**Lab01: Using Visual Studio 2017 C/C++ IDE**

窗体顶端

**1. 实验目的**

To practice how to create workspace, project, file under C/C++ environment. Get adept to VS Debugger.

**2. 实验步骤**

窗体底端

**2.1 Using VS IDE to run a program**

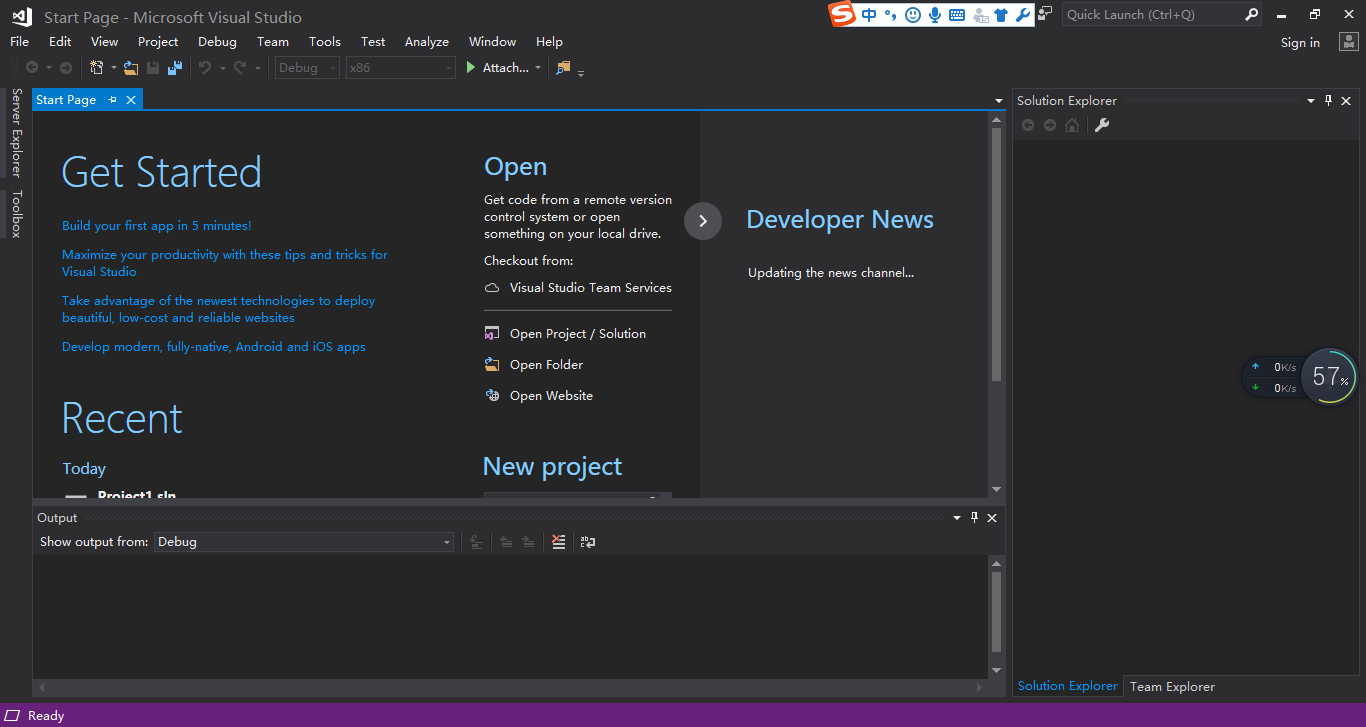
If you know how to create a program with this **IDE**, please jump to **2.2.**

We now describe how to create an empty Visual Studio 2017 workspace. We then create an empty project within that workspace, and, finally, we create a C source program within that empty project. As we describe the steps, we will expect that you will be carrying them out in your computer's copy of Visual Studio 2017.

Start Visual Studio 2017. You are likely to find the icon as figure 1 and double click it.

