

**Swinburne University of Technology***School of Science, Computing and Engineering Technologies***ASSIGNMENT COVER SHEET**

---

**Subject Code:** COS30008  
**Subject Title:** Data Structures and Patterns  
**Assignment number and title:** 2, Iterators  
**Due date:** Monday, April 17, 2023, 10:30  
**Lecturer:** Dr. Markus Lumpe

---

**Your name:** \_\_\_\_\_**Your student ID:** \_\_\_\_\_

|                   |               |               |                        |                         |               |              |              |              |              |                |                |
|-------------------|---------------|---------------|------------------------|-------------------------|---------------|--------------|--------------|--------------|--------------|----------------|----------------|
| Check<br>Tutorial | Tues<br>08:30 | Tues<br>10:30 | Tues<br>12:30<br>BA603 | Tues<br>12:30<br>ATC627 | Tues<br>14:30 | Wed<br>08:30 | Wed<br>10:30 | Wed<br>12:30 | Wed<br>14:30 | Thurs<br>08:30 | Thurs<br>10:30 |
|                   |               |               |                        |                         |               |              |              |              |              |                |                |

Marker's comments:

| Problem | Marks | Obtained |
|---------|-------|----------|
| 1       | 16    |          |
| 2       | 22    |          |
| 3       | 92    |          |
| Total   | 130   |          |

**Extension certification:**

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

Signature of Convener: \_\_\_\_\_

## Problem Set 2: Iterators

| The ASCII Table |     |      |     |     |      |     |     |      |     |     |      |
|-----------------|-----|------|-----|-----|------|-----|-----|------|-----|-----|------|
| Dec             | Hex | Char | Dec | Hex | Char | Dec | Hex | Char | Dec | Hex | Char |
| 00              | 00  | NUL  | 32  | 20  | SP   | 64  | 40  | @    | 96  | 60  | `    |
| 01              | 01  | SOH  | 33  | 21  | !    | 65  | 41  | A    | 97  | 61  | a    |
| 02              | 02  | STX  | 34  | 22  | "    | 66  | 42  | B    | 98  | 62  | b    |
| 03              | 03  | ETX  | 35  | 23  | #    | 67  | 43  | C    | 99  | 63  | c    |
| 04              | 04  | EOT  | 36  | 24  | \$   | 68  | 44  | D    | 100 | 64  | d    |
| 05              | 05  | ENQ  | 37  | 25  | %    | 69  | 45  | E    | 101 | 65  | e    |
| 06              | 06  | ACK  | 38  | 26  | &    | 70  | 46  | F    | 102 | 66  | f    |
| 07              | 07  | BEL  | 39  | 27  | '    | 71  | 47  | G    | 103 | 67  | g    |
| 08              | 08  | BS   | 40  | 28  | (    | 72  | 48  | H    | 104 | 68  | h    |
| 09              | 09  | HT   | 41  | 29  | )    | 73  | 49  | I    | 105 | 69  | i    |
| 10              | 0A  | LF   | 42  | 2A  | *    | 74  | 4A  | J    | 106 | 6A  | j    |
| 11              | 0B  | VT   | 43  | 2B  | +    | 75  | 4B  | K    | 107 | 6B  | k    |
| 12              | 0C  | FF   | 44  | 2C  | ,    | 76  | 4C  | L    | 108 | 6C  | l    |
| 13              | 0D  | CR   | 45  | 2D  | -    | 77  | 4D  | M    | 109 | 6D  | m    |
| 14              | 0E  | SO   | 46  | 2E  | .    | 78  | 4E  | N    | 110 | 6E  | n    |
| 15              | 0F  | SI   | 47  | 2F  | /    | 79  | 4F  | O    | 111 | 6F  | o    |
| 16              | 10  | DLE  | 48  | 30  | 0    | 80  | 50  | P    | 112 | 70  | p    |
| 17              | 11  | DC1  | 49  | 31  | 1    | 81  | 51  | Q    | 113 | 71  | q    |
| 18              | 12  | DC2  | 50  | 32  | 2    | 82  | 52  | R    | 114 | 72  | r    |
| 19              | 13  | DC3  | 51  | 33  | 3    | 83  | 53  | S    | 115 | 73  | s    |
| 20              | 14  | DC4  | 52  | 34  | 4    | 84  | 54  | T    | 116 | 74  | t    |
| 21              | 15  | NAK  | 53  | 35  | 5    | 85  | 55  | U    | 117 | 75  | u    |
| 22              | 16  | SYN  | 54  | 36  | 6    | 86  | 56  | V    | 118 | 76  | v    |
| 23              | 17  | ETB  | 55  | 37  | 7    | 87  | 57  | W    | 119 | 77  | w    |
| 24              | 18  | CAN  | 56  | 38  | 8    | 88  | 58  | X    | 120 | 78  | x    |
| 25              | 19  | EM   | 57  | 39  | 9    | 89  | 59  | Y    | 121 | 79  | y    |
| 26              | 1A  | SUB  | 58  | 3A  | :    | 90  | 5A  | Z    | 122 | 7A  | z    |
| 27              | 1B  | ESC  | 59  | 3B  | ;    | 91  | 5B  | [    | 123 | 7B  | {    |
| 28              | 1C  | FS   | 60  | 3C  | <    | 92  | 5C  | \    | 124 | 7C  |      |
| 29              | 1D  | GS   | 61  | 3D  | =    | 93  | 5D  | ]    | 125 | 7D  | }    |
| 30              | 1E  | RS   | 62  | 3E  | >    | 94  | 5E  | ^    | 126 | 7E  | ~    |
| 31              | 1F  | US   | 63  | 3F  | ?    | 95  | 5F  | _    | 127 | 7F  | DEL  |

