CSC 3210

Computer Organization and Programming

Lab Work 12

Dr. Zulkar Nine

mnine@gsu.edu

Georgia State University

Fall 2021

Learning Objective

Integer Arithmetic

Disclaimer

- The process shown in these slides might not work in every single computer due to Operating system version, Microsoft Visual Studio versions and everything.
- If you find any unusual error, you can inform the instructor.
- Instructor will help you resolve the issue.

Attendance!

Lab Work 12 Instructions

- Lab 12 : Expression evaluation

Plan early …

- You have one week time to submit the lab
- Start early
- If you have issues
  - Email TA or instructor
  - Stop by during office hours
- Start working at the last moment is not a good idea.

- AppendixA shows how to check memory data and Appendix B shows how to create a new project.

Problems in this lab

- You might see similar questions in the quizzes and exam.
- During the exam you might need solve similar problems without visual studio.

MUL Instruction

- In 32-bit mode, MUL (unsigned multiply) instruction **multiplies** an 8-, 16-, or 32-bit operand **by** either AL, AX, or EAX.
- The instruction formats are:

754	→	Multiplicand
× 8	→	Multiplier
6032	→	Product

```
MUL  reg/mem8
MUL  reg/mem16
MUL  reg/mem32
```

Multiplier operands

MUL Instruction

- The instruction forma

```
MUL  reg/mem8
MUL  reg/mem16
MUL  reg/mem32
```

Multiplier operands

7 5 4	→	Multiplicand
× 8	→	Multiplier
6 0 3 2	→	Product

Because the destination operand is **twice the size** of the multiplicand and multiplier, overflow **cannot occur**.

Table 7-2 MUL Operands.

Multiplicand	Multiplier	Product
AL	reg/mem8	AX
AX	reg/mem16	DX:AX
EAX	reg/mem32	EDX:EAX

The colon (:) means **concatenation**. This means that **DX** are the bits 16-31 and **AX** are bits 0-15 of the input number

over sized data

MUL Instruction

MUL Examples

The following statements multiply AL by BL, storing the product in AX. The Carry flag is clear (CF = 0) because AH (the upper half of the product) equals zero:

```
mov  al,5h
mov  bl,10h
mul  bl
```

```
; AX = 0050h, CF = 0
```

The following diagram illustrates the movement between registers:

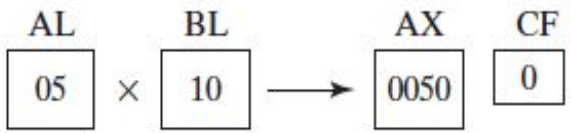


Table 7-2 MUL Operands.

Multiplicand	Multiplier	Product
AL	reg/mem8	AX
AX	reg/mem16	DX:AX
EAX	reg/mem32	EDX:EAX

The Carry flag indicates whether or not the upper half of the product contains **significant digits (?)**

Because the destination operand is **twice the size** of the multiplicand and multiplier, overflow **cannot occur**.

MUL: Example1

- **Example1: 100h * 2000h**, unsigned 16-bit operands:

```
.data
val1 WORD 2000h
val2 WORD 0100h
.code
mov ax,val1
mul val2     ; DX:AX = 00200000h, CF=1
```

Table 7-2 MUL Operands.

Multiplicand	Multiplier	Product
AL	reg/mem8	AX
AX	reg/mem16	DX:AX
EAX	reg/mem32	EDX:EAX

Because the destination operand is **twice the size** of the multiplicand and multiplier, overflow **cannot occur**.

Result = 200000h

The Carry flag indicates whether or not the upper half of the product contains **significant digits (?)**

MUL Instruction

MUL Examples

The following statements multiply AL by BL, storing the product in AX. The Carry flag is clear (CF = 0) because AH (the upper half of the product) equals zero:

```
mov  al,5h
mov  bl,10h
mul  bl
```

```
; AX = 0050h, CF = 0
```

The following diagram illustrates the movement between registers:

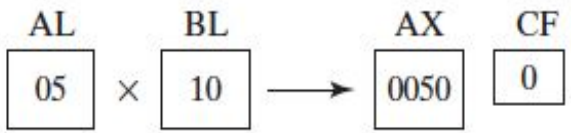


Table 7-2 MUL Operands.

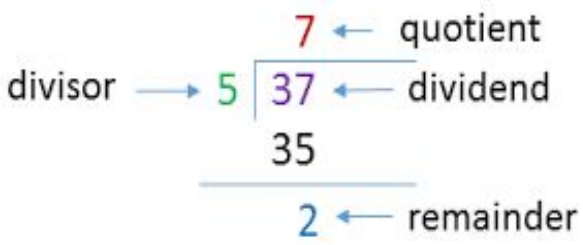
Multiplicand	Multiplier	Product
AL	reg/mem8	AX
AX	reg/mem16	DX:AX
EAX	reg/mem32	EDX:EAX

The Carry flag indicates whether or not the upper half of the product contains **significant digits (?)**

Because the destination operand is **twice the size** of the multiplicand and multiplier, overflow **cannot occur**.

DIV Instruction

- The DIV (unsigned divide) instruction performs 8-bit, 16-bit, and 32-bit division on unsigned integers



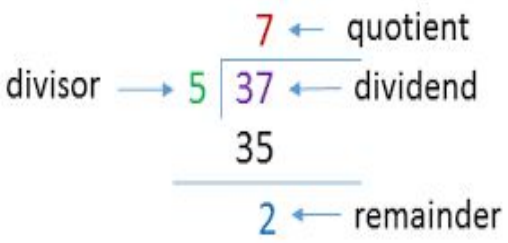
- A **single operand** is supplied (register or memory operand), which is assumed to be the **divisor**
- Instruction formats:

```
DIV reg/mem8
DIV reg/mem16
DIV reg/mem32
```

divisor

DIV Instruction

- The DIV (unsigned divide) instruction performs 8-bit, 16-bit, and 32-bit division on unsigned integers




```
DIV reg/mem8
DIV reg/mem16
DIV reg/mem32
```

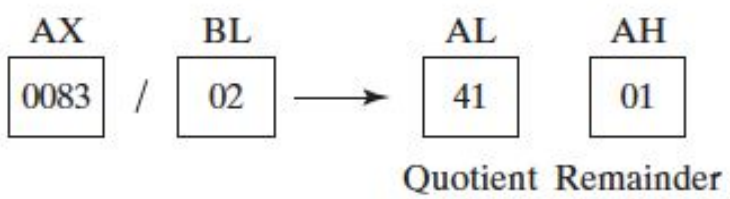
divisor

Dividend	Divisor	Quotient	Remainder
AX	*r/m8*	AL	AH
DX:AX	*r/m16*	AX	DX
EDX:EAX	*r/m32*	EAX	EDX

DIV: Example1

- The following instructions perform 8-bit unsigned division (83h/2),
  - producing a **quotient** of **41h**
  - and a **remainder** of **1**

```
mov ax,0083h        ; dividend
mov bl,2            ; divisor
div bl              ; AL = 41h, AH = 01h
```



Dividend	Divisor	Quotient	Remainder
AX	*r/m8*	AL	AH
DX:AX	*r/m16*	AX	DX
EDX:EAX	*r/m32*	EAX	EDX

Dividend	Divisor	Quotient	Remainder
AX	*r/m8*	AL	AH
DX:AX	*r/m16*	AX	DX
EDX:EAX	*r/m32*	EAX	EDX

Signed Arithmetic Expressions

Var1 = 10
Var2 = 15
Var3 = 5

- **Example1:** **var4 = (var1 / var2) * (var3 – 5)**

  ; Assume **signed** operands 32 bit

Table 7-2 MUL Operands.

