

University of Toronto  
Faculty of Applied Science and Engineering

ECE 106S

PROGRAMMING FUNDAMENTALS

Spring 2007

Final Examination

Examiner: T.S. Abdelrahman, C. Gibson, and A. Goel

Duration: Two and a Half Hours

This exam is open textbook and open notes. Use of computing and/or communicating devices is NOT permitted.

Do not remove any sheets from this test book. Answer all questions in the space provided. No additional sheets are permitted.

Work independently. The value of each part of each question is indicated. The total value of all questions is 100.

Write your name and student number in the space below. Do the same on the top of each sheet of this exam book.

Name: \_\_\_\_\_  
(Underline last name)

Student Number: \_\_\_\_\_

Q1. \_\_\_\_\_

Q9. \_\_\_\_\_

Q2. \_\_\_\_\_

Q10. \_\_\_\_\_

Q3. \_\_\_\_\_

Q11. \_\_\_\_\_

Q4. \_\_\_\_\_

Q12. \_\_\_\_\_

Q5. \_\_\_\_\_

Q13. \_\_\_\_\_

Q6. \_\_\_\_\_

Q14. \_\_\_\_\_

Q7. \_\_\_\_\_

Q15. \_\_\_\_\_

Q8. \_\_\_\_\_

Q16. \_\_\_\_\_

Total

**Question 1. (13 marks).** *General.*

Answer the following questions either by **Yes** or **No**, or by providing a **very brief** and **direct** answer when indicated.

- (a) **Yes** or **No**? One should always arrange the `catch` blocks for a given `try` statement starting with `Exception` and ending with the most specific exception.
- (b) **Yes** or **No**? It is not safe for a function to return a pointer to an object that was created on the stack inside the function.
- (c) What is the worst case complexity of search in a hash table of size  $N$  and with  $n$  items in it, such that  $n/N = 0.5$ . Use big-O notation.
- (d) **Yes** or **No**? Can a hash function be made better by using a *truly* random number generator?
- (e) **Yes** or **No**. An  $O(\log(n))$  algorithm is always faster than an  $O(n)$  algorithm when  $n > 1$ ?
- (f) **Yes** or **No**. The fact that a function is virtual matters only when the function is invoked through a pointer to object
- (g) How would you force a derived class to implement a virtual method from the base class?
- (h) A 2-tree is a binary tree in which each node has exactly two empty subtrees or two non-empty subtrees, each of which is also a 2-tree. **Yes** or **No**: any 2-tree has an odd number of nodes?
- (i) We decide to double the size of a hash table when the density factor  $\lambda$  is greater than 0.5. In order to do so, we must rehash all the elements already in the table. Explain why this rehashing is necessary.
- (j) What Unix command would you type to create a new directory called `ece106`?

(k) After creating a new file called `a.h` you notice that it is readable by all other students. What single Unix command would you type to ensure that other students cannot read the file, without changing any other attributes?

(l) You are asked to count the number of nodes in a binary search tree. What is the complexity (in Big-O notation) of the most efficient algorithm that could perform this task?

(m) Consider the following recursive function:

```
int Fun (int n) {  
    if (n == 4) return (2);  
    else return (2* Fun(n+1));  
}
```

What is the value returned by the function call `Fun(2)`?



**Question 4. (6 marks).** *Objects.*

Study the following class definition (`usepair.h`) and implementation (`usepair.cc`).

```
struct pair {
    int first;
    int second;
};

class usePair {
private:
    bool valid;
    struct pair* thepair;

public:
    usePair();
    usePair(int f, int s);
    ~usePair();
    void setFirst(int f);
    void setSecond (int s);
    usePair operator+ (usePair rhs);
    void print();
};
```

```
#include "usepair.h"
#include "iostream"
using namespace std;

usePair::usePair() {
    valid = true;
    thepair = new struct pair;
    thepair->first = 0;
    thepair->second = 0;
}

usePair::usePair(int f, int s) {
    valid = true;
    thepair = new struct pair;
    thepair->first = f;
    thepair->second = s;
}

usePair::~~usePair() {
}

void usePair::setFirst(int f) {
    thepair->first = f;
}

void usePair::setSecond(int s) {
    thepair->second = s;
}
```

```

usePair usePair::operator+ (usePair other) {
    usePair sum;
    sum.thepair->first = thepair->first + other.thepair->first;
    sum.thepair->second = thepair->second + other.thepair->second;
    other.thepair->first = 0;
    other.thepair->second = 0;
    return (sum);
};

void usePair::print() {
    cout << "(" << thepair->first
         << "," << thepair->second
         << ")" << endl;
}

