THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Department of Computer Science and Engineering

COMP2012: Object-Oriented Programming and Data Structures (Spring 2011/12)

Midterm Examination

Date: Saturday, 17 March 2012

Time: 3:00pm – 5:00pm (2 hours)

Rules:

- This is a closed-book and close-note examination
- Write your answers in the space provided. Your answers will be graded according to correctness, efficiency, precision, and clarity
- During the examination, you must put aside your calculators, mobile phones, PDAs and all other electronic devices. In particular, all mobile phones should be turned off completely
- This booklet has ?? single-sided pages. Please check that all pages are properly printed before you start working
- You may use the reverse side of the pages for your rough work or continuation of work

Student Name	
Student ID	
ITSC email	_____ @stu.ust.hk
I declare that I have not violated the Academic Honor Code in this examination Your signature: _____	

Question	Score / Max. Score
1	/ 20
2	/20
3	/20
4	/40
Total	/100

1. Multiple-choice Questions (20 marks)

Write down your answers in the boxes provided below. Each question has exactly one correct answer. Each correct answer is worth 2 marks

(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)	(j)
B	C	D	D	D	B	D	D	C	A

(a) Which of the followings is true about Arrays in C++?

A) They cannot have more than 10,000 elements
B) All elements in an array must have the same type
C) They must be initialized when you define an array
D) The elements in an array are always automatically initialized as zero.

(b) Which of the followings is a correct way to dynamically allocate an array of 100 integers

A) int arr[100];
B) int arr[100]; int* p = arr;
C) int* p = new int[100];
D) int* p[100]; for (int i=0; i<100; i++) p[i] = new int;

(c) Given the following function *foo*, what is the return value of *foo(15)*?

```
int foo(int n) {
   if ( n > 0 )
       return foo(n/2)+1;
   return 0;
}
```

A) 1
B) 2
C) 3
D) 4

(d) Given the following C++ class definition:

```
class Test {
private:
    int a;
public:
    int b;
};
int main() {
   Test testObj;
   // Add an expression here
   return 0;
}
```

Which of the following(s) will cause compilation error when an expression is added to the main function?

A) testObj.a = 10;
B) testObj.b = 10;
C) Test.b = 10;
D) Both (A) and (C)

(e) A node in a doubly linked list contains:

A) The data value
B) A pointer to the next node
C) A pointer to the previous node
D) All of the above

(f) Suppose the number of items is known. Which of the following type of data structures requires a full traversal for insertion at the tail?

A) An array
B) A singly linked list, with only a head pointer
C) A doubly linked list, with both head and tail pointers
D) A circular, doubly linked list, with only a head pointer

(g) Which of the following function prototypes causes a compilation error?

A) void func(int a, int b = 0);
B) void func(int, int);
C) void func(int a, int b = 0, int c = 5);
D) void func(int a, int b = 0, int c);

(h) Which of the followings is FALSE about constructors?

A) A constructor has no return type
B) A class constructor is always called when an object of that class is created.
C) We can define more than one constructor for a class
D) A compilation error will occur if we do not initialize all member variables in a constructor

(i) Which of the followings is FALSE about static member functions?

A) They cannot access non-static data members of the class
B) They cannot access non-static member functions of the class
C) They cannot be overloaded
D) They exist independently of any objects of a class

(j) Which of the followings is FALSE about general OOP design principles?

A) Programmers should always implement the default constructor for a class
B) We use private member variables instead of the public ones for data protection
C) Constructors should be used for object initialization
D) Class interface should be separated from class implementation

2. Basic C++: Class, Pointer and Recursion (20 marks)

In this question, you are going to implement a class named *IntArray*, which is used to store an array of integers of size *n*. In addition, a pointer *data* with type *int** is used to dynamically allocate an array of integers. The class definition of IntArray is given below:

```
class IntArray {
public:
      static const int DEFAULT = 0;
       IntArray(int inputN); // (a)
       IntArray(const IntArray& other); // (b)
       ~IntArray(); // (c)
      int sumByLoop(int start, int end); // (d)
      int sumByRecursion(int start, int end); // (e)
private:
       int n;
       int *data;
};
```

(a) (5 marks) Implement the first constructor: ***IntArray(int inputN);*** An array of integers is dynamically created by the given size (inputN). All values in the array should be initialized as DEFAULT.

```
IntArray::IntArray(int inputN) {


      n = inputN ;
      data = new int[n];
      for (int i=0; i<n; i++)
            data[i] = DEFAULT; // or IntArray.DEFAULT


}
```

(b) (5 marks) Implement the copy constructor of IntArray

```
IntArray::IntArray(const IntArray& other) {


      n = other.n ;
      data = new int[n];
      for (int i=0; i<n; i++)
            data[i] = other.data[i];



}
```

(c) (2 marks) Implement the destructor of IntArray. *(Note: No memory leak is allowed)*

```
IntArray::~IntArray() {


      delete [] data;


}
```

