STRING IN C

Dr. Sumit Srivastava
Dept. of CSE, BIT Mesra Ranchi
Email:- sumit@bitmesra.ac.in

String

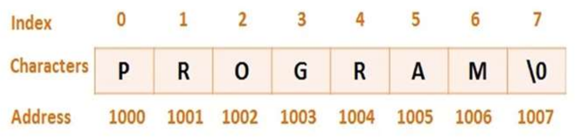
- A String is collection of characters in a linear sequence.
- A String is a sequence of characters terminated with a null character '\0'.
- Strings are used for storing text/characters.
- The String is stored as an array of characters.
- ✓ (The difference between a character array and a C string is that the string is terminated with a unique character '\0'.)

String

- C Strings are single dimensional array of characters ending with a null character('\0').
- Null character marks the end of the string.
- Strings constants are enclosed by double quotes and character are enclosed by single quotes.
- For Example
- String constant : "BITMesraRanchi"
- Character constant: 'B'

String

- If the size of a C string is N, it means this string contains N-1 characters from index 0 to N-2 and last character at index N-1 is a null character.
- Each character of string is stored in consecutive memory location and occupy 1 byte of memory.



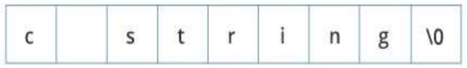
Index	0	1	2	3	4	5	6	7
Characters	P	R	O	G	R	A	M	\0
Address	1000	1001	1002	1003	1004	1005	1006	1007

String

For example:

```
char c[] = "c string";
```

- When the compiler encounters a sequence of characters enclosed in the double quotation marks, it appends a null character \0 at the end by default.



Declaration of Strings

Syntax of String Declaration

char str_name[size];

Here,

- str_name is the string variable's name
- The size is the maximum number of characters the string can hold, excluding the null character.

Example of String Declaration

Here, we have declared a string of 5 characters.

```
char c[] = "abcd";
char c[50] = "abcd";
char c[] = {'a', 'b', 'c', 'd', '\0'};
char c[5] = {'a', 'b', 'c', 'd', '\0'}
```

Example of String Declaration

```
// C Program to illustrate the String declaration
#include <stdio.h>
int main() {
    char message[31]; // declaring the string variable
    printf("Enter a message (up to 30 characters): ");
    scanf("%30s", message); // reading input from the user and storing it in the string
    printf("The message you entered is: %s\n", message); // printing the string
    return 0;
}
```

Initialization of String

- There are 4 ways in which we can initialize string in C language. These are by-.
  1. Assigning a string literal with size
  2. Assigning a string literal without size
  3. Assigning character by character with size
  4. Assigning character by character without size

Assigning String Literal With Size

It allows for the direct assignment of array size and value at once.

- **Syntax:**

  char string name[Size] = “String_Literal”;

Here.
- The string name is the name of the string variable.
- The size is the maximum number of characters in the string (basically, it is the space that will be allocated to the array).
- The “string_Literal” is the string that you are going to assign.

Example

```
// C Program to illustrate the Assigning String Literal With Size

#include<stdio.h> //Header File

int main() //Main Method

{

    char Name[7] = “Mesra"; //Assign A String With Size

    printf("Name : %s", Name); //Print Statement

    return 0;

}
```

Example

Output:

```
Name: Mesra
```

Explanation:

In the example-

•We initialized the character Array named ‘Name’ by assigning it with ‘Programming'.

•This will occupy 6 characters + 1 Null Character ('/0'), which indicates the ending of the string.

•If you don't provide the space for a Null character, then the compiler will automatically add a Null character at the end.

•Also, if the size we provide is lower than the actual character count, then the result will be up to the defined size only.

Example

```
// C Program to illustrate the Assigning String Literal With Size
#include<stdio.h> //Header File
int main() //Main Method
{
    char Name[4] = “Mesra"; //Assign A String With Size
    printf("Name : %s", Name); //Print Statement
    return 0;
}
```

Assigning A String Literal Without Size

- A string variable may be initialized by assigning a literal string to it without specifying the array's size.
- The compiler automatically allocates the null terminator and sufficient memory.

