**Lab Session 11**



**High Level Language Interface**



**Objectives**

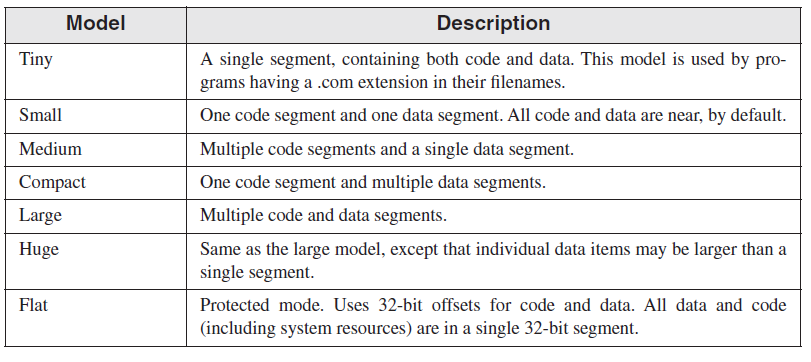
* General Conventions, Model Directive
* Implementing Inline Assembly Code

**.Model Directive**

.MODEL directive determines

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

**Memory Model**



**Memory Options**

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

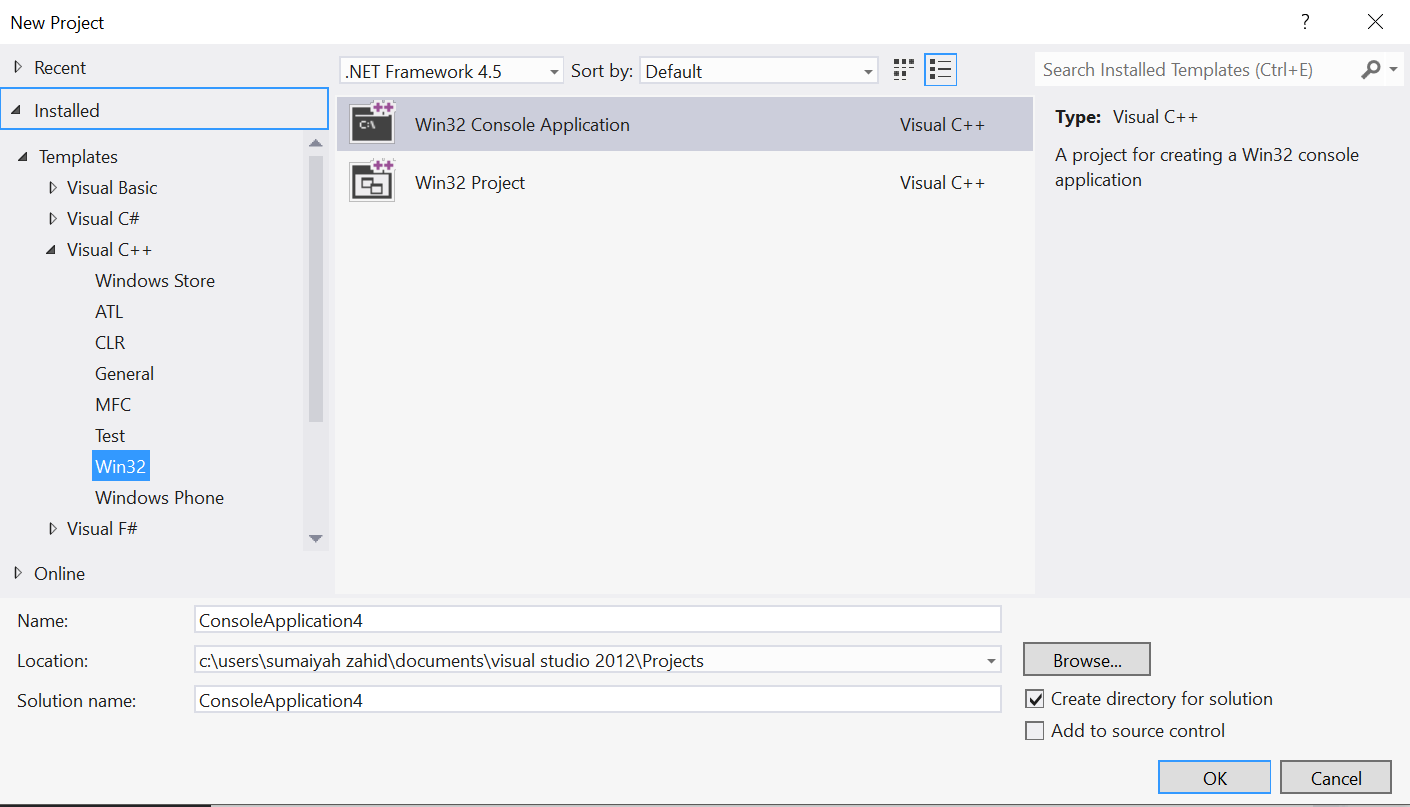
Stack distance -> can be NEARSTACK (the default) or FARSTACK

