Hong Kong University of Science and Technology
COMP151: Object Oriented Programming

Spring 2002, Midterm Examination

Monday 11 March, 7:15 –9.00 PM

Student Name: _____

Student ID: _____ Lab Section/TA Name: _____

Instructions:

1. ***This is a closed-book, closed-notes examination.***
2. ***Check that you have all 13 pages (including this cover page).***
3. ***Write your name, student ID, lab section/TA name on this page.***
4. ***Answer all questions in the space provided. Rough work should be done on the back pages.***

Question	Score
1	
2	
3	
4	
5	
6	
7	
Total	

Question 1 : You are given the definition of the class **Weekend**. Write the implementation of the default constructor and the destructor of class Weekend. The default constructor initializes the 2-dimentional array *days* to the values: "*Saturday"* and "*Sunday"*. The destructors deallocates the memory allocated for *days*.

```
class Weekend
{
private:

        const char** days;              // Dynamically allocated array
           int num;
public:
     Weekend(){                          //Default constructor
     ------------//to be filled-----------------

     ~Weekend(){                    //Destructor
     ------------//to be filled-----------------

     void print() const {for (int i=0;i<2;++i) cout<<days[i]<<endl;}
};
```

The following program demonstrates the use of class Weekend.

```
void main() {
     Weekend x;
     x.print();
}
```

The output of the main program should be:

```
OUTPUT
Saturday
Sunday
```

Write the implementation of `Weekend::Weekend()`

Write the implementation of `Weekend::~Weekend()`

Question 2 : You are given the class **Date** and the class **Person**:

```
class Date{
     int y,m,d;
public:
     Date(int x, int y, int z) {y=x; m=y; d=z;}
};

class Person{
     const Date birthday;
public:
     Person(int x, int y,int z){
     ------------//to be filled-----------------
};
```

Write the implementation of `Person::Person(int x, int y, int z):`

Question 3 (20 points) : You are given the class **Movie** and two functions **f1()** and **f2 ()**.

```
class Movie {

char * name;      //the title of the movie
int size;         //the number of characters of the title

public:
      //default constructor
      Movie() {
      cout<<"Default Constructor"<<endl;
       ------------//to  be filled-----------------

      //copy constructor
      Movie(const Movie &c) {
            cout<<"Copy Constructor"<<endl;
      ------------//to  be filled-----------------


      //conversion constructor
      Movie(const char * x) {
            cout<<"Conversion Constructor"<<endl;
      ------------//to  be filled-----------------

      ~Movie() {delete [ ] name;}
      void print() {cout<<name<<endl;}
      //transforms to uppercase letters the name of the movie
      void cap() {for(int i=0; i<size; i++) name[i]=toupper(name[i]);}
      //transforms to lowercase letters the name of the movie
      void low() {for(int i=0; i<size; i++) name[i]=tolower(name[i]);}


};

Movie f1(Movie x) { x.cap(); return x;}
Movie f2(Movie& x) { x.cap(); return x;}
```

A. (6 points) Complete the implementation of the default, copy and conversion constructor.

A1. Complete the implementation of the default constructor. The default constructor sets the name of the movie to the default value "*matrix*".

```
Movie::Movie() {
            cout<<"Default Constructor"<<endl;
```

A2. Complete the implementation of the copy constructor

```
Movie::Movie(const Movie &c) {
            cout<<"Copy Constructor"<<endl;
```

A3. Complete the implementation of the conversion constructor

```
Movie::Movie(const char *x) {
            cout<<"Conversion Constructor"<<endl;
```

B.(12 points) Write the output of the following program:

```
void main() {
      cout<<"------------------0--------------------"<<endl;
      Movie x;
      cout<<"------------------1--------------------"<<endl;
      Movie a=f1(x);
      x.print();
      a.print();
      cout<<"------------------2--------------------"<<endl;
      Movie b=f2(x);
      x.print();
      cout<<"------------------3--------------------"<<endl;
      Movie c="Spy Game";
      cout<<"------------------4--------------------"<<endl;
      Movie d(x);
      cout<<"------------------5--------------------"<<endl;
```

```
        Movie y=" the lord of the rings";
        Movie e=y;
        y.cap();
        e.print();
        y.low();
        cout<<"------------------6--------------------"<<endl;

}
```

OUTPUT

```
cout<<"------------------0--------------------"<<endl;

cout<<"------------------1--------------------"<<endl;
cout<<"------------------2--------------------"<<endl;

cout<<"------------------3--------------------"<<endl;


cout<<"------------------4--------------------"<<endl;


cout<<"------------------5--------------------"<<endl;


cout<<"------------------6--------------------"<<endl;
```

