Lab Session 12

High Level Language Interface

Objectives

- General Conventions, Model Directive
- Implementing Inline Assembly Code

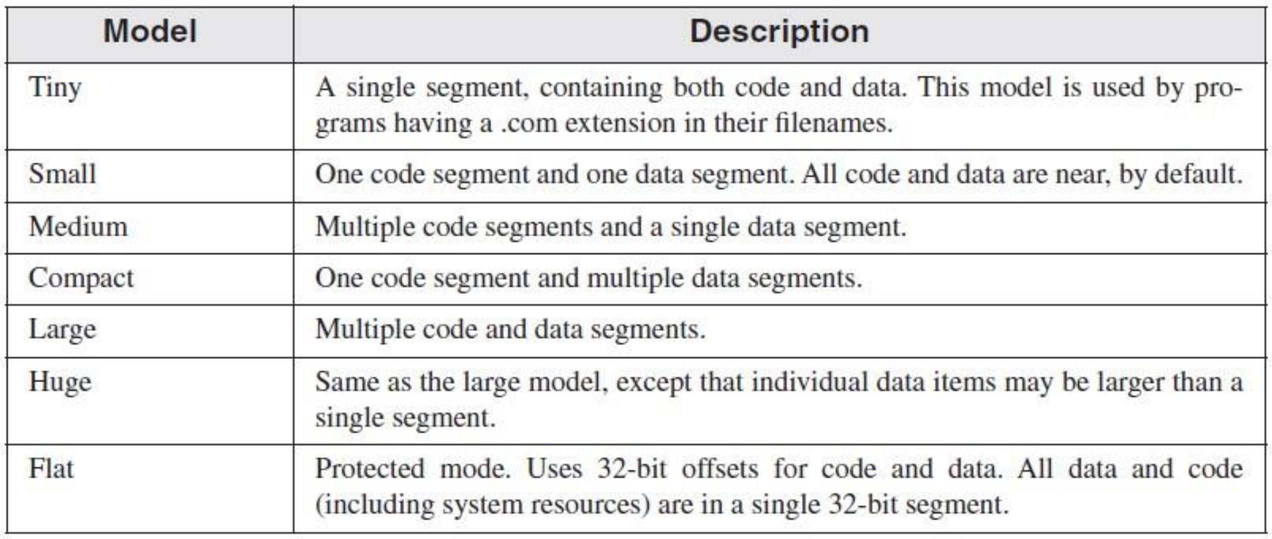
.Model Directive

.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.

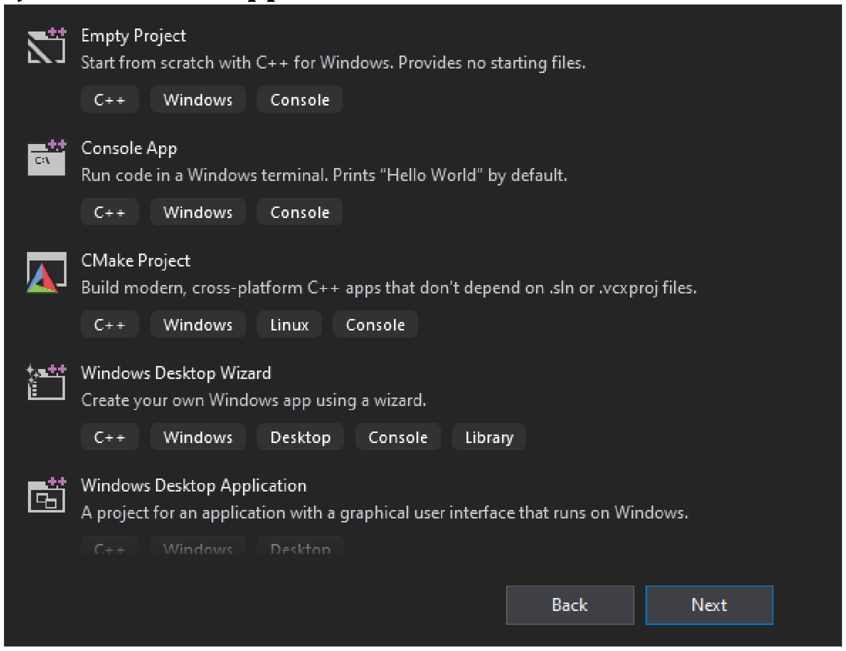
Memory Options

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**


Empty Project
Start from scratch with C++ for Windows. Provides no starting files.

