Common mistakes made with scanf (and another called printf for reference)

The purpose of this document is to highlight typical errors made when using *scanf*. In order to understand the mistakes and where they originate, we have included a short discussion on *printf* and the use of *scanf*. If you only want to look at the list for quick reference, scroll to the bottom.

Background

In order to utilise the screen (output) and keyboard (input), the programmer must use functions (tools) from a library (toolbox of pre-existing functions). These functions are the *printf* and *scanf* functions from the *stdio.h* library.

The *printf* function enables the programmer to display characters on the screen (console) as output from the program. *printf* is used to prompt the user of the program.

The *scanf* funtion enables the programmer to take characters from the keyboard as input into the program. *scanf* is used to get information from the user.

Use of printf

Different variable types are represented differently in the memory. We therefore need to tell the computer what type the variable is in when we print it as text to the screen. We use the % control to tell the compiler what the **type** is what we need to print. We then place what we need to print after the string. If multiple references are made, insertions happen in the order they are given.

To print a % on the screen therefore, we need to use %%

Type	Example	Ouput
string	printf("Hello world");	Hello world
string	printf("**%s**", "Hello world");	Hello world
character	printf("**%c**", 's');	s
integer	printf("**%d**", 234);	234
float	printf("**%f**", 13.5);	13.500000
rounded float	printf("**%0.2f**", 13.5);	13.50
complex	printf("I am **%s**, **%d** years old");	I am JP, 10 years old
complex	printf("I am **%0.2f %%** certain", 95.2);	I am 95.2% certain.
string	printf("I am 92.5%% certain");	I am 95.2% certain.

To insert a new-line, we use escape characters, such as "\n", which is the escape sequence for newline. :

printf("Hello world\n");

Use of scanf

To get variables from the user (keyboard) into the program, we use *scanf*. *scanf* is the same as *printf*, in that we need to specify the type of the information read in as text from the keyboard. Because

scanf reads the typed information directly into the memory address of a variable, we need to use the memory address of the variable we want to read it into, rather than the name of the variable. We get the address by using the **ampersand (&)** character. For example, the address of a variable called *myVariable*, is *&myVariable.*

Type	Example	Notes

integer	int myInteger; //declaration
	scanf("**%d**", &myInteger); //assignment
		Gets the information from the keyboard, **interprets** it as an integer, and puts it into a variable called myInteger.

float	float myFloat; //declaration
	scanf("**%f**", &myFloat); //assignment
		Gets the information from the keyboard, **interprets** it as a float, and puts it into a variable called myFloat.

string	// you do not need to know how to assign yet scanf("**%s**", &myString); //assignment	Gets the information from the keyboard, **interprets** it as a string, and puts it into a variable called myString.

Common mistakes with scanf

The majority of mistakes with *scanf* relate to using *printf* approaches (which is why it is included in this document). The most common errors are listed here.

Error	Example
Reading into the variable, instead of the memory address. (no ampersand)	scanf("%d", myInteger);
Use of escape characters in scanf	scanf("%d**\n**", &myInteger);
Reading into the wrong type of variable read into (interpret as one type and put into variable of another type)	scanf("**%f**", &my**Integer**);
Use of a space between the % and the type indicator	scanf("% f", &myInteger);
Use of rounding in scanf	scanf("%**0.2**f", &myFloat);