Multiplicand	Multiplier	Product
AL	reg/mem8	AX
AX	reg/mem16	DX:AX
EAX	reg/mem32	EDX:EAX

A good reason **for checking the Carry flag after executing MUL** is to know whether the upper half of the product can safely be ignored.

Dividend	Divisor	Quotient	Remainder
AX	*r/m8*	AL	AH
DX:AX	*r/m16*	AX	DX
EDX:EAX	*r/m32*	EAX	EDX

Signed Arithmetic Expressions

- **Example1: var4 = (var1 / var2) * (var3 – 5)**

```
; Assume signed operands 32 bit

mov  eax,var1
CDQ
idiv  var2        ; EAX = var1 / var2
Sub var3, 5                    ; var3 = var3 - 5
imul  var3        ; EAX = EAX * var3
jo   TooBig       ; check for carry
mov  var4,eax       ; save product

TooBig:
```

Table 7-2 MUL Operands.

Multiplicand	Multiplier	Product
AL	reg/mem8	AX
AX	reg/mem16	DX:AX
EAX	reg/mem32	EDX:EAX

A good reason **for checking the Carry flag after executing MUL** is to know whether the upper half of the product can safely be ignored.

Dividend	Divisor	Quotient	Remainder
AX	*r/m8*	AL	AH
DX:AX	*r/m16*	AX	DX
EDX:EAX	*r/m32*	EAX	EDX

Lab 12

Submission

Submission (1)

- **Convert the following pseudo code into assembly code. ('//' represents comments)**

```
array1  13h, 14h, 15h, 16h      // Each element is 1 Byte long
array2 12h, 13h, 14h, 15h     // Each element is 1 Byte long
length1 = number of items in Array1.  // length1 is a Symbolic constant
length2 = number of items in Array2  // length2 is a Symbolic constant
sample1 = 30h       // this variable is 1 byte long initialized with 30
sample2 = 5h.        // this variable is 1 byte long
maxlength = max of length1 and length2. // maxlength is  1 byte long variable
index = 0.   // this is a variable initialized with 0
While (  index < maxlength ){
    If (array1[index] > array2[index]
        exp1 = (array1[index] * sample1) / (array2[index] * Sample2)
    // only store the quotient of the division in exp1 and exp1 is        //16
  bit long variable
    else
    exp1 = 0
    index = index + 1
}
```

Submission (2)

- array1 elements are 1 byte long
- array2 elements are 1 byte long
- length1 and length2 are symbolic constant
- sample1 and sample2 are 1 byte long variables initialized with 30h, and 5h respectively
- maxlength is a 1 byte long variable and uninitialized in the beginning.
  - Inside the code section, compute the largest between length1 and lenght2 and store it in the maxlength
- Index is a variable that is 1 byte long and initialized with 0
- Exp1 is a variable that is 2 byte long.
- You are free to use any register and user defined temporary memory variables to store the values.
- Your code must produce correct results.
- Also handle corner cases (e.g. if the divisor is zero, assign 0 to exp1)

Submission (3)

- Submit the screenshot of your code.
- Debug your code until you reach INVOKE ExitProcess, 0
- Take a screenshot of the watch window showing variable `exp1`.
  - Submit the screenshot.
- Also, Rename the asm file using your last name as Lastname.asm
  - Submit the ASM file as well.

Appendix A

Checking Memory Data

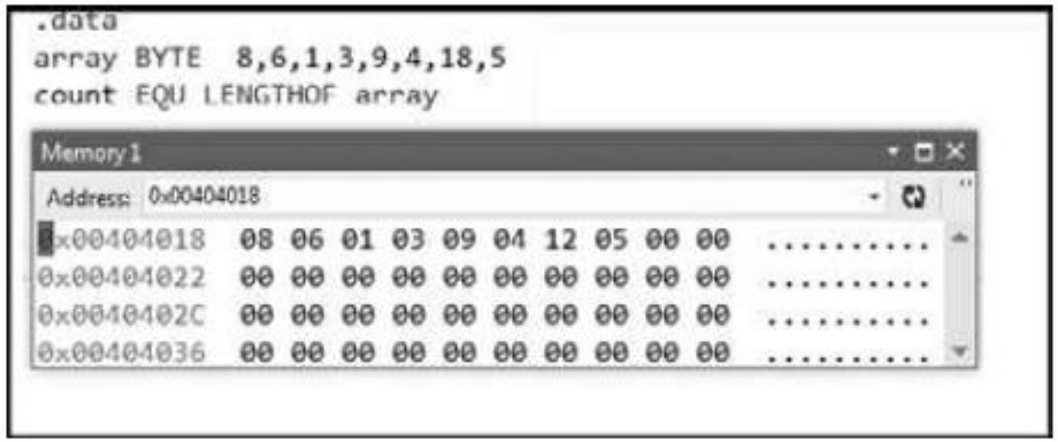
Checking Memory Data

- Use **Memory window** to verify the values of memory locations.
  - **To activate Memory window,** run the debugger, go to debug menu and click on windows, open it, go to Memory then choose **Memory1**.
    - When you run your program and step over every line you will see the changed values marked with red color.