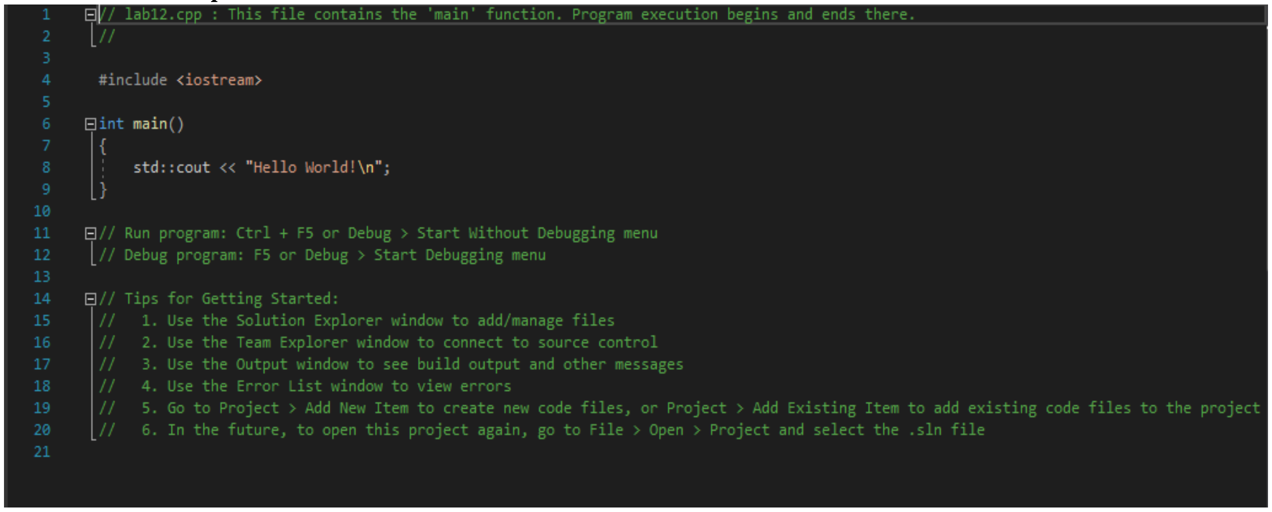
Console App
Run code in a Windows terminal. Prints "Hello World" by default.

CMake Project
Build modern, cross-platform C++ apps that don't depend on .sln or .vcxproj files.

