Initialization

(i) `char ch[] = "Hello";` - automatically puts '\0' at the end.

(ii) `char arr[] = {'H','e','l','l','o'};` - doesn't put '\0' at the end automatically

(Q) What is the size of the string `char str[] = "PhysicsWallah"` (indices 0 1 2 3 4 5 6 7 8 9 10 11 12)

size = 13 + 1 = 14 ('\0')

(Q) Is the following code correct `char str[13] = "PhysicsWallah"`

(Q) `char str[50] = "Physics Wallah";`

size of arr = 50

size of string = 14

(Q) `char str[] = "Physics Wallah";`

Predict str after:

(a) `str[0] = 'M';` "Mhysics Wallah"

(b) `str[1] = 97;` (97 ↓ ascii) "Ahysics Wallah"

* `arr[i] = *(arr+i) = *(i+arr) = i[arr]`

Input and output of a string without loops

- gets()
- %[^\n]s
- puts()

`char str[] = "College Wallah is the best";`

`puts(str);` // `printf("%s", str);` replacement

↳ automatically puts '\n' at end of string

```
char str[40];
scanf("%s", str);      // darshak
printf("%s", str);     // darshak
```

```
char str[40];
scanf("%s", str);      // Raghav garg   (only the first word of the string is considered.)
printf("%s", str);     // Raghav
```

```
char str[40];
gets(str);
printf("your input was: %s", str);
```

gives correct output.

```
char str[40];
scanf("%[^\n]s", str);
printf("your input was : %s", str);
```

without declaring size we cannot declare an array.

```
int main() {
char str[] = "College Wallah";
char *ptr = str;
int i=0;
while (*ptr != 0)
{
printf("%c", *ptr);
ptr++;
i++;
}
```

↑ iska har ek character modify kr skte hai.

output: "College Wallah";

i = 14

pointer to character

Note: char *ptr = "Physics Wallah";

character pointer can be used to store address of a string

Note: Such direct initialisation using pointers results in a read-only memory allocation of character arrays and hence, causes undefined behaviour when we try to change the characters.

ptr[0] = 'm'; Error!

individual characters

```
char* ptr = "College Wallah";
printf("%s",ptr);
```

Important

using printf("%s",*ptr); output:- C

```
char str[] = "Physics Wallah";
str = "College Wallah";
printf("%s",str);
```

gives error

* we cannot directly modify the whole string but we can modify each character

but

str[0] = p // output physics Wallah

Pointer initialisation: we can modify entire string but not the individual character

In normal initialisation: we cannot modify individual character but not the entire string

```
char *ptr = "College Wallah";
ptr = "Physics Wallah";
printf("%s",ptr);
```

// Physics Wallah

```
ptr = "College Wallah";
printf("%s",ptr);
```

// College Wallah

```
int main(){
    char str[] = "College Wallah";
    char* ptr = str;
    ptr = "Physics Wallah";
    printf("%s",str);
    return 0;
}
```

output: College Wallah

```
int main(){
    char str[] = "College Wallah";
    // char* ptr = str;
    // ptr = "Physics Wallah";
    char* p = str;
    *p = 'P';
    printf("%s",str);
    return 0;
}
```

output: Pollege Wallah

```
char *ptr = "Physics Wallah";
printf("%s \n", ptr);
ptr = "College Wallah";
printf("%s", ptr);
```

Works perfectly!

Physics Wallah ← ptr

then it points to

College Wallah ← ptr

That why correct output

Pointers change the address to which they point after initializing a new character array

Copy one string to another

```
char d1[] = "Physics Wallah";
char *d2 = d1; // shallow copy
d1[0] = 'M';
printf("%s", d2);
```

Output: Mhysics Wallah

M Physics Wallah (S2)
↑
d2

} shallow copy.

Not a deep copy: Here d2 points to the same character array and hence, changes in d1 is also reflected in d2

Deep Copy

```
char str1[] = "Hello";
char str2[] = ↓ copy
```

deep copy

boolean

strlen(char *str)	Returns the length of string
strcpy(char *s1, char *s2)	Copies the contents of string s2 to string s1
strcat(char *s1, char *s2)	Concat s1 string with s2 and stores the result in s1
strcmp(char *s1, char *s2)	Compares the two strings
strncpy(char *s2, char *s1, int len)	Copy substring of size len starting from s1 character pointer into s2.

include <string.h>

```
char d1[] = "Physics Wallah";
char d2[] = d1;   → gives error
char d2[15] = d2 → gives error
```

} but we can do this with pointer

strlen

```
char *str = "Raghav";
int x = strlen(str);
printf("%d", x);
```

actual length of string - 7. (including '\0')

- output - 6 (it ignores the '\0')

```
int y = sizeof(*str);
```

→ 7

ready the '\0'

returns memory in bytes

strcpy

```
char d1[] = "Raghav Garg";
char d2[12];
strcpy(d2, d1);
printf("%s", s2);
```

// Raghav Garg

d2 → destination

d1 → source

deep copy

sizeof d2 should be large enough to copy strings

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

int main()
{
char str1[30]="SARTHAK ";
char* str2="SETHI";
strcat(str1,str2);
printf("%s",str1);
return 0;
}
```

works perfectly

strcat.c > main()

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

int main()
{
char* str1="SARTHAK ";
char* str2="SETHI";
strcat(str1,str2);
printf("%s",str1);
return 0;
}
```

trace trap

strcat

```
char a1[]= "Raghav";
char a2[]= "Garg";
strcat(a1,a2);   → gives error
```

but this should be long enough to store the result of two strings.

```
char a1[11]= "Raghav";
char a2[]= "Garg";
strcat(a1,a2);   → Raghav Garg
```

```
char *a1 = "Raghav"   (cannot change a1. (read only (behaviour))
char *a2= "Garg"
strcat(a1,a2);   → error
```

