

C programming

basic to advance

Syllabus

1. Learn C Basics First-----	04
2. Datatypes and Variables, Constants-----	08
3. Arithmetic operator-----	17
4. Conditional Statement-----	28
5. switch...case statement-----	65
6. While loops-----	86
7. For loops-----	98
8. C – Arrays-----	160
9. Functions in C-----	204
10. library function-----	238
11. Recursion-----	254
12. Pointers-----	263
13. Structures-----	289
14. Union-----	316
15. Strings-----	325

16. File handling exercises-----	378
17. Memory Management-----	421
18. C Dynamic Memory-----	425
19. C - Header Files-----	430
20. C - Error Handling-----	447
21. C Files I/O-----	451
22. Type Casting in C-----	456

1. Learn C Basics First

➤ History of c programming

- **History of C language** is interesting to know. Here we are going to discuss a brief history of the c language.
- **C programming language** was developed in 1972 by Dennis Ritchie at bell laboratories of AT&T (American Telephone & Telegraph), located in the U.S.A.
- **Dennis Ritchie** is known as the **founder of the c language**.
- It was developed to overcome the problems of previous languages such as B, BCPL, etc.
- Initially, C language was developed to be used in **UNIX operating system**. It inherits many features of previous languages such as B and BCPL.
- Let's see the programming languages that were developed before C language.

Language	Year	Developed By
Algol	1960	International Group
BCPL	1967	Martin Richard
B	1970	Ken Thompson
Traditional C	1972	Dennis Ritchie
K & R C	1978	Kernighan & Dennis Ritchie
ANSI C	1989	ANSI Committee
ANSI/ISO C	1990	ISO Committee
C99	1999	Standardization Committee

➤ Features of C Language

C is the widely used language. It provides many **features** that are given below.

1. Simple
2. Machine Independent or Portable
3. Mid-level programming language
4. structured programming language
5. Rich Library
6. Memory Management
7. Fast Speed
8. Pointers
9. Recursion
10. Extensible

➤ Install compiler or code block

Download code block and install

➤ Header file

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

➤ what is main function

The main () has function definition (the code of a function) but it doesn't have any function declaration. Though we often use int main () or void main ()

➤ Key word in c programming

C KEYWORDS OR RESERVED WORDS

BeginnersBook.com

auto	break	case	char
const	continue	default	do
int	long	register	return
short	signed	sizeof	static
struct	switch	typedef	union
unsigned	void	volatile	while
double	else	enum	extern
float	for	goto	if

➤ What is output

Printf(" ");

➤ C programming structure

[Here is a basic Programming of c](#)

```
#include <stdio.h>
int main()
{
/* This is first program in C */
printf("Hello, World! \n");
return 0;
}
```

➤ Write first c programming

[Here is a basic Programming of c](#)

```
#include <stdio.h>
int main()
{
    /* This is first program in C */
    printf("Hello, World! \n");
    return 0;
}
```

2

2. Datatypes and Variables, Constants

➤ What is data types and list of data types?

A data type constrains the values that an expression, such as a variable or a function, might take. This data type defines the operations that can be done on the data, the meaning of the data, and the way values of that type can be stored.

C Basic Data Types	32-bit CPU		64-bit CPU	
	Size (bytes)	Range	Size (bytes)	Range
char	1	-128 to 127	1	-128 to 127
short	2	-32,768 to 32,767	2	-32,768 to 32,767
int	4	-2,147,483,648 to 2,147,483,647	4	-2,147,483,648 to 2,147,483,647
long	4	-2,147,483,648 to 2,147,483,647	8	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
long long	8	9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	8	9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4	3.4E +/- 38	4	3.4E +/- 38
double	8	1.7E +/- 308	8	1.7E +/- 308

➤ Size of data types

C Basic Data Types	32-bit CPU		64-bit CPU	
	Size (bytes)	Range	Size (bytes)	Range
char	1	-128 to 127	1	-128 to 127
short	2	-32,768 to 32,767	2	-32,768 to 32,767
int	4	-2,147,483,648 to 2,147,483,647	4	-2,147,483,648 to 2,147,483,647
long	4	-2,147,483,648 to 2,147,483,647	8	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
long long	8	9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	8	9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4	3.4E +/- 38	4	3.4E +/- 38
double	8	1.7E +/- 308	8	1.7E +/- 308

➤ What is variables and use to or not use to variables

In mathematics, a variable is a symbol which works as a placeholder for expression or quantities that may vary or change; is often used to represent the argument of a function or an arbitrary element of a set. In addition to numbers, variables are commonly used to represent vectors, matrices and functions.