Figure 1: 7-Bit Character Mapping.

The goal of this problem set is to develop a small console application that counts the occurrences of each printable non-whitespace character in a given input text file. The application consists of three parts: a class `CharacterMap` that implements the logic to map characters (we use 8-Bit characters in this problem set) to the frequency of their occurrence in a given text file, a class `CharacterCounter` that defines an object adapter for an array of 256 `CharacterMap` objects to record the occurrences of characters, a class `CharacterFrequencyIterator` that allows for a systematic and ordered traversal of the counted frequencies, and a `main` function that drives the counting process.

This problem set is comprised of three problems. The test driver (i.e., `main.cpp`) uses `P1`, `P2`, and `P3` as variables to enable/disable the test associated with a corresponding problem. To enable a test just uncomment the respective `#define` line. For example, to test problem 2 only, enable `#define P2`:

```
// #define P1
#define P2
// #define P3
```

In Visual Studio, the code blocks enclosed in `#ifdef PX ... #endif` are grayed out, if the corresponding test is disabled. The preprocessor definition `#ifdef PX ... #endif` enables conditional compilation. XCode does not use this color coding scheme.

## Problem 1

To effectively define the character counting process, we need a data type `CharacterMap`, which establishes the required association between a character and its number of occurrences. The following class specification suggests a possible solution:

```
#pragma once

#include <cstdint>

class CharacterMap
{
private:
    unsigned char fCharacter;
    size_t fFrequency;

public:
    CharacterMap( unsigned char aCharacter = '\\0',
                 int aFrequency = 0 ) noexcept;

    void increment() noexcept;
    void setCharacter( unsigned char aCharacter ) noexcept;

    bool operator<( const CharacterMap& aOther ) const noexcept;

    unsigned char character() const noexcept;
    size_t frequency() const noexcept;
};
```

Class `CharacterMap` defines one constructor with default arguments. This is a C++ idiom that provides us a default constructor “for free”. Remember, a default constructor does not take any arguments and can be called anywhere. In particular, we need a default constructor when we define an array of objects, here `CharacterMap` objects. We can write

```
CharacterMap lMap;
```

to create an object `lMap` of type `CharacterMap` whose fields are initialized with `'\\0'` (the zero character) and `0` (the zero frequency), respectively.

