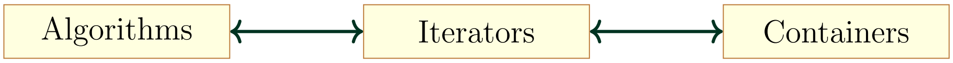
Contents

1 Purpose

- Practice using the STL components: standard sequence container[1] classes[2], standard associative container[3] classes[4], iterators, and callable objects

- Get the feel for how to connect algorithms to containers with the help of iterators.



2 General Requirements

- The implementation of the tasks in this assignment may not use explicit loops; that is, no `for`, `while` or `do/while` loops. However, you may use loops in your test drivers.

- Define the following type name abbreviations fro tasks 1, 2, 3 and 4.

```
using WordsVector = std::vector<std::string>;
using WordsMap = std::map<std::string, size_t>;
```

[1] A sequence container provides access based on the **position** of an element in the sequence.

[2] such as `std::array<>`, `std::vector<>`, `std::list<>` and `std::forward_list<>`

[3] An associative container provides access to the elements based on a **key**.

[4] The Standard Library offers two categories of associative containers:

- **Ordered associative containers** are usually implemented as Self-balancing binary search trees.
  `std::set<>`, `std::multiset<>`, `std::map<>`, and `std::multimap<>`
- **Unordered associative containers** are implemented as hash tables.
  `std::unordered_set<>`, `std::unordered_multiset<>`, `std::unordered_map<>`, and `std::unordered_multimap<>`

3 Task 1: Copy the Words in a Text File Into a Vector

Consider the following incomplete function that takes a text file consisting of exactly one English word per line as parameter and returns a `WordsVector` containing all the words in that file.

Fill in the blanks to complete the call to the `std::copy` algorithm. If you find your expression for a blank too long, you may define a variable representing that expression and use that variable in the corresponding blank.

```
WordsVector read_words_into_vector(const std::string& inFileName)
{
   std::ifstream ifs(inFileName); // Create an input file stream
   if (!ifs.is_open()) {          // Check that the file is open
      cout << "Could not open file " + inFileName << endl;
      throw std::runtime_error("Could not open file " + inFileName);
   }
```

```
   WordsVector words_vector;  // an empty vector
   std::copy(__________,       // start of input stream
             __________,       // end of input stream
             __________);      // destination
   return words_vector;
}
```

4 Task 2: Count Frequency of the Words in a Vector Using a Lambda

Complete the following function that takes a single parameter, namely wvec, which is a vector storing words. The function is to count the number of occurrences of each individual word occurring in wvec.

To keep track of the words and their occurrences in wvec, the function uses a WordsMap object named wmap in which the *keys* are the words and the *values* are the frequency of the words.

```
WordsMap map_and_count_words_using_lambda(const WordsVector& wvec)
{
   WordsMap wmap;
   std::for_each(         ,      // start of source
                          ,      // end of source
                          );     // lambda expression
   return wmap;
}
```

The call to the std::for_each algorithm passes wvec's elements (words) one by one to for_each's third argument, which is a unary callable expression (a lambda in this task) taking exactly one parameter. To keep track of the occurrences of each word, however, the lambda needs write access to wmap. As a result, you will need to find a way to make wmap accessible and writable inside the lambda.

5 Task 3: Count Frequency of the Words in a Vector Using a Functor

Complete the following function that is to count the number of occurrences of each individual word occurring in a given vector, namely wvec, the function's single parameter.

This function behaves the same as that of Task 2, except that it uses an object, namely wcf, of a function-object class named WordCountFunctor as the third argument in the call to the std::for_each algorithm.

```
WordsMap map_and_count_words_using_functor(const WordsVector& wvec)
{
   WordCountFunctor wcf{};
   wcf = std::for_each(
                 _________,   // start of source
                 _________,   // end of source
                 _________); // a functor keeping trac of the frequencies

   return wcf.getmap();
}
```

Again, whether we use a lambda, a functor, or a free function as the third argument in a call to the std::for_each algorithm, that argument must be a function taking exactly one parameter through which wvec's elements (words) are passed.

Therefore, to keep track of the occurrences of the words, the functor wcf must be equipped with its own WordsMap object where it can store the words and their corresponding frequency.

