**Workshop 9: Multi-Threading**

In this workshop, you backup an encrypted text file of characters to a binary file, restore the encrypted data from the binary file and add multi-threading to the solution.

**Learning Outcomes**

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

* process partitioned data on two or more threads
* write a set of characters to a file in binary mode
* read a set of characters from a file in binary mode
* bind a function to its arguments

**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).

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 part 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/9.1.0/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.

If, when you try to compile/submit the workshop on matrix, you encounter an error regarding /lib64/libstdc++.so.6: version 'CXXABI\_1.3.9 not found', add the following line at the end of your .bashrc file (this is a hidden text file located in the home folder--~--of your matrix account):

export LD\_LIBRARY\_PATH=/usr/local/gcc/9.1.0/lib64:$LD\_LIBRARY\_PATH

After you edit the file, logout and login again--this problem should go away.

**Part 1 (0%)**

The three source files provided for this workshop are

* SecureData.h -- already implemented, **do not modify**
* SecureData.cpp -- partially implemented
* w9.cpp -- already implemented, **do not modify**

The SecureData class holds text in encoded form along with the number of bytes in the text including the null terminator.

The Cryptor function object holds the encryption/decryption logic used by the SecureData class. This object is fully implemented.

Your tasks for this part of the workshop are to complete the backup() and restore() member functions of the SecureData class. Your code in each case should open the binary file in the appropriate mode and throw an exception if opening fails.

For backup, your code simply writes the data to the file.

For restoration, your code deallocates the existing data, determines the number of bytes to be read, allocates memory for them, and reads the data from the file.

**w9 Module (supplied)**

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):

w9.exe input.txt encoded.dat.txt K

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

**⚠️Important:** The output file encoded.dat is not automatically checked; before submission make sure that the file contains binary encoded data (and not the original text). Submissions that do not have encoded data in encoded.dat will be rejected.

**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/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 345\_w9\_p1

and follow the instructions.

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

**Part 2 (100%)**

Your second task is to multi-thread the conversion in the code() member function of the SecureData class.

Select a reasonable number of threads and partition the data equally amongst all of the threads. Use std::bind from the functional library to bind the converter.

**w9 Module (supplied)**

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):

w9.exe input.txt encoded.dat.txt K

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

**⚠️Important:** The output file encoded.dat is not automatically checked; before submission make sure that the file contains binary encoded data (and not the original text). Submissions that do not have encoded data in encoded.dat will be rejected.

**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:

* why was necessary to use binary files in this workshop and how did you use **them**
* **why would be important to bind a function to its arguments, and how was it useful in this workshop**
* **the advantages of using multiple threads, and how did you accomplished multi-threading in this workshop**

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/9.1.0/bin/g++) and make sure that everything works properly. Add switch to your compilation command -pthread to enable threading capabilities.

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

~profname.proflastname/submit 345\_w9\_p2

and follow the instructions.

**⚠️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.