Module 06: CS31003: Compilers

Run-time Environments

Pralay Mitra
Partha Pratim Das

Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

pralay@cse.iitkgp.ac.in
ppd@cse.iitkgp.ac.in

August 19, 2021

Module Objectives

- Understand the Run-Time Environment for Program Execution
- Understand Symbol Tables, Activation Records (Stack Frames) and interrelationships
- Understand Binding, Layout and Translation for various Data Types and Scopes

Module Outline

1. Objectives & Outline
2. Binding
   - Properties
3. Memory
4. Activation Record
5. AR in VS: Function
   - Lean Debug Code
   - Safe Debug Code
6. Opt. & I/O
7. Non-int Types
   - double
   - Pointer
   - struct
   - Array
   - Function Pointer
   - Nested Blocks
   - Global / Static
   - Mixed

Lab Focus

- Binding Protocol
- Memory Organization
- Symbol Table, Activation Record, Stack Frame
- Function Call Protocol (`int`)
- Optimization & IO

Properties of a Symbol

- A symbol has multiple Properties based on its context
  - **Declaration**: A declaration states the (lexical) name of the symbol with its data type and qualifier
    - ▷ Variable

```
// Symbol Name = "sum", Symbol Type = "int"
int sum;

// Symbol Name = "array_size, Symbol Type = "const int"
const int array_size = 10;
```

    - ▷ Function

```
// Symbol Name = "info", Symbol Type = "int --> int"
int fibo(int);
```

- Declaration are maintained in the Symbol Table
- Declaration are processed at multiple phases
  - Lexcical Analyzer tokenizes the symbol (`sum` or `fibo`) and creates entry in Symbol Table
  - Syntax Analyzer adds the type information (`int` or `int` → `int`) on Symbol Table
  - The symbol's size information is also entered. This will be used to created the final offset of the symbol in the Activation Record

Properties of a Symbol

- A symbol has multiple Properties based on its context
  - **Initialization**: Set at a declaration, this states the initial value of the symbol

```
// Symbol Name = "sum", Symbol Type = "int",
// Symbol Initialization = "0"
int sum = 0;

// Symbol Name = "p", Symbol Type = "int*",
// Symbol Initialization = "&sum"
int *p = &sum;
```

- Initialization is maintained in the Symbol Table along with the Declaration of the symbol
- Initialization is processed at multiple phases
  - Lexcical Analyzer tokenizes the initialization constant (`0`)
  - Syntax Analyzer adds the initialization information on Symbol Table
  - Semantic Analyzer evaluates the constant initialization expression (like `const double pi = 4.0*atan(1.0);` and updates Symbol Table
  - Note that `class Shape { ... virtual void Draw() = 0; ... };` is not an initialization, but semantic specifier for pure virtual functions

Properties of a Symbol

- A symbol has multiple Properties based on its context
  - **Definition**:
    - ▷ Variable: Any assignment and / or direct / indirect "write" to the symbols

```
// Symbol Name = "sum", Symbol Update = By direct assignment
sum = sum + 1;
// Symbol Name = "*p", Symbol Update = By indirect assignment
*p = *p + 1;
```

    - ▷ Function: A function can have only one definition or function body

```
int fibo(int n) {
    if (0 == n) return 1;
    else return n*fibo(n-1);
}
```

  - Definitions typically result in TAC during intermediate code generation that use various symbol information from the Symbol Table
  - This process may involve compiler-defined (un-named) temporary variables that also go into the Symbol Table. For example `sum = sum + 1;` may be translated to:

```
t1 = sum + 1 // t1 is un-named temporary
sum = t1
```

Properties of a Symbol

- A symbol has multiple Properties based on its context
  - **Binding**: The physical memory address of the symbol

```
// Symbol Name = "sum", Symbol Type = "int"
// Symbol Binding = &sum // Address of sum
int sum;
```

- For example, consider the output of the following program:

```
#include <stdio.h>
int main() {
    int a = 10;
    printf("a = %d\n&a = %p\n", a, &a);
    return 0;
}
a = 10 // Value of 'a'
&a = 0x7ffe7be8ad9c // Address or binding of 'a'
```

- During Target Code Generation phase, the symbol offsets in the Symbol Table are converted into address expressions (like `[ebp] + offset`) that can automatically create the Activation Record at run-time, thereby achieving the binding in an elegant way

Symbol Table to Activation Record: Functions

Symbol Table	**Activation Record**
3-Address Code	**Target Code**
Compile Time	*Run Time*
• Parameters • Local Variables • Temporary • Nested Block Nested blocks are flattened out in the Symbol Table of the Function they are contained in so that all local and temporary variables of the nested blocks are allocated in the activation record of the function.	• Variables ○ Parameters ○ Local Variables ○ Temporary ○ Non-Local References • Stack Management ○ Return Address ○ Return Value ○ Saved Machine Status • Call-Return Protocol

Example: main() & add(): Source, TAC, and Symbol Table

```
int add(int x, int y) {
    int z;
    z = x + y;
    return z;
}
void main(int argc,
          char* argv[]) {
    int a, b, c;
    a = 2;
    b = 3;
    c = add(a, b);
    return;
}
```

```
add:    t1 = x + y
        z = t1
        return z
main:   t1 = 2
        a = t1
        t2 = 3
        b = t2
        param a
        param b
        c = call add, 2
        return
```

ST.glb				
add	int × int → int	func	0	0
main	int × array(*, char*) → void	func	0	0

ST.add()		Parent *ST.glb*		
y	int	param	4	+8
x	int	param	4	+4
z	int	local	4	0
t1	int	temp	4	–4

ST.main()		Parent *ST.glb*		
argv	array(*, char*)	param	4	+8
argc	int	param	4	+4
a	int	local	4	0
b	int	local	4	–4
c	int	local	4	–8
t1	int	temp	4	–12
t2	int	temp	4	–16

Columns: Name, Type, Category, Size, & Offset

Storage Organization

Typical sub-division of run-time memory into code and data areas with the corresponding bindings

Memory Segment		**Bound Items**
Text		*Program Code*
Const		*Program Constants*
Static		*Global & Non-Local Static*
Heap		*Dynamic*
... Heap grows downwards here ... ... **Free Memory** ... Stack grows upwards here		
Stack		*Automatic*

Activation Record

Actual Params	The actual parameters used by the calling procedure (often placed in registers for greater efficiency).
Returned Values	Space for the return value of the called function (often placed in a register for efficiency). Not needed for void type.
Return Address	The return address (value of the program counter, to which the called procedure must return).
Control Link	A control link, pointing to the activation record of the caller.
Access Link	An ”access link” to locate data needed by the called procedure but found elsewhere, e.g., in another activation record.
Saved Machine Status	A saved machine status (state) just before the call to the procedure. This information typically includes the contents of registers that were used by the calling procedure and that must be restored when the return occurs.
Local Data	Local data belonging to the procedure.
Temporary Variables	Temporary values arising from the evaluation of expressions (in cases where those temporaries cannot be held in registers).

Fibo

```
int fibo(int n)
{
    if (n < 2)
        return n;
    else
        return
            fibo(n-1)+
            fibo(n-2);
}

int main()
{
    int m = 10;
    int f = 0;

    f = fibo(m);

    return 0;
}
```

```
fibo:   t1 = 2
        if (n < t1) goto L100
        goto L101
L100:   return n
        goto L102
L101:   t2 = 1
        t3 = n - t2
        param t3
        t4 = call fibo,1
        t5 = 2
        t6 = n - t5
        param t6
        t7 = call fibo, 1
        t8 = t4 + t7
        return t8
        goto L102
L102:   goto L102

main:   param m
        t1 = call fibo, 1;
        f = t1;
```

Activation Tree / Call Graph – Fibo

Module 06

Pralay Mitra & P P Das

Objectives & Outline

Binding

Properties

Memory

AR / SF

Function

Lean Debug Code

Safe Debug Code

Opt. & I/O

Non-int Types

double

Pointer

struct

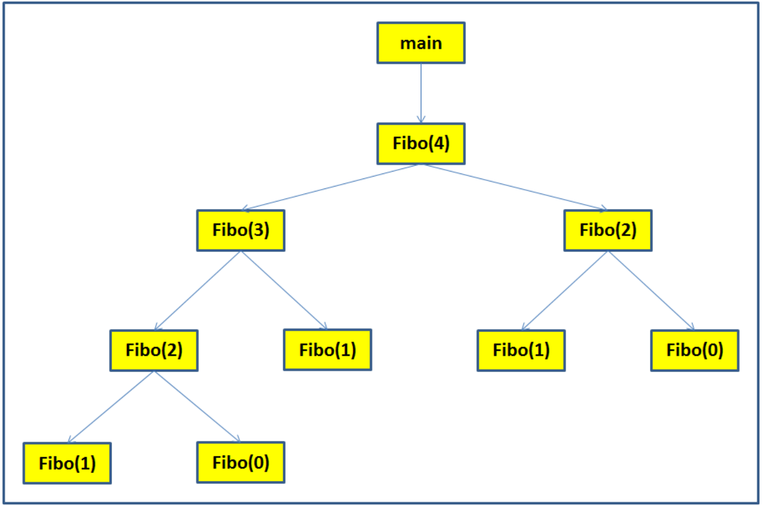
Array

Fn. Ptr.

Nested Blocks

Global / Static

Mixed



main

Fibo(4)

Fibo(3)

Fibo(2)

Fibo(2)

Fibo(1)

Fibo(1)

Fibo(0)

Fibo(1)

Fibo(0)

Activation Records in Action on Stack – Fibo

AR of main()	
Prm	
RV	...
Lnk	crtmain()

AR of fibo(4)	
Prm	4
RV	...
Lnk	main()

AR of fibo(3)	
Prm	3
RV	...
Lnk	fibo(4)

AR of fibo(2)	
Prm	2
RV	...
Lnk	fibo(3)

AR of fibo(1)	
Prm	1
RV	...
Lnk	fibo(2)

AR of main()	
Prm	
RV	...
Lnk	crtmain()

AR of fibo(4)	
Prm	4
RV	...
Lnk	main()

AR of fibo(3)	
Prm	3
RV	...
Lnk	fibo(4)

AR of fibo(2)	
Prm	2
RV	...
Lnk	fibo(3)

AR of fibo(0)	
Prm	0
RV	...
Lnk	fibo(2)

AR of main()	
Prm	
RV	...
Lnk	crtmain()

AR of fibo(4)	
Prm	4
RV	...
Lnk	main()

AR of fibo(3)	
Prm	3
RV	...
Lnk	fibo(4)

AR of fibo(1)	
Prm	1
RV	...
Lnk	fibo(3)

●
●
●
●
●

AR of main()	
Prm	
RV	...
Lnk	crtmain()

AR of fibo(4)	
Prm	4
RV	...
Lnk	main()

AR of fibo(2)	
Prm	2
RV	...
Lnk	fibo(4)

AR of fibo(1)	
Prm	1
RV	...
Lnk	fibo(2)

●
●
●
●
●

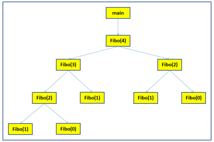
AR of main()	
Prm	
RV	...
Lnk	crtmain()

AR of fibo(4)	
Prm	4
RV	...
Lnk	main()

AR of fibo(2)	
Prm	2
RV	...
Lnk	fibo(4)

AR of fibo(0)	
Prm	0
RV	...
Lnk	fibo(2)

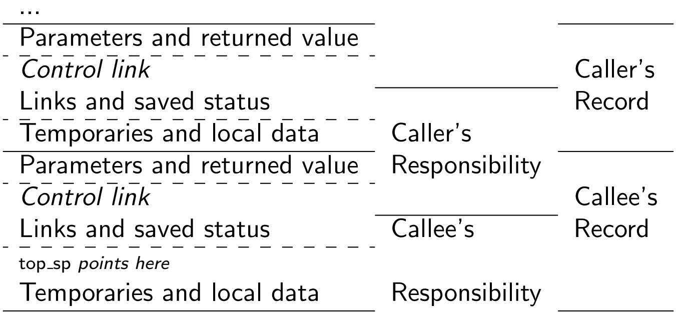
●
●
●
●
●



Calling & Return Sequences

- **Calling Sequences**:
  Consists of code that allocates an activation record on the stack and enters information into its fields.
  The code in a calling sequence is divided between
  - The calling procedure (the "caller") and
  - The procedure it calls (the "callee").
- **Return Sequence**:
  Restores the state of the machine so the calling procedure can continue its execution after the call.

