Example2.c - 1

```
.text:00000000 _sub:      push   ebp
.text:00000001            mov    ebp, esp
.text:00000003            mov    eax, [ebp+8]
.text:00000006            mov    ecx, [ebp+0Ch]
.text:00000009            lea    eax, [ecx+eax*2]
.text:0000000C            pop    ebp
.text:0000000D            retn
.text:00000010 _main:     push   ebp ☒
.text:00000011            mov    ebp, esp
.text:00000013            push   ecx
.text:00000014            mov    eax, [ebp+0Ch]
.text:00000017            mov    ecx, [eax+4]
.text:0000001A            push   ecx
.text:0000001B            call   dword ptr ds:__imp__atoi
.text:00000021            add    esp, 4
.text:00000024            mov    [ebp-4], eax
.text:00000027            mov    edx, [ebp-4]
.text:0000002A            push   edx
.text:0000002B            mov    eax, [ebp+8]
.text:0000002E            push   eax
.text:0000002F            call   _sub
.text:00000034            add    esp, 8
.text:00000037            mov    esp, ebp
.text:00000039            pop    ebp
.text:0000003A            retn
```

eax	**0xcafe ⌘**
ecx	**0xbabe ⌘**
edx	**0xfeed ⌘**
ebp	**0x0012FF50 ⌘**
esp	**0x0012FF24 ♍**

0x0012FF30	0x12FFB0 (char ** argv)⌘
0x0012FF2C	0x2 (int argc) ⌘
0x0012FF28	Addr after “call _main” ⌘
0x0012FF24	0x0012FF50(saved ebp)♍
0x0012FF20	undef
0x0012FF1C	undef
0x0012FF18	undef
0x0012FF14	undef
0x0012FF10	undef
0x0012FF0C	undef

Key: executed instruction ☒, modified value ♍, arbitrary example start value ⌘

Example2.c - 2

```
.text:00000000 _sub:     push   ebp
.text:00000001           mov    ebp, esp
.text:00000003           mov    eax, [ebp+8]
.text:00000006           mov    ecx, [ebp+0Ch]
.text:00000009           lea    eax, [ecx+eax*2]
.text:0000000C           pop    ebp
.text:0000000D           retn
.text:00000010 _main:    push   ebp
.text:00000011           mov    ebp, esp ☒
.text:00000013           push   ecx
.text:00000014           mov    eax, [ebp+0Ch]
.text:00000017           mov    ecx, [eax+4]
.text:0000001A           push   ecx
.text:0000001B           call   dword ptr ds:__imp__atoi
.text:00000021           add    esp, 4
.text:00000024           mov    [ebp-4], eax
.text:00000027           mov    edx, [ebp-4]
.text:0000002A           push   edx
.text:0000002B           mov    eax, [ebp+8]
.text:0000002E           push   eax
.text:0000002F           call   _sub
.text:00000034           add    esp, 8
.text:00000037           mov    esp, ebp
.text:00000039           pop    ebp
.text:0000003A           retn
```

eax	0xcafe
ecx	0xbabe
edx	0xfeed
ebp	0x0012FF24 ♍
esp	0x0012FF24

0x0012FF30	0x12FFB0 (char ** argv)
0x0012FF2C	0x2 (int argc)
0x0012FF28	Addr after “call _main”
0x0012FF24	0x0012FF50 (saved ebp)
0x0012FF20	undef
0x0012FF1C	undef
0x0012FF18	undef
0x0012FF14	undef
0x0012FF10	undef
0x0012FF0C	undef

Example2.c - 3

```
.text:00000000 _sub:     push   ebp
.text:00000001           mov    ebp, esp
.text:00000003           mov    eax, [ebp+8]
.text:00000006           mov    ecx, [ebp+0Ch]
.text:00000009           lea    eax, [ecx+eax*2]
.text:0000000C           pop    ebp
.text:0000000D           retn
.text:00000010 _main:    push   ebp
.text:00000011           mov    ebp, esp
.text:00000013           push   ecx ☒
                         mov    eax, [ebp+0Ch]
                         mov    ecx, [eax+4]
                         push   ecx
                         call   dword ptr ds:__imp__atoi
                         add    esp, 4
                         mov    [ebp-4], eax
                         mov    edx, [ebp-4]
                         push   edx
                         mov    eax, [ebp+8]
                         push   eax
                         call   _sub
                         add    esp, 8
                         mov    esp, ebp
                         pop    ebp
                         retn
```