6 Task 4: Remove duplicated Words in a Vector

Implement the following function to remove the duplicated words in the supplied `words_vector`. A sketch of this function is listed as comments:

```cpp
WordsVector remove_duplicates(const WordsVector& words_vector)
{
   WordsVector words_vec{ words_vector }; // make a copy of the supplied words_vector
   // 1- use std::sort to sort words_vec alphabetically
   //    so that we can locate the duplicate words in it.

   // 2- use std::unique to rearrange the words in the sorted words_vec
   //    so that each word appears once in the front portion of words_vec.
   //    store the returned iterator, which points to the element
   //    immediately after all the unique elements in the front of words_vec.

   // 3- use std::vector's erase member function to erase the range of non-unique
   //    words in words_vec, starting at the iterator stored in step 2 above
   //    to the end of words_vec.
   return words_vec;
}
```

7 Task 5: Palindromes and No Explicit Loops

Recall that a palindrome is a word or phrase that reads the same backward and forward, such as "`Was it a car or a cat I saw?`". The reading process ignores spaces, punctuation, and capitalization.

Write a function named `isPalindrome` that takes a parameter of the type `std::string` representing a *phrase* and determines whether that string is a palindrome.

Your implementation may *not* use

- more than one local `string` variable
- raw arrays, STL container classes

7.1 A suggested sketch of the function

1. use `std::remove_copy_if` to move only the alphabet characters from `phrase` to `temp`;
    - Since `temp` is initially empty, you will need to use an `inserter` iterator when you fill it with the alphabet characters of `phrase`.
    - As the last argument in the call to `std::remove_copy_if`, pass a unary predicate, a regular free function in this task, named, say, `is_alphabetic`, that takes a `char ch` as its single parameter and determines whether `ch` is an alphabetic character.
2. To allow case insensitive comparison, convert all the characters in `temp` to the same letter-case, either uppercase or lowercase.
    - To do this use the `std::transform` algorithm, passing `temp` as both the source and the destination streams, effectively overwriting `temp` during the transformation process.
        - As the last argument in the call to `std::transform`, use a lambda that takes a `char ch` as its only parameter and returns `ch` in the selected letter-case.
3. use `std::equal` to compare the first half of `temp` with its second half, moving forward in the first half starting at `temp.begin()` and moving backward in the second half starting at `temp.rbegin()`.
    - Store in `result` the `bool` value returned from the call to `std::equal`;
4. return `result`

8 Task 6: Counting Strings of Equal lengths

Some algorithms, such as the `count_if` algorithm shown below, take a parameter that is a either unary or binary **predicate**, a callable expression that returns a `bool` value.

```
// Returns the number of elements in the range [first,last) for which pred is true.
template <class InputIterator, class UnaryPredicate>        // template header
  typename iterator_traits<InputIterator>::difference_type  // return type
    count_if (InputIterator first,                          // start of source range
              InputIterator last,                           // end of source range
              UnaryPredicate pred);                         // a unary predicate
```

A unary predicate has exactly one parameter, whereas a binary predicate has exactly two parameters. However, depending on what we want it to do, a predicate may requires more arguments than it allows. This task involves such situation[5].

Write three functions that have the same return type and parameter lists of the form

```
int count_using_xxx (const std::vector<std::string>& vec, int n);
```

where `xxx` is either `lambda`, `free_func`, or `functor` (function object). Using the `count_if` algorithm, each function must count and return the number of elements in `vec` that are of length `n`. For example, if `vec` is defined like this

```
std::vector<std::string> vec { "C", "BB", "A", "CC", "A", "B",
                               "BB", "A", "D", "CC", "DDD", "AAA", "CCC" };
```

then the calls taking the arguments `(vec, 1)`, `(vec, 2)`, `(vec, 3)`, and `(vec, 4)` must return 6, 4, 3, and 0, respectively.

Taking exactly one string parameter of the type `std::string`, your unary predicate in each version must determine whether the length of that string parameter is `n`. Specifically, the predicate in

version 1

```
int count_using_lambda (const std::vector<std::string>& vec, int n);
```

must use a lambda expression that captures `n` by value in its introducer.

version 2

```
int count_using_Functor(const std::vector<std::string>& vec, int n);
```

must use a functor (function object) named that stores `n` at construction.

version 3

```
int count_using_Free_Func(const std::vector<std::string>& vec, int n);
```

must use a free binary function `bool freeFunc(std::string, int)` that is turned into a "unary" function by fixing its 2nd argument to `n` using std::bind.[6]

[5] Similar situation exists in tasks 2 and 3, where the third argument to the `for_each` algorithm is a callable expression that takes exactly one parameter.

[6] Specifically, `auto unaryFreeFunc = std::bind(freeFunc, _1, n);` where `_1` refers to the first and only argument of `unaryFreeFunc`. As a result, a call such as `unaryFreeFunc("hello")` is equivalent to the call `freeFunc("hello", n)`.

