Department of

Computer Science and Engineering

Title: Debugging and Tracing

Computational Thinking and Problem Solving

CSE 100

Green University of Bangladesh

Objectives:

1. To identify and fix errors in the program
2. To improve the efficiency and performance of the program
3. To understand the behavior of the program
4. To improve the quality of the program

Problem Analysis:

Debugging: Debugging involves the process of identifying and resolving errors or bugs in a program. This is typically done by using a debugger, which is a tool that allows programmers to step through the code line by line, inspect the values of variables, and identify the source of errors. Debugging can help programmers identify and fix issues such as logical errors, syntax errors, and runtime errors.

Tracing: Tracing involves following the flow of a program and analyzing its behavior. This is typically done by adding print statements or log messages to the code, which output information about the program's execution. Tracing can help programmers understand how the program works, identify areas of inefficiency, and optimize the program's performance.

Here are the steps to perform debugging and tracing in CodeBlocks for a C program:

Step 1: Open the CodeBlocks IDE and create a new C project. To create a project, click on the *File* pull-down menu, open *New* and then *Project*, as shown in Figure 1.

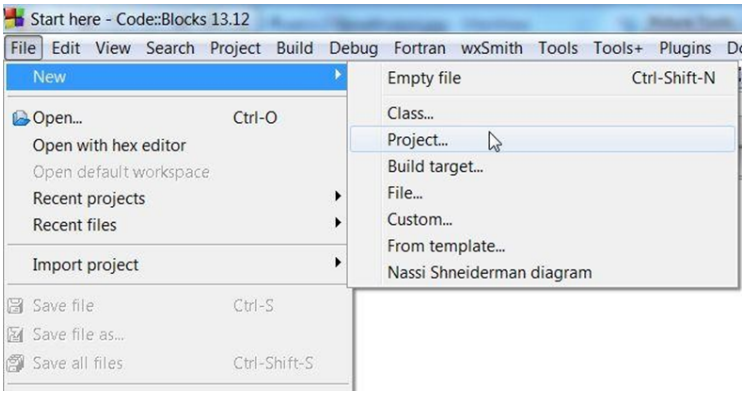

Figure 1: New project menu

This will bring up the New from template window, shown in Figure 2.

Step 2: Opening (clicking on) Console Application will then allow you to write a program with input and output on the console. The other applications are for developing more advanced types of applications.

Step 3: After selecting the Console application, click on the Go button to begin using the Console Application Wizard.

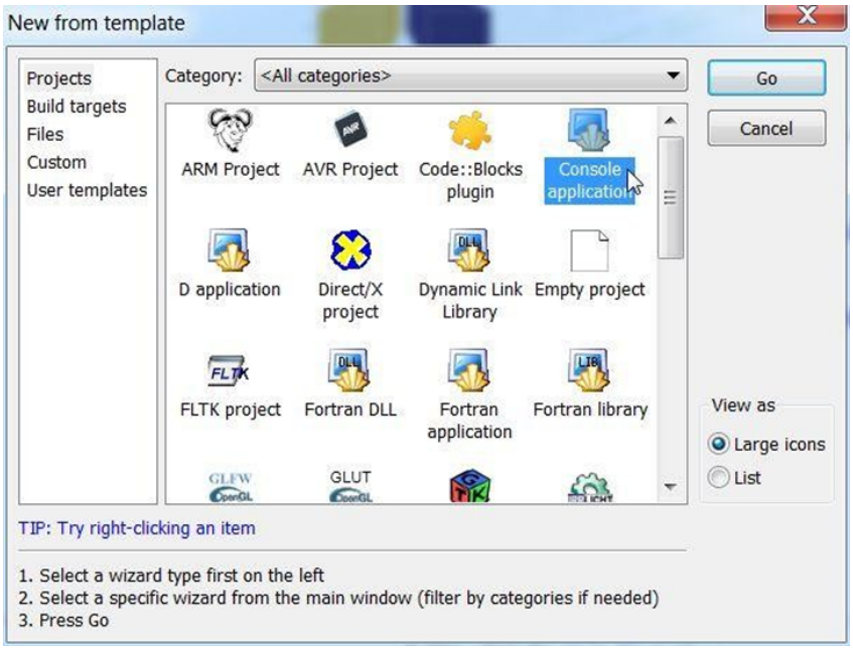

Figure 2: New from template - console application chosen

Step 4: The next window, shown in Figure 3 allows you to choose the language that you will use. Select the language as *C*, then press *Next*.

