Lab 1 - Getting Started with MASM in Visual Studio 2022 and Debugging the Code
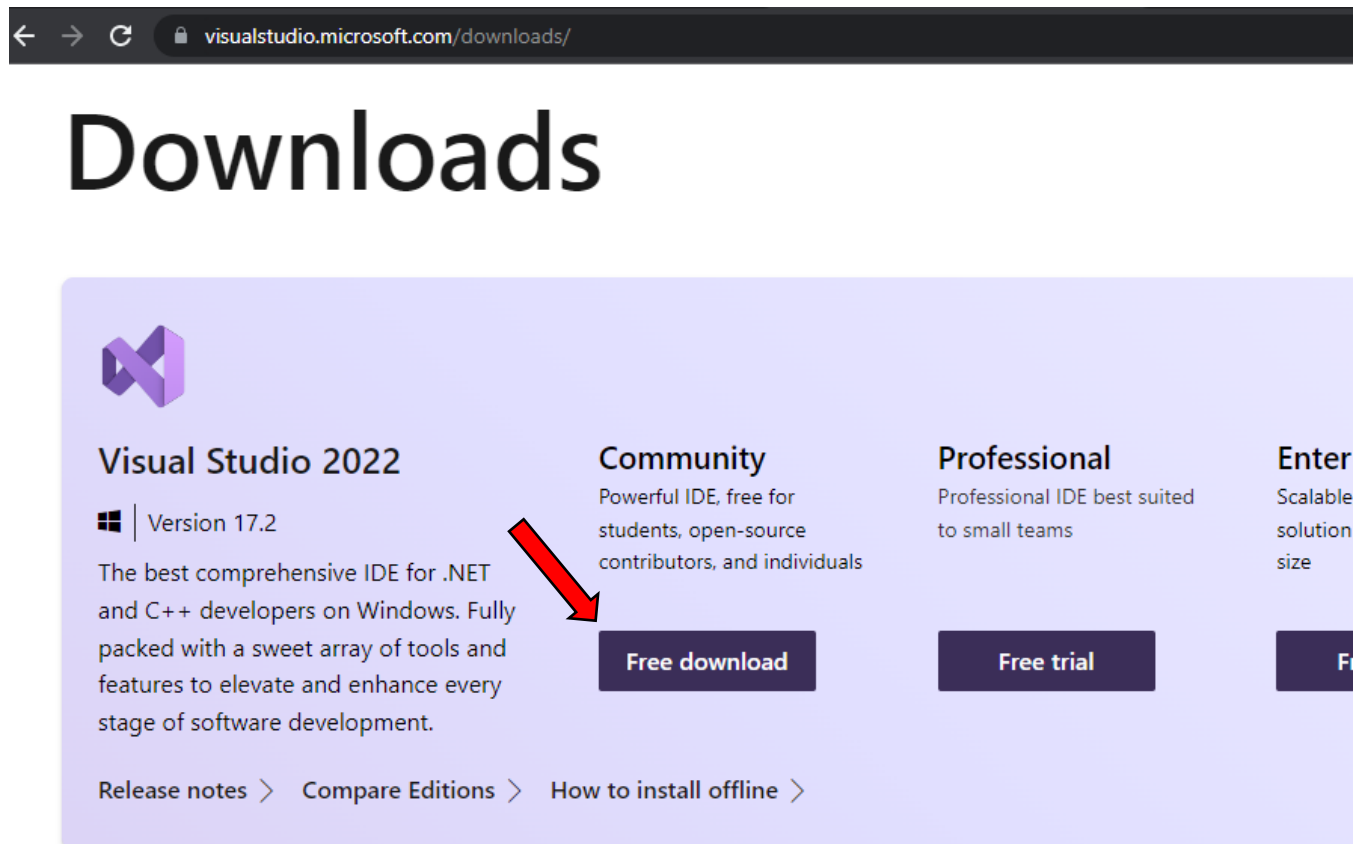
Reference Link for Visual Studio 2022: http://asmirvine.com/gettingStartedVS2022/index.htm
Reference Link for Visual Studio 2019: http://asmirvine.com/gettingStartedVS2019/index.htm

1.1 Installation of Visual Studio 2022

Visit this **Link** to download installer of Visual Studio Community 2022.