You Must be in the Debugging Mode to see the memory or the register window

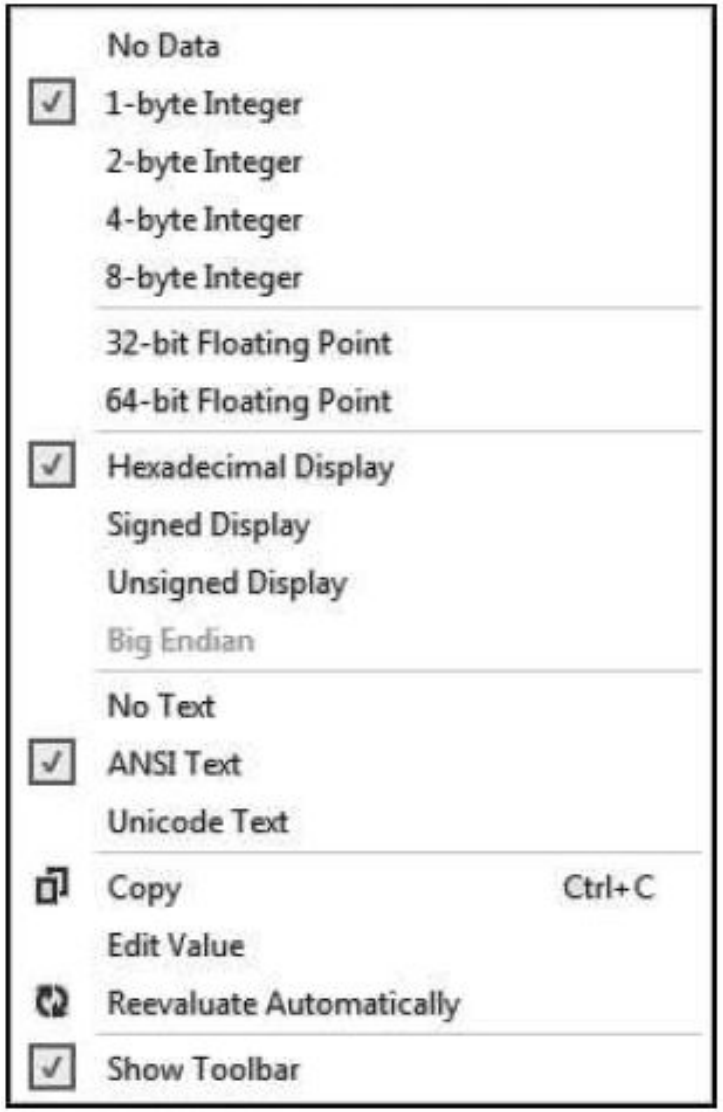
Checking Memory Data

- **To activate Memory window,**
  - if you want to see the location of your variable in the memory,
    - Memory window search box (on the top of the memory window, Address:)
    - write **&** follow it with the variable name: example: **&myVal**l.
    - This will take you to the memory locations of your program (.data section).



Checking Memory Data

- **To activate Memory window,**
  - You can right-click inside the memory window
  - You will see **Popup menu for the debugger's memory window**
  - You can choose how you want to group your bytes: by 1,2,4, or by 8
  - You can also presents data in **hexadecimal**, **signed**, or **unsigned** display



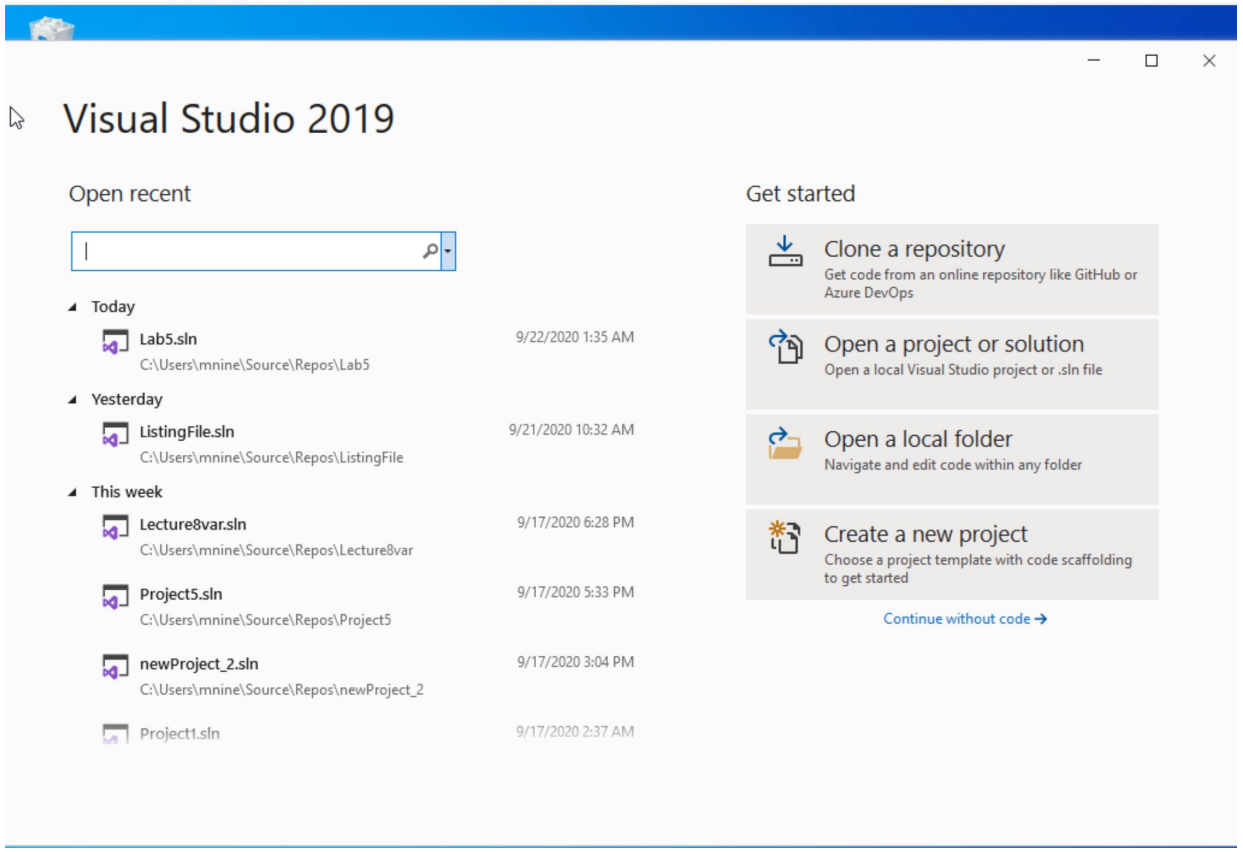
Appendix B

Create a Project

Step 1: Create a project (1)