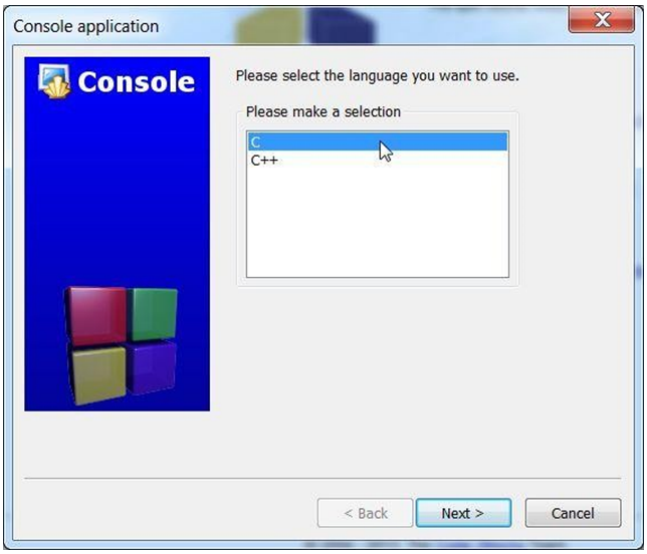

Figure 3: Selection of C dialect

Step 5: Start by filling in the *Project Title* – see Figure 4 . You will notice that the Project Filename automatically becomes the same name. If you wish, you can change the file name, but in general we will leave it as it is.

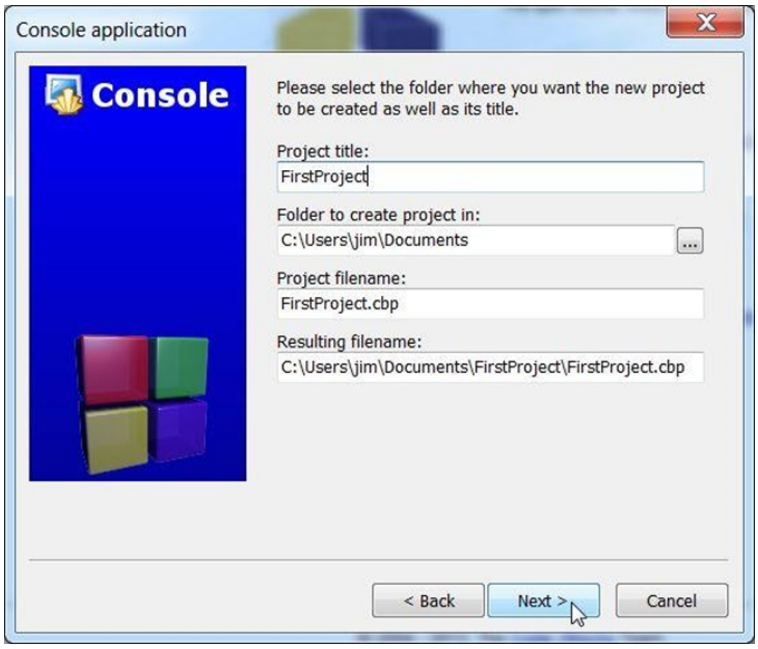

Figure 4: Filling the Project information

Step 6: The next window to pop up, also named *Console application* will allow you to specify the compiler and project configurations. This specifies where the Debug and Release compiled versions of your program will be placed. An example is shown in Figure 5.

