PL 1994 qual

1. Implementation of parameter passing
   a. Pass-by-value: code generator needs to know the size of the parameters to emit code which, when called, will grow the call stack the proper amount. Type effects code output.
      Pass-by-reference: code generator needs the number of reference parameters, and since all pointers are the same size, it doesn't need the type info. However, the code emitted is based on type, so the code generator phase of the compiler needs to know the types of the arguments in order to generate the right instructions.
   b. Both can be used.
      Integer: value – copy onto stack; reference – multiple return values
      Record: value – lots of copying; reference – efficient
   c. Parameters of the function passed in are stored in that functions symbol table and are not used by the function it is passed to. Return type is important.

2. Design of C++
   a. Imperfect: memory leaks, harder to program, illegal references, run out of memory.
      Practical: efficiency, may not need garbage collection, pointers look like integers, C didn't have it.
   b. Imperfect: assignment works poorly, direct manipulation breaks polymorphism, dangling pointers
      Practical: efficient, easier, memory management is cleaner, destructor automatic – good for exceptions
   c. Imperfect: reduced flexibility, can't redefine non-virtual
      Practical: efficient, C compatible, don't pay for dynamic lookup if it isn't used

3. Subtyping and inheritance
   a. Yes. Assume f:int → int, g:real → int. Since int is a subtype of real, g is a subtype of f.
   b. Class c2 can be used anywhere class c1 can. Class c2 has all the data members of class c1.
   c. No inheritance. In C++, subtyping must come via public inheritance.
   d. Class c2: public c1 {
      public:
      virtual short h();
      }
   e. from typecast, c2 is a c1 to the type checker. Layout of c2 and c1 are the same, so the offsets for c1 are valid in c2.
   f. Break it by rearranging the order of the virtual functions.
   g. Class d2 must list members in the same order as class d1, with new stuff at the bottom. Types of d2 may be subtypes of corresponding types in d1.
   h. Smalltalk subtyping is implicit. If two classes have the same contents, they are considered subtypes. In C++ subtyping must be explicit through public inheritance.
   i. According to the definition of subtype given in the problem, it is now a subtype. C++ would not recognize it, but Smalltalk would. It is a subtype because D is a subtype of B.

4. Polymorphism and representation of data
   a. ML avoids code replication through pattern matching and its runtime type specification system. C++ doesn't support this for efficiency. Lookup is slow. C++ can use compile time optimizations based on type.
   b. Vtable – dynamic method lookup; overloading – sort of polymorphic
   c. Smalltalk – inheritance; no types: a method can work with any object
      C++ - templates, overloading, inheritance
      ML – runtime type specification, pattern matching