COMPILING MODULES

Workshop 1 (draft V0.91)
Submission is not open yet.

In process of doing your first workshop, you are to sub-divide a program into modules, compile each module separately and construct an executable from the results of each compilation.

LEARNING OUTCOMES

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

- organize source code into modules, using header and implementation files;
- compile and link modular programs;
- distinguish the contents of a header and an implementation file;
- describe to your instructor what you have learned in completing this workshop.

SUBMISSION POLICY

This workshop has only one section;

Workshop1 is to be completed before the end of the next day after your workshop period. To get the mark for this workshop you must attend the lab.

Start the workshop few days earlier and come to the workshop sessions with questions and attendance.

Remember: you must be present at the lab in order to get credit for the workshop.

Reflection, Citation and Sources

After the workshop is completed, 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. Add any other comments you wish to make.

Also, when submitting Workshops, Project and assignment deliverables, a file called **sources.txt** must be present. This file will be submitted with your work automatically.

You are to write either of the following statements in the file "**sources.txt**":

I have done all the coding by myself and only copied the code that my professor provided to complete my workshops and assignments.

Then add your name and your student number as signature

OR:

Write exactly which part of the code of the workshops or the assignment are given to you as help and who gave it to you or which source you received it from.

You need to mention the workshop name or assignment name and also mention the file name and the parts in which you received the code for help.

Finally add your name and student number as signature.

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.

Late Submission Penalties:

Late submission: 1 to 6 days -50% after that submission rejected.

- If the content of sources.txt or reflect.txt is missing, the mark for the whole workshop will be **0.**

WORKSHOP DUE DATES

You can see the exact due dates of all assignments by adding -due after the submission command:

Run the following script from your account (use your professor's Seneca userid to replace `profname.proflastname`, and replace 2?? with your course code, i.e. 244 or 200 and NXX with your section ID i.e., NAA, NBB, etc.):

~profname.proflastname/submit 2??/NXX/WS01/ws -due<ENTER>

Example:
~john.doe/submit 200/NAA/WS01/ws -due
~Jane.doe/submit 244/NBB/WS01/ws -due.

ORIGINAL SOURCE CODE (THE WORDSTAT PROGRAM)

`wordStat` is a program that reads a text file from standard input and analyzes and reports the number of words and their occurrences in the text file.

To feed the text file into the program through standard input we need to use the redirection operator (recall ULI101) of the operating system (that is " < ").

Using command line, if the executable of the program is called "ws", and we need to analyze a text file called "text.txt", we would do so with the following command in **Matrix** (or another Linux command line interface):

`ws < text.txt` <ENTER>

See the instructions below to find out how to setup your **Visual Studio** to do the same thing.

NOTE: For this part, if you have not setup your computer, it is better to do this lab at school and on a lab computer since it has "Git" and "Tortoise Git" installed. If you do have "Git" or "Tortoise Git" installed on your own computer, you can do this on your own personal computers.

NOTE: Installation guides for preparing your computer for the subject can be found in this playlist:
https://www.youtube.com/playlist?list=PLxB4x6RkylosAh1of4FnX7-g2fk0MUeyc
OR:
https://tinyurl.com/244setup

RETRIEVE THE ORIGINAL PROGRAM:

First go to MyApps on the lab computers and launch:
TortoiseGit
Putty
Visual Studio 2019

Workshop Steps:

1. Open https://github.com/Seneca-244200/OOP-Workshops and click on "Clone or download" Button; this will open "Clone with HTTPS" window.

> **NOTE**: If the window is titled "Clone with SSH" then click on "Use HTTPS":

2. Copy the https URL by clicking on the button on the right side of the URL:

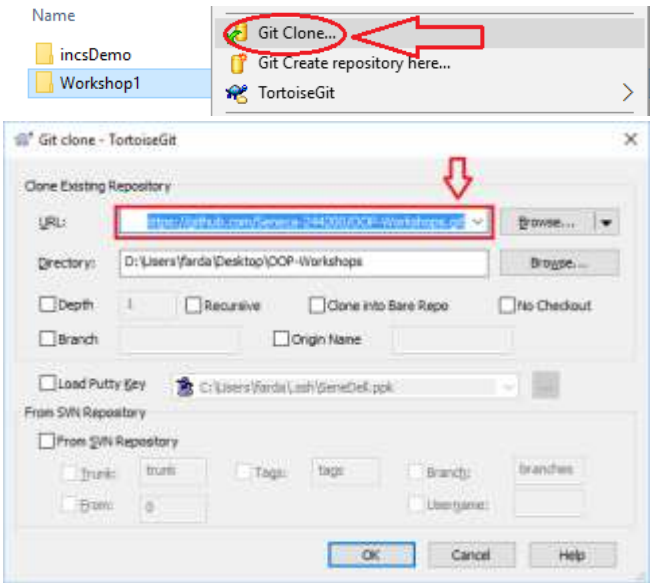
3. Open File Explorer on your computer and select or create a directory for your workshops.

