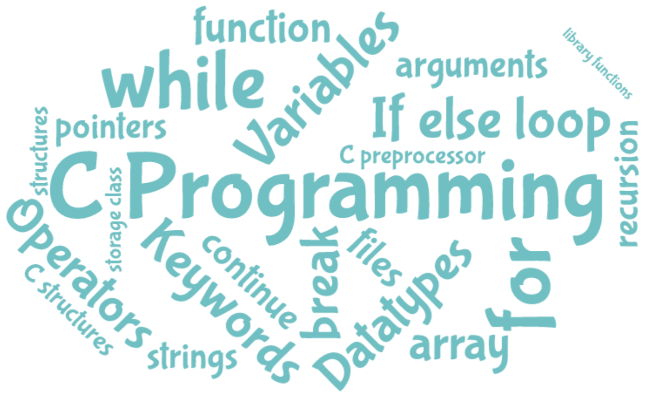
{C}

PROGRAMMING

IT 112: Introduction to Programming



Dr. Manish Khare
Dr. Bakul Gohel

STRING

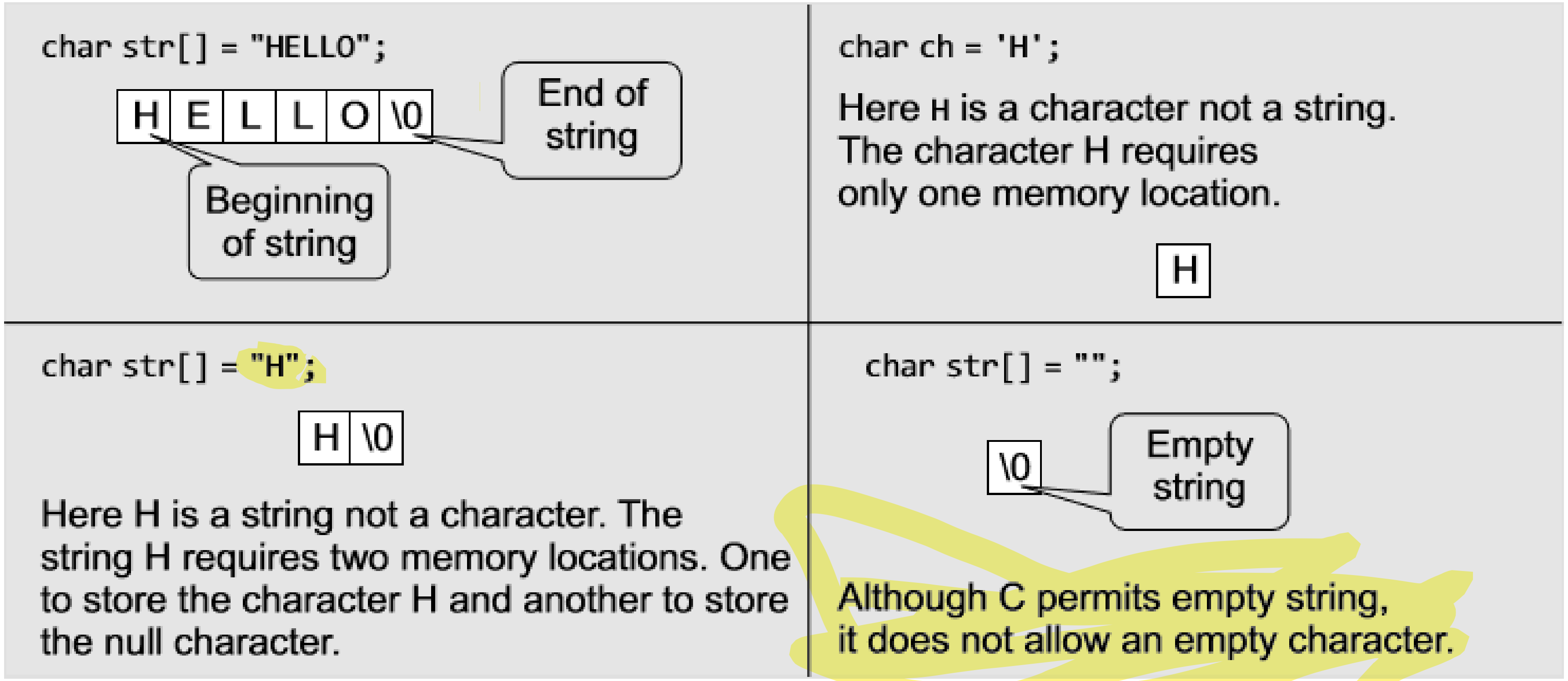
Strings

- Strings are actually one-dimensional array of characters terminated by a **null** character '\0’.
- Thus a null-terminated string contains the characters that comprise the string followed by a **null**.
- char name[10] = {'s', 't', 'r', 'i', 'n', 'g' ,'\0’};
- So we can say that a string is just a one-dimensional array of characters with a null character ('\0') as it's the last element.

- A string literal is just a sequence of characters enclosed in double quotes (""). It is also known as a string constant.

- If you follow the rule of array initialization then you can write the previous statement as follows −
  - char greeting[] = "Hello";

Difference between character storage and string storage



```
char str[] = "HELLO";
```

```
char ch = 'H';
```

Here H is a character not a string. The character H requires only one memory location.