After downloading and installing the Visual Studio Installer, open the Visual Studio Installer from Start Menu. And select the "Desktop development with C++" and start the Installation.
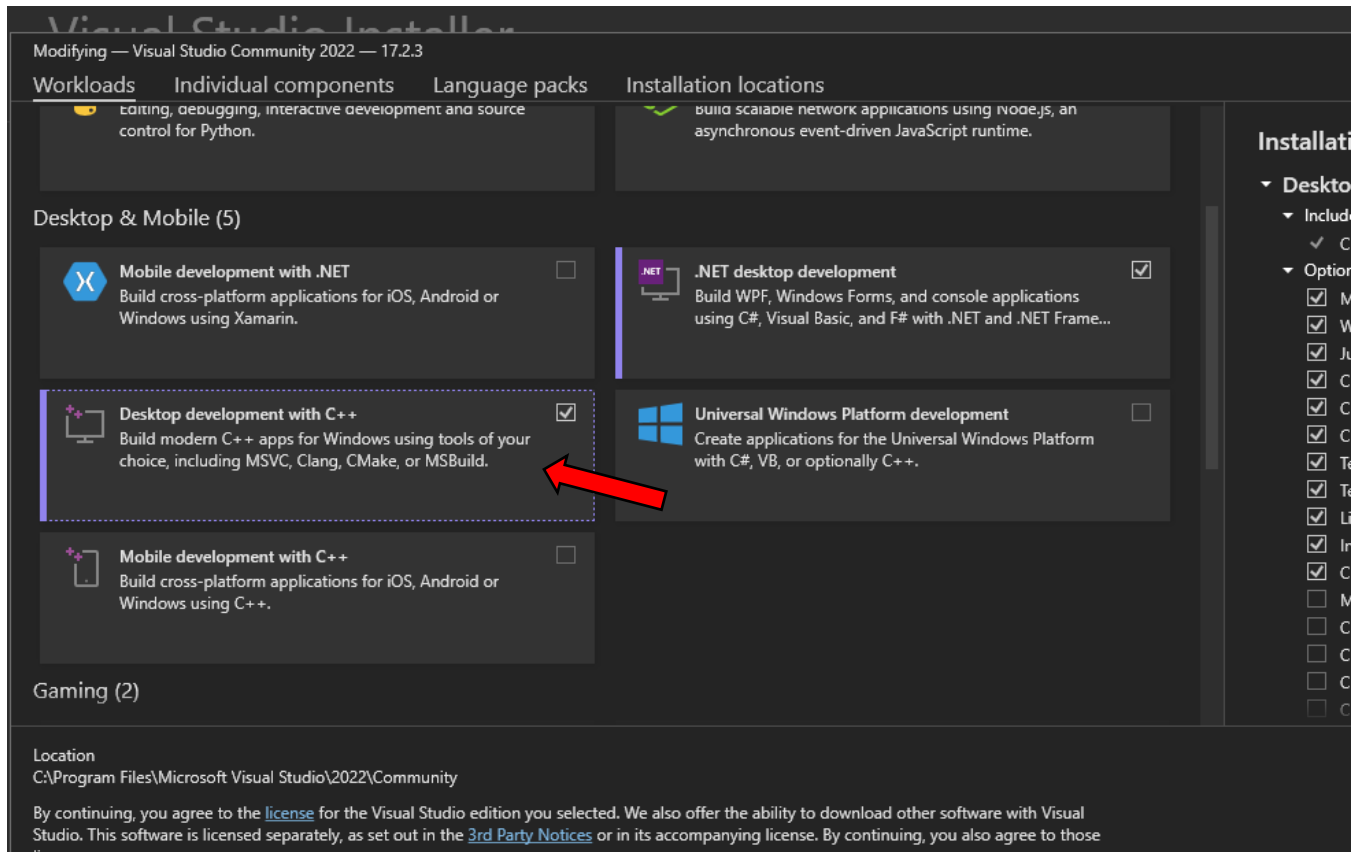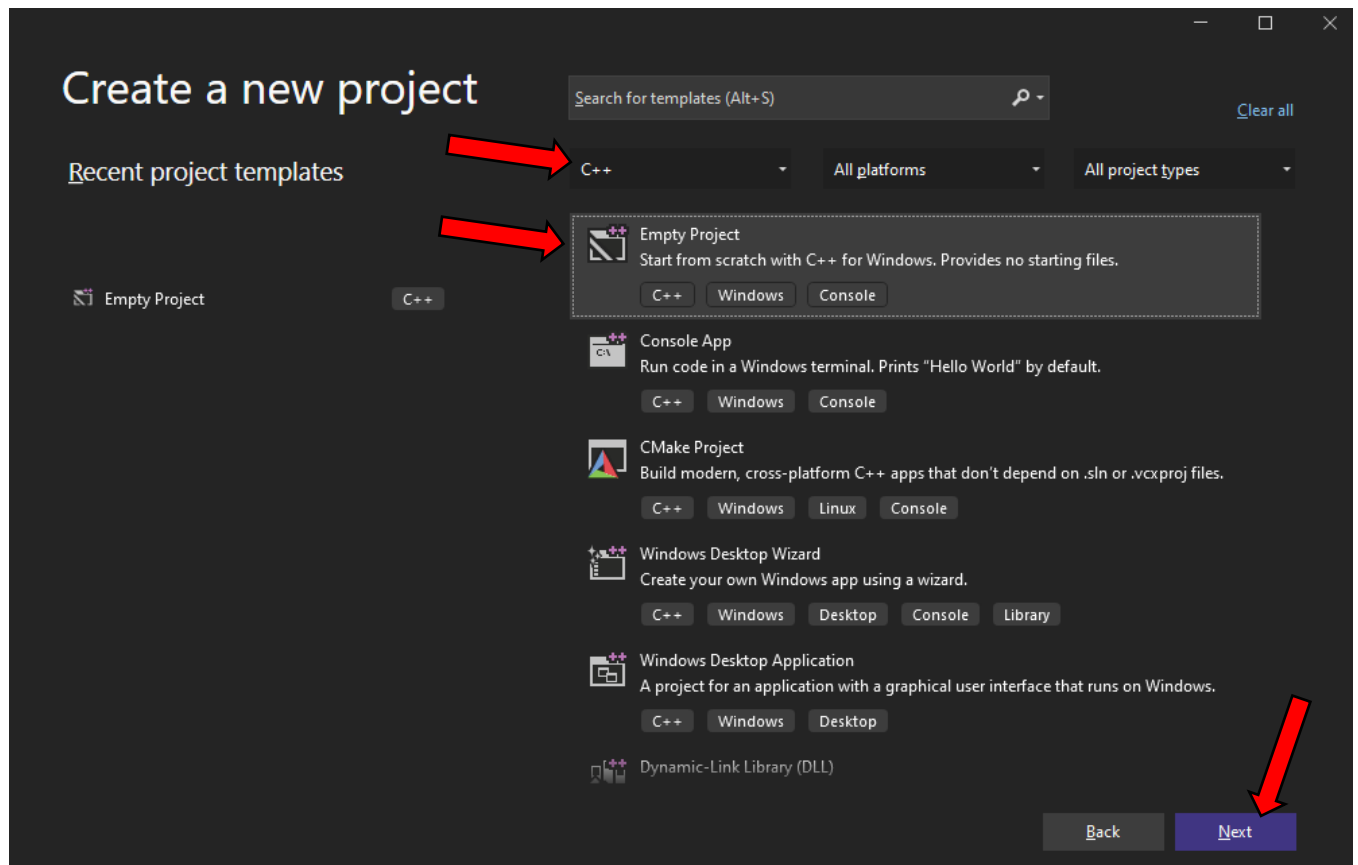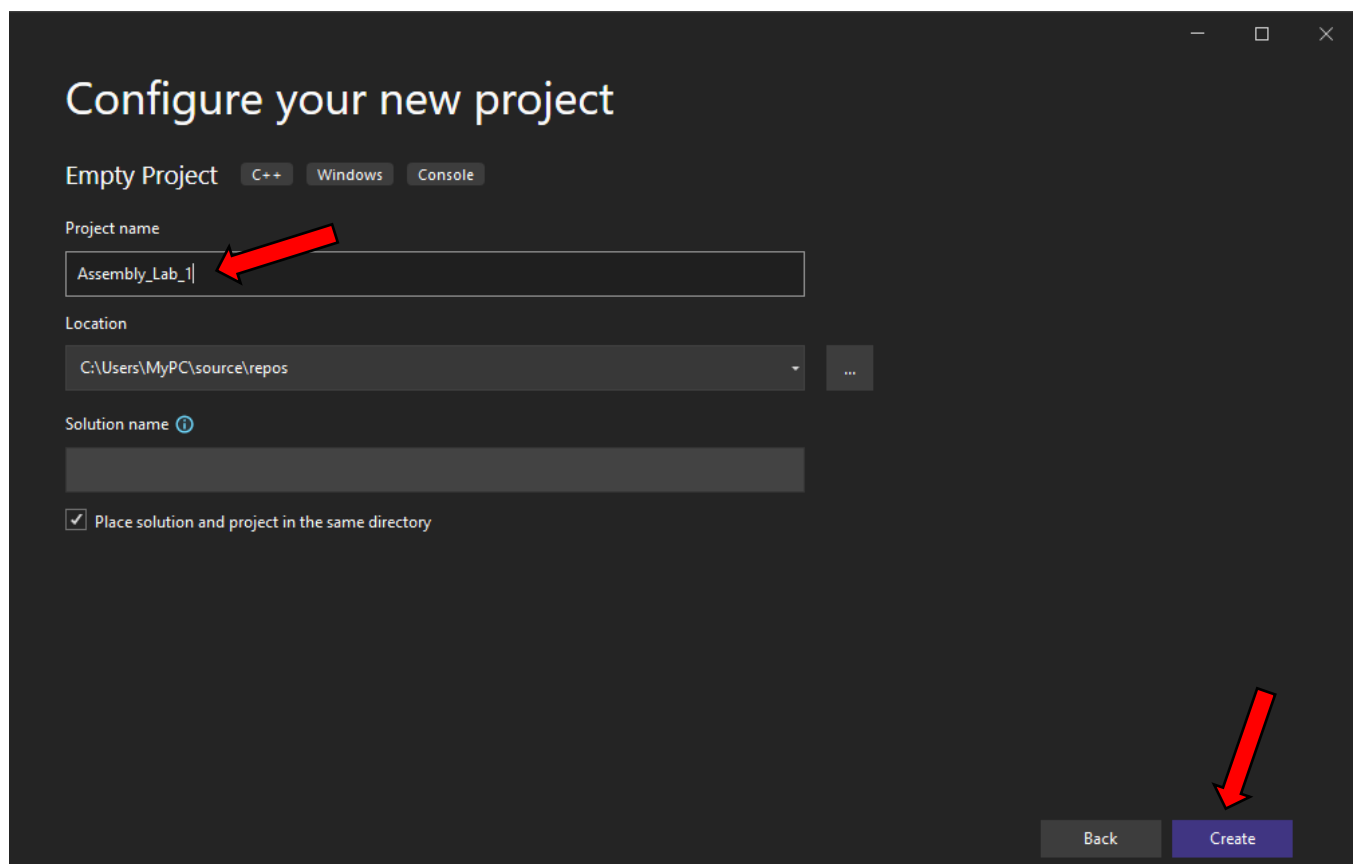
1.2 Configure Visual Studio for MASM

After the successful installation of Visual Studio, Open the Visual Studio. When you start Visual Studio, click the "Create a new project" panel on the startup window:
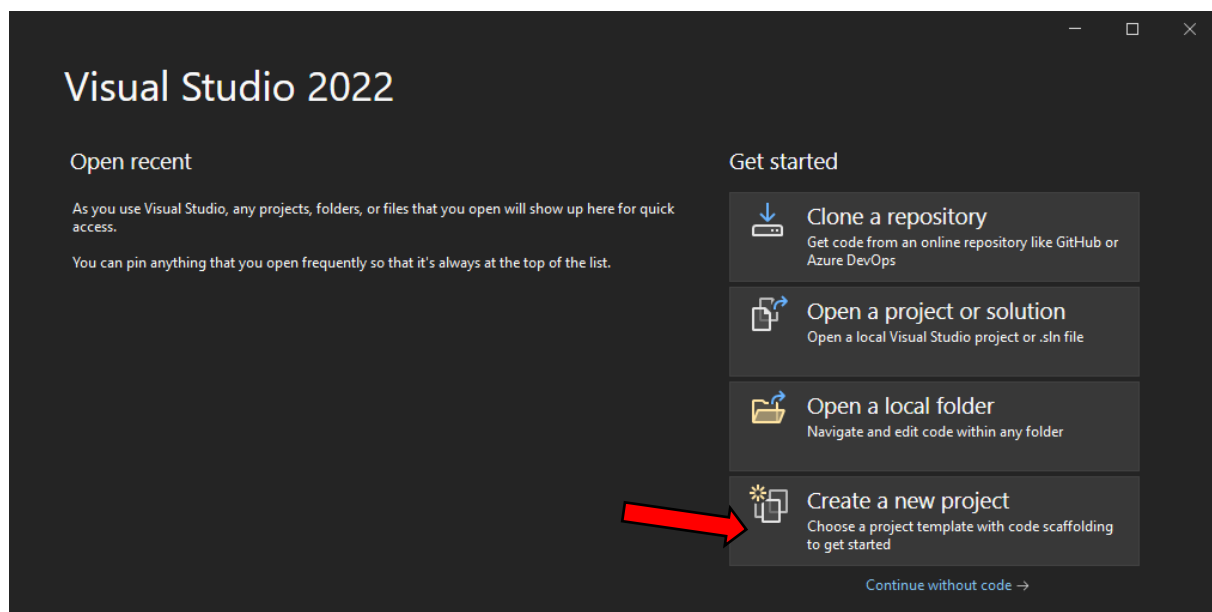