```

Now consider the following main function, which uses the above class.

```

#include "iostream"
#include "usepair.h"
using namespace std;

int main () {
    usePair a(1,1);
    usePair b(4,16);
    usePair c = b;
    usePair d;

    a.print(); // Statement # 1
    b.print(); // Statement # 2
    c.print(); // Statement # 3
    d.print(); // Statement # 4

    a.setFirst(0);
    b.setFirst(8);
    c.setSecond(20);
    d.setFirst(5);
    d.setSecond(10);

    a.print(); // Statement # 5
    b.print(); // Statement # 6
    c.print(); // Statement # 7
    d.print(); // Statement # 8

    d=a+b;
    c=a;

    a.print(); // Statement # 9
    b.print(); // Statement # 10
    c.print(); // Statement # 11
    d.print(); // Statement # 12

    return (0);
}

```

Indicate what is being printed by each statement in the above main function. For simplicity, each statement that produces output has been given a number, and you can write the output of each statement in the table below.

Statement #	Output
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	

**Hint:** A picture is worth a thousand words.

**Question 5. (5 marks).** *Inheritance.*

Consider the following class definitions.

```
class BaseC {
private:
    int a;
    void help();
protected:
    int b;
public:
    int c;
    int d;
    void print();
};
```

```
class DerivedC : public BaseC {
private:
    int x;
    void nohelp();
public:
    int w;
    int z;
    void noprint();
};
```

The following declarations and statements are made in the nohelp member function of the class DerivedC.

```
BaseC base;
BaseC* baseptr;
DerivedC derived;
DerivedC* derivedptr;
```

Indicate by placing an **X** in the appropriate column whether each of the following statements is *correct* code, or *incorrect* code. The statements also appear in main. Assume each statement or group of statements in each row of the table to be independent of the others.

	Correct	Incorrect
derived.a = 8;		
derived.b = 10;		
derived.x = 12;		
derived.w = 4;		
baseptr = derivedptr;		
derivedptr = baseptr;		
base = derived;		
derived = base;		
baseptr = new DerivedC(); baseptr->c = 0;		
baseptr = new DerivedC(); baseptr->noprint();		

**Question 6. (6 marks).** *Inheritance.*

What output does the following code generate?

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

class A {
public:
    A();
    virtual void Foo();
};

class B : public A {
public:
    B();
    virtual void Foo();
};

A::A() {
    this->Foo();
}

void A::Foo() {
    cout << "A::Foo()" << endl;
}

B::B() {
    this->Foo();
}

void B::Foo() {
    cout << "B::Foo()" << endl;
}

int main() {
    B objectB;
    return 0;
}
```

Output:

**Question 7. (8 marks).** *Inheritance.*

Study the following class definitions and implementation.

```
// firstOne.h
class firstOne {
private:
    int x;
    char* first;

public:
    firstOne();
    firstOne(char* s);
};

// secondOne.h
class secondOne : public firstOne {
private:
    int y;
    char* second;

public:
    secondOne(char* s);
    secondOne(int v);
};

// firstOne.cc
firstOne::firstOne() {
    x = 0;
    first = new char[1];
    first = '\0';
    cout << "Constructor 1 of firstOne done" << endl;
}

firstOne::firstOne(char* s) {
    x = 0;
    int size = strlen(s);
    first = new char[size+1];
    strcpy(first,s);
    cout << "Constructor 2 of firstOne done" << endl;
}

// secondOne.cc
secondOne::secondOne(char* s) {
    y = 0;
    int size = strlen(s);
    second = new char[size+1];
    strcpy(second,s);
    cout << "Constructor 1 of secondOne done" << endl;
}

secondOne::secondOne(int v):firstOne("X") {
    y = v;
    second = new char[1];
    second = '\0';
    cout << "Constructor 2 of secondOne done" << endl;
}
```

Now consider the following main function, which uses the above classes.

```
#include "iostream"
#include "firstOne.h"
#include "secondOne.h"
using namespace std;

