**Workshop 1: Across Translation Units**

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

**Learning Outcomes**

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

* declare a local variable that remains in memory for the lifetime of the program
* guard a class definition from repetition
* access a variable defined in a different translation unit
* receive program arguments from the command line
* 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.**

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 usbmitted up to **1 (one) day** late (the day that is 5 days after the lab period); submissions received on this day are considered **late** 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 submited late (more than 4 days after the lab period): 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.

***In-Lab***

This workshop consists of three modules:

* w1 (partially supplied)
* process (supplied)
* String

Enclose all your source code within the sdds namespace and include the necessary guards in each header file.

**w1 Module (partially supplied)**

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!**

**process Module**

The process module is supplied and you do not need to change its header or implementation files. The implementation file defines a single function named process() that receives the address of an unmodifiable C-style null terminated string, constructs a String object from the C-style string and inserts that object into standard output followed by a newline.

**String Module**

Your String module defines a class named String that holds a C-style null-terminated string of up to 3 characters *excluding the null byte terminator*. The class includes the following member and helper functions:

* a one-argument constructor that receives the address of an unmodifiable C-style null-terminated string and copies the string at that address into an instance variable. Check for receipt of the null address and store an empty string in that case.
* a query named display() that receives a reference to an std::ostream object and inserts the string stored in the instance variable.
* a **free helper** operator that inserts the saved string into the left operand. This operator should count how many times it has been called (by incrementing a counter), and prints this value to the screen as shown below. **Do not use global variables to store the counter!**

COUNTER: STRING

Do not use the string class of the standard library in this workshop. Use only cstring functions where necessary.

In the namespace sdds, this module should contain a *global variable* named g\_maxSize of type unsigned int (an integer that has non-negative values). Initialize it with 3. Declare the variable in the header and define it in the implementation file.

**Sample Output**

When the program is started with the command:

w1.exe Welcome to C++!

the output should look like:

Command Line:

1: w1.exe

2: Welcome

3: to

4: C++!

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

Global variable:

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

Old Value: 3

New Value: 4

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

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

Processing:

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

1: Wel

2: to

3: C++

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

**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 305XXX\_w1\_lab

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

**⚠️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 String class to manage a C-style null-terminated string of any length. When a new instance is created, the custom constructor should accept at most g\_maxSize characters (excluding the null terminator). No other modules need to be changed.

**Sample Output**

When the program is started with the command:

w1.exe Welcome to C++!

the output should look like:

Command Line:

1: w1.exe

2: Welcome

3: to

4: C++!

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

Global variable:

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

Old Value: 3

New Value: 4

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

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

First processing:

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

1: Welc

2: to

3: C++!

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

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

Second processing:

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

4: Welco

5: to

6: C++!

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

Welcome

Welcome

School of Software Design and Data Science

7: Welcome

8: Welcome

9: Welcome

**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 String 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 305XXX\_w1\_home

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

**⚠️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.