Caller-save, or space for local var? This time it turns out to be space for local var since there is no corresponding pop, and the address is used later to refer to the value we know is stored in a.

eax	0xcafe
ecx	0xbabe
edx	0xfeed
ebp	0x0012FF24
esp	0x0012FF20 ♍

0x0012FF30	0x12FFB0 (char ** argv)
0x0012FF2C	0x2 (int argc)
0x0012FF28	Addr after “call _main”
0x0012FF24	0x0012FF50 (saved ebp)
0x0012FF20	0xbabe (int a) ♍
0x0012FF1C	undef
0x0012FF18	undef
0x0012FF14	undef
0x0012FF10	undef
0x0012FF0C	undef

Example2.c - 4

```
.text:00000000 _sub:     push   ebp
.text:00000001           mov    ebp, esp
.text:00000003           mov    eax, [ebp+8]
.text:00000006           mov    ecx, [ebp+0Ch]
.text:00000009           lea    eax, [ecx+eax*2]
.text:0000000C           pop    ebp
.text:0000000D           retn
.text:00000010 _main:    push   ebp
.text:00000011           mov    ebp, esp
.text:00000013           push   ecx
.text:00000014           mov    eax, [ebp+0Ch]
                         mov    ecx, [eax+4]
                         push   ecx
                         call   dword ptr ds:__imp__atoi
                         add    esp, 4
                         mov    [ebp-4], eax
                         mov    edx, [ebp-4]
                         push   edx
                         mov    eax, [ebp+8]
                         push   eax
.text:0000002F           call   _sub
.text:00000034           add    esp, 8
.text:00000037           mov    esp, ebp
.text:00000039           pop    ebp
.text:0000003A           retn
```

Getting the base of the argv char * array (aka argv[0])

eax	0x12FFB0 ♍
ecx	0xbabe
edx	0xfeed
ebp	0x0012FF24
esp	0x0012FF20

0x0012FF30	0x12FFB0 (char ** argv)
0x0012FF2C	0x2 (int argc)
0x0012FF28	Addr after “call _main”
0x0012FF24	0x0012FF50 (saved ebp)
0x0012FF20	0xbabe (int a)
0x0012FF1C	undef
0x0012FF18	undef
0x0012FF14	undef
0x0012FF10	undef
0x0012FF0C	undef

Example2 - 5

```
.text:00000000 _sub:     push   ebp
.text:00000001           mov    ebp, esp
.text:00000003           mov    eax, [ebp+8]
.text:00000006           mov    ecx, [ebp+0Ch]
.text:00000009           lea    eax, [ecx+eax*2]
.text:0000000C           pop    ebp
.text:0000000D           retn
.text:00000010 _main:    push   ebp
.text:00000011           mov    ebp, esp
.text:00000013           push   ecx
.text:00000014           mov    eax, [ebp+0Ch]
.text:00000017           mov    ecx, [eax+4] ☒
                         push   ecx
                         call   dword ptr ds:__imp__atoi
                         add    esp, 4
                         mov    [ebp-4], eax
                         mov    edx, [ebp-4]
                         push   edx
                         mov    eax, [ebp+8]
                         push   eax
                         call   _sub
                         add    esp, 8
                         mov    esp, ebp
                         pop    ebp
                         retn
```

Getting the char * at argv[1]
(I chose 0x12FFD4 arbitrarily since it's out of the stack scope we're currently looking at)

eax	0x12FFB0
ecx	0x12FFD4♍(arbitrary⌘)
edx	0xfeed
ebp	0x0012FF24
esp	0x0012FF20

0x0012FF30	0x12FFB0 (char ** argv)
0x0012FF2C	0x2 (int argc)
0x0012FF28	Addr after “call _main”
0x0012FF24	0x0012FF50 (saved ebp)
0x0012FF20	0xbabe (int a)
0x0012FF1C	undef
0x0012FF18	undef
0x0012FF14	undef
0x0012FF10	undef
0x0012FF0C	undef

