Part I: Multiple Choice (14 questions, 28 points total)

Answer all of the following questions. Choose the best answer for each question. READ EACH QUESTION CAREFULLY. Record your answer on the accompanying answer sheet. All multiple choice questions are equally weighted. Budget your time so you spend enough on the programming questions at the end.

1.

```
int f1 (int & x, int  y) {
     int temp;
     temp = x;
     x = x+y;
     y = temp;
     return temp;
}

int main () {
     int a=10, b=20;
     f1 (a, b);
     //        line A
...
}
```

What is the value of `b` when we reach *line A* in the main program?

A. 10
B. 20
C. 30
D. Impossible to determine.
E. There is a syntax error: the line before line A should be written `f1 (&a, b);`

2. What is NOT true about constructors?

A. A constructor can initialize public member data of a class.
B. A constructor can initialize private member data of a class.
C. A constructor in a base class can be overridden by the derived class.
D. A constructor cannot have a return type, not even void.
E. Multiple constructors for a class are legal, and they all have the same name.

3. **`IntQueue` is a correctly implemented Queue data structure which stores integers.**

```
IntQueue myQueue;

myQueue.enqueue(6);
int i1 = myQueue.dequeue();

if (myQueue.isEmpty())
    myQueue.enqueue(7);
if (myQueue.size() == 2)
    myQueue.enqueue(9);

myQueue.enqueue(1);
myQueue.enqueue(3);
int i3 = myQueue.dequeue();
```

After the above code is executed, what is stored in `myQueue`? (i.e., in what order does the data appear in the queue, with the front element of the queue on the left and the end element of the queue on the right?)

A. 3 1
B. 7 9 1
C. 7 1
D. 6 1 3
E. 1 3

4. **Suppose you see this in a correctly working program:**

```
Foo * v1 = new Myclass [3];
Foo *v2 = v1;
```

From this alone, you can conclude that:

I. Myclass has a copy constructor.
II. Myclass has a constructor with one argument.
III. A destructor for Myclass has been implemented.

A. III only
B. II only
C. I only
D. I and III
E. None of I, II, or III

5. Below is a listing of possible orders of growth for functions.

1. Exponential
2. Linear
3. Logarithmic
4. N*log(N)
5. Quadratic

What is the correct ordering of these growth functions in decreasing complexity? (first = biggest order of growth, last = smallest order of growth)

A. 1 4 5 3 2
B. 1 5 4 3 2
C. 1 5 3 4 2
D. 1 5 4 2 3
E. None of the above

6. The worst case complexities for MergeSort and QuickSort on data of size N are:

A. MergeSort: O(N)
QuickSort: O(N*logN)
B. MergeSort: O(logN)
QuickSort: O(N*logN)
C. MergeSort: $O(N^2)$
QuickSort: O(N*logN)
D. MergeSort: O(N * logN)
QuickSort: $O(N^2)$
E. MergeSort: $O(N^2)$
QuickSort: $O(N^2)$

7.
```
class A  {
public:
    A() { cout << “A’s constructed.”;  }
    void fun() { cout << “I am A”;  }
};

class B : public A {
public:
    B() { cout << “B’s constructed.”;  }
    void fun() { cout << “I am B”; }
};
```

Suppose the following statements are executed - what is the output?

```
int main()  {
  A* foob;
  A* fab;
  fab = new B;
  fab->fun();
  return 0;
}
```

A. B’s constructed. I am A.
B. A’s constructed. A’s constructed. B’s constructed. I am B.
C. A’s constructed. B’s constructed. I am B.
D. A’s constructed. B’s constructed. I am A.
E. Nothing, the compiler would report an error since you can’t allocate memory of type B to a pointer of type A.

8. Consider the following function `fooz`.

```
void fooz(int N) {
        int x = 0;
        for (int i=0; i < N*N; i++) {
                while (x < N/2) {
                        x = x+2;
                }
                x = 0;
        }
}
```

What is the asymptotic complexity of `fooz` in terms of N?

A. O(N^3)
B. O(N^2 log N)
C. O(N^2)
D. O(N log N)
E. O(N)

9. **Suppose you are given a binary search tree containing more than one node. What is true about the node containing the smallest value in the binary search tree?**

I. It is located in the left subtree of the root.
II. Its left subtree must be empty
III. It cannot be the root

A. I only
B. II only
C. III only
D. I and II
E. None are true

10.
```
void displayData (int count) {
   Stack *m_stacks;
   m_stacks  = new Stack[count];
   for (int i = 0; i < count; i++)
        m_stacks[i].Push(5);
   delete m_stacks;
}
```

Something may (or may not) be incorrect in the function above. Choose the statement below which best describes any error or problem.

A. **new Stack[count];**
is an error, because an array size must be a constant.
B. **delete m_stacks;**
is a memory leak since [] are required when deallocating an array.
C. **m_stacks = new Stack[count];**
is an error, because m_stacks is a pointer, not an array.
D. **m_stacks [i];**
is an error, because m_stacks is a pointer, not an array.
E. **m_stacks [i];**
is an error because m_stacks[i] is uninitialized.
F. No errors are present.

