Practical Foundations of Debugging

Chapter 10

Arrays and Structures in Memory – Part 1

Designed, written and illustrated by Dmitry Vostokov

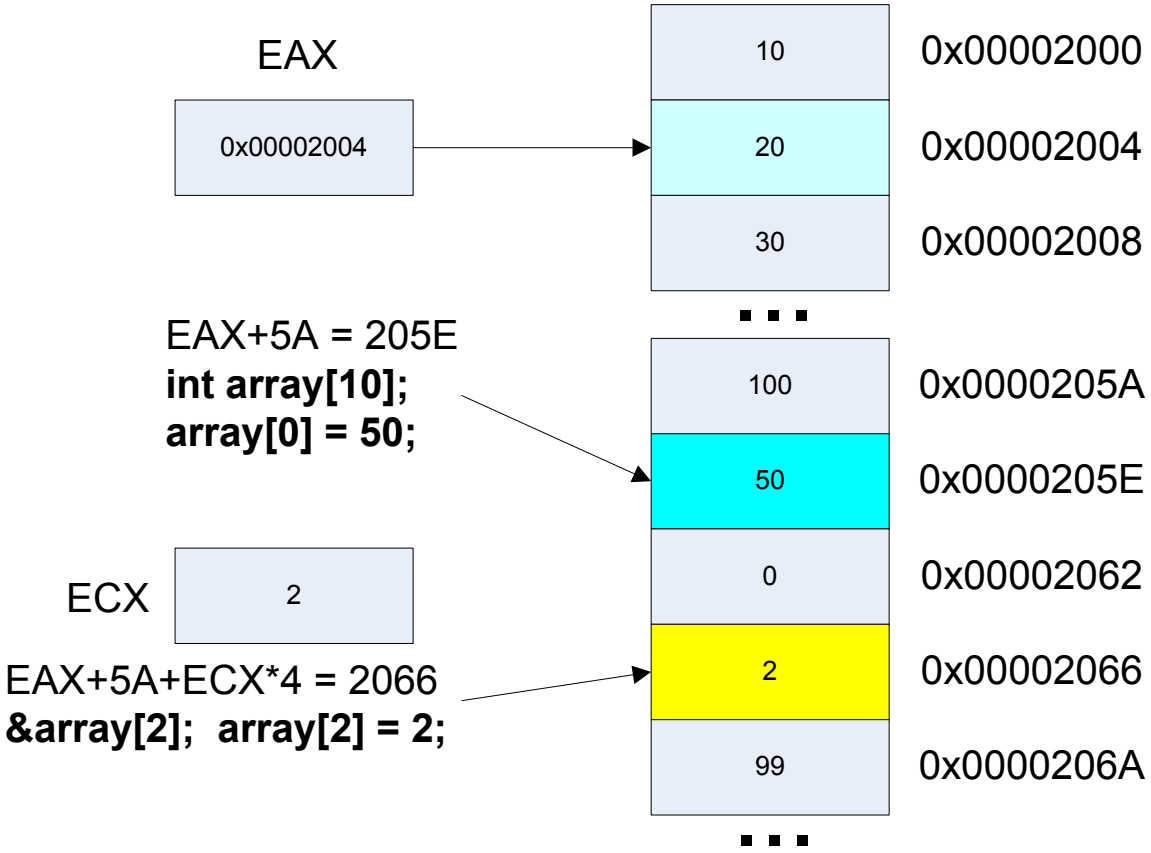
“Memory”

- When we are talking about memory we mean virtual memory
- When you see array manipulations in C or C++ try to mentally visualize/translate them into memory/pointer manipulations