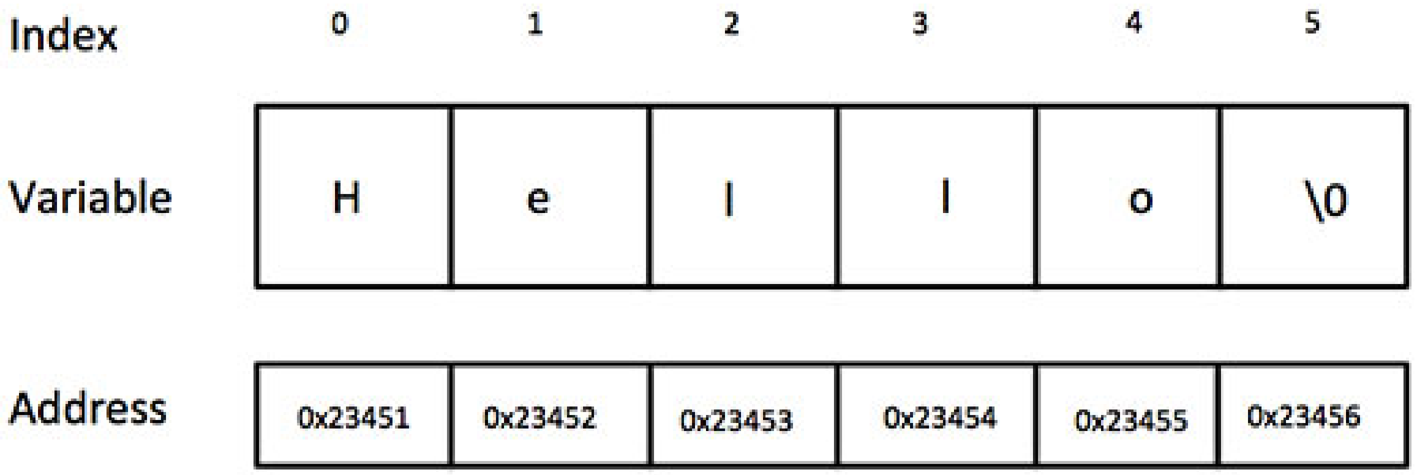
```
char str[] = "H";
```

Here H is a string not a character. The string H requires two memory locations. One to store the character H and another to store the null character.

```
char str[] = "";
```

Although C permits empty string, it does not allow an empty character.

- Following is the memory presentation of the above defined string in C.



Index	0	1	2	3	4	5
Variable	H	e	l	l	o	\0
Address	0x23451	0x23452	0x23453	0x23454	0x23455	0x23456

- Actually, you do not place the *null* character at the end of a string constant. The C compiler automatically places the '\0' at the end of the string when it initializes the array.

- The statement char str[] = "HELLO"; declares a constant string, as we have assigned a value to it while declaring the string.
- However, the general form of declaring a string is

  char str[size];
- When we declare the string like this, we can store size–1 characters in the array because the last character would be the null character.
- For example, char mesg[100]; can store a maximum of 99 characters.