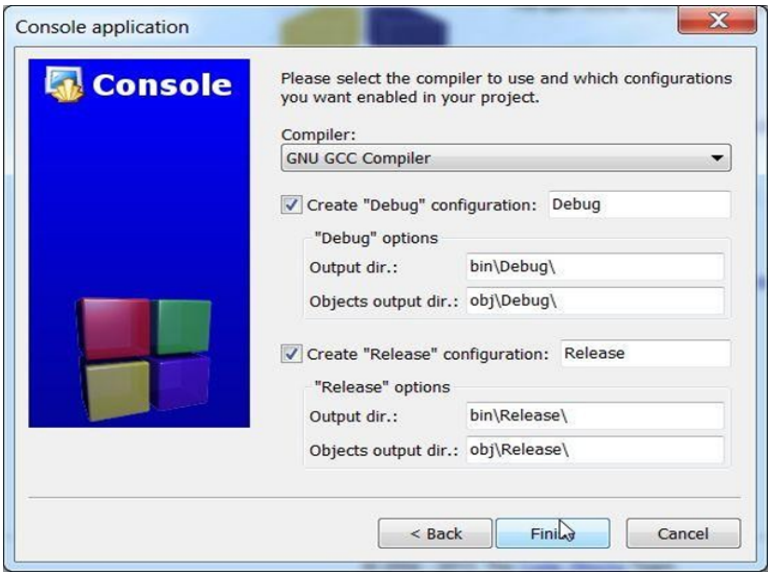

Figure 5: Configuring the debug and release output folders

Leave this setting as it is and press *Next*. The system will then return to the [First Program] window and you are ready to write your program.

Step 6: When your programs become more complicated, there will be a need to trace the program execution step by step or place breakpoints where you wish the program to pause. This is where a debugger is utilized. A debugger can pause your program and you can watch the values of the variables that you have defined.

Figure 6 shows a sample program that can be traced “line by line” while watching what happens as each line of code is executed.

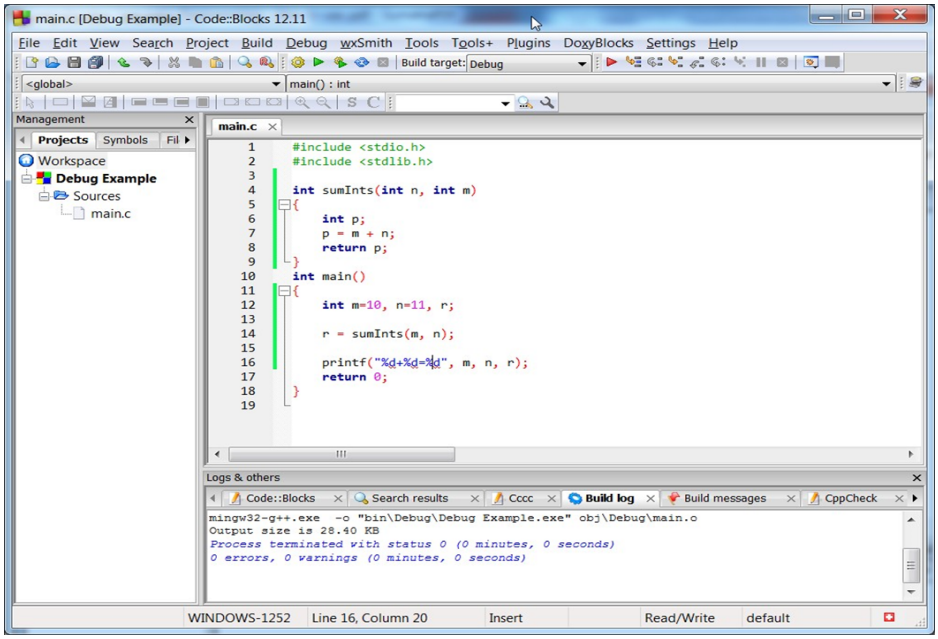

Figure 6: Sample program to debug

Step 7: First, it is necessary to set a place in the code to have the program pause. This is done by using the Debug pull-down menu and clicking on *Run to Cursor* (or use shortcut key F4). The cursor should be over the first line of code where you wish to start the tracing process. This starts the debugging process.