9 Task 7: Sorting Strings on length and Value

Consider the following function that prints the sorted version of a supplied vector. It uses a multiset object that is constructed using std::multiset's default compare type parameter, which by default is std::less<T>.

```
1  void multisetUsingDefaultComparator(const std::vector<std::string>& vec)
2  {
3     std::multiset<std::string> strSet; // an empty set
4
5     // to print a sorted verstion of the supplied vector vec,
6     // we first copy vec to our strSet and then print the strSet.
7
8     // note: since std::multiset does not provide push_front or push_back members,
9     // we can't use a front or back inserter when we copy vec to our empty strSet,
10    // meaning that we must use a general inserter:
11
12    std::copy(vec.begin(), vec.end(),                // source start and finish
13              std::inserter(strSet, strSet.begin())); // destination start using
14                                                      // a general inserter
15
16    // create an ostream_iterator attached to cout, using a space " " as a separator
17    std::ostream_iterator<std::string> out(cout, " ");
18
19    // output the set elements to the cout
20    std::copy(strSet.begin(), strSet.end(), out);
21 }
```

For example, the code

```
std::vector<std::string> vec =
{ "C", "BB", "A", "CC", "A", "B", "BB", "A", "D", "CC", "DDD", "AAA" };

multisetUsingDefaultComparator(vec);
```

will produce the following output:

```
A A A AAA B BB BB C CC CC D DDD
```

Renaming the function multisetUsingMyComparator(), modify the declaration on line 3 so that the same code will produce the following output:

```
A A A B C D BB BB CC CC AAA DDD
```

The effect is that the string elements in strSet are now ordered into groups of strings of increasing lengths 1, 2, 3, ..., with the strings in each group sorted lexicographically.

10 Task 8: Generate the First N Fibonacci numbers

The Fibonacci sequence is defined using the following formula:

$$F_n = \begin{cases} 0, & \text{if } n = 0; \\ 1, & \text{if } n = 1; \\ F_{n-1} + F_{n-2} & \text{if } n > 1 \end{cases}$$

Write a function that has the following prototype and that uses the `std::generate_n` algorithm to generate the first `n` terms of the Fibonacci sequence into a `std::vector<int>` and returns that vector.

```
std::vector<int> getnerate_Fibonacci(int n);
```

Hint: See examples 1, 2, and 7 in Lecture Notes 11.

11 Deliverables

Implementation files: assignment5.h, assignment5.cpp. In addition, include the supplied test driver file assignment_5_test_driver.cpp.

README.txt A text file, as described in the course outline.

12 Grading scheme

Functionality	• Correctness of execution of your program • Proper implementation of all specified requirements • Efficiency	60
OOP Style	• Encapsulating only the necessary data inside objects • Information hiding • Proper use of C++ constructs and facilities • No global variables • No use of the operator delete • No C-style memory functions such as malloc, alloc, free, etc.	20
Documentation	• Description of purpose of program • Javadoc comment style for all methods and fields • Comments for non-trivial code segments	10
Presentation	• Format, clarity, completeness of outpu • User friendly interface	5
Code Readability	• Meaningful identifiers, indentation, spacing	5

13 Test Driver

13.1 assignment.h

```cpp
#ifndef ASSIGNMENT5_H_
#define ASSIGNMENT5_H_

#include <map>
#include <string>
#include <vector>
// Type aliases
using WordsVector = std::vector<std::string>;
using WordsMap = std::map<std::string, size_t>;

WordsVector read_words_into_vector(const std::string& inFileName);
WordsMap map_and_count_words_using_lambda(const WordsVector& wvec);
WordsMap map_and_count_words_using_functor(const WordsVector& wvec);
WordsVector remove_duplicates(const WordsVector& words_vector);
bool is_palindrome(const std::string& phrase);
size_t count_using_lambda(const std::vector<std::string>& vec, int n);
size_t count_using_Free_Func(const std::vector<std::string>& vec, int n);
size_t count_using_Functor(const std::vector<std::string>& vec, int n);
void multisetUsingMyComparator(const std::vector<std::string>& vec);
void multisetUsingDefaultComparator(const std::vector<std::string>& vec);
std::vector<int> getnerate_Fibonacci(int n);
#endif
```

