This handout is a tutorial on how to trace and debug programs using the VS Integrated Debugger.

Create a new project on your network drive. Download the source file `debugging.cpp` from the course website to your project folder and add it to your project.

Build and run this program. It does not give the correct result. For example, it says that the sum of the integers from 1 to 5 is 25, whereas the correct answer is 15. Also, it has an infinite loop. You cannot quit the program. (To terminate the program, use the close window button of the command prompt window.) So this program has (at least) two bugs in it. If you see them, do not fix them just yet.

**Starting the Debugger**

The Integrated Debugger helps us to debug programs by allowing us to trace program variables and expressions while the computer executes the program. To turn on the debugger, go to Debug, then Step Into (or press the F11 key). This will cause VS to change the lower windows to watch windows on the left and the call stack initially on the right. Debug icons will appear in the toolbar. The command prompt window will be under the VS window. You can access it by clicking on its icon in the bottom taskbar.

There should be a yellow arrow at the left side of the edit window indicating the current statement to be executed. Click on the Locals tab of the lower left window to show the currently declared variables and their values. Since we haven't actually started running the program, yet, there are none listed. The computer will display their values as we execute the program.

**Single-Stepping Through Code**

Now we can execute the program and watch the values of the variables. We would like to execute the program one line at a time. We can do this by using Debug, then Step Over (or pressing the F10 key or click the Debug toolbar icon with the arrow going around code). Each time you Step Over, one line of your program is executed.

Try stepping over one line now. Notice that the arrow on the left side of the program window moves to the next line. Every time you step over, a line of your program is executed and the arrow moves. In this first step, the declarations make `number`, `result`, and `answer` known to the computer, so they show up in the lower left window. Note that their values are random ones, since they are uninitialized.

Step over until the arrow is on the statement: `cin >> number`, then step over again. The system is waiting for input, so the command prompt window pops up. Enter a number for `number` just as when
you run your program normally. For the purposes of this tutorial, enter 5 in response. Notice that the variable window now shows \texttt{number} as having the value you typed in, namely 5.

Step over again. Executing this line caused the program to call the \texttt{Summation} function and assign the return value to \texttt{result} in one step. Notice that when the assignment statement is executed, that the new value of \texttt{result} is immediately displayed in the variable window. Unfortunately, it is the wrong answer, so we conclude that our function \texttt{Summation} is not working correctly.

\section*{Setting Breakpoints}

We don't need to execute the rest of our program right now, so terminate it. We also don't need to single step through the entire program. Instead, we will set a \textbf{breakpoint}, which tells the system to execute the program until it encounters this point. To set a breakpoint, right-click on the statement you want the program to stop at and choose Insert Breakpoint. For our problem that would be: \texttt{result = Summation (number)}. This puts a red dot next to the statement.

To run the program, go to Debug, then Start Debugging (or press the \texttt{F5} key). Enter 5 as input again. The program stops before executing the line with the breakpoint. Recall that last time when we stepped over, the computer executed both the call and the return in one step. But now we would like to see the execution of the function itself, so instead of stepping over, we can go to Debug, then Step Into (or press the \texttt{F11} or click the Debug toolbar icon that has the arrow going into code), which tells the computer to trace into the function call.

Step into now. Notice that the arrow is now at the first line of the function. Also, notice that \texttt{n} is now in the variable window and that it has the value passed in, namely, the value of \texttt{number}. The variables \texttt{number}, \texttt{result}, and \texttt{answer} are no longer present, since they are only known in the main program.

Now we want to trace the function execution step by step so we use Step Over again. Repeatedly step over and watch which lines of the function are executed. In particular, notice the value of \texttt{sum} after we execute the line \texttt{sum = sum + n}. From this, we see that we are not adding the correct value to \texttt{sum} since the first time through the loop, we want to add 1, the second time we want to add 2, and so forth. Inspection of the code reveals that we have inadvertently added \texttt{n} to \texttt{sum} each iteration, rather than \texttt{count}. At this point, we could stop the debugger, edit and recompile the program, and run it again. However, we know there is at least one other bug that causes the infinite loop, so we'll continue debugging.

A note about using Step Into: Generally, if the line is not a function call, Step Into and Step Over will have the same behavior. However, be aware that the input and output operators (\texttt{>>} and \texttt{<<}) are function calls, so you probably want to use Step Over on them. Otherwise, you'll get sent to the code for these calls, too. If this happens, just go to Debug, then Step Out (or type \texttt{Shift+F11} or click the Debug
toolbar icon with the arrow coming out of code). Step out will finish executing the current function and stop when it returns to the caller.

Since we know what's wrong with the function, we can get rid of the breakpoint. To do this, right-click on the red dot and choose Delete Breakpoint.

**Watching Expressions**

To figure out what's wrong with the loop, we can step through the code that asks the user for a response and determine that the program is reading the correct value from the keyboard for the variable `answer`, so the problem must be the loop condition. We can watch the loop condition expression to see what's wrong. To watch an expression, highlight the expression in the code, then right-click and choose Add Watch. Since our loop condition consists of two parts, it might be interesting to look at the values of each part separately: `answer != 'N'` and `answer != 'n'`, as well as the entire expression: `(answer != 'N') || (answer != 'n')`.

Now set a breakpoint at the line `while ((answer != 'N') || (answer != 'n'))`. Restart the debugger using Debug, then Restart (or typing `Ctrl-Shift-F5` or clicking the Debug left turn arrow icon). Type any integer for the number, then when it asks for an answer, type an N. If we look at the watch expressions, we see that `answer != 'N'` is false, `answer  != 'n'` is true, and `(answer != 'N') || (answer != 'n')` is true. To have the program run until the breakpoint again, do Debug, then Continue (or type F5). This time type an n for the answer. This time we see that `answer != 'N'` is true, `answer != 'n'` is false, and `(answer != 'N') || (answer != 'n')` is true. The logical operator is not correct, we want the condition to be true only if both parts are true, so we need `&&` rather than `||`.

When you are done watching an expression, you can delete it by selecting it and pressing the Delete key. If you want to see the variable values again, click on the Locals tab.

**Exiting the Debugger**

Now we have found both bugs that we know about, we could delete the breakpoint, but you don't have to. To stop the debugger, go to Debug, then Stop Debugging (or type `Shift+F5` or click the Debug stop icon on toolbar). You can fix the bugs and rebuild and run the program. If you use the normal Start Without Debugging, then the breakpoints are ignored.