CORE EXAMINATION, PRAGMATIC SECTION: SOLUTIONS
Department of Computer Science
New York University
September 23, 2005

Programming Languages and Compilers

PL&C: Question 1

A. Define the term "polymorphism" as used in programming languages.

Answer: Polymorphism, meaning "many shaped", refers to the ability of a single entity (such as a procedure or function) to take on many types dynamically. For example, a polymorphic function in ML whose type is 'a list → 'a can, for each type 'a taken from the infinite set of types, take a list of type 'a and return a value of type 'a. As another example, in an object oriented language a procedure expecting a parameter of type A will accept an object of type A or any type derived (directly or indirectly) from A.

B. Describe the difference in polymorphism between i) subtyping due to class derivation in an object-oriented language and b) interfaces as in Java.

Answer: The difference between these two situations arises from the range of types over which the formal parameter to a procedure can vary. In the first case (subtyping due to class derivation), the formal parameter can range over all types derived from the declared parameter type, as well as the declared parameter type themselves. That is, the set of possible types for the argument is constrained by the subtyping relationship. For example, if a procedure is declared as

```
void f(A a) { ... }
```

where A is a class, then the actual parameters that can be passed to f must be of type A or any class that is a descendant of A.

In the second case (interfaces), the parameter to a procedure can range over any type that implements the required methods of the interface (and, in Java, declares itself to have implemented the interface). There is no requirement that the range of types over which a parameter can vary be related by subtyping.

C. In ML, write a polymorphic function MINLIST that takes two parameters:

- a list L
- a function LESS such that LESS(A,B) returns true of A is less than B

and which, by using LESS, returns the minimum value of L.

Answer:

```
fun MINLIST [x] less_than = x
 |  MINLIST (x::y::xs) less_than =
           if less_than x y then MINLIST (x::xs) less_than
                            else MINLIST (y::xs) less_than
```

D. What is the type of MINLIST?

Answer:

```
'a list -> ('a -> 'a -> bool) -> 'a
```

E. Write an example of a call to MINLIST where the first parameter, L, is not simply a list of integers.

Answer:

```
let
    fun less_length L1 L2 = if length L1 < length L2 then true else false
in
    MINLIST [[1,2],[3,4,5],[6],[7,8,9]] less_length
end
```

F. Write, in Java, a polymorphic procedure MINARRAY (analogous to MINLIST above), which returns the minimum value of an array of objects where the type of the objects supports the LESS operation.

Answer:

```
interface LT {
  bool LESS(Object o);
}

LT MINARRAY(LT[] a) {
  if (a.length == 1)
     return a[0];
  else {
     LT min = a[0];
     for(int i=1; i<a.length;i++)
        if (a[i].LESS(min))  min = a[i];
     return min;
  }
}
```

PL&C: Question 2

A. What does it mean for a compiler optimization to be a "local optimization".

Answer: The scope of the optimization is within a basic block of the flow graph.

B. Give an example of local optimization (other than copy propagation), describe the benefits of the optimization, and give a high level description of an algorithm for performing that optimization.

Answer: See the Dragon book, Section 9.4 for a description of optimizations applied to basic blocks.

C. Write, in pseudo-code, a detailed algorithm for performing local copy propagation.

Answer:

See the dragon book, section 10.7. Although this section covers global copy propagation, the algorithm for computing gen and kill sets in the global optimization is essentially the local optimization.

PL&C: Question 3

Consider the following C++ declarations:

```
class C { virtual void proc (int x) {..} ..  };
..
C obj;
C* ptr;
..
obj.proc (5);          // (1)
ptr -> proc (5);       // (2)
```

A. Explain briefly how the code generated for (2) differs from that generated for (1). Estimate how many more machine instructions must be executed.

Answer:

The code for (1) is a simple call. In this case there will be one instruction to push each argument on the stack (the explicit argument 5, and the object itself), and one call instruction.

The code for (2) is a dispatching call. The operation to be called is determined at run-time, by examining the dispatch table (or vtable) of the class of the object denoted by ptr. Every instance of a class has a pointer to the dispatch table, and each operation has a statically known position in that table. Note that all classes derived from C, have a pointer to their version of proc at the same location in the dispatch table. So the call in (2) requires one additional indirect load, to retrieve the address of the operation to be called, followed by an indirect call through that address. This adds at least one more instruction to the call. This cost is independent of the number of subclasses of C.

B. Consider now the following Java declaration:

```
interface Itf {void update (int x)..};
```

Why is the following declaration illegal?

```
Itf thing = new Itf();
```

Answer: An interface describes a series of operations with no implementation, and an interface includes no data. Therefore it makes no sense to create an instance of an interface.

Is there some equivalent to the interface construct in other object-oritented languages?