13.2 assignment_5_test_driver

```cpp
#include <cassert>
#include <iostream>
using std::cout;
using std::endl;
using std::cin;

#include "assignment5.h"
// function prototypes
void validate_words_vector(const WordsVector& word_vector);
void print_words_vector(const WordsVector& word_vector);
WordsVector task_1_Test_Drive(const std::string& infilename);
void validate_word_map(const WordsMap& wmap);
void print_word_map(const WordsMap& wmap);
void task_2_Test_Drive(const WordsVector& words_vector);
void task_3_Test_Drive(const WordsVector& words_vector);
void validate_unique_words_vector(const WordsVector& word_vector);
void task_4_Test_Drive(const WordsVector& words_vector);
void task_5_Test_Drive();
void task_6_test_driver();
void task_7_test_driver();
void task_8_test_driver(int n);
```

```

// Task 1
void validate_words_vector(const WordsVector& word_vector)
{
   assert(word_vector.size() == 574);
   assert(word_vector.back() == "yoke");
   assert(word_vector[0] == "wink");
   assert(word_vector[200] == "fool");
   assert(word_vector[400] == "work");
   assert(word_vector.at(100) == "gainful");
   assert(word_vector.at(300) == "dirty");
   assert(word_vector.at(500) == "coast");
}
void print_words_vector(const WordsVector& word_vector)
{
   for (const auto& word : word_vector)
   {
      cout << word << endl;
   }
   cout << "Number of words: " << word_vector.size() << endl;

}

WordsVector task_1_Test_Drive(const std::string& infilename)
{
   WordsVector words_vector = read_words_into_vector(infilename);
   validate_words_vector(words_vector);
   cout << "All words extracted OK\n";
   //cout << "All words in the input file\n";
   //print_words_vector(words_vector);
   return words_vector;
}
```

```

// Task 2
void validate_word_map(const WordsMap& wmap)
{
   const auto& [word1, count1] { *wmap.begin() };
   cout << word1 << ": " << count1 << endl;
   assert(word1 == "air" && count1 == 6);

   const auto& [word2, count2] { *std::prev(wmap.end()) };
   cout << word2 << ": " << count2 << endl;
   assert(word2 == "yoke" && count2 == 8);

   // PreC++17 way of accessing a map's element, say the first element
   std::pair<std::string, size_t> key_value_pair{ *wmap.begin() };
   std::string key = key_value_pair.first;
   size_t value = key_value_pair.second;
   assert(key == "air" && value == 6);
}

void print_word_map(const WordsMap& wmap)
{

   for (const auto& [word, count] : wmap)
   {
      cout << word << ": " << count << endl;
   }
}

void task_2_Test_Drive(const WordsVector& words_vector)
{
   WordsMap word_map_using_lambda = map_and_count_words_using_lambda(words_vector);
   validate_word_map(word_map_using_lambda);
   cout << "word_map_using_lambda is OK\n";
   cout << "All words in the map generated using lambda\n";
   print_word_map(word_map_using_lambda);
}
```

```cpp

// Task 3
void task_3_Test_Drive(const WordsVector& words_vector)
{
   WordsMap word_map_using_functor = map_and_count_words_using_functor(words_vector);
   validate_word_map(word_map_using_functor);
   cout << "word_map_using_functor is OK\n";
   //cout << "All words in the map generated using functor\n";
   //print_word_map(word_map_using_functor);
}
```

```cpp

// Task 4
void validate_unique_words_vector(const WordsVector& word_vector)
{
   assert(word_vector.size() == 100);
   assert(word_vector.back() == "yoke");
   assert(word_vector[0] == "air");
   cout << "Unique words OK\n";
}

void task_4_Test_Drive(const WordsVector& words_vector)
{
   WordsVector unique_words_vector = remove_duplicates(words_vector);
   cout << "All unique words\n";
   print_words_vector(unique_words_vector);
   validate_unique_words_vector(unique_words_vector);
}
```

```cpp

// Task 5
void task_5_Test_Drive()
{
   std::string str_i_saw = std::string("was it a car or A Cat I saW?");
   bool result_i_saw = is_palindrome(str_i_saw);
   assert(result_i_saw == true);
   cout << "the phrase \"" + str_i_saw + "\" is a palindrome\n";

   std::string str_u_saw = std::string("was it A Car or a cat U saW?");
   bool result_u_saw = is_palindrome(str_u_saw);
   assert(result_u_saw == false);
   cout << "the phrase \"" + str_u_saw + "\" is not a palindrome\n";
}
```

```

// Task 6
void task_6_test_driver()
{
   std::vector<std::string> vecstr
   { "count_if", "Returns", "the", "number", "of", "elements", "in", "the",
      "range", "[first", "last)", "for", "which", "pred", "is", "true."
   };
   assert(count_using_lambda(vecstr, 5) == 4);
   assert(count_using_Free_Func(vecstr, 5) == 4);
   assert(count_using_Free_Func(vecstr, 5) == 4);

   assert(count_using_lambda(vecstr, 3) == 3);
   assert(count_using_Free_Func(vecstr, 3) == 3);
   assert(count_using_Free_Func(vecstr, 3) == 3);

   cout << "Task 6 OK" << endl;
}
```

```

// Task 7
void task_7_test_driver()
{
   std::vector<std::string> vec =
   { "C", "BB", "A", "CC", "A", "B", "BB", "A", "D", "CC", "DDD", "AAA" };
   multisetUsingDefaultComparator(vec);
   cout << ’\n’;
   multisetUsingMyComparator(vec);
   cout << endl;
}
```

```

// Task 8
void task_8_test_driver(int n)
{
   cout << "Fibonacci Sequence" << endl;
   std::vector<int> fibs = getnerate_Fibonacci(n);
   std::copy(fibs.begin(), fibs.end(), std::ostream_iterator<int>(cout, " "));
   assert(fibs[9] == 34);
   assert(fibs[14] == 377);
}
```

```cpp

int main()
{
   std::string infilename{ R"(C:\Users\msi\CPP\words.txt)" }; // adjust the file location

   WordsVector words_vector = task_1_Test_Drive(infilename);
   task_2_Test_Drive(words_vector);
   task_3_Test_Drive(words_vector);
   task_4_Test_Drive(words_vector);

   task_5_Test_Drive();
   task_6_test_driver();
   task_7_test_driver();
   task_8_test_driver(15);

   return 0;
}
```