(d) (3 marks) Sum the numbers between the *start* index and the *end* index inclusively. You can assume the *start* index and the *end* index are between *0* and *n-1*. In addition, the *start* index is always less than or equal to the *end* index. You MUST use a loop to implement it.

```
int IntArray::sumByLoop(int start, int end) {

   int sum = 0;
   for (int i=start; i<=end; i++)
       sum = sum + data[i];
    return sum;



}
```

(e) (5 marks) Sum the numbers between the *start* index and the *end* index inclusively. You can assume the *start* index and the *end* index are between *0* and *n-1*. In addition, the *start* index is always less than or equal to the *end* index. You MUST use recursion to implement it.

```
int IntArray::sumByRecursion(int start, int end) {

   if ( start == end )    // or start <= end
      return data[start];  // or data[end]

   return data[start]+sumByRecursion(start+1,end);

   // or data[end]+sumByRecursion(start, end-1)




}
```

3. Sorting algorithms (20 marks)

(a) (10 points) Here is a quick sort implementation of an array in the range defined by the left and the right indices. The array is required to sort in ascending order. In each recursive pass, the pivot should be selected in the middle of the given range. A partition algorithm is applied such that values in the left sub-range should be less than the pivot, while values in the right sub-range should be greater than the pivot. After the partition process, two recursive calls will be invoke to sort two sub-ranges. Please fill in the blanks. :

```
void quick_sort(int list[], int left, int right) {
            int pivot, left_arrow, right_arrow;
            left_arrow = left; // Assign the current leftmost index
            right_arrow = right; // Assign the current rightmost index
            pivot = list[(left + right)/2 ]; // item in the middle
            do {  // The partition algorithm
                  // Locate an item from the right which is < pivot
                  while (_list[right_arrow] > pivot_ )// ignore equal sign
                        right_arrow--;
                  // Locate an item from the left which is > pivot
                  while (list[left_arrow] < pivot ) // ignore equal sign
                        left_arrow++;
                  // Make a swap if both items exist
                  if (left_arrow <= right_arrow ) {    // ignore equal sign
                        swap(list[left_arrow], list[right_arrow]);
                        left_arrow++;  // update the left_arrow
                        right_arrow--; // update the right_arrow
                  }
            // until the left_arrow moves beyond the right_arrow
            } while (right_arrow >= left_arrow);       // ignore equal sign
            if (___left < right_arrow_____)            // ignore equal sign
                  quick_sort(list, left,right_arrow);  // Recursive call
            if (left_arrow < right)
                  quick_sort(list, left_arrow, right); // Recursive call
      }
```

-1 mark for each blank

- Ignore the equal sign for all the checking cases

(b) (5 marks) Use MergeSort to sort the array {66, 77, 11, 88, 99, 22, 33, 44, 55}. The initial array and the first pass are given as follows. Please complete all the remaining passes.

Initial Array: 66, 77, 11, 88, 99, 22, 33, 44, 55

After the 1st pass: [66 77] [11 88] [22 99] [33 44] [55]

After the 2nd pass: [11 66 77 88][22 33 44 99][55]

After the 3rd pass: [11 22 33 44 66 77 88 99][55]

The last pass (sorted): [11 22 33 44 55 66 77 88 99]

- Marks will only be given if all the passes are correct

(c) (5 marks) Use InsertionSort to sort the array {34 8 64 51 32 21} in ascending order. The initial array and the first pass are given as follows. Please complete all the remaining passes.

Initial Array: 34 8 64 51 32 21

After 1st pass: 8 34 64 51 32 21

8 34 64 51 32 21

8 34 51 64 32 21

8 32 34 51 64 21

The last pass (sorted): 8 21 32 34 51 64

- Marks will only be given if all the passes are correct

4. Abstract Data Type (ADT) –Vector in 3D (40 marks)

In this question, you are asked to implement **Vector3** - an ADT class (given below) for 3-dimensional vectors. It represents a point *(x, y, z)* in the 3-dimensional space.

```
class Vector3 {
public:
        // This constructor should initialize the vector
        // according to the given parameters (px, py, pz)
        Vector3(double px=0, double py=0, double pz=0);

        // Copy constructor
        Vector3(const Vector3& c);

        // Standard overloaded operator+
        // for the addition of 2 vectors
        Vector3 operator+(const Vector3& v) const;

        // Standard overloaded operator+
        // for the dot product of 2 vectors
        Vector3 operator*(const Vector3& v) const;

        // Calculate the cross product of itself and another
        // vector v which is given by the parameter. (itself × v)
        // The (x, y, z) values of itself will be updated to
        // store the product.
        void cross(const Vector3& v);

        // Overloaded operator[]
        // Suppose A is a Vector3 object
        // A[0] should return the x-coordinate
        // A[1] should return the y-coordinate
        // A[2] should return the z-coordinate
        // For all other values of the index, return 0
        double operator[](const int index) const;

private:
        double x; //stores the x-coordinate
        double y; //stores the y-coordinate
        double z; //stores the z-coordinate
};
```