Example2 - 6

```
.text:00000000 _sub:    push    ebp
.text:00000001          mov     ebp, esp
.text:00000003          mov     eax, [ebp+8]
.text:00000006          mov     ecx, [ebp+0Ch]
.text:00000009          lea     eax, [ecx+eax*2]
                        pop     ebp
                        retn
                        push    ebp
                        mov     ebp, esp
                        push    ecx
                        mov     eax, [ebp+0Ch]
                        mov     ecx, [eax+4]
                        push    ecx ☒
                        call    dword ptr ds:__imp__atoi ☒
                        add     esp, 4 ☒
                        mov     [ebp-4], eax
                        mov     edx, [ebp-4]
                        push    edx
                        mov     eax, [ebp+8]
                        push    eax
                        call    _sub
                        add     esp, 8
                        mov     esp, ebp
                        pop     ebp
                        retn
```

Saving some slides…
This will push the address of the string at argv[1] (0x12FFD4). atoi() will read the string and turn in into an int, put that int in eax, and return. Then the adding 4 to esp will negate the having pushed the input parameter and make 0x12FF1C undefined again (this is indicative of cdecl)

eax	0x100♍ (arbitrary⌘)
ecx	0x12FFD4
edx	0xfeed
ebp	0x0012FF24
esp	0x0012FF20

0x0012FF30	0x12FFB0 (char ** argv)
0x0012FF2C	0x2 (int argc)
0x0012FF28	Addr after “call _main”
0x0012FF24	0x0012FF50 (saved ebp)
0x0012FF20	0xbabe (int a)
0x0012FF1C	undef ♍
0x0012FF18	undef ♍
0x0012FF14	undef
0x0012FF10	undef
0x0012FF0C	undef

Example2 - 7

```
.text:00000000 _sub:     push   ebp
.text:00000001           mov    ebp, esp
.text:00000003           mov    eax, [ebp+8]
.text:00000006           mov    ecx, [ebp+0Ch]
.text:00000009           lea    eax, [ecx+eax*2]
.text:0000000C           pop    ebp
.text:0000000D           retn
.text:00000010 _main:    push   ebp
.text:00000011           mov    ebp, esp
.text:00000013           push   ecx
                         mov    eax, [ebp+0Ch]
                         mov    ecx, [eax+4]
                         push   ecx
                         call   dword ptr ds:__imp__atoi
                         add    esp, 4
                         mov    [ebp-4], eax ☒
                         mov    edx, [ebp-4] ☒
                         push   edx ☒
                         mov    eax, [ebp+8]
                         push   eax
                         call   _sub
                         add    esp, 8
                         mov    esp, ebp
                         pop    ebp
.text:0000003A           retn
```

First setting “a” equal to the return value. Then pushing “a” as the second parameter in sub(). We can see an obvious optimization would have been to replace the last two instructions with “push eax”.

eax	0x100
ecx	0x12FFD4
edx	0x100 ♍
ebp	0x0012FF24
esp	0x0012FF1C ♍

0x0012FF30	0x12FFB0 (char ** argv)
0x0012FF2C	0x2 (int argc)
0x0012FF28	Addr after “call _main”
0x0012FF24	0x0012FF50 (saved ebp)
0x0012FF20	0x100 (int a) ♍
0x0012FF1C	0x100 (int y) ♍
0x0012FF18	undef
0x0012FF14	undef
0x0012FF10	undef
0x0012FF0C	undef

Example2 - 8

```
.text:00000000 _sub:     push   ebp
.text:00000001           mov    ebp, esp
.text:00000003           mov    eax, [ebp+8]
.text:00000006           mov    ecx, [ebp+0Ch]
.text:00000009           lea    eax, [ecx+eax*2]
.text:0000000C           pop    ebp
.text:0000000D           retn
.text:00000010 _main:    push   ebp
.text:00000011           mov    ebp, esp
.text:00000013           push   ecx
.text:00000014           mov    eax, [ebp+0Ch]
.text:00000017           mov    ecx, [eax+4]
.text:0000001A           push   ecx
.text:0000001B           call   dword ptr ds:__imp__atoi
.text:00000021           add    esp, 4
.text:00000024           mov    [ebp-4], eax
.text:00000027           mov    edx, [ebp-4]
.text:0000002A           push   edx
                         mov    eax, [ebp+8] ☒
                         push   eax ☒
                         call   _sub
                         add    esp, 8
                         mov    esp, ebp
                         pop    ebp
.text:0000003A           retn
```