Addressing modes

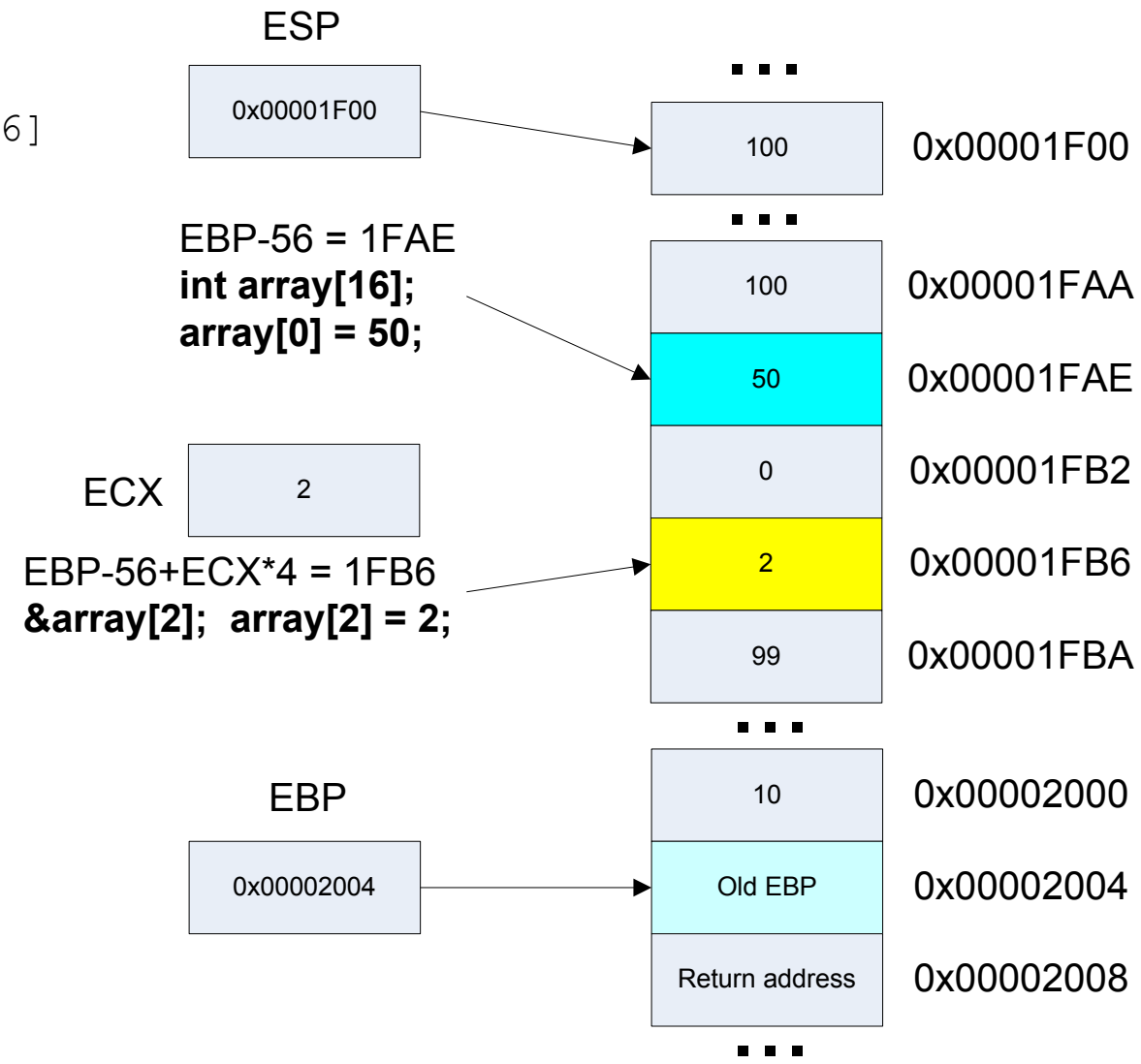
- `[EAX]` `[base]`
- `[EAX+0x5A]` `[base+displacement]`
- `[EAX+ECX*4+0x5A]` `[base+index*scale+displacement]`

  `scale = 1, 2, 4, 8`



Addressing modes – stack based arrays

- `[EBP]`
- `[EBP-0x56]`
- `[EBP+ECX*4-0x56]`



Arrays and pointers relationship rule

- Arrays are always passed as a pointer to the first array element

```
int main()
{
   int array[10];

   func(array, 10);
   func2(array, 10);

   return 0;
}
```

- The following declarations are equivalent:

```
void func(int *array, int numElem);
void func2(int array[], int numElem)
{
   array[0] = 0;
}
```

Arrays and pointers are not interchangeable in general

```
int array[10];
int *pElement;

pElement = &array[1];  // pointer to 2nd element
pElement = &array[2];  // pointer to 3rd element

array = &array[1];     // error: cannot convert from 'int *' to 'int [10]'
```

Why?

- 'array' is r-value, compile time-constant, doesn't have it's own address
- pElement is l-value, it's a pointer and it has it's own address

```
pElement = array;      // OK
```

Rule for indexing arrays and pointers

```
int array[10];
int *pElement;

pElement = array;

array[i] = 0;
pElement[i] = 0;

Why?

*(array+i) = 0;
*(pElement+i) = 0;

Consequence:

i[array] = 0;        // OK
i[pElement] = 0;     // OK
```

Pointer arithmetic internals

```
Type array[N];
Type *pType;
int i;

// In assembly language we use raw (byte) addresses regardless of
   actual Type

pType + i;      *(pType + i) ~ pType[i]
array + i;      *(array + i) ~ array[i]
// in C/C++ means calculate address of i-th element
// in assembly language we calculate byte address of i-th element
//    pType + i*sizeof(Type)
//    array + i*sizeof(Type)
```

Indexed access

```
int array[SIZE];

// using indexes
for (int i = 0; i < SIZE; ++i)
   array[i] = 0;

for (int i = 0; i < sizeof(array)/sizeof(array[0]); ++i)
   array[i] = 0;

// using pointers
int *pElement = array;
pElement = &array[0];

int *pNextToLastElement = array+SIZE;
pNextToLastElement = &array[SIZE];

for (; pElement != pNextToLastElement; ++pElement)
   *pElement = 0;
```