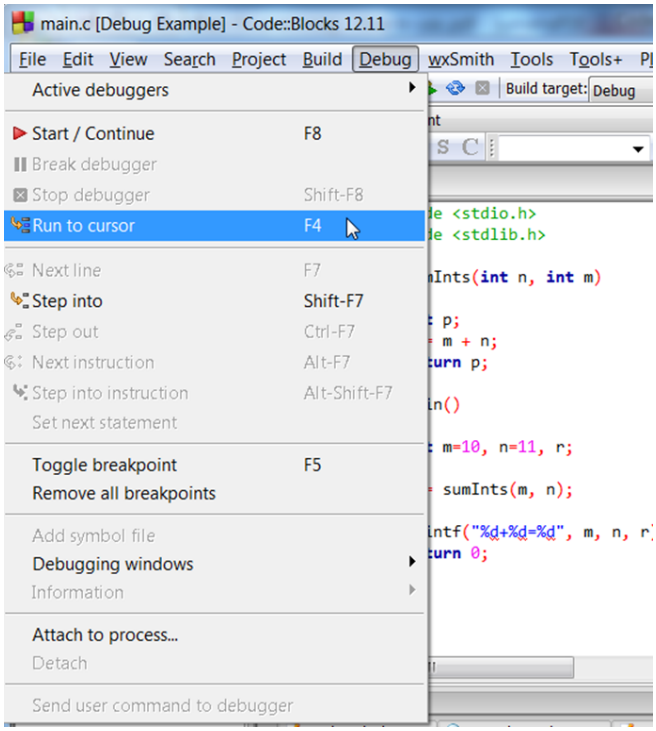

Figure 7: Start debugging at the point the cursor is in the editor window

The program will generate an empty output window. It is empty, since that program has yet to execute any line that displays something. To watch certain variables during the execution of the program, you should open the *Watches* window.

This will show you the variables in your code. This is accomplished by going to the *Debug* pull-down menu and clicking on *Debugging Windows* and then *Watches*. The result is shown in Figure 8.

These are the watches that the debugger is displaying. Notice that *m* and *n* have the correct values. Variable *r* has not been assigned its value on line 14 yet. The current value is a random value. Line 14 has a yellow marker on the left side. This indicates that the program has paused on that line, which is the breakpoint.

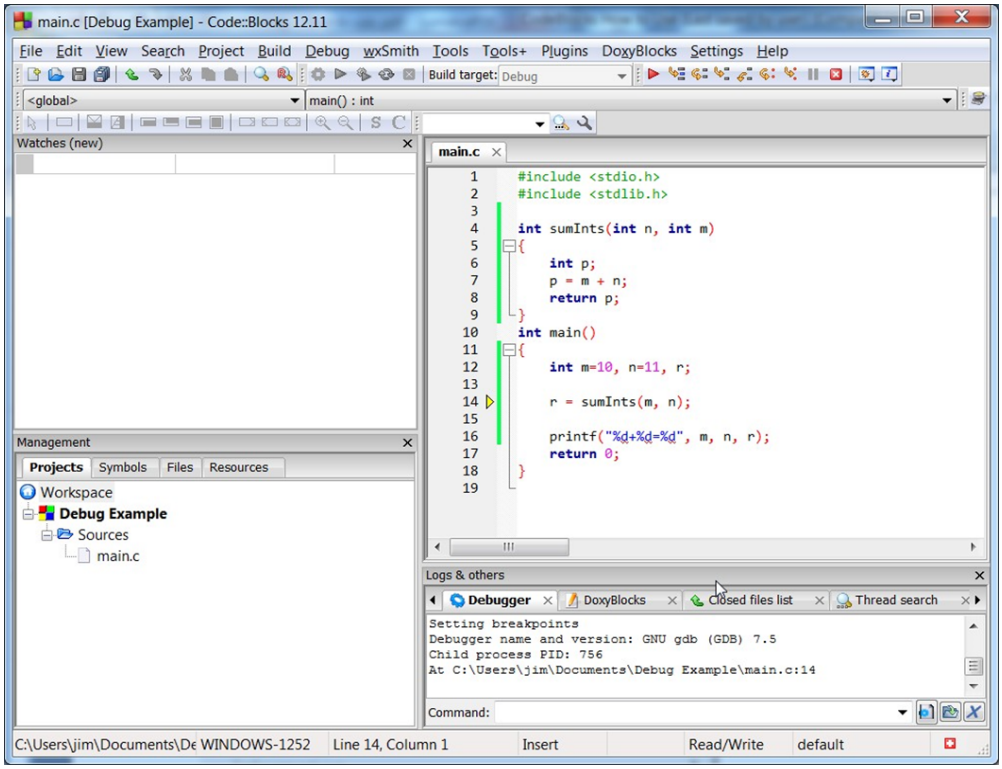

Figure 8: The debugged program is at line 14.

To determine how your program will function when calling functions such as:

z = addem(x,y);

Step info (*Shift-F7*) can be selected from the Debug pull-down menu. The next step is line 8. The local variables *r* from main have not yet been initialized, as shown in the Watches window – line 14 has not been executed yet, as shown in the Figure.

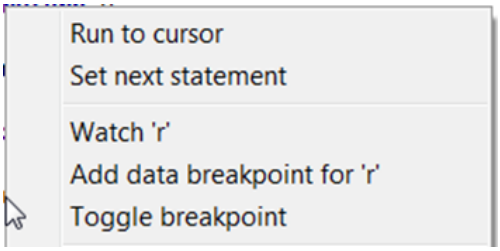

Figure 9: Right-click context menu in Debug mode.