- Reading Strings

If we declare a string by writing

Char str [100];

Then Str can be read by using three ways:

Using scanf function

Using gets() function

Using getchar(), getch() or getche() function repeatedly

- Strings can be read using scanf() by writing

```
scanf("%s", str);
```

$\uparrow$ base address

- Although the syntax of using scanf() function is well known and easy to use, the main pitfall of using this function is that the function terminates as soon as it finds a blank space.
- For example, if the user enters Hello World, then the str will contain only Hello.
- This is because the moment a blank space is encountered, the string is terminated by the scanf() function.

- The next method of reading a string is by using the **gets()** function.
- The string can be read by writing

  gets(str);
- gets() is a simple function that overcomes the drawbacks of the scanf() function.
- The gets() function takes the starting address of the string which will hold the input.
- The string inputted using gets() is automatically terminated with a null character.

- Strings can also be read by calling the **getchar()** function repeatedly to read a sequence of single characters (unless a terminating character is entered) and simultaneously storing it in a character array as shown below.

```
i=0;
ch = getchar;// Get a character
while(ch != '*')
{
        str[i] = ch;// Store the read character in str
        i++;
        ch = getchar();// Get another character
}
str[i] = '\0';// Terminate str with null character
```

- Note that in this method, you have to deliberately append the string with a null character.
- The other two functions automatically do this.

➢ Writing Strings

Strings can be displayed on screen using three ways:

Using printf() function

Using puts() function

Using putchar() function repeatedly

- Strings can be displayed using printf() by writing

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

- We use the format specifier %s to output a string.
- Observe carefully that there is no ‘&’ character used with the string variable.

- A string can be displayed by writing

  ```
  puts(str);
  ```

- Strings can also be written by calling the putchar() function repeatedly to print a sequence of single characters.

```
i=0;
while(str[i] != '\0')
 {
   putchar(str[i]);// Print the character on the screen
   i++;
 }
```

- In C, we can store a string either in fixed length format or in variable length format

- String
  - Fixed Length
  - Variable Length
    - Length Controlled
    - Delimited

Operations on Strings

- Finding the length of a string
- Converting characters of a string into Upper case
- Converting characters of a string into Lower case
- Concatenating two strings to form a new string
- Appending a string to another string
- Comparing two strings
- Reversing a String
- Extracting a substring from left of the string
- Extracting a substring from Right of the string
- Extracting a substring from the middle of the string
- Inserting a string in another string
- Indexing
- Deleting a String from main string
- Replacing a pattern with another pattern in a string

Predefined string functions

- We can perform different kinds of string functions like joining of 2 strings, comparing one string with another or finding the length of the string.

Function	Use
strlen	calculates the length of string
strcat	Appends one string at the end of another
strncat	Appends first n characters of a string at the end of another
strcpy	Copies a string into another
strncpy	Copies first n characters of one string into another
strcmp	Compares two strings
strncmp	Compares first n characters of two strings
strchr	Finds first occurrence of a given character in a string
strrchr	Finds last occurrence of a given character in a string
strstr	Finds first occurrence of a given string in another string

Function	Use
strcspn	returns the length of the initial part of the string s1 **not** containing any of the characters of the string s2.
strspn	returns the length of the initial part of the string s1 only containing the characters of the string s2.
strpbrk	returns the first occurrence of any of the characters of the string s2 in the string s1.
strtok	finds s2 in s1 and returns a pointer to it and returns NULL if not found.

Character Manipulation Function

Sr.No.	Function & Description
1	int isalnum(int c) This function checks whether the passed character is alphanumeric.
2	int isalpha(int c) This function checks whether the passed character is alphabetic.
3	int iscntrl(int c) This function checks whether the passed character is control character.
4	int isdigit(int c) This function checks whether the passed character is decimal digit.
5	int isgraph(int c) This function checks whether the passed character has graphical representation using locale.