(1) Start Visual Studio

(2) Click Create a new Project

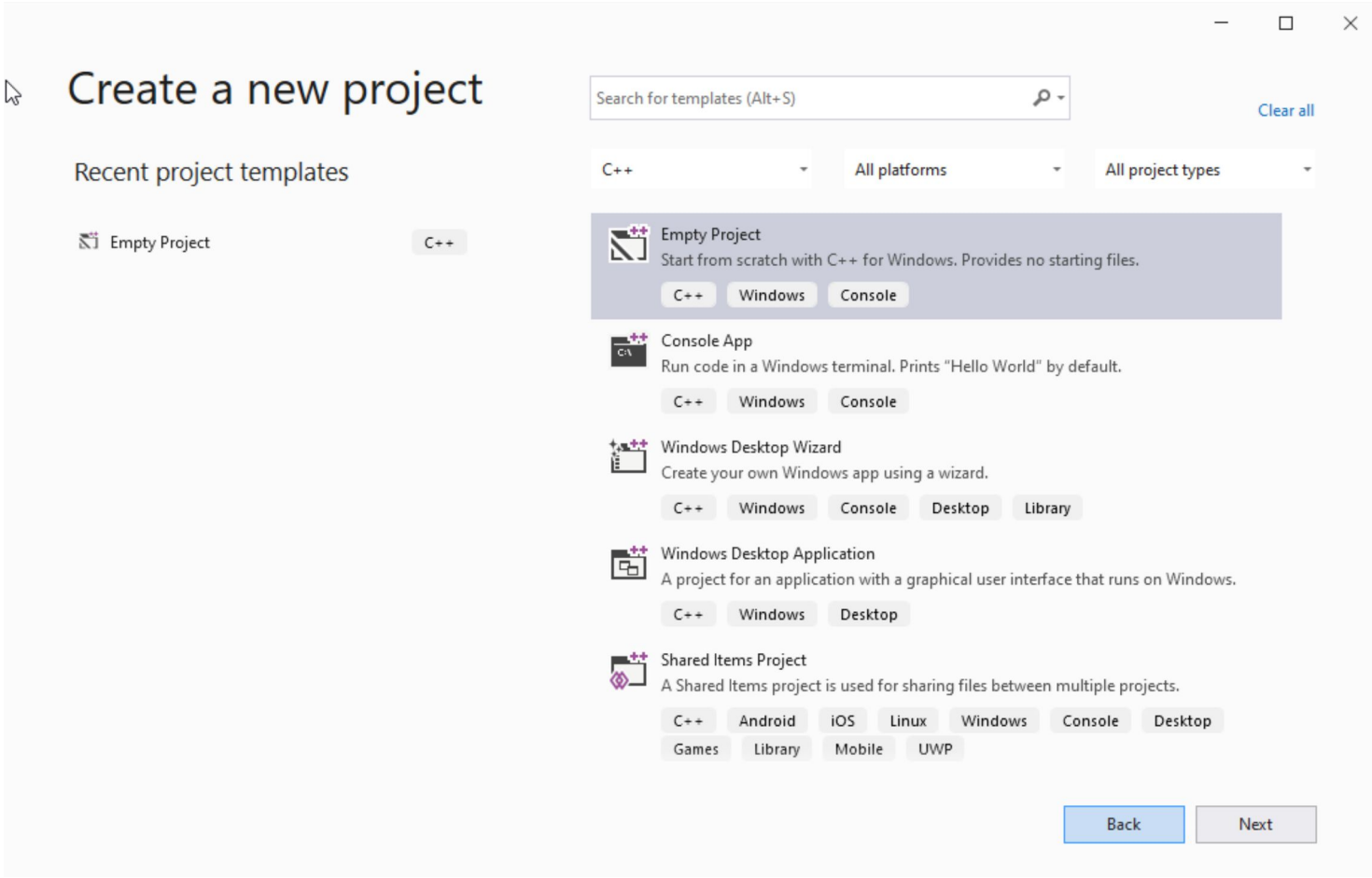


Step 1: Create a project (2)

(1) Select C++ as language

(2) Select Empty Project

(3) Click Next

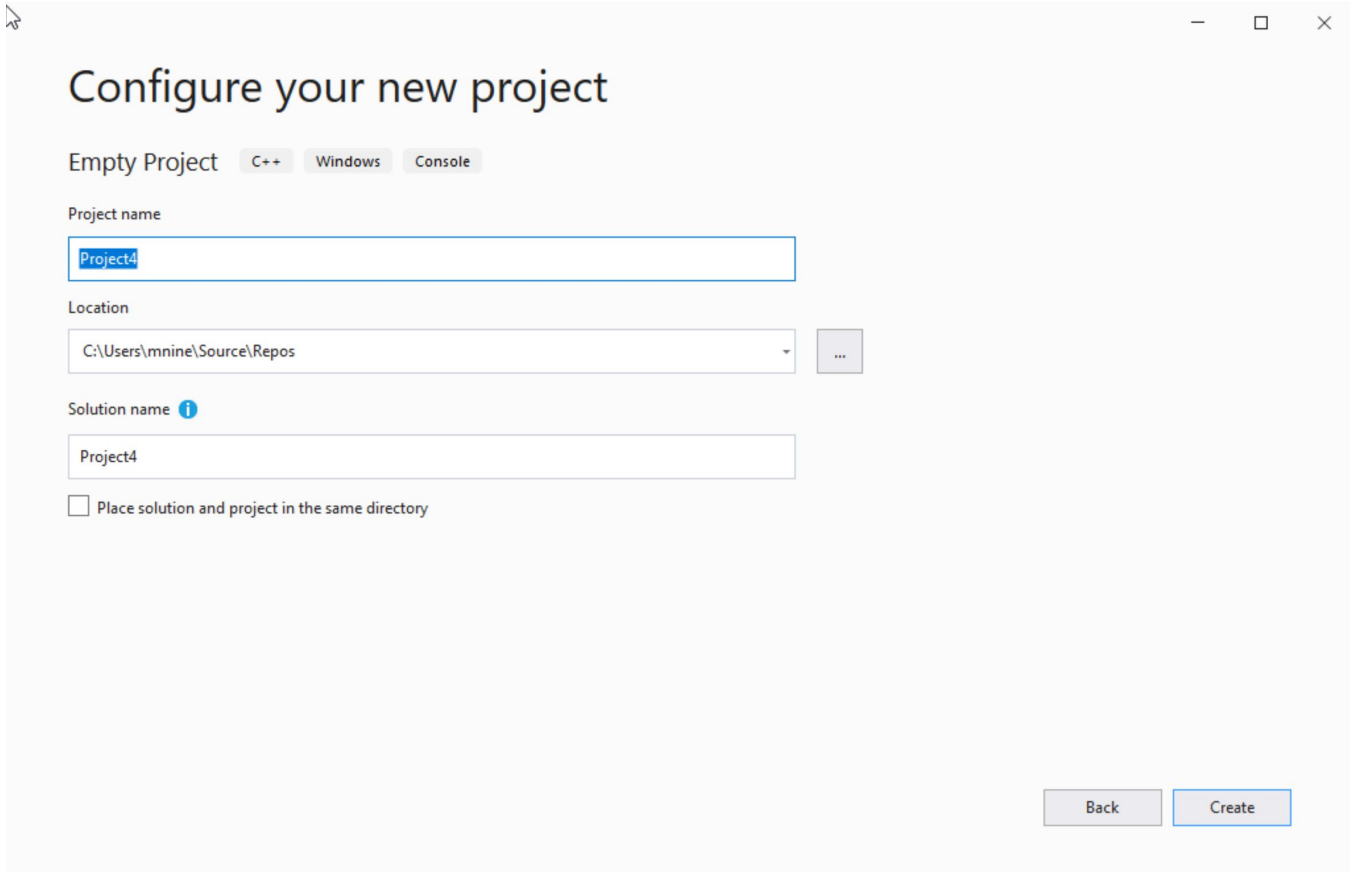


Step 1: Create a project (3)

