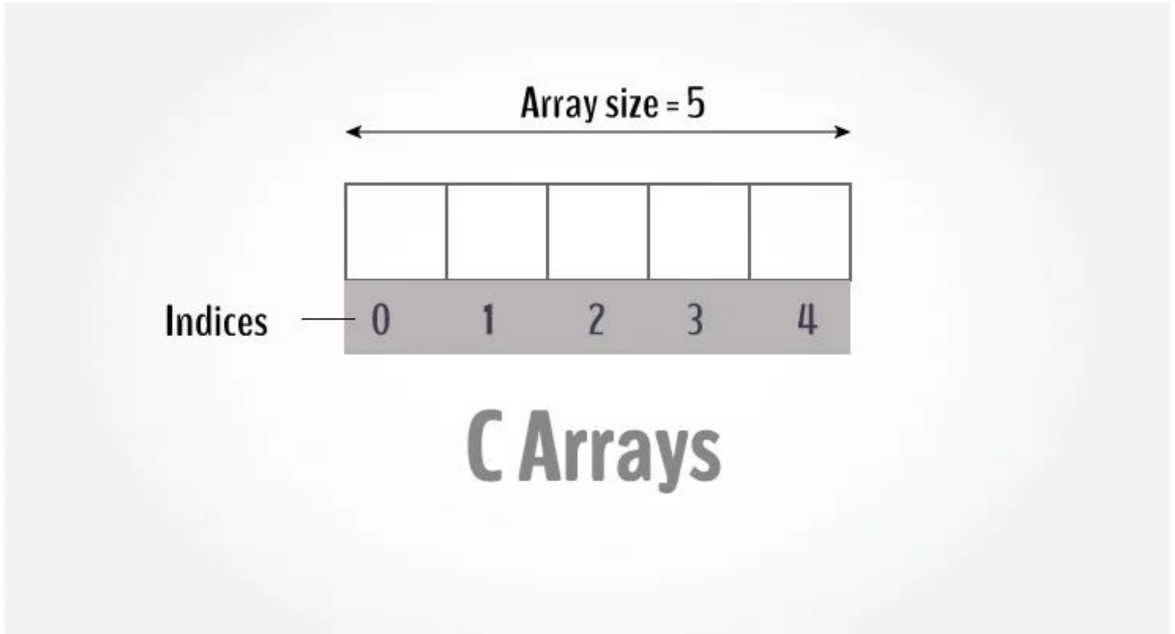
Arrays

What are arrays?

- An array is a variable that can store multiple values.
- For example, if you want to store 100 integers, you can create an array for it.



How to declare an array

dataType arrayName[arraySize];

For eg: int i[20];

Here, we declared an array, i, of integer type. And its size is 20. Meaning, it can hold 20 integer values.

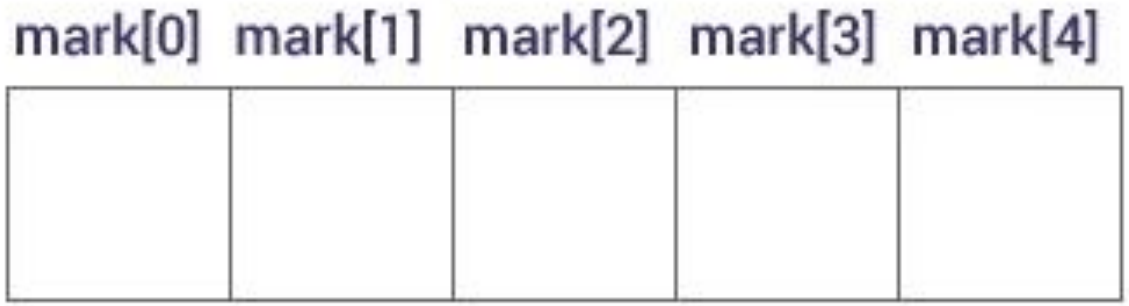
float marks[10];

Here, we declared an array, marks, of floating-point type. And its size is 10. Meaning, it can hold 10 floating-point values.

Access array elements

You can access elements of an array by indices.

Suppose you declared an array mark as above. The first element is mark[0], the second element is mark[1] and so on.



- Arrays have 0 as the first index, not 1. In this example, mark[0] is the first element.
- If the size of an array is n, to access the last element, the n-1 index is used. In this example, mark[4]
- Suppose the starting address of mark[0] is **2120d**. Then, the address of the mark[1] will be **2124d**. Similarly, the address of mark[2] will be **2128d** and so on.