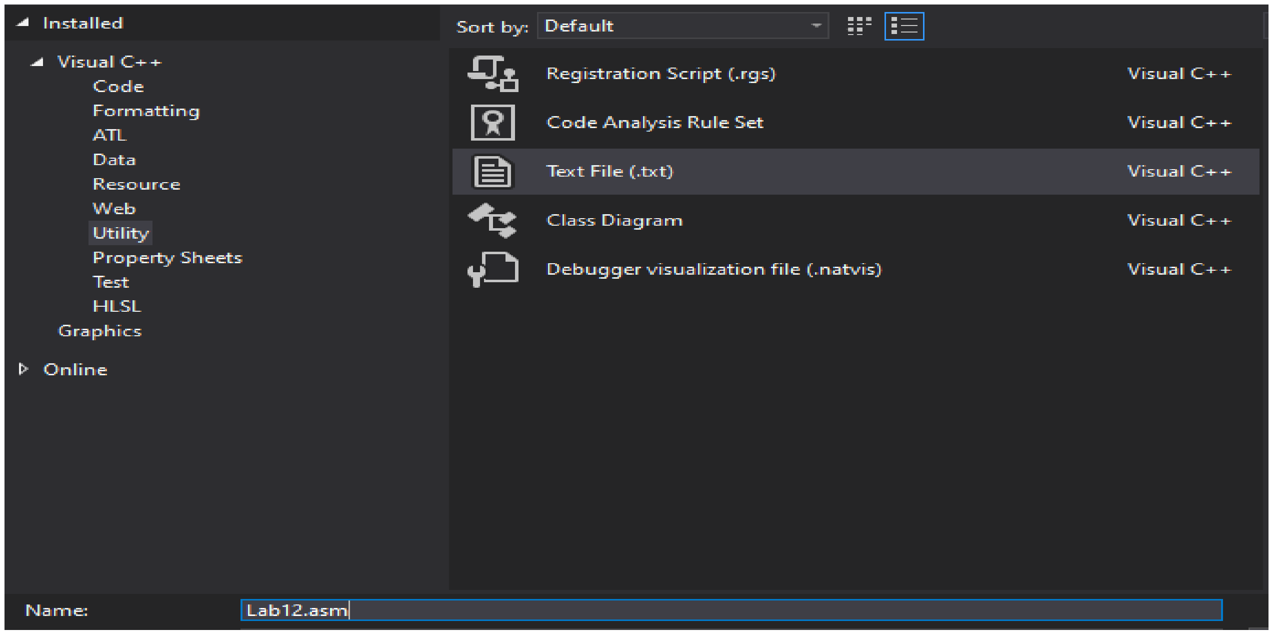
Windows Desktop Wizard
Create your own Windows app using a wizard.

Windows Desktop Application
A project for an application with a graphical user interface that runs on Windows.

2. Build the example Code.


```
// lab12.cpp : This file contains the 'main' function. Program execution begins and ends there.
//

#include <iostream>

int main()
{
    std::cout << "Hello World!\n";
}

// Run program: Ctrl + F5 or Debug > Start Without Debugging menu
// Debug program: F5 or Debug > Start Debugging menu

// Tips for Getting Started:
//   1. Use the Solution Explorer window to add/manage files
//   2. Use the Team Explorer window to connect to source control
//   3. Use the Output window to see build output and other messages
//   4. Use the Error List window to view errors
//   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
//   6. In the future, to open this project again, go to File > Open > Project and select the .sln file

