# 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:** Xuan Tuan Minh Nguyen **Your student id:** 103819212

---

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:_____

```cpp
#include "FibonacciSequenceGenerator.h"
#include <stdexcept>
#include <climits>


FibonacciSequenceGenerator::FibonacciSequenceGenerator(const std::string& aID) noexcept : fID(aID), fPrevious(0), fCurrent(1) {
    //Construct the FibonacciSequenceGenerator
}

const std::string& FibonacciSequenceGenerator::id() const noexcept {
    //Get the ID of the FibonacciSequenceGenerator
    return this->fID;
}

const long long& FibonacciSequenceGenerator::operator*() const noexcept {
    //Get the current value, which will be use as the main value of FibonacciSequenceGenerator (using operator*())
    return this->fCurrent;
}

FibonacciSequenceGenerator::operator bool() const noexcept {
    //Return true if there are any next available number (not reach limit of long long). Depends on hasNext() function
    return this->hasNext();
}

void FibonacciSequenceGenerator::reset() noexcept {
    //Reset the previous and current to 0 and 1 consecutively
    this->fPrevious = 0;
    this->fCurrent = 1;
}

bool FibonacciSequenceGenerator::hasNext() const noexcept {
    //Check that current value must greater than 0 and lower than the long long limit (which is 2^64 - 1)
    return this->fCurrent >= 0 && this -> fPrevious <= (LLONG_MAX - this->fCurrent);
}

void FibonacciSequenceGenerator::next() noexcept {
    //If value is greater than limit of long long -> Raise overflow_error
    if (!this->hasNext())
        throw std::overflow_error("Fibonacci sequence overflow");
    else {
        //Set a temporary value as previous + current
        long long temporary = this->fCurrent + this->fPrevious;
        //Previous be the current value
        this->fPrevious = this->fCurrent;
```

```
45          //Current value be the temporary value
46          this->fCurrent = temporary;
47
48      }
49  }
```

```cpp
#include "FibonacciSequenceIterator.h"

FibonacciSequenceIterator::FibonacciSequenceIterator(const FibonacciSequenceGenerator& aSequenceObject, long long aStart) noexcept : fSequenceObject(aSequenceObject), fIndex(aStart) {
    //Initialize the FibonacciSequenceIterator, make sure that the sequence object must be reset before iteration
    this->fSequenceObject.reset();
}

const long long& FibonacciSequenceIterator::operator*() const noexcept {
    //Getter, using the operator*() of FibonacciSequenceGenerator
    return this->fSequenceObject.operator*();
}

FibonacciSequenceIterator& FibonacciSequenceIterator::operator++() noexcept {
    //++(FibonacciSequenceIterator)
    // Raise the fIndex to one value
    ++this->fIndex;
    //Check if could go to next value
    if (this->fSequenceObject.hasNext()) {
        //If can then jump to next value
        this->fSequenceObject.next();
    }
    //Return the iterator
    return *this;
}

FibonacciSequenceIterator FibonacciSequenceIterator::operator++(int) noexcept {
    //(FibonacciSequenceIterator)++
    //Copy iterator as temporary
    FibonacciSequenceIterator temporary = *this;
    //Raise the iterator to one value (both index and FibonacciSequenceGenerator object)
    ++temporary;
    //Return the copy
    return temporary;
}

bool FibonacciSequenceIterator::operator==(const FibonacciSequenceIterator& aOther) const noexcept {
    //Compare if the two index of both objects are true and same id as well
    return this->fIndex == aOther.fIndex && this->fSequenceObject.id() == aOther.fSequenceObject.id();
}

bool FibonacciSequenceIterator::operator!=(const FibonacciSequenceIterator&
```

```
    aOther) const noexcept {
42      //Negative of operator==
43      return !(*this == aOther);
44  }
45
46  FibonacciSequenceIterator FibonacciSequenceIterator::begin() const noexcept ↪
    {
47      //Copy the object as temporary
48      FibonacciSequenceIterator temporary = *this;
49      //Set the copy's index as 0
50      temporary.fIndex = 0;
51      //Reset the copy's FibonacciSequenceGenerator (Previous value will be 0 ↪
        and Current value will be 1)
52      temporary.fSequenceObject.reset();
53      //Return the copy
54      return temporary;
55  }
56
57  FibonacciSequenceIterator FibonacciSequenceIterator::end() const noexcept {
58      //Copy the object as temporary
59      FibonacciSequenceIterator temporary = *this;
60      //Do while loop while object is has next value
61      while (temporary.fSequenceObject.hasNext()) {
62          //Move fSequenceObject to one value
63          temporary.fSequenceObject.next();
64          //Raise the index to one value
65          ++temporary.fIndex;
66      }
67
68      //Return the copy
69      return temporary;
70  }
71
```