6	int islower(int c) This function checks whether the passed character is lowercase letter.
7	int isprint(int c) This function checks whether the passed character is printable.
8	int ispunct(int c) This function checks whether the passed character is a punctuation character.
9	int isspace(int c) This function checks whether the passed character is white-space.
10	int isupper(int c) This function checks whether the passed character is an uppercase letter.

11	int isxdigit(int c) This function checks whether the passed character is a hexadecimal digit.
12	int tolower(int c) This function converts uppercase letters to lowercase.
13	int toupper(int c) This function converts lowercase letters to uppercase.

- These functions are defined in **"string.h"** header file, so we need to include this header file also in our code by writing
  - **#include <string.h>**

strlen

➢ **strlen(s1)** calculates the length of string s1.

```
#include <stdio.h>
#include <string.h>
int main()
{
  char name[ ]= "Hello";
  int len1, len2;
  len1 = strlen(name);
  len2 = strlen("Hello World");
  printf("length of %s = %d\n", name, len1);
  printf("length of %s = %d\n", "Hello World", len2);
  return 0;
}
```

Output

```
length of Hello = 5

length of Hello World =
11
```

strlen doesn't count '\0' while calculating the length of a string.

strcat

- **strcat(s1, s2)** concatenates(joins) the second string s2 to the first string s1.

```
#include <stdio.h>
#include <string.h>
int main()
{
  char s2[ ]= "World";
char s1[20]= "Hello";
strcat(s1, s2);
printf("Source string = %s\n", s2);
printf("Target string = %s\n", s1);
return 0;
}
```

Output

```
Source string = World

Target string =
HelloWorld
```

strncat

- **strncat(s1, s2, n)** concatenates(joins) the first ‘n’ characters of the second string s2 to the first string s1.

```
#include <stdio.h>
#include <string.h>
int main()
{
  char s2[ ]= "World";
 char s1[20]= "Hello";
 strncat(s1, s2, 2);
  printf("Source string = %s\n", s2);
  printf("Target string = %s\n", s1);
  return 0;}
```

Output

```
Source string = World

Target string =
HelloWo
```

strcpy

- **strcpy(s1, s2)** copies the second string s2 to the first string s1.

```
#include <stdio.h>
#include <string.h>
int main()
{
  char s2[ ]= "Hello";
  char s1[10];
  strcpy(s1, s2);
  printf("Source string = %s\n", s2);
  printf("Target string = %s\n", s1);
  return 0;
}
```

Output

```
Source string = Hello

Target string = Hello
```

strncpy

- **strncpy(s1, s2, n)** copies the first ‘n’ characters of the second string s2 to the first string s1.

```
#include <stdio.h>
#include <string.h>
int main()
{
  char s2[ ]= "Hello";
  char s1[10];
  strncpy(s1, s2, 2);
  s1[2] = '\0';   /* null character manually added */
  printf("Source string = %s\n", s2);
  printf("Target string = %s\n", s1);
  return 0;
}
```

Output

```
Source string = Hello

Target string = He
```

strcmp

- **strcmp(s1, s2)** compares two strings and finds out whether they are same or different. It compares the two strings character by character till there is a mismatch. If the two strings are **identical**, it returns a **0**. If not, then it returns the difference between the ASCII values of the first non-matching pair of characters.

```
#include <stdio.h>
#include <string.h>
int main()
{
  char s1[ ]= "Hello";
  char s2[ ]= "World";
  int i, j;
  i = strcmp(s1, "Hello");
  j = strcmp(s1, s2);
  printf("%d \n %d\n", i, j);
  return 0;
}
```

Output

```
0
-15
```

The difference between the ASCII values of H(72) and W(87) is -15.

strncmp

- **strncmp(s1, s2, n)** compares the first ‘n’ characters of s1 and s2.

```
#include <stdio.h>
#include <string.h>
int main()
{
 char s1[ ]= "Hello";
 char s2[ ]= “Heral";
 int i, j;
 i = strncmp(s1, s2, 2);
 printf("%d\n", i);
 return 0;}
```

Output

```
0
```

strspn

- **strspn(s1, s2)** returns the length of the initial part of the string s1 only containing the characters of the string s2.

```
#include <stdio.h>
#include <string.h>
int main()
{
  char s1[ ]= "123abc";
  char s2[ ] = "123456790";
  int i = strspn(s1, s2);
  printf("%d\n", i);
  return 0;
}
```

Output

```
3
```

strrev

- **strrev(s1)** reverses all the characters in the string except the null character.

