# # 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).

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`:

```bash

**/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`):

```bash

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%)

This workshop consists of three modules:

- `w2` (supplied)

- `TimedEvents`

- `RecordSet`

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 record objects. Your class predefines the maximum number of record objects at 7. 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 records of anonymous structure type (the structure has no name). The structure should contain the following fields:

1. a string with the event name.
2. a string with the predefined units of time
3. the duration of the recorded 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
* `recordEvent()`: 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:

1. stores the C-style string pointed to by the parameter into the name attribute
2. stores `"nanoseconds"` as the units of time
3. 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 20, aligned on the left; the \*\*duration\*\* should be in a field of size 12, 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)).

## ### `RecordSet` Module

Design and code a class named `RecordSet` 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 records present (the record delimiter should be a single space `' '`)
  2. allocates memory for that number of records in the array
  3. re-reads the file and loads the records into the array.
* a copy constructor
* a copy assignment operator
* a destructor
* `size\_t size()`: a query that returns the number of records stored in the current object.
* `std::string getRecord(size\_t)`: a query that returns the record at the index received as the function parameter in an object of type `std::string`. 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 [Input and Output Operators](https://ict.senecacollege.ca/~oop244/pages/content/custo.html).

## ### Sample Output

When the program is started with the command:

```

w2.exe gutenberg\_shakespeare

```

the output should look like:

```

Command Line:

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

1: ws

2: gutenberg\_shakespeare

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

0-arg Constructor - a.size = 0 records -> (a) Record 0: []

(a) Record 100: []

(a) Record 1000: []

(a) Record last: []

1-arg Constructor - b.size = 1293934 records -> (b) Record 0: [This]

(b) Record 100: [USED]

(b) Record 1000: [or]

(b) Record last: [Shakespeare]

Copy Constructor - c.size = 1293934 records -> (c) Record 0: [This]

(c) Record 100: [USED]

(c) Record 1000: [or]

(c) Record last: [Shakespeare]

Copy Assignment - a.size = 1293934 records -> (a) Record 0: [This]

(a) Record 100: [USED]

(a) Record 1000: [or]

(a) Record last: [Shakespeare]

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

Execution Times:

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

0-arg Constructor 400 nanoseconds

1-arg Constructor 143568386 nanoseconds

Copy Constructor 41302993 nanoseconds

Copy Assignment 40913697 nanoseconds

Destructor 18210121 nanoseconds

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

```

\*\*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/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 professorâ€™s Seneca userid):

```

~profname.proflastname/submit 345\_w2\_p1

```

and follow the instructions.

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

â€ƒ

â€ƒ

# ## Part 2 (100%)

For this part of the workshop, upgrade the `RecordSet` class to include a \*\*move constructor\*\* and a \*\*move assignment operator\*\*. No other modules need to be changed.

## ### Sample Output

When the program is started with the command:

```

w2.exe gutenberg\_shakespeare

```

the output should look like:

```

Command Line:

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

1: ws

2: gutenberg\_shakespeare

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

0-arg Constructor - a.size = 0 records -> (a) Record 0: []

(a) Record 100: []

(a) Record 1000: []

(a) Record last: []

1-arg Constructor - b.size = 1293934 records -> (b) Record 0: [This]

(b) Record 100: [USED]

(b) Record 1000: [or]

(b) Record last: [Shakespeare]

Copy Constructor - c.size = 1293934 records -> (c) Record 0: [This]

(c) Record 100: [USED]

(c) Record 1000: [or]

(c) Record last: [Shakespeare]

Copy Assignment - a.size = 1293934 records -> (a) Record 0: [This]

(a) Record 100: [USED]

(a) Record 1000: [or]

(a) Record last: [Shakespeare]

Move Constructor - d.size = 1293934 records -> (a) Record 0: []

(a) Record 100: []

(a) Record 1000: []

(a) Record last: []

(d) Record 0: [This]

(d) Record 100: [USED]

(d) Record 1000: [or]

(d) Record last: [Shakespeare]

Move Assignment - a.size = 1293934 records -> (a) Record 0: [This]

(a) Record 100: [USED]

(a) Record 1000: [or]

(a) Record last: [Shakespeare]

(d) Record 0: []

(d) Record 100: []

(d) Record 1000: []

(d) Record last: []

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

Execution Times:

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

0-arg Constructor 400 nanoseconds

1-arg Constructor 150157814 nanoseconds

Copy Constructor 41364391 nanoseconds

Copy Assignment 40878296 nanoseconds

Move Constructor 300 nanoseconds

Move Assignment 100 nanoseconds

Destructor 17899923 nanoseconds

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

```

\*\*Note:\*\* See that in the sample output above the \*move operations\* are \*\*many orders of magnitude\*\* faster than the \*copy operations\*. If your output doesn't have such a significant difference in times, keep working on your implementation (the actual numbers will be different every time you run the application).

# ### 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 explanationâ€”\*\*but do not limit it to\*\*â€”the following points:

- the reason for the significant time difference between the copy and move operations

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.

Then, run the following command from your account (replace `profname.proflastname` with your professorâ€™s Seneca userid):

```

~profname.proflastname/submit 345\_w2\_p2

```

and follow the instructions.

\*\*:warning: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.