This figure shows the Debug drop down menu before a program has started to execute. The following are useful items.

**Start** (F5) is used to actually start program execution, or used to resume execution if program had been temporarily halted by debugger.

**Start Without Debugging** (Ctrl+F5) will cause the program to execute completely with no halting for breakpoints or temporary stops. Program will stop at command window with prompt “Press any key to Continue”

**Step Into** (F11) will cause program to start execution and stop at first line of execution of current function, can then single step through rest of program or continue or stop.

**Step Over** (F10) will cause program to execute through a function, so function code executes, then stops at line of code after function call.

**Run to Cursor** (Ctrl+F10), only short cut key available, but similar to setting breakpoint at the same location. Will cause program to run through program, but stop where the cursor is placed in the source code in editor. Can then single step through or observe value of variables or whatever want to do.

This figure shows the Debug drop down menu when a program is currently running, but has been stopped by the debugger. Most are same as described above, with the others meaning:

**Continue** (F5) used to continue program execution.

**Stop Debugging** (Shift+F5), will stop the debugger.

**Step Out** (Shift+F11), will cause debugger to continue with the execution of current function and stop at line immediately after the function call.

**New Breakpoint…** will allow the user to place a halt or stop somewhere in the code. Once program reaches this location the execution will stop. Can then single step through and watch variables.

**Clear All Breakpoints**, will remove all breakpoints that have been set by user.

**To single step** through program or execute one line at a time press the F11 key, (some systems it is the F8 key)
To place a breakpoint in code, in the editor, point and then click the mouse pointer in the grey column next to the left hand side of the code. If done correctly a red circle will appear there. This indicates a breakpoint at this location. So when program runs it will stop execution at this spot.

You can place as many breakpoints in code as you want to this way. To remove any one simply click again on the red circle or click **Clear All Breakpoints** to remove them all.
This figure shows the screen when the debugger is running, which will allow the user to execute one line of code at a time and watch variables as they change.

The red circle shows where the program ran to then stopped and the yellow arrow shows what line of code will be executed next when the user presses F11, (sometimes F8).

The bottom of the screen is very important. It has or should have three small windows, layered one on the top of the other, with tabs on the bottom to choose which one is visible.

One displays the auto or automatic variables, another displays the local variables and the third labeled “Watch 1”, allows you to enter any variable or variable you wish to watch change as the program executes one line at a time. To add items to the watch window simply click on the “Watch 1” tab then click on the top line, type in the variable name and push enter.

Now as the program executes one line at a time the variables values can be watch as they change.

You can add or remove items from the “Watch 1” window at any time. And once you find your logic error you can then stop the debugger, fix the code and then re-run the program.

The debugger can be used to watch any kind of variable, whether it is built in data type or user created. Some data type variables, such as those for arrays, will have a “+” next to the name, meaning that if you click once on the “+”, then all array locations will then be visible in the watch window.

```
#include <iostream>
using namespace std;

void FindLargest(int, int, int &);

void main()
{
    int M = 127, N = 34, Large;
    FindLargest(M, N, Large);
    cout << "Largest is = " << Large << endl;
    cin.ignore();
}

void FindLargest(int x, int y, int &L)
{
    if (x > y)
        L = x;
    else
        L = y;

    cout << "Largest is = " << L << endl;
    cin.ignore();
}
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>127</td>
</tr>
<tr>
<td>N</td>
<td>34</td>
</tr>
<tr>
<td>Large</td>
<td>127</td>
</tr>
</tbody>
</table>