Swinburne University Of Technology

Faculty of Science, Engineering and Technology

ASSIGNMENT COVER SHEET

Subject Code: COS30008
Subject Title: Data Structures & Patterns
Assignment number and title: 4 - Iterators
Due date: April 15, 2014, 10:30 a.m.
Lecturer: Dr. Markus Lumpe

Your name:_____ **Your student id:**_____

Check Tutorial	Fri 10:30	Fri 12:30	Fri 14:30

Marker's comments:

Problem	Marks	Obtained
1	42	
Total	42	

Extension certification:

This assignment has been given an extension and is now due on _____

Signature of Convener:_____

Problem Set 4: Iterators

 [1]

In mathematics, the *Fibonacci numbers* (or *Fibonacci sequence*) are positive numbers in the following sequence

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181, 6765, …

For $n \geq 3$, we can define this sequence recursively by

$$\text{Fibonacci}(n) = \text{Fibonacci}(n - 1) + \text{Fibonacci}(n - 2),$$

with seed values

$$\text{Fibonacci}(1) = 1 \text{ and } \text{Fibonacci}(2) = 1.$$

Fibonacci numbers appear in numerous places, including computer science and biology. Unfortunately, evaluating a Fibonacci sequence for a given n in a recursive and bottom-up fashion is computationally expensive and may exceed available resources (in terms of both space and time). The recursive definition calculates the smaller values of Fibonacci(n) first and then builds larger values from them.

An alternative mathematical formulation of the Fibonacci sequence is due to *dynamic programming*, a technique developed by Richard E. Bellmann in the 1940s while working for the RAND corporation. Dynamic programming uses memorization to save values that have already been calculated. This yields a top-down approach that allows Fibonacci(n) to be split into sub-problems and then calculate and store values. This method yields a very efficient iterative algorithm to generating the Fibonacci sequence.

The iterative formulation of the Fibonacci sequence uses two storage cells, `previous` and `current`, to keep track of the values computed so far:

```
Fibonacci( n ) =
  previous := 0;
  current := 1;
  for i := 1 to n do
   next := current + previous;
   previous := current;
   current := next;
   end;
```

For $n \geq 1$, this algorithm produces the desired sequence in linear time, but only requiring constant space.

[1] Source: http://en.wikipedia.org/wiki/File:Fibonacci.png

Problem 1: Class FibonacciIterator

Using the dynamic programming solution we can construct a C++ iterator that produces the Fibonacci sequence up to a given n. This iterator implements the standard interface for a C++ forward iterator. The following class specification suggests a possible solution:

```
#pragma once

class FibonacciIterator
{
private:
  long fMaxN;          // maximum n
  long fCurrentN;      // current n
  long fPrevious;      // previous Fibonacci number
  long fCurrent;       // current Fibonacci number

public:

  // Default constructor to set up Fibonacci sequence
  FibonacciIterator( long aMaxN );

  // iterator methods

  const long& operator*() const;          // return current Fibonacci number
  FibonacciIterator& operator++();        // prefix, next Fibonacci number
  FibonacciIterator operator++( int );    // postfix (extra unused argument)
  bool operator==( const FibonacciIterator& aOther ) const;
  bool operator!=( const FibonacciIterator& aOther) const;

  // auxiliaries
  FibonacciIterator begin() const;        // return new iterator positioned at n==1
  FibonacciIterator end() const;          // return new iterator positioned at n+1
};
```

The iterator requires four member variables. The values `fPrevious` and `fCurrent` serve are the storage cells to compute the Fibonacci sequence. The values `fMaxN` and `fCurrentN` denote the target n and the current n, respectively. For Fibonacci(n), initially `fMaxN` is set to n and `fCurrentN` to 1, the start. In each iteration (i.e., iterator increment) the value of `fCurrentN` is increased by 1 until the iterator reaches the position `fCurrentN == fMaxN`.

The implementation of `FibonacciIterator` follows standard practice and is similar to the `IntArrayIterator` and the `CharacterCounterIterator` studied in class and tutorials.

Build the C++ console application, called `FibonacciIterator`, that takes one argument (i.e., a number string) and outputs the corresponding Fibonacci sequence to the console screen. Use the following main function in your application (see C++ reference for details on `atoi`):

Main & Test

```
#include <iostream>
#include <cstdlib>

#include "FibonacciIterator.h"

using namespace std;

int main( int argc, char *argv[] )
{
  if ( argc < 2 )
  {
    cerr << "Missing argument!" << endl;
    cerr << "Usage: FibonacciIterator number" << endl;
    return 1;
  }

  cout << "Fibonacci sequence up to " << argv[1] << endl;

  FibonacciIterator lIterator( atoi( argv[1] ) );

  for ( ; lIterator != lIterator.end(); lIterator++ )
    cout << *lIterator << endl;

  cout << "Once more:" << endl;

  FibonacciIterator lIterator2 = lIterator.begin();

  for ( ; lIterator2 != lIterator2.end(); lIterator2++ )
    cout << *lIterator2 << endl;

  return 0;
}
```

Result (command line argument: 20):

```
Fibonacci sequence up to 20
1
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
1597
2584
4181
6765
Once more:
1
1
2
3
5
8
13
21
```

```
34
55
89
144
233
377
610
987
1597
2584
4181
6765
```

The final program will require approx. 80 lines of code including comments and very spacious formatting.

Submission deadline: Tuesday, April 15, 10:30 a.m.

Submission procedure: on paper, code of class `FibonacciIterator`.