# Workshop 9

*Multi-Threading and Encryption*

In this workshop, you encrypted a text file of characters while backing it up to a binary file, decrypt the characters while restoring the encrypted data from the binary file and add multi-threading to your 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 *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). The *at-home* portion of the lab 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 and student number.

You are responsible to back up your work regularly.

### Late Submission Penalties:

- *In-lab* portion submitted late, with *at-home* portion: **0** for *in-lab*. Maximum of 7/10 for the entire workshop.
- If any of *in-lab*, *at-home* or *reflection* portions is missing, the mark for the workshop will be **0**/10.

## SPECIFICATIONS – IN LAB

The modules that constitute this workshop are listed below along with their file names

1. **SecureData** module**:**
   **SecureData.h** (supplied)
   **SecureData.cpp** (incomplete)
2. **Main** module**: w8.cpp** (supplied)

The data file supplied with this workshop is: **text.dat**

```
Compound types are types that a programmer constructs from a language's
fundamental types and/or other compound types.  In C++11, compound types
include pointers and references to existing types, enumerations of named
constant values, arrays of objects of an existing type, classes and function
types.  C++11 clarified the difference between lvalues and rvalues and
references to each, which helped simplify many of the facilities available
in the standard library.

This chapter describes the non-function types in detail.  The description
includes a review of one-dimensional arrays in both static and dynamic memory
and shows how to allocate multi-dimensional arrays in both static and dynamic
memory.  The description of classes reviews class definitions, introduces
move-constructors and move-assignment operators and includes declarations of
special members such as bit fields, class variables and class functions.
```

The output from your executable running Visual Studio with the following command line argument should look like

```
Command Line : C:\Users\...\Debug\in_lab.exe text.dat encoded.dat C

921 bytes copied from file text.dat into memory (null byte added)
Data encrypted in memory

922 bytes copied from binary file encoded.dat into memory.
Data decrypted in memory

Compound types are types that a programmer constructs from a language's
fundamental types and/or other compound types.  In C++11, compound types
include pointers and references to existing types, enumerations of named
constant values, arrays of objects of an existing type, classes and function
types.  C++11 clarified the difference between lvalues and rvalues and
references to each, which helped simplify many of the facilities available
in the standard library.

This chapter describes the non-function types in detail.  The description
includes a review of one-dimensional arrays in both static and dynamic memory
and shows how to allocate multi-dimensional arrays in both static and dynamic
memory.  The description of classes reviews class definitions, introduces
move-constructors and move-assignment operators and includes declarations of
special members such as bit fields, class variables and class functions.
```

The last command line argument (**c**) is the key to be used in the encryption and decryption.

### Your Tasks

Your tasks for this part of the workshop are to implement the functionality for the **SecureData** module:

1. Complete the definition of the **SecureData::backup** member function
2. Complete the definition of the **SecureData::restore** member function

## In-Lab Submission (30%)

To test and demonstrate 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 gcc compiler 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 345XXX_w9_lab**<ENTER>

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

## SPECIFICATIONS – AT HOME

In this part of the workshop you implement multi-threading in the **SecureData::code(char key)** member function.

### 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. Take your time. Explain in your own words what you have learned in completing this workshop. Include in your explanation but do not limit it to the following points (40%):

- Binary file operations
- Binding a function to its arguments
- Multi-threading

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

Include all corrections to the Quiz(zes) you have received (30%).

## At-Home Submission (70%)

To test and demonstrate 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 gcc compiler 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 345XXX_w9_home**<ENTER>

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