Assignment # 2-A

PART A

Please open buggy.c file and looking inside, you will see that the code has a bug. As in the lecture, there are at least 2 basic methods to debug the code. 1) Use printf() function revealing the unseen values of certain variable to be visible, and 2) Use debugger tools. Thus, in this assignment, you are required to debug your code using debugger tool offered in the vscode and the C/C++ extension by Microsoft.

The objective of buggy.c program is to print out 2 vertical #-brick blocks separated by empty space (line) between the vertical blocks such as follows:

```
./buggy (or .\buggy.exe for Window's user)
#
#

#
#
```

As can be seen, when we execute the code, the supposed output, meaning the intended output should be as above where each vertical block consists of 2-#-bricks and in between the vertical blocks, has empty space. However, due to some bug (logical error), we although get 2 vertical block with empty space in between, the number of #-bricks in every vertical block is not 2 but 3-#-bricks such as follow:

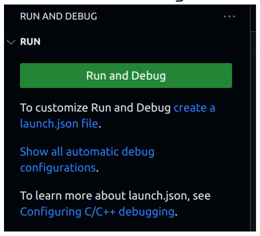
```
./buggy
#
#
#

#
#
#
```

Thus, your job is debug the code using debugger tool provided by vscode and C/C++ extension.

Your task is to record the debugging process using print screen of every steps taken when we click the the button 'step over'. (Watch video lectures or the slide for the whole debugging process if not clear). The general procedures are as the followings:

1) Compile buggy.c source code once.
2) Set the breakpoint accordingly.
3) Click run and debug icon on the most-left-thin panel shown here:
4) Then, a new debug panel appears with big-button "Run and Debug" shown below:

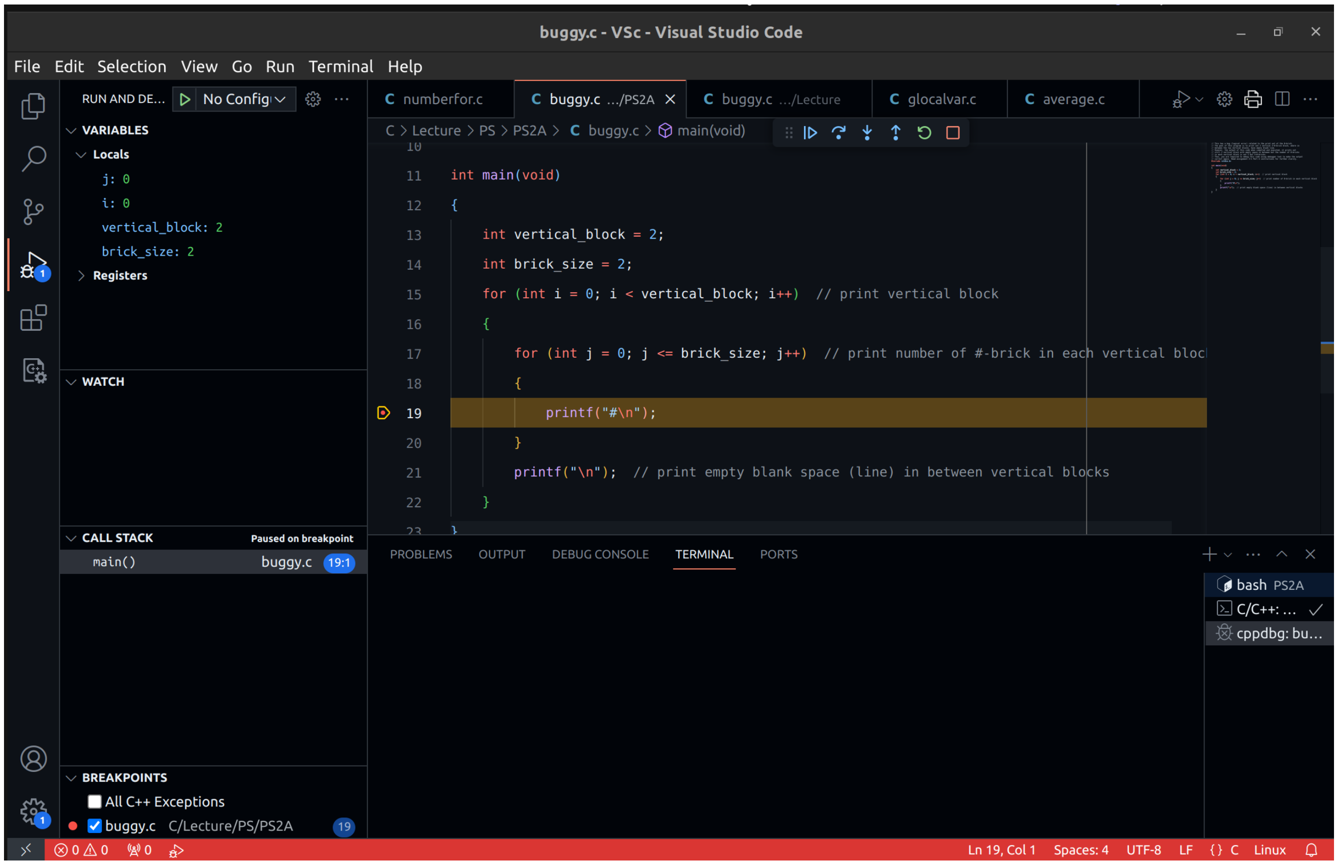


To customize Run and Debug create a launch.json file.

Show all automatic debug configurations.

To learn more about launch.json, see Configuring C/C++ debugging.

5) Click that "Run and Debug" button to start the debugging process.
6) Then a new debugging shell appears and your terminal window will be refreshed and your debugging panel will show variables that are used in buggy.c such as shown below:



VARIABLES
Locals

```
j: 0
i: 0
vertical_block: 2
brick_size: 2
```

Registers

WATCH

CALL STACK — Paused on breakpoint

```
main()    buggy.c 19:1
```

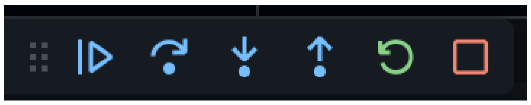
BREAKPOINTS

- All C++ Exceptions
- buggy.c C/Lecture/PS/PS2A 19

```c
int main(void)
{
    int vertical_block = 2;
    int brick_size = 2;
    for (int i = 0; i < vertical_block; i++)  // print vertical block
    {
        for (int j = 0; j <= brick_size; j++)  // print number of #-brick in each vertical bloc
        {
            printf("#\n");
        }
        printf("\n");  // print empty blank space (line) in between vertical blocks
    }
}
```

```
⊗ 0 ⚠ 0   0        Ln 19, Col 1   Spaces: 4   UTF-8   LF   {} C   Linux
```

7) Be aware of the debugging menu where it contains several icons-button such as shown below.



8) To start debugging, click the 'step-over' button, which is shown as
9) Every time you click that 'step-over' button, you need to print screen the process and observe:
    a) changes of how and which line of your code (highlighted by yellow color block surrounding your line of code and also, yellow icon next to the numbers representing lines of code) is being executed.
    b) changes at your terminal window whether something or # symbol is printed out.
    c) changes to the value of variable i and j.
10) Click 'step-over' button until you finish debugging your code where in each click, print screen your screen and keep observing the three items mentioned in item 9) above.
11) Once finish debugging, click 'stop' button to end debugging process:

Once done printing screen of every 'step-over' debugging steps, place in order those images in your words or text document.

Then, in every of your printed screen images necessarily, comment below them about what's happening to your code relating to the three items mentioned in step 9 above. In one those images, comment on where is the source of logical error and what needs to be done to correct the error and thus, correcting the output of this code.

Once done, we need to do the followings:

1) Convert that words or text containing those printed-screen images witj your comments into pdf file.
2) Modify buggy.c code accordingly removing the bug or logical error. Test or compile/execute your modified whether now you have the correct output shown as such below:

```
./buggy
#
#

#
#
```

To end, submit those two files by uploading to your assignment section in MS team.

Now, make us proud!

HINT:

Set breakpoint at line of code that prints the #-brick.

PART B

In PART B of this assignment, you are required to code .c program according to unique number (generated by c program) obtained from your lecturer.

Furthermore, when you write your own code, observe the 3 criteria of correctness – getting the right output, design – efficient design with less repetition of codes and styles – observing the locations of {} as well as indentation and etc.

Finally, comment your codes necessarily to explain its functionality in your code.