(1) You can change the project name as you like

(1) Also, you can change the project location

(2) Click Next

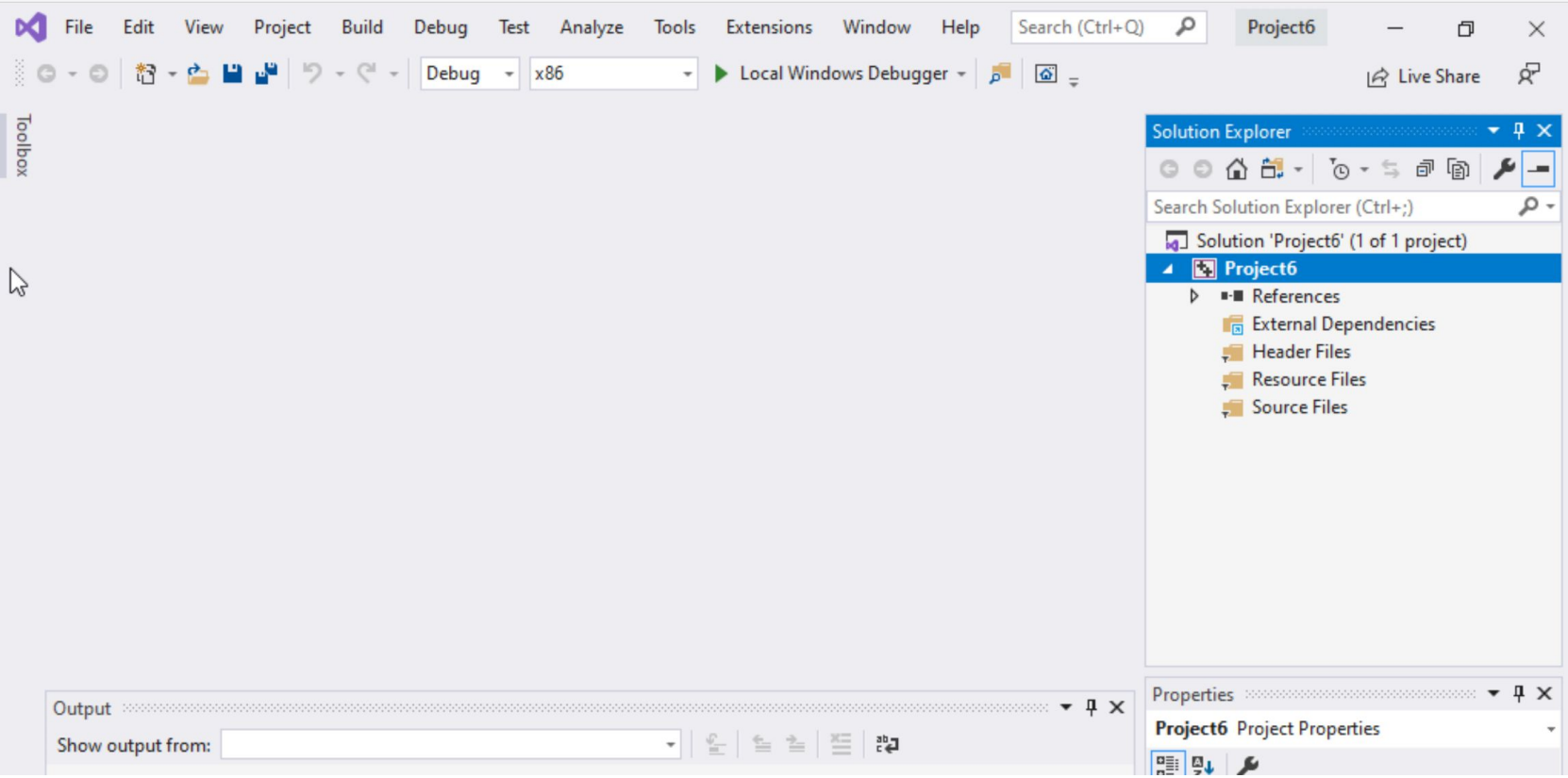


Step 1: Create a project (4)

Delete the

Following folders:

- Header files
- Resources Files, and
- Source Files



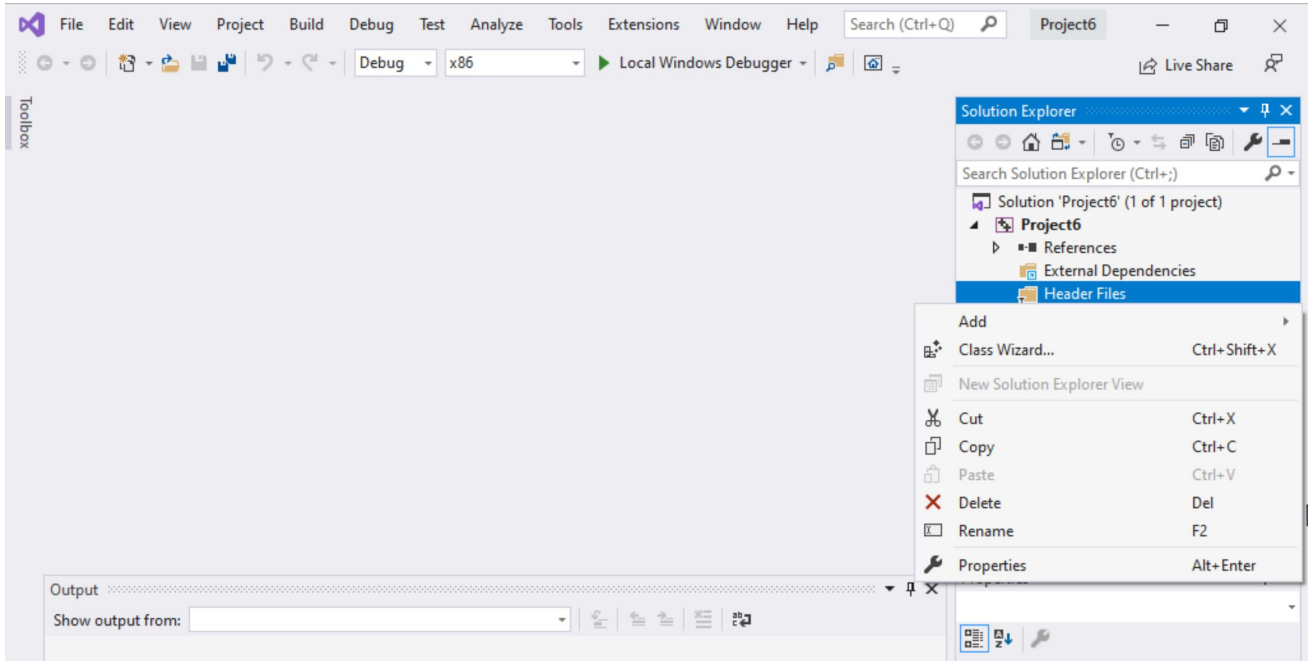
Step 1: Create a project (5)

