(VERSION WITH ANSWERS)
CORE EXAMINATION
Department of Computer Science
New York University
February 8, 2008

This is the common examination for the M.S. program in CS. It covers four core computer science topics: Programming Languages and Compilers, Operating Systems, and Algorithms. The exam has two parts. The Systems Part (PL&C and OS) lasts three hours and covers the first three topics. The Algorithms Part (Algo), given this afternoon, lasts one and one-half hours, and covers the last topic.

You will be assigned a seat in the examination room.

Use the proper booklet or answer sheet 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 and ALGS2. But ALGS3 question has an answer sheet and not a booklet. DO NOT put your name on the exam booklet or answer sheet. 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!

SYSTEMS PART: Programming Languages, Compilers and Operating Systems

Question 1 – please use the Exam Booklet labeled PL&C1

Programming Languages: Fun with Functional Lists

Scheme and ML have built-in list data structures: a list is a sequence of pairs, with a pair's head pointing to the actual element and a pair's tail pointing to either another pair or to a null value terminating the list.

Consider this Java implementation of a pair:

```
public class Pair<T> {
  private final T head;
  private final Pair<T> tail;

  private Pair() { head = null; tail = null; }
  private static final Pair EMPTY = new Pair();

  @SuppressWarnings("unchecked")
  public static final <T> Pair<T> empty() {
    return (Pair<T>)EMPTY;
  }

  public Pair(T h, Pair<T> t) {
    if (null == t) throw new NullPointerException();
    head = h;
    tail = t;
  }

  public boolean isEmpty() {
    return this == EMPTY;
  }

  public T head() {
    if (isEmpty()) throw new UnsupportedOperationException();
    return head;
  }

  public Pair<T> tail() {
    if (isEmpty()) throw new UnsupportedOperationException();
    return tail;
  }
}
```

The EMPTY value serves as the canonical empty list value and the static empty() method provides type-safe access to this value, since static fields cannot have generic types in Java.

1. (3 Points) Write the *recursive* definition of the length() method for Pair<T> in Java, which counts the elements on a non-circular list. In other words, this version of length() should invoke itself on some smaller sublist.

Answer:

```
public int length() {
  return isEmpty() ? 0 : 1 + tail.length();
}
```

2. (3 Points) Write the *iterative* definition of the length() method for Pair<T> in Java, which counts the elements on a non-circular list. In other words, this version of length() must not invoke itself.

Answer:

```
public int length() {
  Pair<T> p = this;
  int     l = 0;

  while (! p.isEmpty()) {
    l++;
    p = p.tail;
  }

  return l;
}
```

3. (4 Points) We can easily test whether a list of pairs is circular by iteratively traversing the list with two pointers. For each step, the first pointer advances one pair and the second pointer advances two pairs. If either pointer encounters the emtpy list, the overall list is *not* circular. If, after advancement, both pointers reference the same pair, the overall list *is* circular. Otherwise, we continue with the traversal.

   Write the definition of the `isCircular()` method for `Pair<T>` in Java. Your implementation should use the minimal number of tests for the empty list.

Answer:

```
public boolean isCircular() {
   if (isEmpty()) return false;

   Pair<T> p1 = this, p2 = this;
   while (true) {
     if (p2.tail.isEmpty()) return false;
     if (p2.tail.tail.isEmpty()) return false;
     p2 = p2.tail.tail;
     p1 = p1.tail;

     if (p1 == p2) return true;
   }
}
```

Question 2 – please use the Exam Booklet labeled PL&C2

Program Language:

1. (2 Points) *Function overloading* is the ability to give distinct functions in a program the same name. In the context of function overloading, what is the difference between *dynamic overloading* (also called *overriding* in Java) and *static overloading*? Write a very small C++ program that exhibits both forms of overloading.

Answer:

Dynamic overloading is the assigning of the same name to distinct functions, where the overload resolution – that is, determining which of the distinct functions is being referenced in a function call – is performed at run-time. Static overloading refers to assigning of the same name to distinct functions, where the overload resolution is performed at compile time.

```
class A {
public:
virtual void foo() { cout << "A"; }
virtual void foo(int x) { count << "A: " << x; }
};

class B : public A {
public:
virtual void foo() { cout << "B"; }
};
```

2. (2 Points) In order for dynamic overloading to occur in C++, what must the relationship be between the distinct functions with the same name? Be precise.

