COMPREHENSIVE EXAMINATIONS IN COMPUTER SCIENCE 1972 - 1978

edited by

Frank M. Liang

STAN-CS-78-677
NOVEMBER 1978
(Second Printing, August 1979)

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

COMPUTER SCIENCE COMPREHENSIVE EXAMINATIONS 1972 - 1978

by
the faculty and students of the
Stanford University
Computer Science Department

edited by
Frank M. Liang

Abstract

Since Spring 1972, the Stanford Computer Science Department has periodically given a "comprehensive examination" as one of the qualifying exams for graduate students. Such exams generally have consisted of a six-hour written test followed by a several-day programming problem. Their intent is to make it possible to assess whether a student is sufficiently prepared in all the important aspects of computer science. This report presents the examination questions from thirteen comprehensive examinations, along with their solutions.

The preparation of this report has been supported in part by NSF grant MCS 77-23738 and in part by IBM Corporation.

Foreword

This report probably contains as much concentrated computer science per page as any document in existence – it is the result of thousands of person-hours of creative work by the entire staff of Stanford's Computer Science Department, together with dozens of highly-talented students who also helped to compose the questions. Many of these questions have never before been published. Thus I think every person interested in computer science will find it stimulating and helpful to study these pages.

Of course, the material is so concentrated it is best not taken in one gulp; perhaps the wisest policy would be to keep a copy on hand in the bathroom at all times, for those occasional moments when inspirational reading is desirable.

By publishing these examinations we aim to help future students prepare for future exams, and to provide a resource for anyone who wishes to make up similar test questions. Furthermore, I think this is an important historical document, showing what at least one faculty has perceived to be the core of Computer Science during the 1970s.

Speaking of history, I should say a few words about the development of Stanford's Comprehensive Exams. Our department originally gave specialized qualifying examinations in different areas; by the late 1960s there were five such official areas (Artificial Intelligence, Hardware, Mathematical Theory of Computation, Numerical Analysis, and Programming Languages and Systems), each of which was intended to assess a student's qualifications for research work in that area. A student was required to pass the Programming Languages and Systems qual, plus two of the other four quals. Unforunately from the students' standpoint (but fortunately from the standpoint of our discipline), computer science was growing by leaps and bounds, so that every year it took longer and longer to learn everything necessary for one particular qual. As a result it became humanly impossible for a student to pass three quals without almost totally ignoring the subject matter of the other two quals *not* being taken; our students were being forced into an overly-specialized educational pattern.

A major reform was therefore adopted, beginning in the spring of 1972: Instead of three specialized "area quals," we switched to a system that would provide both breadth and depth. Each graduate student was now to pass a new exam called the comprehensive qual, after which he or she was to pass just *one* of the area quals (in the intended thesis area). The purpose of the comprehensive qual was to define and enforce the minimum standard of competence, in all areas of computer science, that we wished each graduate student to have.

Our original intention was to include only "interdisciplinary" questions in the comprehensive exams, questions that couldn't be asked in ordinary courses or in the previous area quals because the answer required knowledge from more than one subject area. Unfortunately, when the first comprehensive qual committee met in 1972, we realized that such notions were too idealistic; almost every interdisciplinary question we could think of was either trivial or an unsolved research problem! Even worse, only the student members of the committee were able to understand questions in more than about two of the subareas of computer science; the faculty couldn't keep up with all the exploding knowledge any better than the students. Nevertheless, we did come up with a few interesting questions that span two or more subareas, and through the years such questions continue to appear. The comprehensive exams began to be subdivided into named areas in 1974, so that a student's weakness in a particular area could be more easily identified by the grading committee. This of course had the unfortunate corollary that interdisciplinary questions became rarer, but the programming problems tend to ameliorate this defect.

It is perhaps necessary to point out that students were never expected to get perfect scores on these exams. In recent years the criterion of "passing" the written test has been to require roughly 1/3 of the points in each area and 2/3 of the total points summed over all areas.

A reading list was given each time to help students prepare for the exam. The cumulative reading list appears at the end of the report, together with dates indicating when each publication was inserted into or deleted from the list. All exams were "open book".

Frank Liang deserves an enormous vote of thanks for undertaking to collect and edit the examinations into publishable form.

D. E. Knuth
June 1978

Syllabus

The Comprehensive Exam is meant generally to cover the material from the following courses (given by Stanford University's Computer Science Department): CS 111 (assembly language); 112 (hardware); 137A (numerical analysis); 140A, 140B, and 246 (systems); 144A (data structures); 156 (theory of computation); and 224 (artificial intelligence). Since the precise content of these courses varies somewhat, the actual scope of the exam will be determined by the references given in the reading list. Please note that the reading list includes some material involving structured programming as well as the history and culture of Computer Science even though it does not correspond to any particular course.

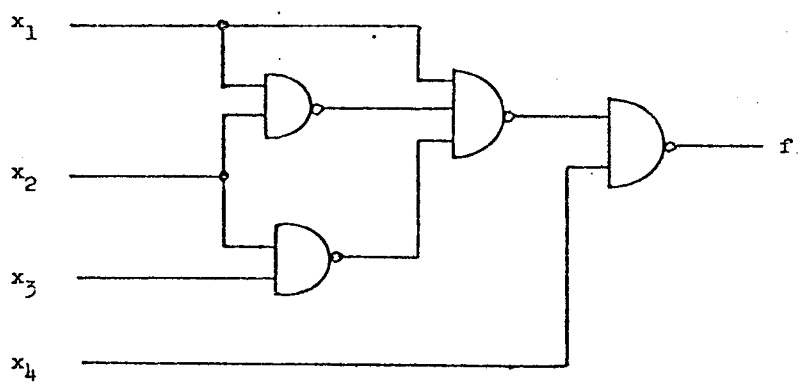
The exam also assumes a certain mathematical sophistication and a knowledge of programming. The mathematical sophistication required includes knowledge of techniques such as induction, recursion, "divide and conquer" (e.g., techniques in sorting algorithms, case arguments, etc.), and will be at the level of an upper division undergraduate in the mathematical sciences. The programming knowledge required will be an ALGOL-like language (e.g., ALGOL W or SAIL), the basic elements of LISP, and possibly some assembly language.

Table of Contents

Spring 1972 Comprehensive Exam

Problem 1. *(5 points)*

Fill in the Karnaugh map shown below for the function realized by the following logic network.



x_1x_2

x_3x_4	00	01	11	10
00				
01				
11				
10				

f

Problem 2. *(5 points)*

Complete the following ALGOL W program by inserting a single *assignment statement* in the box shown. Your program, when executed, should write the value "1" and nothing else (not "0") !

```
BEGIN COMMENT A STRANGE PROGRAM;
   INTEGER ARRAY A(0::0);
   INTEGER PROCEDURE I;
      BEGIN
         [              ];
         0
      END I;
   PROCEDURE WRITEZERO(INTEGER X);
      BEGIN
         X:=0;
         WRITE(X)
      END WRITEZERO;
   WRITEZERO(A(I))
END.
```

Problem 3. *(20 points)*

Discuss the historical development of general-purpose digital computers up to 1950. Mention the names of the principal contributors and the places they worked, and give some noteworthy characteristics of the machines.

Problem 4. *(15 points)*

Using AND-gates and OR-gates, draw a two-stage logic circuit to realize the function

$$f = \bar{x}_1\bar{x}_2x_4 + x_1x_2x_4 + x_1x_2\bar{x}_4 + x_1\bar{x}_2x_3x_4$$

with the minimal number of logic gates and literals (a literal is either an occurrence of a variable or an occurrence of the complement of a variable). Assume that each input variable and its complement are available to your circuit. If a literal appears as input to more than one gate, each appearance counts.

Problem 5. *(20 points)*

Describe, in general terms, the uses of tree data structures in

(a) artificial intelligence applications
(b) system programming applications
(c) numerical analysis applications
(d) hardware applications.

Give as many different kinds of uses as you can.

Problem 6. *(60 points)*

Consider the following context-free grammar:

Start symbol A
Nonterminal symbols A B C
Terminal symbols a b c

Productions	$A \to a$	$B \to b$	$C \to c$
	$A \to aBC$	$B \to bCA$	$C \to cAB$

(a) *(10 points)* Show that this grammar is ambiguous, by constructing two different parse trees for the terminal string *abcabcabc*.

(b) *(20 points)* Prove that if $x_1 \ldots x_n$ is any string of a's, b's, and c's, for $n \geq 1$, the string $(abc)^{3n-1}x_1 \ldots x_n$ is in the language defined by the above grammar. *Hint:* Show that the following are true:

$A \to^* abcAb$
$A \to^* abcAc$
$A \to^* abcabcA$
$A \to^* abcababcAa$

(c) *(20 points)* Consider the following ALGOL W program, which is a "top-down analyzer" based on the above grammar:

```
BEGIN INTEGER P; STRING(150) S;
      LOGICAL PROCEDURE MATCH((STRING(1) VALUE X);
         BEGIN LOGICAL L;
            IF S(P|1)=X THEN BEGIN P:=P+1; L:=TRUE END ELSE L:=FALSE;
            L
         END MATCH;
      LOGICAL PROCEDURE A;
         BEGIN INTEGER Q; LOGICAL L;
            Q:=P;
            IF MATCH("A") AND B AND C THEN L:=TRUE
            ELSE BEGIN P:=Q; L:=MATCH("A") END;
            L END A;
      LOGICAL PROCEDURE B; ... (analogous to A, by symmetry)
      LOGICAL PROCEDURE C; ... (analogous to A, by symmetry)
   READCARD(S);
   P:=0;
   IF A AND MATCH(".") THEN WRITE("ACCEPT")
      ELSE WRITE("REJECT")
END.
```

Suppose that an input card containing the string "ABCABABCABCAC." is punched. Explain what the program does, giving evidence to the grader that you understand the recursive sequence of calculations. Why does the program print "REJECT", even though the corresponding string *abcababcabcac is* in the language defined by the grammar?

(d) *(10 points)* Find a string $x_1 \ldots x_n$ of *a*'s, *b*'s, and *c*'s ($n \leq 71$) which is not in the above language but for which the above program will write "ACCEPT" when $x_1 \ldots x_n$ is punched on the input card. Or, give an informal but convincing proof that no such string exists.

Problem 7. *(20 points)*

Prove by resolution (or by "iterative consensus") that the following set of clauses is unsatisfiable:

$p \vee \sim q,\ q \vee \sim r,\ r \vee \sim p,\ p \vee q \vee r,\ \sim p \vee \sim q \vee \sim r.$

Try to use the minimum number of resolutions necessary, and give an informal proof that your method uses the minimum number.

Problem 8. *(15 points)*

A certain computer has "ideal" 8-digit floating-point decimal arithmetic, in the sense that the result of floating-point addition or subtraction is the true sum or difference of the operands, rounded to 8 digits. Formally,

$$\begin{aligned}\text{floatingadd}(a,b) &= \text{round}(a+b)\\ \text{floatingsubtract}(a,b) &= \text{round}(a-b)\end{aligned}$$

where

$$\text{round}(x) = \begin{cases} 0\,, & \text{if } x = 0 \\ 10^{-k}\lfloor 10^k x + .5\rfloor\,, & \text{if } 10^7 \le 10^k x < 10^8 \\ -\text{round}(-x)\,, & \text{if } x < 0\,. \end{cases}$$

Here $\lfloor x \rfloor$ denotes the greatest integer $\le x$.

(a) What is the smallest value of a floating-point number ϵ such that floatingadd$(1,\epsilon) > 1$?

(b) What is the smallest value of a floating-point number ϵ such that floatingsubtract$(1,\epsilon) < 1$?

(For the purposes of this problem, assume that the range of exponents in these floating-point numbers is unlimited.)

Problem 9. *(15 points)*

The concept of a *buffer* is central to both logic design and programming.

(a) What is a buffer?

(b) Give an example to illustrate the use of buffers in logic design.

(c) Give an example to illustrate the use of buffers in systems programming.

Problem 10. *(25 points)*

A linked list of n elements has been stored in locations $x[1]$ through $x[n]$, with the links in locations $p[1]$ through $p[n]$. The first element of the list is $x[f]$, the next is $x[p[f]]$, the third is $x[p[p[f]]]$, etc. The link $p[p[\ldots p[f]\ldots]]$ (iterated n times) is 0.

The following algorithm rearranges the list, putting the first element in $x[1]$, the second in $x[2]$, $\ldots$, the last in $x[n]$. (You need not prove that the algorithm works, but it will be helpful if you understand in your head how and why it works.)

```
for $k$ := 1 step 1 until $n$ do begin
   while $f < k$ do $f := p[f]$;
   $t := x[k]$; $x[k] := x[f]$; $x[f] := t$;
   $q := p[f]$; $p[f] := p[k]$; $p[k] := f$;
   $f := q$
end
```

(Here t is an auxiliary variable having the same type as x, and q is an auxiliary integer variable.)

Suppose that the *final* contents of the arrays, after this algorithm has been performed, are as follows ($n = 8$) :

```
   j  = 1 2 3 4 5 6 7 8
x[j] = R N M O E G V C
p[j] = 6 7 6 8 5 6 8 8
```

(If you understand the algorithm, you will know the final value of f.)

What were their *initial* contents, just before the algorithm was performed, and what was the initial value of f?

```
   j  = 1 2 3 4 5 6 7 8
x[j] =
p[j] =
   f  =
```

PROGRAMMING PROBLEM

This programming problem is designed to emphasize topics in hardware design and the creation of nontrivial programs, as well as numerical analysis.

The problem is to determine the value of

$$\int_0^1 \sqrt{x+x^4}\,dx$$

in decimal notation, correct to 18 decimal places.

Hint: It may be helpful to first transform the integral before blindly applying a formula for numerical integration.

All programming should be done in either ALGOL W or SAIL. You should not use any subroutines not written by yourself, except for fundamental routines such as input/output or square root. Thus, you will probably have to write some extended-precision arithmetic subroutines.

Your grade will be based on (1) clarity of the program and its documentation, (2) how close you are to getting the correct 18-digit answer, and (3) running time of the program.

Winter 1973 Comprehensive Exam

Problem 1.

A hash table of size N contains K items. Collisions are resolved with random or linear quotient probing. There is a probability P that the items for which searches are conducted are in the table. What is the expected number of probes necessary to search for one of these items?

Problem 2.

Consider a three-digit chopped decimal floating-point number system F, with numbers

$$x = \pm .d_1 d_2 d_3 \times 10^e,$$

with $-100 \le e \le 100$, and $0 \le d_i \le 9$. Let F be normalized (i.e. $x \ne 0 \supset d_1 \ne 0$). The number zero is contained in F and has the unique representation: $+.000 \times 10^{-100}$.

We denote by $\oplus$ the operation of floating-point addition. For all x and y in F, the value of $x \oplus y$ is defined as the floating-point number closest to $x+y$ whose magnitude is less than or equal to the magnitude of $x+y$. We denote by $\otimes$ the operation of floating-point multiplication. For all x and y in F, the value of $x \otimes y$ is defined as the floating-point number closest to $x \times y$ whose magnitude is less than or equal to the magnitude of $x \times y$.

(a) How many different real numbers can be exactly represented by F ?

(b) Find examples of x, y, $z \in$ F to show that the following statements are *not* generally true:
(1) $(x \otimes y) \otimes z = x \otimes (y \otimes z)$
(2) $(x \oplus y) \oplus z = x \oplus (y \oplus z)$

(c) Find an example where $(x \oplus y) \oplus z$ has a relative error of at least 50% .

Problem 3.

Several processors must each be able to access a critical data base. Since any of them may alter it, they must not access the data base simultaneously. Each processor may run at a different speed. The following mechanism is proposed to accomplish this:

Each processor before accessing the data base will check a special "lock" word in memory. The lock contains a 1 if any other processor is accessing the data base, and a 0 if not.

Will the following instruction sequence executed by each processor accomplish the goal of shared but not simultaneous access? Explain your answer.

```
TEST    LDA     lock        (load accumulator from lock)
        BRP     TEST        (branch if accumulator > 0 to TEST)
        SET     lock        (set lock to 1)
     access data base
```

```
        .
        .
        .
      end access
        CLR      lock              (zero lock)
```

Can you improve on this program? You may rearrange the given instructions, delete some, and/or add other reasonably implementable instructions.

Problem 4.

You have been hired to design a syntax-checker for Stanford's new, optimal programming language, NIL.

The language is as follows: A program consists of a series of statements, each of which is one word. A program must begin with BEGIN and end with END. The "initial segment" of a program, following BEGIN, may have as many ADD1 statements as you wish, possibly none, but each ADD1 statement must be immediately followed by a SUBTRACT1 statement. (This is the only place a SUBTRACT1 statement can occur.) After the "initial segment" comes a "final segment" which consists of a non-zero number of "nochange" statements. Each "nochange" statement may be either an ADD0 or a TIMES1 statement.

(a) Describe the syntax of NIL by a regular expression. Design a syntax checker which may be either a transition graph or a finite automaton (designate which) that accepts exactly the programs of this language.

(b) You were successful and now the user community has demanded a more flexible language. Accordingly, the language is revised so that in the initial segment of the program the ADD1 and SUBTRACT1 statements can occur in any order. Of course the program can still only compute the same value (using the obvious meaning of the statements). Give a formal grammar of the syntax of this initial segment. Will a regular language suffice? Why or why not? Will a context-free language suffice? Why or why not?

Problem 5.

Consider a sequence g_k $(k = 0, 1, \ldots)$ which satisfies the relationship

$$g_k = c - sk + \sum_{p=1}^{\infty} a_p \lambda_p^k$$

where c, s, and $\{a_p, \lambda_p\}_{p=1}^{\infty}$ are unknown constants with $|\lambda_p| < 1$ and $|\lambda_p| \geq |\lambda_{p+1}|$.

(a) Given numerical values g_0, g_1, g_2, g_3, show how to use Aitken-extrapolation to determine an approximate value of s.

(b) Under what circumstances will your algorithm yield the exact value of s when g_0, g_1, g_2, g_3 are given?

(c) Perform the calculation to determine s, given the following values for g_k:

$g_0 = -2.5, \; g_1 = 14.5, \; g_2 = 4.5, \; g_3 = 8.$

Problem 6.

Consider a complete resolution proof procedure that will find all possible resolvents (and factors) of a set of clauses, all resolvents (and factors) of those, etc. The procedure terminates on the empty clause or in the absence of further distinct resolvents. For each of the sets of clauses given below, determine: (1) Does the procedure terminate? (2) Is the set satisfiable? Explain.

(a) $\{\sim p(x) \vee p(f(x))\}$
$\{p(a)\}$
$\{q(a)\}$
$\{q(a) \vee \sim q(b)\}$

(b) $\{p(f(x)) \vee \sim q(x)\}$
$\{\sim p(g(a))\}$
$\{q(a) \vee r(a)\}$
$\{\sim r(x) \vee \sim q(x)\}$

(c) $\{p(x) \vee q(x)\}$
$\{\sim p(a) \vee r(a)\}$
$\{\sim q(y) \vee r(y)\}$
$\{\sim r(a) \vee p(a)\}$
$\{\sim p(z) \vee \sim r(z)\}$

(Here a, b are constants; x, y, z are variables; p, q, r are predicates; and f, g are functions.)

Problem 7.

Show that for each of the three common fixed point binary number representations (signed magnitude, ones' complement, two's complement), there is a reasonable floating point number representation such that the same comparison instruction can be used (1) to test whether one fixed point number is greater than another and (2) to test whether one normalized floating point number is greater than another.

Problem 8.

TWENTY QUESTIONS

1. Which machine is faster? (a) IBM 360/65; (b) CDC 6600.

2. What is the main innovation of the IBM 360/85?

3. Which of the following are stack machines?

(a) PDP-10
(b) B 5500
(c) 360/65

4. Which language is designed especially for string processing?
(a) ALGOL W
(b) SNOBOL
(c) LISP

5. What is the fastest general-purpose digital computer?

6. What is the cheapest general-purpose digital computer?

7. Are there any general-purpose 20 ns cycle time computers? If yes, give name.

8. What is the essential difference between combinational and sequential hardware circuits?

9. How many switching functions of 3 variables are there?

10. How fast can sorting be done on a single sequential machine?

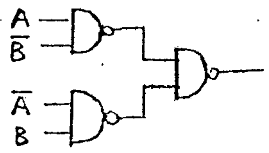
11. Let $f(x) = (x-a)^2$, and assume that x_0 is close enough to a. What is the rate of convergence for the iteration $x_{n+1} = x_n - f(x_n)/f'(x_n)$, $n = 0, 1, \ldots$?

12. Let $f(x) = \cos x$. For a given accuracy, which method requires fewer operations for finding a zero of $f(x)$: Newton's method or the method of regula falsi? Why?

13. Approximately how long is one nanosecond?
(a) one millimeter; (b) one inch; (c) one foot; (d) one kilometer.

14. Name two reasonable algorithms that might be used to determine which page to replace when a new page is brought into main memory in a paging memory system.

15. What switching function does the following logic diagram implement? Describe in simple form.



16. Consider 1's complement versus 2's complement representation of integers. From a hardware standpoint, in which system is it easier to implement the complement operation? Why?

17. What information must be saved for later restoration when an interrupt is received and serviced?

18. Is a queue or a stack the more common data structure in parsing? In I/O routines?

19. Give a good candidate for the smartest A.I. program.

20. Write down at least one "computer science joke."

PROGRAMMING PROBLEM

You are given the six distinct objects that can be made from four unit cubes placed face to face so that none of them is a rectangular parallelpiped, and the one object that can be made from three unit cubes with the same restrictions.

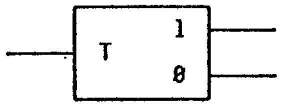
The problem is to write a program that will determine all of the distinct ways in which these seven objects can be assembled into a 3x3x3 cube. This is the classical SOMA cube puzzle. It is well-known that there are 240 solutions. Since this problem is classical, you are expected to do it entirely on your own. It is not open book, except for programming reference manuals. You may use or build a version of the puzzle to experiment with while working on the problem.

All programming should be done in either ALGOL W or SAIL. The grade will be based on (1) clarity of program documentation, (2) structure and elegance of the program, and (3) running time of program. Partial credit will be given for incomplete answers. Note: your program will take at least a minute or so to run.

Spring 1973 Comprehensive Exam

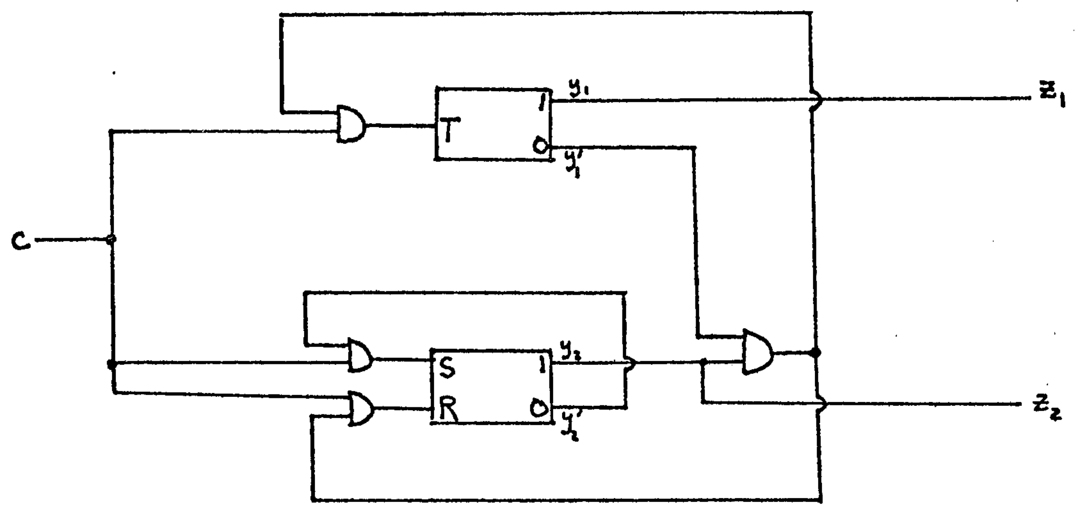
Problem 1. *(15 points)*

(a) A T flip-flop is shown schematically as



There is a single input, and the state of the flip-flop changes with each pulse on the input line. Show how a T flip-flop could be made from a single S-R (set-reset) flip-flop, OR gates, and AND gates.

(b) The circuit shown below contains one T flip-flop and one S-R flip-flop. There is a single clock input, c, and two outputs, z_1 and z_2. Assume that initially $z_1 = 0$ and $z_2 = 0$. Fill in the table below to show the values of the indicated variables after the arrival of each clock pulse.



	T	S	R	y_1	y_2	z_1	z_2
Initial state (no clock pulse present)	0	0	0	0	0	0	0
After arrival of 1st clock pulse							
After arrival of 2nd clock pulse							
After arrival of 3rd clock pulse							
After arrival of 4th clock pulse							
After arrival of 5th clock pulse							
After arrival of 6th clock pulse							

Problem 2. *(15 points)*

For each of the languages given below, decide whether the language is regular, context-free and not regular, or recursive and not context-free. Explain your decision in some convincing (not necessarily formal) manner.

In the following, assume that i, j are positive integers and that n is a fixed positive integer.

(a) $\{ a^i b^j \mid j = n-i \}$
(b) $\{ a^i b^j \mid j = n+i \}$
(c) $\{ a^i b^j \mid j = n*i \}$
(d) $\{ a^i b^j \mid j = n/i \}$ ("/" interpreted as integer division)
(e) $\{ a^i b^j \mid j = i^n \}$

Problem 3. *(30 points)*

Let G be the grammar whose productions are given below. Capitals ($A, B, \ldots$) denote nonterminal symbols, lower case letters ($a, b, c, \ldots$) denote terminals, and ϵ denotes the empty string. The nonterminal S is the start symbol.

$$
\begin{aligned}
S &\rightarrow AC \\
A &\rightarrow CB \mid aB \\
B &\rightarrow C \mid Ab \\
C &\rightarrow c
\end{aligned}
$$

(a) Is the grammar left-recursive? That is, could a top-down recognizer loop forever when parsing sentences with this grammar? Why or why not?

(b) Find the precedence table for this grammar and determine if it is a (1,1) precedence grammar.

For parts (c), (d) and (e), use the grammar above and add the production $C \rightarrow \epsilon$.

(c) Determine which nonterminals produce the empty string.

(d) Answer part (a) for the modified grammar.

(e) Answer part (b) for the modified grammar.

Problem 4. *(30 points)*

Distinguish between *paging* and variable length *segmentation* (without paging) with respect to:

(a) memory utilization
(b) memory allocation problems
(c) ease of program sharing
(d) implementation of virtual memory.

Note: typical machines
paging: (360/67, XDS Sigma 7, Honeywell 6180)
segmentation: (B6700, Honeywell 6180)

Problem 5. *(15 points)*

Architecture Problem (systems)

What machine architecture features are needed (or are useful) to accomplish:

(a) looping
(b) dynamic program relocation
(c) recursion
(d) input/output
(e) implementation of P and V mutual exclusion primitives
(f) debugging at machine/assembly language level.

Problem 6. *(45 points)*

GPS SOLVES THE MONKEY, APE AND BANANAS PROBLEM

Problem: A room contains a monkey, an ape, a box, and some bananas which are hanging from the ceiling. The monkey wants to eat the bananas, but he cannot reach them unless he is standing on the box, when it is under the bananas. The ape is willing and able to help the monkey. How can the monkey get the bananas?

GPS task formulation and task environment:

initial state:	monkey's place = place1 box's place = place2 ape's place = place3 contents-of-monkey's-hand = empty contents-of-ape's-hand = empty
goal state:	contents-of-monkey's-hand = bananas
places:	place1, place2, place3, under-bananas
animals:	monkey, ape

operators:

CLIMB:	input-	monkey's place = box's place
	action-	monkey's place becomes on-box
WALK:	input-	x is in the set of places, a is in set of animals
	action-	a's place becomes x
MOVE-BOX:	input-	x is in the set of places
		ape's place is in set of places
		ape's place = box's place
	actions-	ape's place becomes x
		box's place becomes x
GET-BANANAS:	input-	box's place = under-bananas
		a is in set of animals
		a's place = on-box
	action-	contents-of-a's-hand becomes bananas

differences:	D0 = (monkey's place)
	D1 = (ape's place)
	D2 = (box's place)
	D3 = (contents-of-monkey's-hand)
difference ordering:	(D3, D2, D1, D0)
table of connections:	consider all operators relevant to all differences

Write out the trace of the GPS solution to this problem. GPS is to be taken to mean the problem solving system described in chapter 8 of Newell and Simon, *Human Problem Solving*, particularly as summarized by Figure 8.7. Your answer should be laid out in goal-subgoal structure, indicating differences found, operators applied, and intermediate states achieved. A model for your answer is Fig. 8.10 in *HPS*, although your trace does not have to be quite as thorough and detailed as that one.

Problem 7. *(15 points)*

```
BEGIN
REAL A;
REAL PROCEDURE P; A+1

A ← 0;
   BEGIN
   REAL A;
   PROCEDURE Q(REAL PROCEDURE R); WRITE(R);

   A ← 1;
   Q(P);
   END
END.
```

```
BEGIN
REAL A;
REAL PROCEDURE P(REAL A); A+1;

A ← 0;
   BEGIN
   REAL A;
   PROCEDURE Q(REAL PROCEDURE R); WRITE(R(A));

   A ← 1;
   Q(P);
   END
END.
```

(a) What will each of these two ALGOL W programs print?

(b) What issue in language design is illustrated by this example? Name a language which does this differently from ALGOL. What are the arguments for each side?

Problem 8. *(30 points)*

You are on duty as a consultant for a computer center and you have just been asked to provide subroutines to compute the error function and the inverse error function. Neither of these is in your library, but you do have access to an otherwise well-equipped library of general numerical mathematics subroutines. Your visitor needs immediate answers to his programming problem because he has planned on using these functions later this morning. However, he is interested in computing only a few numbers with these subroutines (at unknown points) and great accuracy is not important. Neither are speed or elegance, but inputting tables will not solve his problem (or yours).

(a) Provide algorithms using only general library subroutines to compute the error function, defined by

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \,,$$

and the inverse error function, defined by

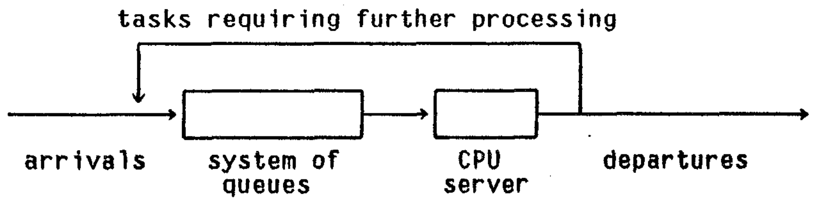
$$\mathrm{inverf}(z) = x \iff \mathrm{erf}(x) = z.$$

You need only outline your solution, but you must give a brief description of any library subroutines you would use. Pay particular attention to what information or parameters would be necessary or useful. For some representative values of these parameters, give an estimate of the accuracy you would expect your algorithms to achieve.

(b) Briefly comment on the efficiency of your solution and suggest alternative approaches you might try if you wanted to add these functions to your library (and had a week to do so).

PROGRAMMING PROBLEM

You have been asked by a local computer system manufacturer to simulate the design of a multiprogramming operating system. You are seeking to find out, for example, at what rate tasks can be processed (throughput), to what extent memory is utilized, and how task priority and processing time requirements affect the turnaround time for a particular task. The basic system model is shown below.



Tasks arrive at Poisson intervals and bring with them certain demands for systems resources. Call these demands d_i, where i runs from 1 to n. When a task arrives, we could form a demand vector, $\underset{\sim}{d} = (d_1, d_2, \ldots, d_n)$, which represents that task's resource requirements. We will assume that these demands remain fixed for the life of the task. Examples of possible demands include CPU time, main memory, disk memory, peripheral equipment, priority, or special data bases.

With respect to the problem of scheduling tasks to be run, we first observe that no task will be put in "ready to run" status until it is possible to allocate to that task all resources which the task has demanded. In the case of CPU time demand, this philosophy is tempered by the multiprogramming notion to the extent that each task which is ready to run will ultimately be given a quantum of time, q seconds, during which the CPU is devoted to running that task. If the task requires further processing when the q seconds are up, then it will be returned to the ready to run queue. Otherwise, it will depart from the system.

The system you will model has four resources to allocate:

1. CPU computation time
2. Main memory
3. Disk memory
4. Priority

The first and fourth of these resources are limited in the sense that no CPU time demand may exceed some limit, L_1, and no priority may fall outside the range $1 \leq \text{priority} \leq L_4$. The second and third resources are limited in the sense that no more than L_2 words of main memory and L_3 disk tracks can be allocated at one time. The implication is that some arriving tasks may have to wait until other tasks depart before they may be initiated.

Since we are simulating the real world, we must find a way to generate task arrivals and task resource demands. We do this by drawing demands and arrival times from suitably distributed random variables.

Arrival of a new task occurs at intervals which are distributed exponentially, with mean I (mean interarrival time). Upon arrival, a task is assigned a demand vector $\underset{\sim}{d}$, with d_1, d_2, d_3 (CPU time in units of q seconds, main memory in words, and disk memory in tracks, respectively) chosen from exponentially distributed random variables whose means are m_1, m_2, and m_3 respectively. The priority of the task, d_4, is given an integer value between 1 and L_4, using a uniform distribution. The larger the value of d_4, the higher the priority of the task.

By now it should be apparent that the system of queues that must be maintained is split into two parts:

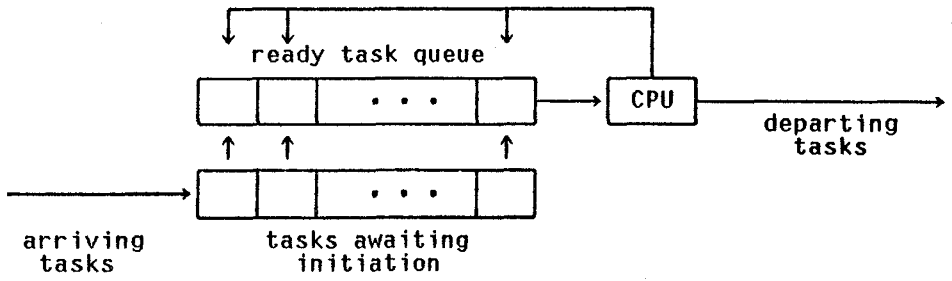
(1) Tasks, ordered by priority, which are awaiting initiation because there were insufficient resources available when the task first arrived.

(2) Tasks, ordered by priority, which have been initiated (i.e. allocated necessary resources) and are ready to run, but which are queued up waiting to get a quantum of time on the CPU.

The scheduler is run each time a task's quantum runs out. The scheduler first attempts to move tasks from the queue of tasks awaiting initiation to the queue of tasks which are awaiting CPU access, and then the scheduler attempts to start up the highest priority task which is ready to run.

An arriving task will be given a demand vector and will be queued at the rear of the tasks in its priority awaiting initiation. Ultimately (we hope), it will be placed in the ready queue by the scheduler.

The figure below shows the queueing system in more detail.



Given the following parameters, run your simulation until 1000 tasks have been completed. Note that it may be necessary to initiate more than 1000 tasks. Collect statistics as described below.

Parameters:

I	= mean interarrival time = 30 seconds
m_1	= mean CPU time required = 20 q units
m_2	= mean main memory required = 200 K bytes
m_3	= mean disk space required = 20 tracks
L_1	= maximum CPU time allowed = 240 q units
L_2	= maximum memory available = 2000 K bytes
L_3	= maximum disk space available = 200 tracks
L_4	= number of priority levels = 4
q	= quantum time = 0.5 seconds

Note that the total CPU times of all tasks waiting for initiation or in the ready queue may exceed 240 q, but no single task may have a CPU demand in excess of 240 q. Furthermore, when a task departs, it returns all disk and main memory that it demanded.

Statistics:

Define T to be the total time the system is in operation, i.e. T is the difference between the time at which the 1000th task departs from the system and the time at which the first task enters the waiting queue.

(1) Average throughput (broken down by priority) = $\frac{\text{total number of priority } i \text{ tasks run}}{T}$

(2) The mean and variance of queue lengths in both waiting and ready queues, broken down by priority. Collect these statistics incrementally each time the scheduler is run.

(3) Memory utilization. Let t_i be the interval of time from the moment task i first enters the ready queue to the time task i departs from the system. Let $d_2(i)$ be the memory demand assigned to task i when it enters the system. Then memory utilization is defined as:

$$\left(\sum_{i=1}^{1000} t_i * d_2(i)\right) \Big/ (L_2 * T)$$

where i runs over all tasks which have been completed, and T is as defined earlier.

(4) Disk utilization. (Same as (3), except replace $d_2(i)$ by $d_3(i)$ and L_2 by L_3.)

(5) For each priority class, prepare a scatter plot of the total time a task is in the system against the CPU time demand the task makes. Note that this "total task time" is not the same as the t_i in (3).

Winter 1974 Comprehensive Exam

Problem 1.

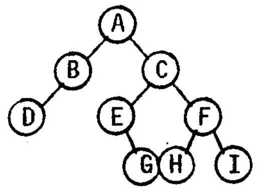
Consider the following seven steps which are executed (recursively) in some binary tree traversal algorithms:

1. Visit the root
2. Visit the left successor of the root
3. Traverse the left subtree of the left successor of the root
4. Traverse the right subtree of the left successor of the root
5. Visit the right successor of the root
6 Traverse the left subtree of the right successor of the root
7. Traverse the right subtree of the right successor of the root

(To be completely precise, steps 2, 3, and 4 should be preceded by the phrase "If the root has a left successor"; and steps 5, 6, and 7 should be preceded by the phrase "If the root has a right successor".)

(a) Express the standard traversal algorithms of preorder, inorder, and postorder in terms of the above steps.

(b) Consider the following tree; call it S.



(1) Let the "inside out" algorithm be defined by executing the steps in the order 4 6 3 7 2 5 1. Apply this algorithm to tree S. In what order are the nodes visited?

(2) Which of the algorithms, when applied to tree S, visit the nodes in the order I H F C B G E A D?

(3) Give a permutation of (A B C D E F G H I) which cannot represent the order in which the nodes of S are visited by one of the 7! algorithms which can be defined in terms of the steps above. Prove your answer.

(c) Suppose all 7! algorithms which can be defined in terms of the steps above are run on some arbitrary (but fixed) tree T, resulting in 7! permutations of the nodes of T. How many of these permutations are distinct?

(d) Given an algorithm A, consider the algorithm A^R defined by reversing the order of the seven steps. (E.g. if A is 4 5 2 3 1 6 7 then A^R is 7 6 1 3 2 5 4.) Are the nodes of all trees traversed by A^R in the opposite order to the order they are traversed by A? Prove your answer.

Problem 2.

(a) How would you recognize that you have obtained the nth degree polynomial $P(x)$ that is the minimax (Chebyshev) approximation to a function $f(x)$ in $C[a,b]$?

(b) Use your answer to (a) to find the straight line that is the best Chebyshev approximation to the quadratic $ax^2 + bx + c$ in $[-1,1]$.

(c) Find the answer to the problem in (b) by expanding the quadratic in a series of Chebyshev polynomials $T_0(x) = 1$, $T_1(x) = x$, $T_2(x) = 2x^2-1$. State the general theorem you are using.

(d) Suppose the interval in (b) is changed to $[0,1]$. Explain how you would apply the methods of (b) and (c), but do not carry out the details.

(e) Prove that

$$f(x) = (x-x_1)(x-x_2)\ldots(x-x_n)$$

where the x_i are at your disposal, is minimized with respect to the maximum norm in $[-1,1]$ by choosing $x_i \doteq \cos[(2i-1)\pi/2n]$.

Problem 3.

Ternary operations

Dr. Art D. Ware once had the marvelous idea of using ternary logic for improving the speed of computer hardware. He uses "trits" instead of bits, and the following functions:

	-1	0	+1
-1	-1	-1	0
0	-1	0	+0
+1	0	+1	+1

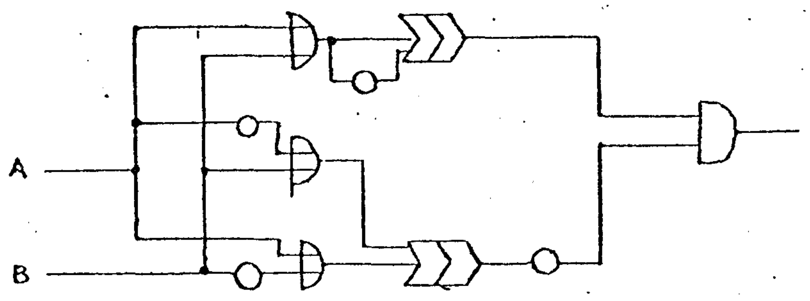
	-1	0	+1
-1	-1	0	0
0	0	0	0
+1	0	0	+1

	-1	0	+1
-1	-1	0	+1
0	0	0	+1
+1	+1	+1	+1

-1	0	+1
+1	-1	0

-1	0	+1
+1	0	-1

There are three binary operators, And ( ), Or ( ), and Sup ( ); and two unary operators, Rot ( ) and Neg ( ). Use these components to build a half-adder and then a full adder. To help you get started we give below the construction of a multiplier modulo three.



Mul(a,b) = And(Sup(Or(a,b),Neg(Or(a,b))),Neg(Sup(Or(a,Neg(b)),Or(Neg(a),b)))).

Your end result should have a similar form.

Problem 4.

Consider the polynomial $f(x) = x^3 + 3x^2 + 2x - 1$.

(a) Show that there is a positive zero α and find an interval of length 1 (i.e. an interval of the form $[a,a+1]$) which contains α.

(b) Consider the iterative method

$$x_{n+1} = x_n - mf_0(x_n) ,$$

where $x_0 \in [a, a+1]$ and

$$f_0(x) = \begin{cases} f(a) , & x < a \\ f(x) , & a \le x \le a+1 \\ f(a+1) , & a+1 < x \end{cases} .$$

Find a value of m for which this method converges to α for any $x_0 \in [a, a+1]$. Show that for your value of m the sequence x_n does converge to α . Discuss what happens if x_0 is an arbitrary real number, considering convergence or non-convergence, and if convergent, the speed of convergence.

(c) Consider the iterative method

$$x_{n+1} = \frac{1}{2}(1 - x_n^3 - 3x_n^2) , \quad x_0 = a + \frac{1}{2} .$$

Is this method convergent? Explain.

(d) Consider Newton's method

$$x_{n+1} = x_n - f(x_n)/f'(x_n) , \quad x_0 = a + 1 .$$

Discuss the convergence or non-convergence.

Problem 5.

The Algac 60 executes Algol 60 programs directly except for input/output. It reads a number by a statement x := *read* and writes by a procedure *write*(x). It has a clock interrupt to a statement labelled *foo* every algosecond. The procedure *return* is used to return from the interrupt. Write the program at *foo* and establish conventions for communicating with it so as to read and write arrays of numbers.

<u>Problem 6</u>.

Prove or disprove each of the following four statements.

(a) $L(G)$ is regular, where G is the grammar $(\{S, A, B, C, D\}, \{0, 1\}, P, S)$ and the productions P are:

$$
\begin{array}{ll}
S & \rightarrow AB \\
A & \rightarrow CCC \\
B & \rightarrow DDD \\
CD & \rightarrow DC \\
DC & \rightarrow CD \\
CDC & \rightarrow DCD \\
DCD & \rightarrow CDC \\
C & \rightarrow 1 \\
D & \rightarrow 0
\end{array}
$$

(b) $\{ww^R : w \text{ in } \{a, b\}^*\}$ is linear context free.

(c) $\{wcw^R : w \text{ in } \{a, b\}^*\}$ is regular.

(d) $\{a^n b^n c^n : n \geq 1\}$ is context free.

<u>Problem 7</u>.

Consider a game playing program for a game in which there are always exactly two moves to consider, A and B. The program explores the game tree to a fixed depth D, and uses an evaluation function that assigns a different value to each bottom position (the positions at depth D from the starting position). The program always considers the moves in a fixed order (A then B) because it is difficult to tell a priori which of the two moves is likely to be preferred. In order to reduce the size of the game tree considered, the program uses a restricted "alpha-beta" pruning procedure which only does one level cutoffs. (In Nilsson's terminology: when deciding whether search should be discontinued below a given node, its provisional backed-up value is compared with the provisional backed-up value of its parent node only.) Thus there are no deep cutoffs.

What is the expected number of bottom positions this game playing program will consider as a function of D?

<u>Problem 8</u>.

Base 7 adder

Design, as efficiently as you can, using And (ᗜ), Or (Ψ), and Neg (ϕ) functions, a three-bit, base seven full adder. The input is seven bits, including the carry. The output is three bits and a carry. Say why you think that your design is good.

Problem 9.

The actual parameters of an ALGOL 60 procedure may be called by name or by value. Explain clearly the difference between these two mechanisms, and say which you would choose in a situation where either could be used. In what circumstances is it essential to call parameters by name?

The following procedure prints a table of values of the function

$$f(x) = ax^2 + bx + d \text{ for } x = 2, 3, \ldots, 10;$$

the method used is unusual.

```
procedure David (a, b, c);
  real a, b, c; value a, b, c;
begin
  integer i; real y;
  y := 2;
  for i := 1 step 1 until 9 do begin
    print (a*y*y + b*y +c);
    c := c + b + a;
    b := b + 2*a;
  end
end David;
```

Trace the action of this procedure when activated by the program segment

$p := p_0;\ q := q_0;$
$David\ (p + q,\ p,\ q);$

and deduce the values of $f(2)$ and $f(3)$ computed

(a) with the procedure as given;
(b) if the part **value** a, b, c; is omitted.

Note that the action of this procedure is the same in ALGOL W as in ALGOL 60. Only the notation of the procedure heading is different in the two languages.

Problem 10.

A cubical barrel contains 27 identical spherical apples in a regular 3x3x3 array. A worm is to start at an outside apple and eat his way from apple to apple visiting each apple once and going to a touching apple. Write a sentence of predicate calculus with equality whose satisfiability expresses the possibility of the worm doing this and ending up in the center. It is not necessary to determine whether the worm can do it in order to write the sentence.

(a) How might you quibble about this problem?
(b) Solve it without quibbling.

Problem 11.

QUICKIE QUESTIONS

(a) What is Professor Knuth's middle name?

(b) Perform the following 9-bit ones complement addition:

```
  1 1 1 0 0 1 1 0 1
+ 1 1 1 0 1 0 0 1 0
+ 1 1 0 1 0 0 0 0 1
-------------------
```

(c) Compute 9−1+2+3+4∗.

(d) What is the importance of $\hat{g}$, in connection with the heuristic power of $\hat{h}$, especially considering the admissibility and optimality of A^*?

(e) Is ALGOL W a context-free language?

(f) Apply Warshall's algorithm to the following binary matrix:

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

(g) What data structure would you use to represent a 100 × 100 array which you know will always have less than five non-zero elements?

(h) What is a nybble?

(i) Prove, in any way you can, $\exists x \forall y\ (Fx \supset Fy)$.

(j) What is the biggest architectural flaw of the IBM S/360?

(k) What is strength reduction?

(l) Alphabet soup: expand the following abbreviations:

APL
BNF
COBOL
AVL
CCW

(m) For integration of a first order differential equation with initial condition, multistep methods such as Adams' method are generally more efficient than the Runge-Kutta method. Under what circumstances or in what portion of the calculations would the Runge-Kutta method be preferred?

(n) Let $S_1 = \sum_{n=1}^{100000} \frac{1}{n^2}$ and $S_2 = \sum_{n=100000}^{1} \frac{1}{n^2}$.

If these sums are computed on a finite precision computer of the usual type, which one would you expect to have the smaller round-off error and hence give a better approximation to

$$\sum_{n=1}^{\infty} \frac{1}{n^2}?$$

Why?

(o) How would you calculate $(x) = x - \sqrt{x^2-\alpha}$ correctly to the number of digits used in x when x is very large compared to α?

(p) In a few words what is the most important property of a spline function approximation which makes it better than a polynomial approximation for interpolating a given function at a given set of points?

(q) What is the biggest architectural flaw of the PDP-10?

PROGRAMMING PROBLEM

A *semigroup* is a set of objects S together with a binary operation $*$ such that $(x*y)*z = x*(y*z)$ for all x, y, z in S. To define a semigroup on the four objects A, B, C, D, we must specify a *multiplication table* for the operation $*$, namely a 16-tuple

(A$*$A, A$*$B, A$*$C, A$*$D, B$*$A, ..., D$*$D)

which satisfies the 64 relations

$$
\begin{gathered}
(A*A)*A = A*(A*A)\\
(A*A)*B = A*(A*B)\\
\vdots\\
(D*D)*D = D*(D*D)
\end{gathered}
$$

Two semigroups defined by multiplication tables $*$ and $*'$ are *isomorphic* if there is a permutation p of (A B C D) such that $p(x*y) = p(x)*'p(y)$ for all x, $y \in S$.

You are to write a program which prints out all such semigroup multiplication tables in lexicographic (i.e. dictionary) order, except that you should list only the lexicographically first table from each class of isomorphic semigroups. In other words, you are to find all the nonisomorphic semigroups on four objects.

Example: For two objects, the answer would be

AAAA, AAAB, AABB, ABAB, ABBA;

since the set of all semigroup multiplication tables is

AAAA,	AAAB,	AABB,	ABAB,	ABBA,	ABBB,	BAAB,	BBBB;
1	2	3	4	5	6	7	8

and the pairs (1,8), (2,6), (5,7) are isomorphic.

This is intended as a programming problem and not as a problem in group theory. The problem is classical, but you are expected to do it entirely on your own; the definitions above should be sufficient. It is not open book, except for programming reference manuals.

As your program generates solutions, have it print out the elapsed time, along with any other measures of performance you think important. Your grade will be based on the structure and elegance of your program and the clarity of your documentation, as well as its performance.

You may find the following facts helpful:

(1) There are less than 10,000 semigroups on four objects.
(2) Less than 1000 of these are isomorphically distinct.
(3) The grader's program took less than ninety seconds to run.

Spring 1974 Comprehensive Exam

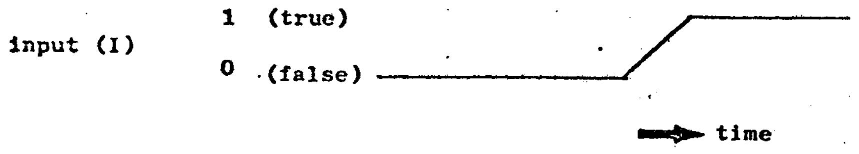
HARDWARE

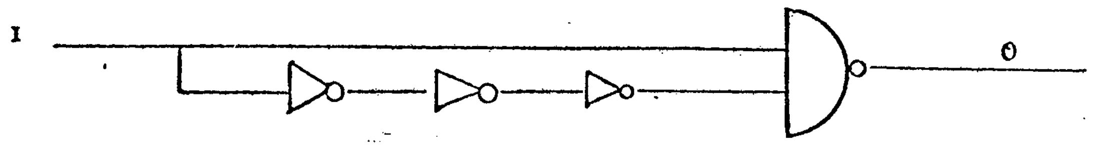
1. (7,3)-counter. *(20 points)*

A (7,3)-counter is a combinational circuit with 7 inputs and 3 outputs, in which the 3-bit output is the binary representation of the number of 1's present on the inputs. Assuming that a (7,3)-counter costs \$1 and a one-bit full adder costs \$2, design a combinational network of (7,3)-counters and full adders to calculate the sum of fourteen 10-bit unsigned integers. Give a strategy for the design and state the number of each building block needed, making the cost of your design as low as possible.

2. Trigger. *(10 points)*

The following circuit might be called the "poor man's trigger". Sketch the behavior of the output signal when the circuit is presented with the following input signal:





Explain briefly the reasons for the behavior observed.

SYSTEMS

3. Software factory.

You are the director of a large software consulting company. The XYZ company has just given you a fixed price contract to supply all the software for its new machine which it will market in *two years*. The system should sell for about \$300K to small universities. The contract allows you the normal outrageous profit if you can produce the software with less than ten man-years of effort. Assume that you have available whatever software talent is needed. Lay out a software development schedule in some detail – show what software is to be developed, when development starts and stops, and some indication as to the way in which major packages would be implemented. Explain clearly any additional assumptions you are forced to make in order to make the question more well-defined.

4. Operating systems/architecture. *(20 points)*

A systems designer at Veeblefetzer and Beebleberry has noticed that machines which use base and bound registers to implement *virtual addressing* must still use physical memory addresses for I/O, while using virtual addresses for program references in the CPU. This is because the transformation from virtual to real addresses is done by the CPU.

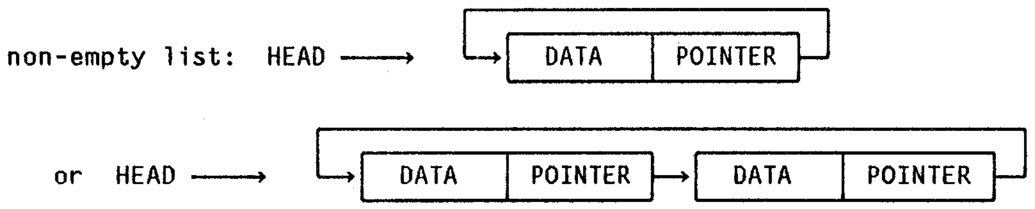
The CPU address calculation logic (e.g. indexing, indirect addressing) produces a virtual address. This value is added to the contents of the current base register producing the correct physical memory address.

The designer decides to make things more symmetric by building the virtual-to-real mapping into the memory. This way, he reasons, programs can issue I/O requests referring to virtual buffer addresses and these can be used in the various I/O devices since the virtual addresses will be transformed to physical ones at the memory. He also hopes to permit dynamic program relocation even during I/O, since all addresses entering the memory request queue will be virtual. Evaluate this strategy, pointing out as many flaws or weaknesses as you can.

5. Circular lists. *(10 points)*

A programmer has designed a circular list format as follows:

empty list: HEAD ≡ 0



He writes the following routine to insert a new data element DNEW into the list. FREE is the index of an available node in PointerArray.

```
IF HEAD = 0 THEN BEGIN
    Pointer[FREE] ← FREE;
    HEAD ← FREE
END
ELSE BEGIN
    Pointer[FREE] ← Pointer[HEAD];
    Pointer[HEAD] ← FREE
END;
Data[FREE] ← DNEW;
```

Specify a change to the data structure which will allow you to simplify the insertion routine by eliminating the conditional. The change should preserve the circularity property. Rewrite the insertion routine for the modified data structure.

6. Block structure. *(20 points)*

Implementation of certain features for block structured languages can cause some serious difficulties. Write a short paragraph or two in response to each of the following questions:

(a) Algol W implements records and references but places only references on the stack, while records are kept in separate storage not a part of the stack. Why is this?

(b) It is unusual to find an interactive Algol system which allows a user to modify his program incrementally, that is, adding and deleting text at will, recompiling as little of the program as possible. Why is it so difficult to build such a system? Cite several specific problems that you can think of.

NUMERICAL ANALYSIS

7. Euler's method. *(30 points)*

Consider the ordinary differential equation

$$\frac{dy}{dt} = \lambda y, \ t \geq 0, \ y(0) = y_0.$$

(a) Define Euler's method for this equation.

(b) Let $v_n = v(t_n)$, $t_n = nk$, $n = 0, 1, \ldots, k > 0$, be the approximation to $y(t_n)$. Solve the Euler difference equation and show that $v_n = y_0 \exp(n\lambda k - n\lambda^2 k^2/2 + O(k^3))$. *Hint:* $\ln(1+x) = x - x^2/2 + x^3/3 - \ldots, -1 < x \leq 1$.

(c) Using the expression obtained in (b), compute an expression for $|v_n - y(t_n)|$. What can you conclude from this?

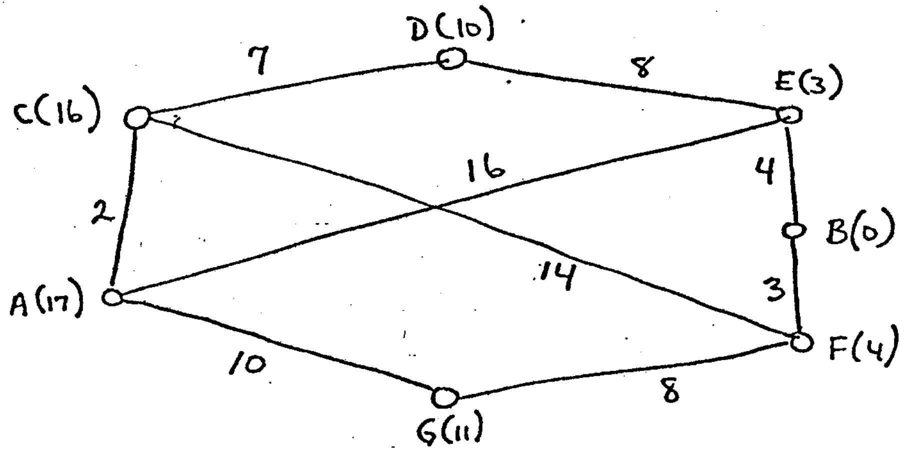
(d) Let $\lambda = \lambda_1 + i\lambda_2$, $i = \sqrt{-1}$, λ_1 and λ_2 real, $\lambda_1 \leq 0$. Then $|y(t)| \leq |y_0|$. Derive a relation for k which is both necessary and sufficient so that $|v_n| \leq |y_0|$.

(e) Discuss the usefulness of Euler's method for integrations over long t-intervals if $\lambda_1 = 0$. What if $|\lambda_1|$ is very much smaller than $|\lambda_2|^2$? Are these conclusions valid if one is only interested in the solution over a very short t-interval?

ARTIFICIAL INTELLIGENCE

8. Search. *(15 points)*

The U. S. Postal Service has developed a simple model to help speed the mail. The model deals with 7 cities and the bi-directional mail shipment between them. The following graph represents the model:



The number on an arc is the actual transit time of mail traveling that route. The number in parentheses at each city is an *estimate* of the transit time from that city to city *B*.

(a) Use the A^* algorithm (Nilsson) to "search" for the apparent shortest path from *A* to *B*. Use the estimates at the cities for the heuristic estimate of distance to the goal. Illustrate your application of the algorithm so that the grader can understand it.

(b) Does A^* find the actual shortest path? If not, why not?

9. Trends. *(15 points)*

Nilsson's survey article on artificial intelligence suggests that a major and fundamental shift in approach to the AI core problem areas has occurred in recent years. Describe and discuss this shift, giving examples from the AI "application" areas if you wish.

THEORY OF COMPUTATION

10. Turing machine implementation. *(20 points)*

(a) Describe the finite control for a Turing machine that recognizes

$$* \, a^n b^{(2^n)} \, *$$

given these constraints:

1. The alphabet of the machine is {", a, b, x}.
2. The symbol x will never appear on the input a.
3. The contents of the tape at completion may be anything.
4. The head begins on the lefthand *.
5. $n \geq 0$.

First explain how your implementation works, in English. Then give a precise description of the machine, using some Turing machine formalism (e.g. quadruples or flow graphs). Be sure to mention how the TM indicates success or failure..

(b) Can a NDPDA (non-deterministic push down automaton) recognize this language? Why or why not?

(c) Is this language context sensitive?

11. Propositional calculus. *(10 points)*

The following formula in the propositional calculus is either a tautology, satisfiable but not a tautology, or unsatisfiable. Determine which and prove your answer.

$$\{[(a \supset b) \supset (c \equiv d)] \vee [(\sim b \supset \sim a) \supset ((\sim c \vee \sim d) \wedge (c \vee d))]\} \wedge [(c \equiv b) \vee [d \equiv (b \vee \sim a)]]$$

PROGRAMMING PROBLEM

A "generalized Gray code" of degree 4 is a cyclic permutation $(p_0, p_1, \ldots, p_{15})$ of the sixteen 4-bit words 0000, 0001, ..., 1111 such that p_i and $p_{(i+1) \bmod 16}$ differ in only one bit position for all i. For example, one such code is

(0000, 0001, 0011, 0010, 0110, 0111, 0101, 0100, 1100, 1101, 1111, 1110, 1010, 1011, 1001, 1000).

Two Gray codes are considered equivalent if one is obtainable from the other by combining any of the following four operations:

(a) Reversal, i.e. $(p_0, p_1, \ldots, p_{15})$ is equivalent to $(p_{15}, p_{14}, \ldots, p_0)$.

(b) A cyclic shift, i.e. $(p_0, p_1, \ldots, p_{15})$ is equivalent to $(p_k, p_{k+1}, \ldots, p_{k-1})$.

(c) Complementation of any set of bit positions, i.e. $(p_0, p_1, \ldots, p_{15})$ is equivalent to $(p_0 \oplus x, p_1 \oplus x, \ldots, p_{15} \oplus x)$, where x is any bit pattern and $\oplus$ denotes exclusive or.

(d) Permutation of bit positions, using the same permutation on each of the p's.

As an example of operations (c) and (d), we can replace each bit pattern $abcd$ by $cadb \oplus 0110 = c\bar{a}\bar{d}b$ in the generalized Gray code shown above, obtaining

(0110, 0100, 1100, 1110, 1111, 1101, 0101, 0111, 0011, 0001, 1001, 1011, 1010, 1000, 0000, 0010).

Write a well-structured computer program which lists all of the *distinct* generalized Gray codes of degree 4, giving for each equivalence class the code which is first in lexicographic order.

For example, it is not difficult to work out the case of degree 3 by hand, obtaining

(000, 001, 011, 010, 110, 111, 101, 100)

as the unique answer!

For degree 4, there are 9 solutions, and they can be found in McCluskey, *Introduction to Logic Design and Switching Theory*, 1965, page 62. This information will help you determine whether your program is correct; obviously it is not sufficient for you to simply list the solutions. You should try to make your algorithm as efficient as possible, making use of symmetry and other work-saving ideas.

All programs must be written in ALGOL W or SAIL. You will be given an account with $100. Programs will be evaluated according to the criteria of efficiency, clarity, documentation, correctness of results, structure, and the algorithm used in the program. Include the program run-time in your documentation.

Winter 1975 Comprehensive Exam

HARDWARE

1. Counter circuit. *(20 points)*

The four binary outputs of a circuit at time t represent the binary encoding of an integer $q(t)$ in the range [0,15] . The circuit has a single binary input. On receipt of an input signal, the circuit generates

$$q(t + dt) = 3 \times q(t) + 1 \pmod{16}$$

where $\times$ and + imply multiplication and addition over the integers. The initial state is $q(0) = 0$.

Design such a circuit using not more than three memory elements. Include brief descriptions of the actions of any circuit elements you employ.

2. Prime number circuit. *(20 points)*

(a) Given four binary signals which encode the numbers 0 through 15 in an 8-4-2-1 code, construct a circuit whose output is 1 if and only if the number represented is prime. (The integer 1 is not prime.) Assume that AND gates, OR gates, and inverters are available. The circuit should require a minimal number of gates.

(b) Pick one of the leads to one of the gates. Assume that this lead breaks in such a way that the gate thinks the corresponding input is always a 1. What is the behavior of the circuit now? Give a systematic procedure for detecting which gate is involved, under the restriction that you can only look at the final output of the circuit.

PROGRAMMING LANGUAGES

3. Precedence grammar. *(10 points)*

Show that the grammar

$<S> ::= a<A>d \mid b<A>d$
$<A> ::= c \mid c<A>$

is an operator precedence but not a simple precedence grammar. Suggest a change to the grammar which leaves the strings generated by S unchanged but gives the grammar the simple precedence property.

4. Parameter passing. *(15 points)*

Consider an imaginary language in which calls can be by value, reference, or name, as specified in the procedure definition. Simulate the output of the following program. Assume the obvious conventions for print format, etc. (e.g. your simulation should begin:

```
1  X=4 W=225
2  X=4 W=229 )
```

```
BEGIN
INTEGER X, INTEGER W, INTEGER I
MACRO SPEAK
   BEGIN
   I := I+1
   PRINT (I, "X= ", X, "W= ", W)
   END
PROCEDURE ONE (W INTEGER VALUE)
   BEGIN
   INTEGER X
   X := 7
   SPEAK
   W := X + W
   X := W
   SPEAK
   END
PROCEDURE TWO (W INTEGER REFERENCE)
   BEGIN
   INTEGER X
   X := 7
   SPEAK
   W := X + W
   X := W
   SPEAK
   END
PROCEDURE THREE (W INTEGER NAME)
   BEGIN
   INTEGER X
   X := 7
   SPEAK
   W := X + W
   X := W
   SPEAK
   END

I := 0
X := 4
W := 225
SPEAK
W := X + W
SPEAK
ONE(X)
SPEAK
TWO(X)
SPEAK
THREE(X)
SPEAK
END
```

5. Error handler. *(20 points)*

You are maintaining an implementation of a compiled high-level block-structured programming language in which the blocks have names. (SAIL is an example where the names are optional.) It has been decided that the run-time error handler should print out the names of the blocks which constitute the lexical (static) scope of the statement in which an error occurs. The *only* information immediately available to the error handler is the value of the program counter at the place where the error occurred.

(a) What additional information must the compiler make available to the error handler?

(b) Design a suitable data structure for the storage of the information.

(c) Give an algorithm for the error handler to use for printing the block names.

Example:

```
BEGIN "problem"  COMMENT block names in quotes;
   PROCEDURE FIRST (REAL VALUE X; INTEGER VALUE IO);
   BEGIN "part 1"
      BEGIN "case 16a"  INTEGER I,J,K;
      WHILE TRUE DO
         IF SQRT(X) > 10 THEN ...
      END "case 16a";...
   END "part 1";
   PROCEDURE SUPERVISE; BEGIN "supe"
      FIRST (-15.,26);...
   END "supe";
SUPERVISE;
END "problem";
```

The error message should include

```
SQRT: negative argument
Block structure is  problem
                    part 1
                    case 16a
```

SYSTEMS

6. Storage reclamation. *(10 points)*

Briefly describe the advantages and disadvantages of the following two schemes for storage reclamation:

(a) garbage collection
(b) reference counts.

Assume the following: the data structures involved are large and cannot be wholly contained in core; a virtual store is provided by a paging system; core space is at a premium; disk accesses are expensive, but computational power is fairly cheap; the storage reclamation problem arises in the context of a LISP system which normally includes only a small number of self-referential structures.

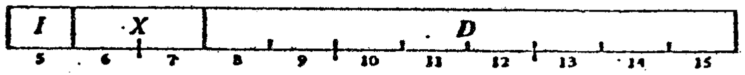
7. Breakpoints. *(20 points)*

The documentation below describes the memory reference instructions of a minicomputer (the Data General Nova).

Describe the design of a mechanism for inserting breakpoints in programs in such a machine. What are the key issues? What must the code that is executed when a breakpoint is encountered do? How about the code that executes the return from a breakpoint?

2.1 MEMORY REFERENCE INSTRUCTIONS

Bits 5–15 have the same format in every memory reference instruction whether the effective address is used for storage or retrieval of an operand or to alter program flow. Bit 5 is the indirect bit, bits 6 and 7 are the

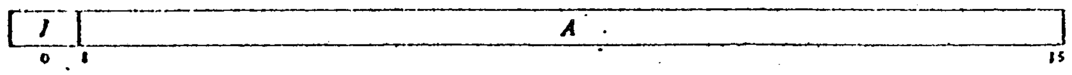


index bits, and bits 8–15 are the displacement. The effective address *E* of the instruction depends on the values of *I*, *X*, and *D*. If *X* is 00, *D* addresses one of the first 256 memory locations, *ie D* is a memory address in the range 00000–00377. This group of locations is referred to as page zero.

If *X* is nonzero, *D* is a displacement that is used to produce a memory address by adding it to the contents of the register specified by *X*. The displacement is a signed binary integer in twos complement notation. Bit 8 is the sign (0 positive, 1 negative), and the integer is in the octal range −200 to +177 (decimal −128 to +127). If *X* is 01, the instruction addresses a location relative to its own position, *ie D* is added to the address in PC, which is the address of the instruction being executed. This is referred to as relative addressing. If *X* is 10 or 11 respectively, it selects AC2 or AC3 as a base register to which *D* is added.

X	*Derivation of address*
00	Page zero addressing. *D* is an address in the range 00000–00377.
01	Relative addressing. *D* is a signed displacement (−200 to +177) that is added to the address in PC.
10	Base register addressing. *D* is a signed displacement (−200 to +177) that is added to the address in AC2.
11	Base register addressing. *D* is a signed displacement (−200 to +177) that is added to the address in AC3.

If *I* is 0, addressing is direct, and the address already determined from *X* and *D* is the effective address used in the execution of the instruction. Thus a memory reference instruction can directly address 1024 locations: 256 in page zero, and three sets of 256 in the octal range 200 less than to 177 greater than the address in PC, AC2 and AC3. If *I* is 1, addressing is indirect, and the processor retrieves another address from the location



specified by the address already determined. In this new word bit 0 is the indirect bit: bits 1–15 are the effective address if bit 0 is 0; otherwise they specify a location for yet another level of address retrieval. This process continues until some referenced location is found with a 0 in bit 0; bits 1–15 of this location are the effective address *E*.

Move Data Instructions

These two instructions move data between memory and the accumulators. In the descriptions of all memory reference instructions, *E* represents the effective address.

LDA **Load Accumulator**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	1	*A*		*I*	*X*		*D*							

Load the contents of location *E* into accumulator *A*. The contents of *E* are unaffected, the original contents of *A* are lost.

STA **Store Accumulator**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	0	*A*		*I*	*X*		*D*							

Store the contents of accumulator *A* in location *E*. The contents of *A* are unaffected, the original contents of *E* are lost.

Modify Memory Instructions

These two instructions alter a memory location and test the result for a skip. They are used to count loop iterations or successively modify a word for a series of operations.

ISZ **Increment and Skip if Zero**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	0	*I*	*X*		*D*							

Add 1 to the contents of location *E* and place the result back in *E*. Skip the next instruction in sequence if the result is zero.

DSZ **Decrement and Skip if Zero**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	1	1	*I*	*X*		*D*							

Subtract 1 from the contents of location *E* and place the result back in *E*. Skip the next instruction in sequence if the result is zero.

Jump Instructions

These two instructions allow the programmer to alter the normal program sequence by jumping to an arbitrary location. They are especially useful for calling and returning from subroutines.

JMP **Jump**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	*I*	*X*		*D*							

Load *E* into PC. Take the next instruction from location *E* and continue sequential operation from there.

JSR **Jump to Subroutine**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	1	*I*	*X*		*D*							

Load an address one greater than that in PC into AC3 (hence AC3 receives the address of the location following the JSR instruction). Load *E* into PC. Take the next instruction from location *E* and continue sequential operation from there. The original contents of AC3 are lost.

NOTE: The effective address calculation is completed before PC+1 is loaded into AC3. Thus a JSR that specifies AC3 as a base register does execute properly; *ie* the previous contents of AC3 are used in the address calculation.

NUMERICAL ANALYSIS

8. Order of convergence. *(15 points)*

We wish to solve the equation

$$f(x) = 0$$

by the following iteration formula:

$$x_{n+1} = x_n + \alpha \frac{f(x_n)}{f'(x_n)} + \beta \frac{(f(x_n))^2 f''(x_n)}{(f'(x_n))^3} .$$

(a) Determine the constants α and β such that the convergence will be of highest possible order.

(b) Give the order.

9. Predictor-corrector. *(20 points)*

For integrating the differential equation

$$y' = f(x, y) , \qquad y(x_0) = y_0$$

the following predictor-corrector pair of formulas has been proposed:

$$\text{P:} \quad y_{n+1} = y_n + \frac{h}{12}(23y'_n - 16y'_{n-1} + 5y'_{n-2})$$

$$\text{C:} \quad y_{n+1} = -y_n + 2y_{n-1} + \frac{h}{4}(y'_{n+1} + 8y'_n + 3y'_{n-1}) .$$

Note that the first is just the Adams-Bashforth predictor of order 3. It can be shown that the second formula is also of order 3. Hence, both formulas would give correct results if $y(x)$ were a polynomial of degree 3 or less.