Filling memory

```
// using initialization

void func()
{
    int array2El[2] = { 1, 2 };
    int array[SIZE] = {1}; // fills the rest with 0's
}

// using memset (to fill array with a constant value)
memset (array, 0, sizeof(array));  // fills bytes, not dwords

// using FillMemory
FillMemory (array, sizeof(array), 0);

// using ZeroMemory
ZeroMemory (array, sizeof(array));

// using SecureZeroMemory (W2K3)
SecureZeroMemory (array, sizeof(array));

void Sample()  // from MSDN
{
  WCHAR szPassword[MAX_PATH];

  if (GetPasswordFromUser(szPassword, MAX_PATH))     // this function retrieves a
                                                      //     password
     UsePassword(szPassword);
  SecureZeroMemory(szPassword, sizeof(szPassword));   // clear the password from memory
}
```

Accessing array elements directly

```
#include "arrayUtils.h"

const int SIZE = 100;

int g_array[SIZE];

int main(int argc, char* argv[])
{
    int arrayNI[SIZE/2];                // uninitialized array
    int array[SIZE/2] = {0};

    __asm nop;                          // No Operation instruction

    g_array[0] = 1;
    g_array[10] = 1;

    g_array[10] += g_array[0];
    ++g_array[0];
    g_array[10] *= g_array[0];

    __asm nop;

    array[0] = 1;
    array[10] = 1;

    array[10] += array[0];
    ++array[0];
    array[10] *= array[0];

    useArrays(g_array, array, arrayNI);  // prevent optimizing arrays away

    return 0;
}
```

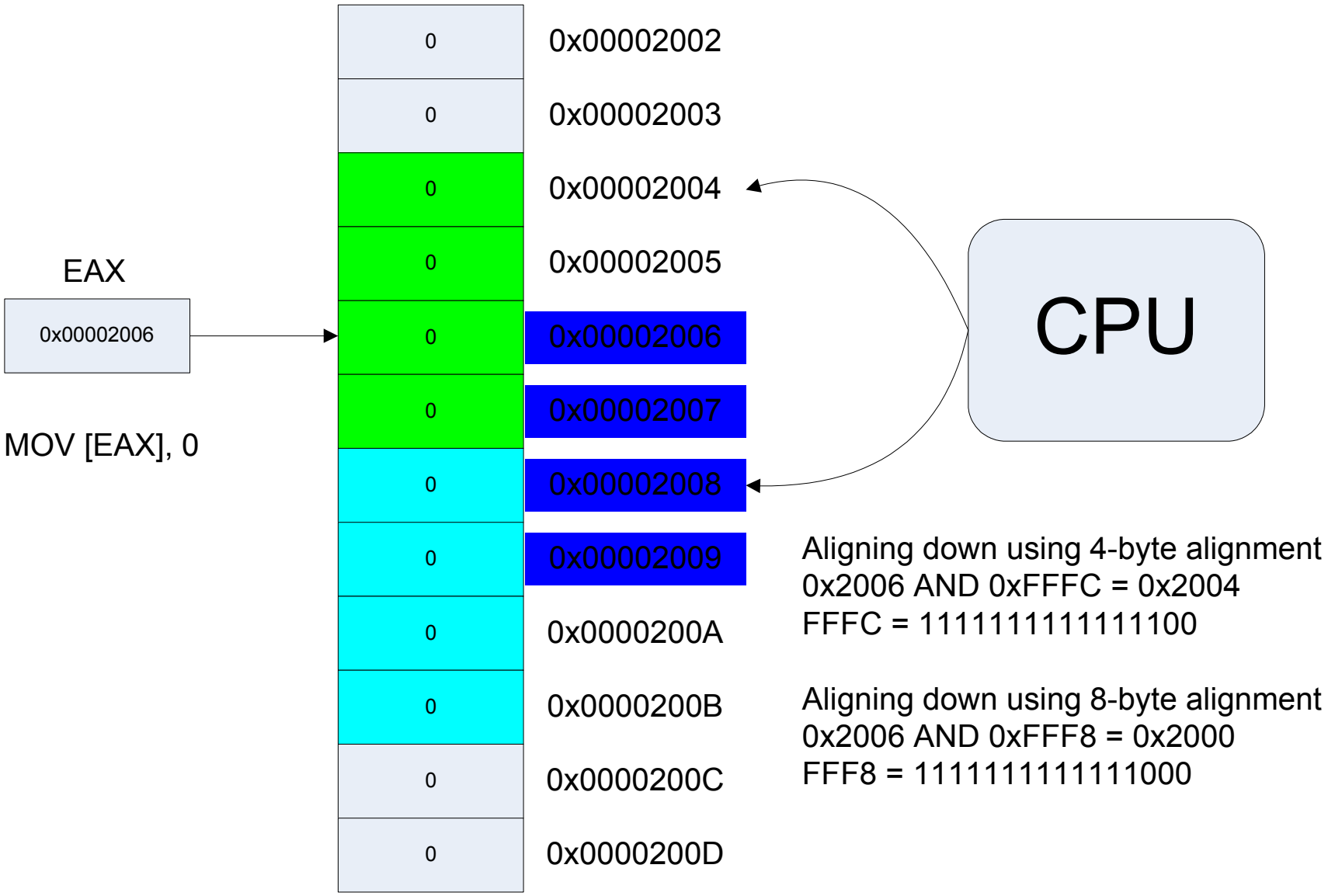
Debug build

```
Arrays!main:
push    ebp
mov     ebp,esp
sub     esp,0x260                       ; reserving arrayNI, array space
push    ebx
push    esi
push    edi
lea     edi,[ebp-0x260]                 ; filling local variables space
mov     ecx,0x98                        ;   with 'cc' pattern
mov     eax,0xcccccccc
rep     stosd
mov     dword ptr [ebp-0x19c],0x0       ; zeroing 1 array integer (pipelining)
mov     ecx,0x31                        ; zeroing 49 array integers
xor     eax,eax
lea     edi,[ebp-0x198]
rep     stosd
nop
mov     dword ptr [Arrays!g_array (00428500)],0x1          ; g_array[0] = 1;
mov     dword ptr [Arrays!g_array+0x28 (00428528)],0x1     ; &g_array[10] = 1;
mov     eax,[Arrays!g_array+0x28 (00428528)]
add     eax,[Arrays!g_array (00428500)]
mov     [Arrays!g_array+0x28 (00428528)],eax
mov     eax,[Arrays!g_array (00428500)]
add     eax,0x1
mov     [Arrays!g_array (00428500)],eax
mov     eax,[Arrays!g_array+0x28 (00428528)]
imul    eax,[Arrays!g_array (00428500)]
mov     [Arrays!g_array+0x28 (00428528)],eax
nop
```