Inserting a character in string

```
college → college
int main()
{
char str[20]="college";
for(int i=strlen(str) ; i> 2; i--)
{
str[i]= str[i-1];
}
str[2]='l';
```

strcmp - returns 0 if two strings are equal

```c
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)
```

Functions in strings

- strlen (const char* str); -returns the length of the string excluding null character

- strcpy (char *dest, const char *src) - copies one string to another

```c
#include <stdio.h>
#include <string.h>
int main() {
    char src[] = "Hello";
    char dest[10];
    strcpy(dest, src);
    printf("Copied string: %s\n", dest);
    return 0;
}
```

the destination should be of greater or equal length as of source (including null character)

Common Errors:

- Buffer Overflow: Destination is not large enough.

```c
char src[] = "This is too long";
char dest[5];
strcpy(dest, src); // Undefined behavior
```

- strncpy (char *dest, const char *src, size_t n) : copies at most n characters of a string

```c
#include <stdio.h>
#include <string.h>
int main() {
    char src[] = "Hello";
    char dest[10];
    strncpy(dest, src, 3);
    dest[3] = '\0'; // Manually null-terminate
    printf("Partial Copy: %s\n", dest);
    return 0;
}
```

*if src is longer than n then dest is not null terminated automatically and we have to manually terminate the string.

output: Hel

- strcmp (const char *str1, const char*str2) -compares two strings lexicographically

```c
#include <stdio.h>
#include <string.h>
int main() {
    char str1[] = "Hello";
    char str2[] = "World";
    int result = strcmp(str1, str2);
    printf("Comparison Result: %d\n", result); // Negative because 'H' < 'W'
    return 0;
}
```

returns the difference of the characters that are not equal (if the strings are not equal)

here it return H-W = -15

- strcat (char *dest, const char *src) - concatenates two strings

```c
#include <stdio.h>
#include <string.h>
int main() {
    char str1[20] = "Hello, ";
    char str2[] = "World!";
    strcat(str1, str2);
    printf("Concatenated String: %s\n", str1);
    return 0;
}
```

if dest is too small than undefined behaviour is encountered.

null character is at the end of the string

- strncat (char*dest, const char *src, size_t n)

concat at most n characters from src to destination

```c
#include <stdio.h>
#include <string.h>
int main() {
    char str1[20] = "Hello, ";
    char str2[] = "World!";
    strncat(str1, str2, 3);
    printf("Partially Concatenated String: %s\n", str1); // "Hello, Wor"
    return 0;
}
```

if dest lacks null character then behaviour is undefined

it automatically puts null character

- strchr(const char *str, int c)

returns a pointer to that location

ascii code.

finds the first occurrence of character in a string

```c
#include <stdio.h>
#include <string.h>
int main() {
    char str[] = "Hello, World!";
    char *ptr = strchr(str, 'W');
    if (ptr) {
        printf("Found 'W' at position: %ld\n", ptr - str);
    } else {
        printf("'W' not found\n");
    }
    return 0;
}
```

if char not found it returns null pointer

returns 7 indexing starts from 0

- strstr(const char* haystack, const char *needle)

returns the pointer to the first char of substring

Finds the first occurrence of a substring in a string

```c
#include <stdio.h>
#include <string.h>
int main() {
    char str[] = "Hello, World!";
    char *ptr = strstr(str, "World");
    if (ptr) {
        printf("Substring found: %s\n", ptr);
    } else {
        printf("Substring not found\n");
    }
    return 0;
}
```

if the substring is not found then it returns a null pointer.

if it is found returns a pointer to that string

- strtok (char* str, const char *delim) : Tokenizes a string based on delimeters.

```c
#include <stdio.h>
#include <string.h>
int main() {
    char str[] = "Hello,World,Tokenize,Me";
    char *token = strtok(str, ",");
    while (token) {
        printf("Token: %s\n", token);
        token = strtok(NULL, ",");
    }
    return 0;
}
```

cannot use with string literals (we cannot modify string literals)

returns 0 if str1 == str2 str1 < str2 -1 and > 0 if str1 > str2

- strncmp (const char *str1, const char *str2, size_t n) : compares upto n characters lexicographically

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

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

    if (strncmp(str1, str2, 6) == 0) {
        printf("The first 6 characters are identical.\n");
    } else {
        printf("The first 6 characters are different.\n");
    }
    return 0;
}
```

Common error: using n larger than length of either of the strings

it compares until it encounters 'null' termination

- strpbrk (const char *str, const char * accept)

(str: string to be scanned; accept: the set of characters to match)

Finds the first character in a string that matches any character in a set of characters

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

int main() {
    char str[] = "Hello, World!";
    char accept[] = "aeiou"; // Vowels

    char *ptr = strpbrk(str, accept);

    if (ptr) {
        printf("First vowel in '%s': %c\n", str, *ptr);
    } else {
        printf("No vowels found.\n");
    }
    return 0;
}
```

returns null if no character is found

* it returns pointer to the first occurrence.

- strerror (int errcode) : returns a string describing that error code.

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

int main() {
  printf("%s\n", strerror(0));
  printf("%s\n", strerror(1));
  printf("%s\n", strerror(2));
  printf("%s\n", strerror(3));
  return 0;
}
```

Output:
Success
Operation not permitted
No such file or directory
No such process