Calling & Return Sequences



...

Parameters and returned value

Control link

Links and saved status

Temporaries and local data

Parameters and returned value

Control link

Links and saved status

top_sp *points here*

Temporaries and local data

Caller's Responsibility

Callee's Responsibility

Caller's Record

Callee's Record

Calling & Return Sequences

- **Calling Sequences**:
  The calling sequence and its division between caller and callee is as follows:
  a) The caller evaluates the actual parameters.
  b) The caller stores a return address and the old value of top_sp into the callee's activation record. The caller then increments top_sp to the position shown – just past the caller's local data and temporaries and the callee's parameters and status fields.
  c) The callee saves the register values and other status information.
  d) The callee initializes its local data and begins execution.

Calling & Return Sequences

- **Return Sequence**:
  A suitable, corresponding return sequence is:
  a) The callee places the return value next to the parameters.
  b) Using information in the machine-status field, the callee restores top_sp and other registers, and then branches to the return address that the caller placed in the status field.
  c) Although top_sp has been decremented, the caller knows where the return value is, relative to the current value of top_sp; the caller therefore may use that value.

AR in VS: Function

Function Call and `int` Data Type

Example: main() & add(): Source & TAC

```
int add(int x, int y) {
    int z;
    z = x + y;
    return z;
}
void main(int argc,
          char* argv[]) {
    int a, b, c;
    a = 2;
    b = 3;
    c = add(a, b);
    return;
}
```

```
add:    t1 = x + y
        z = t1
        return z
main:   t1 = 2
        a = t1
        t2 = 3
        b = t2
        param a
        param b
        c = call add, 2
        return
```

ST.glb				
add	int × int → int	func	0	0
main	int × array(*, char*) → void	func	0	0

ST.add()				
y	int	param	4	+8
x	int	param	4	+4
z	int	local	4	0
t1	int	temp	4	−4

ST.main()				
argv	array(*, char*)	param	4	+8
argc	int	param	4	+4
a	int	local	4	0
b	int	local	4	−4
c	int	local	4	−8
t1	int	temp	4	−12
t2	int	temp	4	−16

Columns: Name, Type, Category, Size, & Offset

main() & add(): Peep-hole Optimized

```
int add(int x, int y) {
    int z;
    z = x + y;
    return z;
}
void main(int argc,
          char* argv[]) {
    int a, b, c;
    a = 2;
    b = 3;
    c = add(a, b);
    return;
}
```

```
add:    z = x + y
        return z
main:   a = 2
        b = 3
        param a
        param b
        c = call add, 2
        return
```

ST.glb				
add	int × int → int	func	0	0
main	int × array(*, char*) → void	func	0	0
ST.add()				
y	int	param	4	+8
x	int	param	4	+4
z	int	local	4	0

ST.main()				
argv	array(*, char*)	param	4	+8
argc	int	param	4	+4
a	int	local	4	0
b	int	local	4	–4
c	int	local	4	–8

Columns: Name, Type, Category, Size, & Offset

main(): x86 Assembly (MSVC++, 32-bit)

```
PUBLIC    _main
EXTRN    __RTC_CheckEsp:PROC
; Function compile flags: /Odtp /RTCsu
_TEXT    SEGMENT
_c$ = -12     ; size = 4
_b$ = -8      ; size = 4
_a$ = -4      ; size = 4
_argc$ = 8    ; size = 4
_argv$ = 12   ; size = 4
_main    PROC

; 6    : void main(int argc, char *argv[]) {

    push   ebp
    mov    ebp, esp
    sub    esp, 12 ; 0000000cH
    mov    DWORD PTR [ebp-12], 0xccccccccH
    mov    DWORD PTR [ebp-8], 0xccccccccH
    mov    DWORD PTR [ebp-4], 0xccccccccH

; 7    :     int a, b, c;
; 8    :     a = 2;

    mov    DWORD PTR _a$[ebp], 2

; 9    :     b = 3;

    mov    DWORD PTR _b$[ebp], 3
```

```
; 10   :     c = add(a, b);

    mov    eax, DWORD PTR _b$[ebp]
    push   eax
    mov    ecx, DWORD PTR _a$[ebp]
    push   ecx
    call   _add
    add    esp, 8 ; pop params
    mov    DWORD PTR _c$[ebp], eax

; 11   :     return;
; 12   : }

    xor    eax, eax
    add    esp, 12 ; 0000000cH
    cmp    ebp, esp
    call   __RTC_CheckEsp
    mov    esp, ebp
    pop    ebp
    ret    0
_main    ENDP
_TEXT    ENDS
```

- No Edit + Continue
- No Run-time Check
- No Buffer Security Check

add(): x86 Assembly (MSVC++, 32-bit)

```
PUBLIC    _add
EXTRN    __RTC_Shutdown:PROC
EXTRN    __RTC_InitBase:PROC
; Function compile flags: /Odtp /RTCsu
rtc$IMZ    ENDS
_TEXT    SEGMENT
_z$ = -4     ; size = 4
_x$ = 8      ; size = 4
_y$ = 12     ; size = 4
_add    PROC

; 1    : int add(int x, int y) {

    push   ebp
    mov    ebp, esp
    push   ecx
    mov    DWORD PTR [ebp-4], 0xccccccccH

; 2    :    int z;
; 3    :    z = x + y;

    mov    eax, DWORD PTR _x$[ebp]
    add    eax, DWORD PTR _y$[ebp]
    mov    DWORD PTR _z$[ebp], eax
```

```
; 4    :    return z;

    mov    eax, DWORD PTR _z$[ebp]

; 5    : }

    mov    esp, ebp
    pop    ebp
    ret    0
_add    ENDP
_TEXT    ENDS
```

- No Edit + Continue
- No Run-time Check
- No Buffer Security Check

Run-Time Error Checking on Stack Frame in Visual Studio

- **Enable Stack Frame Run-Time Error Checking (/GZ)**[1]: Used to enable and disable the run-time error checks feature (prefer /RTC). With this option, uninitialized variables are automatically assigned to `0xccccccccH` (at byte level). It is distinct and easy to identify if the program ends up using an uninitialized variable. Interestingly, in x86 assembly, the op-code `0xcc` is the `int 3` op-code, which is the software breakpoint interrupt. So, if you ever try to execute code in uninitialized memory that has been filled with that fill value, you'll immediately hit a breakpoint, and the operating system will let you attach a debugger (or kill the process).

[1] **Source**:
http://msdn.microsoft.com/en-us/library/hddybs7t.aspx
http://stackoverflow.com/questions/370195/when-and-why-will-an-os-initialise-memory-to-0xcd-0xdd-etc-on-malloc-free-new

ARs of main() and add(): Compiled Code

AR of main()

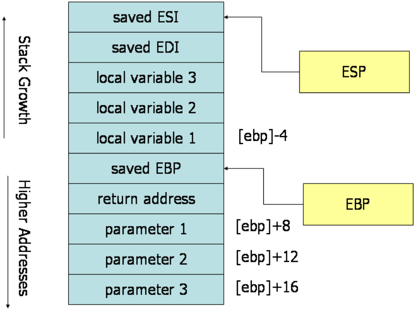
1012	−12	c
1016	−8	b = 3
1020	−4	a = 2
1024		ebp
1028		RA
1032	+8	argc
1036	+12	argv

ebp = 1024

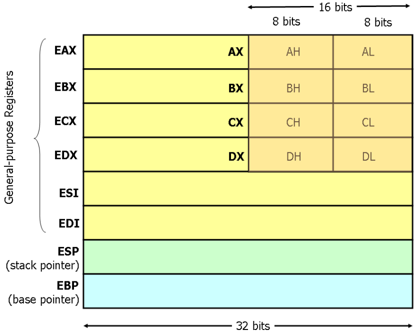
AR of add()

992	−4	z = 5
996		ebp = 1024
1000		RA
1004	+8	ecx = 2: x
1008	+12	eax = 3: y

ebp = 996



Registers of x86



Register	Purpose	Remarks
EAX, EBX, ECX, EDX	General Purpose	Available in 32-, 16-, and 8-bits
ESI	Extended Source Index	General Purpose Index Register
EDI	Extended Destination Index	General Purpose Index Register
ESP	Extended Stack Pointer	Current Stack Pointer
EBP	Extended Base Pointer	Pointer to Stack Frame
EIP	Extended Instruction Pointer	Pointer to Instruction under Execution

Source: http://flint.cs.yale.edu/cs421/papers/x86-asm/asm.html

Code in Execution: main(): Start Address: 0x00

Loc.	Code	esp	ebp	eax	ecx	Stack / Reg.	Value
	; _a$=–4 ; _b$=–8 ; _c$=–12	1028	?	?	?		
0x00	push ebp	1024				[1024] =	ebp
0x01	mov ebp, esp		1024				
0x03	sub esp, 12 ; 0x0000000c	1012					
0x06	mov DWORD PTR [ebp–12], 0xcccccccc ;#fill					c = [1012] =	#fill
0x0d	mov DWORD PTR [ebp–8], 0xcccccccc ;#fill					b = [1016] =	#fill
0x14	mov DWORD PTR [ebp–4], 0xcccccccc ;#fill					a = [1020] =	#fill
0x1b	mov DWORD PTR _a$[ebp], 2					a = [1020] =	2
0x22	mov DWORD PTR _b$[ebp], 3					b = [1016] =	3
0x29	mov eax, DWORD PTR _b$[ebp]			3		eax =	[1016] = 3
0x2c	push eax	1008				y = [1008] =	eax = 3
0x2d	mov ecx, DWORD PTR _a$[ebp]				2	ecx =	[1020] = 2
0x30	push ecx	1004				x = [1004] =	ecx = 2
0x31	call _add	1000				RA = [1000] =	epi = 0x36
						epi = _add (0x50)	
	; On return	1004		5	2	epi =	[1000]
0x36	add esp, 8	1012					
0x39	mov DWORD PTR _c$[ebp], eax					c = [1012] =	eax = 5
0x3c	xor eax, eax			0		eax =	0
0x3e	add esp, 12 ; 0x0000000c	1024					
0x41	cmp ebp, esp					status = ?	
0x43	call __RTC_CheckEsp	1020				[1020] =	epi = 0x48
0x48	mov esp, ebp	1024					
0x4a	pop ebp	1028	?			ebp =	[1024]
0x4b	ret 0	1032					