Any type key words and sepals character and Number not use to variable and other use.

➤ What is constants and use to why?

In computer programming, a constant is a value that should not be altered by the program during normal execution, i.e., the value is constant. When associated with an identifier, a constant is said to be "named," although the terms "constant" and "named constant" are often used interchangeably..

Its not change the value there use.

➤ What is Assign or defiant value

A=10;

B=30;

Its assign or defiant value.

➤ What is user input

Any information or data sent to a computer for processing is considered input.

... Input or user input is sent to a computer using an input device. The picture is an illustration of the difference between input and output. The input example (top) shows data being sent from a keyboard to a computer.

```
scanf(" ");
```

The scanf() function is used for input. It reads the input data from the console.

➤ Try input/output or printf/scanf

The printf() and scanf() functions are used for input and output in C language. Both functions are inbuilt library functions, defined in stdio.h (header file).

printf() function

The printf() function is used for output. It prints the given statement to the console.

scanf() function

The scanf() function is used for input. It reads the input data from the console.

Example 1: C Output

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

```
int main()
```

```
{
```

```
    // Displays the string inside quotations
```

```
    printf("C Programming");
```

```
    return 0;  
}
```

Output

C Programming

Example 2: Integer Output

```
#include <stdio.h>  
  
int main()  
{  
    int testInteger = 5;  
    printf("Number = %d", testInteger);  
    return 0;  
}
```

Output

Number = 5

Example 3: float and double Output

```
#include <stdio.h>  
  
int main()  
{  
    float number1 = 13.5;  
    double number2 = 12.4;  
  
    printf("number1 = %f\n", number1);  
    printf("number2 = %lf", number2);  
    return 0;  
}
```

```
}
```

Output

```
number1 = 13.500000
```

```
number2 = 12.400000
```

Example 4: Print Characters

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

```
int main()
```

```
{
```

```
    char chr = 'a';
```

```
    printf("character = %c", chr);
```

```
    return 0;
```

```
}
```

Output

```
character = a
```

Example 5: Integer Input/Output

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

```
int main()
```

```
{
```

```
    int testInteger;
```

```
    printf("Enter an integer: ");
```

```
    scanf("%d", &testInteger);
```

```
    printf("Number = %d", testInteger);
```

```
    return 0;
```

```
}
```

Output

Enter an integer: 4

Number = 4

Example 6: Float and Double Input/Output

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

```
int main()
```

```
{
```

```
    float num1;
```

```
    double num2;
```

```
    printf("Enter a number: ");
```

```
    scanf("%f", &num1);
```

```
    printf("Enter another number: ");
```

```
    scanf("%lf", &num2);
```

```
    printf("num1 = %f\n", num1);
```

```
    printf("num2 = %lf", num2);
```

```
    return 0;
```

```
}
```

Output

Enter a number: 12.523

Enter another number: 10.2

num1 = 12.523000

```
num2 = 10.200000
```

Example 7: C Character I/O

```
#include <stdio.h>

int main()
{
    char chr;

    printf("Enter a character: ");

    scanf("%c",&chr);

    printf("You entered %c.", chr);

    return 0;
}
```

Output

```
Enter a character: g
```

```
You entered g
```

I/O Multiple Values

Here's how you can take multiple inputs from the user and display them.

```
#include <stdio.h>

int main()
{
    int a;

    float b;

    printf("Enter integer and then a float: ");
```

```
// Taking multiple inputs
scanf("%d%f", &a, &b);

printf("You entered %d and %f", a, b);

return 0;
}
```