Unique number	Assignment requirement(s)

1	• Create an narray_1.c file (_1 in the file means assignment based on unique number 1)
	• Declare an integer array variable of size 5 called scores.
	• Get user input using for loop statement to initialize every elements of your array with appropriate integer values like the following when compiled code is executed:
	
	`./narray_1 (or for window's user: ./narray_1.exe`
	`scores 1: XX`
	`scores 2: XX`
	`scores 3: XX`
	`scores 4: XX`
	`scores 5: XX`
	
	where XX is just integer number of your preferences to initialize the value to every element in your integer array, for example, scores[0], scores[1], scores[2], scores[3] and scores[4].
	
	• Then, using while loop statement, print out the elements and their values accordingly. For example, on the screen after the elements are initialized, the array is printed out such as follows:
	
	`./narray_1 (or for window's user: ./narray_1.exe`
	`scores 1: XX`
	`scores 2: XX`
	`scores 3: XX`
	`scores 4: XX`
	`scores 5: XX`
	
	`scores[0] = XX`
	`scores[1] = XX`
	`scores[2] = XX`
	`scores[3] = XX`
	`scores[4] = XX`
	
	• Finally, find the average of these 5 integers and print it out onto the screen with 3 significant digits after decimal point, which on the screen after every elements and their values are printed out, we will see the following:

```
./narray_1 (or for window's user: ./narray_1.exe
scores 1: XX
scores 2: XX
scores 3: XX
scores 4: XX
scores 5: XX

scores[0] = XX
scores[1] = XX
scores[2] = XX
scores[3] = XX
scores[4] = XX

Average = XX.XXX  // Note the average is non-integer value
```

- Finally, submit your narray_1.c file together with image file containing the output on terminal windows when the compiled compiled machine code is executed.

2

- Create an narray_2.c file (_2 in the file means assignment based on unique number 2)
- Declare a integer type array variable of size 5, called temps_C. This array represent the temperatures in degree Celsius.
- Get user input using for loop statement to initialize every elements of your array with appropriate integer values like the following when compiled code is executed:

```
./narray_2 (or for window's user: ./narray_2.exe
temperature 1: XX
temperature 2: XX
temperature 3: XX
temperature 4: XX
temperature 5: XX
```

where XX.X is just integer number of your preferences to initialize the value to every element in your integer array, for example, temps_C[0], temps_C[1], temps_C[2], temps_C[3] and temps_C[4].

- Then, using while loop statement, print out the values in deg Celsius of every elements and also, their converted values to deg Fahrenheit, which will be float type number set to one significant digit after decimal point accordingly. For example, on the screen after the elements are initialized, we will see the following outputs when printing out the value of every element in degree Celsius and their converted values in degree Fahrenheit:

```
./narray_2 (or for window's user: ./narray_2.exe
temperature 1: XX
temperature 2: XX
temperature 3: XX
temperature 4: XX
temperature 5: XX

temps_C[0] = XX deg C -> XX.X deg F
temps_C[1] = XX deg C -> XX.X deg F
temps_C[2] = XX deg C -> XX.X deg F
temps_C[3] = XX deg C -> XX.X deg F
temps_C[4] = XX deg C -> XX.X deg F
```

- Finally, submit your narray_2.c file together with image file containing the outputs on terminal window when the compiled output machine code is executed.

3

- Create an narray_3.c file (_3 in the file means assignment based on unique number 3)
- Declare a int type array variable of size 4, called temps_F. This array represent the temperatures in degree Fahrenheit.
- Get user input using for loop statement to initialize every elements of your array with appropriate integer values like the following when compiled code is executed:

```
./narray_3 (or for window's user: ./narray_3.exe
temperature 1: XX
temperature 2: XX
temperature 3: XX
temperature 4: XX
```

where XX is just float number of your preferences to initialize the value to every element in your integer array, for example, temps_F[0], temps_F[1], temps_F[2] and temps_F[3].