int main () {
    firstOne a;
    firstOne b("G");
    secondOne c("N");
    secondOne d(8);
}
```

Indicate what is being printed by each statement in the above main function using the table below. If no output is produced, indicate so by writing "NONE".

Statement	Output
firstOne a;	
firstOne b("G");	
secondOne c("N");	
secondOne d(8);	

**Question 8. (7 marks).** *Inheritance.*

Consider the following definitions for the `Vehicle` and the `Car` classes.

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

class Vehicle {
private:
    int v_id;
    char *model;
public:
    Vehicle(int id = 0, char *m = "Unknown");
    virtual ~Vehicle();
    int get_id();
    virtual void move(int _x, int _y);
};

Vehicle::Vehicle(int id, char *m) {
    v_id = id;
    model = new char [strlen(m) + 1];
    strcpy(model, m);
    cout << "Vehicle: constructor: " << v_id << endl;
}

Vehicle::~~Vehicle() {
    move(0, 0);
    model = NULL;
}

int Vehicle::get_id() {
    move(0, 0);
    return v_id;
}

void Vehicle::move(int _x, int _y) {
    cout << "Vehicle: move: " << v_id << " " << model << endl;
}
```

```

class Car : public Vehicle {
private:
    int x, y;
    void print_position(void);
public:
    Car(int id, char *m, int _x = 0, int _y = 0);
    virtual ~Car();
    int get_id();
    virtual void move(int _x, int _y);
};

Car::Car(int id, char *m, int _x, int _y) : Vehicle(id, m) {
    x = _x; y = _y;
}

Car::~~Car() {
    print_position();
}

int Car::get_id() {
    print_position();
    return Vehicle::get_id();
}

void Car::move(int _x, int _y) {
    x += _x; y += _y;
    cout << "Car: move: x = " << x << ", y = " << y << endl;
    Vehicle::move(_x, _y);
}

void Car::print_position(void) {
    cout << "Car: position: x = " << x << ", y = " << y << endl;
}

// main program
int main()
{
    Vehicle v; /* 1 */
    Vehicle *vp = &v; /* 2 */
    Car c(10, "Zip"); /* 3 */
    Car *cp = &c; /* 4 */
    /* 5 */
    cout << vp->get_id() << endl; /* 6 */
    vp->move(1, 3); /* 7 */
    cp->move(2, 1); /* 8 */
    *vp = *cp; /* 9 */
    vp->move(1, 1); /* 10 */
    vp = cp; /* 11 */
    cout << vp->get_id() << endl; /* 12 */
    cp->move(1, 1); /* 13 */
} /* 14 */

```

Show the output produced by each line of the program in the table below. If a line has no output, write N/A.

Line #	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	


**Question 9. (4 marks).** *Exceptions.*

Consider the following code that uses exceptions.

```
int myfunction () {
    char* name = new char[4];
    strcpy(name, "Tom");
    try {
        bool cond;
        int x = 0;
        string s = "hello";
        int* a = new int[10];

        :
        :

        // If cond is true, an error has occurred
        if (cond) {

        }
        :
        :
    }
    catch (myExceptionType & e) {
        cout << "An error has occurred" << endl;
    }
    :
    delete name;
}
```

Write the body of the `if (cond)` statement so as:

- (a) an exception is thrown that will be caught by the above catch statement (assume a default constructor exists), and
- (b) no memory leaks exist when the exception is thrown.

Write your answer in the box above.

**Question 10. (10 marks).** *Exceptions.*

Assume that you are given the following class definitions and the program on the next page:

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

class Error {
private:
    int error;
public:
    Error(int err);
    virtual int gerror();
};

Error::Error(int err) {
    error = err;
}

int Error::gerror() {
    return error;
}

class OffError : public Error {
private:
    int offset;
public:
    OffError(int err, int offset);
    virtual int gerror();
};

OffError::OffError(int err, int off) : Error(err) {
    offset = off;
}

int OffError::gerror() {
    return Error::gerror() + offset;
}
```

```

int f1(int n) {
    try {
        if (n < 0 || n >= 100)
            throw Error(n);
    } catch (OffError & e) {
        cout << "f1: Offset error: " << e.gerror() << endl;
    }
    cout << n * n << endl;
    return n * n;
}

void f2(int n) {
    int x;
    x = f1(n);
    if (x > 100)
        throw OffError(x, n);
    try {
        x = f1(n * n);
        if (x > 100)
            throw OffError(x, n * n);
    } catch (Error e) {
        cout << "f2: Error: " << e.gerror() << endl;
    }
    cout << x << endl;
}

int main()
{
    int i;
    cin >> i; // 2, 4, 10, 30, 110
    try {
        f2(i);
    } catch (OffError & e) {
        cout << "main: Offset error: " << e.gerror() << endl;
    }
    cout << "end" << endl;
    return 0;
}

```

Answer the questions on the following page.

(a) What is printed if the user enters 2?

(b) What is printed if the user enters 4?

(c) What is printed if the user enters 10?

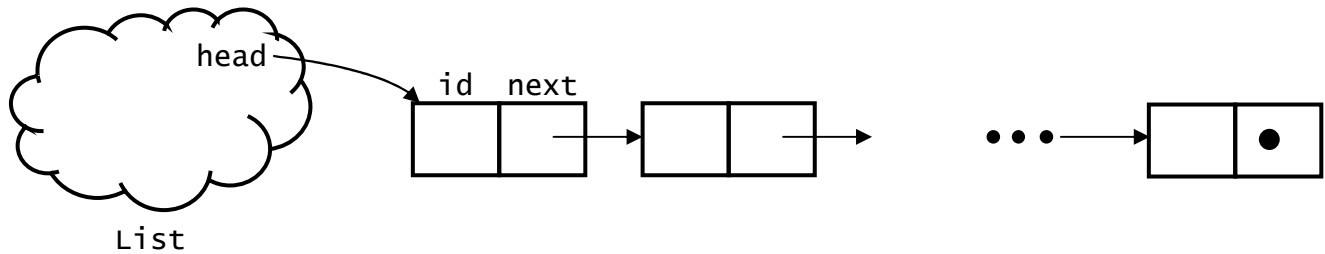
(d) What is printed if the user enters 30?

(e) What is printed if the user enters 110?



**Question 12. (5 marks).** *Recursion.*

A singly-linked list is shown below. The list consists of a `List` object with a `head` field and `Node` objects with the `id` and the `next` fields. Write a recursive `List_Tail` function that returns a pointer to the last element of the list. A template of the function is shown below.