Pushing argc as the first parameter (int x) to sub()

eax	0x2 ♍
ecx	0x12FFD4
edx	0x100
ebp	0x0012FF24
esp	0x0012FF18 ♍

0x0012FF30	0x12FFB0 (char ** argv)
0x0012FF2C	0x2 (int argc)
0x0012FF28	Addr after “call _main”
0x0012FF24	0x0012FF50 (saved ebp)
0x0012FF20	0x100 (int a)
0x0012FF1C	0x100 (int y)
0x0012FF18	0x2 (int x) ♍
0x0012FF14	undef
0x0012FF10	undef
0x0012FF0C	undef

Example2 - 9

```
.text:00000000 _sub:    push   ebp
.text:00000001          mov    ebp, esp
.text:00000003          mov    eax, [ebp+8]
.text:00000006          mov    ecx, [ebp+0Ch]
.text:00000009          lea    eax, [ecx+eax*2]
.text:0000000C          pop    ebp
.text:0000000D          retn
.text:00000010 _main:   push   ebp
.text:00000011          mov    ebp, esp
.text:00000013          push   ecx
.text:00000014          mov    eax, [ebp+0Ch]
.text:00000017          mov    ecx, [eax+4]
.text:0000001A          push   ecx
.text:0000001B          call   dword ptr ds:__imp__atoi
.text:00000021          add    esp, 4
.text:00000024          mov    [ebp-4], eax
.text:00000027          mov    edx, [ebp-4]
.text:0000002A          push   edx
.text:0000002B          mov    eax, [ebp+8]
.text:0000002E          push   eax
.text:0000002F          call   _sub ☒
.text:00000034          add    esp, 8
.text:00000037          mov    esp, ebp
.text:00000039          pop    ebp
.text:0000003A          retn
```

eax	0x2
ecx	0x12FFD4
edx	0x100
ebp	0x0012FF24
esp	0x0012FF14 ♍

0x0012FF30	0x12FFB0 (char ** argv)
0x0012FF2C	0x2 (int argc)
0x0012FF28	Addr after “call _main”
0x0012FF24	0x0012FF50 (saved ebp)
0x0012FF20	0x100 (int a)
0x0012FF1C	0x100 (int y)
0x0012FF18	0x2 (int x)
0x0012FF14	0x00000034 ♍
0x0012FF10	undef
0x0012FF0C	undef

Example2 - 10

```
.text:00000000 _sub:     push    ebp ☒
.text:00000001           mov     ebp, esp ☒
.text:00000003           mov     eax, [ebp+8]
.text:00000006           mov     ecx, [ebp+0Ch]
.text:00000009           lea     eax, [ecx+eax*2]
.text:0000000C           pop     ebp
.text:0000000D           retn
.text:00000010 _main:    push    ebp
.text:00000011           mov     ebp, esp
.text:00000013           push    ecx
.text:00000014           mov     eax, [ebp+0Ch]
.text:00000017           mov     ecx, [eax+4]
.text:0000001A           push    ecx
.text:0000001B           call    dword ptr ds:__imp__atoi
.text:00000021           add     esp, 4
.text:00000024           mov     [ebp-4], eax
.text:00000027           mov     edx, [ebp-4]
.text:0000002A           push    edx
.text:0000002B           mov     eax, [ebp+8]
.text:0000002E           push    eax
.text:0000002F           call    _sub
.text:00000034           add     esp, 8
.text:00000037           mov     esp, ebp
.text:00000039           pop     ebp
.text:0000003A           retn
```

eax	0x2
ecx	0x12FFD4
edx	0x100
ebp	0x0012FF10 ♍
esp	0x0012FF10 ♍

0x0012FF30	0x12FFB0 (char ** argv)
0x0012FF2C	0x2 (int argc)
0x0012FF28	Addr after “call _main”
0x0012FF24	0x0012FF50 (saved ebp)
0x0012FF20	0x100 (int a)
0x0012FF1C	0x100 (int y)
0x0012FF18	0x2 (int x)
0x0012FF14	0x00000034
0x0012FF10	0x0012FF24(saved ebp)♍
0x0012FF0C	undef