Next, in the Create a new project dialog window, select C++ in the list. Select "Empty Project" and click on Next button.
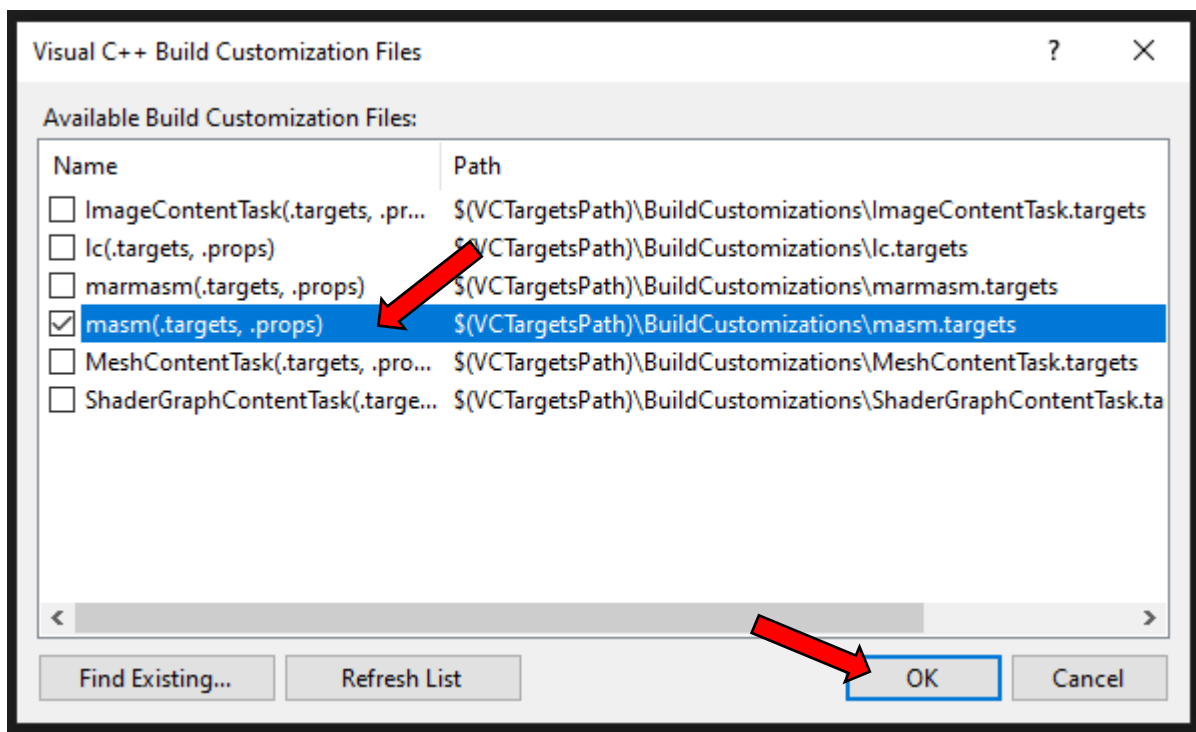
Enter Project name and clink on "Create" button.

The newly created project will open in the Visual Studio window. Click on "Solution Explorer" Tab on the right side of the screen and right click on the Project in the Solution Explorer and select Build Dependencies -> Build Customizations.
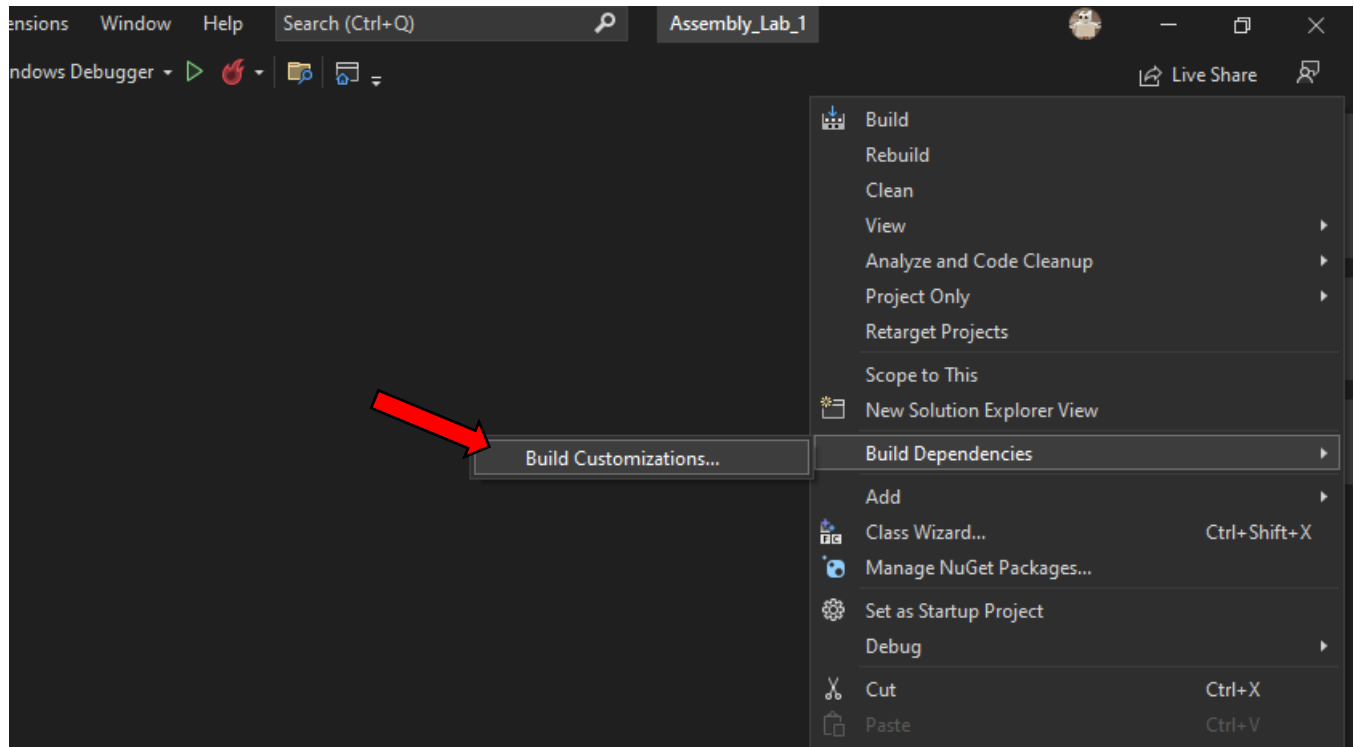
Tick the "masm" checkbox and press OK button.