- Then, using while loop statement, print out the values in deg Fahrenheit of every element and also, their converted values to deg Celsius, which will be float type number with one significant digit after decimal point accordingly. For example, on the screen after the elements are initialized, we will see the following outputs when printing out the value of every element in degree Fahrenheit and their converted values in degree Celsius:

```
./narray_3 (or for window's user: ./narray_3.exe
temperature 1: XX
temperature 2: XX
temperature 3: XX
temperature 4: XX

temps_F[0] = XX deg F -> XX.X deg C
temps_F[1] = XX deg F -> XX.X deg C
temps_F[2] = XX deg F -> XX.X deg C
temps_F[3] = XX deg F -> XX.X deg C
```

- Finally, submit your narray_3.c file together with image file containing the outputs on terminal window when the compiled output machine code is executed.

4	• Create an narray_4.c file (_4 in the file means assignment based on unique number 4)
	• Declare an integer type array variable of size 5, called intnums. This array represents the 5 integer numbers.
	• Get user input using for loop statement to initialize every elements of your array with appropriate integer values like the following when compiled code is executed:
	
	`./narray_4 (or for window's user: ./narray_4.exe`
	`integer number 1: XX`
	`integer number 2: XX`
	`integer number 3: XX`
	`integer number 4: XX`
	`integer number 5: XX`
	
	where XX is just integer number of your preferences when initializing the value to every element in your integer array, for example, intnums[0], intnums[1], intnums[2], intnums[3] and intnums[4].
	
	• Then, using while loop statement, print out the elements and their respective integer values and also, their converted values, which required to be float type with 1 significant digit after decimal point when those numbers are being squared respectively. For example, on the screen after the elements are initialized, we will see the following outputs when printing out the value of every element in integer numbers and their converted values to being squared or power of 2:
	
	`./narray_4 (or for window's user: ./narray_4.exe`
	`integer number 1: XX`
	`integer number 2: XX`
	`integer number 3: XX`
	`integer number 4: XX`
	`integer number 5: XX`
	
	`intnums[0] = XX -> squared value = XX.X`
	`intnums[1] = XX -> squared value = XX.X`
	`intnums[2] = XX -> squared value = XX.X`
	`intnums[3] = XX -> squared value = XX.X`
	`intnums[4] = XX -> squared value = XX.X`
	
	• Finally, submit your narray_4.c file together with image file containing the outputs on terminal window when the compiled output machine code is executed.

5	• Create an narray_5.c file (_5 in the file means assignment based on unique number 5) • Declare an integer type array variable of size 4, called intnums. This array represents the 4 integer numbers. • Get user input using for loop statement to initialize every elements of your array with appropriate integer values like the following when compiled code is executed:

	./narray_5 (or for window's user: ./narray_5.exe integer number 1: XX integer number 2: XX integer number 3: XX integer number 4: XX where XX is just integer number of your preferences when initializing the value to every element in your integer array, for example, intnums[0], intnums[1], intnums[2], and intnums[3]. • Then, using while loop statement, print out the elements and their respective integer values and also, their remainder values when those numbers are being divided by 3 respectively. For example, on the screen after the elements are initialized, we will see the following outputs when printing out the value of every element in integer numbers and their remainder values to being divided by 3: ./narray_5 (or for window's user: ./narray_5.exe integer number 1: XX integer number 2: XX integer number 3: XX integer number 4: XX intnums[0] = XX -> remainder when divided by 3 = XX intnums[1] = XX -> remainder when divided by 3 = XX intnums[2] = XX -> remainder when divided by 3 = XX intnums[3] = XX -> remainder when divided by 3 = XX • Finally, submit your narray_5.c file together with image file containing the outputs on terminal window when the compiled output machine code is executed.
6	• Create an narray_6.c file (_6 in the file means assignment based on unique number 6) • Declare a double type array variable of size 5, called dnums. This array represents the 5 double type numbers. • Get user input using for loop statement to initialize every elements of your array with appropriate double values with at 2 significant digits like the following when compiled code is executed: ./narray_6 (or for window's user: ./narray_6.exe double number 1: XX.XX double number 2: XX.XX double number 3: XX.XX double number 4: XX.XX double number 5: XX.XX where XX is just integer number of your preferences when initializing the value to every element in your integer array, for example, dnums[0], dnums[1], dnums[2], dnums[3] and dnums[4].