We mostly uses *.model flat, STDCALL*

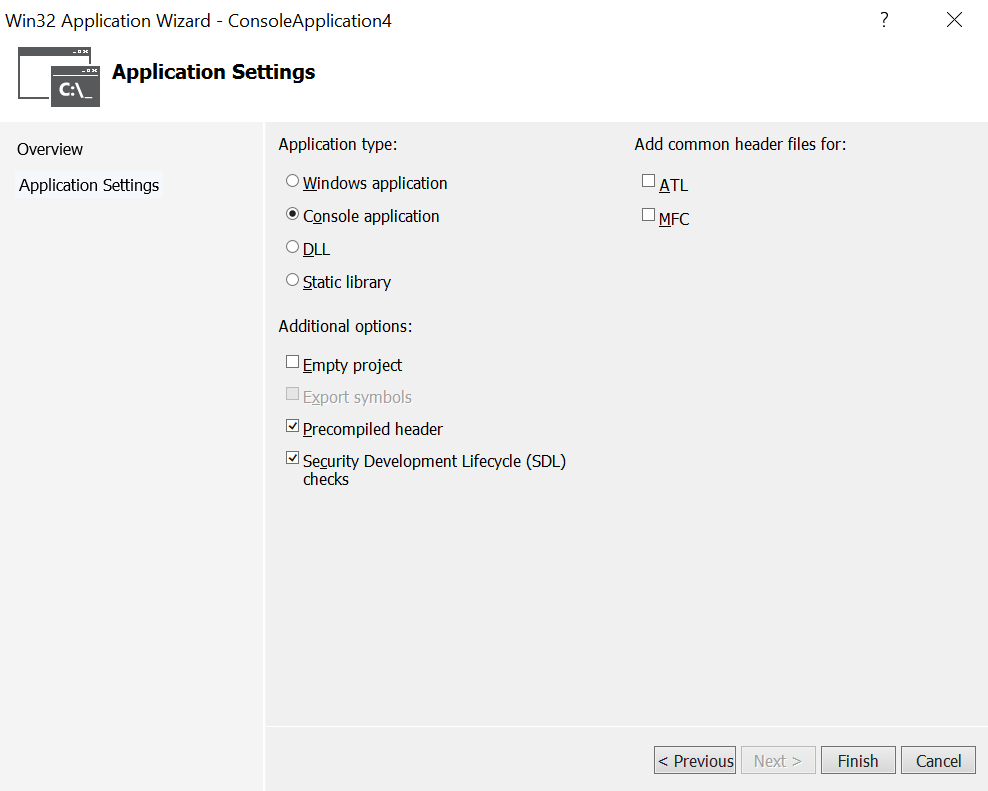
**STDCALL** is the language specifier used when calling MS-Windows functions.