Add a new empty Assembly file to the project by right clicking on the Project and selecting Add -> New Item with "C++ File". Enter a filename ending with .asm like "test.asm".
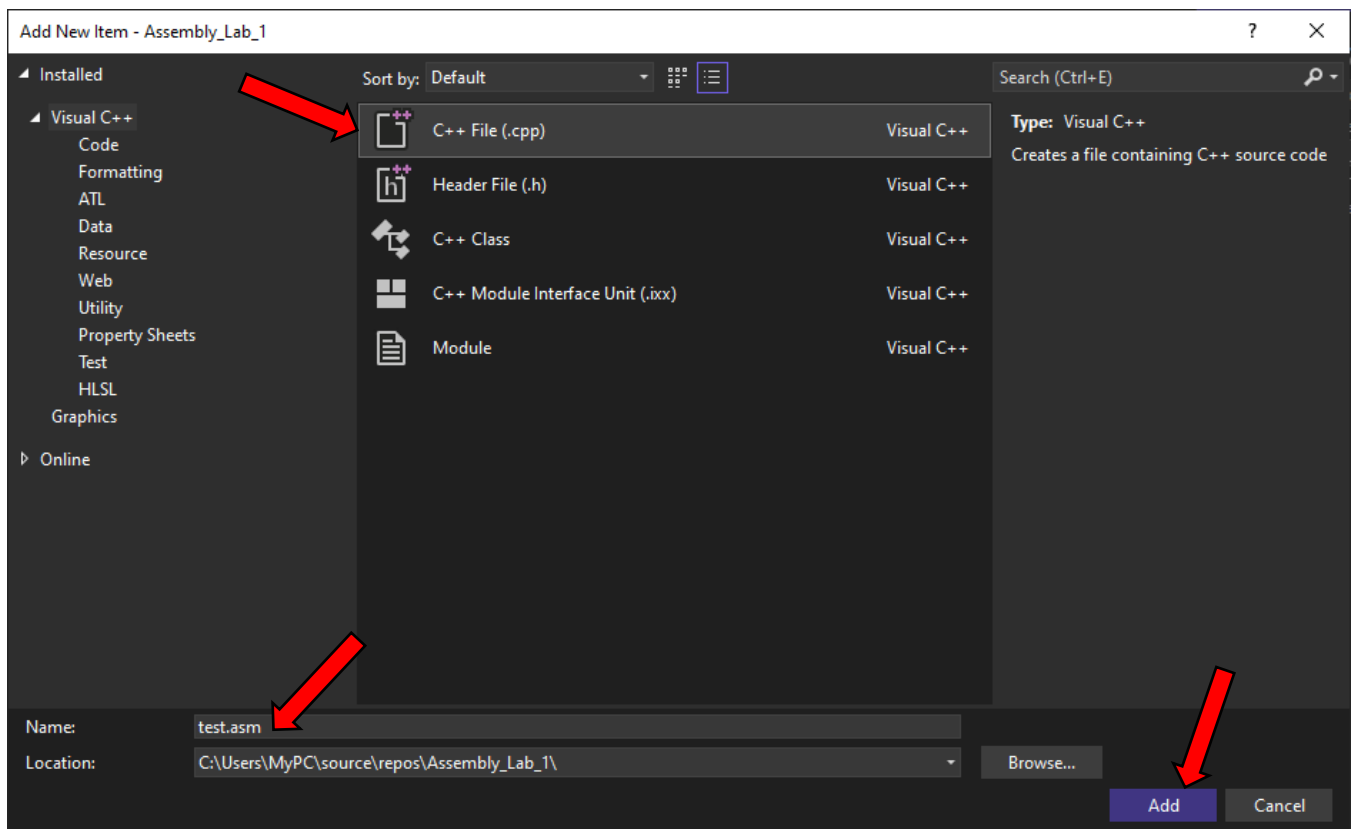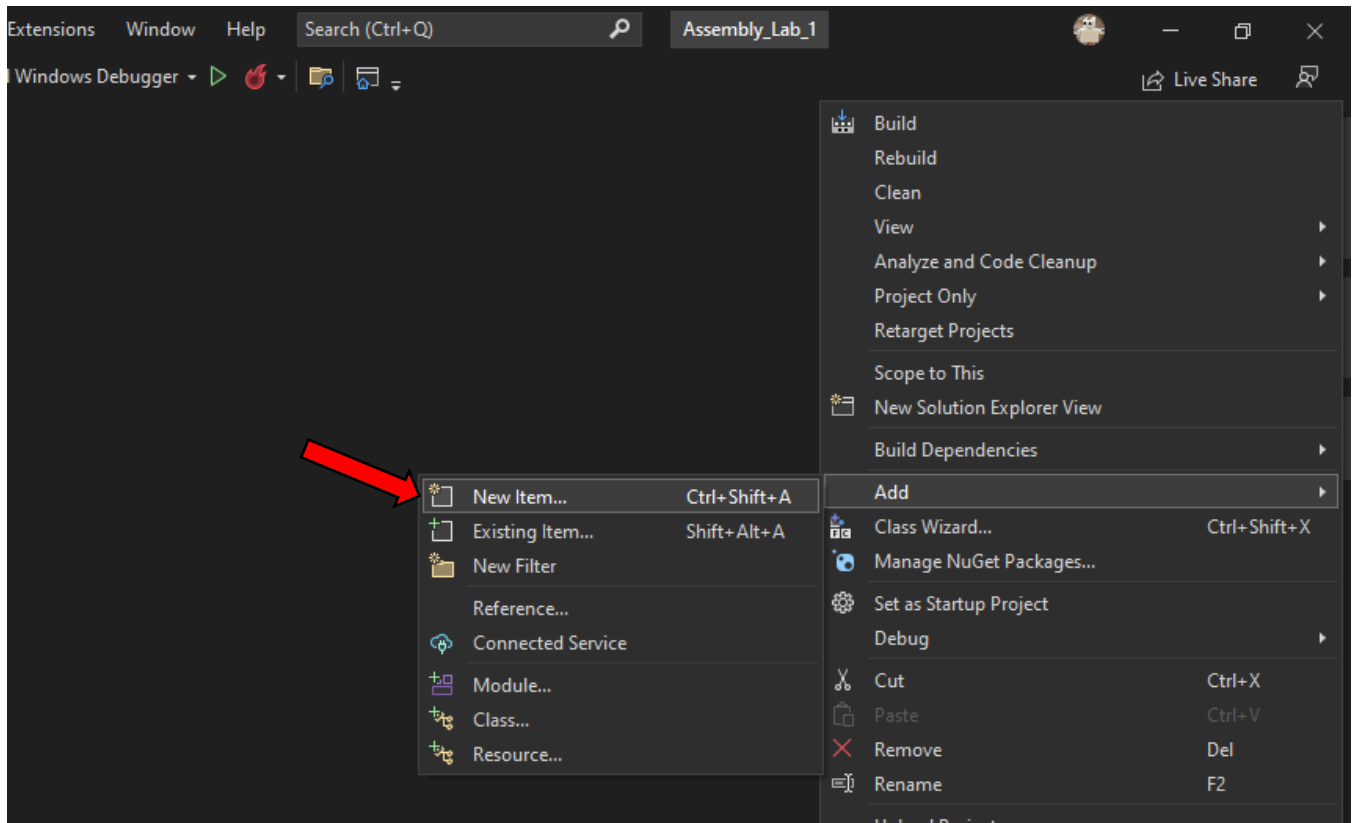
An empty Assembly files will be created. Now you can write the Assembly code and execute the code.

Copy and Paste this Sample code in the "test.asm" file and Build and Execute the code.

Copy these code lines and paste into the "test.asm" assembly file. Select x86 from "Solution Platform" dropdown list and then press Local Windows Debugger button to build and execute the code.

```
; This Code adds two 32-bit integers.

.386
.model flat,stdcall
.stack 4096
ExitProcess proto,dwExitCode:dword

.code
main proc
        mov     eax,5        ; Moving 5 to EAX Register
        add     eax,6        ; Adding 6 to the register EAX
                             ; and store the result in the EAX itself

        invoke ExitProcess,0
main endp
end main
```
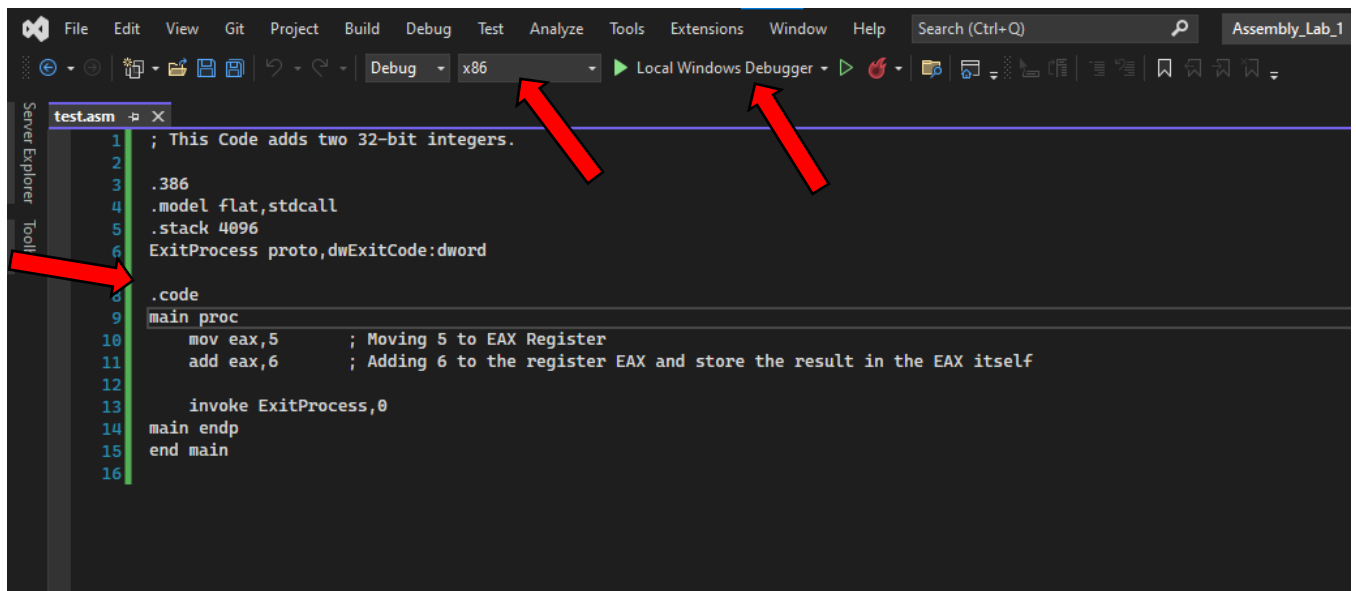