- Then, using while loop statement, print out the elements and their respective double type values and also, their converted values with 12 significant digits after decimal point when those numbers are being cubed, i.e. power of 3 respectively. For example, on the screen after the elements are initialized, we will see the following outputs when printing out the value of every element in integer numbers and their converted values to being cubed or power of 3:

```
./narray_6 (or for window's user: ./narray_6.exe
double number 1: XX.XX
double number 2: XX.XX
double number 3: XX.XX
double number 4: XX.XX
double number 5: XX.XX

dnums[0] = XX.XX -> cube value = XX.XXXXXXXXXXXX
dnums[1] = XX.XX -> cube value = XX.XXXXXXXXXXXX
dnums[2] = XX.XX -> cube value = XX.XXXXXXXXXXXX
dnums[3] = XX.XX -> cube value = XX.XXXXXXXXXXXX
dnums[4] = XX.XX -> cube value = XX.XXXXXXXXXXXX
```

- Finally, submit your narray_6.c file together with image file containing the outputs on terminal window when the compiled output machine code is executed.

7

- Create an narray_7.c file (_7 in the file means assignment based on unique number 7)
- Declare a float type array variable of size 5, called masses. This array of 5 float numbers represent a mass value in kg.
- Get user input using for loop statement to initialize every elements of your array with appropriate float values with at 2 significant digits like the following when compiled code is executed:

```
./narray_7 (or for window's user: ./narray_7.exe
mass 1: XX.XX
mass 2: XX.XX
mass 3: XX.XX
mass 4: XX.XX
mass 5: XX.XX
```

where XX is just integer number of your preferences when initializing the value to every element in your float array, for example, fnums[0], fnums[1], fnums[2], fnums[3] and fnums[4].

- Then, using while loop statement, print out the elements and their respective float type values and also, their converted values with 5 significant digits after decimal point when those numbers are being converted weight, i.e. multiplied with 9.81 respectively. For example, on the screen after the elements are initialized, we will see the following outputs when printing out the value of every element in integer numbers and their

	converted values to being cubed or power of 3:
	
	`./narray_7 (or for window's user: ./narray_7.exe`
	`mass 1: XX.XX`
	`mass 2: XX.XX`
	`mass 3: XX.XX`
	`mass 4: XX.XX`
	`mass 5: XX.XX`
	
	`masses[0] = XX.XX kg -> weight = XX.XXXXX N`
	`masses[1] = XX.XX kg -> weight = XX.XXXXX N`
	`masses[2] = XX.XX kg -> weight = XX.XXXXX N`
	`masses[3] = XX.XX kg -> weight = XX.XXXXX N`
	`masses[4] = XX.XX kg -> weight = XX.XXXXX N`
	
	• Finally, submit your narray_7.c file together with image file containing the outputs on terminal window when the compiled output machine code is executed.

8	• Create an narray_8.c file (_8 in the file means assignment based on unique number 8)
	• Declare a float type array variable of size 4, called lengths. This array of 4 float numbers represent a length value in unit meter (m).
	• Get user input using for loop statement to initialize every elements of your array with appropriate float values with at 2 significant digits like the following when compiled code is executed:
	
	`./narray_8 (or for window's user: ./narray_8.exe`
	`length 1: XX.XX`
	`length 2: XX.XX`
	`length 3: XX.XX`
	`length 4: XX.XX`
	
	where XX.XX is just float number of your preferences when initializing the value to every element in your integer array, for example, lengths[0], lengths[1], lengths[2] and lengths[3].
	
	• Then, using while loop statement, print out the elements and their respective float type values and also, their converted values with 5 significant digits after decimal point when those numbers are being converted to feet, i.e. multiplied with 3.28084 respectively. For example, on the screen after the elements are initialized, we will see the following outputs when printing out the value of every element in integer numbers and their converted values to being cubed or power of 3:
	
	`./narray_8 (or for window's user: ./narray_8.exe`
	`length 1: XX.XX`
	`length 2: XX.XX`
	`length 3: XX.XX`
	`length 4: XX.XX`
	
	`lengths[0] = XX.XX m -> US length = XX.XXXXX ft`
	`lengths[1] = XX.XX m -> US length = XX.XXXXX ft`

```
lengths[2] = XX.XX m -> US length = XX.XXXXX ft
lengths[3] = XX.XX m -> US length = XX.XXXXX ft
lengths[4] = XX.XX m -> US length = XX.XXXXX ft
```

- Finally, submit your narray_8.c file together with image file containing the outputs on terminal window when the compiled output machine code is executed.

