Homework 5

Yuji Shimojo

CMSC 330

Instructor: Prof. Reginald Y. Haseltine

July 13, 2013

HOMEWORK 5

Question 1

Consider the following Java definition of a mutable string class.

```
class MutableString
{
   private char[] chars = new char[200];
   private int size = 0;
   public boolean set(char aChar, int index)
   {
      if (index < 0 || index >= chars.length)
         return false;
      chars[index] = aChar;
      return true;
   }
   public String toString()
   {
      String result = "";
      for (int i = 0; i < size; i++)
         result += chars[i];
      return result;
   }
}
```

Suppose this class was rewritten in C++ in two ways, the first with the array that represents the list as a stack-dynamic variable, and then with the list as a heap dynamic variable.

HOMEWORK 5

Explain when constructors and destructors would be needed. Explain why no constructors or destructors are needed in the Java implementation.

Answer 1

First, I ran the following Java class, which I added a main method.

MutableString Program in Java

```
class MutableString
{
   public static void main(String [] args)
   {
      MutableString mutableString = new MutableString();
      System.out.println(mutableString.set('A', 0));
      System.out.println(mutableString.toString());
   }

   private char[] chars = new char[200];
   private int size = 1;
   public boolean set(char aChar, int index)
   {
      if (index < 0 || index >= chars.length)
         return false;
      chars[index] = aChar;
      return true;
   }
   public String toString()
```

```
    {
        String result = "";
        for (int i = 0; i < size; i++)
            result += chars[i];
        return result;
    }
}
```

Output

```
true
A
```

I rewrote the above program in C++ by two approaches as below. The first is using stack-dynamic variables, and the second is using heap dynamic variables, so that it allocates heap memory dynamically when the class objects is instantiated.

MutableString Program in C++ #1

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

char chars[200];
int size = 1;

bool set(char aChar, int index)
{
```

```
    if (index < 0 || index >= sizeof(chars))
        return false;
    chars[index] = aChar;
    return true;
}
string toString()
{
    string result = "";
    for (int i = 0; i < size; i++)
        result += chars[i];
    return result;
}

int main (int argc, const char * argv[])
{
    cout << set('A',0) << endl;
    cout << toString() << endl;
    return 0;
}
```

Output

```
1
A
```

MutableString Program in C++ #2

```
#include <iostream>
```

HOMEWORK 5

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


class MutableString
{
private:
   char chars[200];
   int size;
public:
   // constructor
   MutableString()
   {
      size = 1;
      cout << "constructor is called!" << endl;
   }
   // destructor
   ~MutableString()
   {
      cout << "destructor is called!" << endl;
   }
   bool set(char aChar, int index)
   {
      if (index < 0 || index >= sizeof(chars))
         return false;
      chars[index] = aChar;
      return true;
   }
```

```
    string toString()
    {
        string result = "";
        for (int i = 0; i < size; i++)
            result += chars[i];
        return result;
    }
};

int main (int argc, const char * argv[])
{
    MutableString mutableString;
    cout << boolalpha << mutableString.set('A',0) << endl;
    cout << mutableString.toString() << endl;
    return 0;
}
```

Output

```
constructor is called!
true
A
destructor is called!
```

The output of the second program shows when the constructor and the destructor are called. Constructors are needed when the object is instantiated, and destructors are needed when the scope of the object is terminated.

HOMEWORK 5

Question 2

Consider the following C++ template class.

```
template <typename T, int length>
class Vector
{
public:
   Vector(T values[length])
   {
      for (int i = 0; i < length; i++)
         list[i] = values[i];
   }
   friend bool operator<(const Vector<T, length>& left, const Vector<T, length>& right)
   {
      bool result = true;
      for (int i = 0; i < length; i++)
         result &= left.list[i] < right.list[i];
   return result;
   }
private:
   T list[length];
};

int main()
{
```

```
    int first[] = {1, 2}, second[] = {2, 3};
    Vector<int, 2> vector1(first), vector2(second);
    cout << (vector1 < vector2) << endl;
    return 0;
}
```

The class Vector cannot be instantiated for any arbitrary type. For example, consider the following instantiation for a wrapper integer class.

```
class Int
{
public:
    Int(int i = 0) {this->i = i;}
private:
    int i;
};

int main()
{
    Int first[] = {Int(1), Int(2)}, second[] = {Int(2), Int(3)};
    Vector<Int, 2> vector1(first), vector2(second);
    cout << (vector1 < vector2) << endl;
    return 0;
}
```

Explain why the second implementation fails. What must be added to that class so this program will compile? Suppose this program were written in Java. Explain how Java allows the constraints on a generic type parameter to be specified and how they would be specified in this case

Java does have one limitation, however. Although wrapper classes can be used to instantiate generic type parameters, primitive types cannot. Explain why.

Answer 2

Output of the First Implementation

```
true
```

The second implementation fails because the operands of friend bool operator are Vector objects. When the operator is used in main method, the type of arguments is indicated as integer.

I created binomial comparison operating program in Java by using Vector class, generics, and compareTo method as below.

GenericTypeParameterTest class

```
import java.util.Vector;
public class GenericTypeParameterTest
{
   public static void main(String[] args)
   {
```

```
        int resultInt=0;
        // Instantiates a generic type of Vector object
        Vector<String> strings = new Vector<String>();
        strings.add("abc");
        strings.add("def");
        resultInt = strings.elementAt(0).compareTo(strings.elementAt(1));
        System.out.println(isLeftSmall(resultInt));

        // Instantiates a generic type of Vector object
        Vector<Integer> integers = new Vector<Integer>();
        integers.add(1);
        integers.add(2);
        resultInt = integers.elementAt(0).compareTo(integers.elementAt(1));
        System.out.println(isLeftSmall(resultInt));
    }
    // compareTo method which returns a boolean value
    static boolean isLeftSmall(int resultInt)
    {
        boolean result = true;
        if (resultInt < 0)
            result = true;
        else if (resultInt > 0)
            result = false;
        return result;
    }
}
```

HOMEWORK 5

Output

true

true