If everything was configured correctly, the code will execute, and a blank console window will popup saying that the code is "exited with code 0". It means our code run and closed correctly without any error.
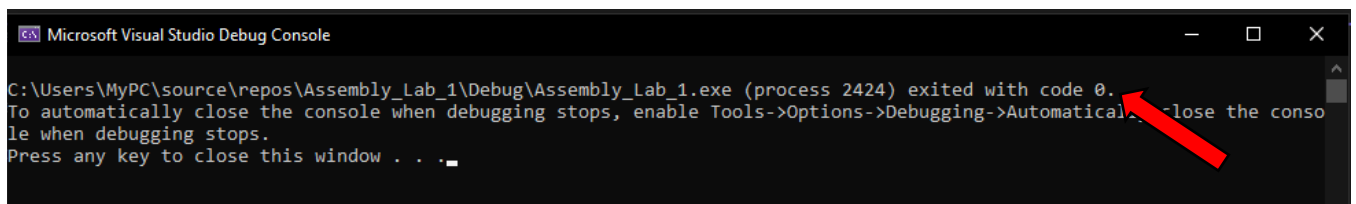
1.3 Adding Kip Irvine Book (7th Edition) Support in Visual Studio
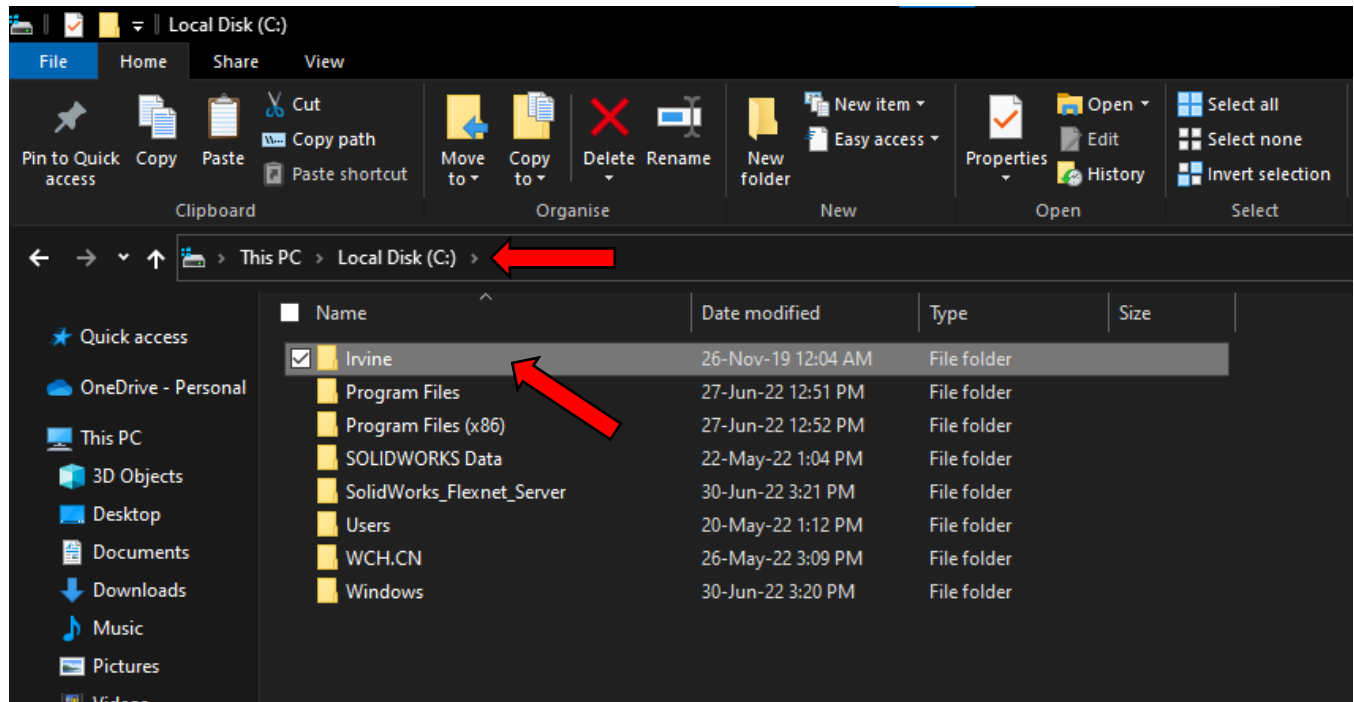
To execute the example codes and use the Kip Irvine libraries for input and output operations, we have you add these libraries in the Project properties of our newly created assembly project. For that purpose, first of all download the "Irvine.zip" file from this **Link**.

Extract the zip file and place the Irvine folder in the C: Drive (e.g., C:\Irvine).

Then open the Visual Studio and open the Assembly Project we created earlier. Go to "Solution Explorer" and right click on the Project and click on "Properties". A dialog box will open. On the Left side of the box, go to the "Microsoft Macro Assembler" tab.