To proceed to the next line of code, select *Next* line from the *Debug* menu. Pressing F7 is a useful keyboard shortcut and will become second nature as you become familiar with the system.

To step through the statements of function sumInt, select *Step Into*, now. Debugging will now enter sumInt. As you pass over line 7, the debug Watches reflects the change of local variable p.

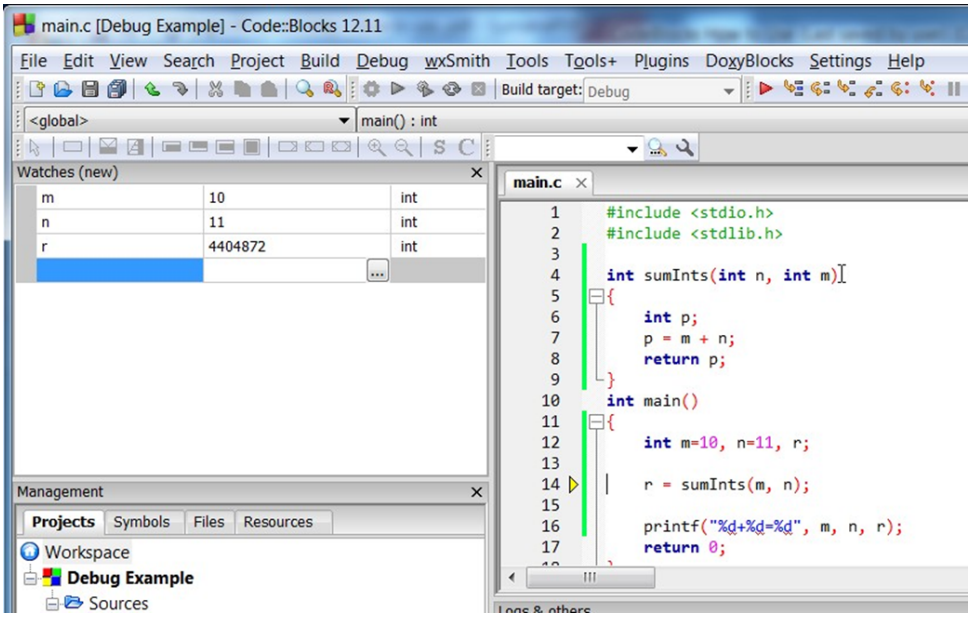

Figure 10: Program before invoking sumInts().

When you are done debugging, you can click on *Continue* from the *Debug* menu, and your program will run to completion.

This is better than selecting to ***Stop* debugger**. The reason it is better to *Continue*, is because the program comes to a natural end, rather than aborting. However if your program is stuck in a loop, or you are sure you can exit safely, you can select from the *Debug* menu ***Stop Debugger***.

You can further define places in your program to pause and allow you to inspect the code. This is done by setting breakpoints in your code. You can have zero or more breakpoints in your code. When the debugger encounters a breakpoint, the program pauses and the debugger will allow you to inspect your code. The breakpoint remains until you remove it. It can be toggled with F5 or by right-clicking a non-empty line it and select *Add breakpoint* from the pop-up menu which appears.

A breakpoint has been set at line 7. The red circle on the left indicates that there is a breakpoint in the code.

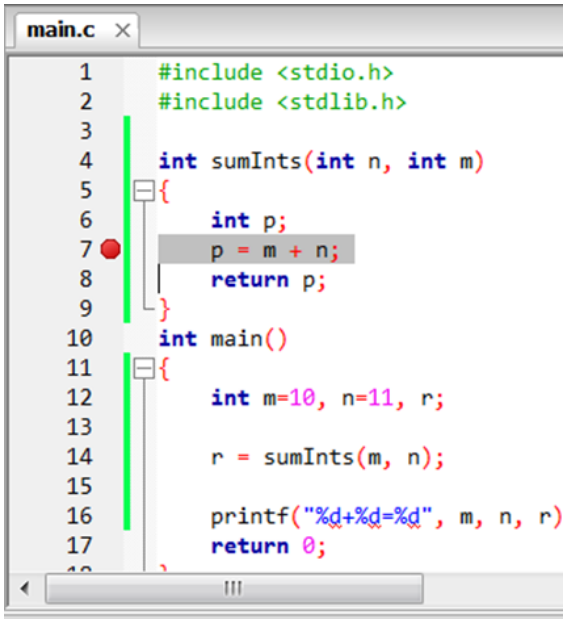

Figure 11: Breakpoint at line 7.