Debug build - continue

```
mov     dword ptr [ebp-0x19c],0x1           ; array[0] = 1;
mov     dword ptr [ebp-0x174],0x1           ; array[10] = 1;
mov     eax,[ebp-0x174]
add     eax,[ebp-0x19c]
mov     [ebp-0x174],eax
mov     eax,[ebp-0x19c]
add     eax,0x1
mov     [ebp-0x19c],eax
mov     eax,[ebp-0x174]
imul    eax,[ebp-0x19c]
mov     [ebp-0x174],eax
lea     eax,[ebp-0xcc]                      ; arrayNI,  &arrayNI[0]
push    eax
lea     ecx,[ebp-0x19c]                     ; array,    &array[0]
push    ecx
push    0x428500                            ; g_array,  &g_array[0]
call    Arrays!ILT+135(?useArraysYAXPAH00Z) (0041108c)
add     esp,0xc
xor     eax,eax
push    edx
mov     ecx,ebp
push    eax
lea     edx,[Arrays!main+0x105 (00411b25)]
call    Arrays!ILT+410(_RTC_CheckStackVars (0041119f)
pop     eax
pop     edx
pop     edi
pop     esi
pop     ebx
add     esp,0x260                           ; adjusting stack pointer
cmp     ebp,esp
call    Arrays!ILT+935(__RTC_CheckEsp) (004113ac)
mov     esp,ebp
pop     ebp
ret
```

Memory alignment



Release build

```
Arrays!main:
push    ebp
mov     ebp,esp
and     esp,0xfffffff8        ; aligning a pointer down to 8-byte boundary
sub     esp,0x194             ; allocating stack for 2 arrays
push    edi
mov     ecx,0x31              ; zeroing 49 integers
xor     eax,eax
lea     edi,[esp+0xc]
mov     dword ptr [esp+0x8],0x0 ; zeroing 1 more (pipelining)
rep     stosd
nop
mov     ecx,0x2               ; optimizations, compiler figured out results
mov     eax,0x4
mov     [Arrays!g_array (004072c0)],ecx
mov     [Arrays!g_array+0x28 (004072e8)],eax
nop
mov     [esp+0x30],eax        ; array[10] = 4
lea     eax,[esp+0xd0]        ; eax := arrayNI
mov     [esp+0x8],ecx         ; array[0] = 2
push    eax
lea     ecx,[esp+0xc]         ; array
push    ecx
push    0x4072c0              ; g_array
call    Arrays!useArrays (00401070)
add     esp,0xc               ; adjusting stack after function call
xor     eax,eax
pop     edi
mov     esp,ebp               ; no need to adjust esp, will use ebp value
pop     ebp
ret
```

