CS1010E: Programming Methodology

Take Home Lab 0: Introduction

TBA

Setup [#1]

Problem Description

The website http://www.comp.nus.edu.sg/~cs1010e/ contains information regarding the setup needed to write, compile, and run your C programs. Go to the website and click the "Home" tab for more information.

Final Objective

Setup your machine for CS1010E.

Bash [#2]

Problem Description

"Bash is a Unix shell and command language written by Brian Fox for the GNU Project as a free software replacement for the Bourne shell. First released in 1989, it has been distributed widely as the default shell for Linux distributions and Apple's macOS (formerly OS X). In 2016 it was also made available by Microsoft for use in Windows 10 Anniversary Update, albeit not installed by default." – Wikipedia

Bash comes before graphical user interface (GUI). As such, it does not take mouse as an input. All the commands it accepts are written through your keyboard. Therefore, navigating through folders and files can be troublesome if you are not familiar with the system.

Fortunately, I provide a simple tutorial for you to get yourself familiar with Bash.

Final Objective

Get yourself familiar with Bash terminal through this tutorial.

Assumptions

The following assumptions are considered to be true, they limit the inputs to the following restrictions:

▷ You have setup your machine as in Task 1

Tasks

The problem is split into 7 tasks. In the sample run, please note the following:

- ↩ is the *invisible* [**newline**] character.
- User input in blue and program output in purple color.
- Comments are in green color and are not part of the input and/or output.

Task 1/7

Download the basher file at http://www.comp.nus.edu.sg/~adi-yoga/CS1010E/basher.sh

1. NOTE: *You do not need to understand what the following commands do*
2. Download the file using the command
   `wget "http://www.comp.nus.edu.sg/~adi-yoga/CS1010E/basher.sh"`
3. Run the file using the command `sh basher.sh`

Task 2/7

Go to sub-folder: `cd`

1. NOTE: *To go to a sub-folder, type* `cd sub_folder_name`
2. Go to sub-folder `basher` using the command `cd basher`

Task 3/7

Listing a folder content: `ls`

1. NOTE: *To list the current folder content, type* `ls`
2. NOTE: *To list a sub-folder content, type* `ls sub_folder_name`
3. NOTE: *To list a sub-folder content, type* `ls ..`
4. NOTE: *To list ALL information about the current folder content (including hidden files or folders), type* `ls -all`
5. List the current folder content using the command `ls`
6. List the content of the sub-folder `training` using the command `ls training`

Task 4/7

Reading a file content: `cat` and `less`

1. NOTE: *To read a file content, type* `cat file_name`
2. NOTE: *To read a file content of a* ***large file****, type* `less -e file_name`
   - Press `ENTER` to go to the next line
   - Pressing `ENTER` at the end of the file automatically close the file
   - To quit reading in the middle (maybe because the file is too long), press `q`
   - For this to work properly, make sure that the Bash terminal window is small (adjust the window to be small)
3. Go to `training` folder using the command `cd training`
4. Read the content of a small file `small.txt` using the command `cat small.txt`
5. Read the content of a large file `large.txt` using the command `less -e large.txt` and quit by pressing `ENTER` several times
6. Read the content of a large file `large.txt` using the command `less -e large.txt` and immediately quit by pressing `q`

Task 5/7

Go to parent folder: `cd ..`

1. NOTE: *To go to a sub-folder, type* `cd ..`
2. Go to parent folder using the command `cd ..`
3. Go to parent folder (*again*) using the command `cd ..`
4. You are now in you main folder

Task 6/7

Go to multiple folder(s)

1. Go to folder `basher/training` using the command `cd basher/training`
2. Go back to main folder using the command `cd ../..`

Task 7/7

Auto-complete name(s)

1. NOTE: *Name can be auto-completed, press* `TAB` *for auto-complete*
2. Go to folder `basher` using the command `cd basher`
3. Type `cd tr` **but do NOT press** `ENTER`
4. Auto-complete by pressing `TAB` then press `ENTER`
5. You are now in folder `basher/training`

VIM [#3]

Problem Description

"Vim is a highly configurable text editor built to make creating and changing any kind of text very efficient. It is included as "vi" with most UNIX systems and with Apple OS X." – http://www.vim.org

VIM may seem primitive in this day and age considering that it does not support graphical user interface (GUI) involving mouse. However, it has the advantage of being available to most machines with low overhead. You are to get yourself familiar with VIM through a simple tutorial.