Code in Execution: add(): Start Address: 0x50

Loc.	Code	esp	ebp	eax	ecx	Stack/Reg.	Value
	;_x$=8 ;_y$=12 ;_z$=–4	1000	1024	3	2		
0x50	push ebp	996				[996] =	ebp = 1024
0x51	mov ebp, esp		996				
0x53	push ecx	992					
0x54	mov DWORD PTR [ebp–4], 0xccccccccH ;#fill					z = [992] =	#fill
0x5b	mov eax, DWORD PTR _x$[ebp]			2		eax =	x = [1004] = 2
0x5e	add eax, DWORD PTR _y$[ebp]			5		eax =	eax+=y= ([1008]=3)
0x61	mov DWORD PTR _z$[ebp], eax					z = [992] =	eax = 5
0x64	mov eax, DWORD PTR _z$[ebp]			5		eax =	z = [992] = 5
0x67	mov esp, ebp	996					
0x69	pop ebp	1000	1024			ebp =	[1024]
0x6a	ret 0	1004				epi =	[1000] = 0x36

main(): x86 Assembly (MSVC++, 32-bit): Safe Debug Code

```
PUBLIC    _main
; Function compile flags: /Odtp /RTCsu /ZI
_TEXT    SEGMENT
_c$ = -32     ; size = 4
_b$ = -20     ; size = 4
_a$ = -8      ; size = 4
_argc$ = 8    ; size = 4
_argv$ = 12   ; size = 4
_main    PROC ; COMDAT

; 6    : void main(int argc, char *argv[]) {

    // PROLOGUE of _main
    // Save the ebp of the caller of _main
    push   ebp
    // Set the ebp of _main
    mov    ebp, esp
    // Create space for local and temporary in the AR of _main
    sub    esp, 228                ; 000000e4H = 32 + 4 + 192
    // Save machine status
    push   ebx
    push   esi
    push   edi
    // Fill the fields of the AR with 0xccccccccH
    lea    edi, DWORD PTR [ebp-228]
    mov    ecx, 57                 ; 00000039H = 228/4
    mov    eax, -858993460         ; ccccccccH
    rep stosd                      ; Store String (doubleword) from eax
                                   ; at edi repeating ecx times
```

main(): x86 Assembly (MSVC++, 32-bit): Safe Debug Code

```
; 7     :      int a, b, c;
; 8     :      a = 2;

    // Copy 2 in DWORD starting at _a$[ebp]
    mov     DWORD PTR _a$[ebp], 2

; 9     :      b = 3;

    // Copy 3 in DWORD starting at _b$[ebp]
    mov     DWORD PTR _b$[ebp], 3

; 10    :      c = add(a, b);

    // Push parameters in the AR of _add
    // Note the right-to-left order
    mov     eax, DWORD PTR _b$[ebp]
    push    eax  ; Value of b is passed
    mov     ecx, DWORD PTR _a$[ebp]
    push    ecx  ; Value of a is passed
    // Return Address gets pushed
    call    _add
    // Re-adjust esp on return from _add
    add     esp, 8 ; pop params
    // Copy return value from eax
    mov     DWORD PTR _c$[ebp], eax

; 11    :      return;
; 12    : }
```

```
    // EPILOGUE of _main
    xor     eax, eax
    // Restore machine status
    pop     edi
    pop     esi
    pop     ebx
    // Annul the space for local and
    // temporary in the AR of _main
    add     esp, 228 ; 000000e4H
    // Check the correctness of esp
    cmp     ebp, esp
    call    __RTC_CheckEsp
    mov     esp, ebp
    // Restore the ebp of the caller
    // of _main
    pop     ebp
    // Return type void -
    // nothing to return
    ret     0
_main    ENDP
_TEXT    ENDS
```

- DWORD PTR: Double Word Pointer – Refers to 4 consecutive bytes
- add() returns int value through eax
- C++ style comments added for better understanding

Activation Record of main()

Offset	Addr.	Stack	Description
	784	`edi`	Saved registers
	788	`esi`	
	792	`ebx`	
	796	`0xcccccccc`	Buffer for Edit & Continue (192 bytes)
	...	`0xcccccccc`	
	...	`0xcccccccc`	
	988	`0xcccccccc`	Local data w/ buffer
−32	992	`c`	
	996	`0xcccccccc`	
	1000	`0xcccccccc`	
−20	1004	`b = 3`	
	1008	`0xcccccccc`	
	1012	`0xcccccccc`	
−8	1016	`a = 2`	
	1020	`0xcccccccc`	
ebp →	1024	`ebp (of Caller of main())`	Control link
	1028	`Return Address`	RA (Caller saved)
+8	1032	`argc`	Params (Caller saved)
+12	1036	`argv`	

add(): x86 Assembly (MSVC++, 32-bit): Safe Debug Code

```
PUBLIC    _add
; Function compile flags: /Odtp /RTCsu /ZI
_TEXT    SEGMENT
_z$ = -8        ; size = 4
_x$ = 8         ; size = 4
_y$ = 12        ; size = 4
_add    PROC    ; COMDAT

; 1    : int add(int x, int y) {

    // PROLOGUE of _add
    // Save the ebp of the caller of _add (_main)
    push   ebp
    // Set the ebp of _add
    mov    ebp, esp
    // Create space for local and temporary in the AR of _add
    sub    esp, 204                    ; 000000ccH = 8 + 4 + 192
    // Save machine status
    push   ebx
    push   esi
    push   edi
    // Fill the fields of the AR with 0xccccccccH
    lea    edi, DWORD PTR [ebp-204]
    mov    ecx, 51                     ; 00000033H = 204/4
    mov    eax, -858993460             ; ccccccccH
    rep stosd
```

add(): x86 Assembly (MSVC++, 32-bit): Safe Debug Code

```
; 2    :     int z;
; 3    :     z = x + y;

    mov    eax, DWORD PTR _x$[ebp]
    add    eax, DWORD PTR _y$[ebp]
    mov    DWORD PTR _z$[ebp], eax

; 4    :     return z;

    mov    eax, DWORD PTR _z$[ebp]

; 5    : }

    // EPILOGUE of _add
    // Restore machine status
    pop    edi
    pop    esi
    pop    ebx
    // Annul the space for local and
    // temporary in the AR of _add
    mov    esp, ebp
    // Restore the ebp of the caller
    // of _add (_main)
    pop    ebp
    // Return through eax -
    // no direct return
    ret    0
_add    ENDP
_TEXT    ENDS
```

- `add()` returns `int` value through `eax`

Activation Record of add()

Offset	Addr.	Stack	Description
	552	`edi`	Saved registers
	556	`esi`	
	560	`ebx`	
	564	`0xcccccccc`	Buffer for Edit & Continue (192 bytes)
	...	`0xcccccccc`	
	...	`0xcccccccc`	
	756	`0xcccccccc`	Local data w/ buffer
–8	760	`z = 5`	
	764	`0xcccccccc`	
ebp →	768	`ebp (of main()) = 1024`	Control link
	772	`Return Address`	RA (Caller saved)
+8	776	`ecx = 2: x`	Params (Caller saved)
+12	780	`eax = 3: y`	

Code in Execution: main(): Start Address: 0x00

Loc.	Code	esp	ebp	eax	ecx	Stack / Reg.	Value
		1028	?	?	?		
0x00	push ebp	1024				[1024] =	ebp
0x01	mov ebp, esp		1024				
0x03	sub esp, 228	796					
0x09	push ebx	792				[792] =	ebx
0x0a	push esi	788				[788] =	esi
0x0b	push edi	784				[784] =	edi
0x0c	lea edi, [ebp-228]					edi =	796
0x12	mov ecx, 57				57	ecx =	57
0x17	mov eax, 0xccccccccH ;#fill			#fill		eax =	#fill
0x1c	rep stosd					[796:1023] =	#fill
0x1e	mov _a$[ebp], 2 ; _a$=-8					a = [1016] =	2
0x25	mov _b$[ebp], 3 ; _b$=-20					b = [1004] =	3
0x2c	mov eax, _b$[ebp]					eax =	[1004] = 3
0x2f	push eax	780		3		[780] =	eax = 3
0x30	mov ecx, _a$[ebp]				2	ecx =	[1016] = 2
0x33	push ecx	776				[776] =	ecx = 2
0x34	call _add	772				[772] =	epi = 0x39
					epi =	_add (0x50)	
	; On return	776		5	51	epi =	[772]
0x39	add esp, 8	784					
0x3c	mov _c$[ebp], eax ; _c$=-32					c = [992] =	eax = 5
0x3f	xor eax, eax			0		eax =	0
0x41	pop edi	788				edi =	[784]
0x42	pop esi	792				esi =	[788]
0x43	pop ebx	796				ebx =	[792]
0x44	mov esp, ebp	1024					
0x46	pop ebp	1028	?			ebp =	[1024]
0x47	ret 0	1032					

Code in Execution: add(): Start Address: 0x50

Loc.	Code	esp	ebp	eax	ecx	Stack/Reg.	Value
		772	1024	3	2		
0x50	push ebp	768				[768] =	ebp
0x51	mov ebp, esp		768				
0x53	sub esp, 204	564					
0x59	push ebx	560				[560] =	ebx
0x5a	push esi	556				[556] =	esi
0x5b	push edi	552				[552] =	edi
0x5c	lea edi, [ebp-204]					edi =	564
0x62	mov ecx, 51				51	ecx =	51
0x67	mov eax, 0xccccccccH ;#fill			#fill		eax =	#fill
0x6c	rep stosd					[564:767] =	#fill
0x6e	mov eax, _x$[ebp] ;_x$=8			2		eax =	x = [776] = 2
0x71	add eax, _y$[ebp] ;_y$=12			5		eax =	eax+=y=[780]=3
0x74	mov _z$[ebp], eax ;_z$=-8					z = [760] =	eax = 5
0x77	mov eax, _z$[ebp]			5		eax =	z = [760] = 5
0x7a	pop edi	556				edi =	[552]
0x7b	pop esi	560				esi =	[556]
0x7c	pop ebx	564				ebx =	[560]
0x7d	mov esp, ebp	768					
0x7f	pop ebp	772	?			ebp =	[768]
0x80	ret 0	776				epi =	[772]

Notes on Stack Frame in Visual Studio

- **Debug Information Format – Edit + Continue (/ZI)**[2]: 192 are bytes allocated in the frame to support the Edit + Continue feature. It allows one to edit the code while a breakpoint is active and add local variables to a function.
- **Buffer Security Check (/GS)**[3]: Detects some buffer overruns that overwrite a function's return address, exception handler address, or certain types of parameters. On functions that the compiler recognizes as subject to buffer overrun problems, the compiler allocates space on the stack before the return address. On function entry, the allocated space is loaded with a *security cookie* that is computed once at module load. On function exit, and during frame unwinding on 64-bit operating systems, a helper function is called to make sure that the value of the cookie is still the same. A different value indicates that an overwrite of the stack may have occurred. If a different value is detected, the process is terminated.

