2003 Programming Languages Comp. Solutions

written by 2006 Ph.D. First-Years

1. *Garbage Collection:*

   (a) You could do mark-and-sweep. Follow pointers because you know what their types are at compile-time. No, it cannot find and delete all inaccessible objects; any garbage collector can at most hope to delete all unreachable objects, but not all inaccessible objects.

   (b) You need to implement conservative garbage collection whereby you treat every value that could possibly point to a region in the heap as a pointer.

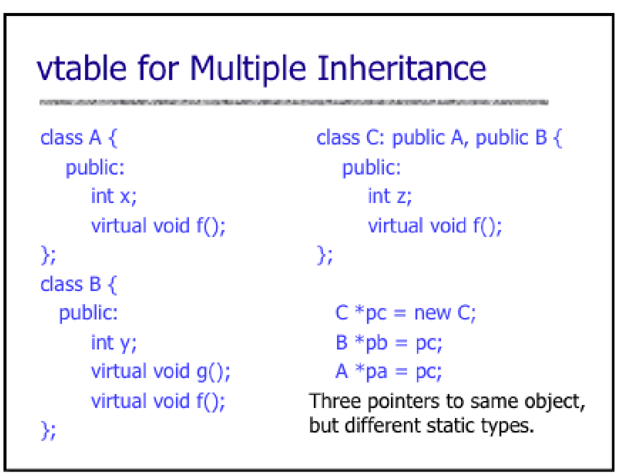
   (c) You need to lock thread-local heaps.

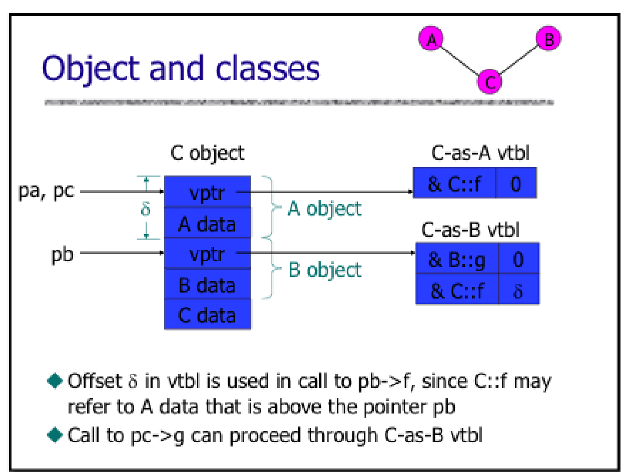
2. *Subtyping:*

   (a) Return types are covariant; function arguments are contravariant. e.g., `circle <: shape`. A function returning a `circle` is a subtype of a function returning a `shape` because if a caller expects a `shape` to be returned, then it's perfectly okay to return a `circle` (covariance). A function taking in a `shape` as an argument is a subtype of a function taking in a `circle` as an argument, because if the caller expects to pass in a `circle`, it's perfectly okay to treat it as simply a `shape` within the function and only use the features that are available in `shape` and ignore the `circle`-specific features (contravariance).

   (b) `ref circle` is not a subtype of `ref shape`, because you can put a `square` in a `ref shape`, but not in a `ref circle`.

3. *Method Lookup:* Deltas are needed in the vtable because the compiler needs to put in a fixed offset in the compiled code when compiling virtual function calls, but when a class inherits from multiple classes, the offset of the function in the subclass will probably be different than the offset in one of the superclasses. The delta is used to go backwards to the beginning of the object so that all fields can be accessed. See below (taken from Prof. John Mitchell's CS242 lecture notes) for an illustrated example.





4. *Implementing Exceptions:*

   (a) The pointer that refers to the base pointer of the next activation record up in the stack (the one for the current method's caller). This is called the *control link* in CS242 lecture notes.

   (b) No, because when an exception is raised at run-time, the activation records on the stack can be different depending on what series of method calls led up to that exception. There is no way for a compiler to know at compile-time what records are going to be on the stack when an exception is raised, and thus how many pointers to follow.

   (c) It's difficult to optimize to re-arrange instructions around side effects in general, and an exception is a particularly nasty side-effect.

   (d) Whether that function is allowed to throw exceptions.

   (e) Yes, only for checked exceptions, though. Unchecked exceptions do not need to be declared in the method declaration.