Example2 - 11

```
.text:00000000 _sub:    push   ebp
.text:00000001          mov    ebp, esp
                        mov    eax, [ebp+8] ☒
                        mov    ecx, [ebp+0Ch] ☒
                        lea    eax, [ecx+eax*2]
                        pop    ebp
                        retn
.text:00000010 _main:   push   ebp
.text:00000011          mov    ebp, esp
.text:00000013          push   ecx
.text:00000014          mov    eax, [ebp+0Ch]
.text:00000017          mov    ecx, [eax+4]
.text:0000001A          push   ecx
.text:0000001B          call   dword ptr ds:__imp__atoi
.text:00000021          add    esp, 4
.text:00000024          mov    [ebp-4], eax
.text:00000027          mov    edx, [ebp-4]
.text:0000002A          push   edx
.text:0000002B          mov    eax, [ebp+8]
.text:0000002E          push   eax
.text:0000002F          call   _sub
.text:00000034          add    esp, 8
.text:00000037          mov    esp, ebp
.text:00000039          pop    ebp
.text:0000003A          retn
```

Move “x” into eax, and “y” into ecx.

eax	0x2 ♍ (no value change)
ecx	0x100 ♍
edx	0x100
ebp	0x0012FF10
esp	0x0012FF10

0x0012FF30	0x12FFB0 (char ** argv)
0x0012FF2C	0x2 (int argc)
0x0012FF28	Addr after “call _main”
0x0012FF24	0x0012FF50 (saved ebp)
0x0012FF20	0x100 (int a)
0x0012FF1C	0x100 (int y)
0x0012FF18	0x2 (int x)
0x0012FF14	0x00000034
0x0012FF10	0x0012FF24 (saved ebp)
0x0012FF0C	undef

Example2 - 12

```
.text:00000000 _sub:    push   ebp
.text:00000001          mov    ebp, esp
.text:00000003          mov    eax, [ebp+8]
                        mov    ecx, [ebp+0Ch]
                        lea    eax, [ecx+eax*2]
                        pop    ebp
                        retn
                        push   ebp
                        mov    ebp, esp
                        push   ecx
                        mov    eax, [ebp+0Ch]
                        mov    ecx, [eax+4]
                        push   ecx
.text:0000001B          call   dword ptr ds:__imp__atoi
.text:00000021          add    esp, 4
.text:00000024          mov    [ebp-4], eax
.text:00000027          mov    edx, [ebp-4]
.text:0000002A          push   edx
.text:0000002B          mov    eax, [ebp+8]
.text:0000002E          push   eax
.text:0000002F          call   _sub
.text:00000034          add    esp, 8
.text:00000037          mov    esp, ebp
.text:00000039          pop    ebp
.text:0000003A          retn
```

Set the return value (eax) to 2*x + y. Note: neither pointer arith, nor an "address" which was loaded. Just an afficient way to do a calculation.

eax	0x104 ♍
ecx	0x100
edx	0x100
ebp	0x0012FF10
esp	0x0012FF10

0x0012FF30	0x12FFB0 (char ** argv)
0x0012FF2C	0x2 (int argc)
0x0012FF28	Addr after "call _main"
0x0012FF24	0x0012FF50 (saved ebp)
0x0012FF20	0x100 (int a)
0x0012FF1C	0x100 (int y)
0x0012FF18	0x2 (int x)
0x0012FF14	0x00000034
0x0012FF10	0x0012FF24 (saved ebp)
0x0012FF0C	undef

