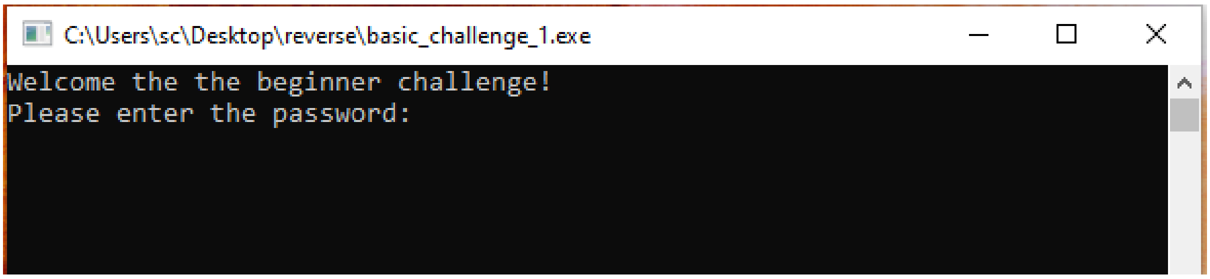
Basic Challenge 1

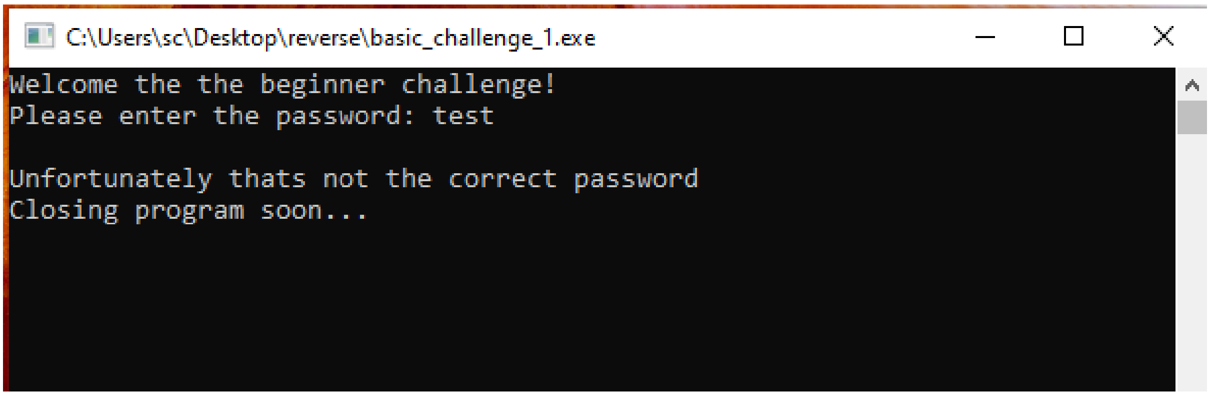
S Chowdhury

Here we will solve the first challenge in this series. We have to figure out the correct password for this program. First, lets run the program.



```
Welcome the the beginner challenge!
Please enter the password:
```

The program asks for a password. Lets give “test” as the password.



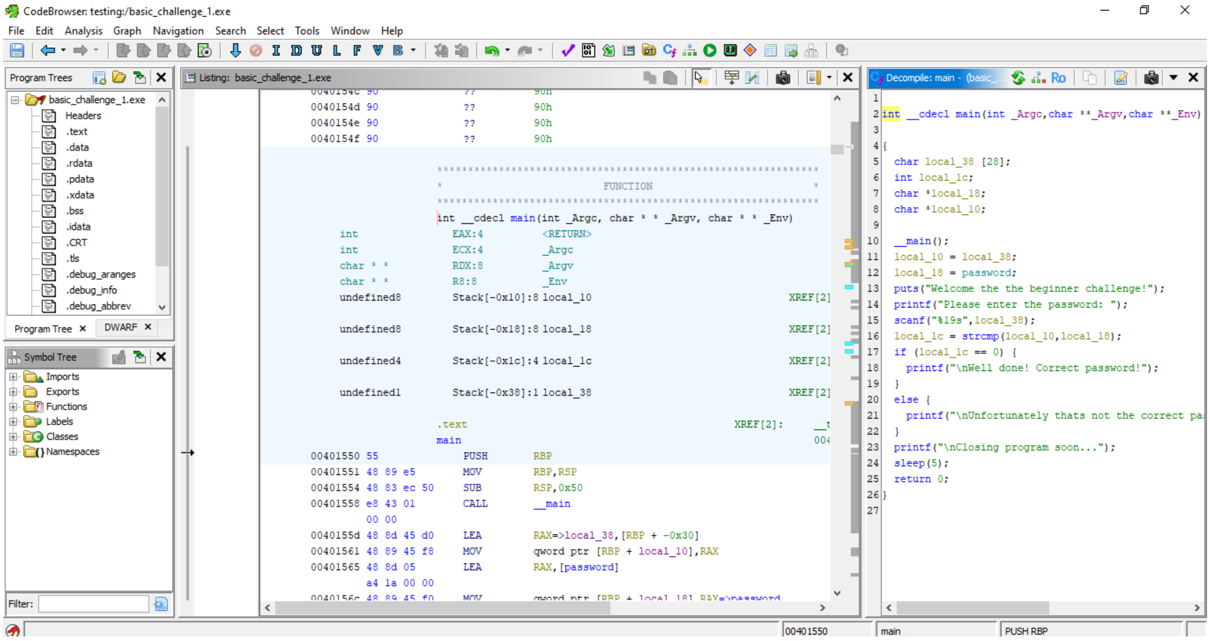
```
Welcome the the beginner challenge!
Please enter the password: test

Unfortunately thats not the correct password
Closing program soon...
```

That’s not the correct password.

The first basic trick we can use is to view the strings of the program. On Linux, you can use this command:

strings [filename]

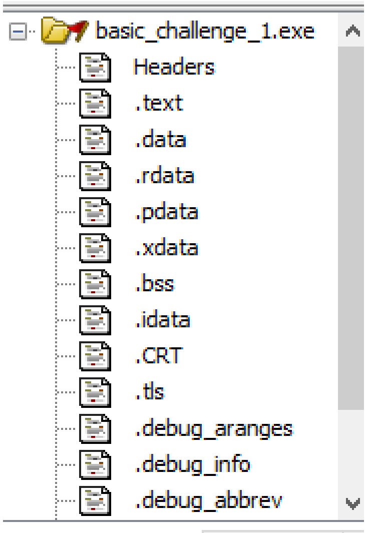
On Windows, we can open Ghidra and see the strings from there. Here’s the program opened in Ghidra:



```
0040154c 90              ??         90h
0040154d 90              ??         90h
0040154e 90              ??         90h
0040154f 90              ??         90h

                 **************************************************************
                 *                          FUNCTION                          *
                 **************************************************************
                 int __cdecl main(int _Argc, char * * _Argv, char * * _Env)
     int            EAX:4          <RETURN>
     int            ECX:4          _Argc
     char * *       RDX:8          _Argv
     char * *       R8:8           _Env
     undefined8     Stack[-0x10]:8 local_10                           XREF[2]

     undefined8     Stack[-0x18]:8 local_18                           XREF[2]

     undefined4     Stack[-0x1c]:4 local_1c                           XREF[2]

     undefined1     Stack[-0x38]:1 local_38                           XREF[2]

                 .text                                     XREF[2]:    __t
                 main                                                  004
00401550 55              PUSH       RBP
00401551 48 89 e5        MOV        RBP,RSP
00401554 48 83 ec 50     SUB        RSP,0x50
00401558 e8 43 01        CALL       __main
         00 00
0040155d 48 8d 45 d0     LEA        RAX=>local_38,[RBP + -0x30]
00401561 48 89 45 f8     MOV        qword ptr [RBP + local_10],RAX
00401565 48 8d 05        LEA        RAX,[password]
         a4 1a 00 00
0040156c 48 89 45 f0     MOV        qword ptr [RBP + local_18],RAX=>password
```

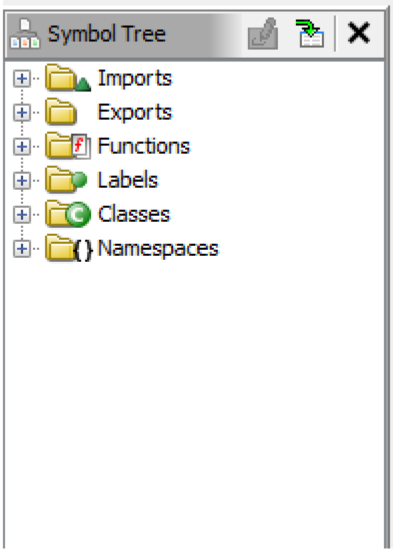
```
int __cdecl main(int _Argc,char **_Argv,char **_Env)

{
  char local_38 [28];
  int local_1c;
  char *local_18;
  char *local_10;

  __main();
  local_10 = local_38;
  local_18 = password;
  puts("Welcome the the beginner challenge!");
  printf("Please enter the password: ");
  scanf("%19s",local_38);
  local_1c = strcmp(local_10,local_18);
  if (local_1c == 0) {
    printf("\nWell done! Correct password!");
  }
  else {
    printf("\nUnfortunately thats not the correct pa
  }
  printf("\nClosing program soon...");
  sleep(5);
  return 0;
}
```

On the top left, we can see the **Program Trees** section. Here, we can see the different sections of the executable.

