Laboratory 6

September 14, 2020 *Andrey Javier Lizarazo Hernández.*

1 *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?*

R:

The meaning of the ***Code Construct*** *is that it is a functional property and it is an abstraction of code, it does not give details of the implementation, but it does include loops, linked lists, declarations, etc.*

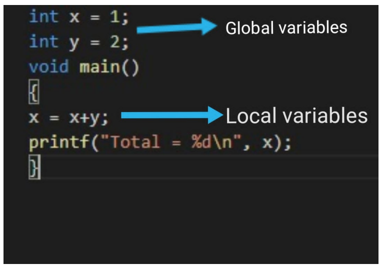
The assembly code presents us with the structure and functionality, but this code can vary too much, depending on the language that the program implements, where it was executed or compiled, and the equipment where the disassembly is done, this can change part of the structure.

2 *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.*

R:
Global variables *can be accessed and used by any function in a program.*

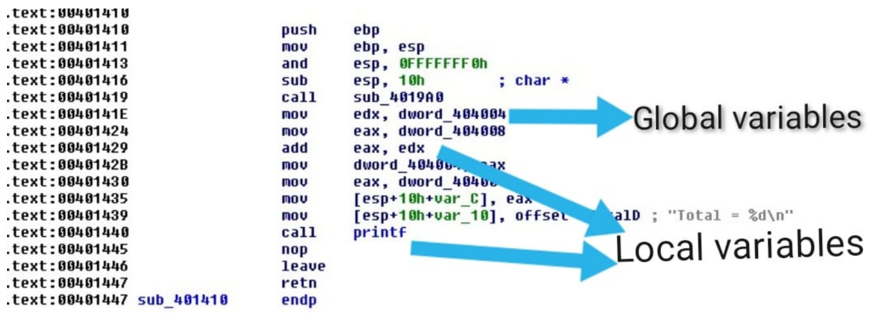
Local variables *can be accessed only by th e function in which they are defined.*

(this can be better appreciated in the following example)



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

(a) Code in c



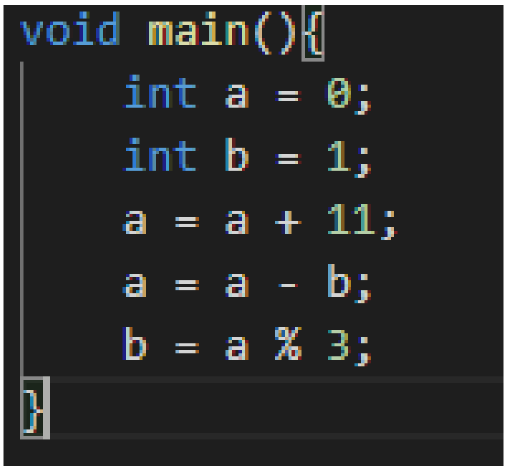
```
.text:00401410
.text:00401410                 push    ebp
.text:00401411                 mov     ebp, esp
.text:00401413                 and     esp, 0FFFFFFF0h
.text:00401416                 sub     esp, 10h        ; char *
.text:00401419                 call    sub_4019A0
.text:0040141E                 mov     edx, dword_404004
.text:00401424                 mov     eax, dword_404008
.text:00401429                 add     eax, edx
.text:0040142B                 mov     dword_40400[illegible] eax
.text:00401430                 mov     eax, dword_4040[illegible]
.text:00401435                 mov     [esp+10h+var_C], eax
.text:00401439                 mov     [esp+10h+var_10], offset [illegible]alD ; "Total = %d\n"
.text:00401440                 call    printf
.text:00401445                 nop
.text:00401446                 leave
.text:00401447                 retn
.text:00401447 sub_401410      endp
```

(b) Disassembled

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

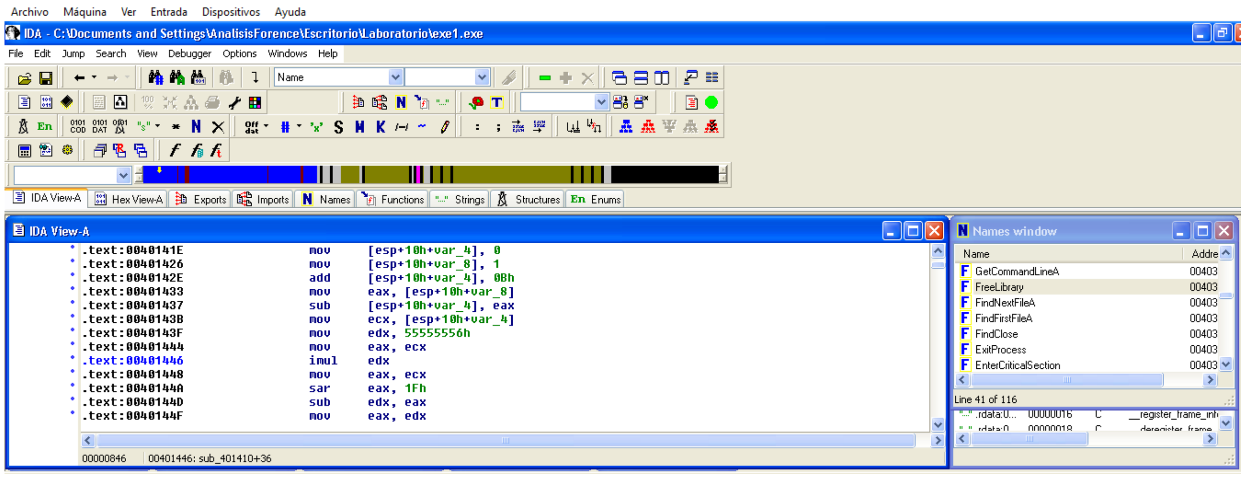
R:*Many types of operations are used when creating code and in programs the most common are (addition, subtraction, increment, decrement and modulus), in a code in c you can easily see how these operations are performed on certain variables , but this changes when you want to disassemble the code already compiled or executed.*

Next we will present an example where a c code is given and disassembled to see how it works already executed:



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

Figure 2: This is the example c code used to perform the disassembly (addition, substraction, increment, decrement and modulo)


Archivo Máquina Ver Entrada Dispositivos Ayuda
IDA - C:\Documents and Settings\AnalisisForence\Escritorio\Laboratorio\exe1.exe
File Edit Jump Search View Debugger Options Windows Help
IDA View-A
Hex View-A
Exports
Imports
Names
Functions
Strings
Structures
Enums
Names window

```
.text:0040141E                 mov     [esp+10h+var_4], 0
.text:00401426                 mov     [esp+10h+var_8], 1
.text:0040142E                 add     [esp+10h+var_4], 0Bh
.text:00401433                 mov     eax, [esp+10h+var_8]
.text:00401437                 sub     [esp+10h+var_4], eax
.text:0040143B                 mov     ecx, [esp+10h+var_4]
.text:0040143F                 mov     edx, 55555556h
.text:00401444                 mov     eax, ecx
.text:00401446                 imul    edx
.text:00401448                 mov     eax, ecx
.text:0040144A                 sar     eax, 1Fh
.text:0040144D                 sub     edx, eax
.text:0040144F                 mov     eax, edx

00000846    00401446: sub_401410+36
```

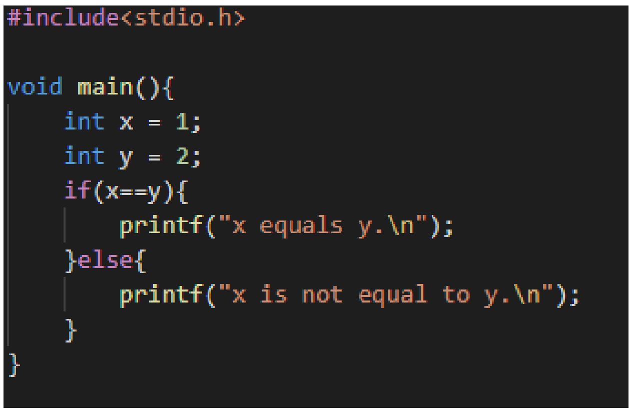
Name	Addre
GetCommandLineA	00403
FreeLibrary	00403
FindNextFileA	00403
FindFirstFileA	00403
FindClose	00403
ExitProcess	00403
EnterCriticalSection	00403

Line 41 of 116

- *The two variables a and b are created with mov and are left with the value of* $[esp + 10h + var_4]$ *and* $[esp + 10h + var_8]$ *each with a value of 0.1 respectively. It should be clarified that they are seen as local variables.*
- *The value of a is stored in an eax and modify the variable that was had and then 0x0b is added to eax, incrementing a by 11. b is then subtracted from a, this is achieved thanks to the add and sub instructions.*
- *Then it is observed how a module is made at the time of being unmounted, happens when performing the div or idiv instruction.*
- *Finally (addition, subtraction, increment, decrement and modulus) are represented in assembly code as (add), (sub), (div), (idiv).*

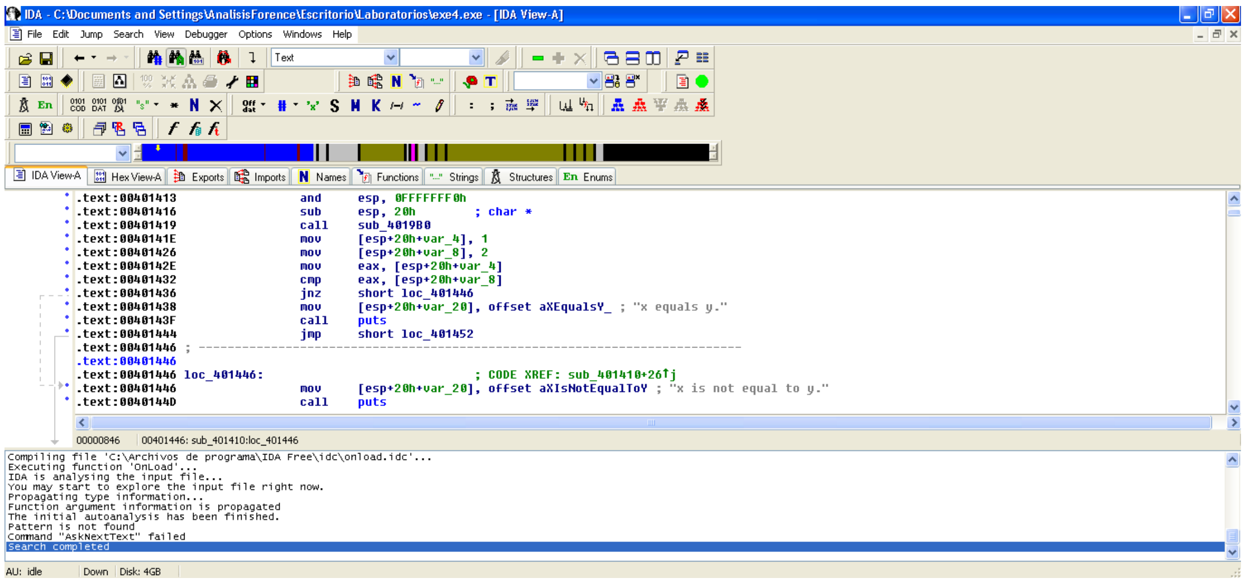
4 *Read the section "Recognizing if Statements" and explain to your classmates how to recognize an if/else structure in assembly code.*

R:
The (if / else) is used to represent logical operations of two variables given a condition, the logic in the disassembly does not change, but the way of knowing that code has these conditions is observed as follows:



```c
#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");
    }
}
```

Figure 3: This is the example c code used to perform the disassembly (if/else)

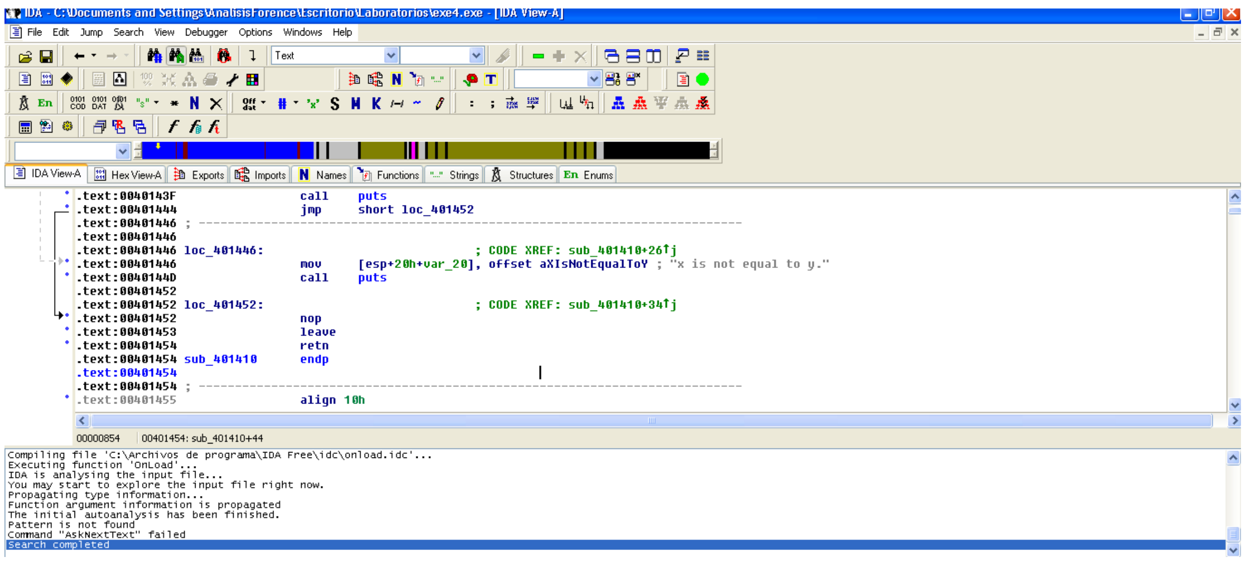

IDA - C:\Documents and Settings\AnalisisForence\Escritorio\Laboratorios\exe4.exe - [IDA View-A]
File Edit Jump Search View Debugger Options Windows Help
Text
IDA View-A
Hex View-A
Exports
Imports
Names
Functions
Strings
Structures
Enums

```
.text:00401413                 and     esp, 0FFFFFFF0h
.text:00401416                 sub     esp, 20h        ; char *
.text:00401419                 call    sub_4019B0
.text:0040141E                 mov     [esp+20h+var_4], 1
.text:00401426                 mov     [esp+20h+var_8], 2
.text:0040142E                 mov     eax, [esp+20h+var_4]
.text:00401432                 cmp     eax, [esp+20h+var_8]
.text:00401436                 jnz     short loc_401446
.text:00401438                 mov     [esp+20h+var_20], offset aXEqualsY_ ; "x equals y."
.text:0040143F                 call    puts
.text:00401444                 jmp     short loc_401452
.text:00401446 ; ---------------------------------------------------------------------------
.text:00401446
.text:00401446 loc_401446:                             ; CODE XREF: sub_401410+26↑j
.text:00401446                 mov     [esp+20h+var_20], offset aXIsNotEqualToY ; "x is not equal to y."
.text:0040144D                 call    puts
```

```
00000846    00401446: sub_401410:loc_401446
```

```
Compiling file 'C:\Archivos de programa\IDA Free\idc\onload.idc'...
Executing function 'OnLoad'...
IDA is analysing the input file...
You may start to explore the input file right now.
Propagating type information...
Function argument information is propagated
The initial autoanalysis has been finished.
Pattern is not found
Command "AskNextText" failed
Search completed
```

```
AU:  idle    Down    Disk: 4GB
```


IDA - C:\Documents and Settings\AnalisisForence\Escritorio\Laboratorios\exe4.exe - [IDA View-A]
File Edit Jump Search View Debugger Options Windows Help
IDA View-A Hex View-A Exports Imports Names Functions Strings Structures Enums

```
.text:0040143F                 call    puts
.text:00401444                 jmp     short loc_401452
.text:00401446 ; ---------------------------------------------------------------------------
.text:00401446
.text:00401446 loc_401446:                             ; CODE XREF: sub_401410+26↑j
.text:00401446                 mov     [esp+20h+var_20], offset aXIsNotEqualToY ; "x is not equal to y."
.text:0040144D                 call    puts
.text:00401452
.text:00401452 loc_401452:                             ; CODE XREF: sub_401410+34↑j
.text:00401452                 nop
.text:00401453                 leave
.text:00401454                 retn
.text:00401454 sub_401410      endp
.text:00401454
.text:00401454 ; ---------------------------------------------------------------------------
.text:00401455                 align 10h
```

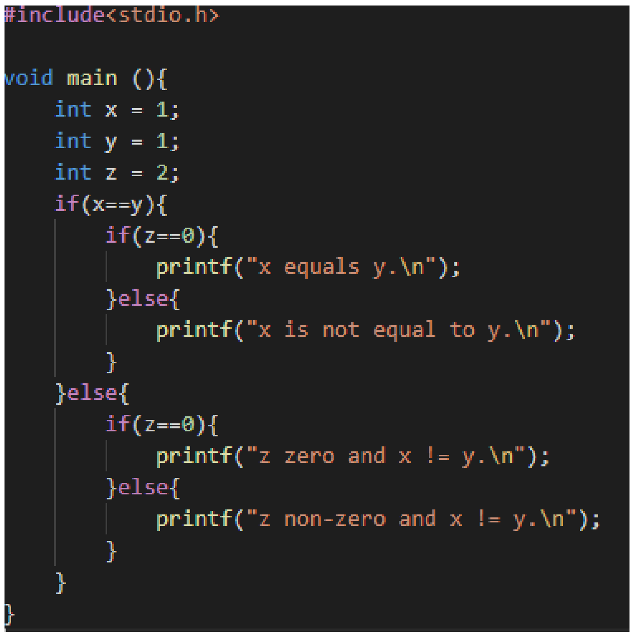
```
00000854    00401454: sub_401410+44
```

```
Compiling file 'C:\Archivos de programa\IDA Free\idc\onload.idc'...
Executing function 'OnLoad'...
IDA is analysing the input file...
You may start to explore the input file right now.
Propagating type information...
Function argument information is propagated
The initial autoanalysis has been finished.
Pattern is not found
Command "AskNextText" failed
Search completed
```

- *The (printf) can be used to obtain keywords and thus found the code disassembled, in this case, you can find in IDA pro the disassembled code using the keywords that are seen in the (printf), such as "x equals y".*
- *The values of x and y are stored in* $[esp + 20h + var_4]$ *and* $[esp + 20h + var_8]$*. Taking the value of 2 and 1 respectively.*
- *Each value is saved in an eax, in order to later put the equality condition. This decision corresponds to the conditional jump (jnz).*
- *If the values are not equal, the jump occurs, and the code prints "x is not equal to y."; otherwise, the code continues the path of execution and prints "x equals y."Notice also the jump (jmp) that jumps over the else section of the code at.*
- *Finally, (if/else) are represented in assembly code as (jump), (jnz).*

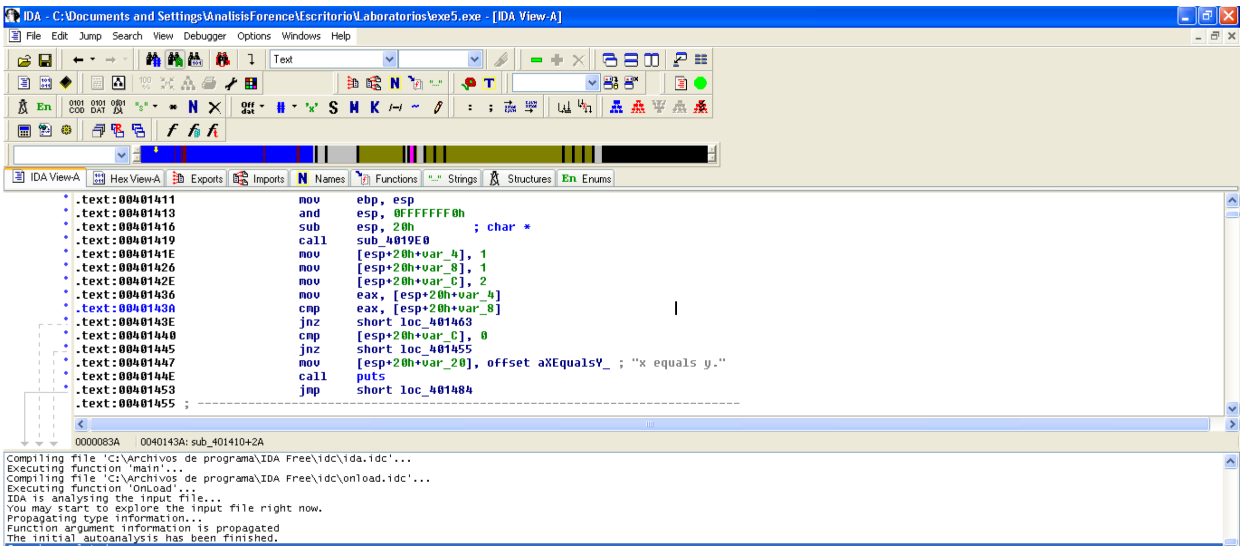
5 Read the section "Recognizing Nested if Statements" and explain to your classmates how to recognize a "Nested IF" structure in assembly code.

R:
The (Nested IF) is used to encapsulate several conditions, in such a way that the main condition is given and if it is fulfilled, it enters the encapsulated conditions. This can be analyzed in the following example



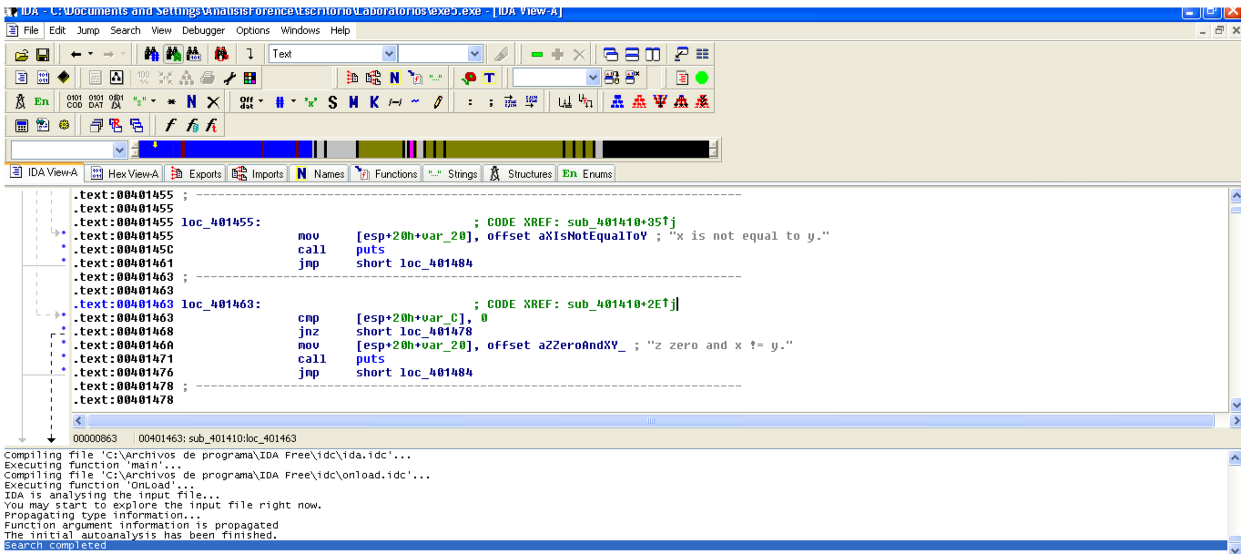
```c
#include<stdio.h>

void main (){
    int x = 1;
    int y = 1;
    int z = 2;
    if(x==y){
        if(z==0){
            printf("x equals y.\n");
        }else{
            printf("x is not equal to y.\n");
        }
    }else{
        if(z==0){
            printf("z zero and x != y.\n");
        }else{
            printf("z non-zero and x != y.\n");
        }
    }
}
```

Figure 4: This is the example c code used to perform the disassembly(Nested IF)

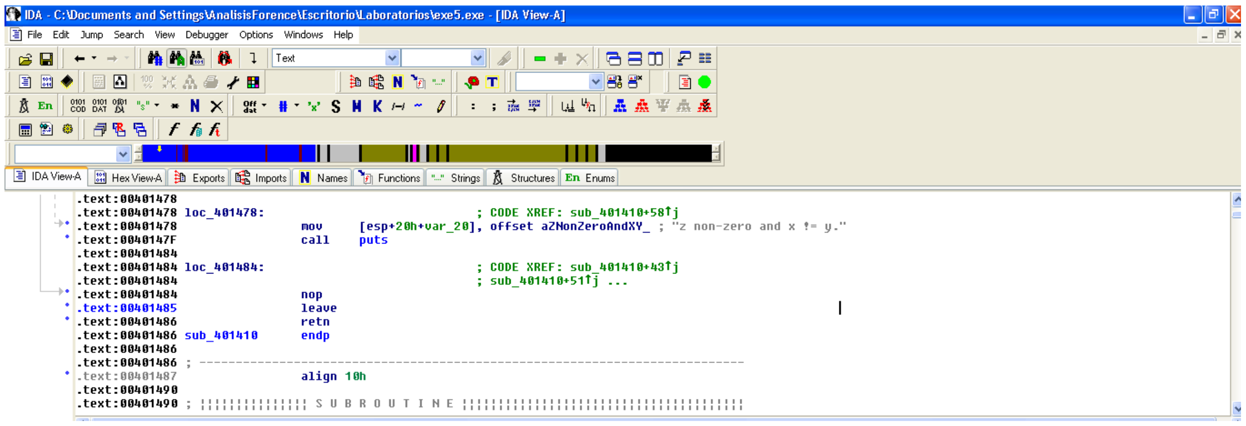

IDA - C:\Documents and Settings\AnalisisForence\Escritorio\Laboratorios\exe5.exe - [IDA View-A]
File Edit Jump Search View Debugger Options Windows Help
Text
IDA View-A
Hex View-A
Exports
Imports
Names
Functions
Strings
Structures
Enums

```
.text:00401411                 mov     ebp, esp
.text:00401413                 and     esp, 0FFFFFFF0h
.text:00401416                 sub     esp, 20h        ; char *
.text:00401419                 call    sub_4019E0
.text:0040141E                 mov     [esp+20h+var_4], 1
.text:00401426                 mov     [esp+20h+var_8], 1
.text:0040142E                 mov     [esp+20h+var_C], 2
.text:00401436                 mov     eax, [esp+20h+var_4]
.text:0040143A                 cmp     eax, [esp+20h+var_8]
.text:0040143E                 jnz     short loc_401463
.text:00401440                 cmp     [esp+20h+var_C], 0
.text:00401445                 jnz     short loc_401455
.text:00401447                 mov     [esp+20h+var_20], offset aXEqualsY_ ; "x equals y."
.text:0040144E                 call    puts
.text:00401453                 jmp     short loc_401484
.text:00401455 ; ---------------------------------------------------------------------------

0000083A    0040143A: sub_401410+2A
```

```
Compiling file 'C:\Archivos de programa\IDA Free\idc\ida.idc'...
Executing function 'main'...
Compiling file 'C:\Archivos de programa\IDA Free\idc\onload.idc'...
Executing function 'OnLoad'...
IDA is analysing the input file...
You may start to explore the input file right now.
Propagating type information...
Function argument information is propagated
The initial autoanalysis has been finished.
```



```
IDA - C:\Documents and Settings\AnalisisForence\Escritorio\Laboratorios\exe5.exe - [IDA View-A]

.text:00401455 ; ---------------------------------------------------------------------------
.text:00401455
.text:00401455 loc_401455:                             ; CODE XREF: sub_401410+35↑j
.text:00401455                 mov     [esp+20h+var_20], offset aXIsNotEqualToY ; "x is not equal to y."
.text:0040145C                 call    puts
.text:00401461                 jmp     short loc_401484
.text:00401463 ; ---------------------------------------------------------------------------
.text:00401463
.text:00401463 loc_401463:                             ; CODE XREF: sub_401410+2E↑j
.text:00401463                 cmp     [esp+20h+var_C], 0
.text:00401468                 jnz     short loc_401478
.text:0040146A                 mov     [esp+20h+var_20], offset aZZeroAndXY_ ; "z zero and x != y."
.text:00401471                 call    puts
.text:00401476                 jmp     short loc_401484
.text:00401478 ; ---------------------------------------------------------------------------
.text:00401478

00000863  00401463: sub_401410:loc_401463

Compiling file 'C:\Archivos de programa\IDA Free\idc\ida.idc'...
Executing function 'main'...
Compiling file 'C:\Archivos de programa\IDA Free\idc\onload.idc'...
Executing function 'OnLoad'...
IDA is analysing the input file...
You may start to explore the input file right now.
Propagating type information...
Function argument information is propagated
The initial autoanalysis has been finished.
Search completed
```



```
IDA - C:\Documents and Settings\AnalisisForence\Escritorio\Laboratorios\exe5.exe - [IDA View-A]

.text:00401478
.text:00401478 loc_401478:                             ; CODE XREF: sub_401410+58↑j
.text:00401478                 mov     [esp+20h+var_20], offset aZNonZeroAndXY_ ; "z non-zero and x != y."
.text:0040147F                 call    puts
.text:00401484
.text:00401484 loc_401484:                             ; CODE XREF: sub_401410+43↑j
.text:00401484                                         ; sub_401410+51↑j ...
.text:00401484                 nop
.text:00401485                 leave
.text:00401486                 retn
.text:00401486 sub_401410      endp
.text:00401486
.text:00401486 ; ---------------------------------------------------------------------------
.text:00401487                 align 10h
.text:00401490
.text:00401490 ; ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦ S U B R O U T I N E ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
```

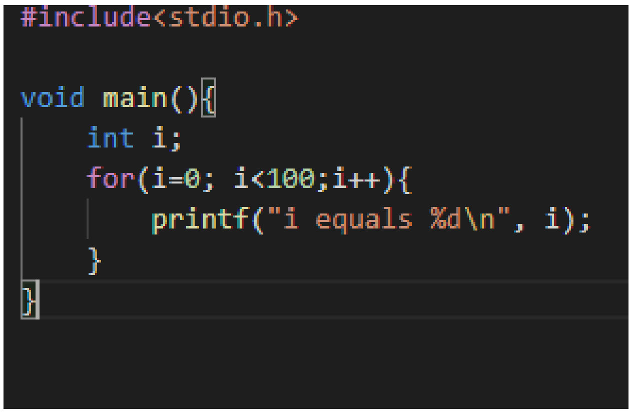
- *The (printf) can be used to obtain keywords and thus found the code disassembled, in this case, you can find in IDA pro the disassembled code using the keywords that are seen in the (printf), such as "x equals y".*
- *The values of x, y, z are stored in* $[esp + 20h + var_4]$, $[esp + 20h + var_8]$ *and* $[esp + 20h + var_c]$. *Taking the value of 1, 1, 2 respectively*
- *As you can see, three different conditional jumps occur. The first occurs if* var_4 *does not equal* var_8 *a.The other two occur if* var_C *is not equal to zero at*
- *This is very similar to the previous example "point 5", but a (Nested IF) can be identified when we see a quantity of (jump), (jnz) together and taking into account the dependence of change in the variables with each conditional.*

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

R:

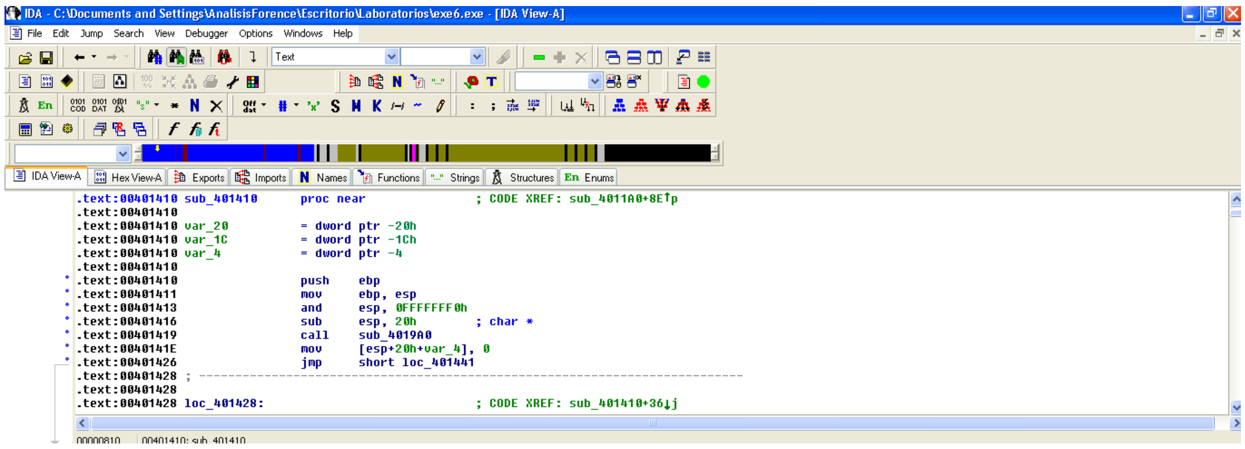
Loops are very important and used in code, one of the most used is the ”For”, ”For” loops always have four components: initialization, comparison, execution instructions, and the increment or decrement.

Next, we will show with an example of how to recognize a ”For”.



```
#include<stdio.h>

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

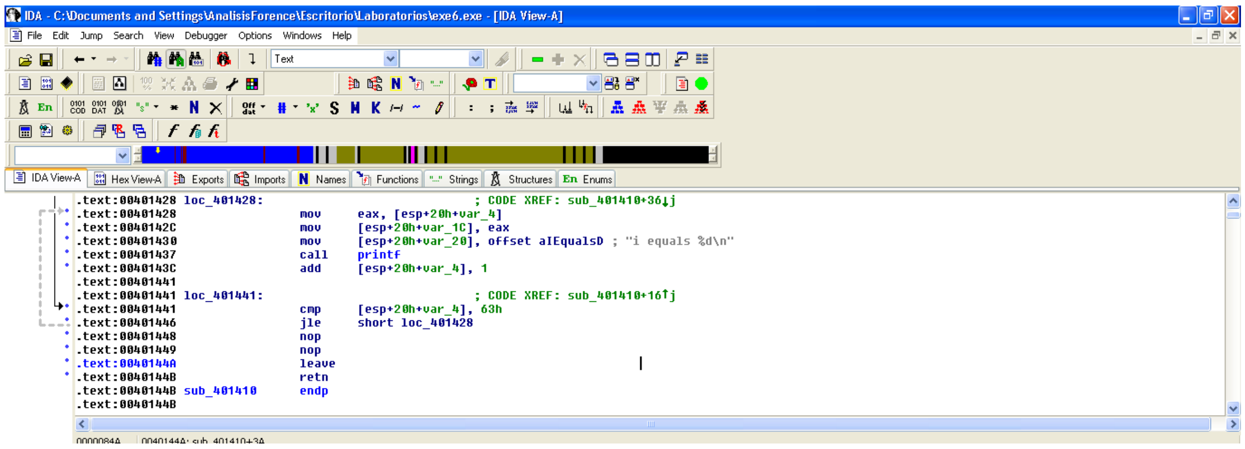
Figure 5: This is the example c code used to perform the disassembly(FOR)



IDA - C:\Documents and Settings\AnalisisForence\Escritorio\Laboratorios\exe6.exe - [IDA View-A]

```
.text:00401410 sub_401410      proc near               ; CODE XREF: sub_4011A0+8E↑p
.text:00401410
.text:00401410 var_20          = dword ptr -20h
.text:00401410 var_1C          = dword ptr -1Ch
.text:00401410 var_4           = dword ptr -4
.text:00401410
.text:00401410                 push    ebp
.text:00401411                 mov     ebp, esp
.text:00401413                 and     esp, 0FFFFFFF0h
.text:00401416                 sub     esp, 20h        ; char *
.text:00401419                 call    sub_4019A0
.text:0040141E                 mov     [esp+20h+var_4], 0
.text:00401426                 jmp     short loc_401441
.text:00401428 ; ---------------------------------------------------------------------------
.text:00401428
.text:00401428 loc_401428:                             ; CODE XREF: sub_401410+36↓j
```

00000810 00401410: sub_401410


IDA - C:\Documents and Settings\AnalisisForence\Escritorio\Laboratorios\exe6.exe - [IDA View-A]
File Edit Jump Search View Debugger Options Windows Help
IDA View-A
Hex View-A
Exports
Imports
Names
Functions
Strings
Structures
Enums

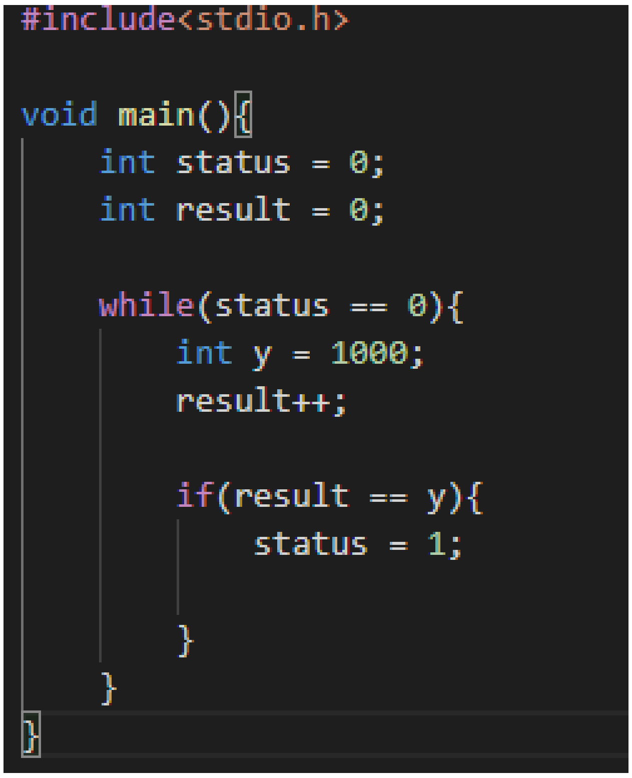
```
.text:00401428 loc_401428:                             ; CODE XREF: sub_401410+36↓j
.text:00401428                 mov     eax, [esp+20h+var_4]
.text:0040142C                 mov     [esp+20h+var_1C], eax
.text:00401430                 mov     [esp+20h+var_20], offset aIEqualsD ; "i equals %d\n"
.text:00401437                 call    printf
.text:0040143C                 add     [esp+20h+var_4], 1
.text:00401441
.text:00401441 loc_401441:                             ; CODE XREF: sub_401410+16↑j
.text:00401441                 cmp     [esp+20h+var_4], 63h
.text:00401446                 jle     short loc_401428
.text:00401448                 nop
.text:00401449                 nop
.text:0040144A                 leave
.text:0040144B                 retn
.text:0040144B sub_401410      endp
.text:0040144B
```

```
0000084A    0040144A: sub_401410+3A
```

- *The (printf) can be used to obtain keywords and thus found the code disassembled, in this case, you can find in IDA pro the disassembled code using the keyword that is seen in the (printf), such as "i equals".*
- *In assembly, the for loop can be recognized by locating the four components—initialization, comparison, execution instructions, and increment/decrement the initial value is given by* $[esp + 20h + var_4]$*, 0.*
- *Then the initial value is modified with what we know eax. the comparison is made using the instructions (jmp), (jge)*
- *Finally, the decision is made by the conditional jump. In this example If the jump is not taken, the printf instruction will execute, and an unconditional jump occurs at.*

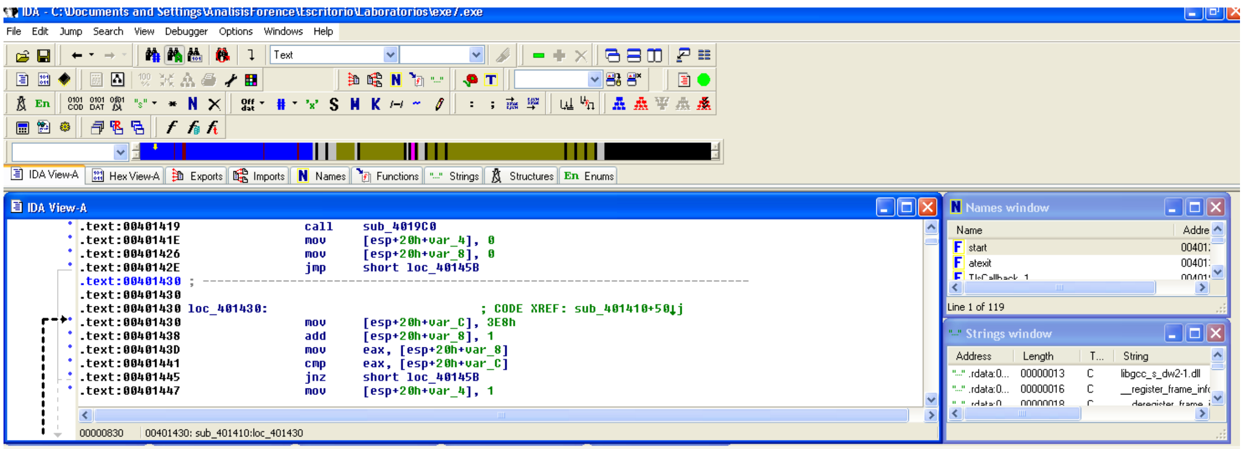
7 Read the section "Recognizing Loops" and explain to your classmates how to recognize a WHILE structure in assembly code.

R:*This loop is also widely used when making a code, the while iterates as many times as possible depending on whether the conditional it has is met. this can be analyzed in IDA pro as follows:*



```c
#include<stdio.h>

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

    while(status == 0){
        int y = 1000;
        result++;

        if(result == y){
            status = 1;

        }
    }
}
```

Figure 6: This is the example c code used to perform the disassembly (WHILE)



```
.text:00401419                 call    sub_4019C0
.text:0040141E                 mov     [esp+20h+var_4], 0
.text:00401426                 mov     [esp+20h+var_8], 0
.text:0040142E                 jmp     short loc_40145B
.text:00401430 ; ---------------------------------------------------------------------------
.text:00401430
.text:00401430 loc_401430:                             ; CODE XREF: sub_401410+50↓j
.text:00401430                 mov     [esp+20h+var_C], 3E8h
.text:00401438                 add     [esp+20h+var_8], 1
.text:0040143D                 mov     eax, [esp+20h+var_8]
.text:00401441                 cmp     eax, [esp+20h+var_C]
.text:00401445                 jnz     short loc_40145B
.text:00401447                 mov     [esp+20h+var_4], 1
```

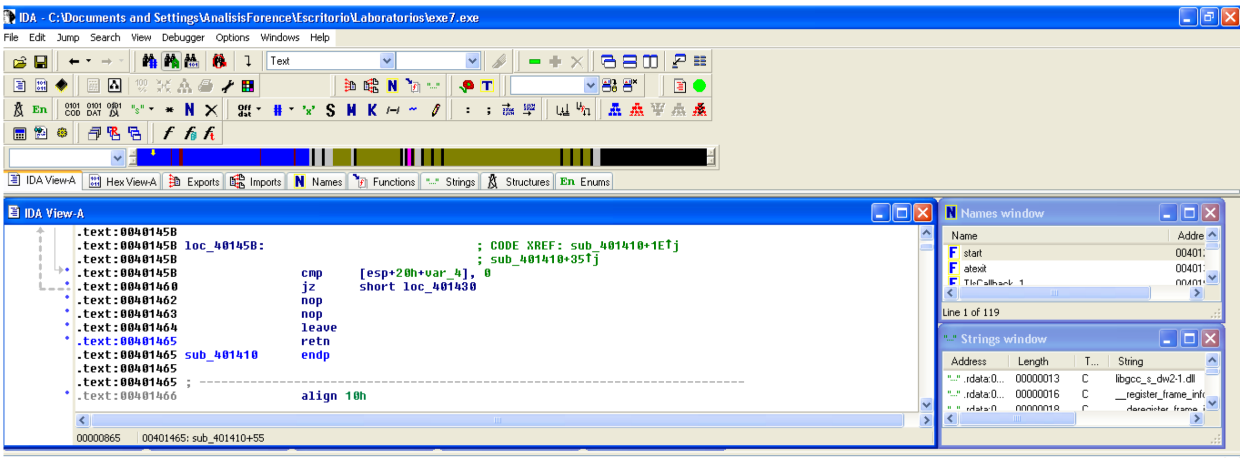
00000830 | 00401430: sub_401410:loc_401430

Name	Addre
start	00401:
atexit	00401:

Line 1 of 119

Address	Length	T...	String
.rdata:0...	00000013	C	libgcc_s_dw2-1.dll
.rdata:0...	00000016	C	__register_frame_inf

IDA - C:\Documents and Settings\AnalisisForence\Escritorio\Laboratorios\exe7.exe
File Edit Jump Search View Debugger Options Windows Help
IDA View-A
Hex View-A
Exports
Imports
Names
Functions
Strings
Structures
Enums
Names window
Strings window


IDA - C:\Documents and Settings\AnalisisForence\Escritorio\Laboratorios\exe7.exe
IDA View-A
Names window
Strings window

```
.text:0040145B
.text:0040145B loc_40145B:                             ; CODE XREF: sub_401410+1E↑j
.text:0040145B                                         ; sub_401410+35↑j
.text:0040145B                 cmp     [esp+20h+var_4], 0
.text:00401460                 jz      short loc_401430
.text:00401462                 nop
.text:00401463                 nop
.text:00401464                 leave
.text:00401465                 retn
.text:00401465 sub_401410      endp
.text:00401465
.text:00401465 ; ---------------------------------------------------------------------------
.text:00401466                 align 10h
```

00000865 00401465: sub_401410+55

- *This is very similar to the ”for” loop in point 6, unlike that, it lacks an increment section.*
- *Finally, This is very similar to the ”for” loop in point 6, but unlike that point, it lacks an increment section, so the only way to stop this code from running repeatedly is with a conditional jump that occurs at the start and an unconditional jump at the end. Also in this way you can recognize a ”while”.*

8 Bibliography:

- *Bejtlich, R. (s.f.). Practical Malware Analysis. No Starch press.*