Example2 - 13

```
.text:00000000 _sub:     push   ebp
.text:00000001           mov    ebp, esp
.text:00000003           mov    eax, [ebp+8]
.text:00000006           mov    ecx, [ebp+0Ch]
.text:00000009           lea    eax, [ecx+eax*2]
.text:0000000C           pop    ebp ☒
.text:0000000D           retn
.text:00000010 _main:    push   ebp
.text:00000011           mov    ebp, esp
.text:00000013           push   ecx
.text:00000014           mov    eax, [ebp+0Ch]
.text:00000017           mov    ecx, [eax+4]
.text:0000001A           push   ecx
.text:0000001B           call   dword ptr ds:__imp__atoi
.text:00000021           add    esp, 4
.text:00000024           mov    [ebp-4], eax
.text:00000027           mov    edx, [ebp-4]
.text:0000002A           push   edx
.text:0000002B           mov    eax, [ebp+8]
.text:0000002E           push   eax
.text:0000002F           call   _sub
.text:00000034           add    esp, 8
.text:00000037           mov    esp, ebp
.text:00000039           pop    ebp
.text:0000003A           retn
```

eax	0x104
ecx	0x100
edx	0x100
ebp	0x0012FF24 ♍
esp	0x0012FF14 ♍

0x0012FF30	0x12FFB0 (char ** argv)
0x0012FF2C	0x2 (int argc)
0x0012FF28	Addr after “call _main”
0x0012FF24	0x0012FF50 (saved ebp)
0x0012FF20	0x100 (int a)
0x0012FF1C	0x100 (int y)
0x0012FF18	0x2 (int x)
0x0012FF14	0x00000034
0x0012FF10	undef ♍
0x0012FF0C	undef

Example2 - 14

```
.text:00000000 _sub:     push    ebp
.text:00000001           mov     ebp, esp
.text:00000003           mov     eax, [ebp+8]
.text:00000006           mov     ecx, [ebp+0Ch]
.text:00000009           lea     eax, [ecx+eax*2]
.text:0000000C           pop     ebp
.text:0000000D           retn ☒
.text:00000010 _main:    push    ebp
.text:00000011           mov     ebp, esp
.text:00000013           push    ecx
.text:00000014           mov     eax, [ebp+0Ch]
.text:00000017           mov     ecx, [eax+4]
.text:0000001A           push    ecx
.text:0000001B           call    dword ptr ds:__imp__atoi
.text:00000021           add     esp, 4
.text:00000024           mov     [ebp-4], eax
.text:00000027           mov     edx, [ebp-4]
.text:0000002A           push    edx
.text:0000002B           mov     eax, [ebp+8]
.text:0000002E           push    eax
.text:0000002F           call    _sub
.text:00000034           add     esp, 8
.text:00000037           mov     esp, ebp
.text:00000039           pop     ebp
.text:0000003A           retn
```

eax	0x104
ecx	0x100
edx	0x100
ebp	0x0012FF24
esp	0x0012FF18 ♍

0x0012FF30	0x12FFB0 (char ** argv)
0x0012FF2C	0x2 (int argc)
0x0012FF28	Addr after “call _main”
0x0012FF24	0x0012FF50 (saved ebp)
0x0012FF20	0x100 (int a)
0x0012FF1C	0x100 (int y)
0x0012FF18	0x2 (int x)
0x0012FF14	undef ♍
0x0012FF10	undef
0x0012FF0C	undef

Example2 - 15

```
.text:00000000 _sub:     push   ebp
.text:00000001           mov    ebp, esp
.text:00000003           mov    eax, [ebp+8]
.text:00000006           mov    ecx, [ebp+0Ch]
.text:00000009           lea    eax, [ecx+eax*2]
.text:0000000C           pop    ebp
.text:0000000D           retn
.text:00000010 _main:    push   ebp
.text:00000011           mov    ebp, esp
.text:00000013           push   ecx
.text:00000014           mov    eax, [ebp+0Ch]
.text:00000017           mov    ecx, [eax+4]
.text:0000001A           push   ecx
.text:0000001B           call   dword ptr ds:__imp__atoi
.text:00000021           add    esp, 4
.text:00000024           mov    [ebp-4], eax
.text:00000027           mov    edx, [ebp-4]
.text:0000002A           push   edx
.text:0000002B           mov    eax, [ebp+8]
.text:0000002E           push   eax
.text:0000002F           call   _sub
.text:00000034           add    esp, 8 ☒
.text:00000037           mov    esp, ebp
.text:00000039           pop    ebp
.text:0000003A           retn
```