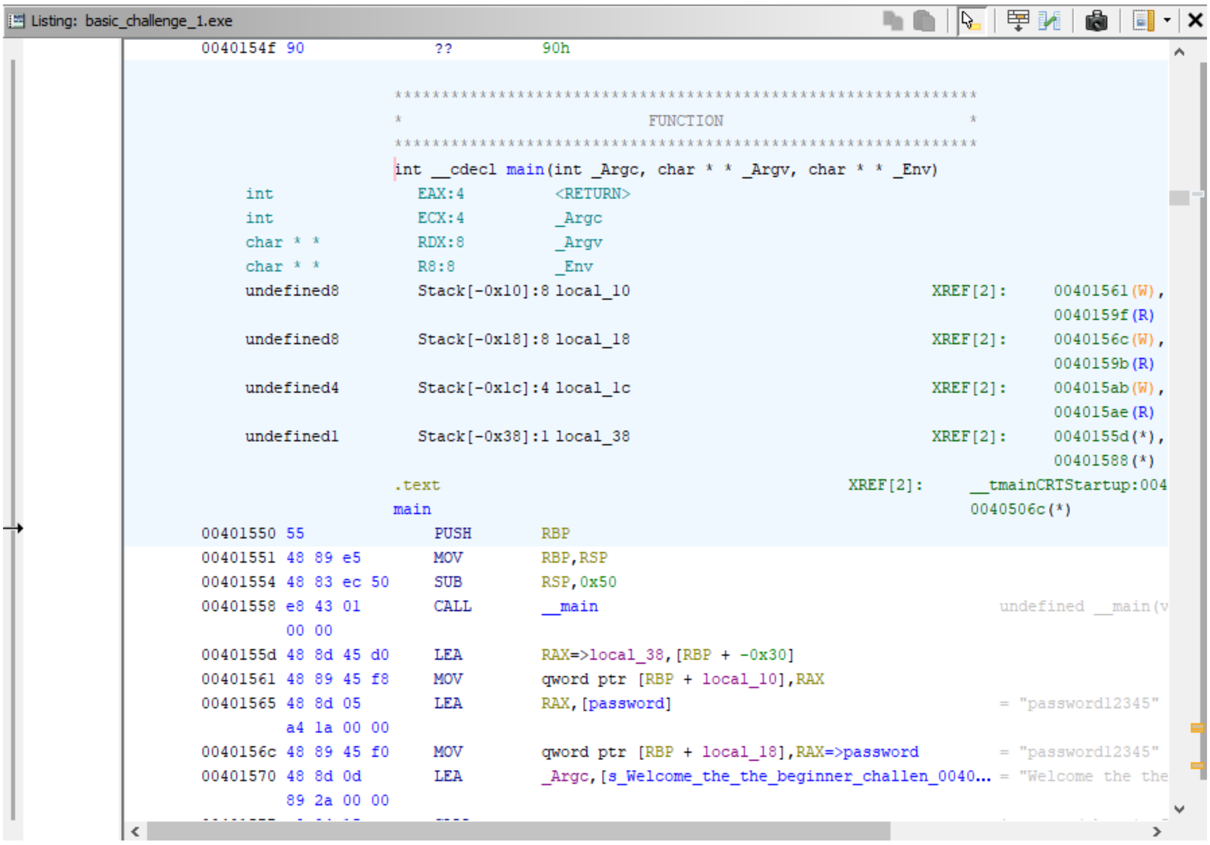


For now, you should know that the *.text* contains the code, the *.data* contains static global variables, the *.bss* section contains global variables that can be changed during runtime and the *Headers* contains important information for the Windows operating system to run the file.

On the bottom left, we can see the **symbol tree**:

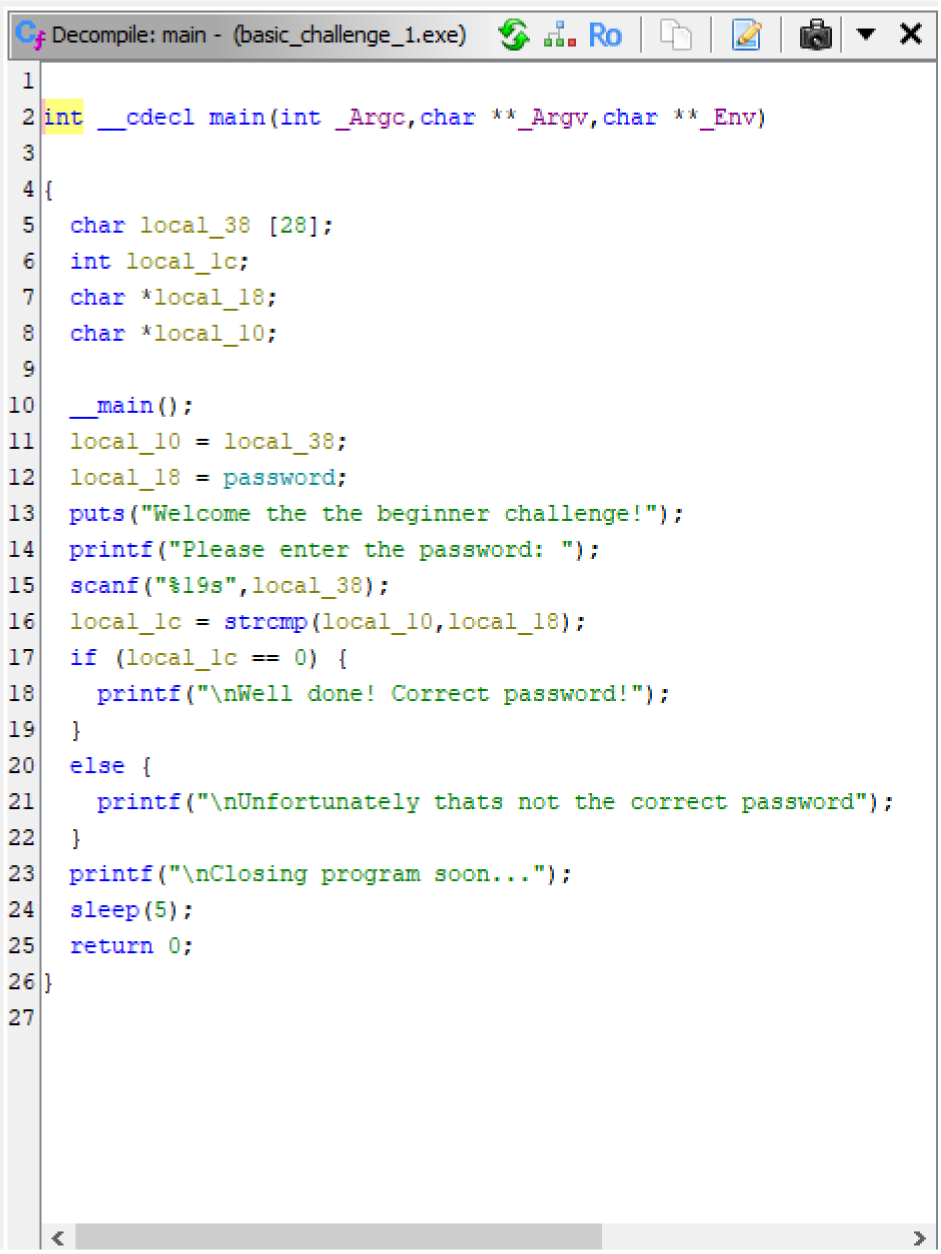


In the middle, we can see the **Listing Window**:



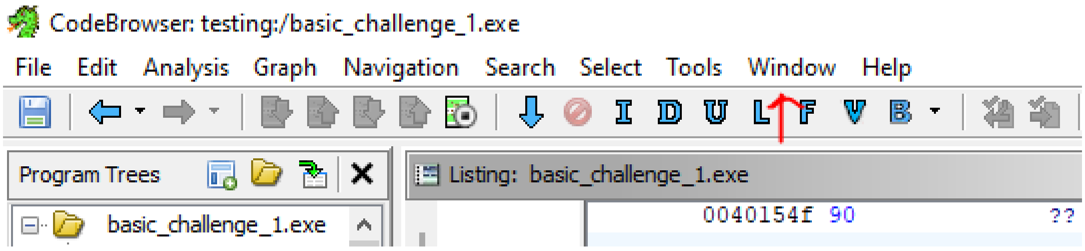
```
0040154f 90          ??         90h

                     ****************************************************************
                     *                          FUNCTION                            *
                     ****************************************************************
                     int __cdecl main(int _Argc, char * * _Argv, char * * _Env)
     int               EAX:4          <RETURN>
     int               ECX:4          _Argc
     char * *          RDX:8          _Argv
     char * *          R8:8           _Env
     undefined8        Stack[-0x10]:8 local_10                     XREF[2]:     00401561(W),
                                                                                0040159f(R)
     undefined8        Stack[-0x18]:8 local_18                     XREF[2]:     0040156c(W),
                                                                                0040159b(R)
     undefined4        Stack[-0x1c]:4 local_1c                     XREF[2]:     004015ab(W),
                                                                                004015ae(R)
     undefined1        Stack[-0x38]:1 local_38                     XREF[2]:     0040155d(*),
                                                                                00401588(*)
                     .text                                 XREF[2]:     __tmainCRTStartup:004
                     main                                               0040506c(*)
00401550 55          PUSH       RBP
00401551 48 89 e5    MOV        RBP,RSP
00401554 48 83 ec 50 SUB        RSP,0x50
00401558 e8 43 01    CALL       __main                                          undefined __main(v
         00 00
0040155d 48 8d 45 d0 LEA        RAX=>local_38,[RBP + -0x30]
00401561 48 89 45 f8 MOV        qword ptr [RBP + local_10],RAX
00401565 48 8d 05    LEA        RAX,[password]                                  = "password12345"
         a4 1a 00 00
0040156c 48 89 45 f0 MOV        qword ptr [RBP + local_18],RAX=>password        = "password12345"
00401570 48 8d 0d    LEA        _Argc,[s_Welcome_the_the_beginner_challen_0040... = "Welcome the the
         89 2a 00 00
```

On the right, we can see the decompiled C code:



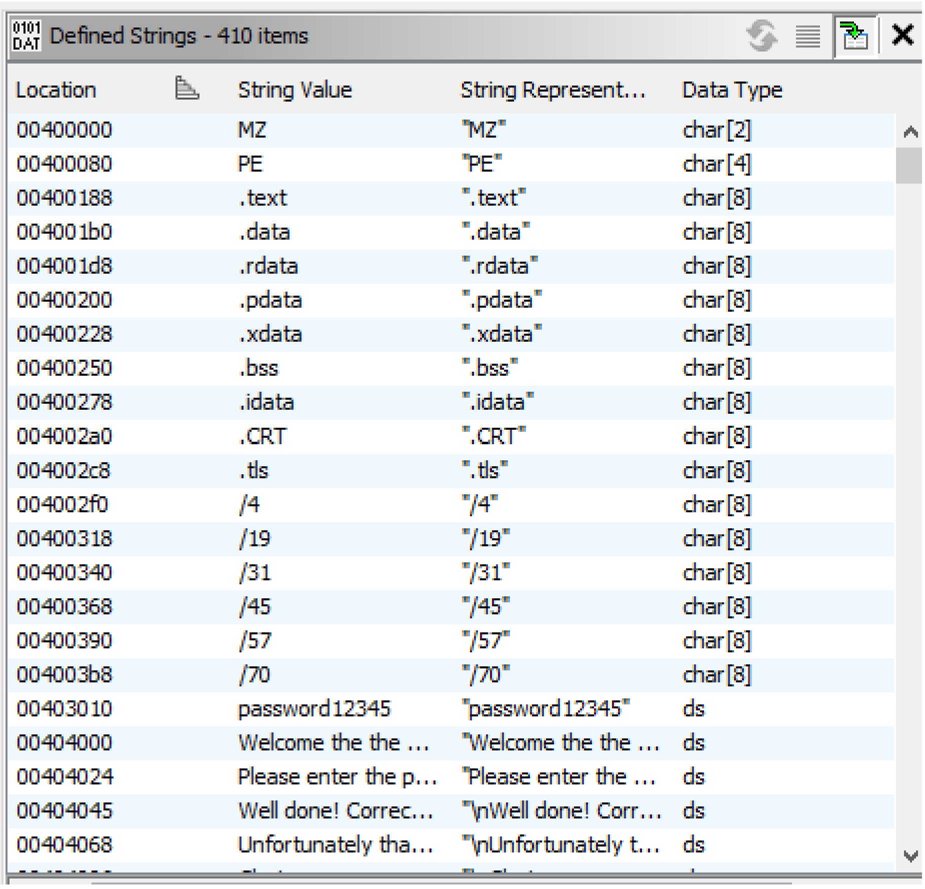
```

int __cdecl main(int _Argc,char **_Argv,char **_Env)

{
  char local_38 [28];
  int local_1c;
  char *local_18;
  char *local_10;

  __main();
  local_10 = local_38;
  local_18 = password;
  puts("Welcome the the beginner challenge!");
  printf("Please enter the password: ");
  scanf("%19s",local_38);
  local_1c = strcmp(local_10,local_18);
  if (local_1c == 0) {
    printf("\nWell done! Correct password!");
  }
  else {
    printf("\nUnfortunately thats not the correct password");
  }
  printf("\nClosing program soon...");
  sleep(5);
  return 0;
}

```

Now, lets see the strings inside this executable from Ghidra. Click *Window>Defined Strings* to see this information.



After clicking on *Window* click on the *Defined Strings* option.

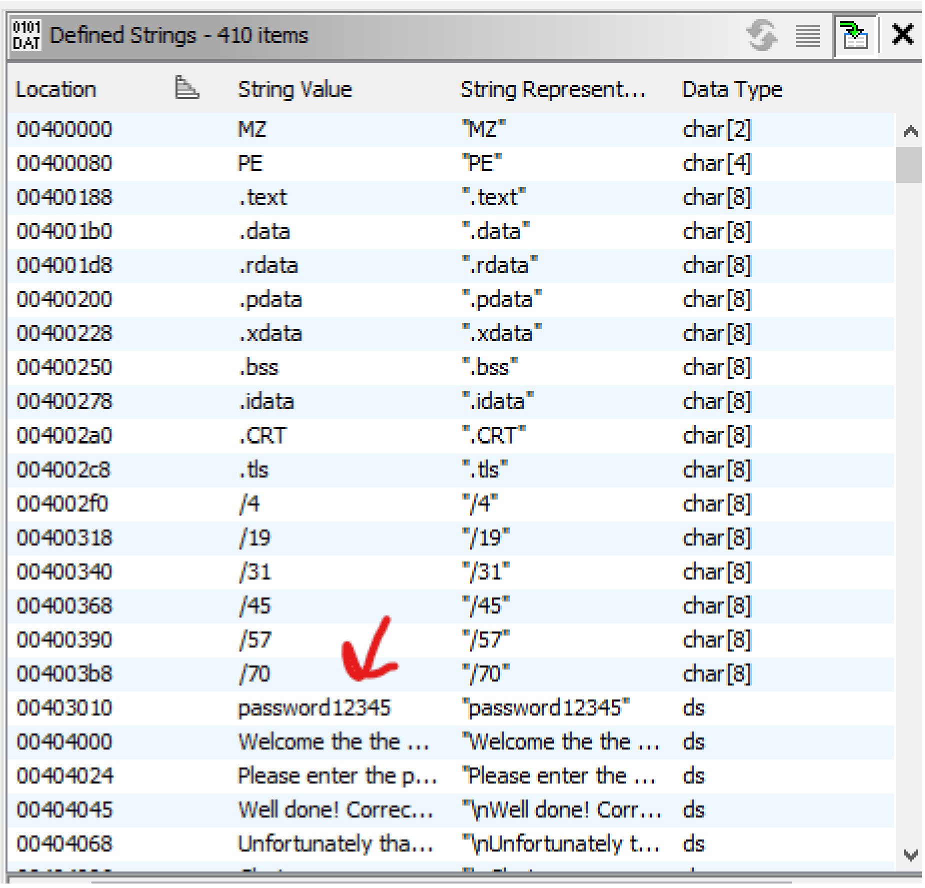
In that menu, we can see this information:



Defined Strings - 410 items

Location	String Value	String Represent...	Data Type
00400000	MZ	"MZ"	char[2]
00400080	PE	"PE"	char[4]
00400188	.text	".text"	char[8]
004001b0	.data	".data"	char[8]
004001d8	.rdata	".rdata"	char[8]
00400200	.pdata	".pdata"	char[8]
00400228	.xdata	".xdata"	char[8]
00400250	.bss	".bss"	char[8]
00400278	.idata	".idata"	char[8]
004002a0	.CRT	".CRT"	char[8]
004002c8	.tls	".tls"	char[8]
004002f0	/4	"/4"	char[8]
00400318	/19	"/19"	char[8]
00400340	/31	"/31"	char[8]
00400368	/45	"/45"	char[8]
00400390	/57	"/57"	char[8]
004003b8	/70	"/70"	char[8]
00403010	password12345	"password12345"	ds
00404000	Welcome the the ...	"Welcome the the ...	ds
00404024	Please enter the p...	"Please enter the ...	ds
00404045	Well done! Correc...	"\nWell done! Corr...	ds
00404068	Unfortunately tha...	"\nUnfortunately t...	ds

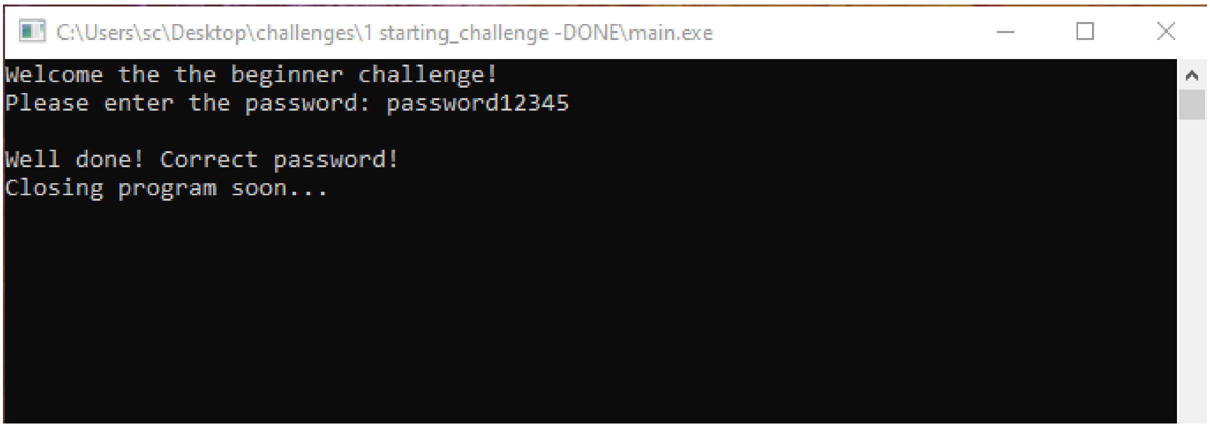
Note, there seems to be a string that could be the password:



Defined Strings - 410 items

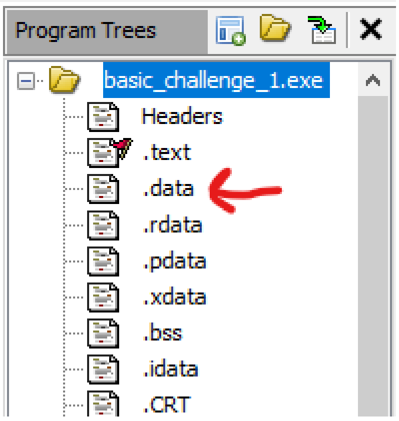
Location	String Value	String Represent...	Data Type
00400000	MZ	"MZ"	char[2]
00400080	PE	"PE"	char[4]
00400188	.text	".text"	char[8]
004001b0	.data	".data"	char[8]
004001d8	.rdata	".rdata"	char[8]
00400200	.pdata	".pdata"	char[8]
00400228	.xdata	".xdata"	char[8]
00400250	.bss	".bss"	char[8]
00400278	.idata	".idata"	char[8]
004002a0	.CRT	".CRT"	char[8]
004002c8	.tls	".tls"	char[8]
004002f0	/4	"/4"	char[8]
00400318	/19	"/19"	char[8]
00400340	/31	"/31"	char[8]
00400368	/45	"/45"	char[8]
00400390	/57	"/57"	char[8]
004003b8	/70	"/70"	char[8]
00403010	password12345	"password12345"	ds
00404000	Welcome the the ...	"Welcome the the ...	ds
00404024	Please enter the p...	"Please enter the ...	ds
00404045	Well done! Correc...	"\nWell done! Corr...	ds
00404068	Unfortunately tha...	"\nUnfortunately t...	ds

Let’s try to put “password12345” into the program and see if it’s correct:

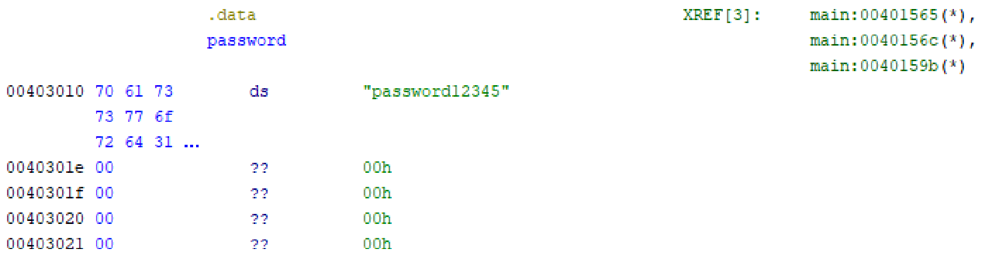

C:\Users\sc\Desktop\challenges\1 starting_challenge -DONE\main.exe

```
Welcome the the beginner challenge!
Please enter the password: password12345

Well done! Correct password!
Closing program soon...
```

So, that’s the correct password. Lets investigate how this program works. Let’s visit the *.data* section.