Syntax:

```
char stringName[] = "string literal“;
```

Example

```
// C Program to illustrate the Assigning String Literal Without Size
#include <stdio.h>
int main()
{
    char myString[] = "Hello BIT!"; // Assigning a string literal without size
    printf("%s",myString);
    return 0;
}
```

Assigning Character By Character With Size

- We can initialize a string by assigning it characters individually and specifying the maximum size of the string as an array size.
- **Syntax:**

```
char array_name[static size] = {'C', 'H',' A', 'R', '\0'}
```

Example

```
// C Program to illustrate the Assigning Character By Character With Size

#include<stdio.h>
int main()
{

     char arr[5] = {'c', 'h', 'a', 'r','\0' }; //Assigning character by character
                                               with size and null-terminator ('\0') at
                                               the end of the string is compulsory.

     printf("%s",arr);// Output char

     return 0;
}
```

Assigning Character By Character Without Size

- we can also initialize a string in C by assigning it characters individually without specifying the size of the array.
- The compiler will automatically allocate enough memory to store the string, including the null terminator.
- **Syntax:**

```
char str[ ] = {'s', 't', 'r', 'i', 'n', 'g',}
```

Example

```
// C Program to illustrate the Assigning Character By Character Without Size

#include <stdio.h>
int main( )

{

     char str[]={'H', 'e', 'l', 'l', 'o'}; // We made an array type character
                                           variable named 'str' to store characters
                                           one by one in it, without size specification.

     printf("%s",str);

     return 0;

}
```

Access Strings

- We can access a string by referring to its index number inside square brackets [].
- **Example:**

```
#include <stdio.h>
int main()
{
    char greetings[] = "Hello World!";
    printf("%c", greetings[0]);

    return 0;
}
```

Modify Strings

- To change the value of a specific character in a string, refer to the index number, and use single quotes:
- **Example:**

```
#include <stdio.h>
int main()
{
    char greetings[] = "Hello World!";
    greetings[0] = 'J';
    printf("%s", greetings);
    return 0;
}
```

Loop Through a String

- You can also loop through the characters of a string, using a for loop:
- **Example:**

```
#include <stdio.h>
int main()
{
    char carName[] = "Volvo";
    int i;
    for (i = 0; i < 5; ++i)
      {
         printf("%c\n", carName[i]);
      }
    return 0;
}
```

Read String from the user

- To read a string use the scanf() function
- The scanf() function reads the sequence of characters until it encounters whitespace (space, newline, tab, etc.).

Example

- **scanf() to read a string**

```
#include <stdio.h>
int main()
{
      char name[20];
      printf("Enter name: ");
      scanf("%s", name);
      printf("Your name is %s.", name);
      return 0;
}
```

- Output:

```
Enter name: BIT Mesra
Your name is BIT.
```

Strings – Special Characters

Strings - Special Characters

- Because strings must be written within quotes, C will misunderstand this string, and generate an error.

```
char txt[] = "We are the so-called “students“
              of the BIT Mesra.";
```

- The solution to avoid this problem, is to use the **backslash escape character**.
- The backslash (\) escape character turns special characters into string characters:

Strings - Special Characters

Escape character	Result	Description
\'	'	Single quote
\"	"	Double quote
\\	\	Backslash

Escape Character	Result
\n	New Line
\t	Tab
\0	Null

Strings - Special Characters

The sequence \" inserts a double quote in a string:

```
#include <stdio.h>
int main()
{
    char txt[] = "We are the so-called \“students\" of the BIT Mesra.";
   printf("%s", txt);

 return 0;
}
```

Output: We are the so-called “students" of the BIT Mesra.

Strings - Special Characters

The sequence \' inserts a single quote in a string:

```
#include <stdio.h>
int main()
{
    char txt[] = "It\'s alright.";
    printf("%s", txt);

  return 0;
}
```

Output: It's alright.