Accessing arrays using function parameters

```
#include "arrayUtils.h"

void useArrays(int *array_1, int *array_2, int *array_3)
{
   array_3[0] = 1;
   array_3[10] = 1;

   array_3[10] += array_3[0];
   ++array_3[0];
   array_3[10] *= array_3[0];

   asm nop;

   array_1[0] = 1;
   array_1[10] = 1;
   array_2[0] = 1;
   array_2[10] = 1;

   array_1[10] += array_2[0];
   ++array_2[0];
   array_1[10] *= array_2[0];
}
```

Debug build

```
Arrays!useArrays:
push    ebp
mov     ebp,esp
sub     esp,0xc0
push    ebx
push    esi
push    edi
lea     edi,[ebp-0xc0]
mov     ecx,0x30
mov     eax,0xcccccccc
rep     stosd
mov     eax,[ebp+0x10]                    ; eax := &array_3[0]
mov     dword ptr [eax],0x1
mov     eax,[ebp+0x10]
mov     dword ptr [eax+0x28],0x1
mov     eax,[ebp+0x10]
mov     ecx,[eax+0x28]
mov     edx,[ebp+0x10]
add     ecx,[edx]
mov     eax,[ebp+0x10]
mov     [eax+0x28],ecx
mov     eax,[ebp+0x10]
mov     ecx,[eax]
add     ecx,0x1
mov     edx,[ebp+0x10]
mov     [edx],ecx
mov     eax,[ebp+0x10]
mov     ecx,[ebp+0x10]
mov     edx,[eax+0x28]
imul    edx,[ecx]
mov     eax,[ebp+0x10]
mov     [eax+0x28],edx
nop
```

Debug build - continue

```
mov     eax,[ebp+0x8]                       ; eax := &array_1[0]
mov     dword ptr [eax],0x1
mov     eax,[ebp+0x8]
mov     dword ptr [eax+0x28],0x1
mov     eax,[ebp+0xc]                       ; eax := &array_2[0]
mov     dword ptr [eax],0x1
mov     eax,[ebp+0xc]
mov     dword ptr [eax+0x28],0x1
mov     eax,[ebp+0x8]
mov     ecx,[eax+0x28]
mov     edx,[ebp+0xc]
add     ecx,[edx]
mov     eax,[ebp+0x8]
mov     [eax+0x28],ecx
mov     eax,[ebp+0xc]
mov     ecx,[eax]
add     ecx,0x1
mov     edx,[ebp+0xc]
mov     [edx],ecx
mov     eax,[ebp+0x8]
mov     ecx,[ebp+0xc]
mov     edx,[eax+0x28]
imul    edx,[ecx]
mov     eax,[ebp+0x8]
mov     [eax+0x28],edx
pop     edi
pop     esi
pop     ebx
add     esp,0xc0
cmp     ebp,esp
call    Arrays!ILT+935(__RTC_CheckEsp) (004113ac)
mov     esp,ebp
pop     ebp
ret
```

Release build

```
Arrays!useArrays:
mov     eax,[esp+0xc]             ; eax := &array_3[0]
mov     dword ptr [eax],0x2
mov     dword ptr [eax+0x28],0x4
nop
mov     eax,[esp+0x4]             ; eax := &array_1[0]
mov     ecx,[esp+0x8]             ; ecx := &array_2[0]
mov     edx,0x1
mov     [eax+0x28],edx
mov     [eax],edx
mov     [ecx],edx
mov     [ecx+0x28],edx
inc     dword ptr [eax+0x28]
mov     edx,[ecx]
inc     edx
mov     [ecx],edx
mov     ecx,edx
mov     edx,[eax+0x28]
imul    edx,ecx
mov     [eax+0x28],edx
ret
```

Accessing array elements using indexes

```
int indexedAccess(int *arrayParam, int index)
{
     const int SIZE = 20;
     int array[SIZE];

     // using indexes
     for (int i = 0; i < SIZE; ++i)
            array[i] = 0;

     __asm nop;

     for (int i = 0; i < sizeof(array)/sizeof(array[0]); ++i)
            array[i] = 0;

     __asm nop;

     // using pointers
     int *pElement = array;

     __asm nop;

     pElement = &array[0];

     int *pNextToLastElement = array+SIZE;

     __asm nop;

     pNextToLastElement = &array[SIZE];

     for (; pElement != pNextToLastElement; ++pElement)
            *pElement = 0;

     __asm nop;

     useArrays(array, arrayParam, &arrayParam[1]);

     return arrayParam[index];
}
```

Calling indexedAccess Debug and Release builds

```
int main(int argc, char* argv[])
{
    // …
    __asm nop;

    indexedAccess(g_array, 1);
    indexedAccess(array, 2);

    return 0;
}
```

```
nop
push    0x1
push    0x428500                        ; g_array
call    Arrays!ILT+165(?indexedAccessYAHPAHHZ) (004110aa)
add     esp,0x8
push    0x2                             ; array
lea     eax,[ebp-0x19c]
push    eax
call    Arrays!ILT+165(?indexedAccessYAHPAHHZ) (004110aa)
add     esp,0x8

nop
push    0x1
push    0x4072c0                        ; g_array
call    Arrays!indexedAccess (004010d0)
lea     edx,[esp+0x10]                  ; array
push    0x2
push    edx
call    Arrays!indexedAccess (004010d0)
add     esp,0x10
```

