

Lab Session 12

High Level Language Interface

Objectives

- General Conventions, Model Directive
- Implementing Inline Assembly Code

.Model Directive

.MODEL directive determines

- memory model type
- procedure naming scheme
- parameter passing convention

```
.MODEL memorymodel [,modeloptions]
```

Memory Model

Model	Description
Tiny	A single segment, containing both code and data. This model is used by programs having a .com extension in their filenames.
Small	One code segment and one data segment. All code and data are near, by default.
Medium	Multiple code segments and a single data segment.
Compact	One code segment and multiple data segments.
Large	Multiple code and data segments.
Huge	Same as the large model, except that individual data items may be larger than a single segment.
Flat	Protected mode. Uses 32-bit offsets for code and data. All data and code (including system resources) are in a single 32-bit segment.

Memory Options

Language specifier -> determines calling and naming conventions for procedures and public symbols

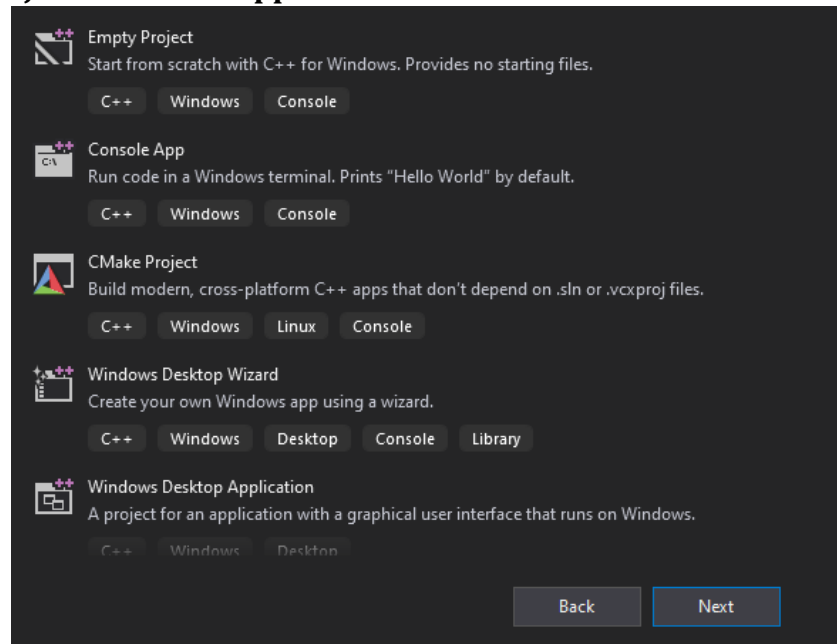
Stack distance -> can be NEARSTACK (the default) or FARSTACK

We mostly uses `.model flat, STDCALL`

STDCALL is the language specifier used when calling MS-Windows functions.

