FINAL MOCK EXAM - CS 1124 Spring 2013

Disclaimer: This mock exam is only for practice. It was made by tutors in the Polytechnic Tutoring Center and is not representative of the actual exam given by the CS Department

1. Given: class Base {/*code*/}; and class Derived : public Base {/*code*/};
Consider each question separately and answer YES or NO.

a) If Base has virtual functions then Derived needs to override them all _____

b) Any virtual function in Derived must be virtual in Base _____

c) Base destructor should be virtual _____

e) Any object of type Base is an object of type Derived _____

f) If Base has friend operator<< then it is also a friend of Derived _____

g) Assigning an object of type Base to Derived is called slicing _____

h) Pure virtual functions in Base are PV in Derived (by default) _____

i) System provided Derived copy constructor calls Base copy constructor before executing _____

j) ~Derived always calls ~Base after its body is executed_____

2. Given the recursive function f defined below. What is the output of f (6); ?

```
void f (int x) {
        if (x > 3) {
                f (x/3);
                cout << 'b';
                f (x/2);
                cout << 'c';
                f (x/4);
        }
        cout << 'a';
}
```

Output:

3. What are the 4 main operations that iterators support?

-
-
-

4. Using an STL vector, write a function remove(`vector<int>` theVector, `int` target) that will remove the selected element from the list. Return true if it was found and removed, or false otherwise. Use iterators to solve the problem.

Code:

5. Write a function, throwIfNegativeOne, which takes an integer. If the integer is -1 it should throw an instace of ErrorMessage containing the message: "NEGATIVE ONE". Your function should work with the following code.

```
struct ErrorMessage{
	string message;
};

int main()
{
	int value;
	cin >> value;
	try{
		throwIfNegativeOne(value);
	}catch(ErrorMessage e){
		cout << "Revieved error message: " << e.message << endl;
	}catch(...){
		cout << "An exception occured..." << endl;
	}
}
```

Code:

6. Write a templated function, multiply, that takes any two numbers of the same type (int, float, double, Rational, Complex, etc.) and returns them multiplied together.

Code:

7. **Given:**

```
class Parent
{
protected:
        int a;
public:
        Parent(int x=0): a(x) {}
        void display() { cout << a << " I am the parent.\n";}
};

class Child : public Parent
{
public:
        Child(): Parent(4) {}
        void display () { cout << a << " I am the child.\n";}
};
```

What is displayed by the following code?

```
int main ()
{
        Parent* p = new Child;
        p->display();
}
```

Output:

8. Write a class called NodeItr that can be used to iterate through a list made up of the following Node struct:

```
struct Node
{
    Node(int d = 0, Node* l = NULL) : data(d), link(l) {}
    int data;
    Node* link;
};
```

For this to work, you must write a constructor that takes in a Node pointer, an accessor to get the node data, a mutator to modify the node data, a accessor to determine whether the node is NULL, and overload the pre-increment, post-increment, ==, and != operators. Your code should work with the following main function:

```
int main()
{
    Node* headPtr = new Node;
    Node* tmp = headPtr;

    //add 4 more nodes to the list
    for (int i = 0; i < 4; i++)
    {
        tmp->link = new Node;
        tmp = tmp->link;
    }

    //fill list with values 1,2,3,4,5
    int count = 1;
    NodeItr itr(headPtr);
    while (itr != NULL)
    {
        itr++.setData(count);
              count++;
    }

    for (NodeItr i(headPtr); !i.isNull(); i++)
    {
              cout << i.getData() << endl;
    }
}
```

FINAL MOCK EXAM ANSWER KEY- CS 1124 Spring 2013

Disclaimer: This mock exam is only for practice. It was made by tutors in the Polytechnic Tutoring Center and is not representative of the actual exam given by the CS Department

1. Given: class Base {/*code*/}; and class Derived : public Base {/*code*/};
Consider each question separately and answer YES or NO.

a) If Base has virtual functions then Derived needs to override them all **NO**

b) Any virtual function in Derived must be virtual in Base **NO**

c) Base destructor should be virtual **YES**

e) Any object of type Base is an object of type Derived **NO**

f) If Base has friend operator<< then it is also a friend of Derived **NO**

g) Assigning an object of type Base to Derived is called slicing **NO**

h) Pure virtual functions in Base are PV in Derived (by default) **YES**

i) System provided Derived copy constructor calls Base copy constructor before executing **YES**

j) ~Derived always calls ~Base after its body is executed **YES**

2. Given the recursive function f defined below. What is the output of f (6); ?

```
void f (int x) {
        if (x > 3) {
                f (x/3);
                cout << 'b';
                f (x/2);
                cout << 'c';
                f (x/4);
        }
        cout << 'a';
}
```