[2] **Source**:
http://msdn.microsoft.com/en-us/library/958x11bc.aspx
http://stackoverflow.com/questions/3362872/explain-the-strange-assembly-of-empty-c-main-function-by-visual-c-compiler

[3] **Source**:
http://msdn.microsoft.com/en-us/library/8dbf701c.aspx

AR in VS: Opt. & I/O

I/O and Optimized Build

Example: main() & add(): Using I/O

```
#include <stdio.h>

int add(int x, int y) {
    int z;
    z = x + y;
    return z;
}
void main() {
    int a, b, c;

    scanf("%d%d", &a, &b);
    c = add(a, b);
    printf("%d\n", c);

    return;
}
```

Let us build in Debug Mode

add(): Debug Build

```
PUBLIC    _add
EXTRN     __RTC_Shutdown:PROC
EXTRN     __RTC_InitBase:PROC
; Function compile flags: /Odtp /RTCsu
_TEXT     SEGMENT
_z$ = -4      ; size = 4
_x$ = 8       ; size = 4
_y$ = 12      ; size = 4
_add     PROC

; 3     : int add(int x, int y) {

    push    ebp
    mov     ebp, esp
    push    ecx
    mov     DWORD PTR [ebp-4], 0xccccccccH

; 4     :     int z;
; 5     :     z = x + y;

    mov     eax, DWORD PTR _x$[ebp]
    add     eax, DWORD PTR _y$[ebp]
    mov     DWORD PTR _z$[ebp], eax
```

```
; 6     :     return z;

    mov     eax, DWORD PTR _z$[ebp]

; 7     : }

    mov     esp, ebp
    pop     ebp
    ret     0
_add     ENDP
_TEXT     ENDS
```

- No change from earlier – as expected

main(): Debug Build

```
PUBLIC    _main
EXTRN    __imp__printf:PROC
EXTRN    __imp__scanf:PROC
EXTRN    @_RTC_CheckStackVars@8:PROC
EXTRN    __RTC_CheckEsp:PROC
; Function compile flags: /Odtp /RTCsu
_TEXT    SEGMENT
_c$ = -28      ; size = 4
_b$ = -20      ; size = 4
_a$ = -8       ; size = 4
_main    PROC

; 8    : void main() {

    push   ebp
    mov    ebp, esp
    sub    esp, 28   ; 0000001cH
    push   esi
    mov    eax, 0xccccccccH
    mov    DWORD PTR [ebp-28], eax
    mov    DWORD PTR [ebp-24], eax
    mov    DWORD PTR [ebp-20], eax
    mov    DWORD PTR [ebp-16], eax
    mov    DWORD PTR [ebp-12], eax
    mov    DWORD PTR [ebp-8], eax
    mov    DWORD PTR [ebp-4], eax
```

```
; 9    :     int a, b, c;
; 10   :
; 11   :     scanf("%d%d", &a, &b);

    mov    esi, esp
    lea    eax, DWORD PTR _b$[ebp]
    push   eax ; Address of b is passed
    lea    ecx, DWORD PTR _a$[ebp]
    push   ecx ; Address of a is passed
    push   OFFSET $SG2756
    call   DWORD PTR __imp__scanf
    add    esp, 12 ; 0000000cH
    cmp    esi, esp
    call   __RTC_CheckEsp

; 12   :     c = add(a, b);

    mov    edx, DWORD PTR _b$[ebp]
    push   edx ; Value of b is passed
    mov    eax, DWORD PTR _a$[ebp]
    push   eax ; Value of a is passed
    call   _add
    add    esp, 8 ; pop params
    mov    DWORD PTR _c$[ebp], eax
```

- Library function `scanf` called by convention
- `lea` used for address parameter in `scanf`

main(): Debug Build

```
; 13    :     printf("%d\n", c);

    mov     esi, esp
    mov     ecx, DWORD PTR _c$[ebp]
    push    ecx ; Value of c is passed
    push    OFFSET $SG2757
    call    DWORD PTR __imp__printf
    add     esp, 8
    cmp     esi, esp
    call    __RTC_CheckEsp

; 14    :
; 15    :     return;
; 16    : }

    xor     eax, eax
    push    edx
    mov     ecx, ebp
    push    eax
    lea     edx, DWORD PTR $LN6@main
    call    @_RTC_CheckStackVars@8
    pop     eax
    pop     edx
    pop     esi
    add     esp, 28 ; 0000001cH
    cmp     ebp, esp
    call    __RTC_CheckEsp
    mov     esp, ebp
    pop     ebp
    ret     0
```

```
$LN6@main:
    DD     2
    DD     $LN5@main
$LN5@main:
    DD     -8 ; fffffff8H
    DD     4
    DD     $LN3@main
    DD     -20 ; ffffffecH
    DD     4
    DD     $LN4@main
$LN4@main:
    DB     98 ; 00000062H
    DB     0
$LN3@main:
    DB     97 ; 00000061H
    DB     0
_main   ENDP
_TEXT   ENDS
```

- Library function `printf` called by convention
- Run-time checks at the end

Example: main() & add(): Using I/O

```
#include <stdio.h>

int add(int x, int y) {
    int z;
    z = x + y;
    return z;
}
void main() {
    int a, b, c;

    scanf("%d%d", &a, &b);
    c = add(a, b);
    printf("%d\n", c);

    return;
}
```

Let us build in Release Mode

add(): Release Build

```
PUBLIC    _add
; Function compile flags: /Ogtp
_TEXT    SEGMENT
; _x$ = ecx
; _y$ = eax

; 4    :      int z;
; 5    :      z = x + y;

    add    eax, ecx

; 6    :      return z;
; 7    : }

    ret    0
_add    ENDP
_TEXT    ENDS
```

- Parameters passed through registers
- No save / restore of machine status
- No use of local (z)

main(): Release Build

```
PUBLIC    _main
; Function compile flags: /Ogtp
_TEXT    SEGMENT
_b$ = -8        ; size = 4
_a$ = -4        ; size = 4
_main    PROC ; COMDAT

; 8    : void main() {

    push   ebp
    mov    ebp, esp
    sub    esp, 8

; 9    :      int a, b, c;
; 10   :
; 11   :      scanf("%d%d", &a, &b);

    lea    eax, DWORD PTR _b$[ebp]
    push   eax
    lea    ecx, DWORD PTR _a$[ebp]
    push   ecx
    push   OFFSET
        ??_C@_04LLKPOCGK@?$CFd?$CFd?$AA@
    call   DWORD PTR __imp__scanf
```

```
; 12   :      c = add(a, b);

    mov     edx, DWORD PTR _a$[ebp]
    add     edx, DWORD PTR _b$[ebp]

; 13   :      printf("%d\n", c);

    push    edx
    push    OFFSET
        ??_C@_03PMGGPEJJ@?$CFd?6?$AA@
    call    DWORD PTR __imp__printf
    add     esp, 20 ; 00000014H

; 14   :
; 15   :      return;
; 16   : }

    xor     eax, eax
    mov     esp, ebp
    pop     ebp
    ret     0
_main    ENDP
_TEXT    ENDS
```

- No unnecessary save / restore of machine status
- Call to add() optimized out!

Handling beyond `int` Types

- `double`
- Pointer
- `struct`
- Array
- Function Pointer
- Nested Blocks
- Global / Static
- Mixed

AR in VS: double

`double` **Data Type**

Example: main() & d_add(): double type

```
double d_add(double x, double y) {
    double z;
    z = x + y;
    return z;
}
void main() {
    double a, b, c;
    a = 2.5;
    b = 3.4;
    c = d_add(a, b);
    return;
}
```

```
d_add:  z = x + y
        return z
main:   a = 2.5
        b = 3.4
        param a
        param b
        c = call d_add, 2
        return
```

ST.glb				
d_add	dbl × dbl → dbl	function	0	0
main	void → void	function	0	0

ST.d_add()				
x	dbl	param	8	0
y	dbl	param	8	16
z	dbl	local	8	24

ST.main()				
a	dbl	local	8	0
b	dbl	local	8	8
c	dbl	local	8	16

Columns are: Name, Type, Category, Size, & Offset

d_add(): double type

```
PUBLIC    _d_add
EXTRN    __fltused:DWORD
EXTRN    __RTC_Shutdown:PROC
EXTRN    __RTC_InitBase:PROC
; Function compile flags: /Odtp /RTCsu
_TEXT    SEGMENT
_z$ = -8 ; size = 8
_x$ = 8  ; size = 8
_y$ = 16 ; size = 8

; 1     : double d_add(double x, double y) {

    push   ebp
    mov    ebp, esp
    sub    esp, 8
    mov    DWORD PTR [ebp-8], 0xccccccccH
    mov    DWORD PTR [ebp-4], 0xccccccccH

; 2     :     double z;
; 3     :     z = x + y;

    fld    QWORD PTR _x$[ebp]
    fadd   QWORD PTR _y$[ebp]
    fstp   QWORD PTR _z$[ebp]

; 4     :     return z;

    fld    QWORD PTR _z$[ebp]
```

```
; 5     : }

    mov    esp, ebp
    pop    ebp
    ret    0
_d_add    ENDP
_TEXT    ENDS
```

- `QWORD PTR`: Quad Word Pointer – Refers to 8 consecutive bytes
- Uses FPU register stack for operations
- `fld`: Load Floating Point Value
- `fadd`: Adds the destination and source operands and stores the sum in the destination location
- `fstp`: Store Floating Point Value
- Return value (local variable `z`) passed through FPU register stack (`fld`)

main(): double type

```
PUBLIC    _main
EXTRN     __RTC_CheckEsp:PROC
CONST     SEGMENT
__real@400b333333333333 DQ
    0400b333333333333r    ; 3.4
__real@4004000000000000 DQ
    04004000000000000r    ; 2.5
CONST     ENDS
; Function compile flags: /Odtp /RTCsu
_TEXT     SEGMENT
_c$ = -24 ; size = 8
_b$ = -16 ; size = 8
_a$ = -8  ; size = 8
_main     PROC

; 6     : void main() {
    push    ebp
    mov     ebp, esp
    sub     esp, 24 ; 00000018H
    mov     eax, 0xccccccccH
    mov     DWORD PTR [ebp-24], eax
    mov     DWORD PTR [ebp-20], eax
    mov     DWORD PTR [ebp-16], eax
    mov     DWORD PTR [ebp-12], eax
    mov     DWORD PTR [ebp-8], eax
    mov     DWORD PTR [ebp-4], eax

; 7     :     double a, b, c;
; 8     :     a = 2.5;
    fld     QWORD PTR __real@4004000000000000
    fstp    QWORD PTR _a$[ebp]
```

```
; 9     :     b = 3.4;
    fld     QWORD PTR __real@400b333333333333
    fstp    QWORD PTR _b$[ebp]
; 10    :     c = d_add(a, b);
    sub     esp, 8 ; push b
    fld     QWORD PTR _b$[ebp]
    fstp    QWORD PTR [esp]
    sub     esp, 8 ; push a
    fld     QWORD PTR _a$[ebp]
    fstp    QWORD PTR [esp]
    call    _d_add
    add     esp, 16 ; 00000010H - pop params
    fstp    QWORD PTR _c$[ebp]
; 11    :     return;
; 12    : }
    xor     eax, eax
    add     esp, 24 ; 00000018H
    cmp     ebp, esp
    call    __RTC_CheckEsp
    mov     esp, ebp
    pop     ebp
    ret     0
_main     ENDP
_TEXT     ENDS
```