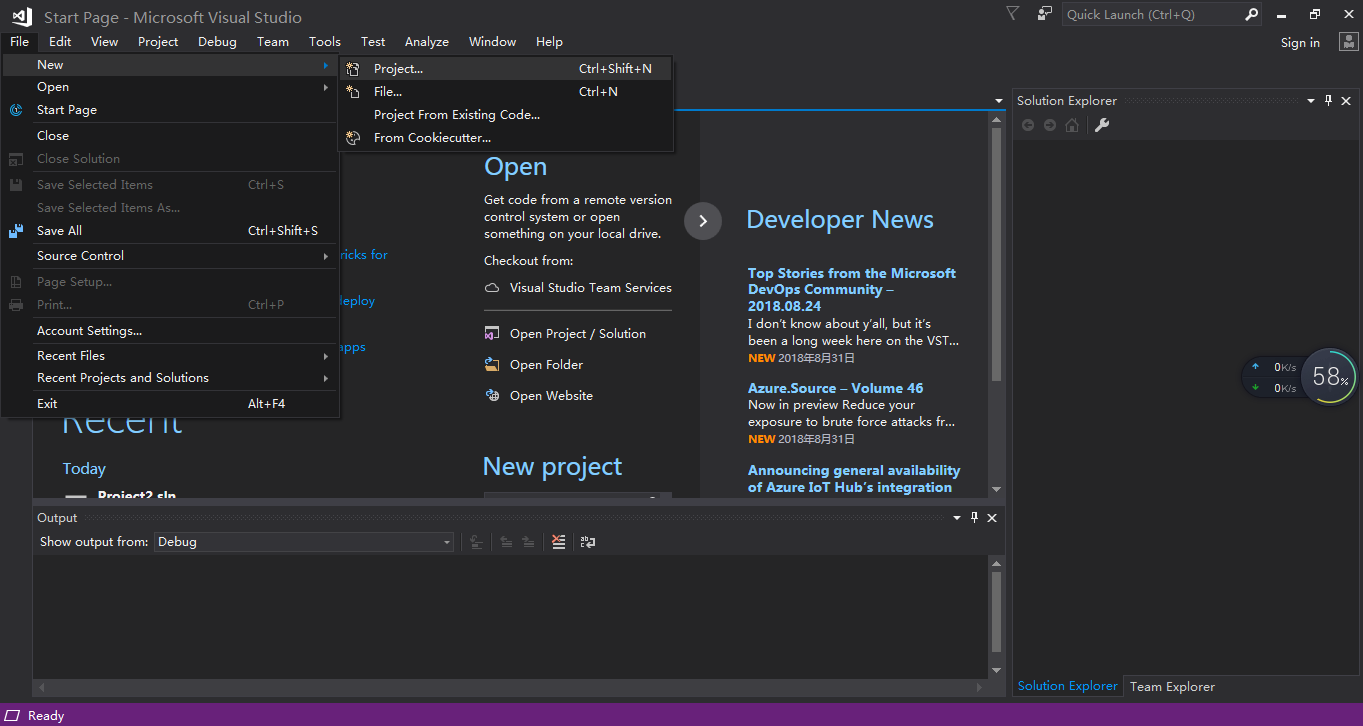
|  |
| --- |
| 说明: 说明: 说明: C:\Users\h\AppData\Local\Microsoft\Windows\INetCache\Content.Word\新图片(5).bmp |
| Figure 1: Double Click to Start Visual Studio 2017 |

Once Start Visual Studio 2017 starts, you will see a window that will look like the one in the figure below. Besides the usual menu and tool bars, you can see the window is divided into three panes. The lower pane is the status area, where the various tools of Visual Studio 2017 (for example, compiler, debugger, text search) report their progress and errors. Above that, the pane on the right is the project area, where the various workspaces, projects, and files will be shown once we create them. Finally, the pane on the left is where we will see the source files when we start editing them.



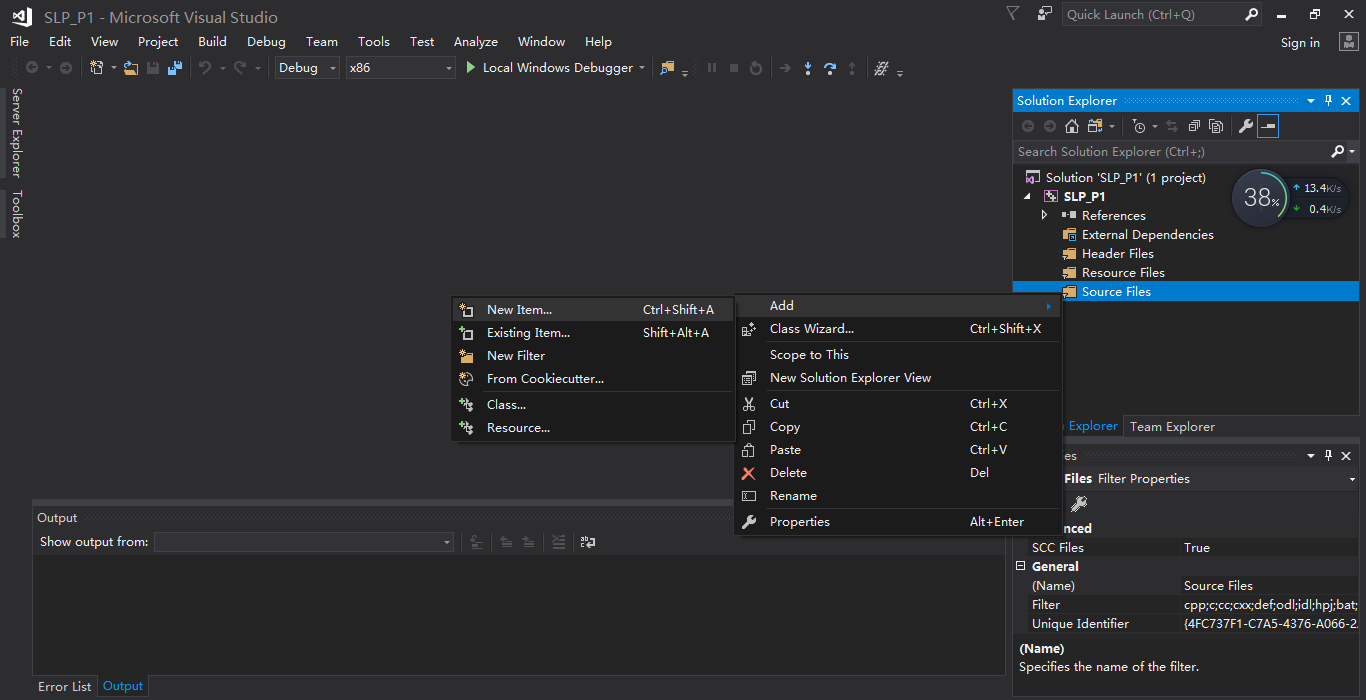
|  |
| --- |
| Figure 2: First Visual Studio 2017 Screen |

We are now ready to create an empty workspace. Pull down the **File** menu, and choose **New...**. You will get a new submenu, and choose **Project…** , like the picture below. Click on **Blank Workspace**, and select a name for your new workspace. As the picture shows, we have named ours "SLP\_P1," but you may name yours anything you like. Finally, press “OK”.