The program is started by selecting from the Debug pull-down menu, Start. This will run the program in the debugger until a breakpoint is encountered, at which point the program will pause.

When the program pauses at the breakpoint, a red circle with a yellow triangle mark will appear at the breakpoint.

You can set multiple breakpoints. The keyboard shortcut F5 allows you to toggle the breakpoint at any line.

This screen shows breakpoints on lines 7 and 16, but line 7 indicates that the code has executed to that point.

Selecting Continue from the Debugger menu will run the program till the next breakpoint. Now the program stops at line 16, because the program reached the second breakpoint . Press Ctrl-F7 to continue. Now the program runs till the end of the program, because there are no further breakpoints to encounter.

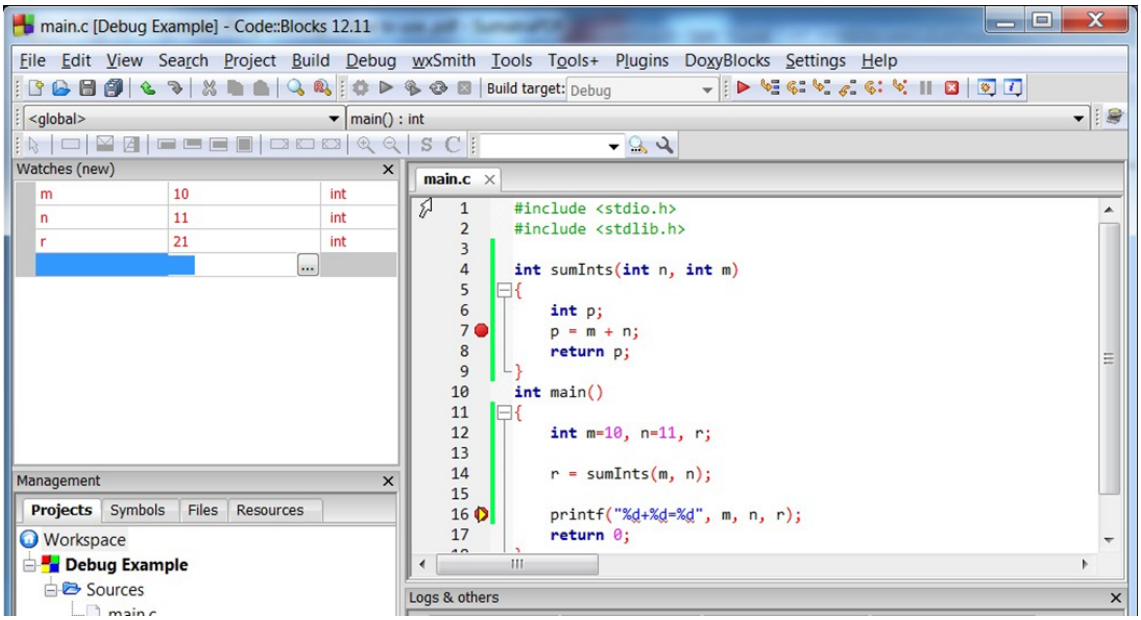

Figure 12: Breakpoint at line 16, reached.

Discussion and Conclusion

Based on the focused objectives, be familiar with debugging and tracing in a complicated program and able to fix errors in the program. The additional lab exercise made me more confident in the fulfillment of the objectives.

Lab Task (Please implement yourself and show the output to the instructor)

1. Write a C program that reads in a list of integers from the user and calculates the sum and average of the numbers. Use debugging and tracing techniques to identify and fix any errors in the program.
2. Write a C program that reads in an integer from the user and outputs whether the number is positive, negative, or zero. Use tracing techniques to output information about the program's execution and to analyze the flow of the program.

Lab exercise (submit as a report)

1. Write a C program that reads in a single integer from the user and outputs whether the number is even or odd. Use debugging and tracing techniques to identify and fix any errors in the program.
2. Write a C program that reads in two floating-point numbers from the user and calculates their average. Use debugging and tracing techniques to identify and fix any type errors in the program.

Policy

Copying from the internet, classmates, seniors, or from any other source is strongly prohibited. 100% marks will be deducted if any such copying is detected.