- No push / pop for QWORD – using explicit manipulation of esp with load / store.
- Return value returned through FPU register stack (fstp)

ARs of main() and d_add(): double type

```
; Function compile flags: /Odtp /RTCsu
```

- No Edit + Continue
- No Run-time Check
- No Buffer Security Check

AR of main()

1000	−24	`c =`
1004		`5.9`
1008	−16	`b =`
1012		`3.4`
1016	−8	`a =`
1020		`2.5`
1024		`ebp`
1028		`RA`

ebp = 1024

AR of d_add()

968	−4	`z =`
972		`5.9`
976		`ebp = 1024`
980		`RA`
984	+8	`x`
988		`2.5`
992	+16	`y`
996		`3.4`

ebp = 976

AR in VS: Pointer

Pointer Data Type

Example: main() & swap()

```
void swap(int *x, int *y) {
    int t;
    t = *x;
    *x = *y;
    *y = t;
    return;
}
void main() {
    int a = 1, b = 2;
    swap(&a, &b);
    return;
}
```

```
swap:   t = *x;
        *x = *y;
        *y = t;
        return
main:   a = 1
        b = 2
        t1 = &a
        t2 = &b
        param t1
        param t2
        call swap, 2
        return
```

ST.glb

swap	int* × int* → void	func	0	0
main	void → void	func	0	0

ST.swap()

y	int*	prm	4	0
x	int*	prm	4	4
t	int	lcl	4	8

ST.main()

a	int	lcl	4	0
b	int	lcl	4	4
t1	int*	lcl	4	8
t2	int*	lcl	4	12

Columns are: Name, Type, Category, Size, & Offset

swap()

```
PUBLIC    _swap
; Function compile flags: /Odtp /RTCsu
_TEXT    SEGMENT
_t$ = -4       ; size = 4
_x$ = 8        ; size = 4
_y$ = 12       ; size = 4
_swap    PROC

; 1    : void swap(int *x, int *y) {

    push   ebp
    mov    ebp, esp
    push   ecx
    mov    DWORD PTR [ebp-4], 0xccccccccH

; 2    :     int t;
; 3    :     t = *x;

    mov    eax, DWORD PTR _x$[ebp]
    mov    ecx, DWORD PTR [eax]
    mov    DWORD PTR _t$[ebp], ecx
```

```
; 4    :     *x = *y;

    mov    edx, DWORD PTR _x$[ebp]
    mov    eax, DWORD PTR _y$[ebp]
    mov    ecx, DWORD PTR [eax]
    mov    DWORD PTR [edx], ecx

; 5    :     *y = t;

    mov    edx, DWORD PTR _y$[ebp]
    mov    eax, DWORD PTR _t$[ebp]
    mov    DWORD PTR [edx], eax

; 6    :     return;
; 7    : }

    mov    esp, ebp
    pop    ebp
    ret    0
_swap    ENDP
_TEXT    ENDS
```

- Pointer dereferencing handled in two instructions

main()

```
PUBLIC    _main
; Function compile flags: /Odtp /RTCsu
_TEXT    SEGMENT
_b$ = -20     ; size = 4
_a$ = -8      ; size = 4
_main    PROC
; 8    : void main() {
    push   ebp
    mov    ebp, esp
    sub    esp, 24 ; 00000018H
    mov    eax, 0xccccccccH
    mov    DWORD PTR [ebp-24], eax
    mov    DWORD PTR [ebp-20], eax
    mov    DWORD PTR [ebp-16], eax
    mov    DWORD PTR [ebp-12], eax
    mov    DWORD PTR [ebp-8], eax
    mov    DWORD PTR [ebp-4], eax
; 9    :     int a = 1, b = 2;
    mov    DWORD PTR _a$[ebp], 1
    mov    DWORD PTR _b$[ebp], 2
; 10   :     swap(&a, &b);
    lea    eax, DWORD PTR _b$[ebp]
    push   eax
    lea    ecx, DWORD PTR _a$[ebp]
    push   ecx
    call   _swap
    add    esp, 8
; 11   :     return;
; 12   : }
```

```
    xor    eax, eax
    push   edx
    mov    ecx, ebp
    push   eax
    lea    edx, DWORD PTR $LN6@main
    call    @_RTC_CheckStackVars@8
    pop    eax
    pop    edx
    add    esp, 24 ; 00000018H
    cmp    ebp, esp
    call   __RTC_CheckEsp
    mov    esp, ebp
    pop    ebp
    ret    0
```

- lea used to pass reference parameters a and b

main()

```
$LN6@main:
    DD    2
    DD    $LN5@main
$LN5@main:
    DD    -8 ; fffffff8H
    DD    4
    DD    $LN3@main
    DD    -20 ; ffffffecH
    DD    4
    DD    $LN4@main
$LN4@main:
    DB    98 ; 00000062H
    DB    0
$LN3@main:
    DB    97 ; 00000061H
    DB    0
_main    ENDP
_TEXT    ENDS
```

ARs of main() and swap()

```
; Function compile flags: /Odtp /RTCsu
```

980	−4	t = 1
ebp → **984**		ebp = 1024
988		RA
992	+8	ecx = 1016: x
996	+12	eax = 1004: y
1000		0xcccccccc
1004	−20	b = 2
1008		0xcccccccc
1012		0xcccccccc
1016	−8	a = 1
1020		0xcccccccc
ebp → **1024**		ebp
1028		RA

$\text{ebp} = 1024$

AR in VS: struct

struct Data Type

Example: main() & C_add(): struct type

```
typedef struct {
    double re;
    double im;
} Complex;

Complex C_add(Complex x, Complex y) {
    Complex z;

    z.re = x.re + y.re;
    z.im = x.im + y.im;
    return z;
}

void main() {
    Complex a = { 2.3, 6.4 }, b = { 3.5, 1.4 }, c = { 0.0, 0.0 };
    c = C_add(a, b);
    return;
}
```

```
C_add:  z.re = x.re + y.re
        z.im = x.im + y.im
        *RV = z
        return
main:   a.re = 2.3
        a.im = 6.4
        b.re = 3.5
        b.im = 1.4
        c.re = 0.0
        c.im = 0.0
        param a
        param b
        c = call C_add, 2
        return
```

ST.glb: ST.glb.parent = null

Complex	struct{dbl, dbl}	type	0	ST.Complex
C_add	Complex × Complex → Complex	function	0	ST.C_add
main	void → void	function	0	ST.main

ST.C_add(): ST.C_add.parent = ST.glb

RV	Complex*	param	4	0
x	Complex	param	16	20
y	Complex	param	16	36
z	Complex	local	16	52

ST.Complex: ST.Complex.parent = ST.glb

re	dbl	local	8	0
im	dbl	local	8	8

ST.main(): ST.main.parent = ST.glb

a	Complex	local	16	0
b	Complex	local	16	16
c	Complex	local	16	32
RV	Complex	local	16	48

Columns are: Name, Type, Category, Size, & Offset

C_add(): struct type

```
; Function compile flags: /Odtp /RTCsu
_TEXT    SEGMENT
_z$ = -20       ; size = 16
$T1 = 8         ; size = 4
_x$ = 12        ; size = 16
_y$ = 28        ; size = 16
_C_add    PROC

; 7    : {

    push   ebp
    mov    ebp, esp
    sub    esp, 24                   ; 00000018H
    mov    eax, -858993460           ; ccccccccH
    mov    DWORD PTR [ebp-24], eax
    mov    DWORD PTR [ebp-20], eax
    mov    DWORD PTR [ebp-16], eax
    mov    DWORD PTR [ebp-12], eax
    mov    DWORD PTR [ebp-8], eax
    mov    DWORD PTR [ebp-4], eax

; 8    :     Complex z;
; 9    :
; 10   :     z.re = x.re + y.re;

    movsd    xmm0, QWORD PTR _x$[ebp]
    addsd    xmm0, QWORD PTR _y$[ebp]
    movsd    QWORD PTR _z$[ebp], xmm0
```

```
; 11    :     z.im = x.im + y.im;

    movsd    xmm0, QWORD PTR _x$[ebp+8]
    addsd    xmm0, QWORD PTR _y$[ebp+8]
    movsd    QWORD PTR _z$[ebp+8], xmm0

; 12    :
; 13    :     return z;

    mov    eax, DWORD PTR $T1[ebp]
    mov    ecx, DWORD PTR _z$[ebp]
    mov    DWORD PTR [eax], ecx
    mov    edx, DWORD PTR _z$[ebp+4]
    mov    DWORD PTR [eax+4], edx
    mov    ecx, DWORD PTR _z$[ebp+8]
    mov    DWORD PTR [eax+8], ecx
    mov    edx, DWORD PTR _z$[ebp+12]
    mov    DWORD PTR [eax+12], edx
    mov    eax, DWORD PTR $T1[ebp]
```

- xmm0: xmm0 through xmm7 are 64-bit Registers in Streaming SIMD Extensions (SSE)

C_add(): struct type

```
; 14   : }

    push   edx
    mov    ecx, ebp
    push   eax
    lea    edx, DWORD PTR $LN5@C_add
    call   @_RTC_CheckStackVars@8
    pop    eax
    pop    edx
    mov    esp, ebp
    pop    ebp
    ret    0
    npad    3
$LN5@C_add:
    DD    1
    DD    $LN4@C_add
$LN4@C_add:
    DD    -20         ; ffffffecH
    DD    16          ; 00000010H
    DD    $LN3@C_add
$LN3@C_add:
    DB    122         ; 0000007aH
    DB    0
_C_add    ENDP
_TEXT    ENDS
```

main(): struct type

```
PUBLIC    _C_add
PUBLIC    _main
PUBLIC    __real@0000000000000000
PUBLIC    __real@3ff6666666666666
PUBLIC    __real@4002666666666666
PUBLIC    __real@400c000000000000
PUBLIC    __real@401999999999999a
;    COMDAT __real@401999999999999a
CONST SEGMENT __real@401999999999999a
    DQ 0401999999999999ar    ; 6.4
CONST    ENDS
;    COMDAT __real@400c000000000000
CONST SEGMENT __real@400c000000000000
    DQ 0400c000000000000r    ; 3.5
CONST    ENDS
;    COMDAT __real@4002666666666666
CONST SEGMENT __real@4002666666666666
    DQ 04002666666666666r    ; 2.3
CONST    ENDS
;    COMDAT __real@3ff6666666666666
CONST SEGMENT __real@3ff6666666666666
    DQ 03ff6666666666666r    ; 1.4
CONST    ENDS
;    COMDAT __real@0000000000000000
CONST SEGMENT __real@0000000000000000
    DQ 00000000000000000r    ; 0
CONST    ENDS
;    COMDAT rtc$TMZ
```