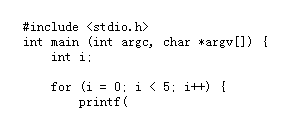
|  |
| --- |
| Figure 3: New Workspace window |
| 说明: 说明: 说明: C:\Users\h\AppData\Local\Microsoft\Windows\INetCache\Content.Word\新图片(5).bmp |
| Figure 4: Choose Empty Project and Name it |

Now try to insert a C or C++ source file into the new project. Right click **Source Files** in the project pane, select **Add** an**d New Item…**. Then choose the right file type and name it “hello.c”.



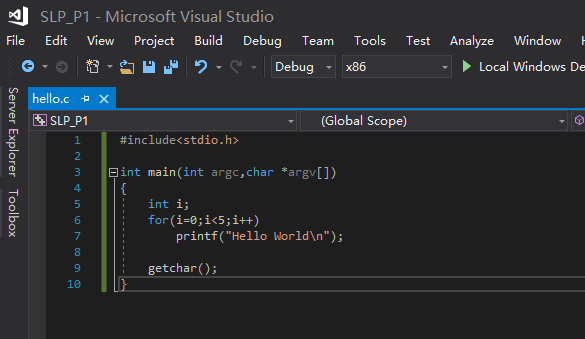
|  |
| --- |
| Figure 5: Insert Source File |
| 说明: 说明: 说明: C:\Users\h\AppData\Local\Microsoft\Windows\INetCache\Content.Word\新图片(8).bmp |
| Figure 6: Insert a C Source File |

Type the following into the edit window:



Note two things:

* As you type, words that have meaning in C, like int and for, change color.
* When you type the parenthesis right after printf, a small window appears. printf is a standard C procedure, that is to say, one that comes with the language when you include the header file stdio.h. So you don't have to write it, and Visual Studio already knows about it. Visual Studio tries to help you remember what parameters this procedure takes by showing you the prototype of the procedure within the small window.



|  |
| --- |
| Figure 7: Typing code |

Now, finish writing the program by typing:

|  |
| --- |
| "Hello World\n"); } getchar(); } |

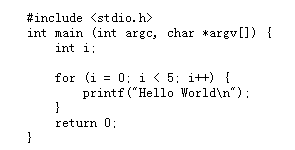
You have just completed the program. You may now quick compile and debug the program by selecting 说明: 说明: 说明: C:\Users\h\AppData\Local\Microsoft\Windows\INetCache\Content.Word\新图片(5).bmp on the toolbar. The compiler will tell you what it does through the status pane at the bottom of the window and at the same time start to debug the program. If you are not yet ready to run the program. You can select **Build Solution** bypull down the **Build** menu. You also may run without debugging by **Start Without Debugging** from the **Debug** pull-down menu. Now you have a program that you can execute, from within or from without Visual Studio.

Before you exit, you should save everything

**2.2 Using Visual Studio Debugger**

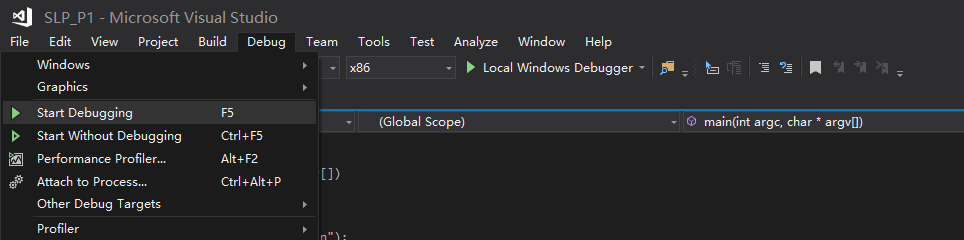
If you know how to debug a program with this **IDE**, please jump to **2.3.**

We will now start to use the Visual Studio debugger. We will apply it to the program that we have created. The program is the following:



Within Visual Studio, open the solution or project by selecting **Open… Project/Solution...** from the **File** pull-down menu. You will need to click on the file of extension .sln. You should have gotten your program back.

