**Workshop 7: STL Algorithms**

In this workshop, you use the Algorithm category of the Standard Template Library.

You are going to create an application that manages a collection of songs.

**Learning Outcomes**

Upon successful completion of this workshop, you will have demonstrated the abilities to:

* copy data from a file into a sequential container
* use the numeric library to accumulate data values
* use a lambda expression to specify an operation on each value in a data set
* use the algorithm library to sort data values

**Submission Policy**

The workshop is divided into two coding parts and one non-coding part:

* *Part 1*: worth 0% of the workshop's total mark, is optional and designed to assist you in completing the second part.
* *Part 2*: worth 100% of the workshop's total mark, is due on **Sunday at 23:59:59** of the week of your scheduled lab. Submissions of *Part 2* that do not contain the *reflection* are not considered valid submissions and are ignored.
* *reflection*: non-coding part, to be submitted together with *Part 2*. The reflection does not have marks associated to it, but can incur a **penalty of max 40% of the whole workshop's mark** if your professor deems it insufficient (you make your marks from the code, but you can lose some on the reflection).

The workshop should contain ***only work done by you this term*** or provided by your professor. Work done in another term (by you or somebody else), or work done by somebody else and not **clearly identified/cited** is considered plagiarism, in violation of the Academic Integrity Policy.

Every file that you submit must contain (as a comment) at the top **your name**, **your Seneca email**, **Seneca Student ID** and the **date** when you completed the work.

* If the file contains only your work, or work provided to you by your professor, add the following message as a comment at the top of the file:

I have done all the coding by myself and only copied the code that my professor provided to complete my workshops and assignments.

* If the file contains work that is not yours (you found it online or somebody provided it to you), **write exactly which parts of the assignment are given to you as help, who gave it to you, or which source you received it from.** By doing this you will only lose the mark for the parts you got help for, and the person helping you will be clear of any wrong doing.

**Compiling and Testing Your Program**

All your code should be compiled using this command on matrix:

/usr/local/gcc/10.2.0/bin/g++ -Wall -std=c++17 -g -o ws file1.cpp file2.cpp ...

* -Wall: compiler will report all warnings
* -std=c++17: the code will be compiled using the C++17 standard
* -g: the executable file will contain debugging symbols, allowing *valgrind* to create better reports
* -o ws: the compiled application will be named ws

After compiling and testing your code, run your program as following to check for possible memory leaks (assuming your executable name is ws):

valgrind ws

To check the output, use a program that can compare text files. Search online for such a program for your platform, or use *diff* available on matrix.

**Part 1 (0%)**

The first portion of this workshop consists of modules:

* w7 (supplied)
* SongCollection

Enclose all your source code within the sdds namespace and include the necessary guards in each header file.

In the specs below, functions marked with DO NOT USE MANUAL LOOPS should not use for or while in the implementation; these functions should use STL Algorithms. The STL Algorithms you choose should use lambda expressions to process the collection; the lambdas should **capture the necessary variables from context only by value** (the stream object is the only one allowed to be captured by reference). Check course notes to identify which algorithm is appropriate in each situation. Using manual loops will lead to **rejection** of the workshop or **severe penalties** (at the discretion of your professor).

**SongCollection Module**

This modules defines a simple structure called Song, capable of storing the following information about a single song:

* artist
* title
* album
* price
* release year
* length of the song

No need to add any member functions to this structure. Choose appropriate types for each attribute.

Also, define a class called SongCollection that manages a collection of objects of type Song.

***Public Members for SongCollection***

* a custom constructor that receives as a parameter the name of the file containing the information about the songs to be added to the collection. This function should load into the attributes all the songs in the file.

If the filename is incorrect, this constructor should throw an exception.

Each line from the file contains information about a single song in the following format:

TITLE ARTIST ALBUM YEAR LENGTH PRICE

The fields are not separated by delimiters; each field has a fixed size: TITLE, ARTIST and ALBUM have **exactly** 25 characters; while YEAR, LENGTH and PRICE have **exactly** 5 characters.

The length of a song in the input file is stored in seconds.

Any blank space at the begining/end of a token is not part of the token and should be removed.