Make sure that you have already done the necessary setup above.

Final Objective

Get yourself familiar with VIM through this tutorial.

Assumptions

The following assumptions are considered to be true, they limit the inputs to the following restrictions:

- ▷ You have setup your machine as in Task 1
- ▷ You are familiar with Bash commands as in Task 2

Tasks

The problem is split into 4 tasks. In the sample run, please note the following:

- ↩ is the *invisible* `[newline]` character.
- User input in blue and program output in purple color.
- Comments are in green color and are not part of the input and/or output.

Task 1/4

Create a new file: `hello_world.c`

1. Open `terminal`
2. Open VIM by typing `vim hello_world.c`, make sure that you do not already have a file called `hello_world.c` in your current directory
   - You should get a window similar to the one below:

3. You will start in the `--COMMAND--` mode

Task 2/4

Writing your first C program

1. To start typing, press `i`
   - This is called the `--INSERT--` mode
   - You can note that you are in `--INSERT--` mode by looking at the lower left corner of the screen as shown below (note how the word `--INSERT--` is not present in the picture above):

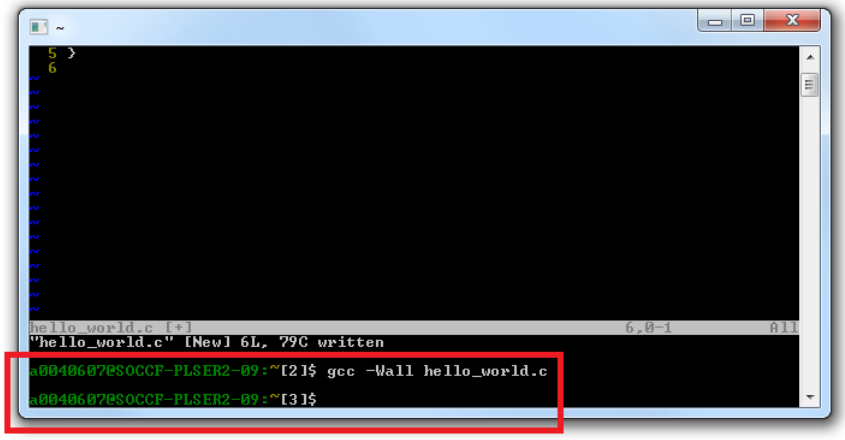
2. Type the following program:

3. Exit the `--INSERT--` mode by pressing `ESC` to go back to `--COMMAND--` mode
4. Save the program by typing the command `:w` in `--COMMAND--` mode
5. Exit VIM by typing the command `:q` in `--COMMAND--` mode

Task 3/4

Compile your first C program: `gcc -Wall hello_world.c`

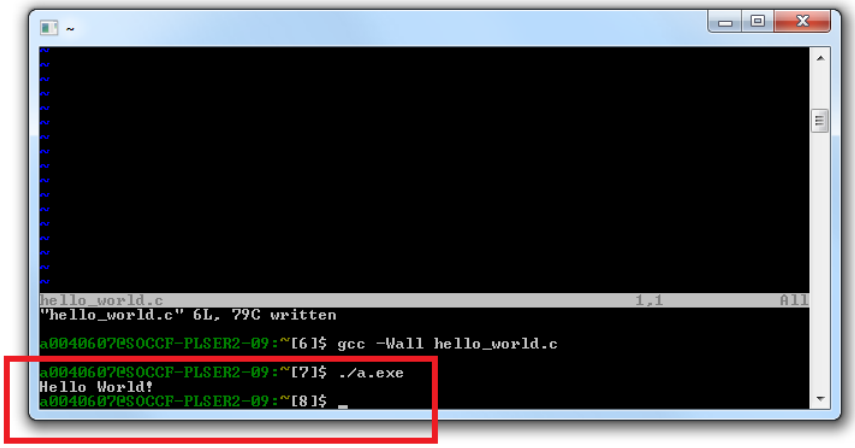
- You should get no message from the `terminal` as shown below:



Task 4/4

Running your first C program: `./a.out` (or `./a.exe` in Windows)

- You should get an output as shown below:



VIM Debugging [#4]

Problem Description

"Debugging is the process of finding and resolving of defects that prevent correct operation of computer software or a system." – Wikipedia

Writing a program that works in the first try is **hard**. Most of the time, we will find *bugs* in our program. Since debugging is necessary, we will introduce to you several aspects of debugging.