- xmm0: xmm0 – xmm7 are 64-bit Reg. in Streaming SIMD Extensions (SSE)

```
; Function compile flags: /Odtp /RTCsu
_TEXT    SEGMENT
$T1 = -88                        ; size = 16
_c$ = -68                        ; size = 16
_b$ = -44                        ; size = 16
_a$ = -20                        ; size = 16
_main    PROC
; 16    : void main() {
    push   ebp
    mov    ebp, esp
    sub    esp, 88                    ; 00000058H
    push   edi
    lea    edi, DWORD PTR [ebp-88]
    mov    ecx, 22                    ; 00000016H
    mov    eax, -858993460            ; ccccccccH
    rep stosd
; 17    :     Complex a = { 2.3, 6.4 },
             ; b = { 3.5, 1.4 }, c = { 0.0, 0.0 };
    movsd xmm0, QWORD PTR __real@4002666666666666
    movsd QWORD PTR _a$[ebp], xmm0
    movsd xmm0, QWORD PTR __real@401999999999999a
    movsd QWORD PTR _a$[ebp+8], xmm0
    movsd xmm0, QWORD PTR __real@400c000000000000
    movsd QWORD PTR _b$[ebp], xmm0
    movsd xmm0, QWORD PTR __real@3ff6666666666666
    movsd QWORD PTR _b$[ebp+8], xmm0
    movsd xmm0, QWORD PTR __real@0000000000000000
    movsd QWORD PTR _c$[ebp], xmm0
    movsd xmm0, QWORD PTR __real@0000000000000000
    movsd QWORD PTR _c$[ebp+8], xmm0
; 18    :
```

main(): struct type

```
; 19    :       c = C_add(a, b);
    sub     esp, 16 ; 00000010H
    mov     eax, esp
    mov     ecx, DWORD PTR _b$[ebp]
    mov     DWORD PTR [eax], ecx
    mov     edx, DWORD PTR _b$[ebp+4]
    mov     DWORD PTR [eax+4], edx
    mov     ecx, DWORD PTR _b$[ebp+8]
    mov     DWORD PTR [eax+8], ecx
    mov     edx, DWORD PTR _b$[ebp+12]
    mov     DWORD PTR [eax+12], edx
    sub     esp, 16 ; 00000010H
    mov     eax, esp
    mov     ecx, DWORD PTR _a$[ebp]
    mov     DWORD PTR [eax], ecx
    mov     edx, DWORD PTR _a$[ebp+4]
    mov     DWORD PTR [eax+4], edx
    mov     ecx, DWORD PTR _a$[ebp+8]
    mov     DWORD PTR [eax+8], ecx
    mov     edx, DWORD PTR _a$[ebp+12]
    mov     DWORD PTR [eax+12], edx
    lea     eax, DWORD PTR $T1[ebp]
    push    eax
    call    _C_add
```

- xmm0: xmm0 – xmm7 are 64-bit Reg. in Streaming SIMD Extensions (SSE)

```
    add     esp, 36 ; 00000024H = 16 + 16 + 4
    mov     ecx, DWORD PTR [eax]
    mov     DWORD PTR _c$[ebp], ecx
    mov     edx, DWORD PTR [eax+4]
    mov     DWORD PTR _c$[ebp+4], edx
    mov     ecx, DWORD PTR [eax+8]
    mov     DWORD PTR _c$[ebp+8], ecx
    mov     edx, DWORD PTR [eax+12]
    mov     DWORD PTR _c$[ebp+12], edx
; 20    :
; 21    :       return;
; 22    : }

    xor     eax, eax
    push     edx
    mov     ecx, ebp
    push     eax
    lea     edx, DWORD PTR $LN7@main
    call     @_RTC_CheckStackVars@8
    pop     eax
    pop     edx
    pop     edi
    add     esp, 88                     ; 00000058H
    cmp     ebp, esp
    call     __RTC_CheckEsp
    mov     esp, ebp
    pop     ebp
    ret     0
```

main(): struct type

```
    npad    3
$LN7@main:
    DD    3
    DD    $LN6@main
$LN6@main:
    DD    -20                    ; ffffffecH
    DD    16                    ; 00000010H
    DD    $LN3@main
    DD    -44                    ; ffffffd4H
    DD    16                    ; 00000010H
    DD    $LN4@main
    DD    -68                    ; ffffffbcH
    DD    16                    ; 00000010H
    DD    $LN5@main
$LN5@main:
    DB    99                    ; 00000063H
    DB    0
$LN4@main:
    DB    98                    ; 00000062H
    DB    0
$LN3@main:
    DB    97                    ; 00000061H
    DB    0
_main    ENDP
_TEXT    ENDS
```

AR in VS: Array

Array Data Type

Example: main() & Sum(): Using Array & Nested Block

```
#include <stdio.h>

int Sum(int a[], int n) {
    int i, s = 0;
    for(i = 0; i < n; ++i) {
        int t;
        t = a[i];
        s += t;
    }
    return s;
}
void main() {
    int a[3];
    int i, s, n = 3;
    for(i = 0; i < n; ++i)
        a[i] = i;
    s = Sum(a, n);
    printf("%d\n", s);
}
```

```
Sum:    s = 0
        i = 0
L0:     if i < n goto L2
        goto L3
L1:     i = i + 1
        goto L0
L2:     t1 = i * 4
        t_1 = a[t1]
        s = s + t_1
        goto L1
L3:     return s
```

Block local variable t is named as t_1 to qualify for the unnamed block within which it occurs.

```
main:   n = 3
        i = 0
L0:     if i < n goto L2
        goto L3
L1:     i = i + 1
        goto L0
L2:     t1 = i * 4
        a[t1] = i
        goto L1
L3:     param a
        param n
        s = call Sum, 2
        param "%d\n"
        param s
        call printf, 2
        return
```

Parameter s of printf is handled through varargs.

ST.glb: ST.glb.parent = null				
Sum	array(*, int) × int → int	function	0	ST.Sum
main	void → void	function	0	ST.main
ST.main(): ST.main.parent = ST.glb				
a	array(3, int)	local	12	0
i	int	local	4	12
s	int	local	4	16
n	int	local	4	20
t1	int	temp	4	24

ST.Sum(): ST.Sum.parent = ST.glb				
a	int[]	param	4	0
n	int	param	4	4
i	int	local	4	8
s	int	local	4	12
t_1	int	local	4	16
t1	int	temp	4	20

Columns are: Name, Type, Category, Size, & Offset

main()

```
PUBLIC    _main
; Function compile flags: /Odtp /RTCsu
_TEXT    SEGMENT
_n$ = -32      ; size = 4
_s$ = -28      ; size = 4
_i$ = -24      ; size = 4
_a$ = -16      ; size = 12
_main    PROC

; 12   : void main() {

    push    ebp
    mov     ebp, esp
    sub     esp, 32 ; 00000020H
    push    esi
    mov     eax, -858993460 ; ccccccccH
    mov     DWORD PTR [ebp-32], eax
    mov     DWORD PTR [ebp-28], eax
    mov     DWORD PTR [ebp-24], eax
    mov     DWORD PTR [ebp-20], eax
    mov     DWORD PTR [ebp-16], eax
    mov     DWORD PTR [ebp-12], eax
    mov     DWORD PTR [ebp-8], eax
    mov     DWORD PTR [ebp-4], eax

; 13   :     int a[3];
; 14   :     int i, s, n = 3;

    mov     DWORD PTR _n$[ebp], 3
```

```
; 15   :     for(i = 0; i < n; ++i)

    mov     DWORD PTR _i$[ebp], 0
    jmp     SHORT $LN3@main
$LN2@main:
    mov     eax, DWORD PTR _i$[ebp]
    add     eax, 1
    mov     DWORD PTR _i$[ebp], eax
$LN3@main:
    mov     ecx, DWORD PTR _i$[ebp]
    cmp     ecx, DWORD PTR _n$[ebp]
    jge     SHORT $LN1@main

; 16   :         a[i] = i;

    // Index in edx
    mov     edx, DWORD PTR _i$[ebp]
    // Right-hand Expression in eax
    mov     eax, DWORD PTR _i$[ebp]
    // Index expression directly used
    mov     DWORD PTR _a$[ebp+edx*4], eax
    jmp     SHORT $LN2@main
$LN1@main:
```

- Array reference in a uses index expression in code – no temporary used
- for loop condition implemented as cmp and conditional jump jge

main()

```
; 17    :      s = Sum(a, n);

    mov     ecx, DWORD PTR _n$[ebp]
    push    ecx
    lea     edx, DWORD PTR _a$[ebp]
    push    edx
    call    _Sum
    add     esp, 8
    mov     DWORD PTR _s$[ebp], eax

; 18    :      printf("%d\n", s);

    mov     esi, esp
    mov     eax, DWORD PTR _s$[ebp]
    push    eax
    push    OFFSET $SG2765
    call    DWORD PTR __imp__printf
    add     esp, 8
    cmp     esi, esp
    call    __RTC_CheckEsp

; 19    : }

    xor     eax, eax
    push    edx
    mov     ecx, ebp
    push    eax
    lea     edx, DWORD PTR $LN8@main
    call    @_RTC_CheckStackVars@8
```

```
    pop     eax
    pop     edx
    pop     esi
    add     esp, 32 ; 00000020H
    cmp     ebp, esp
    call    __RTC_CheckEsp
    mov     esp, ebp
    pop     ebp
    ret     0
    npad    3
$LN8@main:
    DD     1
    DD     $LN7@main
$LN7@main:
    DD     -16 ; fffffff0H
    DD     12 ; 0000000cH
    DD     $LN6@main
$LN6@main:
    DB     97 ; 00000061H
    DB     0
_main   ENDP
_TEXT   ENDS
END
```

- lea used to pass parameter in a

Sum()

```
PUBLIC    _Sum
EXTRN    __RTC_Shutdown:PROC
EXTRN    __RTC_InitBase:PROC
; Function compile flags: /Odtp /RTCsu
_TEXT    SEGMENT
_t$2755 = -12; size = 4
_s$ = -8     ; size = 4
_i$ = -4     ; size = 4
_a$ = 8      ; size = 4
_n$ = 12     ; size = 4
_Sum    PROC
; 3    : int Sum(int a[], int n) {
    push   ebp
    mov    ebp, esp
    sub    esp, 12 ; 0000000cH
    mov    DWORD PTR [ebp-12], 0xccccccccH
    mov    DWORD PTR [ebp-8], 0xccccccccH
    mov    DWORD PTR [ebp-4], 0xccccccccH
; 4    :     int i, s = 0;
    mov    DWORD PTR _s$[ebp], 0
; 5    :     for(i = 0; i < n; ++i) {
    mov    DWORD PTR _i$[ebp], 0
    jmp    SHORT $LN3@Sum
$LN2@Sum:
    mov    eax, DWORD PTR _i$[ebp]
    add    eax, 1
    mov    DWORD PTR _i$[ebp], eax
$LN3@Sum:
    mov    ecx, DWORD PTR _i$[ebp]
    cmp    ecx, DWORD PTR _n$[ebp]
    jge    SHORT $LN1@Sum
```

```
; 6    :          int t;
; 7    :          t = a[i];
    mov    edx, DWORD PTR _i$[ebp]
    mov    eax, DWORD PTR _a$[ebp]
    mov    ecx, DWORD PTR [eax+edx*4]
    mov    DWORD PTR _t$2755[ebp], ecx
; 8    :          s += t;
    mov    edx, DWORD PTR _s$[ebp]
    add    edx, DWORD PTR _t$2755[ebp]
    mov    DWORD PTR _s$[ebp], edx
; 9    :      }
    jmp    SHORT $LN2@Sum
$LN1@Sum:
; 10   :      return s;
    mov    eax, DWORD PTR _s$[ebp]
; 11   : }
    mov    esp, ebp
    pop    ebp
    ret    0
_Sum    ENDP
_TEXT    ENDS
```