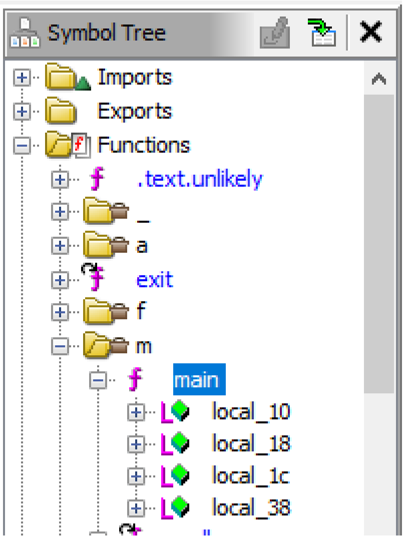
In the *.data* section, we can see a global variable named “password” and we can see the value:



```
                          .data                                    XREF[3]:     main:00401565(*),
                          password                                              main:0040156c(*),
                                                                                main:0040159b(*)
        00403010 70 61 73        ds         "password12345"
                 73 77 6f
                 72 64 31 ...
        0040301e 00              ??         00h
        0040301f 00              ??         00h
        00403020 00              ??         00h
        00403021 00              ??         00h
```

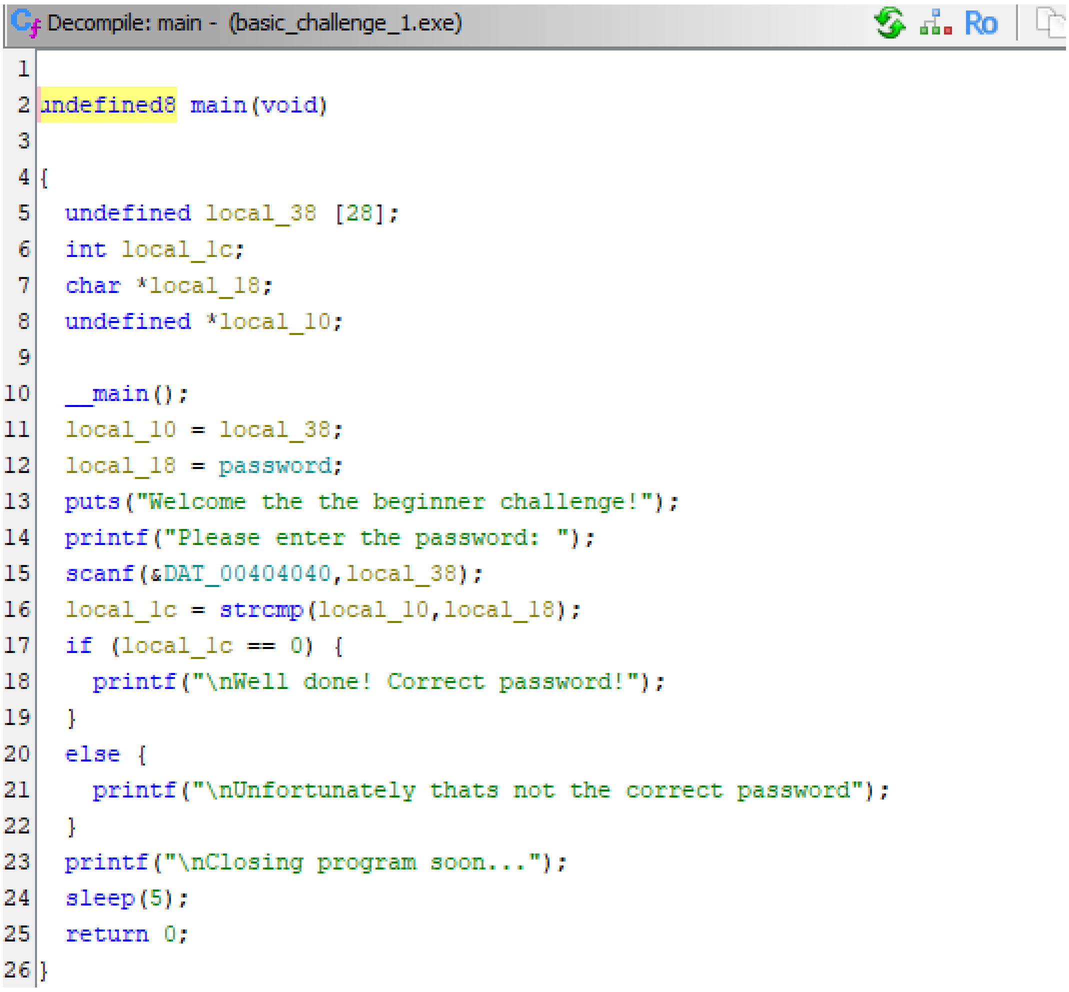
Please note that the “00403010” is a memory location where the string is located. The “ds” means “defined string” and the values “70 61 73 73…” represents the hexadecimal representation of the string.

Return to the main function:



Click main from the *Functions* folder in the **Symbol Tree**.

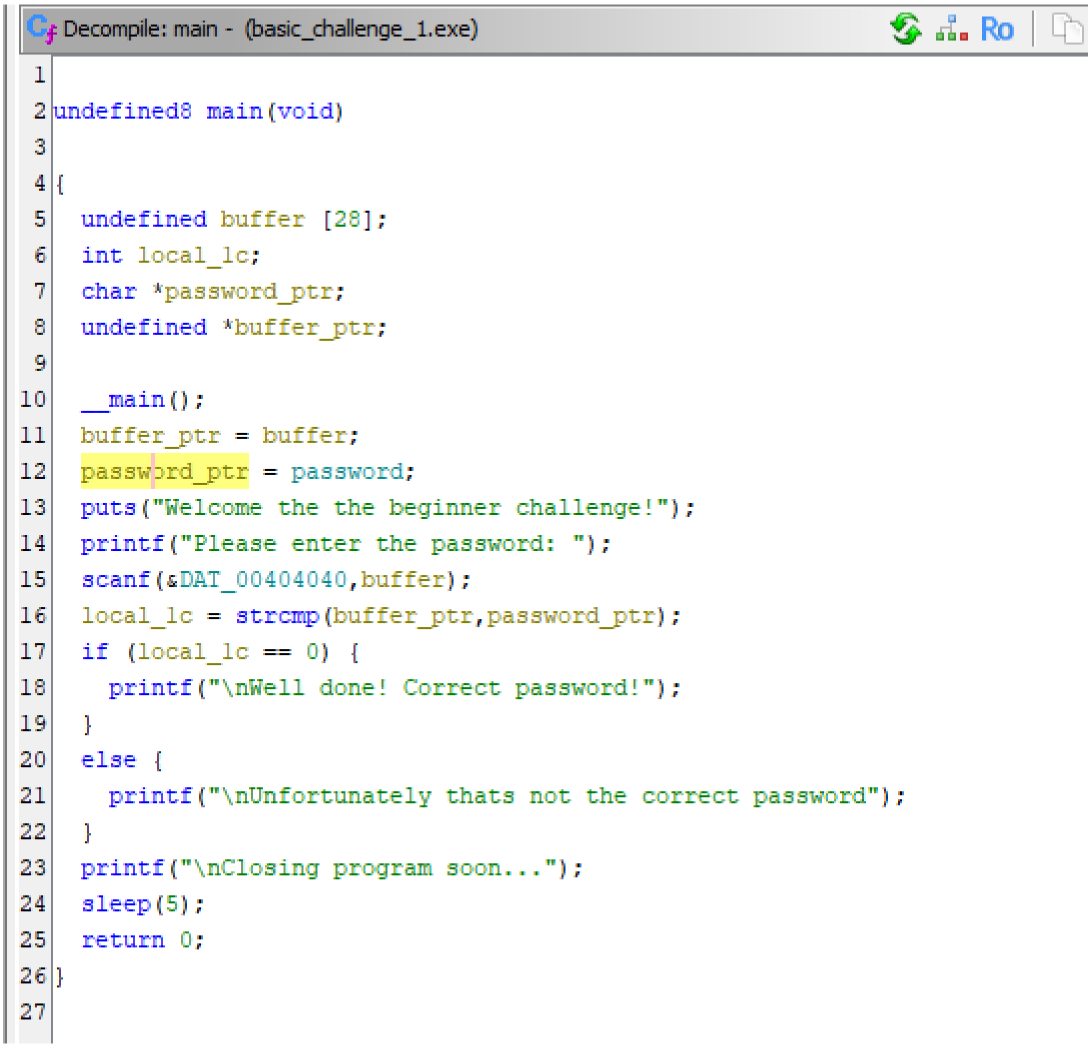
To bring the decompiler, exit the “Defined Strings” window and the decompiler should return. Otherwise, click *Window* > *Decompiler*.



```
 1
 2 undefined8 main(void)
 3
 4 {
 5   undefined local_38 [28];
 6   int local_1c;
 7   char *local_18;
 8   undefined *local_10;
 9
10   __main();
11   local_10 = local_38;
12   local_18 = password;
13   puts("Welcome the the beginner challenge!");
14   printf("Please enter the password: ");
15   scanf(&DAT_00404040,local_38);
16   local_1c = strcmp(local_10,local_18);
17   if (local_1c == 0) {
18     printf("\nWell done! Correct password!");
19   }
20   else {
21     printf("\nUnfortunately thats not the correct password");
22   }
23   printf("\nClosing program soon...");
24   sleep(5);
25   return 0;
26 }
```

We can see on line 5, local_38’s an array, from line 7 and 12 that local_18’s a pointer that stores the memory location of the *password* variable and from line 8 and 11 that local_10’s a pointer that stores the memory location of *local_38* array. We can rename variables in Ghidra, by right clicking the variable and selecting “Rename variable”. So lets rename the variables.

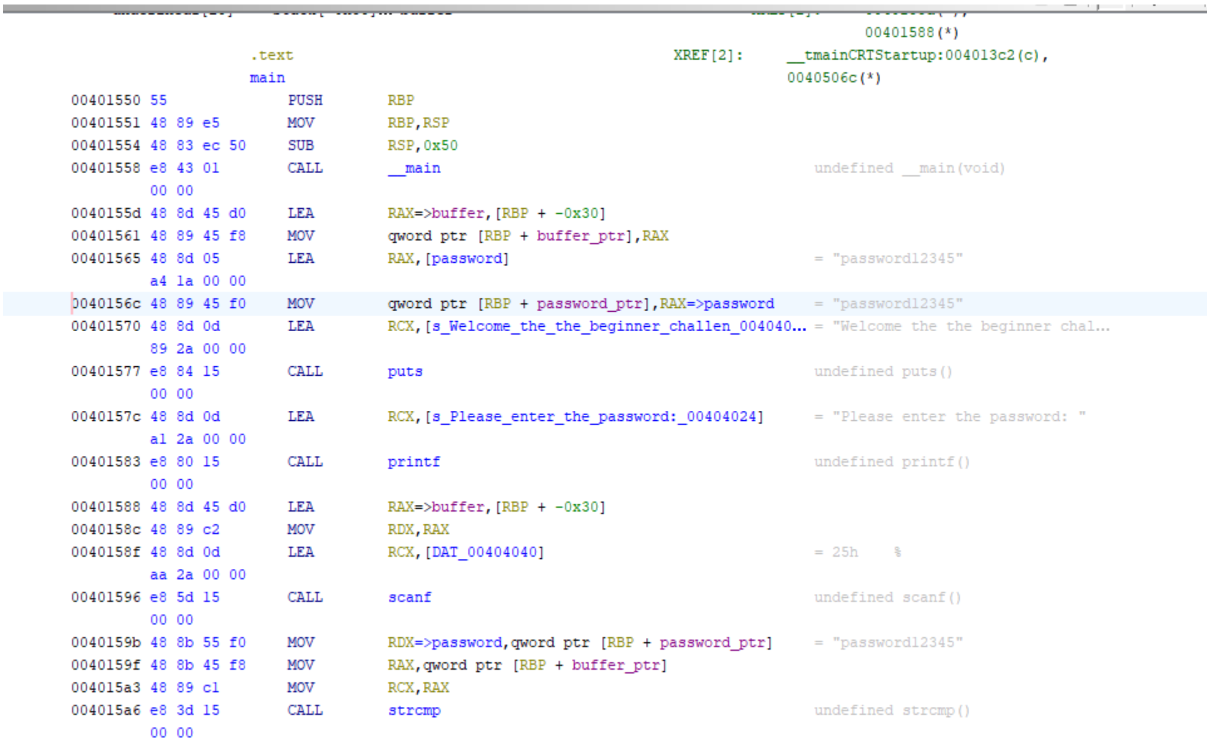
- local_38 -> buffer
- local_10 -> buffer_ptr
- local_18 -> password_ptr



```
 1
 2 undefined8 main(void)
 3
 4 {
 5   undefined buffer [28];
 6   int local_1c;
 7   char *password_ptr;
 8   undefined *buffer_ptr;
 9
10   __main();
11   buffer_ptr = buffer;
12   password_ptr = password;
13   puts("Welcome the the beginner challenge!");
14   printf("Please enter the password: ");
15   scanf(&DAT_00404040,buffer);
16   local_1c = strcmp(buffer_ptr,password_ptr);
17   if (local_1c == 0) {
18     printf("\nWell done! Correct password!");
19   }
20   else {
21     printf("\nUnfortunately thats not the correct password");
22   }
23   printf("\nClosing program soon...");
24   sleep(5);
25   return 0;
26 }
27
```

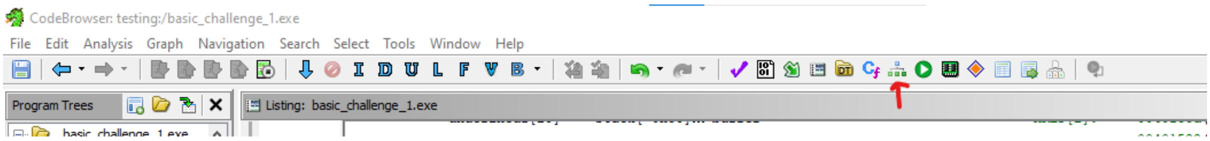
We can see on line 15, the *scanf* function will take user input and store into “buffer” variable. Then on line 16, the *strcmp* function gets used to compare the string inside the “buffer” variable and the “password” variable. This value gets stored in *local_1c*. Now, if *strcmp* returns 0, then the strings the same, and it prints that you have the correct password.

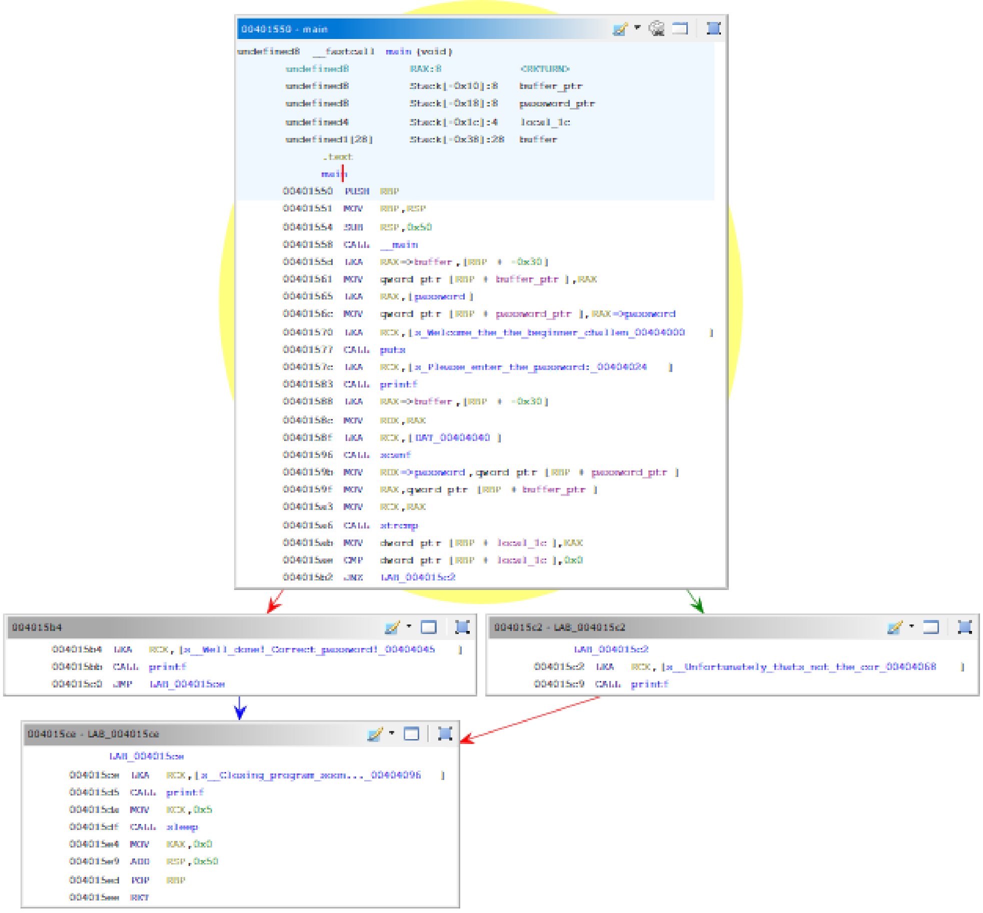
Now, lets investigate the assembly code for the main function.



```
                                                                              00401588(*)
                            .text                                XREF[2]:     __tmainCRTStartup:004013c2(c),
                            main                                              0040506c(*)
00401550 55                 PUSH       RBP
00401551 48 89 e5           MOV        RBP,RSP
00401554 48 83 ec 50        SUB        RSP,0x50
00401558 e8 43 01           CALL       __main                                 undefined __main(void)
         00 00
0040155d 48 8d 45 d0        LEA        RAX=>buffer,[RBP + -0x30]
00401561 48 89 45 f8        MOV        qword ptr [RBP + buffer_ptr],RAX
00401565 48 8d 05           LEA        RAX,[password]                         = "password12345"
         a4 1a 00 00
0040156c 48 89 45 f0        MOV        qword ptr [RBP + password_ptr],RAX=>password   = "password12345"
00401570 48 8d 0d           LEA        RCX,[s_Welcome_the_the_beginner_challen_004040…  = "Welcome the the beginner chal…
         89 2a 00 00
00401577 e8 84 15           CALL       puts                                   undefined puts()
         00 00
0040157c 48 8d 0d           LEA        RCX,[s_Please_enter_the_password:_00404024]    = "Please enter the password: "
         a1 2a 00 00
00401583 e8 80 15           CALL       printf                                 undefined printf()
         00 00
00401588 48 8d 45 d0        LEA        RAX=>buffer,[RBP + -0x30]
0040158c 48 89 c2           MOV        RDX,RAX
0040158f 48 8d 0d           LEA        RCX,[DAT_00404040]                     = 25h    %
         aa 2a 00 00
00401596 e8 5d 15           CALL       scanf                                  undefined scanf()
         00 00
0040159b 48 8b 55 f0        MOV        RDX=>password,qword ptr [RBP + password_ptr]   = "password12345"
0040159f 48 8b 45 f8        MOV        RAX,qword ptr [RBP + buffer_ptr]
004015a3 48 89 c1           MOV        RCX,RAX
004015a6 e8 3d 15           CALL       strcmp                                 undefined strcmp()
         00 00
```

We can press this button to see the function in graph mode:





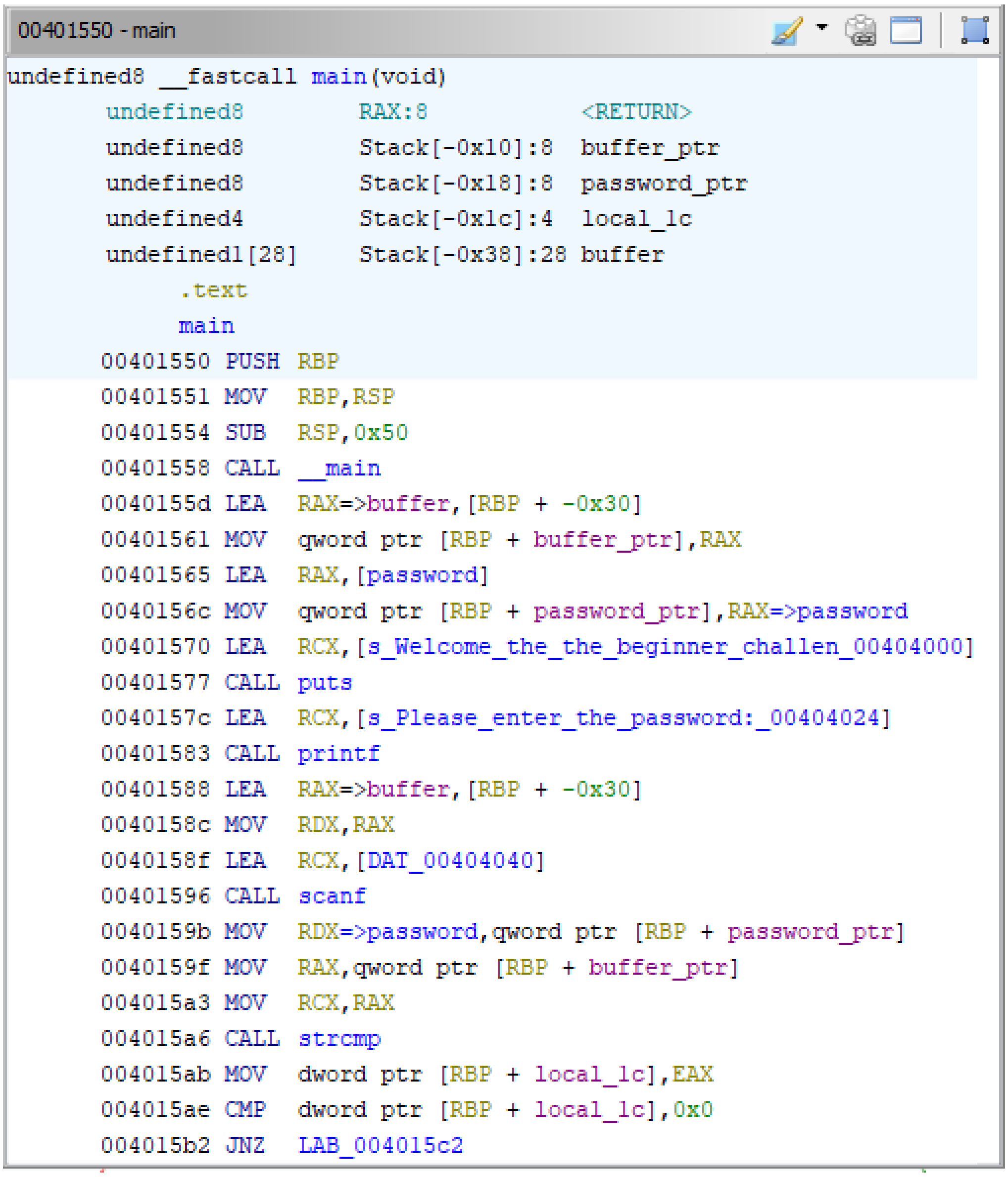
```
00401550 - main

undefined8 __fastcall main(void)
     undefined8       RAX:8            <RETURN>
     undefined8       Stack[-0x10]:8   buffer_ptr
     undefined8       Stack[-0x18]:8   password_ptr
     undefined4       Stack[-0x1c]:4   local_1c
     undefined1[28]   Stack[-0x38]:28  buffer
             .text
             main
     00401550  PUSH  RBP
     00401551  MOV   RBP,RSP
     00401554  SUB   RSP,0x50
     00401558  CALL  __main
     0040155d  LEA   RAX=>buffer,[RBP + -0x30]
     00401561  MOV   qword ptr [RBP + buffer_ptr],RAX
     00401565  LEA   RAX,[password]
     0040156c  MOV   qword ptr [RBP + password_ptr],RAX=>password
     00401570  LEA   RCX,[s_Welcome_the_the_beginner_challen_00404000]
     00401577  CALL  puts
     0040157c  LEA   RCX,[s_Please_enter_the_password:_00404024]
     00401583  CALL  printf
     00401588  LEA   RAX=>buffer,[RBP + -0x30]
     0040158c  MOV   RDX,RAX
     0040158f  LEA   RCX,[DAT_00404040]
     00401596  CALL  scanf
     0040159b  MOV   RDX=>password,qword ptr [RBP + password_ptr]
     0040159f  MOV   RAX,qword ptr [RBP + buffer_ptr]
     004015a3  MOV   RCX,RAX
     004015a6  CALL  strcmp
     004015ab  MOV   dword ptr [RBP + local_1c],EAX
     004015ae  CMP   dword ptr [RBP + local_1c],0x0
     004015b2  JNZ   LAB_004015c2
```

```
004015b4
     004015b4  LEA   RCX,[s__Well_done!_Correct_password!_00404045]
     004015bb  CALL  printf
     004015c0  JMP   LAB_004015ce
```

```
004015c2 - LAB_004015c2
             LAB_004015c2
     004015c2  LEA   RCX,[s__Unfortunately_thats_not_the_cor_00404068]
     004015c9  CALL  printf
```

```
004015ce - LAB_004015ce
             LAB_004015ce
     004015ce  LEA   RCX,[s__Closing_program_soon..._00404096]
     004015d5  CALL  printf
     004015da  MOV   ECX,0x5
     004015df  CALL  sleep
     004015e4  MOV   EAX,0x0
     004015e9  ADD   RSP,0x50
     004015ed  POP   RBP
     004015ee  RET
```

Lets investigate the first block.



```
undefined8 __fastcall main(void)
        undefined8          RAX:8           <RETURN>
        undefined8          Stack[-0x10]:8  buffer_ptr
        undefined8          Stack[-0x18]:8  password_ptr
        undefined4          Stack[-0x1c]:4  local_1c
        undefined1[28]      Stack[-0x38]:28 buffer
              .text
             main
       00401550 PUSH  RBP
       00401551 MOV   RBP,RSP
       00401554 SUB   RSP,0x50
       00401558 CALL  __main
       0040155d LEA   RAX=>buffer,[RBP + -0x30]
       00401561 MOV   qword ptr [RBP + buffer_ptr],RAX
       00401565 LEA   RAX,[password]
       0040156c MOV   qword ptr [RBP + password_ptr],RAX=>password
       00401570 LEA   RCX,[s_Welcome_the_the_beginner_challen_00404000]
       00401577 CALL  puts
       0040157c LEA   RCX,[s_Please_enter_the_password:_00404024]
       00401583 CALL  printf
       00401588 LEA   RAX=>buffer,[RBP + -0x30]
       0040158c MOV   RDX,RAX
       0040158f LEA   RCX,[DAT_00404040]
       00401596 CALL  scanf
       0040159b MOV   RDX=>password,qword ptr [RBP + password_ptr]
       0040159f MOV   RAX,qword ptr [RBP + buffer_ptr]
       004015a3 MOV   RCX,RAX
       004015a6 CALL  strcmp
       004015ab MOV   dword ptr [RBP + local_1c],EAX
       004015ae CMP   dword ptr [RBP + local_1c],0x0
       004015b2 JNZ   LAB_004015c2
```

The first two instructions,

PUSH RBP

MOV RBP, RSP

We see at the start of function calls.

At **0040155d** the *Load Effective Address* instruction gets used to store the address of the “buffer” variable into the RAX register. Note that RBP-0x30 represents the memory address on the stack where the variable “buffer” gets stored. Note that the name changes we did in the decompiler shows up in the assembly.

At **00401561**, we move whatever was in RAX into the local variable “buffer_ptr”.

In the next two lines, we store the memory address of the “password” global variable into RAX then store that value into local variable “password_ptr”.

At **00401570**, me move the memory address of a string into RCX, then on the next line we can see that *puts* function gets called, to print to console. Note, this program is using the assembly calling convention where we put the first input of the function into the RCX variable. There are different calling conventions, in 32-bit programs you would see the inputs getting pushed onto the stack in reverse order.

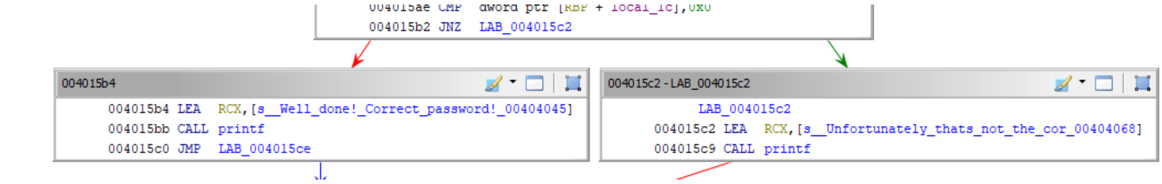
Then, another string gets stored into RCX then the *printf* functions shows that string on the console.

At **00401588**, the address of “buffer” gets stored into RAX, then this address gets moved to RDX and a different address gets stored into RCX. After that, the *scanf* function gets called. Note, in the calling convention that this program uses, the first input gets stored into RCX and the second input gets stored in RDX. Now, these instructions stores the user input into the ”buffer” variable.

At **0040159b**, the “password_ptr” variable gets stored into RDX, the “buffer_ptr” gets stored in RAX, then RAX is moved into RCX. After that, the *strcmp* function gets called. The inputs to this function are stored in RCX and RDX, which contains “buffer_ptr” variable and “password_ptr” variable. The *strcmp* function returns 0 if the strings the same. Right after the function call, the value in the EAX register gets moved into local variable “local_1c”. After the *strcmp* function runs, the return value of that function gets stored in the EAX register, which is why we see the value in EAX getting stored in a this variable.

At **004015ae**, the local variable “local_1c” gets compared to 0x0. Now this *cmp* command will subtract the second argument from the first argument, however the result will not get stored. Rather, this is used to update the flags in the flags register. If “local_1c” – 0x0 equals to 0, then the zero flag will get set.

At **004015b2**, we see the instruction *JNZ LAB_004015c2*. This means that if the 0 flag is not set, then jump to that location in memory.



```
004015ae CMP   dword ptr [RBP + local_1c],0x0
004015b2 JNZ   LAB_004015c2
```

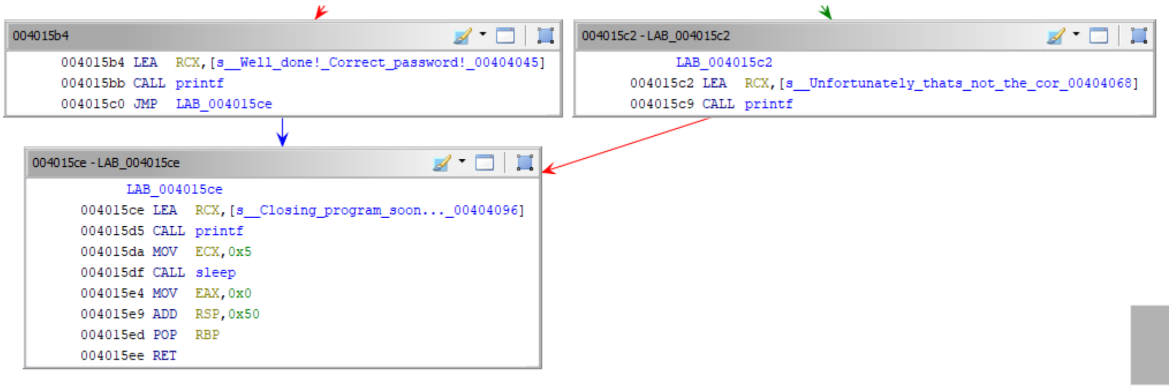
004015b4

```
004015b4 LEA   RCX,[s__Well_done!_Correct_password!_00404045]
004015bb CALL  printf
004015c0 JMP   LAB_004015ce
```

004015c2 - LAB_004015c2

```
         LAB_004015c2
004015c2 LEA   RCX,[s__Unfortunately_thats_not_the_cor_00404068]
004015c9 CALL  printf
```

Note that if the 0 flag is not set, that means that “local_1c” does not equal 0, which means that the *strcmp* function did not return 0. Remember, if *strcmp* returns 0, then the strings equal. We can see in the screenshot that *LAB_004015c2* represents the code that runs if the password’s not correct. Meanwhile, if the correct password was entered, then the zero flag would have been set at the *cmp* instruction, therefore it would not jump and instead move to instruction **004015b4**. This block prints out the success message.



004015b4

```
004015b4 LEA   RCX,[s__Well_done!_Correct_password!_00404045]
004015bb CALL  printf
004015c0 JMP   LAB_004015ce
```

004015c2 - LAB_004015c2

```
         LAB_004015c2
004015c2 LEA   RCX,[s__Unfortunately_thats_not_the_cor_00404068]
004015c9 CALL  printf
```

004015ce - LAB_004015ce

```
         LAB_004015ce
004015ce LEA   RCX,[s__Closing_program_soon..._00404096]
004015d5 CALL  printf
004015da MOV   ECX,0x5
004015df CALL  sleep
004015e4 MOV   EAX,0x0
004015e9 ADD   RSP,0x50
004015ed POP   RBP
004015ee RET
```

The final block just prints that the program would close and moves 5 in ECX then class the *sleep* function to pause the program for 5 seconds.

Then, the program moves 0 into EAX, adds 50 to RSP, POP RBP and then return. Remember how the *main* function in C returns an integer? That's why we move 0 into EAX here, because in the original program we have "return 0;" at the end. We have to add 50 to RSP, the stack pointer, because we need to clean up the stack after the function's done executing. The POP RBP resets the RBP to what the value was before the function was called, and return just returns to where the function was called. Note that in the original program the *main* function is where the program starts, but in the actual executable, the *main* function gets called from somewhere else.