Inside indexedAccess function (Debug build)

```
Arrays!indexedAccess:
00411ce0 55              push    ebp
00411ce1 8bec            mov     ebp,esp
00411ce3 81ec54010000    sub     esp,0x154
00411ce9 53              push    ebx
00411cea 56              push    esi
00411ceb 57              push    edi
00411cec 8dbdacfeffff    lea     edi,[ebp-0x154]
00411cf2 b955000000      mov     ecx,0x55
00411cf7 b8cccccccc      mov     eax,0xcccccccc
00411cfc f3ab            rep     stosd
00411cfe c745f814000000  mov     dword ptr [ebp-0x8],0x14                ; SIZE = 20
00411d05 c7459400000000  mov     dword ptr [ebp-0x6c],0x0                ; int i = 0
00411d0c eb09            jmp     Arrays!indexedAccess+0x37 (00411d17)    ; goto loop comparison expression
00411d0e 8b4594          mov     eax,[ebp-0x6c]                          ; eax := i
00411d11 83c001          add     eax,0x1                                 ; ++eax
00411d14 894594          mov     [ebp-0x6c],eax                          ; i := eax
00411d17 837d9414        cmp     dword ptr [ebp-0x6c],0x14               ; i < SIZE ?
00411d1b 7d0d            jge     Arrays!indexedAccess+0x4a (00411d2a)    ; >= -> exit loop
00411d1d 8b4594          mov     eax,[ebp-0x6c]                          ; eax := i
00411d20 c74485a000000000 mov    dword ptr [ebp+eax*4-0x60],0x0           ; array[i] = 0
00411d28 ebe4            jmp     Arrays!indexedAccess+0x2e (00411d0e)
00411d2a 90              nop
00411d2b c7458800000000  mov     dword ptr [ebp-0x78],0x0                ; another copy of I, second loop
00411d32 eb09            jmp     Arrays!indexedAccess+0x5d (00411d3d)
00411d34 8b4588          mov     eax,[ebp-0x78]
00411d37 83c001          add     eax,0x1
00411d3a 894588          mov     [ebp-0x78],eax
00411d3d 837d8814        cmp     dword ptr [ebp-0x78],0x14
00411d41 730d            jnb     Arrays!indexedAccess+0x70 (00411d50)    ; jnb (jump if not below) ~ jge
00411d43 8b4588          mov     eax,[ebp-0x78]
00411d46 c74485a000000000 mov    dword ptr [ebp+eax*4-0x60],0x0
00411d4e ebe4            jmp     Arrays!indexedAccess+0x54 (00411d34)
00411d50 90              nop
00411d51 8d45a0          lea     eax,[ebp-0x60]                          ; array
00411d54 89857cffffff    mov     [ebp-0x84],eax                          ; pElement = array
00411d5a 90              nop
00411d5b 8d45a0          lea     eax,[ebp-0x60]                          ; &array[0]
00411d5e 89857cffffff    mov     [ebp-0x84],eax                          ; pElement = &array[0]
00411d64 8d45f0          lea     eax,[ebp-0x10]                          ; array+SIZE
00411d67 898570ffffff    mov     [ebp-0x90],eax                          ; pNextToLastElement = array+SIZE
00411d6d 90              nop
```