Insert "**C:\Irvine**" into the "Include Path" text box. Then Click on Apply button.
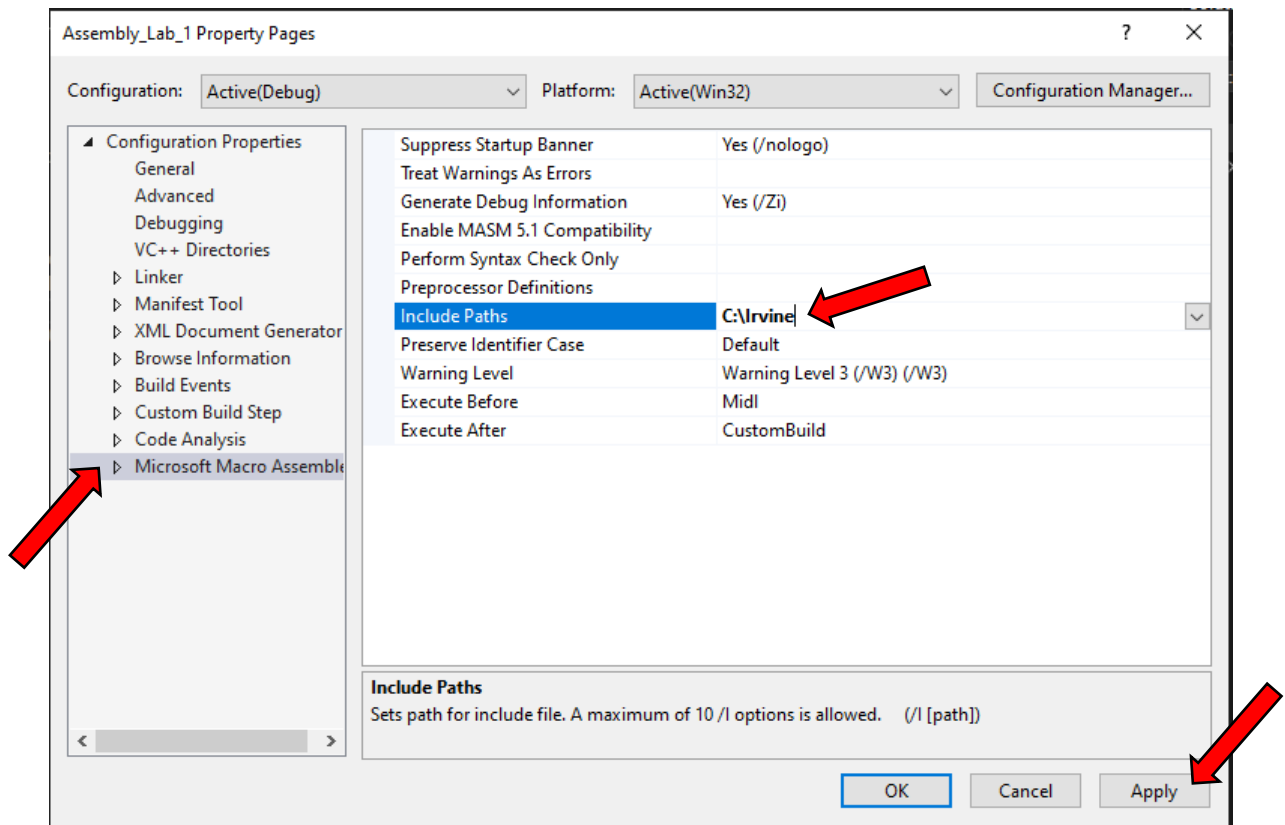
Then Go to Linker -> General tab and paste this string "**c:\Irvine;%(AdditionalLibraryDirectories)**" in Additional Library Directories text box.
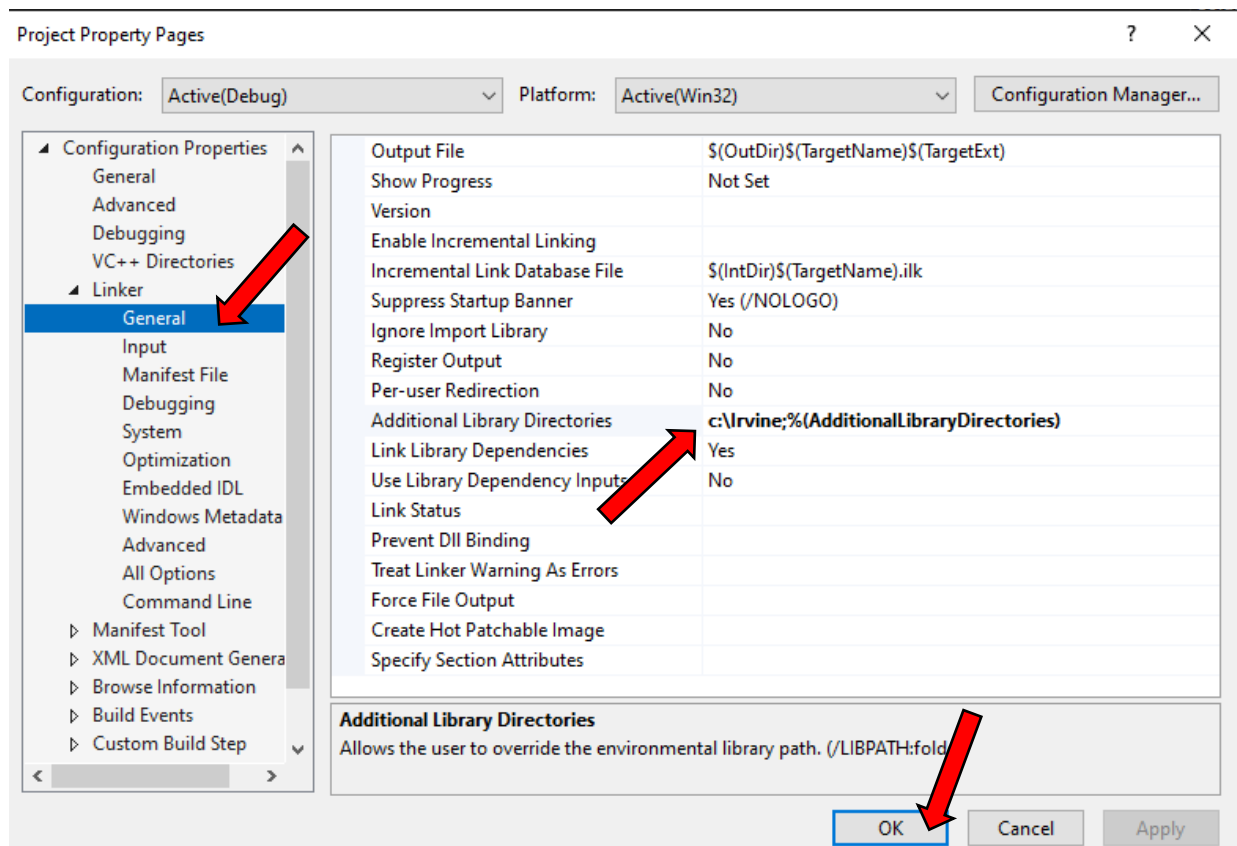
Then go to Linker -> Input tab and paste this string "**user32.lib;irvine32.lib;%(AdditionalDependencies)**" in the Additional Dependencies text box and press the OK button.
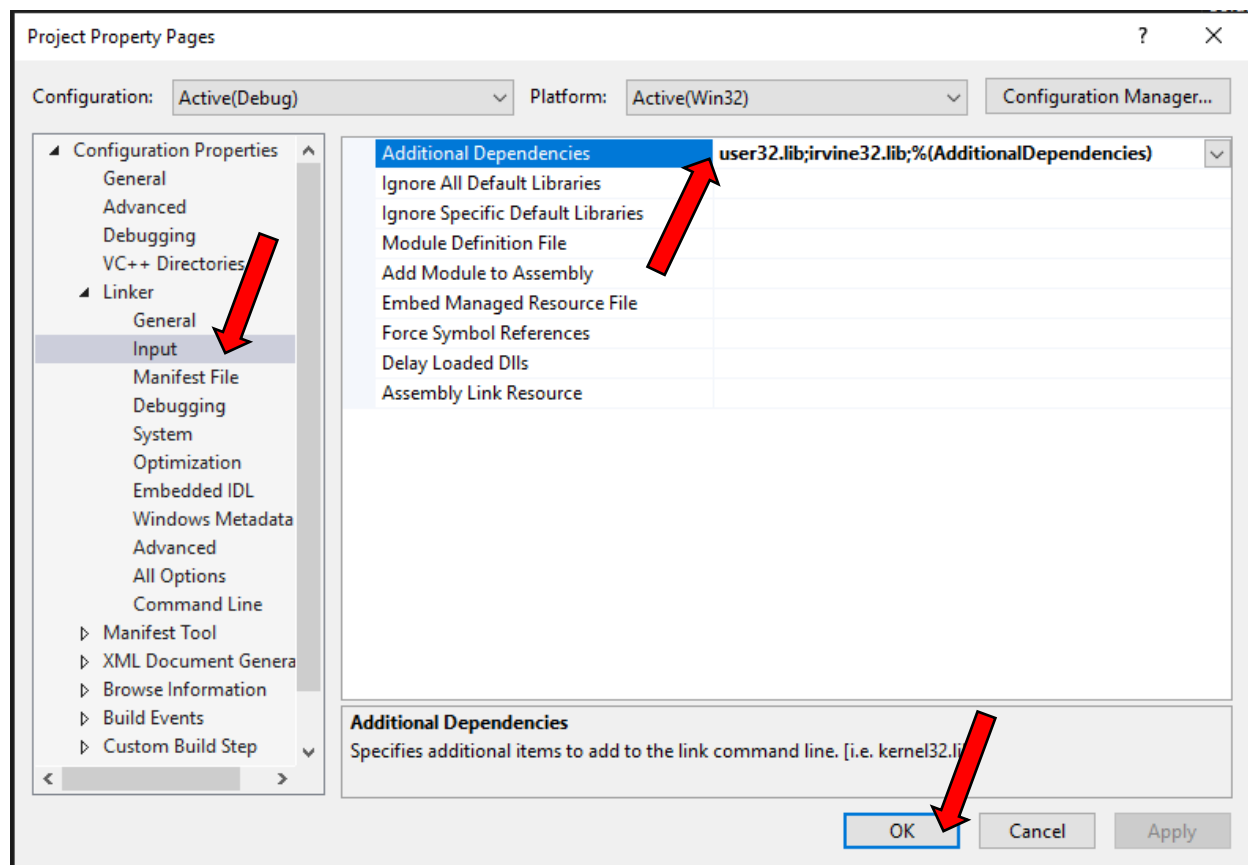

All Done!

Important Points:

➤ After all these configurations, now you can build and execute the examples code given in the book in your Visual Studio project. But you have to perform all these steps every time you create a new project.

➤ To solve this problem, I am providing you the pre-configured Visual Studio project containing a demo Assembly file and where all the linking and libraries configuration is performed built-in. You just have to open the project and then open the given demo assembly file and then replace the code of that file according to your requirements.

➤ Download the pre-configured 32-bit project for **Visual Studio 2022** from this **Link**.