```

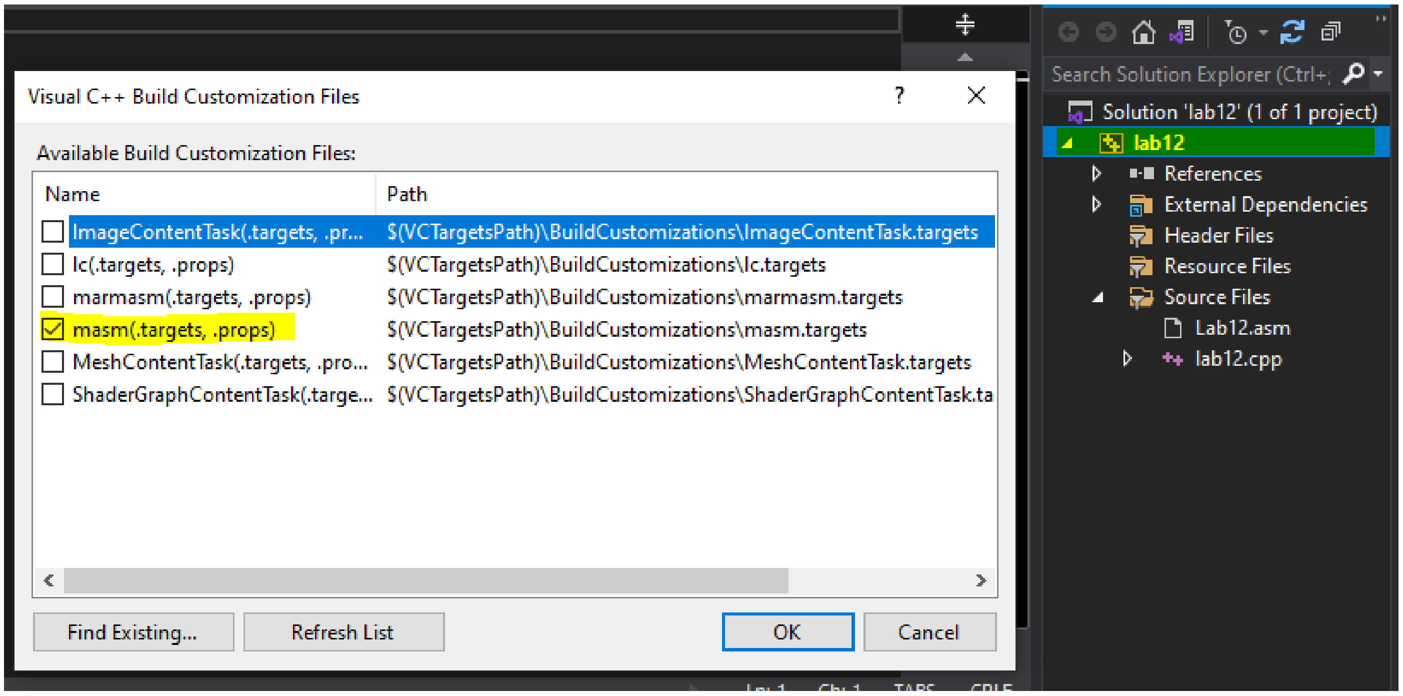
3. Add Source File . Utility >> .txt file. Change the name and extension to .asm.



Registration Script (.rgs)	Visual C++
Code Analysis Rule Set	Visual C++
Text File (.txt)	Visual C++
Class Diagram	Visual C++
Debugger visualization file (.natvis)	Visual C++

Name: Lab12.asm

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

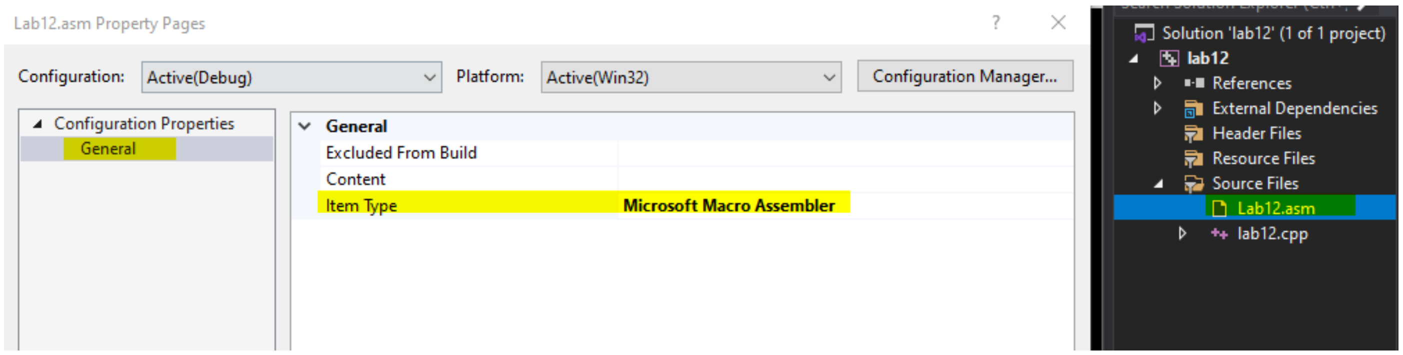


Visual C++ Build Customization Files

Available Build Customization Files:

	Name	Path
☐	ImageContentTask(.targets, .pr...	$(VCTargetsPath)\BuildCustomizations\ImageContentTask.targets
☐	lc(.targets, .props)	$(VCTargetsPath)\BuildCustomizations\lc.targets
☐	marmasm(.targets, .props)	$(VCTargetsPath)\BuildCustomizations\marmasm.targets
✓	masm(.targets, .props)	$(VCTargetsPath)\BuildCustomizations\masm.targets
☐	MeshContentTask(.targets, .pro...	$(VCTargetsPath)\BuildCustomizations\MeshContentTask.targets
☐	ShaderGraphContentTask(.targe...	$(VCTargetsPath)\BuildCustomizations\ShaderGraphContentTask.ta

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



Configuration: Active(Debug)
Platform: Active(Win32)

General	
Excluded From Build	
Content	
Item Type	Microsoft Macro Assembler

6. Copy C++ code in .cpp file.
7. Copy Assembly Code in .asm file.
8. Build solution and observe result.

Example:

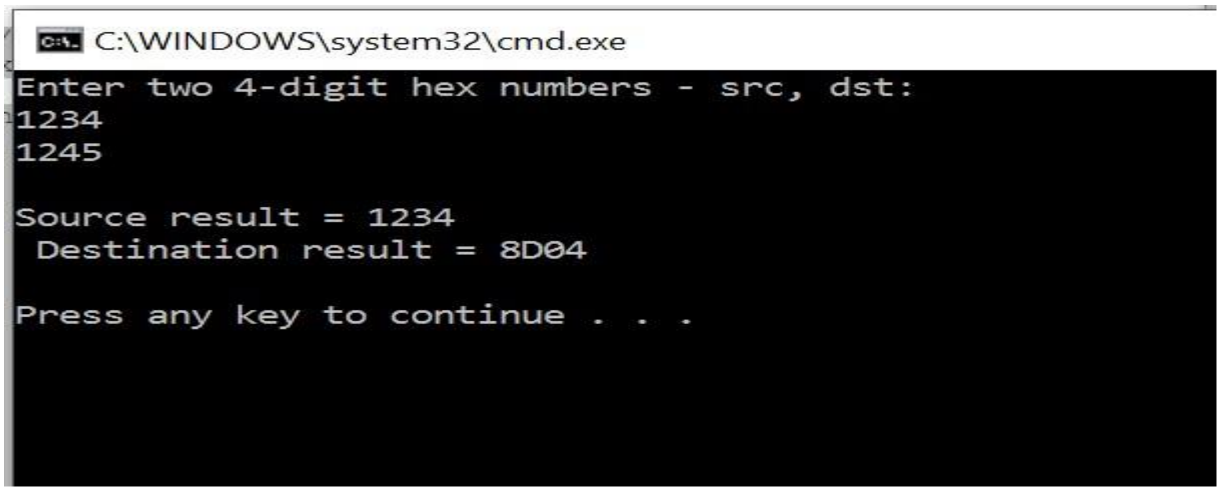
C++ Code:

```
#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 . . .
```