  This is because the size of a float is 4 bytes.

How to initialize an array

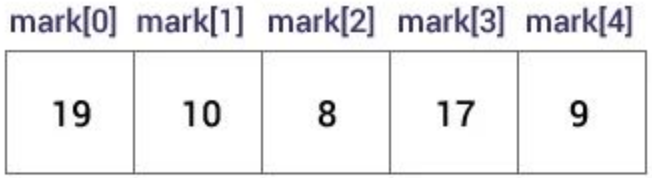
It is possible to initialize an array during declaration. For example,

int mark[5] = {19,10,8,17,9};

You can also initialize an array like this.

int mark[ ] = {19,10,8,17,9};

Here, we haven't specified the size. However, the compiler knows its size is 5 as we are initializing it with 5 elements.



Changing value of array elements

```
int mark[5] = {19, 10, 8,17,9};

// make the value of the third element to 3.

mark[2] = 3;

// make the value of fifth element to 0

mark[4]= 0;
```

Input and Output Array elements

```
// take input and store it in the 3rd element
scanf("%d", &mark[2]);
// take input and store it in the ith element
scanf("%d", &mark[i-1]);
// print the first element of the array
printf("%d", mark[0]);
// print the third element of the array
printf("%d", mark[2]);
// print ith element of the array
printf("%d", mark[i-1]);
```

Problems

- WAP to take 5 values from the user and store them in an array and print the results from the array.
- WAP to find the average of n numbers using array.
- WAP to find the largest among three numbers
- WAP to find largest element in an array.

What is #include <stdio.h>?

a) Preprocessor directive

b) Inclusion directive

c) File inclusion directive

d) None of the mentioned

Which of the following are C preprocessors?

a) #ifdef

b) #define

c) #endif

d) all of the mentioned

What is the sizeof(char) in a 32-bit C compiler?
a) 1 bit
b) 2 bits
c) 1 Byte
d) 2 Bytes

What will happen if the following C code is executed?

```c
#include <stdio.h>
int main()
{
    int main = 3;
    printf("%d", main);
    return 0;
}
```

a) It will cause a compile-time error
b) It will cause a run-time error
c) It will run without any error and prints 3
d) It will experience infinite looping

What will be the output of the following C code?

```c
#include  <stdio.h>
int main()
{
   signed char chr;
   chr = 128;
   printf("%d\n", chr);
   return 0;
}
```

a) 128
b) -128

c) Depends on the compiler

d) None of the mentioned

What is the difference between the following 2 C codes?

```c
#include <stdio.h> //Program 1
int main()
{
    int d, a = 1, b = 2;
    d =  a++ + ++b;
    printf("%d %d %d", d, a, b);
}
#include <stdio.h> //Program 2
int main()
{
    int d, a = 1, b = 2;
    d =  a++ +++b;
    printf("%d %d %d", d, a, b);
}
```

a) No difference as space doesn’t make any difference, values of a, b, d are same in both the case
b) Space does make a difference, values of a, b, d are different
c) Program 1 has syntax error, program 2 is not
d) Program 2 has syntax error, program 1 is not

Problems

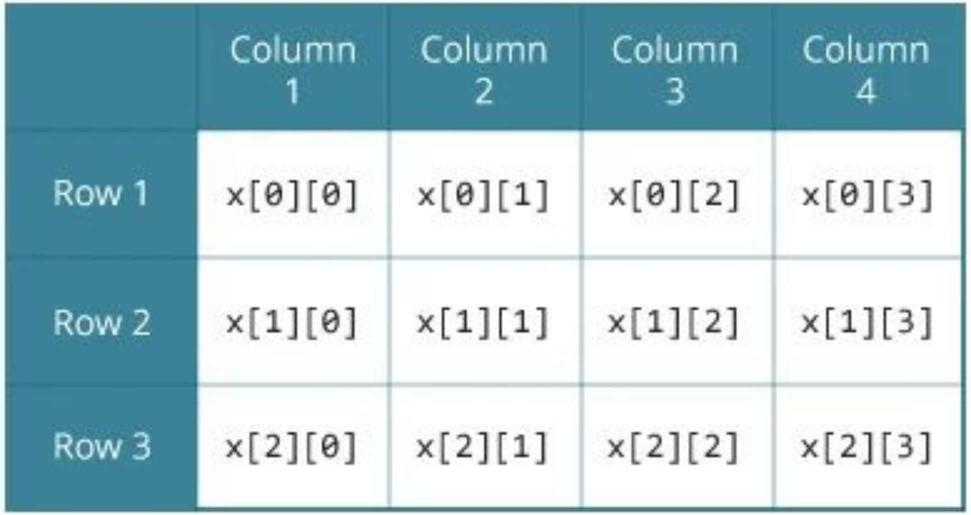
- WAP to input a character in a variable and display it using other variable.
- WAP to check whether two numbers are equal, greater or lesser than each other.
- WAP to print the odd numbers from 1 to 20.
- WAP to check the prime number.
- WAP to calculate the factorial of a number.
- WAP to check whether a number is palindrome or not.
- Print the pattern:

```
1

12

123

1234

12345
```