Output:
abacaa

3. What are the 4 main operations that iterators support?

- assignment (operator=)
- incrementation (operator++)
- equality checking (operator==)
- dereferencing (operator*)

4. Using an STL vector, write a function remove(`vector<int>` theVector, `int` target) that will remove the selected element from the list. Return true if it was found and removed, or false otherwise. Use iterators to solve the problem.

Code:

```
bool remove(vector<int>& theVector, int target)
{
        for (vector<int>::iterator i = theVector.begin(); i != theVector.end(); i++)
        {
                if (*i == target)
                {
                        *i = theVector[theVector.size()-1];
                        theVector.pop_back();
                        return true;
                }
        }
        return false;
}
```

5. Write a function, throwIfNegativeOne, which takes an integer. If the integer is -1 it should throw an instace of ErrorMessage containing the message: "NEGATIVE ONE". Your function should work with the following code.

```
struct ErrorMessage{
	string message;
};

int main()
{
	int value;
	cin >> value;
	try{
		throwIfNegativeOne(value);
	}catch(ErrorMessage e){
		cout << "Revieved error message: " << e.message << endl;
	}catch(...){
		cout << "An exception occured..." << endl;
	}
}
```

Code:

```
void throwIfNegativeOne(int num) throw (...){
	if(num == -1){
		ErrorMessage e;
		e.message = "NEGATIVE ONE";
		throw e;
	}
}
```

6. Write a templated function, multiply, that takes any two numbers of the same type (int, float, double, Rational, Complex, etc.) and returns them multiplied together.

Code:

```
template<class T>
T multiply( T num1, T num2){
	return num1 * num2;
}
```

7. Given:

```
class Parent
{
protected:
        int a;
public:
        Parent(int x=0): a(x) {}
        void display() { cout << a << " I am the parent.\n";}
};

class Child : public Parent
{
public:
        Child(): Parent(4) {}
        void display () { cout << a << " I am the child.\n";}
};
```

What is displayed by the following code?

```
int main ()
{
        Parent* p = new Child;
        p->display();
}
```

Output:
4 I am the parent.
(It calls Parent's display, it isn't virtual!)

8. Write a class called NodeItr that can be used to iterate through a list made up of the following Node struct:

```
struct Node
{
    Node(int d = 0, Node* l = NULL) : data(d), link(l) {}
    int data;
    Node* link;
};
```

For this to work, you must write a constructor that takes in a Node pointer, an accessor to get the node data, a mutator to modify the node data, a accessor to determine whether the node is NULL, and overload the pre-increment, post-increment, ==, and != operators. Your code should work with the following main function:

```
int main()
{
    Node* headPtr = new Node;
    Node* tmp = headPtr;

    //add 4 more nodes to the list
    for (int i = 0; i < 4; i++)
    {
        tmp->link = new Node;
        tmp = tmp->link;
    }

//fill list with values 1,2,3,4,5
int count = 1;
NodeItr itr(headPtr);
while (itr != NULL)
{
    itr++.setData(count);
          count++;
}

for (NodeItr i(headPtr); !i.isNull(); i++)
{
          cout << i.getData() << endl;
}
}
```

Code:

```
class NodeItr
Code:
{
friend bool operator==(const NodeItr& lhs, const NodeItr& rhs);
public:
    NodeItr(Node* n) : node(n) {}

       void setData(int value){
              node->data = value;
       }

       int getData() const {
              return node->data;
       }

       bool isNull() const {
              return node == NULL;
       }

   NodeItr operator++(int i) //post-increment
       {
        NodeItr tmp(node);
        if (node)
        {
            node = node -> link;
        }
        return tmp;
   }

       NodeItr& operator++() //pre-increment
       {
            if (node)
            {
                node = node -> link;
            }
            return *this;
       }

private:
    Node* node;
};

bool operator==(const NodeItr& lhs, const NodeItr& rhs)
{
    return lhs.node == rhs.node;
}

bool operator!=(const NodeItr& lhs, const NodeItr& rhs)
{
    return !(lhs==rhs);
}
```