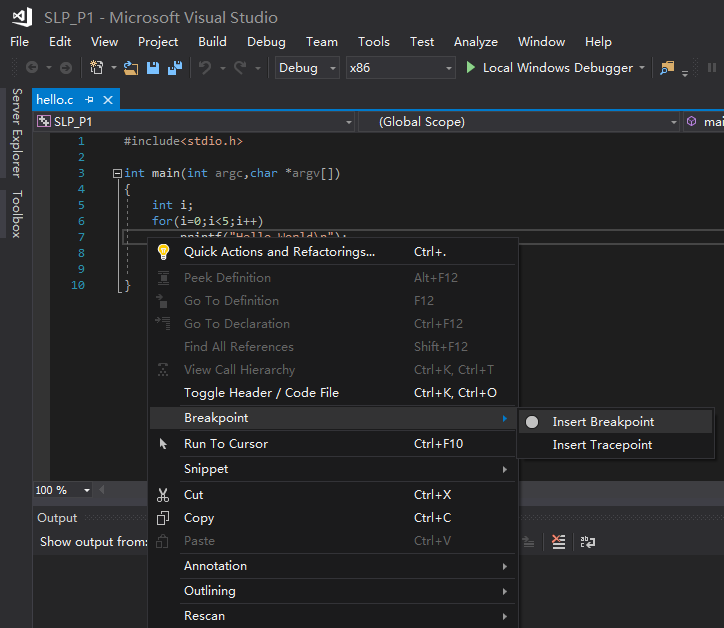
When you start the debugger, Visual Studio automatically starts running your program. Do this now, by selecting **Start Debugging** from within in the **Debug** pull-down menu. You can also achieve the same effect by pressing F5 or by clicking on the 说明: 说明: 说明: C:\Users\h\AppData\Local\Microsoft\Windows\INetCache\Content.Word\新图片(5).bmpicon on the toolbar.



|  |
| --- |
| Figure 1: Starting the debugger |

When you do so, you should see the console window appear and it will disappear after you press any key. But once you delete the last statement “getchar()” in the program, the window will quickly appear and disappear even before you can see anything on it. We bet you did not expect this. What happened is that the debugger started, and finished executing your program.

For the debugger to be useful, we need to set *breakpoints*. A breakpoint, as its name implies, is a point in the program at which execution will be stopped (or "broken"). Once execution stops, you can use the debugger to snoop around, inspect variable values, and observe the general state of the computer. Let's try to insert a breakpoint in the line that calls printf. While the mouse pointer is over that line, press the right button of the mouse. A menu will pop up. Select **Breakpoint…** **Insert Breakpoint**, and click again. You can achieve the same effect by press function key F9 or left click on the left most column.



|  |
| --- |
| **Left click on the left most column of the line** |

|  |
| --- |
| Figure 2: Inserting a breakpoint |

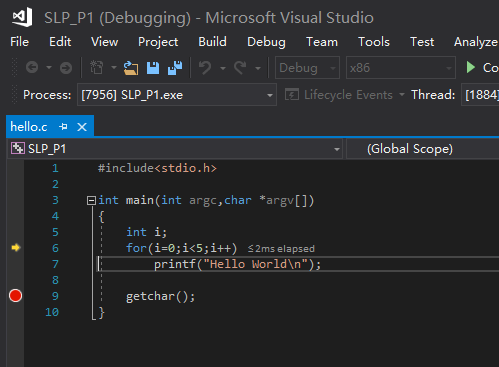
A red circle will have appeared to the left of the line. You can think of this red circle as a stop sign. You may insert as many breakpoints as you want. The debugger will stop execution at each of them. Let's see what happens now if you start the debugger. Press F5 again. You will see that this time, the console window stays around, empty, and that a yellow arrow appears near the "stop sign." This arrow indicates that the program is currently executing at that point.

From within the debugger, we can control execution in several ways. Let's try two of these. In the debug toolbar, you see the following two icons:

* 说明: 说明: 说明: C:\Users\h\AppData\Local\Microsoft\Windows\INetCache\Content.Word\新图片(8).bmp Will continue execution, and automatically stop it at the beginning of the next line of source code. (This can also be done by selecting **Step Over** in the **Debug** pull-down menu.)
* 说明: 说明: 说明: C:\Users\h\AppData\Local\Microsoft\Windows\INetCache\Content.Word\新图片(9).bmpWill continue execution until the program reaches the current location, or until it reaches a breakpoint, whichever comes first. (This can also be done by selecting **Continue** in the **Debug** pull-down menu or press F5.)

Try clicking on the **Step Over** button. You will see that the yellow arrow moved to the next line, indicating that the execution is now stopped at the end of the loop. You can see that the program did execute the printf because a single "Hello World" now appears in the console window. Now, try clicking the **Run to Cursor** button. The yellow arrow is back on the stop sign, indicating that the program's execution continued, coming around the loop, and not stopping until the breakpoint was again reached. If you click on these buttons a couple of times, you will see that the execution goes around the loop, stopping in every iteration, and printing "Hello World" one at a time.

Let's now delete the breakpoint and insert another one in the line "getchar();". To quickly delete the breakpoint, place the cursor over the red circle and left click again. The stop sign is gone. Now, insert a breakpoint in the " getchar();" line. Things will look like this:

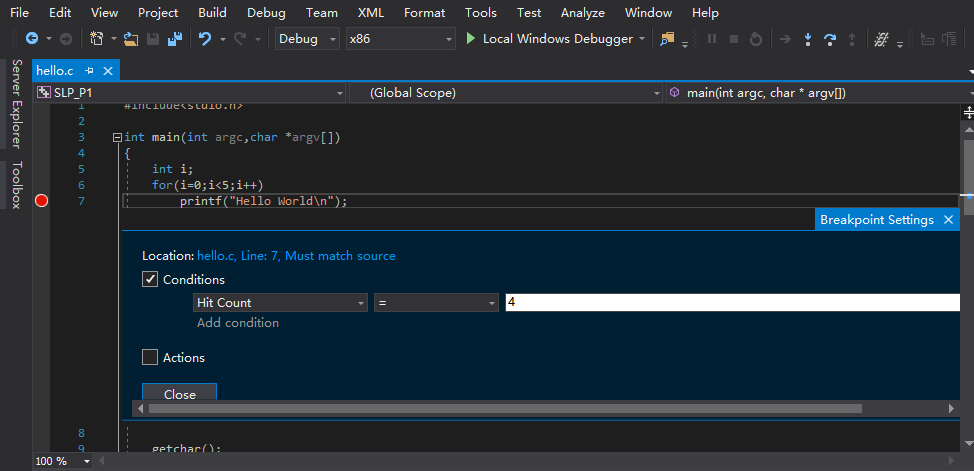


|  |
| --- |
| Figure 3: After inserting the second breakpoint |

Click on **Continue** and the program will continue executing until the next breakpoint.Try it.