Steps to follow

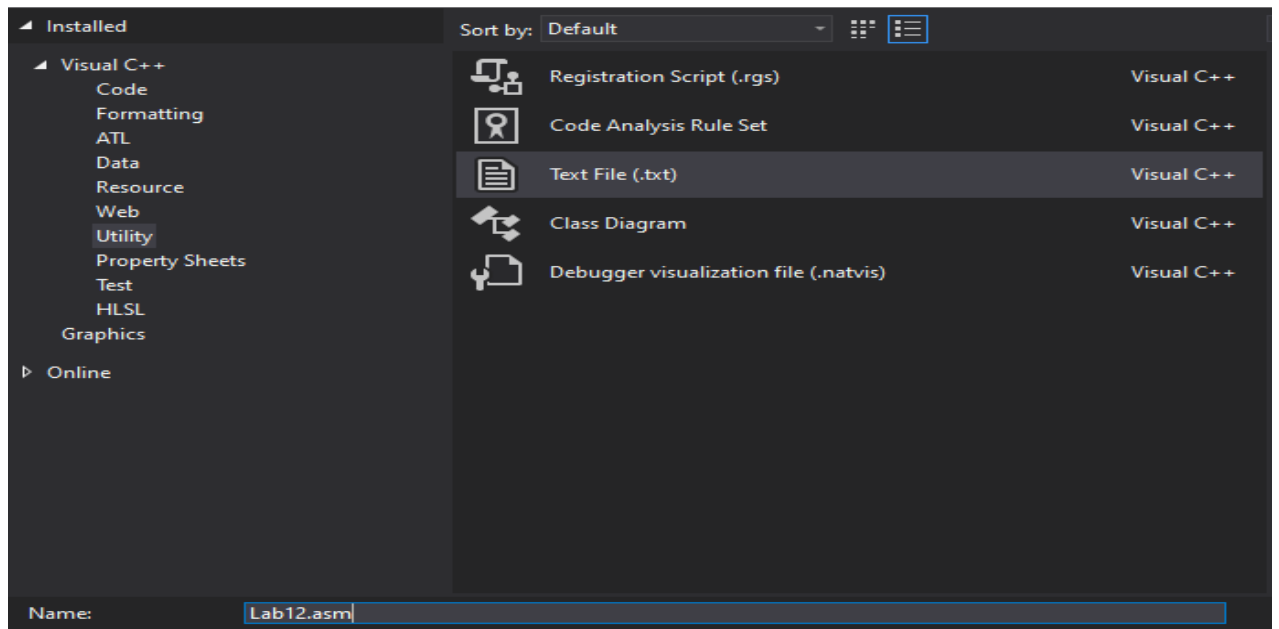
1. Select **New Project > Console Application**



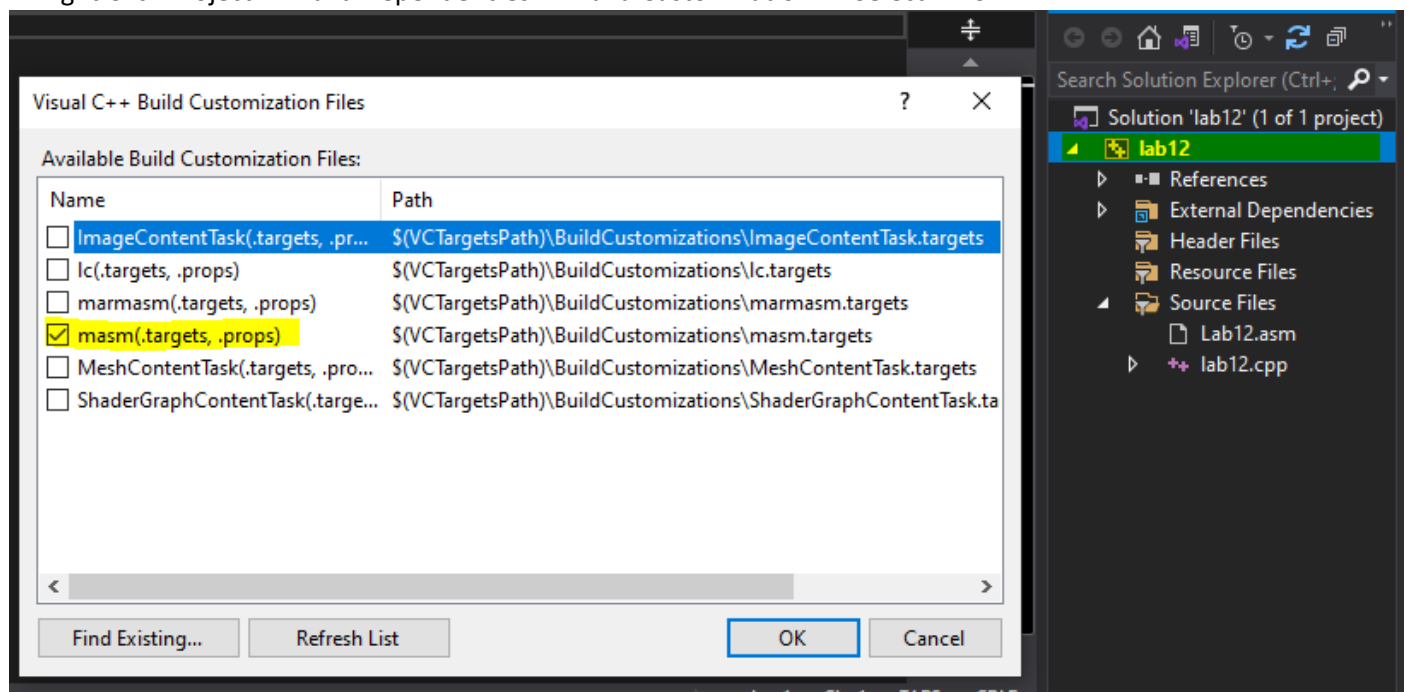
2. Build the example Code.

```
1 // lab12.cpp : This file contains the 'main' function. Program execution begins and ends there.
2 //
3
4 #include <iostream>
5
6 int main()
7 {
8     std::cout << "Hello World!\n";
9 }
10
11 // Run program: Ctrl + F5 or Debug > Start Without Debugging menu
12 // Debug program: F5 or Debug > Start Debugging menu
13
14 // Tips for Getting Started:
15 // 1. Use the Solution Explorer window to add/manage files
16 // 2. Use the Team Explorer window to connect to source control
17 // 3. Use the Output window to see build output and other messages
18 // 4. Use the Error List window to view errors
19 // 5. Go to Project > Add New Item to create new code files, or Project > Add Existing Item to add existing code files to the project
20 // 6. In the future, to open this project again, go to File > Open > Project and select the .sln file
21
```