The vector operations are defined as follows:

Addition of two vectors:

$(x_1, y_1, z_1) + (x_2, y_2, z_2) = (x_1+x_2, y_1+y_2, z_1+z_2)$

For example: $(1, 2, 3) + (6, 4, 2) = (7, 6, 5)$

Dot product of two vectors:

$(x_1, y_1, z_1) \cdot (x_2, y_2, z_2) = (x_1x_2, y_1y_2, z_1z_2)$

For example: $(1, 2, 3) \cdot (6, 4, 2) = (6, 8, 6)$

Cross product of two vectors:

$(x_1, y_1, z_1) \times (x_2, y_2, z_2) = (y_1z_2 - z_1y_2, -x_1z_2 + z_1x_2, x_1y_2 - y_1x_2)$

For example: $(1, 2, 3) \times (6, 4, 2) = (-8, 16, -8)$

(a) (3 marks) Implement the constructor that takes three parameters.

```
Vector3::Vector3(double px, double py, double pz) {
        x = px;
        y = py;
        z = pz;

}
```

(b) (3 marks) Implement the copy constructor.

```
Vector3::Vector3(const Vector3& v) {
        x = v.x;
        y = v.y;
        z = v.z;

}
```

(c) (2 marks) Implement the overloaded `operator+`.

```
Vector3 Vector3::operator+(const Vector3& v) const {
        return Vector3(x + v.x, y + v.y, z + v.z);


}
```

(d) (2 marks) Implement the overloaded `operator*`.

```
Vector3 Vector3::operator*(const Vector3& v) const {
        return Vector3(x * v.x, y * v.y, z * v.z);



}
```

(e) (4 marks) Implement the `cross` member function.

```
void Vector3::cross(const Vector3& v) {



        //must use temporary variables to store the old values first
        double oldX = x;
        double oldY = y;
        double oldZ = z;

        x = oldY * v.z - oldZ * v.y;
        y = -oldX * v.z + oldZ * v.x;
        z = oldX * v.y - oldY * v.x;
}
```

(f) (3 marks) Implement the overloaded `operator[]`.

```
double Vector3::operator[](const int index) const {
        if(index == 0) return x;
        else if(index == 1) return y;
        else if(index == 2) return z;
        else return 0;




}
```

(g) (7 marks) Implement the main function to complete the following tasks. You must use the appropriate member functions for all the tasks.

- Create and initialize the following vectors (Vector3 objects)

Variable name	**Value**
A	(1, 2, 3)
B	(4, 5, 6)
C	Copy from A.

- Compute the sum of A and B. Store the sum to a vector variable D.
- Compute the dot product of A and B. Store the product to a vector variable E.
- Compute the cross product of B and A. (B × A) The result should be stored to B.
- Output the z-coordinate of C to the screen using cout.

```
int main() {

   Vector3 A(1, 2, 3);
   Vector3 B(4, 5, 6);
   Vector3 C(A);
   Vector3 D = A + B;
   Vector3 E = A * B;
   B.cross(A);
   cout << C[2];

   return 0;
}
```

(h) Instead of using 3 double variables, you decide to use a double array to store them.

```
class Vector3 {
public:
        //...class member functions are omitted here...

private:
        //data is a 3-item double array storing the coordinates
        //data[0] should store the x-coordinate
        //data[1] should store the y-coordinate
        //data[2] should store the z-coordinate
        double data[3];
};
```

(1) (3 marks) Update the implementation of the constructor based on the above new design

```
Vector3::Vector3(double px, double py, double pz) {

        data [0] = px;
        data [1] = py;
        data [2] = pz;
}
```

(2) (3 marks) Show the updated implementation of the copy constructor.

```
Vector3::Vector3(const Vector3& v) {

        data [0] = v.data [0];
        data [1] = v.data [1];
        data [2] = v.data [2];
}
```

(3) (4 marks) Do you need to implement your own destructor (say Yes/No)? Please explain why. If it is necessary, please also implement the destructor:

No. It is not necessary to delete the content as the array is not dynamically allocated.

(Zero marks will be given if the student wrote "Yes")

(i) (6 marks) Suppose you want to move the operator+ member function to the global scope. That is, instead of having operator+ overloaded inside the class, you want to overload the global operator+ function:

```
Vector3 operator+(const Vector3& v1, const Vector3& v2);
```

Before you can do that, you need to add an extra line inside the class declaration. What is that line? Please also explain why it is necessary.

(This question is cancelled due to possible confusion. 6 marks are awarded to everyone. Full mark of the midterm is still 100.)

It depends on whether you have the operator[] overloaded in the new design. If you assume that the operator[] is omitted in the new design, then you need:

friend Vector3 operator+(const Vector3& v1, const Vector3& v2);

This is required to enable the global function operator+ to access the private data members of the Vector3 class.

If you assume that the operator[] is still overloaded in the new design, then the “friend statement” is not necessary as the private data can be accessed via the use of that.

--- END OF THE PAPER ---