Answer: An interface is closely related to an abstract class with no data members (in C++ parlance) or to an abstract tagged null record (in Ada jargon).

What values can legally be used to assign to variable thing?

Answer: A reference to an instance of any class that implements the interface.

C. Given the method declaration:

```
void tryOut (Itf thing1) {...};
```

In the body of this method, discuss the run-time mechanism, i.e. generated code and data-structures, that might be needed to execute:

```
thing1.update (10);    //  (3)
```

Answer: An interface dispatching call is more complex than a regular dispatching call, because different classes that implement an interface do not necessarily have the interface operation at the same position in their dispatch table. The simplest approach, used in early implementations of Java, is to locate the operation by name, which has a cost proportional to the number of operations in the table. There are more efficient implementations that involve multiple dispatch tables (one per implemented interface) and therefore gain performance at a modest cost in space. Each interface table is a permutation of the primary dispatch table. The following steps are needed to implement an interface call::

a) Retrieve class information from pointer in object. (one dereference)
b) Locate interface table (indexing at fixed position).
c) Retrieve position of operation from entry in interface table (indexing)
d) Perform indirect call.

This is a constant overhead, i.e. independent of the number of interfaces implemented by a class, or the number of operations defined in the class.

OS: Question 1

Consider an operating system in which files are implemented using i-nodes, the free list is implemented as a bit vector, and blocks are 2 KBytes. At a given point in time, the first few blocks on the free list are 8, 12, 15, 19, 21, 30. At this point, a user process creates a new file /D/F, writes 3 KBytes of data to it, and closes it. Assume that the number of blocks in D does not need to change.

A. How many new blocks must be allocated? What are they used for?

Answer: 3 new blocks. One for the inode for F, and two for the data.

B. How are these new blocks (or this new block) found?

Answer: The free list will be searched linearly for the first three free blocks. Blocks 8, 12, and 15 will be allocated for use.

C. What changes are made to the free list?

Answer: Bits 8, 12, and 15 set to 0.

D. What changes are made to the directory D?

Answer: An entry is added to D. This entry will certainly contain the name "F" of the new file and the block number of the inode. It may also contain some of the further attributes of the file F. (Which attributes are saved in the directory and which are saved in the inode is a matter of implementation.)

E. How is the file structured, in terms of blocks?

Answer: The inode contains the block numbers of the two data blocks.

F. Name four attributes that would typically be recorded for the new file. How would they be recorded?

Answer: Owner, time of creation, permissions, size (there are others). These are recorded either in the inode or in the directory entry.

G. Suppose that at a later time a user process wants to read byte number 2500 of file /D/F, and that at that time the root directory is in memory, but not the directory D or the file F. What blocks need to be loaded into memory, and how does the operating system find these blocks? (Do not worry about finding the physical location of the block on the disk; just describe how the OS finds the block number.)

Answer:
Look up inode of directory D by name in the root directory.
Load inode of directory D from disk.
Find addresses of data block(s) of D in inode.
Load data blocks of D.

Look up address of inode of file F by name in D.
Load inode of F.
Find address of second data block of F in inode.
Load second data block of F.

OS: Question 2

Virtual memory provides the illusion of memory being larger than that available on the actual hardware, i.e., the physical memory.

A. Mapping virtual to physical addresses

Virtual memory is implemented by mapping virtual addresses, which are used by applications, to physical addresses, which are used to access memory in hardware.

- What is the name of the main data structure used for mapping from virtual to physical memory?

  The *page table*.

- What is the unit of mapping in this data structure?

  The *page*, which, for example, is 4 KB on Windows 2000 and XP.

- What other information besides virtual and physical addresses is typically contained in this data structure?

  The *valid* bit, the *modified* bit, the *protection* bits (read/write/execute).

- How is this other information utilized by an operating system and/or hardware?

  - The valid bit determines whether a virtual page is currently mapped to a physical page.
  - The modified bit determines whether a page has been written to since it was swapped in. If this bit is set, the page has to be written to the swap space/file before swapping it out. Otherwise, it can simply be discarded/overwritten.
  - The protection bits determine whether the data in a page can be read, written, or executed by an application.

B. Protection

Virtual memory is not only used to provide applications with more memory than actually available, but also to isolate applications from each other, i.e., for protection.

- How many mappings between virtual and physical addresses does an operating system maintain?

  One per *address space*, which typically coincides with an application.

C. Sharing

While virtual memory is used to isolate applications from each other, it can also be used for sharing.

- How can virtual memory be used for sharing?

  Through *shared memory*, that is, by mapping pages in different address spaces and at possibly different addresses to the same physical page.

- In fact, which part of an application should always be shared and why?

  An application's executable code should always be shared between all of its instances to preserve memory and to thus reduce the need for memory swapping between physical memory and the disk.