C/C++ Programming

Azlan Mukhtar

Introduction

- Visual C++ and GCC compiler
- Useful compiler commands/switches
- Debugging

Using MSVC Compiler

- Basic
  - cl.exe main.cpp
- With assembly listing
  - cl.exe /Famain.asm main.cpp
- With optimization, listing, debug info and link option
  - cl.exe /O2 /Zi /FA source.cpp /link /out:output.exe
- cl.exe options
  - https://msdn.microsoft.com/en-us/library/9s7c9wdw.aspx
- Debugging an exe
  - https://msdn.microsoft.com/en-us/library/0bxe8ytt.aspx

Variables

- Local variable
  - Dynamically allocated on stack
  - Temporarily available
- Global variable
  - Usually located inside memory section
  - Static location, always accessible from anywhere

Variables: Example

```
// variables.cpp
#include <cstdio>
int global_var1 = 5;
int global_result = 0;

int multiply(int a, int b) {
        global_result = a * b * 2;
        return global_result;
}

int main() {
        int local_var1 = 0;
        int local_var2 = 0;

        scanf("%d %d", &local_var1, &local_var2);
        local_var2 = multiply(local_var1, local_var2);
        global_result = global_var1 + local_var1 + local_var2;

        return 0;
}
```

Using Visual Studio

```
1   // variables.cpp : Defines the entry point for the console application.
2   //
3
4   #include "stdafx.h"
5
6   #include <cstdio>
7   int global_var1 = 5;
8   int global_result = 0;
9
10  int multiply(int a, int b) {
11      global_result = a * b * 2;
12      return global_result;
13  }
14
15  int main() {
16      int local_var1 = 0;
17      int local_var2 = 0;
18
19      scanf("%d %d", &local_var1, &local_var2);
20      local_var2 = multiply(local_var1, local_var2);
21      global_result = global_var1 + local_var1 + local_var2;
22
23      return 0;
24  }
```

Using Visual C++ IDE for compiling and debugging

Inspecting compiler result

```
.text:00401356 ; =============== S U B R O U T I N E =======================
.text:00401356
.text:00401356 ; Attributes: library function static
.text:00401356
.text:00401356 ; int __cdecl mainCRTStartup()
.text:00401356                 public _mainCRTStartup
.text:00401356 _mainCRTStartup proc near
.text:00401356                 call    ___security_init_cookie
.text:0040135B                 jmp     __tmainCRTStartup
.text:0040135B _mainCRTStartup endp
.text:0040135B
.text:00401360
.text:00401360 ; =============== S U B R O U T I N E =======================
```

Entry Point

```
.text:004012E4
.text:004012E4 loc_4012E4:                             ; CODE XREF: __tmainCRTStartup+E6↑j
.text:004012E4                 mov     eax, __environ
.text:004012E9                 mov     ___initenv, eax
.text:004012EE                 push    eax
.text:004012EF                 push    ___argv
.text:004012F5                 push    ___argc
.text:004012FB                 call    _main_0
.text:00401300                 add     esp, 0Ch
.text:00401303                 mov     [ebp+mainret], eax
.text:00401306                 cmp     [ebp+managedapp], esi
.text:00401309                 jnz     short $LN30
.text:0040130B                 push    eax             ; code
.text:0040130C                 call    _exit
.text:00401311 ; ---------------------------------------------------------------------------
```

__tmainCRTStartup

Inspecting compiler result - cont

```
.text:00401040 ; =============== S U B R O U T I N E =======================================
.text:00401040
.text:00401040
.text:00401040 ; int __cdecl main()
.text:00401040 _main           proc near               ; CODE XREF: _main_0↑j
.text:00401040
.text:00401040 local_var2      = dword ptr -8
.text:00401040 local_var1      = dword ptr -4
.text:00401040
.text:00401040                 sub     esp, 8
.text:00401043                 xor     eax, eax
.text:00401045                 mov     [esp+8+local_var1], eax
.text:00401049                 mov     [esp+8+local_var2], eax
.text:0040104C                 lea     eax, [esp+8+local_var2]
.text:0040104F                 push    eax
.text:00401050                 lea     ecx, [esp+0Ch+local_var1]
.text:00401054                 push    ecx
.text:00401055                 push    offset format   ; "%d %d"
.text:0040105A                 call    _scanf
.text:0040105F                 mov     ecx, [esp+14h+local_var1]
.text:00401063                 mov     eax, [esp+14h+local_var2]
.text:00401067                 mov     edx, ?global_var1@@3HA ; int global_var1
.text:0040106D                 imul    eax, ecx
.text:00401070                 add     eax, eax
.text:00401072                 add     edx, eax
.text:00401074                 add     edx, ecx
.text:00401076                 mov     ?global_result@@3HA, edx ; int global_result
.text:0040107C                 xor     eax, eax
.text:0040107E                 add     esp, 14h
.text:00401081                 retn
.text:00401081 _main           endp
```