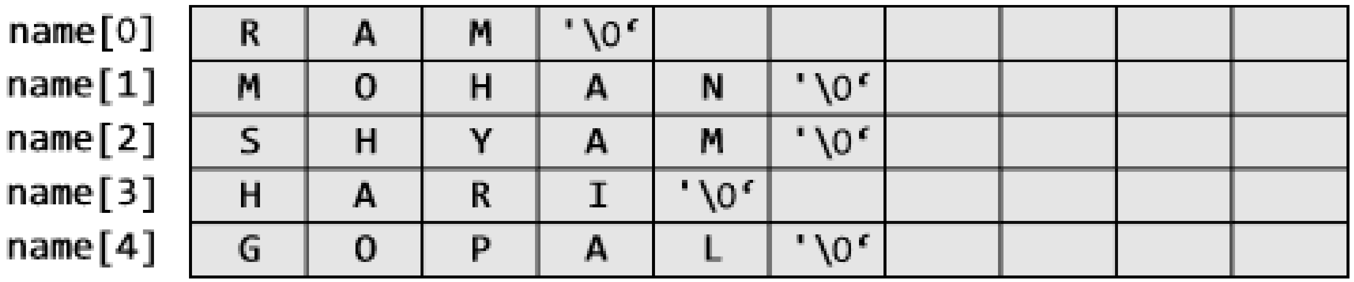
```
#include <stdio.h>
#include <string.h>
int main()
{
  char s1[ ]= "Hello";
 printf("Source string = %s\n", s1);
  strrev (s1);
  printf("Target string = %s\n", s1);

return 0;
}
```

Array of Strings

- Till now we have seen that a string is an array of characters.
- For example, if we say char name[] = "Mohan", then the name is a string (character array) that has five characters.
- Now, suppose that there are 20 students in a class and we need a string that stores the names of all the 20 students.
- How can this be done? Here, we need a string of strings or an array of strings.
- Such an array of strings would store 20 individual strings.
- An array of strings is declared as **char names[20][30];**

- Here, the first index will specify how many strings are needed and the second index will specify the length of every individual string.
- So here, we will allocate space for 20 names where each name can be a maximum 30 characters long.
- Let us see the memory representation of an array of strings.
- If we have an array declared as char name[5][10] = {"Ram", "Mohan", "Shyam", "Hari", "Gopal"};



name[0]	R	A	M	'\0'						
name[1]	M	O	H	A	N	'\0'				
name[2]	S	H	Y	A	M	'\0'				
name[3]	H	A	R	I	'\0'					
name[4]	G	O	P	A	L	'\0'				

- By declaring the array names, we allocate 50 bytes.
- But the actual memory occupied is 27 bytes.
- Thus, we see that about half of the memory allocated is wasted.

Pointers and Strings

- In C, strings are treated as arrays of characters that are terminated with a binary zero character (written as `'\0'`). Consider, for example,

```
    char str[10];
    str[0] = 'H';
    str[1] = 'i';
    str[2] = '!':
    str[3] = '\0';
```

- C provides two alternate ways of declaring and initializing a string. First, you may write

```
     char str[10] = {'H', 'i', '!', '\0'};
```

- But this also takes more typing than is convenient. So, C permits

```
    char str[10] = "Hi!";
```

- When the double quotes are used, a `null` character (`'\0'`) is automatically appended to the end of the string.
- When a string is declared like this, the compiler sets aside a contiguous block of the memory, i.e., 10 bytes long, to hold characters and initializes its first four characters as `Hi!\0`.

- Now, consider the following program that prints a text.

```
#include <stdio.h>
int main()
{
char str[] = "Hello";
char *pstr;
pstr = str;
printf("\n The string is : ");
while(*pstr != '\0')
{
printf("%c", *pstr);
pstr++;
}
return 0;
}
```