- a is reference parameter – &a[0]
- Local variable declaration int t; in block is renamed to _t$2755 instead of _t$ to track unnamed block

Activation Records of main() & Sum()

Offset	Addr.	Stack	Description
−12	960	`t (_t$2755)`	Local data w/ buffer
−8	964	`s`	
−4	968	`i`	
ebp →	972	`ebp (of main())`	Control link
	976	`Return Address`	
+8	980	`a`	Reference Param – &a[0]
+12	984	`n`	
	988	`esi`	Saved registers
−32	992	`n`	Local data w/ buffer
−28	996	`s`	
−24	1000	`i`	
	1004	`0xcccccccc`	
−16	1008	`a [0]`	
	1012	`a [1]`	
	1016	`a [2]`	
	1020	`0xcccccccc`	
ebp →	1024	`ebp (of Caller of main())`	Control link
	1028	`Return Address`	

AR in VS: Function Pointer

Function Pointer

Example: main(), function parameter & other functions

```
int trans(int a, int(*f)(int), int b)
{ return a + f(b); }

int inc(int x) { return x + 1; }

int dec(int x) { return x - 1; }

void main() {
    int x, y, z;

    x = 2;
    y = 3;
    z = trans(x, inc, y) +
        trans(x, dec, y);
    return;
}
```

```
trans:  param b
        t1 = call f, 1
        t2 = a + t1
        return t2

inc:    t1 = x + 1
        return t1

dec:    t1 = x - 1
        return t1
```

```
main:   x = 2
        y = 3
        param x
        param inc
        param y
        t1 = call trans, 3
        param x
        param dec
        param y
        t2 = call trans, 3
        z = t1 + t2
        return
```

ST.glb: ST.glb.parent = null

trans	int × ptr(int → int) × int → int	func	0	0
inc	int → int	func	0	0
dec	int → int	func	0	0
main	void → void	func	0	0

ST.trans(): ST.trans.parent = ST.glb

a	int	prm	4	0
f	ptr(int → int)	prm	4	4
b	int	prm	4	8
t1	int	tmp	4	12
t2	int	tmp	4	16

ST.inc(): ST.inc.parent = ST.glb

x	int	prm	4	0
t1	int	tmp	4	4

ST.dec(): ST.dec.parent = ST.glb

x	int	prm	4	0
t1	int	tmp	4	4

ST.main(): ST.main.parent = ST.glb

x	int	lcl	4	0
y	int	lcl	4	4
z	int	lcl	4	8
t1	int	tmp	4	12
t2	int	tmp	4	16

Columns are: Name, Type, Category, Size, & Offset

main()

```
PUBLIC    _inc
PUBLIC    _dec
PUBLIC    _trans
PUBLIC    _main
; Function compile flags: /Odtp /RTCu
_TEXT    SEGMENT
_z$ = -12      ; size = 4
_x$ = -8       ; size = 4
_y$ = -4       ; size = 4
_main    PROC

; 17   : {

    push   ebp
    mov    ebp, esp
    sub    esp, 12 ; 0000000cH
    push   esi

; 18   :     int x, y, z;
; 19   :
; 20   :     x = 2;

    mov    DWORD PTR _x$[ebp], 2

; 21   :     y = 3;

    mov    DWORD PTR _y$[ebp], 3
```

```
; 22   :     z = trans(x, inc, y) + trans(x, dec, y);
    mov    eax, DWORD PTR _y$[ebp]
    push   eax
    push   OFFSET _inc // Function Pointer
    mov    ecx, DWORD PTR _x$[ebp]
    push   ecx
    call   _trans
    add    esp, 12 ; 0000000cH
    mov    esi, eax
    mov    edx, DWORD PTR _y$[ebp]
    push   edx
    push   OFFSET _dec // Function Pointer
    mov    eax, DWORD PTR _x$[ebp]
    push   eax
    call   _trans
    add    esp, 12 ; 0000000cH
    add    esi, eax
    mov    DWORD PTR _z$[ebp], esi

; 23   :     return;
; 24   : }

    xor    eax, eax
    pop    esi
    mov    esp, ebp
    pop    ebp
    ret    0
_main    ENDP
_TEXT    ENDS
```

trans() and inc() & dec()

```
; Function compile flags: /Odtp /RTCu
_TEXT    SEGMENT
_a$ = 8         ; size = 4
_f$ = 12        ; size = 4
_b$ = 16        ; size = 4
_trans    PROC

; 12   : {

    push   ebp
    mov    ebp, esp

; 13   :     return a + f(b);

    mov    eax, DWORD PTR _b$[ebp]
    push   eax
    // Function Pointer
    call   DWORD PTR _f$[ebp]
    add    esp, 4
    add    eax, DWORD PTR _a$[ebp]

; 14   : }

    pop    ebp
    ret    0
_trans    ENDP
_TEXT     ENDS
```

```
_TEXT    SEGMENT
_x$ = 8       ; size = 4
_dec    PROC
; 7    : {
    push   ebp
    mov    ebp, esp
; 8    :     return x - 1;
    mov    eax, DWORD PTR _x$[ebp]
    sub    eax, 1
; 9    : }
    pop    ebp
    ret    0
_dec    ENDP
_TEXT   ENDS

_TEXT    SEGMENT
_x$ = 8       ; size = 4
_inc    PROC
; 2    : {
    push   ebp
    mov    ebp, esp

; 3    :     return x + 1;
    mov    eax, DWORD PTR _x$[ebp]
    add    eax, 1
; 4    : }
    pop    ebp
    ret    0
_inc    ENDP
_TEXT    ENDS
```

Nested Blocks

Example: Nested Blocks: Source & TAC

```
int a;
int f(int x) { // function scope f
    int t, u;
    t = x; // t in f, x in f
    { // un-named block scope f_1
         int p, q, t;
         p = a; // p in f_1, a in global
         t = 4; // t in f_1, hides t in f
         { // un-named block scope f_1_1
             int p;
             p = 5; // p in f_1_1, hides p in f_1
         }
         q = p; // q in f_1, p in f_1
    }
    return u = t; // u in f, t in f
}
```

```
f: // function scope f
   // t in f, x in f
   t = x
   // p in f_1, a in global
   p@f_1 = a@glb
   // t in f_1, hides t in f
   t@f_1 = 4
   // p in f_1_1, hides p in f_1
   p@f_1_1 = 5
   // q in f_1, p in f_1
   q@f_1 = p@f_1
   // u in f, t in f
   u = t
```

ST.glb: ST.glb.parent = null

a	int	global	4	0	null
f	int → int	func	0	0	ST.f

ST.f(): ST.f.parent = ST.glb

x	int	param	4	0	null
t	int	local	4	4	null
u	int	local	4	8	null
f_1	null	block	–		ST.f_1

ST.f_1: ST.f_1.parent = ST.f

p	int	local	4	0	null
q	int	local	4	4	null
t	int	local	4	8	null
f_1_1	null	block	–		ST.f_1_1

ST.f_1_1: ST.f_1_1.parent = ST.f_1

p	int	local	4	0	null

Columns: Name, Type, Category, Size, Offset, & Symtab

Grammar and Parsing for this example is discussed with the Parse Tree in 3-Address Code Generation

Nested Blocks Flattened

```
f: // function scope f
   // t in f, x in f
   t = x
   // p in f_1, a in global
   p@f_1 = a@glb
   // t in f_1, hides t in f
   t@f_1 = 4
   // p in f_1_1, hides p in f_1
   p@f_1_1 = 5
   // q in f_1, p in f_1
   q@f_1 = p@f_1
   // u in f, t in f
   u = t
```

ST.f(): ST.f.parent = ST.glb					
x	int	param	4	0	null
t	int	local	4	4	null
u	int	local	4	8	null
f_1	null	block	–		ST.f_1

ST.f_1: ST.f_1.parent = ST.f					
p	int	local	4	0	null
q	int	local	4	4	null
t	int	local	4	8	null
f_1_1	null	block	–		ST.f_1_1

ST.f_1_1: ST.f_1_1.parent = ST.f_1					
p	int	local	4	0	null

Columns: Name, Type, Category, Size, Offset, & Symtab

```
f: // function scope f
   // t in f, x in f
   t = x
   // p in f_1, a in global
   p#1 = a@glb  // p@f_1
   // t in f_1, hides t in f
   t#3 = 4      // t@f_1
   // p in f_1_1, hides p in f_1
   p#4 = 5      // p@f_1_1
   // q in f_1, p in f_1
   q#2 = p#1    // q@f_1, p@f_1
   // u in f, t in f
   u = t
```

ST.f(): ST.f.parent = ST.glb					
x	int	param	4	0	null
t	int	local	4	4	null
u	int	local	4	8	null
p#1	int	blk-local	4	0	null
q#2	int	blk-local	4	4	null
t#3	int	blk-local	4	8	null
p#4	int	blk-local	4	0	null

Example: Nested Blocks: main()

```
_DATA    SEGMENT
COMM     _a:DWORD
_DATA    ENDS
PUBLIC    _f
; Function compile flags: /Odtp /RTCsu
_TEXT    SEGMENT
_p$1 = -24 ; size = 4 // p#4
_t$2 = -20 ; size = 4 // t#3
_q$3 = -16 ; size = 4 // q#2
_p$4 = -12 ; size = 4 // p#1
_u$ = -8   ; size = 4
_t$ = -4   ; size = 4
_x$ = 8    ; size = 4
_f    PROC

; 2    : int f(int x) { // function scope f

    push   ebp
    mov    ebp, esp
    sub    esp, 24          ; 00000018H
    mov    eax, -858993460  ; ccccccccH
    mov    DWORD PTR [ebp-24], eax
    mov    DWORD PTR [ebp-20], eax
    mov    DWORD PTR [ebp-16], eax
    mov    DWORD PTR [ebp-12], eax
    mov    DWORD PTR [ebp-8], eax
    mov    DWORD PTR [ebp-4], eax
```

```
; 3    : int t, u;
; 4    : t = x; // t in f, x in f

    mov    eax, DWORD PTR _x$[ebp]
    mov    DWORD PTR _t$[ebp], eax

; 5    : { // un-named block scope f_1
; 6    :   int p, q, t;
; 7    :   p = a; // p in f_1, a in global

    mov    ecx, DWORD PTR _a
    mov    DWORD PTR _p$4[ebp], ecx

; 8    :   t = 4; // t in f_1, hides t in f

    mov    DWORD PTR _t$2[ebp], 4

; 9    :   { // un - named block scope f_1_1
; 10   :     int p;
; 11   :     p = 5; // p in f_1_1, hides p in f_1

    mov    DWORD PTR _p$1[ebp], 5

; 12   :   }
; 13   :   q = p; // q in f_1, p in f_1

    mov    edx, DWORD PTR _p$4[ebp]
    mov    DWORD PTR _q$3[ebp], edx
```

Nested Blocks: main()

```
; 14    :    }
; 15    :    return u = t; // u in f, t in f

    mov     eax, DWORD PTR _t$[ebp]
    mov     DWORD PTR _u$[ebp], eax
    mov     eax, DWORD PTR _u$[ebp]

; 16    : }

    mov     esp, ebp
    pop     ebp
    ret     0
_f    ENDP
_TEXT    ENDS
```

