C Programming

Strings

String literals

A string literal is any sequence of characters enclosed in **double quotes**. For example, if I have the following:

```
printf("Hello");
```

In this case, "Hello" is a string literal. All the characters in the string literal are stored in a contiguous block of memory.

A string literal ends with a special character called the **NULL character / NULL terminator**.

The NULL character is used to tell the compiler where the end of the string is located. The NULL character is represented by a **'\0'**

'H'	'e'	'l'	'l'	'o'	'\0'

NULL character

When the printf("Hello") executes, the OS will automatically go to the memory address of the first character, i.e., 'H', and display the contents of every memory location that follows up to but excluding the NULL character. This is the only way the OS knows where the string literal ends.

So, this begs the question ….

How do you display the following string literals to standard output:

- "I can program in C", said the student.

  ```
  printf("\"I can program in C\", said the student.");
  ```

- C:\ is the root directory

  ```
  printf("C:\\ is the root directory");
  ```

The Escape character, i.e., a single backslash, is really used as an instruction character to tell the OS to do something.

Character	Meaning
\n	new line
\"	display a double quote
\'	display a single quote
\0	NULL character
\t	display a tab
\b	remove a space, backspace
\a	alert, ping, chime noise

Long character strings

There will be times when you need to display a string that is quite long in a program. For example:

"This is a very long string that might span over a few lines in whichever code editor you like"

We can write the code for this string literal in the following 3 ways:

- ```
  printf("This is a very long string that might span over a few lines in whichever code editor you like");
  ```
- ```
  printf("This is a very long string that\
  might span over a few lines in whichever\
  code editor you like");
  ```
- ```
  printf("This is a very long string that "
  "might span over a few lines in whichever "
  "code editor you like");
  ```
  ```

Strings and Arrays

In C, a string is an array of characters.

The number of elements in the array, i.e., the size of the array, is equal to the number of characters in the string **PLUS** an additional element for the NULL character to identify the end of the string.

For example:

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

Another way to create a string is:

```
char greeting[6] = "Hello";
```

if the size of the array is bigger than the number of characters in the string PLUS the NULL character, the OS will fill all remaining elements in the array with the NULL character.
e.g., if your array is of size 10

```
char greeting[10] = "Hello";
```

'H'	'e'	'l'	'l'	'o'	'\0'	'\0'	'\0'	'\0'	'\0'

Display a string

C gives you very good flexibility to display a string in a program. Let's have a look at a simple C program to show how the different ways a string can be displayed to standard output, i.e., the screen

```
#include <stdio.h>

int main(void)
{
   char greeting[6] = "Hello";

   printf("%s", greeting);
```

```
    printf("\n");

    // Right-justifies the string by counting 20 spaces from the screen
margin
    printf("%20s", greeting);

    printf("\n");

    // Left-justifies the string by counting 20 spaces from the screen
margin
    printf("%-20s", greeting);

    printf("\n");

    // Displays the first 3 characters of the string
    printf("%.3s", greeting);

    printf("\n");

   // Displays the first 3 characters of the string, width 20
   printf("%20.3s", greeting);

 return 0;
}
```

Repl 17.1: https://replit.com/@michaelTUDublin/171-Display-a-string

The gets() and puts() functions

1. puts()

puts() is a built-in function that is used to display a string. puts() is only used to display a string and cannot be used to display any other regular data type variable, e.g., a character variable or an array of integers, floats, etc.,

e.g.,

```
#include <stdio.h>
```

```c
#define SIZE 21

int main()
{
  char name[SIZE];

  printf("Enter your name\n");
  scanf("%s", name); // no ampersand needed in the scanf() for strings

  printf("Hello ");
  puts(name); // automatically moves cursor to a new line after the
string

  return 0;

} // end main()
```

Repl 17.2: https://replit.com/@michaelTUDublin/172-puts

Note:
puts() will display a string to standard output and will automatically move the cursor onto a new line.

2. gets()

gets() is a built-in function that reads a string from standard input, i.e., the keyboard. gets() will read all characters, including whitespace characters, up to the newline character, i.e., the enter key.

e.g., let's modify the program above so that it uses gets()

```c
#include <stdio.h>

#define SIZE 21

int main()
{
   char name[SIZE];
```

```
    printf("Enter your name\n");
    //scanf("%s", name); // no ampersand needed in the scanf() for
strings
     gets(name);
    // this is the alternative for the gets() function because it is now
    // become deprecated, i.e., some compilers will not support gets()
    //fgets(name, SIZE-1, stdin);


    printf("Hello ");
    puts(name); // automatically moves cursor to a new line after the
string
    return 0;

} // end main()
```

Repl 17.3: https://replit.com/@michaelTUDublin/173-gets

Assigning a string to a pointer

Let's look again at how we can initialise a string:

- `char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};`

- `char greeting[6] = "Hello";`

Now, there is a 3rd way to initialise a string as follows:

- `char *ptr = "Hello";`

	'H'	'e'	'l'	'l'	'o'	'\0'
Memory address	*F123*	*F124*	*F125*	*F126*	*F127*	*F128*

The string "Hello" is placed inside RAM in a **contiguous** block of memory. The pointer ptr will store the memory address of the first character, i.e., 'H' and essentially point at this character. By moving the pointer to the next memory address and checking if it contains the NULL character, the OS will know where the string ends and may display all characters in the string up to that point.

Let's take a look at a program that creates a string and assigns it to a pointer..

```
#include <stdio.h>

int main()
{
   // ptr contains the address of the 1st character in the string, 'H'
   char *ptr = "Hello";

   // (i) printf("%s", ptr);

   // (ii) display the string character by character
```

```
    // while the contents of the address pointed by ptr is NOT
    // the NULL character
    while (*ptr != '\0')
    {
        printf("%c", *ptr);

        // Must increment the address in prt to point at the next
        // address, i.e., the next character in the string
        ptr++;

    } // end while

    return 0;

} // end main
```

In summary, have a look at the following two string declarations….

```
char *p = "some text";

char my_array[] = "some text";
```

's'	'o'	'm'	'e'		't'	'e'	'x'	't'	'\0'

F123

Question:

1. Can I write the following C code:

```
    // can I say:
    p = my_array;

    printf("\n%c", *p);
```

YES, you can write this. You are assigning the memory address of the first element in the array, i.e., &my_array[0] to the pointer p. Pointer p will now contain that memory address and is pointing at the first element of the array.

2. Can I write the following C code:

```
// can I say:
my_array = p;


printf("\n%c", *my_array);
```

NO. This will not work because the array name is the same as the address of the 1st element of the array. Since the array memory block is contiguous and fixed in memory while the program is running, the OS will not allow any code to change the location of the array.