➤ Download the pre-configured 32-bit project for **Visual Studio 2019** from this **Link**.
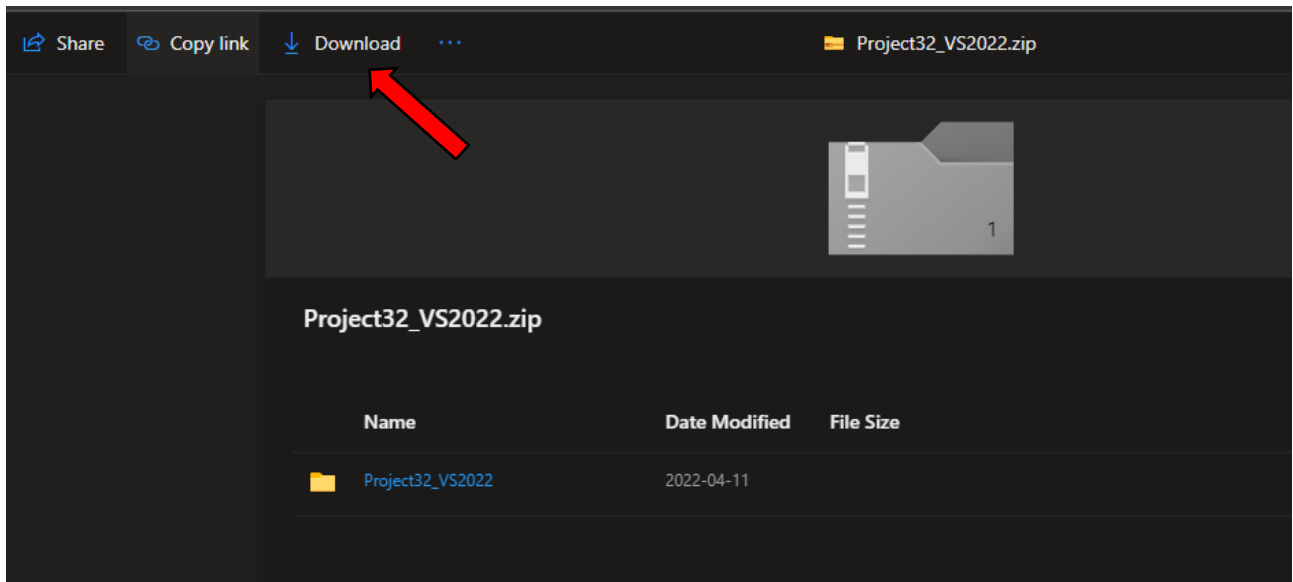
After downloading, extract the zip file and then open the Visual Studio solution file.
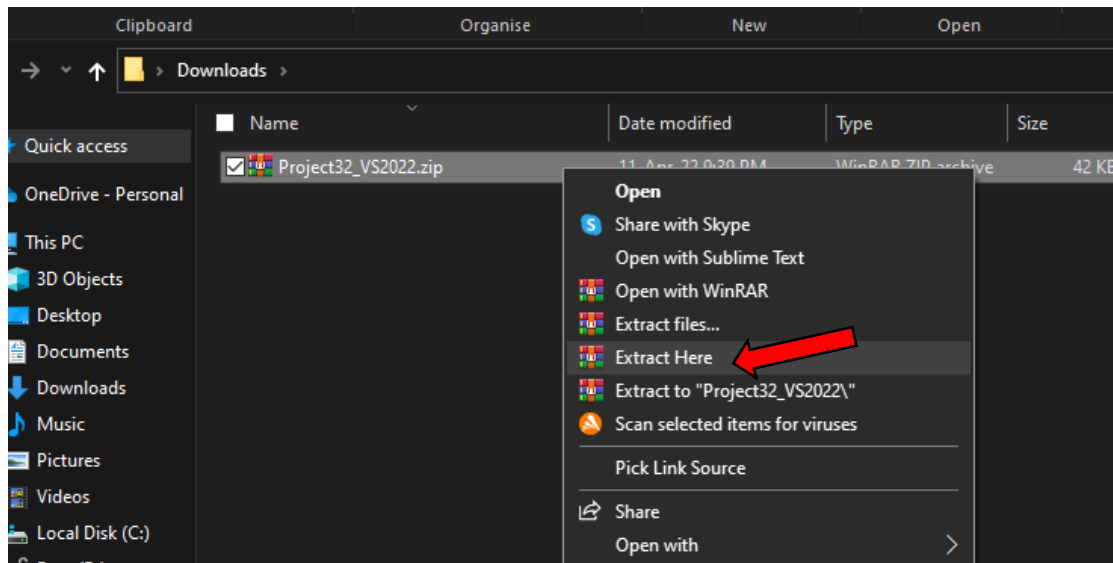
Then open the "Project32_VS2022" folder and open the "Project.sln" file.
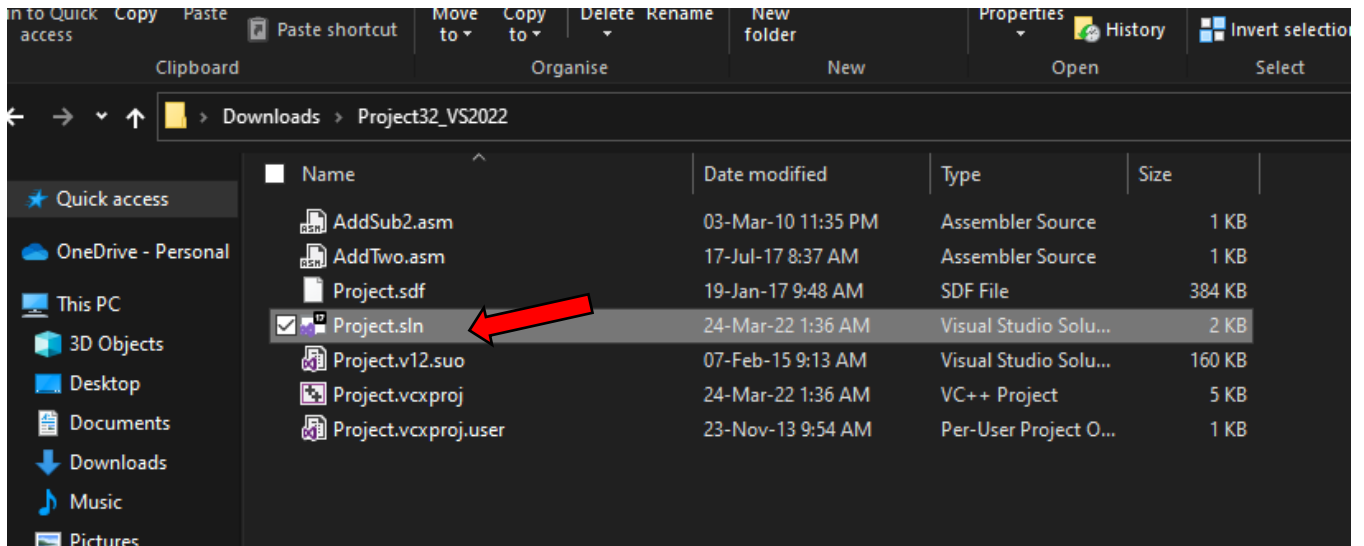
The project will be opened in Visual Studio. Then go to Solution Explorer and open the demo assembly file named "AddTwo.asm".
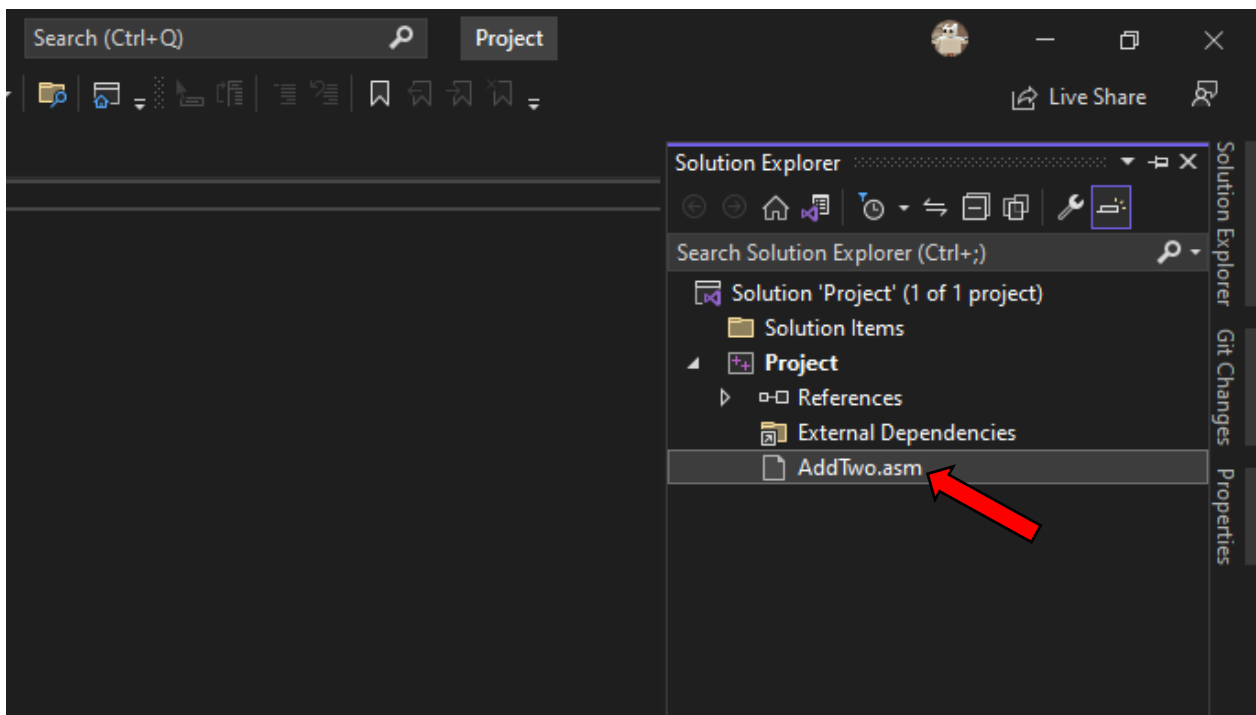
Double click on the "AddTwo.asm" file to open the file. It contains a demo assembly code. Now you can delete the code and paste/write your code here to build and execute your assembly code.