29

Strings - Special Characters

The sequence \\ inserts a single backslash in a string:

```
#include <stdio.h>
int main()
{
    char txt[] = "The character \\ is called backslash.";
    printf("%s", txt);

  return 0;
}
```

Output: The character \ is called backslash.

30

Strings - Special Characters

```
#include <stdio.h>
int main()
{
    char txt[] = "Hello\nWorld!";
    printf("%s", txt);

  return 0;
}
```

Output:

```
Hello
World!
```

```
#include <stdio.h>
int main()
{
    char txt[] = "Hello\tWorld!";
    printf("%s", txt);

  return 0;
}
```

Output:

```
Hello        World!
```

31

Strings - Special Characters

```
#include <stdio.h>
int main()
{
    char txt[] = {'H', 'e', 'l', 'l', 'o', '\0'};
    printf("%s", txt);

  return 0;
}
```

Output: Hello

32

String Functions

String Functions

- C also has many useful string functions, which can be used to perform certain operations on strings.
- To use them, you must include the <string.h> header file in your program:

```
#include <string.h>
```

String Length

- To get the length of a string, you can use the **strlen()** function.
- **Example:**

```
#include <stdio.h>
#include <string.h>
int main()
{
  char alphabet[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
  printf("%d", strlen(alphabet));
  return 0;
}
```

- **Output: 26**

String Length

- To get the length of a string, you can use the **strlen()** function.
- **Example:**

```
#include <stdio.h>
#include <string.h>
int main()
{
  char alphabet[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
  printf("Length is: %d\n", strlen(alphabet));
  printf("Size is: %d\n", sizeof(alphabet));
  return 0;
}
```

- **Output: Length is: 26**

  Size is: 27

String Length

- **Example:**

```
#include <stdio.h>
#include <string.h>
int main()
{
    char alphabet[50] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    printf("Length is: %d\n", strlen(alphabet));
    printf("Size is: %d\n", sizeof(alphabet));
return 0;
}
```

- **Output: Length is: 26**

  Size is: 50

Strings - Special Characters

- We used sizeof to get the size of a string/array.
- The sizeof and strlen behaves differently, as sizeof also includes the \0 character when counting.
- sizeof will always return the memory size (in bytes), and not the actual string length.

Concatenate Strings

To concatenate (combine) two strings, you can use the **strcat()** function.

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

int main() {
  char str1[20] = "Hello ";
  char str2[] = "World!";

  // Concatenate str2 to str1 (the result is stored in str1)
  strcat(str1, str2);

  // Print str1
  printf("%s", str1);
  return 0;
}
```

Output: Hello World

Copy Strings

To copy the value of one string to another, you can use the **strcpy()** function.

```
#include <stdio.h>
#include <string.h>
int main()
{
  char str1[20] = "Hello World!";
  char str2[20];

  // Copy str1 to str2
  strcpy(str2, str1);

  // Print str2
  printf("%s", str2);

  return 0;
}
```

Output: Hello World

Compare Strings

To compare two strings, you can use the **strcmp()** function.

It returns 0 if the two strings are equal, otherwise a value that is not 0.

```
char str1[] = "Hello";
char str2[] = "Hello";
char str3[] = "Hi";

// Compare str1 and str2, and print the result
printf("%d\n", strcmp(str1, str2));  // Returns 0 (the strings are equal)

// Compare str1 and str3, and print the result
printf("%d\n", strcmp(str1, str3));  // Returns -4 (the strings are not equal)
```

Output: 0
-4

String Input Output in C

Standard library functions for reading strings

- **gets()** : Reads a line from stdin and stores it into given character array.
- **scanf()** : Reads formatted data from stdin.
- **getchar()** : Returns a character from stdin stream.
- **fscanf()** : Read formatted data from given stream.

String Input Output in C

Standard library functions for printing strings

- **puts()** : Writes a string to stdout stream excluding null terminating character.
- **printf()** : Print formatted data to stdout.
- **putchar()** : Writes a character to stdout stream.
- **fprintf()** : Writes formatted output to a stream.