# CSC 180 lab 8: More on input

Thurs Nov 1 or Mon Nov 5, 2001

**1.** Enter the following program from a lecture example (or copy it from ~ajr/lab8/age.c):

```
#include <stdio.h>

int main()
{
    int age;
    printf("How old are you?\n");
    scanf("%d", &age);
    printf("Big deal, I'm %d\n", age + 3);
    return 0;
}
```

Compile and test it.

**2.** What happens if you type a word instead of a number?

**3.** When typing in a word instead of a number, on one computer I got the output "Big deal, I'm 134517995"; on another computer I got the output "Big deal, I'm 3". How can the results be different without this representing a bug in the C compiler?

**4.** We can improve the program as follows:

```
#include <stdio.h>

int main()
{
    int age;
    printf("How old are you?\n");
    if (scanf("%d", &age) == 1)
        printf("Big deal, I'm %d\n", age + 3);
    else
        printf("That is not a valid age.\n");
    return 0;
}
```

Modify your program to resemble the above. Compile and test it, including testing it with non-numeric input.

**5.** It would be nicer if when complaining about the invalid entry, the program instead looped around and solicited the input again. What happens when you do the following?

```
    while (scanf("%d", &age) != 1)
        printf("That is not a valid age.\n");
    printf("Big deal, I'm %d\n", age + 3);
```

Try it. Explain the results. (Press control-C to stop your program, or just close the window.)

**6.** This illustrates why we rarely scanf() numbers in practice. The alternative is to read user input as a string and then analyze it. We use *fgets*() to read one line of user input as a string. There is a function called *sscanf*() which analyzes a string just the way scanf() analyzes the standard input.

Modify the program, without the above loop, to declare a char array of size 50, call fgets(array, 50, stdin), and then call sscanf on the array, as was done in class.

**7.** Put in a loop with fgets() and sscanf() so that your program keeps re-prompting "How old are you?" after it says that an age is invalid, until it gets a numeric input.

*(over)*

**8.** If fgets() returns NULL, that indicates EOF. Make sure your program behaves reasonably if you press control-D. (It should simply exit. But if you ignore this error condition, it will probably loop quickly, re-prompting like crazy.)

**9.** A negative age is impossible. An age less than four is unlikely. Make your program emit error messages to this effect, and loop around and get another age attempt, until either a reasonable age is correctly input, or the user presses control-D to signal EOF, or something worse happens (such as the destruction of civilization and the termination of electrical power to the computer (don't attempt to test this)).

**10.** Since we are inputting a string, we can add a few other alternatives. If they just press return, instead of "That is not a valid age", output "Don't be shy!", but still loop around and re-solicit the input. If they type "guess", print "Ok, I guess 18", set age to 18, and proceed.

Use strcmp() to compare the user input with "\n" and "guess\n". You'll have to #include <string.h> to call strcmp(). You can look up more information about this function in the index of our textbook.

---

Answer to question 3 above: Use of the 'age' variable without initializing it is invalid, and any computer behaviour is permissible when you break the rules in this way; it could be worse than merely supplying a crazy number. So normally we should be very careful not to do this. In the case of this program, we could be careful to avoid using an uninitialized variable 'age' by testing the scanf result, which we do in question 4.