Global / Static / Function / Extern Data

Example : Global & Function Scope: main() & add(): Source & TAC

```
int x, ar[2][3], y;
int add(int x, int y);
double a, b;
int add(int x, int y) {
    int t;
    t = x + y;
    return t;
}
void main() {
    int c;
    x = 1;
    y = ar[x][x];
    c = add(x, y);
    return;
}
```

```
add:     t#1 = x + y
         t = t#1
         return t

main:    t#1 = 1
         x = t#1
         t#2 = x * 12
         t#3 = x * 4
         t#4 = t#2 + t#3
         y = ar[t#4]
         param x
         param y
         c = call add, 2
         return
```

ST.glb: ST.glb.parent = null					
x	int	global	4	0	null
ar	array(2, array(3, int))	global	24	4	null
y	int	global	4	28	null
add	int × int → int	func	0	32	ST.add()
a	double	global	8	32	null
b	double	global	8	40	null
main	void → void	func	0	48	ST.main()

ST.add(): ST.add.parent = ST.glb				
x	int	param	4	0
y	int	param	4	4
t	int	local	4	8
t#1	int	temp	4	12

ST.main(): ST.main.parent = ST.glb				
c	int	local	4	0
t#1	int	temp	4	4
t#2	int	temp	4	8
t#3	int	temp	4	12
t#4	int	temp	4	16

Columns: Name, Type, Category, Size, Offset, & Symtab

Grammar and Parsing for this example is discussed with the Parse Tree in 3-Address Code Generation

Example: Global & Function Scope: main()

```
_DATA    SEGMENT
COMM     _x:DWORD
COMM     _ar:DWORD:06H // 4 * 6 = 24
COMM     _y:DWORD
COMM     _a:QWORD
COMM     _b:QWORD
_DATA    ENDS
PUBLIC     _add
PUBLIC     _main
; Function compile flags: /Odtp /RTCsu
_TEXT    SEGMENT
_c$ = -4       ; size = 4
_main    PROC

; 9     : void main() {

    push   ebp
    mov    ebp, esp
    push   ecx
    mov    DWORD PTR [ebp-4], -858993460
           ; ccccccccH

; 10    :     int c;
; 11    :     x = 1;

    mov    DWORD PTR _x, 1
```

```
; 12    :     y = ar[x][x];

    imul   eax, DWORD PTR _x, 12
    mov    ecx, DWORD PTR _x
    mov    edx, DWORD PTR _ar[eax+ecx*4]
    mov    DWORD PTR _y, edx

; 13    :     c = add(x, y);

    mov    eax, DWORD PTR _y
    push   eax
    mov    ecx, DWORD PTR _x
    push   ecx
    call   _add
    add    esp, 8
    mov    DWORD PTR _c$[ebp], eax

; 14    :     return;
; 15    : }

    xor    eax, eax
    add    esp, 4
    cmp    ebp, esp
    call   __RTC_CheckEsp
    mov    esp, ebp
    pop    ebp
    ret    0
_main    ENDP
_TEXT    ENDS
```

Example: Global & Function Scope: add()

```
; Function compile flags: /Odtp /RTCsu
_TEXT    SEGMENT
_t$ = -4     ; size = 4
_x$ = 8      ; size = 4
_y$ = 12     ; size = 4
_add    PROC

; 4    : int add(int x, int y) {

    push   ebp
    mov    ebp, esp
    push   ecx
    mov    DWORD PTR [ebp-4], -858993460
           ; ccccccccH

; 5    :     int t;
; 6    :     t = x + y;

    mov    eax, DWORD PTR _x$[ebp]
    add    eax, DWORD PTR _y$[ebp]
    mov    DWORD PTR _t$[ebp], eax
```

```
; 7    :     return t;

    mov    eax, DWORD PTR _t$[ebp]

; 8    : }

    mov    esp, ebp
    pop    ebp
    ret    0
_add    ENDP
_TEXT    ENDS
```

Example:Global, Extern & Local Static Data

```
// File Main.c
extern int n;
int Sum(int x) {
    static int lclStcSum = 0;

    lclStcSum += x;
    return lclStcSum;
}
int sum = -1;
void main() {
    int a = n;

    Sum(a);
    a *= a;
    sum = Sum(a);
    return;
}

// File Global.c
int n = 5;
```

```
      lclStcSum = 0
Sum:  lclStcSum = lclStcSum + x
      return lclStcSum

        sum = -1
main:   a = glb_n
        param a
        call Sum, 1
        a = a * a
        param a
        sum = call Sum, 1
        return
```

ST.glb (Main.c)

n	int	extern	4	0
Sum	int → int	func	0	4
sum	int	global	4	0
main	void → void	func	0	8

ST.glb (Global.c)

n	int	global	4	0

Columns are: Name, Type, Category, Size, & Offset

ST.Sum()

x	int	param	4	0
lclStcSum	int	static	4	4

ST.main()

a	int	local	4	0

main()

```
PUBLIC    _sum // Global int sum;
_BSS    SEGMENT
?lclStcSum@?1??Sum@@9@9 DD 01H DUP (?)
    ; ‘Sum’::‘2’::lclStcSum // int lclStcSum = 0;
_BSS    ENDS
_DATA    SEGMENT
_sum    DD    0ffffffffH // int sum = -1;
_DATA    ENDS
PUBLIC    _Sum
PUBLIC    _main
EXTRN    _n:DWORD // extern int n;
; Function compile flags: /Odtp /RTCsu
; File ..\main.c
_TEXT    SEGMENT
_a$ = -4        ; size = 4
_main    PROC

; 13    : void main() {
    push    ebp
    mov     ebp, esp
    push    ecx
    mov     DWORD PTR [ebp-4], -858993460
    ; ccccccccH

; 14    :     int a = n;
    mov     eax, DWORD PTR _n
    mov     DWORD PTR _a$[ebp], eax
; 15    :
```

```
; 16    :     Sum(a);
    mov     ecx, DWORD PTR _a$[ebp]
    push    ecx
    call    _Sum
    add     esp, 4

; 17    :     a *= a;
    mov     edx, DWORD PTR _a$[ebp]
    imul    edx, DWORD PTR _a$[ebp]
    mov     DWORD PTR _a$[ebp], edx

; 18    :     sum = Sum(a);
    mov     eax, DWORD PTR _a$[ebp]
    push    eax
    call    _Sum
    add     esp, 4
    mov     DWORD PTR _sum, eax

; 19    :     return;
; 20    : }
    xor     eax, eax
    add     esp, 4
    cmp     ebp, esp
    call    __RTC_CheckEsp
    mov     esp, ebp
    pop     ebp
    ret     0
_main    ENDP
_TEXT    ENDS
```

Sum()

```
; Function compile flags: /Odtp /RTCsu
; File ..\main.c
_TEXT    SEGMENT
_x$ = 8       ; size = 4
_Sum     PROC

; 4    : {

    push   ebp
    mov    ebp, esp

; 5    :     static int lclStcSum = 0;
; 6    :
; 7    :     lclStcSum += x;

    mov    eax, DWORD PTR ?lclStcSum@?1??Sum@@9@9
    add    eax, DWORD PTR _x$[ebp]
    mov    DWORD PTR ?lclStcSum@?1??Sum@@9@9, eax

; 8    :     return lclStcSum;

    mov    eax, DWORD PTR ?lclStcSum@?1??Sum@@9@9

; 9    : }

    pop    ebp
    ret    0
_Sum     ENDP
_TEXT    ENDS
```

```
    TITLE    $HOME\Global.c
PUBLIC    _n // int n;
_DATA    SEGMENT
_n     DD     05H // int n = 5;
_DATA    ENDS
END
```

AR in VS: Mixed

Typical Code Snippets

Example: Binary Search

```
int bs(int a[], int l,
       int r, int v) {
    while (l <= r) {
        int m = (l + r) / 2;
        if (a[m] == v)
            return m;
        else
            if (a[m] > v)
                r = m - 1;
            else
                l = m + 1;
    }
    return -1;
}
```

```
100: if l < = r goto 102
101: goto 121
102: t1 = l + r
103: t2 = t1 / 2
104: m = t2
105: t3 = m * 4
106: t4 = a[t3]
107: if t4 == v goto 109
108: goto 111
109: return m
110: goto 100
```

```
111: t5 = m * 4
112: t6 = a[t5]
113: if t6 > v goto 115
114: goto 118
115: t7 = m - 1
116: r = t7
117: goto 100
118: t8 = m + 1
119: l = t8
120: goto 100
121: t9 = -1
122: return t9
```

ST.glb				
bs	array(*, int) × int × int× int → int	func	0	0

Columns: Name, Type, Category, Size, & Offset

Temporary variables are numbered in the function scope – the effect of the respective block scope in the numbering is not considered. Hence, we show only a flattened symbol table

ST.bs()				
a	array(*, int)	param	4	+16
l	int	param	4	+12
r	int	param	4	+8
r	int	param	4	+4
m	int	local	4	0
t1	int	temp	4	–4
t2	int	temp	4	–8
t3	int	temp	4	–12
t4	int	temp	4	–16
t5	int	temp	4	–20
t6	int	temp	4	–24
t7	int	temp	4	–28
t8	int	temp	4	–32
t9	int	temp	4	–36

Example: Transpose

```
int main() {
    int a[3][3];
    int i, j;
    for (i = 0; i < 3; ++i) {
        for (j = 0; j < i; ++j) {
            int t;
            t = a[i][j];
            a[i][j] = a[j][i];
            a[j][i] = t;
        }
    }
    return;
}
```

```
100: t01 = 0
101: i = t01
102: t02 = 3
103: if i < t02 goto 108
104: goto 134
105: t03 = i + 1
106: i = t03
107: goto 103
108: t04 = 0
109: j = t04
110: if j < i goto 115
111: goto 105
112: t05 = j + 1
113: j = t05
114: goto 110
115: t06 = 12 * i
116: t07 = 4 * j
117: t08 = t06 + t07
118: t09 = a[t08]
119: t = t09
120: t10 = 12 * i
121: t11 = 4 * j
122: t12 = t10 + t11
123: t13 = 12 * j
124: t14 = 4 * i
125: t15 = t13 + t14
126: t16 = a[t15]
127: a[t12] = t16
128: t17 = 12 * j
129: t18 = 4 * i
130: t19 = t17 + t18
131: a[t19] = t
132: goto 112
133: goto 105
134: return
```

ST.glb

main	void → void	func

ST.main()

a	array(3, array(3, int))	param	4	0
i	int	local	4	−4
j	int	local	4	−8
t01	int	temp	4	−12
t02	int	temp	4	−16
t03	int	temp	4	−20
t04	int	temp	4	−24
t05	int	temp	4	−28
t06	int	temp	4	−32
t07	int	temp	4	−36
t08	int	temp	4	−40
t09	int	temp	4	−44
t10	int	temp	4	−48
t11	int	temp	4	−52
t12	int	temp	4	−56
t13	int	temp	4	−60
t14	int	temp	4	−64
t15	int	temp	4	−68
t16	int	temp	4	−72
t17	int	temp	4	−76
t18	int	temp	4	−80
t19	int	temp	4	−84