The method `increment()` add one to the frequency and the method `setCharacter()` assigns a desired character to the map.

The getters `character()` and `frequency()` just return the corresponding values of the associated member variables.

The `operator<` compares the frequency of `this` object and the `aOther` object. We use this operator to sort character maps in increasing order in Problem 3.

To test your implementation of `CharacterMap`, uncomment `#define P1` and compile your solution.

The test driver should produce the following output:

```
Test CharacterMap
Initial: 0 - 0
After updates: S - 2
lMapA < lMapB: false
Test CharacterMap complete
```

## Problem 2

Using instances of class `CharacterMap`, we can define class `CharacterCounter` as follows:

```
#pragma once

#include "CharacterMap.h"

class CharacterCounter
{
private:
    size_t fTotalNumberOfCharacters;
    CharacterMap fCharacterCounts[256];

public:
    CharacterCounter() noexcept;

    void count( unsigned char aCharacter ) noexcept;

    const CharacterMap& operator[]( unsigned char aCharacter ) const noexcept;
};
```

The class `CharacterCounter` records the total number of counted characters and the frequencies of those characters. The class has one constructor to properly initialize the data members (i.e., set the character to values from 0 to 255), a `count` method that takes a character (an `unsigned char` value), and an indexer (i.e., `operator[]`) that returns the corresponding character map.

The implementations of the constructor and the `count` method are straightforward. The constructor initializes all data members with 0, whereas `count` has to increment the corresponding data members. The elements of `fCharacterCounts` are `CharacterMap` objects. We can use the parameter `aCharacter` as index into the array `fCharacterCounts`. In C++ characters are integer values. A `CharacterMap` object has an `increment()` method that allows us to count characters.

In general, a `CharacterCounter` object is like an array with a fixed dimension of 256.

To test your implementation of `CharacterCounter`, uncomment `#define P2` and compile your solution.

The test driver should produce the following output (use `Main.cpp` as command line argument):

```
Test CharacterCounter
The frequencies:
!: 3
": 54
#: 22
&: 5
': 2
(: 36
): 36
*: 1
+: 2
,: 5
-: 2
.: 21
/: 6
0: 7
1: 11
2: 15
```

```
3: 7
5: 1
6: 1
8: 1
:: 105
;: 43
<: 114
=: 3
>: 11
?: 1
A: 16
B: 6
C: 40
F: 2
I: 15
M: 23
O: 1
P: 18
S: 5
T: 8
U: 2
[: 6
]: 6
_: 3
a: 113
b: 1
c: 80
d: 92
e: 173
f: 47
g: 17
h: 39
i: 76
l: 73
m: 23
n: 108
o: 50
p: 41
q: 9
r: 131
s: 88
t: 175
u: 70
v: 8
w: 2
y: 7
z: 1
{: 13
}: 13
Test CharacterCounter complete.
```

### Problem 3

Finally, we wish to define an iterator for objects of class `CharacterCounter`. This iterator provides us with a systematic and ordered traversal of character maps. In particular, the iterator provides us with a "sorted view" of the character maps that adheres to a decreasing order of frequencies. To achieve this behavior, the iterator maintains a private array of indices, `fMappedIndices`, that provide the iterator with an "indirect address mode" for character maps. The indices are sorted based on the frequency recorded in the associate character map. For example, given an array of five character maps

0: ('a', 10)

1: ('c', 2)

2: ('e', 12)

3: ('g', 4)

4: ('k', 3)

the array `fMappedIndices` equals `[2, 0, 3, 4, 1]` which orders the character maps in decreasing order of frequency: ('e', 12), ('a', 10), ('g', 4), ('k', 3), ('c', 2).

To sort character maps, we use *stable Insertion Sort* (compare lecture week 1). In this context, *stable* means that if two `CharacterMap` objects are equal they do not change places.