```
Node* List_Tail(List *l) {
```

```
}
```

**Question 13. (11 marks).** *Complexity Analysis.*

Determine the *worst-case* time complexity (expressed in big-O notation) for each of the program segments below as a function of the size of the input  $n$ . Show the details of your analysis and clearly indicate your final result.

(a) (2 marks) The size of the input is  $n$ .

```
for (int i=0; i < n; ++i) {  
    for (int j=0; j < 100000000 ; ++j) {  
        O(1)  
    }  
}
```

$T(n) =$

(b) (2 marks) The size of the input is  $n$ .

```
for (int i=0; i < n; ++i) {  
    for (int j=0; j < 3*n ; ++j) {  
        O(1)  
    }  
}
```

$T(n) =$

(c) (2 marks). The size of the input is  $n$ .

```
int j = 0;  
for (int i=1; i <= n; i=i*2) {  
    for (int k=0; k < j ; ++k) {  
        O(1)  
    }  
}
```

$T(n) =$

(d) (5 marks). Assume for simplicity that  $n$  is a power of two.

```
int recursive(int n) {
    int x,y;
    if (n <= 1) return 0;
    else {
        for (int i=0; i < n ; ++i) o(1);
        recursive (n/2);
        recursive (n/2);
    }
}
```

Write the recurrence equation for  $T(n)$ .

Solve the recurrence equation (by inspection) to obtain an expression of  $T(n)$  in terms of  $n$ .

Express  $T(n)$  using the **big-O notation**.

$T(n) =$

**Question 14. (6 marks).** *Hash Tables.*

Assume you are given a hash table with size  $M = 11$ , and the following hash function:

$$h(\text{key}) = (\text{key} \% M)$$

(Note: the “%” is the modulus operator in C++)

Assume that **cubic probing** is used to resolve key collisions, and that the hash table is initially empty. If  $i$  is the initial hash index generated by the hash function above, cubic probing examines locations  $i + (k)^3$ , where  $k = 0, 1, 2, 3, \dots$  until an empty location is found.

Show the final contents of the hash table after all of the following operations have been performed, in the sequence shown. Next to each operation, indicate the number of probes performed when inserting that value.

insert 7

insert 3

insert 18

insert 29

insert 14

insert 40

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

**Question 15. (5 marks).** *The Make Utility.*

Consider the Makefile below.

```
1 all: maintool findtool mixtool
2
3 findtool: findtool.o
4         g++ findtool.o -o findtool
5
6 mixtool: tree.o list.o list.h
7         g++ list.o tree.o -o mixtool
8
9 maintool: findtool.o tree.o list.o
10        g++ tree.o findtool.o -o maintool
11
12 list.o: list.c list.h
13        g++ -g -Wall -c list.c -o list.o
14
15 tree.o: tree.c list.h
16        g++ -g -Wall -c tree.c -o tree.o
17
18 findtool.o: findtool.c findtool.h
19        g++ -g -Wall -c findtool.c -o findtool.o
```

The following table shows several invocations of the `Make` utility using the above correct Makefile. For each invocation, indicate the commands that are executed as a result of the invocation, *in the order in which they are invoked*. To simplify providing an answer, the lines of the Makefile are numbered as shown above; just indicate the line number corresponding to a command in the table provided below.

Assume that no files generated as a result of calling `make` exist before the first call of `make`. Also, the invocations of `Make` are performed *in the order shown in the table (i.e., the different commands are not independent)*.

Assume that the Makefile exists in the same directory as all the `.cc` and `.h` files.

Recall that the `touch` command simply updates the timestamp of its argument to the current time.

<b>Make Invocation</b>	<b>Commands Executed (indicate line number)</b>
make maintool	
touch list.h make mixtool	
make findtool	
make all	
touch list.h make	

**Question 16. (0 marks; no credit).**

Do you feel your performance on this exam reflects your understanding of the course material? If not, please explain why.

**THIS PAGE IS INTENTIONALLY BLANK FOR ANSWER OVERFLOWS**