Lecture 12: User Input

Standard streams:

In any OS, there are standard streams for streaming data with its various types. The main three streams to interact with are: Standard Input, Standard Output, and Standard Error.
Standard Input is for taking data from the command line, Standard Output is for giving output to the command line, and Standard Error is a separated output stream specified for producing error messages. You might see some representations for these streams as 0 for Standard Input, 1 for Standard Output, and 2 for Standard Error. In C, we will be representing these streams with ‘stdin’ for Standard Input, ‘stdout’ for Standard Output, and ‘stderr’ for Standard Error.

When we want to get user data in our case, we can get this data from several sources such as Standard Input, a file, a network, or some other type of source. In our lecture, we will be covering how to get user input from the Standard Input stream.

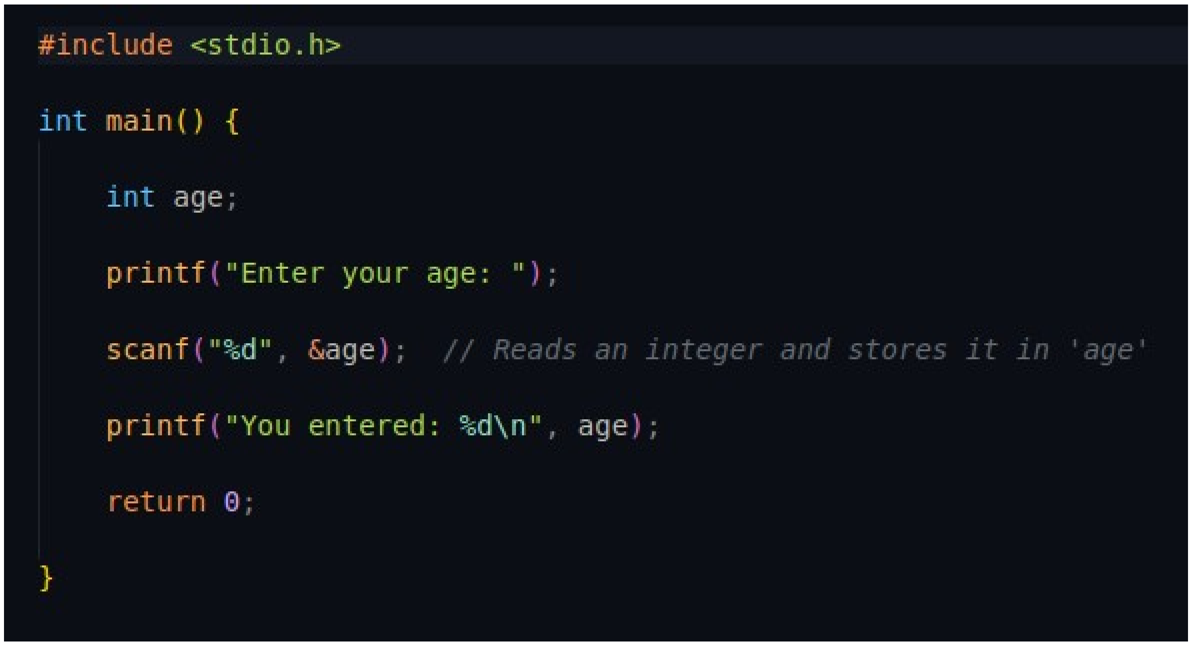
C functions to receive input from the user:

The C language provides us with several libraries that have several ways of handling input from the Standard Input stream. The <stdio.h> library have some that we will be covering them in this lecture:

1. `scanf(“”)`:
   This function takes formatted from the stdin stream, and assign it to a given variable, in a syntax similar to that of the printf functions.

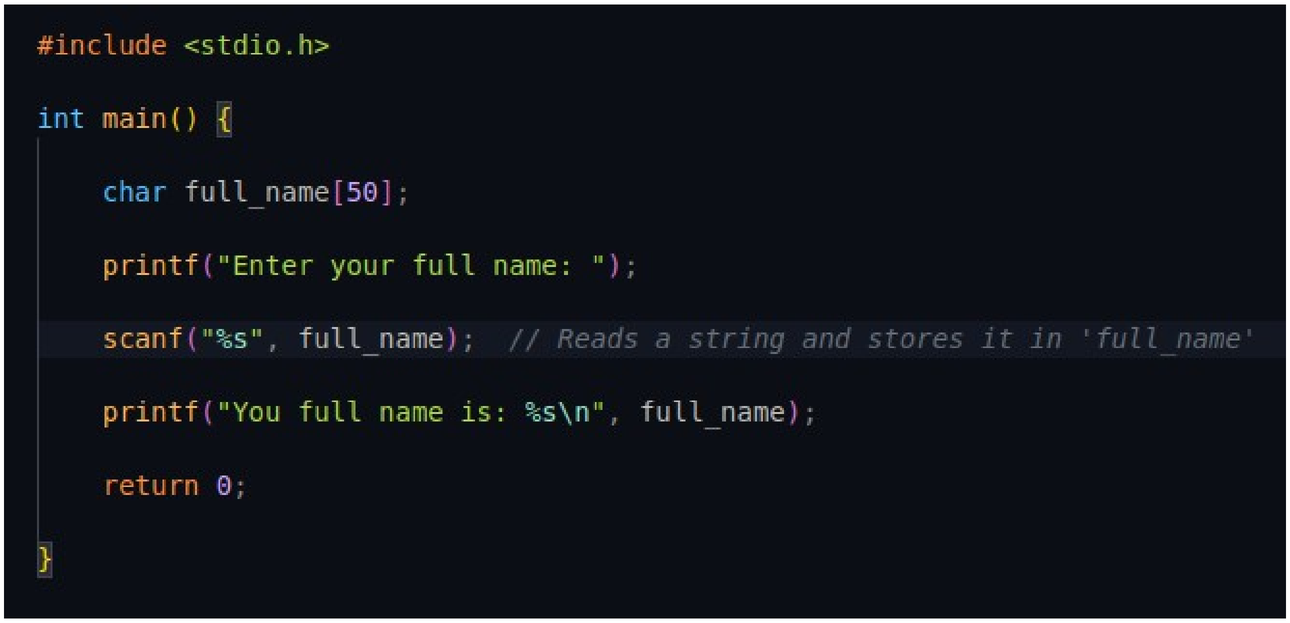
   ```
   scanf(“%d %f %c %s”, int, float, char, char*);
   ```

   Here is an example for that:



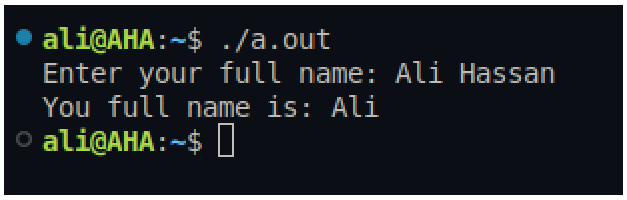
```c
#include <stdio.h>

int main() {

    int age;

    printf("Enter your age: ");

    scanf("%d", &age);  // Reads an integer and stores it in 'age'

    printf("You entered: %d\n", age);

    return 0;

}
```

One downside of the `scanf()` function is that it stops the scanning and taking the input once it encounters a white space in the input. For example:



```c
#include <stdio.h>

int main() {

    char full_name[50];

    printf("Enter your full name: ");

    scanf("%s", full_name);  // Reads a string and stores it in 'full_name'

    printf("You full name is: %s\n", full_name);

    return 0;

}
```

When running the above code, and entering a sentence with white space in it, the output will be as follows:



```
ali@AHA:~$ ./a.out
Enter your full name: Ali Hassan
You full name is: Ali
ali@AHA:~$ 
```

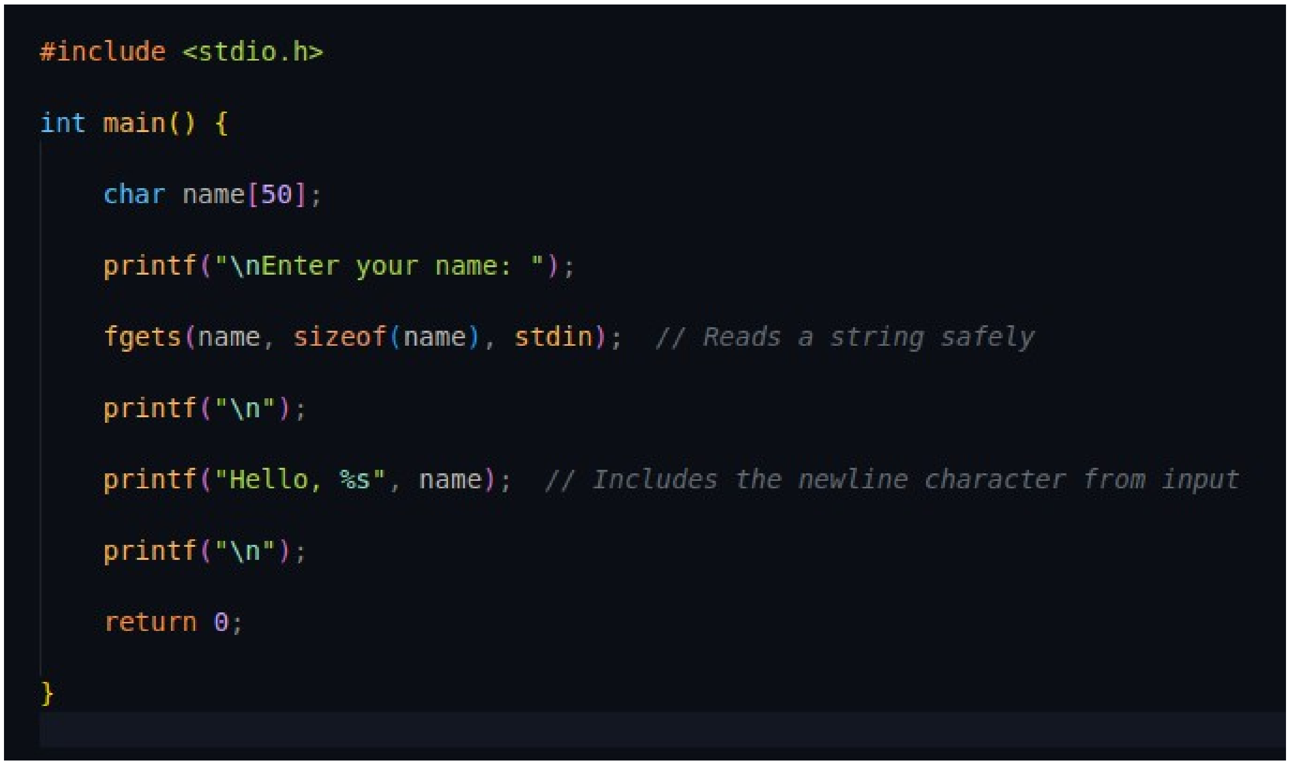
As we see, the scanning stopped once the function encountered white space in the middle of the sentence, which is why this function is not the ideal method of getting data from the user.

2. `fgets(a, b, c)`:
   This is another function provided in the `<stdio.h>` library that also takes input from the user, and stores it in a variable, but it has a couple more arguments that it takes that make it’s functionality a bit more efficient and practical than other options.
   The syntax of the function is as follows:

   ```
   fgets(var, size of the var, stream to read from);
   ```

   But what does these value mean?
   - `Var`: this is the variable that will store the input that will come from the user
   - `size of the var`: this is the value that will indicate the maximum size that the `Var` can hold. For example, if we declared an integer, then we will tell the function that the maximum value the variable can hold is the size of the integer, to prevent any memory collision that can happen if the user gave us a value that is bigger than what the variable can hold.
   - `Stream to read from`: this is the value that will tell the function what will be the stream that it needs to watch to get the values it waits from. For example, we can tell the function to take the values from the stderr stream, so that we can store of later use the errors that we might encounter. Or in our case, we will be telling the function to take the values from the stdin stream, so that we can take the input that will given by the user.

   Here is an example of the above:



```c
#include <stdio.h>

int main() {

    char name[50];

    printf("\nEnter your name: ");

    fgets(name, sizeof(name), stdin);  // Reads a string safely

    printf("\n");

    printf("Hello, %s", name);  // Includes the newline character from input

    printf("\n");

    return 0;

}
```

The first thing to notice is that the function can deal with the white spaces that it counters in the user input with no problems, which gives it another advantage over the previous function.

Things to watch out when dealing with user input:

The number one rule in the coding world is DON'T TRUST THE USER FOR INPUTTING DATA!!!. Whenever we are expecting the user to give us any source of data, we will need to do some operations afterwards to make sure that the data is valid for use, for example:

- Validate the type of the data given by the user.
- Validate the value of the input.
- Validate the nature of the values give by the user.
- Make sure that there is no more or less data than needed.
- Watch out for data that could mess up the work ;).

Best wishes, Eng. Ali Hassan.