* void SongCollection::display(std::ostream& out) const: print the content of the collection into the parameter (one song / line). Use the insertion operator (see below). ***DO NOT USE MANUAL LOOPS!***

***Free Helpers***

* std::ostream& operator<<(std::ostream& out, const Song& theSong): inserts one song into the first parameter, using the following format (the width of each field is specified in brackets):
* | TITLE(20) | ARTIST(15) | ALBUM(20) | YEAR(6) | LENGTH | PRICE |

Look in the sample output to see how the numbers should be formatted and the alignment of each field.

**Add any other member that is required by your design!**

**w7 Module (supplied)**

The tester module for the first portion has been supplied. **Do not modify the existing code!**

When doing the workshop, you are encouraged to write your own tests, focusing on a single implemented feature at the time, until you get all functionality in place.

**Sample Output**

When the program is started with the command (the file songs.txt is provided):

ws songs.txt

the output should look like the one from the sample\_output.txt file.

**Test Your Code**

To test the execution of your program, use the same data as shown in the output example above.

Upload your source code to your matrix account. Compile and run your code using the latest version of the g++ compiler (available at /usr/local/gcc/10.2.0/bin/g++) and make sure that everything works properly.

Then, run the following command from your account (replace profname.proflastname with your professor’s Seneca userid):

~profname.proflastname/submit 345\_w7\_p1

and follow the instructions.

***This part represents a milestone in completing the workshop and is not marked!***

**Part 2 (100%)**

The second part of this workshop upgrades your solution to include more functionality in the SongCollection module.

**⚠️Important:** Your solution should not use the same algorithm twice.

**SongCollection Module**

***Public Members***

* void SongCollection::display(std::ostream& out) const: update this function to display the playtime of the entire collection (the total playtime is the sum of the length attribute of all songs). See the sample output for the format. ***DO NOT USE MANUAL LOOPS!***
* void sort(): receives as a parameter the name of the field used to sort the collection of songs in ascending order. The parameter can have one of the values title, album, or length. ***DO NOT USE MANUAL LOOPS!***
* void cleanAlbum(): removes the token [None] from the album field of the songs that do not have a valid album. ***DO NOT USE MANUAL LOOPS!***
* bool inCollection() const: receives the name of an artist as a parameter, and returns true if the collection contains any song by that artist. ***DO NOT USE MANUAL LOOPS!***
* std::list<Song> getSongsForArtist() const: receives the name of an artist as a parameter, and returns the list of songs of that artist available in the collection. ***DO NOT USE MANUAL LOOPS!***

**w7 Module**

The tester module has been supplied. **Do not modify the existing code!**

When doing the workshop, you are encouraged to write your own tests, focusing on a single implemented feature at the time, until you get all functionality in place.

**Sample Output**

When the program is started with the command (the input file is provided):

ws songs.txt

the output should look like the one from the sample\_output.txt file.

**Reflection**

Study your final solution, reread the related parts of the course notes, and make sure that you have understood the concepts covered by this workshop. **This should take no less than 30 minutes of your time and the result is suggested to be at least 150 words in length.**

Create a **text** file named reflect.txt that contains your detailed description of the topics that you have learned in completing this particular workshop and mention any issues that caused you difficulty and how you solved them. Include in your explanation—**but do not limit it to**—the following points:

* list all the STL functions that you have used in this workshop; explain your reasoning for choosing each one, and not another.
* present the advantages of using STL algorithms vs. implementing the functionality by yourself using loops.
* the function that sorts the collection of songs receives as a parameter the field name to use; this can create errors if the client specifies an invalid field (or misspells it). How would you redesign that function to prevent the client to specify an invalid string, allowing any error to be detected by the compiler?

To avoid deductions, refer to code in your solution as examples to support your explanations.

**Submission**

To test and demonstrate execution of your program use the same data as shown in the output example above.

Upload the source code and the reflection file to your matrix account. Compile and run your code using the latest version of the g++ compiler (available at /usr/local/gcc/10.2.0/bin/g++) and make sure that everything works properly.

Then, run the following command from your account (replace profname.proflastname with your professor’s Seneca userid):

~profname.proflastname/submit 345\_w7\_p2

and follow the instructions.