Answer: For dynamic overloading of two functions with the same name to occur, one of the functions must be defined in an ancestor class of the class in which the other function is defined. In addition, the function defined in the ancestor class must be declared as `virtual`. Finally, the signature of the two functions (i.e. the parameter types and the result type) must be the same. Two functions can also be dynamically overloaded if they are dynamically overloaded with the same function in a common ancestor class.

3. (2 Points) What does the following program print? Explain your answer.

```
#include <iostream>
using namespace std;

class A {
public:
  void f() { cout << "A: " << this->bar() << endl; }
protected:
  virtual int bar() { return 6; }
};


class B : public A {
public:
  void f() { cout << "B: " << this->bar() << endl; }
protected:
  virtual int bar() { return 30; }
};


void foo(A &z)
{
  z.f();
}

int main()
{
  A a;
  B b;
  foo(a);
  foo(b);
}
```

Answer:

The program would print:

```
A: 6
A: 30
```

In the call to `foo(a)`, the formal parameter `z` becomes bound to an object of type `A`, and thus `A::f()` would be called. Within `A::f()`, the call `this->bar()` would invoke `A::bar()`, because `this` points to an object of type `A`.

In the call to `foo(b)`, the formal parameter `z` would becomes bound to an object of type `B`, but since `z` is declared to be of type `A` and `A::f()` is not declared using the `virtual` keyword (and thus no dynamic overloading of `f()` occurs), `A::f()` would be called. Within `A::f()`, the call `this->bar()` calls `B::bar()`, because `bar()` is declared using the `virtual` keyword (and is thus dynamically overloaded) and `this` points to an object of type `B`.

4. (1 Point)

   In the above program, if the `virtual` keyword was removed from the definition of `B::bar` (but remained in the definition of `A::bar`), what would the program print? Why?

Answer:

The program would still print

```
A: 6
A: 30
```

because dynamic dispatch of `bar()` within `A::f()` would still occur since `bar()` is declared `virtual` in `A` and the `this` object is implicitly declared to be a `A` pointer.

5. (1 Point)

   If the keyword `virtual` was removed from the definition of `A::bar` (but remained in the definition of `B::bar`), what would the program print? Why?

Answer:

The program would now print

```
A: 6
A: 6
```

because no dynamic dispatching of `bar()` within `A::f()` would occur since `bar()` is not declared virtual in `A` and the `this` object is implicitly declared to be an `A` pointer.

6. (1 Point)

   If the keyword `virtual` was added to the definition of `B::f` but not to `A::f`, what would the program print? Why?

Answer:

The program would still print

```
A: 6
A: 30
```

because no dynamic dispatching of `f()` within `foo()` would occur since `f()` would not be declared virtual in `A` and `z` is statically declared to be of type `A`.

7. (1 Point)

   If the keyword `virtual` was added to the definition of `A::f` but not to `B::f`, what would the program print? Why?

Answer:

The program would now print

```
A: 6
B: 30
```

because dynamic dispatching of `f()` within `foo()` would occur since `f()` would be declared virtual in `A` and `z` is statically declared to be of type `A`.

Question 3 – please use the Exam Booklet labeled PL&C3

COMPILER:

A) (6 points) Consider the language over the alphabet $\{0, 1\}$ which consists of strings with the same number of 0's and 1's.

(a) (3 points)

Is this a regular language? Justify your answer.

Answer: The language is not regular. If it were regular then so would be its intersection with the regular language 0^*1^*. But this intersection yields the language $\{0^i1^i \mid i \geq 0\}$ which is known to be non-regular, as can be proven using the pumping lemma.

Alternately, we can prove directly that no (deterministic) finite-state automaton (DFA) can recognize the language $\{0^i1^i \mid i \geq 0\}$. Assume to the contrary that there exists a DFA A which recognizes this language, and let A has N states. Let $m > N$ be some integer greater than N, and let us observe the accepting run of A over the word a^mb^m. Since A has only N states, the run of A over a^mb^m must close a cycle (visit the same state twice) while still reading the prefix a^m. Assume that this cycle is of size k, $0 < k \leq N$. It can be shown that the run of A over the shorter word $a^{m-k}b^m$ reaches the same accepting state as the run of A over a^mb^m. Thus, automaton A accepts the word $a^{m-k}b^m$ which is not in the language $\{0^i1^i \mid i \geq 0\}$ – a contradiction.

