CORE EXAMINATION with Solutions MAY 1996

Department of Computer Science

New York University

May 17, 1996

Programming Languages & Compilers

PL&C1.

(a) What is operator overloading?

(b) Is there operator overloading in Pascal? Explain.

(c) We want to perform arithmetic on fractions. A fraction is represented by two integer components: its numerator and its denominator. In Ada or C++ write a type or class definition for Fraction, and write the definition of an overloaded operator "+" that computes the sum of two fractions.

Solution:

(a) Operator overloading is a syntactic facility that allows operator symbols to have different meanings depending of the type of the arguments to which they are applied. Some languages allow user-defined overloading, among them Ada and C++. (In type theory, operator overloading is called “ad-hoc polymorphism” and considered a cheap trick).

All programming languages use overloading at least for arithmetic operations, where (A + B) may mean integer addition or floating-point addition (two completely different operations) depending on the types of A and B.

(b) As mentioned above, operator overloading is present in Pascal for arithmetic operators, but there is not facility provided for the programmer to add new overloadings.

(c) In Ada, we will define a package that contains the type and operations that apply to the type:

```
package Fractions is
    type Fraction is private;
    function Numerator   (F : Fraction) return integer;
    function Denominator (F : Fraction) return integer;

    function "+" (F1, F2 : Fraction) return Fraction;
    ...  other useful overloaded operators.
private
   type Fraction is record
```

```
      Num : Integer;
      Den : Integer:
   end record;
end Fractions;
```

The bodies of operations, including overloaded operators, is in the body of the package:

```
package body Fractions is
   function "+" (F1, F2 : Fraction) return Fraction is
      Result: Fraction;
   begin
      Result.Num := F1.Num * F2.Den + F1.Den * F2.Num;
      Result.Den := F1.Den * F2.Den;
      --  In  practice, we would simplify the result using a GCD
operation.
      return Result;
   end "+";
   ...      implementation of other operations.
end Fractions;
```

The syntax of the C++ solution will depend on whether the operation is defined as a class method (in which case it has a single explicit parameter) or as a friend of the class (in which case it has two parameters, as in Ada).

PL&C2.

(a) Briefly explain the purpose of destructors in C++.

(b) Define a class NameTable whose representation is an array of strings of different lengths. Write the body of the destructor for this class.

PL&C3. The semantics of Java specify that a program is illegal if it cannot be determined statically that all variables have a value when they are referenced. For example, the following program is illegal and rejected by the compiler:

```
int this, that, res;
        if (maybe)
           this = 5;
        else
           that = 6;

        res = that;
```

because on one branch of the if-statement the variable `that` is not assigned.

The compiler must compute a data-flow attribute (let's call it 'defined') that must have the value True for all variable references. Describe how this attribute can be computed. You can use data flow equations or a syntax directed approach.

Solution. This is a forward data-flow problem with intersection. For every basic block in the program we must determine which variables are defined (i.e. have received a value) on entrance to the block, and which have a value on exit from the block. Let IN (b) and OUT (b) denote these values. Within the block, each assignment statement makes its left-hand side defined, so for each block we define GEN (b) as the set of variables that receive a value within b. Then the following equation holds:

OUT (b) = IN (b) + GEN (b)

where "+" denotes set union. The statement of the problem indicates that at a point of use, all paths leading to it must have provided a value for the variable being used, so that

IN (b) = Intersection (OUT (b')) for b' in Pred (b)

that is to say, what is defined on entry to a block is the intersection of what is defined on exit from all the predecessors of the block.

Operating Systems

OS1. Consider a memory management system with segmentation and demand paging. As you know, this means that each entry in the segment table of a process points to the page table for that segment.

A. Assume simple hardware with no TLB (translation lookaside buffer) or cache. How many memory references are needed to translate a virtual address to the corresponding physical address? Describe each of these references.

B. To implement demand paging, a page replacement policy must be selected. Normally, LRU (or some approximation) is used but for this problem consider three others FIFO (first in first out), LIFO (last in first out), and RANDOM (an arbitrary page is evicted). One of these three is likely to be quite bad; which one and why?

OS2. Consider a system with four units of a resource of the same type, shared by three processes P_1, P_2 and P_3. Each of these processes needs NO MORE than two units of the above resource at any given time.

Show that the system is dead-lock free assuming that you have only the three processes P_1, P_2 and P_3 and that the resource is the only shared resource.

ALGORITHMS

ALGS1. We find code for different sorting algorithms. The first, Algorithm A, claims that it works in time $O(n \log n)$ on average, but in the worst case, can take time $O(n^2)$. The second, Algorithm B, says that it works in time $O(n^2)$ in all cases. The third, Algorithm C, takes time $O(n!)$.

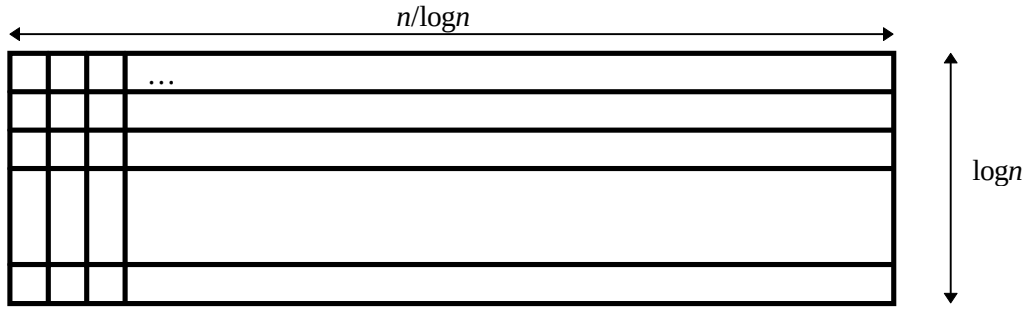
(a) Which is the "best algorithm," and which is the "worst?"

(b) What is a possible algorithm for Algorithm A, for Algorithm B, and for Algorithm C? For each, either state the name of an algorithm, or give a one-sentence description of an algorithm.

(c) Algorithm D is worst-case $O(n^{1.5})$, and algorithm E in the worst-case runs in quadratic time. In experiments using data sets of the same size, algorithm E runs faster than algorithm D. Explain this.

ALGS2. (a) Describe briefly the procedure MERGE that takes as input two sorted lists *A* and *B* each of size *n* and returns a sorted list *C* of length 2*n* consisting of the values in lists *A* and B. What is the time complexity of MERGE?

(b) Consider the following generalization. You are given $\log n$ sorted lists each of size $n/\log n$, (see figure—the rows are the sorted lists). Once again, you wish to merge the lists into a single sorted list of size *n*. Describe how the algorithm MERGE can be generalized to this case, **using a heap data structure**. What is the time complexity of the resulting MERGE process?



ALGS3. A simple undirected graph $G = (V, E)$ without self-loops has at most one edge between every pair of vertices and no edge from a vertex to itself. A graph is *p*-colorable if all vertices can be assigned one of the *p* colors with no edge receiving the same color at both of its ends.

Let *d*(*v*) denote the degree of a vertex *v*, i.e., the number of edges incident at *v*. Let *d*(*G*) denote $\max_{v \in V} d(v)$, the maximum degree of the vertices of the graph *G*.

Design an efficient algorithm and prove its correctness, which provides a $(d(G)+1)$-coloring of a given graph *G*. What is the time complexity when $d(G) = 3$ and *V* has *n* vertices?