1.4 Debugging your Assembly Code in Visual Studio

First, you must set a breakpoint. When you set a breakpoint in a program, you can use the debugger to execute the program until it reaches the breakpoint. At that point, the program pauses, and the debugger drops into single-step mode. Here's how to do it:
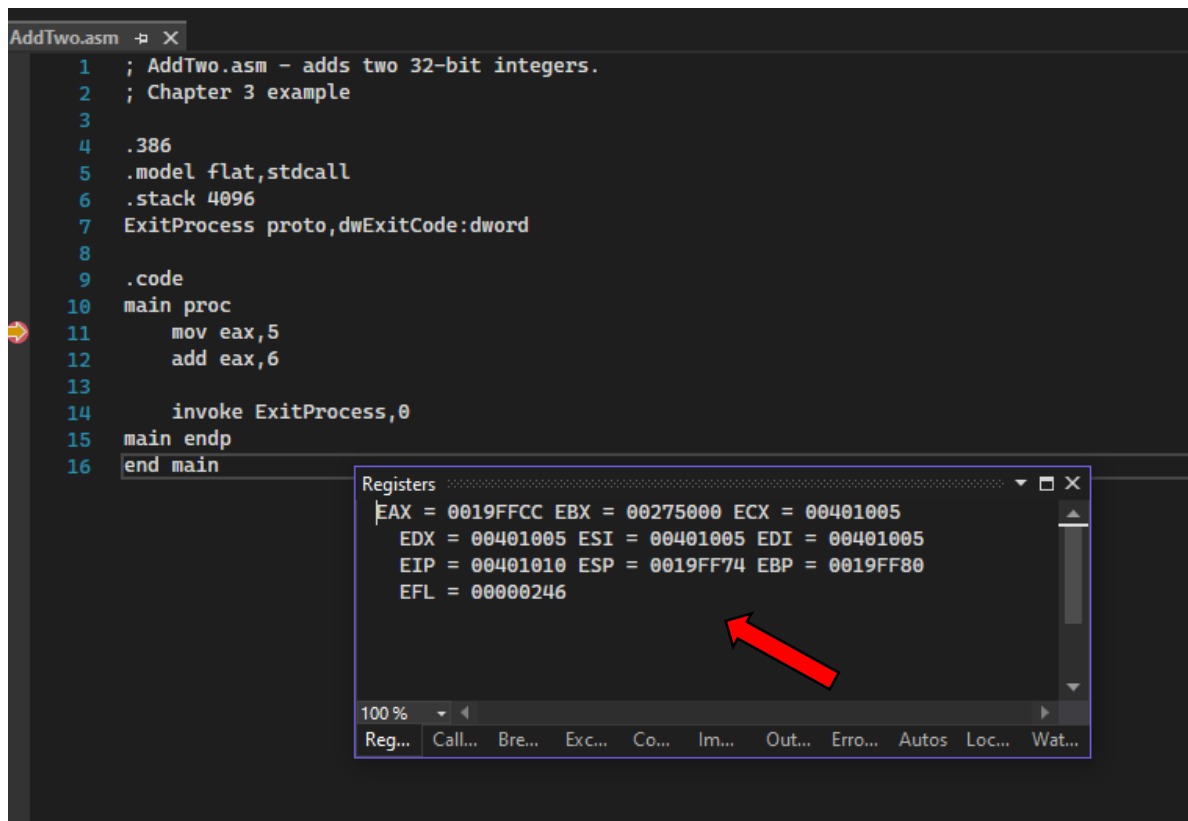
1. Make sure the ASM source code file is open in the editor window.
2. Click the mouse along the border to the left of the **mov eax,5** statement. A large red dot should appear in the margin.
3. Select Start Debugging from the Debug menu. The program should run and pause on the line with the breakpoint.
4. Press the F10 key (called Step Over) to execute the current statement. Continue pressing F10 until the program is about to execute the invoke statement.
5. Press F10 one more time to end the program.

You can remove a breakpoint by clicking its dot with the mouse. Take a few minutes to experiment with the Debug menu commands. Set more breakpoints and run the program again. Here's what your program will look like when paused at the breakpoint:
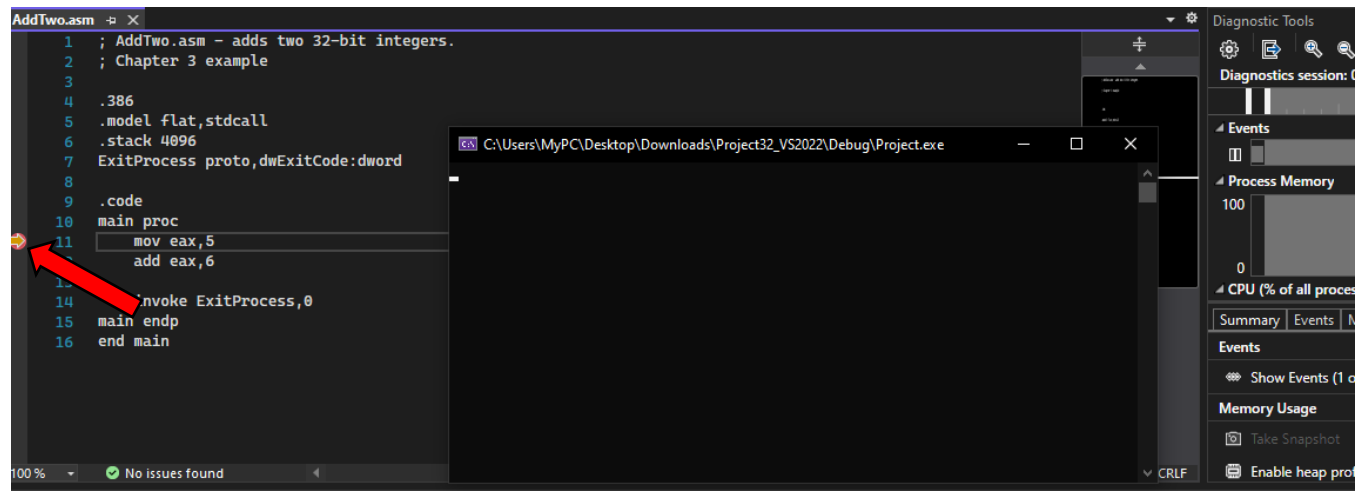
The code will pause at that line and when you press F10 key then the remaining lines will be executed one by one by pressing F10 key again and again.

You can watch the values of the registers during this process. To show the Registers window press "**Alt+Ctrl+G**" to open the window.

```
AddTwo.asm  ⊣ ×
     1    ; AddTwo.asm - adds two 32-bit integers.
     2    ; Chapter 3 example
     3
     4    .386
     5    .model flat,stdcall
     6    .stack 4096
     7    ExitProcess proto,dwExitCode:dword
     8
     9    .code
    10    main proc
⇨   11        mov eax,5
    12        add eax,6
    13
    14        invoke ExitProcess,0
    15    main endp
    16    end main
```

```
Registers                                                    ▼ ☐ ✕
 EAX = 0019FFCC EBX = 00275000 ECX = 00401005
    EDX = 00401005 ESI = 00401005 EDI = 00401005
    EIP = 00401010 ESP = 0019FF74 EBP = 0019FF80
    EFL = 00000246


100 %   ▼ ◀
Reg...  Call... Bre... Exc... Co... Im... Out... Erro... Autos Loc... Wat...
```

In this Registers window you can watch the values of the registers in real-time and understand the behavior and flow of your code.

Note: In the Registers Window, the values of the registers are showing in Hexadecimal form. Each digit of the Hexadecimal represents 4-bit binary data. Hence in 32-bit mode the values of these registers use 8 digits of Hexadecimal (32 binary bits).