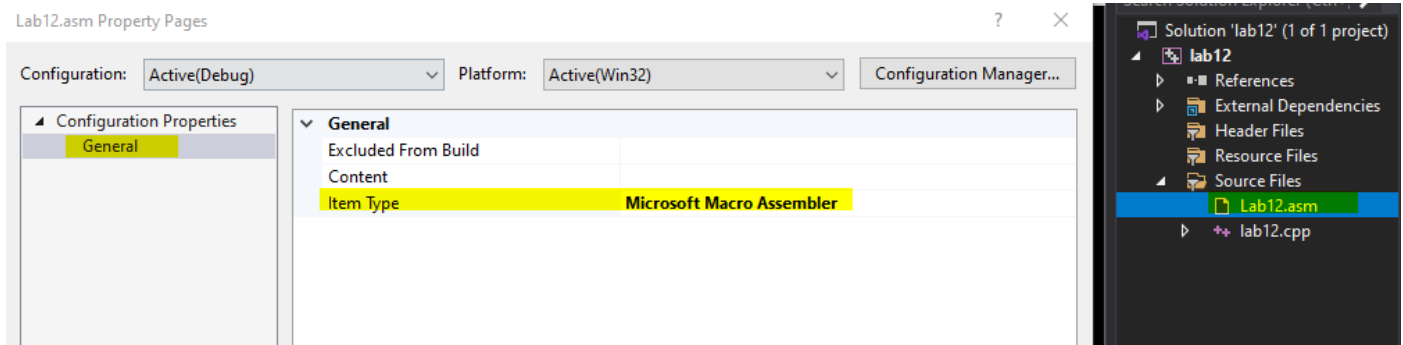
3. Add Source File . Utility >> .txt file. Change the name and extension to .asm.