Now Clone (download) the original source code of `w1.cpp` (Workshop1) from GitHub in one of the following three ways: (methods 1 and 2 are preferred)

1. Using `TortoiseGit`:

   a. Right click on the selected directory and select "Git Clone":

   This will open the "Git Clone" window with the URL already pasted in the "URL" text box; if not, paste the URL manually.



b. Click on OK.

This will create on your computer a clone (identical directory structure) of the directory on `Github`. Once you have cloned the directory, you can open the directory `OOP-Workshops/WS01` and start doing your workshop. Note that you will repeat this process for all workshops and milestones of your project in this subject.

IMPORTANT: If your professor makes any changes to the workshop, you can right click on the cloned repository directory and select `TortoiseGit/pull` to update and sync your local workshop to the one on `Github` without having to download it again. Note that this will only apply the changes made and will not affect any work that you have done on your workshop.

2. Using the command line:

a. Open the `git` command line on your computer.
b. Change the directory to your workshops directory.
c. Issue the following command at the command prompt in your workshops directory:

```
git clone https://github.com/Seneca-244200/OOP-Workshops.git<ENTER>
```

IMPORTANT: The URL for all the workshops are the same throughout the semester. The only thing that changes, is the workshop number.

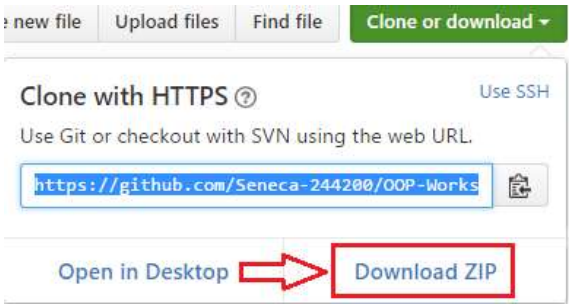
This will create on your computer a clone (identical directory structure and content) of the `OOP-Workshops` directory on `Github`. Once you have cloned the directory, you can open subdirectory `OOP-Workshops/WS01` and start doing your workshop. Note that you will repeat this process for all workshops and milestones of your project in this subject.

IMPORTANT: If your professor makes any changes to the workshop, you can issue the command `git pull<ENTER>` in the cloned repository directory to update and sync your local workshop to the one on `Github` without having to download it again. Note that this will only apply the changes

made and will not affect any work that you have done on your workshop.

3. Using the "Download ZIP" option:

   a. Open https://github.com/Seneca-244200/OOP-Workshops and click on "Clone or download" button and click on "Download ZIP".



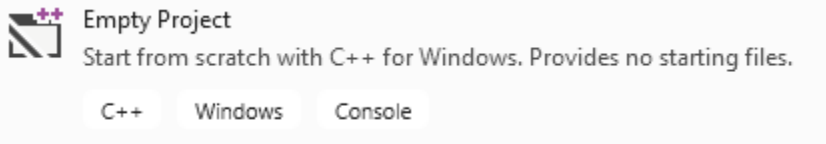
   b. This will download to your computer a zipped file copy of the workshop repository in `Github`. You can extract this file to where you want to do your workshop.

IMPORTANT: Note that, if your professor makes any changes to the workshop, to get them you have to separately download another copy of the workshop and manually apply the changes to your working directory to **make sure nothing of your work is overwritten by mistake**.

STEP 1: TEST THE PROGRAM

1. On Windows, using Visual Studio (VS)

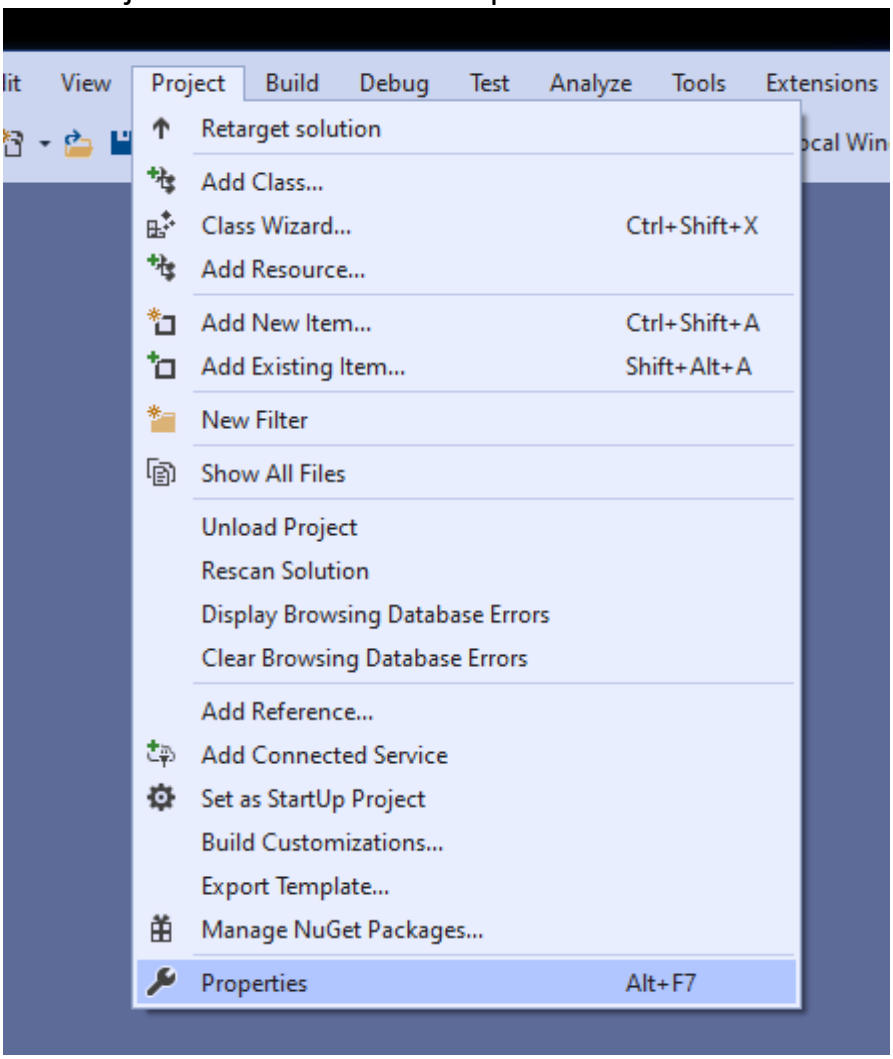
   a. Open Visual studio 2019 and create an Empty C++ Windows Console Project:



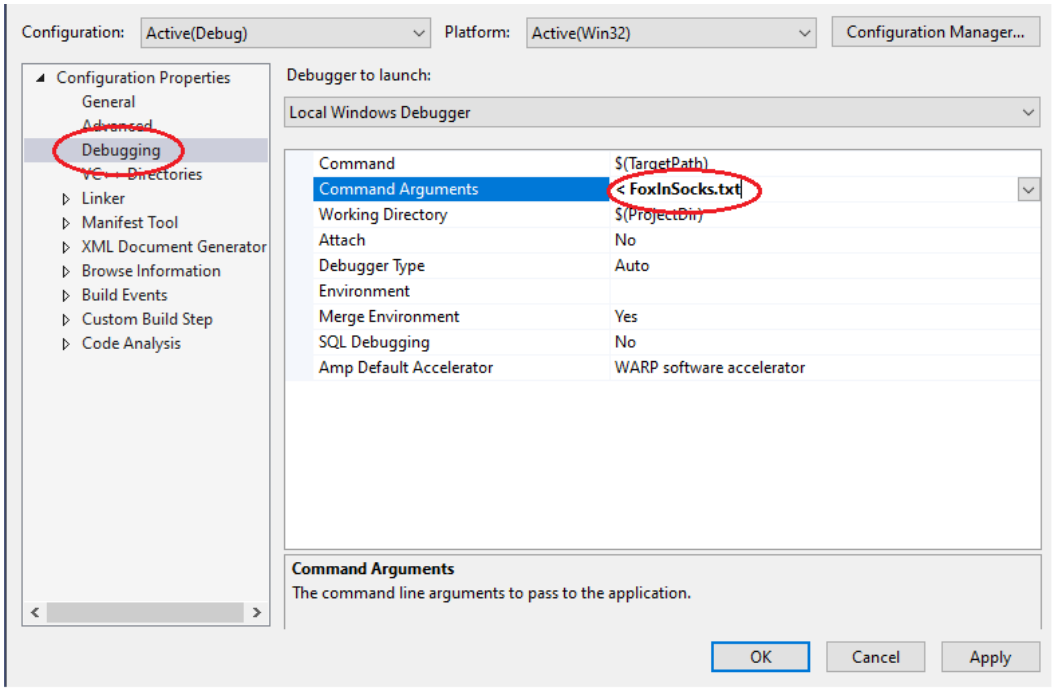
   b. In VS, (if not open already) open Solution Explorer *(click on View/Solution Explorer)* and then add `w1.cpp` file to your project:
      - Right click on "Source Files"
      - Select "Add/Existing Item"
      - Select `w1.cpp` from the file browser
      - Click on "Ok"

c. To add the command line arguments needed to feed the "FoxInSocks.txt" file to the program through the standard input do the following:

In "Project" menu click on "Properties":



In “Properties” window click on “Debugging” and then in “Command Arguments” section enter: **“< FoxInSocks.txt”** and click on ok.



d. Compile and run the program by selecting “Debug/Start without Debugging” or pressing “Ctrl-F5”.