There are two other ways in which we can control program execution, ways that we will not be showing. The **Step Into** 说明: 说明: 说明: C:\Users\h\AppData\Local\Microsoft\Windows\INetCache\Content.Word\新图片(8).bmp button is equivalent to **Step Over**—unless the program is stopped at a line that has a procedure call. In that case, **Step Into** will stop in the first line of the callee. The **Step Out** button continues execution until the current function returns, and then stops it at the caller.

There are also many other ways of setting breakpoints. For instance, we can specify that execution does not stop until the fourth time the program happens upon a breakpoint. To modify breakpoints in this manner, set a breakpoint before “printf()”, then right click the breakpoint, select **Conditions…**,**Hit Count…,**and set the value to 4 as showing in figure 4 on the pop-up menu. Or we can specify that the program does not stop unless a particular condition (for example, i == 4) is true. Henceforth, we won't be covering the various features of Visual C++ in detail: we will assume that you are sufficiently familiar with it to strike out on your own—aided by courage and the comprehensive Visual Studio help facility.



|  |
| --- |
| Figure 4: Edit Conditional Breakpoint Dialog |

**2.3 Examining Data**

If you already know how to watch the value of variables when debug a program with this **IDE**, please jump to **2.4.**

There are three main ways to inspect the values of variables in Visual Studio:

* While debugging a program, the value of a variable appears in a small window when the mouse pointer pauses briefly over the name of that variable in the source code window.
* While debugging, a pane below the source code window shows the values of selected variables. There are three useful selections: the **Auto** selection, the **Locals** selection and the **Watch 1**.
  + The **Auto** tab shows variables that are automatically deemed of interest at the point where the program is currently executing. These may or may not be the variables that you are interested in; Visual Studio guesses what you might be interested in, but it cannot be all knowing.
  + The "**Locals**" tab shows all variables that are local to the currently executing function, including the parameters to the function.
  + In the **Watch** pane we can type any valid C expression whose value we are interested in. The expression is evaluated in the scope of the current breakpoint. These expressions can be simple variables, or any other expression that could appear on the right-hand side of a C assignment.

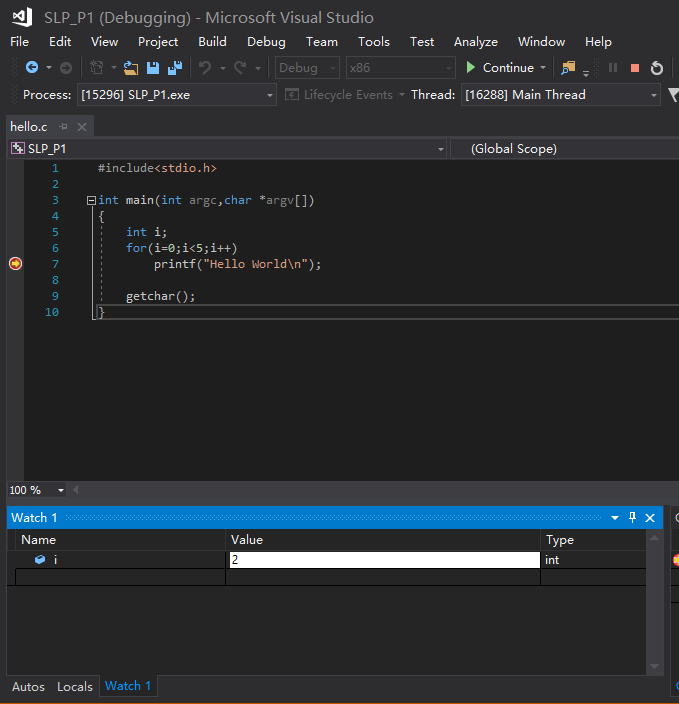
You can also modify the current value of a variable *while* the program is running, by displaying the value using the **Variable** or **Watch** windows, or both, and double-clicking on the value.

Let's practice. Open the workspace where you saved our "Hello World" program, and insert a breakpoint in the printf line. Then start up the debugger (F5) and position the mouse pointer over one of the appearances of i in the for line. You will see that the value of i appears. Press F5 again to let the debugger go through one iteration, and again place the mousepointer over i. Has the value changed? Convince yourself that you understand what is going on.

Now pay attention to the variable pane below. It should show the value for i. It may show in black or in red. Red signifies that that value has changed since the program was last stopped. This meaning of "red" is consistent across Visual Studio, as we will see. But be warned: Visual Studio is not always good at keeping track of changes when we open and close variable and watch panes. You should not trust the color of the values absolutely; always keep track yourself, when in doubt. Press F5 again, just to see the value of i in the variable pane change.

Now, switch to **Watch** pane,this new pane will be empty. Click on it, type "i", thenEnter. You will see that the value of i again appears and behaves like the entry in the variable pane. Repeat the process, but this time type "i == 4", and then again type "i + 2". You will see that the values of these expressions at the current point of execution appear and are recomputed every time the program hits a breakpoint.

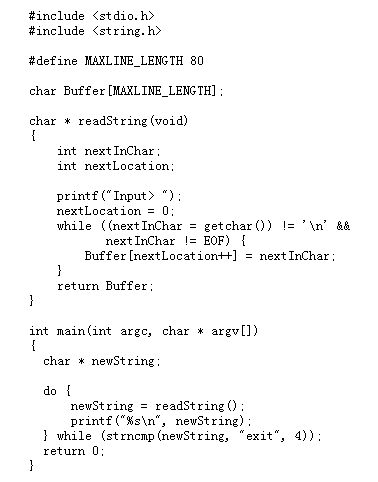
Try changing the value of i, for example, to 0 and continue running the program. How many of the "Hello World" do you get now? Do you understand why?



|  |
| --- |
| Figure 1: Examining data using the **Watch** pane |

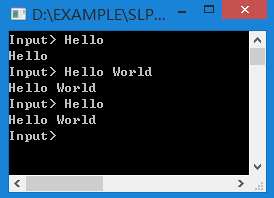
**2.4 Debug and fix "Echo.cpp" program**

Time to debug! The program below has a problem. It is intended to take anything we type into it and output it back. But it doesn't quite do that. For example, if we type in "Hello", it does output "Hello". If we then type in "Hello World", it correctly outputs "Hello World". However, if we again type "Hello", it insists on printing "Hello World". It's really enamored with the "World" word, and won't drop it! For your playing pleasure, the program is also available in the file [Echo.cpp](http://swjx.scu.edu.cn/moodle/file.php/49/systemlevelprogramming/week1/lab1/Echo.cpp).



First we ought to understand the code. To do so, you have to be a little familiar with C. This shouldn't be a problem, since you are already likely to be familiar with C++, and the two are fairly similar. However, you may need to review a thing or two and this will be a good opportunity. Next, we need to understand what some of the library functions do. The program calls the functions strncmp() and getchar(). The documentation for all library functions is available within the Visual C++ help facility. An easy way of getting the relevant documentation is to highlight the name of the function you want to know more about, and then press F1. You can also get to the documentation by starting from the **Help** pull-down menu. Check out the documentation, and examine the program above until you understand what it tries to do and how.

Do you understand the program now? Okay, here is an example of what it does when we type the instances of "Hello" as described above:

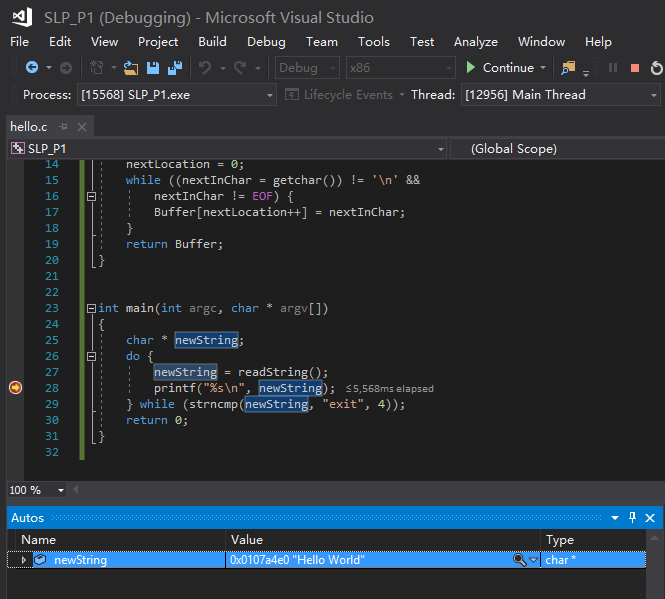


|  |
| --- |
| Figure 1: Behavior of the Echo program |

Try it yourself. [Download](http://swjx.scu.edu.cn/moodle/file.php/49/systemlevelprogramming/week1/lab1/Echo.cpp) the file, make a new Visual Studio C++ solution, and run it.

Programmers often debug by working backwards from the known erroneous symptoms. That is what we will do here. The symptom is that when we run through the loop in main() the third time (when we give the program input as in Figure 1), the printf prints the wrong string. Let's first make sure that the value of the string is indeed wrong at that point: if it weren't, we would have to blame printf() for the problem, unlikely as that might be.