Inside indexedAccess function (Debug build) - continue

```
00411d6e 8d45f0          lea     eax,[ebp-0x10]                          ; &array[SIZE]
00411d71 898570ffffff    mov     [ebp-0x90],eax                          ; pNextToLastElement = &array[SIZE]
00411d77 eb0f            jmp     Arrays!indexedAccess+0xa8 (00411d88)
00411d79 8b857cffffff    mov     eax,[ebp-0x84]                          ; ++pElement
00411d7f 83c004          add     eax,0x4
00411d82 89857cffffff    mov     [ebp-0x84],eax
00411d88 8b857cffffff    mov     eax,[ebp-0x84]                          ; pElement
00411d8e 3b8570ffffff    cmp     eax,[ebp-0x90]                          ; pElement - pNextToLastElement ?
00411d94 740e            jz      Arrays!indexedAccess+0xc4 (00411da4)    ; equal ? -> exit loop
00411d96 8b857cffffff    mov     eax,[ebp-0x84]                          ; *pElement = 0
00411d9c c70000000000    mov     dword ptr [eax],0x0
00411da2 ebd5            jmp     Arrays!indexedAccess+0x99 (00411d79)    ; goto and increment pElement
00411da4 90              nop
00411da5 8b4508          mov     eax,[ebp+0x8]                           ; arrayParam
00411da8 83c004          add     eax,0x4                                 ; arrayParam+4 ~ &arrayParam[1]
00411dab 50              push    eax
00411dac 8b4d08          mov     ecx,[ebp+0x8]                           ; arrayParam
00411daf 51              push    ecx
00411db0 8d55a0          lea     edx,[ebp-0x60]                          ; array
00411db3 52              push    edx
00411db4 e8d3f2ffff      call Arrays!ILT+135(?useArraysYAXPAH00Z) (0041108c)
00411db9 83c40c          add     esp,0xc
00411dbc 8b450c          mov     eax,[ebp+0xc]                           ; eax := index
00411dbf 8b4d08          mov     ecx,[ebp+0x8]                           ; ecx := arrayParam
00411dc2 8b0481          mov     eax,[ecx+eax*4]                         ; arrayParam[index] (return value)
00411dc5 52              push    edx
00411dc6 8bcd            mov     ecx,ebp
00411dc8 50              push    eax
00411dc9 8d15ea1d4100    lea     edx,[Arrays!indexedAccess+0x10a (00411dea)]
00411dcf e8d0f3ffff      call  Arrays!ILT+415(_RTC_CheckStackVars (004111a4)
00411dd4 58              pop     eax
00411dd5 5a              pop     edx
00411dd6 5f              pop     edi
00411dd7 5e              pop     esi
00411dd8 5b              pop     ebx
00411dd9 81c454010000    add     esp,0x154
00411ddf 3bec            cmp     ebp,esp
00411de1 e8cbf5ffff      call    Arrays!ILT+940(__RTC_CheckEsp) (004113b1)
00411de6 8be5            mov     esp,ebp
00411de8 5d              pop     ebp
00411de9 c3              ret
```

Inside indexedAccess function (Release build)

```
// Both functions useArray and indexedAccess are in the same compilation unit

Arrays!indexedAccess:
nop
nop
nop
nop
nop
mov     eax,[esp+0x4]                   ; arrayParam                    ;
mov     dword ptr [eax+0x4],0x2         ; arrayParam[1]
mov     dword ptr [eax+0x2c],0x4        ; arrayParam[11]
nop
mov     ecx,[esp+0x8]                   ; index
mov     dword ptr [eax+0x28],0x1        ; arrayParam[10]
mov     dword ptr [eax],0x2             ; arrayParam[0]
mov     eax,[eax+ecx*4]                 ; arrayParam[index]
ret

// If you think VC7 compiler is too smart see next slides where I put both functions
    into different compilation units
```

Inside indexedAccess function (Release build, different compilation units, Speed)

```
Arrays!indexedAccess:
004010d0 83ec50          sub     esp,0x50                ; allocating space for array[20]
004010d3 33c0            xor     eax,eax                 ; zeroing all elements, pipelining, first loop
004010d5 890424          mov     [esp],eax
004010d8 89442404        mov     [esp+0x4],eax
004010dc 89442408        mov     [esp+0x8],eax
004010e0 8944240c        mov     [esp+0xc],eax
004010e4 89442410        mov     [esp+0x10],eax
004010e8 89442414        mov     [esp+0x14],eax
004010ec 89442418        mov     [esp+0x18],eax
004010f0 8944241c        mov     [esp+0x1c],eax
004010f4 89442420        mov     [esp+0x20],eax
004010f8 89442424        mov     [esp+0x24],eax
004010fc 89442428        mov     [esp+0x28],eax
00401100 8944242c        mov     [esp+0x2c],eax
00401104 89442430        mov     [esp+0x30],eax
00401108 89442434        mov     [esp+0x34],eax
0040110c 89442438        mov     [esp+0x38],eax
00401110 8944243c        mov     [esp+0x3c],eax
00401114 89442440        mov     [esp+0x40],eax
00401118 89442444        mov     [esp+0x44],eax
0040111c 89442448        mov     [esp+0x48],eax
00401120 56              push    esi
00401121 89442450        mov     [esp+0x50],eax
00401125 90              nop
00401126 33c9            xor     ecx,ecx                 ; second loop
00401128 894c2404        mov     [esp+0x4],ecx
0040112c 894c2408        mov     [esp+0x8],ecx
00401130 894c240c        mov     [esp+0xc],ecx
00401134 894c2410        mov     [esp+0x10],ecx
00401138 894c2414        mov     [esp+0x14],ecx
0040113c 894c2418        mov     [esp+0x18],ecx
00401140 894c241c        mov     [esp+0x1c],ecx
00401144 894c2420        mov     [esp+0x20],ecx
00401148 894c2424        mov     [esp+0x24],ecx
0040114c 894c2428        mov     [esp+0x28],ecx
00401150 894c242c        mov     [esp+0x2c],ecx
00401154 894c2430        mov     [esp+0x30],ecx
00401158 894c2434        mov     [esp+0x34],ecx
0040115c 894c2438        mov     [esp+0x38],ecx
00401160 894c243c        mov     [esp+0x3c],ecx
00401164 894c2440        mov     [esp+0x40],ecx
00401168 894c2444        mov     [esp+0x44],ecx
0040116c 894c2448        mov     [esp+0x48],ecx
00401170 894c244c        mov     [esp+0x4c],ecx
00401174 894c2450        mov     [esp+0x50],ecx
00401178 90              nop
```