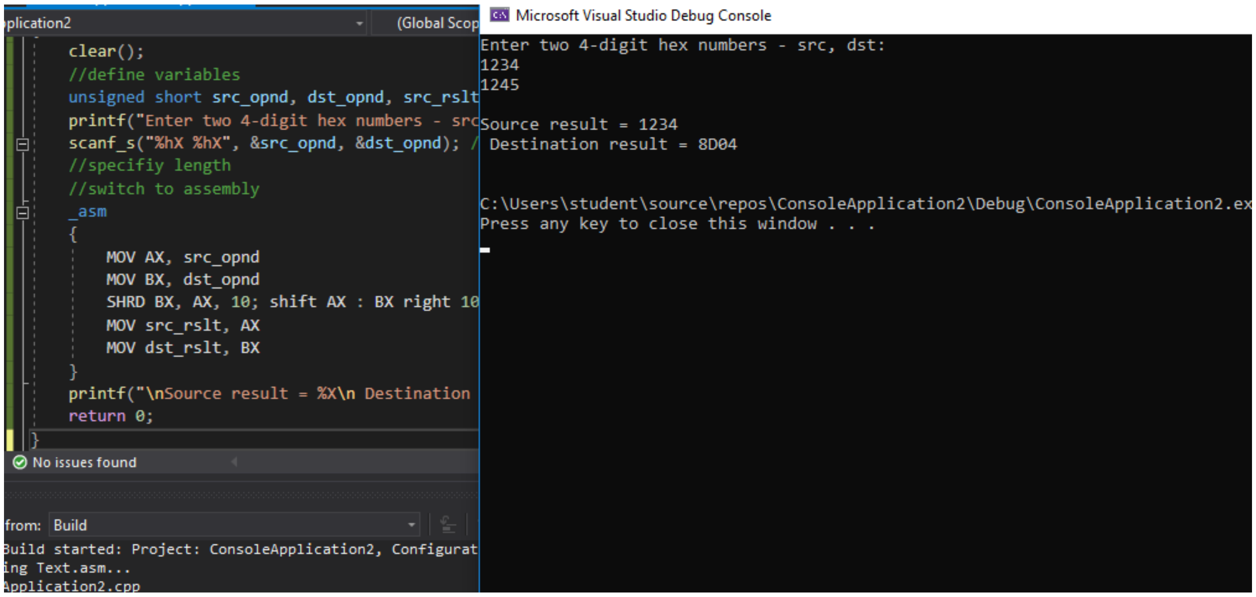
ACTIVITIES:

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.

LAB 12

Q1: Code + Output:



```
plication2                                   (Global Scop
    clear();
    //define variables
    unsigned short src_opnd, dst_opnd, src_rslt
    printf("Enter two 4-digit hex numbers - src
    scanf_s("%hX %hX", &src_opnd, &dst_opnd); /
    //specifiy length
    //switch to assembly
    _asm
    {
        MOV AX, src_opnd
        MOV BX, dst_opnd
        SHRD BX, AX, 10; shift AX : BX right 10
        MOV src_rslt, AX
        MOV dst_rslt, BX
    }
    printf("\nSource result = %X\n Destination
    return 0;
}
```

No issues found

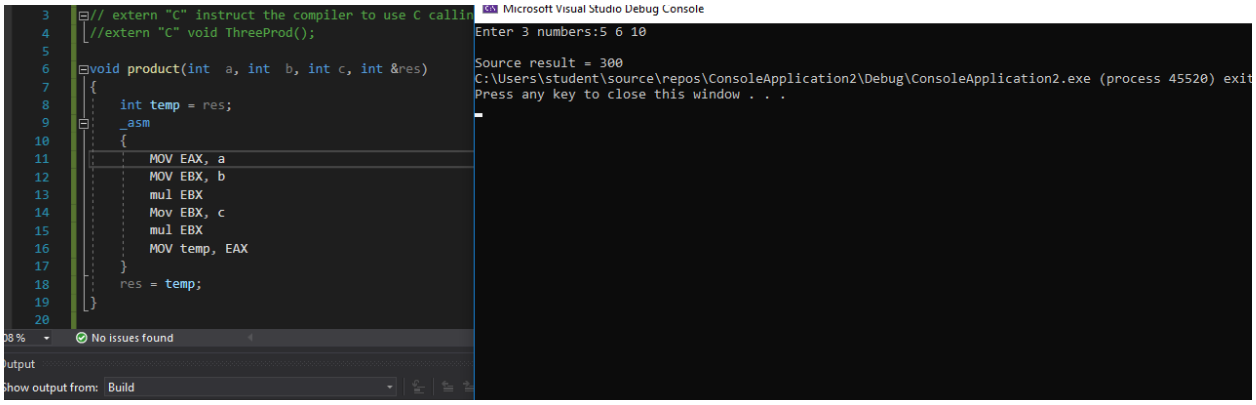
from: Build

```
Build started: Project: ConsoleApplication2, Configurat
ing Text.asm...
Application2.cpp
```

