Thursday, 07-10-2014

SWE315 : C++

Lesson 10 - MidTerm

Notes:

- **Open material** - Books, homework. Internet just for looking into reference material.
  - Example websites: http://www.cplusplus.com/reference/
  - For ++ for example: http://en.cppreference.com/w/cpp/language/operators or http://www.tutorialspoint.com/cplusplus/increment_decrement_operators_overloading.htm
- **Timed exam** – If you need accommodation, let me know beforehand.

Solution:

1. Please send solution to: zbaharav@cogswell.edu
2. You know the drill by now: Simply hit reply, and no zipped directories etc..
3. You will need to attach this word document, where at the bottom (see place for this) you paste
   a. Fib1.h
   b. Fib1.cpp
   c. Screen shots of the console output.
   d. In addition, you will need to fill-in the answer for the two-questions below in the appropriate place. See the 'green highlighted' area where you need to fill-in.

====

(see also hints-and-guidelines below!)

Question 1 (80pts)

Create a class called *Fib1* , which creates and holds a basic Fibonacci series (http://en.wikipedia.org/wiki/Fibonacci_number) . Our series will start with 0,1, and every subsequent element is the sum of the previous two.
** It is important (for your default constructor and otherwise), that the minimum length of series allowed is 2 elements. If the user requests a series shorter than 2, a length 2 series will be created.
The class should have the following variables and interfaces:

a. Variables:
   i. Integer pointer *vec* .
   ii. Integer variable *len* (holding the length of vec).
b. Interfaces:
   i. Constructor :
      1. Empty constructor.

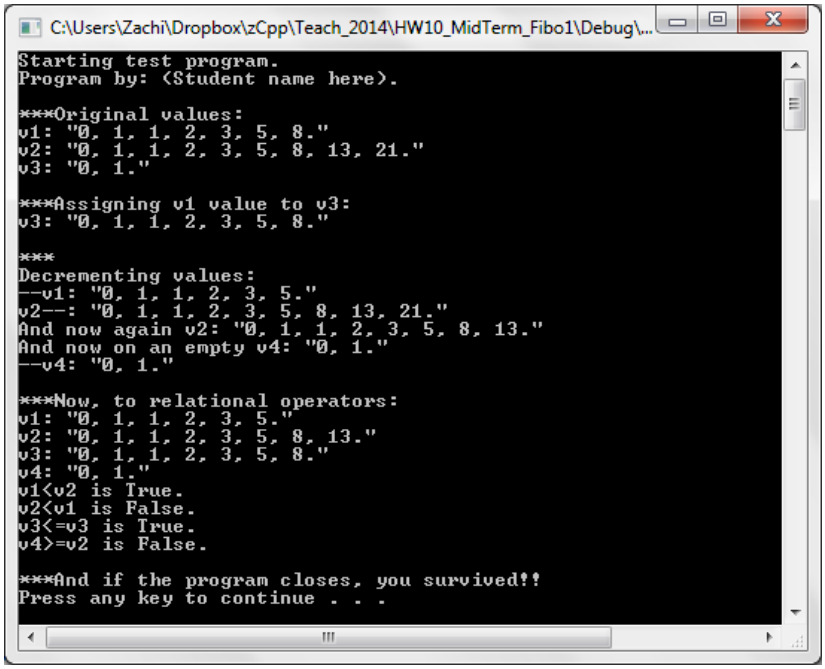
2. Constructor given an integer, length of series:
```
Fib1(int n);
```
3. Copy constructor:
```
Fib1(const Fib1& v);
```
4. Assignment operator:
```
Fib1& operator=(const Fib1& v);
```

ii. Destructor, to clear memory allocated.

iii. Overload the following operators:
1. <<
2. -- (prefix) : This operator subtracts one element from the Fibonacci series. Keep in mind: The minimum length should be 2! (see at the opening specs)
3. -- (postfix) <-- This is MORE challenging, so you may wish to leave this for the very end.
4. Relational operators: < , > , <= , >= : These operators determine the relation according to the length of the series. Note/hint (which you do NOT have to use): There is a very simple way to implement these using only one function that actually does anything…

Please use the following test-program to validate your solution, and produce the screen shot as below. (you will need to change the 'student name').

Screen shot :



Test program:

```
// main.cpp
// Test program for Fib1.cpp


#include <iostream>
#include "Fib1.h"

using std::cout;

int main(void)
{
	{
		Fib1 v1(7);
		Fib1 v2(9);
		Fib1 v3(0);
		Fib1 v4;

		cout << "Starting test program.\n";
		cout << "Program by: (Student name here).\n";
		cout << "\n***Original values:\n";

		cout << "v1: \"" << v1 << "\"\n";
		cout << "v2: \"" << v2 << "\"\n";
		cout << "v3: \"" << v3 << "\"\n";

		cout << "\n***Assigning v1 value to v3:\n";
		v3= v1;
		cout << "v3: \"" << v3 << "\"\n";

		/*
		cout << "\n***Incrementing values:\n";
		cout << "++v1: \"" << ++v1 << "\"\n";
		cout << "v2++: \"" << v2++ << "\"\n";
		cout << "and now again v2: \"" << v2 << "\"\n";
		cout << "\n***And trying it on empty Fib:\n";
		cout << "++v4: \"" << ++v4 << "\"\n";
		*/

		cout << "\n***\nDecrementing values:\n";
		cout << "--v1: \"" << --v1 << "\"\n";
		cout << "v2--: \"" << v2-- << "\"\n";
		cout << "And now again v2: \"" << v2 << "\"\n";
		cout << "And now on an empty v4: \"" << v4 << "\"\n";
		cout << "--v4: \"" << --v4 << "\"\n";



		cout << "\n***Now, to relational operators:\n";
		cout << "v1: \"" << v1 << "\"\n";
		cout << "v2: \"" << v2 << "\"\n";
		cout << "v3: \"" << v3 << "\"\n";
		cout << "v4: \"" << v4 << "\"\n";
		cout << "v1<v2 is " << ((v1<v2)? "True":"False") << ".\n";
		cout << "v2<v1 is " << ((v2<v1)? "True":"False") << ".\n";
		cout << "v3<=v3 is " << ((v3<=v3)? "True":"False") << ".\n";
```

```
        cout << "v4>=v2 is " << ((v4>=v2)? "True":"False") << ".\n";

    }

    cout << "\n***And if the program closes, you survived!!\n";
    system("pause");

    return 0;
}
```