11.
```
#include <iostream>
using namespace std;

class Mammal {
public:
      virtual talk() {cout << "Talking ";}
};

class Cat: public Mammal {
public:
      virtual talk() {cout <<"Meow ";}
};

int main ()
{
      Cat c;
      Mammal m = c;
      Mammal * m_ptr = new Cat;
      m.talk();
      m_ptr->talk();
      return 0;
}
```

What will be output by the program above?

A. Talking Meow
B. Meow Talking
C. Meow Meow
D. Talking Talking
E. None of the above. The program contains a compile error.

12. Suppose you see this in a valid C++ implementation file:

`Foobar * fb = new Fooz;`

Which of the following are possibilities?

I. `Foobar` is the name of a class, and class `Foobar` is derived from the class `Fooz`
II. `Fooz` is the name of a class, and class `Fooz` is derived from the class `Foobar`
III. `Fooz` is the name of a member variable of the class `Foobar`

A. I only
B. II only
C. III only
D. II and III
E. I and II

13.
```
class Plane{
public:
    virtual landing() ;
    virtual int takeoff(  ) = 0;
};
```

Which of the following statements are true based on the code above?

I. It is illegal to declare variables of type `Plane`
II. It is legal to declare an instance of `Plane`, but illegal to call the function `takeoff()`
III. It is illegal to declare variables of type `Plane*`
IV. The function `takeoff` always returns an int equal to 0

A. I only
B. II only
C. III only
D. IV only
E. I and III

14. The following program reads a number from `cin` and invokes the function `secret` on it:

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

void secret(int n) {
  if (n > 1) {
     secret(n-1);
  }
  cout << n << " ";
}

int main() {
   int i;
   cin >> i;
   secret(i);
   return 0;
}
```

Indicate which of the following outputs might possibly be produced by the program, given that the user enters an appropriate number:

I. 1 2 3 4 5
II. 5 4 3 2 1
III. 0 1 2 3 4
IV. 4 3 2 1 0

A. I only
B. II only
C. III only
D. IV only
E. I and III
F. II and IV

Part II. Programming Problem (1 question, 22 points total)

16. (4 parts)

An important aspect of object oriented programming is the concept of code reusability. If you can reuse old code, you don't have to write it again and you can spend the time you saved playing Ultimate. We will assume that the following code has already been written.

```
struct ListNode {
    int m_value;
    ListNode *next;
};

class List {
  public:
       // Constructor
      List( );

       // Returns true if list is empty, false otherwise
      bool isEmpty();

       // Creates a new node containing value and inserts it at the
       // front of the list
      virtual void insert(int value);

       // Removes the 1st node containing value in the list, returns
       // false if value is not present, true otherwise.
      virtual bool remove(int value);

       // Returns true if value is present in list, false otherwise.
       bool find(int value);

       // Returns 1st value in the list, but does not remove the node.
      int returnTop();

       // destructor
      virtual ~List() {deleteList(head);}

  protected:
      ListNode *head;
       void deleteList(ListNode * front);
};
```

a) The `List` destructor above contains one line of code: a call to the `deleteList` method. Recursively implement the `deleteList` method below so that the `List` destructor deallocates all appropriate dynamic memory (you should not change the `List` destructor at all).

```
void List::deleteList(ListNode * front) {
     if (front != NULL) {
          deleteList(front->next);
          delete front;
     }
}
```

Now your assignment is to create a new list class `OrderedList`, based on `List`. You should NOT alter the `List` class specification. This means creating a class derived from `List` that maintains an "ordered list". Your `OrderedList` class should have an `insert` method and a `remove` method which meet the following requirements:

1. **The `remove` method from the `List` class removes from the list a node containing `value`. If the list contains multiple nodes with the `value`, then `remove` will only remove the first occurrence of such a node in the list. The `remove` method for your `OrderedList` class should remove ALL nodes containing `value` in the list.**
2. **The `insert` method from the `List` class places nodes at the head of the list, like a stack. The `insert` method for your `OrderedList` class should insert new nodes in order, e.g. if 5 is entered and the list currently contains 1 6 7, then the new list order should be 1 5 6 7.**

b) Write your declaration of the `OrderedList` class below.

```
class OrderedList: public List {
public:
     OrderedList();
     virtual bool remove(int value);
     virtual void insert(int value);
     ~OrderedList();
};
```

c) Code your implementation of `OrderedList::remove` below. (Hint: you might find it useful to use the `find` method and `remove` method from the `List` class in your implementation.)

```
bool OrderedList::remove(int value) {
     bool temp_return = find(value);
     while (find(value))
          List::remove(value);
     return temp_return;
}
```

d) Code your implementation of `OrderedList::insert` below.

```
void OrderedList::insert(int value) {

     // if list is empty or if value is smallest element of
     // list, then add new node to the front of the list
     if (isEmpty() || (head->m_value > value)) {
          List::insert(value);
     }
     else { // otherwise, find where to insert new node
          ListNode * tempNode = new ListNode;
          tempNode->m_value = value;
          ListNode * prev = head;
          ListNode * curr = head->next;
          while (curr != NULL) {
               if (curr->m_value >= value) {
                    prev->next = tempNode;
                    tempNode->next = curr;
                    return;
               }
               prev = curr;
               curr = curr->next;
          }
          prev->next = tempNode;
          tempNode->next = curr;
     }
}
```