CORE EXAMINATION
Department of Computer Science
New York University
May 14, 2004

This is the common examination for the M.S. program in CS. It covers core computer science topics: Languages and Compilers, Operating Systems, and Algorithms. The exam has two parts. The first part lasts three hours and covers the first two topics. The second part, given this afternoon, lasts one and one-half hours, and covers algorithms.

Use the proper booklet for each question. Each booklet is marked with the Area and Question number, in the form PL&C1, PL&C2, PLC&C3, OS1, OS2, ALGS1, ALGS2, ALGS3. Use the appropriate booklet for each question. DO NOT put your name on the exam booklet. Instead, your exam number must be on every booklet.

You will be graded according to your exam number, shown on the envelope containing the booklets. Remember your exam number: when grades are given out, they will be published according to this number, not by name.

Make sure your name and signature are on the envelope. This is the only place where your name appears. Please include all the booklets inside the envelope. You can keep the exam.

Good luck!

Programming Languages and Compilers

Question 1

a. Define the term *multiple inheritance*, as used in C++.

b. To illustrate multiple inheritance, define in C++ a class `seaplane` which is derived from a class `plane` and a class `boat`, both of which you should also define.

c. Java does not support multiple inheritance, but Java's proponents often claim that Java interfaces can be used in a similar way. Illustrate this by defining a `seaplane` class in Java which has characteristics of both planes and boats. Be sure to define whatever other classes or interfaces you need.

d. Java's designers intentionally did not include multiple inheritance, claiming that the benefits of multiple inheritance are outweighed by the complexity of using multiple inheritance and the complexity of implementing it.

- Give an example where the use of multiple inheritance in C++ leads to increased complexity that the programmer must deal with, and
- Give an example where the use of multiple inheritance in C++ leads to increased complexity of the implementation (i.e. the compiler).

Question 2

a. Explain the function of the stack pointer and frame pointer in setting up the stack frame for a function call. Draw a picture of a typical stack frame to illustrate your answer.

b. For the following c function, write the prologue code to set up the stack frame and the corresponding epilogue code to release it:

```
int try (int x) {
  int here, there;
  ...
  return here * there;
}
```

c. There is an option for omitting the frame pointer in GCC. What does this mean and how does it work?

d. What language constructs (in C, or C++ or Ada, your choice) make the frame pointer indispensable (so that the gcc option mentioned in c) has no effect)

Question 3

We say that a program has a memory leak if its execution consumes heap space unnecessarily, so that memory usage is proportional to the running time of the program. A typical memory leak occurs when objects are created in a loop, and used only once for each iteration.

a. In a language without a garbage collector, such as Ada or C++, write a code fragment that shows such a memory leak. Assume that the object which is created repeatedly holds a numeric value and a string of some dynamic size. Write the corresponding type or class declaration.

b. Destructors can be used to automatically reclaim objects when they go out of scope. Write a destructor (or a Finalize procedure) for the type or class of part a), and modify the code so that the memory leak disappears.

c. In Java, the presence of the garbage collector removes some but not all sources of memory leaks. For example, for logging/recovery purposes it is often necessary to create a static container (say a Vector) that stores the objects created for each invocation of a given method. Write a Java fragment that displays this construct.

d. The code in c) has a memory leak if the container is not flushed periodically. Explain how this flushing can be done.

Operating Systems

OS 1:

Consider a system with **32-bit** logical addresses, which implements **segmentation with paging**. Segment numbers are **16-bits** long and the page size is **1 KB** (2^{10} bytes). The Segment Table Entry (STE) and Page Table Entry (PTE) structures are defined as shown below in C/C++ syntax (the `unsigned` type represents a 32-bit unsigned integer, `PTE*` points to an array of PTEs and is equivalent to `PTE[]` in Java):

```
struct PTE
  unsigned frame_no;
  /* other fields, not relevant for this question */
;

struct STE
  unsigned segment_no;
  int segment_length;
  /* pointer to the page table for this segment */
  PTE* segment_page_table;
  /* other fields, not relevant for this question */
;
```

Note that the STE structure above explicitly stores the segment number. The rationale for this decision is that one expects each process to have a small number of segments, but with well-known segment numbers.

Provide the pseudo-code (using C/C++ or Java-like syntax; partial credit for an English description) for the following function, which translates a logical address to a physical address:

```
/*
 * Inputs:
 * VA: the logical address
 * segTablePtr: pointer to the segment table (an array of STEs)
 * numEntries: number of entries in the segment table
 *
 * Output is either:
 * the physical address, or
 * a call to a routine "invalid()" for handling illegal accesses
 */
unsigned GetPhysicalAddress( unsigned VA, STE* segTablePtr, int =
numEntries );
```

For full credit, your solution must include checks for invalid accesses.

OS 2:

Consider three file allocation strategies: **contiguous**, **linked**, and **Unix i-node**. For linked allocation, the pointer to the next block is stored within the block itself. For simplicity the Unix i-node allocation does not include indirect blocks.

Assume that the directory entry for a file stores a pointer to the starting block and the number of blocks for the contiguous case, pointers to the starting block and the ending block for the linked case, and a pointer to the i-node for the Unix i-node case.

For a file consisting of 100 blocks, how many disk I/O operations are required for each of the above allocation strategies for the following three operations:

a. Adding a block at the beginning of the file

b. Removing block 50 from the file

c. Adding a block at the end of the file

Assume that the directory entry, the i-node in the Unix i-node case, and the data to be added (in cases *a* and *c*) are all present in memory. Assume also that you can ignore the cost of querying and updating the free block list.

For full credit, your solution must include any disk I/O operations involving the block containing the directory entry and/or the i-node that might be required for maintaining file system consistency.