Hints and guidelines:

1. You should know how to create the simple class with variables and methods.
2. You will need to dynamically allocate a 1D array.
3. Remember to clear this space in the destructor.
4. Important:
    a. Implement the simple things first.
    b. Comment stuff from main.cpp when you only implement parts.
    c. Only after you did well with the assignment operator, head into the ++ overloading.

Question 2 (10pts)

- Will the following program compile?
- Will the following program run?
- If it will run, what is the output?

```
#include <iostream>

int sum2(int& a, int& b)
{
    a = 3;
    b = 4;
    return a+b;
}

int main(void)
{
    int a=1;
    int b=2;

    int c = sum2(a,a);
    std::cout << a << b << c ;

    return 0;
}
```

Answer (**please explain your answers**):

B is not touched during the whole process. Only a is transferred as reference.
So in sum2, a and b reference the SAME variable.

So this variable is initially set to 3, then it is set to 4, and then it is summed with itself ➔ returning 4+4, which is 8, and assigned in c.

Thus, the printout would be : 428

Question 3 (10pts)

- Will the following program compile?
- Will the following program run?
- If it will run, what is the output?

```
#include <iostream>

class A {
public:
  A() { std::cout << 'a'; }
  ~A() { std::cout << 'A'; }
};

class B {
public:
  B() { std::cout << 'b'; }
  ~B() { std::cout << 'B'; }
};

int main() { B b; A a; return 0;}
```

Answer **(please explain your answers)**:

First B is created ➔ b
Then A is created ➔a
Then things are destructed in reverse order (we saw it in our String1 class program):
➔ first A and then B

Thus, the printout is: baAB

=== End of MidTerm ====

```
// Fib1.h

#ifndef FIB1_H_
#define FIB1_H_

#include <iostream>
```

```
class Fib1
{
private:
	int* vec;
	int len;

public:
	// Constructors
	Fib1();
	Fib1(int n);
	Fib1(const Fib1& v);		// Copy constructor
	Fib1& operator=(const Fib1& v);	// Assignemnt operator

	// Destructor
	~Fib1();

	// Overloaded
	friend std::ostream& operator<<(std::ostream& os, const Fib1& v);
	Fib1 operator++();			// ++ prefix
	Fib1 operator++(int);		// postfix ++

	inline bool operator<(const Fib1& rhs) const { return (this->len < rhs.len)
;}
	inline bool operator>(const Fib1& rhs) const {return (rhs < *this);}
	inline bool operator<=(Fib1& rhs) const {return !(*this > rhs);}
	inline bool operator>=(Fib1& rhs) const {return !(*this < rhs);}

};
#endif
```

// Fib1.cpp

```
#include "Fib1.h"
#include <iostream>

using std::cout;
using std::endl;

// Constructors

// Default constructor
Fib1::Fib1()
{
	len = 2;
	vec = new int[len];
	vec[0] = 0  , vec[1] = 1;
}

// specified length constructor
Fib1::Fib1(int n)
{
	len = (n<2) ? 2 : n;

	vec = new int[len];
	vec[0] = 0  , vec[1] = 1;
```

```
	for (int ii=2; ii < len; ++ii)
		vec[ii] = vec[ii-1]+vec[ii-2];
}

// Copy constructor
Fib1::Fib1(const Fib1& v)
{
	len = v.len;
	vec = new int[len];
	for (int ii=0; ii < len; ++ii)
		vec[ii] = v.vec[ii];
}

// Assignemnt operator
Fib1& Fib1::operator=(const Fib1& v)
{
	if (this == &v)
		return *this;
	delete [] vec;
	len = v.len;
	vec = new int[len];
	for (int ii=0; ii < len; ++ii)
		vec[ii] = v.vec[ii];
	return *this;
}

// Destructor
Fib1::~Fib1()
{
	delete [] vec;
}

// Overloaded
std::ostream& operator<<(std::ostream& os, const Fib1& v)
{
	for (int ii=0; ii<v.len-1 ; ++ii)
		os << v.vec[ii] << ", ";
	os << v.vec[v.len-1] << ".";
	return os;
}

// ++ prefix
Fib1 Fib1::operator++()
{

	// Clear the old one
	delete [] vec;
	len++;

	vec = new int[len];
	vec[0] = 0  , vec[1] = 1;
	for (int ii=2; ii < len; ++ii)
		vec[ii] = vec[ii-1]+vec[ii-2];

	return *this;
}

// postfix ++
```

```
Fib1 Fib1::operator++(int)
{
	Fib1 V(*this);

	++(*this);

	return V;

}
```

=== End of solution