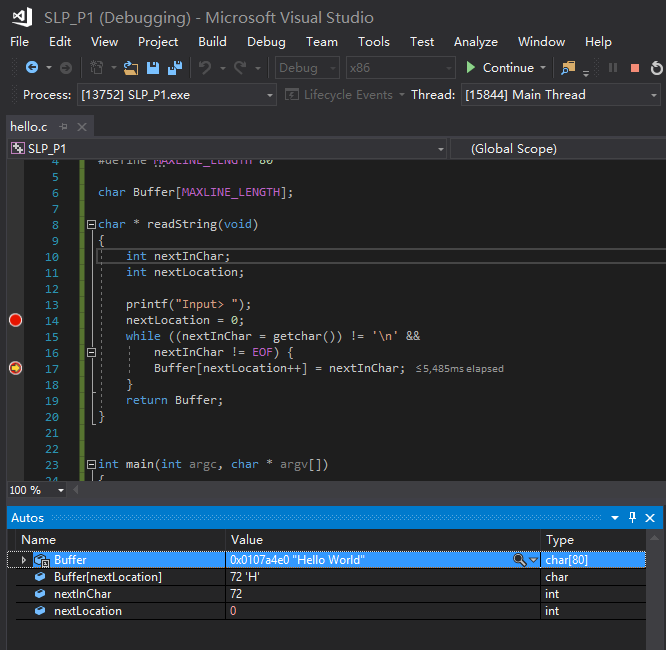
Insert a breakpoint at the printf() line, then press F5, and type "Hello" into the console window. Press F5 again, and type "Hello World". Once more, press F5 and type "Hello". You will have just taken the program to the brink, to the point immediately before the bug manifests itself. Take a look at the value of newString. It should be "Hello", but it is "Hello World" instead. So the problem is not within printf after all. We have discarded this one possibility.



|  |
| --- |
| Figure 2: Printf is OK |

If the value is already wrong when printf is called, then it has to be wrong when readString returns. Let's examine the way readString constructs newStringby placing a breakpoint within it. We would like to insert the breakpoint inside the while loop, but think of what would happen if we did that. How many times would we have to press F5 to get the program to the point where our problem arises? We would have to do this once for every character that we typed in up to that point. That's 18 times, including the new line at the end of each input.

Instead, we can insert a breakpoint before the while loop, let it go twice, and *then* set the breakpoint inside the loop. This way, we only need to press F5 three times. Let's do this, and then let's take a look at the value of Buffer. At this point, the program hasn't yet started to fill in the string that readString will return, but Buffer already has the value "Hello World"! How could this happen?



|  |
| --- |
| Figure 3: "Hello World" is already there! |

This happened because, in C, there is no string data type. Strings in C are no more than arrays of characters. It is a convention that strings are null-terminated, that is, that the string contained within a character array ends with the first zero ("\0"). But this is a convention that is not enforced by mere usage of the character arrays. The zeros that terminate strings have to be inserted explicitly by programs that manipulate them. In this case, the array Buffer still contains the value that it had at the end of the previous call to readString. Whereas the current call will overwrite the first five characters in the array with "H", "e", "l", "l", and "o"—the remaining six characters from the previous call will remain as part of the character array that readString will return this time.

Your task is to fix this error.

C and C++ are similar, but as you have just seen, C has fewer built-in types and other abstract machinery than C++ does. For this reason, many people prefer to program in C++. But C's relative lack of abstraction is also its strength. Because C programs are more similar to the way the computer actually executes programs, C affords greater control and insight during execution.

**2.5 Using Memory and Register Window**

By using watch window, we can display the value and address of variables. Now let’s go further to inspect memory and CPU registers. Debug the following code and set a breakpoint before instruction *printf*. Get the address of a in watch window.

#include <stdio.h>

int main(void)

{

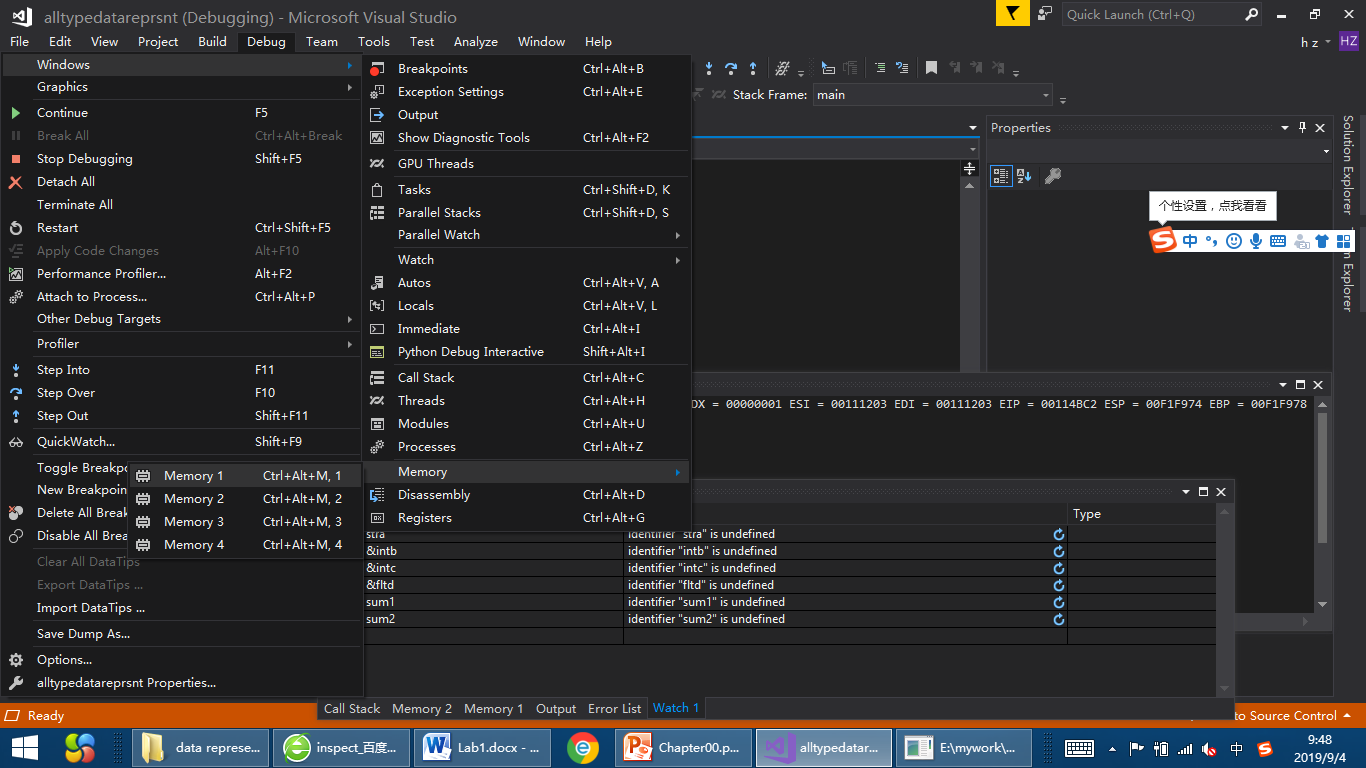
float a = 10.0f;