2. On Linux, in your `matrix` account.

e. Upload `w1.cpp` and `FoxInSocks.txt` to your matrix account (Ideally to a designated directory for your oop244 workshop solutions). Then, enter the following command to compile the source file and create an executable called `w1`:

```
g++ w1.cpp -Wall -std=c++11 -o ws<ENTER>
```

- `-Wall`: display all warnings
- `-std=c++11`: compile using C++11 standards
- `-o w1`: name the executable `w1`

f. Type the following to run and test the execution:

```
ws < FoxInSocks.txt<ENTER>
```

STEP 2: CREATE THE MODULES

1. On Windows, using Visual Studio (VS)

   In solution explorer, add three new modules to your project:

- `WordStat;` A module to hold the main() function and its relative functions. (see below)
- `Word;` A module to hold the functions related to Word processing and analysis.
- `Tools;` A module to hold the general helper functions.

The `WordStat` module has an implementation (`.cpp`) file but no header file. The Word and Tools modules have both implementation (`.cpp`) and header (`.h`) files:

- Add `Word.h` and `Tools.h` to the "Header Files" directory (right click on "Header Files" and select "Add/New Item" and add a header file)

  Make sure you add the **compilation safeguards*** and also have all the code in Word and Tools Modules in a namespace called "**sdds**".

  ***** **compilation safeguards** refer to a technique to guard against multiple inclusion of header files. It does so by applying macros that check against a defined name:

```
#ifndef NAMESPACE_HEADERFILENAME_H // replace with relevant names
#define NAMESPACE_HEADERFILENAME_H

// Your header file content goes here

#endif
```

  If the name isn't yet defined, the **#ifndef** will allow the code to proceed onward to then define that same name. Following that the header is then included. If the name is already defined, meaning the file has been included prior (otherwise the name wouldn't have been defined), the check fails, the code proceeds no further and the header is not included again.

  Additionally, here is an instructional **video** showing how the compiler works and why you need these safeguards in all of your header files. **Do note that this video describes the intent and concept behind safeguards, the naming scheme isn't the standard for our class. Follow the standard for safeguards as described in your class**. https://www.youtube.com/watch?v=EGak2R7QdHo

- Add `WordStat.cpp`, `Word.cpp` and `Tools.cpp` to the "Source Files" directory (right click on "Source Files" and select "Add/New Item" and add a C++ file)

  The content of WordStat.cpp file should be exactly as following:

```
#include "Word.h"
using namespace sdds;

int main() {
   programTitle();
   wordStats(true);
   return 0;
}
```

Separate the rest of the functions in `w1.cpp` and copy them into Word and Tools modules as you find fit. Copy the body of the functions into the **cpp** files and the prototype into the **header** files.

To test that you have done this correctly, you can compile each module separately, by right clicking on `Tools.cpp` or `Word.cpp` and select `compile` from the menu. If the compilation is successful, most likely you have done it correctly.

> **NOTE**: The equivalent of this on matrix is to add `-c` to the compile command:
>
> **`g++ Tools.cpp –Wall -std=c++11 -c`**<ENTER>
>
> This will only compile `Tools.cpp` and will not create an executable.

Now remove `w1.cpp` from the project. You can do this by right clicking on the filename in solution explorer and selecting remove in the menu (make sure you do not delete this file but only remove it).

Compile and run the project (as you did before in Step 1) and make sure everything works.

2. On Linux, in your `matrix` account.

Upload the five files from earlier (Tools.cpp, Tools.h, Word.cpp, Word.h and WordStat.cpp) to your matrix account and compile the source code using the following command.

`g++ Tools.cpp Word.cpp WordStat.cpp -Wall -std=c++11 -o ws`<ENTER>

Run the program like before with the **`FoxInSocks.txt`** and make sure that everything still works properly.

SUBMISSION

To test and demonstrate execution of your program use the same data as the output example above.

If not on matrix already, upload `Tools.cpp`, `Tools.h`, `Word.cpp`, `Word.h`, `WordStat.cpp`, `reflect.txt` and `sources.txt` to your `matrix` account. Compile and run your code and make sure that everything works properly.

Then, run the following script from your account (use your professor's Seneca userid to replace `profname.proflastname`, and replace 2?? with your course code, i.e. 244 or 200 and NXX with your section ID i.e., NAA, NBB, etc.):

`~profname.proflastname/submit 2??/NXX/WS01/ws` <ENTER>

and follow the instructions generated by the command and your program.

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.