**Steps to follow**

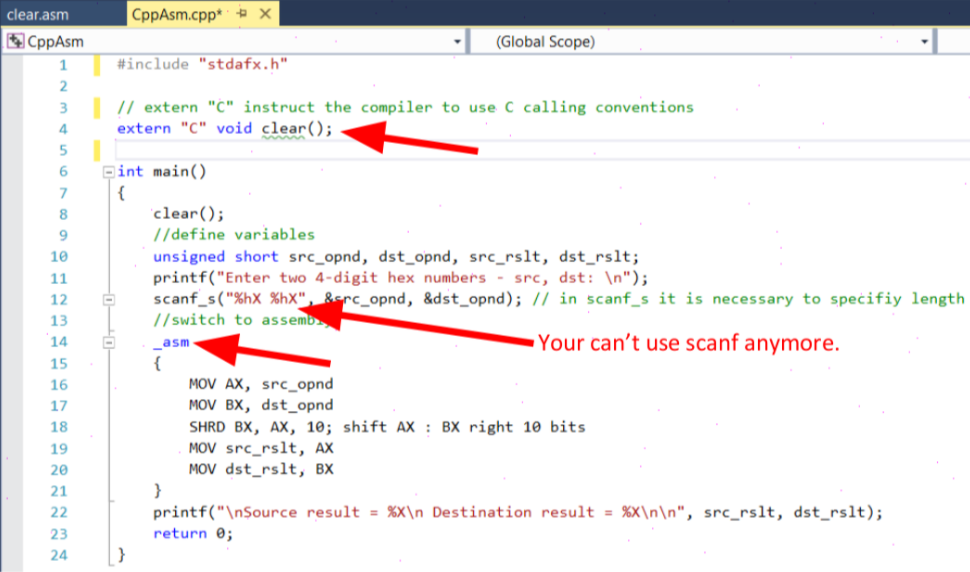
1. Select **New Project > Visual C++ > Win32 >Win32 Console Application**



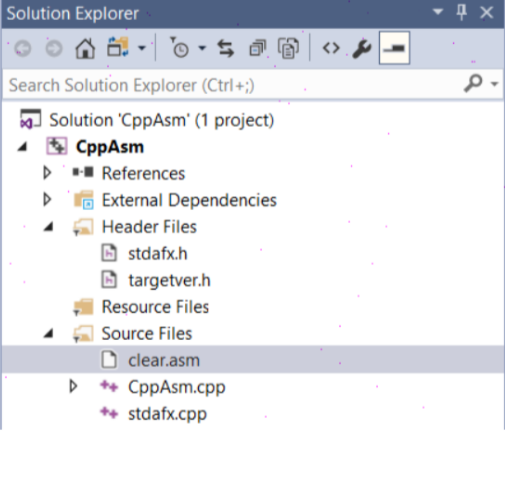
1. Make sure to check Pre Compiled Header