4. Right Click Project >> Build Dependencies >> Build Customization >> Select MASM.



5. Right Click on .asm File.>> Properties >> General >> Item Type >> MASM.



6. Copy C++ code in .cpp file.

7. Copy Assembly Code in .asm file.

8. Build solution and observe result.

Example:

C++ Code:

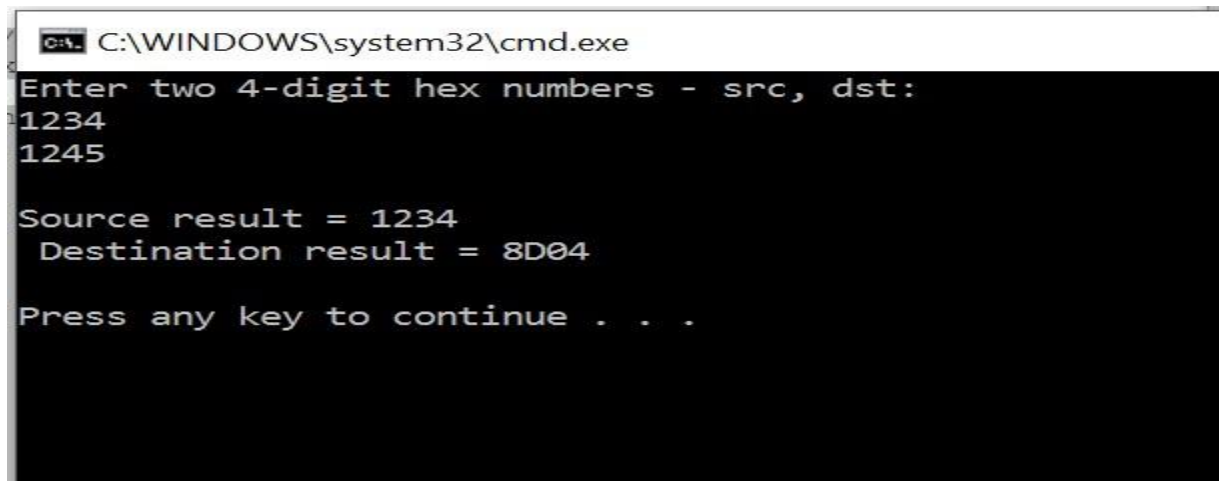
```
#include <stdio.h>
// extern "C" instruct the compiler to use C calling conventions
extern "C" void clear();
int main()
{
    clear();
    //define variables
    unsigned short src_opnd, dst_opnd, src_rslt, dst_rslt;
    printf("Enter two 4-digit hex numbers - src, dst: \n");
    scanf_s("%hX %hX", &src_opnd, &dst_opnd); // in scanf_s it is necessary to
    //specifiy length
    //switch to assembly
    _asm
    {
        MOV AX, src_opnd
        MOV BX, dst_opnd
        SHRD BX, AX, 10; shift AX : BX right 10 bits
        MOV src_rslt, AX
        MOV dst_rslt, BX
    }
    printf("\nSource result = %X\n Destination result = %X\n\n", src_rslt, dst_rslt);
    return 0;
}
```

Assembly Code:

```
.686                                ;Target processor. Use instructions for Pentium class machines
.MODEL FLAT, C                      ;Use the flat memory model. Use C calling conventions
.STACK 2048                         ;Define a stack segment of 1KB (Not required for this example)
.DATA                              ;Create a near data segment. Local variables are declared after
                                ;this directive (Not required for this example)

var_1 dword 10
str_1 byte 50,100,34,5,6,78,12,45,67
str_2 byte 5000 dup(?)
.CODE                               ;Indicates the start of a code segment.
clear PROC
    xor eax, eax
    xor ebx, ebx
    ret
clear ENDP
END
```

Output:



```
C:\WINDOWS\system32\cmd.exe
Enter two 4-digit hex numbers - src, dst:
1234
1245

Source result = 1234
Destination result = 8D04

Press any key to continue . . .
```

ACTIVITIES:

1. Write a program in C++ which takes input from user and contains a procedure in assembly named **ThreeProd** that displays the product of three numeric parameters passed in a function argument.
2. Write a program in C++ which takes input from user and contains a procedure in assembly named **GCD**(Greatest common divisor) which calculates their GCD.