To delete :

Select the folders

Right click on it

Select delete



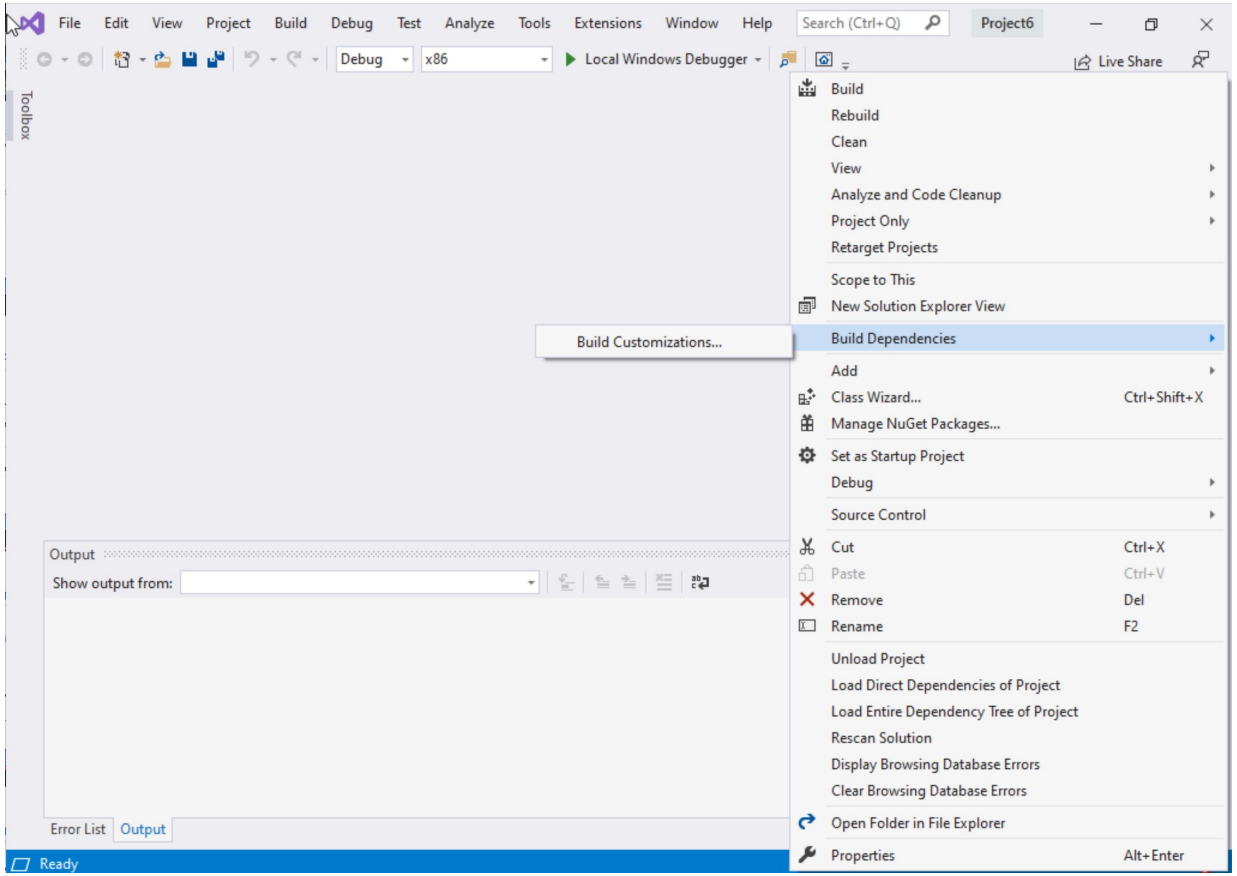
Step 1: Create a project (6)

Select Project Name on solution explorer

Right click on it

Go to Build Dependencies

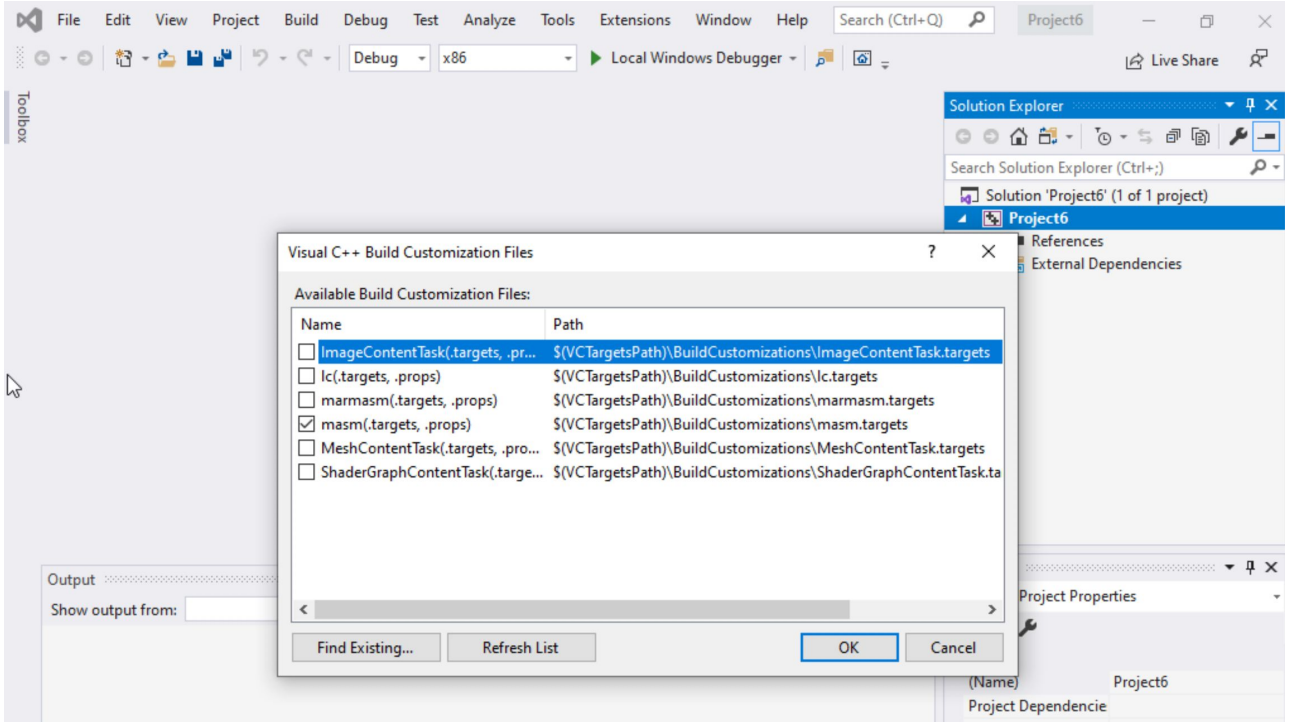
Click on Build Customizations



Step 1: Create a project (7)

