Lab6 - Code Construct

Forensis Analysis and Incident Management

Andrés Felipe Zapata Rozo
andresf.zapata@urosario.edu.co

September 13, 2020

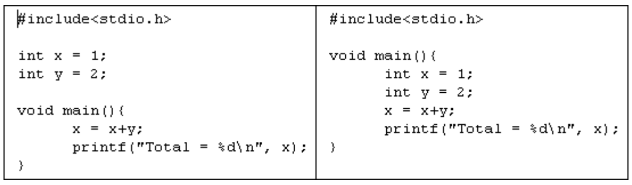
- Read the introduction of the section 6 "Recognizing C Code Constructs in Assembly" and explain what means a "Code Construct". What aspects may impact the way as assembly code is generated?

  The *Code Construct* is a code abstraction that establish functional properties but not details of the implementation, for example the existence of loops and conditionals.

  An aspect that can impact the way as assembly code is generated is the architecture where the source code was compiled.

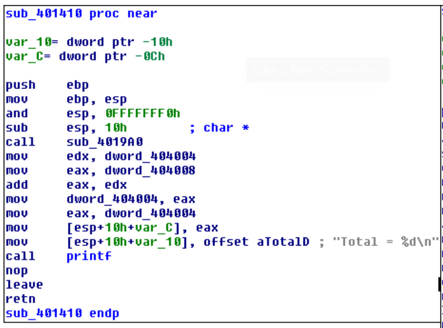
- Read the section "Gobal vs Local Variables" and identify what are the differences in the compilation of a code that employs global vs one that employs local Variables.

  Compilation using local variables initialize the variables in the execution of the function, but using global variables these variables are stored in the *eax* and *edx* register in this case.

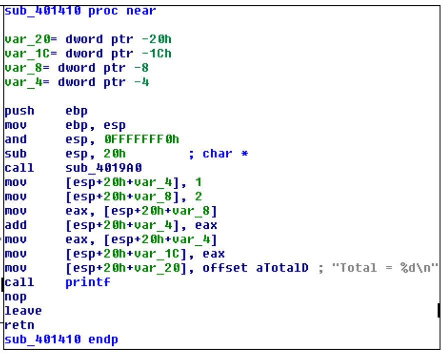


```
#include<stdio.h>

int x = 1;
int y = 2;

void main(){
       x = x+y;
       printf("Total = %d\n", x);
}
```

```
#include<stdio.h>

void main(){
       int x = 1;
       int y = 2;
       x = x+y;
       printf("Total = %d\n", x);
}
```



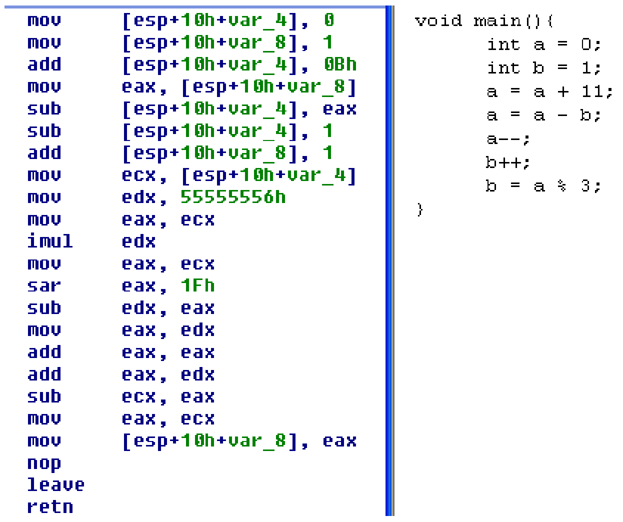
```
sub_401410 proc near

var_10= dword ptr -10h
var_C= dword ptr -0Ch

push    ebp
mov     ebp, esp
and     esp, 0FFFFFFF0h
sub     esp, 10h        ; char *
call    sub_4019A0
mov     edx, dword_404004
mov     eax, dword_404008
add     eax, edx
mov     dword_404004, eax
mov     eax, dword_404004
mov     [esp+10h+var_C], eax
mov     [esp+10h+var_10], offset aTotalD ; "Total = %d\n"
call    printf
nop
leave
retn
sub_401410 endp
```



```
sub_401410 proc near

var_20= dword ptr -20h
var_1C= dword ptr -1Ch
var_8= dword ptr -8
var_4= dword ptr -4

push    ebp
mov     ebp, esp
and     esp, 0FFFFFFF0h
sub     esp, 20h        ; char *
call    sub_4019A0
mov     [esp+20h+var_4], 1
mov     [esp+20h+var_8], 2
mov     eax, [esp+20h+var_8]
add     [esp+20h+var_4], eax
mov     eax, [esp+20h+var_4]
mov     [esp+20h+var_1C], eax
mov     [esp+20h+var_20], offset aTotalD ; "Total = %d\n"
call    printf
nop
leave
retn
sub_401410 endp
```

- Read the section "Disassembling Arithmetic Operations" and explain to your classmates how the operations (addition, subtraction, increment, decrement and modulo) are represented in assembly code.

  The initialization of the variables are in the two first lines of the image, in the line 3 we can see the operation $a = a + 11$, after this, the variable b is moved to the register *eax* to be used in subtraction bellow, the next subtraction and additions correspond to decrease and increase operators, finally the instruction between the line 8 and the line 20 corresponds to the module operation.

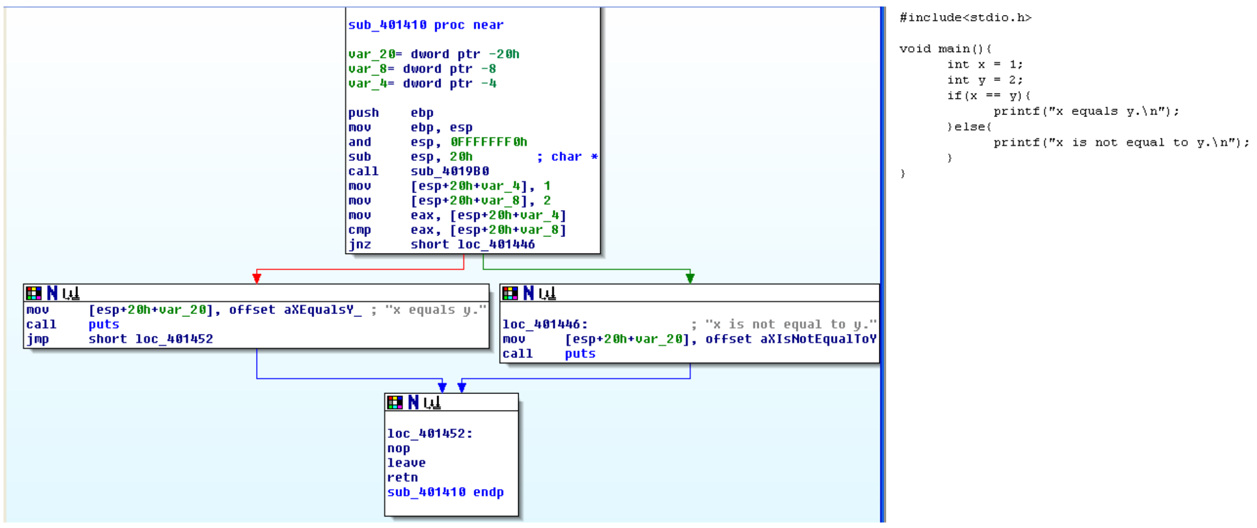


```
mov     [esp+10h+var_4], 0
mov     [esp+10h+var_8], 1
add     [esp+10h+var_4], 0Bh
mov     eax, [esp+10h+var_8]
sub     [esp+10h+var_4], eax
sub     [esp+10h+var_4], 1
add     [esp+10h+var_8], 1
mov     ecx, [esp+10h+var_4]
mov     edx, 55555556h
mov     eax, ecx
imul    edx
mov     eax, ecx
sar     eax, 1Fh
sub     edx, eax
mov     eax, edx
add     eax, eax
add     eax, edx
sub     ecx, eax
mov     eax, ecx
mov     [esp+10h+var_8], eax
nop
leave
retn
```

```
void main(){
        int a = 0;
        int b = 1;
        a = a + 11;
        a = a - b;
        a--;
        b++;
        b = a % 3;
}
```

- Read the section ”Recognizing if Statements” and explain to your classmates how to recognize an if/else structure in assembly code.

  In these case the easy way to recognize and if/else structure is find a *cmp* followed to a *jnz* or similar instruction, and the *else* statement is the instruction followed by the *jnz*. and the *if* statement is execute after the jump.



```
sub_401410 proc near

var_20= dword ptr -20h
var_8= dword ptr -8
var_4= dword ptr -4

push    ebp
mov     ebp, esp
and     esp, 0FFFFFFF0h
sub     esp, 20h        ; char *
call    sub_4019B0
mov     [esp+20h+var_4], 1
mov     [esp+20h+var_8], 2
mov     eax, [esp+20h+var_4]
cmp     eax, [esp+20h+var_8]
jnz     short loc_401446
```

```
mov     [esp+20h+var_20], offset aXEqualsY_ ; "x equals y."
call    puts
jmp     short loc_401452
```

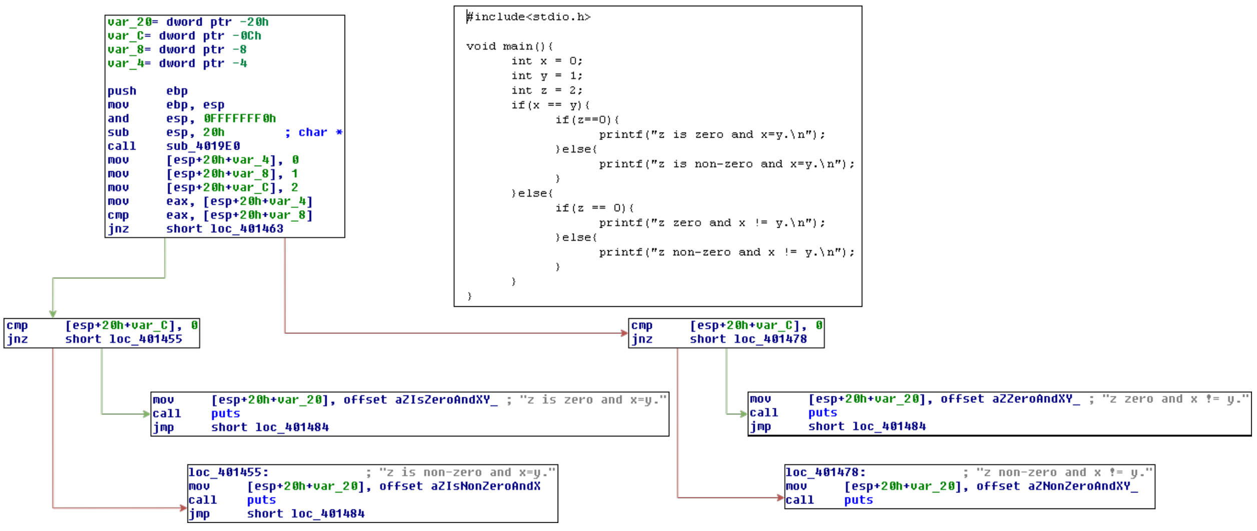
```
loc_401446:                     ; "x is not equal to y."
mov     [esp+20h+var_20], offset aXIsNotEqualToY
call    puts
```

```
loc_401452:
nop
leave
retn
sub_401410 endp
```

```
#include<stdio.h>

void main(){
        int x = 1;
        int y = 2;
        if(x == y){
                printf("x equals y.\n");
        }else{
                printf("x is not equal to y.\n");
        }
}
```

- Read the section ”Recognizing Nested if Statements” and explain to your classmates how to recognize a ”Nested IF” structure in assembly code.

  Similarly to the above we can recognize the outermost statement *if/else* and the nested statements we can find if we follow the directions of the jumps.



```
var_20= dword ptr -20h
var_C= dword ptr -0Ch
var_8= dword ptr -8
var_4= dword ptr -4

push    ebp
mov     ebp, esp
and     esp, 0FFFFFFF0h
sub     esp, 20h          ; char *
call    sub_4019E0
mov     [esp+20h+var_4], 0
mov     [esp+20h+var_8], 1
mov     [esp+20h+var_C], 2
mov     eax, [esp+20h+var_4]
cmp     eax, [esp+20h+var_8]
jnz     short loc_401463
```

```
cmp     [esp+20h+var_C], 0
jnz     short loc_401455
```

```
mov     [esp+20h+var_20], offset aZIsZeroAndXY_ ; "z is zero and x=y."
call    puts
jmp     short loc_401484
```

```
loc_401455:                     ; "z is non-zero and x=y."
mov     [esp+20h+var_20], offset aZIsNonZeroAndX
call    puts
jmp     short loc_401484
```

```
#include<stdio.h>

void main(){
	int x = 0;
	int y = 1;
	int z = 2;
	if(x == y){
		if(z==0){
			printf("z is zero and x=y.\n");
		}else{
			printf("z is non-zero and x=y.\n");
		}
	}else{
		if(z == 0){
			printf("z zero and x != y.\n");
		}else{
			printf("z non-zero and x != y.\n");
		}
	}
}
```

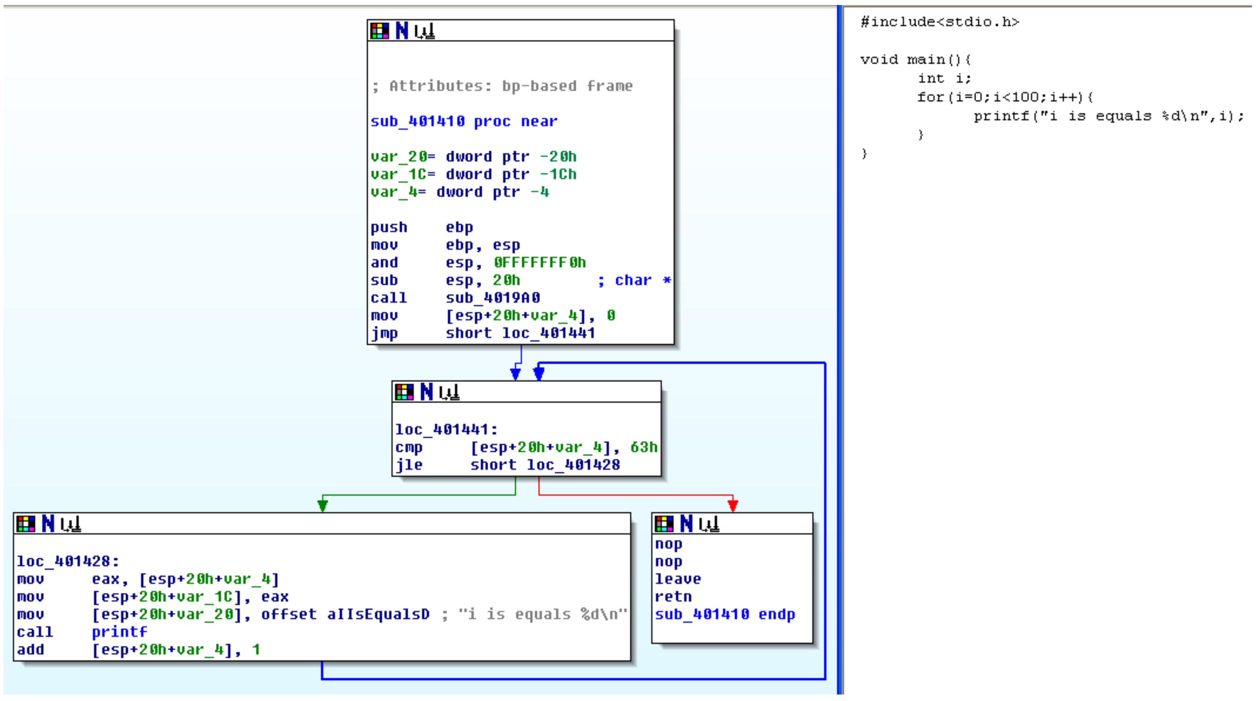
```
cmp     [esp+20h+var_C], 0
jnz     short loc_401478
```

```
mov     [esp+20h+var_20], offset aZZeroAndXY_ ; "z zero and x != y."
call    puts
jmp     short loc_401484
```

```
loc_401478:                     ; "z non-zero and x != y."
mov     [esp+20h+var_20], offset aZNonZeroAndXY_
call    puts
```

- Read the section "Recognizing Loops" and explain to your classmates how to recognize a FOR structure in assembly code.

  To recognizing a loop in assembly is easy when we find instructions like *jl*, *jle*, *jg*, *jge* and the sector where jump go back to the start of the jump.



```
; Attributes: bp-based frame

sub_401410 proc near

var_20= dword ptr -20h
var_1C= dword ptr -1Ch
var_4= dword ptr -4

push    ebp
mov     ebp, esp
and     esp, 0FFFFFFF0h
sub     esp, 20h          ; char *
call    sub_4019A0
mov     [esp+20h+var_4], 0
jmp     short loc_401441
```

```
loc_401441:
cmp     [esp+20h+var_4], 63h
jle     short loc_401428
```

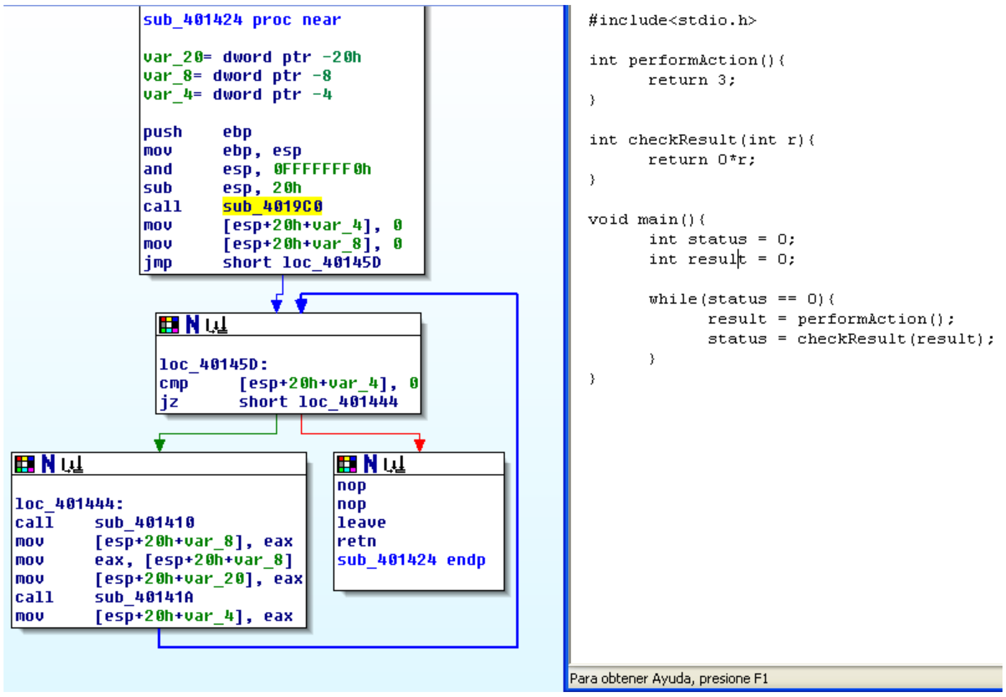
```
loc_401428:
mov     eax, [esp+20h+var_4]
mov     [esp+20h+var_1C], eax
mov     [esp+20h+var_20], offset aIIsEqualsD ; "i is equals %d\n"
call    printf
add     [esp+20h+var_4], 1
```

```
nop
nop
leave
retn
sub_401410 endp
```

```
#include<stdio.h>

void main(){
	int i;
	for(i=0;i<100;i++){
		printf("i is equals %d\n",i);
	}
}
```

- Read the section "Recognizing Loops" and explain to your classmates how to recognize a WHILE structure in assembly code.

  The difference between the for loop and the while loop is that the control variable is wrapped within the statement and the condition of the jump can be more variable.



```
sub_401424 proc near

var_20= dword ptr -20h
var_8= dword ptr -8
var_4= dword ptr -4

push    ebp
mov     ebp, esp
and     esp, 0FFFFFFF0h
sub     esp, 20h
call    sub_4019C0
mov     [esp+20h+var_4], 0
mov     [esp+20h+var_8], 0
jmp     short loc_40145D
```

```
loc_40145D:
cmp     [esp+20h+var_4], 0
jz      short loc_401444
```

```
loc_401444:
call    sub_401410
mov     [esp+20h+var_8], eax
mov     eax, [esp+20h+var_8]
mov     [esp+20h+var_20], eax
call    sub_40141A
mov     [esp+20h+var_4], eax
```

```
nop
nop
leave
retn
sub_401424 endp
```

```
#include<stdio.h>

int performAction(){
        return 3;
}

int checkResult(int r){
        return 0*r;
}

void main(){
        int status = 0;
        int result = 0;

        while(status == 0){
                result = performAction();
                status = checkResult(result);
        }
}
```