Our main() function

Procedure Call/Function

- Procedural programming is derived from structured programming, based upon the concept of the procedure call
- Procedures, also known as routines, subroutines, methods, or functions
- x86 architecture supports procedure call
- The processor supports procedure calls in the following two different ways:
  - CALL and RET instructions.
  - ENTER and LEAVE instructions, in conjunction with the CALL and RET instructions
- The procedure stack, commonly referred to simply as “the stack”, will save the state of the calling procedure, **pass parameters** to the called procedure, and **store local variables** for the currently executing procedure

Procedure: Example

```
// function.cpp
int calculate(int a, int b)  {
        int result = a + b -2;
        return result;
}

int calculate2(int a, int b) {
        int result = a + b * 2;
        return result;
}

int main() {
        int total = 0;
        int result = calculate(5,6);
        total += result;
        total += calculate2(7,8);
        return 0;
}
```

Procedure: Parameter

- Parameters/arguments
- Argument passing:
  - Value
  - Reference
  - Address (pointer)
- Return value

Calling Convention: CDECL

```
// C/C++ codes

_stdcall int
MyFunction2(int a, int b)
{
   return a + b;
}

 x = MyFunction2(2, 3);
```

```
; x86 asm codes

:_MyFunction@8
push ebp
mov ebp, esp
mov eax, [ebp + 8]
mov edx, [ebp + 12]
add eax, edx
pop ebp
ret 8

Start:
push 3
push 2
call _MyFunction@8
```

Calling Convention: STDCALL

```
// C/C++ codes

_stdcall int
MyFunction2(int a, int b)
{
   return a + b;
}

 x = MyFunction2(2, 3);
```

```
; x86 asm codes

:_MyFunction@8
push ebp
mov ebp, esp
mov eax, [ebp + 8]
mov edx, [ebp + 12]
add eax, edx
pop ebp
ret 8

Start:
push 3
push 2
call _MyFunction@8
```

Array

- int arr[] = {2, 3, 5, 7 ,11};
- Accessing array
  - Value: arr[2] → 5
  - Pointer: &arr[2]

Strings

- Null terminated const strings
  - `char *str = "my strings";`
- Null terminates strings array
  - `char str[] = "my strings";`
- Null terminated byte array
  - `char str[] = {'m','y',' ','s','t', 'r','i','n','g','s', '\x0'};`

Structure

- When to use structure
- Passing structure to functions
- Exercise: Parse binary data file

Pointer

- A variable to hold a pointer/address/location
- How does it look
- Passing pointer around
- Pointer arithmetic
- Void pointer
- Dereferencing

Handle

- Standard library
  - FILE*
  - struct tm
- Windows API
  - HANDLE
  - HINSTANCE
  - HWND
  - HPROCESS

Dynamic Memory

- malloc and free (C)
- new and delete (C++)
- HeapAlloc and HeapFree (Windows)
- VirtualAlloc and VirtualFree (Windows)

References

- http://lmgtfy.com/?q=c+programming+tutorial
- Intel® 64 and IA-32 Architectures Developer's Manual: Vol. 1 -CHAPTER 6 - PROCEDURE CALLS, INTERRUPTS, AND EXCEPTIONS
  - http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-vol-1-manual.html
- Procedural Programming
  - http://en.wikipedia.org/wiki/Procedural_programming
- Functions and Stack Frame
  - http://en.wikibooks.org/wiki/X86_Disassembly/Functions_and_Stack_Frames