Strings in C Programming

DECLARATION STATEMENT

A string in C is actually a character array. There are several methods of declaring the variable. This first example declares a variable that can hold 4 characters. Below it is the initialized version of the same declaration. The 5th space is for the end of string character that is automatically added to the end of all strings:

```
char var[5];
char var[5] = "abcd";
char var[] = "abcd";          /* Equivalent to above.   */
```

This type of declaration **precludes the subsequent use of the assignment operator** to change the value stored in var. However, the value may be changed by using functions such as **`strcpy()`**, **`fscanf()`**, and **`fgets()`**.

Another declaration method is to declare a pointer variable. Notice in the first example a size has not be determined. The assignment operator **may** be used to initialize the array later but functions **may not** be used for initialization. Once initialized, the maximum size of the array has been set as far as functions are concerned and functions may be used to change the value. I think the assignment operator may be used to subsequently assign longer strings to the pointer but I am not sure yet. The second example shows initialization during declaration. p345

```
char *var;
char *var = "abcd";
```

SCANF()

The **`scanf()`** function requires the use of addresses of variables.

syntax: `scanf("`*control string(s)*`", &`*variable(s)*`);`
i.e.: `scanf("%d %d", &num1, &num2);`

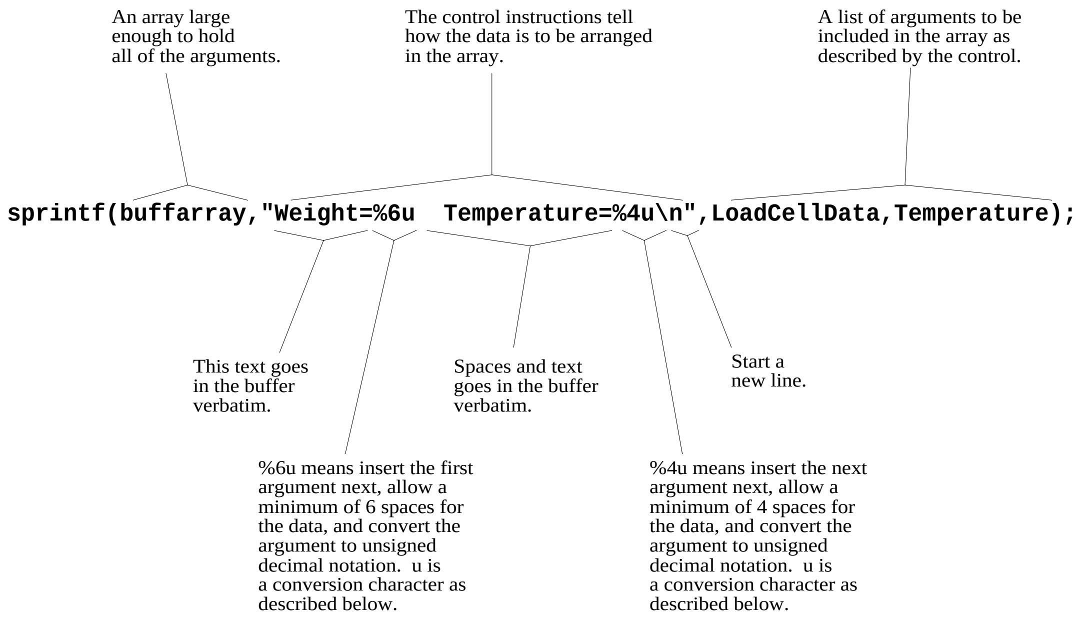
When using the **`scanf()`** function to read a character or string from the keyboard, **empty the buffer** afterward (the carriage return is still in there) using the following code:

```
fflush(stdin);
```

SPRINTF()

The **`sprintf()`** function takes a list of arguments and formats them into an array.

syntax: `sprintf(`*array,* `"`*control string(s)*`",` *variable(s)*`);`



```
sprintf(buffarray,"Weight=%6u  Temperature=%4u\n",LoadCellData,Temperature);
```

An array large enough to hold all of the arguments.

The control instructions tell how the data is to be arranged in the array.

A list of arguments to be included in the array as described by the control.

This text goes in the buffer verbatim.

Spaces and text goes in the buffer verbatim.

Start a new line.

%6u means insert the first argument next, allow a minimum of 6 spaces for the data, and convert the argument to unsigned decimal notation. u is a conversion character as described below.

%4u means insert the next argument next, allow a minimum of 4 spaces for the data, and convert the argument to unsigned decimal notation. u is a conversion character as described below.

The conversion characters are:

d	decimal notation
o	unsigned octal notation
x	unsigned hexadecimal notation
u	unsigned decimal notation
c	a single character
s	string
e	decimal notation of a float or double in the form m.nnnnnnE±xx The number of n's may be specified.
f	decimal notation of a float or double in the form mmm.nnnnn The number of n's may be specified.
g	Use %e or %f, whichever is shorter

PASSING STRINGS TO FUNCTIONS

To pass addresses to a function (referred to as ***pass by reference***), you can use the array name. If your function needs to know how may elements are in the array, you can pass that value as a second argument:

FUNCTION PROTOTYPE

```
void MyFunct(char []);
void MyFunct(char [],int);
```

FUNCTION CALL

```
MyFunct(ArrayName);
MyFunct(ArrayName,HowMany);
```

FUNCTION HEADER

```
void MyFunct(AryNm[])
void MyFunct(AryNm[],Num)
```

If you have declared a pointer to the array (see the sheet on pointers) you can pass the pointer. Be sure your function expects a pointer to an array:

FUNCTION PROTOTYPE

```
void MyFunct(char *);
void MyFunct(char *,int);
```

FUNCTION CALL

```
MyFunct(Ptr);
MyFunct(Ptr,HowMany);
```

FUNCTION HEADER

```
void MyFunct(*P)
void MyFunct(*P,Num)
```