Inside indexedAccess function (Release build, different compilation units) - continue

```
00401179 90              nop
0040117a 90              nop
0040117b 33d2            xor     edx,edx                 ; third loop
0040117d 89542404        mov     [esp+0x4],edx
00401181 89542408        mov     [esp+0x8],edx
00401185 8954240c        mov     [esp+0xc],edx
00401189 89542410        mov     [esp+0x10],edx
0040118d 89542414        mov     [esp+0x14],edx
00401191 89542418        mov     [esp+0x18],edx
00401195 8954241c        mov     [esp+0x1c],edx
00401199 89542420        mov     [esp+0x20],edx
0040119d 89542424        mov     [esp+0x24],edx
004011a1 89542428        mov     [esp+0x28],edx
004011a5 8954242c        mov     [esp+0x2c],edx
004011a9 89542430        mov     [esp+0x30],edx
004011ad 89542434        mov     [esp+0x34],edx
004011b1 89542438        mov     [esp+0x38],edx
004011b5 8954243c        mov     [esp+0x3c],edx
004011b9 89542440        mov     [esp+0x40],edx
004011bd 89542444        mov     [esp+0x44],edx
004011c1 89542448        mov     [esp+0x48],edx
004011c5 8954244c        mov     [esp+0x4c],edx
004011c9 89542450        mov     [esp+0x50],edx
004011cd 90              nop
004011ce 8b742458        mov     esi,[esp+0x58]          ; arrayParam
004011d2 8d4604          lea     eax,[esi+0x4]           ; &arrayParam[1]
004011d5 50              push    eax
004011d6 8d4c2408        lea     ecx,[esp+0x8]           ; array
004011da 56              push    esi
004011db 51              push    ecx
004011dc e8affeffff      call    Arrays!useArrays (00401090)
004011e1 8b542468        mov     edx,[esp+0x68]          ; index
004011e5 8b0496          mov     eax,[esi+edx*4]         ; arrayParam[index]
004011e8 83c40c          add     esp,0xc
004011eb 5e              pop     esi
004011ec 83c450          add     esp,0x50
004011ef c3              ret
```

Inside indexedAccess function (Release build, different compilation units, Size)

```
Arrays!indexedAccess:
004010a3 55              push    ebp
004010a4 8bec            mov     ebp,esp
004010a6 83ec50          sub     esp,0x50
004010a9 56              push    esi
004010aa 57              push    edi
004010ab 6a14            push    0x14                    ; push 20
004010ad 59              pop     ecx                     ; pop 20 into ecx
004010ae 33c0            xor     eax,eax
004010b0 8d7db0          lea     edi,[ebp-0x50]          ; zeroing array
004010b3 f3ab            rep     stosd
004010b5 90              nop
004010b6 6a14            push    0x14
004010b8 59              pop     ecx                     ; ecx := 20
004010b9 8d7db0          lea     edi,[ebp-0x50]          ; zeroing array again, second loop
004010bc f3ab            rep     stosd
004010be 90              nop
004010bf 90              nop
004010c0 90              nop
004010c1 6a14            push    0x14                    ; zeroing loop again, third loop
004010c3 59              pop     ecx
004010c4 8d7db0          lea     edi,[ebp-0x50]
004010c7 f3ab            rep     stosd
004010c9 90              nop
004010ca 8b7508          mov     esi,[ebp+0x8]           ; arrayParam
004010cd 8d4604          lea     eax,[esi+0x4]           ; &arrayParam[1]
004010d0 50              push    eax
004010d1 8d45b0          lea     eax,[ebp-0x50]          ; array
004010d4 56              push    esi
004010d5 50              push    eax
004010d6 e891ffffff      call    Arrays!useArrays (0040106c)
004010db 8b450c          mov     eax,[ebp+0xc]           ; index
004010de 8b0486          mov     eax,[esi+eax*4]         ; arrayParam[index]
004010e1 83c40c          add     esp,0xc
004010e4 5f              pop     edi
004010e5 5e              pop     esi
004010e6 c9              leave                           ; size optimization
004010e7 c3              ret
```

Something to think about

What is the difference between a pointer, a reference and a handle?

What's next?

- Arrays and structures – parts 2 and 3
- Virtual memory and paging
- Multithreading, memory and stacks
- Calling Windows functions
  (stdcall vs. cdecl)
- Strings
- Pointers to pointers (LPSTR *)