Microsoft Visual Studio Debug Console

```
Enter two 4-digit hex numbers - src, dst:
1234
1245

Source result = 1234
 Destination result = 8D04

C:\Users\student\source\repos\ConsoleApplication2\Debug\ConsoleApplication2.ex
Press any key to close this window . . .
```

Q2: Code + Output



```
// extern "C" instruct the compiler to use C callin
//extern "C" void ThreeProd();

void product(int  a, int  b, int c, int &res)
{
    int temp = res;
    _asm
    {
        MOV EAX, a
        MOV EBX, b
        mul EBX
        Mov EBX, c
        mul EBX
        MOV temp, EAX
    }
    res = temp;
}
```

No issues found

utput

how output from: Build

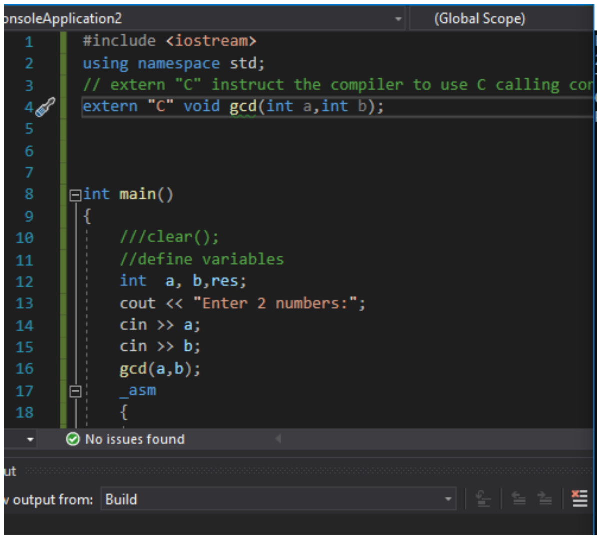
Microsoft Visual Studio Debug Console

```
Enter 3 numbers:5 6 10

Source result = 300
C:\Users\student\source\repos\ConsoleApplication2\Debug\ConsoleApplication2.exe (process 45520) exit
Press any key to close this window . . .
```

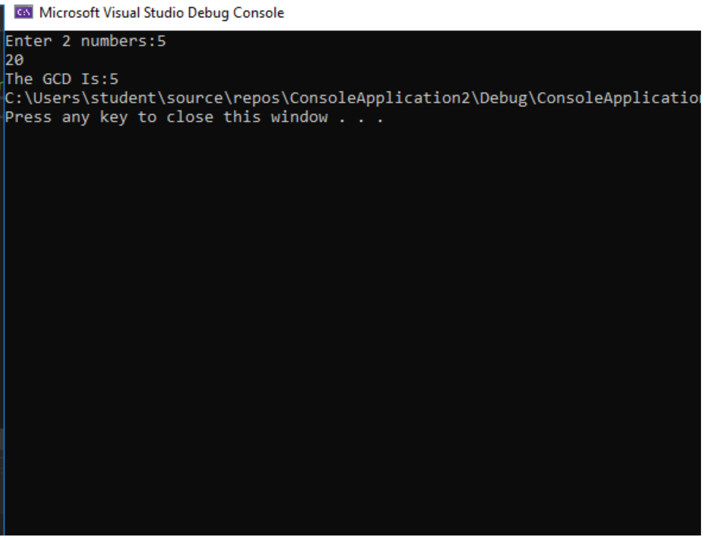
Mohsin Ali Mirza **k200353** **3E-BSCS**

LAB 12

Q3: Code + Output



```
#include <iostream>
using namespace std;
// extern "C" instruct the compiler to use C calling co
extern "C" void gcd(int a,int b);



int main()
{
    ///clear();
    //define variables
    int  a, b,res;
    cout << "Enter 2 numbers:";
    cin >> a;
    cin >> b;
    gcd(a,b);
    _asm
    {
```



```
Enter 2 numbers:5
20
The GCD Is:5
C:\Users\student\source\repos\ConsoleApplication2\Debug\ConsoleApplicatio
Press any key to close this window . . .
```