(a) The following table was obtained by solving

$$y' = y , \qquad y(0) = 1$$

using $h = 0.1$. For each step (except the first two) the predictor was used and the corrector was used sufficiently often to obtain convergence (i.e. agreement to two decimals). Carry this solution forward one more step.

x	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
y	1.00	1.11	1.22	1.36	1.48	1.68	1.77	2.12	

(b) Explain the behavior of this solution. (*Hint:* Study the stability of the corrector formula.)

ARTIFICIAL INTELLIGENCE

10. Representation. *(15 points)*

Two aphorisms in Kernighan and Plaugher are "Say what you mean, simply and directly", and "Choose a data representation that makes the problem simple". These are simple statements of key issues in A.I., but mask some interesting and difficult problems. Here you are asked to represent a problem in two ways and briefly describe the issues involved.

The problem: A robot truck driver, starting at Universal Plastics, Inc., must deliver baubles, bangles, and beads to Gump's, Macy's, and Saks', respectively.

(a) Predicate calculus representation

Assuming the simple operators *drive*(x, y) and *unload*(z) and any others that are necessary, with appropriate preconditions and results, show how to set up (axiomatize) the problem for a theorem prover to find a sequence of operators producing a state satisfying *AT*(baubles, Gump's), *AT*(bangles, Macy's), and *AT*(beads, Saks').

(b) Procedural representation

Write a procedural representation in ALGOL W of the problem solver for accomplishing the same task where the same simple operators are assumed above. Write the procedure to deliver articles to retailers from suppliers, where those names are specified on data cards (the list of the day's jobs), and not necessarily known in advance.

(c) Briefly discuss some of the circumstances under which descriptive and procedural representations seem most appropriate.

11. Combinatorial explosion. *(15 points)*

Donald Michie has written:

"The natural enemy of the worker in the field of artificial intelligence is the 'combinatorial explosion,' and almost his entire craft is concerned with ways of combatting it."

(a) Why is the first clause generally believed to be true? (Be sure to describe what the problem is.)

(b) List and briefly describe the techniques and strategies for heuristic control of search used in A.I. programs to combat the combinatorial explosion.

THEORY OF COMPUTATION

12. Merging. *(20 points)*

Consider the operation of *merging* words together. For example, two words *ab* and *cd* can be merged in six different ways giving *abcd*, *acbd*, *acdb*, *cabd*, *cadb*, and *cdab*. In general, a merge of $x \in \Sigma^*$ and $y \in \Sigma^*$ is a word of length $|x|+|y|$ with both x and y as disjoint subsequences in it.

For two languages L_1 and L_2 their merge is defined as the set of all possible merges of two words $x \in L_1$, $y \in L_2$.

Give the basic ideas of the constructions you would use to prove that

(a) The family of regular languages is closed under merging.

(b) The family of context-free languages is closed under merging with regular languages.

(c) The family of context-sensitive languages is closed under merging.

13. Grammar with *a*'s and *b*'s. *(20 points)*

Give an effective algorithm to determine, for an arbitrary context-free grammar, whether all words generated by it contain an equal number of occurrences of the letters *a* and *b*.

PROGRAMMING PROBLEM: A Scene Recognition System

Three dimensional scenes are to be analyzed for the presence or absence of objects.

We will decompose the scene as follows. Assume the space has dimensions A units by B units by C units. The result is a parallelepiped composed of $A \times B \times C$ unit cubes. Assume that we represent the space by a three dimensional array P. If an object in the scene occupies a cube at (a, b, c) then $P(a, b, c) = 1$ and otherwise $P(a, b, c) = 0$. Thus an "arch" in a space of 4x3x2 might occupy the following positions:

section at $y = 2$	1110 1110
section at $y = 1$	1010 1010
section at $y = 0$	1010 1010.

The scene recognition system will search the space for interesting objects by picking a set of coordinates and inquiring whether that unit cube is occupied or not. The system will attempt to fill out the shape of the object by retrieving other (in general, adjacent) unit cubes to see if they are also occupied. The first part of your task is to design a data structure to represent arbitrary scenes, given that:

(1) P will normally be a sparse array with a high degree of clustering. In other words, the number of occupied unit cubes will be only a small fraction of the total number of such cubes. Furthermore, if a unit cube is occupied, then its six neighbors are very likely to also be occupied. The kind of scene your program will be asked to analyze will normally consist of a few connected objects with regular geometric shapes.

(2) The computer memory is arranged in a two level hierarchy. For any given scene, assume that a significant portion of the data structure that represents your scene must reside in secondary memory. An automatic paging system is used that implements a least recently used page replacement algorithm. The page size is PAGESIZE words. Assume that INCOREPAGES pages of your data structure can be core resident at any one time. (Note that your program will have to simulate the behavior of the paging system in order to properly account for the paging faults it incurs.)

Devise a retrieval algorithm for your data structure. The algorithm should take a coordinate triple (x, y, z) as a parameter and return $P(x, y, z)$.

For the next part of the problem, the scenes to be analyzed are assumed to contain a single object which is a supercube of size $S \times S \times S$ unit cubes. This supercube is randomly placed in the scene space, but is assumed to be aligned with the x, y, and z-axes. Inside the supercube is a spherical cavity. The center of the sphere coincides with one corner of one of the unit cubes, and the radius of the sphere, R, is an integer. The sphere is completely contained but randomly placed within the supercube, and is not tangent to any side of the supercube. We assume that a unit cube is fully contained in the sphere if its interior and the interior of the sphere have a point in common. The task of the scene recognizer is to determine the total volume of the supercube minus the volume of the sphere.

Thus the assignment has six parts:

(1) Describe the data structures you intend to use.

(2) Program the unit cube retrieval algorithm for an arbitrary scene. Comment on the expected performance of your algorithm as a function of the scene it is presented with.

(3) Assume that your scene consists of a supercube SxSxS placed with one corner at (IX,IY,IZ). This supercube contains a spherical hole of radius R centered at (SX,SY,SZ). Generate the data structure representing this scene and compute the number of pages it occupies.

(4) Using the supplied random number generator, select N unit cubes in the AxBxC space. Test the retrieval algorithm of part (2) by determing whether these N unit cubes are occupied or not. Compute the average number of page faults per unit cube retrieved.

(5) Assume that the sphere inside the cube is empty. Write an algorithm for computing the volume of the hollow supercube. In doing this the only way you can obtain information about the scene space is by using the unit cube retrieval program of part (2). Program the algorithm so that it prints out the number of unit cubes examined and the total number of page faults incurred, as well as the volume.

(6) Run your program on the test data given below and print out the quantities described above.

Test data:

```
A = B = C = 255
S = 21
IX = 23, IY = 33, IZ = 47
SX = 31, SY = 42, SZ = 55
R = 5
N = 1000
PAGESIZE = 64
INCOREPAGES = 10
```

Your programs will be graded on the criteria of correctness, clarity, and appropriateness of the data structures. Your data structures should try to minimize the following measures of cost: size of the structures; number of page faults incurred; number of unit cubes examined during the computation of the volume; and CPU time. Note: you should not use language features involving use of storage that you cannot account for in your paging simulation.

Spring 1975 Comprehensive Exam

SYSTEMS

1. Deadlock. *(5 minutes)*

Three parallel processes share four tape units which can be reserved or released only one at a time. Each process needs at most two tape units. Can a deadlock occur in this system?

2. LISP and systems. *(10 minutes)*

(a) The bindings of variables in a LISP program act quite differently from those in ALGOL. Describe briefly the basic difference in the philosophy of the two approaches to determining binding.

(b) How would a paging environment influence your implementation of the CONS (structure building) function?

3. Paging concepts. *(20 minutes)*

(a) A process refers to five pages, A, B, C, D, E, in the following order: A B C D A B E A B C D E.

(1) Assume the system uses a FIFO replacement algorithm. Find the number of page faults starting with an empty store and 3 page frames. Answer the same question for 4 page frames.

(2) Repeat part (1) assuming a LRU replacement algorithm.

(b) Using the hardware mechanisms of current machines that support paging, describe a practical approximation to an LRU page replacement algorithm.

(c) Under what circumstances will page sharing improve the performance of a paged multiprogramming system?

4. Compiler quickies. *(20 minutes)*

(a) Discuss three ways of reducing the storage requirements needed for using precedence techniques.

(b) Give an example of why a compiler writer may want to combine two different parsing techniques in one compiler.

(c) Can precedence functions be assigned to this matrix? (Justify your answer.)

$$\begin{array}{c|cc} & S_1 & S_2 \\ \hline S_1 & = & > \\ S_2 & > & = \end{array}$$

(d) What are the pros and cons of typed and typeless languages? Give examples of each.

5. <u>Process synchronization</u>. *(5 minutes)*

Describe a primitive process synchronization mechanism.

HARDWARE

1. <u>Combinational circuit problem</u>. *(20 minutes)*

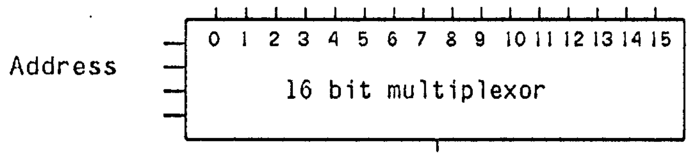
The boolean Fibonacci function f is defined to be TRUE for the binary inputs x_1, x_2, x_3, x_4 corresponding to {1, 2, 3, 5, 8, 13} and FALSE otherwise except that we don't care what value it has for inputs {7, 9}.

(a) Give the minimal sum of products expression for f.

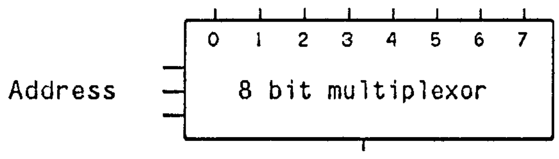
(b) Give the minimal product of sums expression for f.

A multiplexor is a circuit that can select information from one of several input channels and route that input to a single output lead. The input terminal is selected by a binary encoded address supplied to the circuit. A multiplexor can be used as a universal switching function by connecting a fixed 1 or a fixed 0 to each of its inputs.

(c) Implement f on this 16 input multiplexor.

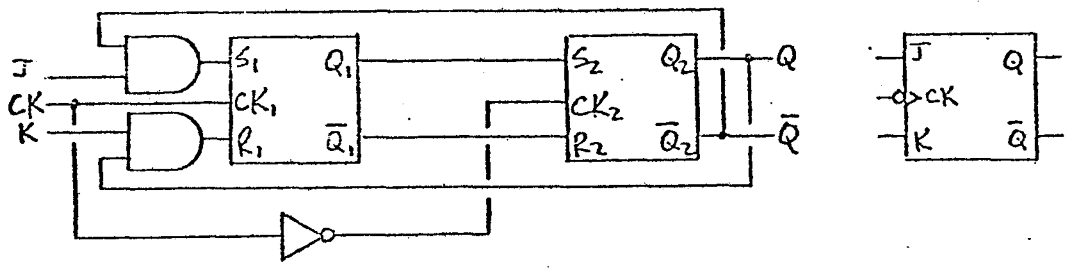


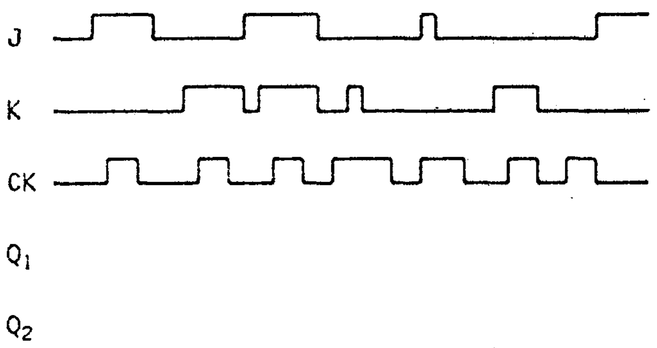
(d) Can you implement f using only this single 8-input multiplexor and inverters? (If yes, do it.)



2. J-K flip-flop. *(10 minutes)*

A logic diagram for a J-K master-slave flip-flop can be drawn in terms of two S-R latches as shown below. Draw the Q_1 and Q_2 outputs that are obtained from the J, K, and CK inputs shown.





3. Binary counter. *(15 minutes)*

Give the logic diagram of a 4-bit synchronous binary counter using J-K master-slave flip-flops. (In a synchronous counter all of the output bits must change simultaneously.)

4. Architecture concepts. *(15 minutes)*

Slow machines usually do things sequentially which faster machines can do at least partially in parallel. Explain briefly what each of the following terms mean and how their use can increase a computer's execution rate.

(a) multiprocessing
(b) memory interleaving
(c) wide buses
(d) associative memory

THEORY OF COMPUTATION

1. Diagonalization problem. *(30 minutes)*

Diagonalization is a technique made famous when Cantor proved that the real numbers are not countable. The technique assumes a list of items and establishes the existence of a new item which differs in some way from each item in the list and is therefore not in the list.

(a) Use diagonalization to show that there can be no effective enumeration of the total recursive functions (defined for all inputs). (*Hint*: Start by assuming that there is such an enumeration.)

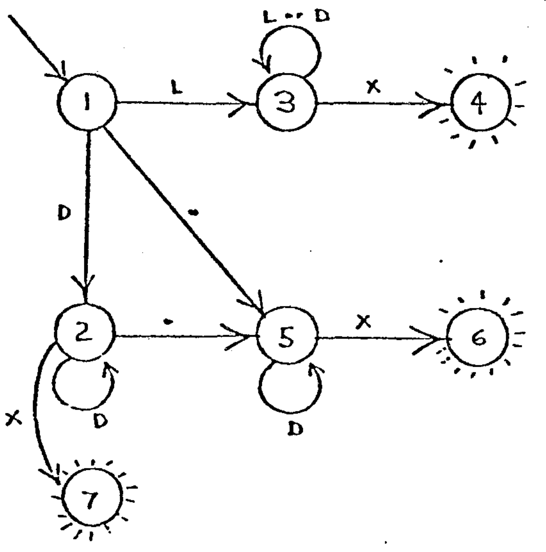
(b) It is possible to give a listing $\{C_1, C_2, C_3, \ldots\}$ of all context-sensitive grammars. Show that there exists a recursive set which is not context sensitive.

2. Finite automata. *(20 minutes)*

Consider the finite state machine M below whose inputs can be interpreted as

L -	Letter
D -	Digit
X -	Delimiter
. -	Decimal point

and where F_M is the set of final states:



$F_M = \{4, 6, 7\}$

(a) Write a regular expression for the strings which end at state 6. For example, the regular expression $\{0,1\}^*\{000\}\{0,1\}^*$ denotes the set of strings over the set $\{0,1\}$ having three consecutive 0's.

(b) Construct a *nondeterministic* machine which recognizes the reversal of the language recognized by M. (Draw its state diagram.)

(c) Construct a *deterministic* machine which recognizes the reversal of the language recognized by M.

(d) Relate this problem to lexical analysis.

3. Context-free language quickies. *(10 minutes)*

Consider the language defined by the grammar

$$\begin{array}{lll} S \to a & A \to b & B \to AaS \\ S \to AbB & A \to SBc & B \to aSA \end{array}$$

with start symbol S and terminal alphabet $\{a, b, c\}$.

(a) Prove that the language contains no strings of length 100.

(b) Find all strings in the language of length 9, having the letter "c" as their middle (fifth) character.

ALGORITHMS & DATA STRUCTURES

1. Algol 60 problem. *(30 minutes)*

The original ALGOL 60 report concluded with the example procedure RK shown on the following page. After fifteen years have gone by, we think we know a little more about control structure and style.

Suppose ALGOL 60 has been extended to include the following new syntax:

```
<basic statement> ::= <loop statement>
<loop statement>  ::= loop <statement list> while <Boolean expression>: <statement list> repeat
<statement list>  ::= <statement> | <statement list>; <statement>
```

and the semantics that, if S and T are statement lists and B is a Boolean expression, the loop statement

loop S **while** B: T **repeat**

is equivalent to

```
S;
if B then begin
  T;
  loop S while B: T repeat
end
```

```
procedure RK(x, y, n, FKT, eps, eta, xE, yE, fi); value x, y; integer n;
Boolean fi; real x, eps, eta, xE; array y, yE; procedure FKT;
comment: RK integrates the system y'_k = f_k(x, y_1, y_2, ..., y_n) (k = 1, 2, ... n)
of differential equations with the method of Runge-Kutta with automatic search for
appropriate length of integration step. Parameters are: The initial values x and y[k]
for x and the unknown functions y_k(x). The order n of the system. The procedure
FKT(x, y, n, z) which represents the system to be integrated, i.e. the set of functions f_k.
The tolerance values eps and eta which govern the accuracy of the numerical integra-
tion. The end of the integration interval xE. The output parameter yE which re-
presents the solution at x = xE. The Boolean variable fi, which must always be
given the value true for an isolated or first entry into RK. If however the functions y
must be available at several meshpoints x_0, x_1, ..., x_n, then the procedure must be
called repeatedly (with x = x_k, xE = x_{k+1}, for k = 0, 1, ..., n-1) and then the later
calls may occur with fi = false which saves computing time. The input parameters
of FKT must be x, y, n, the output parameter z represents the set of derivatives z[k] =   α
f_k(x, y[1], y[2], ..., y[n]) for x and the actual y's. A procedure comp enters as a
non-local identifier;
begin
    array z, y1, y2, y3[1:n]; real x1, x2, x3, H; Boolean out;
    integer k, j; own real s, Hs;
    procedure RK1ST(x, y, h, xe, ye); real x, h, xe; array y, ye;
        comment: RK1ST integrates one single Runge-Kutta step with initial
        values x, y[k] which yields the output parameters xe = x + h and ye[k],
        the latter being the solution at xe.
        Important: the parameters n, FKT, z enter RK1ST as non-local entities;
        begin
              :  (omitted)
         end RK1ST;
Begin of program:
    if fi then begin H := xE - x; s := 0 end else H := Hs;                   β
    out := false;
AA: if (x + 2.01 × H - xE > 0) ≡ (H > 0) then
    begin Hs := H; out := true; H := (xE - x)/2 end if;                      γ
    RK1ST(x, y, 2 × H, x1, y1);
BB: RK1ST(x, y, H, x2, y2); RK1ST(x2, y2, H, x3, y3);
    for k := 1 step 1 until n do
        if comp(y1[k], y3[k], eta) > eps then go to CC;

    comment: comp(a, b, c) is a function designator, the value of which is the
    absolute value of the difference of the mantissae of a and b, after the exponents   δ
    of these quantities have been made equal to the largest of the exponents of the
    originally given parameters a, b, c;

    x := x3; if out then go to DD;

    for k := 1 step 1 until n do y[k] := y3[k];                               ε
    if s = 5 then begin s := 0; H := 2 × H end if;
    s := s + 1; go to AA;
CC: H := 0.5 × H; out := false; x1 := x2;                                     ζ
    for k := 1 step 1 until n do y1[k] := y2[k];
    go to BB;
DD: for k := 1 step 1 until n do yE[k] := y3[k]                               η
end RK
```

Rewrite the *RK* procedure using the new loop statement, eliminating unnecessary go to statements which tend to obscure the control structure of the program in its original form.

Important note: Large segments of the program have been marked α, β, γ, etc. so that you need not recopy them; simply give your answer in terms of these abbreviations. You need not understand precisely what goes on inside those chunks of code in order to solve this problem. Don't make any changes to the actual sequence of computations, and don't try to do any tricky things like eliminating the Boolean variable "*out*". Simply remove as many of the go to statements and meaningless labels as you can by taking advantage of the new loop construct.

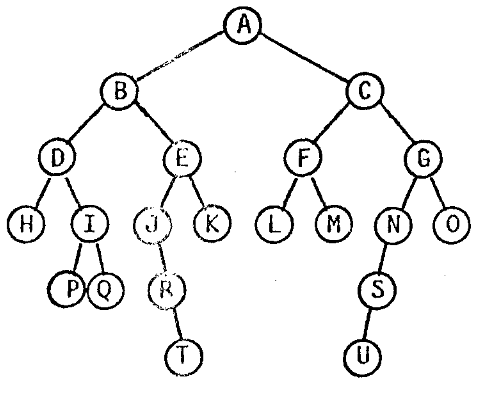
2. Tree traversal. *(10 minutes)*

Traversal of a binary tree in comprehensive-qual order (CQ order) is defined as follows.

```
procedure CQ(t); binary tree t;
  if t ≠ Λ then begin
    if right(t) ≠ Λ then begin
      CQ(left(right(t))); visit(t); CQ(left(t));
      visit(right(t)); CQ(right(right(t)))
    end else
    begin visit(t); CQ(left(t)) end
  end.
```

Here Λ denotes a null binary tree, and *left*(*t*), *right*(*t*) denote respectively the left and right subtrees of a non-null binary tree *t*.

In what order are the nodes of the following binary tree visited, when it is traversed in CQ order? (State the single-letter names of the nodes as they are encountered.)



3. Sparse matrix. *(20 minutes)*

What data structure would you use to represent a sparse 100×100 matrix (1 word entries), if you are told that it contains

(a) At most 5 non-zero entries
(b) At most 50 non-zero entries
(c) At most 500 non-zero entries
(d) At most 5000 non-zero entries.

The operations you will have to do on the matrix are

(1) fetch element $A[i,j]$, given i and j
(2) store element $A[i,j]$, given i and j
(3) fetch all non-zero elements of a given row
(4) fetch all non-zero elements of a given column .

NUMERICAL ANALYSIS

1. Banded linear systems. *(10 minutes)*

It is well known that linear systems $A\underset{\sim}{x} = \underset{\sim}{b}$ can be solved using Gaussian elimination *without pivoting* if the matrix A has certain properties (e.g. A is irreducibly diagonally dominant or A is positive definite). Explain why this is an important practical consideration when dealing with banded matrices $A = (a_{ij})$, $a_{ij} = 0$ if $|i-j| > m$, say, where m is small compared with n, the rank of the matrix.

2. Arithmetic and error analysis. *(10 minutes)*

(a) Do floating-point and real arithmetic have the same arithmetic properties (e.g. associativity, commutativity)? Prove your point.

(b) What advantage does inverse error analysis have over forward error analysis?

3. The rank problem. *(20 minutes)*

It is easy to verify that

$$\begin{pmatrix} 1 & -1 & -1 & \dots & -1 & 1 \\ & 1 & -1 & \dots & -1 & \frac{1}{2} \\ & & \ddots & & & \frac{1}{4} \\ & & & 1 & -1 & \frac{1}{2^{n-2}} \\ & & & & 1 & \frac{1}{2^{n-2}} \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ \\ 0 \\ \frac{1}{2^{n-2}} \end{pmatrix}$$

(a) What does this indicate about the conditioning of the matrix on the left hand side?

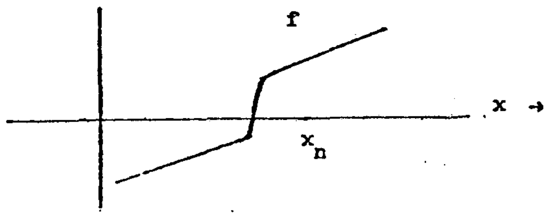
(b) Does the above example say anything about the use of Gaussian elimination to determine the rank of a matrix? (Recall that Gaussian elimination can be viewed as transforming a matrix to upper triangular form.)

4. Newton's method. *(20 minutes)*

Newton's method is one of the best known iterative techniques for evaluating the root of a function. The iterative step of the method is given by

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

(a) The graph below shows a function f near one of its roots. A program using Newton's method has just evaluated f at x_n as indicated. Show geometrically the position for the estimate x_{n+1}.



Can you predict any difficulty that will beset the program during the next few iterations?

(b) Some polynomials, especially those of high degree, are unstable in the sense that small changes in the coefficients will lead to large changes in the zero. Show that the relative root change is given by

$$\frac{\Delta r}{r} \approx \frac{-a_n r^{n-1}}{p'(r)} \frac{\Delta a_n}{a_n}$$

where $p(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_0$ is the polynomial, $p(r) = 0$ (i.e. r is a root), and a_n is the coefficient being changed slightly.

(c) Newton's method is sometimes recommended because of its quadratic convergence. Herman Newfellow has observed that the corrections $p(x_i)/p'(x_i)$ were reduced by a factor of about 1/2 for each of the first 10 iterations of his program. What order of convergence does this indicate? How would you explain this to Herman?

ARTIFICIAL INTELLIGENCE

1. "Understanding systems". *(30 minutes)*

In recent years the A.I. research community has been describing much of its research as "understanding": viz. speech understanding, language understanding, and image understanding.

(a) What A.I. concepts, capabilities, program organizations, representations, and tools are used in "understanding" systems? Which of these are relatively domain specific and which are general? Illustrate and refine your answer by dealing specifically with one of the following areas of program understanding. For the area chosen, concentrate on *one* in detail or survey the few systems which exist in the area. Keep your answers brief and to the point.

(1) English language understanding
(2) speech understanding
(3) organic-chemical mass spectral data interpretation

(b) Suppose we designed an image understanding computer system (ala A.I. vision systems) to ride along on a future ERTS (Earth Resources Technology Satellite) orbiting mission and to radio back to scientists on earth what it "understood" about the scenes being viewed. From the A.I. scientist's point of view, what would be the essential design problem in making the system work?

2. A.I. concepts. *(20 minutes)*

Define each of the following problem-solving concepts and methods, and give a short (one or two sentence) description of the sort of problem domain to which it is applicable. (For instance, "table lookup" might be defined as the selection of data elements from an information structure by processing keys associated with the data. It would be applicable in situations where the set of possible answers is explicit; there are appropriate keys and selection functions; and the number of elements is small enough to allow the storage and retrieval to be reasonably efficient.)

(a) heuristic search
(b) means-end analysis
(c) hill climbing
(d) "demons"
(e) production systems

3. A.I. quickies. *(10 minutes)*

(1) The state-space approach to problem solving is a special case of the problem reduction approach. True or false? Explain. (If true, then why bother dealing with a separate concept of "state-space methods"?)

(2) (True or false). The alpha-beta technique is often faster than the minimax technique but it may achieve a less optimal answer.

(3) It has been suggested that work on programs of a general problem solving nature has been shelved temporarily. Supposing this is correct, what would be the reason for this trend? State your answer briefly.

PROGRAMMING PROBLEM: Simulation of a Packet-switching Network

Computer networks are becoming more widespread and are influencing the way people can use computers. Although networks seem conceptually simple, some aspects of their design were not appreciated until the networks were actually implemented. In this problem you will be able to experiment with some elements of network design as well as learn something about the programming of a simulation.

The basic model for a network that we have in mind is a simplified version of the ARPA network. The network is made up of IMP's, which are minicomputers used for message switching and which are connected to each other by transmission lines. Each IMP is also connected to a HOST, which is a local computer where the user does his actual computations. Each IMP is in fact a computer and is busy managing the message traffic between HOST's. For the purposes of this simulation, we will be mostly concerned about what is going on inside the IMP's and will use a relatively simple model for the activity of each HOST. The traffic on the network consists of variable length messages. A message enters the network at the source (or sending) HOST, which specifies a destination (or receiving) HOST. The message is transmitted through an appropriate set of IMP's and finally leaves the network at a destination HOST.

Messages have the following attributes:

source:	HOST or IMP which sent it
destination:	HOST or IMP to receive it
acknowledge bit:	bit which is set if the message is simply an acknowledgement of a previous message
length:	number of bits in message (specified by sender)
sequence number:	set by sender (32 or 36 bits) -- used to ensure that messages are delivered in the order that they are sent.

For simplicity, we will assume that there are only two sorts of messages:

(1) *Acknowledgement messages*, which have the acknowledge bit set and have length HEADERBITS.

(2) *Data messages*, which come in two flavors:
short (length = HEADERBITS + SHORTBITS) or
long (length = HEADERBITS + LONGBITS).
Short messages occur with probability SHORTPROB and long messages occur with probability (1-SHORTPROB).

Messages are carried on *lines* between IMP's or between an IMP and a HOST. The lines between IMP's carry messages at a fixed rate, LINERATE, in thousands of bits per second. They are full-duplex, that is, they can carry messages simultaneously in both directions (at LINERATE in each direction).

The transmission time of a message is assumed to depend only on the message length. Furthermore, each line connecting an IMP to another IMP is assumed to have a probability LOSSPROB $*$ (length of message) of garbling a message so that it will have to be retransmitted. (Yes, we know this cannot be quite right, but accept it as an approximation.) Error detection circuitry in the receiving IMP will detect broken messages.

The lines connecting an IMP to its HOST are assumed to have zero error and operate at HOSTRATE. In addition, they have a special block signal by which the IMP can prevent the HOST from sending more messages.

HOSTS

For our purposes, assume that each IMP is associated with exactly one HOST.

Each HOST generates messages with a Poisson arrival time distribution. This means that the intervals between the times that the HOST queues messages for transmission to its IMP are exponentially distributed. These may be generated for the simulation by the relation

$$T_{n+1} = T_n - \text{TLAMBDA} * \ln(r)$$
$$T_0 = 0$$

where r is uniformly distributed on (0,1), and TLAMBDA = mean interarrival time.

No HOST sends a message to itself. The distribution of each message from a HOST is equal to the destination of the previous message sent from the HOST with probability SAMEPROB; otherwise, the destination is randomly chosen from among the HOST's other than the sending HOST, each with equal probability (including the HOST of the previous message).

As messages are generated, the HOST places them into an output queue to its corresponding IMP. The IMP accepts messages from the HOST at rate HOSTRATE unless the IMP is blocked (in which case, the messages just remain in the HOST's output queue). If the HOST runs out of buffer space, it stops generating messages until a buffer becomes free.

If an IMP blocks a HOST while a transfer (into the IMP) is in progress, that transfer will be completed, but no new HOST-to-IMP transfers will be started until the IMP unblocks the HOST.

We assume that the HOST is always able to accept a message from its IMP at the HOSTRATE. We will not simulate what the HOST does with the messages it receives.

IMPS

The IMP is the basic message switching processor of the system. Each IMP is connected to some subset of the set of other IMP's as determined by the topology of the network.

When an IMP accepts a message from its corresponding HOST, it adds a sequence number telling the order (first, second, etc.) of that message in the stream from the source to the destination HOST. In other words, there is a separate set of sequence numbers for each HOST-HOST pair.

Internally, each IMP has a *route control table* by which it determines how to route a message. This is simply a table of line numbers indexed by destination IMP or HOST.

Every IMP maintains a buffer pool from which it allocates buffers for its various processes. For simplicity, each buffer is assumed to be long enough to contain one maximum length message.

An IMP tries to keep a buffer available on each of its lines ready to receive input. When an ungarbled message arrives, and there is a free buffer available, the IMP processes it by doing the following:

(1) Send an acknowledgement. (This does not necessarily require a new buffer to create the acknowledgement message.)

(2) If the message is for its HOST, put the message on the output queue for transmission to the HOST. Otherwise, look up the destination line in the route control table and put the message on the output queue for that line.

If there is no buffer available when a message arrives, the message is lost.

After a message has been sent off (to another IMP), the IMP cannot reclaim the message buffer until an acknowledgement has been received from the IMP at the other end of the line. If no acknowledgement has been received for TIMEOUT seconds, the IMP should retransmit the message.

The IMP's processing time to allocate buffers, etc., is assumed to be zero.

Note: Since some messages get delayed in the network due to line loss and retransmission delays, messages may arrive for an IMP's HOST out of sequence order. An IMP must deliver messages to its HOST in correct sequence order. (This implies that the destination HOST never receives the same message twice.)

The problem

We have not specified here the order and the timing by which the IMP sends messages. Part of your design problem is to work out a scheme that avoids excessive delay. Be sure to explain your scheme in the documentation.

You are to build a simulator and simulate the performance of a sample network (see below) from time zero until a simulated time DEADLINE. At time zero, all buffers and queues are empty. Each HOST's first message will be to a destination and will occur at time T_1 for that HOST.

(a) Assume that HOSTs and IMPs have an infinite buffer pool.

(b) Assume that each IMP has only IMPSPACE buffers available and every HOST has only HOSTSPACE buffers available.

For parts (a) and (b) above, print out a distribution (histogram) and averages for each of the following quantities:

(1) Average number of occupied buffers per IMP per unit time. (Combine together the statistics for all IMPs, since each will have the same distribution in the network considered.)

(2) Average number of occupied HOST buffers per unit time. (Again, combine together the statistics for each HOST.)

(3) Delay time for messages between each class of HOST-HOST pairs. (In the sample data set, there are just three such classes.) Combine together all statistics within a given class. You should calculate delay time from when the message is created in the source HOST to when transfer into the destination HOST is complete.

Print out these statistics at every REPORTING_INTERVAL of simulated time. Clear out the accumulated totals so that each interval has fresh statistics.

Document your program. In particular, describe and explain your implementation of the following items: (1) representation of HOSTs and IMPs, (2) handling of time in the simulation, (3) data structures used in the program, and (4) IMP buffer allocation schemes.

Have you thought about what you have done? Explain how you would redesign these network algorithms and your implementation of the simulation to overcome peculiarities and difficulties you observed in this problem. Did the test data set fully expose the capabilities and pathologies of your network algorithms?

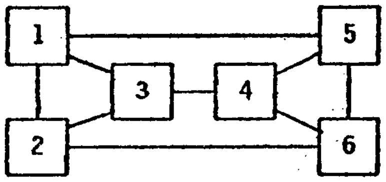
Programs will be graded according to correctness of results, clarity of program and documentation, and appropriateness of data structures. Partial credit will be given for incomplete solutions, but be sure to document fully what you have done, and indicate how you would do the rest.

Test data

```
LINERATE := 50 kb          HOSTRATE := 100 kb       LOSSPROB := .0001 per bit
HEADERBITS := 96           SHORTBITS := 128         LONGBITS := 1000
SHORTPROB := .95           TLAMBDA := .006 sec      SAMEPROB := 0.75
TIMEOUT := .05 sec         IMPSPACE := 25           HOSTSPACE := 10
DEADLINE := 2 sec          REPORTING_INTERVAL := 0.5 sec
```

List of connections

1	2
1	3
1	5
2	3
2	6
3	4
4	5
4	6
5	6



Route control table:

FROM	TO 1	2	3	4	5	6
1	-	2	3	3	5	2
2	1	-	3	3	1	6
3	1	2	-	4	1	2
4	5	6	3	-	5	6
5	1	6	4	4	-	6
6	5	2	4	4	5	-

Classes of pairs

A = { 1↔2, 2↔3, 3↔1, 4↔5, 5↔6, 6↔4 }
B = { 1↔5, 2↔6, 3↔4 }
C = { 1↔4, 1↔6, 2↔4, 2↔5, 3↔5, 3↔6 }

Winter 1976 Comprehensive Exam

SYSTEMS

1. Assembler. *(10 minutes)*

Describe a way in which a one-pass, in-memory assembler could resolve forward references. Assume that the assembler accepts source assembly language and produces, in main memory, an executable image of the program. If the one-pass assembler produces a relocatable object module, how would the forward references be resolved (assuming the object module could not fit in memory)?

2. Operating system. *(5 minutes)*

What is the main job of an operating system? (Answer in at most 2 paragraphs.)

3. Interrupts. *(5 minutes)*

What is an interrupt? What things are typically done by the hardware to service an interrupt? What additional things might typically be done by the operating system software when servicing the interrupt?

4. Linker vs. loader. *(5 minutes)*

What is the distinction between a *linker* (or *link-editor*) and a *loader*?

5. Re-entrance vs. recursion. *(5 minutes)*

What is the difference between re-entrant and recursive code?

6. Multiprogramming vs. multiprocessing. *(10 minutes)*

Are multiprogramming and multiprocessing logically equivalent or is one more complex than the other? In what ways are they the same and in what ways do they differ? As a model of a multiprocessor, consider a collection of autonomous CPU's with shared memory.

7. Paging vs. segmentation. *(10 minutes)*

Distinguish between virtual memories provided by paging and those provided by segmentation. Suppose a library of subroutines is available which is to be shared (i.e. each subroutine is re-entrant, and only one physical copy is to be contained in the main physical store). Which method, paging or segmentation, would offer better flexibility?

8. Cooperating process synchronization. *(5 minutes)*

Describe one way in which concurrent processes can be made to cooperate (i.e. will avoid concurrent execution of so-called "critical" sections).

9. Interpreters vs. compilers. *(5 minutes)*

What is the distinction between an interpreter and a compiler?

NUMERICAL ANALYSIS

1. Numerical arithmetic. *(22 minutes)*

Assume we are given a set of approximately equal numbers $a_1, a_2, \ldots, a_N$, where $N = 2^M$. Define $N_j = N/2^j = 2^{M-j}$.

Consider the following algorithm:

$$\begin{aligned} s(0,k) &= a_k, & 1 \le k \le N \\ s(j,k) &= s(j-1,k) + s(j-1, N_j+k), & 1 \le k \le N_j,\ 1 \le j \le M. \end{aligned}$$

(a) Determine $s(M, 1)$.

(b) Let $t(0,k) = a_k$ for $1 \le k \le N$ and let

$$t(j,k) = fl(t(j-1,k) + t(j-1,N_j+k) = (t(j-1,k) + t(j-1,N_j+k))(1 + \epsilon_{j,k})$$

where $|\epsilon_{j,k}| \le \epsilon$, and ϵ is a prescribed number. (The notation $fl(x+y)$ indicates the floating point addition of x and y.)

Show that

$$t(M,1) = \sum_{n=1}^{N} a_j(1+\eta_j)$$

and give a bound for $|\eta_j|$.

For parts (c) and (d), assume $a_k > 0$.

(c) Give a bound for the relative error

$$\frac{|s(M,1) - t(M,1)|}{|s(M,1)|}.$$

(d) How does the bound given in (c) compare with a bound for the relative error when $s(M, 1)$ is computed in the natural way?

2. Sytems of equations. *(38 minutes)*

We wish to solve the system of equations

$$x = f(x, y)$$
$$y = g(x, y)$$

for x and y. In matrix notation, we have $\underset{\sim}{z} = \underset{\sim}{h}(\underset{\sim}{z})$, where

$$\underset{\sim}{z} = \begin{pmatrix} x \\ y \end{pmatrix}, \quad \underset{\sim}{h}(\underset{\sim}{z}) = \begin{pmatrix} f(x, y) \\ g(x, y) \end{pmatrix} = \begin{pmatrix} f(\underset{\sim}{z}) \\ g(\underset{\sim}{z}) \end{pmatrix}.$$

We seek $\underset{\sim}{z}^*$ such that $\underset{\sim}{z}^* = \underset{\sim}{h}(\underset{\sim}{z}^*)$. Assume that

$$\| \underset{\sim}{h}(\underset{\sim}{z}) - \underset{\sim}{h}(\underset{\sim}{\zeta}) \| \le L \, \| \underset{\sim}{z} - \underset{\sim}{\zeta} \|$$

for all $\underset{\sim}{z} \in R$, $\underset{\sim}{\zeta} \in R$ where R is a subset of $\mathbb{R}^2$ and furthermore $\underset{\sim}{h}(\underset{\sim}{z}) \in R$ if $\underset{\sim}{z} \in R$. For simplicity, we use the maximum norm, which is defined by

$$\left\| \begin{pmatrix} p \\ q \end{pmatrix} \right\| = \max \left(|p|, |q| \right).$$

Assume that $L < 1$.

(a) Show that if we define $\underset{\sim}{z}^{(0)} \in R$ and $\underset{\sim}{z}^{(k+1)} = \underset{\sim}{h}(\underset{\sim}{z}^{(k)})$, then $\underset{\sim}{z}^{(k)} \to \underset{\sim}{z}^*$ as $k \to \infty$.

(b) Derive a bound of the form

$$\| \underset{\sim}{z}^{(k)} - \underset{\sim}{z}^* \| \le M \, \| \underset{\sim}{z}^{(1)} - \underset{\sim}{z}^{(0)} \|$$

and give an explicit expression for M (in terms of L).

(c) Consider the system of equations

$$x + ay = 1$$
$$bx + y = 2$$

with $|a| < 1$, $|b| < 1$.

Show that this system has a solution.

(d) For the system given in (c), consider the iteration defined by

$$x^{(k+1)} = 1 - ay^{(k)}$$
$$y^{(k+1)} = 2 - bx^{(k)}$$

where $x^{(0)}$ and $y^{(0)}$ are arbitrary.

Determine L (as defined above) and prove convergence.

(e) Now consider the iteration defined by

$$x^{(k+1)} = 1 - ay^{(k)}$$
$$y^{(k+1)} = 2 - bx^{(k+1)}.$$

Show that this algorithm converges for an arbitrary choice of $y^{(0)}$.

ARTIFICIAL INTELLIGENCE

Consider the following epistemological problem aboard the starship Enterprise.

Anyone with pointed ears that is aboard the Enterprise is a Vulcan. Vulcans, of course, are rational beings. Furthermore, both Spock and Kirk are on the Enterprise. Finally, Spock has pointed ears. To speculate on whether non-Vulcan, featherless bipeds are truly rational is beyond the scope of this commentary.

1. Formal systems. *(20 minutes)*

Restate the situation in a formal system so that one may infer the existence and identity of a rational being.

(a) Formalize this knowledge in first-order logic.

(b) Draw a relational-structure ("Quillian-like" net) to represent equivalent knowledge in a more visual form.

2. Deductive systems. *(20 minutes)*

There are several ways of implementing a deductive system that can make the inference stated in question (1).

(a) Describe an implementation in one of the new A.I. language systems. Your implementation should use the features of the system to good advantage. Your description may be a precise program in Planner, Micro-planner, QLISP, QA4, Conniver, or Popler; or it may be a more abstract program in your own syntax that clearly shows you understand the essence of the A.I. language features.

(b) Describe an implementation in either LISP or ALGOL-W. In particular describe how you would represent and store your relations and inference mechanisms. The implementation description must be convincing enough so that a typical fellow graduate student could implement it. Spend more than ten minutes at your own risk. For example, your implementation description might read like, "Each letter of the relation will be stored in successive words of the array. We will find a stored relation in our data base by shifting a 'pattern array' along a data array until it matches."

3. Evaluation. *(5 minutes)*

Comment on the relative advantages and disadvantages of the two systems defined in problem 2 above.

4. Strategy. *(5 minutes)*

Does your A.I. system implementation (question 2(a)) work via a depth-first, breadth-first, heuristic progressive deepening (best-first), or other strategy?

5. Extensions. *(10 minutes)*

How (if not there already) could you extend your LISP or ALGOL-W implementation (question 2(b)) to a heuristic progressive deepening (best-first) strategy, assuming an evaluation function is given?

ALGORITHMS

1. Priority queues. *(15 minutes)*

A discrete event simulation, as discussed by Hoare in *Structured Programming*, makes use of a "sequence set". This set contains the events which will take place at specified times, usually known as "event notices". As the simulation proceeds, future notices are filed in the set according to their time of occurrence. Each time the simulator completes the processing of an event notice, it takes the *chronologically next* event from the sequence set of event notices. The sequence set thus forms a *priority queue* of event notices.

(a) What advantage would a binary tree data structure have over a linear list for representing the priority queue of event notices for a large number of pending events?

(b) Suppose that the priority queue is to be represented by a binary tree. Event notices are filed in this tree so that the first node visited by the traversal method selected will be the next event which should occur in the simulation. After an event occurs, its notice is deleted from the tree. We expect that the method of traversal will in some way determine the algorithm for adding event notices to the tree.

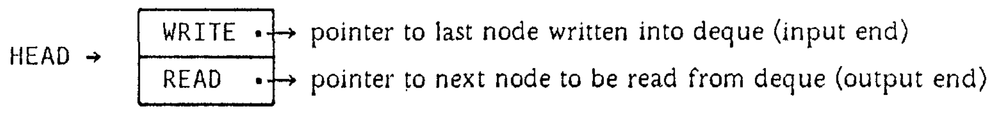
Using four notices with the event times $t = 1$, $t = 2$, $t = 3$, and $t = 4$, show that the method of traversal does not uniquely determine the shape of the tree to be constructed. Do this for two types of trees: one to be traversed in inorder and one in postorder. (Postorder traverses the root after both subtrees have been traversed; inorder traverses the root after the left subtree has been traversed.)

(c) Assume that notices with equal event times are represented as a right branch working as a FIFO queue. What advantage does a postorder tree have over a inorder tree?

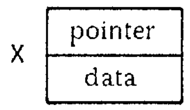
2. Deques and lists. *(20 minutes)*

(a) Explain how a doubly-linked list can be represented using only a single pointer field (one word) for each data carrying node in the list, and a head node containing two pointers.

(b) Suppose that the solution to question (a) is used to implement an input and output restricted deque. The head node contains two pointers, WRITE and READ :


HEAD →
WRITE
READ
pointer to last node written into deque (input end)
pointer to next node to be read from deque (output end)

Normal deque entries have a pointer word and a data word:



Write down insertion and deletion algorithms for this I/O restricted deque:

```
         INSERT (X,HEAD)      puts X at tail of deque
    X := DELETE (HEAD)        sets X equal to head of deque and deletes the head
```

(c) Assume that three data elements named X, Y, and Z, at locations 57, 19, and 124 respectively, have been entered into an initially empty input/output restricted deque. Draw an illustration of the deque, including the values in HEAD and all pointer fields. Assume HEAD is at location 100.

(d) Illustrate the state of the above deque after the insertion of a new node W (location 67).

3. Data structures. *(10 minutes)*

Describe briefly an appropriate data structure for each of the following situations (one or two lines indicating that you understand the issues will be sufficient for full credit):

(a) Tridiagonal matrix (greater than 10 × 10) on which matrix multiplication will be performed. Try to save space.

(b) An alphabetized list (length >> 10) requiring fast insertion but only occasional ordered readout.

(c) Two stacks (limited available space). Suggest two representations.

(d) Sparse matrix (100 × 100 with at most 10 non-zero entries). Store, fetch, and listing of non-zero entries of a particular row or column will be required.

(e) Symbol table (length >> 50) requiring fast read/write access.

4. Algorithm analysis. *(15 minutes)*

(a) Suppose we are given n real numbers $x_1, x_2, \ldots, x_n$. Show that

$$\max_{1 \le i < n} |x_i - \theta|$$

is minimized when

$$\theta^* = \frac{\alpha + \beta}{2}, \text{ where } \alpha = \min_{1 \le i \le n} x_i,\ \beta = \max_{1 \le i \le n} x_i.$$

(b) Carefully describe an algorithm for computing θ^* efficiently with respect to the number of comparisons. You may assume that n is even.

THEORY OF COMPUTATION

1. Halting problem. *(30 minutes)*

The undecidability of the halting problem may be used to show that certain programs cannot be written thus saving one the time and effort of attempting such programs only to eventually give up in failure. Use the halting problem to do *one* of the following two problems.

(a) Given a language which allows recursive procedures, show that there is no algorithmic method to test whether a given procedure will ever call itself at run time (i.e. whether or not there is an input value which will cause the procedure to call itself).

(b) Show that there exists no algorithmic method to test whether or not two programs have the same I/O characteristics (i.e. whether or not there is an input value such that one program halts on this input and the other one doesn't or they both halt but give different output).

2. Language classification. *(20 minutes)*

Classify each of the following languages according to whether they are:

(a) regular
(b) context-free
(c) context-sensitive
(d) recursively enumerable
(e) not recursively enumerable.

(Partial credit will be given for correct classifications even though they are not the most specific classification possible.)

(1) $\{ 0^n 1^m \mid n \geq m \geq 0 \}$

(2) $\{ 0^i 1^j 0^k \mid i=j \text{ or } j=k \}$

(3) $\{ 0^x 1^y \mid x + y \text{ is divisible by } 3 \}$

(4) $\{ (x, y, z) \mid$ the xth algorithm running on input y terminates in z steps $\}$

(5) $\{ 1^a \mid a \text{ is a prime} \}$

(6) the set of all palindromes in $\{0, 1\}^*$

(7) $\{ u \in \{0, 1\}^* \mid$ there exists an integer n such that u is the binary representation of n^2 $\}$

(8) $\{ x \mid$ the xth algorithm terminates on all input $\}$

(9) $\{ u \in \{0, 1\}^* \mid$ there exists an integer n such that u is the binary representation of $5n$ $\}$

(10) the set of names of all graduate students in the Computer Science Department

3. True or false. *(10 minutes)*

For each of the following statements, state whether it is true, false, or unknown (still an open problem).

(1) All regular languages can be recognized by a finite automaton.

(2) $[(\forall x)(\exists y)[P(x) \supset Q(y)]] \equiv [[(\exists x)P(x)] \supset [(\exists y)Q(y)]]$ is a tautology.

(3) All context-free grammars can be recognized by a deterministic pushdown automaton.

(4) P = NP

(5) The problem of whether or not there are ten 4's in the infinite expansion of π is undecidable.

(6) Any language which can be recognized by a nondeterministic linear bounded automaton can also be recognized by a determinisitic linear bounded automaton.

(7) $[(\forall x)[P(x) \vee Q(x)]] \equiv [[(\forall x)P(x)] \vee [(\forall x)Q(x)]]$ is a tautology.

(8) $[(\forall x)[P(x) \wedge Q(x)]] \equiv [[(\forall x)P(x)] \wedge [(\forall x)Q(x)]]$ is a tautology.

(9) The intersection of two regular languages is still regular.

(10) The intersection of two context-free languages is still context-free.

HARDWARE

1. BCD to XS3 notation. *(30 minutes)*

Design a combinational circuit to convert one BCD (8421) digit to the corresponding digit representation in excess-3 code. Assume that non-code input combinations do not occur at the circuit inputs. Use as few gates as possible (assume double-rail inputs).

(a) Design a two-stage circuit using AND gates and OR gates.

(b) Design a multi-stage circuit using AND gates and OR gates.

(c) Design a two-stage circuit using NOR gates.

2. JK ripple counter. *(15 minutes)*

(a) Draw a circuit for a three-stage ripple counter constructed from JK flipflops.

(b) Show the complete sequence of 3-bit output values (corresponding to both stable and unstable states) which appear as the counter passes through a complete cycle.

3. Definitions. *(15 minutes)*

Define each of the following terms and give an example.

(a) destructive read-out memory
(b) volatile memory
(c) random-access memory
(d) control memory
(e) decoder

PROGRAMMING PROBLEM: Pattern Transformation Compiler

Pattern directed computation has proven to be very useful in many areas of computer science. It has been used in natural language processing, algebraic simplification, and is the basis for the idea of compilation, to name only a few. Any pattern directed computation may be represented as transformation rules of the form

<pattern> $\rightarrow$ <substitution>

where the pattern is matched against the input and replaced by the substitution. Given a procedure which applies the rules in a certain order and decides when to stop, a list of these transformations will define a computation. (In fact, if the procedure is general enough, then any effective computation can be expressed as a list of such transformations.) However, a computation stated in terms of a list of transformations often proves to be very inefficient. One possible improvement would be to compile the transformations into code which may be directly executed.

You are to write a program which will take a list of transformations and compile them into any reasonable language of your choice (e.g. ALGOL W or SAIL).

Input syntax

In general, it is very difficult to compile a list of transformations into a usable form because the strings in the transformation may take any form. To make the problem more interesting, we shall restrict the form of the rules. We assume that the strings in the rules are in fact LISP lists for an expression of binary operators, according to the following syntax

```
<exp>        ::= <number> | <variable> | (<operator><exp><exp>)
<number>     ::= any unsigned integer
<variable>   ::= any string of alphabetic characters
<operator>   ::= any string of alphabetic characters
```

The allowable operators may vary with the list of transformations being considered. In attacking this problem, you may make as much use of the syntax as you see fit.

Transformations

In order to make transformations useful, one needs to have some kind of general pattern matching capability available. To keep the compilation at a reasonable level, you need only consider the following mechanisms for pattern matching.

(1) The pattern and the expression being matched must satisfy the syntactic form for an <exp> as described above, with the exception that ?X--- and ?N---, where the --- is replaced by a number, may be used any place an <exp> is required.

(2) The form ?N---, with the --- replaced by a number, will match any <number>.

(3) The form ?X---, with the --- replaced by a number, will match any <exp>.

(4) For any part of the right hand side of a transformation where an <exp> is syntactically correct, there may be an arithmetic expression of numbers and forms ?N---, with the interpretation that the expression is replaced by the value of that expression. You need only implement addition, subtraction, and multiplication.

Although the forms ?N--- and ?X--- match general forms, they will essentially be given a value on the first time they match in a pattern and will then only match the same expression they matched the first time. Thus, the pattern (PLUS ?X1 ?X1) will match (PLUS ALPHA ALPHA) and (PLUS (TIMES 2 V) (TIMES 2 V)), but will not match (PLUS ALPHA (TIMES 2 V)). Furthermore, once one of these forms matches on the left side of a transformation, it will have the same value any place it is used on the right side of that transformation. Thus if the transformation (PLUS ?X1 ?N1) → (PLUS ?N1 ?X1) is applied to the expression (PLUS (TIMES 2 V) 5), the result is (PLUS 5 (TIMES 2 V)).

Interpretation of a list of transformations

For the purposes of this problem, it will be assumed that any list of transformations has the following implicit control structure. To apply a list of transformation to an expression *E*, first apply the list of transformations to each subexpression of *E*. After the subexpressions have been fully transformed, take the first transformation whose left side matches with *E* and apply that transformation to *E*. Repeat this last step until no transformation has a left side which matches *E* (i.e. after applying a transformation to *E*, do not try to reapply the list of transformations to the new subexpressions of *E*, however, do start at the top of the list of transformations to find a transformation to apply to the new *E*). Thus, if one were to apply the transformations

(i) (PLUS 0 ?X1) → ?X1
(ii) (PLUS ?X1 ?N1) → (PLUS ?N1 ?X1)

to the string (PLUS V (PLUS 0 0), a trace of the execution would go as follows.

```
considering V
   none of the rules match so nothing is done
considering (PLUS 0 0)
   considering 0
      none of the rules match
   considering 0 (the second subexpression)
      none of the rules match
   rule (i) matches: (PLUS 0 0) → 0
   considering 0 (the result)
      none of the rules match
   (the expression is now (PLUS V 0))
rule (ii) matches (but not rule (i)): (PLUS V 0) → (PLUS 0 V)
rule (i) matches: (PLUS 0 V) → V
none of the rules match
done
```

Thus, the above transformations change (PLUS V (PLUS 0 0)) into V.

Problem

Using the language of your choice, write a program which will take a list of transformations as described above and compile them into code for the procedure inherent in those transformations. The code for the transformations which your program produces should run as fast as possible.

In addition to a listing of your program, you should turn in a listing of the code your program produces when run on the transformations listed below, and listings of the expressions produced when the code you compiled is run on the expressions listed below.

<u>Sample transformations</u>

```
(PLUS 0 ?X1) → ?X1
(PLUS ?N1 ?N2) → ?N1+?N2
(PLUS ?N1 (PLUS ?N2 ?X1)) → (PLUS ?N1+?N2 ?X1)
(PLUS ?X1 ?N1) → (PLUS ?N1 ?X1)
(PLUS ?X1 ?X1) → (TIMES 2 ?X1)
(SUB ?X1 ?X2) → (PLUS ?X1 (TIMES -1 ?X2))
(TIMES 1 ?X1) → ?X1
(TIMES 0 ?X1) → 0
(TIMES ?N1 ?N2) → ?N1*?N2
(TIMES ?N1 (TIMES ?N2 ?X1)) → (TIMES ?N1*?N2 ?X1)
(TIMES ?X1 ?N1) → (TIMES ?N1 ?X1)
(PLUS ?X1 (TIMES ?N1 ?X1)) → (TIMES 1+?N1 ?X1)
(PLUS (TIMES ?N1 ?X1) (TIMES ?N2 ?X1)) → (TIMES ?N1+?N2 ?X1)
```

<u>Sample expressions</u>

```
(PLUS (TIMES -19 V) (TIMES 2 (PLUS (TIMES V 7) (PLUS (TIMES V 3)
     (TIMES V (TIMES U (SUB U U)))))))

(SUB (SUB (SUB (SUB VA VB) (SUB VC VD)) (SUB VE VF))
     (SUB (SUB (SUB VA VB) (SUB VC VD)) (SUB VE VF)))
```

Your programs will be graded according to the following criteria: correctness, clarity of program and documentation, appropriateness of data structures, and efficiency of both the compiler code and the compiled code.

Spring 1976 Comprehensive Exam

THEORY OF COMPUTATION

1. Regular sets. *(30 minutes)*

(a) Are the following sets regular? Prove your answers.

(i) $\{ x \mid x \in (0+1)^*$, where x has a 1 in the exact center $\} = \{ (0+1)^n 1 (0+1)^n \mid n \geq 0 \}$

(ii) $\{ xyx \mid x, y \in (0+1)^*$ where x is non-empty and begins and ends in 1 $\}$

(b) Given finite automata M_1 and M_2 accepting regular sets R_1 and R_2, show that it is decidable whether $R_1 \subseteq R_2$.

2. Bad function. *(30 minutes)*

A LISP predicate *term* is called a termination tester if for any S-expression *e*, *term*[*e*] = T if and only if the LISP evaluation of *e* terminates; otherwise *term*[*e*] = F or is undefined. Thus we must have *term*[(CAR (QUOTE (A)))] = T, and *term*[((LABEL FOO (LAMBDA (X) (FOO X))) NIL)] may have the value F or may be undefined. Let *term** be the S-expression representation of *term*.

(a) Why can't there be a termination tester that is always defined?

(b) Write a LISP function *bad* such that for any termination tester *term*, *bad*[*term**] is defined, but *term*[*bad*[*term**]] is undefined. Show that your *bad* works. (Partial credit will be given for an informal description of *bad*.)

Hint: A two line answer is possible using the LISP system function

$subst\,[x, y, z]$ = if $atom\ z$ then [if $z\ eq\ y$ then x else z]
else $cons[subst[x, y, car\ z],\ subst[x, y, cdr\ z]]$

and the function $quine[x] = subst[x$,X,(X (QUOTE X))].

NUMERICAL ANALYSIS

1. Quadratic roots. *(15 minutes)*

Suppose that the equation $x^2 + a_1 x + a_2$ with real coefficients possesses real roots α, β. Show that if x_0 is chosen sufficiently close to α, the iteration

$$x_{k+1} = -\frac{a_1 x_k + a_2}{x_k}$$

converges to α if $|\alpha| > |\beta|$, the iteration

$$x_{k+1} = -\frac{a_2}{x_k + a_1}$$

converges to α if $|\alpha| < |\beta|$, and the iteration

$$x_{k+1} = -\frac{x_k^2 + a_2}{a_1}$$

converges to α if $2|\alpha| < |\alpha+\beta|$.

2. Floating-point sums. *(45 minutes)*

Consider t-digit base-β floating point arithmetic. Given two floating point numbers x, y, we denote by

$$fl(x+y),\ fl(x-y),\ fl(x\cdot y),\ fl(x/y)$$

the results of floating point addition, subtraction, multiplication, and division, respectively, where the result is rounded. We define $fl(x+y+z)$ to mean $fl(fl(x+y)+z)$. Similarly, if a_i, $i = 1, 2, \ldots, N$ are floating point numbers, we define

$$fl(\sum_{i=1}^{N} a_i)$$

to mean floating point addition of the terms in the order from $i=1$ to $i=N$.

(a) Let $u = (1/2)\,\beta^{1-t}$ and suppose that $Nu \le 0.01$. Show that

$$fl(\sum_{i=1}^{N} a_i) = \sum_{i=1}^{N} a_i(1 + 1.01(N+1-i)\theta_i u)$$

where $|\theta_i| \le 1$.

(b) Suppose that you are asked to write an ALGOL W or FORTRAN program to obtain an approximation to the function

$$f(x) = \sum_{n=1}^{\infty} \frac{1}{n^2 + x}, \text{ where } 0 \le x \le 1.$$

You decide that you will sum the first 10000 terms of the series. Does the order of summation of the terms make any difference in the accuracy of your result? Explain.

(c) Consider the two sums

$$S_1 = fl\Big(\sum_{n=1}^{N} \frac{1}{n^2 + x}\Big),\ S_2 = fl\Big(\sum_{n=N}^{1} \frac{1}{n^2 + x}\Big).$$

Use the result (or method) of part (a) to find bounds for the round-off errors committed in each case, in terms of N, t, β for $0 \le x \le 1$. Assume that N is small enough so that N^2 can be found without round-off. Then for $n \le N$,

$$fl(n^2+x) = (n^2+x)(1+\delta),\ |\delta| \le u.$$

(d) Evaluate approximately both of the bounds found in part (c) for the case of single precision arithmetic on the IBM 360 ($\beta = 16$, $t = 6$) with N=10000. Do these bounds confirm the conclusion you reached in part (b)?

(e) For x=0 and N=10000, using single precision arithmetic in an ALGOL W program it was found that $S_1 = 1.643517$, $S_2 = 1.644833$. The true value of $f(0)$ is given by $f(0) = \pi^2/6 = 1.644934\ldots$. By obtaining upper and lower bounds for the truncation error resulting from the use of S_1 or S_2 in place of $f(0)$, estimate the actual round-off error in each of the above two results. How do these errors compare with the bounds found in part (d)?

Note: the following may be useful in parts (d) and (e):

$$\sum_{n=1}^{N} \frac{1}{n} < 1 + \ln N, \qquad \frac{1}{N+1} < \sum_{n=N+1}^{\infty} \frac{1}{n^2} < \frac{1}{N}, \qquad \ln 10000 \approx 9.21,\ \ 2^{-10} \approx 10^{-3}.$$

SYSTEMS

1. Storage allocation. *(20 minutes)*

In a higher level language, it is necessary to allocate storage of several types, including variables, arrays, strings, records, etc. This allocation can be done at compile time, at run time using a stack, or at run time using a more general storage allocator, for which the order in which blocks of storage are freed is not a simple function of the order in which they were allocated. For each of the following features, indicate the simplest kind of allocation which will suffice (assuming compile time is simpler than run time, and stack simpler than general). Give a brief justification for your choice (1 or 2 sentences).

(a) values stored for variables in a language with the free variable lookup environment determined on the basis of the run-time calling environment (as in LISP)

(b) values stored for variables in a language with the free variable lookup environment determined on the basis of the source code block structure (as in ALGOL)

(c) records and references (as in ALGOL W)

(d) arrays, whose bounds are determined on block entry (as in ALGOL)

(e) strings of arbitrary length (as in SAIL)

(f) a READ statement which converts tokens into unique ATOM references (as in LISP)

(g) overlapping arrays (as in FORTRAN's COMMON)

(h) concurrent processes (as in SAIL)

(i) variables which refer to procedures (as in SAIL PROCEDURE variables)

(j) recursive call by value (as in ALGOL and LISP)

2. Microcode usage. *(20 minutes)*

You are a systems programmer at a laboratory which has just gotten a new micro-programmed machine which has enough micro-code memory to simulate your old machine with a good deal left over to spare. You want to make use of this additional power to speed up the execution of your favorite higher level language (in answering, you can use whichever one you are familiar with, e.g. LISP, SAIL, ALGOL W).

(a) What could you use the micro-code for? Think of several different ways in which it could be used.

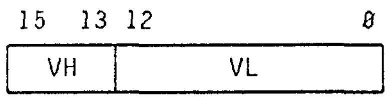
(b) How would you go about evaluating the relative benefit to be gained from different uses? Include measurements which demand simulating program operation, and those which don't.

(c) How would you expect the choice of micro-code to be different if you were on a machine whose major limitation for typical programs in your language was not in instruction execution speed, but in core size?

3. Memory mapping. *(20 minutes)*

The essential characteristics of a certain computer's memory mapping hardware can be described as follows:

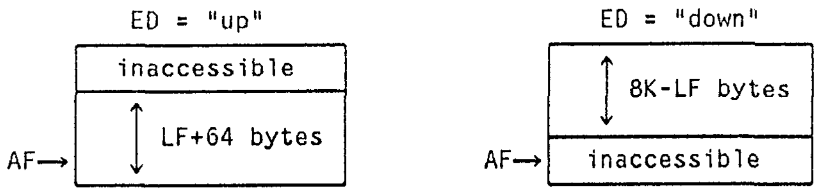
1. Programs generate 16-bit virtual addresses, which are conceptually divided into fields as shown:



2. The 3-bit value VH selects one of eight *mapping registers* MR0-MR7. (The mapping registers occupy fixed locations, which are normally accessible only to the monitor.) Each MR has the following fields.

   AF (address field) - 18 bits, low order 6 are zero (positive integer)
   LF (length field) - 13 bits, low order 6 are zero (positive integer)
   ED (expansion direction) - 1 bit ("up" or "down")
   NR (non-resident) - 1 bit ("true" or "false")

   If NR="true" then this mapping register describes no physical memory. Otherwise, the other fields describe a block of contiguous physical memory as shown:



3. Accesses proceed as follows: the mapping register selected by VH is inspected. If NR="true" then the access traps. The physical address PA=AF+VL is formed. If it lies outside the block of addresses defined by AF, LF, and ED, the access traps. Otherwise, the access is made to physical address PA.

(a) Describe how this hardware can be made to simulate a single set of base-bounds registers.

(b) Describe how this hardware can be the basis of a paged address space.

(c) Describe how this hardware can be the basis of a segmented address space. Is it possible to achieve Multics-style segmentation on this machine? How or why not?

(d) When this hardware is used for paging, the pages are very large. What are the advantages and disadvantages of this?

DATA STRUCTURES

1. Implementation of structures. *(24 minutes)*

What data structure would you use to carry out the following sets of operations? Describe one good way you could implement the data structure, and the operations, on a computer. Give the best order-of-magnitude bound you can on the average time for each operation, using your data structures. (You need not prove your time estimates.)

(a) Insert an item.
Delete the most recently inserted item.

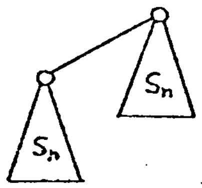
(b) Insert an item.
Delete the least recently inserted item.

(c) Insert an item with an associated floating-point value.
Delete the item with the smallest value.

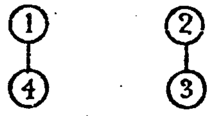
(d) Insert an item with a floating-point value.
Delete all items with a given value.

2. S_n tree checker. *(36 minutes)*

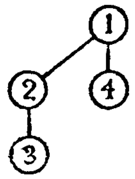
An S_n tree is an ordered tree having a key associated with each node. The class of these trees is defined for $n \geq 0$ as follows: An S_0 tree consists of a single node with an arbitrary key. An S_{n+1} tree, for $n \geq 0$, consists of two S_n trees combined by adding a single new branch, as shown at the top of the next page. The S_n tree of the pair whose root node has the larger key is made the leftmost son of the root of the other S_n tree. (Ties between the roots can be decided arbitrarily.)



For example, the two trees below are S_1 trees:



and combining them as described above gives an S_2 tree:



(Note: what we've actually defined is an "ordered S_n tree with heap-ordered keys"; we'll continue to use the briefer, if less accurate, description "S_n tree".) It is easy to see that any S_n tree has 2^n nodes, and that the smallest key value is in the root.

(a) Suppose that an S_n tree is represented internally as a binary tree using the natural correspondence between forests and binary trees. That is, each node has the internal format:

KEY	
LCHILD	RSIB

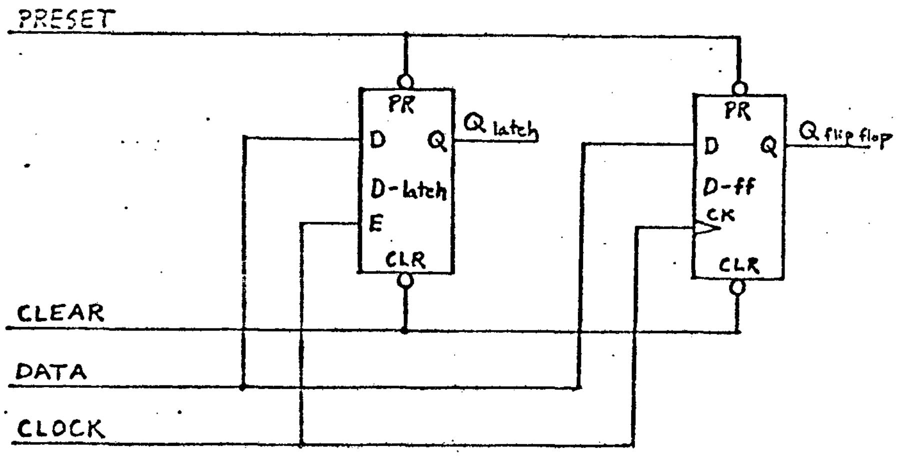
where KEY is the key value of the node, and LCHILD and RSIB are pointers to the leftmost child of this node and the next sibling to the right of this node, respectively. Show the representation of the S_2 tree using nodes in the above format.

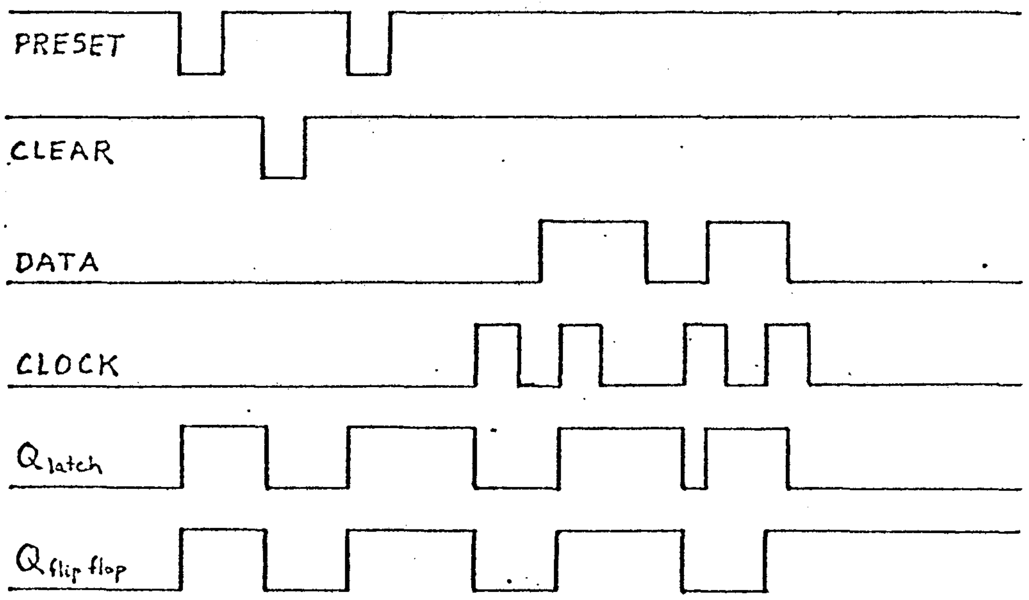
(b) Write a procedure CHECK which, given a pointer S and integer N, determines whether or not S points to the root of an S_N tree in the above binary tree representation. The procedure should return normally if it accepts S; otherwise, it should call procedure ERROR with a string argument describing why it rejects S (e.g. ERROR("Parent larger than child")).

The procedure may not modify the input tree during execution. CHECK should require at most time proportional to 2^N and space proportional to N. (You should assume that S points to a nonempty binary tree with nodes in the format given in part (a); there is no need to check for circularity in the structure.) You may write CHECK in an Algol-like notation, and explicit recursion is allowed.

HARDWARE

The following example demonstrates the behavior of the *D–latch* and the *D–flip–flop*, which are 1-bit memory elements:





1. Shift registers. *(30 minutes)*

Design a four-bit serial-in, serial-out shift register using

(a) D-flip-flops
(b) D-latches.

2. Number representations. *(30 minutes)*

(a) Modify the shift register of question 1(a) so that is possible to shift out the ones' complement of the stored magnitude.

(b) Repeat part (a) for shifting out the two's complement of the stored magnitude.

ARTIFICIAL INTELLIGENCE

1. Resolution proof. *(15 minutes)*

Prove by resolution that $\exists x\ \forall y\ (P(x) \supset P(y))$ is a theorem of first order logic.

2. Tower of Hanoi. *(45 minutes)*

In the tower of Hanoi problem, there are three spikes and on spike 1 there is a stack of n disks of successively decreasing size. The problem is to move one disk at a time from one spike to another and end up with all n disks on spike 2, observing the restriction that no disk is ever on top of a smaller disk.

(a) Describe an algorithm that solves the problem in $O(3^n)$ steps.

(b) Describe an algorithm that solves the problem in $O(2^n)$ steps.

(c) Describe an algorithm that "solves" it in n steps. What is meant by "solve" in this case?

(d) Discuss what is required for a computer program to "honestly solve" it in $O(1)$ steps.

PROGRAMMING PROBLEM

Write a program to solve a large sparse system of linear equations by Gaussian elimination. Recall that an $n \times n$ system of linear equations is a matrix equation

$$\underset{\sim}{A}\underset{\sim}{x} = \underset{\sim}{b},$$

where A is an $n \times n$ real-valued matrix, $\underset{\sim}{x}$ is a $n \times 1$ vector of variables, and $\underset{\sim}{b}$ is a $n \times 1$ vector of constants.

There are two steps to Gaussian elimination:

(1) Convert the matrix A to upper triangular form by means of row operations. A row operation consists of adding an appropriate multiple of a row of A to another row of A, with the same transformation being applied to the vector $\underset{\sim}{b}$.

(2) Backsolve: if A is in upper triangular form, the equation for x_n is $a_{nn}x_n = b_n$, which can be solved directly. Substituting the value for x_n into the other equations, we can solve for x_{n-1}, then for x_{n-2}, and so on.

The standard method for performing step (1) is to use a_{11} to zero all other entries in the first column by adding to row i the multiple $-a_{i1}/a_{11}$ of row 1; then to use a_{22} to zero all other entries below it in column 2; and so on. This is called Gaussian elimination without pivoting, and it only works if the diagonal entries are nonzero when they are needed for elimination. Also, they must be large enough so that the method is stable. For this problem, you may assume that Gaussian elimination without pivoting will work. (You may consult other references for a further discussion of Gaussian elimination.)

The major difficulty with this method is the problem of *fill-in*, that is, the creation of new nonzero matrix elements by the elimination process. You must devise a data structure which will take advantage of the sparseness of A by not storing or performing unnecessary arithmetic on matrix elements which are zero, and keeping track of the fill-in. You are to assume that A is large and very sparse (e.g. 300×300 with 2000 nonzero entries), but that A has no other perceivable structure (i.e. A doesn't necessarily have a small bandwidth).

It may help to use a two-pass method, first computing the possible fill-in locations and then doing the arithmetic. It may also help to use *row elimination*, in which the first row is zeroed, then the second, and so on; instead of column elimination.

Winter 1977 Comprehensive Exam

SYSTEMS

1. Systems quickies. *(30 points)*

(1) Explain the concept of a SIMULA *class*. How does it contribute to program structuring and resource protection?

(2) What are the advantages, if any, of semaphore operations over "test-and-set" operations as a high-level concept?

(3) Suppose you have to multiply two large square matrices A and B using a virtual memory system. How would you lay the matrices out so as to minimize your program's working set size, assuming that your program can generate the elements of A and B equally conveniently in any order?

(4) State an undesirable aspect associated with each of the following scheduling algorithms:

- (a) first come, first served
- (b) shortest job next
- (c) round robin.

(5) Suppose hierarchical resource allocation is enforced, i.e. the resources $r_1, \ldots, r_n$ are ordered so that no process can request resource r_j if it already holds resource r_i for some $i > j$. Can deadlock occur? (Explain informally, either constructing an example of deadlock or sketching a proof that deadlock is impossible.)

2. Compiler issues. *(30 points)*

Suppose you are writing a compiler for a new language, and you have some say in the design of the details of certain features. Describe in general terms how you would deal with the following issues in a *recursive* Algol-like language. (Please don't quibble with the syntax in the examples below; just take them as a guide to the semantic problem being raised, and not as a rigid requirement for your language.)

(a) *(7 points)* Suggest a run-time implementation for string variables of arbitrary, dynamically varying lengths. Would this differ from your implementation of string variables whose lengths are given as part of their declarations?

(b) *(8 points)* Suppose the language is to have extensible data types, where declarations can be freely intermixed with statements. How would you represent at compile time the definition of type "vector" in the fourth line of the following example source program?

```
begin
   integer length;
   length := 3;
   type vector = integer array [1::length];
   length := 25;
   vector x;
   length := 75;
   vector array y [1::length];
      . . .
end.
```

Give two answers to this question, depending on how the language is defined, where type "vector" involves the value of *length* either

(i) at the time "vector" is declared, or
(ii) at the time "vector" is used in another declaration.

The example program under interpretation (i) declares x to be a vector of length 3 and y to be an array of 75 vectors of length 3; but under interpretation (ii) x has length 25 and y consists of 75 vectors of length 75.

(c) *(15 points)* Suppose the language is to have *procedure variables*, i.e. variables whose "value" denotes a procedure. (i) Discuss what language constraints you would put on the set of procedure values which a procedure variable might assume. What effect do such constraints have on compile-time vs. run-time error checking, especially with respect to parameters? (ii) Explain briefly how you would implement procedure variables at run time, paying particular attention to problems you might encounter with static vs. dynamic environments as found in coroutines or in the following example:

```
begin
   procedure-var any;   (a procedure variable)
   procedure foo (procedure-var result q);
   begin
      integer m;
      procedure baz;
         print (m);
      m := 3;
      q := baz;
   end;
   foo (any);
   any;
end.
```

NUMERICAL ANALYSIS

1. Polyalgorithms for nonlinear equations. *(5 points)*

Most of the "state of the art" programs for finding real zeros of a general scalar equation $f(x) = 0$ use the bisection method initially and switch to something like Newton's method or the secant method for the final iterations. What motivates this strategy?

2. Nonlinear equation solvers. *(5 points)*

Why is the secant method often more efficient than Newton's method even though its order of convergence is smaller?

3. The influence of arithmetic on algorithms. *(20 points)*

The modulus of a complex number $z = x + iy$ is usually defined by

(1) $|z| = (x^2+y^2)^{1/2}$.

Suppose we want to compute $|z|$ using floating-point arithmetic on a computer where the representations of nonzero real numbers are constrained to lie between M and M^{-1} in absolute value. Results with magnitude greater than M will cause overflow and those with magnitude less than M^{-1} will cause underflow. Suppose further that we want to determine $|z|$ maintaining as much significance in the final result as possible and avoiding underflow or overflow unless $|z|$ lies outside the interval $[M^{-1}M]$.

(a) *(2 points)* What problem do we immediately encounter if we attempt to use formula (1)?

It has been suggested that (1) be reformulated as follows if $x, y \neq 0$. Set $u = \max(|x|, |y|)$ and $v = \min(|x|, |y|)$ and use

(2) $|z| = u(1 + (v/u)^2)^{1/2}$,

where v/u is replaced by zero if this division causes underflow. This formulation overcomes much of the difficulty encountered using (1), and it is generally satisfactory if binary arithmetic is used. However, it is unsatisfactory if hexadecimal arithmetic is used. Suppose you have the hexadecimal representation

$s = \pm 16^e \cdot f$

where e is an integer exponent and f is a binary fraction containing t bits (where t is divisible by 4) normalized so that $1/16 \leq f < 1$.

(b) *(9 points)* At most how many significant bits can be expected in the fractional part of the representation of $(1 + (v/u)^2)$?

The formulation

(3) $z = (2u)(0.25 + (v/2u)^2)^{1/2}$

has been suggested to correct the situation in part (b). There are values of z for which a computation based on (3) is somewhat less accurate than one based on (2), but statistically (3) is a significant improvement.

(c) *(9 points)* If $w = 0.25 + (v/2u)^2$, at most how many correct bits can we expect in the fractional part of w?

4. <u>The solution of sparse linear systems</u>. *(30 points)*

(a) *(25 points)* Describe a modified Gaussian elimination algorithm to reduce an $n \times n$ matrix of the form

$$\begin{pmatrix} b_1 & c_1 & & & & & a_1 \\ a_2 & b_2 & c_2 & & & & \\ & a_3 & b_3 & c_3 & & & \\ & & \cdot & \cdot & \cdot & & \\ & & & \cdot & \cdot & \cdot & \\ & & & & \cdot & \cdot & \cdot \\ & & & & a_{n-1} & b_{n-1} & c_{n-1} \\ c_n & & & & & a_n & b_n \end{pmatrix}$$

to upper triangular form. Assume that pivoting is not necessary. Such matrices often arise — they result from periodic spline approximations and periodic boundary-value problems for ordinary differential equations. Make sure that your algorithm requires only the storage of three vectors of length n (say $a_1 \ldots a_n$, $b_1 \ldots b_n$, and $c_1 \ldots c_n$), and a small additional working area whose size is independent of n. You need not write a program, but you must give all relevant formulas. Note that your operations only operate on the matrix elements, not on any assumed "right-hand side" of a matrix equation.

(b) *(5 points)* Give a condition on the matrix above which ensures that Gaussian elimination without pivoting will work.

ARTIFICIAL INTELLIGENCE

1. <u>A.I. terms</u>. *(8 points)*

Match each item in column A with the one from column B that is most related to it.

	A		B
(1)	AND/OR trees	(1)	COBOL
(2)	add and delete lists	(2)	backward reasoning
(3)	consequent theorems	(3)	effects of operators
(4)	cooperating knowledge sources	(4)	forward reasoning
(5)	symbol-mapping problem	(5)	HEARSAY-II
(6)	Skolem functions	(6)	inheritance of properties in "isa" hierarchies
(7)	unification	(7)	matching
(8)	set-of-support	(8)	problem reduction
		(9)	removal of existential quantifiers
		(10)	resolution strategy

2. English to logic. *(12 points)*

Write predicate calculus expressions to represent each of the following three sentences, using the predicates *dog*(x), *dogcatcher*(x), *town*(x), *lives_in*(x, y), and *has_bitten*(x, y).

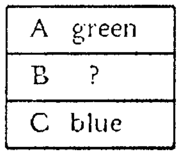
(1) Every town has a dogcatcher who has been bitten by every dog in town.

(2) No town has a dog who has bitten every dogcatcher in town.

(3) At least one town has a dogcatcher who has been bitten by none of the dogs in town.

3. Blocks. *(10 points)*

Suppose three blocks are stacked A on B on C, where A is green, C is blue, and the color of B is unknown (it might be green with blue polka dots).



List the axioms needed to prove by resolution that there is a green block immediately on top of a non-green block and show the resolution proof. *Hint:* Include, for example, the axioms *on*(A, B), ~*green*(C).

4. Wffs vs. clauses. *(15 points)*

In pure resolution proof procedures, wffs are represented as clauses. What are some of the disadvantages of this approach? Give examples of wffs which we might not wish to convert to clause form, and explain why the clause form is undesirable.

5. Semantic information processing. *(15 points)*

Good arguments have been given in support of each of the following statements. Select one of the statements and present as much evidence for it as you can, considering typical implementations of semantic information processing systems.

(1) In terms of representational power, there is really very little difference (if any) between the predicate calculus and the better semantic network formalisms.

(2) Semantic networks provide a much richer representational formalism than the predicate calculus does.

ALGORITHMS

1. Binary search trees. *(45 points)*

A *binary tree* T either is empty or consists of a root node (containing $info[T]$) and two binary trees $left[T]$, $right[T]$. We can represent an empty tree by $T = 0$ and we can let T be a positive integer in the other case, so that a binary tree structure can be represented by means of three integer arrays $info[\ ]$, $left[\ ]$, $right[\ ]$. (We could also, of course, use records and references if we don't have to program in FORTRAN.) The set $nodes(T)$ is defined recursively as follows:

$$nodes(T) = \begin{cases} \emptyset\ , & \text{if } T \text{ is empty;} \\ \{T\} \cup nodes(left[T]) \cup nodes(right[T])\ , & \text{otherwise.} \end{cases}$$

A *binary search tree* is a binary tree T for which there is an ordering "<" defined on all the elements $info[T]$, and we have

$info[T] < info[T]$ for all $t \in nodes(left[T])$,
$info[T] > info[T]$ for all $t \in nodes(right[T])$.

Furthermore if T is not empty both $left[T]$ and $right[T]$ must be binary search trees.

The following algorithm is commonly used to insert new elements x into a binary search tree T :

```
recursive procedure ins (integer x; reference integer T);
  if T = 0 then begin
    T ← new_node; info[T] ← x;
    left[T] ← 0; right[T] ← 0
  end
else if x < info[T] then ins(x, left[T])
else if x > info[T] then ins(x, right[T]);
```

Here "reference" is essentially the same as the ALGOL W specification "value result"; and "*new_node*" is an integer procedure which returns the value of an unused position in the arrays $info[\ ]$, $left[\ ]$, and $right[\ ]$.

Let $BST(x_1, x_2, \ldots, x_n)$ denote the binary search tree obtained by the sequence of operations

$T \leftarrow 0;\ ins(x_1, T);\ ins(x_2, T);\ \ldots;\ ins(x_n, T)$.

For example,

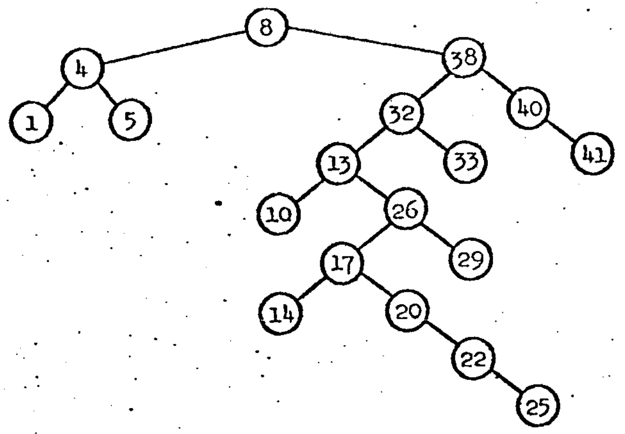
$BST(8,\ 38,\ 32,\ 40,\ 13,\ 26,\ 4,\ 41,\ 17,\ 20,\ 14,\ 5,\ 33,\ 29,\ 22,\ 1,\ 10,\ 25)$

is a binary search tree which may be drawn as shown at the top of the next page.

(a) *(5 points)* Draw the binary search tree

$BST(23,\ 8,\ 38,\ 32,\ \ldots,\ 25)$,

where the elements after 23 are the same as in the example above.



(b) *(25 points)* ("Insertion at the top.") Fill the blanks in the following program so that "*topins*" constitutes a valid insertion procedure for binary search trees. After *topins*(x, T) has been applied to $T = BST(x_1, \ldots, x_n)$, the new value of T should be $BST(x, x_1, \ldots, x_n)$. Thus the sequence of operations

$T \leftarrow 0$; *topins*(x_1, T); *topins*(x_2, T);...;*topins*(x_n, T)

should produce $BST(x_n, \ldots, x_2, x_1)$.

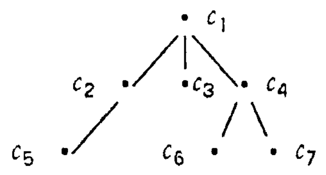
```
procedure topins (integer $x$; reference integer $T$);
  begin integer $u$;
  $u \leftarrow new\_node$; $info[u] \leftarrow x$;
  $magic\ (T, x, left[u], right[u])$;
  $T \leftarrow u$
  end;
recursive procedure $magic$ (integer $T$, $x$; reference integer $L$, $R$);
  if $T = 0$ then [          ]
  else if $x < info[T]$ then
    begin $magic\ (left[T], x,$ [      ], [      ]);
      $R \leftarrow T$
    end
  else if $x > info[T]$ then
    begin [                    ]
    end
   else begin comment [          ];
    $L \leftarrow$ [        ];
    $R \leftarrow$ [        ];
    end;
```

Suggest a better name for the procedure "*magic*": [   ].

(c) *(15 points)* Sketch an informal proof that the program in your answer to part (b) is correct.

2. Worst case cost. *(15 points)*

Each node α of a (finite) tree has been assigned a nonnegative cost $c(\alpha)$ in such a manner that the sum of all costs on nodes of the path from the root to α is less than or equal to the number of nodes on this path. For example, in the tree



we must have

$$
\begin{aligned}
&c_1 \le 1 \ , \\
&c_1 + c_2 \le 2 \ , \\
&c_1 + c_3 \le 2 \ , \\
&c_1 + c_4 \le 2 \ , \\
&c_1 + c_2 + c_5 \le 3 \ , \\
&c_1 + c_4 + c_6 \le 3 \ , \\
&c_1 + c_4 + c_7 \le 3 \ .
\end{aligned}
$$

Given this condition, what is the maximum possible value of the total cost of the tree (i.e., the sum of all $c(\alpha)$)? State and prove your answer in terms applicable to an arbitrary oriented tree, not just the tree shown.

THEORY OF COMPUTATION

1. Unification. *(10 points)*

What is the most general unifier of α and β, where

$$
\begin{aligned}
\alpha &= P(Q(P(Q(w),\, P(z,\, Q(P(y,\, y))))),\, P(x,\, P(z,\, Q(x)))) \\
\beta &= P(Q(P(t,\, P(t,\, u))),\, P(v,\, P(Q(u),\, Q(P(t,\, u))))) \ ?
\end{aligned}
$$

That is, find the most general substitution which makes α equal to β.

2. Three three machines. *(25 points)*

Consider an infinite tape containing the symbols

$$\ldots x_{n+1} x_n \ldots x_1 x_0$$

where there is some $n \ge 0$ such that x_j is 0 or 1 for $0 \le j \le n$ but $x_j = \_$ (blank) for all $j > n$. If the tape represents a binary number, the following "finite-state transducer" adds three to that number and halts:

Initially $j = 0$, state = q_1.

If state =	and if x_j =	then set x_j =	and then set $j \leftarrow j + 1$ and go to state
q_1	0	1	q_2
q_1	1	0	q_3
q_1	_	1	q_2
q_2	0	1	*halt*
q_2	1	0	q_2
q_2	_	1	*halt*
q_3	0	0	q_2
q_3	1	1	q_2
q_3	_	0	q_2

(a) *(10 points)* Construct a similar state/action table for a finite-state machine that *multiplies* the given number by 3 and halts. Use only as many states as necessary.

(b) *(10 points)* Is it possible to construct a similar machine that multiplies by any given (fixed) positive integer m? If not, prove the impossibility. If so, what is the minimum number of states required, as a function of m? (The device must start with $j = 0$ and increase j by 1 at every step.)

(c) *(5 points)* Answer part (b) for a machine that produces the *cube* of the input number.

3. Context-free grammars. *(25 points)*

Let L be the context-free language defined by the two productions

$S \rightarrow aSbS$
$S \rightarrow \epsilon$

where ϵ denotes the empty string. The terminal alphabet is $\{a,b\}$ and the only nonterminal symbol is S. Let M be the context-free language defined by the above two production plus a third one:

$S \rightarrow bSaS$.

(a) *(5 points)* Show that the context-free grammar given for M is ambiguous, by exhibiting a string of M which has two different leftmost derivations.

(b) *(5 points)* Give a context-free grammar which defines the language consisting of all nonempty strings of L.

(c) *(10 points)* Give a context-free grammar which defines the language consisting of all strings of L whose length is a multiple of 3.

(d) *(5 points)* Explain how to generalize your construction in part (c) so that *any* context-free grammar can be converted into a grammar for the same language but restricted to strings whose length is a multiple of 3.

HARDWARE

1. Definitions. *(8 points)*

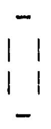
Define each of the following terms:

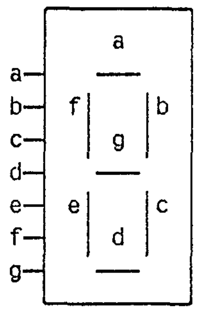
(a) dynamic memory
(b) cache memory
(c) PROM
(d) direct memory access.

2. BCD-to-7-segment decoder. *(30 points)*

Design a combinational circuit to convert a BCD (8421 code) digit to the corresponding 7-bit digit to drive a 7-segment LED display. The display is shown below, with each input requiring a "1" to turn the corresponding segment on. Try to use as few gates as possible in your design.

Example: A BCD "zero" (0000) is displayed by turning on segments a, b, c, d, e, and f, as shown below.





7-segment LED display

(a) *(25 points)* Design a two-level tree circuit for the decoder using AND, OR, NAND, or NOR gates. Assume that double-rail inputs are available, and that non-code input combinations do not occur at the decoder input. (Note: there is a worksheet on the next page.)

(b) *(5 points) Suppose the specifications for the decoder are changed to require that the circuit reject false input data, that is, non-BCD inputs should turn off all the segments.* How would you redesign the circuit? (Just state briefly what changes you would make to the design in part (a).)

Decimal	BCD code D C B A	a	b	c	d	e	f	g	Display Segments on
0	0 0 0 0	1	1	1	1	1	1	0	a, b, c, d, e, f
1	0 0 0 1	0	1	1	0	0	0	0	b, c
2	0 0 1 0	1	1	0	1	1	0	1	a, b, d, e, g
3	0 0 1 1	1	1	1	1	0	0	1	a, b, c, d, g
4	0 1 0 0	0	1	1	0	0	1	1	b, c, f, g
5	0 1 0 1	1	0	1	1	0	1	1	a, c, d, f, g
6	0 1 1 0	0	0	1	1	1	1	1	c, d, e, f, g
7	0 1 1 1	1	1	1	0	0	0	0	a, b, c
8	1 0 0 0	1	1	1	1	1	1	1	a, b, c, d, e, f, g
9	1 0 0 1	1	1	1	0	0	1	1	a, b, c, f, g
10	1 0 1 0	d	d	d	d	d	d	d	any
11	1 0 1 1	d	d	d	d	d	d	d	any
12	1 1 0 0	d	d	d	d	d	d	d	any
13	1 1 0 1	d	d	d	d	d	d	d	any
14	1 1 1 0	d	d	d	d	d	d	d	any
15	1 1 1 1	d	d	d	d	d	d	d	any

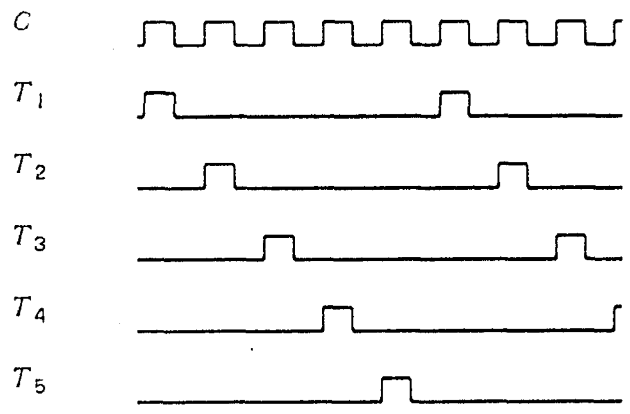
$$\begin{array}{c|cccc} & 00 & 01 & 11 & 10 \\ \hline 00 & 1 & 0 & 1 & 1 \\ 01 & 0 & 1 & 1 & 0 \\ 11 & d & d & d & d \\ 10 & 1 & 1 & d & d \end{array} \qquad \begin{array}{c|cccc} & 00 & 01 & 11 & 10 \\ \hline 00 & 1 & 1 & 1 & 1 \\ 01 & 1 & 0 & 1 & 0 \\ 11 & d & d & d & d \\ 10 & 1 & 1 & d & d \end{array} \qquad \begin{array}{c|cccc} & 00 & 01 & 11 & 10 \\ \hline 00 & 1 & 1 & 1 & 0 \\ 01 & 1 & 1 & 1 & 1 \\ 11 & d & d & d & d \\ 10 & 1 & 1 & d & d \end{array}$$

$$\begin{array}{c|cccc} & 00 & 01 & 11 & 10 \\ \hline 00 & 1 & 0 & 1 & 1 \\ 01 & 0 & 1 & 0 & 1 \\ 11 & d & d & d & d \\ 10 & 1 & 0 & d & d \end{array} \qquad \begin{array}{c|cccc} & 00 & 01 & 11 & 10 \\ \hline 00 & 1 & 0 & 0 & 1 \\ 01 & 0 & 0 & 0 & 1 \\ 11 & d & d & d & d \\ 10 & 1 & 0 & d & d \end{array} \qquad \begin{array}{c|cccc} & 00 & 01 & 11 & 10 \\ \hline 00 & 1 & 0 & 0 & 0 \\ 01 & 1 & 1 & 0 & 1 \\ 11 & d & d & d & d \\ 10 & 1 & 1 & d & d \end{array}$$

$$\begin{array}{c|cccc} & 01 & 01 & 11 & 10 \\ \hline 00 & 0 & 0 & 1 & 1 \\ 01 & 1 & 1 & 0 & 1 \\ 11 & d & d & d & d \\ 10 & 1 & 1 & d & d \end{array}$$

3. Multi-phase clock design. *(22 points)*

A multi-phase clock having 5 timing pulses, T_1, T_2, ..., T_5, is desired as a clock for a computer. The timing diagram is shown below, where the clock outputs T_i are generated from an input signal C.



(a) *(2 points)* What is the minimum number of flip-flops which must be used? How many states does the circuit have?

(b) *(3 points)* Draw a state diagram assuming a state transition at each C pulse. Make a state assignment.

(c) *(17 points)* Using the minimum number of J-K flip-flops with positive-edge clock inputs, realize the 5-phase clock circuit. (Please use the worksheet below.)

Transition table

$c = 0$	$c = 1$

Excitation table

$c = 0$	Excitation

PROGRAMMING PROBLEM

You are working (temporarily) for a medium-sized software firm. A bright young manager has decided that your company should offer a package to keep track of patients' histories for small teams of doctors in private practice. Marketing thinks this is a great idea because (1) there are a lot of doctors out there, (2) they can easily afford to buy a small computer system, or to get time on a time-shared system, and (3) the current system for keeping records is awful. (A history is taken from the patient each time he comes into the office and is added to the top of the patient's file. This can result in a very large amount of paper in which relevant information can get buried. Also, the patient is not always a reliable source of information for the history.)

You are to write a small prototype system to fill this need. You have the latitude to decide whether the target system will run on a microcomputer with floppy disk memory or on a time-sharing system. You may also decide what sort of terminal to assume. The prototype system should be written in a high-level language such as ALGOL W or SAIL on some available computer, but you hope to be able to transfer most of your code directly to the target system, only changing a few modules.

The prototype system is not expected to handle all data the doctor will want to store and access, but it will provide a basis around which the eventual system can be built. It will certainly establish the style of communication that will be used and the internal data structures, so care should be used in designing these. On Tuesday, 1 February, the system will be presented to a committee of management and marketing personnel, and, if approved, it will be released for field tests. An additional consideration for you is the fact that you strongly suspect that you will not be with the company by the time the field tests are completed. (A grant allowing you to do your own research looks like it will come through, or a friend has approached you about setting up your own consulting firm.) Therefore, on Tuesday the program and documentation should be in such a state that any random programmer klutz should be able to take over what you've done and make all necessary modifications with a minimum of bother and rewriting. Finally, you should have a set of test data which "adequately" demonstrates that the program works.

Specifications.

Every time a patient comes into the office the doctor takes a history and physical examination. The history consists of the patient's description of the chronological events leading up to the visit to the doctor. This may span days or years: a cold or the flu may have started only a day ago, while "ulcers" may have smoldered for years. The history is supplemented by a review of the patient's general well being, ranging from changes in weight or appetite to changes in lung function, urinary habits, etc. In addition there may be more distant medical history or prior hospitalizations or major illnesses which may or may not already exist in the patient's file. (You may assume for now that all dates are known exactly.) Finally, the doctor performs a physical examination of the patient. This will vary in completeness depending upon the presenting complaint, but can include examination of the head, eyes, ears, nose, and throat (HEENT), neck, chest, heart, abdomen, genitalia, rectum, and extremities, along with evaluation of the neurological system.

The above description was provided for you by your doctor consultant; use whatever of it you think appropriate for the task. You know for sure though that your system should be able to provide at least the following information on request:

(1) Current and past diseases or conditions the patient has had within a given bodily system. (Systems include nervous, musculo-skeletal, head-eye-ear-nose-throat, gastrointestinal, cardiovascular, respiratory, endocrine, and genitouretal.) You may ignore for now the problem of cross-referencing a condition belonging to one system which may cause problems in another system, e.g. diabetes (endocrine) producing blindness (HEENT).

(2) Medications and associated dates and results.

(3) Allergies.

(4) Relevant information from previous examinations, including significant test results.

Note: This problem is clearly very straightforward to program in a haphazard way. The goal here is not just to get the job done, or even to get the job done efficiently, but to do it within the constraints of the situation proposed. The user interface should be designed carefully: the program should neither step on the doctors' ego nor waste their time. Design decisions and program description should be laid out well for the person who will be modifying and maintaining the program. Any tradeoffs you consider in writing this program should be decided using these and other real-world criteria, and should be documented.

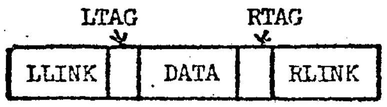
ALGORITHMS AND DATA STRUCTURES

1. Threaded binary trees. *(25 points)*

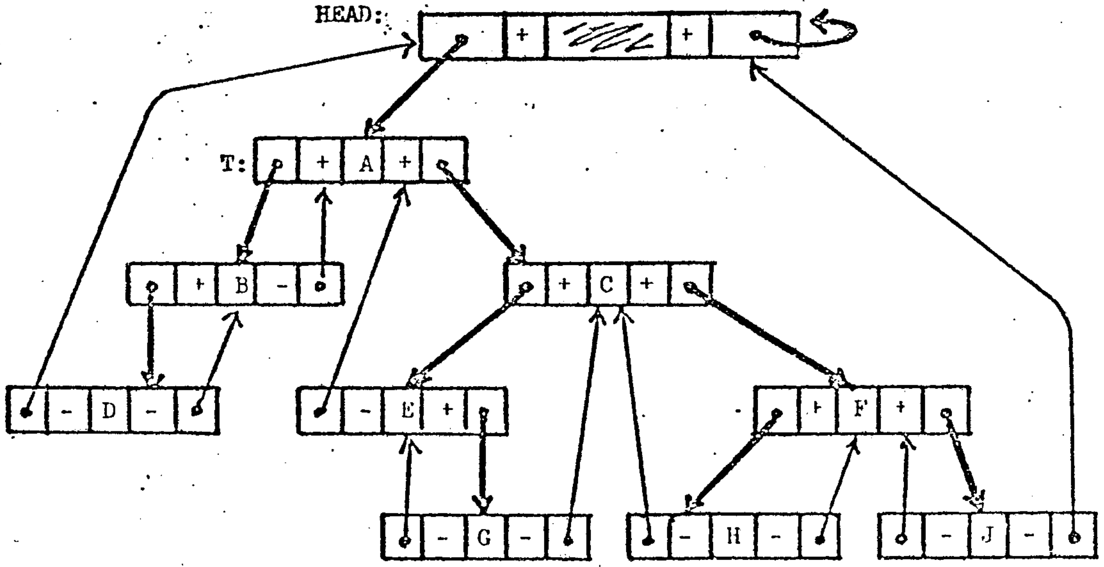
Suppose that T is a left and right threaded binary tree as in Knuth Vol. 1, and that P points to some node in T. Recall that each node of T contains two link fields and two one-bit tag fields as well as any relevant data fields. The conventions are

LTAG(P) = "+" :	LLINK(P) points to P's non-null left child.
LTAG(P) = "-" :	P has a null left child, and LLINK(P) points to P's predecessor in symmetric order.
RTAG(P) = "+" :	RLINK(P) points to P's non-null right child.
RTAG(P) = "-" :	P has a null right child, and RLINK(P) points to P's successor in symmetric order.

There is also a header node at location HEAD with the conventions as shown in the following example tree, where the fields of the nodes are drawn in the form



Example:



Write an algorithm which moves upward in such a tree; that is, write

Algorithm Parent: If P points to a node in a threaded binary tree, this algorithm sets Q to point to P's parent. If P points at the root, Q is set to point to the header node (i.e. Q ← HEAD).

Write the algorithm in Knuthian style. Do not access the DATA fields of the nodes, and do not alter the data structure in any way. Your algorithm should use bounded storage; that is, it should not use an auxiliary stack explicitly or by recursion. Your algorithm should also be relatively efficient; it should run at worst in time proportional to the height of the tree, where the height of a tree is defined to be the number of the deepest non-empty level.

Also, state why your algorithm is correct.

2. <u>NR(k) - CNF satisfiability</u>. *(35 points)*

The following is standard nomenclature for boolean expressions:

A *literal* is either x or $\neg x$ where x is a variable.
A *clause* is a sum (disjunction, OR) of zero or more literals (the empty clause has truth value *false*).

A boolean expression is in *conjunctive normal form* (CNF, "product of sums") if it is a product (conjection, AND) of zero of more clauses; the empty product has truth value *true*. The problem of testing boolean expressions in CNF for satisfiability is well-known to be NP-complete.

Call a CNF boolean expression *k-negation restricted* (NR(k)) if every clause contains at most k negated variables. For example,

$$(\bar{x}_1 + x_2 + x_3)(x_3 + \bar{x}_4 + \bar{x}_5)(\bar{x}_1 + x_2)$$

is NR(2) but is not NR(1). Note that an expression which is NR(0) cannot contain any negations; thus, if all clauses are non-empty, it is trivially satisfied by letting all variables be *true*.

(a) *(10 points)* Show that the satisfiability problem for NR(2) - CNF boolean expressions is NP-complete. *Hint:* Remember that CNF satisfiability with at most three literals per clause is NP-complete.

It is not hard to see that the problem of satisfiability for expressions in NR(1) - CNF is in P. Note that if every clause contains at least two literals, the expression can be satisfied by letting all variables be *true*. Trouble can only come from the empty clause, or from a clause consisting of a single negated variable.

So, suppose an expression of length m is written in the standard encoding on one tape of a multi-tape Turing machine. We scan looking for a trouble-making clause; if none are found, we output *satisfiable* and halt. If we find an empty clause, we output *unsatisfiable* and halt. Otherwise, suppose we find a clause of the form $(\bar{x}_i)$; our only hope is then to make x_i *false*. So, we scan the entire tape, performing the following operations.

(1) If x_i appears in a clause, delete x_i from that clause.

(2) If $\bar{x}_i$ appears in a clause, delete the entire clause from the expression.

Now, by copying onto a work tape, we can eliminate the gaps caused by deletions in $O(m)$ time as well. Thus, we have reduced the problem by at least one literal in time $O(m)$. The total time cost of this deterministic Turing machine is hence $O(m^2)$, and we conclude that NR(1) - CNF satisfiability is in P.

(b) *(25 points)* Design a "fast" algorithm to solve the NR(1) - CNF satisfiability problem on a random access machine. For definiteness, suppose that the problem is initially represented as follows: variables are represented by integers in the range from 1 to n, "not" is represented by arithmetic negation, a clause is represented as a sequence of literals in sequential locations surrounded by "( )", and an expression by a sequential sequence of clauses surrounded by "[ ]". For example,

$$(x_1 + \bar{x}_2 + x_4)(x_3 + x_4 + x_2 + x_1)(\bar{x}_2)$$

would initially be represented as follows.

[	(	1	-2	4	)	(	3	4	2	1	)	(	-2	)	]

Let m denote the number of words in the expression in this representation; we will assume that $m \geq n$.

Design an algorithm and data structures which will test the satisfiability of an NR(1) - CNF expression inputted in the above representation in time at most proportional to m on a random access machine (RAM with uniform cost criterion).

Explain your data structures, drawing pictures if appropriate. Write your algorithm at a high level; for example, you may assume that the grader knows how to insert into a doubly-linked list.

Hint: Start from the Turing machine presented above, and restructure things so that everything can be done in "one pass".

Try your algorithm on the example

$$(x_1)(\bar{x}_1 + x_2 + x_3)(\bar{x}_2 + x_3 + x_4)(\bar{x}_3 + x_4 + x_5)(\bar{x}_4 + x_5 + x_6)(\bar{x}_5)(\bar{x}_6)$$

which is *not* satisfiable.

ARTIFICIAL INTELLIGENCE

1. Representation. *(40 points)*

The following report describes a situation which you will be asked to formalize, as described below.

Report 42-7003M to the intergalactic anthropologists:

On a recent visit to the planet Xmxcv, we noted an interesting development in their use of computers to help in the management of their society. As you may remember from our earlier report (40-6891L), the inhabitants of Xmxcv reproduce communally, so there is no identification of offspring with an individual parent. In order to provide a family structure, they follow a set of adoption customs which have been passed down from antiquity, and are strictly obeyed.

(1) Every child must be the adopted offspring (called *kyd* in their language) of one and only one adult.

(2) At all times, each family must have exactly one kyd which is designated as a *bygshot*. A bygshot is said to *sqysh* each of the other kyds (if there are any) in the family. A kyd who is not a bygshot can trade roles with the bygshot in the same family through a custom called *swyching*. The one who was the bygshot becomes an ordinary kyd in the family, and the other kyd becomes the bygshot.

(3) Any adult may adopt a kyd out of its current family, by carrying out a custom called *glomming*. The result is that the kyd is in the family of the adult who glommed and no longer in the family it was in previously. A kyd's status as a bygshot (or not one) cannot be changed as the result of a glom.

(4) Each village has a single adult (called the *burokrat*) who is in charge of all glomming and swyching. He will only do one ceremony per day, carried out at precisely midnight in front of the assembled elders. The ceremony can be either a glom or a swych.

(5) Life proceeds so slowly on Xmxcv that for all practical purposes, no new adults or kyds ever need be accounted for, and "once a kyd, always a kyd".

This set of customs has worked for many years, but is beginning to bog down under the pressure of the population explosion (see our report 40-6802L). As families get larger, and there are more of them, it has become more complicated to figure out the proper sequence of gloms and swyches to reach a desired arrangement. As a result, the elders have formed two committees, one of logicians and one of programmers, to help clarify the situation. The logicians are attempting to provide a formal axiomatization of the system, in order to be able to decide whether certain kinds of processes are possible or impossible, while the programmers are building a computer program which will help plan the right sequence of ceremonies to get a family arranged in a desired way. We will report later on the success of these ventures.

Exploration Team 98A43, Intergalactic Council

Read the report carefully and choose *one* of the following two problems:

(a) You are a member of the logicians team. Write a set of axioms in first order predicate calculus which formally characterize the relevant parts of the customs described above. From these axioms outline a proof (using any proof methods you like) that *No kyd can be a bygshot on the day before or following the midnight at which it is glommed from one family into another.* You do not need to go through all of the symbolic manipulations of the proof, but it should be clear from your description of it exactly which axioms will be used when.

(b) You are a member of the programmers team. Write a goal-reduction scheme which plans a sequence of ceremonies which will achieve a specific goal. Your answer can be either in the form of a program in an AI language like MicroPlanner, or as a set of operators with a difference-operator table, sets of preconditions, and add and delete lists, as for a GPS or STRIPS-like scheme. The goal given as input to the program is always of the form: *Get specific kyd x to sqysh specific kyd y in a family whose adult is z.* Demonstrate (with an informal trace of its behavior) how your program would work on the following situation:

Family 1:	adult = Axcvbv	Family 2:	adult = Excvbv
	bygshot = Bxcvbv		bygshot = Fxcvbv
	other kyds = Cxcvbv, Dxcvbv		other kyds = Gxcvbv

Goal: Bxcvbv sqyshes Fxcvbv in family of Excvbv

2. System integration in AI programs. *(10 points)*

In organizing AI systems for language, speech, and vision, there has been a debate between those who want to keep a strictly structured hierarchical organization, in which each component can be described and run independently, and those who favor a more heterarchical organization, in which component boundaries are not as strongly structured.

(a) For the problem areas of speech understanding and visual scene analysis, describe the most common division into components.

(b) Briefly list the advantages and disadvantages of the two conflicting approaches.

3. Applicability of AI techniques. *(10 points)*

Consider three different kinds of AI programs: a chess player, a language understanding system, and a travel route chooser. For each of the following techniques, answer two questions:

(1) In which of these programs does the technique seem most applicable? Give examples, if you know of places where it has been applied.

(2) In which does it not seem applicable, and why?

Techniques:

(a) the alpha-beta minimax strategy

(b) the A* search strategy

(c) goal reduction

HARDWARE

1. Architecture. *(10 points)*

(a) What is a cache (buffer) memory? What properties should a program have to gain the most benefit from a cache?

(b) What is a pipelined execution unit? When does pipelining work particularly well?

2. Combinatorial circuits. *(20 points)*

On the following page is circuit F, with 3 inputs a, b, c and 3 outputs x, y, z. This circuit is constructed from NOT gates, and AND and OR gates with either 2 or 3 inputs.

(a) *(5 points)* Describe the behavior of the lines u and v as functions of a, b, c. *Hint:* Note that the values of u and v are symmetric functions of a, b, and c; that is, they depend only on the number of inputs which are 1.

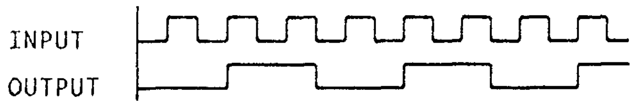
(b) *(10 points)* Construct a circuit G which has the same input-output behavior as F but uses at most 5 gates. Beware, it is easy to make careless mistakes on this.

(c) *(5 points)* Suppose that different varieties of gates cost different amounts; when would circuit F be superior to circuit G? That is, what is interesting about circuit F?

(d) *(0 points)* Describe your reaction to the existence of circuit F; are you surprised?

3. Sequential circuits. *(15 points)*

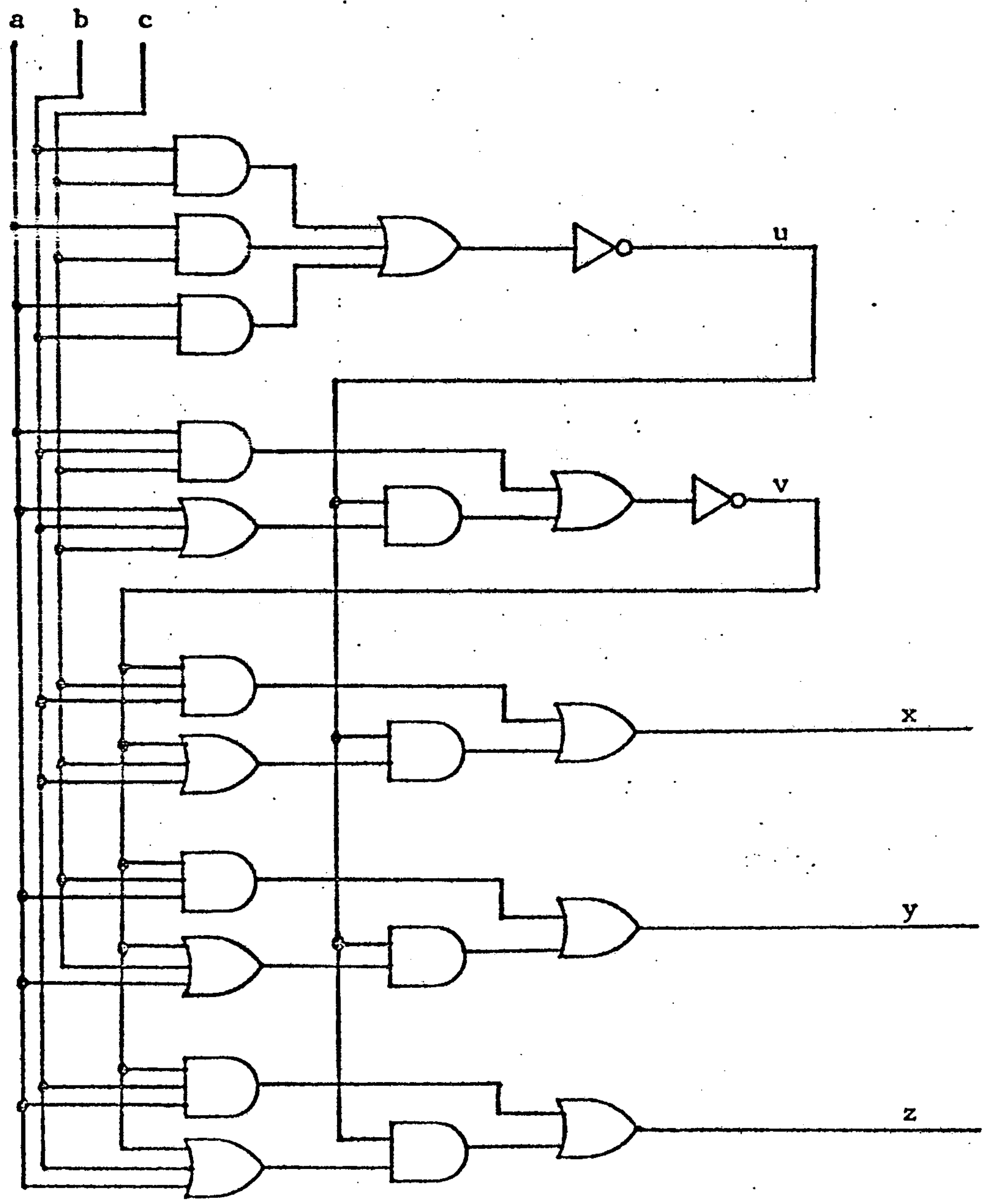
Design a one input, one output sequential circuit with the following behavior:



Use two negative-edge triggered JK master-slave flip-flops as your memory elements. Describe the sequence of state changes that your circuit will go through.

Design the circuit so that it does not need initialization; that is, if the flip-flops begin in any combination of states, the circuit will eventually behave as shown above.

Hint: Begin by implementing a period three counter with INPUT as the clock.



Circuit F

4. Excess-3 adder. *(15 points)*

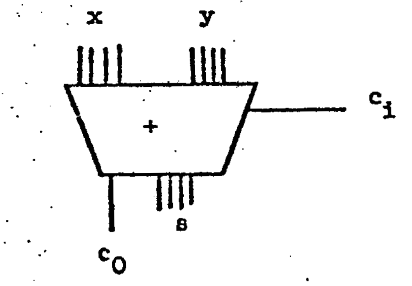
Excess-3 code is used to represent the decimal digits from 0 to 9 using 4 bits as follows:

0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

(a) *(3 points)* Name a good characteristic of excess-3 code.

(b) *(12 points)* You are given the following circuits as building blocks:

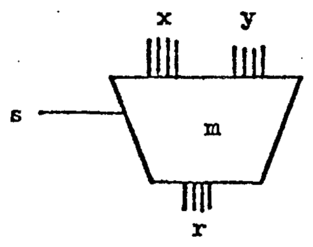
(i) 4-bit binary adder: takes two 4-bit binary numbers as input and a 1-bit carry-in, and produces a 4-bit binary sum and a 1-bit carry-out.



$s = x + y + c_i \bmod 16$

$c_0 = x + y + c_i \text{ div } 16$

(ii) 2-to-1 multiplexor: takes two 4-bit quantities and selects one of them according to switch *s*.



$r = (\text{if } s = 1 \text{ then } y \text{ else } x)$

Using the above building blocks, design a circuit which takes as input two 4-bit decimal digits represented in excess-3 code and a 1-bit carry-in, and produces as output the 4-bit sum digit in excess-3 code and a 1-bit carry-out.

NUMERICAL ANALYSIS

1. Solution of linear systems. *(5 points)*

Give at least two reasons why one might prefer an iterative scheme such as Jacobi, Gauss-Seidel, or successive overrelaxation (SOR) over a finite algorithm like Gaussian elimination (LU decomposition) for solving $A\underset{\sim}{x} = \underset{\sim}{b}$.

2. Condition of linear systems. *(5 points)*

Consider the linear system $A\underset{\sim}{x} = \underset{\sim}{b}$. We consider the perturbed system $A(\underset{\sim}{x}+\delta\underset{\sim}{x}) = (\underset{\sim}{b}+\delta\underset{\sim}{b})$. It can be shown that $\|\delta\underset{\sim}{x}\|/\|\underset{\sim}{x}\| \le \mu(A)\|\delta\underset{\sim}{b}\|/\|\underset{\sim}{b}\|$ where $\mu(A)$ is the condition number of A. What does "scaling" the matrix A mean? How can scaling help to decrease the sensitivity of the system of linear equations to perturbations in the right-hand side $\underset{\sim}{b}$?

3. Polynomial interpolation/approximation. *(6 points)*

Let f be a given continuous function on $[-1,1]$. Let $\mathbb{P}_n$ denote the space of polynomials of degree n. Further let $E_n(f) = \inf\{\|f-P\|_\infty \mid P \in \mathbb{P}_n\}$. Suppose you are given $\{(x_i,f_i)\}_{i=0}^{n}$, where $f_i = f(x_i)$. Let $P_n \in \mathbb{P}_n$ interpolate $\{(x_i,f_i)\}$. In general, P_n will not approximate f very well because of Runge's phenomenon. What is Runge's phenomenon? Suppose you are allowed to select the x_i's. Can you give a practical scheme for picking the x_i's so that the resulting P_n is close to being optimal, i.e., $\|P_n-f\|_\infty$ is not much larger than $E_n(f)$?

4. Non-linear equations. *(21 points)*

Remember that Newton-Raphson iteration can be derived using Taylor's theorem in the following fashion

(1) $0 = f(\alpha) = f_n + \epsilon_n f'(x_n+\theta_1\epsilon_n) \quad (0 \le \theta_1 \le 1)$
$\approx f_n + \epsilon_n f_n'$

where $\epsilon_n = x_n - \alpha$, x_0 and f are given, and α is a root of f. Hence $\epsilon_n = -f_n/f_n'$, $x_{n+1} = x_n + \epsilon_n$ is an obvious algorithm.

The secant method is often used when f' is unavailable or is expensive to evaluate. Sometimes however, cheap higher order derivatives of f are available and it is relevant to consider higher order methods.

(a) *(16 points)* Construct an iterative scheme using f, f' and f'' in a manner analogous to the Newton-Raphson approach above. Your scheme should be of the form $\epsilon_n = \phi(f_n, f_n', f_n'')$ where ϕ is a rational function of its arguments (no square roots!). *Hint:* Consider (1) and note that $f'(x_n+\theta_1\epsilon_n) = f_n' + \epsilon_n f''(x_n+\theta_2\epsilon_n)/2$ where $0 \le \theta_2 \le 1$.

(b) *(5 points)* Consider $f(x) = 1/x - 1$. Set $x_0 = 3/2$. Apply three steps of Newton-Raphson and three steps of your scheme from (a) to this f, with the given initial guess. Remark on the observed convergence behavior (your scheme should be better than Newton-Raphson!).

5. Periodic splines. *(23 points)*

Let $\{(x_i, f_i)\}_{i=1}^{n}$ be given. Consider a cubic spline S with knots at the x_i's which satisfies $S(x_i) = f_i$ $(i = 1, 2, \ldots, n)$. Let $\sigma_i = S''(x_i)/6$ $(i = 1, 2, \ldots, n)$. For $x_i \le x \le x_{i+1}$ we know from the linearity of S'' that

$$S''(x) = \frac{6}{h_i}[(x_{i+1}-x)\sigma_i + (x-x_i)\sigma_{i+1}], \qquad \text{where } h_i = x_{i+1}-x_i.$$

It can be shown that for $x_i \le x \le x_{i+1}$,

$$S'(x) = -3\frac{(x_{i+1}-x)^2}{h_i}\sigma_i + 3\frac{(x-x_i)^2}{h_i}\sigma_{i+1} + \frac{f_{i+1}-f_i}{h_i} - (\sigma_{i+1}-\sigma_i)h_i$$

and

$$S(x) = \frac{(x_{i+1}-x)^3}{h_i}\sigma_i + \frac{(x-x_i)^3}{h_i}\sigma_{i+1} + (f_i - \sigma_i h_i^2)\frac{(x_{i+1}-x)}{h_i} + (f_i - \sigma_{i+1}h_i^2)\frac{(x-x_i)}{h_i}.$$

From these results it is possible to write down a system of linear equations for the σ_i's given boundary conditions such as $S''(x_1) = S''(x_n) = 0$ for natural splines.

Consider the so-called periodic boundary conditions for S given by $D^\alpha S(x_1+) = D^\alpha S(x_n-)$ for $\alpha = 0,1,2$ and $D = (d/dx)$. Write down the resulting linear system for the σ_i's in the equidistant case, i.e., each $h_i = h > 0$. Would Gaussian elimination without pivoting be stable for solving the resulting linear system? (Be sure to verify all of the proper continuity conditions.)

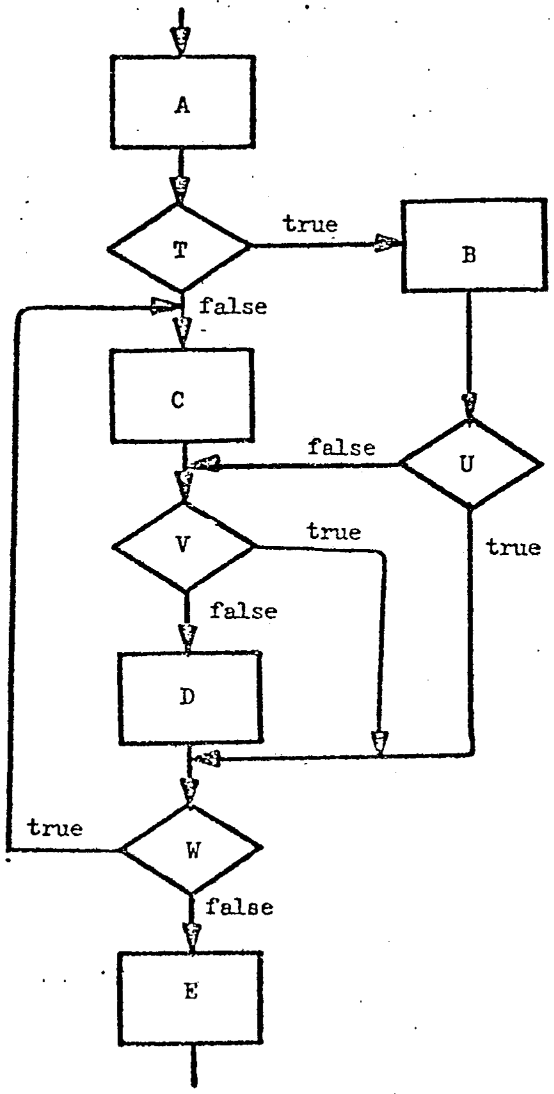
SYSTEMS

1. Storage management. *(5 points)*

A recursive program is available to multiple processes on a reentrant basis. Present an adequate storage management scheme for parameter values and local variables.

2. Structured programming. *(10 points)*

Restructure the flowchart below according to the precepts of structured programming.



3. Fragmentation. *(10 points)*

In a paged system some memory is unusable due to internal fragmentation and due to the need to have a resident page table (one word per page). Programs and data occupy segments of average lengths s which begin at page boundaries.

(a) What is the optimal page size p given only the above memory usage considerations?

(b) What other factors should be considered and what cost factors need to be known in order to obtain a more realistic evaluation of an optimal page size?

4. Parsing of expressions. *(15 points)*

(a) Write in BNF syntax rules adequate to describe realistic arithmetic expressions containing: var + - / * ↑ ( ).

(b) Sketch the parse tree for your system for the expression:

$$A * (B + C) + D - (-E + F) \uparrow G$$

(c) Is this expression parsable using operator precedence?

5. Exponential service time. *(5 points)*

What criteria should be applied to observations of service times to warrant the frequently made assumption in scheduling that service times have an exponential distribution?

6. Deadlock. *(10 points)*

There are p processes competing for r identical resource units. Each process needs a maximum of m resource units ($m \le r$).

(a) Under what condition can no deadlock occur?

(b) Give an example of a request sequence leading to deadlock.

(c) Write a resource allocation algorithm which will prevent deadlock.

7. Error recovery. *(5 points)*

What aspects of grammars contribute to good error recovery?

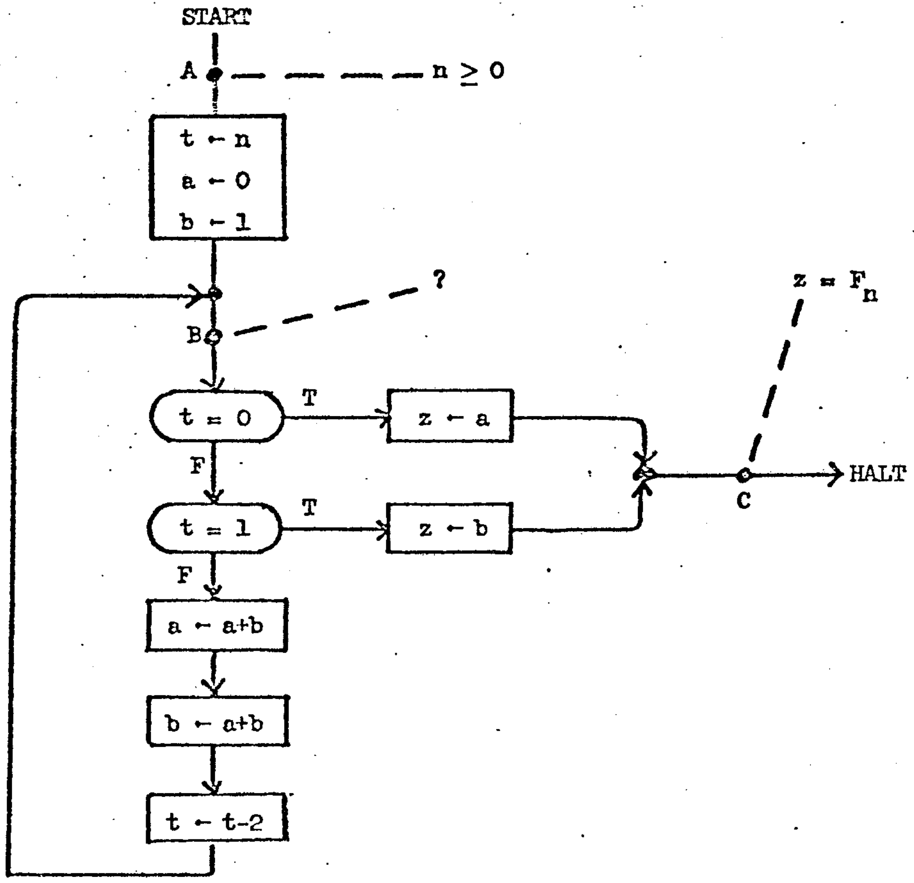
THEORY OF COMPUTATION

1. Inductive assertion. *(10 points)*

The Fibonacci numbers are a famous sequence of integers defined by the recurrence

$F_0 = 0,\ F_1 = 1$, and
$F_{n+2} = F_{n+1} + F_n$ for $n \geq 0$.

Thus, the sequence begins 0, 1, 1, 2, 3, 5, 8, 13, ... The following is a flowchart program over the integers which computes the Fibonacci numbers; the input is n, the output is z, and variables t, a, and b are temporaries.



Supply an inductive assertion for the loop cut-point B which will enable a proof of partial correctness.

2. Decidability true/false. *(10 points)*

Are the following problems decidable? (answer true or false)

(a) *(3 points)* For any word w and type-0 grammar G, whether or not $w \in L(G)$.

(b) *(3 points)* For any Turing machine M and any input word w of length n, whether or not M will halt within n^5 steps.

(c) *(4 points)* For any integer m, whether or not for all n in the range $3 \leq n < m$, the equation $x^n + y^n = z^n$ has no solution in integers x, y, z; that is, whether or not Fermat's Conjecture holds for all exponents less than m.

3. Monotone machines. *(15 points)*

We will define a Monotone automaton to be a machine with a finite state control and finite input tape. The head can move both ways on the input tape. Initially the input has the form $\$w¢$ where $w \in \{0,1\}^*$, and the $\$$ and ¢ symbols serve as end-markers. In addition to reading, the head can also change a 0 into a 1, but *not* a 1 into a 0.

Note that the end-markers allow the machine to avoid moving off the ends of the input tape unawares. The symbols $\$$ and ¢ are used always and only for this purpose.

Remark: Two-way read-only automata are known to be no more powerful than normal, one-way finite automata.

(a) *(10 points)* Find a language $L_1 \subseteq \{0,1\}^*$ which is not regular, but which is recognized by a Monotone automaton. Sketch the actions of an automaton which recognizes your L_1.

(b) *(5 points)* Find a language $L_2 \subseteq \{0,1\}^*$ which is not recognized by any Monotone automaton, but which is context free. You may use the following lemma without proving it.
Lemma. Let k be any nonnegative integer. If $L \subseteq \{0,1\}^*$ is a language that is recognized by some Monotone machine, and all words in L have at most k zeros, then L must be regular.

4. Context-free grammar. *(25 points)*

Let G be the context-free grammar with start symbol S, nonterminals $\{S, A, B\}$, terminals $\{0,1\}$, and productions

$$S \to A \mid B$$
$$A \to 0 \mid B0B$$
$$B \to 1 \mid A1A$$

(a) *(15 points)* Prove that G is ambiguous.

(b) *(10 points) Conjecture*: No string in $L(G)$ contains four consecutive 0's or four consecutive 1's.

Either give a proof of this conjecture, or find a counterexample.

PROGRAMMING PROBLEM

Let t be a positive integer. An *arithmetic progression of length t* is a sequence of the form $<a, a+d, a+2d, \ldots, a+t-1d>$, where a and d are integers and $d > 0$. For example,

> <1, 3, 5, 7> is an arithmetic progression of length 4,
> <2, 11, 20> is an arithmetic progression of length 3,
> <6> is an arithmetic progression of length 1, and
> <5, 7, 10> is not an arithmetic progression.

Let n be a positive integer, and let $S_n = \{1, 2, 3, \ldots, n\}$. Consider coloring each of the integers in S_n either red or blue; an assignment of exactly one of these two colors to each element of S_n will be called a *two-coloring* of S_n. Note that two-colorings of S_n correspond to partitions of S_n into two disjoint and exhaustive subsets, the red elements and the blue elements. In particular, there are 2^n two-colorings of S_n.

Let r and b also be positive integers. A two-coloring of S_n will be called *(r, b)-constrained* if and only if both of the following conditions hold:

(i) there is no entirely red arithmetic progression of length r, and
(ii) there is no entirely blue arithmetic progression of length b.

For example, consider the following two-coloring of S_8:

1	2	3	4	5	6	7	8
red	red	blue	blue	red	red	blue	blue.

Note that it is (3,3)-constrained. However, it is not (4,2)-constrained, since there are six entirely blue arithmetic progressions of length 2.

We will say that S_n is *(r, b)-colorable* if and only if there exists an (r, b)-constrained two-coloring of S_n. The above example demonstrates that S_8 is (3,3)-colorable. Now, it is not too difficult to check by hand that S_9 is not (3,3)-colorable; that is, any two-coloring of S_9 must contain a monochromatic arithmetic progression of length 3. The proof is a case analysis, and the reader is encouraged to try her hand at it, to improve her grasp of these concepts.

Note that, for r and b fixed, it becomes increasingly difficult to find an (r, b)-constrained two-coloring of S_n as n gets larger. As more and more arithmetic progressions appear in S_n, it becomes harder and harder to avoid making long ones monochromatic. In fact, in 1925, Van der Waerden proved that, for every r and b, there exists an integer n such that S_n is not (r, b)-colorable. Hence, for each r and b, there exists a smallest integer n such that S_n is not (r, b)-colorable; call this integer $f(r, b)$. For example $f(3,3) = 9$.

If $n \leq m$ then $S_n \subseteq S_m$, and any two-coloring of S_m can be restricted to a two-coloring of S_n. Furthermore, if the two-coloring of S_m is (r, b)-constrained, the restricted two-coloring of S_n will be (r, b)-constrained also. Thus, if $n \leq m$ and S_m is (r, b)-colorable, so is S_n. From this, we can see that, for any r, b, and n, the set S_n is (r, b)-colorable if and only if $n < f(r, b)$.

Some of the known values of f are given in the following table.

b \ r	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	3	6	7	10	11
3	3	6	9	18	22	32
4	4	7	18			
5	5	10	22			
6	6	11	32			

Write a program to compute $f(3,4)$, $f(3,5)$, $f(3,6)$; you may of course check your answers against the above table. In addition, in each of the three cases, output at least one (r, b)-constrained coloring of $S_{f(r,b)-1}$.

Your programs will be graded for correctness, clarity, and efficiency. Give some thought to various algorithms and data structures before beginning to code; there is usually more than one way to organize combinatorial computing. In the program's documentation, include a description of your algorithm and data structures, and a justification of their appropriateness.

The languages of choice for the project are Algol W or SAIL.

Winter 1978 Comprehensive Exam

ALGORITHMS AND DATA STRUCTURES

1. Sorting. *(25 points)*

You are given an array of length N of records, each with a field containing a key chosen from an ordered set, and you are asked to sort the array into nondecreasing order. Assume a record is large compared with a key or a pointer field. In each of the problems below, you are given further characteristics of the problem, such as the nature of the keys, the size of N, the amount of auxiliary storage allowed (beyond a small fixed amount for bookeeping), and the desired behavior of the algorithm.

Suggest a sorting algorithm, describe how it works, and explain why it is appropriate to the task. You need only say enough to convince the grader that you understand the issues; do not write complicated programs.

(a) Arbitrary keys, large N (say $N \geq 1000$), no auxiliary storage, good worst-case time.

(b) Keys are 5-digit integers (zip codes), large N, (N + a fixed constant) pointer fields allowed, linear time.

(c) Arbitrary keys, large N, $O(\log N)$ words of auxiliary storage allowed, good average-case time (assuming the input is in random order).

(d) Arbitrary keys, $N = 4$, no auxiliary storage.

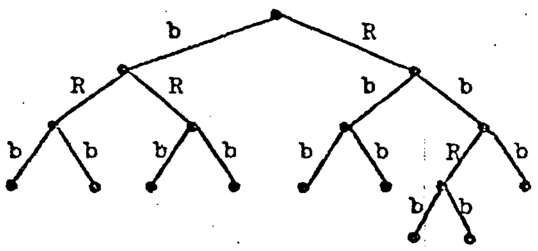
(e) Arbitrary keys, large N, no auxiliary storage, good average-case time (assuming each record is expected to be very close to its proper place).

2. RB trees. *(35 points)*

An *RB tree* (discovered by L.G.R.S. Arby, inventor of the roast beef sandwich) is a binary tree whose edges are colored red or black such that

(i) Every internal node has two descendants.
(ii) Every edge leading to a leaf is black.
(iii) No path from the root to a leaf passes through two consecutive red edges.

Define the *B-level* of a node to be the number of black edges on the path from the root to the node. An *RB balanced tree of height h* is an RB tree whose leaves all have B-level h. For example, the tree shown below is an RB balanced tree of height 2.



(a) Suggest a representation for this data structure. Draw a picture of a model node, label its fields, and indicate any conventions. Draw a pictuıe of the above example using your representation. *(5 points)*

(b) Write a Pidgin-Algol procedure which, when given a node as input, checks the form of the tree rooted at that node. (You may assume that it really is a tree, i.e. there are no cycles and no multiple in-edges.) Your procedure should produce one of the following outputs:

(i) "There is a node with only one descendant."
(ii) "There is a red edge leading to a leaf."
(iii) "There are consecutive red edges."
(iv) "There are leaves of B-levels *m* and *n*."
(v) "The tree is RB balanced of height *h*."

where *m*, *n* and *h* are integers whose values should be given. *(20 points)*

(c) What are the minimum and maximum number of leaves in an RB balanced tree of height *h*? *(10 points)*

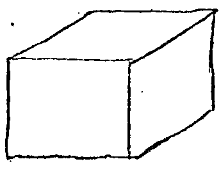
ARTIFICIAL INTELLIGENCE

1. Vision. *(40 points)*

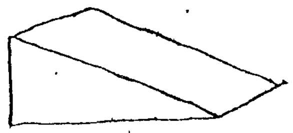
Suppose we wish to analyze a blocks world scene. Since we know that noise, sub-optimal lighting conditions, etc. often make it impossible to process an image correctly, we have decided to integrate knowledge from three sources. In addition to the usual digital camera, we have also installed a laser range finder (in the same position as the camera) to provide depth information, and directly overhead we have installed a sonic range finder, to provide information about the height of objects above the table.

Our blocks world consists of one or more blocks sitting on a table top with one or more walls in the background. The scene can contain any of the following objects, and the object may appear in any orientation:

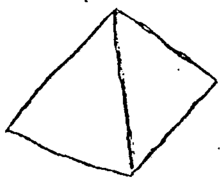
Block:



Wedge:



Pyramid:



(a) The following table consists of the light intensity values returned by our digital camera (normalized in some weird fasion). Use the tracing paper provided to produce a line drawing using this data. We suggest that you mark the corners of your drawing so you can overlay your line drawing on the tables of values returned by the range finders. Is there enough information in this data to determine exactly what is in the blocks world scene? Why or why not? *(10 points)*

4	3	3	3	4	3	4	4	5	4	3	4	5	4	5	4	3	3	3	4	3	4	3	4	5	7	7	8	9	9
5	4	5	3	5	4	3	5	4	3	3	3	4	3	4	3	4	3	4	3	3	5	4	5	3	7	8	8	8	9
4	4	4	3	3	4	3	1	2	2	1	2	2	1	2	0	5	4	5	3	5	4	3	4	4	7	7	8	8	9
5	4	5	3	4	4	4	0	1	1	0	1	2	0	1	1	4	5	5	4	5	5	4	5	5	7	8	7	8	9
5	4	3	3	3	4	3	1	2	2	1	0	1	1	0	1	4	4	3	3	4	4	4	4	4	7	7	7	8	8
4	3	4	3	3	5	4	2	1	1	2	1	2	2	2	1	3	3	3	4	3	3	3	3	5	7	8	8	8	9
5	4	3	4	3	4	3	2	1	0	1	0	2	1	7	8	7	8	9	8	8	7	4	5	5	8	8	9	8	9
4	3	3	5	4	5	4	1	0	1	1	1	1	8	7	7	8	9	9	8	8	9	3	5	5	7	7	8	8	9
4	4	4	3	4	5	4	1	2	2	3	2	8	7	7	8	8	9	8	8	9	8	3	5	4	7	8	8	9	9
5	4	3	3	3	4	3	3	4	4	7	8	9	8	8	7	7	8	9	8	9	9	4	4	5	7	8	8	8	9
4	3	4	4	3	4	3	4	5	7	8	7	8	9	9	8	8	7	8	9	9	8	5	4	5	7	7	7	8	8
4	3	3	5	4	5	4	3	8	7	7	8	8	9	8	9	8	7	7	8	8	9	3	4	5	8	7	8	8	9
3	4	3	2	3	3	2	3	2	3	4	3	4	3	2	2	8	9	8	9	9	8	9	8	7	7	8	8	9	9
3	4	3	2	2	2	3	4	3	4	3	4	3	2	2	2	7	8	8	9	8	9	8	8	7	7	8	8	9	9
2	3	2	3	2	2	4	3	4	3	2	3	2	2	2	3	8	8	9	8	8	8	7	7	8	8	7	8	8	8
3	3	2	2	3	3	3	3	3	4	3	2	3	3	3	4	9	9	8	7	7	8	7	8	9	9	7	8	8	9
4	3	4	2	4	3	2	3	3	4	3	3	2	2	3	3	8	8	7	8	7	8	8	9	8	9	8	8	8	8
7	3	3	2	3	4	3	3	2	3	4	3	2	3	2	2	7	8	8	7	8	9	8	9	9	8	7	7	8	9
8	9	8	2	3	4	3	2	3	4	3	2	3	3	2	3	8	7	8	8	8	9	8	8	9	8	7	7	8	8
9	8	7	7	8	3	3	2	2	3	3	3	3	4	3	2	7	8	8	7	8	8	9	8	9	8	8	7	8	9
8	7	8	8	9	8	3	2	3	4	3	3	2	3	4	3	8	7	8	8	9	8	9	8	8	8	8	8	7	8
7	8	8	7	8	9	8	9	9	8	7	7	8	9	8	7	8	8	8	9	8	8	9	9	7	7	8	8	7	9
9	7	8	8	7	7	9	9	8	8	9	8	8	8	7	8	9	8	7	7	8	9	9	8	9	8	7	8	8	7
8	9	8	8	7	8	8	8	9	8	8	9	8	7	7	8	8	8	7	8	8	9	8	7	8	8	9	8	9	8

(b) The following table is the data returned by the laser range finder. (The smaller numbers denote points closer to the source) What contribution does this knowledge make to analyzing the image? Add any new information to your line drawing. *(5 points)*

4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	3	3	2	1	0
4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	3	3	2	1	0
4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	3	3	2	1	0
4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	3	3	3	3	3	3	3	3	3	3	3	3	2	1	0
5	5	5	5	5	5	5	5	5	5	5	5	5	5	4	4	4	4	4	4	4	4	4	4	4	4	3	2	1	0
5	5	5	5	5	5	5	5	5	5	5	5	5	5	4	4	4	4	4	4	4	4	4	4	4	4	3	2	1	0
5	5	5	5	5	5	5	5	5	5	5	5	5	1	1	1	1	1	1	1	1	2	4	4	4	4	3	2	1	0
5	5	5	5	5	5	5	5	5	5	5	5	1	1	1	1	1	1	1	2	3	4	4	4	4	4	3	2	1	0
6	6	6	6	6	6	6	6	6	6	6	1	1	1	1	1	1	1	1	2	3	5	5	5	5	5	3	2	1	0
6	6	6	6	6	6	6	6	6	6	1	1	1	1	1	0	0	0	1	2	3	5	5	5	5	5	3	2	1	0
6	6	6	6	6	6	6	6	6	0	0	0	0	0	0	0	0	0	1	2	3	5	5	5	5	5	5	3	1	0
6	6	6	6	6	6	6	6	0	0	0	0	0	0	0	0	0	1	2	3	4	5	5	5	5	5	4	3	1	0
6	6	6	6	6	6	6	0	0	0	0	0	0	0	0	0	0	1	2	3	4	5	5	5	5	4	3	2	1	0
5	5	5	5	5	5	5	0	0	0	0	0	0	0	0	0	1	1	2	3	4	4	4	4	4	4	3	2	1	0
5	5	5	5	5	5	5	0	0	0	0	0	0	0	0	0	0	1	2	3	4	4	4	4	4	4	3	2	1	0
4	4	4	4	4	4	4	0	0	0	0	0	0	0	0	0	0	1	2	3	3	3	3	3	3	3	3	2	1	0
4	4	4	4	4	4	4	1	1	1	1	1	0	0	0	0	0	1	2	3	3	3	3	3	3	3	3	2	1	0
3	3	3	3	3	3	3	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	1	0
3	3	3	3	3	3	3	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	1	0
3	3	3	3	3	3	3	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	1	0
2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
2	2	2	2	2	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	2	2	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(c) The following table is the set of values returned by the sonic range finder located directly over the table. The coordinates of each value have been transformed so that the scene appears to be viewed from exactly the same position as the camera and laser range finder. Since there are some surfaces whose values are undefined under this transformation, the undefined values are marked with an X. The values denote the vertical distance from the table top. What contribution does this knowledge make to analyzing the image? Add the new information to the line drawing. *(5 points)*

X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	1	1	1	1	1	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	1	1	1	1	1	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X	X	X	1	1	1	1	1	1	1	1	X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	X	1	1	1	1	1	1	1	1	1	X	X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	X	1	1	1	1	1	1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X
X	X	X	X	X	X	X	X	1	1	1	1	1	1	1	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X
0	0	0	0	0	0	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	0	0	X	X	X	X	X
0	0	0	0	0	0	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	0	0	X	X	X	X	X
0	0	0	0	0	0	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	0	0	0	X	X	X	X
0	0	0	0	0	0	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	0	0	0	0	X	X	X	X
0	0	0	0	0	0	0	X	X	X	X	X	X	X	X	X	X	X	X	X	0	0	0	0	0	0	0	X	X	X
0	0	0	0	0	0	0	X	X	X	X	X	X	X	X	X	X	X	X	0	0	0	0	0	0	0	0	X	X	X
0	0	0	0	0	0	0	X	X	X	X	X	X	X	X	X	X	X	X	0	0	0	0	0	0	0	0	0	X	X
0	0	0	0	0	0	0	X	X	X	X	X	X	X	X	X	X	X	0	0	0	0	0	0	0	0	0	0	X	X
0	0	0	0	0	0	0	X	X	X	X	X	X	X	X	X	X	0	0	0	0	0	0	0	0	0	0	0	0	X
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

(d) Is there a shadow in the scene? If so, point out which line or lines are actually pseudo-edges due to the shadow. Give two examples of how a shadow can provide useful information in analyzing an image. *(5 points)*

(e) Label the edges and vertices in the line drawing using the Huffman-Clowes vertex and edge labeling scheme. In general, why is this information useful? *(5 points)*

(f) Which of the three possible objects is in the scene? How would a program figure out which object it is? *(10 points)*

2. Short answer questions. *(10 points)*

For each of the following terms, briefly explain what it means, and say why the concept has been useful or important to AI research.

(a) pattern directed invocation
(b) augmented transition networks
(c) frame problem (as discussed by McCarthy and Hayes)
(d) blackboard
(e) means-ends analysis

3. Robotics (of sorts). *(10 points)*

The following is an excerpt from an article appearing in STARLOG magazine, issue no. 11, dated January, 1978:

> *If Anthony Reichelt has his own way, there will soon be a robot in every home. As president of Quasar Industries, Reichelt is planning on marketing a 5'2" domestic helper that would, among other things, serve dinner, answer the door, babysit, vacuum and polish floors, mow lawns and trim hedges. And that's just the beginning, according to Reichelt. "We have modifications in store that allow him to read fairy tales to children, teach French, climb stairs, monitor for smoke and fire and serve as a burglar alarm. The only thing he won't do is windows. But then again, he won't take Thursdays off either."*
>
> *The ambulatory machine, which has been in research and development for eight years, will retail for about $4,000 and be ready for stores in less that two years.*

Quite understandably, even people who are not acquainted with the field of artificial intelligence have been rather skeptical about their claims. When pressed for details, the salesmen usually back down slightly, and talk about their robot's "limitations". They say that it can understand speech, but it is restricted to a vocabulary of 250 or so words. It can see, but its vision becomes a little fuzzy about 10 feet away.

Even given the stated "limitations", is this robot capable of doing a reasonably competent job of, say, teaching French, given the current state of the art in artificial intelligence? Discuss what problems would be encountered by a robot working in that domain, and which problems currently can and cannot be solved.

HARDWARE

1. Short-answer questions. *(10 points)*

Describe briefly:

(1) base address register
(2) Gray code
(3) PROM
(4) direct memory access (DMA)
(5) programmable logic array (PLA)
(6) bit
(7) tri-state logic
(8) multivalue logic
(9) microcode
(10) CMOS (complementary metal oxide semiconductor)

2. Describe two hardware techniques which can be used to increase the information transfer rate between memory and processor. *(6 points)*

3. What is the 1's complement of 1010? *(1 point)*
What is the 2's complement of 1010? *(1 point)*

4. List four types of addressing modes. *(4 points)*

5. Given that computers do indeed fail because of hardware failures, briefly discuss two methods to protect computers against these events. *(4 points)*

6. Give two two-input gates, each one forming a complete set by itself. (A set of gates is complete if any Boolean function can be realized using only gates from that set.) *(4 points)*

7. In the land of the little green men from Mars, the following statement is true:

one of the roots of $x^2 - 129x + 716$ is 123.

How many fingers do they have? *(5 points)*

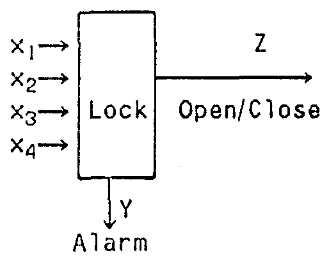
8. Assume you want to build a digital lock. The first method is to build a combinational lock.

You want the lock to be shut whenever the input combination does not match the key.

For added protection, you want an alarm to ring whenever somebody keys in an input combination that differs from the key by at least one digit.

However, since you do not want to keep the alarm ringing, you provide the lock with a "rest state", for which the lock is closed and the alarm is silent.

This can be represented as:



Key: $x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 0$
$Z = 1 \Leftrightarrow$ lock open
$Y = 1 \Leftrightarrow$ alarm on
Rest state: $x_1 = 0, x_2 = 0, x_3 = 0, x_4 = 0$
Logic: Lock open $\Leftrightarrow$ input combination = key
Alarm on $\Leftrightarrow$ input combination differs from either key or rest state by at least one digit.

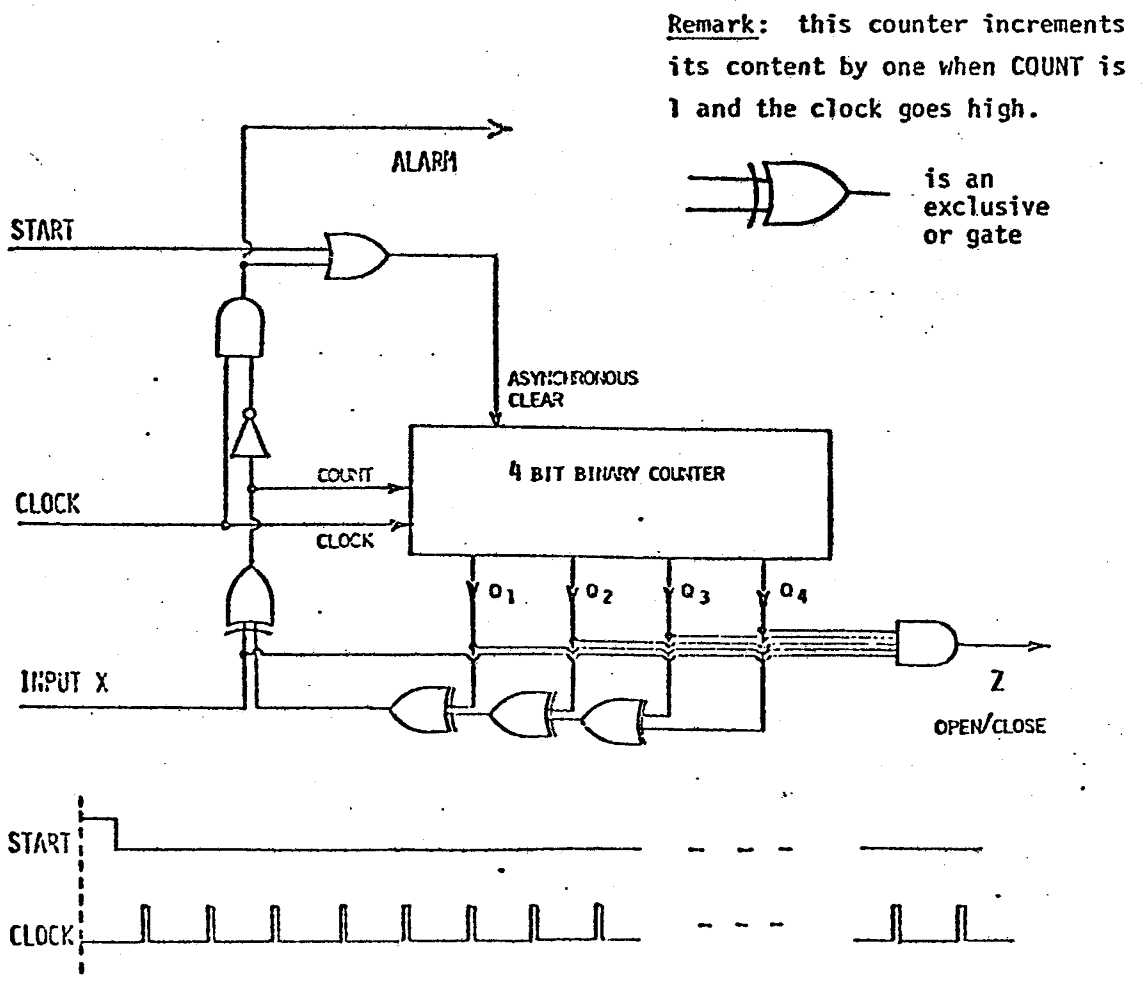
How would you realize such a circuit with NAND gates? (You may use gates with any number of inputs; note that a single input NAND gate = inverter.) *(10 points)*

9. It is also possible to use a sequential lock. Here, there is only one input line and the correct sequence of input signals (the sequential key) must be keyed in. The figure on the next page gives an example of such a sequential digital lock. Before sending any input, the Start button is depressed, producing a logic one on the line marked Start. Then the key is sequentially keyed in on line x. One key digit occurs at each clock pulse. Line x is high if the digit is a 1, otherwise it is low. The lock opens when a logic one is received on line z.

(a) When is the count line high? *(5 points)*

(b) When does the alarm go off? *(5 points)*

(c) What is the shortest key? *(5 points)*



NUMERICAL ANALYSIS

1. General principles. *(5 points)*

Consider a procedure which has been suggested for computing exp(x) on a computer with standard floating point arithmetic, using the everywhere convergent Taylor series:

$$\exp(x) = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \dots$$

The proposed computation is to be carried out as follows: Initialize the partial sum s to 1. Then for $i = 1, 2, \ldots,$

(1) compute x ** i
(2) form $i!$
(3) compute their quotient
(4) add the latter to s.

Stop when s does not change as a result of the last addition.

Explain at least three defects of the suggested procedure from the viewpoint of numerical analysis.

2. Iterative approximation. *(30 points)*

You are asked to produce a software routine, called sqrt(c), that attempts to compute $\sqrt{c}$ for any floating-point number $c \geq 0$, as quickly and accurately as possible. The routine will be executed on a newly designed small computer with binary-based normalized floating-point arithmetic (assume that floating-point division is available in hardware).

(a) Describe the procedure that you would implement, discussing at least the following topics:

(1) the choice of iterative procedure to be used (justify your suggestion in terms of speed and reliability), and
(2) obtaining an initial approximation (explain how to compute it, and how it may influence the efficiency of the sqrt routine).

(b) Let sqrt(c) denote the value computed by your routine, and $\sqrt{c}$ denote the exact value.

(1) Can you say that, for a machine-representable c, sqrt(c) = $\sqrt{c}$? Why or why not?
(2) Can you say that (sqrt(c)) ** 2 = c? Why or why not?

3. Linear systems. *(25 points)*

(a) Suppose that we wish to solve the linear system $Ax = b$, and that x is an alleged solution. Does a "small" residual vector (that is, $\|b - Ax\|$ is small) guarantee that x is an accurate solution (that is, $\|x - A^{-1}b\|$ is small)? Why or why not?

(b) Explain the following statement: In solving the linear system $Ax = b$ numerically, Gaussian elimination with partial pivoting is guaranteed to produce a "small" residual vector if no "growth" occurs in forming the factorization.

(c) In solving the linear system $Ax = b$, suppose that we first apply Gaussian elimination with partial pivoting (assume that no growth ocurs), followed by iterative improvement. Does the solution produced by a convergent iterative improvement procedure necessarily have a smaller computed residual than the unimproved solution? Why or why not?

SYSTEMS

1. Compilation. *(25 points)*

Given the following grammar:

(1)	<expression>	→	<expression> **or** <term>
(2)	<expression>	→	<term>
(3)	<term>	→	<term> **and** <factor>
(4)	<term>	→	<factor>
(5)	<factor>	→	**not** <primary>
(6)	<factor>	→	<primary>
(7)	<primary>	→	identifier
(8)	<primary>	→	(<expression>)

Use the following string in parts (a), (b) and (c).

X **or** (Y **or** Z **and not** V)

(*Hint:* for parts (a) and (b), construct a parse tree first.)

(a) Give the sequence of production numbers to be applied for this string in a bottom up parse which is left to right and a rightmost derivation.

(b) Give the sequence of production numbers to be applied for this string in a top down parse which is left to right and a leftmost derivation.

(c) Suppose this expression appears in a condition to be tested. Using the branch instructions

```
BT      operand, label  /*branch to label if operand is true*/
BF      operand, label  /*branch to label if operand is false*/
```

generate the optimal code to transfer to L1 if the expression is true and L2 if it false (where the values of X, Y, Z and V are true or false with equal probability).

2. Short questions. *(10 points)*

(a) Describe one drawback of

(1) FIFO page replacement strategy
(2) LRU page replacement strategy.

(b) Give an example of a deadly embrace.

(c) What are the advantages of spooling?

(d) Describe a set of hardware features which make it possible for an operating system to protect one user's program in memory from access by another user's program.

3. Parallel processing. *(15 points)*

When the only indivisible program operations are reads and writes of memory, the critical section problem is difficult to solve. Suppose we add one additional indivisible operation "add-to-store" which adds a value (positive or negative) to a memory location, at the same time copying the sum into a private register of the processor. This operation will be denoted by *add* (x, y); it returns a value equal to the sum of x and y, and as a side effect stores the sum as the new value of x. The following is then a possible solution to the critical section problem:

```
Process i: while true do
  begin
    while add (x, 1) ≠ 1 do junk := add (x, -1);
    critical section;
    junk := add (x, -1);
    non-critical section
  end
```

where x is a global variable with an initial value of 0.

Does the above procedure solve the critical section problem? That is, is it true that

(i) At most one process at a time can execute its critical section.

(ii) If no process is in its critical section, and one or more processes are trying to enter their critical sections, then eventually some process will enter.

Carefully justify your answer.

4. Linking. *(10 points)*

Suppose an ALGOL compiler permits external procedures and variables to be declared. The keyword **external** indicates that a variable or procedure is defined elsewhere, and **entry** indicates it is used elsewhere. Each unit of compilation is called a module.

Suppose two modules which reference each other are separately compiled.

(a) Describe what information must be output by the compiler with each module to link them together.

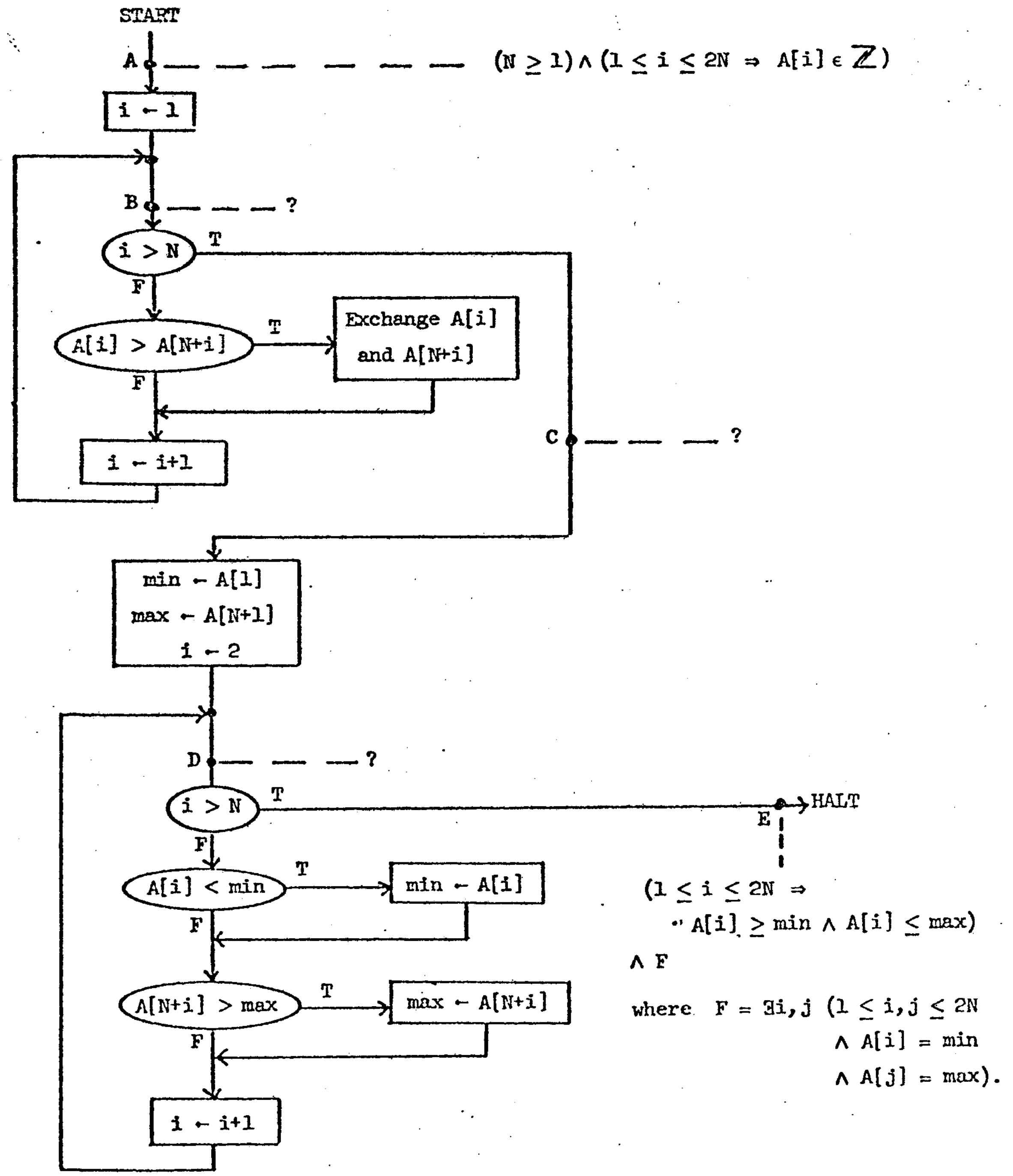
(b) Describe a typical implementation of the linking process.

(c) Suppose you also desire to do type checking among modules. Describe what information the compilers must supply. Briefly describe how you would check type compatibility. (Remember you should check the types of parameters to procedures as well.)

THEORY OF COMPUTATION

1. Proof of a program. *(30 points)*

The following program allegedly computes both the minimum and maximum elements of an array *A* of length 2*N*.


START
A
$(N \geq 1) \wedge (1 \leq i \leq 2N \Rightarrow A[i] \in \mathbb{Z})$
i ← 1
B
?
i > N
T
F
A[i] > A[N+i]
T
Exchange A[i] and A[N+i]
F
i ← i+1
C
?
min ← A[1]
max ← A[N+1]
i ← 2
D
?
i > N
T
HALT
E
F
A[i] < min
T
min ← A[i]
F
A[N+i] > max
T
max ← A[N+i]
F
i ← i+1
$(1 \leq i \leq 2N \Rightarrow A[i] \geq \min \wedge A[i] \leq \max)$
$\wedge F$
where $F \equiv \exists i,j\ (1 \leq i,j \leq 2N \wedge A[i] = \min \wedge A[j] = \max)$.

(a) Attach assertions at points B, C, and D suitable for a proof of correctness for this program by the method of inductive assertions.

(b) Prove (informally) that assertion B is maintained throughout the loop.

(c) Prove that assertion C follows from assertion B.

(d) Assuming assertion C, prove that assertion D holds initially and is maintained throughout the loop.

(e) Prove that assertion E follows from assertion D.

2. NP-completeness. *(30 points)*

The zero matrix problem ZM is defined as follows:

Given an $n \times n$ matrix A whose elements are 0 or 1 and an integer k, does there exist a $k \times k$ submatrix of A full of zeros? That is, are there distinct indices $i_1, i_2, \ldots, i_k$ such that all the elements in rows $i_1, \ldots, i_k$ and simultaneously in columns $i_1, \ldots, i_k$ are zero? For example, if $k = 2$ and

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

a zero submatrix of order 2 exists because the elements

$$\begin{matrix} A[1,1] & A[1,3] \\ A[3,1] & A[3,3] \end{matrix}$$

are all zero.

(a) Is ZM in NP? Briefly justify your answer.

(b) Which of the following known NP-complete problems would you use to prove by polynomial-time reduction that ZM is NP-hard?

Hamiltonian circuit
set cover
clique
satisfiability

(Definitions of these problems appear on p. 378 of Aho, Hopcroft and Ullman.)

Would you reduce this problem to ZM, or ZM to this problem?

(c) Briefly sketch the reduction mentioned in part (b).

PROGRAMMING PROBLEM: Warehouse Location Problem

The Adams Dornick Manufacturing Company has just built a new factory and now must decide where to build warehouses to distribute dornicks most effectively. The marketing department has prepared a list of cities to supply; each city is supplied through the warehouse nearest to it. The demand for dornicks in each city is proportional to the population of the "metropolitan area" of the city. The cost of shipping a dornick is proportional to the distance it is shipped; however it costs more per mile to ship by truck from the warehouse to the customer than it does to ship by rail from the factory to the warehouse.

Adams has enough capital to build a certain fixed number of warehouses; it only needs to know the best place to put them in order to minimize shipping costs. As the EDP manager of Adams, you are to design and write a program to solve this problem.

The problem is characterized by the following parameters:

(1)	`N`	= the number of cities to be supplied
(2)	`M`	= the number of warehouses to be built
(3)	`source`	= the city number of the location of the factory
(4)	`trRate`	= the ratio of truck cost to rail cost
(5)	`pop[i]`	= the population of the ith city, $1 \le i \le N$
(6)	`d[i,j]`	= the distance from the ith to the jth city, $1 \le i,j \le N$.

Your program should select M cities in which to build warehouses so that the total distribution cost is low. It should produce a report suitable for presentation to Edward Neufville Adams, III, the president of Adams, describing the situation. This report should include at least the following information:

(1) the location of the factory
(2) the location of each warehouse
(3) the cities supplied by each warehouse
(4) the cost of supplying each city
(5) the total cost of the traffic through each warehouse
(6) the total cost of the solution.

It could also include any other information you feel may be relevant or desired. For example, E.N.A. may want to know the total trucking cost, the distance from the factory to each warehouse, or portions of the original input.

The catch. If there are a large number of cities and a moderate number of warehouses, the number of potential solutions $\binom{N}{M}$ is too large to permit an exhaustive search in a reasonable amount of time. Therefore, you must devise heuristics for finding good solutions.

Whatever technique you use, it should be general, that is, it should work roughly as well for a different set of parameters from the one given in the sample data. For example, if you make use of the fact that Boston and Manchester are so close together that it would be senseless to put warehouses in both cities, it should be because your program "deduced" this fact rather than because you somehow included this external knowledge in the code.

Test data

A moderately large example of this problem is provided for you to solve. A listing of the data appears on the following two pages. The distance matrix was prepared by George Dantzig many years ago for a travelling salesman problem; it has a city in each state, except that it assumes a salesman would travel from Boston to Washington through Rhode Island, Connecticut, New York, New Jersey, Pennsylvania, Delaware, and Maryland. The population table is taken from an almanac. In the example, populations are divided by 100,000 and rounded to the nearest integer. "Metropolitan Boston" includes Rhode Island, Connecticut, Massachusetts, and New York; "metropolitan Washington" includes the District of Columbia, Maryland, Pennsylvania, Delaware, and New Jersey. The "metropolitan area" of every other city is the state including that city.

The data has the following form:

```
(line 1)        N  M  source  trRate
(line 2)        pop[1]  name[1]
    .               .
    .               .
    .               .
(line N+1)      pop[N]  name[N]
(line N+2)      -1
(line N+3)      d[2,1]  -1
(line N+4)      d[3,1]  d[3,2]  -1
    .               .
    .               .
    .               .
(last line)      ...  d[N,N-1]  -1  -1
```

You may write your program in any "Algol-like" language (including ALGOL W, SAIL, PASCAL, PL/1) or in your favorite dialect of LISP. Your program will be graded according to the criteria of correctness, efficiency, clarity, aptness, justification, and human factors. Besides merely getting it to work, you should spend some effort to make it understandable, efficient, easy to use and modify, and general. Since there are many ways of attacking this problem, you should be careful to explain your choices of algorithm and data structure and why you believe them to be appropriate.

Listing of test data file (COMP.DAT[1,SWB]):

```
42 6 3 2
7   Manchester
4   Montpelier
89  Detroit
107 Cleveland
17  Charleston
32  Louisville
52  Indianapolis
111 Chicago
44  Milwaukee
38  Minneapolis
7   Pierre
6   Bismarck
7   Helena
34  Seattle
21  PortlandOre
7   Boise
11  SaltLakeCity
5   CarsonCity
200 LosAngeles
18  Phoenix
10  SantaFe
22  Denver
3   Cheyenne
15  Omaha
28  DesMoines
47  KansasCity
22  Topeka
26  OklahomaCity
112 Dallas
19  LittleRock
39  Memphis
22  Jackson
36  NewOrleans
34  Birmingham
46  Atlanta
68  Jacksonville
26  Columbia
51  Raleigh
46  Richmond
241 Washington
278 Boston
10  PortlandMe
-1
8 -1
39 45 -1
37 47 9 -1
50 49 21 15 -1
61 62 21 20 17 -1
58 60 16 17 18 6 -1
59 60 15 20 26 17 10 -1
62 66 20 25 31 22 15 5 -1
81 81 40 44 50 41 35 24 20 -1
103 107 62 67 72 63 57 46 41 23 -1
108 117 66 71 77 68 61 51 46 26 11 -1
145 149 104 108 114 106 99 88 84 63 49 40 -1
181 185 140 144 150 142 135 124 120 99 85 76 35 -1
```

```
187 191 146 150 156 142 137 130 125 105 90 81 41 10 -1
161 170 120 124 130 115 110 104 105 90 72 64 34 31 27 -1
142 146 101 104 111 97 91 85 86 75  51 59 29 53 48 21 -1
174 178 133 138 143 129 123 117 118 107 83 84 54 46 35 26 31 -1
185 186 142 143 140 130 126 124 128 118 93 101 72 69 58 58 43 26 -1
164 165 120 123 124 106 106 105 110 104 86 97 71 93 82 62 42 45 22 -1
137 139 94 96 94 80 78 77 84 77 56 64 65 90 87 58 36 68 50 30 -1
117 122 77 80 83 68 62 60 61 50 34 42 49 82 77 60 30 62 70 49 21 -1
114 118 73 78 84 69 63 57 59 48 28 36 43 77 72 45 27 59 69 55 27 5 -1
85 89 44 48 53 41 34 28 29 22 23 35 69 105 102 74 56 88 99 81 54 32 29 -1
77 80 36 40 46 34 27 19 21 14 29 40 77 114 111 84 64 96 107 87 60 40 37 8 -1
87 89 44 46 46 30 28 29 32 27 36 47 78 116 112 84 66 98 95 75 47 36 39 12 11 -1
91 93 48 50 48 34 32 33 36 30 34 45 77 115 110 83 63 97 91 72 44 32 36 9 15
         3 -1
105 106 62 63 64 47 46 49 54 48 46 59 85 119 115 88 66 98 79 59 31 36 42 28 33
         21 20 -1
111 113 69 71 66 51 53 56 61 57 59 71 96 130 126 98 75 98 85 62 38 47 53 39 42
         29 30 12 -1
91 92 50 51 46 30 34 38 43 49 60 71 103 141 136 109 90 115 99 81 53 61 62 36 34
         24 28 20 20 -1
83 85 42 43 38 22 26 32 36 51 63 75 106 142 140 112 93 126 108 88 60 64 66 39
         36 27 31 28 28 8 -1
89 91 55 55 50 34 39 44 49 63 76 87 120 155 150 123 100 123 109 86 62 71 78 52
         49 39 44 35 24 15 12 -1
95 97 64 63 56 42 49 56 60 75 86 97 126 160 155 128 104 128 113 90 67 76 82 62
         59 49 53 40 29 25 23 11 -1
74 81 44 43 35 23 30 39 44 62 78 89 121 159 155 127 108 136 124 101 75 79 81 54
         50 42 46 43 39 23 14 14 21 -1
67 69 42 41 31 25 32 41 46 64 83 90 130 164 160 133 114 146 134 111 85 84 86 59
         52 47 51 53 49 32 24 24 30 9 -1
74 76 61 60 42 44 51 60 66 83 102 110 147 185 179 155 133 159 146 122 98 105 107
         79 71 66 70 70 60 48 40 36 33 25 18 -1
57 59 46 41 25 30 36 47 52 71 93 98 136 172 172 148 126 158 147 124 121 97 99
         71 65 59 63 67 62 46 38 37 43 23 13 17 -1
45 46 41 34 20 34 38 48 53 73 96 99 137 176 178 151 131 163 159 135 108 102 103
         73 67 64 69 75 72 54 46 49 54 34 24 29 12 -1
35 37 35 26 18 34 36 46 51 70 93 97 134 171 176 151 129 161 163 139 118 102
         101 71 65 65 70 84 78 58 50 56 62 41 32 38 21 9 -1
29 33 30 21 18 35 33 40 45 65 87 91 117 166 171 144 125 157 156 139 113 95 97
         67 60 62 67 79 82 62 53 59 66 45 38 45 27 15 6 -1
3 11 41 37 47 57 55 58 63 83 105 109 147 186 188 164 144 176 182 161 134 119
         116 86 78 84 88 101 108 88 80 86 92 71 64 71 54 41 32 25 -1
5 12 55 41 53 64 61 61 66 84 111 113 150 186 192 166 147 180 188 167 140 124
         119 90 87 90 94 107 114 77 86 92 98 80 74 77 60 48 38 32 6 -1 -1
```

Spring 1978 Comprehensive Exam

ALGORITHMS AND DATA STRUCTURES

1. Bin packing. *(20 points)*

Let $X = \{x_1, x_2, \ldots, x_n\}$ be the weights of n objects with $0 < x_i < 1$. We wish to pack the objects into a *minimum* number of bins, subject to the following constraints:

(i) At most *two* objects can be put in the same bin.
(ii) All bins have unit capacity, that is, x_i and x_j can share a bin if and only if $x_i + x_j \le 1$.

(a) Design an $O(n \log n)$ algorithm that produces an optimal packing for any X with n objects.

(b) Prove that the packing produced by your algorithm indeed uses the minimum number of bins possible.

2. Maximum flow. *(20 points)*

Let T be a directed binary tree with n leaves. Associated with each arc $(FATHER(v), v)$ in T is a non-negative *capacity* $c(FATHER(v), v)$ for the amount of water that can flow from node $FATHER(v)$ to node v. We assume $c(ground, root) = \infty$, that is, there is an infinite supply of water at the root. A *flow* in T is a non-negative function f on the set of arcs such that

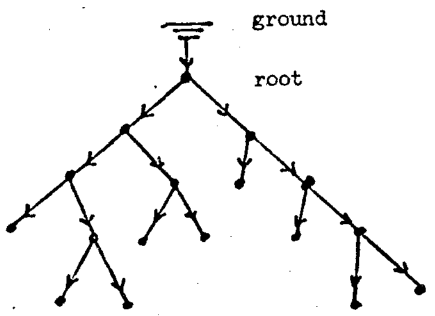
(i) $f(FATHER(v), v) \le c(FATHER(v), v)$
(ii) $f(FATHER(v), v) = f(v, LSON(v)) + f(v, RSON(v))$.

The value $f(ground, root)$, which is equal to $\sum_{v \text{ leaf}} f(FATHER(v), v)$ because of condition (ii), is called the *value* of the flow f, denoted by $W(f)$. A flow f is a *maximum flow* in T if $W(f)$ is as large as possible.

(a) Give an algorithm to compute the *value* of a maximum flow f for any binary tree T with its associated capacity function c.

(b) Give an algorithm to generate a *maximum flow* f by computing the arc assignments $f(FATHER(v), v)$, for any binary tree T with its capacity function c.

(Both (a) and (b) have $O(n)$ algorithms. Partial credit will be given to less efficient algorithms.)



3. Data structures. *(20 points)*

You are to design a data structure which implements the following operations efficiently (in the average case) using a hashing scheme:

$ENTER(a, b)$	Enters the pair (a, b) into the table.
$FIND(a, b)$	Returns the pair (a, b) or indicates a failure if it is not present.
$FINDALL(a, *)$	Returns a list of all pairs whose first element is a.
$FINDALL(*, b)$	Returns a list of all pairs whose second element is b.
$REMOVE(a, b)$	Removes the pair (a, b) from the table.
$REMOVEALL(a, *)$	Removes all pairs $(a, *)$ from the table.
$REMOVEALL(*, b)$	Removes all pairs $(*, b)$ from the table.

Briefly describe how each of the operations is implemented using your data structure.

ARTIFICIAL INTELLIGENCE

1. Formal representation. *(18 points)*

Consider the following problem:

(1) There are five houses, each of a different color and inhabited by men of different nationalities, with different pets, drinks, and cigarettes.

(2) The Englishman lives in the red house.

(3) The Spaniard owns the dog.

(4) Coffee is drunk in the green house.

(5) The Ukranian drinks tea.

(6) The green house is immediately to the right of the ivory house.

(7) The Old Gold smoker owns snails.

(8) Kools are smoked in the yellow house.

(9) Milk is drunk in the middle house.

(10) The Norwegian lives in the first house on the left.

(11) The man who smokes Chesterfields lives in the house next to the man with the fox.

(12) Kools are smoked in the house next to the house where the horse is kept.

(13) The Lucky Strike smoker drinks orange juice.

(14) The Japanese smokes Parliaments.

(15) The Norwegian lives next to the blue house.

The problem continues, "Now, who drinks water? And who owns the zebra?"

Your problem is not to "solve" this puzzle, but to express the conditions of this problem in first order logic. You may invent whatever individuals, predicates and functions you need, but make clear what each is to represent. You do not have to translate every statement to a WFF in your representation; rather, you need do only enough so that the rest of the translation is transparent. It should therefore be necessary to translate only statements 1, 2, 3, 4, 6, 10, 12, and 15 to convince the grader that you understand the material.

2. A.I. systems. *(24 points)*

People often tend to emphasize the positive in describing abilities and capabilities of their A.I. programs and systems. In particular, from reading Winston one might get the feeling that A.I. jumps from one flaming success to the next. Unfortunately, there are deficiencies and inabilities with every proposed formalism and system, even if it takes a critical "reading between the lines" to notice them.

For *six* of the following programs or ideas, state the key failings and problems of that system. Typical answers might point out problems with the particular methodology, domain problems that the system cannot solve, or difficulties in a particular approach.

(a) Evans' Analogy program

(b) Winston's Concept Formation system

(c) Waltz's Vision system

(d) Newell and Simon's GPS

(e) Fikes and Nilsson's STRIPS

(f) production systems

(g) Winograd's Shrdlu

(h) Minsky's Frames

(i) Buchanan et. al. DENDRAL

(j) Hewitt et. al. Planner

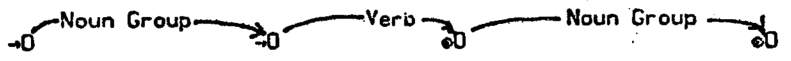
3. Natural language. *(10 points)*

(a) What is an Augmented Transition Network? How does it differ from an ordinary transition network?

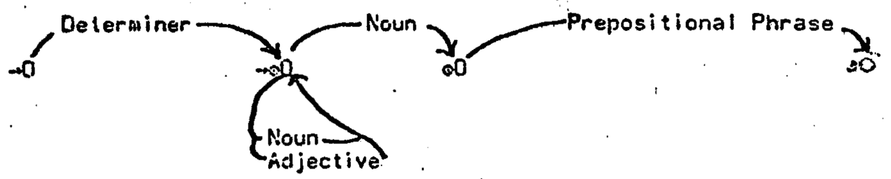
(b) Name four possible "notes" for the arcs of an ATN which is to be employed in parsing English.

(c) Given the following transition network describing a sentence (where → represents a possible initial node, and ● a possible terminal node), state an English sentence that is syntactically ambiguous in this grammar. Be sure to state the possible word classes of each of the words in your sentence.

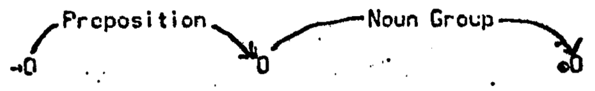
SENTENCE:



Noun Group:

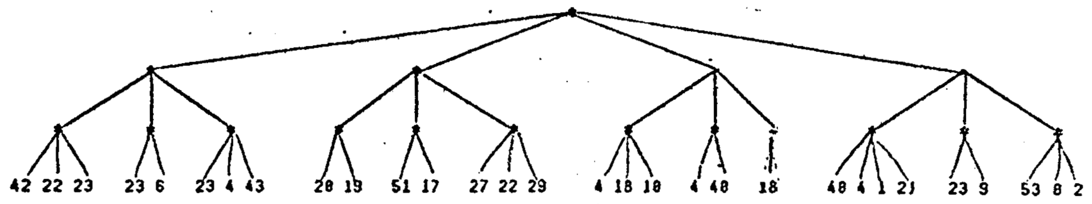


Prepositional phrase:



4. Alpha-beta search. *(8 points)*

(a) Search the game tree below from left to right using the alpha-beta technique. Assume the top level is a maximizing level. Underline the nodes where static evaluation will occur, and draw a circle around the terminal node of the principal variation.



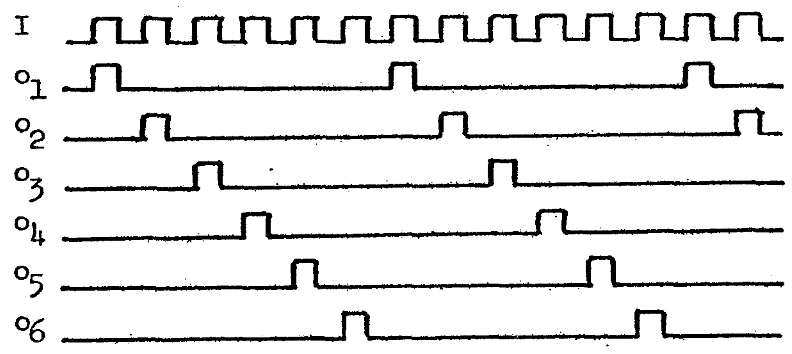
(b) If you had searched the tree from right to left, many more nodes would have been visited. What technique might a program employ to minimize the number of nodes visited in an alpha-beta search?

(c) Now search the tree again with initial conditions $\alpha = 20$, $\beta = 25$, underlining the nodes where static evaluation occurred. How might a game playing program be written so as to usually visit this smaller number?

HARDWARE

1. 6-phase clock. *(13 points)*

Design a circuit to provide a 6-phase non-overlapping clock. Given input I, provide outputs $o_1, o_2, \ldots, o_6$, as shown below.



Try to keep it glitch-free.

2. Implement $ab\bar{c} + a\bar{b}c + \bar{a}bc + \bar{a}\bar{b}c$ in as few gates as possible. *(4 points)*

3. Design a circuit to provide odd parity for an ASCII character. That is, given a 7 bit input, provide a 1 bit output so that there are always an odd number of the 8 bits on. *(10 points)*

4. Expand the following acronyms: *(8 points)*

   CCD, FIFO, LSTTL, CMOS, ECL, SOS, UART, PROM.

5. Discuss the advantages and disadvantages of microprogramming as compared with hard-wired control. In particular, when would you use one rather than the other? *(5 points)*

6. You have recently purchased an 8-bit microcomputer with a 16 bit address bus, and you want to connect 64K bytes of RAM and 64K bytes of PROM, leaving room for future expansion. Pick some technique for doing this, and justify your choice. *(10 points)*

7. Control word design. *(10 points)*

You are given: 16 registers, an ALU with 16 logic and 16 arithmetic functions, and a shifter with 8 operations, all connected to a common bus.

(a) Formulate a control word to specify the various micro-operations for the CPU.

(b) Specify the number of bits for each field of the control word, and give a general idea of the encoding scheme for each field.

(c) Show the control word(s) for the operation: $R7 := R1 + R14$.

NUMERICAL ANALYSIS

1. Iteration. *(20 points)*

(a) Three well-known methods for solving a non-linear scalar equation $f(x) = 0$ are the bisection method, Newton-Raphson method and the secant method. Describe each method briefly and discuss their advantages and disadvantages. Mention order of convergence, amount of computation, and global and local characteristics.

(b) As an alternative method for the solution of $f(x) = 0$, one might seek an iteration formula involving the second derivative of f. You decide to find an iteration formula of the form

$$x_{n+1} = x_n - \frac{f_n}{f_n'} + \beta \frac{f_n^2 f_n''}{(f_n')^3}$$

where $f_n = f(x_n)$, $f_n' = f'(x_n)$, $f_n'' = f''(x_n)$.

Determine the constant β so that the convergence will be of highest possible order, and give the order.

2. Minimax approximation. *(10 points)*

(a) How would you recognize that you have obtained the nth degree polynomial that is the minimax (Chebyshev) approximation to a function $f(x)$ in $C[a,b]$?

(b) Use your answer to (a) to find the straight line that is the best Chebyshev approximation to the quadratic $ax^2 + bx + c$ in $[-1,1]$.

(c) Find the answer to the problem in (b) by expanding the quadratic in a series of Chebyshev polynomials $T_0(x) = 1$, $T_1(x) = x$, $T_2(x) = 2x^2-1$. State the general theorem you are using.

3. Linear systems. *(20 points)*

Define the condition number of a nonsingular matrix A in the L_p-norm by

$$\mu_p(A) = \|A\|_p \|A^{-1}\|_p$$

(a) Prove that if the perturbed matrix $(A + \delta A)$ is singular, then we must have

$$\mu_p(A) \geq \frac{\|A\|_p}{\|\delta A\|_p}.$$

(b) Show that with the L_2-norm, equality holds in (a) when

$$\delta A = \frac{-yx^T}{x^Tx}, \quad x = A^{-1}y,$$

where y is a vector for which $\|A^{-1}\|_2\|y\|_2 = \|A^{-1}y\|_2$.

(c) Use the inequality in (a) to get a lower bound for $\mu_\infty(A)$ for the matrix

$$A = \begin{bmatrix} 1 & -1 & 1 \\ -1 & \epsilon & \epsilon \\ 1 & \epsilon & \epsilon \end{bmatrix}, \quad 0 < |\epsilon| < 1.$$

4. Arithmetic and error analysis. *(10 points)*

(a) Do floating-point and real arithmetic have the same arithmetic properties (e.g. associativity, commutativity)? Prove your point.

(b) Briefly explain the idea of backward error analysis in the context of the solution of a linear system $Ax = b$.

(c) What advantage does backward error analysis have over forward error analysis?

SYSTEMS

1. LR(k)-grammars. *(12 points)*

The following grammar "abstracts" the statement structure of BEGIN blocks in PASCAL:

$$\begin{aligned} S &\to b\ SL\ e \\ S &\to b\ SL;\ e \\ S &\to a \\ SL &\to S \\ SL &\to S;\ SL \end{aligned}$$

(a) Is the grammar ambiguous? Provide some argument to justify your answer.

(b) The grammar is not LR(1). Why? (*Hint:* think about the nontrivial parsing decisions; you should not have to construct the characteristic finite state machine to answer.)

(c) Is the grammar LR(2)? Justify your answer.

2. Symbol table. *(10 points)*

A symbol table entry for a simple variable in PASCAL contains the following information:

```
name, level, offset, type, direct or indirect
```

For each item of information, give a use for that item in the compilation process. Be sure to include an explanation of how addressing of the variable is done.

3. Optimization. *(6 points)*

What kind of optimization is illustrated in the examples below? Give the name commonly used for each optimization transformation.

(a)
```
FOR I := 1 TO 3 DO          ⇒      A[1] := 1;
   A[I] := I;                      A[2] := 2;
                                   A[3] := 3;
```

(b)
```
Z := A + B * C;             ⇒      TEMP0001 := B * C;
X := B * C + D;                    Z := A + TEMP0001;
                                   X := TEMP0001 + D;
```

(c)
```
FOR I := 1 TO N DO          ⇒      TEMP0002 := 5;
   A[I] := 5 * I;                  FOR I := 1 TO N DO BEGIN
                                      A[I] := TEMP0002;
                                      TEMP0002 := TEMP0002 + 5
                                   END;
```

4. Register allocation. *(10 points)*

When writing a code generator for a register machine, it will at times be necessary to generate code using a register for some purpose, although the register contains information which is potentially useful for other computations within the basic block. This problem is similar in many ways to the page replacement problem for virtual memory systems.

(a) Compare the register problem with the page replacement problem in the case of a strictly one pass compiler. Mention similarities and differences.

(b) What is the principal difference between the two problems in a multipass compiler?

5. Protection. *(12 points)*

A general protection scheme can be thought of as a matrix with rows for users and columns for resources. The matrix entry specifies the rights of the user with respect to that particular resource. Thus, the matrix entry might contain flags for read permission, write permission, and append permission.

(a) Suppose that the matrix were maintained in columns by the operating system. A given column is associated with each particular resource. Provide a scenario showing how the operating system might enforce protection.

(b) If the matrix is maintained by rows associated with users, it may be considered a capability list for a given user. Sometimes a user wants to permit another user, who is trusted to be careful, to operate on a resource which the first user has a capability to operate on. Describe an extension to the matrix model which will make it possible to pass a capability to the second user so that he may use it, but prevent him from passing it on further.

6. Linking loaders. *(10 points)*

Linking loaders perform two distinct operations. They resolve external references and relocate object code to run in specific machine locations.

(a) A "linker only" would take sections of object code and produce one large section of object code plus some tables to be used by the loader. Describe the function of this table and a data structure which implements it.

(b) A "loader only" would take sections of object code and produce one large section of object code (possibly in core), plus some tables to be used by the linker. Describe the function of this table and a data structure which implements it.

THEORY OF COMPUTATION

1. Regular languages. *(13 points)*

Consider the language

$L = \{a_{2m+1}a_{2m}b_m a_{2m-1}a_{2m-2}b_{m-1}\ldots a_1 a_0 b_0 \mid a_i, b_i \in \{0,1\},\ m \geq 0,$ and the two binary numbers $x = a_{2m+1}a_{2m}\ldots a_0,\ y = b_m b_{m-1}\ldots b_0$ satisfy $x = y^2\}$.

For example, $\underline{01}1\underline{10}0\underline{01}1$ is in L since $x = 011001 = 25_{10}$ and $y = 101 = 5_{10}$.

Is L a regular language? Prove your answer.

2. True or false. *(12 points)*

Answer true or false for each of the following statements. Give a brief argument to justify your answer.

(1) The intersection of two recursively enumerable sets is still recursively enumerable.

(2) It is undecidable whether a Turing machine over the alphabet $\{0,1\}$ accepts at least one word that begins with a 1.

(3) For a language L, define $Transpose(L) = \{u_2 u_1 \mid |u_1| = |u_2|,$ and $u_1 u_2 \in L\}$. If L is context-free, then $Transpose(L)$ is always context-free.

3. Logic. *(15 points)*

On the farflung Islets of Langerhans, a peculiar form of logic is employed. There are only two sentential connectives, a unary operator, ~, and a binary operator, *, and one rule of inference, Reductio ad nauseum (or R for short):

R] $\dfrac{\begin{array}{r}{\sim}A*B \\ B\end{array}}{A}$ (Here we follow the Langerhan tradition of letting ${\sim}A*B$ mean $({\sim}A)*B$.

There is also a single axiom schema, Producia desde nil (or P):

P] ${\sim}({\sim}A*B)*(B*C)$

Prove each of the following WFF's of Langerhanish logic, using this axiom schema and rules of inference. Proof of the last WFF is, of course, sufficient to guarantee credit for the entire problem.

(a) ${\sim}D*{\sim}({\sim}A*B)$

(b) ${\sim}E*{\sim}D$

(c) E

4. Verification. *(5 points)*

Consider the pidgin-algol program fragment:

```
for $i$ ← 1 step 1 until $n$ do
   begin
      $t$ ← $a[i]$;
      $a[i]$ ← $a[n-i+1]$;
      $a[i]$ ← $t$;
   end;
```

What does this program fragment do to the array a? Letting $a_0[j]$ denote the initial values of the array, write an invariant of the loop that will suffice to prove your answer.

5. Verification. *(15 points)*

(a) Write a pidgin-Algol program fragment to rearrange the elements of the array $a[1{:}n]$ so that all elements less than the real number c precede the others.

In parts (b) and (c) you may use the relation *permutes*(A, B), which states that the array A is a permutation of the array B.

(b) State as a formula of logic the relation between the initial and final arrays that insures the correctness of the program.

(c) Attach suitable invariants to one or more suitable points in your program so that you could prove the partial correctness of your program. You need not prove the invariance, but the partial correctness of your program must follow from it.

PROGRAMMING PROBLEM: Production System Interpreter

Part 1: The Interpreter

This programming problem centers around the implementation of a production system interpreter. Production systems are a form of automata, just as Turing machines and random access machines are automata. More recently, they have been suggested as a model for human information processing (see, for example, Newell & Simon's *Human Problem Solving*). The production system you are to program has elements from both of these domains.

For our purposes, we define a production system as follows:

A production system consists of two pieces, a *Working Memory* (WM) and a *Production Memory* (PM). The working memory is a series of strings, and the production memory, a series of productions. Using a semicolon (;) to separate the elements in a series of strings, we get the following BNF for working memory and production memory:

```
<working memory>     ::= <memory string> | <memory string>; <working memory>
<production memory>  ::= <production> | <production>; <production memory>
<memory string>      ::= (any string of characters except an unquoted semicolon)
```

Note that in describing the BNF of this system, we are describing the external representation, that is, the format in which the productions are to be read and written. You may want to find some more appropriate data structure to represent the internal form that your interpreter is to manipulate.

In all of the following examples, you may find it convenient to read and write productions with additional spaces following the separating semicolons (;) and greater-than signs (>) and keywords in actions. That is, your system should recognize the input of the working memory

```
ABCD; BCDE; EFG
```

and the working memory

```
ABCD;  BCDE;EFG
```

as referring to the same internal form. However, spaces are to be considered as significant in all other contexts.

A production consists of two parts, a *pattern* and an *action*, separated by a greater-than sign:

```
<production> ::= <pattern> > <action>
```

Patterns and actions are themselves series of pattern and action elements.

```
<pattern> ::= <pattern-element> | <pattern-element>; <pattern>
<action>  ::= <action-element>  | <action-element>; <action>
```

The production system interpreter operates in a two phase cycle. First, the patterns of the production are *matched* to the elements of working memory. A match will either succeed or fail, and a successful match will bind certain elements to certain strings. If the match succeeds, the production is *excited*. Then, one of the excited productions is selected for execution, and the action elements of that production are evaluated in order. All productions are then unexcited. Note that a single production can become excited with several different variable bindings.

Patterns are made up of two elements, variables and constants. Variables are to be distinguished by preceding them with the character ? (question mark). Variables are single alphabetic characters. Thus, the form ?X is able to match any substring (including the null string, <null>). The form ?3X implies a match of any three character string.

The second and subsequent occurrences of a variable in a pattern imply repetition of the original string matched. Matching a variable to a substring binds that variable to that substring for the subsequent action evaluation.

Thus, the following are successful matches:

memory string	pattern	binds to ?X	binds to ?Y
ABC	ABC		
ABCDEFGHI	ABC?XHI	DEFG	
ABCABCAB	?X?Y?X?Y?X	AB	C
ABC	?X?Y	AB	C
ABC	?X?Y	A	BC
ABC	?2X?Y	AB	C
ABC	?X?Y	ABC	<null>
ABCD	?X?Y	<null>	ABCD
ABCDEFGHI	ABC?XHI?Y	DEFG	<null>
ABCDEFGHIJKL	?10X?Y	ABCDEFGHIJ	KL
?ABC	'?A?X	BC	

and the following are unsuccessful matches:

memory string	pattern	reason
ABCABCAB	?1X?Y?X?Y?X	As ?1X binds X to A, the rest of pattern won't match.
ABC	?2X?2Y	Not enough characters.
ABC	D?X	D doesn't match A.
ABC	ABCD	Ran out of memory element.
ABCDE	ABCD	Ran out of pattern element.
ABC	'?A	' is the quoting character.

In general, the pattern side of a production may consist of several elements; for that production to become excited, each element of the pattern must be successfully matched to a different element in the working memory. Variable binding transfer over the match. Thus, the working memory

ABC; ABD; BCE

will match with the pattern

?2XD; ?XC; ?Y

binding X to AB and Y to BCD, but will not match with the pattern

?2XE; ?XC; ?Y .

Matching the pattern side of a production to working memory also binds another type of variable, the dollar variables. Generally, if a pattern of the form X; Y; Z; matches to a working memory of the form A; B; C; D, with X matched to B, Y matched to C and Z matched to A, then \$1 is bound to the element B, \$2 to the element C, and \$3 to the element A. Performing NOTICE \$2 would make working memory appear as C; A; B; D.

Actions consist of three types of elements, those that move things around (including delete) from the working memory, those that add new elements to either working memory or production memory, and those that interact with the *outside environment.*

Implement the following actions:

DELETE $n	Delete $n from working memory. Thus, the working memory element matching the second element of this production's pattern would be deleted if the command had been DELETE $2.
NOTICE $n	Move working memory element $n to the front of working memory.
PLACE <string expression>	Compute the value of <string expression>, and place an element at the front of working memory with that value. The BNF of string expressions is defined below.
BUILD <string expression>	Construct a new production memory production, the same production that would have resulted from reading <string expression> in as defining a production.
PERFORM <string expression>	This action is for interaction with the outside environment. Your simulator should then update its model of the outside environment, and perhaps place some new element at the front of working memory. Systems that employ PERFORM (such as problem three of part two) must also build the environment for their performance.
HALT	Guess what.

Just as in > and ; , spaces after one of the action keywords should be ignored.

String expressions are defined:

```
<string expression> ::= <string variable> | <string constant> | <string variable> (i,j)
<string expression> ::= <string expression><string expression>
<string variable>   ::= $n | <?variable>
```

If $n is used, it refers to the entire string matched by production pattern element n. Naming a variable (as ?X) inserts the bound value of that variable into the expression. The parenthesized *i,j* is for substringing; it refers to the *i*th through *j*th characters of <string variable>. Everything else is a constant.

Of course, it is sometimes necessary to quote a character, such as ?, $, (, and) which would have some other meaning to the production system. For that purpose, the quote character ' should be implemented. ' can, of course, itself be quoted.

Note that activation of a production does not necessarily insure that that production will fire. If several productions match a working memory, there must be some *conflict resolution* method for deciding which production to fire. Typical methods include:

Production order. The first active production on the production list is selected to fire.

Greatest latency. The active production that has not been fired for the longest time is selected.

Random selection. A production is selected from among the active productions, at random.

Best match. The active production requiring the fewest variable assignments is selected for firing.

Working memory order. The active production matching the earliest element of working memory is selected.

Of course, not all of these methods are *complete*; that is, some of them will not necessarily resolve conflicts.

In your system, implement *production order* and at least one other conflict resolution scheme. Let the resolution scheme be selectible by switch.

You should also provide adequate tracing facilities so that, for each step, the graders can see which productions were excited, which fired, and what effect that firing had on working memory. Tracing should also be switchable on and off. If you manage to avoid checking every production that might be excited, you don't need to trace the "excited" productions.

In summary, the first part of this problem is to write a production system simulator that will read in (1) switch settings, (2) a series of productions, and (3) a working memory, and simulate those productions until either a preset limit is reached, or a halt statement is encountered.

If this is too confusing, perhaps an example will help. We want a production system that will take expressions using three types of bracketing (), {} and [], which will place the element TRUE at the front of working memory if the expression is "well-formed", FALSE otherwise, and then halt. We also want a second expression in working memory, of the form COUNT <parentheses>,<brackets>,<braces> which will state, in unary, the number of matching parenthesis, bracket and brace pairs we removed.

We assume the input will be a string containing an equals sign (=), and then then some arrangement of the characters {} [] and (). We also assume that we are switched to "production order conflict resolution". In all problems, you can mark the initial input with some special marker.

Our production system should read in the productions (the numbers on the left are for reference, they are not part of the productions):

```
1.   =?X> DELETE $1; PLACE ?X; PLACE COUNT,,
2.   (); COUNT?A,?B,?C> DELETE $1; PLACE COUNT ?A1,?B,?C; PLACE TRUE; HALT
3.   []; COUNT?A,?B,?C> DELETE $1; PLACE COUNT ?A,?B1,?C; PLACE TRUE; HALT
4.   {}; COUNT?A,?B,?C> DELETE $1; PLACE COUNT ?A,?B,?C1; PLACE TRUE; HALT
5.   ?X()?Y; COUNT?A,?B,?C> DELETE $1; PLACE ?X?Y; DELETE $2; PLACE COUNT?A1,?B,?C
6.   ?X[]?Y; COUNT?A,?B,?C> DELETE $1; PLACE ?X?Y; DELETE $2; PLACE COUNT?A,?B1,?C
7.   ?X{}?Y; COUNT?A,?B,?C> DELETE $1; PLACE ?X?Y; DELETE $2; PLACE COUNT?A,?B,?C1
8.   ?X> PLACE FALSE;  HALT
```

Thus, if we read the working memory

={[]()[]}

we would get the following simulation:

Step	Excited	Firing	Bindings	New Working Memory
1.	1,5,6,6,8	1	?X←{[]()[]}	{[]()[]}; COUNT,,
2.	5,6,6,8	5	?X←{[] ?Y←[]}	{[][]}; COUNT1,,
3.	6,6,8	6	?X←{ ?Y←[]}	{[]}; COUNT1,1,
4.	6,8	6	?X←{ ?Y←}	{}; COUNT1,11,
5.	4,8	4		TRUE; COUNT 1,11,1

The system has halted after step 5. Note that in steps 2 through 4, there are two different ways that production 6 can match the working memory.

Part 2: Sample Production Systems

Having created this production system interpreter, you are now to write several simple production systems to demonstrate its features.

1. Write a simple production system to convert from balanced ternary to unary. Write another simple production system to convert from unary to balanced ternary. See Knuth, Vol 2. pp. 173-175 for a definition of balanced ternary.

2. Write a simple production system to take boolean expressions and compute their value. Your boolean expressions should follow the grammar:

<boolean expression> ::= <disjunctive expression> | <boolean expression>+<disjunctive expression>
<disjunctive expression> ::= <negated expression> | <disjunctive expression> * <negated expression>
<negated expression> ::= - <primary expression> | <primary expression>
<primary expression> ::= T | F | (<boolean expression>)

and have the following semantics:

T+T=T	T+F=T	T*T=T	T*F=F	-T=F
F+T=T	F+F=F	F*T=F	F*F=F	-F=T

3. Write a production system that can find its way through mazes such as the one shown in figure 1 on the next page. The PERFORM action will be important here. There are eight things your production system can PERFORM; it can LOOK or MOVE in any of the four directions (NORTH, SOUTH, EAST, WEST). Executing the action PERFORM LOOK WEST inserts as the first element of working memory either the string BLOCKED or the string OPEN depending upon whether there is a wall separating the cell you are in from the cell to your west. The action PERFORM MOVE WEST will either move you to the cell to the west, if there is no wall separating you and that cell, or be a no-op, if there is. Notice that you must also build an outside "environment", to represent the maze, in writing this production system. Run your production system on the maze in figure 1.

4. (Optional.) Write some other interesting production system that exercises some of the nice features of your interpreter. Be sure to make clear what exactly you have done.

It is acceptable to program the production system in ALGOL, PASCAL, LISP, SAIL, or PL/1 (or even FORTRAN or BASIC should you be so foolhardy.) It is not acceptable to do this problem in SNOBOL, or other languages with special string pattern matching facilities.

A complete solution of this problem includes not only the interpreter, but also the sample production systems, traces of their execution, and copious documentation. Your work will be graded on several criteria, such as programming style, program structure, breadth (which of the additional features you coded), efficiency (both of code and data structures), and, most important, clarity.

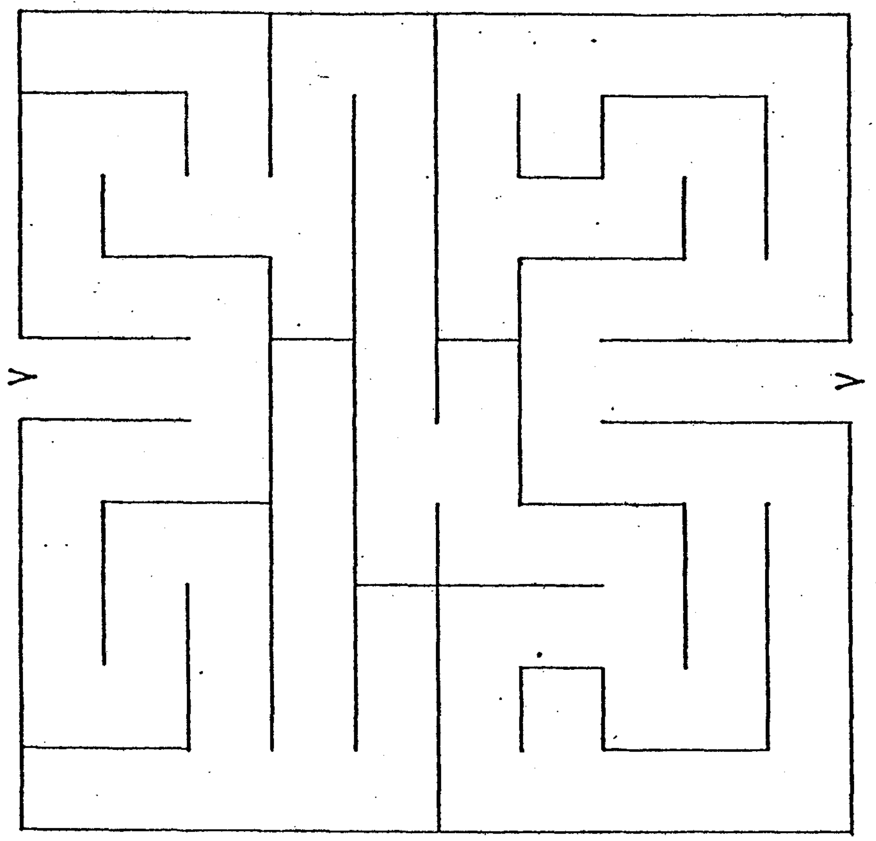


Figure 1

Answers

SPRING 1972 COMPREHENSIVE EXAM

1.

1	1	1	1
Ø	Ø	Ø	1
Ø	Ø	Ø	1
1	1	1	1

2. A(0) := 1 . (When the program runs, A(0) gets set to 1, then 0, then 1 .)

3. Charles Babbage designed a computing machine in the first half of the nineteenth century; it used decimal arithmetic, with about 1000 words of 50 digits each. His "Analytical Engine" was never built, but it was well-known in England as a possibility; frequent references to it appear in the literature during the 1950's.

About 1935 George Stibitz at Bell Labs constructed a small calculator out of relays, capable of doing complex-number arithmetic in the binary system. He demonstrated it at the Math Society meeting in 1940, hooked up via teletype between Dartmouth College (New Hampshire) and Bell Labs (New Jersey).

Meanwhile John Atanasoff at Iowa State was working on a computer with electronic arithmetic. This machine (also binary) was designed specifically to solve linear equations (it wasn't general-purpose).

Konrad Zuse of Germany was working on a general-purpose computer, using relay circuitry, floating-point binary arithmetic and a program controlled by punched movie film. His machine became working in 1941, but it was unknown outside Germany (due to the war) and details are just now becoming known in America.

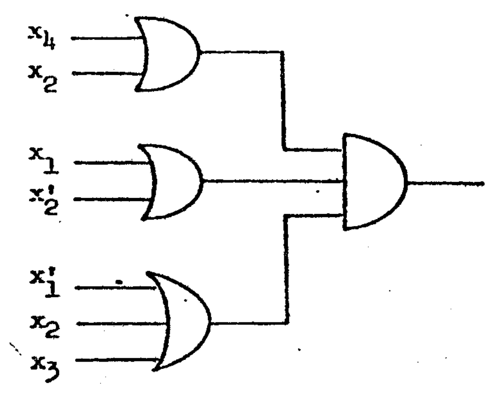
The next general-purpose computer to be built was the Harvard Mark I, built by IBM and Howard Aiken and completed in 1944. It also used relay technology. The program was punched on paper tape; it used 24-digit decimal arithmetic and had 72 storage registers for memory.

The first electronic computer (much faster than relay speeds) was the ENIAC, developed principally by J. P. Eckert and John Mauchly at the University of Pennsylvania during 1943–1946. (Many other people, notably J. G. Brainerd and H. H. Goldstine, were significant contributors to this project.) ENIAC had a plugboard controlled program and decimal arithmetic, with *parallel* processing going on in twenty 10-digit-plus-sign accumulators.

The same group, with the help of John von Neumann, designed EDVAC, the first stored-program computer, during 1944-1946. This machine was binary and was to have about 2000 words of mercury delay-line memory with 32 bits per word. The circuitry was serial with a speed of one bit per microsecond. The EDVAC was not completed until 1951, but its initial design was published in 1945 and stimulated the development of many similar machines.

The first stored-program computer to be actually working was Wilkes's EDSAC at Cambridge University, finished in 1949. Other noteworthy early machines based largely on EDVAC were Williams-Kilburn's Manchester machine (with index registers and electrostatic memory), the IAS machine (built largely under von Neumann's supervision at Princeton; it was the ancestral 7094), and J. Forrester's Whirlwind at MIT (introducing core memory). Eckert and Mauchly went on to build BINAC, and later its decimal counterpart UNIVAC, which introduced simultaneous read-write-compute.

4.



(The corresponding "OR of AND's" would require one more literal.)

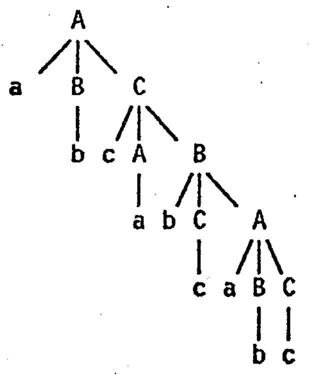
5. (a) Heuristic problem-solving, whether depth-first or breadth-first, is essentially based on constructing a tree in which the nodes represent problems or subproblems. As a special case, the nodes of game-playing trees represent positions in a game. Trees are also useful for representing formulas, e.g. in predicate calculus or for symbol manipulation. Similarly they can be used to represent structure in the sentences of natural languages. Tree structures also facilitate information retrieval from large files by having each page of the file be either a "leaf" or a node with a high branching factor.

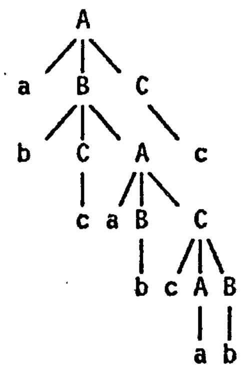
(b) Binary trees are often used to maintain symbol tables so that they may be easily kept in sorted order. Compilers often represent algebraic formulas or parse diagrams as trees. Equivalence declarations can be neatly handled using oriented trees. Also, priority queues have a convenient representation as binary trees.

(c) Adding a sequence of minimal error propagation is readily viewed in terms of binary trees. Multivariate polynomials are sometimes best represented as trees.

(d) Trees are used in the representation of environments or control flow in MULTICS-like systems and also in the representation of networks for design automation.

6. (a)





(b) $A \to^* abC \to^* abcAb$

$A \to^* aBc \to^* abcAc$

$A \to^* abC \to^* abcaB \to^* abcabcA$

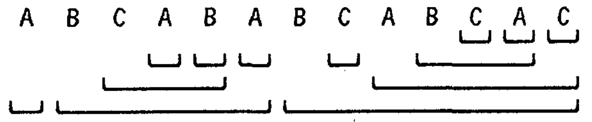
$A \to^* abC \to^* abcaB \to^* abcabCa \to^* abcabcaBa \to^* abcabcabcAa$

From the last derivation and using the third derivation with the first two, we find that $A \to^* (abc)^3Ax$ for $x \in \{a, b, c\}$. Hence the result follows by induction after we also show that $A \to^* abcabcx$ for $x \in \{a, b, c\}$.

(c) Procedure A, beginning at position P, will return TRUE if and only if $x_P = a$. The variable P will move to $f(P)$ on exit from A, where

$$f(P) = \begin{cases} P, & \text{if } x_P \neq a \\ P+1, & \text{if } x_P = a \text{ and } x_{P+1} \neq b \\ P+1, & \text{if } x_P = a,\ x_{P+1} = b, \text{ and } x_{f(P+1)} \neq c \\ f(f(P+1)), & \text{if } x_P = a,\ x_{P+1} = b, \text{ and } x_{f(P+1)} = c. \end{cases}$$

Similar statements apply to B and C. Therefore the action of the program can be diagrammed by drawing nested intervals from P to $f(P)$ as follows:



The program therefore executes MATCH(".") when scanning the second character. Because it does *not have the ability to back up and reconsider all of its previous decisions, it fails to discover the* parse $A \to^* aBc \to abCAc$ where $C \to^* cab$ and $A \to^* abcabca$.

(d) The $f(P)$ argument in part (c) implies that if A is called with $P = P_0$ and exits TRUE with $P = P_1$, the sequence $x_{P_0} \ldots x_{P_1 - 1}$ is derivable from A. (Proof by induction on $P_1 - P_0$.) Hence the program can't ACCEPT without having effectively discovered a valid parse tree.

7. Resolve $p \vee \sim q$ with $p \vee q \vee r$ getting $p \vee r$.
Resolve this with $r \vee \sim p$ getting r.
Resolve $p \vee \sim q$ with $\sim p \vee \sim q \vee \sim r$ getting $\sim q \vee \sim r$.
Resolve this with $q \vee \sim r$ getting $\sim r$.
Resolve r with $\sim r$ getting $\emptyset$.

Five steps are necessary because it takes two steps to single out any one literal and then two more to get its negation.

8. (a) $\lfloor 10^7 + 10^7\epsilon + .5 \rfloor > 10^7$ implies $\epsilon \geq 5\times10^{-8}$. Answer: 5×10^{-8}.
(b) $\lfloor 10^8 - 10^8\epsilon + .5 \rfloor < 10^8$ implies $\epsilon > 5\times10^{-9}$. Answer: 5.0000001×10^{-9}.

9. (a) The dictionary defines "buffer" as something used to lessen a shock. In computer terminology it generally stands for a place where data is held or gathered by one process before being used by another. Intuitively it stands "between" two programs, two devices, etc.

(b) A buffer is often used between a CPU or memory unit and an I/O device. For example, the data from the device is assembled into words, then written into memory.

(c) Data may be read into an area of memory (used as a buffer) and processed serially from that area while reading into another. Similarly, data may be stored into a buffer after which it is written out from the buffer. Also, in Algol-like languages, a run-time stack serves as a buffer to hold values of parameters from the main program.

10. The final value of f is zero. Working backwards we can determine that, just before k was set to 4, the configuration must have been

```
   j  = 1 2 3 4 5 6 7 8
 x[j] = R N M O E G V C
 p[j] = 6 7 6 8 5 6 8 0
   f  = 2.
```

(Note that $f = 2$ since it must have been 2 then 7 then 8 and then 0 because $p[7] = 8$ and $p[2] = 7$.) The answer is

```
   j  = 1 2 3 4 5 6 7 8
x[j] = M C G V E R N O   (misspelling intentional)
p[j] = 8 0 4 2 3 7 1 5
   f  = 6.
```

WINTER 1973 COMPREHENSIVE EXAM

1. The expected number of probes is different depending on whether the item is known to be present or not present.

When the item is not present, the expected number of probes is

$$1+\frac{K}{N}+\frac{K(K-1)}{N(N-1)}+\frac{K(K-1)(K-2)}{N(N-1)(N-2)}+\cdots+\frac{K(K-1)\cdots(1)}{N(N-1)\cdots(N-K+1)},$$

where K/N is the probability of a collision on the first probe, $K(K-1)/N(N-1)$ is the probability of a collision on the second probe, etc. The sum is

$$1+\frac{K}{N-K+1}=\frac{1}{1-K/(N+1)}\approx\frac{1}{1-\alpha},\quad \text{where } \alpha=\frac{K}{N}.$$

When the item is known to be present, then the expected number of probes to find it is the same as the expected number of probes to place it in the table, as determined above. We get

$$\frac{1}{K}\sum_{k=1}^{K}\frac{1}{1-(k-1)/(N+1)}\approx\frac{1}{K}\sum_{k=0}^{K}\frac{1}{1-k/N}\approx\frac{N}{K}\int_0^{\alpha}\frac{dx}{1-x}=-\frac{1}{\alpha}\log(1-\alpha).$$

Thus, the final answer is approximately

$$\frac{1-P}{1-\alpha}-\frac{P}{\alpha}\log(1-\alpha),\quad \text{where } \alpha=\frac{K}{N}.$$

2. (a) $2\cdot9\cdot10\cdot10\cdot(100+100+1)+1=361801$.

(b) (1) Let $x=+.500\times10^0$, $y=+.200\times10^0$, $z=+.666\times10^0$. Then $(x\otimes y)\otimes z=(+.100\times10^0)\otimes(+.666\times10^0)=+.666\times10^{-1}$, but $x\otimes(y\otimes z)=(+.500\times10^0)\otimes(+.133\times10^0)=+.665\times10^{-1}$.

(2) Let $x=+.500\times10^{-3}$, $y=+.500\times10^{-3}$, $z=+.100\times10^0$. Then $(x\oplus y)\oplus z=(+.100\times10^{-2})\oplus(+.100\times10^0)=+.101\times10^0$, but $x\oplus(y\oplus z)=(+.500\times10^{-3})\oplus(+.100\times10^0)=+.100\times10^0$.

(c) Let $x=+.100\times10^4$, $y=+.100\times10^0$, $z=-.100\times10^4$. Then $(x\oplus y)\oplus z=(+.100\times10^4)\oplus(-.100\times10^4)=+.000\times10^{-100}$, but true result $=y=+.100\times10^0$, 100% error.

3. The given instruction sequence can fail in certain situations. Suppose processor 1 executes its LDA from a zero lock, and then immediately (before processor 1 does its SET) processor 2 executes its LDA, still from a zero lock. Both processors then set the lock and access the data base simultaneously.

We can reduce the time between testing the lock and setting it by reversing the instructions BRP and SET. But then consider the following sequence of actions: processor 1 sets the lock and accesses the data base; processor 2 executes its LDA; processor 1 finishes and executes its CLR; processor 2 executes its SET. This results in an infinite wait loop because the CLR was missed by processor 2.

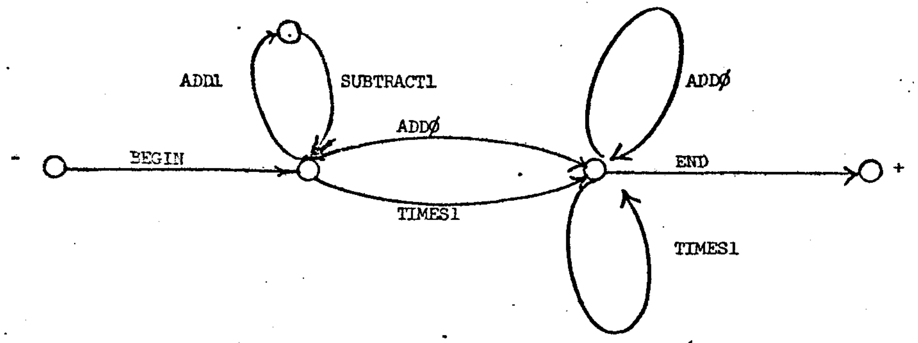
What is needed is an instruction that does the testing and setting of the lock in a single memory cycle so that no other processor may intervene. Let us call this the TAS (test and set) instruction, which loads the accumulator from the specified memory word and sets the memory word to 1, in a single read/write cycle. Then a correct instruction sequence would be:

```
    TEST    TAS     lock
            BRP     TEST
      access data base
              .
              .
              .
      end access
            CLR     lock
```

The increment and skip on zero (ISZ) or similar instruction may also be used, but then care must be taken to subtract in order to balance the add's.

Some other problems which might need to be considered: busy waiting; arbitrarily long waits (need priority scheme); and processor interrupts.

4. (a) `BEGIN (ADD1 SUBTRACT1)* (ADD0 + TIMES1)* END`



(b) <initial segment> → ADD1 <initial segment> SUBTRACT1 | SUBTRACT1 <initial segment> ADD1 | <initial segment> <initial segment> | <empty>

Note that the above language is context free. Also, it is easy to construct a pushdown automaton to accept the language.

However, the language is not regular. Suppose that it could be recognized by a finite automaton. Then upon reading the sequence $\text{ADD1}^{(i)}$, the automaton would have to be in a different state for each i in order to accept a following $\text{SUBTRACT1}^{(j)}$ if and only if $j = i$. Thus a finite number of states is not sufficient.

5. (a) In order to estimate s by the Aitken extrapolation method, we must eliminate c. We can write

$$g_k - g_{k+1} = s + \sum_{p=1}^{\infty} a_p(1-\lambda_p)\lambda_p^k = s + \sum_{p=1}^{\infty} a_p'\lambda_p^k$$

where $a_p' = a_p(1-\lambda_p)$. Thus, we may estimate s by applying the Aitken method to the sequence $h_k = g_k - g_{k+1}$.

(b) Given h_0, h_1, h_2, and $h_0' = h_0 - (h_1-h_0)^2/(h_2-2h_1+h_0)$, we have $h_0' = s$ when $h_k = s + a_1'\lambda_1^k$, or $g_k = c - sk + a_1\lambda_1^k$.

(c) We have $h_0 = -17.0$, $h_1 = 10.0$, $h_2 = -3.5$, so $h_0' = -17.0 - (27)^2/(-3.5-20-17) = 1 \approx s$.

6. (a) The procedure does *not* terminate.

1]	$\{\sim p(x) \vee p(f(x))\}$	5]	$\{p(f(a))\}$	from 1, 2
2]	$\{p(a)\}$	6]	$\{p(f(f(a)))\}$	from 1, 5
3]	$\{q(a)\}$	7]	$\{p(f(f(f(a))))\}$	from 1, 7
4]	$\{q(a) \vee \sim q(b)\}$			

The procedure will continue to produce clauses of the form $\{p(f(f\ldots f(a)\ldots))\}$, and no other resolvents. In particular the empty clause will not be derived, so the set of clauses is satisfiable.

(b) The procedure terminates.

1]	$\{p(f(x)) \vee \sim q(x)\}$	5]	$\{p(f(a)) \vee r(a)\}$	from 1, 3
2]	$\{\sim p(g(a))\}$	6]	$\{q(a) \vee \sim q(a)\}$ (tautology)	from 3, 4
3]	$\{q(a) \vee r(a)\}$	7]	$\{r(a) \vee \sim r(a)\}$ (tautology)	from 3, 4
4]	$\{\sim r(x) \vee \sim q(x)\}$			

There are no further resolvents. Again, since the empty clause cannot be derived, the set is satisfiable.

(c) The procedure terminates with the empty clause.

1]	$\{p(x) \vee q(x)\}$	6]	$\{\sim r(a)\}$	from 4, 5
2]	$\{\sim p(a) \vee r(a)\}$	7]	$\{\sim p(a)\}$	from 6, 2
3]	$\{\sim q(y) \vee r(y)\}$	8]	$\{\sim q(a)\}$	from 6, 3
4]	$\{\sim r(a) \vee p(a)\}$	9]	$\{q(a)\}$	from 7, 1
5]	$\{\sim p(z) \vee \sim r(z)\}$	10]	$\varnothing$	from 8, 9

The set is unsatisfiable since a contradiction was derived.

7. Let i and j denote integers which are assumed to have integer represenions in sign-magnitude, ones' complement, and two's complement. Let F_i and F_j denote the real numbers whose floating-point representations have the same bit pattern as the integer representations of i and j, respectively.

In order for the comparison instruction to work, we first note that the sign bits for floating-point and integer representations must be the same. (Consider the case when i and j have opposite signs.) Secondly, we want $F_i = F_j$ iff $i = j$, and therefore the floating-point representations must be unique. This requires the use of normalized mantissas.

For nonnegative numbers, all three fixed-point representations are the same. In order for the nonnegative floating-point representations to have the same order properties, they must have the (biased) exponent on the left. We then have $F_i < F_j$ iff $i < j$ for $i, j \geq 0$. Now for negative numbers, we need $F_{-i} = -F_i$, so that the same negation operation works for both fixed and floating-point numbers. We then have, for $i, j < 0$,

$$F_i < F_j \text{ iff } -F_{-i} < -F_{-j} \text{ iff } F_{-i} > F_{-j} \text{ iff } -i > -j \text{ iff } i < j.$$

For ones' complement, the definition of a normalized mantissa is that the sign bit and the first bit of the mantissa are different. This definition also applies to two's complement, except for the case of $-.5_{10}$, which has mantissa $.10000\ldots$. It should be noted that arithmetic is somewhat complicated for the cases of ones' and two's complement.

The above scheme is used on the Univac 1103A and the CDC 6600 (ones' complement), and the PDP-6 Type 166 (two's complement).

8. Twenty questions.

1) (b) CDC 6600
2) cache memory
3) (b) B 5500
4) (b) SNOBOL
5) Illiac IV, STAR
6) PDP 8-e, Intel MCS-4, TI 960
7) no
8) memory
9) $2^{2^3} = 2^8 = 256$
10) constant $\times$ $n \log n$, where n = number of keys
11) linear, since $f(x)$ has a double root at a
12) regula falsi: lower rate of convergence, but 1/2 as many function evaluations per step
13) (c) 186,000 miles/sec $\times$ 5280 ft/mile $\times$ 10^{-9} sec $\approx$ 1 ft
14) least recently used, pages not in working set
15) exclusive-or
16) ones' complement, because complementation is the same as negation
17) instruction counter, registers, other state information
18) stack; queue
19) Dendral, MATHLAB
20) Here are a few:

Q: What is Jim Wilkinson's favorite drink?
A: Reduction on the blocks with a zero chaser.

After constructing a small computer-controlled vehicle with a general hemispherical shape, the experimenter exclaimed: "My bug has a program!"

Q: How would you describe the state of a number-crunching machine?
A: Chompin' at the bit.

To iterate is human, to recurse is divine.

See Knuth, Vol. 1, index.

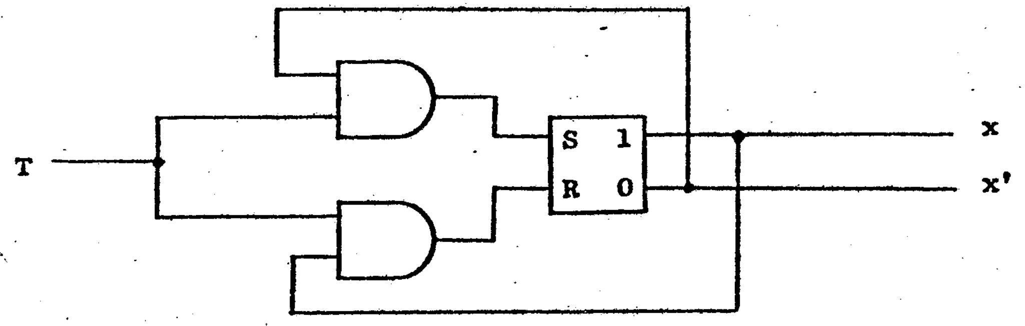
graffiti: "Free the FORTRAN 4"
scribbled below: "WATFOR?"

Here is how the statement "All odd numbers ≥ 3 are prime" would be proved by various scientists.

Mathematician: "3 is a prime, 5 is a prime, 7 is a prime, by induction they're all prime."
Physicist: "3 is a prime, 5 is a prime, 7 is a prime, 9 is *not* a prime – must be experimental error."
Engineer: "3 is a prime, 5 is a prime, 7 is a prime, 9 is a prime – they're all prime."
Computer scientist: "3 is a prime, 3 is a prime, 3 is a prime, ... "

SPRING 1973 COMPREHENSIVE EXAM

1. (a)



(b)

	T	S	R	y_1	y_2	z_1	z_2
Initial state (no clock pulse present)	0	0	0	0	0	0	0
After arrival of 1st clock pulse	0	1	0	0	1	0	1
After arrival of 2th clock pulse	1	0	1	1	0	1	0
After arrival of 3th clock pulse	0	1	0	1	1	1	1
After arrival of 4th clock pulse	0	0	0	1	1	1	1
After arrival of 5th clock pulse	0	0	0	1	1	1	1
After arrival of 6th clock pulse	0	0	0	1	1	1	1

2. Each of the languages takes the form $\{ a^i b^j \mid j = f(n,i) \}$.

(a) (d) $f(n,i) = n-i$ or n/i. The corresponding languages are regular because the languages are finite. ($i \le n$, $j \le n$)

(b) (c) $f(n,i) = n+i$ or $n*i$. The corresponding languages are context free, but not regular. Informally, this is because each of the languages requires an arbitrarily large amount of memory in order to be recognized. More formally, consider a finite automaton trying to recognize one of these languages and look at the set of states it enters while recognizing the substring x^i. Since this set of states is finite, for some $k \ne i$ the finite automaton is in the same state at the end of a^k as of a^i. But then if it accepts $a^i b^{f(n,i)}$ it also accepts $a^k b^{f(n,i)}$ and this is *not* in the language. (Clearly the important thing about f is that $f(n,i)$ as a function of i takes on an infinite set of values.)

Both languages are context free, which can be shown by giving grammars or designing push-down automata to accept them. Grammars for each are given below.

$j = n+i$:	$S \to aTB \mid aB$	$j = n*i$:	$S \to aSB \mid aB$
	$T \to aTb \mid ab$		$B \to b^n$
	$B \to b^{n+1}$		

(e) $f(n,i) = i^n$ (or n^i). For $n > 1$, the language is *not* context free. For $n = 1$, in the first case it is context free, and in the second case it is regular. To show that the language is not context free for $n > 1$, we show that the *uvwxy* theorem (Hopcroft and Ullman, p. 57) does not hold. Suppose by the theorem that $z = uvwxy$, $|vx| > 0$ is such that uv^iwx^iy is in L for all $i \ge 0$. By the simple form of the language, $vx = a^k b^l$ for some k, l both nonzero. Now consider $(i+k)^n - i^n$. As a function of i, this is unbounded as $i \to \infty$. (*Proof:* Consider the binomial expansion.) So simply choose an integer m such that $(km+k)^n - (km)^n > l$. Then if the *uvwxy* theorem holds, uv^mwx^my is in L and hence, by definition of L, $uv^mwx^my = a^i b^{i^n}$ for some i. Then $uv^{m+1}wx^{m+1}y = a^{i+k} b^{i^n+l}$ is in L. But by our choice of m, $i^n + l \ne (i+k)^n$. Hence this last string cannot be in L. The contradiction shows L is not context free; however it is easily shown to be recursive.

3. (a) The grammar is not left-recursive. Clearly there is no direct left recursion. For any non-terminal symbol U, let $L(U)$ be the set of all leftmost symbols in strings producible from U. Then we have

$$L(S) = \{A,\ C,\ a,\ c\}$$
$$L(A) = \{C,\ a,\ c\}$$
$$L(B) = \{C,\ A,\ c,\ a\}$$
$$L(C) = \{c\}$$

Since none of the sets $L(U)$ contains U, there is no indirect left recursion either.

(b) If the definitions given in Gries are used, the precedence table is

	S	A	B	C	a	b	c
S							
A				=		=	<
B						>	>
C		<	=	<	<	>	><
a		<	=	<	<		<
b						>	>
c					>	>	>

If the table is constructed as described in a paper by Wirth and Weber ("EULER: A Generalization of ALGOL, and its Formal Definition: Part I", *CACM*, Vol. 9, No. 1, January 1966), there are some additional > relations. In either case, there is a conflict $C > c$, $C < c$, so this is not a (1,1)-precedence grammar.

(c) This is most easily done by starting with the added production and working backwards. Clearly, C produces the empty string so any non-terminal that produces C will also produce the empty string. As a result, B can produce the empty string. It then follows that A and S can also produce the empty string.

(d) With the addition of the production $C \to \epsilon$, a top-down recognizer could loop forever when parsing sentences with this grammar. When the recognizer is looking for an A as a C followed by a B, the C can be satisfied by the empty string. Now, however, in looking for a B, the recognizer might try to use the production $B \to Ab$. It must now find an A again, and this loop will continue forever.

(e)

	S	A	B	C	a	b	c	
S								
A				=		=	<	<
B						>	>	>
C		<	<=	<	<	><	><	><
a		<	<=	<	<	><	><	><
b						>	>	>
c					>	>	>	>
					>	>	>	>

Again there are conflicts in the table so this new grammar is not a (1,1)-precedence grammar.

4. (a) Paging introduces internal fragmentation within the fixed page size. Segmentation introduces external fragmentation due to variable length segments.

(b) Paging systems typically must solve the page replacement problem (which page to replace when a page fault occurs). Some paging systems with limited mapping ability must also solve page *placement* problem. All must determine when to bring a page into memory (e.g. demand paging, working set preloading, etc.).

Segmentation systems must solve a related segment-replacement problem. Since segments are variable length, there is also a garbage collection problem to be solved. The system must also decide when to fetch a segment.

(c) Program sharing is awkward since one can typically only make two processes share pages. Furthermore, if the programs are not self-relative, they must be placed in the same place in each process' virtual memory. This restriction can make sharing of program libraries very difficult since virtual memory for the entire library must be set aside and not used by any process wishing to access any subset of the library.

When each program (procedure, etc.) is a separate segment, then sharing is easily accomplished since each procedure has its own virtual space. Typically segmented memory spaces are very much larger than paged memory spaces (MULTICS has 2^{18} words each; the XDS Sigma 7 has 2^8 pages of 512 words each.).

(d) Paging permits a system with small physical storage to simulate much larger storage space (hence, "virtual storage"). Not all pages of the virtual space need be present in the physical memory at once.

Segmentation, without paging, usually requires all of a segment to be in memory at once (note that this does not mean that the maximum possible segment be present, only as much as has been declared or accessed so far). Segments are allocated to sequential portions of memory and relocation is achieved by the use of base registers, descriptors, and indirect addressing. The maximum permissible segment is typically bounded by the physical memory size. Of course, one can combine paging and segmentation (as in MULTICS) to obtain the advantages of both.

5. (a) index registers, general registers
(b) base registers, self-relative instruction set
(c) push-down stack instruction set
(d) interrupts with masking, enabling, and disabling primitives; byte-string oriented instructions: edit, move, compare, translate, translate and test
(e) test-and-set instruction
(f) "previous instruction counter"-register; metabits on each memory word to permit trapping on any access, write access, read access, execute access

```
6.     Given: IS: monkey's place (MP) = place1, box's place (BP) = place2
                  ape's place (AP) = place3, contents-of-monkey's-hand (CMH) = empty (E)
                  contents-of-ape's-hand (CAH) = empty
              GS: CMH = bananas (B)

Goal 1 Transform IS into GS
 Goal 2 Reduce difference D3 between IS and GS
  Goal 3 Apply Get Bananas (GB) to IS with a=monkey
   Goal 4 Reduce D2 on IS
    Goal 5 Apply Move-Box (MB) to IS with x=under bananas (UB)
     Goal 6 Reduce D1 on IS
      Goal 7 Apply Walk (W) to IS with x=place2, a=ape
     State S1: MP=place1, BP=place2, AP=place2, CMH=E, CAH=E
     Goal 8 Apply MB to S1 with x=UB
   State S2: MP=place1, BP=UB, AP=UB, CMH=E, CAH=E
   Goal 9 Apply GB to S2 with a=monkey
    Goal 10 Apply climb
     Goal 11 Reduce D0 on S2
      Goal 12 Apply W with x=UB, a=monkey
     State S3: MP=UB, BP=UB, AP=UB, CMH=E, CAH=E
     Goal 13 Apply climb
```

State S4: MP=on-box, BP=UB, etc.
Goal 14 Apply GB to S4
State S5: CMH=bananas, etc.
(Goal 1 succeeds)

7. (a) The first program writes "1" and the second writes "2".
(b) The issue this problem was intended to point out is the lexical scope properties of ALGOL. In LISP, the value of A used in the first program would be "1", the dynamically most recent. There are times when we want to use the dynamic value (this example is probably one of them). However, using dynamic bindings makes the program harder to read and precludes some code optimizations.

8. (a) The obvious approach is to use some procedure for numerical integration. (We may assume that the exponential function is available to us.) For simplicity, we choose the trapezoidal rule. The error involved in using the trapezoidal rule with step size h is given by

$$\int_0^x f(x)\,dx = T - \frac{h^2}{12}xf''(u), \text{ for some } u,\ 0 \le u \le x.$$

In our case $f(x) = e^{-x^2}$ and $f''(x) = (4x^2-2)e^{-x^2}$. By examining the third derivative, we find that $|f''(x)| \le 2$ for $0 \le x < \infty$. Thus the error in our computation of erf(x) is bounded by $(h^2x)/(3\sqrt{\pi})$ so to achieve an error of ϵ we must choose $h \le \sqrt{3\sqrt{\pi}\epsilon/x}$. For example, to compute erf(3.0) to an accuracy of 10^{-4} we would need $h \approx .0133$, i.e. about 225 steps.

Now that we have a way of computing erf(x), we can compute the inverse error function inverf(z) by finding the root of [erf(x) - z]. Since the derivative of erf(x) is readily available (it is $(2/\sqrt{\pi})e^{-x^2}$), we may use Newton's method. However, even with this well-behaved function, Newton's method is not guaranteed to converge, and the choice of initial guess is crucial (thus it may be better to use a safer method, such as bisection). For example, if we take $x_0 = 1.0$, then the iteration diverges for $z \le .4$. It turns out that choosing $x_0 = z$ is a much better approximation. Now assuming that we find a root x such that $|\text{erf}(x)-z| < \epsilon$, we would like to know the error $d = w-x$, where $w = \text{inverf}(z)$? We have

$$\text{erf}(w) = \text{erf}(x) + d*\text{erf}'(u), \text{ for some } u \text{ between } w \text{ and } x.$$

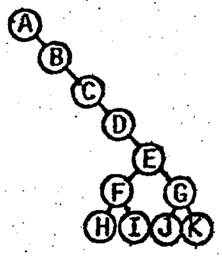
Since $|\text{erf}(w)-\text{erf}(z)| < \epsilon$, we find that $|d| \le \epsilon(\sqrt{\pi}/2)e^{-x^2}$. For example, if $x = 2.5$ and $\epsilon = 10^{-4}$, then d can be as large as .05.

(b) If we compute inverf(.999) by our methods above, we find that we need on the order of 1000 evaluations of e^{-x^2}. A much better way to solve our problem is to use power series or rational function approximations for erf and inverf. These may be found in most handbooks of numerical methods.

WINTER 1974 COMPREHENSIVE EXAM

1. (a) preorder: 1 2 3 4 5 6 7 ; inorder: 3 2 4 1 6 5 7 ; postorder: 3 4 2 6 7 5 1.
(b) (1) G E D H I F B C A. (2) 4 7 5 2 6 1 3 , 7 4 5 2 6 1 3 , 7 5 4 2 6 1 3 , 7 5 2 4 6 1 3 , 7 5 2 6 4 1 3 , 7 5 2 6 1 4 3 , 7 5 2 6 1 3 4 . (3) There are many solutions, all based on the recursive nature of the algorithms as applied to the grandson trees. For example, any permutation with (G,E) or (F,H,I) not adjacent cannot be achieved. Nor can any permutation have (A,B,C) and (F,H,I) in different relative orders.

(c) It depends on the tree. Run any of the algorithms on the tree, and mark those steps which operate on something non-null. Let n be the number of marked steps. Then $0 \le n \le 7$, and the answer is $(n!)$. For a counterexample to many solutions, see the tree below, for which all 7! algorithms are distinct.



(d) Yes. The result is easily proved by induction.

2. (a) $R(x) = f(x)-P(x)$ has $n+2$ maxima and minima which are equal in absolute value and alternate in sign.

(b) Let $P(x) = Ax + B$. Then $R(x) = ax^2 + bx + c - (Ax + B)$ and $R'(x) = 2ax + b - A$, so the extrema of $R(x)$ in $(-1,1)$ occurs at $x_0 = (A-b)/2a$. By part (a), we must have $R(-1) = -R(x_0) = R(1)$. Solving these equations for A and B, we find that $P(x) = bx + c + a/2$.

(c) $ax^2+bx+c = (a/2)T_2(x)+bT_1(x)+(c+a/2)T_0(x)$. The best linear polynomial is $bT_1(x)+(c+a/2)T_0(x)$ $= bx+c+a/2$, as before. Here we have used the fact that, in general, $(1/2^{n-1})T_n(x)$ is the polynomial beginning with x^n having the smallest maximum.

(d) The interval [0,1] maps onto [-1,1] by the transformation $x' = 2x-1$. Then $x = (x'+1)/2$, so we consider $a((x'+1)/2)^2+b(x'+1)/2+c$ and proceed as in (c). Then we map back to [0,1].

(e) $f(x) = (x-x_1)\dots(x-x_n)$ is minimized when $2^n f(x) = T_n(x) = \cos n\theta$, where $\cos\theta = x$. The zeros of $T_n(x)$ are $x_i = \cos\theta_i$, where $\cos n\theta_i = 0$. Therefore $x_i = \cos[(2i-1)\pi/2n)]$.

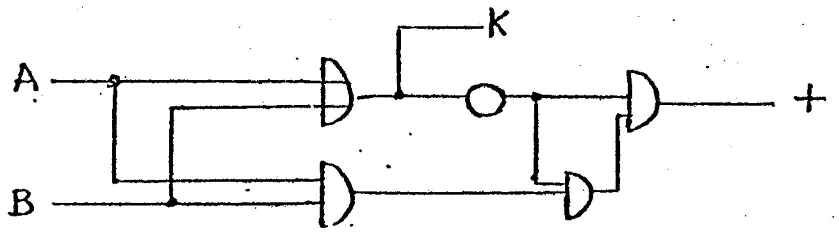
3. The tables for a half-adder are

Add	-1	0	+1
-1	+1	-1	0
0	-1	0	+1
+1	0	+1	-1

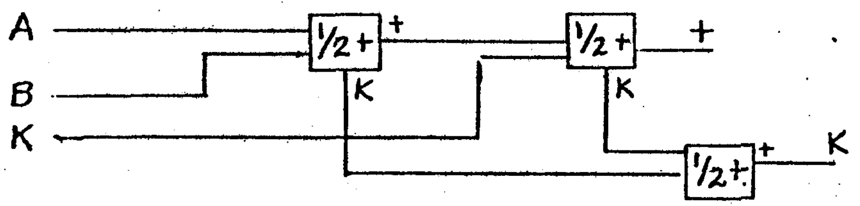
Carry	-1	0	+1
-1	-1	0	0
0	0	0	0
+1	0	0	+1

We get these as follows: Add = And(And(Neg(Or),And),Neg(Or)) ; Carry = Or.

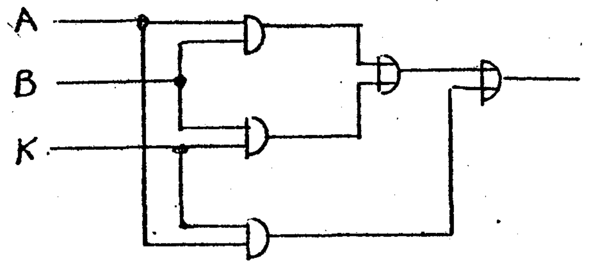
The diagram for the half-adder is



A full adder can be made from 3 half-adders, as shown below, but this is rather slow. An easy improvement which produces a faster carry is also shown below.



Full adder



Fast carry

4. (a) $f(0) = -1$, $f(1) = 5$. Therefore, f has a zero in $[0,1]$.

(b) Let $g(x) = x - mf_0(x)$. The iteration $x_{n+1} = g(x_n)$ converges if $|g'(x)| \leq L < 1$. But $g'(x) = 1 - mf'(x)$ in $(0,1)$, and $f'(x) = 3x^2+6x+2$. Since $f'(0) = 2$, $f'(1) = 11$, and f' is increasing, we obtain the restriction $0 < m < 2/11$. Let us try $m = 1/11$. It can then be shown that $g(x) \in [0,1]$ for all $x \in [0,1]$. Therefore all conditions are satisfied for convergence. If $x_0 < 0$, then the iteration becomes $x_{n+1} = x_n + 1/11$, so after about $11|x_0|$ iterations x_n reaches $[0,1]$ and we will have convergence (very slow!). If $x_0 > 1$, the iteration becomes $x_{n+1} = x_n - 5/11$, so again we will eventually have convergence, after about $(11/5)x_0$ iterations.

(c) Does not converge because $|g'(x)| > 1$ near the root.

(d) Converges; use the fact that $f'(x)$, $f''(x) > 0$ for $x \in [0,1]$.

5. The object of this problem is to illustrate that clock interrupts can also serve as device interrupts. When the program wants data it sets *rdflag* := true and tests it for false as a signal that the data has arrived. An array *rdarray* of size 1:*m* is used for communicating the actual data. Similarly, when the program wants to write, it fills *wrtarray* of size 1:*n* and sets *wrtflag* := true. The interrupt routine will set *wrtflag* := false when the array is ready for more data. In addition, the routine requires two hardware flags *readready* and *writeready* which indicate when data is ready for reading or when the output unit is ready for more information.

```
foo: if rdflag ∧ readready then begin
        readready ← false;
        rptr := rptr + 1;
        rdarray[rptr] := read;
        if rptr = m then begin rptr := 1; rdflag ← false end
     end;
     if wrtflag ∧ writeready then begin
        writeready ← false;
        write(wrtarray[wptr]);
        wptr := wptr + 1;
        if wptr > n then begin wptr := 1; wflag ← false end;
     end;
     return;
```

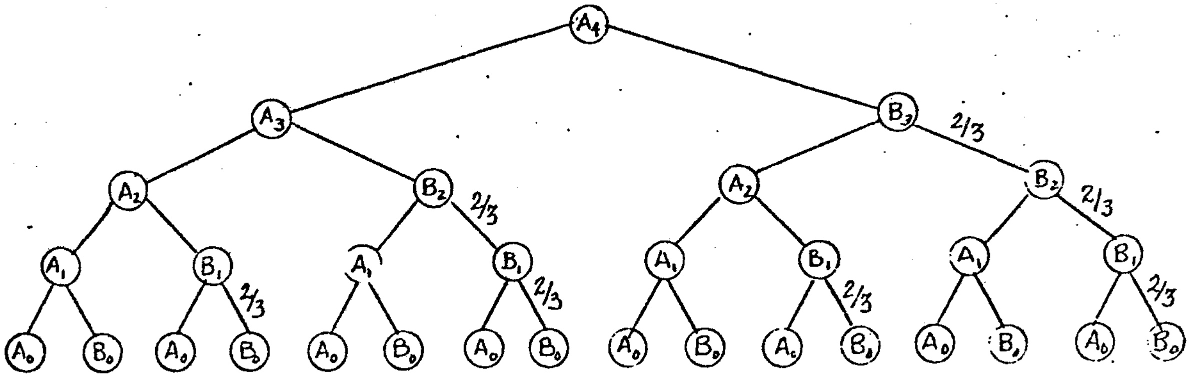
6. (a) True. It is easy to see that $L(G)$ contains only words from $\{0,1\}^*$ of length ≤ 6. This means that the language is finite, and therefore regular.

(b) True; consider the grammar with productions $\{ S \to aSa,\ S \to bSb,\ S \to \epsilon \}$.

(c) False. Consider the word $a^n c a^n$. A finite automaton accepting the language must be in a different state after reading each of the first n a's. However, this must be true for all n, which is impossible.

(d) False. Suppose the language were context free. Then for sufficiently large k, the word $a^k b^k c^k$ can be written in the form $uvwxy$ where not both v and x are empty and where all words of the form $uv^i wx^i y$ must also be in the language ($uvwxy$ theorem). Now it can be seen that neither v nor x can contain two distinct letters, so at least one of the three letters a, b, c does not occur in v or x. But then the word uv^2wx^2y is not in the language, a contradiction.

7.



The "no deep cutoff" approximation assumes that most cutoffs are not deep and makes the probability of no cutoff, where one might occur given the presence of the parent node, the same throughout the tree (in this case, 2/3).

Let A_i and B_i represent the expected number of bottom positions in the corresponding trees above. For a tree of depth d, we are interested in the value of A_d. Clearly $A_0 = B_0 = 1$, and for $i \geq 1$ we have the recurrence relations $A_i = A_{i-1} + B_{i-1}$ and $B_i = A_{i-1} + (2/3)B_{i-1}$. These recurrences may be solved using generating functions, obtaining

$$A_d = \frac{1}{b-a}\left[\left(1+\frac{3}{b}\right)\left(\frac{1}{b}\right)^d - \left(1+\frac{3}{a}\right)\left(\frac{1}{a}\right)^d\right],\ \text{where } a = \frac{\sqrt{37}-5}{2} \text{ and } b = \frac{\sqrt{37}+5}{2}.$$

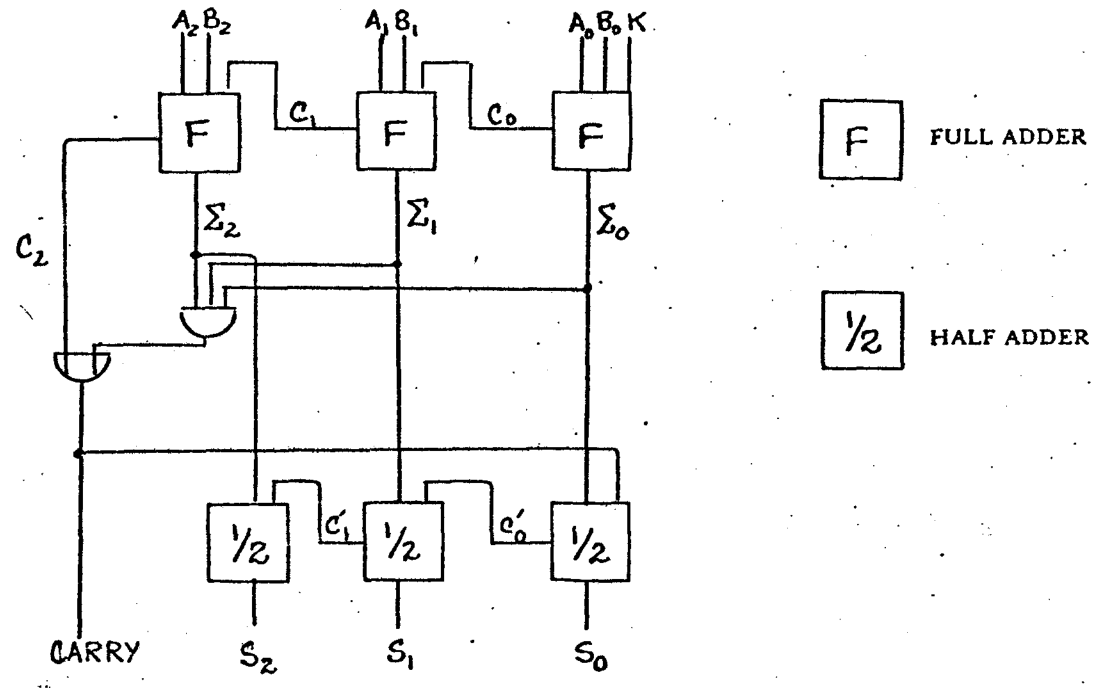
We find that A_d is approximately $1.075(1.847)^d$.

8. (See logic diagram at top of next page.)

9. Call by name: substitution rule. Call by value: value is passed. Use call by name for returning results in variables and also for passing arrays. Use call by value when value only is passed and variable is not to be changed in the outside world.

(a) $f(2) = 6p_0 + 5q_0$; $f(3) = 12p_0 + 10q_0$

(b) $f(2) = 6p_0 + 5q_0$; $f(3) = 52p_0 + 34q_0$



10. We ensure that the universe has exactly the three distinct elements $\{1,2,3\}$ by writing

(1) $\forall x\ (\ x=1 \lor x=2 \lor x=3\ ) \land \sim(1=2) \land \sim(1=3) \land \sim(2=3)$

The adjacency relations between them are written

(2) $\forall x \sim A(x,x) \land \forall x\ (A(x,y) \supset A(y,x)) \land A(1,2) \land A(2,3) \land \sim A(1,3)$

The adjacency of the apples with coordinates (x_1, y_1, z_1) and (x_2, y_2, z_2) is expressed by a predicate of six arguments $Adj(x_1, y_1, z_1, x_2, y_2, z_2)$ defined by

(3) $\forall x_1 y_1 z_1 x_2 y_2 z_2\ [\ Adj(x_1, y_1, z_1 x_2, y_2, z_2) \equiv (\ x_1=x_2 \land y_1=y_2 \land A(z_1, z_2)\ ) \lor$
$(\ x_1=x_2 \land A(y_1, y_2) \land z_1=z_2\ ) \lor (\ A(x_1, x_2) \land y_1=y_2 \land z_1=z_2\ )\ ]$

A path meeting the conditions of the problem will have an associated predicate $P(x_1, y_1, z_1, x_2, y_2, z_2)$ expressing the fact that the apple with coordinates (x_2, y_2, z_2) follows the apple with coordinates (x_1, y_1, z_1) in the path. This predicate will satisfy the following conditions.

Every square has a unique successor except the center.

(4) $\forall x_1 y_1 z_1\ (\ x_1 \neq 2 \lor y_1 \neq 2 \lor z_1 \neq 2\ ) \supset \exists !\, x_2 y_2 z_2\ P(x_1, y_1, z_1, x_2, y_2, z_2)$

The beginning square is not the center.

(5) $a \neq 2 \lor b \neq 2 \lor c \neq 2$

Every square except the beginning square has a unique predecessor.

(6) $\forall x_2 y_2 z_2\ (\ x_2 \neq a \lor y_2 \neq b \lor z_2 \neq c\ ) \supset \exists !\, x_1 y_1 z_1\ P(x_1, y_1, z_1, x_2, y_2, z_2)$

Consecutive squares on the path are adjacent.

(7) $\forall x_1 y_1 z_1 x_2 y_2 z_2\ P(x_1, y_1, z_1, x_2, y_2, z_2) \supset Adj(x_1, y_1, z_1, x_2, y_2, z_2)$

The satisfiability of the conjunction of these seven sentences expresses the solvability of the problem.

The quibble is that since the problem is unsolvable, the sentence *false* also expresses its solvability.

11. QUICKIE QUESTIONS

(a) Ervin. See index of Vol. 1, 2, or 3.
(b) 1 0 1 0 0 0 0 1 0 . (two end-around carries)
(c) -12. (Polish notation)
(d) See Nilsson, *Problem Solving Methods in Artificial Intelligence.*
(e) No; consider the requirements for placing declarations.
(f)
$$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad \text{(transitive closure)}$$
(g) 3 by 5 array.
(h) Half a byte.
(i) If F is true for all arguments, then the sentence is true. If F is false for any argument, let x be one such argument and the sentence is true.
(j) No dynamic relocation.
(k) E.g. changing x^2 to $x*x$ and $2*x$ to $x+x$ (done by compilers).
(l) A Programming Language
Backus Naur (Normal) Form
COmmon Business Oriented Language
Adel'son-Vel'skii & Landis, American Volleyball League
Channel Control Word, Counter ClockWise
(m) At the beginning to obtain initial values.
(n) S_2, because with S_1 the smaller terms of the sum are lost.
(o) $(x) = \alpha/(x+\sqrt{x^2-\alpha})$
(p) Smooth, less oscillatory.
(q) Hard to replace console lights.

SPRING 1974 COMPREHENSIVE EXAM

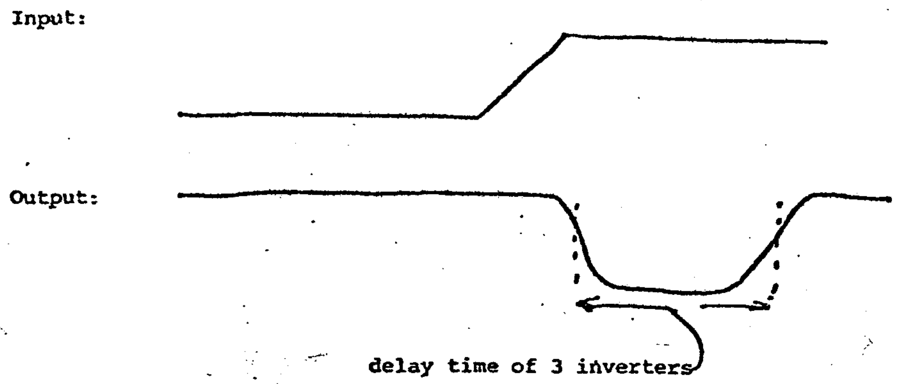
1. We first note that the sum of 14 10-bit numbers has at most 14 bits.

Consider the numbers to be summed as a 14×10 array of bits. We divide this into two 7×10 arrays, and sum each of the columns of 7 bits using a (7,3)-counter. This compresses each 7×10 array into a 3×12 array, at a total cost of 20 counters. Now we can regard the two 3×12 arrays as one 6×12 array and then compress it into a single 3×14 array, using 12 counters. Finally, we use two 14-bit adders to obtain the result. Total cost: \$20 + \$12 + 2 × \$28 = \$88.

Actually, we could have first compressed the 3×14 array into a 2×14 array, using 14 counters. (This works because the sum of each 3-bit column has only two significant bits.) Then we need just one 14-bit adder to complete the sum, thus reducing the total cost to \$74.

At this point we may realize that a (7,3)-counter can in fact simulate a one-bit full adder, as follows: Consider three of the inputs as the two bits to be summed plus a carry, and connect the other four inputs to ground (i.e. 0). Then the two low-order bits of the output represent the sum and the carry out. Thus we need never use an adder at all! Furthermore, a little thought reveals that an optimal strategy for summing the bits is to sum the columns starting with the least significant bit. (We assume that duplication of lines is not allowed.) We find that at most four counters are needed to sum each column (including carry bits from sums of earlier columns), for a total cost of 13 × \$4 = \$52. A more careful analysis can reduce the cost to as little as \$40 (is this optimal?).

2.



The delays in the inverters cause the output of the last inverter to change after I has changed. Moral: All logic circuits have delays. The circuit is dubbed a "trigger" because an output pulse is induced by an input transition from 0 to 1.

3. This problem requires a more precise definition of the functions of the computer system.

First of all, is it to be batch-oriented, single-user interactive, or timeshared? The cheapest to build is probably single-user, because batch requires a job-control facility; and timesharing requires scheduling, swapping, and protection mechanisms. However, a $300K machine would probably not be used as a single-user system. We choose a timeshared operation as the most flexible; a crude batch facility could always be added as another timesharing job. (To implement a timesharing system, we must assume that the computer hardware permits some kind of protection mechanisms, for memory and for privileged execution.)

We must certainly provide an assembler, loader, and debugger. These can be quite simple; esoteric features should be avoided in the interest of getting these pieces working quickly. Note that we will probably be using them to test our own software.

The chief components of the operating system are the file system (including device drivers for terminals, line printer, etc.) and the scheduling and memory management system (to perform swapping, etc.). A secondary component would be some kind of command interpreter that is capable of invoking "sub-systems" (i.e. user programs). The functional specification of the operating system should be done early, because the authors of other software will need to make use of its facilities (chiefly for file operations).

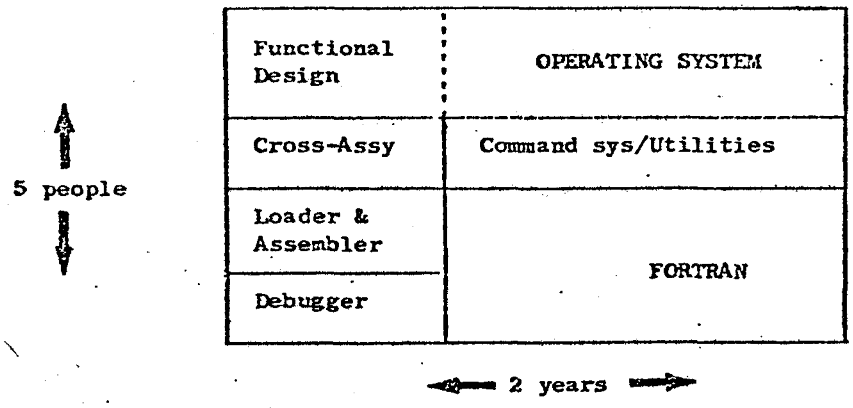
We must also provide at least one high-level language. If the machine is destined for scientific markets, this language will doubtless be FORTRAN; if for business, COBOL/RPG. Second languages (if we have time) could be ALGOL, PL/I, or BASIC.

Lastly, we will need some utility programs, which can be written as "user programs" rather than as part of the operating system. These include programs for listing disk directories, deleting files, backing up files on magnetic tape, etc. We will probably also want a simple text editor which can be used to prepare source text files.

As the above systems are being written, we must remember to allow time for writing documentation! (If a "standard" FORTRAN is used, perhaps some documentation effort can be avoided.)

A crucial question is how to get started on the new machine. We have two options: bootstrapping or cross-compiling. Cross-compiling is easier for this case, so we choose it. To that end, we define a "binary file" format: we compile programs on our existing computer into this format, load them into the new computer, and check them out. We will choose to do our coding in assembly language, so we build a cross-assembler on our existing machine. If the new machine is not available yet (or if its production schedule slips!) we can also build a simulator to check out code for the new computer on the existing computer.

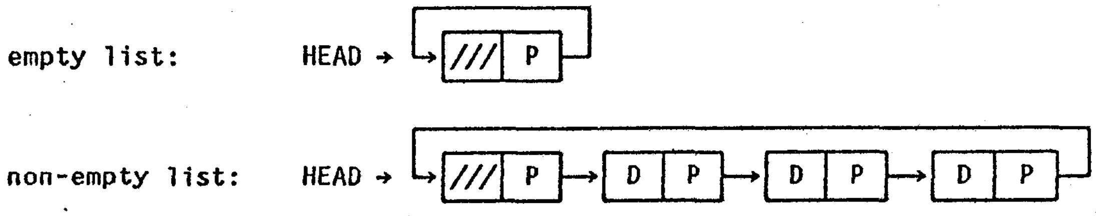
A crude time-chart for the project is shown below.



4. There are a number of problems associated with this strategy. The common intent behind base register systems, in addition to providing virtual memory, is to permit more than one process to co-exist in the main memory at one time, each one isolated in its own virtual memory. Since only one process ever has control over the CPU at a time, the physical mapping device (base register) is typically reloaded each time the system scheduler switches processes. However, I/O may be requested by many processes (into the same virtual but different physical memory locations). The composite mapper would have to know for which process an I/O operation is being performed in order to transform the device controller's memory requests into the correct physical location in memory. This in turn requires that the mapper, if it is in the memory, have direct access to all the base register values that might be needed for serving the active processes in the system. Memory access requests would have to be tagged by some process identifier so that the correct base register value is used.

As for dynamic program relocation, there is a serious problem with this idea as well. Even though the I/O requests can be redirected instantly by changing the associated base register, it is still necessary to copy the memory locations to be moved from one physical location to another. If I/O is going on (especially into memory), it is not clear how much of the memory to copy. If a buffer is being read from memory to an I/O device, the buffer cannot be reused until the reading is done. More precisely, if a buffer is being written into, then the related base register value would have to be changed so that I/O would write to the new location, then some portion of the buffer would be copied from the old to the new location. If a buffer is being read from memory to a peripheral device, it should be copied to the new location and then the base register value should be changed. Since the I/O proceeds asynchronously with respect to copying, it is not clear that the two will cooperate harmoniously. The conclusion is that buffers must stay wired down until I/O is complete, and also that a common memory map is not very helpful.

5. There is a simple solution: make the empty list contain a single element which has no data associated with it.



To insert DNEW after HEAD:

```
Pointer[FREE] := Pointer[HEAD];
Pointer[HEAD] := FREE;
Data[GREE] := DNEW;
```

6. (a) The main problem here is that a record which is created while within an inner block would disappear when the inner block is exited. However, a global reference to such a record would still point to the place in the stack where the record was. This is part of the famous "retention" vs. "deletion" argument which flares up now and then when stack oriented block structured languages are discussed.

(b) The major problem with interactive and dynamic compilation is that a simple change (e.g. insertion of a "begin-end" pair) may cause many blocks to change their lexical levels and may even necessitate a complete re-binding of variables. Furthermore, some changes in program text may necessitate reparsing (e.g. insertion of a **comment** to effectively nullify a statement). The problem is to minimize the amount of reparsing absolutely needed.

7. (a) $v_{n+1} = (1 + \lambda k)v_n$.

(b) $v_n = (1 + \lambda k)^n v_0$, $v_0 = y_0$. Using the hint, we have $1 + \lambda k = \exp(\lambda k - \lambda^2k^2/2 + \lambda^3k^3/3 - \dots)$ for $-1 < \lambda k \le 1$. Therefore $v_n = y_0 \exp(n\lambda k - n\lambda^2k^2/2 + O(k^3))$ for $-1 < \lambda k \le 1$.

(c) Since the differential equation has the solution $y(t) = y_0 e^{\lambda t}$, we obtain

$$|v_n - y(t)| = |y_0|\,|e^{\lambda t}|\,|1 - \exp(-\lambda^2 kt/2 + O(k^2)|$$

Since the exponent of e is $O(k)$ as $k \to 0$ the above relation implies that the iteration converges uniformly in t on every finite t interval, as $k \to 0$.

(d) We have $|y(t)| = |y_0|\,|e^{\lambda t}| = |y_0|\, e^{\lambda_1 t}$, which implies $|y(t)| \le |y_0|$ since $\lambda_1 \le 0$.

From part (b), $|v_n| \le |v_0|$ iff $|1 + \lambda k| \le 1$. Since $|1 + \lambda k|^2 = |1 + (\lambda_1 + i\lambda_2)k|^2 = 1 + 2\lambda_1 k + \lambda_1{}^2k^2 + \lambda_2{}^2k^2$, we find that $|1 + \lambda k| \le 1$ iff

$$k \le \frac{2|\lambda_1|}{\lambda_1{}^2 + \lambda_2{}^2}.$$

(e) If one wishes to compute an approximate solution to our problem over a long t-interval, it is essential that $|v_n| \le |v_0|$ and hence that the condition derived in part (d) be satisfied. If $\lambda_1 = 0$ then we must take $k = 0$ to satisfy the condition in (d), so Euler's method is useless in this case. Similarly if $|\lambda_2|^2$ is much larger than $|\lambda_1|$ then the condition of (d) implies that k must be very small and consequently that Euler's method is very inefficient for this problem. If a solution is only desired over a very short t-interval the restriction $|v_n| \le |v_0|$ is not so essential and the method has limited usefulness.

8. (a) We shall illustrate the algorithm by showing the steps: For each node on OPEN, we shall give the path from A to that node, the sum of the costs on the arcs of the path, and (in parentheses) the estimate of the distance from the OPEN node to B.

	OPEN	CLOSED
1)	A (17)	
2)	AC 2(16), AE 16(3), AG 10(11)	A(0)
3)	Expand C, because arc costs + estimate is minimum.	
	AE 16(3), AG 10(11), ACD 9(10), ACF 16(4)	A(0), C(2)

Note that ACA is also a possible path, but A is already on CLOSED, with a lower cost than that of the ACA path (i.e. 0 ≤ 4).

4) Since there is a tie between E and D, we will split it arbitrarily. We shall expand E.

AG 10(11), ACD 9(10), ACF 16(4), AEB 20(0) A(0), C(2), E(16)

Again, the path AEA is not expanded because A is on CLOSED with a lower cost. Path AED is not expanded because D is already on OPEN with a lower cost.

5) Expand D.

AG 10(11), ACF 16(4), AEB 20(0), ACDE 17(3) A(0), C(2), E(16), D(9)

The path ACDC is not expanded because C is on CLOSED with a lower cost.

6) Choose B, because its total cost is 20 (in a tie with F), and we always resolve ties in favor of a goal node. Hence A^* finds path AEB to be the "shortest" one.

(b) No, the shortest path is ACB. The reason that A^* fails is that the estimates are not all lower bounds on the distance from the node to the goal. In particular, the estimate at F is incorrect, and this causes the exploration of paths involving F to be delayed just long enough so that a sub-optimal path is found by A^*.

9. In recent years, there has been a shift toward "knowledge-based" AI programs. These programs incorporate domain-specific knowledge as well as knowledge about how to use the knowledge. This is in contrast to "knowledge-impoverished" programs (e.g. first-order logic theorem provers which use resolution).

The most striking example of knowledge-based systems are the new approaches to natural language understanding (Winograd, Schank). The knowledge-impoverished natural language systems (e.g. attempts at machine translation in the 1950's) were not successful. Other examples of knowledge-based systems are DENDRAL, theorem proving (Bledsoe), automatic programming (Elspas, Sussman), and scene interpretation.

10.(a) The algorithm we will present attempts iteratively to reduce n, keeping the constraint that the tape hold $*a^n b^{(2^n)}*$ for some n. This is done by deleting characters, replacing them with the character x, and ignoring all occurrences of x.

To reduce n by 1, we need to delete one a and half of the b's. This can be done by deleting the leftmost a and then deleting every other b (thus there is no need to count them). Failure occurs if we find a b among the a's, an a among the b's, or an odd number of b's at any step.

The final step should correspond to $n = 0$, i.e. the tape should contain $*b*$ (ignoring all x's). Therefore, if no a is seen at the start of some step, a check is made to see if there is exactly one b.

A flow chart of the TM is shown on the next page. For the sake of clarity, we separate the steps of reading, writing, and moving the head.

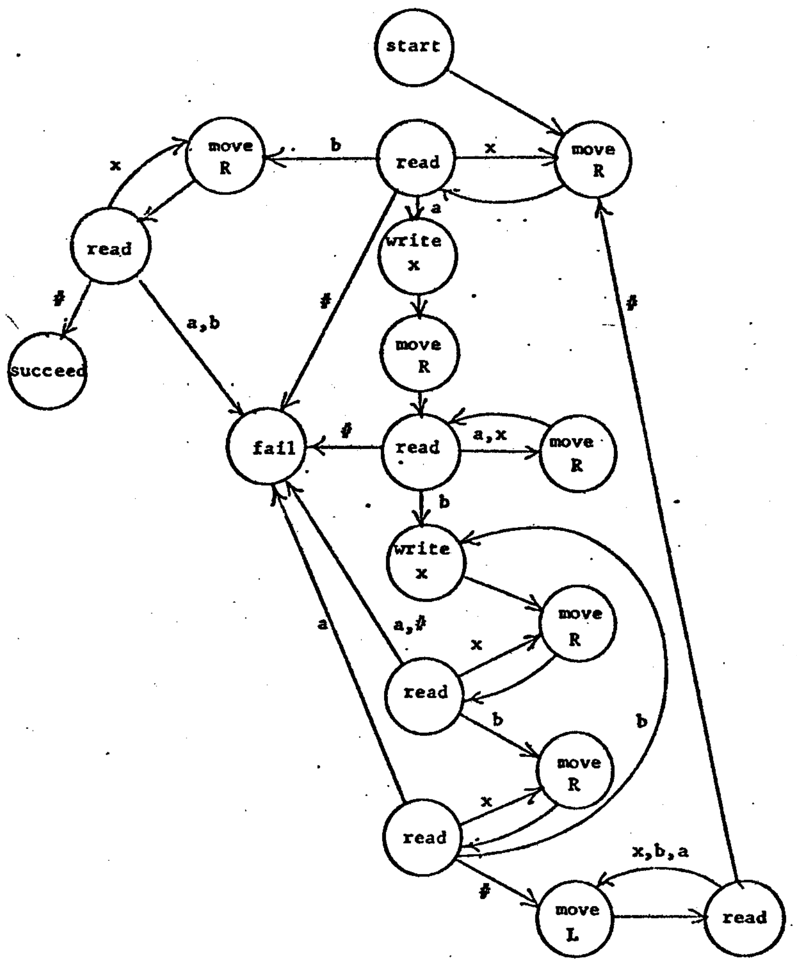
(b) No, because the language is not context-free (use the *uvwxy* theorem).

(c) Yes, the language is context-sensitive, because it can be recognized by a linear bounded automaton, namely, the one described above.

11. The formula is satisfiable, but not a tautology.

Since $(c \equiv d) \equiv \sim((\sim c \vee \sim d) \wedge (c \vee d))$ and $(a \supset b) \equiv (\sim b \supset \sim a)$, the first part of the conjunction may be written $[(a \supset b) \supset (c \equiv d)] \vee [(a \supset b) \supset \sim(c \equiv d)]$, which is a tautology.

For the second part of the conjunction, set b to *false*, then we have $\sim c \vee (d \equiv \sim a)$, which is clearly satisfiable, but not a tautology. In particular, the assignments a = *true*, b = *false*, c = *false*, d = *true* satisfy the original formula, but a = *true*, b = *false*, c = *true*, d = *true* falsify it.



WINTER 1975 COMPREHENSIVE EXAM

1. The binary encoding of the values which q takes are

0000, 0001, 0100, 1101, 1000, 1001, 1100, 0101, 0000, ...

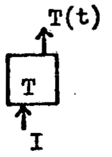
Only 8 of the possible 16 states arise. Note that the third bit is always zero.

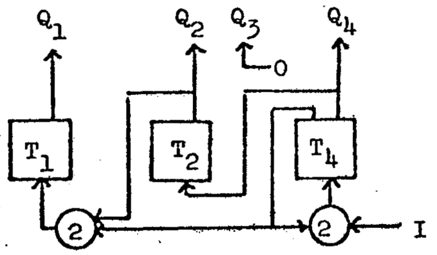
	Input			Output		
I	q_1	q_2	q_4	Q_1	Q_2	Q_4
1	0	0	0	0	0	1
1	0	0	1	0	1	0
1	0	1	0	1	1	1
1	1	1	1	1	0	0
1	1	0	0	1	0	1
1	1	0	1	1	1	0
1	1	1	0	0	1	1
1	0	1	1	0	0	0

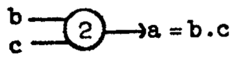
One can observe that Q_1 changes whenever $q_2q_4 = 10$ and that Q_2Q_4 form a mod 4 counter. Thus

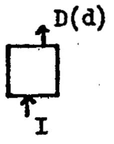
$$Q_1 = (q_1 \oplus q_2\bar{q}_4) , \quad Q_2 = (q_2 \oplus q_4) , \quad Q_4 = \bar{q}_4 I .$$

Either T flip-flop $T \leftarrow t \oplus I$

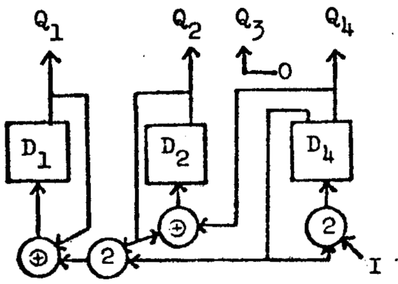


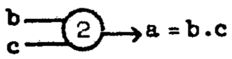


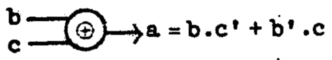
AND gate




or D flip flop $D \leftarrow d.I$



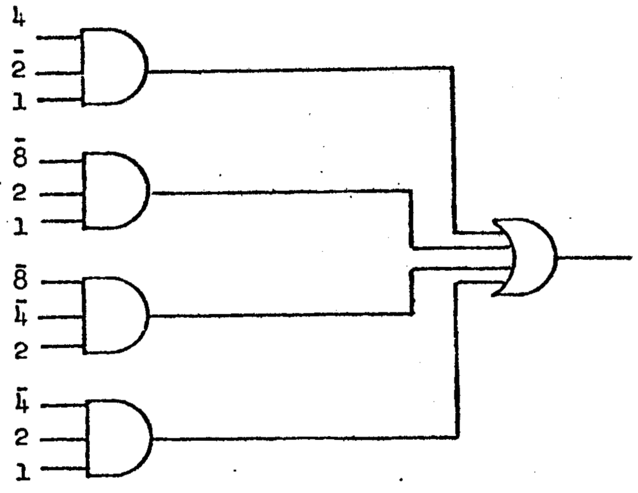
AND gate


XOR gate


2. (a)

		21			
		00	01	11	10
	00	0	0	1	1
84	01	0	1	1	0
	11	0	1	0	0
	10	0	0	1	0

sum of products: 4 AND + 1 OR
product of sums: 5 OR + 1 AND



(b) Pick one of the input leads to the OR gate. Then the circuit will always give a 1 output, and this is the only break of the type being considered which will cause this behavior. If a lead to an AND gate is chosen, there will be at least one non-prime input for which the output will be 1 rather than 0. However, it may not be possible to tell which gate is at fault.

3. Operator precedence

	a	b	c	d
a			<	=
b			<	=
c			<	>
d				

Simple precedence

	S	A	a	b	c	d
S						
A						=>
a		=			<	
b		=			<	
c		=			<	>
d						

The standard trick for eliminating precedence clashes is to introduce <*B*> ::= <*A*> and to redefine *S* to be <*S*> ::= *a*<*B*>*d* | *b*<*B*>*d* . We then have

	S	A	B	a	b	c	d
S							
A							>
B							=
a		<	=			<	
b		<	=			<	
c		=				<	>
d							

A much simpler solution is <*S*> ::= *a*<*A*> | *b*<*A*> , <*A*> ::= *cd* | *c*<*A*> .

4.

1	X = 4	W = 225
2	X = 4	W = 229
3	X = 7	W = 4
4	X = 11	W = 11
5	X = 4	W = 229
6	X = 7	W = 4
7	X = 11	W = 11
8	X = 11	W = 229
9	X = 7	W = 11
10	X = 18	W = 18
11	X = 18	W = 229

5. (a) The beginning and ending values of the PC (program counter) for each block, and the name of each block.

(b) Assume that code is generated in the same order as statements in the source file. Then it will be most convenient for the compiler to store the PC and name information immediately upon finding each **begin**. We put this information into a linear list, and also store the location of the list entry for each **begin** on a temporary stack. When the matching **end** is found, its address is put into the list entry and the **begin** is removed from the stack. The stack also enables the compiler to construct a FATHER link for each block upon finding the **begin**.

(c) Search through the linear list to find the smallest **end** address greater than the PC. Search back along the FATHER links until a **begin** address is found which is less than or equal to the PC. Then follow the FATHER links back to the beginning, printing the block names in reverse order.

In the above solution, we have used a linear list in order to conserve space. Note that timing considerations are generally not as important when dealing with errors. A tree structure could also be used, with LSON - RSIBLING linking for the blocks.

6. The chief advantage of both systems is that they free the programmer from worrying about reclaiming the storage he has used up.

(a) The disadvantage of garbage collection is that it takes time proportional to the total amount of space in use, since all of it must be examined. In an extended physical memory system, the garbage collector must access many pages on secondary store, and this is an inherently slow process. Secondly, garbage collection is a completely disruptive process. Computation cannot proceed while it is in progress. However, garbage collection is attractive from the viewpoint that there is almost no overhead until storage runs out.

(b) Reference counts require that every allocated data item include space for the reference count. In a system like LISP, where the allocation unit is a single word, the space overhead may be quite significant. In addition, every operation which affects the accessibility of a data item must update the reference count. The time overhead for this updating may be undesirable, especially when the referenced item is in secondary memory. Finally, the reference count scheme cannot detect self-referencing structures which are otherwise inaccessible.

7. In general, the problem is that we have to save the state of the machine when a breakpoint is encountered, and to restore it before we pass control back to the user. This saved state is exactly what the user will want to interrogate during his debugging session. We therefore need to save information such as the contents of the accumulators, any status bits, and the value of the program counter.

The broken instruction must be replaced with one which transfers control to the debugger. Since we need to know where we came from, different breakpoints must branch to different locations. Of course, we need a list which associates breakpoints with the corresponding broken instructions and their locations.

On return from a breakpoint, we will be executing the broken instruction from a different location in memory, which may cause some problems. For example, self-relative addressing will not work. Also, if the broken instruction is a jump to a subroutine, we must somehow make sure that the proper return address is saved.

8. Let γ be the solution of the equation $f(x) = 0$ and let $d_n = x_n - \gamma$ be the error in the approximation x_n. Then we have

$$d_{n+1} = d_n + \alpha\frac{f(x_n)}{f'(x_n)} + \beta\frac{f(x_n)^2 f''(x_n)}{f'(x_n)^3}.$$

Now

$$0 = f(\gamma) = f(x_n) + (\gamma - x_n)f'(x_n) + \frac{(\gamma-x_n)^2}{2}f''(x_n) + \frac{(\gamma-x_n)^3}{6}f'''(x_n) + \ldots$$

so

$$f(x_n) = d_n f'(x_n) - \frac{d_n^2}{2}f''(x_n) + \frac{d_n^3}{6}f'''(x_n) + \ldots .$$

Thus

$$\begin{aligned} d_{n+1} &= d_n + \alpha\left(d_n - \frac{d_n^2}{2}\frac{f''(x_n)}{f'(x_n)} + \frac{d_n^3}{6}\frac{f'''(x_n)}{f'(x_n)} + \ldots\right) + \beta\left(d_n - \frac{d_n^2}{2}\frac{f''(x_n)}{f'(x_n)} + \ldots\right)^2\frac{f''(x_n)}{f'(x_n)} \\ &= d_n(1+\alpha) + d_n^2\left(-\frac{\alpha}{2}+\beta\right)\frac{f''(x_n)}{f'(x_n)} + d_n^3\left(\frac{\alpha}{6}\frac{f'''(x_n)}{f'(x_n)} - \beta\left(\frac{f''(x_n)}{f'(x_n)}\right)^2\right) + \ldots . \end{aligned}$$

(a) To obtain the highest order of convergence we want to choose α and β such that $1 + \alpha = 0$ and $-\frac{\alpha}{2} + \beta = 0$. Thus $\alpha = -1$ and $\beta = -\frac{1}{2}$.

(b) Order of convergence is 3, i.e. cubic convergence.

9. (a) predictor: $y_8 = 2.12 + \frac{.1}{12}(\,23(2.12) - 16(1.77) + 5(1.68)\,) = 2.36$

corrector: $y_8 = -2.12 + 2(1.77) + \frac{.1}{4}(\,2.36 + 8(2.12) + 3(1.77)\,) = 2.04$

$y_8 = -2.12 + 2(1.77) + \frac{.1}{4}(\,2.04 + 8(2.12) + 3(1.77)\,) = 2.03$

(b) Solution decreases while true solution increases. We examine the behavior of the corrector for $y' = y$:

$$y_{n+1} = -y_n + 2y_n + \frac{h}{4}(y_{n+1} + 8y_n + 3y_{n-1})$$
$$(1 - \frac{h}{4})y_{n+1} + (1 - 2h)y_n - (2 + \frac{3h}{4})y_{n-1} = 0.$$

If $h = 0$, then we have $y_{n+1} + y_n - 2y_{n-1} = 0$. Characteristic equation: $z^2 + z - 2 = (z + 2)(z - 1) = 0$. The root $z = 1$ corresponds to the true solution e^x, but the root $z = -2$ corresponds to $(-2)^n$ which oscillates and grows rapidly in magnitude. Thus the solution is unstable.

10.(a) Prove: AT(baubles, Gump's) $\wedge$ AT(bangles, Macy's) $\wedge$ AT(beads, Sak's)
Initial state: AT(baubles, Universal Plastics) $\wedge$ AT(bangles, Universal Plastics) $\wedge$ AT(beads, Universal Plastics) $\wedge$ $truck(t)$

Axioms:

A1. $(\forall xyz)\ AT(x,y) \wedge AT(x,z) \supset y=z$
A2. $(\forall xyz)\ AT(x,y) \wedge AT(z,y) \wedge LOAD(x) \supset IN(x,z)$
A3. $(\forall xy)\ truck(x) \wedge drive(x,y) \supset AT(x,y)$
A4. $(\forall xyz)\ truck(x) \wedge AT(x,z) \wedge IN(y,x) \wedge UNLOAD(y) \supset AT(y,z)$

(b)
```
BEGIN
   STRING(20) SUPPLIER, RETAILSTORE, ARTICLE;
   READ (SUPPLIER);
   WHILE SUPPLIER ¬= "DONE" DO BEGIN
      READON (RETAILSTORE, ARTICLE);
      DRIVETO (SUPPLIER);
      LOAD (ARTICLE);
      DRIVETO (RETAILSTORE);
      UNLOAD (ARTICLE);
      READ (SUPPLIER)
   END;
   WRITE ("DONE")
END.

"UNIVERSALPLASTICS" "GUMP'S" "BAUBLES"
"UNIVERSALPLASTICS" "MACY'S" "BANGLES"
"UNIVERSALPLASTICS" "SAK'S"  "BEADS"
"DONE"
```

(c) A descriptive representation seems appropriate when: the knowledge of the task is highly modular; the knowledge base needs to be changed often; the task can be described with assertions about the world; or the problem can be set up as an assertion to prove.

A procedural representation seems appropriate when: the sequence of actions is important; the relationships between objects are time-dependent; the primitive actions are fixed; or the method of solution is known and is unlikely to change.

11. The combinatorial explosion is the proliferation of nodes in a search space resulting from the large number of possible combinations of primitive elements in a program. AI programs typically involve search through such a space. The evaluation of any one node in a possible solution path usually depends on the values of all the successor nodes. Thus, the number of nodes to be examined grows exponentially with the depth of the search tree.

Search can be controlled using one of the following techniques:

(1) Heuristics for pruning branches of the search tree: knowledge of task domain; branch and bound techniques (e.g. alpha-beta pruning); strong evaluation function with threshold (expand only the best nodes).

(2) Heuristics for reordering branches of the search tree: expand best nodes first; examine "easy" branches first at OR-nodes and "hard" branches first at AND-nodes in an AND-OR tree.

(3) Problem reduction: establish subproblems which are easier to solve than the original problem.
Planning: establish a series of intermediate goals to guide the problem solver through the search space.

(4) Learning: save the results of expanding commonly encountered nodes to avoid re-searching below these nodes. Adjust the evaluation function as a result of previous experience.

12.(a) Given two finite automata with state sets K_1 and K_2, we construct a non-deterministic automaton with the Cartesian product $K_1 \times K_2$ as its set of states. Only one state component is changed in each transition of the new automaton. These transitions shall correspond in the obvious way to the state transitions of the original automata. The initial and final states of the new automaton are the pairs of initial and final states of the original automata. This new automaton accepts exactly the merge of the languages accepted by the original two automata. (The above construction could also be given in terms of regular grammars.)

(b) Similar to part (a), using a pushdown automaton and a finite automaton.

(c) Given context-sensitive grammars for L_1 and L_2, we construct a new context-sensitive grammar which contains the union of the sets of production rules for the original grammars. (We assume that the two grammars have distinct symbols.) We introduce a new start symbol S and add the production $S \to S_1S_2$, where S_1 and S_2 are the start symbols of the original grammars. In addition, we add all productions of the form $a_ib_j \to b_ja_i$, where a_i and b_j are terminal symbols in the grammars for L_1 and L_2 respectively. The resulting context-sensitive grammar generates exactly the merge of L_1 and L_2.

13. Assume the grammar contains no useless nonterminals. (All terminals except a and b can also be deleted.) For a nonterminal A, define $n(A)$ to be the set of all i such that A can generate a word with i more a's than b's. The sets $n(A)$ can be determined by iterating the following process:

> For all productions $A \to \alpha$ such that α contains only terminals or nonterminals B_i for which one element of $n(B_i)$ is already known, compute values for $n(A)$.

If any $n(A)$ is found to have more than one value, then the grammar cannot have the desired property. Otherwise, the process terminates when no new values for any $n(A)$ are obtained. The value of $n(S)$ then gives the answer.

SPRING 1975 COMPREHENSIVE EXAM

Systems

1. No. A deadlock can occur only if all of the tape units are reserved and all processes are waiting indefinitely for more units to become available. If all four tape units are reserved, then one of the processes has all the tape units it needs and will be able to finish. Its two units then become available, and the other processes can finish.

2. (a) In ALGOL, variable bindings are *static* and are semantically determined at compile time from the block structure and scope rules. In LISP, bindings are *dynamic* and depend on the order in which routines are called. A LISP variable is bound to its most recent definition on a value stack.

(b) We should try to place the new cell on the same page as one of the substructures (CDR or CAR), in order to reduce paging during traversal of the structure. Note this implies that each page should have a free storage list associated with it.

3. (a) FIFO: 9 faults with 3 page frames; 10 faults with 4 page frames (FIFO anomaly).
LRU: 10 faults with 3 page frames; 8 faults with 4 page frames.

(b) Associate a reference bit with each page which is set every time the page is referenced. When we need to select a page for removal, we advance a pointer circularly through the list of in-core pages and select the first page whose reference bit is clear. Pages which are skipped over have their bits cleared.

(c) On a system where there are many commonly used programs (e.g. compilers and editors), it is advantageous to have users share the common pages. Such sharing reduces both the overhead of moving programs to and from the drums and the required amount of drum storage. Data pages may also be shared by programs, and this is often the most efficient way to transfer data from producers to consumers.

4. (a) The problem is to reduce the amount of storage required for the precedence matrix. Methods for doing this include: sparse matrix techniques, precedence functions (if they exist), and pointers to identical rows or columns.

(b) Most compilers use a simpler technique for the scanner than for the context-free parser. Some compilers use top-down methods for good error recovery but switch to operator precedence for expressions (to avoid backup). In FORTRAN, a special parser might be used for FORMAT statements.

(c) No. From the matrix we conclude that $g(S_1) = f(S_1) > g(S_2) = f(S_2) > g(S_1)$, which is unsatisfiable in precedence functions.

(d) Typed languages: ALGOL, SAIL, FORTRAN, PASCAL, SIMULA. Advantages: many more errors detectable at compile time; compiler knows more about what a program is doing and can perform optimizations; system can provide type conversion automatically. Disadvantages: less flexible language; compiler is harder to write.

Untyped languages: BCPL, LISP, APL, EULER, GEDANKEN. Advantages: more flexibility possible in programming resulting sometimes in greater efficiency; compiler is easier to write because of the uniform treatment of addresses. Disadvantages: programs are harder to debug (some bugs result in subtle smashing of code or data).

4. The test-and-set instruction provides a primitive way to synchronize processes. A process (1) waits for a flag to be clear, (2) sets the flag, (3) enters its critical region and (4) clears the flag when it is done. The disadvantage of this technique is that it involves busy waiting.

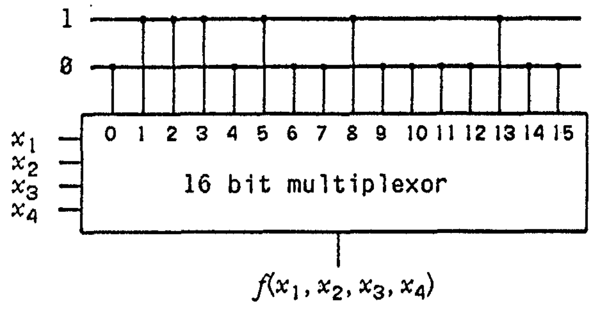
A better primitive method uses semaphores (Dijkstra), which allow processes to be queued while they are waiting and restarted when the shared resource (e.g. critical region) becomes available to them. The basic operations in the program are P (wait) and V (signal).

Hardware

1. (a) Sum of products: $f = x_1\bar{x}_2\bar{x}_3 + \bar{x}_3x_4 + \bar{x}_1\bar{x}_2x_3$

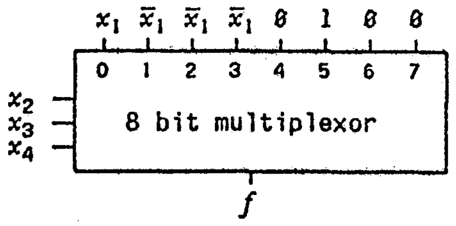
(b) Product of sums: $f = (x_1 + x_3 + x_4)(\bar{x}_1 + \bar{x}_3)(\bar{x}_2 + x_4)$

(c)

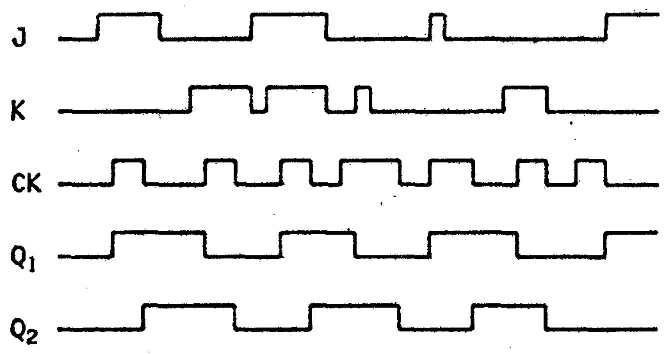


(d)

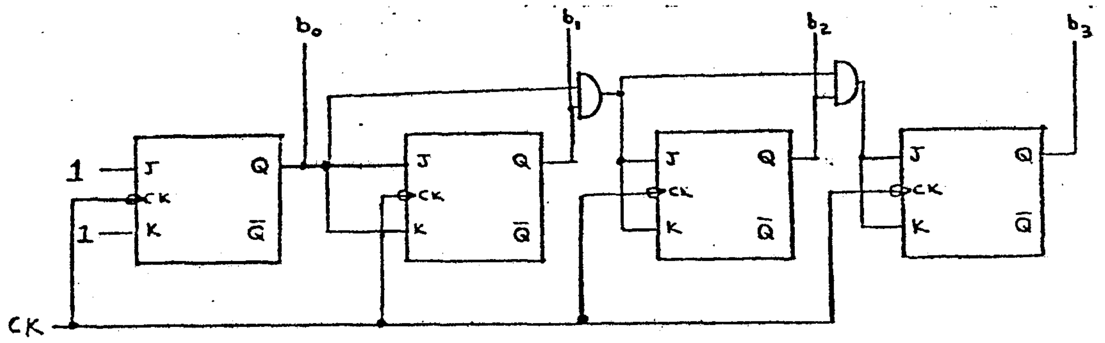
	$x_2x_3x_4$ 0	1	2	3	4	5	6	7
x_1 0	0	1	1	1	0	1	0	0
1	1	0	0	0	0	1	0	0



2.



3.



4. (a) Multiprocessing means that there are several processors capable of operating simultaneously on a system. For example, swapping drums and other I/O devices usually have their own processors which run fairly independently of the CPU. Some systems may even have several CPUs.

(b) Memory interleaving means that successive addresses refer to words which are physically located in different memory banks. Since reading a word from memory is usually destructive, there is a delay time for that bank during which local hardware restores the contents of the word. The effect of this delay can be minimized if the next word fetched is from a different bank.

Another use for interleaving arises on memories having multiple ports, say for the CPU and swapping drum. Usually the I/O ports have higher priority. If memory were not interleaved, a drum could effectively lock the CPU out of a bank of memory altogether during a rapid block transfer. Interleaving reduces this "*interference*".

(c) Wide buses permit the parallel transfer of more information at the same time. If each access to memory fetches two words at once, the memory bandwidth is potentially doubled, and a CPU may need only about half as many bus cycles to carry out the instructions.

(d) Associative memory means memory addressed by part of its content. This speeds up the search for an entry in a table because of the parallel operation of the hardware which carries out the association. Associative memory is often used to speed up virtual-to-physical address translation or in implementing a cache memory.

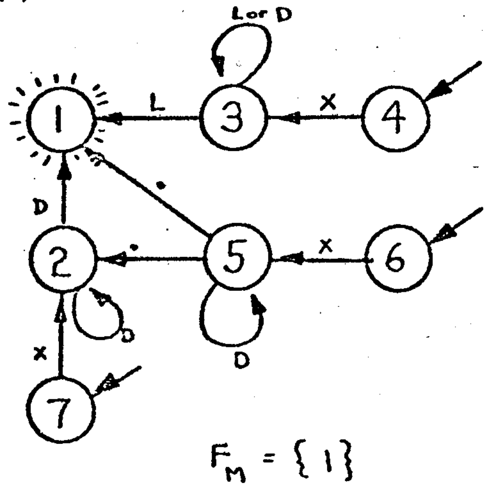
Theory of Computation

1. (a) Assume that there is an effective enumeration $\{ T_1, T_2, T_3, \ldots \}$ of all total recursive functions. Define a new function F as follows: $F(x) = T_x(x) + 1$ for all x. Then F is a total recursive function, but it is not in the enumeration because it differs from each T_i at the argument i. This contradiction shows that there can be no effective enumeration of all total recursive functions.

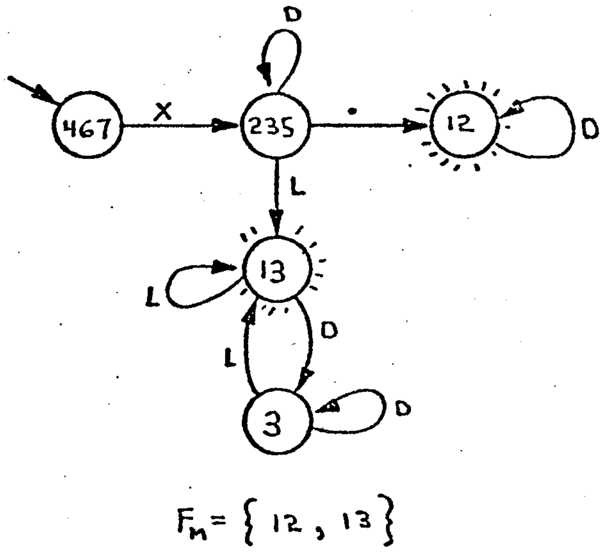
(b) Every context sensitive language is recursive. If every recursive language were context sensitive, then we would have a contradiction of part (a). Alternatively, let $\{ C_1, C_2, C_3, \ldots \}$ be an effective enumeration of all context sensitive grammars, over a fixed (countable) alphabet. Define $S = \{ a^i \mid a^i \text{ is not in } L(C_i) \}$. Then S is recursive but it cannot be generated by any context sensitive grammar in the above enumeration.

2. (a) `D*.D*x`

(b)



(c)



(d) The finite machine of part (a) is essentially that part of a scanner which recognizes the numeric constant and variable tokens in a language. Final state 7 corresponds to an integer constant, state 6 to a floating point constant, and state 4 to a variable name. The reverse scanner might be used if, for some reason, it is desired to read the source backwards. Of course an actual scanner should be based upon a *deterministic* machine.

3. (a) The language contains no strings of even length. This is easily seen by noting that every production increases the length of the string by zero or two. Since we start with a single symbol (namely S), we cannot generate any strings of even length. (In fact, all terminal strings have length $4k+1$ for some $k \geq 0$.)

(b) First note that the shortest string derivable from B has length 3, and that c occurs only in the production $A \to SBc$. Hence the desired strings must be of the form $aBcbB$ where each B is replaced by either aab or baa. There are a total of four such strings.

Algorithms and Data Structures

1.
```
procedure RK α
Begin of program:
  β
loop γ
check stepsize: RK1ST(x, y, H, x2, y2); RK1ST(x2, y2, H, x3, y3);
  for k:=1 step 1 until n do
    if comp(y1[k], y3[k], eta) > eps then begin
      δ
      ζ
      go to check stepsize
    end;
  x:=x3;
while not out:
  ε
repeat
  η
end RK
```

The remaining go to cannot be eliminated without introducing extra computation. But it isn't such a harmful one, and one would expect an example at the end of a language definition to illustrate most of the features of that language.

2. F L M A J R T B P D H I Q E K C G N S U O.

3. (a) A sequential list containing entries $(i, j, A[i,j])$ for nonzero $A[i,j]$.
(b) An orthogonal list arrangement (see Knuth section 2.2.6).
(c) Same as (b), or possibly a hash table of size $M \approx 750$ which finds $A[i,j]$ given the key $<i,j>$. If the hash function has the form $(ai + bj) \bmod M$, it will be possible to perform operations (3) and (4) without too much delay.
(d) 10000 consecutive locations with $A[i,j]$ in locations $L + 100*i + j$ for some L.

Numerical Analysis

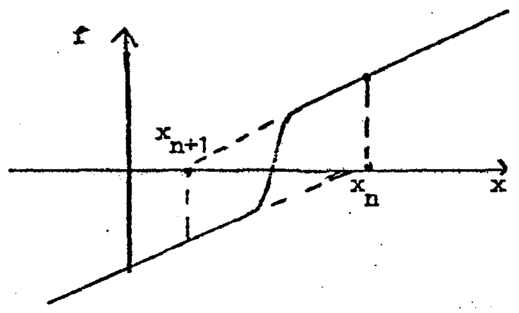
1. Partial pivoting could increase the bandwidth from $2m+1$ to $4m+1$ thus significantly increasing the amount of storage as well as the number of arithmetic operations required to carry out the elimination.

2. (a) No. For example, the floating point number system is not associative with respect to +. To show this, consider the special case of two digit base 10 arithmetic. Then $fl(10 + fl(-10 + 0.1)) = 0$ but $fl(fl(10 - 10) + 0.1) = 0.1$.
(b) Forward error analysis must be carried out in floating point arithmetic which leads to problems as in part (a). Inverse error analysis can be carried out in real arithmetic, which is much easier, and also usually produces simpler bounds.

3. (a) We use the l_∞ norm, $\|\underset{\sim}{x}\|_\infty = \max_{1\le i\le n} |x_i|$, where $\underset{\sim}{x} = (x_1, \ldots, x_n)$. Writing our equation in the form $A^{-1}\underset{\sim}{b} = \underset{\sim}{x}$, where $\underset{\sim}{x} = (1, 1/2, 1/4, \ldots, 2^{2-n}, 2^{2-n})$ and $\underset{\sim}{b} = (0, 0, \ldots, 0, 2^{2-n})$, we have $\|A^{-1}\|_\infty 2^{2-n} = \|A^{-1}\|_\infty \|\underset{\sim}{b}\|_\infty \ge \|A^{-1}\underset{\sim}{b}\|_\infty = \|\underset{\sim}{x}\|_\infty = 1$ so that $\|A^{-1}\|_\infty \ge 2^{n-2}$. Now $\|A\|_\infty = 1$ so $\mathrm{cond}(A) = \|A\|_\infty \|A^{-1}\|_\infty \ge 2^{n-2}$. Thus for large n, the matrix A is quite ill-conditioned.

(b) Since $\det(A) = 1$, A is clearly nonsingular and the diagonal elements are certainly not small. However, A is "nearly singular" for large n since its condition number is then very large. Methods based upon Gaussian elimination must rely upon the size of the diagonal elements, once transformed to upper diagonal form, to determine rank. If the lower r rows have diagonals which are "essentially zero" we would conclude that the rank is $n-r$. Our example shows that there are nearly singular matrices (i.e. which are only a small perturbation away from being singular) which have large diagonal elements. Therefore, this approach will not be suitable for floating point computation.

4. (a)



The iteration clearly will not converge and from the picture it is reasonable to conclude that $x_n = x_{n+2} = \ldots = x_{n+2m} = \ldots$ and $x_{n+1} = x_{n+3} = \ldots = x_{n+2m+1} = \ldots$ for all $m \geq 0$.

(b) Let $\tilde{p}(x) = \tilde{a}_n x^n + \sum_{0 \leq i \leq n-1} a_i x^i$, where $\tilde{a}_n = a_n + \Delta a_n$, and suppose that $\tilde{r} = r + \Delta r$ is the root of $\tilde{p}$ corresponding to the root r of the original polynomial p. Then $0 = \tilde{p}(\tilde{r}) = \Delta a_n \tilde{r}^n + p(\tilde{r})$. Using a Taylor expansion to terms $O(\Delta r^2)$, we obtain $p(\tilde{r}) \approx p(r) + (\Delta r)p'(r) = (\Delta r)p'(r)$. Thus to terms $O(\Delta r^2 + \Delta r \Delta a_n)$, we have $(\Delta r)p'(r) \approx -\Delta a_n(r + \Delta r)^n \approx -\Delta a_n r^n$, or,

$$\frac{\Delta r}{r} \approx -\frac{a_n r^{n-1}}{p'(r)} \frac{\Delta a_n}{a_n} .$$

(c) The observed convergence is linear. This may have happened because the iterates are not close enough to the root or they are converging to a multiple root.

Artificial Intelligence

1. (a) (The following is an example of an acceptable answer.)

"English language understanding"

A program which "understands" English must have some way of representing the meaning of sentences. This meaning needs to be incorporated into the program's *model* of the world. For example, in Winograd's SHRDLU a model of the "blocks world" is kept and updated by the program. In Colby's PARRY, the variables of the program model the emotions of the paranoid. In Schank's work, a semantic net carries the meaning in a data structure.

The importance of the model derives in part from the necessity of the program to "reason" about its world. For example SHRDLU knows about the transitivity of "on top of" and also knows what it has done. The MYCIN program knows "why" it made each step of a diagnosis. The models in these cases include the prior activities of the program. In general A.I. developments, reasoning has been based on theorem proving with new techniques involving the *resolution principle*. Older general techniques include the *means-ends analysis* of GPS. Today's more successful programs use more domain specific techniques. SHRDLU reasons about its model using the goal directed computations of MICROPLANNER. In a sense, it knows "what follows from what" although it does not prove theorems in general. Like several of the new A.I languages, MICROPLANNER does the bookkeeping associated with backtracking. This tool allows a program to gracefully abandon a fruitless subgoal.

Sometimes there are several sources of knowledge for a program to draw upon or there is knowledge to be used only in particular contexts. The first case is called the *multiple expert* problem and is exemplified by the HEARSAY program. This program subjugates the various experts of an English-speaking chess playing program (e.g. sound expert, syntax expert, chess expert) to a *master* which arbitrates between them. The second case is handled by a new software concept – the pattern-directed or preconditioned interrupt known as a *demon*. If the entire calculation is driven by patterns of the form <precondition> → <action>, then the system is called a *production system*. These various concepts can be viewed either as fancy developments in program organization, or as variations on the procedural representation of knowledge.

In summary, English understanding is a rich enough area to have connections with all the major areas of A.I. research, including heuristic search, knowledge representation, and new program organizations.

(b) We must first determine what objects are to be recognized by the image understanding system. Then the main A.I. design problem is to decide how the objects will be described to the program, as well as how the program will exploit these descriptions. In our case, the satellite has to be able to recognize objects such as rivers, river bends, deltas, mountains, islands, etc.

2. (a) For any nontrivial task involving search, the search space is very large. Rules and procedures called heuristics are therefore employed to steer or limit the search. Often these rules are highly task specific and depend on informal knowledge about the problem.

(b) Means-ends analysis is a problem solving method which is applicable when the initial and goal states are explicitly defined and when a path to the goal can be achieved by the incremental application of a given set of state operators. The method is to iteratively select an operator to transform the current state into another state which is closer to the goal state, until the goal is reached.

(c) Hill climbing is a technique for finding the extrema of a scalar function of n variables. This is useful for improving the value of a performance evaluation function of n variables. Unfortunately, the technique generally cannot distinguish between local and global extrema; thus it may give a result which is far from optimal.

(d) A demon is a conditional interrupt which operates autonomously in a program. It activates whenever the "condition" occurs in the program. Demons are useful for representing a process which can be conceptualized in terms of a few independent pattern-invoked actions or subprocesses in addition to the main computation.

(e) Production systems are a way of representing a program in terms of <situation, action> pairs which are called productions. The basic idea is that the action (subprocess or event) takes place when the situation (pattern matching conditional expression) becomes true. This technique is useful when the task consists of many independent pattern-invoked subprocesses or when it is often desired to add new subprocesses or to change the order of operations.

3. (1) True. Each application of an operator can be viewed as the reduction of the problem to a simpler subproblem. The special case of state space methods remains useful as a separate concept because it is simpler and because some problems are more easily conceptualized as moves in a state space rather than as problem reduction steps.

(2) False. The alpha-beta technique always achieves exactly the same answer as minimax.

(3) The detailed knowledge of task domains has been found to be both useful and necessary for achieving high levels of performance (i.e. achieving significant search reduction). The attention of A.I. researchers has turned to acquiring and utilizing such detailed knowledge for specific problem domains. Another drawback of general problem solving methods was that they tended to force awkward, non-natural representations of problems and domain-dependent knowledge.

WINTER 1976 COMPREHENSIVE EXAM

Systems

1. Make the symbol table entry for the undefined variable be the head of a linked list of all forward references to the variable. When the variable is defined, the list can be traversed to fix up the forward references. To use this method, we must place some restrictions on the arithmetic which can be done on forward references. Also, if the object code is too large to "fix up" while in memory during the assembly phase, then the loader can be used to perform this task, using the same method.

2. The main job of an operating system is to allocate the resources of the system to competing tasks (jobs, processes, or users). The operating system allocates memory and CPU resources and manages a file on behalf of the running processes. Operating systems may also offer utility services through system calls. These frequently deal with I/O, interprocess communication, and file manipulation.

3. An interrupt is an externally generated signal which indicates the completion of an I/O operation or the occurrence of some condition which the operating system should know about. To resolve conflicting interrupts, there is usually some method for establishing priority. At predetermined stages in its normal instruction cycle, the CPU hardware will test for an interrupt condition and execute an instruction in a specified location associated with that particular interrupt. Usually this instruction does a subroutine branch or exchanges the current PSW for a new one to reach the actual interrupt service routine. The service routine might save all the general registers and other information in order to avoid any damage to the state of the interrupted program.

4. Linkers are primarily concerned with resolving external references between object modules. Loaders are generally used to perform relocation of object modules and reformatting of them into absolute load modules. Some loaders can perform linking chores as well (e.g. automatically loading library modules to satisfy unresolved external references).

5. Re-entrant routines are capable of being executed by more than one process concurrently. The routine must be pure procedure, that is, the calling routine must supply all local storage. Recursive code is typically re-entrant and also "calls itself". To maintain control, new local variables must be created for each recursive entry into the routine.

6. The fundamental difference between multiprogramming and multiprocessing is simultaneity versus concurrency. Both systems must deal with the interactions among several processes, and the handling of resource management, interprocess communication, and process management is similar. However, in a multiprocessing environment, there may be the possibility of two processes executing the operating system simultaneously. Cooperation among processes cannot make single-CPU assumptions which might be valid in a multiprogramming environment.

7. Paged systems allocate fixed length pages on secondary storage and identical length page frames in the main store. Pages are moved to and from secondary storage as needed (e.g. demand paging or pre-paging). A large address space can be offered to each process, even if the real main store is smaller than the maximum virtual address space. Segmentation is a method of creating virtual storage composed of variable length segments. To manage the mapping from virtual to real memory, segment tables are maintained with pointers to the base of each segment and the extent of each segment. Symbolic segmentation allows a user to refer to any elements of a subroutine library without necessarily requiring that a particular part of the address space be allocated to the library routine. In a paged space, the library routines must be located at a fixed place in the virtual address space in order to execute. In effect, a segmented space creates a very large number of linear

virtual spaces, each with a name. The paged space creates only one such virtual linear address space.

8. One popular way is the use of "P" and "V" operations on objects called semaphores (invented by E. W. Dijkstra). Processes wishing to use a protected resource signal "P" to the associated semaphores. If the resource is in use, the process is queued by the operating system and put to sleep. When a process finishes the critical section, it signals "V" to the semaphore which causes the operating system to reschedule the process which is currently at the head of the queue, waiting on that semaphore.

An alternative is a "busy wait" scheme using "lock" and "unlock" primitive operations. Here, a process attempts to successfully lock the critical section. Locking fails if the critical section is already locked. The process loops, repeatedly attempting to lock until successful. When done with the critical section, a process unlocks it to allow other processes to proceed.

9. An interpreter accepts source input in some well-defined form and executes it (i.e. it interprets the semantics and carries out the associated commands). A compiler, on the other hand, accepts source input and transforms it into something which can be interpreted later (e.g. relocatable or absolute object module). Incremental compilers often transform source language into an intermediate interpretive form which can be interpreted on command from an interacting user.

Numerical Analysis

1. (a) The algorithm describes pairwise addition. Thus $s(M, 1) = \Sigma_{1 \le j \le N}\, a_j$.

(b) We have $t(1, k) = (t(0, k) + t(0, k+2^{M-1}))(1 + \epsilon_{1,k}) = (a_k + a_{k+2^{M-1}})(1 + \epsilon_{1,k})$. Then by induction $t(M, 1) = \Sigma_{1 \le j \le N}\, a_j(1 + \eta_j)$, where $(1 + \eta_j) = \Pi_{1 \le p \le M}\,(1 + \epsilon_{p,j_p})$. Hence $|\eta_j| \le M\epsilon + O(\epsilon^2)$.

(c) From (a) and (b), $s(M, 1) - t(M, 1) = -\Sigma_{1 \le j \le N}\, a_j\eta_j$, so $|s(M, 1) - t(M, 1)| \le \Sigma_{1 \le j \le N}\, a_j|\eta_j| \le \max_j |\eta_j|\, \Sigma_{1 \le j \le N}\, a_j$. Thus

$$\frac{|s(M, 1) - t(M, 1)|}{|s(M, 1)|} \le M\epsilon + O(\epsilon^2).$$

(d) Suppose we compute the sum in the "natural way" as follows: $w(0) = 0$, $w(j) = fl(a_j + w(j-1))$ for $j = 1, 2, \ldots, N$. Then $w(N) = \Sigma_{1 \le j \le N}\, a_j(1 + \zeta_j)$, where $|\zeta_1| \le (N-1)\epsilon + O(\epsilon^2)$ and $|\zeta_j| \le (N-j+1)\epsilon + O(\epsilon^2)$, $j = 2, 3, \ldots, N$. Hence $|\Sigma_{1 \le j \le N}\, a_j - w(N)| \le \Sigma_{1 \le j \le N}\, a_j|\zeta_j| \le \max_{1 \le j \le N} |\zeta_j| \cdot \Sigma_{1 \le j \le N}\, a_j$ and so

$$\frac{|\Sigma_{1 \le j \le N}\, a_j - w(N)|}{|\Sigma_{1 \le j \le N}\, a_j|} \le (N-1)\epsilon + O(\epsilon^2).$$

We see that the maximum relative error for pairwise addition is much smaller than for addition performed in the usual manner.

2. (a) Since $\underset{\sim}{z}^* = \underset{\sim}{h}(\underset{\sim}{z}^*)$, we have $\|\underset{\sim}{z}^{(k+1)} - \underset{\sim}{z}^*\| = \|\underset{\sim}{h}(\underset{\sim}{z}^{(k)}) - \underset{\sim}{h}(\underset{\sim}{z}^*)\| \le L\,\|\underset{\sim}{z}^{(k)} - \underset{\sim}{z}^*\|$ and hence $\|\underset{\sim}{z}^{(k)} - \underset{\sim}{z}^*\| \le L^k\,\|\underset{\sim}{z}^{(0)} - \underset{\sim}{z}^*\|$. Since $0 < L < 1$, $\|\underset{\sim}{z}^{(k)} - \underset{\sim}{z}^*\| \to 0$ as $k \to \infty$.

(b) We have $\|\underset{\sim}{z}^{(k+1)} - \underset{\sim}{z}^{(k)}\| = \|\underset{\sim}{h}(\underset{\sim}{z}^{(k)}) - \underset{\sim}{h}(\underset{\sim}{z}^{(k-1)}\| \le L\,\|\underset{\sim}{z}^{(k)} - \underset{\sim}{z}^{(k-1)}\|$ and hence $\|\underset{\sim}{z}^{(k+1)} - \underset{\sim}{z}^{(k)}\| \le L^k\,\|\underset{\sim}{z}^{(1)} - \underset{\sim}{z}^{(0)}\|$. Writing $\underset{\sim}{z}^{(m)} - \underset{\sim}{z}^{(n)} = (\underset{\sim}{z}^{(m)} - \underset{\sim}{z}^{(m-1)}) + (\underset{\sim}{z}^{(m-1)} - \underset{\sim}{z}^{(m-2)}) + \ldots + (\underset{\sim}{z}^{(n+1)} - \underset{\sim}{z}^{(n)})$, we obtain $\|\underset{\sim}{z}^{(m)} - \underset{\sim}{z}^{(n)}\| \le (L^{m-1} + L^{m-2} + \ldots + L^n)\,\|\underset{\sim}{z}^{(1)} - \underset{\sim}{z}^{(0)}\| \le L^n(1-L)^{-1}\,\|\underset{\sim}{z}^{(1)} - \underset{\sim}{z}^{(0)}\|$. Letting $m \to \infty$, we have

$$\|\underset{\sim}{z}^* - \underset{\sim}{z}^{(n)}\| \le \frac{L^n}{1-L}\,\|\underset{\sim}{z}^{(1)} - \underset{\sim}{z}^{(0)}\|.$$

(c) Write the system in the form

$$\begin{pmatrix} 1 & a \\ b & 1 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}.$$

The determinant of the matrix is $1 - ab \neq 0$, since $|a||b| < 1$. Therefore the system has a solution.

(d) We have $\underset{\sim}{z}^{(k+1)} = \underset{\sim}{h}(\underset{\sim}{z}^{(k)})$, where

$$\underset{\sim}{h}(\underset{\sim}{z}) = \underset{\sim}{c} - \begin{pmatrix} 0 & a \\ b & 0 \end{pmatrix} \underset{\sim}{z}.$$

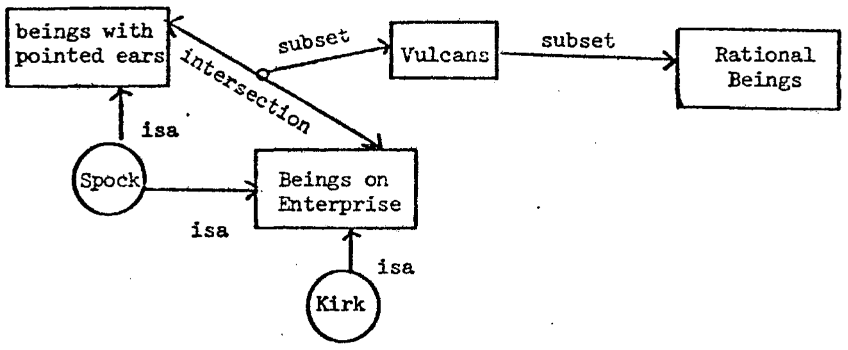
Now $\|\underset{\sim}{h}(\underset{\sim}{z}) - \underset{\sim}{h}(\underset{\sim}{\zeta})\| \leq \max(|a|, |b|)\, \|\underset{\sim}{z} - \underset{\sim}{\zeta}\|$. Thus $L = \max(|a|, |b|) < 1$ and hence $\underset{\sim}{z}^{(k)} \to \underset{\sim}{z}^*$ as $k \to \infty$.

(e) Let $e^{(k)} = x^{(k)} - x^*$ and $f^{(k)} = y^{(k)} - y^*$. Then $e^{(k+1)} = -af^{(k)}$ and $f^{(k+1)} = -be^{(k+1)}$. Hence $f^{(k+1)} = (ab)f^{(k)} = \ldots = (ab)^{k+1}f^{(0)} \to 0$ as $k \to \infty$. A similar analysis shows that $e^{(k)} \to 0$ as $k \to \infty$.

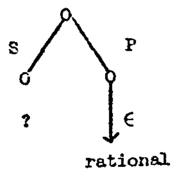
Artificial Intelligence

1. (a) Define the following predicates: $V(x) \equiv x$ is a Vulcan; $E(x) \equiv x$ is aboard the Enterprise; $P(x) \equiv x$ has pointed ears; and $R(x) \equiv x$ is rational. Then the given conditions are $\forall x\ (P(x) \supset (E(x) \supset V(x)))$, $\forall y\ (V(y) \supset R(y))$, $E(\text{Spock})$, $E(\text{Kirk})$, and $P(\text{Spock})$; and the proposition to be proved is $\exists z\ R(x)$.

(b)



An alternative solution (HAM-like) is shown at the top of the next page. Here the query can be represented as

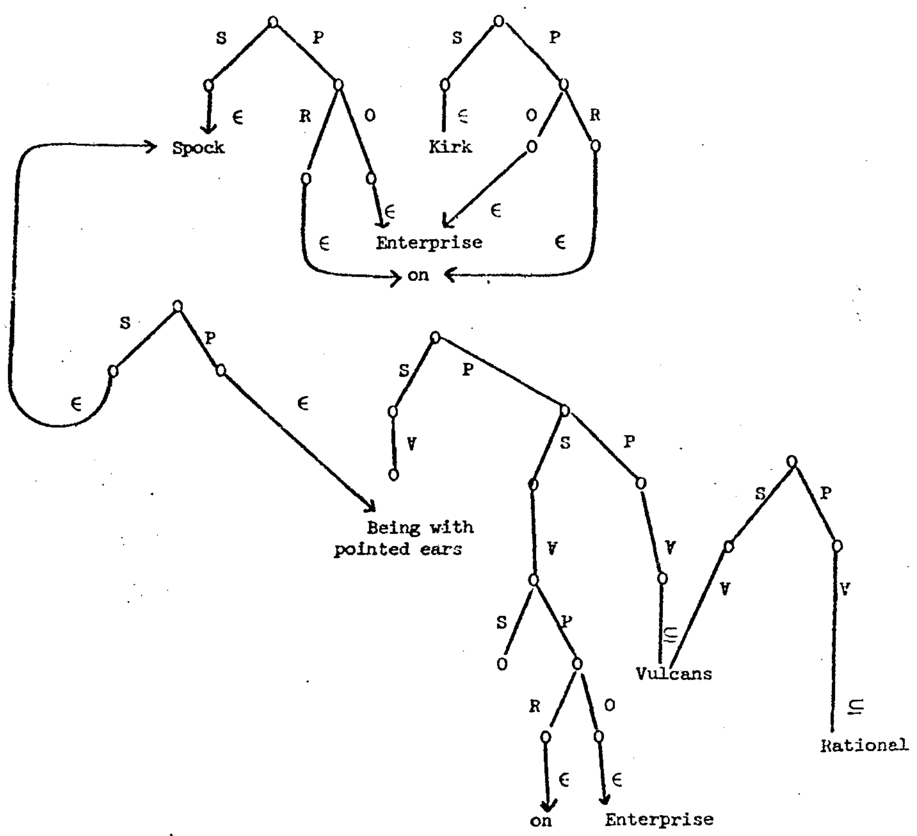


which means, "who is rational?" (actually, "who belongs to the class of rational beings?").

2. (a) We describe a solution in a hypothetical language with the following conventions. When ←*x* occurs in a pattern, this binds *x*. Then $*x* will be the binding, which must exist if used.

```
th_consequent {(←x is rational)
          [(goal (←x is Vulcan))]}
th_consequent {(←x is Vulcan)
          [th_and (goal) (←x on Enterprise))
               (goal ($x has pointed_ears))]}

assert [(Spock on Enterprise)]
assert [(Kirk on Enterprise)]
assert [(Spock has pointed_ears)]
goal   [(←x is rational)]
```



(b) One possible LISP implementation is described below.

Relations will be stored in a hash array. Patterns will be stored in a linear list. Relations and patterns themselves will be represented as lists. Rules of inference will be stored as properties of patterns. A sample rule might be of the form:

```
{To achieve (nice ←x)
     [th_and (goal)(cleaned_up $x))
          (goal (shiny $x))]}
```

These rules are interpreted by the interpreter defined below. Procedures for retrieving and storing relations and patterns in their respective data structures would have to be implemented. In conjunction with these, a pattern matcher would have to be implemented. Given a pattern, it will return all relations in the hash array that match the pattern, as well as the corresponding instantiations of pattern variables.

Deduction will be carried out by a small interpreter. The interpreter will have two main entries: *assert* for asserting a relation, and *goal* for achieving the instantiation of a relation of a given pattern. When a relation is asserted, the pattern list is searched for patterns that match the relation and that have antecedent inference rules associated with them. These rules are retrieved and their bodies interpreted. The goal mechanism is a bit more complicated. The interpreter will keep an explicit tree of goals to be achieved and their corresponding subgoals, and it will visit nodes in the tree depth-first. Whenever a goal is attempted two things happen: an entry for it is created in the goal tree and the pattern list is searched for entries that match the goal pattern and have consequent rules of inference associated with them. These rules are retrieved and interpreted. The tree is thus grown depth-first. If a goal succeeds, the corresponding node is marked accordingly and the interpreter backs up the tree to the parent goal and attempts to achieve any remaining needed subgoals. If a goal fails, the corresponding node is marked as "failed" and the failure is carried to the parent goal by the interpreter. Whenever the root node is marked as either "successful" or "failed" the interpreter terminates, with the corresponding result. Variable instantiations that arise from goal attempts are carried within the goal node itself. If a goal succeeds, its variable bindings are propagated to its parent goal. Otherwise they remain attached to it (or possibly are removed). This explicit goal tree will thus hold the necessary information for implementing backtracking.

Some alternatives to the above solution are: (1) a mini-resolution theorem prover, (2) a QLISP-like system, or (3) a deduction program on the Quillian net.

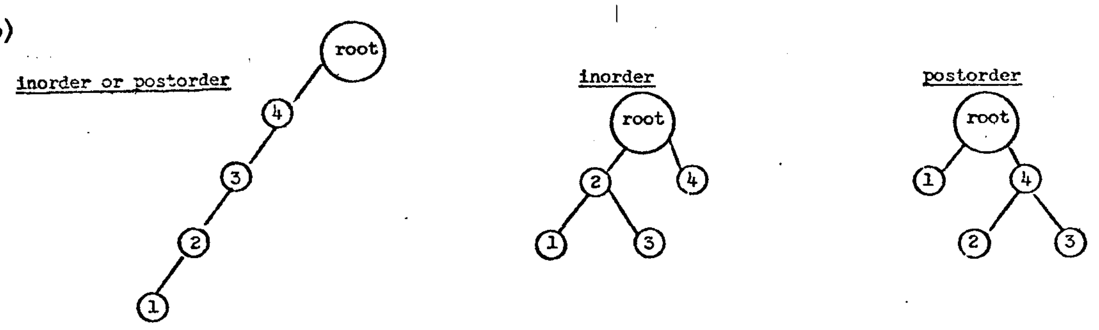
5. If we assume an evaluation function has been given, the described implementation can be modified as follows.

The interpreter, in addition to keeping a goal tree, will have a goal list in which all currently active goals appear. This list is kept sorted according to the values of the evaluation function on each goal. Whenever a new goal is created it is placed both in the goal tree and in the goal list in its proper place (depending on the value of the evaluation function applied to it). The current goal is always taken to be the "best" goal in the goal list. The interpreter will have to be modified as follows: whenever new subgoals are created to achieve some goal and inserted into the goal list, the "best" candidate from the goal list is taken to be the next goal to be attempted by the interpreter. The rest of the interpreter structure will remain the same as before.

Analysis of Algorithms

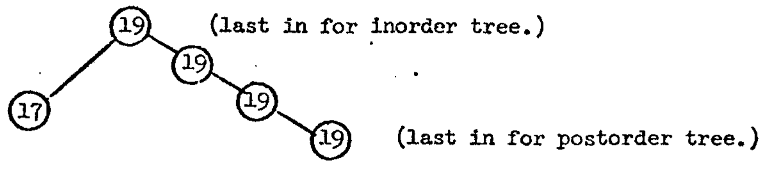
1. (a) Binary search takes $O(\lg n)$ time, while sequential search takes $O(n)$ time.

(b)



The structure of the tree is only loosely determined by the method of traversal.

(c) Sample subtree:



The number in the node is the event time of the notice of that node.

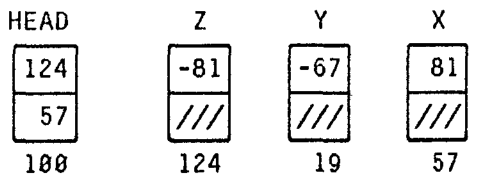
Insertion for the postorder tree is often faster because the event time of a given node is ≥ event times of all nodes in its subtrees. Thus insertion can take place fairly high in the tree without the need to follow a branch to the the end to find an empty slot as in the normal inorder tree.

2. (a) We replace the two pointers for each node by a single pointer value which is the difference of the two pointers. Thus if we have one of the original pointers we can obtain the other one from this value. The head node points to the beginning and end of the list. (If the list is empty, the head node points to itself.) See Hoare, *Structured Programming*, p. 140.

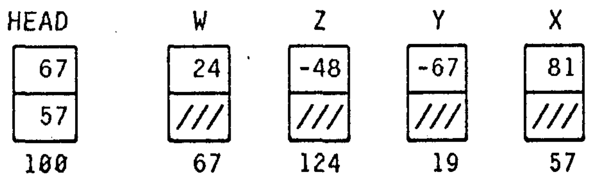
(b)
```
procedure WRITE(X,HEAD);
   if HEAD ≠ WRITE(HEAD) then
      PTR(WRITE(HEAD)) ← PTR(WRITE(HEAD)) - X + HEAD;
   PTR(X) ← WRITE(HEAD) - HEAD;
   WRITE(HEAD) ← X;

procedure DELETE(HEAD);
   if HEAD ≠ READ(HEAD) then begin
      X ← READ(HEAD);
      READ(HEAD) ← HEAD - PTR(X);
      if READ(HEAD) ≠ HEAD then
         PTR(READ(HEAD)) ← PTR(READ(HEAD)) - X + HEAD;
   end;
```

(c)



(d)



3. (a) 3 linear arrays.
(b) Binary tree.
(c) (1) Put tops of stacks at opposite ends of a block of space and have them grow toward each other. (2) Use dynamic allocation of list space, with garbage collection.
(d) Sequential list of entries $(i,\ j,\ A[i,j])$ for all nonzero $A[i,j]$.
(e) Hash table.

4. (a) Clearly $\max_{1\le i\le n}|x_i-\theta| \ge |\beta-\theta|,\ |\theta-\alpha|$. Therefore $2\cdot\max_{1\le i\le n}|x_i-\theta| \ge |\beta-\theta|+|\theta-\alpha| \ge |\beta-\alpha|$, or $\max_{1\le i\le n}|x_i-\theta| \ge |\beta-\alpha|/2$. Equality occurs when $\theta = (\alpha+\beta)/2$, so this value of θ minimizes $|x_i-\theta|$.
(b) We can find α and β using essentially $3n/2$ comparisons, as follows. First we compare x_{2i-1} with x_{2i} for $i = 1, 2, \ldots, n/2$ ($n/2$ comparisons). The larger elements from each comparison are saved in an array MAX(i) and the smaller elements in MIN(i). Then β can be computed as the maximum element of MAX(i), and α as the minimum element of MIN(i). Each of these computations requires $(n/2)-1$ comparisons, for a total of $(3n/2)-2$ comparisons.

Theory of Computation

1. For both parts, let $\{P_i \mid i = 1, 2, \ldots\}$ be an effective listing of all computable functions. For example, P_i might be the ith ALGOL W program and $P_i(x)$ would be the output given by that program on input x.
(a) Suppose that there is an algorithm to decide whether or not a given procedure calls itself. Consider the procedure ALPHA(x) defined as follows:

```
PROCEDURE ALPHA (INTEGER X);
BEGIN INTEGER I,J;
I ← X DIV 2;
J ← X MOD 2;
IF J = 0 AND $P_i(i)$ halts THEN ALPHA (X+1);
END;
```

Clearly ALPHA(2*i) calls ALPHA recursively if and only if $P_i(i)$ halts. Thus we can solve the halting problem, which is a contradiction.

(b) Suppose that there is an algorithm to decide whether or not two programs have the same I/O characteristics. Now let $g(i)$ be such that $P_{g(i)}(x) = 0$ if $P_i(i)$ halts, and undefined otherwise. Also, let i_0 be such that $p_{i_0}(x) = 0$ for all x. Then $p_i(i)$ halts if and only if $P_{g(i)}$ and P_{i_0} have the same I/O characteristics. Thus we can solve the halting problem, a contradiction.

2. (1) context free, (2) context free, (3) regular, (4) recursively enumerable, (5) context sensitive, (6) context free, (7) context sensitive, (8) not recursively enumerable, (9) regular, (10) regular.

3. (1) true, (2) true, (3) false, (4) unknown, (5) false, (6) unknown, (7) false, (8) true, (9) true, (10) false.

Hardware

1.

	BCD				Excess-3			
	B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
(0)	0	0	0	0	0	0	1	1
(1)	0	0	0	1	0	1	0	0
(2)	0	0	1	0	0	1	0	1
(3)	0	0	1	1	0	1	1	0
(4)	0	1	0	0	0	1	1	1
(5)	0	1	0	1	1	0	0	0
(6)	0	1	1	0	1	0	0	1
(7)	0	1	1	1	1	0	1	0
(8)	1	0	0	0	1	0	1	1
(9)	1	0	0	1	1	1	0	0
(10)	1	0	1	0	d	d	d	d
(11)	1	0	1	1	d	d	d	d
(12)	1	1	0	0	d	d	d	d
(13)	1	1	0	1	d	d	d	d
(14)	1	1	1	0	d	d	d	d
(15)	1	1	1	1	d	d	d	d

E_3

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	0	1	1	1
11	d	d	d	d
10	1	1	d	d

E_2

B_3B_2 \ B_1B_0	00	01	11	10
00	0	1	1	1
01	1	0	0	0
11	d	d	d	d
10	0	1	d	d

E_1

B_3B_2 \ B_1B_0	00	01	11	10
00	1	0	1	0
01	1	0	1	0
11	d	d	d	d
10	1	0	d	d

E_0

B_3B_2 \ B_1B_0	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	d	d	d	d
10	1	0	d	d

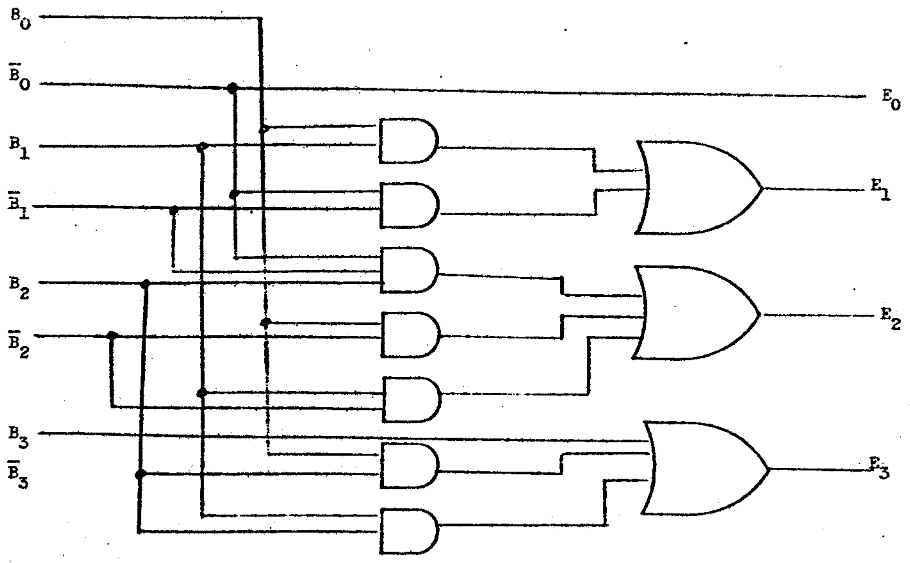
$$E_0 = \overline{B}_0$$

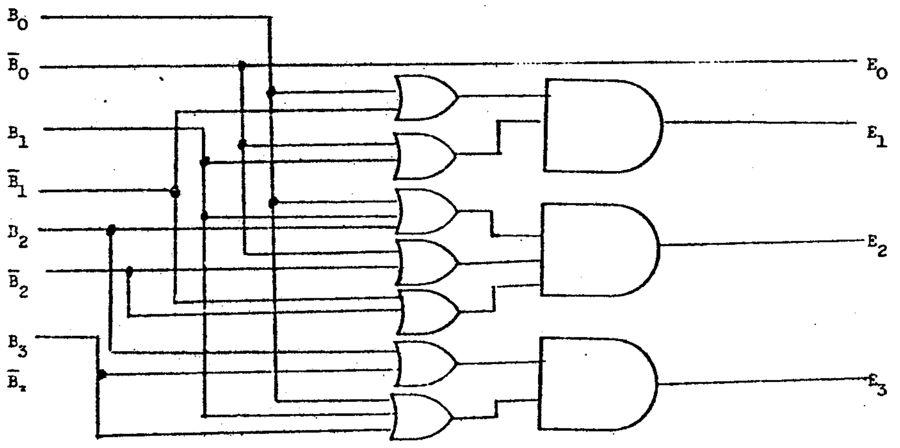
$$E_1 = B_1B_0 + \overline{B}_0\overline{B}_1$$

$$E_2 = \overline{B}_0\overline{B}_1B_2 + B_0\overline{B}_2 + B_1\overline{B}_2$$

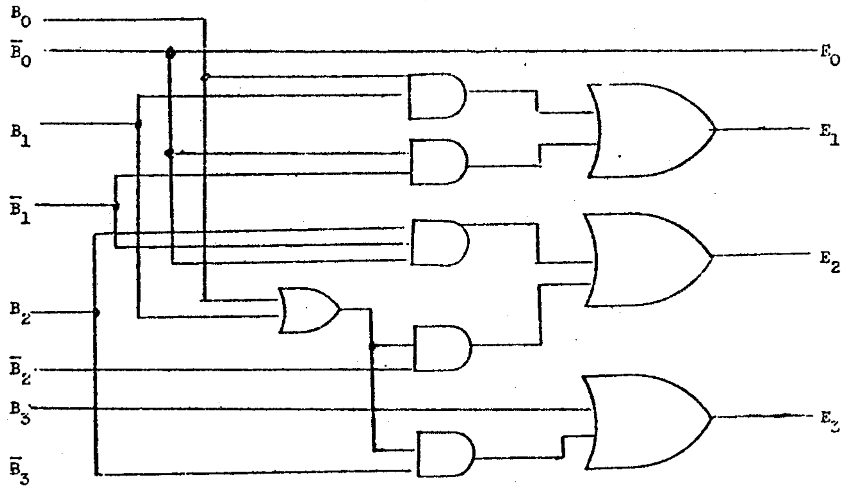
$$E_3 = B_3 + B_0B_2 + B_1B_2$$

(a) Either of the following diagrams:



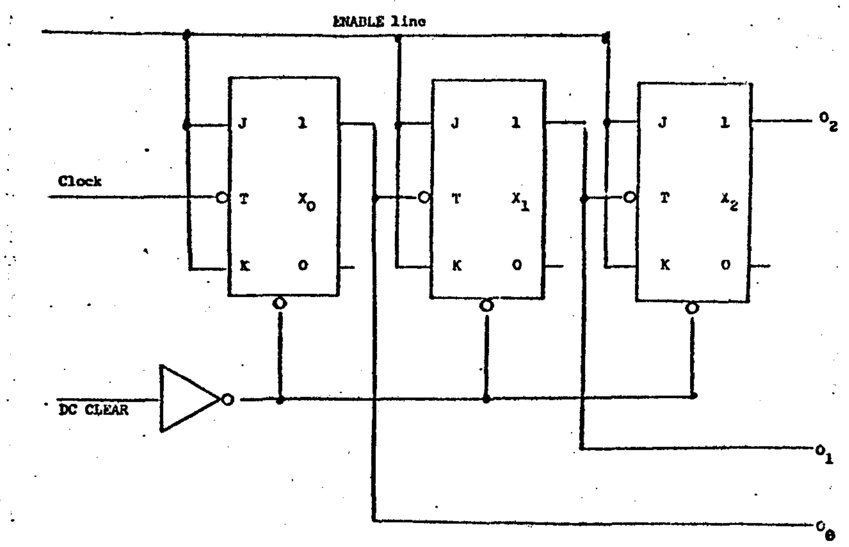


(b)



(c) Same as the second diagram in part (a) except that all gates are replaced by NOR gates.

2. (a)



(b) Outputs listed in the order $o_2o_1o_0$:

$$000 \xrightarrow{ck} 001 \xrightarrow{ck} 000 \longrightarrow 010 \xrightarrow{ck} 011 \xrightarrow{ck} 010 \longrightarrow 000$$

$$\longrightarrow 100 \xrightarrow{ck} 101 \xrightarrow{ck} 100 \longrightarrow 110 \xrightarrow{ck} 111 \xrightarrow{ck} 110 \longrightarrow 100 \longrightarrow 000$$

All arrows which are not labelled "ck" are transitions from unstable states (i.e. a flip-flop has been triggered but hasn't responded).

3. (a) A memory unit whose contents are destroyed in the process of being read. Example: core.

(b) A memory device which requires power to retain information. Example: any TTL RAM.

(c) A memory device in which any location in the device may be directly accessed, with approximately equal access time for each location. Example: core.

(d) Memory which is used for the storage of instructions and not available for storage of working data. Typically implemented via ROMs.

(e) A device which has n inputs and 2^n outputs, where the input is interpreted as a n-bit binary number. For each of the 2^n possible inputs, a unique output line is selected. Example: address decoders.

SPRING 1976 COMPREHENSIVE EXAM

Theory of Computation

1. (a) (i) The set is not regular. Its intersection with the regular set 0^*10^* is $\{0^n10^n \mid n \geq 0\}$, so by Theorem 3.6 in Hopcroft and Ullman it is sufficient to show that $\{0^n10^n \mid n \geq 0\}$ is not regular.

Suppose that a finite automaton accepting $\{0^n10^n \mid n \geq 0\}$ has k states. In scanning the first $k+1$ symbols of $0^{k+1}10^{k+1}$, some state is visited twice; hence the (non-empty) string scanned between visits can be removed from the input without affecting the input's acceptance. This shows that the finite automaton accepts a string which is not of the form 0^n10^n.

(ii) The set is regular. Taking $x = 1$ we find that the set contains $1(0+1)^*1$. But clearly it can contain no more than this, since all strings in the set begin and end in 1. So the set is in fact equal to $1(0+1)^*1$.

(b) By Theorem 3.6 in Hopcroft and Ullman, $\overline{R_2} \cap R_1$ is a regular set, and this set is empty if and only if $R_1 \subseteq R_2$. It is significant that a constructive proof of this theorem is given; by Church's thesis we assert that a Turing machine can be constructed which reads M_1 and M_2 and produces M_3 accepting $\overline{R_2} \cap R_1$. Now if M_3 has k states we simply run M_3 on all inputs of length $< k$ and see whether or not any input is accepted. By Theorem 3.11, $\overline{R_2} \cap R_1$ is empty if and only if no input of length $< k$ is accepted by M_3.

2. (a) If *term* always terminated, it would be a decision procedure for termination of LISP computations. This would enable us to solve the halting problem for Turing machines, which we know is unsolvable.

(b) One possible solution is

```
bad[u] ← subst[u, U, quine[(LABEL FOO
                  (LAMBDA (X) (COND ((U (QUINE X)) (FOO X)) (T T))))]
```

This function was obtained by first constructing from *term* a new LISP function FOO which tells whether a LISP function terminates when it looks at its own S-expression representation, and gives its answer by not terminating when the answer is yes and giving T otherwise. Then *bad*[*term*∗] is the S-expression representation of *foo*[*foo*∗] and the fact that it doesn't terminate follows from its construction but also from carrying out the indicated computation on recursion.

Numerical Analysis

1. For each part of the problem, we use the result that the iteration $x_{k+1} = f(x_k)$ converges to a root α of the equation $x = f(x)$ if $|f'(x)| < 1$ in a neighborhood of α (containing x_0). Since the functions involved are continuous, it is in fact sufficient to show that $|f'(\alpha)| < 1$. Also note that if α, β are the roots of $x^2 + a_1x + a_2 = 0$, then we have $\alpha + \beta = -a_1$ and $\alpha\beta = a_2$.

(a) Here $f(x) = -\dfrac{a_1x_k+a_2}{x} = -a_1 - \dfrac{a_2}{x}$, so $f'(x) = \dfrac{a_2}{x^2} = \dfrac{\alpha\beta}{x^2}$.

Thus $|f'(\alpha)| = \left|\dfrac{\alpha\beta}{\alpha^2}\right| = \left|\dfrac{\beta}{\alpha}\right| < 1$ since $|\alpha| > |\beta|$.

(b) Here $f(x) = -\dfrac{a_2}{x+a_1}$, so $f'(x) = \dfrac{a_2}{(x+a_1)^2} = \dfrac{\alpha\beta}{(x-\alpha-\beta)^2}$.

Thus $|f'(\alpha)| = \left|\dfrac{\alpha\beta}{\beta^2}\right| = \left|\dfrac{\alpha}{\beta}\right| < 1$ since $|\alpha| < |\beta|$.

(c) Here $f(x) = -\dfrac{x^2+a_2}{a_1}$, so $f'(x) = -\dfrac{2x}{a_1} = \dfrac{2x}{\alpha+\beta}$. Thus $|f'(\alpha)| = \left|\dfrac{2\alpha}{\alpha+\beta}\right| < 1$ since $2|\alpha| < |\alpha+\beta|$.

2. (a)
$$\begin{aligned}
fl\left(\sum_{i=1}^{N} a_i\right) &= (fl(a_1+\ldots+a_{N-1}))(1+\delta_1) + a_N(1+\delta_1)\\
&= ((fl(a_1+\ldots+a_{N-2}))(1+\delta_2) + a_{N-1}(1+\delta_2))(1+\delta_1) + a_N(1+\delta_1)\\
&= (fl(a_1+\ldots+a_{N-2}))(1+\delta_1)(1+\delta_2) + a_{N-1}(1+\delta_1)(1+\delta_2) + a_N(1+\delta_1)\\
&= \ldots\\
&= a_1(1+\delta_1)\ldots(1+\delta_{N-1}) + a_2(1+\delta_1)\ldots(1+\delta_{N-1}) + a_3(1+\delta_1)\ldots(1+\delta_{N-2}) + \ldots\\
&\qquad + a_{N-1}(1+\delta_1)(1+\delta_2) + a_N(a+\delta_1)\\
&= a_1(1 + 1.01(N-1)\theta_1{'}u) + a_2(1 + 1.01(N-1)\theta_2u) + \ldots\\
&\qquad + a_{N-1}(1 + 1.01(2)\theta_{N-1}u) + a_N(1 + 1.01(1)\theta_Nu)\\
&= \sum_{i=1}^{u} a_i(1 + 1.01(N+1-i)\theta_iu).
\end{aligned}$$

(b) Yes, you should sum the smallest terms first so that they will contribute to the sum instead of getting lost when they are added to the large partial sum from the first few terms.

(c) We have $fl(n^2+x) = (n^2+x)(1+\delta_1)$ and $fl\left(\frac{1}{n^2+x}\right) = \frac{1}{n^2+x}(1+\delta_1)(1+\delta_2)$, so there are two extra factors $1+\delta_i$.

$$\text{Therefore } S_1 = fl\left(\sum_{n=1}^{N} \frac{1}{n^2+x}\right) = \sum_{n=1}^{N} \frac{1}{n^2+x}(1 + 1.01(N+3-n)\theta_i u),$$

so $|\text{error in } S_1| \le \frac{1}{2}\beta^{1-t} \sum_{n=1}^{N} \frac{1.01}{n^2+x}(N+3-n) \le \frac{1}{2}\beta^{1-t}(1.01) \sum_{n=1}^{N} \frac{N+3}{n^2} \le \frac{1}{2}(1.01)\beta^{1-t}(N+3)(1.64)$.

$$S_2 = fl\left(\sum_{n=N}^{1} \frac{1}{n^2+x}\right) = fl\left(\sum_{n'=1}^{N} \frac{1}{(N+1-n')^2+x^2}\right)$$

$$= \sum_{n'=1}^{N} \frac{1 + 1.01(N+3-n')\theta_n u}{(N+1-n')^2+x^2} = \sum_{n=1}^{N} \frac{1 + 1.01(N+2)\theta_2' u}{n^2+x},$$

so $|\text{error in } S_2| \le \frac{1}{2}\beta^{1-t}(1.01)\sum_{n=1}^{N} \frac{n+2}{n^2} \le \frac{1}{2}\beta^{1-t}(1.01)(1 + \ln N + 2(1.64))$.

(d) For $\beta = 16$, $t = 6$, $N = 10000$:

$|\text{error in } S_1| \le (1/2)(1.01)(16^{-5})(10003)(1.64) \approx 10^{-6}\cdot10^4(1.64)/2 \approx 10^{-2}$.

$|\text{error in } S_2| \le (1/2)(1.01)(16^{-5})(1 + \ln 10000 + 3.28) \approx (1/2)(1.01)(10^{-6})(10.21 + 3.28) \approx 7 \times 10^{-6}$

so the roundoff is much smaller in the second case.

(e) Truncation error $\le \sum_{n=N+1}^{\infty} \frac{1}{n^2}$, which is between $\frac{1}{N+1}$ and $\frac{1}{N}$; so it is about 10^{-4}.

Actual roundoff in first result is

$1.644934 - (1.643517 + .0001) = 1.644934 - 1.643617 = .001317 < .0100$.

Actual roundoff in second result is

$1.644934 - (1.644833 + .0001) = 1.644934 - 1.644933 = .000001 < 7\times10^{-6}$.

Systems

1. (a) Run time stack, since the space needed depends on the depth of recursion, and bindings follow stack discipline.

(b) Run time stack, for the same reason as in (a). The fact that binding is on block entry rather than procedure call does not change the needs.

(c) Run time general, since there is no way of knowing ahead of time how many records of what type will be created, and records become free when there are no references to them, which does not follow scoping discipline.

(d) Run time stack, as in (a).

(e) Run time general, as in (c).

(f) Run time general, since there must be a way to allocate new atom headers when the READ is executed, and they are not freed up when the function or block is exited.

(g) Compile time. If in FORTRAN, it must be.

(h) Run time general, since processes are created and destroyed independently of the normal stack discipline. This can be integrated into a stack, as with INTERLISP's spaghetti stack.

(i) Compile time. Pointers to pieces of code can be allocated like any other variables.

(j) Run time stack, as in (a).

2. (a) Op-codes for commonly used instructions (like CONS, MEMBER, etc.).

Micro-coded aid for frequent internal operations such as variable binding, stack manipulations, variable referencing, or function call (as in the LISP interpreter).

Compact encodings for programs and data. E.g. ByteLisp packs most instructions into 9 bits, allowing 4 to be stored per word. Schemes for encoding LISP list structure have been proposed which get a 2-to-1 reduction for typical data. These compacting strategies could be done without microcode, but the encoding and decoding expense would be too great.

Virtual storage management. Much more sophisticated schemes can be used without incurring to much overhead, when address translation is done in microcode.

(b) Static measurements of how often different operations are called for in existing programs. This requires an analyzer which goes through programs gathering statistics. An example in LISP is finding out how many calls in typical programs are to functions with no arguments, one argument, two arguments, etc. A result might be a decision to put in an opcode which was specially designed for calls to functions of one argument, thereby avoiding some of the overhead associated with variable numbers of arguments.

Static measurements of how data is actually distributed in typical programs. For example, if most of list structure involves long CDR chains and short CAR chains, the optimal encoding would be different than if it was a balanced binary tree.

Dynamic measurements of how often different operations are done. This requires a version of the language which has the capability to gather statistics as its runs. At the level of function calls, it can be done with a tracing mechanism, but to handle internal things like stack manipulations and variable lookup, it is necessary to build in special measurement facilities.

(c) If space is the major bottleneck, then the emphasis would be on supporting virtual memory management and on encoding the programs and data into a more compact form.

3. (The machine under discussion is the DEC PDP-11/45.)

(a) All we want to do is (i) make virtual addresses 0 through BOUND-1 valid, all others invalid, and (ii) map virtual address VA into physical address BASE+VA. This we can do as long as BOUND and BASE are multiples of 64 bytes (low order 6 bits zero).

To do the simulation, let BOUND = $n * 8K + m$. Make MR's 0-n resident, upward expanding, and full length (NR="false", ED="up", LF=max='17700), except that MR n has length m (LF=m-64). Make AF of MR i be BASE + i*8K. Make all other MR's nonresident (NR="true").

(b) To use the hardware for paging, make MR's 0-7 upward expanding and full length (ED="up", LF=max='17700), and require that each AF be a multiple of 8K (low 13 bits zero). We get an eight-page address space, each page of 8K byte length.

(c) If we make all MR's upward expanding, we get a segmentation of sorts: there are 8 segments of 8K byte maximum length. We don't have segmentation in the Multics sense because, for example, there is no provision for control of inter-segment references (indexing from a pointer in segment i can easily produce an address in segment $j \neq i$). To be comparable to Multics, the addressing scheme also needs provision for dynamic linkage of segments by symbolic name. Segmentation is clearly a bad approach here because of the restriction to only 8 small segments in a program. Multics allows up to 2^{18} segments to be linked together.

(d) The primary reason for having memory mapping on this machine at all is to allow a large physical memory to be addressed by a machine with a small virtual address space. Large pages are a problem due to the potential for internal fragmentation, but having upward- and downward-expandable, variable-length "pages" may help here. On the other hand, smaller pages would require more overhead to set up; since the mapping registers are in fixed locations (instead of being kept in core, pointed to by some register), the overhead of loading and unloading them can be considerable. Even if page tables are maintained only in mainstore, the space they take up can be considerable when pages are small.

Data Structures

1. (a) A *stack*. This could be implemented as an array *stack*(1::n) with a pointer *top* to the top of the stack. Initially *top* = 0.

```
    Insertion is:
        if top=n then overflow else begin top = top+1; stack(top) ← item end;
    Deletion is:
        if top=0 then empty else begin item ← stack(top); top = top−1 end;
```

Each operation requires $O(1)$ time, average and worst-case. The stack could also be implemented using a linked list.

(b) A *queue*. This could be implemented as an array *queue*(1::n), with two pointers *front* and *back*. Initially *front*=1, *back*=1. We imagine that position 1 follows position n.

```
    Insertion is:
        if (back mod n)+1 = front then overflow else begin
            queue (back) ← item;
            back ← (back mod n)+1
        end;
    Deletion is:
        if back = front then empty else begin
            item ← queue (front);
            front ← (front mod n)+1
        end;
```

Each operation requires $O(1)$ time, average and worst-case. The queue can store $n-1$ items (at least one array position is left empty so that an empty queue and a full queue can be distinguished). A linked list could also be used to implement a queue.

(c) A *heap*. A heap is a complete binary tree such that the root is the smallest element and each child of a vertex is at least as large as its parent. We can number the elements from 1 to n, such that 1 is the root and $2x$ and $2x+1$ are the children of x. Then we can store the entire tree in an array.

To insert, we add the new element at position $n+1$ and "sift up" by interchanging along the path from $n+1$ to the root so that the elements on this path are ordered. Insertion requires $O(1)$ time on the average.

To delete, we return the element at position 1, put the element at position n in position 1, and "sift down" by comparing the current elements to its two children and exchanging it with smaller if necessary. Deletion requrres $O(\log n)$ time on the average.

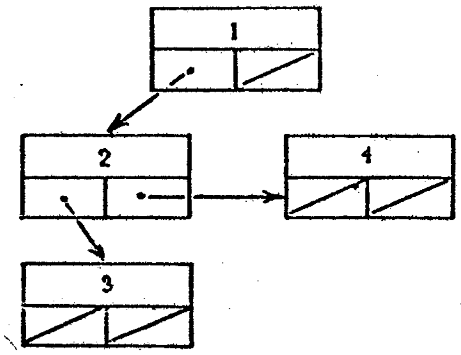
(d) A *hash table*. A hash table is an array of list heads plus a hash function. The hash function, given an arbitrary real number, computes a position in the array.

To insert an item, we compute its hash function and add the item to that list. This always requires $O(1)$ time.

To delete all items of a given value, we compute its hash function and delete all items of the desired value from that list. If the hash table is enough (say larger than the number of items it contains) and the hash function distributes the values evenly into the table, then deletion requires $O(1 + \text{no. of items deleted})$ on the average.

There are many variations possible on this basic hashing scheme. Another reasonable answer to this question is a *balanced tree* of some sort (AVL tree or 2-3 tree). For these structures, the average time per insertion is $O(\log n)$ and per deletion is $O(\log n + \text{number of items deleted})$.

2. (a)



(b) The following ALGOL W solution attempts to mimic the given definition of S_n trees as closely as possible. The main difficulty comes in ensuring that null RSIBLINGS are properly checked for; here this is accomplished by stopping the recursion at N=1 rather than N=0.

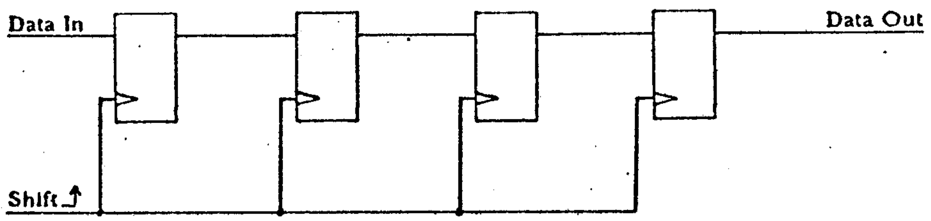
```
PROCEDURE CHECK (REFERENCE (NODE) VALUE S; INTEGER VALUE N);
  BEGIN
  IF RSIB(S)≠NULL THEN ERROR("Root has brother");
  IF N=0 THEN
    BEGIN IF LCHILD(S)≠NULL THEN ERROR("Illegitimate child of root") END
  ELSE CHECK1(S,LCHILD(S),N)
  END CHECK;

PROCEDURE CHECK1 (REFERENCE (NODE) VALUE P,C; INTEGER VALUE N);
  BEGIN
  IF C=NULL THEN ERROR("Missing child");
  IF KEY(P)>KEY(C) THEN ERROR("Parent larger than child");
  IF N=1 THEN BEGIN
    IF LCHILD(C)≠NULL THEN ERROR("Illegitimate child");
    IF RSIB(C)≠NULL THEN ERROR("Unexpected sibling")
    END
  ELSE BEGIN
    CHECK1(C,LCHILD(C),N-1);
    CHECK1(P,RSIB(C),N-1)
    END
  END CHECK1;
```

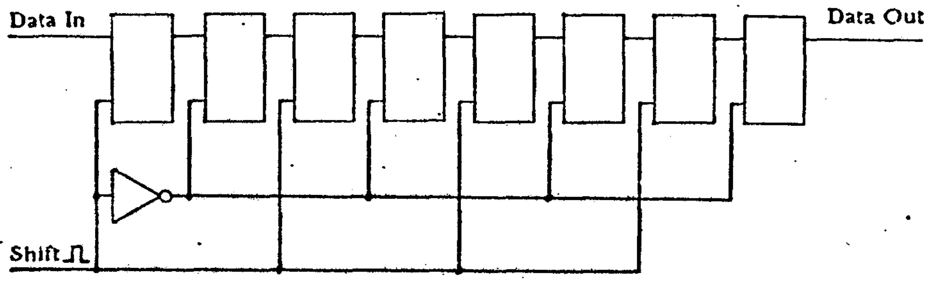
Cleaner solutions are possible by noticing that an S_n tree is really a root whose sons, from left to right, are roots of S_{n-1}, S_{n-2}, ..., S_0 trees. A good non-recursive implementation using this idea requires only two cells of stack space for each level it descends into the tree.

Hardware

1. (a)

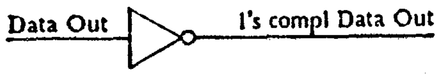


(b)

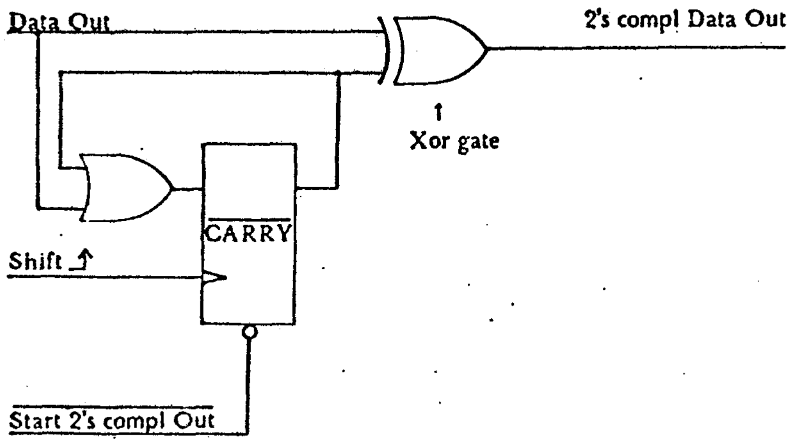


We must assume that the inverter is sufficiently fast. In particular, the delay through the inverter must be less than the delay through a latch, minus the latch's hold time.

2. (a) Just invert Data Out:



(b) Transmit all Data Out bits unchanged until a '1' is produced, then invert the rest:



Some signal ($\overline{\text{Start 2's compl Out}}$ above) must be used to define when the register actually contains the number whose 2's complement is to be shifted out. There will be glitches in 2's compl Data Out when $\overline{\text{CARRY}}$ is set, due to the unequal delay through the last shift register stage and $\overline{\text{CARRY}}$.

Artificial Intelligence

1. The negation of the goal statement is $\forall x \, \exists y \, (P(x) \wedge {\sim}P(y))$ and this gives the clauses $P(x)$ and ${\sim}P(y(x))$ to be proved inconsistent. A single resolution does it.

2. (a) A depth first search avoiding previously visited vertices will take $O(3^n)$ steps since this is the size of the graph.

(b) A recursive program that generates moving all but one disk to the third spike as a subgoal followed by moving that disk and moving the pile back will take $O(2^n)$ steps.

(c) A suitable problem reduction program can discover that the top disk can be moved anywhere, so that any condition on the position of the top disk can always be satisfied. This program will reduce the problem to the $n-1$ disk problem and then to the $n-2$ disk problem, etc. Such a program wil establish a method of moving the disks in $O(n)$ steps although there are 2^n moves.

(d) To "solve" the problem in $O(1)$ steps requires that the program explicitly use mathematical induction or some equivalent mode of reasoning.

WINTER 1977 COMPREHENSIVE EXAM

Systems

1. (a) A *class* is a specification of a packet or template of data and/or procedures and/or a sequence of statements which are to be associated as a unit. Some examples: (1) the data might be two reals representing a complex number and the associated procedures would do arithmetic on these numbers; or (2) the data might be an integer "info" and a pointer "link" to an object of the same class, and the associated procedures might insert and delete from linked lists; or (3) the class might represent a coroutine. Thus, a class is like records in ALGOL W or SAIL but with procedures associated too. Class objects can be created dynamically and be pointed to, and these objects can survive the procedures which created them. We can prove properties about the procedures of a class and use these properties to prove the correctness of a larger program, whenever that program doesn't meddle with the data in an unauthorized manner. Furthermore, a class can be used to specify a "pearl" or a level of program development in which all code for an implementation decision is grouped together.

(b) Test-and-set wastes time doing "busy waiting".

(c) Matrix A would be laid out grouped by rows and matrix B by columns, so that memory references would tend to be grouped and thus require fewer pages of storage.

(d) (1) No flexibility in priority, favors long jobs. (2) Infinite overtake (long jobs may never be run). (3) Swapping overhead.

(e) No. A process cannot request a resource whose index is less than the index of a resource it holds. Thus if process A requires resource i which process B holds, process B cannot request a resource which A holds, because A's resources must all have index less than i, which B already has.

2. (a) For string variables of varying length you can't use the stack to store the string itself because the stack can only contain quantities of a fixed length. Therefore, the entry in the stack will merely contain a *pointer* to the actual string, which is stored in a general data area. The *length* of the string must be indicated in some way, either by including the length with the pointer in the stack; by having the length be the first datum in the string area; or implicitly by having a special end-of-string character.

If the string variable had a known, fixed length, it could be stored on the stack. Even in this case, if the compiler writer assumed that users usually declared strings much longer than necessary and wasted the extra space, the "varying" implementation could be used with the implication that all character positions beyond what was actually stored would be understood to be blank.

(b) In case (i), we know exactly what a "vector" is when the type is declared, so we may as well set up a template of the usual form for an array. We would set up code to fill in the bounds entries right away to whatever they evaluate to at run time; in the example this value is 3. Later references to type vector would either copy or point to this template to describe the variable being declared.

In case (ii), type definitions could become arbitrarily complex before we come to the time to evaluate everything in actually declaring a variable. Since there is not much sense in partially evaluating something while arbitrarily complex segments remain unevaluated, we would probably go to some kind of macro or call-by-name scheme to keep track of the type definition. Accordingly, what we would store would be the unevaluated input text, or some coded version of it.

(c) The major kind of error checking for procedures is the number and types of parameters and the returned result. If a procedure variable can potentially hold any procedure, this checking would in the general case have to be left until run time. However, we could take a stricter view of the type procedure and divide it into multiple types according to the parameters. Thus we could say, for instance, that a procedure or procedure variable has type

boolean procedure (real, real procedure (integer), integer).

Enforcing this discipline would move all type checking to compile time.

Procedure variables also allow for the possibility of calling a procedure from outside its static (block) environment. Since this violates the normal stack discipline, one might dismiss this possibility as an error, and the given example as nonsense. However, coroutines are a valid example of a situation where the stack discipline is insufficient; there must be some way for a procedure which is currently dormant to keep its environment intact. This calls for a procedure to have associated with it pointers to both its caller (dynamic enviroment) and its enclosing block (static environment). As long as a reference to a given procedure exists, the static environment must not be destroyed.

Numerical Analysis

1. The indicated strategy uses a safe but slow method, namely bisection, which guarantees that the error is halved at each step until close to the root. Then the superior convergence rates of either the secant method or Newton's method (orders ≈ 1.6 or 2, respectively) can be used to advantage if the function f is smooth and has a simple zero, since hopefully one has gotten close enough to the root for the asymptotic estimates to apply. Thus, we attempt to avoid the unreliability of the secant method and Newton's method when far from a zero.

2. The secant method only requires one function evaluation per iteration while Newton's method requires the evaluation of both the function and its derivative each iteration. Any additional iterations which may be required for the secant method to achieve desired accuracy is often offset by the expense of computing the derivatives for Newton's method.

3. (a) Underflow or overflow will occur if the magnitude of either x or y lie outside $[M^{-1/2}, M^{1/2}]$ while $|z|$ can still lie in $[M^{-1}, M]$.

(b) $t-3$ bits. If $1/16 \le (v/u)^2 < 1/8$ then the binary representation of the fractional part of $(v/u)^2$ looks like $.0001xxx\ldots$, which has only $t-3$ significant bits.

(c) $t-1$ bits. Since $1/4 \le w \le 1/2$ the fractional part of w looks like $.01xxx\ldots$. Thus, we cannot expect w to have more than $t-1$ significant bits.

4. (a) We store the elements of the original matrix in three vectors of length N, $\underset{\sim}{a} = (a_1, \ldots, a_n)$, $\underset{\sim}{b} = (b_1, \ldots, b_n)$, and $\underset{\sim}{c} = (c_1, \ldots, c_n)$. The resultant matrix will be an upper triangular matrix whose only nonzero elements lie on the main diagonal, on the superdiagonal, and in the last column. We will store our result in the same three vectors $\underset{\sim}{a}$, $\underset{\sim}{b}$, and $\underset{\sim}{c}$. We will store the diagonal and superdiagonal in $\underset{\sim}{b}$ and $\underset{\sim}{c}$ respectively, as before. We will store the first $n-2$ elements of the last column in $a_1, \ldots, a_{n-2}$, i.e., we can relate our vectors $\underset{\sim}{a}$, $\underset{\sim}{b}$, and $\underset{\sim}{c}$ to the matrix as shown below.

$$\begin{pmatrix} b_1 & c_1 & & & & & & a_1 \\ & b_2 & c_2 & & & & & a_2 \\ & & \cdot & \cdot & & & & \cdot \\ & & & \cdot & \cdot & & & \cdot \\ & & & & \cdot & \cdot & & \cdot \\ & & & & & b_{n-2} & c_{n-2} & a_{n-2} \\ & & & & & & b_{n-1} & c_{n-1} \\ & & & & & & & b_n \end{pmatrix}$$

It is easy to see that the elements c_i, $i = 1, \ldots, n-2$ are not changed in the elimination process and that the first row of the matrix is already in the desired form. We assume that the values of the original matrix are already stored in the vectors $\underset{\sim}{a}$, $\underset{\sim}{b}$, and $\underset{\sim}{c}$. At each of the first $n-2$ steps of the algorithm we will introduce a nonzero element in the last row as we zero out the leftmost nonzero element; we will store this element over c_n.

The algorithm:

```
begin
for $i$ := 2 until $n-2$ do
  comment reduction of the $(i-1)$st row and column;
  begin
  $cob := c_{i-1}/b_{i-1}$; $aob := a_{i-1}/b_{i-1}$;
  $b_i := b_i - a_i * cob$; $a_i := - a_i * aob$;
  $b_n := b_n - c_n * aob$; $c_n := - c_n * cob$;
  end
comment reduction of the $(n-2)$nd row and column;
$cob := c_{n-2}/b_{n-2}$; $aob := a_{n-2}/b_{n-2}$;
$b_{n-1} := b_{n-1} - a_{n-1} * cob$; $c_{n-1} := c_{n-1} - a_{n-1} * aob$;
$a_n := a_n - c_n * cob$; $b_n := b_n - c_n * aob$;
comment reduction of $(n-1)$st row and column;
$b_n := b_n - a_n * c_{n-1}/b_{n-1}$
end
```

(b) Pivoting is not necessary if the matrix is positive definite or irreducibly diagonally dominant.

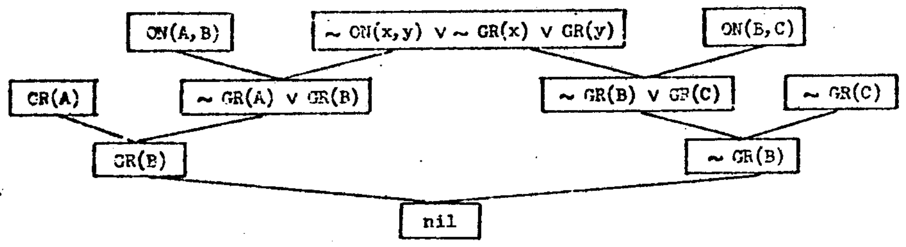
Artificial Intelligence

1. (1) AND/OR trees
   (2) add and delete lists
   (3) consequent theorems
   (4) forward reasoning
   (5) symbol-mapping problem
   (6) Skolem functions
   (7) unification
   (8) set-of-support

   (8) problem reduction
   (3) effects of operators
   (2) backward reasoning
   (5) HEARSAY-II
   (6) inheritance of properties in "isa" hierarchies
   (9) removal of existential quantifiers
   (7) matching
   (10) resolution strategy

2. (a) $(\forall x)\{town(x) \Rightarrow (\exists y)[dogcatcher(y) \wedge lives\text{-}in(y,x) \wedge (\forall z)[dog(z) \wedge lives\text{-}in(z,x) \Rightarrow has\text{-}bitten(z,y)]]\}$
   (b) $\sim(\exists x)\{town(x) \wedge (\exists y)[dog(y) \wedge lives\text{-}in(y,x)] \wedge (\forall z)[dogcatcher(z) \wedge lives\text{-}in(z,x) \Rightarrow has\text{-}bitten(y,z)]\}$
   (c) $(\exists x)\{town(x) \wedge (\exists y)[dogcatcher(y) \wedge lives\text{-}in(y,x)] \wedge (\forall z)[dog(z) \wedge lives\text{-}in(z,x) \Rightarrow \sim has\text{-}bitten(z,y)]\}$

3. Axioms: $ON(A,B)$, $ON(B,C)$, $GR(A)$, $BL(C)$, $\sim GR(C)$.
   Negation of theorem: $\sim ON(x,y) \vee \sim GR(x) \vee GR(y)$.
   One possible refutation:



4. (a) Clause form sometimes results in redundant effort. For example, suppose we are trying to prove $(A \vee B) \wedge C$. The negation of this expression produces the following clauses:

(1) $\sim A \vee \sim C$ (2) $\sim B \vee \sim C$ (3) $\sim D \vee C$ (4) D (5) B

Suppose we begin by resolving (1) and (3) yielding (6) $\sim A \vee \sim D$. Next resolve (6) and (4) yielding (7) $\sim A$. Not being able to resolve (7) against anything, we try (2) and (3) yielding (8) $\sim B \vee \sim D$. Resolving (8) against (4) and the result against (5) produces the contradiction. An analysis of the search for a proof reveals that the process of getting rid of $\sim C$ was the same for both clauses (1) and (2): we used clause (3) and then (4) in two identical sequences. This sort of repetition was caused by having to "distribute" $\sim C$ into more than one goal clause. A natural deduction system, for example, could have proved C once and for all and then taken up the subgoal of proving $A \vee B$.

Often formulas are given (by the domain expert) in a form which suggests the most efficient use. For example, the formula $A \vee B \Rightarrow C$ suggest either (i) to prove C, first prove A and B; or (ii) when A and B are known true, assert that C is true. Converting $A \wedge B \Rightarrow C$ into clause form yields $\sim A \vee \sim B \vee C$. Now we have destroyed the heuristic information regarding how to use this expression. $\sim A \vee \sim B \vee C$ could also have arisen from $A \wedge \sim C \Rightarrow \sim B$, for example.

5. (a) Look at the implementations used for both. These usually involve simple list structures or property list structures that are pretty much the same for both formalisms. So these different formalisms are merely a different way of talking about fundamentally the same representation. Both have to be able to handle quantification and all the logical connectives.

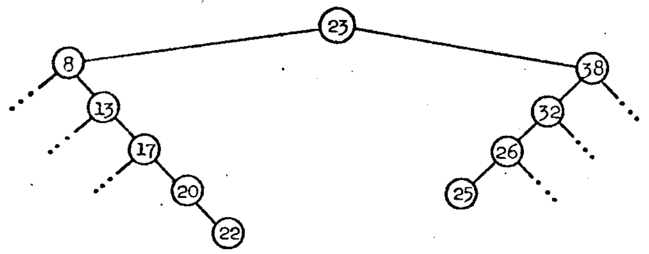
(b) Most implementations of predicate calculus representations deal with "first-order" predicate calculus. Semantic nets allow several "higher-order" constructs such as:

(1) quantification over predicates and functions
(2) typing of arguments
(3) explicit representation of sets, subsets and elements of sets.

In addition most semantic net implementations have two-way indexing between arguments and predicates, which is a feature that most predicate calculus implementations do not have.

Algorithms

1. (a)



where " ... " means the same subtree as before.

(b) Call magic "split" instead. Its body is

```
if $T = 0$ then $L \leftarrow R \leftarrow 0$
else if $x < info[T]$ then begin $split(left[T], x, L, left[T])$; $R \leftarrow T$ end
else if $x > info[T]$ then begin $split(right[T], x, right[T], R)$; $L \leftarrow T$ end
else begin comment $x$ is already in the tree so delete the duplicate;
        $L \leftarrow left[T]$; $R \leftarrow right[T]$;
        comment could also return node $T$ to available storage now;
    end
```

(c) The main thing needed is a clear "invariant" statement about what "split" does, followed by a proof that it does do this.

Procedure $split(T, x, L, R)$ takes a binary search tree T and a value x, and sets L to a binary search tree consisting of all nodes t of T with $info[t] < x$, and sets R to a binary search tree consisting of all nodes t of T with $info[T] > x$. Tree T is destroyed in the process. Furthermore t is an ancestor of u in L if and only if t was an ancestor of u in T, for all $t, u \in nodes\ (L)$; and the same holds for R.

This induction hypothesis is strong enough to prove that $split$ does its thing. For example, if $T = 0$ or $info[T] = x$, the validity is obvious. Otherwise if, say, $x < info[T]$ then $split(left[T], x, L, left[T])$ will (by induction on the size of T, for example) set L correctly and will replace $left[T]$ by the subset of its nodes which are greater than x, hence $R \leftarrow T$ sets R correctly.

Furthermore $BST(x, x_1, \ldots, x_n)$ is the result of $topins(x, BST(x_1, \ldots, x_n))$, for $BST(x_1, \ldots, x_n)$ is the unique binary search tree T such that $\{info[t] \mid t \in T\} = \{x_1, \ldots, x_n\}$ and t is an ancestor of u in T if and only if $i = \min\{k \mid info[t] = x_k\}$ and $j = \min\{k \mid info[u] = x_k\}$ implies $i < j$. Since $topins(x, BST(x_1, \ldots, x_n))$ produces a binary search tree in which x is the root and in which other ancestor relations are present in $BST(x_1, \ldots, x_n)$, the output of $topins$ must be $BST(x, x_1, \ldots, x_n)$.

2. The maximum occurs when all terminal nodes (leaves) have cost equal to their level plus one, and all other nodes have cost zero. *Proof:* If $c(\alpha) > 0$ and α has $k \geq 1$ sons, change $c(\alpha)$ to 0 and add $c(\alpha)$ to the costs of all k sons. This preserves the sum on all paths from the root, so all inequalities are preserved. Furthermore it does not decrease the total cost. After applying this operation a finite number of times we obtain a tree with no smaller cost and all nonzero costs at the leaves; hence there is always an optimum cost assignment of this form. The maximum cost subject to this condition obviously has $c(\alpha) = \text{level}(\alpha)+1$ for all leaves α.

Theory of Computation

1. $u, w \to Q(P(y, y))$; $t, z \to Q(Q(P(y, y)))$; $x, v \to P(Q(Q(P(y, y))), Q(P(y, y)))$.

2. (a) q_1: $0q_1, 1q_2$, (_ *or* 0) *halt*; q_2: $1q_1, 0q_3, 1$ *halt*; q_3: $0q_2, 1q_3, 0q_2$.

(b) Yes. The problem for each j is to output m times the remaining string plus the carry from $x_{j-1} \ldots x_0$, so the minimum necessary and sufficient number of states is the number of distinct carries that can occur. Let y_j be the binary number represented by $x_{j-1} \ldots x_0$, then the carry into position j is $\lfloor my_j/2^j \rfloor$, and this can be any integer from 0 to $m-1$ inclusive when j is sufficiently large. Hence m states are necessary and sufficient.

(c) Not possible. For example, if the machine has n states it cannot form the cube of 2^n, since it must be able to count off $2n$ squares after inputting $x_n = 1$.

3. (a)
$$\begin{tikzcd}[column sep=small] S \arrow[r] & aSbS \arrow[r] & abS \arrow[r] \arrow[dr] & abaSbS \arrow[r] & ababS \arrow[r] & abab \\ & & & abSaSbS \arrow[ur] & & \end{tikzcd}$$

(b) $S \to aSbS \mid abS \mid aSb \mid ab$

(c) Start symbol S_0. $S_0 \to aS_0bS_1 \mid aS_1bS_0 \mid aS_2bS_2$
$S_1 \to aS_0bS_2 \mid aS_1bS_1 \mid aS_2bS_0$
$S_2 \to aS_0bS_0 \mid aS_1bS_2 \mid aS_2bS_1$

(d) Let the given grammar have start symbol S, nonterminals N, terminals A, productions P. The new grammar has start symbol S_0, nonterminals $\{V_0, V_1, V_2 \mid V \in N\}$, terminals A, and productions

$\{V_i \to \beta \mid f(\beta) \equiv i$ (modulo 3) and $t(\beta) = \alpha$ and $V \to \alpha \in P\}$, where

$f(\epsilon) = 0$, $f(a\alpha) = 1 + f(\alpha)$ for $a \in A$, $f(V_i\alpha) = i + f(\alpha)$ for $V \in N$;
$t(\epsilon) = \epsilon$, $t(a\alpha) = at(\alpha)$ for $a \in A$, $t(V_i\alpha) = Vt(\alpha)$ for $V \in N$.

(In other words, the sum of the subscripts plus the number of terminals on the right side is congruent to the left subscript modulo 3.)

Hardware

1. (a) A dynamic memory is a semiconductor memory whose contents need to be periodically refreshed (by reading or writing) to prevent the information from being lost.

(b) A cache memory is a high speed memory in the actual CPU which is used to speed up program execution by remembering recently used instructions or data. For example one might prefetch a block of instructions at a time and allow the CPU to execute out of the cache.

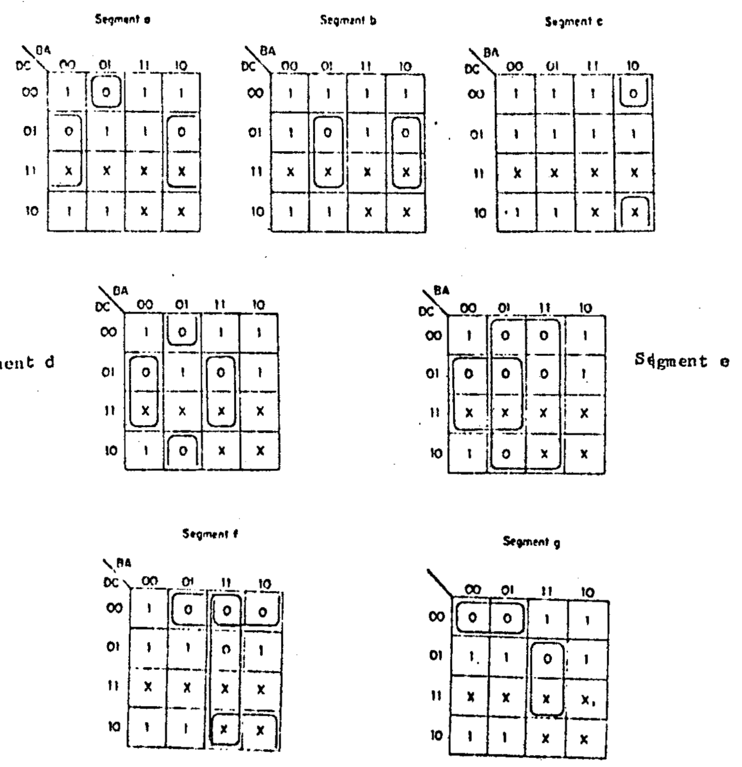
(c) A PROM is a read-only memory (i.e. with fixed content), but whose content can be changed by using special techniques such as ultra-violet light erasure and electrical reprogramming.

(d) Direct memory access occurs when a high-speed peripheral device accesses the main memory directly through a special channel without having to go through the CPU itself.

2. (a) The Karnaugh maps for the 7 output functions a, b, c, d, e, f, g are given below. A simpler design results if one designs the circuit as a product-of-sums (using the 0's of the function) instead of the more usual sum-of-products.

Using De Morgan's theorem, $X \cdot Y = \overline{\overline{X} + \overline{Y}}$, we can use NOR gates exclusively.

Karnaugh maps for Product-of-sums solution



Segment functions

$a = (\overline{A} + B + C + D) . (A + \overline{C})$

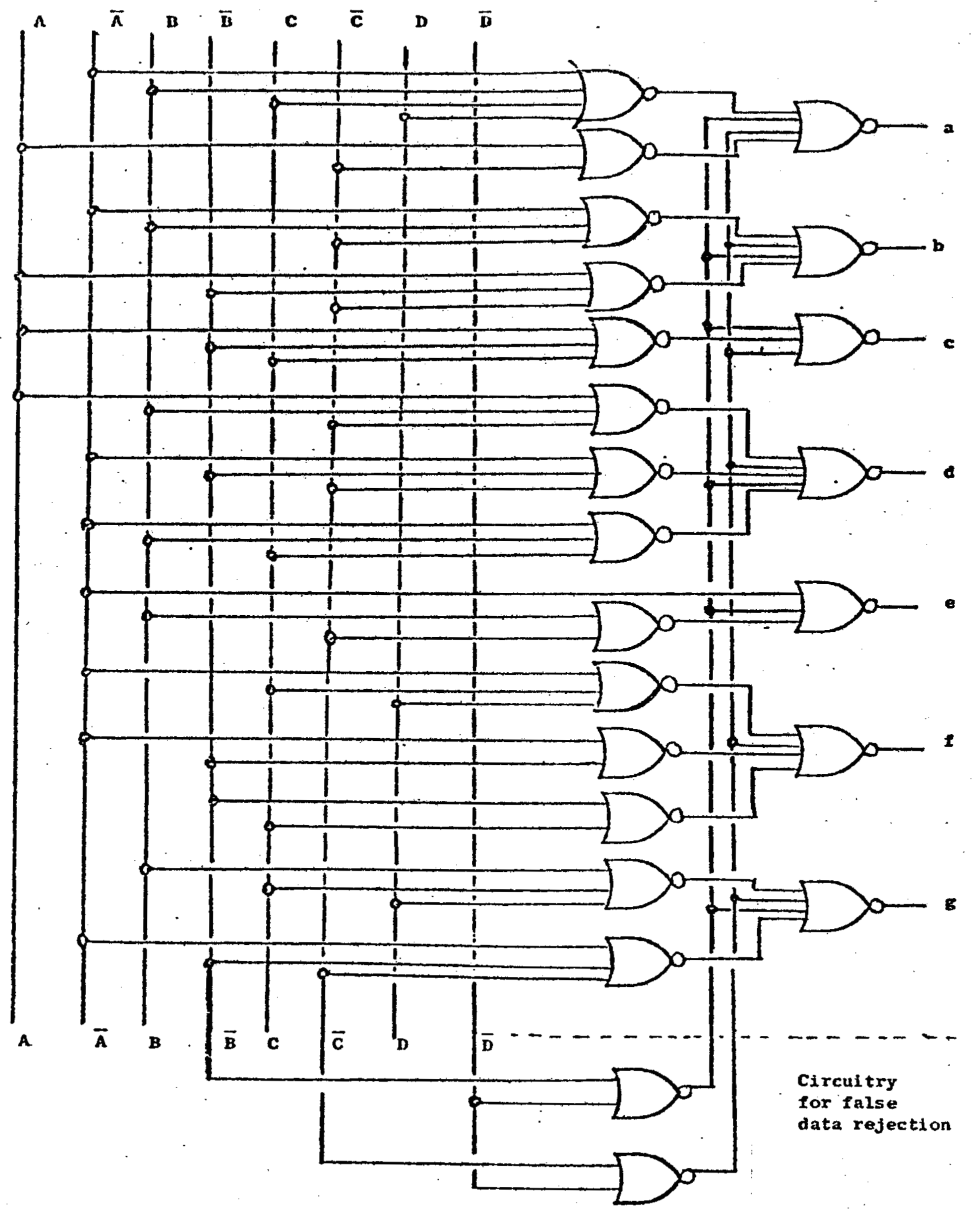
$b = (\overline{A} + B + \overline{C}) . (A + \overline{B} + \overline{C})$

$c = A + \overline{B} + C$ $e = \overline{A} . (B + \overline{C})$

$d = (A + B + \overline{C}) . (\overline{A} + \overline{B} + \overline{C}) . (\overline{A} + B + C)$

$f = (\overline{A} + C + D) . (\overline{A} + \overline{B}) . (\overline{B} + C)$

$g = (B + C + D) . (\overline{A} + \overline{B} + \overline{C})$



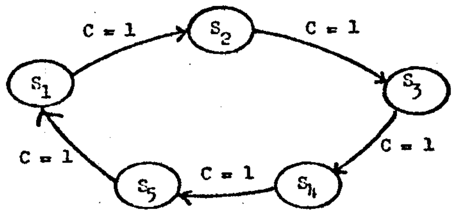
Further savings of gates can be obtained if there is some logic sharing between output functions.

(b) To reject non-code inputs, the entries in the truth table should be all 0's for lines 10 through 15. This corresponds to the addition of prime implicants $(\overline{C} + \overline{D})$ and $(\overline{B} + \overline{D})$ to force all outputs to zero when a non-code input occurs. Both $(\overline{C} + \overline{D})$ and $(\overline{B} + \overline{D})$ should be added to functions a, b, c, d, and g, whereas only implicant $(\overline{B} + \overline{D})$ is needed for function e ($(\overline{C} + \overline{D})$ is already covered), and $(\overline{C} + \overline{D})$ for function f ($(\overline{B} + \overline{D})$ is already covered).

This can be very easily done by adding two more NOR gates to generate $(\overline{C} + \overline{D})$ and $(\overline{B} + \overline{D})$, and feeding their outputs to the appropriate output NOR gates in the previous circuit as shown above.

3. (a) Number of states needed = 5. Minimum number of flip-flops = $\lceil \log_2 5 \rceil = 3$.

(b)



State assignment (Q_1, Q_2, Q_3)

S_1	--	000
S_2	--	001
S_3	--	010
S_4	--	011
S_5	--	100

(c)

Transition Table

	C = 0	C = 1
S_1	0 0 0	0 0 1
S_2	0 0 1	0 1 0
S_3	0 1 0	0 1 1
S_4	0 1 1	1 0 0
S_5	1 0 0	0 0 0
	$Q_1\ Q_2\ Q_3$	

Excitation Table

C = 0	Excitation
0 0 0	0d 0d 1d
0 0 1	0d 1d d1
0 1 0	0d d0 1d
0 1 1	1d d1 d1
1 0 0	d1 0d 0d
$Q_1\ Q_2\ Q_3$	J_1K_1, J_2K_2, J_3K_3

d = don't care

Karnaugh maps of Excitation functions

$Q_3 \backslash Q_1Q_2$	00	01	11	10
0	0	0	d	d
1	0	1	d	d

$J_1 = Q_2Q_3$

$Q_3 \backslash Q_1Q_2$	00	01	11	10
0	d	d	d	1
1	d	d	d	d

$K_1 = 1$

$Q_3 \backslash Q_1Q_2$	00	01	11	10
0	0	1	d	0
1	1	d	d	d

$J_2 = Q_3$

$Q_3 \backslash Q_1Q_2$	00	01	11	10
0	d	0	d	d
1	d	1	d	d

$K_2 = Q_3$

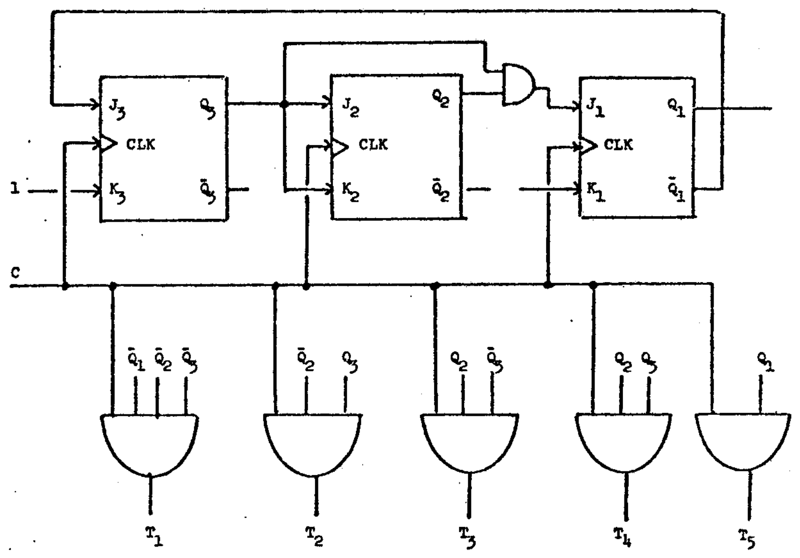
$Q_3 \backslash Q_1Q_2$	00	01	11	10
0	1	1	d	0
1	d	d	d	d

$J_3 = \bar{Q}_1$

$Q_3 \backslash Q_1Q_2$	00	01	11	10
0	d	d	d	d
1	1	1	d	d

$K_3 = 1$

Circuit Diagram



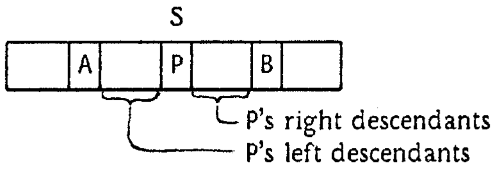
Output functions (using remaining 3 states as don't cares):

$$T_1 = C\bar{Q}_1\bar{Q}_2\bar{Q}_3,\ T_2 = C\bar{Q}_2Q_3,\ T_3 = CQ_2\bar{Q}_3,\ T_4 = CQ_2Q_3,\ T_5 = CQ_1$$

SPRING 1977 COMPREHENSIVE EXAM

Algorithms and Data Structures

1. Consider a sequence which contains the nodes of T in symmetric order. Let S be the subsequence which consists of P and all of P's descendants, and suppose that A immediately precedes S and B immediately follows S.



Now, P is either the left or the right child of its parent. Note that, if P is the left child of its parent, that parent is B, the first node following the entire block S in symmetric order. Similarly, if P is a right child, its parent is A, the node immediately preceding the block S.

We can find these two candidates easily. By following right links from P until we hit a null one, we can find the rightmost node in S, then its right thread will point to B. The other case is handled symmetrically.

There is no easy way to tell whether P is a left or right child of its parent. So, we just try both candidates. Suppose we try B first. Since B is pointed to by a right thread, B must have a non-null left child; we test whether that child is P. If not, A must be the parent.

The above has not considered the case where one or both of A and B are the header node, but everything works out OK. Thus we obtain

Algorithm Parent.

```
P1. [First try B, in case P is a left child.] Q ← P.
P2. [Follow right links.] While RTAG(Q) = "+" do Q ← RLINK(Q).
P3. [And one right thread.] Q ← RLINK(Q).
P4. [Now Q = B, and we know LTAG(B) = "+".] If LLINK(Q) = P then return (Q).
P5. [No good, so now find A; P is a right child.] Q ← P.
P6. [Follow left links.] While LTAG(Q) = "+" do Q ← LLINK(Q).
P7. [And one left thread.] Q ← LLINK(Q).
P8. [Now Q = A, and we know RTAG(A) = "+".] If RLINK(Q) = P then return (Q).
```

2. (a) Since NR(2)-CNF satisfiability is a special case of arbitrary Boolean expression satisfiability, and the latter is in NP, the former is also in NP. To show completeness, we can transform 3-satisfiability into NR(2)-CNF satisfiability.

If every clause has at most three literals, the only way that the NR(2) condition can fail is for a clause to be of the form

$$(\bar{x}_i + \bar{x}_j + \bar{x}_k).$$

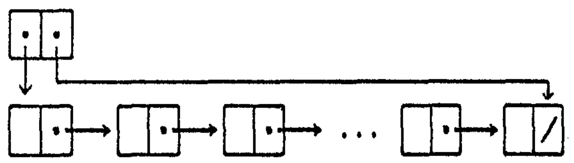
We can replace such a clause by the two clauses

$$(\bar{x}_i + \bar{x}_j + y)(\bar{y} + \bar{x}_k)$$

where y is a brand new variable. It is easy to check that the troublesome clause is satisfiable if and only if both of the replacements are. Furthermore, this transformation can certainly be performed in polynomial time. Thus, any fast algorithm for NR(2)-CNF satisfiability would imply the existence of fast algorithms for 3-satisfiability and hence for all problems in NP.

(b) Here is one possible representation:

Represent clauses as linked lists with the negated variable (if any) last in the list. Represent unordered sets of clauses by linked lists with pointers to both head and and tail, as shown below.



Note that these sets can be merged in constant time.

Also, add an array $A[1{:}n]$ with the convention that $A[i]$ = *false* indicates that the process has discovered that x_i *must* be *false* to have any hope of satisfying the input expression. Otherwise, the value of $A[i]$ is a set (possibly empty) of clauses which can be ignored if x_i is true, but which must be satisfied if x_i is false.

The algorithm initializes all elements of the array A to the empty set, and initializes the variable *input* to a set containing all the clauses of the input expression, represented as stated above. This initialization is clearly linear time. Then, the main loop of the algorithm processes a clause from *input* while maintaining the following invariant: The original expression is satisfiable if and only if there is some assignment of truth values to variables Value(x_i) such that (i) every clause in *input* is satisfied, and (ii) if Value(x_i = *false* then either $A[i]$ = *false* or every clause in $A[i]$ is satisfied.

```
begin
  initialize $A$ and input;
  while input not empty do
    begin
      $C \leftarrow$ any clause in input; remove $C$ from input;
      if $C$ = empty clause then return (unsatisfiable);
      $l \leftarrow$ first literal in $C$; $R \leftarrow$ rest of $C$;
      if $l$ is a negated variable, say $l = \bar{x}_i$, then
        begin
          comment since negated variables are put last, $C$ consists of the single literal $\bar{x}_i$
                    hence $x_i$ must be set to false;
          if $A[i] \neq$ false then
            begin
              input $\leftarrow$ merge ($A[i]$, input);
              $A[i] \leftarrow$ false;
            end;
        end;
      else $l$ is a positive variable, say $l = x_i$, so
        begin
          if $A[i]$ = false then add $R$ to input
            else add $R$ to $A[i]$;
        end;
    end;
  return (satisfiable);
end.
```

Each execution of the while-loop takes constant time with the data structure specified, and a clause of the original input with t literals occupied t+2 words and can generate at most (t+1) executions of the while loop. Hence, the time is $O(m)$.

Artificial Intelligence

1. (a) The solution below uses situation variables to represent the days. A ceremony is thought of as a function from states (days) onto succeeding states (days).

Predicates: *adult*(x), *kyd*(x), *bygshot*(x, s), *inFamily*(x, f, s) and *sqysh*(x, y, s) where x and y are kyds, f is an adult, and s is a day.

Functions on days: *Glom*(f, y, s) = the day following the ceremony in which adult f gloms kyd y at midnight at the end of day s; *Swych*(x, y, s) = the day after x (who was previously a bygshot) swyching with y at midnight of day s.

1. $\forall x, s\ kyd(x) \supset \exists f\ [adult(f) \wedge inFamily(x, f, s) \wedge \forall g\ [inFamily(x, g, s) \supset g{=}f]]$
2. $\forall f, x\ adult(f) \supset \exists y\ [inFamily(y, f, s) \wedge bygshot(x, s) \wedge$
   $\forall z\ [bygshot(z, s) \wedge inFamily(z, f, s) \supset z{=}y]]$
3. $\forall x, y, s\ sqysh(x, y, s) \Leftrightarrow \exists f\ [inFamily(x, f, s) \wedge inFamily(y, f, s) \wedge bygshot(x, s) \wedge x \neq y]$
4. $\forall x, f, s\ inFamily(x, f, Glom(f, x, s))$
5. $\forall x, y, g, f, s\ inFamily(y, g, s) \wedge y \neq x \supset inFamily(y, g, Glom(f, x, s))$
6. $\forall x, y, z, f, s\ inFamily(z, f, s) \supset inFamily(z, f, Swych(x, y, s))$
7. $\forall x, y, z, s\ sqysh(x, y, s) \supset sqysh(y, x, Swych(x, y, s))$
8. $\forall x, y, z, s\ z \neq x \wedge z \neq y \supset [bygshot(z, s) \Leftrightarrow bygshot(z, Swych(x, y, s))]$
9. $\forall x, f, z, s\ bygshot(z, s) \Leftrightarrow bygshot(z, Glom(f, x, s))$

To prove:

$$\forall d1, d2, x, f\ d2{=}Glom(f, x, d1) \supset bygshot(x, d1) \wedge bygshot(x, d2)$$

First to prove it couldn't be a bygshot on the day before: If it was, then by axiom 2, there was no other bygshot in that same family. By axiom 5 together with the uniqueness given by axiom 1, there are no new additions to the family being glommed out of. By axiom 9, none of the old members can become a bygshot at the time of the glomming. Therefore, there is no bygshot in the old family, which contradicts axiom 2.

Second, to prove it couldn't be a bygshot on the day after: If it was, then by axiom 9 it must have been a bygshot on the day before. But we have already proved this impossible.

(b) Goals: *inFamily*(x, z), *Sqysh*(x, y, s), *bygshot*(x)

Operators:

Glom(f, x) Preconditions: f is an adult, x is a kyd, x is not a bygshot
(note this allows you to glom someone in your own family, which is not explicitly forbidden in the problem)
Add: *inFamily*(x, f)
Delete: *inFamily*(x, $*$) for the old family

Swych(x, y) Preconditions: x and y are kyds, x is a bygshot, x and y are in the same family
Add: *bygshot*(y)
Delete: *bygshot*(x)

Differences:

Desired	Operator or subgoal
inFamily(x, f)	*Glom*(f, x)
bygshot(x)	*Swych*($*, x$)
bygshot(x)	*Swych*($x, *$)
Sqysh(x, y, f)	goal: *bygshot*(x)
Sqysh(x, y, f)	goal: *inFamily*(x, f)
Sqysh(x, y, f)	goal: *inFamily*(y, f)

```
Trace:
Goal: Sqysh(b,f,e)
Operator: subgoal bygshot(b) – already satisfied
Operator: subgoal inFamily(b,e)
  Operator: Glom(e,b)
    Preconditions: ~bygshot(b)
      Operator: Swych(b,*)
        Choose * matching preconditions – c or d
        Operation: Swych(b,c)
        Add: bygshot(c)
        Delete: bygshot(b)
    Add: inFamily(b,e)
    Delete: inFamily(b,a)
Operator: subgoal inFamily(f,e) – already satisfied
        Note that at this point a simple planner-like program might think everything was done, since
        the subgoal bygshot(b) was satisfied at one point. However, a GPS-like scheme would still
        have a difference to resolve, and so would search for an appropriate operator.
Operator: subgoal bygshot(b)
  Operator: Swych(*,b)
    Choose * matching preconditions – f because it is bygshot
    Operation: Swych(f,b)
    Add: bygshot(b)
    Delete: bygshot(f)
Goal satisfied
```

2. (a) Speech understanding usually involves:
(1) acoustic processing (involves the actual signals and transformations on them)
(2) phonetic analysis (recognition of phonemes or significant features)
(3) lexical analysis (word recognition)
(4) syntactic analysis (grammatical structures)
(5) semantic analysis (analysis of the meaning of what was said)
(6) pragmatic analysis, or reasoning, or world model (incorporation of what was said into what is known and being done by the system)

Scene recognition involves:
(1) picture processing (involves the actual input corresponding to light intensities on a retina, and transformations on it)
(2) line or feature recognition (often done with a "line finder")
(3) region recognition (may take into account things like texture)
(4) object recognition (often based on a limited domain of possible objects, such as simple straight-edged polyhedra)
(5) comprehension (e.g. as described in Minsky's frames paper)

(b) A modular system can be written by a group in a structured way. Changes to one component have easily controlled effects on how the others operate, and the rules at any one level are easier to comprehend and modify than those dealing with detailed interactions. Modularity forces the representation to be more clearly organized since it is based on communication between different components; and therefore is not usually as ad hoc.

A heterarchical system makes it possible for information from one level to significantly reduce the amount of processing which needs to be done at other levels (e.g. looking for lines only where the edge of an object is suspected). Often, it can make a whole part of the processing unnecessary (e.g. figuring out that a person has used an idiom, without doing a full syntactic analysis). Effort can be allocated to those areas where information is most needed and most likely to be found. Also, the representation can be much simpler since each level can carry all of the useful information in a uniform way.

3. (a) The alpha-beta strategy is used in domains where the planning process has to take into account the efforts of an opponent, trying to minimize the damage he or she can do. It is standardly used in chess. It could be used in travel planning, if nature were considered an opponent (i.e. the program tries to minimize the damage caused by disasters). It is not particularly appropriate in speech understanding since there is no clear place to apply the idea of move and countermove.

(b) The A* strategy is used in domains where there is a good estimate of how successful a plan is likely to be. It is used in things like travel planning (route choosing is a standard domain for illustrating it). It does not apply well in chess, since it does not take into account the negative possible effects of moves by the opponent. It could be used in speech if there were some evaluation of how likely a given parse was to succeed (e.g. in the current SRI system).

(c) Goal reduction is used in domains where the knowledge includes direct goal-reductions of the form: In order to achieve A, do B. This is true in travel planning, and in fact is a standard technique (e.g. McCarthy's original paper on getting to the airport as common sense reasoning). It could be used in chess in a program which had strategies for goals such as "strengthen the king side". This has been talked about often, but not really done yet. It could be applied to a speech system if there were a notion of strategic goals (e.g. figure out the action being referred to, then see who did it), but it seems that nobody has used it in this way.

Hardware

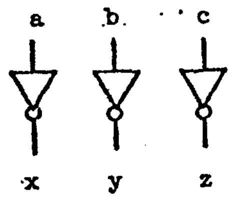
1. (a) A cache is a small chunk of fast memory which is used to hold some of the most frequently referenced words of main memory. The accesses to these popular words can avoid the time delay of probing main memory by instead probing the cache. In order for a cache to be much help, the program should reference small pieces of the memory very frequently instead of probing all of memory at random; in other words, it should display locality of reference.

(b) A pipelined execution unit achieves high speed by overlapping the performances of a sequence of similar computations. At any moment, the different computations are in various stages of completion. Over time, each computation passes through all the intermediary stages in order, with the different computations following each other through the pipe. This organization works especially well on vector arithmetic, for example.

2. (a) Let t be the number of inputs which are one, $0 \leq t \leq 3$. Then, u and v are defined by

t	u	v
0	1	1
1	1	0
2	0	1
3	0	0

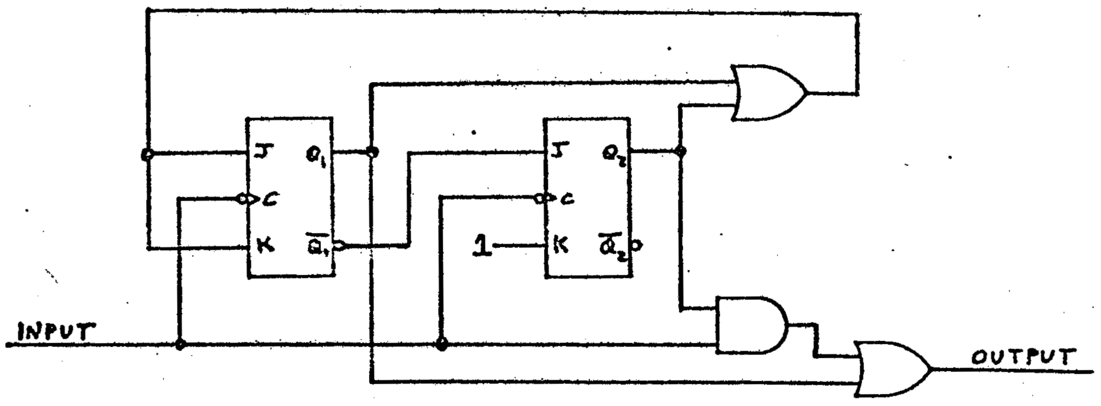
(b)



(c) Circuit F uses only two inverters to invert its three inputs.

(d) Wow! Can you get by with only one inverter? (No, inverting $n \geq 1$ inputs takes at least $\lceil (n+1)/2 \rceil$ inverters.)

3. One possible circuit is



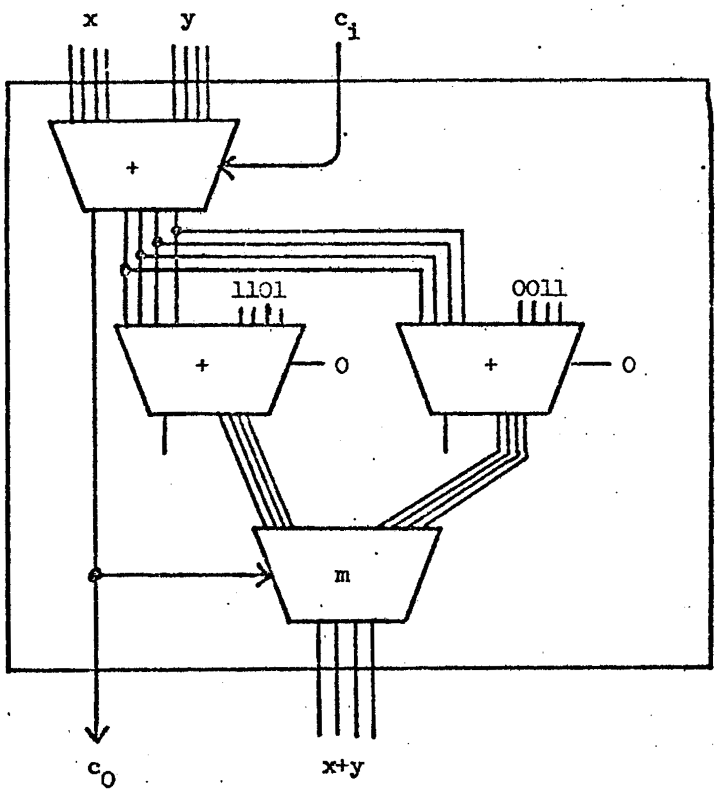
which has the state transitions:

Q_1Q_2

$$\begin{tikzcd}
00 \arrow[r] & 01 \arrow[d] \\
11 \arrow[u] & 10 \arrow[ul]
\end{tikzcd}$$

4. (a) Excess-3 is self-complementing, that is, the 9's complement of a decimal digit in excess-3 code can be computed by complementing each bit of the representation.

(b)



Numerical Analysis

1. (a) A may be large and sparse so that if one forms $PA = LU$ or does a Cholesky decomposition (A symmetric, positive definite) of the form $A = LL^T$, the resulting factors will in general be dense. Hence the finite algorithm may have storage problems that an iterative scheme can avoid.

(b) An iterative scheme often needs only a column or a row of A at one time. Hence if A is so large that only a few rows or columns can be held in fast storage, the iterative scheme may be easier to use than a finite algorithm.

(c) If a good estimate of the solution is known then an iterative scheme can take advantage of this whereas a scheme like Gaussian elimination will not.

2. Consider (1) $A\underset{\sim}{x} = \underset{\sim}{b}$ and (2) $D_1^{-1}AD_2(D_2^{-1}\underset{\sim}{x}) = (D_1^{-1}\underset{\sim}{b})$, where the D_i's are diagonal. Then $D_1^{-1}AD_2$ is a scaled equivalent of A and (2) is a scaled version of (1).

It is often possible to select D_1 and D_2 so that $\mu(D_1^{-1}AD_2) \le \mu(A)$ and hence the sensitivity of (2) to perturbations will be reduced. (Note that if we take $D_2 = I$ then (2) implies $\|\delta\underset{\sim}{x}\|/\|\underset{\sim}{x}\| \le \mu(D_1^{-1}A)\,\|D_1^{-1}(\underset{\sim}{b}+\delta\underset{\sim}{b})\|/\,\|D_1^{-1}\underset{\sim}{b}\|$. Hence the norm on the r.h.s. of the inequality has changed!)

3. If the x_i's are equidistant (or nearly so) the interpolant P_n may oscillate rather badly between the x_i's. This possible radical departure of P_n from f is known as Runge's phenomenon.

If we can select the x_i's we could use Chebyshev interpolation by taking $x_i = \cos[(2i+1)\pi/(2n+2)]$. In this case Powell's theorem (Dahlquist and Bjorck, p.108) tells us that $\|P_n-f\|_\infty \sim (2/\pi)(\ln n)E_n(f)$, and as a special case for $n \le 10$, that $\|P_n-f\|_\infty \le 5E_n(f)$.

4. (a) By Taylor's theorem, we have

$$0 = f(\alpha) = f_n + \epsilon_n f_n' + \epsilon_n^2 f''(x_n+\theta_2\epsilon_n)/2, \quad \text{where } 0 \le \theta_2 \le 1.$$

Rearranging this, we get

$$\epsilon_n = -f_n \,/\, (f_n' + \epsilon_n f''(x_n+\theta_2\epsilon_n)/2).$$

We also have $0 = f(\alpha) = f_n + \epsilon_n f'(x_n+\theta_1\epsilon_n)$ so that $\epsilon_n = -f_n/f'(x_n+\theta_1\epsilon_n)$. Substituting this for ϵ_n on the r.h.s. of the above equation, we get

$$\epsilon_n = \frac{-f_n}{f_n' + f_n f''(x_n+\theta_2\epsilon_n)/2f'(x_n+\theta_1\epsilon_n)} \approx \frac{-f_n}{f_n' - f_n f_n''/2f_n'}.$$

Thus the scheme is

$$x_{n+1} = x_n + \epsilon_n = -2f_n f_n' \,/\, (2(f_n')^2 - f_n f_n'').$$

(b) For the given function the Newton-Raphson (N-R) correction is $\epsilon_n = x_n - x_n^2$ and the new scheme gives $\epsilon_n = 1-x_n$.

step	N-R	error	new scheme	error
0	3/2	1/2	3/2	1/2
1	3/4	-1/4	1	0
2	15/16	-1/16	1	0
3	255/256	1/256	1	0

Clearly the new scheme using more information about the function is better than N-R since it converges in one step!

5. We see that in the interior

$$S'(x_i-) = 2h\sigma_i + h\sigma_{i-1} + h^{-1}[f_i - f_{i-1}]$$

and

$$S'(x_i+) = -2h\sigma_i - h\sigma_{i+1} + h^{-1}[f_{i+1} - f_i].$$

Hence for $i = 2, 3, \ldots, n-1$, we have

$$\sigma_{i-1} + 4\sigma_i + \sigma_{i+1} = D_+D_-f_i = h^{-2}[f_{i+1} - 2f_i + f_{i-1}].$$

From the periodicity conditions it is clear that $\sigma_1 = \sigma_n$ since $S''(x_1+) = S''(x_n-)$. Moreover, $S(x_1+) = S(x_n-)$ implies that $f_1 = f_n$. So we consider S' at the ends from which we see that

$$S'(x_1+) = -2h\sigma_1 - h\sigma_2 + h^{-1}[f_n - f_{n-1}]$$

and

$$S'(x_n-) = 2h\sigma_n + h\sigma_{n-1} + h^{-1}[f_n - f_{n-1}] = 2h\sigma_1 + h\sigma_{n-1} + h^{-1}[f_1 - f_{n-1}].$$

Hence

$$4\sigma_1 + \sigma_2 + \sigma_{n-1} = h^{-2}[f_2 - 2f_1 + f_{n-1}].$$

So the system for $\sigma_1, \sigma_2, \ldots, \sigma_{n-1}$ becomes

$$\begin{pmatrix} 4 & 1 & & & & 1 \\ 1 & 4 & 1 & & & \\ & 1 & 4 & 1 & & \\ & & \cdot & \cdot & \cdot & \\ & & & \cdot & \cdot & 1 \\ 1 & & & & 1 & 4 \end{pmatrix} \begin{pmatrix} \sigma_1 \\ \sigma_2 \\ \cdot \\ \cdot \\ \cdot \\ \sigma_{n-1} \end{pmatrix} = \begin{pmatrix} h^{-2}(f_2 - 2f_1 + f_{n-1}) \\ D_+D_-f_2 \\ \cdot \\ \cdot \\ D_+D_-f_{n-2} \\ h^{-2}(f_1 - 2f_{n-1} + f_{n-2}) \end{pmatrix}$$

(You should also verify that $S''(x_i-) = S''(x_i+)$ and $S(x_i-) = S(x_i+)$.) Gaussian elimination without pivoting will work because the matrix is strictly diagonally dominant.

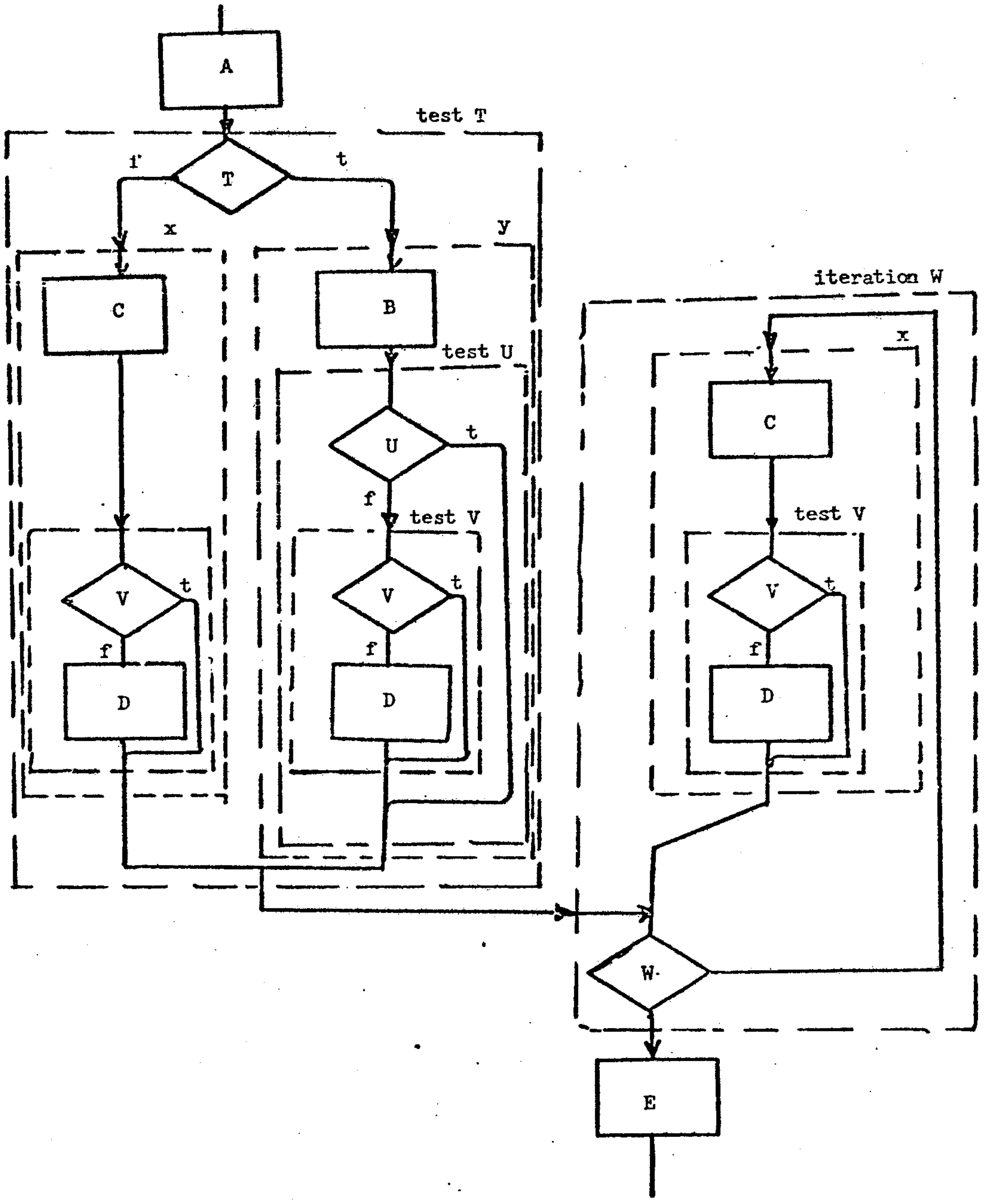
Systems

1. Parameter values and local variables are stored on a stack. Each process calls the program with a stackpointer to its private storage area. The program keeps the stackpointer in registers that are part of the context of a process, i.e. they are saved at process switch time.

2. (a) Let cost be C, and cost per word c. For each segment the storage loss is $p/2$. Each segment requires s/p pages and hence that many page table words. Thus we have $C = cp/2 + cs/p$, which is minimized when $dC/dp = 0 = c/2 - cs/p^2$, so that p(optimal) $= \sqrt{2s}$.

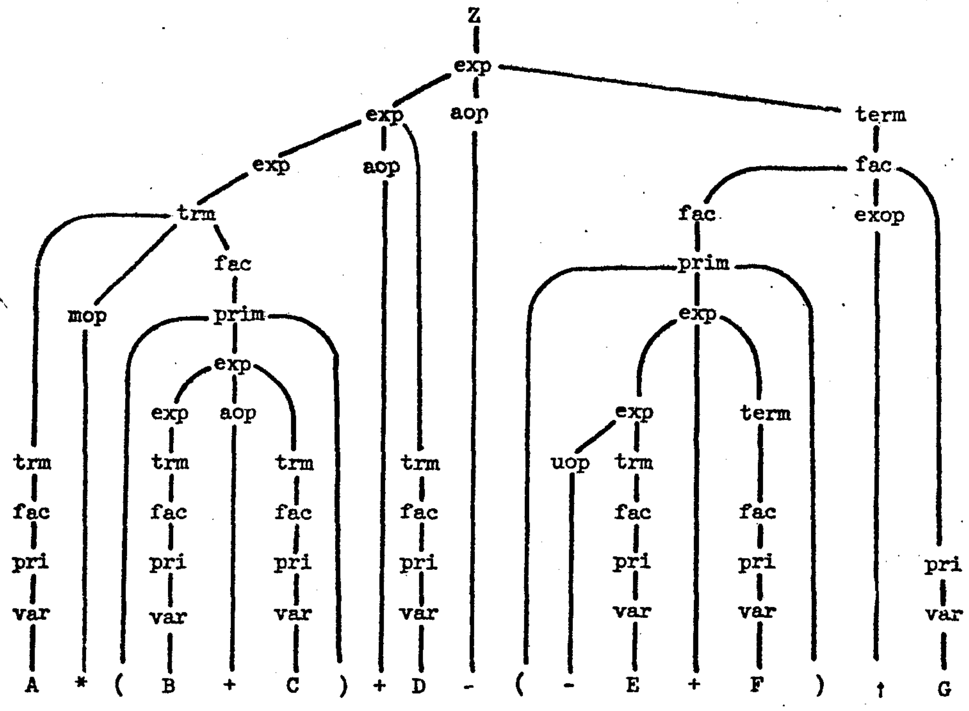
(b) Cost of transfering pages in and out, CPU cost versus memory cost, page faults as a function of page size, hardware block size limitations.

3. The flowchart on the following page gives the complete decomposition, and illustrates suitable lower level procedures.


A
test T
f
T
t
x
y
C
B
test U
U
t
f
test V
V
t
f
D
V
t
f
D
iteration W
x
C
test V
V
t
f
D
W
E

4. (a) Z ::= <exp>
<expr> ::= <term> | <unary op> <term> | <expr> <adop> <term>
<term> ::= <factor> | <term> <multop> <factor>
<factor> ::= <prim> | <factor> <expop> <prim>
<prim> ::= var | (<expr>)
<unary op> := + | -
<multop> ::= * | /
<adop> ::= + | -
<expop> ::= ↑

(b)



(c) The double definition of + and - makes ordinary operator precedence impossible unless the lexical scan differentiates these operators prior to the parse. However, Wirth's "simple operator precedence" can do the job without a special lexical scanner, as shown in *CACM*, January 1966, page 23.

5. The standard deviation should be equal to the mean. The shape of a curve of observations can be plotted or a chi-square test can be run to test the fit. Independence of the service times can be demonstrated if the system is memoryless.

6. (a) Safe if $r \geq p(m-1) + 1$.

(b) Let $h(P_i)$ denote the current allocation to process P_i, for $1 \leq i \leq p$. Then, one example of deadlock occurs when $p = 3$, $m = 3$, $r = 6$, and $h(P_1) = h(P_2) = h(P_3) = 2$.

(c) Find a process P_j which requires the smallest number of additional resources to complete, i.e. $k = m - h(P_j)$ is minimal. If $k \geq r - \Sigma_i\, h(P_i)$, do not allocate to any process other than P_j. (The case where k is strictly greater should never occur.)

7. Determinism, bounded contexts, reserved keywords to identify statements, limitations of strings and comments, terminal occurring only once in the productions.

Theory of Computation

1. The necessary assertion is $(a = F_{n-t}) \wedge (b = F_{n-t+1})$. If a proof of total correctness is desired, it would be necessary to add the conjunct $t \geq 0$.

2. (a) False, the halting problem.

(b) True, just simulate M for n^5 steps.

(c) True. Let t be the smallest exponent for which Fermat's conjecture fails, or let t be ∞ if Fermat's conjecture is always true. Then the decision procedure is

> **if** $m \le t$ **then** *yes* **else** *no*.

3. (a) $L_1 = \{0^n 1 0^n \mid n \ge 0\}$. First, note that L_1 is not regular. This can be proved by considering the equivalence classes under R which contain $0^n 1$ in the language of Hopcroft and Ullman's Theorem 3.1, or by using the *uvwxy* theorem.

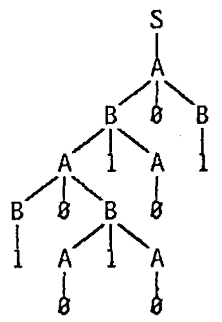
But L_1 can be recognized by a Monotone machine M as follows: the machine makes a preliminary pass over the input to check that $w \in 0^*10^*$, and positions the head over the 1. Then, M enters a loop repeating:

> scan left over 1's until the first 0, and change it to a 1; and
> scan right over 1's until the first 0, and change it to a 1.

If the right end-marker is found by the right scan immediately following the left scan which finds the left end-marker, M accepts.

(b) $L_2 = \{1^n 0 1^n \mid n \ge 0\}$. L_2 is context free, but not regular. Also, all strings in L_2 contain at most one 0. Hence, by the Lemma, L_2 cannot be recognized by any Monotone machine.

4. (a)



The tree on the left and its mirror image are two distinct parses of the sentence
100101001
which has length 9.

(b) Let $u, v, w \in \{0,1\}^*$, and $a, b, c, d \in \{0,1\}$. Suppose that $w \in L(G)$, and $|w| = n$. We can prove by induction on n that

(i)	if $w = a\ b\ v$	then $a \ne b$
(ii)	if $w = u\ c\ d$	then $c \ne d$
(iii)	if $w = u\ a\ b\ c\ d\ v$	then *not* $a = b = c = d$

that is, the conjecture holds, and furthermore, no string in $L(G)$ begins or ends with a double letter. $P(1)$ is trivial, so suppose $S \Rightarrow^* w$ where $|w| > 1$. By symmetry, we can assume that $S \Rightarrow A \Rightarrow^* w$, and since $|w| > 1$, we have $S \Rightarrow A \Rightarrow B_1 0 B_2 \Rightarrow w$. We prove each of the three assertions:

(i) Suppose $w = a\ b\ v$. If a and b both derive from B_1, we are done by induction. If not, we must have $B_1 \Rightarrow 1 = a$ and $b = 0$ so we are OK.

(ii) Symmetrically.

(iii) Suppose $w = u\ a\ b\ c\ d\ v$. If all of a, b, c, d derive from one of the B's, we are done by induction. But if not, two out of a, b, c, d must be the first or last two symbols derived from a B, so we are again done by induction.

WINTER 1978 COMPREHENSIVE EXAM

Algorithms and Data Structures

1. (a) Heapsort. An array A is a heap when $A[i] \geq A[2i]$ and $A[i] \geq A[2i+1]$ for $1 \leq i \leq N/2$. Form a heap by inserting elements $A[i]$ into the partial heap $A[i+1], \ldots, A[N]$, for $i = \lfloor N/2 \rfloor, \lfloor N/2 \rfloor - 1, \ldots, 1$. Next, repeatedly exchange $A[1]$ (the largest remaining element) with $A[j]$ and insert $A[1]$ into the heap $A[1], A[2], \ldots, A[j-1]$, for $j = N, N-1, \ldots, 2$. Insertion consists of exchanging $A[k]$ with the larger of $A[2k]$ and $A[2k+1]$ repeatedly, until $A[k] \geq A[2k]$ and $A[k] \geq A[2k+1]$. Heapsort works in worst-case time $O(n \log n)$, which is the theoretical minimum.

(b) Radix sort. Allocate 10 list headers. Scan through the array, putting each record in the list corresponding to its last digit. Link together the lists in increasing order of last digit, and sort again using the fourth digit. Repeat for each digit. The final list is in sorted order, after $O(N)$ time. This method is familiar to card sorter operators.

(c) Quicksort. Choose a key (say the key from a random record in A). Partition the array into the elements less than the chosen one and those greater. Recursively apply Quicksort to the subarrays, smaller first (to guarantee $O(\log N)$ storage to remember the positions of unsorted subarrays). Quicksort runs in $O(N \log N)$ time on the average, with a smaller constant than Heapsort.

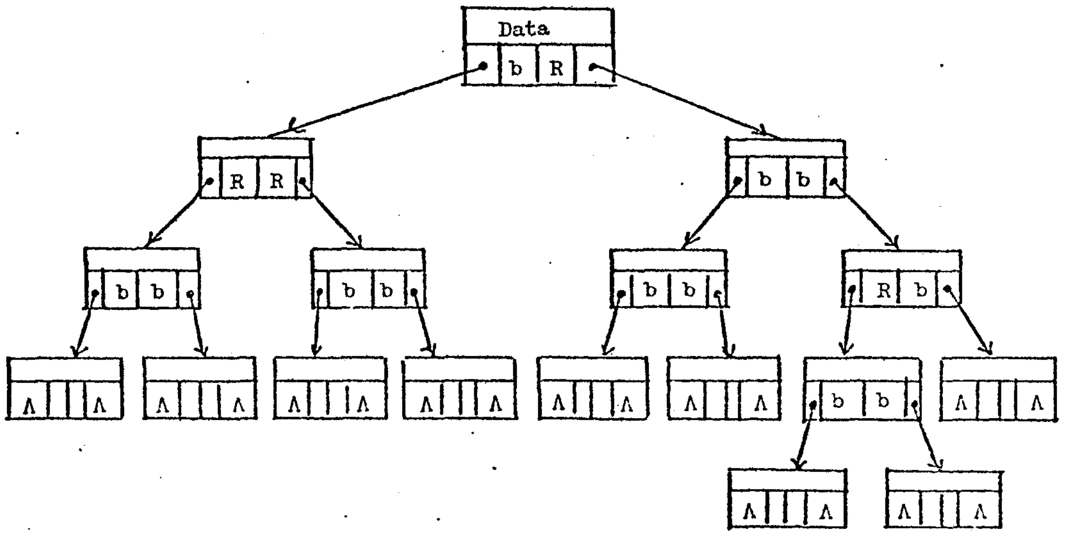
(d) A special case routine. It is possible (and not difficult) to write a short program to sort four elements using only five comparisons in the worst case.

(e) Insertion sort. For $i = 2, 3, \ldots, N$, insert record $A[i]$ into the sorted array $A[1], \ldots, A[i-1]$ by exchanging it with records $A[i-1]$, $A[i-2]$ etc., until we find its proper place. Each record travels one step at a time to its proper place, so the total time is the sum of the distances to the correct places, which we expect to be small.

2. (a)

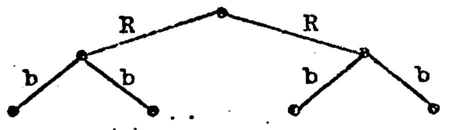
Data	
LeftChild	LeftColor
RightColor	RightChild

LeftChild and RightChild are pointers to other nodes. A leaf will have LeftChild = RightChild = Λ . LeftColor and RightColor take on the values R or b .



(b)
```
procedure CHECK (rbNode node);
begin
  procedure CHECK1 (rbNode node; integer height, depth; boolean underRed);
    comment Check that the tree under node is a legal subtree of an RB-tree of height height,
      if it's at depth depth and under a red edge if underRed is true;
  begin integer d; boolean uR;
    if (LeftChild = Λ) and (RightChild = Λ) then
      begin
        if height ≠ depth then
          ERROR ("There are leaves of B-levels height and depth);
        if underRed then
          ERROR ("There is a red edge leading to a leaf");
      end;
    if (LeftChild = Λ) or (RightChild = Λ) then
      ERROR ("There is a node with only one descendant");
    if underRed then
      if (LeftColor = R) or (RightColor = R) then
        ERROR ("There are consecutive red edges");
    uR ← (LeftColor = R);
    if uR then d = depth else d = depth + 1;
    CHECK1 (LeftChild, height, d, uR);
    uR ← (RightColor = R);
    if uR then d = depth else d = depth + 1;
    CHECK1 (RightChild, height, d, uR);
  end CHECK1;
  integer height; rbNode n;
  height ← 0; n ← node;
  while LeftChild(n) ≠ Λ do
    begin
    if LeftColor(n) = b then height ← height + 1;
    n ← LeftChild(n);
    end;
  CHECK1 (node, height, 0, false);
  PRINT ("The tree is RB balanced of height height");
  end CHECK
```

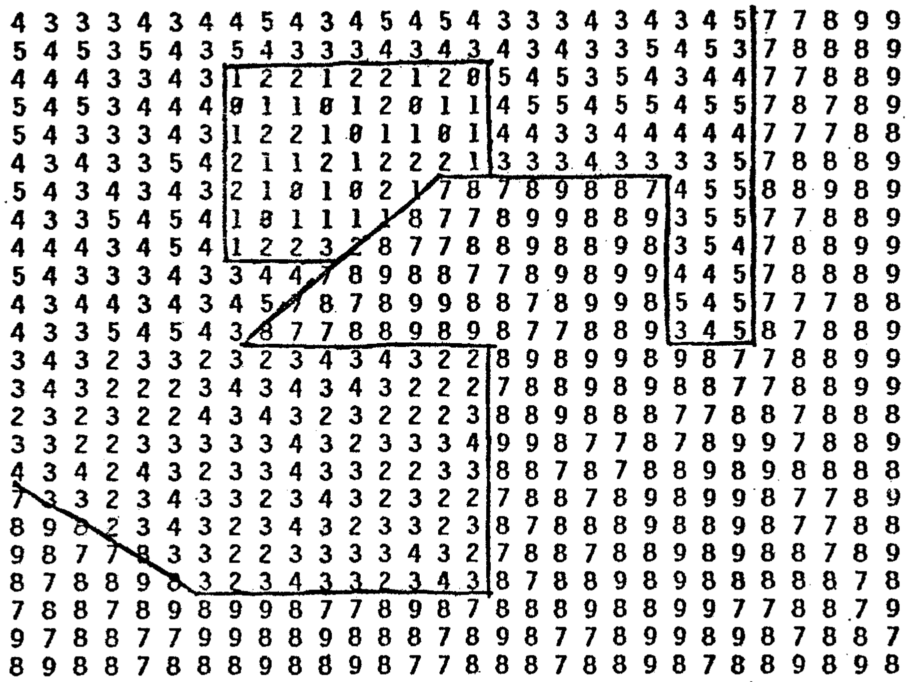
(c) With no red edges at all, we have a complete binary tree of height h which has 2^h leaves. If $h = 1$, the maximum clearly occurs for the tree



which has 4 leaves. For general h, we can view the tree as an RB balanced tree of height 1 whose leaves have been replaced by RB balanced trees of height h-1. We maximize the entire tree by maximizing the subtrees and choosing as many of them as possible. Thus the maximum $M(h)$ satisifies $M(1) = 4$ and $M(h) = 4M(h-1)$, so $M(h) = 4^h$.

Artificial Intelligence

1. (a) The diagram below shows how it is possible to produce a line drawing from the intensity data. A line is drawn wherever the gradient between the values of nearby points is the greatest. The lines below are drawn where the difference in intensity values of adjacent points is greater than or equal to 2. Since there are so many vertices and edges missing from the drawing it is difficult, if not impossible, to identify the object. Also, since it is impossible to determine the exact position of the table relative to the rear wall, it is unclear whether the square region in the upper left is a spot on the wall, or another object partially obscured by a larger object in front.



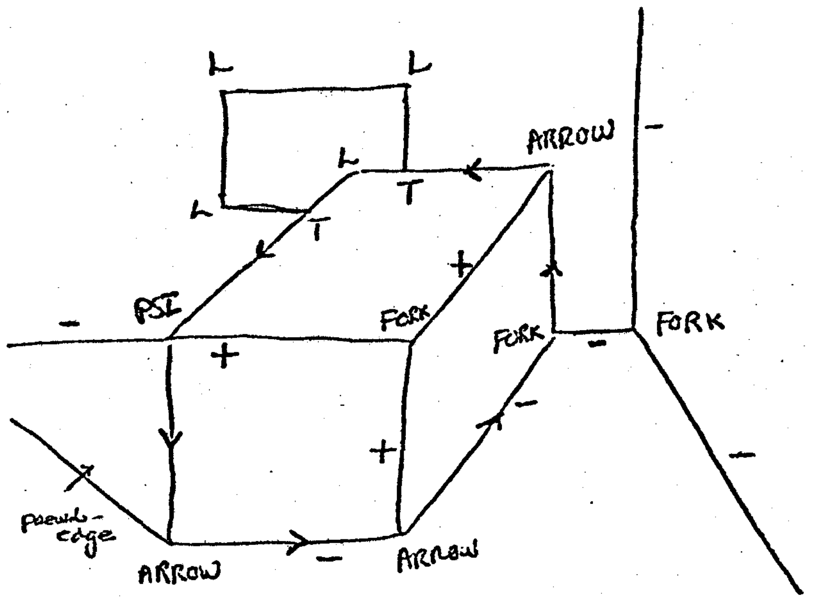
```
4 3 3 3 4 3 4 4 5 4 3 4 5 4 5 4 3 3 3 4 3 4 3 4 5 7 7 8 9 9
5 4 5 3 5 4 3 5 4 3 3 3 4 3 4 3 4 3 4 3 3 5 4 5 3 7 8 8 8 9
4 4 4 3 3 4 3 1 2 2 1 2 2 1 2 0 5 4 5 3 5 4 3 4 4 7 7 8 8 9
5 4 5 3 4 4 4 0 1 1 0 1 2 0 1 1 4 5 5 4 5 5 4 5 5 7 8 7 8 9
5 4 3 3 3 4 3 1 2 2 1 0 1 1 0 1 4 4 3 3 4 4 4 4 4 7 7 7 8 8
4 3 4 3 3 5 4 2 1 1 2 1 2 2 2 1 3 3 3 4 3 3 3 3 5 7 8 8 8 9
5 4 3 4 3 4 3 2 1 0 1 0 2 1 7 8 7 8 9 8 8 7 4 5 5 8 8 9 8 9
4 3 3 5 4 5 4 1 0 1 1 1 1 8 7 7 8 9 9 8 8 9 3 5 5 7 7 8 8 9
4 4 4 3 4 5 4 1 2 2 3 2 8 7 7 8 8 9 8 8 9 8 3 5 4 7 8 8 9 9
5 4 3 3 3 4 3 3 4 4 7 8 9 8 8 7 7 8 9 8 9 9 4 4 5 7 8 8 8 9
4 3 4 4 3 4 3 4 5 7 8 7 8 9 9 8 8 7 8 9 9 8 5 4 5 7 7 7 8 8
4 3 3 5 4 5 4 3 8 7 7 8 8 9 8 9 8 7 7 8 8 9 3 4 5 8 7 8 8 9
3 4 3 2 3 3 2 3 2 3 4 3 4 3 2 2 8 9 8 9 9 8 9 8 7 7 8 8 9 9
3 4 3 2 2 2 3 4 3 4 3 4 3 2 2 2 7 8 8 9 8 9 8 8 7 7 8 8 9 9
2 3 2 3 2 2 4 3 4 3 2 3 2 2 2 3 8 8 9 8 8 8 7 7 8 8 7 8 8 8
3 3 2 2 3 3 3 3 3 4 3 2 3 3 3 4 9 9 8 7 7 8 7 8 9 9 7 8 8 9
4 3 4 2 4 3 2 3 3 4 3 3 2 2 3 3 8 8 7 8 7 8 8 9 8 9 8 8 8 8
7 3 3 2 3 4 3 3 2 3 4 3 2 3 2 2 7 8 8 7 8 9 8 9 9 8 7 7 8 9
8 9 8 2 3 4 3 2 3 4 3 2 3 3 2 3 8 7 8 8 8 9 8 8 9 8 7 7 8 8
9 8 7 7 8 3 3 2 2 3 3 3 3 4 3 2 7 8 8 7 8 8 9 8 9 8 8 7 8 9
8 7 8 8 9 8 3 2 3 4 3 3 2 3 4 3 8 7 8 8 9 8 9 8 8 8 8 8 7 8
7 8 8 7 8 9 8 9 9 8 7 7 8 9 8 7 8 8 8 9 8 8 9 9 7 7 8 8 7 9
9 7 8 8 7 7 9 9 8 8 9 8 8 8 7 8 9 8 7 7 8 9 9 8 9 8 7 8 8 7
8 9 8 8 7 8 8 8 9 8 8 9 8 7 7 8 8 8 7 8 8 9 8 7 8 8 9 8 9 8
```

(b) The most important contribution of the data in this part to analyzing the scene is separating the object from the wall in back. This data establishes that the small square region is in fact part of the wall, and not part of the objects on the table.

(c) The data in this part makes it possible to separate the surface of the table from the walls, and it locates the top face of the object. It is clear from this data that the diagonal line in the lower left is on the table, and not part of the object.

(d) The shadow of the object is the dark region immediately to the left of the object. The diagonal line in the lower left is a pseudo-edge due to the shadow. In the absence of other depth information, a shadow can be very useful source of information. The position of a shadow, for example, could distinguish between an object suspended in space, resting on the floor, or attached to a wall. In the case where one has an object with a hole in it, such as an arch, the shadow can assist in locating the hole.

(e) The diagram below has the edges and vertices labeled, using the same labeling notation as in Winston.



This information is particularly useful as a filter. Usually, due to noise and lighting conditions, there are missing and spurious lines in a line drawing. Since there are only a relatively small number of physically possible labelings compared to all those combinatorially possible, the labeling enables one to discard a large number of possible alternatives.

(f) The object in the scene is, of course, the cube. There are a number of ways a program could establish this fact. One way would be to exploit the constraints of the very limited number of objects in our world. One could have a set of rules like "An object with a triangular face cannot be a cube" or "An object with three rectangular faces visible must be a cube" and eliminate alternatives until the object is identified. Another idea would be to organize the features of each object into a "frame" as described by Minsky. Each frame would identify criterial features which must be present in the object, and the frame could specify a choice of alternative frames to match, suggested by the violations, if any. The frames for each object would be matched with the observations, and the object with the best match would be chosen.

2. (a) Pattern directed invocation is a technique of calling a procedure by matching a desired result with a pattern associated with the procedure. Any procedure whose pattern matches the desired datum is invoked automatically, and it is not necessary for the calling procedure to know the name of the procedure it is invoking. Pattern directed invocation is the basis of a number of AI languages such as PLANNER, CONNIVER and QA4.

(b) An augmented transition network (ATN) is a finite state machine with a pushdown store and a set of "registers" which may hold arbitrary information. Each transition between states has an associated set of conditions and actions. ATNs have proved to be useful devices for recognizing natural language sentences, and a number of AI language understanding systems have been built which employ them, notably, Woods' LUNAR system, and the BBN speech understanding system.

(c) The frame problem arises when one is trying to represent knowledge about a world which may be changed by actions. For any action, one must know exactly which facts about the world remain true and which change, e.g. If I pick up a saucer with a cup sitting in it, I know that the cup will also be picked up, but the table it is sitting on will remain where it is. If the world is large, specifying which facts change and which do not for each action can be difficult.

(d) A blackboard is a data structure which can allow a large number of independent processes (usually specialists in some area) to share information about a common problem they are trying to solve. This has been used successfully in the CMU Hearsay II speech understanding system.

(e) Means-ends analysis is a search technique used in state space searches. One has a set of possible states, and operators which change one state into another. Means-ends analysis is the process of extracting the differences between a current state and a goal state, and matching these differences with relevant operators, perhaps establishing some intermediate state. This idea was the basis for Newell and Simon's GPS program, as well as GPS like planning programs such as STRIPS.

3. If a robot were to teach French, it would have to have an excellent speech understanding capability (assuming, of course, that it will be a better French teacher than a language record). It is true that given the current state of the art, it would be possible for a computer to recognize a vocabulary of 250 words, spoken in isolation, provided it was trained by a particular speaker, and then with less than perfect accuracy. Understanding of continuous speech with a vocabulary that size is possible, as demonstrated by CMU's Hearsay II and Harpie systems, but only if the vocabulary is limited to a narrow domain. Also, this is impossible to do in real time.

Even if the robot was a compact implementation of CMU's speech understanding system, there are many problems encountered in the domain of the language teacher which would make the success of such a robot impossible. Speech understanding systems rely heavily on syntactic, phonological, and other constraints to narrow the range of choices for what has been said. A novice, non-native speaker of a language is likely to make a large number of syntactic and pronounciation mistakes, so these knowledge sources would be unavailable to the system, or severely reduced in their filtering power. In addition, the robot, in order to be an effective teacher, would not only have to understand the sentence despite many mistakes, but be able to recognize minor and subtle mistakes and point out their corrections to the student. Some of these mistakes, such as using the wrong verb tense, would require a large amount of knowledge to detect and correct, and are clearly beyond the capabilities of any language understanding system implemented to date.

Hardware

1. (1) A base address register is a register used to hold the segment start address of the current user. All memory references are relative (within the user segment) and are relocated by adding the base address register.

(2) Gray code is a binary number system in which two successive numbers differ in only a single bit.

(3) A PROM is a programmable read-only memory. It is programmed, once, to have specified values at each address. Thereafter, these values may be read but not written.

(4) Direct memory access is a method of accessing main memory (to remove or insert data) without involving the central processor.

(5) A programmable logic array is a chip consisting of a set of logic with inputs and outputs. A large st of connections is possible between elements in the array. The chip is programmed by making certain connections, to produce the desired output functions.

(6) A bit is a single binary unit of information.

(7) Tri-state logic is a logic system which allows three values: high, low, and floating. When the output is floating it has no effect on anything connected to it.

(8) Multivalue logic is a logic where the possible output values from a unit have more than two values.

(9) Microcode is the program code in control store, which implements a computer instruction set using some hardware which executes micro instructions.

(10)CMOS is an integrated circuit technology which has the characteristics of high density, low power usage, and somewhat slower operation than TTL technology.

2. One possibility is to increase the bandwidth of memory and the memory to processor bus. This will allow more information per transfer and so increase the transfer rate.

Another possibility is interleaved memory. In this scheme memory is divided into "banks". Independent banks can be accessed simultaneously and each bank has an independent bus to the processor.

3. 1's complement: 0101. 2's complement: 0110.

4. Possible answers: indirect, indexed, base register, base register + displacement, absolute, relative.

5. One method is to use redundancy, that is, to have multiple hardware components. A voting scheme can then be used if a failure occurs, thus reducing the possibility of a complete failure.

Another method is to associate error checking information, such as parity bits or CRC, with the data. Examining the error checking bits may indicate when an error which alters the data occurs. This will either allow a fixup or force a retry.

6.

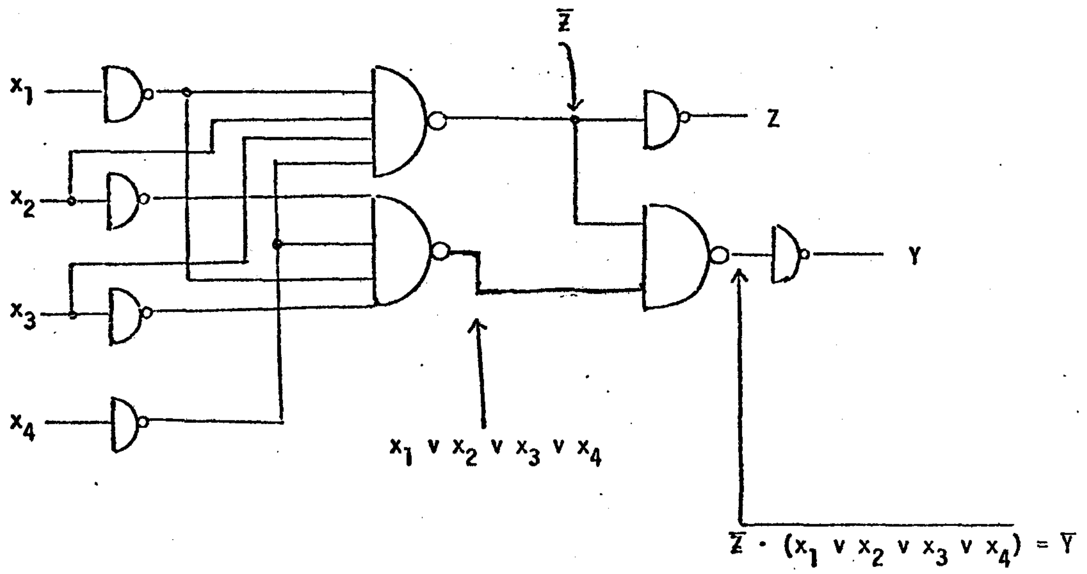
NOR	0	1
0	1	0
1	0	0

NAND	0	1
0	1	1
1	1	0

7. We can write the quadratic as $(x - 123)(x - a)$. We know that $123 * a = 716$ and $123 + a = 129$. Thus $a = 6$, and we have $123 * 6 = 716$. This is possible only if the base is 12. Hence the little green men have 12 fingers.

8. $Z = \bar{x}_1 x_2 x_3 \bar{x}_4$

$Y = Z \cdot (x_1 + x_2 + x_3 + x_4) = Z \cdot (\overline{\bar{x}_1 \bar{x}_2 \bar{x}_3 \bar{x}_4})$


$\bar{Z}$
x_1
x_2
x_3
x_4
Z
Y
$x_1 \vee x_2 \vee x_3 \vee x_4$
$\bar{Z} \cdot (x_1 \vee x_2 \vee x_3 \vee x_4) = \bar{Y}$

9. First compute the effect of the exclusive-or gate network on the counter outputs. The output is $((Q_4 \oplus Q_3) \oplus Q_2) \oplus Q_1$, which is odd parity, i.e. it is high when the number of ones in the counter value is odd.

(a) The count line is high if the counter value has odd parity and the input is zero, or the counter value has even parity and the input is one.

(b) The alarm goes off whenever clock occurs and the value of count is zero.

(c) The shortest key has length 16. Each bit of the key advances the count by one until count = 16, at which point the lock opens. Working out the parity for each count we obtain:

0000	0	0100	1	1000	1	1100	0
0001	1	0101	0	1001	0	1101	1
0010	1	0110	0	1010	0	1110	1
0011	0	0111	1	1011	1	1111	0

To advance the count, the input key should be the opposite of the parity. The lock opens when the counter contains 1111 and the input is 1. Thus the key is 1001011001101001.

Numerical Analysis

1. (1) It is inefficient to form x^i and $i!$ from scratch for every i. (2) There is a risk of overflow in forming x^i as well as $i!$ when $|x|$ and i are large, even though the quotient $x^i/i!$ may be a perfectly reasonable number. (3) Most seriously, this is an extremely poor way to compute $\exp(x)$. In particular, it yields a highly inaccurate result if x is negative and not "small". In this case the terms alternate in sign, and there will be substantial error caused by cancellation, especially since $\exp(x)$ is small for such x.

2. (a) The obvious way to compute sqrt(c) is to use Newton's method for the equation $x^2 - c = 0$. The iteration is

$$x_{n+1} = x_n - \frac{x_n^2 - c}{2x_n} = \frac{1}{2}\left(x_n + \frac{c}{x_n}\right).$$

It can be shown that Newton's method is guaranteed to converge to $\sqrt{c}$ for any $x_0 > 0$. (See Dahlquist and Bjorck, p. 226.)

However, it is not appropriate to simply apply Newton's method for some arbitrary x_0 and let the iteration run until convergence, because we want an efficient as well as reliable procedure. If the initial value is close to $\sqrt{c}$ then the Newton iterates will converge quadratically; thus it is important to obtain a good initial guess.

Since we are working with a binary machine, we can express c in the form $2^{2b}a$, where b is an integer and a lies in the range [1/2,2]. (Note we can do this without introducing any rounding error.) Then $\sqrt{c} = 2^b\sqrt{a}$; and so the square root function needs to be computed only for numbers in the range [1/2,2]. A good initial approximation to $\sqrt{c}$ in this range can be obtained using a simple approximation function (e.g. a straight line).

(b) (1) No, since some representable numbers have non-representable square roots (e.g. $c = 2$).

(2) No. The square root function maps the set of representable numbers onto a smaller subset. Thus, there are at least two distinct representable numbers that will have identical computed square roots.

3. (a) No. Linear transformations do not treat all vectors equally. Since $\underset{\sim}{x} - A^{-1}\underset{\sim}{b} = A^{-1}(A\underset{\sim}{x} - \underset{\sim}{b})$, we have

$$\|\underset{\sim}{x} - A^{-1}\underset{\sim}{b}\| \le \|A^{-1}\|\,\|A\underset{\sim}{x} - \underset{\sim}{b}\|$$

where equality is possible. Thus if $\|A^{-1}\|$ is large, then $\|\underset{\sim}{x} - A^{-1}\underset{\sim}{b}\|$ may be large even if $\|A\underset{\sim}{x} - \underset{\sim}{b}\|$ is small. For example, consider

$$A = \begin{pmatrix} 1 & 1+\epsilon \\ 2 & 2 \end{pmatrix} \qquad A^{-1} = \begin{pmatrix} -\frac{1}{\epsilon} & \frac{1}{2}(1+\frac{1}{\epsilon}) \\ \frac{1}{\epsilon} & -\frac{1}{2\epsilon} \end{pmatrix}$$

where ϵ is small. If $\underset{\sim}{b} = \begin{pmatrix} 2 \\ 4 \end{pmatrix}$ and $\underset{\sim}{x} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, then $A\underset{\sim}{x} - \underset{\sim}{b} = \begin{pmatrix} \epsilon \\ 0 \end{pmatrix}$ which is small, but the exact solution is $\begin{pmatrix} 2 \\ 0 \end{pmatrix}$ so that $\underset{\sim}{x} - A^{-1}\underset{\sim}{b} = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$, which is not small.

(b) Let $\underset{\sim}{x}$ be the solution computed using Gaussian elimination with partial pivoting, and write $(A + \delta A)\underset{\sim}{x} = \underset{\sim}{b}$. If no growth occurs, then we have (see Dahlquist and Bjorck, p. 180)

$$\|\delta A\| \le (n^3 + 3n^2)\, g_n\, \|A\|\, \epsilon,$$

where g_n is the "growth factor", ϵ is machine precision, and we use the infinity norm. Writing $(\underset{\sim}{b} - A\underset{\sim}{x}) = (\delta A)\underset{\sim}{x}$ and assuming $\|\underset{\sim}{x}\| \neq 0$, we get

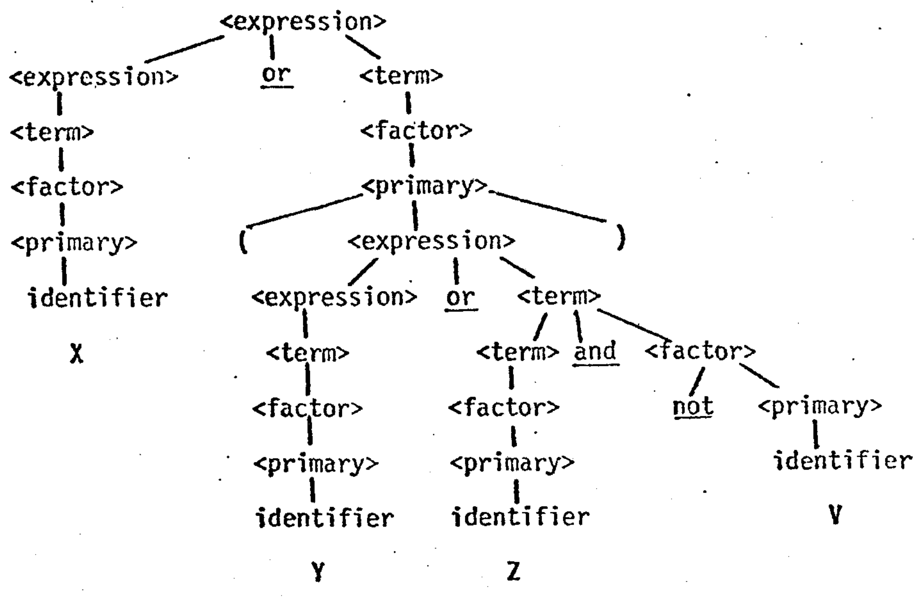
$$\frac{\|\underset{\sim}{b} - A\underset{\sim}{x}\|}{\|\underset{\sim}{x}\|} \le \|\delta A\| \le (n^3 + 3n^2)\, g_n\, \|A\|\, \epsilon.$$

Thus if there is no growth, Gaussian elimination guarantees a residual which is small relative to the computed solution and the original matrix.

(c) No. If iterative improvement converges, it will produce an accurate solution, that is, the final solution will be close to $A^{-1}b$. However, part (a) shows that the accuracy of the solution is not necessarily indicated by the size of the residual, and part (b) shows that Gaussian elimination (without growth) is guaranteed to produce a small residual for the first, unimproved solution. Thus, the norm of the residual does not necessarily decrease during iterative improvement, even though the accuracy of the solution is improving.

Systems

1. The parse tree for *X* or (*Y* or *Z* and not *V*) is



(a) For a bottom up parse we read left to right reducing from the leaves whenever possible. Therefore we obtain

7 6 4 2, 7 6 4 2, 7 6 4, 7 5 3 1 8, 6 4 1.

(b) For a top down parse, we apply productions left to right from the root. Therefore we obtain

1 2 4 6 7, 4 6 8, 1 2 4 6 7, 3 4 6 7, 5 7.

(c)
```
BT      X, L1
BT      Y, L1
BF      Z, L2
BT      V, L2
BT      true, L1
```

2. (a) (1) FIFO does not match typical page-reference patterns, and thus has a high probability of replacing a page that will be used again soon.

(2) LRU has a high overhead (must be updated at every memory reference). However, there are approximations to LRU which can be implemented efficiently.

(b) Assume a system has two tape drives and that (1) Job 1 claims one drive, (2) Job 2 claims one drive, (3) Job 1 requests one drive, (4) Job 2 requests one drive. Now both jobs are blocked because they need a resource held by the other.

(c) I/O to a fast device (drum/disk) lets jobs execute faster and spend less time in memory.

(d) One example is in 370 architecture, which has (1) a memory protect lock for each memory block, (2) a memory protect key in PSW (an access is allowed only if the key matches the lock or the key = 0), and (3) privileged mode, which prevents the user from changing locks and keys.

3. The solution satisfies (a) but not (b).

(a) A process can only enter its critical section if it set $x = 1$. Then $x \geq 1$ will remain true until it leaves the critical section, and this will block other processes.

(b) The following sequence of events is possible:
Process 1 sets $x = 1$ and enters its critical section
Process 2 sets $x = 2$ and fails to enter
Process 1 leaves critical section and sets $x = 1$
Process 1 sets $x = 2$ and fails to enter
Process 2 sets $x = 1$
Process 2 sets $x = 2$ and fails to enter
Process 1 sets $x = 1$

.
. (the last four actions can be repeated indefinitely)
.

4. (a) We need information for each entry/external symbol in a module. The information is contained in two dictionaries. For each entry symbol there is an item in the entry dictionary of the form

symbol	relative address of entry symbol within module

For each external symbol there is an item in the external dictionary of the form

symbol	relative address of an address variable, which should hold the absolute address.

(b) 1. Read in each module and assign starting address.

2. Read in the entry dictionary for each module, update the second field by adding the start address of the module to the relative address within the module. (This yields absolute addresses for each entry symbol.)

3. Read in the **external** dictionary. For each item, find the corresponding item in the **entry** dictionary; if it does not exist give an error for undefined symbol. Otherwise, update the address constant (given by the second **external** item field) to the value in the matching *entry* field.

(c) For a simple language, like Fortran, the compilers must provide type information for each **entry** and **external** symbol representing a variable, function, or formal or actual parameter. In a language with user defined types (e.g. Pascal) a symbol table must be provided.

Checking type compatability involves ensuring that an **external** symbol has the same type as the definition supplied by the **entry** item.

Theory of Computation

1. (a) B: $(1 \le j < i \Rightarrow A[j] \le A[N+j])$
C: $(1 \le j \le N \Rightarrow A[j] \le A[N+j])$
D: $C \wedge (1 \le j < i \Rightarrow min \le A[j] \le A[N+j] \le max) \wedge F$

(b) If assertion $B(i)$ holds before a pass through the loop, then after the loop, $A[j] \le A[N+j]$ for $1 \le j < i$ since these elements are not changed. Furthermore if $A[i] > A[N+i]$ before the loop, then $A[i] \le A[N+i]$ afterward since the test is true and the exchange is performed. Thus $B(i+1)$ holds after the loop.

(c) Upon exit from the loop, $B(N+1)$ holds, which implies C.

(c) The truth of C is not affected by the rest of the program since the array A is never changed. Since $(min = A[1] \le A[N+1] = max)$ holds after the initial assignment, $D(2)$ holds. Assertion F clearly holds throughout the loop.

If $D(i)$ holds before the loop, we have two cases:

Case 1. $A[i] < min$: Then $(1 \le j < i \Rightarrow A[j] \ge min > A[i])$ and $A[i] \ge A[i]$, so after the assignment $(1 \le j < i+1 \Rightarrow A[j] \ge min)$ holds.

Case 2. $A[i] \ge min$: Then $(1 \le j < i \Rightarrow A[j] \ge min)$ so $(1 \le j < i+1 \Rightarrow A[j] \ge min)$ holds after the loop.

Similarly, $(1 \le j < i+1 \Rightarrow A[N+j] \le max)$ after the loop. Combining these facts with assertion C, we find that $D(i+1)$ holds after the loop.

(e) After the loop, $D(N+1)$ holds. Let $1 \le i \le 2N$. Then either:

Case 1. $1 \le i \le N$: Then $min \le A[i] \le A[N+i] \le max$ by $D(N+1)$.

Case 2. $N < i \le 2N$: Then $min \le A[i-N] \le A[i] \le max$ also by $D(N+1)$.

2. (a) Yes. A non-deterministic Turing machine can first check that $k \le n$, and then non-deterministically guess the k indices $i_1, \ldots, i_k$, and finally verify that the k^2 relevant elements are zero. Assuming a cost of n to guess an index and a cost of n^2 to test an element, this algorithm takes time $O(n^4)$.

(b) Reduce the clique problem to ZM.

(c) Given a graph G on n vertices and an integer k, construct the complement of the adjacency matrix. That is, construct a matrix A such that

$$A[i,j] = \begin{cases} 0 & \text{if vertices } i \text{ and } j \text{ are connected in } G \\ 1 & \text{otherwise.} \end{cases}$$

Now solve ZM for the matrix A and the integer k.

The graph G has a clique of size k if and only if there is a solution to ZM for A and k. The transformation takes time $O(N^2)$, where N is the size of the clique problem.

SPRING 1978 COMPREHENSIVE EXAM

Algorithms and Data Structures

1. (a)

```
1. Sort the weights as $x[1] \le x[2] \le \ldots \le x[n]$.
2. Use two pointers: $BIG \leftarrow n$, $SMALL \leftarrow 1$.
3. while $SMALL < BIG$ do begin
       if $x[BIG] + x[SMALL] \le 1$ then begin
           put them in a bin;
           $BIG \leftarrow BIG - 1$;
           $SMALL \leftarrow SMALL + 1$
       end
       else begin
           put $x[BIG]$ alone in a bin;
           $BIG \leftarrow BIG - 1$
       end
   end
4. If $SMALL = BIG$ use a bin for $x[BIG]$.
```

Step 1 uses $O(n \log n)$ time and steps 2–4 use $O(n)$ time.

(b) We prove the optimality of the packing by induction on n. The algorithm is obviously optimal when there is only one object. Now assume that the algorithm is optimal (i.e. uses the minimum number of bins) whenever there are less than n objects, and we shall prove that it is optimal for $\{x_1, \ldots, x_n\}$, where the x_i's have been sorted. There are two cases: (i) $X[1] + X[n] > 1$ and (ii) $X[1] + X[n] \le 1$.

In case (i), $X[n]$ must occupy a bin by itself in any packing. By induction hypothesis, the packing for $X[1], \ldots, X[n-1]$ is optimal, and hence it is optimal for $X[1], \ldots, X[n]$.

In case (ii), we further consider two subcases: (A) In an optimal packing, $X[1]$ is by itself. (B) In an optimal packing, $X[1]$ is paired with some object $X[j]$. In case (A), our packing uses no more bins for $X[2], \ldots, X[n-1]$ than the optimal packing uses for $X[2], \ldots, X[n]$, by induction hypothesis. Hence we are doing at least as well for $X[1], \ldots, X[n]$. In case (B), let us exchange $X[j]$ with $X[n]$ in that optimal packing. Since $X[j] \le X[n]$ and $X[1] + X[n] \le 1$, everything still fits. By induction hypothesis on $X[2], \ldots, X[n-1]$, the packing is optimal.

2. (a) Define $g(v)$ = the value of a maximum flow for the subtree rooted at v, assuming $c(FATHER(v), v) = \infty$.

Then $g(v) = \infty$, if v is a leaf,
$g(v) = \min(g(LSON(v)), c(v, LSON(v))) + \min(g(RSON(v)), c(v, RSON(v)))$, if v is an internal node.

Thus $g(root)$ gives the desired maximum flow value.

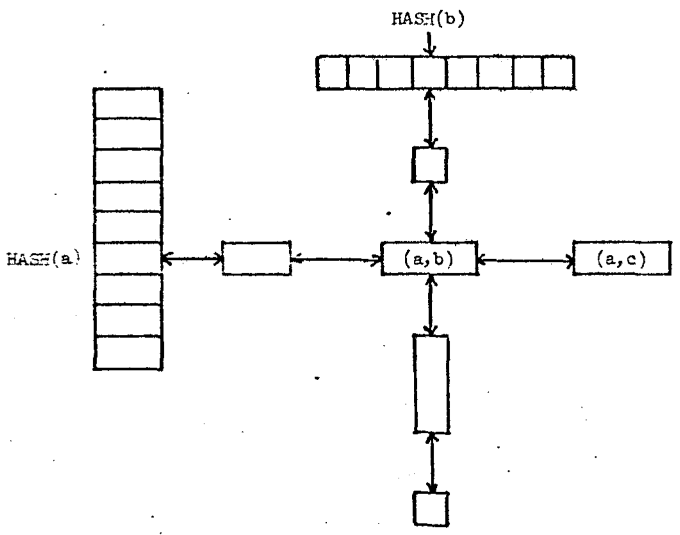
(b) Write $h(FATHER(v), v)$ for $\min(g(v), c(FATHER(v), v))$. So $h(FATHER(v), v)$ is the maximum flow value that can pass through arc $(FATHER(v), v)$ and reach leaves of the subtree rooted at v.

Now compute the maximum flow f from the top down:

$$f(ground, root) = g(root)$$
$$f(v, LSON(v)) = \min(f(FATHER(v), v), h(v, LSON(v)))$$
$$f(v, RSON(v)) = f(FATHER(v), v) - f(v, LSON(v))$$

It is easy to check to f is indeed a flow. Its value is $g(root)$, and therefore maximal.

3.



Each pair (a, b) is stored as a node which is simultaneously in two doubly-linked lists – one hanging from the $hash(a)$ bucket and one hanging from the $hash(b)$ bucket. Thus all elements $(a, *)$ are in the list for $hash(a)$, and all elements $(*, b)$ are in the list for $hash(b)$. The operation $ENTER(a, b)$ takes $O(1)$ time, while the other operations take time proportional to the number of pairs in the list for $hash(a)$ or $hash(b)$. Note that the hash tables for a and b may be either separate or the same.

Artificial Intelligence

1. (This solution goes into a good deal more detail than is expected on the test.)

We let the individuals *House*1, *House*2, *House*3, *House*4, *House*5 stand for the five houses, as viewed from left to right. The colors will of course be *red*, *green*, *ivory*, *yellow* and *blue*; the men *Englishman*, *Spaniard*, *Ukranian*, *Norwegian*, and *Japanese*; the pets *dog*, *snails*, *horse*, *fox* and *zebra*; the drinks *coffee*, *tea*, *milk*, *orange_juice* and *water*; and the cigarettes *Old_Gold*, *Chesterfields*, *Kools*, *Lucky_Strike* and *Parliaments*.

The first problem is to ensure that the different houses, colors etc. are all different. Let us define the predicate *DISTINCT* on five items as follows:

$$\forall\, x1, x2, x3, x4, x5\ (DISTINCT\ (x1, x2, x3, x4, x5) \equiv$$
$$(x1 \neq x2 \wedge x1 \neq x3 \wedge x1 \neq x4 \wedge x1 \neq x5 \wedge x2 \neq x3 \wedge x2 \neq x4 \wedge x2 \neq x5 \wedge x3 \neq x4 \wedge x3 \neq x5 \wedge x4 \neq x5)).$$

Then we can express the distinctness as the WFFs:

1a] *DISTINCT* (*House*1, *House*2, *House*3, *House*4, *House*5)
DISTINCT (*red*, *green*, *ivory*, *yellow*, *blue*)
DISTINCT (*Englishman*, *Spaniard*, *Ukranian*, *Norwegian*, *Japanese*)
DISTINCT (*dog*, *snails*, *horse*, *fox*, *zebra*)
DISTINCT (*coffee*, *tea*, *milk*, *orange_juice*, *water*)
DISTINCT (*Old_Gold*, *Chesterfields*, *Kools*, *Lucky_Strike*, *Parliaments*)

We define the monadic functions *Smokes*, *Owns*, *Drinks*, and *Livesin*, on the men, and *Colorof*, on houses into colors, with the obvious meanings.

Thus, since no man shares a home or a habit with his neighbor, we have:

1b] $\forall x1, x2\ (Smokes(x1) = Smokes(x2) \supset x1{=}x2) \wedge$
$\forall x1, x2\ (Owns(x1) = Owns(x2) \supset x1{=}x2) \wedge$
$\forall x1, x2\ (Drinks(x1) = Drinks(x2) \supset x1{=}x2) \wedge$
$\forall x1, x2\ (Livesin(x1) = Livesin(x2) \supset x1{=}x2) \wedge$
$\forall x1, x2\ (Colorof(x1) = Colorof(x2) \supset x1{=}x2)$

The following parts of the problem become:

2] $Colorof\ (Livesin\ (Englishman)) = red$
3] $Owns\ (Spaniard) = dog$
4] $\forall x\ (Colorof\ (Livesin(x)) = green) \equiv (Drinks(x) = coffee)$
5] $Drinks\ (Ukranian) = tea$
7] $\forall x\ (Smokes(x) = Old\_Gold) \equiv (Owns(x) = snails)$
8] $\forall x\ (Colorof\ (Livesin(x)) = yellow) \equiv (Smokes(x) = Kools)$
13] $\forall x\ (Smokes(x) = Lucky\_Strike) \equiv (Drinks(x) = orange\_juice)$
14] $Smokes\ (Japanese) = Parliaments$

We also need to express the geometry of the world. We define the functions *Right* and *Left*, and state the following axioms:

$Right\ (House1) = House2$	$Left\ (House2) = House1$
$Right\ (House2) = House3$	$Left\ (House3) = House2$
$Right\ (House3) = House4$	$Left\ (House4) = House3$
$Right\ (House4) = House5$	$Left\ (House5) = House4$
$\forall x \sim Right\ (House5) = x$	$\forall x \sim Left\ (House1) = x$

The middle house is, by definition, *House3*; the first house on the left, *House1*. We now get the rest of the problem:

6] $\forall x1, x2\ ((Colorof(x1) = green \wedge Colorof(x2) = ivory) \supset (x1 = Right(x2)))$
9] $\forall x\ (Livesin(x) = House3 \equiv Drinks(x) = milk)$
10] $Livesin\ (Norwegian) = House1$
11] $\forall x1, x2\ ((Smokes(x1) = Chesterfields \wedge Owns(x2) = fox) \supset$
$((Right\ (Livesin(x1)) = Livesin(x2)) \vee (Left\ (Livesin(x1)) = Livesin(x2))))$
12] $\forall x1, x2\ ((Smokes(x1) = Kools \wedge Owns(x2) = horse) \supset$
$((Right\ (Livesin(x1)) = Livesin(x2)) \vee (Left\ (Livesin(x1)) = Livesin(x2))))$
15] $Colorof\ (Right\ (Livesin\ (Norwegian))) = blue \vee Colorof\ (Left\ (Livesin\ (Norwegian))) = blue$.

2. (a) Evans' Analogy program: (1) Only a simple set of possible transformations are recognized. (2) An inflexible notion of figure and change. For example, if the number of subfigures undergoing change (addition, deletion) is not the same between figures, then the program is unable to solve the problem.

(b) Winston's Concept Formation system: (1) This system incorporates only a simplistic notion of learning. If the examples are not given in a most appropriate order, then it is not clear that the system can "learn" anything. (2) The program knows of only a small finite set of primitives. (3) Winston does not mention any actual implementation of his system.

(c) Waltz's Vision system: (1) Too many unrealistic assumptions are made for this program to deal with "real" vision. For example, reality is composed of curved objects, not merely parallelepipeds. In the real world, lighting does not always come from only a single direction and shadows may not be present or may be ambiguous. (2) Waltz relies only upon local line constraints. (3) Waltz's system relies heavily on search; this seems a poor psychological model.

(d) Newell and Simon's GPS: (1) GPS did not prove to be a general problem solver. Many problems are not expressable in means-ends form. (2) A real solution may involve traveling "farther" from the goal in some particular step. GPS would not recognize this. (3) GPS has a purely hierarchical, depth-first search structure. This is both inadequate for general problem solving, and inefficient in requiring repeated deductions.

(e) Fikes and Nilsson's STRIPS: (1) Based on a resolution based theorem prover. (2) Lack of planning ability (resolved somewhat by ABSTRIPS). (3) The usual heirarchical problems of the GPS formalism.

(f) Production Systems: (1) It's a nice idea for a heuristic framework, but leaves completely unstated what exactly is to be said in any set of productions, or how any particular piece of knowledge is to be represented. (2) Great care must be exercised in programming actual production systems to insure that the various productions don't interfere with each other. (3) A purely heterarchical control structure can get you in trouble, too.

(g) Winograd's Shrdlu: (1) A central hypothesis of SHRDLU's linguistic ability – that language can only be analysed in a procedural form – seems denied by equally powerful, but more structured systems like ATN's. (2) SHRDLU's deductive system dealt only in a finite, completely defined world. Even within such systems, there are questions that SHRDLU was unable to answer. (3) Successive utterances in SHRDLU are tied only by anaphora – there is no notion of a conversational dialouge.

(h) Minsky's Frames: (1) Minsky's frame system can be reasonably criticized as a lot of hand waving. While he correctly points out that knowledge in any AI system will need to be functionally organized, he fails to support the rest of his conjectures.

(i) Buchanan et. al. DENDRAL: (1) Much effort is required to create the rule base for any class of compounds. (2) Dendral needs to be told which class of compound the sample belongs to. (3) There are commercial systems that perform the same task without elaborate rule structures.

(j) Hewitt et. al. Planner: (1) Unrestrained automatic backtracking, without either escape mechanisms, or methods of passing "reasons for failure" messages, is an inadequate method. (2) Lack of a data-base context mechanism.

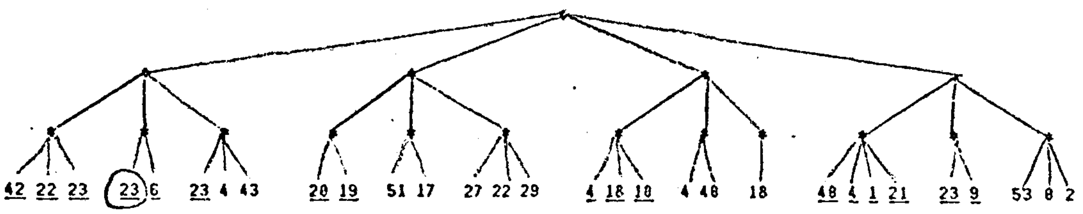
3. (a) A transition network is a set of vertices, connected by arcs (essentially, a finite state automaton). In computational linguistic work, the nodes typically represent states, and the arcs represent grammatical forms which can be employed in going between the states. A transition net can be augmented by hanging "notes" on the arcs of the net. These notes might represent conditions on taking that arc or might indicate additional actions to be performed, such as the storing of some value in a register.

(b) singular or plural, tense, gender, definite or indefinite, number, verb voice.

(c) *Time flies like an arrow*

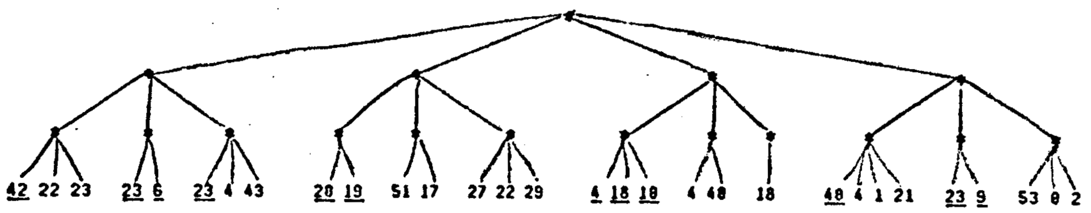
Time	noun,verb
flies	noun,verb
like	verb,preposition
an	determiner
arrow	noun

4. (a)



(b) A program that generated the best (or better) moves first would find greater profit in the alpha-beta technique.

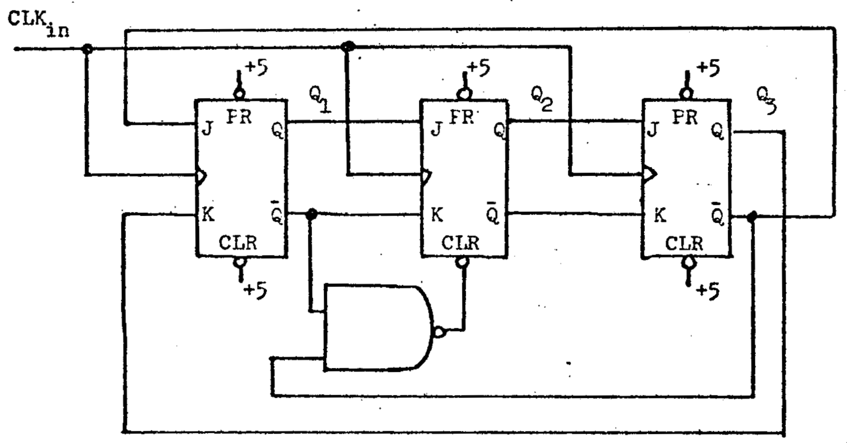
(c)

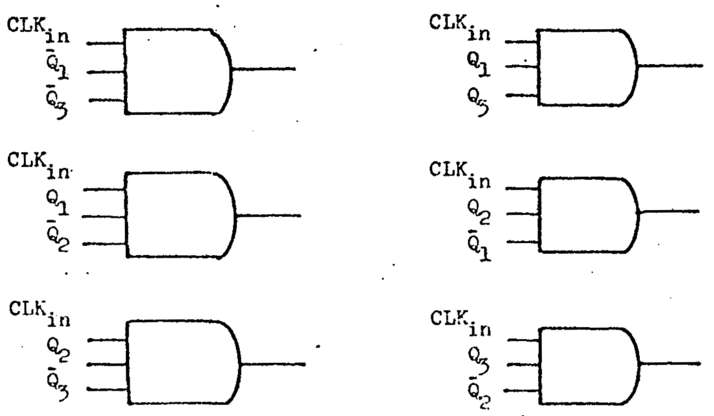


If the program is able to make a good estimate of the correct range of the eventual terminal node, then some searching can be avoided. In this case, guessing that the correct node lies in the 20–25 range permits one to skip searching after the 42 and 40 branches above.

Hardware

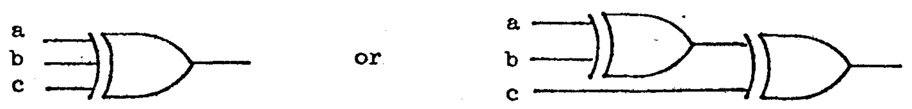
1. Possible solutions: (i) binary count and decode, (ii) 6-bit shift register, (iii) 3-bit switch-tail counter. Solution (iii) is most interesting because it is easier to decode than (i) and allows easy self-correction, unlike (ii). The circuit is shown below.



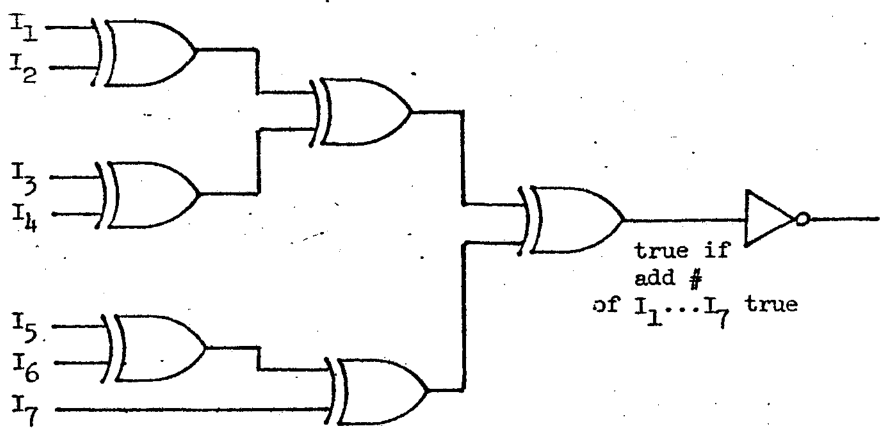


legal states	illegal
000	101
100	010
110	
111	
011	
001	

2.



3.



4. CCD = charge coupled device
FIFO = first in first out
LSTTL = low power schottky transistor transistor logic
CMOS = complementary metal oxide semiconductor
ECL = emitter coupled logic
SOS = silicon on sapphire
UART = universal asynchronous receiver transmitter
PROM = programmable read only memory

5. Microprogramming: lower development time and cost, easier to debug, easier to field modify, fewer parts (therefore cheaper and more reliable).

Hardwired: faster, requires less learning time on small projects, less dependent on single manufacturer.

Use hardwired control for high performance, or for applications that require very few chips. Use microprogramming where speed is not critical, and cost is.

6. One possibility is to have 64K program space (PROM) and 64K data space (RAM). However, this assumes that all programs are in PROM and all data in RAM, which is unlikely to be true.

A better solution is to break the memory into a number of pages (approximately 2K bytes each). This requires hardware to translate 16-bit processor addresses to 20-bit memory bus addresses using a page table. We would also have to implement I/O or memory operations to modify the page table. (Careful design could incorporate this with a virtual memory to allow larger apparent memories.) Page switching will be done under software control. One way to do this is to have an OS in PROM, which is a nice technique for multiprogramming, but still restricts individual programs to 64K addresses. Allowing programs to do their own switching would allow full access, but at the cost of more demands on user software and less protection in a multiprogrammed environment.

7. There are many possible solutions to this problem, depending on the design of the various components and the bus between them.

One possible control word design is given below.

R_4	R_3 R_2 R_1 R_0	D	A	P_1 P_0	X

$R_3 \ldots R_0$	register number or low order 4 bits of instruction code
D	direction: 1 = write to register 0 = read from register or immediate
A	1 = ALU on other end 0 = shifter on other end

P_1P_0

	A = 1	A = 0
0	ALU output port	shifter output port
1	ALU input port 1	shifter input port
2	ALU input port 2	
3	ALU instruction port	shifter instruction port

X	0 = transfer register ($R_3 \ldots R_0$) to/from bus 1 = transfer $R_4 \ldots R_0$ immediate to bus

Example: $R7 := R1 + R14$

	R				D	A	P		X
0	0	0	0	1	0	1	0	1	0
0	1	1	1	0	0	1	1	0	0
code for add					0	1	1	1	1
0	0	1	1	1	1	1	0	0	0

Numerical Analysis

1. (a) In the bisection method we start with an interval $[a_0, b_0]$ such that $f(a_0)$ and $f(b_0)$ have opposite signs. At each stage, we bisect the interval by $m_i = (a_i+b_i)/2$ and choose the half of the interval which retains the change of sign between the endpoints. We continue until the length of the interval is sufficiently small.

In the Newton-Raphson method we start with an initial approximation x_0 which should be close to the root. Then we use the iteration formula

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

until $x_n - x_{n+1}$ is sufficiently small.

In the secant method we start with x_0 and x_1 close to the root, and use the iteration formula

$$x_{n+1} = x_n - \frac{f(x_n)(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}\ .$$

The bisection method always works but is slow (order of convergence = 1). The Newton-Raphson and secant methods are faster (orders of convergence 2 and 1.618, respectively) if x_0 and x_1 are close to the root, but will sometimes fail to converge. Although Newton-Raphson has a higher order of convergence, it requires evaluation of both the function and its derivative at each step (for an effective order of $\sqrt{2}$) and so may sometimes be less efficient than the secant method which requires only one function evaluation per step.

(b) See Winter 1975 exam, problem 8.

2. See Winter 1974 exam, problem 2.

3. (a) Since $A + \delta A$ is singular there exists a nonzero vector x such that $(A + \delta A)x = 0$. Hence $Ax = -\delta Ax$ or $x = -A^{-1}\delta Ax$ so that $\|x\| \le \|A^{-1}\|\,\|\delta A\|\,\|x\|$. Since $\|x\| > 0$, we get $1 \le \|A^{-1}\|\,\|\delta A\|$. Therefore $\mu(A) = \|A\|\,\|A^{-1}\| \ge \|A\| / \|\delta A\|$.

(b) Letting $x = A^{-1}y$ and $\delta A = -yx^T/x^Tx$, we have $(A + \delta A)x = Ax + \delta Ax = AA^{-1}y - yx^Tx/x^Tx = y - y = 0$, so $A + \delta A$ is indeed singular.

We have

$$\|\delta A\|_2 = \max_z \frac{\|\delta Az\|_2}{\|z\|_2} = \max_z \frac{\|yx^Tz\|_2}{x^Tx\,\|z\|_2} = \frac{\|y\|_2}{x^Tx} \max_z \frac{|x^Tz|}{\|z\|_2} = \frac{\|y\|_2}{\|x\|_2},$$

and since $\|A^{-1}\|_2\|y\|_2 = \|A^{-1}y\|_2$, we get

$$\|\delta A\|_2 = \frac{\|A^{-1}y\|_2}{\|x\|_2\|A^{-1}\|_2} = \frac{1}{\|A^{-1}\|_2} \quad \text{so that} \quad \mu_2(A) = \frac{\|A\|_2}{\|\delta A\|_2}.$$

(c) We choose $\delta A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -\epsilon & -\epsilon \\ 0 & -\epsilon & -\epsilon \end{bmatrix}$ so that $A + \delta A = \begin{bmatrix} 1 & -1 & 1 \\ -1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$, which is clearly singular.

Then $\|A\|_\infty$ = max row sum of absolute values = 3, and similarly $\|\delta A\|_\infty = 2|\epsilon|$. Therefore

$$\mu_\infty(A) \ge \frac{3}{2|\epsilon|} = \frac{1.5}{|\epsilon|}.$$

4. (a)(c) See Spring 1975 exam, numerical analysis, problem 2.

(b) In backward error analysis we accept the calculated solution of $Ax = b$ and ask what perturbed matrix $A + \delta A$ or perturbed vector $b + \delta b$ would give this solution. In this way we can obtain bounds on $\|\delta A\|$ and $\|\delta b\|$.

Systems

1. (a) The grammar is unambiguous because the nested $b \ldots e$ blocks can be parsed in only one way and the "statement lists" SL are always constructed from left to right.

(b) After seeing ba with look ahead character ; , it is possible to reduce to bS with look ahead character ; . Now if an e follows the semicolon, S should be reduced to SL, but if an a follows the semicolon this is wrong. Therefore, it is not possible to parse from left to right with a single lookahead character.

(c) Yes, the grammar is LR(2) because in the above situation we can see if an e or an a follows the ; . This is the only real parsing decision made for this grammar.

2. Name is used in the lookup routine to check for multiple accesses to the same identifier.

Level is used to determine the static naming level, which specifies the run time stack frame (activation record) in which the variable is stored.

Offset is the location within that stack frame where the variable is stored.

Type is the declared type of the variable, used to determine if the variable matches on assignment and is proper for operations that require a specific type.

Direct or indirect specify if the address of the variable contains the value of the variable or a pointer to the value of the variable. The latter case is used for reference parameters.

3. (a) loop unrolling (b) common subexpression elimination (c) strength reduction

4. (a) Similarity: Can't tell what will be useful in later computations.

Differences: In a compiler it is feasible to implement true LRU (least recently used), but not in a virtual memory system. Also, the number of registers is fixed in the compilation problem, but the page working set varies.

(b) In a multipass compiler you can look ahead and determine what will be used in the future as well as when it will be used.

5. (a) When you request the resource, the handler for the resource looks you up in the table for the resource. If you are a legitimate user, you get further access to the device without further protection checks.

(b) Add an "ownership" capability to the scheme and consider the capability a resource. You may only pass on a capability (like read permission) if you own the capability. Your trusted friend gets read pemission but not ownership.

6. (a) The load table indicates, for each word in the linked module, whether or not that word needs relocation (i.e. whether or not the word is a memory address). A bit vector (one bit per word) is a good data structure for a relocation map.

(b) The table would indicate all locations which require filling in at link time, and what they link to. This table is usually stored in the form of singly linked lists threaded through the code.

Theory of Computation

1. The language is not regular. Suppose L could be recognized by a finite state machine with n states. Consider $y = 1\underbrace{000\ldots0}_{n}1$ and $x = y^2 = 01\underbrace{00\ldots0}_{n-1}1\underbrace{00\ldots0}_{n+1}1$. The input word w formed by (x, y) begins $011\underbrace{000\ldots\ldots0}_{n-1+\lfloor\frac{n-1}{2}\rfloor}1\ldots\ldots$, where the block of 0's has length $\geq n$ (assuming $n \geq 3$). Therefore, the FSM must loop while reading this block and so must also accept a word w' formed by adding $3p$ zeros to this block of w. But w' corresponds to $y' = 1\underbrace{00\ldots0}_{n+p}1$ and $x' = 01\underbrace{00\ldots0}_{n-1+2p}1\underbrace{00\ldots0}_{n+1}1$, which do not satisfy $x' = (y')^2$, a contradiction.

2. (1) True. Let T_1 and T_2 accept the two r.e. sets. For any input word w, let T_3 first simulate T_1 on w. If T_1 accepts, then T_3 simulates T_2 on w. Thus T_3 accepts exactly those words in $L(T_1) \cap L(T_2)$.

(2) True. We reduce the blank-tape halting problem to the given problem. For any Turing machine T, define T' such that on any input beginning with a 1, T' erases it and then simulates T on a blank tape. It does not matter what T' does on inputs beginning with a 0; let us say that it loops forever. Then T halts on a blank tape if and only if T' halts (and accepts) at least one word that begins with a 1. Therefore if we had an algorithm to decide the latter problem, we could solve the blank-tape halting problem, a contradiction.

(3) False. The language $L = \{ 0^n 1^{3n} \mid n \geq 1 \}$ is context free, but $Transpose(L) = \{ 1^{2n} 0^n 1^n \mid n \geq 1 \}$ is not context free.

3. One possible correct (and brief) proof is the following:

1]	$\sim(\sim A*B)*(B*C)$	from P
2]	$\sim(\sim D* \sim(\sim A*B))* \sim(\sim A*B)*(B*C)$	from P, with $A \leftarrow D$, $B \leftarrow \sim(\sim A*B)$ and $C \leftarrow B*C$
3]	$\sim D * \sim(\sim A*B)$	R 1, 2
4]	$\sim(\sim E*\sim D) * (\sim D* \sim(\sim A*B))$	from P, with $A \leftarrow \sim E$, $B \leftarrow \sim D$, and $C \leftarrow \sim(\sim A*B)$
5]	$\sim E*\sim D$	R 3, 4

We have now succeed in proving that every WFF of the form $\sim E*\sim D$ is a theorem. Hence, $\sim E * \sim(\sim E*\sim D)$ is also a theorem.

6]	$\sim E * \sim(\sim E*\sim D)$	from 5, with $E \leftarrow E$ and $D \leftarrow \sim E*\sim D$
7]	E	R 5, 6

4. The program fragment "reverses" the array a from element 1 to element n. An appropriate invariant is:

$$\forall j \; (\; (1 \leq j \leq n) \supset a[j] = a_0[\textbf{if } j > \min(i, n-i+1) \textbf{ then } j \textbf{ else } n-j+1] \;)$$

5. (a) A suitable program is:

```
       i:=1; j:=n;
loop:  if i>j then go to done;
       if a[i]<c then begin i:=i+1; go to loop end;
       if a[j]≥c then begin j:=j-1; go to loop end;
       t := a[i];
       a[i] := a[j];
       a[j] := a[i];
       i:=i+1; j:=j-1;
       go to loop;
done:
```

(b) Letting $a_0[1{:}n]$ be the initial array, one form of the correctness statement is:

$$permutes(a, a_0) \wedge \forall k, l \; (\; (1 \leq k, l \leq n \wedge a[k]<c \wedge a[l]>c) \supset k<l \;)$$

(c) A suitable invariant (attached to the label *loop*) is:

$$permutes(a, a_0) \wedge \forall k, l \; (\; (1 \leq k<i \wedge j<l \leq n) \supset (a[k]<c \wedge a[l]>c) \;)$$

Comprehensive Examination Reading List

ALGORITHMS AND DATA STRUCTURES

Aho, A. V., Hopcroft, J. E., and Ullman, J. D., *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Massachusetts, 1974, Chapters 2, 3, 4.1 - 4.4. [Spr76 -]

Knuth, D. E., *The Art of Computer Programming*, Volume 1, Addison-Wesley, Reading, Massachusetts, 1968, Chapter 2 (except Section 2.3.4). [Spr72 -]

Nievergelt, J., Reingold, E. M., and Farrar, J., *Computer Approaches to Mathematical Problems*, Prentice-Hall, 1974, sections 2.1, 3.3. [Win75 - Spr75]

ARTIFICIAL INTELLIGENCE

Bobrow, D. and Raphael, B., "New Programming Languages for AI Research," *Computing Surveys*, Volume 6, Number 3, September 1974. [Spr75 -]

Feigenbaum, E. A., *Artificial Intelligence Research*, available in Computer Science Library. [Spr75 - Win77]

Jackson, P. C., *Introduction to Artificial Intelligence*, Petrocelli Books, New York, 1974, pages 1-4, Chapters 3-7. [Win75 - Win78]

Newell, A. and Simon, H. A., *Human Problem Solving*, Prentice-Hall, Englewood Cliffs, N. J., Chapters 3, 4, 8. [Spr73]

Nilsson, N. J., "Artificial Intelligence," in Rosenfeld, Jack L., ed., *Technological and Scientific Applications; Applications in the Social Sciences and the Humanities, Information Processing 74: Proceedings of IFIP Congress 74*, Volume 4, American Elsevier Publishing Company, Inc., New York, 1974, pages 778-801. [Spr75 -]

Nilsson, N., *Problem Solving Methods in Artificial Intelligence*, McGraw-Hill, New York, Chapters 1-6. [Spr72 - Win77, Spr78 -]

Reddy, D. Raj, "Speech Recognition by Machine: A Review", in *Proceedings of the IEEE*, April 1976, pages 501-531. [Spr78 -]

Winston, P. E., *Artificial Intelligence*, Addison-Wesley, Reading, Mass., 1977. [Win78 -]

CULTURE

Goldstine, H. H., *The Computer from Pascal to von Neuman*, Princeton University Press, Princeton, New Jersey. [Spr76 -]

Randall, B., "The History of Digital Computers," Computing Laboratory Technical Report, University of Newcastle upon Tyne, 1974. [Spr76 -]

HARDWARE

Bartee, T. C., *Digital Computer Fundamentals*, 3rd ed., McGraw-Hill, New York, 1972, Chapters 3, 4, 5.1-5.21, 7.1-7.10, 7.12-7.16, 7.20 and 9. [Spr73 - Spr77]

Booth, T. L., *Digital Networks and Computer Systems*, John Wiley and Sons, New York, Chapters 2-5, 6.1-2, 7-10. [Spr72 - Spr76]

Mano, M., *Computer System Architecture*, Prentice-Hall, Englewood Cliffs, New Jersey, 1976, Chapters 1-5, 7, 8, 11.1, 11.2, 11.5, and 12. [Win77 -]

Gschwind, Hans W. and McCluskey, Edward J., *Design of Digital Computers*, Springer-Verlag, New York, 1975, Chapters 2, 3, 5, 6, 7, 8.2, 8.3 (except 8.3.5.1), 8.4. [Win78 -]

NUMERICAL ANALYSIS

Conte, S. D. and De Boor, C., *Elementary Numerical Analysis: An Algorithmic Approach*, McGraw-Hill, New York, Chapters 1, 2, 4, 5, 6. [Spr74 -]

Dahlquist, G. and Bjorck, A., *Numerical Methods*, Prentice-Hall, Englewood Cliffs, New Jersey, 1974, Chapters 1-3, 4.1, 4.3, 4.4, 4.6, 6, 7.1, and 7.3. [Spr76 -]

Forsythe, G. E., Malcolm, M. A., and Moler, C. B., *Computer Methods for Mathematical Computations*, Prentice-Hall, 1977, Chapters 2, 4.4, and 4.5. [Win73 -]

Forsythe, G. E. and Moler, C. B., *Computer Solution of Linear Algebraic Systems*, Prentice-Hall, 1967. [Spr76 -]

Henrici, P., *Elements of Numerical Analysis*, John Wiley and Sons, New York, Chapters 1-4, 6, 9-14. [Spr72 - Win75]

Herriot, J., *Course notes* on spline functions and economization of power series. [Spr72]

PROGRAMMING LANGUAGES AND SYSTEMS

Aho, A. V. and Ullman, J. D., *Principles of Compiler Design*, Wiley, New York, 1975. [Spr78 -]

Brinch Hansen, Per, *Operating System Principles*, Prentice-Hall, 1973. [Spr77 -]

Dahl, O.-J., Dijkstra, E. W., and Hoare, C. A. R., *Structured Programming*, Academic Press, New York, 1972. [Spr74 -]

Denning, Peter, "3rd Generation Computer Systems", *Computing Surveys*, vol. 3, no. 4, December 1971, pages 175-216. [Spr73]

Denning, Peter, "Virtual Memory," *Computing Surveys*, September, 1970. [Spr77 -]

Donovan, J., *Systems Programming*, McGraw-Hill, New York, 1972, Chapters 4, 5, 8, 9. [Spr72 - Spr74]

Gear, C. William, *Computer Organization and Programming*, McGraw-Hill, New York, 1973, Chapter 9. [Spr73 - Win78]

Graham, Robert, *Principles of Systems Programming*, Addison-Wesley, Reading, Massachusetts, 1975. [Spr78 -]

Gries, David, *Compiler Construction for Digital Computers*, John Wiley and Sons, New York, 1971, Chapters 2, 4, 5, 8, 11, 12, 15, 16, 17, 21. [Spr73 - Win78]

Shaw, A. C., *The Logical Design of Operating Systems*, Prentice-Hall, Englewood Cliffs, N. J., 1974. [Spr76 - Win77]

Stone, H. S., *Introduction to Computer Organization and Data Structures*, McGraw-Hill, New York, 1972, all except 11.2.3. [Spr72 -]

Watson, Richard, *Timesharing System Design Concepts*, McGraw-Hill, 1970, Chapters 2, 4, 5. [Win75 -]

THEORY OF COMPUTATION

Aho, A. V., Hopcroft, J. E., and Ullman, J. D., *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Massachusetts, 1974, Chapter 10. [Spr77 -]

Enderton, Herbert, *A Mathematical Introduction to Logic*, Academic Press, 1973, Chapters 1, 2. [Win75 -]

Hopcroft, J. and Ullman, J., *Formal Languages and Their Relation to Automata*, Addison-Wesley, Reading, Massachusetts, Chapters 1-7. [Spr72 -]

Manna, Zohar, *Introduction to Mathematical Theory of Computation*, McGraw-Hill, 1973, Chapters 1, 2, 3. [Win73 -]

Mendelson, E., *Introduction to Mathematical Logic*, Van Nostrand, Chapters 1, 2. [Spr72 -]

Robbin, J. W., *Mathematical Logic*, Benjamin, Chapters 1, 2. [Spr72 - Win73]

PROGRAMMING PROBLEM

Kernighan, B. W. and Plauger, P. J., *The Elements of Programming Style*, McGraw-Hill, New York, 1974. [Spr76 -]

Index

Magic Number ______________

Spring 1977/78 Computer Science Comprehensive Exam
Part 1 -- Written Exam
May 20, 1978 (9:00 - 12:00; 1:30 - 4:30)
Polya Hall, Room 111.

READ THIS FIRST!

1. The exam contains questions drawn from six areas of computer science The total possible score is 360 points, 60 in each area. Hint: 6 hours equals 360 minutes, this may help you plan your time.

2. Please do your best to relax during the lunch break. You may not consult any references or colleagues or write drafts of answers during this period. Just relax.

3. Write your exam number in the upper right-hand corner of each page. Be sure that you have all 39 pages of the exam.

4. Strategic considerations: (a) To pass this exam at the Ph.D. level, you should not leave any of the six subject areas completely blank, as there will be a minimum competence requirement of roughly 20 points in each area. The total scores of everybody who passes this minimum requirement will then be used to determine whether or not the written exam as a whole is passed. Therefore your policy should be first to establish a competency in each area, then to maximize your total score. (b) To pass this exam at the Masters' or CS Minor level, simply try to maximize your total score.

5. Please write legibly, with a pen or sharp pencil. If you need extra space, insert additional sheets; remember to follow instructions 3 and 5 on these sheets. Paper will be available.

6. Show your work, as partial credit will be given for incomplete answers. (But don't show it on the back of a page.)

7. This exam is open book: You may use whatever books and notes you have already brought with you and any library books provided by the committee.

8. Sign the honor code statement blow. (This page will be removed from your exam during the grading process.)

9. The committee suggests that you read over the entire exam quickly once, in order to help in allocating your time. We also suggest that you refrain from panic. GOOD LUCK.

10. The programming problem will be available at 9 a.m. on Thursday, May 25 in Polya 253. It will be due at noon on Tuesday, May 30 in Polya 253. Preliminary results from the written exam will be available when you pick up the programming problems.

In recognition of and in the spirit of the Honor Code, I certify that I have neither received nor given unpermitted aid on this exam.

Signed ______________________________