printf("a = %d\n", a);

return 0;

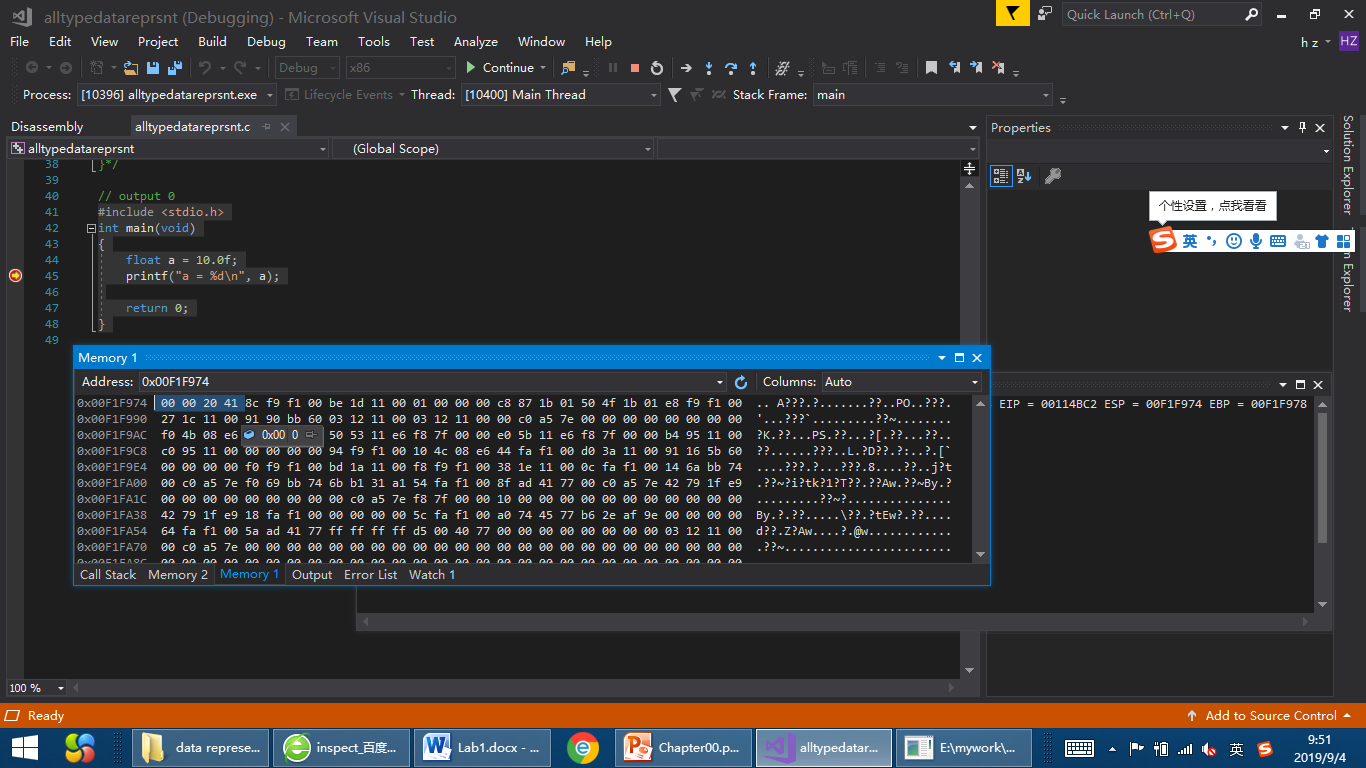
}

Now select **Debug** on menu。Debug—>Windows->Memory-> then any one of the 4 choices.



Type in the address of a and click enter. Now you can see how a single floating variable a is stored in memory. Note that a takes up 4 bytes, the address comes from the lowest byte, as showing in the following figure,

|  |  |  |  |
| --- | --- | --- | --- |
| Highest byte |  |  | Lowest byte |
| 41 | 20 | 00 | 00 |



Define an integer array **arr[5]** in your program and initialize the elements as 1,2,3,4,5. Try to see all the values in the memory window. You can get the address of **arr** in three ways in watch window by typing **arr**，&arr[0],&arr.

In computer architecture, we learn there are some registers in the CPU, they are used to store states or data of a program. In any condition you want to inspect one of them, you can use register window.

Now select **Debug** on menu。Debug—>Windows->Register