13.3 Output

```
All words extracted OK
air: 6
yoke: 8
word_map_using_lambda is OK
All words in the map generated using lambda
air: 6
airplane: 1
amusement: 1
back: 10
beautiful: 8
bells: 7
berry: 3
blot: 1
blue-eyed: 4
bore: 5
bubble: 2
childlike: 2
chop: 2
clap: 5
coast: 1
combative: 5
compete: 9
cooperative: 9
curtain: 4
cushion: 6
defective: 10
defiant: 10
dirty: 8
dynamic: 8
```

```
easy: 8
egg: 4
expensive: 1
extend: 7
extra-small: 1
fast: 11
fearful: 1
feeling: 2
female: 9
flight: 11
flock: 1
fool: 9
friends: 5
gainful: 9
grandiose: 4
greedy: 10
green: 2
grin: 1
groan: 2
guarantee: 9
guitar: 10
gusty: 8
half: 3
hapless: 8
harmonious: 1
hose: 8
impartial: 1
intend: 8
lame: 8
leg: 2
library: 11
limit: 6
melted: 6
mice: 8
milk: 2
moan: 1
noiseless: 7
offbeat: 8
overconfident: 1
overwrought: 1
owe: 8
painful: 9
paper: 5
perform: 10
pickle: 4
power: 8
pushy: 3
quince: 10
rambunctious: 7
reign: 3
representative: 4
roasted: 5
```

```
rot: 7
sassy: 8
sick: 5
snail: 10
somber: 9
spooky: 10
story: 7
stretch: 3
summer: 1
superb: 10
support: 2
swanky: 8
symptomatic: 3
tearful: 6
ticket: 4
unkempt: 4
useless: 5
waiting: 7
wanting: 10
wink: 8
woebegone: 6
work: 10
yam: 5
yoke: 8
air: 6
yoke: 8
word_map_using_functor is OK
All unique words
air
airplane
amusement
back
beautiful
bells
berry
blot
blue-eyed
bore
bubble
childlike
chop
clap
coast
combative
compete
cooperative
curtain
cushion
defective
defiant
dirty
dynamic
```

```
easy
egg
expensive
extend
extra-small
fast
fearful
feeling
female
flight
flock
fool
friends
gainful
grandiose
greedy
green
grin
groan
guarantee
guitar
gusty
half
hapless
harmonious
hose
impartial
intend
lame
leg
library
limit
melted
mice
milk
moan
noiseless
offbeat
overconfident
overwrought
owe
painful
paper
perform
pickle
power
pushy
quince
rambunctious
reign
representative
roasted
```

```
rot
sassy
sick
snail
somber
spooky
story
stretch
summer
superb
support
swanky
symptomatic
tearful
ticket
unkempt
useless
waiting
wanting
wink
woebegone
work
yam
yoke
Number of words: 100
Unique words OK
the phrase "was it a car or A Cat I saW?" is a palindrome
the phrase "was it A Car or a cat U saW?" is not a palindrome
Task 6 OK
A A A AAA B BB BB C CC CC D DDD
A A A B C D BB BB CC CC AAA DDD
Fibonacci Sequence
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```