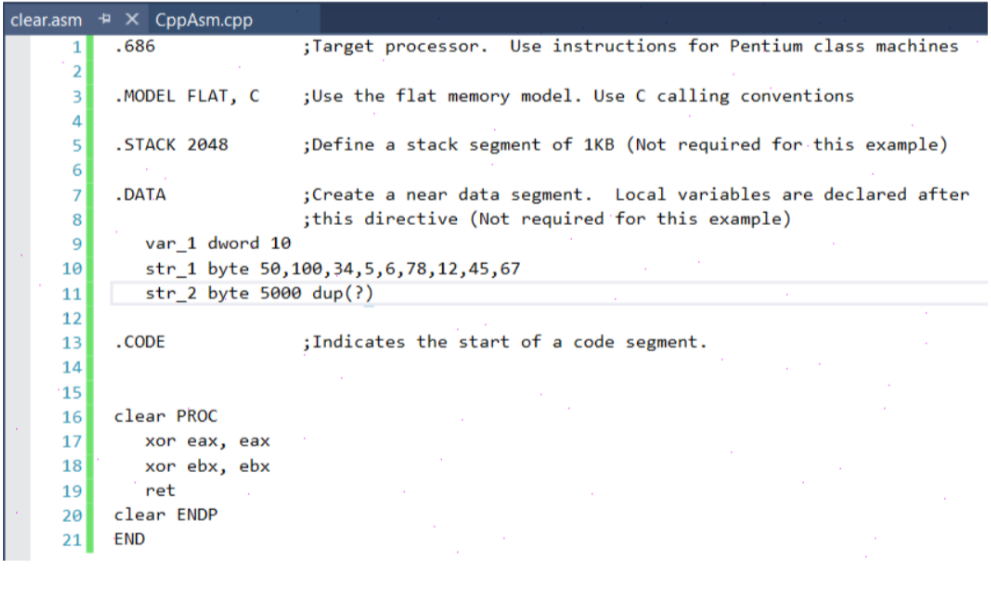
1. Write your C++ code in source file



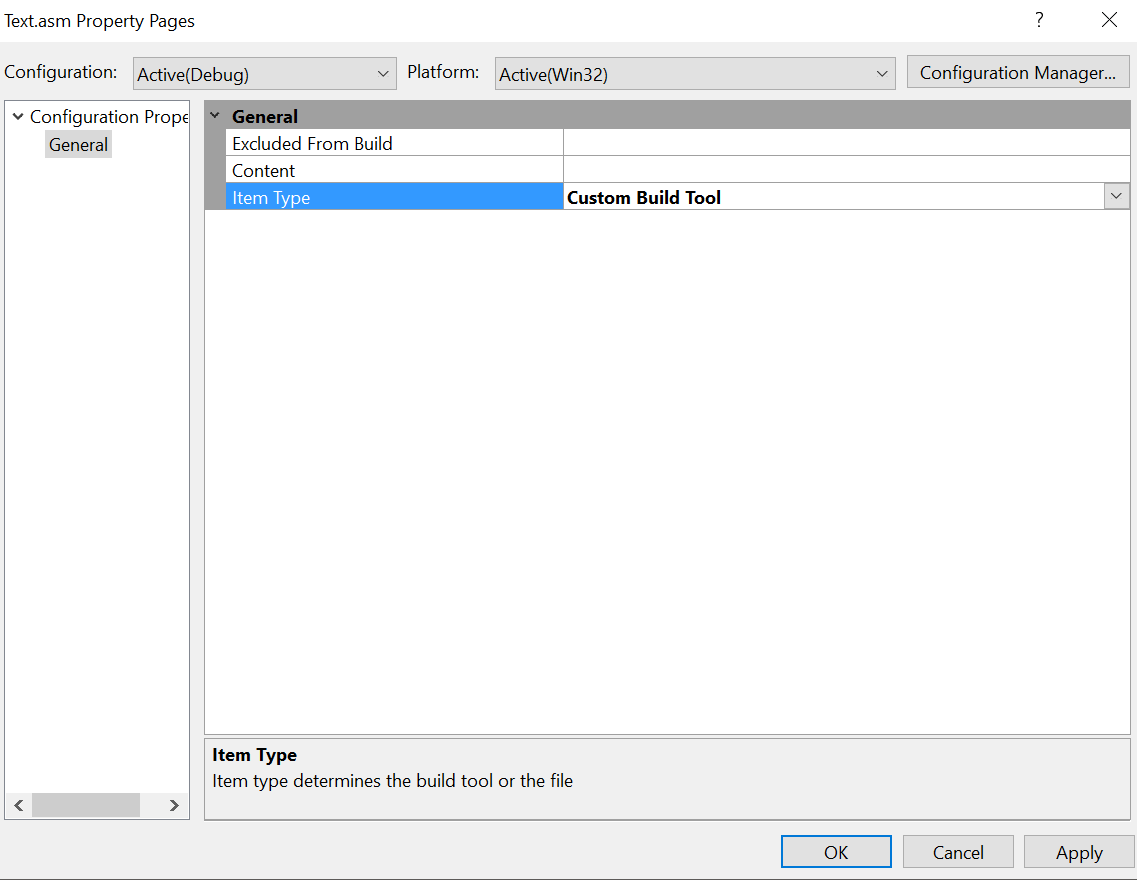
1. Add .asm file to the project



1. Write Assembly code in your .asm file



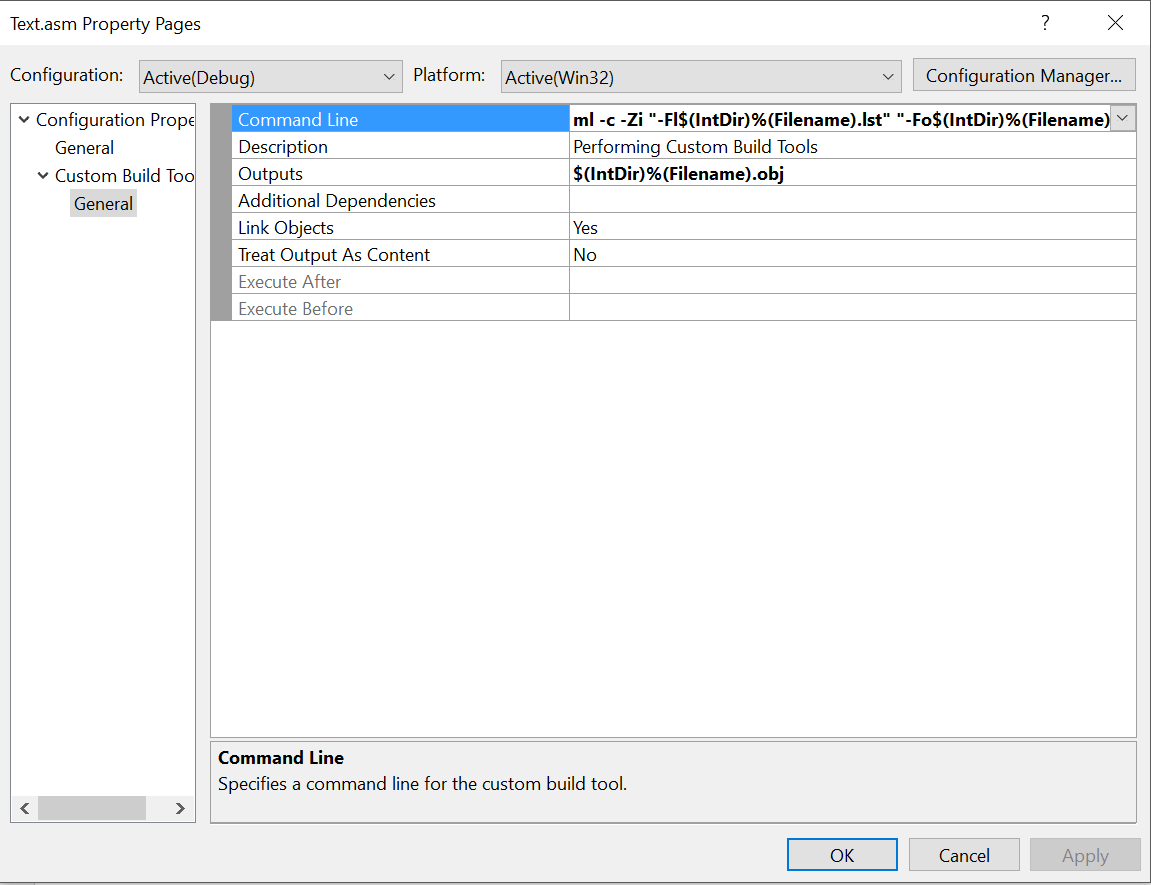
1. Right click on .asm file and select Properties. Select **Configuration Properties > General > Item Type > Custom Build Tool** and Apply the changes.



1. Select **Configuration Properties > Custom Build Tool > General**

**Command Line:** ml -c -Zi "-Fl$(IntDir)%(Filename).lst" "-Fo$(IntDir)%(Filename).obj" "%(FullPath)"

**Outputs:** $(IntDir)%(Filename).obj



**Example:**

**C++ Code:**

#include "stdafx.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

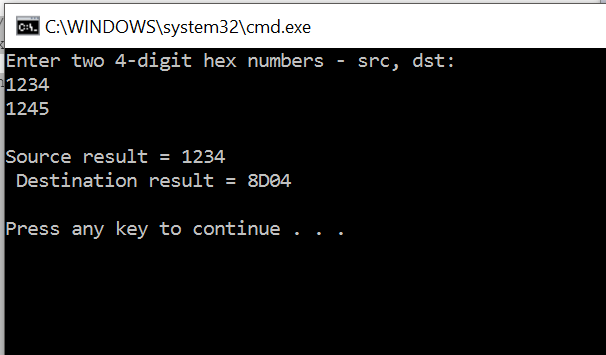
xor ebx, ebx

ret

clear ENDP

END

**Output:**



**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** which calculates their GCD.
3. Write a program in C++ which contains a procedure in assembly named **MinMaxArray** that displays the minimum & maximum values in an array. Pass a size-20 array by reference to this procedure.
4. Write a program in C++ which takes user defined input and implements binary multiplication using inline assembly code.
5. Write a program in C++ which takes user defined string and performs simple encryption using inline assembly code by rotating each plaintext byte a varying number of positions in different directions. For example, in the following array that represents the encryption key, a negative value indicates a rotation to the left and a positive value indicates a rotation to the right. The integer in each position indicates the magnitude of the rotation:

key BYTE -2, 4, 1, 0, -3, 5, 2, -4, -4, 6

Your program should loop through a plaintext message and align the key to the first 10 bytes of the message.

Rotate each plaintext byte by the amount indicated by its matching key array value.

Then, align the key to the next 10 bytes of the message and repeat the process.