9

- Create an narray_9.c file (_9 in the file means assignment based on unique number 9)
- Declare an integer type array variable of size 5, called forces. This array of 5 integer numbers represent a force magnitude value in N.
- Get user input using for loop statement to initialize every elements of your array with appropriate integer values like the following when compiled code is executed:

```
./narray_9 (or for window's user: ./narray_9.exe
force 1: XX
force 2: XX
force 3: XX
force 4: XX
force 5: XX
```

where XX is just integer number of your preferences when initializing the value to every element in your float array, for example, in forces[0], forces[1], forces[2], forces[3] and forces[4].

- Then, using while loop statement, print out the elements and their respective float type values and also, their converted values with 2 significant digits after decimal point when those numbers are being converted to stress value , i.e. dividing it with area respectively. Here, we shall assume the area equal 0.25 m^2. For example, on the screen after the elements are initialized, we will see the following outputs when printing out the value of every element in integer numbers and their converted values to being cubed or power of 3:

```
./narray_9 (or for window's user: ./narray_9.exe
force 1: XX
force 2: XX
force 3: XX
force 4: XX
force 5: XX

forces[0] = XX N -> stress = XX.XX N/m2
forces[1] = XX N -> stress = XX.XX N/m2
forces[2] = XX N -> stress = XX.XX N/m2
forces[3] = XX N -> stress = XX.XX N/m2
forces[4] = XX N -> stress = XX.XX N/m2
```

- Finally, submit your narray_9.c file together with image file containing the outputs on terminal window when the compiled output machine code is executed.

Now, make us proud!!!

HINT:

1. Declaring array having integer type of size 2: int arr[2];

2. Initializing the first element of integer array arr with value of 20:
   arr[0] = 20;
   Above code means that the value 20 is being assigned to the first element of integer array. The first element is represented by index 0. That means the position of element inside of array always can be represented by index number, i.e position number – 1. Further example, 3rd position or 3rd element of an array will be represented by arr[2].

3. for loop:
```
   for (int i = 0; i < 4; i++)
   {
         // Statements;
   }
```
   where int i = 0: i is an integer type counter variable, which is initialized to initial value of 0,
   i < 4: the condition for looping, i.e. as long as the value of i is less than 4, the statements inside the for loop codeblock will be executed until the condition is violated, i.e. the value of i either is equal to 4 or exceed 4. In this case, the statements are repeated or iterated 4 times due to the fact that the range of i value is i = 0, 1, 2, 3
   i++: incremental value by 1 where i++ -> i = i + 1.

4. while loop:
```
   int i = 0;  // counter variable having int data type initialized to 0
   while (i < 4)  // (1 < 4) is the condition
   {
         // Statements;
         i++; // increment by 1: i = i + 1
   }
```

5. Typecast – changing the type of one value to another type temporarily. For example:
```
   // Declare integer type variable x and initialized with value 4
   int x = 4;
   // print value of x to the screen as decimal number using typecast (float)
   printf("x = %f\n", (float)x);  //
```
   As shown in example above, the x variable which has integer type, is temporarily changed to float type by typecast method in order to print the value of x onto the console screen as decimal floating point number.
   When typecasting like shown in the example above, the desired converted type is typed inside parenthesis and the parenthesis placed just before the variable that will be typecast-ed.

6. Printing multiple values of variables and converted values due to use of formula together with format specifiers and typecast in printf function:

   Let's assume the output is expected to be:
```
   arr[0] = 18 inches -> in feet = 1.50 feet
```

where
- value 0 in square bracket of array variable is obtained from i variable,
- integer value 18 is obtained from array variable, for example, from arr[i] when i = 0. In another word, that arr[0] has a value of 18.
- value 1.5 is obtained from formula converting inches to feet, for example: feet = inches / 12, which in this case is achieved by dividing arr[i] when i = 0 with 12. Since the value of arr[i] is an integer type, it needs to be typecast-ed into float type temporarily so that result of the formula will be floating point value. Thus, the formula used will be:

```
      (float)arr[i]/12  where i will take the value of i variable
```

Then the printf statement will be like:

```
printf("arr[%i] = %i inches -> in feet = %.2f feet.\n", i, arr[i], (float)arr[i]/12 );
```