Comprehensive Exam — Programming languages

Fall 2000

Problem 1 — Activation records (8 points)
You want to modify your compiler to support dynamically-sized, stack-allocated variables. Assuming a traditional machine architecture, explain what complications this creates compared to the usual constant-size stack allocation, and how you would handle it.

Problem 2 — Execution (12 points)
One of CS's core themes is execution. We package it in various ways: procedure calls, threads, processes, co-routines, upcalls, interrupt handlers, etc.

1. **(6 points)** Ignoring hardware and language details, what do you fundamentally need to "execute" an instruction stream and why? Please pick two of the above abstractions and, at a high level, classify their constituent parts in your categories.

2. **(6 points)** Whenever we have multiple streams of execution, we have to decide what to which stream to run ("scheduling"). What are some differences between thread scheduling and scheduling which procedure to run? What property of procedure scheduling allows us to use a single stack for procedures but (in general) forces multiple stacks for threads?

Problem 3 — GC fun (18 points)
Assume you want to add garbage collection to C. Unfortunately, unlike most GC implementations, you do not have any compiler support. So, instead you do a hack where you treat all memory/register whose values are valid memory addresses as pointers.

1. (12 points) At a high level, explain how to write a (mostly) conservative garbage collector using this trick.

2. (6 points) When will your collector lose storage? Can you give (possibly) contrived examples of where it would reclaim storage that is not dead? Can your collector compact the heap?

Problem 4 — Naming fun (20 points)
Assume you add overloading to C, where there can be multiple functions with the same name but different type signatures and the compiler determines uses argument types to decide which function to call. For example, given the following two functions:

```
int abs(int x);
float abs(float x);
```

the following calls will be resolved to the first and second respectively:

```
int x1;
float x2;

x1 = abs(x1);
x2 = abs(x2);
```

Assume we want to use separate compilation, and linkers only know how to bind a reference to name N to a single definition of name N. (I.e., emitting both implementations of "`abs`" with the same name will cause a "multiple definition" error.)

1. **(12 points)** Explain how to implement overloading using only local analysis given this constraint. You can simplify your scheme by requiring callers and modules that contain definitions to obey *reasonable* restrictions. Make sure to mention how to (1) stop the user from accidently colliding with your strategy; (2) working in the presence of debugging; and (3) support structures as arguments.

2. **(8 points)** What rules should you follow for resolving ambiguities in the presence of argument promotion? (I.e., when an argument of type T can be legally changed to a T'.) Be sure to handle conflicts that can arise with multi-argument functions.