**Using the Visual C++ Debugger**

*Acknowledgements:* This document is modified from a Web page named *CSE/ENGR 142 Debugger Tips* written by staff at the University of Washington's Computer Science department. We're very grateful!

Use the debugger on the program  **msvs\_exp.cpp**. You'll need to use Visual C++ to create and name a new project directory, then save this C++ file into that directory, and then add the file to the project using the menu sequence Project->Add to Project->Files.

1. [**What is a debugger?**](https://bb.fau.edu/bbcswebdav/pid-1247005-dt-content-rid-13961013_1/courses/COP3014L004_2014S_26249/Intro_to_MSVS_debugger.htm#debugger)
2. [**How to start the debugger**](https://bb.fau.edu/bbcswebdav/pid-1247005-dt-content-rid-13961013_1/courses/COP3014L004_2014S_26249/Intro_to_MSVS_debugger.htm#start)
3. [**How to stop the debugger**](https://bb.fau.edu/bbcswebdav/pid-1247005-dt-content-rid-13961013_1/courses/COP3014L004_2014S_26249/Intro_to_MSVS_debugger.htm#exit)
4. [**Breakpoints**](https://bb.fau.edu/bbcswebdav/pid-1247005-dt-content-rid-13961013_1/courses/COP3014L004_2014S_26249/Intro_to_MSVS_debugger.htm#breakpoints)
5. [**What a debugger session "looks" like**](https://bb.fau.edu/bbcswebdav/pid-1247005-dt-content-rid-13961013_1/courses/COP3014L004_2014S_26249/Intro_to_MSVS_debugger.htm#debugWindow)
6. [**Step Over/Into/Out**](https://bb.fau.edu/bbcswebdav/pid-1247005-dt-content-rid-13961013_1/courses/COP3014L004_2014S_26249/Intro_to_MSVS_debugger.htm#step)
7. [**Use the variable window**](https://bb.fau.edu/bbcswebdav/pid-1247005-dt-content-rid-13961013_1/courses/COP3014L004_2014S_26249/Intro_to_MSVS_debugger.htm#variablewindow)
8. [**Use the watch window**](https://bb.fau.edu/bbcswebdav/pid-1247005-dt-content-rid-13961013_1/courses/COP3014L004_2014S_26249/Intro_to_MSVS_debugger.htm#watchwindow)
9. [**Displaying expressions**](https://bb.fau.edu/bbcswebdav/pid-1247005-dt-content-rid-13961013_1/courses/COP3014L004_2014S_26249/Intro_to_MSVS_debugger.htm#expressions)
10. [**Contexts**](https://bb.fau.edu/bbcswebdav/pid-1247005-dt-content-rid-13961013_1/courses/COP3014L004_2014S_26249/Intro_to_MSVS_debugger.htm#contexts)
11. [**Displaying chars and strings**](https://bb.fau.edu/bbcswebdav/pid-1247005-dt-content-rid-13961013_1/courses/COP3014L004_2014S_26249/Intro_to_MSVS_debugger.htm#char)
12. [**Dereferencing pointers**](https://bb.fau.edu/bbcswebdav/pid-1247005-dt-content-rid-13961013_1/courses/COP3014L004_2014S_26249/Intro_to_MSVS_debugger.htm#pointers)
13. [**Displaying arrays and structures**](https://bb.fau.edu/bbcswebdav/pid-1247005-dt-content-rid-13961013_1/courses/COP3014L004_2014S_26249/Intro_to_MSVS_debugger.htm#arrays)
14. [**Possible problems getting the debugger to work**](https://bb.fau.edu/bbcswebdav/pid-1247005-dt-content-rid-13961013_1/courses/COP3014L004_2014S_26249/Intro_to_MSVS_debugger.htm#MSVC5.0)
15. **What is a****debugger?**A debugger is a tool which helps a programmer find errors ("bugs") in a program. You can make the program stop when it reaches a particular statement: this is called a "breakpoint." While at a **breakpoint**, you can examine the values of your variables and investigate the state of your program. Your C++ program is visible in a window at this time. Then you can continue executing from the breakpoint, perhaps to another breakpoint. Along the way you can set "**watches**" to view the values of variables you are interested in.
16. **How to start the debugger:** The Visual C++  (VC++) debugger is built-in to the Visual C++ development environment. **You can start the debugger by selecting "Debug/Start Debugging"**. Alternatively you can press "F5" to start debugging also.

Before starting the debugger, you probably should set a breakpoint (see below), or your program will simply run to completion as usual. When you started the debugger and it's paused at certain line, your whole VC++ window takes on a totally different look, as you'll see in a minute.

1. **How to stop the debugger:** During the debugging process, you can stop it by selecting "**Debug\Stop Debugging**". Alternatively you can press "Alt+F5" to stop debugging.
2. **Breakpoints:** If you want to stop the program execution at a particular line, use the mouse to left click on that line. When you see a red circle showing up next to that line, you have successfully put a breakpoint there.  See the figure below.
3. If you ever want to remove a breakpoint, put the mouse cursor on that line  and left click.

Some subtle things you may need to know: You **cannot** put a breakpoint on a line that **only** defines a variable or function; put it on the first line afterwards that assigns or "executes" something. When you create a breakpoint while the program is running in debug mode, it is not actually activated until the next time the program reaches that statement.