C. (2 points) The following program has a run time error. Explain why this error occurs.

```
void main() {

        Movie f;
        f=y;
        y.cap();
        f.print();
        y.low();
}
```

Answer:

Question 4 : A stack is an abstract data type with two basic operations: insert a new element to the stack (push) and remove the element that was most recently inserted to the stack (pop). In this question you are given a linked-list implementation of a stack and you have to fill in the push and pop member functions. Each push operation should allocate a new node and each pop operation should deallocate the corresponding node. You are given the definitions for class **node** and **stack**.

```
class node {
```

```
public:

      int item;
      node* next;
      node(int x, node* t) {item=x; next=t;}
};

typedef node* nodePtr;

class stack {
private:
      nodePtr top;
public:
      stack() {top=0;}
      void empty () const{
            if (top==0) cout<<"true"<<endl; else cout<<"false"<<endl;}
      void push(int element){
      ------------//to  be filled-----------------

      int pop(){
      ------------//to  be filled-----------------

};
```

The following program demonstrates the use of the stack:

```
void main() {
      stack d;
      d.empty();
      d.push(5);
      d.push(6);
      cout<<d.pop()<<endl;
      d.empty();
      cout<<d.pop()<<endl;
      d.empty();
}
```

OUTPUT

```
   true
   6
   false
   5
   true
```

Write the implementation of `void push(int element):`

Write the implementation of `int pop():`

Question 5

A. For a class of 100 students, we put their exam grades into an integer array:

```
      int grade[100];
```

Then we would like to sort the grades using the Quick Sort function that you used in your Program Assignment#1:

```
void qsort(void* data, int num_data, size_t data_size,
         int (compare*)(void*, void*));
```

A1. Write one C++ statement to show how you would use the above *qsort* function to sort the 100 grades, assuming an appropriate compare function, called *int_compare*, that you will supply to *qsort*().

A2. Write C++ codes for the *int_compare* function.

(b) One drawback of the design in part (a) is that students' grades and names are separated and after the grades are sorted, we do not know who has the highest grade etc. A better solution is to use a structure to hold both the grade and name of a student as follows:

```
class Record
{
  public:
    int grade;          // student's grade
    char name[32];      // student's name
};
```

B. Assuming now we have an array of Record as:

```
Record record[100];
```

and all the grades and names are entered. Repeat both part A1 and A2 of part (A). That is:

B1 Write one C++ statement to show how you would use the above *qsort* function to sort the 100 records according to the value of the grade, assuming an appropriate compare function, called *record_compare*.

B2 Write C++ codes for the *record_compare* function.

Question 6

Each sub-question contains a complete or incomplete C++ program with at least one error. For each question, first point out which line(s) is/are wrong and then write the correct C++ codes for that/those line(s). Additional information about each program segment is given as C++ comments.

Here are some rules to keep while you make the corrections:

- The meaning of each program segments should be obvious and you have to keep their meanings.
- You should make as few changes as possible.
- After your modification, the program is supposed to compile and run.
- You are NOT allowed to simply delete any lines.

Let's see an example:

```
1. #include {iostream.h}
2. int main()
3  {
4.   int y = 5;
5.   cout << y << endl;
6.   return 0;
7  }
```

For the example, your answer should be:
Answer:

```
i.  incorrect line: 1
ii. line 1 should be:
     #include <iostream.h>
```

(a)

```
1. // in an include file called "example.h"
2. #if_not_define EXAMPLE_H
3. #let_us_define EXAMPLE_H
4. .... // some C++ codes
5. #end_if
```

Answer:

(b)

```
1.  class X
```

```
2.  {
3.    private:
4.    int value;
5.    const X* next;
6.    public:
7.    X X()
8.    {
9.        value = 0;
10.       next = 0;
11.   }
12.   void X::add(const X& y)
13.   {
14.        next = y;
15.   }
16.   void intrude(int a) const
17.   {
18.        if (next) { next->value = a; }
19.   }
20. }
```

Answer:

(c)

```
1.  #include <iostream.h>
2.  class Y
3.  {
4.    int value;
5.    public:
6.      Y(int a = 1) { value = a; }
7.    void print() const { cout << value << endl; }
8.  };
9.  int main()
10. {
      // Create an array of 100 Y objects
11.   Y* many = new Y(5) [100];
12.   cout << y.value << endl;
12.   return 0;
13. }
```

Answer: