start gdb

```
$ gdb ./level1
```

set breakpoint to main

```
(gdb) b main
```

execute the program:

```
(gdb) run
```

once gdb hits the breakpoint, take a look at the disassembly:

```
(gdb) disassemble
Dump of assembler code for function main:
   0x000000000040069d <+0>:     push   rbp
   0x000000000040069e <+1>:     mov    rbp,rsp
=> 0x00000000004006a1 <+4>:     sub    rsp,0x90
   0x00000000004006a8 <+11>:    mov    DWORD PTR [rbp-0x84],edi
   0x00000000004006ae <+17>:    mov    QWORD PTR [rbp-0x90],rsi
   0x00000000004006b5 <+24>:    mov    rax,QWORD PTR fs:0x28
   0x00000000004006be <+33>:    mov    QWORD PTR [rbp-0x8],rax
   0x00000000004006c2 <+37>:    xor    eax,eax
   0x00000000004006c4 <+39>:    mov    QWORD PTR [rbp-0x78],0x4007d8
   0x00000000004006cc <+47>:    lea    rax,[rbp-0x70]
   0x00000000004006d0 <+51>:    mov    edx,0x64
   0x00000000004006d5 <+56>:    mov    esi,0x0
   0x00000000004006da <+61>:    mov    rdi,rax
   0x00000000004006dd <+64>:    call   0x400560 <memset@plt>
<SNIP>
```

We can see that the first thing main does is allocate memory for the stack. Sometime after that, the address `0x4007d8` is copied to `rbp-0x78` (to the stack).

Later we see that same value is being loaded to `rdx` before calling strcmp. (from the stack)

```
0x00000000004006f8 <+91>:    mov    rdx,QWORD PTR [rbp-0x78]
0x00000000004006fc <+95>:    lea    rax,[rbp-0x70]
0x0000000000400700 <+99>:    mov    rsi,rdx
0x0000000000400703 <+102>:   mov    rdi,rax
0x0000000000400706 <+105>:   call   0x400580 <strcmp@plt>
```

hmm... this looks like a parameter to strcmp! Let's investigate:

```
(gdb) x/s 0x4007d8
0x4007d8:       "abcdefghjkl123!"
```

Try and see if this string is the correct password

Solution 2: run “string level1” from the command line :)