MultiDimensional Array

In C programming, you can create an array of arrays. These arrays are known as multidimensional arrays. For example, float x[3][4].

Here, x is a two-dimensional (2d) array. The array can hold 12 elements. You can think the array as a table with 3 rows and each row has 4 columns.



	Column 1	Column 2	Column 3	Column 4
Row 1	x[0][0]	x[0][1]	x[0][2]	x[0][3]
Row 2	x[1][0]	x[1][1]	x[1][2]	x[1][3]
Row 3	x[2][0]	x[2][1]	x[2][2]	x[2][3]

Similarly, you can declare a three-dimensional (3d) array. For example,

float c[2][4][3]. Here, the array c can hold 24 elements.

Initializing multi dimensional array

Initialization of a 2d array

```
/

/ Different ways to initialize two-dimensional array


int c[2][3] = {{1, 3, 0}, {-1, 5, 9}};


int c[][3] = {{1, 3, 0}, {-1, 5, 9}};


int c[2][3] = {1, 3, 0, -1, 5, 9};
```

Initializing multi dimensional array

Initialization of a 3d array

```
int test[2][3][4] = {
    {{3, 4, 2, 3}, {0, -3, 9, 11}, {23, 12, 23, 2}},
    {{13, 4, 56, 3}, {5, 9, 3, 5}, {3, 1, 4, 9}}};
```

Addition of multidimensional matrix

```
#include <stdio.h>
int main()
{
int sum[2][2],i,j,r=2,c=2;
int a[2][2]={{1,2},{3,4}};
int b[2][2]={{1,1},{1,1}};
for(i=0;i<r;i++)
for(j=0;j<c;j++)
sum[i][j]=a[i][j]+b[i][j];
printf("Matrix Addition\n");
for(i=0;i<r;i++)
        for(j=0;j<c;j++){
                printf("%d  ",sum[i][j]);
                if(j==c-1){
                        printf("\n");
                        }
        }
return 0;
}
```

Infinite Loop

An infinite loop is a looping construct that does not terminate the loop and executes the loop forever. It is also called an indefinite loop or an endless loop. It either produces a continuous output or no output.

An infinite loop is useful for those applications that accept the user input and generate the output continuously until the user exits from the application manually. In the following situations, this type of loop can be used:

- All the operating systems run in an infinite loop as it does not exist after performing some task. It comes out of an infinite loop only when the user manually shuts down the system.
- All the servers run in an infinite loop as the server responds to all the client requests. It comes out of an indefinite loop only when the administrator shuts down the server manually.
- All the games also run in an infinite loop. The game will accept the user requests until the user exits from the game.

For loop (infinite)

Syntax: for(; ;)

```
#include <stdio.h>
int main()
{
   for(;;)
   {
     printf("Hello World\n");
   }
return 0;
}
```

While loop (infinite)

Syntax: while(1)

```
#include <stdio.h>
int main()
{
  int i=0;
  while(1)
  {
      i++;
      printf("Value of i is :%d\n",i);
  }
return 0;
}
```

Order of precedence with regards to operators in C

- Unary Operators (++,--,....)
- Mathematical Operators (* / % followed by + -)
- Relational Operators (<, <=, >, >=) followed by (== and !=)
- Logical Operators (&& ||)
- Assignment Operator (=)

