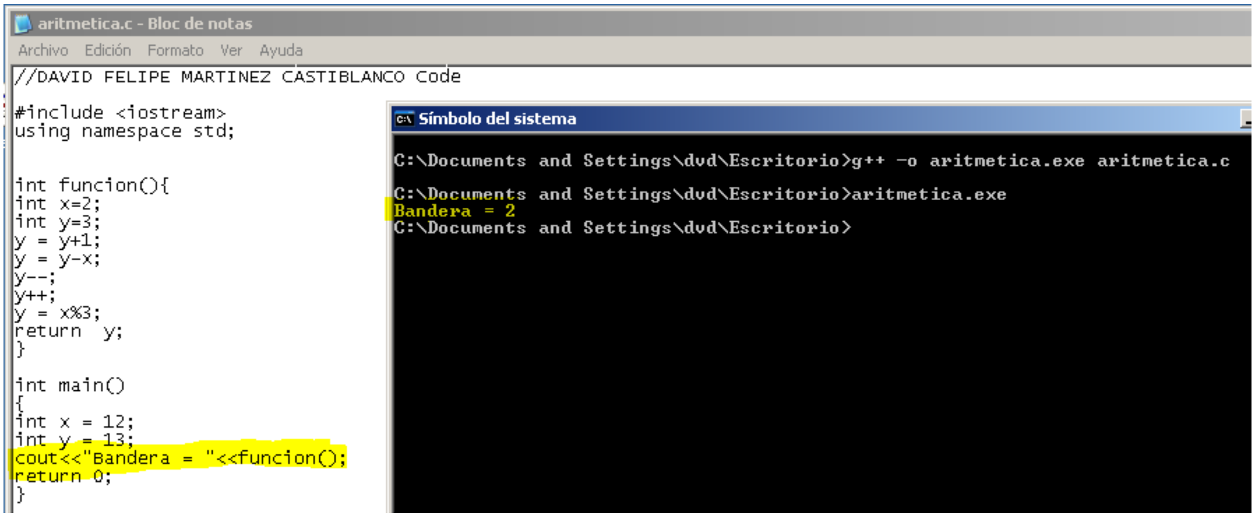
DAVID FELIPE MARTINEZ CASTIBLANCO

Code Constructs Laboratory

-
    - Code constructs can be seen as little code parts which do not refer to specific algorithms, for example, if statements have the same structure regardless the specific algorithm inside.
    - Some code constructs are if statements, while loops, switch statements, for statements, etc.
    - In past laboratory I told why assembly always change depending on the CPU architecture. For the same reason code constructs besides are different in assembly generation.
-
    - As I said past times, memory manage and usage is so important on assembly code generation, and its linked with the way of local and global variables manage.
    - Global variables are saved in memory locations and these can be accessed directly by knowing the direction, but local variables are saved on the stack, at a constant relative offset to ebp, it means that if the program want search this location, just need to stay on ebp (stack), and then move some constant negative positions.

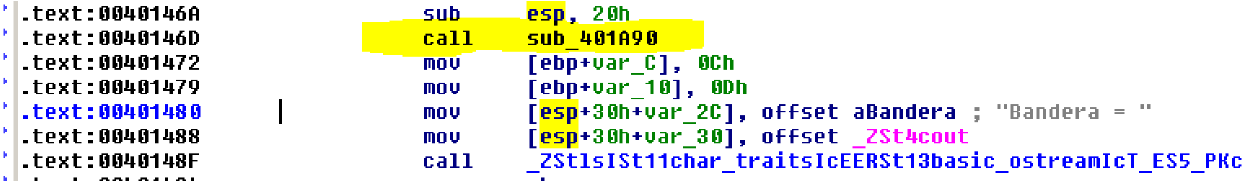
- **ARITHMETICAL OPERATIONS**
- First than all, create and compile some basic operations on C++.


aritmetica.c - Bloc de notas
Archivo Edición Formato Ver Ayuda
Símbolo del sistema

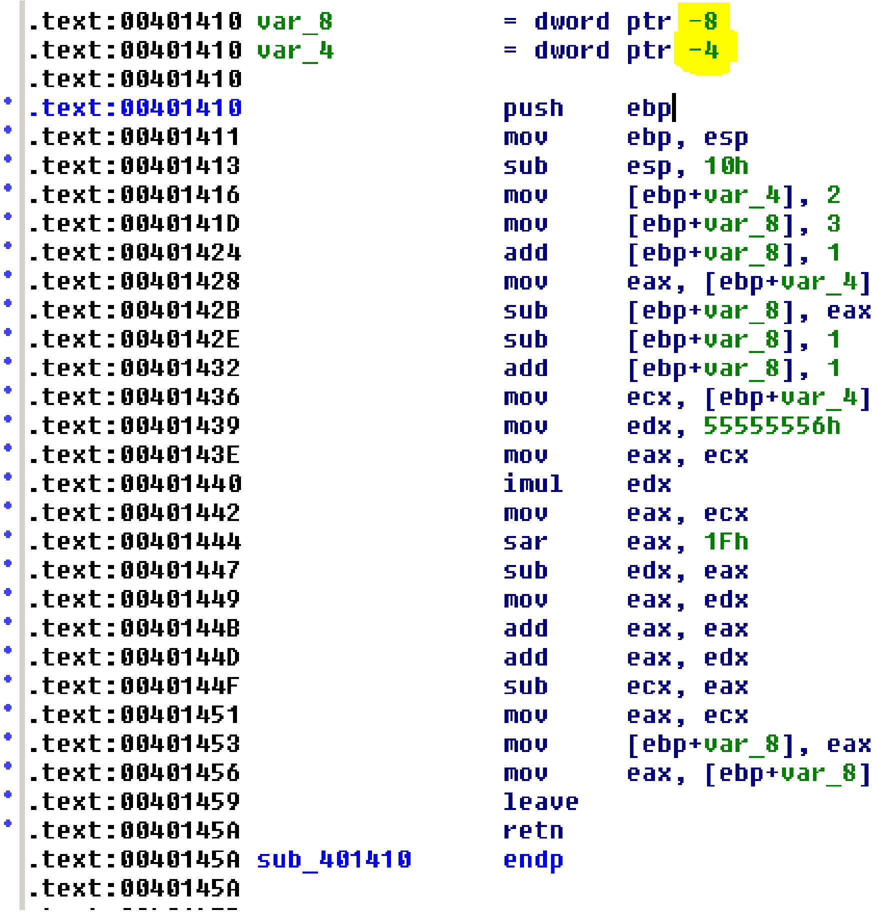
```
//DAVID FELIPE MARTINEZ CASTIBLANCO Code

#include <iostream>
using namespace std;


int funcion(){
int x=2;
int y=3;
y = y+1;
y = y-x;
y--;
y++;
y = x%3;
return  y;
}

int main()
{
int x = 12;
int y = 13;
cout<<"Bandera = "<<funcion();
return 0;
}
```

```
C:\Documents and Settings\dvd\Escritorio>g++ -o aritmetica.exe aritmetica.c

C:\Documents and Settings\dvd\Escritorio>aritmetica.exe
Bandera = 2
C:\Documents and Settings\dvd\Escritorio>
```

- Now, examine it on IDAPro. I put a key word on the code, so its easy to search by “Bandera =” and search the previous function call to examine all basic operations I wrote.



```
.text:0040146A                 sub     esp, 20h
.text:0040146D                 call    sub_401A90
.text:00401472                 mov     [ebp+var_C], 0Ch
.text:00401479                 mov     [ebp+var_10], 0Dh
.text:00401480                 mov     [esp+30h+var_2C], offset aBandera ; "Bandera = "
.text:00401488                 mov     [esp+30h+var_30], offset _ZSt4cout
.text:0040148F                 call    _ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc
```

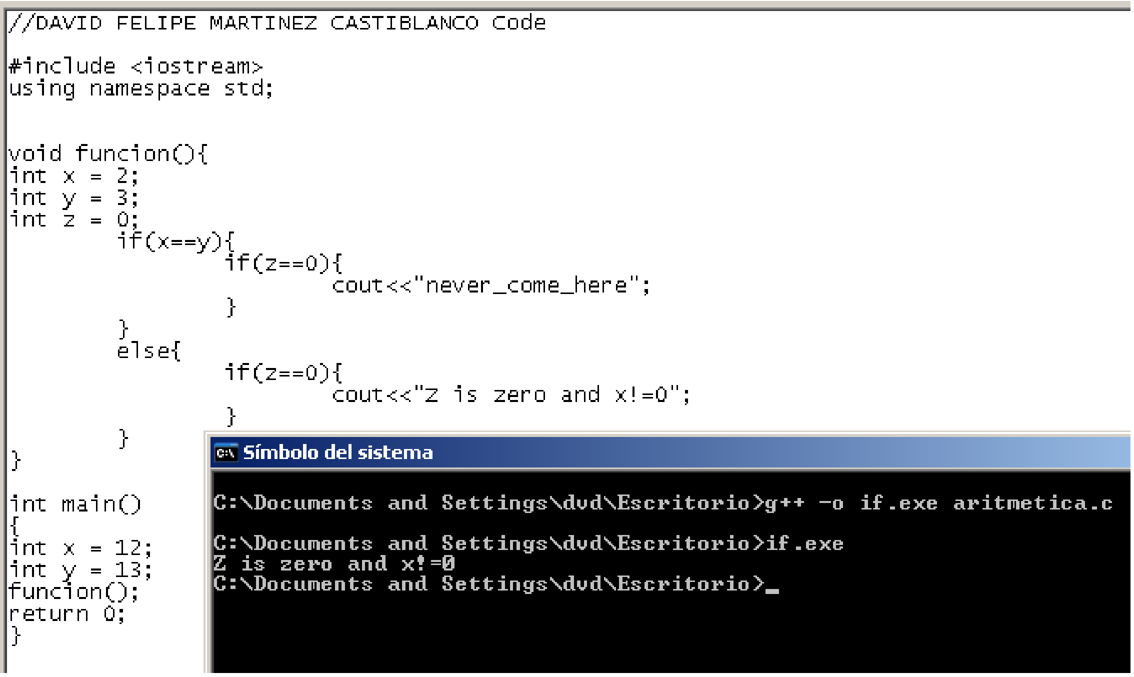


```
.text:00401410 var_8           = dword ptr -8
.text:00401410 var_4           = dword ptr -4
.text:00401410
.text:00401410                 push    ebp
.text:00401411                 mov     ebp, esp
.text:00401413                 sub     esp, 10h
.text:00401416                 mov     [ebp+var_4], 2
.text:0040141D                 mov     [ebp+var_8], 3
.text:00401424                 add     [ebp+var_8], 1
.text:00401428                 mov     eax, [ebp+var_4]
.text:0040142B                 sub     [ebp+var_8], eax
.text:0040142E                 sub     [ebp+var_8], 1
.text:00401432                 add     [ebp+var_8], 1
.text:00401436                 mov     ecx, [ebp+var_4]
.text:00401439                 mov     edx, 55555556h
.text:0040143E                 mov     eax, ecx
.text:00401440                 imul    edx
.text:00401442                 mov     eax, ecx
.text:00401444                 sar     eax, 1Fh
.text:00401447                 sub     edx, eax
.text:00401449                 mov     eax, edx
.text:0040144B                 add     eax, eax
.text:0040144D                 add     eax, edx
.text:0040144F                 sub     ecx, eax
.text:00401451                 mov     eax, ecx
.text:00401453                 mov     [ebp+var_8], eax
.text:00401456                 mov     eax, [ebp+var_8]
.text:00401459                 leave
.text:0040145A                 retn
.text:0040145A sub_401410      endp
.text:0040145A
```

- the following assembly sequence corresponds to the function steps
  - First, note that x and y are declared as local variables, and IDA recognize them by put them with negative pointers.
  - first, to make “y = y+1”, brings variable y (var_8), “add” is used to sum 1, and next, with “mov” the result is moved to eax.
  - To make “y = y-x”, like the previous part, brings to eax variable x(var_4) with “mov”, and next use “sub” to substract what is inside of eax(var_4) to y (var_8).
  - To make “y--”, uses “sub” directly, to substract 1 to variable y (var_8).
  - “y++” is the same as previous step but using “add”
  - now to make “y = x%3”, move the variable x to eax and ecx (copy), move some number (unknown) to edx, “imul” is used to make something like “eax=eax*edx”, then “sar” is used to make a signed division between eax and 1Fh, this result is subtracted to edx, then sum eax with itself and edx, finally, substract it to ecx (copy of variable x), with the module result, it is moved to variable y (var_8).
  - Finally, var_8 is moved to eax and returned.
  - By facility, sometimes I just referred to eax, edx, ecx as normal or variable numbers, but assembly really are accessing to the variables inside these directions. besides, when I said that some z data is moved to “var_xxx” what was really happening is that z was

being written on [ebp+var_xxx], taking in account that var_xxx is inside the stack as a local variable.

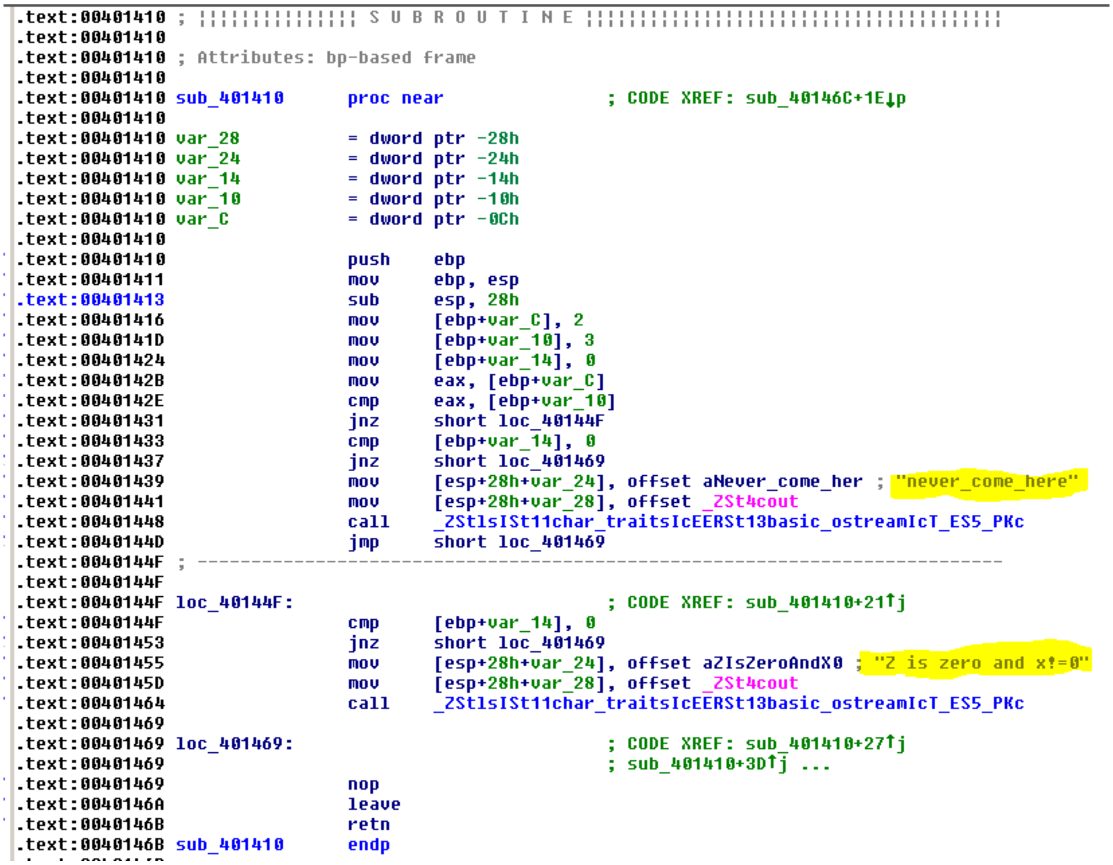
- **IF**
- Then, analyze some nested if statement. Note that the if statement structure is embedded on the nested if statements, so I will analyze them in the nested if statements.



```
//DAVID FELIPE MARTINEZ CASTIBLANCO Code

#include <iostream>
using namespace std;


void funcion(){
int x = 2;
int y = 3;
int z = 0;
        if(x==y){
                if(z==0){
                        cout<<"never_come_here";
                }
        }
        else{
                if(z==0){
                        cout<<"z is zero and x!=0";
                }

        }
}

int main()
{
int x = 12;
int y = 13;
funcion();
return 0;
}
```

```
C:\Documents and Settings\dvd\Escritorio>g++ -o if.exe aritmetica.c

C:\Documents and Settings\dvd\Escritorio>if.exe
Z is zero and x!=0
C:\Documents and Settings\dvd\Escritorio>_
```

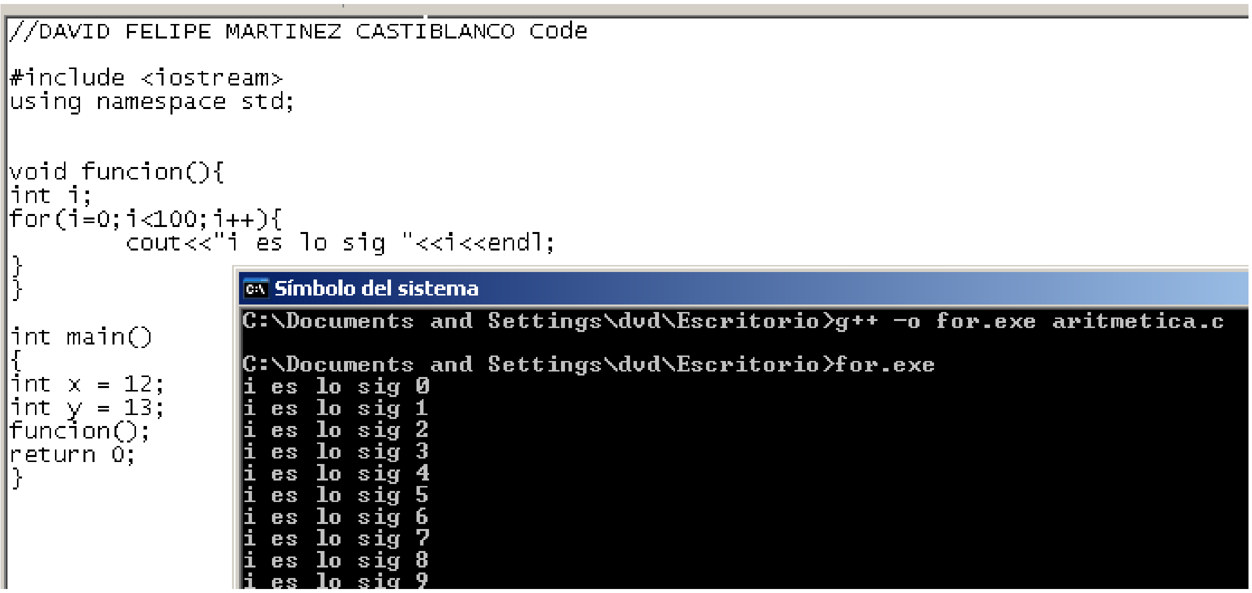
- This time, function code was found thanks to the key word “never_come_here” implied on the if statement.



```
.text:00401410 ; ||||||||||||||| S U B R O U T I N E |||||||||||||||||||||||||||||||||||||||
.text:00401410
.text:00401410 ; Attributes: bp-based frame
.text:00401410
.text:00401410 sub_401410      proc near               ; CODE XREF: sub_40146C+1E↓p
.text:00401410
.text:00401410 var_28          = dword ptr -28h
.text:00401410 var_24          = dword ptr -24h
.text:00401410 var_14          = dword ptr -14h
.text:00401410 var_10          = dword ptr -10h
.text:00401410 var_C           = dword ptr -0Ch
.text:00401410
.text:00401410                 push    ebp
.text:00401411                 mov     ebp, esp
.text:00401413                 sub     esp, 28h
.text:00401416                 mov     [ebp+var_C], 2
.text:0040141D                 mov     [ebp+var_10], 3
.text:00401424                 mov     [ebp+var_14], 0
.text:0040142B                 mov     eax, [ebp+var_C]
.text:0040142E                 cmp     eax, [ebp+var_10]
.text:00401431                 jnz     short loc_40144F
.text:00401433                 cmp     [ebp+var_14], 0
.text:00401437                 jnz     short loc_401469
.text:00401439                 mov     [esp+28h+var_24], offset aNever_come_her ; "never_come_here"
.text:00401441                 mov     [esp+28h+var_28], offset _ZSt4cout
.text:00401448                 call    _ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc
.text:0040144D                 jmp     short loc_401469
.text:0040144F ; ---------------------------------------------------------------------------
.text:0040144F
.text:0040144F loc_40144F:                             ; CODE XREF: sub_401410+21↑j
.text:0040144F                 cmp     [ebp+var_14], 0
.text:00401453                 jnz     short loc_401469
.text:00401455                 mov     [esp+28h+var_24], offset aZIsZeroAndX0 ; "Z is zero and x!=0"
.text:0040145D                 mov     [esp+28h+var_28], offset _ZSt4cout
.text:00401464                 call    _ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc
.text:00401469
.text:00401469 loc_401469:                             ; CODE XREF: sub_401410+27↑j
.text:00401469                                         ; sub_401410+3D↑j ...
.text:00401469                 nop
.text:0040146A                 leave
.text:0040146B                 retn
.text:0040146B sub_401410      endp
```

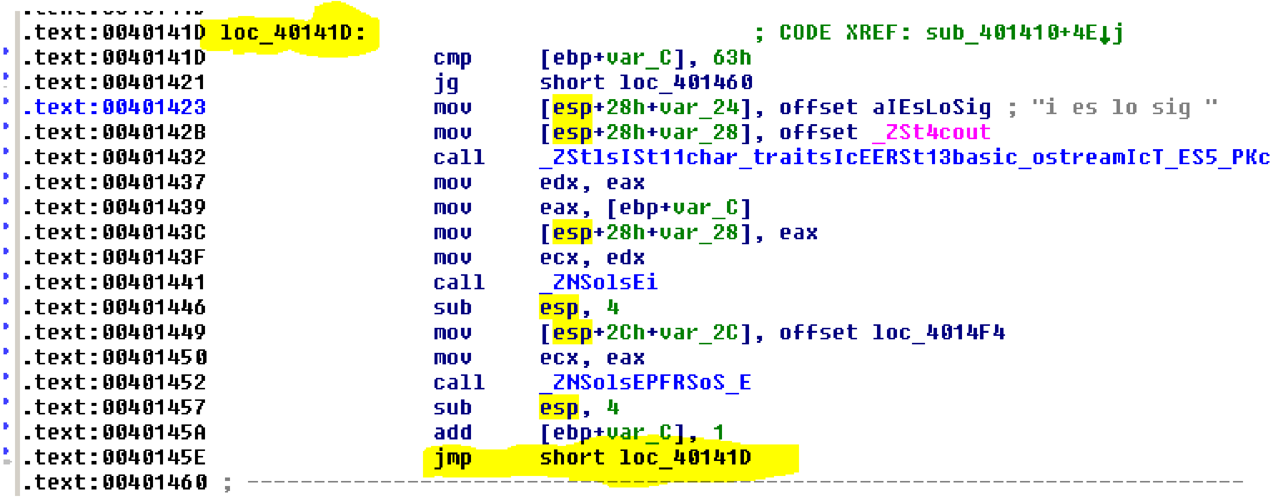
- Same as the past example, local variables are recognized and saved
- Var_c = x = 2, var_10 = y = 3, var_14 = z = 0.
- Note that var_c is moved to eax, and next "cmp" is used to camper it with var_10, in another words x is compared with y, the result (if it is not zero) is used to make a jump to another part of the code "jnz"".
- Note that x is not equal to y, so, effectively the code makes a jump to "loc_40144F", in this part of code there is another "cmp", which is referred to the nested if statement where z is compared with 0. This time result is 0, so "jnz" do not nothing and the next instruction is the string printing.

- **FOR**

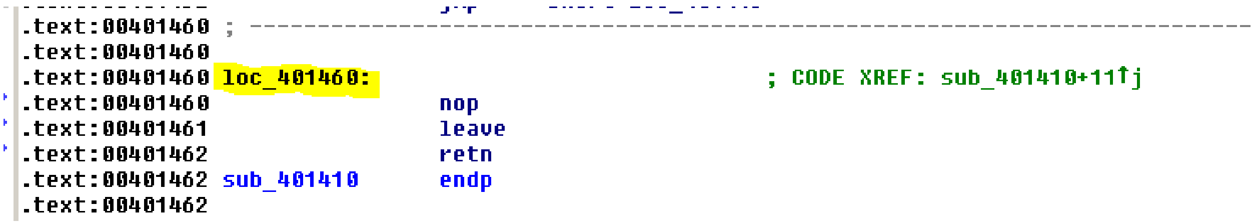


```
//DAVID FELIPE MARTINEZ CASTIBLANCO Code

#include <iostream>
using namespace std;


void funcion(){
int i;
for(i=0;i<100;i++){
        cout<<"i es lo sig "<<i<<endl;
}
}

int main()
{
int x = 12;
int y = 13;
funcion();
return 0;
}
```

```
C:\Documents and Settings\dvd\Escritorio>g++ -o for.exe aritmetica.c

C:\Documents and Settings\dvd\Escritorio>for.exe
i es lo sig 0
i es lo sig 1
i es lo sig 2
i es lo sig 3
i es lo sig 4
i es lo sig 5
i es lo sig 6
i es lo sig 7
i es lo sig 8
i es lo sig 9
```



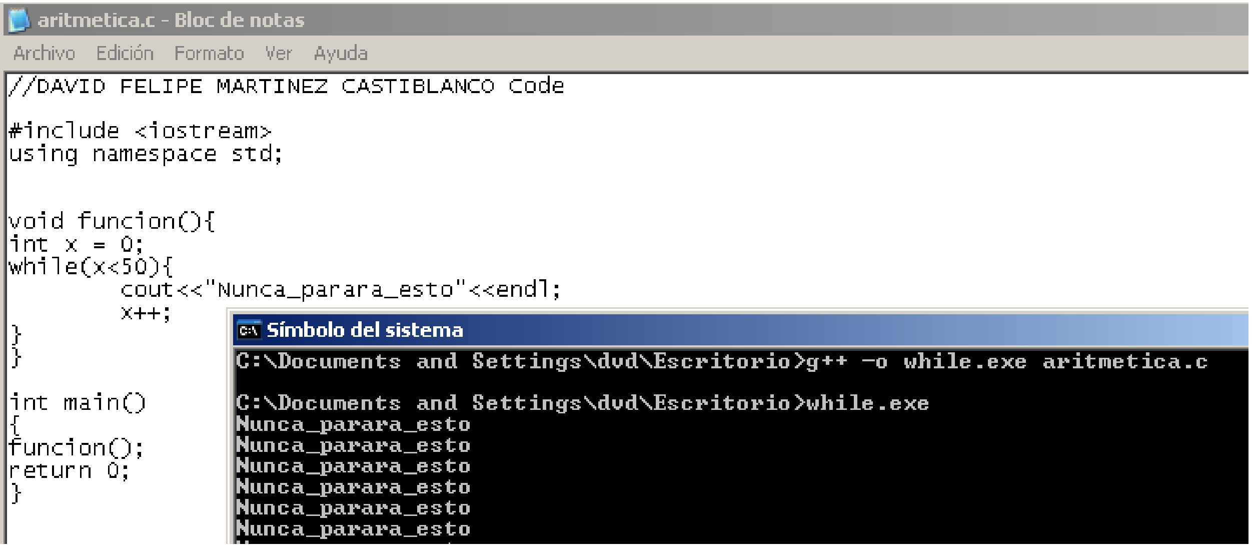
```
.text:0040141D loc_40141D:                              ; CODE XREF: sub_401410+4E↓j
.text:0040141D                 cmp     [ebp+var_C], 63h
.text:00401421                 jg      short loc_401460
.text:00401423                 mov     [esp+28h+var_24], offset aIEsLoSig ; "i es lo sig "
.text:0040142B                 mov     [esp+28h+var_28], offset _ZSt4cout
.text:00401432                 call    _ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc
.text:00401437                 mov     edx, eax
.text:00401439                 mov     eax, [ebp+var_C]
.text:0040143C                 mov     [esp+28h+var_28], eax
.text:0040143F                 mov     ecx, edx
.text:00401441                 call    _ZNSolsEi
.text:00401446                 sub     esp, 4
.text:00401449                 mov     [esp+2Ch+var_2C], offset loc_4014F4
.text:00401450                 mov     ecx, eax
.text:00401452                 call    _ZNSolsEPFRSoS_E
.text:00401457                 sub     esp, 4
.text:0040145A                 add     [ebp+var_C], 1
.text:0040145E                 jmp     short loc_40141D
.text:00401460 ; ---------------------------------------------------------------------------
```



```
.text:00401460 ; ---------------------------------------------------------------------------
.text:00401460
.text:00401460 loc_401460:                              ; CODE XREF: sub_401410+11↑j
.text:00401460                 nop
.text:00401461                 leave
.text:00401462                 retn
.text:00401462 sub_401410      endp
.text:00401462
```

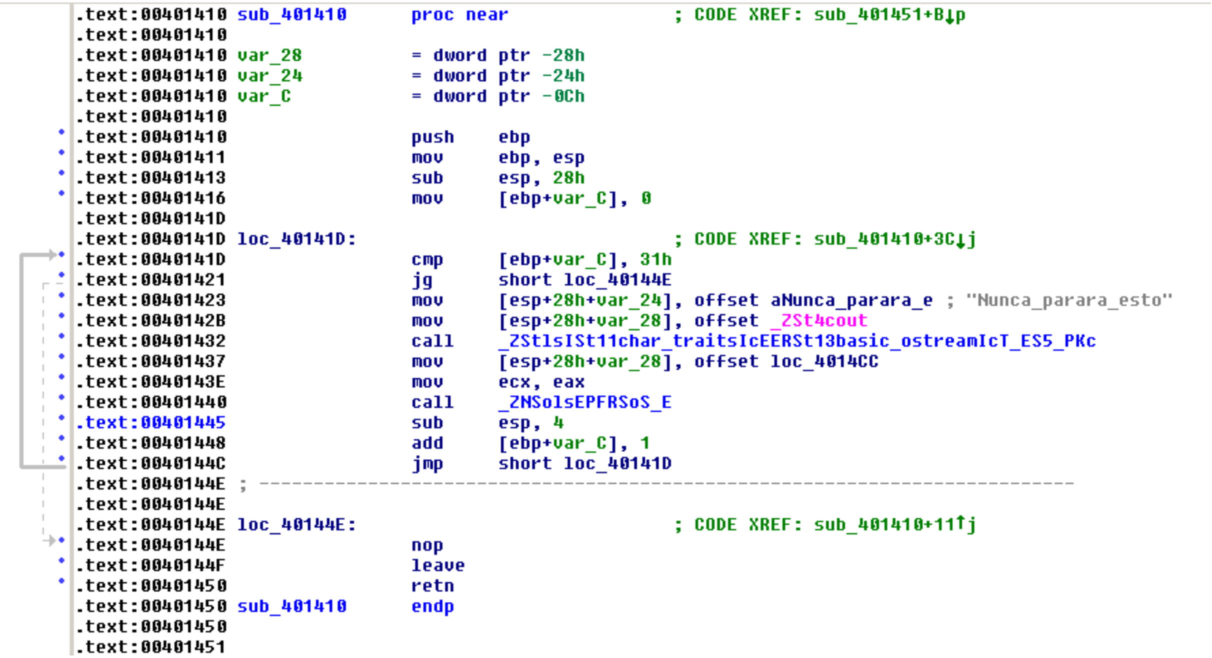
- First than all, note that the block of instructions is the same for every loop inside the for statement, this block is in “loc_40141D”, and at the end of the block there is a jump to the previous location (same block).
- The way to get out of the loop is to make a little jump to “loc_401460” at the start of the code, where for statement ends.
- This previous location is reached if a conditional jump “jg” is performed. This “jg” is just after a “cmp”, which in this case variable I is being compared with 100 (i>=100 or var_c>=100).
- If “jg” is not performed (var_c <100), the following statements run, so there is a print function.
- I am printing variable I inside the for, so, equal as variables are being printed in previous examples, this time variable I is moved and passed to print function.
- Generally, in every for statement, the final part consists on perform the third statement of the for, in this case “i++”, and jump to the same block as I said previously.

- **WHILE**



```
//DAVID FELIPE MARTINEZ CASTIBLANCO Code

#include <iostream>
using namespace std;


void funcion(){
int x = 0;
while(x<50){
        cout<<"Nunca_parara_esto"<<endl;
        x++;
}
}

int main()
{
funcion();
return 0;
}
```

```
C:\Documents and Settings\dvd\Escritorio>g++ -o while.exe aritmetica.c

C:\Documents and Settings\dvd\Escritorio>while.exe
Nunca_parara_esto
Nunca_parara_esto
Nunca_parara_esto
Nunca_parara_esto
Nunca_parara_esto
Nunca_parara_esto
```



```
.text:00401410 sub_401410      proc near               ; CODE XREF: sub_401451+B↓p
.text:00401410
.text:00401410 var_28          = dword ptr -28h
.text:00401410 var_24          = dword ptr -24h
.text:00401410 var_C           = dword ptr -0Ch
.text:00401410
.text:00401410                 push    ebp
.text:00401411                 mov     ebp, esp
.text:00401413                 sub     esp, 28h
.text:00401416                 mov     [ebp+var_C], 0
.text:0040141D
.text:0040141D loc_40141D:                             ; CODE XREF: sub_401410+3C↓j
.text:0040141D                 cmp     [ebp+var_C], 31h
.text:00401421                 jg      short loc_40144E
.text:00401423                 mov     [esp+28h+var_24], offset aNunca_parara_e ; "Nunca_parara_esto"
.text:0040142B                 mov     [esp+28h+var_28], offset _ZSt4cout
.text:00401432                 call    _ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc
.text:00401437                 mov     [esp+28h+var_28], offset loc_4014CC
.text:0040143E                 mov     ecx, eax
.text:00401440                 call    _ZNSolsEPFRSoS_E
.text:00401445                 sub     esp, 4
.text:00401448                 add     [ebp+var_C], 1
.text:0040144C                 jmp     short loc_40141D
.text:0040144E ; ---------------------------------------------------------------------------
.text:0040144E
.text:0040144E loc_40144E:                             ; CODE XREF: sub_401410+11↑j
.text:0040144E                 nop
.text:0040144F                 leave
.text:00401450                 retn
.text:00401450 sub_401410      endp
.text:00401450
.text:00401451
```

- Note that the while structure is very similar to for statements, at the end of the block there is a jump to the same block, and at the start of the block, there is a “cmp” to decide if perform a conditional jump.
- Again, if “jg” is not performed, the next instructions are part of the code I wrote.
- Structure is essentially the same in both loops because you can write every for as a while statement, and vice versa.
- In this structure I don talk about the last add of variable x (var_c) because it is not part of a general loop, I put it to simulate a condition that make the loop stops, but note that missing this add, there will no never condition to stop the while, and the code would be infinitely printing and jumping to “loc_40141D” (same block).
- The main difference is that, while loops not contains a third statement to always perform at the end of the block, there is just a jump. You have to put inside the block something to make this loop stop.