Comprehensive Exam: Programming Languages Autumn 2002

1. (7 points) *Type Inference*

   (a) (3 points) This part of the question asks you to write the ML type of an ML function. Recall that an ML function that takes two integer arguments and returns a Boolean value has type `(int * int) -> bool`. What is the ML type for the following function?

   ```
   fun applytwice(f, x) = f(f(x));
   ```

   Answer: `('a -> 'a) * 'a -> 'a`

   (b) (2 points) Describe types `<type1>`, `<type2>` and `<type3>` that will make the analogous C code below statically typed. You may write the types using correct or mostly correct C syntax. In particular you can write `<type2> f` in a form where `f` appears in the middle of the type, if you prefer (and if this is correct).

   ```
   <type1> applytwice(<type2> f, <type3> x) {
             return f(f(x));
   };
   ```

   Answer: One possibility is `<type1>=<type3>=int` and `<type2>`= int (*) (int). Some prefer to write `int (* f)(int)`

   (c) (2 points) Why is the ML type more useful than the C type for `applytwice`? Specifically, what flexibility does the ML type system give, for functions with the kind of type you wrote in (a), that the C type system does not give you for the function in (b)?

   Answer: The ML function is polymorphic, which means that you can apply `applytwice` to many types of arguments. In contrast, the C function can only be declared and used for one specific choice of types.

2. (5 points) *ALGOL 60 Pass-By-Name*

   In pass-by-name, the result of a procedure call is the same as if the formal parameters were substituted into the body of the procedure. This rule for defining the result of a procedure call by copying the procedure and substituting for the formal parameters is called the *Algol 60 copy rule*. While the copy rule works well for pure functional programs, as illustrated by β-reduction in lambda calculus, the interaction with side effects to the formal parameter may cause unexpected results.

   The following Algol 60 code declares a procedure `P` with one pass-by-name integer parameter. The line `integer x` does not declare local variables – this is just Algol 60 syntax declaring the type of the procedure parameter.

```
begin
   integer i;
   integer array A[1:2];

   procedure P(x);
      integer x;
      begin
         i := x;
         x := i
      end

   i := 1;
   A[1] := 2; A[2] := 3;
   P (A[i]);
   print (i, A[1], A[2])
end
```

(a) (2 points) Explain how the procedure call `P(A[i])` changes the values of `i` and `A` by writing the lines of Algol that you get by substituting the actual parameter for the formal parameter in `P(x)`.

Answer: The program is equivalent to the following one, obtained by replacing the call with a copy of the procedure body in which each occurrence of the formal parameter `x` is replaced by the actual paramter `A[i]`:

```
begin
   integer i;
   integer array A[1:2];
   i := 1;
   A[1] := 2; A[2] := 3;
   begin
      i := A[i];
      A[i] := i
   end;
   print (i, A[1], A[2])
end
```

(b) (3 points) What integer values are printed by the program using pass-by-name parameter passing?

Answer: This block resulting from the call sets `i` to 2 and then sets `A[2]` to 2, so the output is 2, 2, 2.

3. (6 points) *Control Flow and Memory Management.* An *exception* is a command that aborts part of a computation and transfers control to a handler that was established at some earlier point in the computation.

   (a) (3 points) Why is it easier to write programs with exceptions in a language that provides garbage collection than a language where all data on the heap is explicitly manipulated by the program?

   Answer: An exception might casue a program to bypass the "normal" code that deallocated memory and cleans up after a portion of the computation. After an exception, deallocation would have to be performed by the exception handler. However, when the only pointers to garbage are pointers that were in activation records that were eliminated as a result of the exception, it will be impossible for the handler to explicitly deallocate unreachable memory.

   (b) (3 points) In a concurrent programming language, we have two options:
   (i) raising an exception only affects the current thread, or
   (ii) raising an exception aborts all threads that were spawned below the control point associated with the exception handler.

   Which option would be easier to implement? Which would be more convenient to use? Explain briefly. (This question is only worth 3 points, so do not write more than 3 or 4 sentences.)

   Answer: Option (i) is easier to implement but option (ii) may be more convenient in some cases. In general, if we are doing some parallel computation and an error occurs in one branch, then it may make most sense to abort the entire computation. But it is hard to cleanly abort all the subprocesses that might be running on different processors.

4. (12 points) *Method Lookup.*
   Smalltalk and C++ use different implementations of method lookup for objects.

   (a) (4 points) If a Smalltalk program sends message `m` to object `x`, how is the code implementing `m` located? Describe the main data structure and briefly explain how it is used.

   Answer: The method code for `m` is located by searching the method dictionary of the class of `x`, then any superclass of the class of `x`, then the method dictionary of any superclass of any superclass of the class of `x`, and so on. A Smalltalk *method dictionary* generally provides an association between method names (selectors) and method code, together with a pointer to any superclass method dictionaries.

   (b) (4 points) If a C++ program calls virtual member function `f` of object `x`, how is the code for `f` located? Describe the main data structure and briefly explain how it is used.

Answer: The main data structure is the *virtual function table* or vtable. The vtable is a structure of pointers to function bodies. The offset of a virtual member function is calculated at compile time, so that at run time it is only necessary to follow the pointer located by adding the vtable address and the offset.

(c) (4 points) Describe the trade-off's between the two approaches. Which is more flexible? Which is more efficient? Your answer should be 3–5 sentences.

Answer: The C++ mechanism is generally more efficient, but it requires compile-time type information. Smalltalk is a more flexible language, since it doesn't have compile-time type checking, but method lookup is less efficient. (Caching can be used to speed up the Smalltalk lookup, however.) Another factor is that if a class is changed, a C++ derived class or client program may need to be recompiled, since the compile-time-computed offsets may change. However, no such recompilation is needed in Smalltalk since the offsets are determined at run time.