Questions

- What would happen to X in this expression: X += 15; (assuming the value of X is 5).
- In C language, the variables NAME, name, and Name are all the same. TRUE or FALSE?
- What will be the outcome of the following conditional statement if the value of variable s is 10? s >=10 && s < 25 && s!=12
- What does the format %10.2 mean when included in a printf statement?
- How do you access the values within an array?
- What is the difference between =value and ==value?
- What is the equivalent code of the following statement in WHILE LOOP format?

```
for (a=1; a<=100; a++)

        printf ("%d\n", a * a);
```

-

```
#include <stdio.h>
int main()
{
    char a,b,c;
    int d;
    a='A';
    b='4';
    c=a+b;
    d=a+b;
    printf("C has value = %c\n",c);
    printf("d has value = %d",d);
    return 0;
}
```

ASCII

The full form of ASCII is the **American Standard Code for information interchange**. It is a character encoding scheme used for electronics communication. Each character or a special character is represented by some ASCII code, and each ascii code occupies 7 bits in memory.

In C programming language, a character variable does not contain a character value itself rather the ascii value of the character variable. The ascii value represents the character variable in numbers, and each character variable is assigned with some number range from 0 to 127. For example, the ascii value of 'A' is 65.

In the above example, we assign 'A' to the character variable whose ascii value is 65, so 65 will be stored in the character variable rather than 'A'.

ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

Example

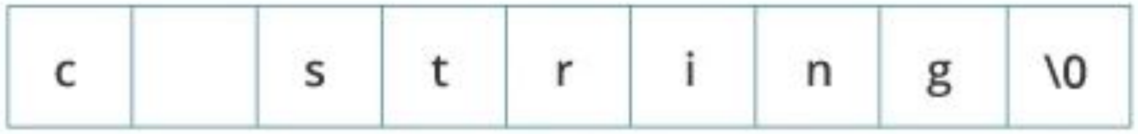
```
#include <stdio.h>
    int main()
    {
    char ch;
    printf("Enter a character:");
    scanf("%c",&ch);  // user input
    printf("\n The ascii value of the ch variable is : %d", ch);
    return 0;
    }
```

Strings

In C programming, a string is a sequence of characters terminated with a null character \0. For example:

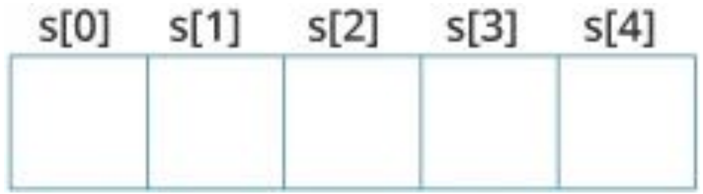
char c[ ] = “c string”

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.



How to declare string?

char s[5];

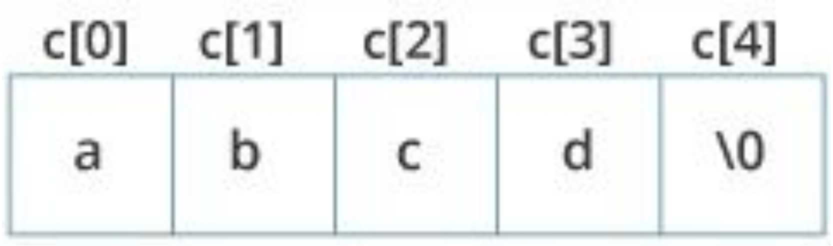


Here, we have declared a string of 5 characters.

How to initialize strings?

You can initialize strings in a number of ways

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


char c[50] = "abcd";


char c[] = {'a', 'b', 'c', 'd', '\0'};


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



c[0]	c[1]	c[2]	c[3]	c[4]
a	b	c	d	\0

Assigning values to strings

```
char c[100];

c = "C programming";  // Error! array type is not assignable.
```

Read String from the user

You can use the scanf() function to read a string.

The scanf() function reads the sequence of characters until it encounters whitespace (space, newline, tab, etc.).

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

Also notice that we have used the code name instead of &name with scanf().

This is because name is a char array, and we know that array names decay to pointers in C.

Thus, the name in scanf() already points to the address of the first element in the string, which is why we don't need to use &

You can use the gets() function to read a line of string. And, you can use puts() to display the string.

```
#include <stdio.h>
int main()
{
    char name[30];
    printf("Enter name: ");
    gets(name);  // read string
    printf("Name: ");
    puts(name);    // display string
    return 0;
}
```

```
#include <stdio.h>
int main()
{
    char name[30];
    printf("Enter name: ");
    fgets(name, sizeof(name), stdin);  // read string
    printf("Name: ");
    puts(name);    // display string
    return 0;
}
```

You need to often manipulate strings according to the need of a problem. Most, if not all, of the time string manipulation can be done manually but, this makes programming complex and large.