Output

Enter integer and then a float: -3

3.4

You entered -3 and 3.400000

➤ Format Specifiers for I/O

Data Type	Format Specifier
int	%d
char	%c
float	%f
double	%lf
short int	%hd
unsigned int	%u

Data Type	Format Specifier
long int	%li
long long int	%lli
unsigned long int	%lu
unsigned long long int	%llu
signed char	%c
unsigned char	%c
long double	%Lf

3. Arithmetic operator

➤ Any type addition

Write a C program to input two numbers from user and calculate their sum. C program to add two numbers and display their sum as output. How to add two numbers in C programming.

Example

Input

Input first number: 20

Input second number: 10

Output

Sum = 30

```
#include <stdio.h>

int main()
{
    int num1, num2, sum;
    printf("Enter first number: ");
    scanf("%d", &num1);
    printf("Enter second number:");
    scanf("%d", &num2);
    sum = num1 + num2;

    printf("Sum of %d and %d = %d", num1, num2, sum);
    return 0;
}
```

➤ Any type Subtraction

Write a C program to input two numbers from user and calculate their subtraction. C program to add two numbers and display their sum as output. How to add two numbers in C programming.

Example

Input

Input first number: 20

Input second number: 10

Output

subtraction = 10

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

```
int main()
```

```
{
```

```
    int num1, num2, subtraction;
```

```
    printf("Enter first number: ");
```

```
    scanf("%d", &num1);
```

```
    printf("Enter second number:");
```

```
    scanf("%d", &num2);
```

```
    subtraction = num1 - num2;
```

```
    printf("subtraction of %d and %d = %d", num1, num2, subtraction);
```

```
    return 0;
```

```
}
```

➤ Any type multiplication

Write a C program to input two numbers from user and calculate their multiplication. C program to add two numbers and display their sum as output. How to add two numbers in C programming.

Example

Input

Input first number: 20

Input second number: 10

Output

multiplication = 200

```
#include <stdio.h>

int main()
{
    int num1, num2, multiplication;
    printf("Enter first number: ");
    scanf("%d", &num1);
    printf("Enter second number:");
    scanf("%d", &num2);
    multiplication = num1 * num2;

    printf("multiplication of %d and %d = %d", num1, num2, multiplication);
    return 0;
}
```

➤ Any type division

Write a C program to input two numbers from user and calculate their division. C program to add two numbers and display their sum as output. How to add two numbers in C programming.

Example

Input

Input first number: 20

Input second number: 10

Output

division = 2

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

```
int main()
```

```
{
```

```
    int num1, num2, division;
```

```
    printf("Enter first number: ");
```

```
    scanf("%d", &num1);
```

```
    printf("Enter second number:");
```

```
    scanf("%d", &num2);
```

```
    division = num1 / num2;
```

```
    printf("division of %d and %d = %d", num1, num2, division);
```

```
    return 0;
```

```
}
```

➤ Any type average

Write a C program to input two numbers from user and calculate their Average. C program to add two numbers and display their sum as output. How to add two numbers in C programming.

Example

Input

Input first number: 20

Input second number: 10

Output

Average = 15

```
#include <stdio.h>

int main()
{
    int num1, num2, sum, Average;
    printf("Enter first number: ");
    scanf("%d", &num1);
    printf("Enter second number:");
    scanf("%d", &num2);
    sum = num1 / num2;
    Average=sum/2;
    printf("Average of %d and %d = %d", num1, num2, Average);
    return 0;
}
```

➤ Any type modulas

Write a C program to input two numbers and perform all arithmetic operations. How to perform all arithmetic operation between two numbers in C programming. C program to find sum, difference, product, quotient and modulus of two given numbers.

Example

Input

First number: 10

Second number: 5

Output

Sum = 15
Difference = 5
Product = 50
Quotient = 2
Modulus = 0

```
#include <stdio.h>
int main()
{
    int num1, num2;
    int sum, sub, mult, mod;
    float div, avg;
    printf("Enter any two numbers: ");
    scanf("%d%d", &num1, &num2);
    sum = num1 + num2;
    sub = num1 - num2;
    mult = num1 * num2;
    div = (float)num1 / num2;
    mod = num1 % num2;

    avg = sum / 2;

    printf("SUM = %d\n", sum);
    printf("DIFFERENCE = %d\n", sub);
    printf("PRODUCT = %d\n", mult);
    printf("QUOTIENT = %f\n", div);
    printf("MODULUS = %d", mod);
    printf("average = %d", avg);
    return 0;
}
```

➤ Any type calculate of Triangle

Write a C program to input side of an equilateral triangle from user and find area of the given triangle. How to find area of an equilateral triangle in C programming. C program to calculate area of an equilateral triangle if its side is given.

Example

Input

Enter side of the equilateral triangle: 10

Output

Area of equilateral triangle = 43.3 sq. units

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

int main()
{
    float side, area;
    printf("Enter side of an equilateral triangle: ");
    scanf("%f", &side);
    area = (sqrt(3) / 4) * (side * side);
    printf("Area of equilateral triangle = %.2f sq. units", area);
    return 0;
}
```

Write a C program to input base and height of a triangle and find area of the given triangle. How to find area of a triangle in C programming. Logic to find area of a triangle in C program.

Example

Input

Enter base of the triangle: 10

Enter height of the triangle: 15

Output

Area of the triangle = 75 sq. units

```
#include <stdio.h>

int main()
{
    float base, height, area;
    printf("Enter base of the triangle: ");
    scanf("%f", &base);
    printf("Enter height of the triangle: ");
    scanf("%f", &height);
    area = (base * height) / 2;
    printf("Area of the triangle = %.2f sq. units", area);
    return 0;
}
```

➤ Any type calculate of circle

Write a C program to input radius of a circle from user and find diameter, circumference and area of the circle. How to calculate diameter, circumference and area of a circle whose radius is given by user in C programming. Logic to find diameter, circumference and area of a circle in C.

Example

Input

Enter radius: 10

Output

Diameter = 20 units

Circumference = 62.79 units

Area = 314 sq. units

```
#include <stdio.h>

int main()
{
    float radius, diameter, circumference, area;
    printf("Enter radius of circle: ");
    scanf("%f", &radius);
```



```

diameter = 2 * radius;
circumference = 2 * 3.14 * radius;
area = 3.14 * (radius * radius);
printf("Diameter of circle = %.2f units \n", diameter);
printf("Circumference of circle = %.2f units \n", circumference);
printf("Area of circle = %.2f sq. units ", area);
return 0;
}

```

➤ Any type calculate of Rectangle

Write a C program to input length and width of a rectangle and calculate perimeter of the rectangle. How to find perimeter of a rectangle in C programming. Logic to find the perimeter of a rectangle if length and width are given in C programming.

Example

Input

Enter length: 5

Enter width: 10

Output

Perimeter of rectangle = 30 units

```

#include <stdio.h>

int main()
{
    float length, width, perimeter;
    printf("Enter length of the rectangle: ");
    scanf("%f", &length);
    printf("Enter width of the rectangle: ");
    scanf("%f", &width);
    perimeter = 2 * (length + width);
    printf("Perimeter of rectangle = %f units ", perimeter);
    return 0;
}

```

Write a C program to input length and width of a rectangle and find area of the given rectangle. How to calculate area of a rectangle in C programming. Logic to find area of a rectangle whose length and width are given in C programming.

Example

Input

Enter length: 5

Enter width: 10

Output

Area of rectangle = 50 sq. units

```
int main()
{
    float length, width, area;
    printf("Enter length of rectangle: ");
    scanf("%f", &length);
    printf("Enter width of rectangle: ");
    scanf("%d", &width);
    area = length * width;
    printf("Area of rectangle = %f sq. units ", area);

    return 0;
}
```

➤ **Calseyas to calculate or farhanayde to calseyass**

Write a program that converts Centigrade to Fahrenheit.

Expected Output :

Input a temperature (in Centigrade): 45

113.000000 degrees Fahrenheit.

```

#include <stdio.h>

int main() {
float temp_f;
float temp_c;

    printf("Input a temperature (in Centigrade): ");
    scanf("%f", &temp_c);
    temp_f = ((9.0 / 5.0) * temp_c) + 32.0;
    printf("%f degrees Fahrenheit.\n", temp_f);
    return(0);
}

```

Write a C program to input temperature in degree Fahrenheit and convert it to degree Centigrade. How to convert temperature from Fahrenheit to Celsius in C programming. C program for temperature conversion. Logic to convert temperature from Fahrenheit to Celsius in C program.

Example

Input

Temperature in fahrenheit = 205

Output

Temperature in celsius = 96.11 C

```

#include <stdio.h>
int main()
{
    float celsius, fahrenheit;
    printf("Enter temperature in Fahrenheit: ");
    scanf("%f", &fahrenheit);
    celsius = (fahrenheit - 32) * 5 / 9;
    printf("%.2f Fahrenheit = %.2f Celsius", fahrenheit, celsius);
    return 0;
}

```

4. Conditional Statement

➤ If statement

Let's see a simple example of C language if statement.

```
#include<stdio.h>

int main(){
    int number=0;
    printf("Enter a number:");
    scanf("%d",&number);
    if(number%2==0){
        printf("%d is even number",number);
    }
    return 0;
}
```

Output

Enter a number:4

4 is even number

Program to find the largest number of the three.

```
#include <stdio.h>

int main()
{
    int a, b, c;
    printf("Enter three numbers?");
```

```
scanf("%d %d %d",&a,&b,&c);  
if(a>b && a>c)  
{  
    printf("%d is largest",a);  
}  
if(b>a && b > c)  
{  
    printf("%d is largest",b);  
}  
if(c>a && c>b)  
{  
    printf("%d is largest",c);  
}  
if(a == b && a == c)  
{  
    printf("All are equal");  
}  
}
```

Output

Enter three numbers?

12 23 34

34 is largest

➤ If-else statement

Let's see the simple example to check whether a number is even or odd using if-else statement in C language.

```
#include<stdio.h>

int main(){
int number=0;
printf("enter a number:");
scanf("%d",&number);
if(number%2==0){
printf("%d is even number",number);
}
else{
printf("%d is odd number",number);
}
return 0;
}
```

Output

enter a number:4

4 is even number

enter a number:5

5 is odd number

Program to check whether a person is eligible to vote or not.

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

```
int main()
{
    int age;
    printf("Enter your age?");
    scanf("%d",&age);
    if(age>=18)
    {
        printf("You are eligible to vote...");
    }
    else
    {
        printf("Sorry ... you can't vote");
    }
}
```

Output

Enter your age?18

You are eligible to vote...

Enter your age?13

Sorry ... you can't vote

➤ Nested If-else statement

The example of an if-else-if statement in C language is given below.

```
#include<stdio.h>

int main(){
int number=0;
printf("enter a number:");
scanf("%d",&number);
if(number==10){
printf("number is equals to 10");
}
else if(number==50){
printf("number is equal to 50");
}
else if(number==100){
printf("number is equal to 100");
}
else{
printf("number is not equal to 10, 50 or 100");
}
return 0;
}
```

Output

enter a number:4

number is not equal to 10, 50 or 100

enter a number:50

number is equal to 50

Program to calculate the grade of the student according to the specified marks.

```
#include <stdio.h>

int main()
{
    int marks;
    printf("Enter your marks?");
    scanf("%d",&marks);
    if(marks > 85 && marks <= 100)
    {
        printf("Congrats ! you scored grade A ...");
    }
    else if (marks > 60 && marks <= 85)
    {
        printf("You scored grade B + ...");
    }
    else if (marks > 40 && marks <= 60)
    {
        printf("You scored grade B ...");
    }
    else if (marks > 30 && marks <= 40)
    {
        printf("You scored grade C ...");
    }
}
```

```

    }
else
{
    printf("Sorry you are fail ...");
}
}

```

Output

Enter your marks?10

Sorry you are fail ...

Enter your marks?40

You scored grade C ...

Enter your marks?90

Congrats ! you scored grade A ...

➤ <, >, <=, >=, ==, !=, &&, ||

Write a C program to accept two integers and check whether they are equal or not.

Test Data : 15 15

Expected Output :

Number1 and Number2 are equal

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

```
void main()
```

```
{
```

```
    int int1, int2;
```

```

printf("Input the values for Number1 and Number2 : ");
scanf("%d %d", &int1, &int2);
if (int1 == int2)
    printf("Number1 and Number2 are equal\n");
else
    printf("Number1 and Number2 are not equal\n");
}

```

Write a C program to check whether a given number is even or odd.

Test Data : 15

Expected Output :

15 is an odd integer

```

#include <stdio.h>

void main()
{
    int num1, rem1;

    printf("Input an integer : ");
    scanf("%d", &num1);
    rem1 = num1 % 2;
    if (rem1 == 0)
        printf("%d is an even integer\n", num1);
    else
        printf("%d is an odd integer\n", num1);
}

```

Write a C program to check whether a given number is positive or negative.

Test Data : 15

Expected Output :

15 is a positive number

```
#include <stdio.h>

void main()
{
    int num;

    printf("Input a number :");
    scanf("%d", &num);
    if (num >= 0)
        printf("%d is a positive number \n", num);
    else
        printf("%d is a negative number \n", num);
}
```

Write a C program to find whether a given year is a leap year or not.

Test Data : 2016

Expected Output :

2016 is a leap year.

```
#include <stdio.h>

void main()
{
    int chk_year;

    printf("Input a year :");
    scanf("%d", &chk_year);
```

```

if ((chk_year % 400) == 0)
    printf("%d is a leap year.\n", chk_year);
else if ((chk_year % 100) == 0)
    printf("%d is a not leap year.\n", chk_year);
else if ((chk_year % 4) == 0)
    printf("%d is a leap year.\n", chk_year);
else
    printf("%d is not a leap year \n", chk_year);
}

```

Exearcise

Write a C program to find maximum between two numbers using if else. C program to input two numbers from user and find maximum between two numbers using if else. How to find maximum or minimum between two numbers using if else in C programming.

Example

Input

Input num1: 10

Input num2: 20

Output

Maximum = 20

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

```
int main()
```

```
{
```

```
    int num1, num2;
```

```
    /* Input two numbers from user */
```

```

printf("Enter two numbers: ");
scanf("%d%d", &num1, &num2);

/* If num1 is maximum */
if(num1 > num2)
{
    printf("%d is maximum", num1);
}

/* If num2 is maximum */
if(num2 > num1)
{
    printf("%d is maximum", num2);
}

/* Additional condition check for equality */
if(num1 == num2)
{
    printf("Both are equal");
}

return 0;
}

```

Write a C program to find maximum between three numbers using ladder if else or nested if. How to find maximum or minimum between three numbers using if else in C programming. Logic to find maximum or minimum between three numbers in C program.

Example

Input

Input num1: 10

Input num2: 20

Input num3: 15

Output

Maximum is: 20

```

#include <stdio.h>

int main()
{
    int num1, num2, num3, max;

    /* Input three numbers from user */
    printf("Enter three numbers: ");
    scanf("%d%d%d", &num1, &num2, &num3);

    if(num1 > num2)
    {
        if(num1 > num3)
        {
            /* If num1 > num2 and num1 > num3 */
            max = num1;
        }
        else
        {
            /* If num1 > num2 but num1 > num3 is not true */
            max = num3;
        }
    }
    else
    {
        if(num2 > num3)
        {
            /* If num1 is not > num2 and num2 > num3 */
            max = num2;
        }
        else
        {
            /* If num1 is not > num2 and num2 > num3 */
            max = num3;
        }
    }
}

```

```

/* Print maximum value */
printf("Maximum among all three numbers = %d", max);

return 0;
}

```

Write a C program to check positive, negative or zero using simple if or if else. C program to input any number from user and check whether the given number is positive, negative or zero. Logic to check negative, positive or zero in C programming.

Example

Input

Input number: 23

Output

23 is positive

```

#include <stdio.h>

int main()
{
    int num;

    /* Input number from user */
    printf("Enter any number: ");
    scanf("%d", &num);

    if(num > 0)
    {
        printf("Number is POSITIVE");
    }
    if(num < 0)
    {
        printf("Number is NEGATIVE");
    }
    if(num == 0)

```



```

{
    printf("Number is ZERO");
}

return 0;
}

```

Write a C program to check whether a number is divisible by 5 and 11 or not using if else. How to check divisibility of any number in C programming. C program to enter any number and check whether it is divisible by 5 and 11 or not. Logic to check divisibility of a number in C program.

Example

Input

Input number: 55

Output

Number is divisible by 5 and 11

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

```

int main()
{
    int num;

```

```

    /* Input number from user */
    printf("Enter any number: ");
    scanf("%d", &num);

```

```

    /*
    * If num modulo division 5 is 0
    * and num modulo division 11 is 0 then
    * the number is divisible by 5 and 11 both
    */
    if((num % 5 == 0) && (num % 11 == 0))
    {
        printf("Number is divisible by 5 and 11");
    }

```

```

    }
    else
    {
        printf("Number is not divisible by 5 and 11");
    }

    return 0;
}

```

Write a C program to check whether a number is even or odd using if else. How to check whether a number is even or odd using if else in C program. C Program to input a number from user and check whether the given number is even or odd. Logic to check even and odd number using if...else in C programming.

Example

Input

Input number: 10

Output

10 is even number

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

```

int main()
{
    int num;

```

```

    /* Input number from user */
    printf("Enter any number to check even or odd: ");
    scanf("%d", &num);

```

```

    /* Check if the number is divisible by 2 then it is even */
    if(num % 2 == 0)
    {
        /* num % 2 is 0 */
        printf("Number is Even.");
    }
    else

```

```

{
    /* num % 2 is 1 */
    printf("Number is Odd.");
}

return 0;
}

```

Write a C program to check leap year using if else. How to check whether a given year is leap year or not in C programming. C Program to input year from user and check whether the given year is leap year or not using ladder if else. Logic to check leap year in C programming.

Example

Input

Input year: 2004

Output

2004 is leap year.

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

```
int main()
{
    int year;
```

```
    /* Input year from user */
    printf("Enter year : ");
    scanf("%d", &year);
```

```
    /*
    * If year is exactly divisible by 4 and year is not divisible by 100
    * or year is exactly divisible by 400 then
    * the year is leap year.
    * Else year is normal year
    */
    if(((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0))
```

```

{
    printf("LEAP YEAR");
}
else
{
    printf("COMMON YEAR");
}

return 0;
}

```

Write a C program to input a character from user and check whether the given character is alphabet or not using if else. How to check whether a character is alphabet or not in C programming. Logic to check if a character is alphabet or not in C program.

Example

Input

Input character: a

Output

'a' is alphabet

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

```
int main()
```

```
{
```

```
    char ch;
```

```
    /* Input a character from user */
```

```
    printf("Enter any character: ");
```

```
    scanf("%c", &ch);
```

```
    if((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z'))
```

```
    {
```

```
        printf("Character is an ALPHABET.");
```

```
    }
```

```
    else
```

```

{
    printf("Character is NOT ALPHABET.");
}

return 0;
}

```

Write a C program to check whether an alphabet is vowel or consonant using if else. How to check vowels and consonants using if else in C programming. C Program to input a character from user and check whether it is vowel or consonant. Logic to check vowel or consonant in C programming.

Example

Input

Input character: a

Output

'a' is vowel

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

```

int main()
{
    char ch;

```

```

    /* Input character from user */
    printf("Enter any character: ");
    scanf("%c", &ch);

```

