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 *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). **In order to get credit for the *in-lab* portion, you must be present in the lab for the entire duration of the lab.**

The *at-home* portion of the workshop 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, student number and the date of completion (use the following template):

```
// Name:
// Seneca Student ID:
// Seneca email:
// Date of completion:
//
// I confirm that the content of this file is created by me,
//   with the exception of the parts provided to me by my professor.
```

You are responsible to back up your work regularly.

Late Submission Penalties

The workshop can be submitted up to **1 (one) day** late (the day that is 5 days after the lab period); submissions received on this day are considered **late** and are subject to penalties:

- only *in-lab* portion submitted late (after the end of the lab period): 0 for *in-lab* portion, max 7/10 for the entire workshop.
- only *at-home* portion submitted late (more than 4 days after the lab period): max 4 for the *at-home* portion, max 7/10 for the entire workshop.

- both *in-lab* **and** *at-home* portions submitted late: max 4/10 for the entire workshop.
- when the submission closes, if the workshop is not complete, the mark for the entire workshop will be 0/10. The workshop is considered complete if there are two separate submissions (*in-lab* submission and *at-home* submission) containing the *in-lab code*, *at-home code* and *reflection*.

The submission is considered closed at the end of the day that is 5 (five) days after the lab period.

The application

The application tracks the current event that happens at a given time in the 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 screen - `A` - *archive*: add the current event to an archive for future reference.

In-Lab

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 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
--------------------
```

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 `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 345XXX_w1_lab
```

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

: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.

At-Home

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.**

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.

Quiz Reflection

Add a section to `reflect.txt` called **Quiz X Reflection**. Replace the **X** with the number of the last quiz that you received and list all questions that you answered incorrectly.

Then for each incorrectly answered question write your mistake and the correct answer to that question. If you have missed the last quiz, then write all the questions and their answers.

Submission (30% for code, 40% for reflection)

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 345XXX_w1_home
```

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

: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.