To solve this, C supports a large number of string handling functions in the standard library "string.h".

Function	**Work of Function**
strlen()	computes string's length
strcpy()	copies a string to another
strcat()	concatenates(joins) two strings
strcmp()	compares two strings
strlwr()	converts string to lowercase
strupr()	converts string to uppercase

strlen()

```
#include <stdio.h>
#include <string.h>
int main()
{
        char a[20]="Program";
        char b[20]={'P','r','o','g','r','a','m','\0'};
        // using the %zu format specifier to print size_t
        printf("Length of string a = %d \n",strlen(a));
        printf("Length of string b = %d \n",strlen(b));
        return 0;
}
```

strcpy()   Syntax: strcpy(char* destination, const char* source);

```
#include <stdio.h>
#include <string.h>
int main() {
  char str1[20] = "C programming";
  char str2[20];
  // copying str1 to str2
  strcpy(str2, str1);
  puts(str2); // C programming
  return 0;
}
```

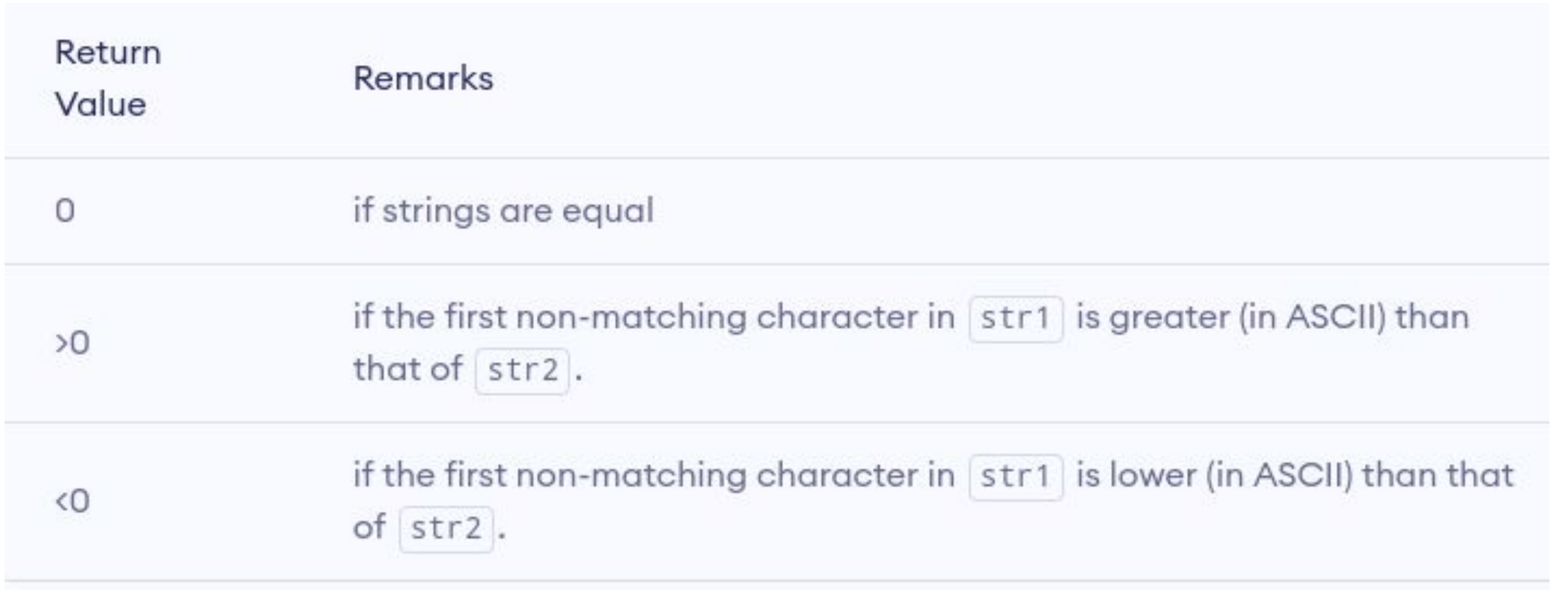
strcat: strcat(char *destination, const char *source)

```
#include <stdio.h>
#include <string.h>
int main() {
   char str1[100] = "This is ", str2[] = "C programming";
   // concatenates str1 and str2
   // the resultant string is stored in str1.
   strcat(str1, str2);
   puts(str1);
   puts(str2);
   return 0;
}
```

strcmp()

The strcmp() compares two strings character by character. If the strings are equal, the function returns 0.

strcmp (const char* str1, const char* str2);



Return Value	Remarks
0	if strings are equal
>0	if the first non-matching character in `str1` is greater (in ASCII) than that of `str2`.
<0	if the first non-matching character in `str1` is lower (in ASCII) than that of `str2`.

```
#include <stdio.h>
#include <string.h>
int main() {
  char str1[] = "abCd", str2[] = "abcd", str3[] = "abcD";
  int result;
  // comparing strings str1 and str2
  result = strcmp(str1, str2);
  printf("strcmp(str1, str2) = %d\n", result);
  // comparing strings str1 and str3
  result = strcmp(str2, str3);
  printf("strcmp(str2, str3) = %d\n", result);
  return 0;
}
```

Read full string after whitespace using scanf

The format specifier `"%[^\n]"` tells to the compiler that read the characters until `"\n"` is not found.

```c
#include <stdio.h>
int main()
{
        char name[30];
        printf("Enter name: ");
        scanf("%[^\n]",name);

        printf("Name is: %s\n",name);
        return 0;
 }
```

WAP to find the frequency of a character

```
#include <stdio.h>
int main() {
        char str[1000], ch;
        int count = 0;

        printf("Enter a string: ");
        fgets(str, sizeof(str), stdin);

        printf("Enter a character to find its frequency: ");
        scanf("%c", &ch);

        for (int i = 0; str[i] != '\0'; ++i) {
        if (ch == str[i])
        ++count;
        }

        printf("Frequency of %c = %d", ch, count);
        return 0;
}
```

C Program to Count the Number of Vowels, Consonants

```
#include <stdio.h>
void main() {
  char line[150];
  int vowels, consonant, digit, space;
  // initialize all variables to 0
  vowels = consonant = digit = space = 0;
  // get full line of string input
  printf("Enter a line of string: ");
  fgets(line, sizeof(line), stdin);
  // loop through each character of the string
  for (int i = 0; line[i] != '\0'; ++i) {
    // convert character to lowercase
    line[i] = tolower(line[i]);
    // check if the character is a vowel
    if (line[i] == 'a' || line[i] == 'e' || line[i] == 'i' ||
        line[i] == 'o' || line[i] == 'u') {
      // increment value of vowels by 1
      ++vowels;
    }
// if it is not a vowel and if it is an alphabet, it is a consonant
    else if ((line[i] >= 'a' && line[i] <= 'z')) {
      ++consonant;
    }

    // check if the character is a digit
    else if (line[i] >= '0' && line[i] <= '9') {
      ++digit;
    }

    // check if the character is an empty space
    else if (line[i] == ' ') {
      ++space;
    }
  }
  printf("Vowels: %d", vowels);
  printf("\nConsonants: %d", consonant);
  printf("\nDigits: %d", digit);
  printf("\nWhite spaces: %d", space);
}
```

Reverse a string

```
#include<stdio.h>

#include<string.h>

int main()

{

   char str[100];

   printf("Enter string: ");

   fgets(str, sizeof(str), stdin);

   strrev(str);

   printf("The reverse of the string is: ");

   puts(str);

   return 0;        }
```

```
#include<stdio.h>
#include<string.h>
int main()
{
   char s1[100], s2[100];
   int i, j,;
   printf("Enter a string: ");
         gets(s1);
   //find String length
   for(i=0; s1[i]!='\0';i++);
         printf("The length of the string is %d\n",i);
         i=strlen(s1)-1;
   //Copying string in reverse order
   for(j=0; i>=0; j++)
   {
         s2[j]=s1[i];
         i–;
   }
   s2[j]='\0';

   printf("The reverse string is:");
   puts(s2);
   return 0;
 }
```

```
#include<stdio.h>
#include<string.h>
main()
{
    int i,j,len;
    char c[]="Programming";
    len=strlen(c);
    for(i=0;i<len;i++)
    {
        for(j=0;j<=i;j++)
        {
            printf("%c",c[j]);
        }
        printf("\n");
    }
}
```