eax	0x104
ecx	0x100
edx	0x100
ebp	0x0012FF24
esp	0x0012FF20 ♍

0x0012FF30	0x12FFB0 (char ** argv)
0x0012FF2C	0x2 (int argc)
0x0012FF28	Addr after “call _main”
0x0012FF24	0x0012FF50 (saved ebp)
0x0012FF20	0x100 (int a)
0x0012FF1C	undef ♍
0x0012FF18	undef ♍
0x0012FF14	undef
0x0012FF10	undef
0x0012FF0C	undef

Example2 - 16

```
.text:00000000 _sub:     push   ebp
.text:00000001           mov    ebp, esp
.text:00000003           mov    eax, [ebp+8]
.text:00000006           mov    ecx, [ebp+0Ch]
.text:00000009           lea    eax, [ecx+eax*2]
.text:0000000C           pop    ebp
.text:0000000D           retn
.text:00000010 _main:    push   ebp
.text:00000011           mov    ebp, esp
.text:00000013           push   ecx
.text:00000014           mov    eax, [ebp+0Ch]
.text:00000017           mov    ecx, [eax+4]
.text:0000001A           push   ecx
.text:0000001B           call   dword ptr ds:__imp__atoi
.text:00000021           add    esp, 4
.text:00000024           mov    [ebp-4], eax
.text:00000027           mov    edx, [ebp-4]
.text:0000002A           push   edx
.text:0000002B           mov    eax, [ebp+8]
.text:0000002E           push   eax
.text:0000002F           call   _sub
.text:00000034           add    esp, 8
.text:00000037           mov    esp, ebp ☒
.text:00000039           pop    ebp
.text:0000003A           retn
```

eax	0x104
ecx	0x100
edx	0x100
ebp	0x0012FF24
esp	0x0012FF24 ♍

0x0012FF30	0x12FFB0 (char ** argv)
0x0012FF2C	0x2 (int argc)
0x0012FF28	Addr after “call _main”
0x0012FF24	0x0012FF50 (saved ebp)
0x0012FF20	undef ♍
0x0012FF1C	undef
0x0012FF18	undef
0x0012FF14	undef
0x0012FF10	undef
0x0012FF0C	undef

Example2 - 17

```
.text:00000000 _sub:     push   ebp
.text:00000001           mov    ebp, esp
.text:00000003           mov    eax, [ebp+8]
.text:00000006           mov    ecx, [ebp+0Ch]
.text:00000009           lea    eax, [ecx+eax*2]
.text:0000000C           pop    ebp
.text:0000000D           retn
.text:00000010 _main:    push   ebp
.text:00000011           mov    ebp, esp
.text:00000013           push   ecx
.text:00000014           mov    eax, [ebp+0Ch]
.text:00000017           mov    ecx, [eax+4]
.text:0000001A           push   ecx
.text:0000001B           call   dword ptr ds:__imp__atoi
.text:00000021           add    esp, 4
.text:00000024           mov    [ebp-4], eax
.text:00000027           mov    edx, [ebp-4]
.text:0000002A           push   edx
.text:0000002B           mov    eax, [ebp+8]
.text:0000002E           push   eax
.text:0000002F           call   _sub
.text:00000034           add    esp, 8
.text:00000037           mov    esp, ebp
.text:00000039           pop    ebp ☒
.text:0000003A           retn
```

eax	0x104
ecx	0x100
edx	0x100
ebp	0x0012FF50 ♍
esp	0x0012FF28 ♍

0x0012FF30	0x12FFB0 (char ** argv)
0x0012FF2C	0x2 (int argc)
0x0012FF28	Addr after “call _main”
0x0012FF24	undef ♍
0x0012FF20	undef
0x0012FF1C	undef
0x0012FF18	undef
0x0012FF14	undef
0x0012FF10	undef
0x0012FF0C	undef