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: 2 - Iterators
Due date: Monday, 22 April, 2024, 10:30
Lecturer: Dr. Markus Lumpe

Your name: Vu Duc Tran **Your student id:** 104175614

Marker's comments:

Problem	Marks	Obtained
1	40	
2	70	
Total	110	

Extension certification:

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

Signature of Convener:_____

FibonacciSequenceGenerator.cpp

```
//
//  FibonacciSequenceGenerator.cpp
//  problem2
//
//  Created by Vu Duc Tran on 21/4/2024.
//
#include <cassert>
#include "FibonacciSequenceGenerator.h"
using namespace std;
FibonacciSequenceGenerator::FibonacciSequenceGenerator(const std::string& aID) noexcept :
fID(aID), fPrevious(0), fCurrent(1) {}

// Get sequence ID
const std::string& FibonacciSequenceGenerator::id() const noexcept {
   return fID;
}

// Get current Fibonacci number
const long long& FibonacciSequenceGenerator::operator*() const noexcept {
   return fCurrent;
}

// Type conversion to bool
FibonacciSequenceGenerator::operator bool() const noexcept {
   return hasNext();
}

// Reset sequence generator to first Fibonacci number
void FibonacciSequenceGenerator::reset() noexcept {
   fPrevious = 0;
   fCurrent = 1;
}

// Tests if there is a next Fibonacci number.
// Technically, there are infinitely many Fibonacci numbers,
// but the underlying integer data type limits the sequence.
bool FibonacciSequenceGenerator::hasNext() const noexcept {
   // Check if the next Fibonacci number is representable
   return fCurrent + fPrevious >= fCurrent;
}
```

```
// Advance to next Fibonacci number
// Function performs overflow assertion check.
void FibonacciSequenceGenerator::next() noexcept {
   assert(fCurrent >= 0 && "Overflow condition reached");
   long long nextFibonacci = fPrevious + fCurrent;
   fPrevious = fCurrent;
   fCurrent = nextFibonacci;
}
```

FibonacciSequenceIterator.cpp

```
//
//  FibonacciSequenceIterator.cpp
//  problem2
//
//  Created by Vu Duc Tran on 21/4/2024.
//

#include <cassert>
#include "FibonacciSequenceIterator.h"

FibonacciSequenceIterator::FibonacciSequenceIterator(const FibonacciSequenceGenerator&
aSequenceObject, long long aStart) noexcept : fSequenceObject(aSequenceObject),
fIndex(aStart - 1) {}

const long long& FibonacciSequenceIterator::operator*() const noexcept {
   return *fSequenceObject; // Return the current Fibonacci number
}

FibonacciSequenceIterator& FibonacciSequenceIterator::operator++() noexcept {
   if (!fSequenceObject.hasNext()) {
      fIndex = -1;
   }
   else {
      ++fIndex;
      fSequenceObject.next();
   }
   return *this;
}

FibonacciSequenceIterator FibonacciSequenceIterator::operator++(int) noexcept {
   FibonacciSequenceIterator old = *this;
   ++(old);
   return old;
}

bool FibonacciSequenceIterator::operator==(const FibonacciSequenceIterator& aOther) const
noexcept {
   return fSequenceObject.id() == aOther.fSequenceObject.id() && fIndex == aOther.fIndex;
}
```

```
bool FibonacciSequenceIterator::operator!=(const FibonacciSequenceIterator& aOther) const
noexcept {
   return !(*this == aOther);
}
// return new iterator positioned at start
FibonacciSequenceIterator FibonacciSequenceIterator::begin() const noexcept {
   return FibonacciSequenceIterator(fSequenceObject);
}

// return new iterator positioned at limit
FibonacciSequenceIterator FibonacciSequenceIterator::end() const noexcept {
   // Return an iterator at the end position
   return FibonacciSequenceIterator(FibonacciSequenceGenerator(fSequenceObject.id()), 0);
}
```