(b) (3 points)

Sketch a parsing algorithm for this language (no need to find a grammar for the language).

Answer: This language can be parsed using a push-down stack by a deterministic automaton. Initially, we place in the stack the symbol Z. The following table identifies the actions of the push-down automaton according to the symbol at the top of the stack and the next incoming character. The character # represents the end of the input.

Top of Stack	Incoming Character	Action
Z	0	Push X
Z	1	Push Y
Z	#	Accept word
X	0	Push X
X	1	Pop X
X	#	Reject word
Y	0	Pop Y
Y	1	Push Y
Y	#	Reject word

(c) (3 points – Bonus)

Write a grammar for this language.

Answer: A possible grammar for this language is:

$$S \rightarrow 0S1 \quad | \quad 1S0 \quad | \quad \epsilon \quad | \quad SS$$

B) (4 points)

Write code or a syntax-directed definition (SDD) that will generate three-address intermediate code corresponding to a for-loop in C of the form

for(e1; e2; e3) stmt

Assume that the code for generating expressions and statements has already been written.

Answer: According to the semantics of C, the elements e1, e3 and stmt are statements, while e2 is a boolean expression. We assume the existence of a procedure "gen_statement(s)" which adds to the compiled output three-address code that execute statement s. Also assume the procedure "gen_exp(exp,T)" which generates the code for computing the expression "exp" and places the result in variable "T". The code executing the presented "for" statement can be generated by the following:

```
$start\_loop := newlabel;$
$end\_loop := newlabel;$
$temp := newvar;$
$gen\_statement(e1);$
$issue(\textbf{Label } L(start\_loop) :);$
$gen\_exp(e2, temp);$
$issue(\textbf{if } temp = 0 \textbf{ then goto } L(end\_loop));$
$gen\_statement(stmt);$
$gen\_statement(e3);$
$issue(\textbf{goto } L(start\_loop));$
$issue(\textbf{Label } L(end\_loop) :);$
```

Question 4 – please use the Exam Booklet labeled OS1

Process Synchronization:

The two loops shown below are written in a C-like language. Each loop is part of a process and the two processes are run concurrently. For convenience, I refer to the loop and process that modifies A as loop A and process A. Similarly for B.

Loop A	Loop B
`while (A>0) {`	`while (B>0) {`
`  P(S);`	`  P(S);`
`  A = A-1;`	`  B = B-1;`
`  X = X+1;`	`  X = X+10;`
`  printf("A: X=%d\n", X);`	`  printf("B: X=%d\n", X);`
`  V(S);`	`  V(S);`
`}`	`}`

The three variables `X, A, B` are shared by the two processes (so changes made by one process to any of these variables are seen by both processes). `X` is initially 0 and the other two variables are each initially 2. The initializations are not shown; they occur before any of the visible statements are executed.

`S` is a (binary) semaphore, `P(S)` and `V(S)` are the standard semaphore operations (some authors refer to `P` and `V` as `down` and `up`). `S` is initially open, that is, the first `P` executed will not block.

There are no other processes, there are no other references to `X, A, B`, and there are no other `printf` statements.

Part A (6 points)

The (inexperienced) author of the above code was surprised that the results printed differed from one run to another even thought no code or data was changed.

Show all the results printed for each possible execution of the above code.

Answer:

```
A: X=1          A: X=1          A: X=1
A: X=2          B: X=11         B: X=11
B: X=12         A: X=12         B: X=21
B: X=22         B: X=22         A: X=22


B: X=10         B: X=10         B: X=10
B: X=20         A: X=11         A: X=11
A: X=21         B: X=21         A: X=12
A: X=22         A: X=22         B: X=22
```

Part B (4 points)

The author next inserted
`V(S); P(S);`
just before each of the `printf` statements in Loop A and Loop B.

For the modified code there were sometimes results that never occurred with the original code. Present the results printed during one execution of the modified code that cannot occur with any execution of the original code. Explain your answer, i.e., explain how the modified code can produce these results and explain why the original code cannot.

Answer:

```
In the original code, each loop iteration is atomic
so the value of X printed is the value computed in this iteration.
In particular, no value of X is repeated.
In the modified code the A and B loops are not atomic.
Suppose the A loop goes first and computes, but does not yet print, X=1.

Now A executes V(S) and it is possible that,
before A can execute P(S),
  B executes P(S);  B computes X=11;
  B executes V(S);  B executes P(S);
  B prints X=11;    B executes V(S);
  B executes P(S);  B computes X=21;
  B executes V(S);  B executes P(S);
  B prints X=21;    B executes V(S);
and the B loop terminates.
Then

  A executes P(S);  A prints X=21  (a DUPLICATED value)
  A executes V(S);  A executes P(S);
  A computes X=22;  A executes V(S);
  A executes P(S);  A prints X=22;
  A executes V(S);

and the A loop terminates.
```

Question 5 – please use the Exam Booklet labeled OS2

File Organization: Consider the organization of secondary memory into logical entities called **files**. The physical memory is divided into contiguous chunks called **blocks**, each of size β bytes and each block number is 32-bit (or 4 bytes long). Assume a separate structure to represent the logical organization of files into a file hierarchy.

1. (2 Points) Describe the FAT method for physical file organization.

   Answer: We use a table called FAT (for File Allocation Table) which normally resides in main memory. There is an entry in FAT for each disk block (the block number is an index into FAT). Each entry, if it represents a block of some file, stores the block number of the next block in the file. The entry is -1 if the block is the last block of its file. Thus FAT represents the linked list structure of all files in the disk.

2. (3 Points) Give a high-level algorithm showing how to find the nth byte in file "foobar" under the FAT method. Indicate how the file directory and the block size β are used.

   Answer:
   1. Use the file directory structure to find the first block number of file "foobar". Typically, the full path name of "foobar" is given in order to use this file structure.
   2. Use this first block to index into FAT. Continue to find the number of the $\lceil n/\beta \rceil$-th block of "foobar" (i.e., follow the links in FAT for $\lceil n/\beta \rceil - 1$ times). If we encounter -1 while doing this, return error ("file has less than n bytes").
   3. Read this block into memory.
   4. Return the $(n \mod \beta)$-th byte of this block.

   REMARK: This algorithm assumes that n is valid. It is not hard to add checks for validity of n while executing this algorithm.

3. (2 Points) Describe the I-node method for physical file organization. Emphasize the information in I-nodes that supports efficient file access, explaining how it works.

   Answer: The directory containing a given file will store a pointer to an I-node. This I-node stores ownership information (user ID, group ID, etc) protection information (file permissions), and bookkeeping information (time of creation, etc).
   For access to file data, it stores 12 block numbers, corresponding to the first 12 data blocks for the file. (Note: this "12" may vary with β to some extent.) Thus, access to the first 12β bytes is very efficient. Then, it has 3 additional indirect block pointers (single indirect, double indirect and triple indirect). The first indirect block holds $\beta/4$ block numbers, and hence they can access $\beta^2/4$ additional bytes. The double indirect pointers points to a block that holds $\beta/4$ single indirect pointers. Using the these indirect pointers, we can acess up to $\beta^3/16$ additional bytes. The third indirect is not needed in 32-bit address space for typical values of β. E.g., $\beta = 4K$.

4. (3 Points) Describe a high-level algorithm showing how to find the nth byte in file "foobar" under the I-Node method. Indicate how the file directory and the block size β are used.

Answer:
1. Use the file directory to get to the I-node for "foobar".
2. Compute $b = \lceil n/\beta \rceil$.
3. If $b < 12$, then we can directly use the address of the bth block from the I-node and read the data block D into memory. Go to Step 6.
4. Otherwise, if $b < 12 + \beta/4$, we can read the first block given by single indirect pointer, and obtain number of the $(b - 12)$-th block. Read this data block D into memory. Go to Step 6.
5. Otherwise, if $b < 12+\beta/4+\beta^2/16$, read in the double indirect block S. Compute $c = \lceil 4(b - 12 - \beta/4)/\beta \rceil$, and use the c-th pointer in S to read in a single indirect block T. Compute $d = (b - 12 - \beta/4) \mod \beta/4)$, and use the d-th pointer in T to read in the data block D.
6. Now the data block D from steps 3, 4 or 5 is in memory. Return the $(n \mod \beta)$th byte of this block.