Select masm(.target, .props)

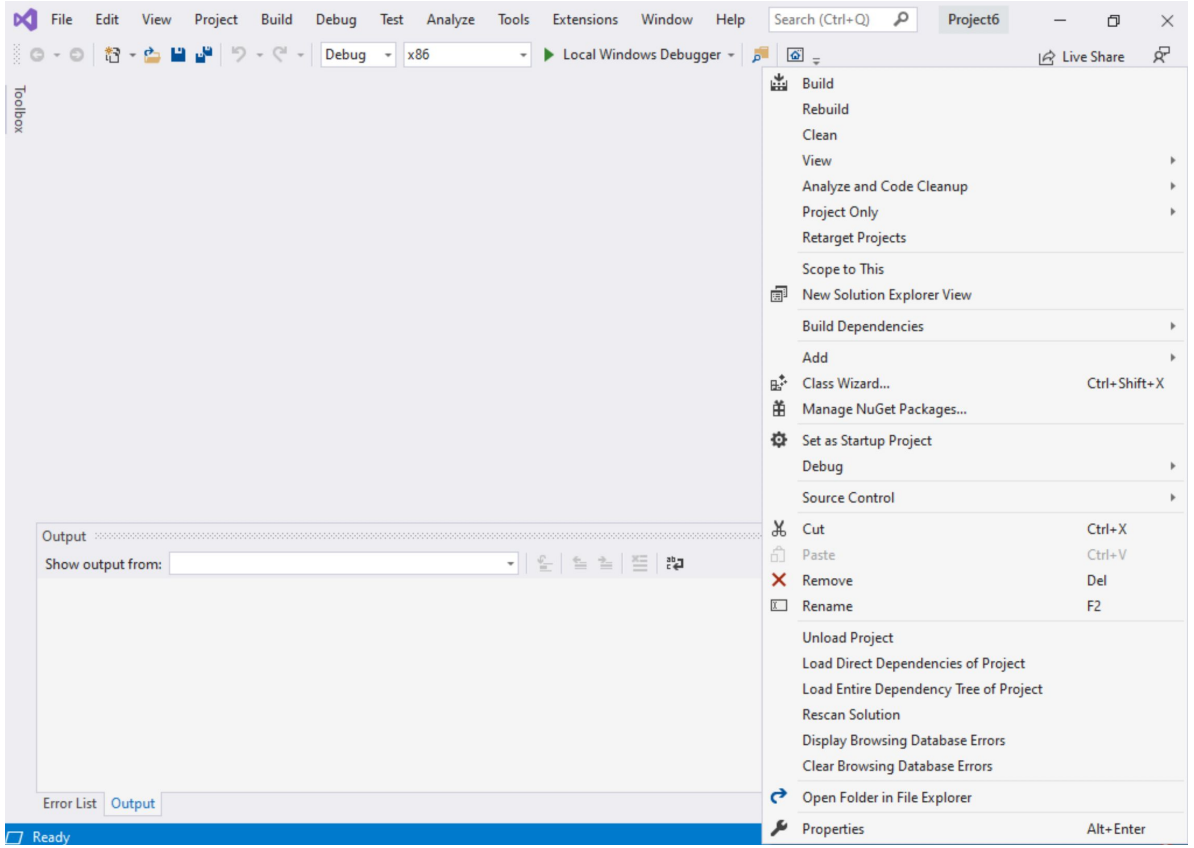
Click ok



Step 1: Create a project (8)

Right click on the Project name in the solution explorer

Click properties



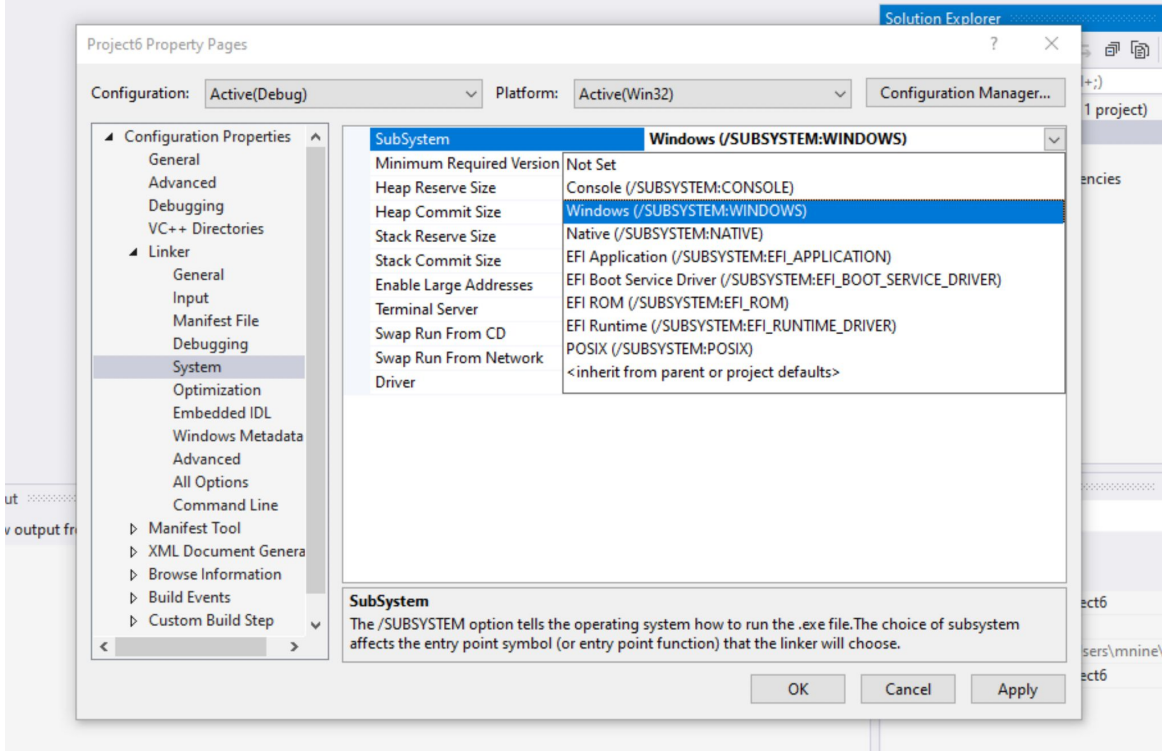
Step 1: Create a project (9)

Expand the ‘Linker’

Select ’System’

Select Windows(/SUBSYSTEM:WINDOWS)