The specification of character map iterator is given below:

```
#pragma once

#include "CharacterCounter.h"

class CharacterFrequencyIterator
{
private:
    const CharacterCounter* fCollection;
    size_t fIndex;
    unsigned char fMappedIndices[256];

    void mapIndices() noexcept;

public:
    CharacterFrequencyIterator( const CharacterCounter* aCollection ) noexcept;

    const CharacterMap& operator*() const noexcept;

    CharacterFrequencyIterator& operator++() noexcept;
    CharacterFrequencyIterator operator++( int ) noexcept;

    bool operator==( const CharacterFrequencyIterator& aOther ) const noexcept;
    bool operator!=( const CharacterFrequencyIterator& aOther ) const noexcept;

    CharacterFrequencyIterator begin() const noexcept;
    CharacterFrequencyIterator end() const noexcept;
};
```

The iterator uses three member variables: `fCollection` which is a raw pointer to an object of type `CharacterCounter`, `fIndex` which is an unsigned integer that represents to current iterator position, and `fMappedIndices` that defines an indirect address mode for element in `fCollection`. We use a raw pointer for `fCollection` to avoid value copies and to facilitate the comparison of two iterators. Using raw pointers is not considered a good practice

in general as raw pointers are non-specific about ownership of objects. We will revisit this issue later in the unit and learn how to use smart pointers to explicitly manage ownership of objects. In this problem set, however, we are not concerned about ownership and the use of a raw pointer does not cause any issues.

The implementation of `SimpleForwardIterator` is standard in the sense that we use and update the member variable `fIndex` to access the current element and advance the iterator position, respectively. The index must be set to 256 (the end) once the iterator reaches a character map whose frequency is 0.

The exception is the array `fMappingIndices`. This array has to be set up as part of the iterator initialization. We use the method `mapIndices()` for this purpose, which is called once in the constructor. This method performs two tasks. First, it initializes the array in a `for`-loop from 0 to 255. Second, it uses *Insertion Sort* to sort the array based on character frequencies. The elements in `fMappingIndices` can be used to access character maps from `fCollection`. For example, the expression

```
(*fCollection)[fMappedIndices[3]]
```

yields a `CharacterMap` object whose character frequency is the 4<sup>th</sup> largest in `fCollection`. Here, the expression `fMappedIndices[3]` is an indirect index into `fCollection`. This technique is similar to the approach we have used in tutorial 5, with the exception that we used only one index (i.e., `fMapPosition`). In this problem set there are 256 indices and they need to be order so that the iterator returns character maps in decreasing order of frequency.

To test your implementation of `CharacterCounter`, uncomment `#define P3` and compile your solution.

The test driver should produce the following output (use `Main.cpp` as command line argument):

```
Test CharacterCounterIterator
The frequencies:
t: 175
e: 173
r: 131
<: 114
a: 113
n: 108
:: 105
d: 92
s: 88
c: 80
i: 76
l: 73
u: 70
": 54
o: 50
f: 47
;: 43
p: 41
C: 40
h: 39
(: 36
): 36
M: 23
m: 23
#: 22
.: 21
P: 18
g: 17
```

```
A: 16
2: 15
I: 15
{: 13
}: 13
1: 11
>: 11
q: 9
T: 8
v: 8
0: 7
3: 7
y: 7
/: 6
B: 6
[: 6
]: 6
&: 5
,: 5
S: 5
!: 3
=: 3
_: 3
T: 2
+: 2
-: 2
F: 2
U: 2
w: 2
*: 1
5: 1
6: 1
8: 1
?: 1
O: 1
b: 1
z: 1
Test CharacterCounterIterator complete.
```

The solution of this problem set requires approx. 150-160 lines of low density C++ code.

**Submission deadline: Monday, April 17, 2023, 10:30.**

**Submission procedure:** Follow the instruction on Canvas. Submit electronically PDF of printed code for classes `CharacterMap.cpp`, `CharacterCounter.cpp`, and `CharacterFrequencyIterator.cpp`. Upload the sources of `CharacterMap.cpp`, `CharacterCounter.cpp`, and `CharacterFrequencyIterator.cpp` to Canvas.

The sources need to compile in the presents of the solution artifacts provided on Canvas.