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
- to count the number of records in a text file
- to monitor the time spent on a particular task using the `std::chrono` library
- to implement copy semantics for a class with a resource
- to implement move semantics for a class with a resource
- to identify the processing-intensive operations in copy and move assignments

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

This workshop consists of three modules:

- **w2** (supplied)
- **Timekeeper**
- **Text**

Enclose all your source code within the **sict namespace** and include the necessary guards in each header file. The output from your executable running Visual Studio with the following command line argument should look like

```
Command Line : C:\Users\...\Debug\w2_in_lab.exe gutenberg_shakespeare
0-arg Constructor - a.size = 0
1-arg Constructor - b.size = 124456
Copy Constructor  - c.size = 124456
Copy Assignment   - a.size = 124456

Execution Times:
0-arg Constructor     0 seconds
1-arg Constructor  1400 seconds
Copy Constructor    171 seconds
Copy Assignment     176 seconds
Destructor          387 seconds
```

Timekeeper Module

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

a. The number records currently stored
b. The start time for the current event (an **std::chrono::steady_clock::time_point** object)
c. The end time for the current event (an **std::chrono::steady_clock::time_point** object)
d. An array of records of anonymous structure (no Tag) type named **record** where each element contains:
    a. The address of a C-style null-terminated string containing a message in a literal string.
    b. The address of a C-style null-terminated string containing the predefined units of time
    c. The duration of the recorded event (an **std::chrono::steady_clock::duration** object)

Your class includes the following member functions:

- a default no-argument constructor
- **start()** a modifier that starts the stopwatch for an event
- **stop()** a modifier that stops the stopwatch for an event
- **recordEvent()** a modifier that receives the address of a C-style null terminated string that holds the description of the event – this function copies the address of the description into the next time record, calculates the duration of the event and copies it into the next

time record, copies the address of the literal string containing a description of the units of time into the next time records, and increments the instance variable that identifies the next empty time record.

- **report()** a query that receives a reference to an **`std::ostream`** object, inserts the title **`"Execution Times:"`** into the object and inserts each of the time records stored in the **Timekeeper** object.

Text Module

Design and code a class named **Text** that manages a dynamically allocated array of **std::string**s. 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 counts the number of records present in the file, allocates memory for that number of pointers to **std::string** object and copies the records into the objects. Note that you can determine the number of records by counting the number of newline characters in a file.
- a copy constructor
- a copy assignment operator
- a destructor
- **size_t size() const** a query that returns the number of records stored in the current object.

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`**. See also **`cplusplus.com`**

Execution

A text file named **gutenberg_shakespeare** has been included in the directory containing the Visual Studio project file. Make sure to include this file name as a command line argument.

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_w2_lab<ENTER>

and follow the instructions. Replace **XXX** with the section letters specified by your instructor.

SPECIFICATIONS – AT HOME

For this part of the workshop, upgrade your w2 module to call the move constructor and the move assignment operator on your **Text** class. Place your code snippet after the one for copy assignment and before the final call to start **Timekeeper t**. Upgrade your **Text** class to include a move constructor and a move assignment operator. Do not modify your **Timekeeper** module.

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

```
Command Line : C:\Users\...\Debug\w2_at_home.exe gutenberg_shakespeare
0-arg Constructor - a.size = 0
1-arg Constructor - b.size = 124456
Copy Constructor  - c.size = 124456
Copy Assignment   - a.size = 124456
Move Constructor  - d.size = 124456
Move Assignment   - a.size = 124456

Execution Times:
0-arg Constructor     0 seconds
1-arg Constructor  1451 seconds
Copy Constructor    170 seconds
Copy Assignment     168 seconds
Move Constructor      0 seconds
Move Assignment       0 seconds
Destructor          362 seconds
```

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. 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%):

- the reason for the difference between the copy and move operations
- the dynamic allocation of addresses to objects instead of objects themselves
- the changes you made in upgrading your **w2** and **Text** modules.

To avoid deductions, cite the code in your solution as an example of your implementation of the concepts that you describe.

Include all corrections to the Quiz 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_w2_home<ENTER>

and follow the instructions. Replace **XXX** with the section letters specified by your instructor.