Click OK



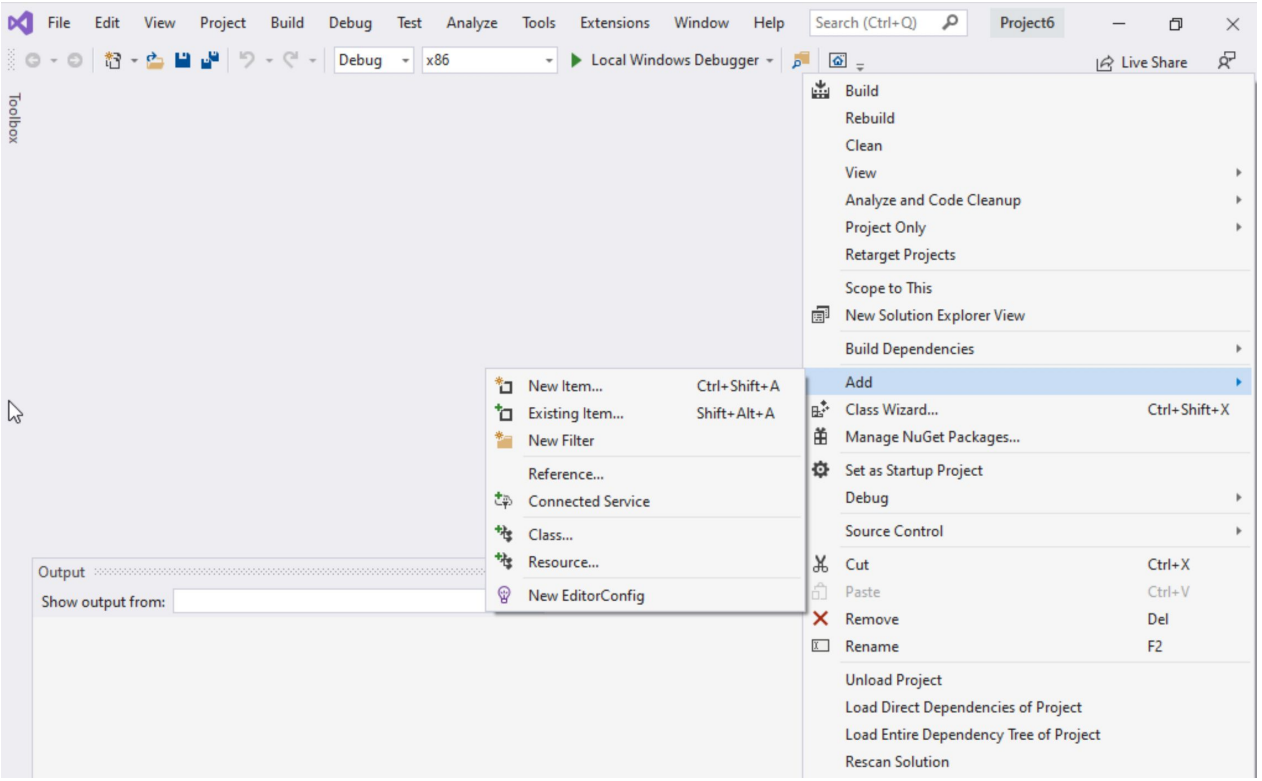
Step 1: Create a project (10)

Select Project name on solution explorer

Right click on it

Expand Add

Choose New Item

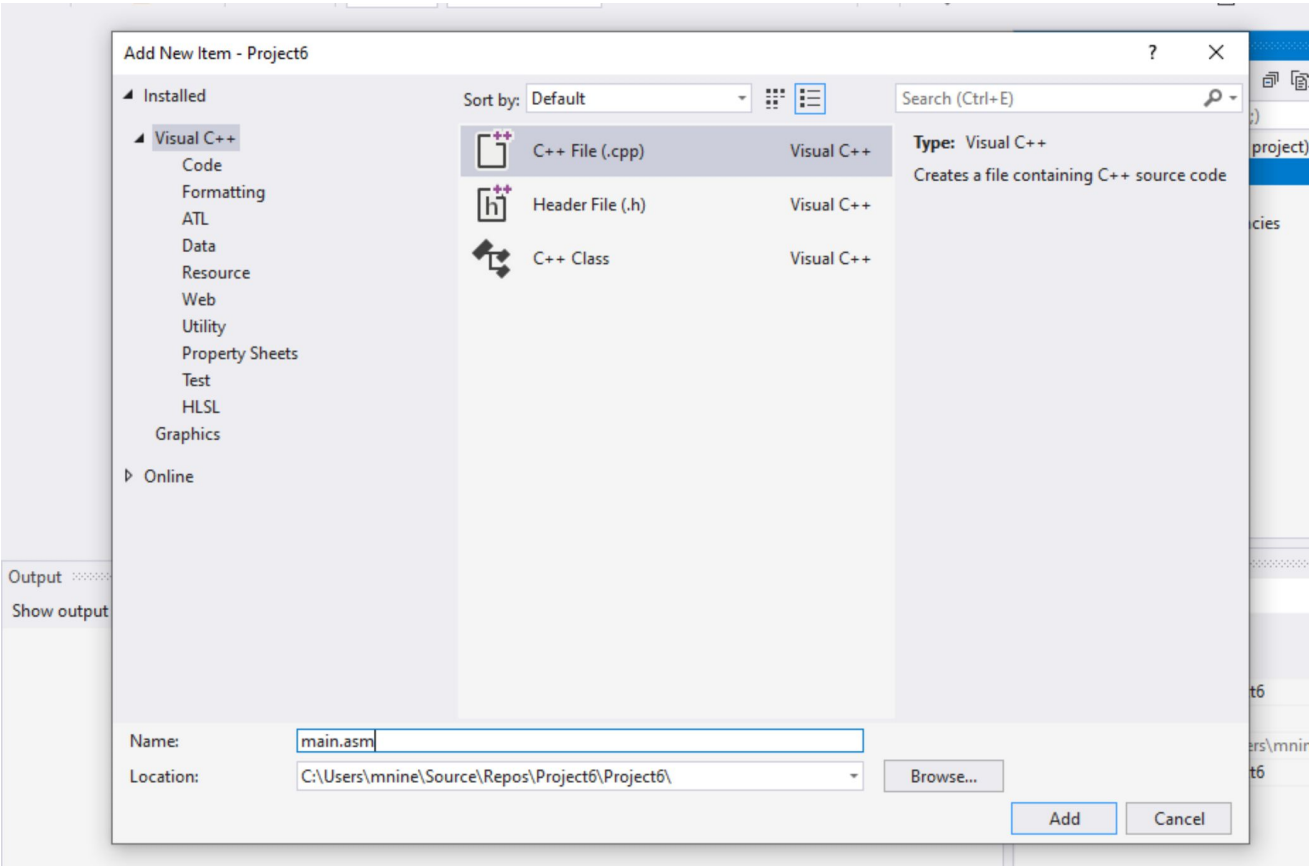


Step 1: Create a project (11)

Select C++ File(.cpp)

Name: main.asm

Click Add



Step 1: Create a project (12)

Select main.asm

Add your code

In the main.asm File.