You **MUST** be familiar with the previous task before proceeding with the following task. Many of the commands are *assumed* to have been learned and will not be given.

To make this easier to work with, open two instances of `terminal`. One will be used for writing the program and the other will be used for compiling and running the program. We will call the `terminal` for writing as `TERM A` and the `terminal` for compiling and running the program as `TERM B`.

Final Objective

Get started with debugging a program using this tutorial.

Assumptions

The following assumptions are considered to be true, they limit the inputs to the following restrictions:

- ▷ You have setup your machine as in Task 1
- ▷ You are familiar with Bash commands as in Task 2
- ▷ You are familiar with VIM commands as in Task 3

Tasks

The problem is split into 6 tasks. In the sample run, please note the following:

- ↩ is the *invisible* `[newline]` character.
- User input in blue and program output in purple color.
- Comments are in green color and are not part of the input and/or output.

Task 1/6

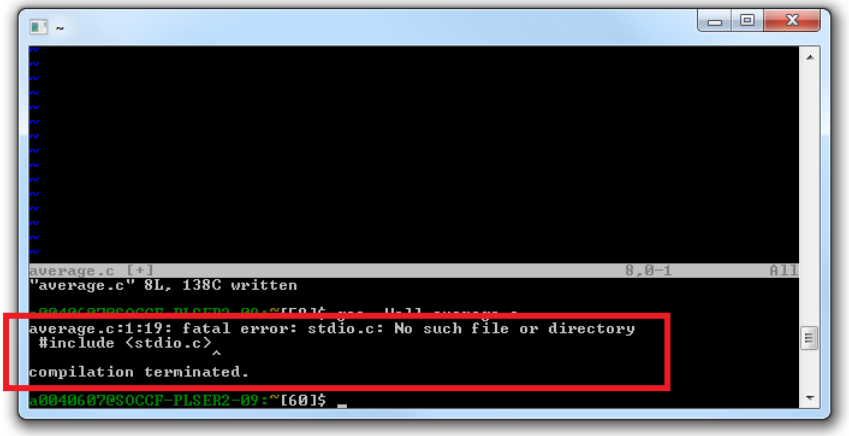
In `TERM A`:

1. Create a new file using VIM called `average.c`
2. Write *exactly* the following program:

3. Save the file *without exiting* using `:w` in `--COMMAND--` mode

In `TERM B`:

1. Compile the program and you should get the following compilation error:



2. The compilation error have the following information:
   - `average.c:1:19` ↦ The error is located in file `average.c` at line 1 and character number 19
   - `#include <stdio.c>` ↦ The error involves the given code
   - `^` ↦ In conjunction with the above error message, the error may involve the following character pointed to by `^`

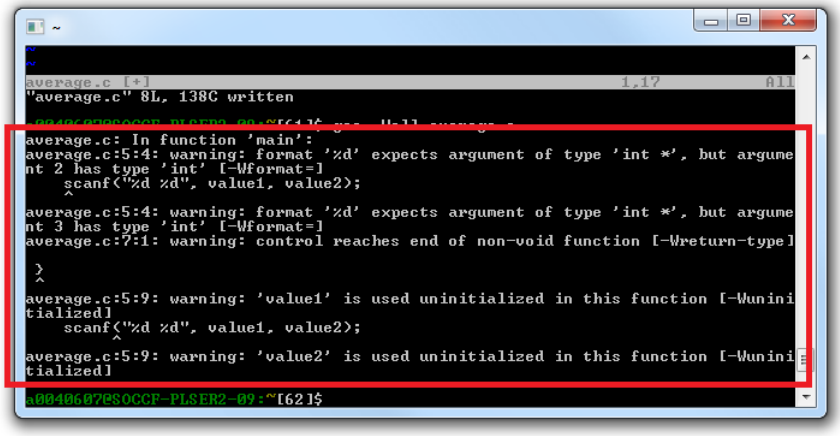
Task 2/6

In `TERM A`:

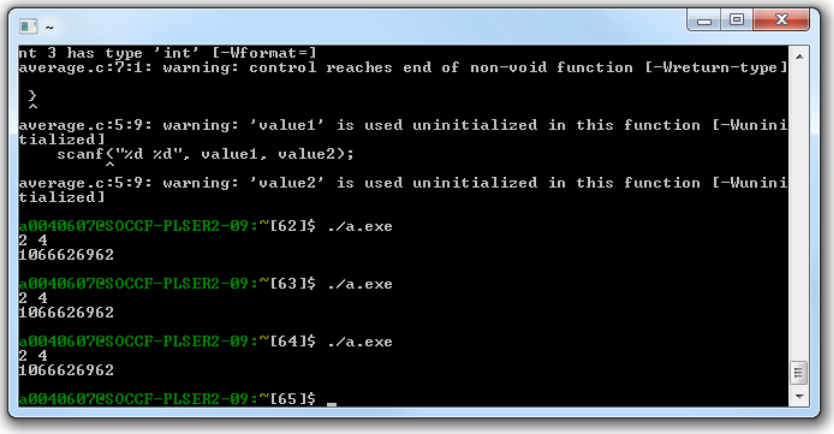
1. Modify the program to correct the error by changing `stdio.c` to `stdio.h`
2. The program is intended to have the following effect:
   - It stores the first input into variable `value1`
   - It stores the second input into variable `value2`
   - It prints the average of the two values *up to 2 decimal places* without ending newline

In `TERM B`:

1. Compile the program and you should get the following compilation *warning*:



2. Compilation warning means that the program compilation is successful but the given warnings may (or may not) cause problems
3. Run the program and give it the following input: `2 4`
   - The solution should be `3.00`
   - The result is other values as shown in my run (your own run may produce different values):



   - This is called *logical error*, an error that is not detected at compilation time but at runtime

Task 3/6

In `TERM A`:

1. Modify the program to correct the logical error by having the following changes:
    - At line 5: change `value1` to `&value1`
    - At line 5: change `value2` to `&value2`
    - NOTE: *These two changes correspond to the first two warnings and it shows that warnings are often useful to eliminate*

In `TERM B`:

1. Compile the program and you should still get one warning but this warning is okay to ignore
2. Run the program and give it the following input: `2 4`
    - The solution should be `3.00`
    - The result shows: `3`
    - This shows another logical error but this error is not reflected anywhere in the warning

Task 4/6

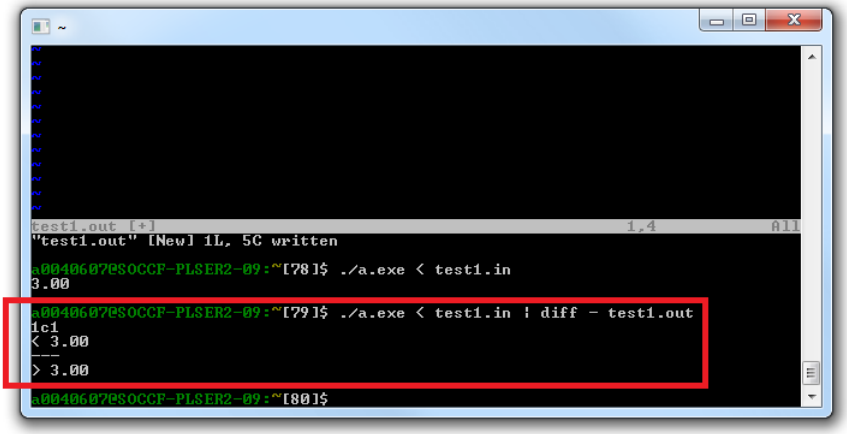
In `TERM A`:

1. To understand the error, you have to know the following concepts which you may/may not have learned depending on the speed of the lecture:
   - `printf("%d\n", ...)` prints an **integer** and not **real** number
   - `printf("%.2f\n", ...)` prints **real** number in 2 decimal places
2. Modify the program to correct the logical error by making several changes as shown below:

3. Since we have been testing for some time, and inputting the test values can be tedious, we will create several test cases:
   - Download the file: `http://www.comp.nus.edu.sg/~adi-yoga/CS1010E/th0_mac.zip` for Mac and `http://www.comp.nus.edu.sg/~adi-yoga/CS1010E/th0_win.zip` for Windows
   - Unpack and place the files in the same folder as `average.c`

In `TERM B`:

1. Compile the program and you should still get one warning but this warning is okay to ignore
2. Run your program using the following command `./a.out < test1.in` (or `./a.exe < test1.in`)
   - The command `<` indicates an *input redirection* where the file `test1.in` is treated as input to the program
   - The result seems correct!
   - NOTE: *You can replace* `test1.in` *with* `test2.in` *or other test numbers*
3. Run *and test* your program using the following command `./a.out < test1.in | diff - test1.out` (or `./a.exe < test1.in | diff - test1.out`)
   - The command `|` indicates a *pipeline* where the result of execution is given to command `diff` to compare **character-by-character** with `test1.out`
   - You should get the following message:



   - The message means there is a difference between your output and the file `test1.out`
   - Although there don't seem to be a difference, it exists in a *not-so-printable* character
   - The difference is in the `[newline]` character `"\n"`

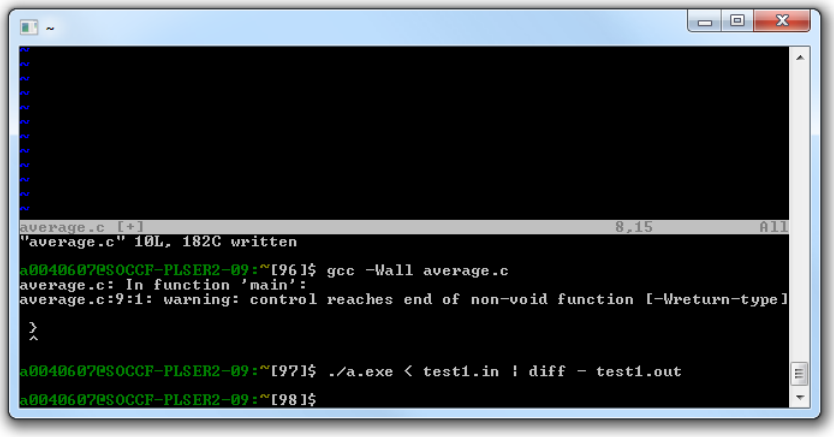
Task 5/6

In `TERM A`:

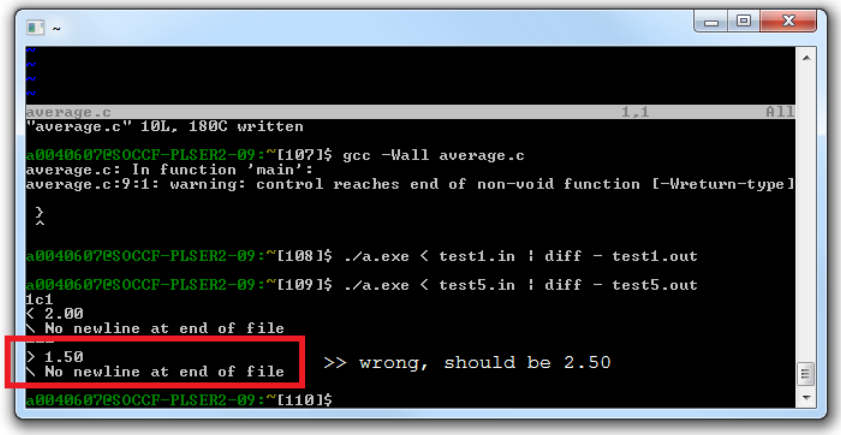
1. Modify the program by changing, at line 8, the code `"%.2f\n"` to `"%.2f"`

In `TERM B`:

1. Compile the program and you should still get one warning but this warning is okay to ignore
2. Run *and test* your program using the following command `./a.out < test1.in | diff - test1.out` (or `./a.exe < test1.in | diff - test1.out`)
   - You should NOT get any message as shown below:



     - No message means your output matches the expected output prefectly
     - *No news is good news!*
   - Try to test your program on `test5.in` with `test5.out` and you will get the following message below:



Task 6/6

In `TERM A`:

1. Modify the program by changing, at line 7, the code `"(value1 + value2)/2"` to `"(value1 + value2)/2.0`

In `TERM B`:

1. Compile the program and you should still get one warning but this warning is okay to ignore
2. Run *and test* your program using the following command `./a.out < test1.in | diff - test1.out` (or `./a.exe < test1.in | diff - test1.out`)
   - You should not get any more message for all test cases
   - Your program is, in fact, as intended

CodeCrunch [#5]

Problem Description

CodeCrunch (https://codecrunch.comp.nus.edu.sg/) is NUS-made automated code tester. Its setup is done by having a set of test inputs (similar to `test1.in`) with the corresponding test outputs (similar to `test1.out`). In fact, it can be thought of as running `gcc -Wall file.c` followed by `./a.out < test1.in | diff - test1.out` (or `./a.exe < test1.in | diff - test1.out`) for all the given test cases where `file.c` is the expected name of your C program file.

In this introduction to CodeCrunch, you should experiment on two things:

1. How to submit to CodeCrunch, and
2. How to interpret questions involving:
   - Problem Description
   - Task
   - Example
   - Input Specification
   - Output Specification
   - Assumption
   - Restrictions (*if any*)
   - Sample Run

Final Objective

Submit `average.c` to CodeCrunch. Read the specification below and learn how to map them to description in Question 4.

Assumptions

The following assumptions are considered to be true, they limit the inputs to the following restrictions:

▷ $0 \leq$ `num1` $\leq 2^{30}$ (*the first number*)

▷ $0 \leq$ `num2` $\leq 2^{30}$ (*the second number*)

Tasks

The problem is split into 5 tasks. In the sample run, please note the following:

- ↩ is the *invisible* `[newline]` character.
- User input in blue and program output in purple color.
- Comments are in green color and are not part of the input and/or output.

Task 1/5

Write a program that reads two `integer` numbers and print the avergae of the **two (2)** numbers up to **two (2)** decimal places.

Sample Run:

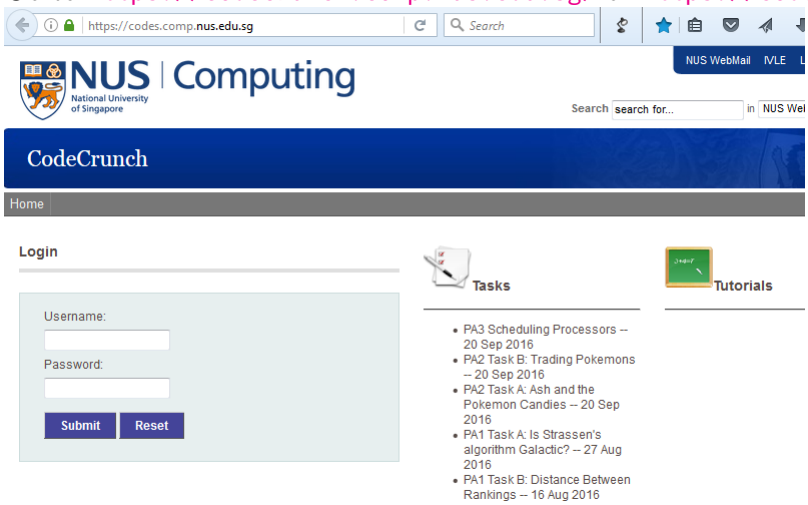
Inputs:	Outputs:
`2 4`	`3.00`

Save your program in the file named `average1.c`. Submit your program to CodeCrunch.

Task 2/5

Login to CodeCrunch

1. Go to `https://codecrunch.comp.nus.edu.sg/` or `https://codes.comp.nus.edu.sg/`

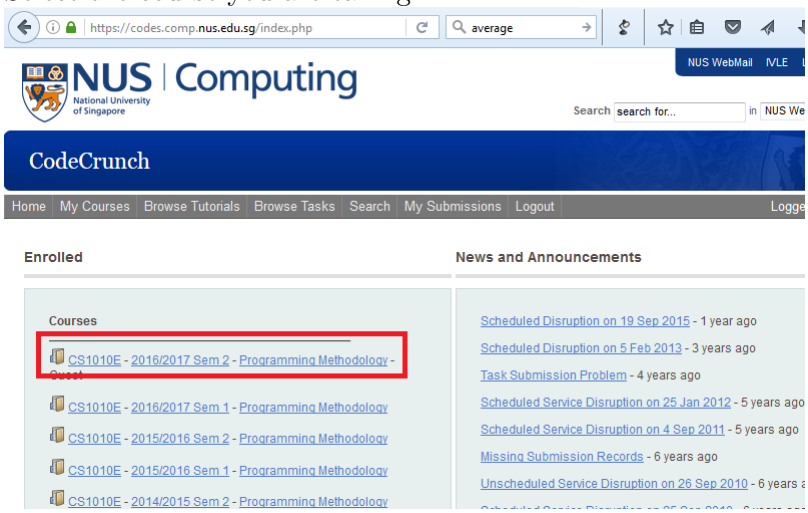


2. Fill in your nusnet ID and password

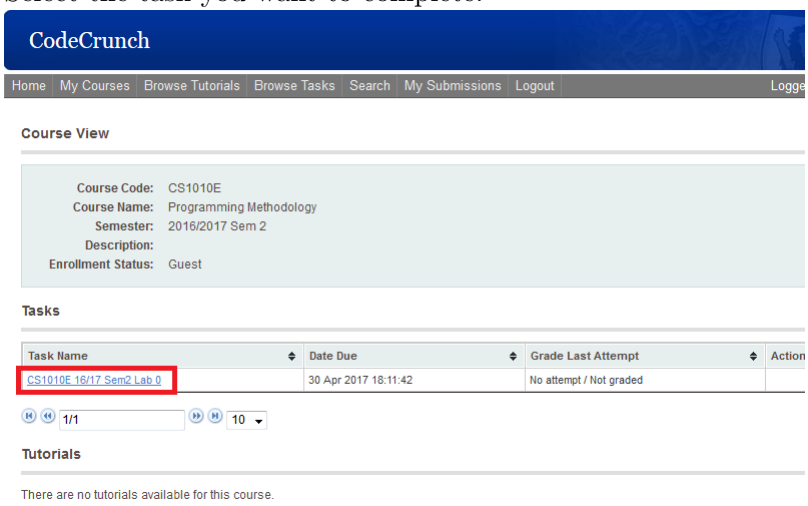
Task 3/5

Navigate to the corresponding task

1. Select the course you are taking:



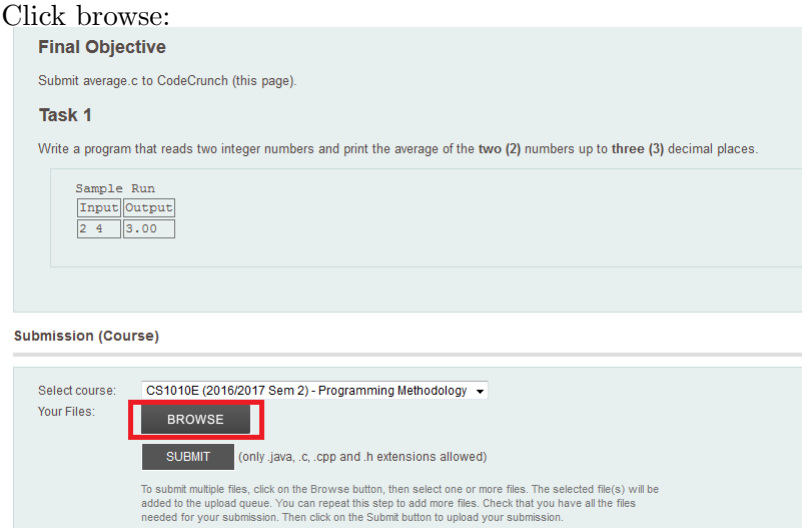
2. Select the task you want to complete:



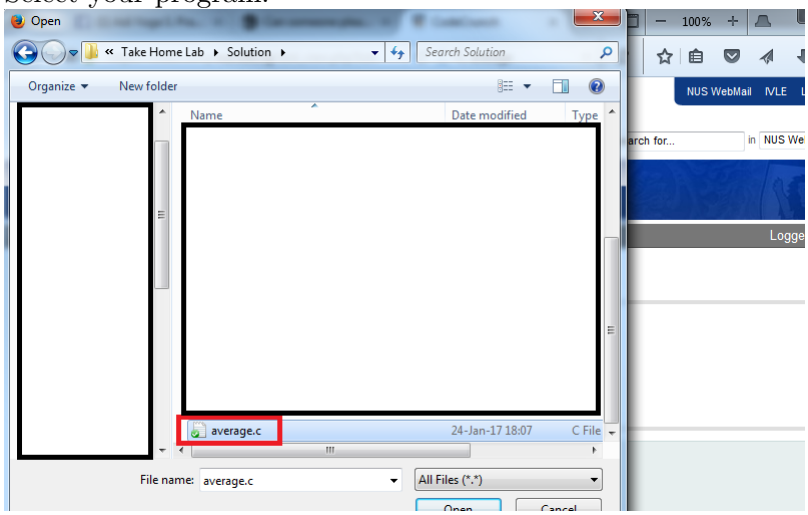
Task 4/5

Submit your program

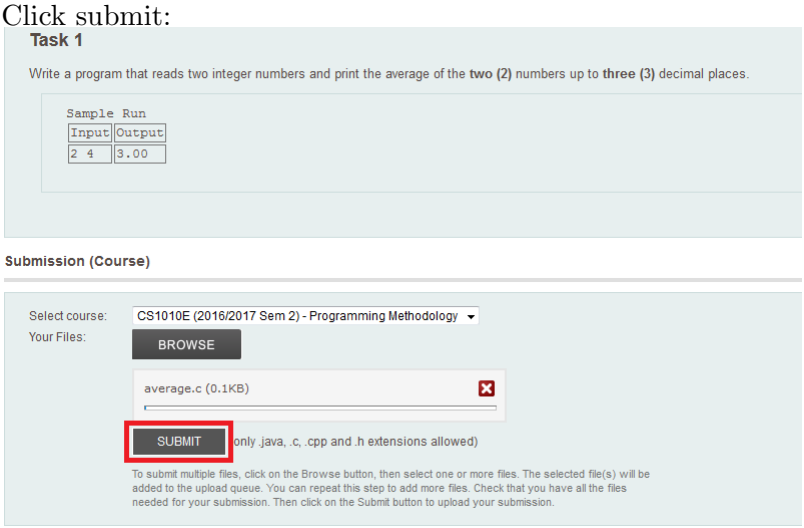
1. Click browse:



2. Select your program:



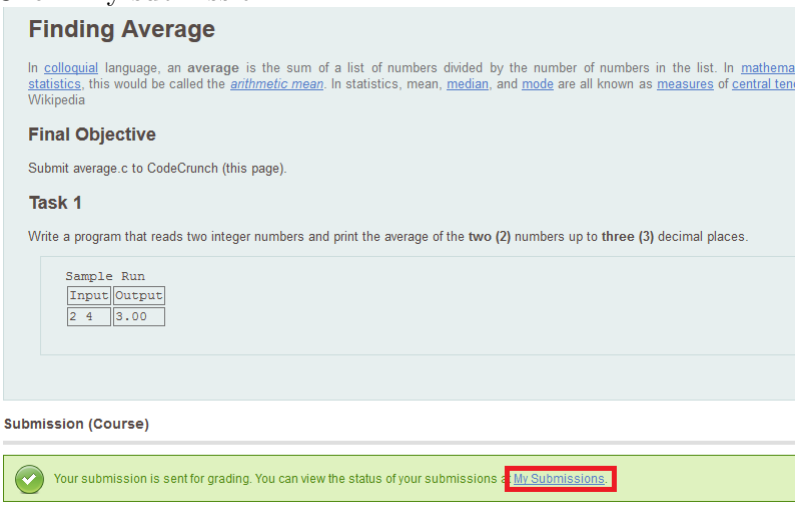
3. Click submit:



Task 5/5

Check your result

1. Click "my submission" link:



2. Your grade is shown on the right-hand side: