**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 *in-lab* section is to be completed during your assigned lab section. It is to be completed and submitted by the end of the workshop period. If you attend the lab period and cannot complete the *in-lab* portion of the workshop during that period, ask your instructor for permission to complete the *in-lab* portion after the period. If you do not attend the workshop, you can submit the *in-lab* section along with your *at-home* section (see penalties below). **In order to get credit for the *in-lab* portion, you must be present in the lab for the entire duration of the lab.**

The *at-home* portion of the workshop is due on the day that is four days after your scheduled *in-lab* workshop (@ 23:59:59), **even if that day is a holiday**.

All your work (all the files you create or modify) must contain your name, Seneca email, student number and the date of completion (use the following template):

// Name:

// Seneca Student ID:

// Seneca email:

// Date of completion:

//

// I confirm that the content of this file is created by me,

// with the exception of the parts provided to me by my professor.

You are responsible to back up your work regularly.

**Late Submission Penalties**

The workshop can be submitted up to **1 (one) day** late (the day that is 5 days after the lab period); submissions received on this day are considered **late** and are subject to penalties:

* only *in-lab* portion submitted late (after the end of the lab period): 0 for *in-lab* portion, max 7/10 for the entire workshop.
* only *at-home* portion submitted late (more than 4 days after the lab period): max 4 for the *at-home* portion, max 7/10 for the entire workshop.
* both *in-lab* **and** *at-home* portions submitted late: max 4/10 for the entire workshop.
* when the submission closes, if the workshop is not complete, the mark for the entire workshop will be 0/10. The workshop is considered complete if there are two separate submissions (*in-lab* submission and *at-home* submission) containing the *in-lab code*, *at-home code* and *reflection*.

The submission is considered closed at the end of the day that is 5 (five) days after the lab period.

***In-Lab***

The in-lab 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. 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 in-lab 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):

w7.exe songs.txt

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

Command Line:

--------------------------

1: w7.exe

2: songs.txt

--------------------------

----------------------------------------------------------------------------------------

| The original collection |

----------------------------------------------------------------------------------------

| Bird Set Free | Sia | This Is Acting | 2016 | 4:12 | 1.21 |

| Fight Song | Rachel Platten | [None] | 2015 | 3:24 | 1.25 |

| ..Baby One More Time | Britney Spears | [None] | 1999 | 3:30 | 1.29 |

| One Million Bullets | Sia | This Is Acting | 2016 | 4:12 | 1.23 |

| (You Drive Me) Crazy | Britney Spears | Baby One More Time | 1999 | 3:18 | 1.29 |

| Cheap Thrills | Sia | This Is Acting | 2016 | 3:31 | 1.29 |

| Dream On | Amy Macdonald | Under Stars | 2017 | 3:19 | 1.29 |

| Dream Is Collapsing | Hans Zimmer | Best of | 2014 | 2:23 | 1.29 |

| Hot N Cold | Katy Perry | [None] | 2008 | 3:40 | 1.14 |

| Circus | Britney Spears | Circus | 2008 | 3:12 | 1.29 |

| Under Stars | Amy Macdonald | Under Stars | 2017 | 3:41 | 1.29 |

| Rafiki's Fireflies | Hans Zimmer | Best of | 2017 | 1:52 | 1.29 |

| Song Of Ocarina | Diego Modena | [None] | 1991 | 3:35 | 0.49 |

| Sometimes | Britney Spears | Baby One More Time | 1999 | 4:05 | 1.29 |

| Alive | Sia | This Is Acting | 2016 | 4:23 | 1.22 |

| Why so Serious? | Hans Zimmer | Best of | 2016 | 9:14 | 1.29 |

| Ain't It Funny | Jennifer Lopez | [None] | | 4:17 | 0.99 |

| Chandelier | Sia | [None] | 2014 | 3:36 | 1.32 |

| Break the Ice | Britney Spears | Blackout | 2007 | 3:16 | 1.29 |

| Moonlight Reggae | Diego Modena | [None] | 2010 | 4:43 | 1.19 |

----------------------------------------------------------------------------------------

**Submission (30%)**

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

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/9.1.0/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 305XXX\_w7\_lab

and follow the instructions. Replace XXX with the section letter(s) specified by your instructor.

**⚠️Important:** Please note that a successful submission does not guarantee full credit for this workshop. If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.

***At-Home***

The *at-home* part of this workshop upgrades your *in-lab* solution to include more functionality in the SongCollection module.

**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 the list of songs of that artist available in the collection. ***DO NOT USE MANUAL LOOPS!***

**w7 Module**

The tester module for the at-home 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 input file is provided):

w7.exe 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.**

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?

**Quiz Reflection**

Add a section to reflect.txt called **Quiz X Reflection**. Replace the **X** with the number of the last quiz that you received and list all questions that you answered incorrectly.

Then for each incorrectly answered question write your mistake and the correct answer to that question. If you have missed the last quiz, then write all the questions and their answers.

**Submission (30% for code, 40% for reflection)**

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/9.1.0/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 305XXX\_w7\_home

and follow the instructions. Replace XXX with the section letter(s) specified by your instructor.

**⚠️Important:** Please note that a successful submission does not guarantee full credit for this workshop. If the professor is not satisfied with your implementation, your professor may ask you to resubmit. Resubmissions will attract a penalty.