# # Workshop 1: Across Translation Units

In this workshop, you implement aspects of linkage, storage duration, namespaces, header guards, and operating system interfaces.

## Learning Outcomes

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

- receive program arguments from the command line

- guard a class definition from repetition

- access a variable defined in a different translation unit

- declare a local variable that remains in memory for the lifetime of the program

- upgrade code to accept and manage a user-defined string of any length

## 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.

# ## The application

The application tracks the current events that happen at given times in a day, and executes a set of actions on the current event.

The application maintains a system clock (representing the time of a day), then loads from a file the actions that must be executed:

- `T` - \*time\*: all following actions must be executed (in sequence) when the system clock reaches the time following `T`

- `S` - \*start\*: at the current time, a new event starts; the name of the event follows `S`

- `E` - \*end\*: the current event has ended and no other event is happening right now

- `P` - \*print\*: display the current event to the screen

- `A` - \*archive\*: add the current event to an archive of current events for future reference.

## Part 1 (0%)

This workshop consists of two modules:

- `w1` (partially supplied)

- `Event`

### `w1` Module (partially supplied)

Study the code supplied and make sure you understand it.

Finish the implementation of the `main` function, by completing the parts marked with `TODO`:

- write the prototype of the `main` function to receive a set of standard command line arguments

- echoes the set of arguments to standard output in the following format:

```

1: first argument

2: second argument

3: third argument

4: ...

```

\*\*Do not modify this module in any other place!\*\*

### `Event` Module

The `Event` module defines a system clock, as a global variable named `g\_sysClock` that stores only positive integers. The value of the clock represents the time of day as the number of seconds since midnight (an integer between 0 and 86400; choose an appropriate type). The clock will be accessed when a new event starts and from the `main` function.

This module also defines a class named `Event` in the namespace `sdds` that stores some information about an event:

- a C-style null-terminated string of up to 128 characters \*including the null byte terminator\* representing the description of the event.

- the time when the event starts, expressed in number of seconds that passed from midnight. The time must be an integer between 0 and 86400 (choose an appropriate type).

## \*\*Public Members\*\*

### - Default constructor

- `display()`: a query that displays to the screen the content of an instance in the following format:

```

COUNTER. HH:MM:SS -> DESCRIPTION

```

If there is no description stored, this query should print:

```

COUNTER. [ No Event ]

```

where

- `COUNTER` is a field of size 3, that represents how many times this function has been called (use a local-to-function variable that remains in memory for the lifetime of the program) \*\*Do not use global/member variables to store the counter!\*\*

- `HH:MM:SS` represents the time when the event started, expressed in hours, minutes and seconds

### - `setDescription()`:

a modifier that receives as a parameter an array of characters. If the parameter is not null and not empty, it means that a new event has started and the information about this event must stored in the current instance. If the **parameter is null or empty string**, this function resets the current instance to an empty state.

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

### Sample Output

The input files `monday.txt` and `tuesday.txt` are already provided; the main module contains a description of the structure for these files.

When the program is started with the command:

```

w1.exe monday.txt tuesday.txt

```

the output should look like:

```

Command Line:

1: w1.exe

2: monday.txt

3: tuesday.txt

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

Day 1

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

1. 00:02:55 -> Computer Starting

2. 00:02:55 -> Computer Starting

3. [ No Event ]

4. 01:15:34 -> User logging in

5. 01:20:09 -> User logging in

6. 09:01:04 -> Browser closed

7. 17:33:33 -> User checks email

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

Day 2

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

8. 01:02:23 -> User starts working on homework

9. [ No Event ]

10. 01:20:34 -> User take a break

11. 01:22:30 -> User plays sudoku

12. 01:26:40 -> User resumes homework

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

Archive

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

13. [ No Event ]

14. 01:20:00 -> Authentication Failed

15. 01:23:20 -> Browser starts

16. 09:01:04 -> Browser closed

17. 17:33:33 -> User checks email

18. 01:02:23 -> User starts working on homework

19. 01:22:30 -> User plays sudoku

20. [ No Event ]

21. 01:26:40 -> User resumes homework

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

```

### Test Your Code

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 `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\_w1\_p1

```

and follow the instructions.

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

## Part 2 (100%)

For this part of the workshop, upgrade your `Event` class such that the description can be a C-style null-terminated string of any length. \*\*Make sure your program doesn't have memory leaks.\*\*

### Sample Output

When the program is started with the command:

```

w1.exe monday.txt tuesday.txt

```

the output should look like:

```

Command Line:

1: w1.exe

2: monday.txt

3: tuesday.txt

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

Day 1

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

1. 00:02:55 -> Computer Starting

2. 00:02:55 -> Computer Starting

3. [ No Event ]

4. 01:15:34 -> User logging in

5. 01:20:09 -> User logging in

6. 09:01:04 -> Browser closed

7. 17:33:33 -> User checks email

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

Day 2

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

8. 01:02:23 -> User starts working on homework

9. [ No Event ]

10. 01:20:34 -> User take a break

11. 01:22:30 -> User plays sudoku

12. 01:26:40 -> User resumes homework

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

Archive

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

13. [ No Event ]

14. 01:20:00 -> Authentication Failed

15. 01:23:20 -> Browser starts

16. 09:01:04 -> Browser closed

17. 17:33:33 -> User checks email

18. 01:02:23 -> User starts working on homework

19. 01:22:30 -> User plays sudoku

20. [ No Event ]

21. 01:26:40 -> User resumes homework

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

```

## ### 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:

- the difference between internal and external linkage using examples from your code

- what are `static` variables and how were they useful in this workshop.

- the changes that you made in upgrading your `Event` class.

## ### 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 professor’s Seneca userid):

```

~profname.proflastname/submit 345\_w1\_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.