**Workshop 2: Move and Copy Semantics**

In this workshop, you work with a large dynamically allocated array of C++ Standard Library strings and compare the performance of copy and move operations on that collection.

**Learning Outcomes**

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

* retrieve records from a text file using an input file stream object
* count the number of records in a text file
* monitor the time spent on a particular task using the std::chrono library
* implement **copy semantics** for a class with a resource
* implement **move semantics** for a class with a resource
* identify the processing-intensive operations in copy and move assignments

**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%)**

This workshop consists of three modules:

* w2 (supplied)
* TimedEvents
* StringSet

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

**w2 Module (supplied)**

**Do not modify this module!** Look at the code and make sure you understand it.

**TimedEvents Module**

Design and code a class named TimedEvents that manages a **statically** allocated array of Event objects. Your class predefines the maximum number of event objects at 10. The **instance variables** for your class should include:

* the number of records currently stored
* the start time for the current event (an object of type std::chrono::steady\_clock::time\_point; see documentation [here](https://en.cppreference.com/w/cpp/chrono/time_point))
* the end time for the current event (an object of type std::chrono::steady\_clock::time\_point)
* an array of Events of structure type. The structure should contain the following fields:
  + a string with the event name.
  + a string with the predefined units of time
  + the duration of the event (an object of type std::chrono::steady\_clock::duration; see documentation [here](https://en.cppreference.com/w/cpp/chrono/duration))

Your class includes the following member functions:

* a default constructor
* startClock(): a modifier that starts the timer for an event
* stopClock(): a modifier that stops the timer for an event
* addEvent(): a modifier that receives the address of a C-style null-terminated string that holds the name of the event. This function will update the next time-record in the array:
  + stores into the name attribute the C-style string at the pointed to address
  + stores "nanoseconds" as the units of time
  + calculates and stores the duration of the event (use std::chrono::duration\_cast<std::chrono::nanoseconds>(), see documentation [here](https://en.cppreference.com/w/cpp/chrono/duration/duration_cast))
* a **friend insertion operator** that receives a reference to an std::ostream object and a TimedEvents object. This operator should insert in the std::ostream object the records from the array in the following format:
* Execution Times:
* --------------------------
* EVENT\_NAME DURATION UNITS
* EVENT\_NAME DURATION UNITS
* ...
* --------------------------

The **name** of the event should be in a field of size 21, aligned on the left; the **duration** should be in a field of size 13, aligned on the right.

Starting and stopping the timer means getting the current time (use std::chrono::steady\_clock::now(); see documentation [here](https://en.cppreference.com/w/cpp/chrono/steady_clock/now)).

**StringSet Module**

Design and code a class named StringSet that manages a **dynamically** allocated array of elements of type std::string. Your class keeps track of the number of strings currently stored and defines the following member functions:

* a no-argument default constructor
* a 1-argument constructor that receives the address of a C-style null terminated string containing the name of a file from which this member function populates the current object. This function
  1. reads the file to count the number of strings present in the file (the record delimiter should be a single space ' ')
  2. allocates memory for that number of strings in the array
  3. re-reads the file and loads the strings into the array.
* size\_t size(): a query that returns the number of strings stored in the current object.
* std::string operator[](size\_t): a query that returns a copy of the string at the index received as the function parameter. If the index is invalid, this function should return an empty string.

To review the syntax for reading from a text file using an std::ifstream object see the chapter in your notes entitled [Custom File Operators](https://ict.senecacollege.ca/~btp200/pages/content/custo.html).

Add any other members that your design requires (without changing the specs above)!

**Sample Output**

When the program is started with the command:

ws gutenberg\_shakespeare

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

**Note:** The execution times will be different every time you run the program! Everything else should match.

**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\_w2\_p1

and follow the instructions.