1. **What a debugger session "looks" like:** Your session should look similar to the image of the debugger session shown below.  You'll see a big window with the code of the current file (with breakpoints shown). Below that, you'll see the *variable window* on the left, and the *watch window* on the right. Right above the variable window you'll see a textbox labeled *Context* with the name of the current function, *main()*. Finally, notice there's a new set of icons up in the top-right window that look like this:  
   More on these later.
2. **Step Over/Into/Out:** One basic use of the debugger is for you to set one or more breakpoints, then when the program pauses at that line, make the program execution continue one line at a time while you watch what happens. We call this ***single-stepping***. You can tell which line is being executed because VC++ puts a yellow arrow next to the line. Debuggers usually provide several commands for this; in VC++ they are:
   * **Step Into:** executes the next line. If this is a function call, go to the next line **inside** the called function.
   * **Step Over:** executes the next line, **without** entering into a called function. Stay in the same function.
   * **Step Out:** Used when inside a function; this executes all the remaining lines in the function, and then stops at the end of the function.

When the program is paused (**at a breakpoint)**, you access these functions from the Debug menu at the top of your window.

Again, when single-stepping a program, and the next statement to be executed is a function call, "**Debug/ Step Over**" will execute the entire function, whereas "**Debug/Step Into**" will step you **into** the function. Use "**Step Into**" for your own not-quite-debugged functions, and "**Step over**" for system functions like printf, or debugged functions of your own. To run a function until its end and then return to the caller, click "**Debug/Step Out**."

If you have stepped as far as you want and now you want the entire program to run to its normal completion, you can click "**Terminate All**" and then click "**Debug/Start Debugging**"to finish execution. (Or, you can click the same button that started the debugger running to do this.) There is also a handy "Run to Cursor" command that will execute all lines between the current line and the line where you've clicked the mouse. As for all of these commands, you can see the function-key and button short-cuts for doing "Step Into","Step Over" and "Step Out" and the others by looking at the "Debug" menu. For example, F8 can be used for "Step Into".

**Really useful hint:** In C++, often functions you didn't write get called automatically. For example, in the example shown, the function call code pr\_message("Hello world!") actually calls a string class constructor to convert a string literal into a string object. If you use "Step Into" on this line, you'll unexpectedly find yourself in a function you didn't write and may not know anything about. **Solution:** Just execute "Step Out" to finish up that function and get back to code you've written.

1. **Use the variable window:** The variable window will display variable values local to a function. (If it's not visible, then select "View/Variables" will pop up the watch window.) As you single-step your program, you'll see the values change - very useful! There are three tabs in the lower left corner of the variable window. You can click on different tabs to display different information:
   * The Auto tab displays information about variables used in the current statement and the previous statement.
   * The Locals tab displays information about variables local to the current function.
   * The Threads tab will not be discussed
   * The Modules tab will not be discussed
2. **Use the watch window:** Watch window displays information about the variables that you are interested in. (If it's not visible, then select "View/Watch" will pop up the watch window.) You can enter variable names in the "**Name**" column of the watch window, the debugger fills in the **Type** and **Value** columns with the corresponding information as your program runs. You use the watch window to focus only on the variables you're interested in.  
   To remove a variable from the watch window, just use the mouse to high-light the variable's name in the "**Name**" column and hit backspace or delete to erase the name.
3. **Displaying expressions:** You can display more than just simple variables in the watch window. You can enter expressions there, and their values will be displayed. This can be useful, for example, in displaying the values of logical expressions controlling if statements and loops.
4. **Displaying chars and strings:** If you declare a variable of type char or char \*, Visual C++ will display the variable value as well as the charater or character string it points to in the "Value" column.
5. **Dereferencing pointers:** If you have a pointer variable in the watch window, there will be a small button next to the variable name. By clicking on the button, you can see the memory location it points to in the "Value" column.
6. **Displaying arrays and structures:** To display the contents of an array or struct, enter the variable name into the watch window as usual. You will see a small button appears next to the variable name. By clicking on the button, you can expand or contract your view of the variable. The button displays a plus sign (+) when the variable is displayed in contracted-form, and a minus sign when it is displayed in expanded form.
7. **Possible problems getting the debugger to work:** If you find the debugger just won't start for you, here are several things you can try. (But these trouble-shooting tips are from an older version of VC++.)
   * First, you need to be sure that the active configuration is set properly. Pull down the "**Build**" menu and select "**Configuration Manager**." Be sure that "**Active Solution Configuration"** is set to "**Debug**" and "**Active solution platform**" is set to Win32. Now, re-compile your program. You should now be able to use the debugger.
   * When you wish to debug, be sure that you either start the program with Build -> Debug -> Start Debugger, or by hitting F5. If you start the program by selecting Build -> Start Without Debugging any breakpoints will not be recognized and the debugger will not work.

**Laboratory Questions**

**Directions:**Copy the program "msvs\_exp.cpp" into Microsoft Visual Studios. Then follow the steps below and answer the following questions.

1. **Question 1:** Place a breakpoint at the line "int x = 1, y = 1;" and run the program using Debug, Start Debugging. The program stops at the first breakpoint. What are the values of x and y at this point in the program?
2. **Question 2:** Use the Step Over command once. What now are the values of x and y at this point in the program?
3. **Question 3:** In your own words, explain why the change in the values of x and y that occurred in Step 2 was not visible at the breakpoint in Step 1.
4. **Question 4:** The yellow arrow should be pointing to the line "pr\_message("Hello world!");". Use the Step Into command to enter this function. Explain what you see.
5. **Question 5:** Now use the Step Out command to return to "pr\_message("Hello world!");". What is the command you used for this action?
6. **Question 6:** Now in the main function, place a breakpoint on the line "x = y / x;". Keep an eye on the window as you continously use the Step Over command until you reach this breakpoint. What line in the program sets the denominator to cause the divide-by-zero error?
7. **Question 7:** Based on your answer in Question 6, correct the code to allow the program to run to completion.