**When to Use a Debugger**

Debugging is the name given to the process of removing bugs from computer programs. This can never be completed entirely, because "there is always one last bug." However, it is possible to remove all bugs except that last one. For this the use of a debugger can be very helpful. This is a program designed to allow you to step through the instructions in another program and to examine the state of that program at every step.

Debugging begins as soon as you wonder why your program is not producing the results you want it to. The first thing to do is to be sure that your C/C++ code compiles and links with no errors. If any error message occurs your code should be modified so that it does not occur.

The second thing to do is to examine your code and compare the intended results with what you actually get. It sometimes happens, however, that a thorough examination of the code does not reveal any bug. Then you may wish to insert some debugging code. This can consist of assert statements to make sure that variables have the values that you expect them to at certain points in your program (e.g.,  assert ( n > 0) ). If the assertions fail, the program stops with an error message.

Debugging code may also consist of statements to print out (using **printf** or **cout**) the values of certain variables at certain places in your code to give you information concerning what is going on when the program is running. If this still does not throw any light on the bug then it is time to fire up the debugger.

**What a Debugger Can Do**

* Step through the program one statement at a time, either "over" or "into" functions.
* Run the program up to a certain point (either to the cursor or to a "breakpoint") and then stop.
* Show the values (or contents) of variables at each point during the execution of the program.

**How to Use the Visual C++ Debugger**

This applies to Version 5.00 of the Microsoft Visual C++ Compiler.  
The debugger in earlier or later versions may differ in some respects.

To use the debugger in the Visual C++ programming environment you must have compiled and linked your program using the "Debug" configuration (not the "Release" configuration). The configuration is set using the menu options **Build | Set Active Configuration**. This is the default configuration, so you don't have to set it unless you previously changed the configuration to "Release".

Usually it is not helpful to step through the statements in a program beginning with the first statement in **main()**. It is better to have the program run until it gets to that part of your code where you think the bug resides. There are two ways to tell the debugger to run to this point. You can locate the cursor at that place in your code and press **Ctrl-F10** (or use **Build | Start Debug | Run to Cursor**). The program will be run by the debugger and if all goes well it will stop at the place in your code that you selected, and the debugger windows will be opened. Of course, it is possible that the program may terminate before your intended stopping place is reached. In which case, choose a place earlier in the program.

The second way to tell the debugger to run to the place in your code that you wish to inspect is to set a "breakpoint" there. To set a break point, position the cursor at the desired place and press **F9** (or use **Right-mouse-button | Insert/Remove Breakpoint**). If a breakpoint has been set then you can remove it by placing the cursor next to it and pressing**F9**. (Breakpoints can also be disabled by using **F9** or by **Right-mouse-button | Disable Breakpoint**.) Once you have set a breakpoint then press **F5** (or use **Build | Start Debug | Go**) to cause the debugger to run the program and halt at the breakpoint, whereupon the debugger windows are opened (if they are not already open).

The windows at the bottom left and bottom right of the screen hold variable names and values. Those in the left window include variables local to the function which is being executed. Those in the right window are user-specified variables. To specify a variable click on the left part of the first empty line then type the name of the variable. After pressing Enter the value of that variable (if it has a value) will be displayed. You can then view how the displayed variables change in value as execution of the program is continued.

Pressing **F1** brings up context-sensitive Help. For example, it tells us the difference between "Auto" variables and "Locals" variables:

The Auto tab displays information about variables used in the current statement and the previous statement.  
The Locals tab displays information about variables that are local to the current function.

To view the value of a variable without having to type in the name, position the cursor somewhere within the variable name and press **Shift-F9** (or **Debug | QuickWatch**).

You can continue program execution in several ways. The simplest is to press **F10** (or **Debug | Step Over**) to execute the next statement (which is indicated by the small yellow arrow in the left margin of the code window). Pressing **F10** repeatedly steps through successive program statements. Attend to the information given in the bottom windows. For example, if you are at a statement

if ( ( j = atoi(buffer) ) >= 2000 )

(which converts the string in buffer to an integer, places the value in variable j and compares this to 2000) then after pressing **F10** you will be informed as to the value returned by atoi().

If you are at a function call then **F10** causes that function to be executed and when the debugger returns control you are at the next statement. At a function call you can press **F11** (or **Debug | Step Into**) to enter that function (if the code for that function was compiled in "Debug" mode). You can then use **F10** to step through the statements in that function. When you are within a function you can press **Shift-F11** (or **Debug | Step Out**) to run through the rest of the function and return to the place where the function was called.

When the debugger starts it opens a window for the program's output, so you can switch between the debugger and this window to see what the program is sending to the screen.

In addition to **F10**, **F11** and **Shift-F11** you can use **Ctrl-F10** (or **Debug | Run to Cursor**) to run to the cursor position. You can also set further breakpoints and use **F5** (or **Debug | Go**) to run to each one in turn. You can have several breakpoints active, and you can disable or remove them at will by positioning the cursor at them and pressing **F9**.

If you want to see the definition of a variable place the cursor within the variable name and use **Right-mouse-button | Go to Definition**. It may happen that the debugger will tell you that it can't do this until it has created the Browser database, in which case tell it to do so.

The Browser is a useful tool which you can use to get the definition of any variable in your program and the places in your program where that variable is used (the "variable references"). You can invoke it by using **Tools | Source Browser**.

If your program takes command line arguments then during debugging you can set these using **Project | Settings | Debug | Program arguments**.

To restart the program from the beginning press **Ctrl-Shift-F5** (or **Debug | Restart**).

To stop the debugger and return to edit mode use **Shift-F5** (or **Debug | Stop Debugging**).

**Summary of Key Commands**

|  |  |  |
| --- | --- | --- |
| F5 |  | Start program execution, run until a breakpoint is reached. |
| Shift-F5 |  | Stop debugging and return to program edit mode. |
| Ctrl-Shift-F5 |  | Restart the program in debugging mode. |
| F9 |  | Insert, disable or remove a breakpoint. |
| Shift-F9 |  | Get the value of a variable. |
| F10 |  | Execute the current statement or function call. |
| Ctrl-F10 |  | Run the program to the point where the cursor is. |
| F11 |  | If the current statement is a function call, step into the function. |
| Shift-F11 |  | If executing a function, run to the end and return to the calling statement. |