```

    /* Condition for vowel */
    if(ch=='a' || ch=='e' || ch=='i' || ch=='o' || ch=='u' ||
       ch=='A' || ch=='E' || ch=='I' || ch=='O' || ch=='U')
    {
        printf("%c is Vowel.", ch);
    }
    else if((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z'))
    {

```

```

    /* Condition for consonant */
    printf("%c' is Consonant.", ch);
}
else
{
    /*
     * If it is neither vowel nor consonant
     * then it is not an alphabet.
     */
    printf("%c' is not an alphabet.", ch);
}

return 0;
}

```

Write a C program to input a character from user and check whether given character is alphabet, digit or special character using if else. How to check if a character is alphabet, digits or any other special character using if else in C programming. Logic to check alphabet, digit or special character in C programming.

Example

Input

Input any character: 3

Output

3 is digit

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

```
int main()
{
    char ch;
```

```

    /* Input character from user */
    printf("Enter any character: ");
    scanf("%c", &ch);

```

```

/* Alphabet check */
if((ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z'))
{
    printf("%c' is alphabet.", ch);
}
else if(ch >= '0' && ch <= '9')
{
    printf("%c' is digit.", ch);
}
else
{
    printf("%c' is special character.", ch);
}

return 0;
}

```

Write a C program to input character from user and check whether character is uppercase or lowercase alphabet using if else. How to check uppercase and lowercase using if else in C programming. Logic to check uppercase and lowercase alphabets in C program.

Example

Input

Input character: C

Output

'C' is uppercase alphabet

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

```
int main()
{
    char ch;
```

```
/* Input character from user */
printf("Enter any character: ");
scanf("%c", &ch);
```

```

if(ch >= 'A' && ch <= 'Z')
{
    printf("%c' is uppercase alphabet.", ch);
}
else if(ch >= 'a' && ch <= 'z')
{
    printf("%c' is lowercase alphabet.", ch);
}
else
{
    printf("%c' is not an alphabet.", ch);
}

return 0;
}

```

Write a C program to input week number(1-7) and print the corresponding day of week name using if else. How to print day of week using if else in C programming. Logic to convert week number to day of week in C programming.

Example

Input

Input week number: 1

Output

Monday

```

#include <stdio.h>

```

```

int main()
{
    int week;

```

```

    /* Input week number from user */
    printf("Enter week number (1-7): ");

```



```
scanf("%d", &week);
```

```
if(week == 1)
{
    printf("Monday");
}
else if(week == 2)
{
    printf("Tuesday");
}
else if(week == 3)
{
    printf("Wednesday");
}
else if(week == 4)
{
    printf("Thursday");
}
else if(week == 5)
{
    printf("Friday");
}
else if(week == 6)
{
    printf("Saturday");
}
else if(week == 7)
{
    printf("Sunday");
}
else
{
    printf("Invalid Input! Please enter week number between 1-7.");
}
```

```
return 0;
```

```
}
```

Write a C program to enter month number between(1-12) and print number of days in month using if else. How to print number of days in a given month using if else in C programming. Logic to find number of days in a month in C program.

Example

Input

Enter month number: 1

Output

It contains 31 days.

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

```
int main()
```

```
{
```

```
    int month;
```

```
    /* Input month number from user */
```

```
    printf("Enter month number (1-12): ");
```

```
    scanf("%d", &month);
```

```
    if(month == 1)
```

```
    {
```

```
        printf("31 days");
```

```
    }
```

```
    else if(month == 2)
```

```
    {
```

```
        printf("28 or 29 days");
```

```
    }
```

```
    else if(month == 3)
```

```
    {
```

```
        printf("31 days");
```

```
    }
```

```
    else if(month == 4)
```

```
    {
```

```
    printf("30 days");
}
else if(month == 5)
{
    printf("31 days");
}
else if(month == 6)
{
    printf("30 days");
}
else if(month == 7)
{
    printf("31 days");
}
else if(month == 8)
{
    printf("31 days");
}
else if(month == 9)
{
    printf("30 days");
}
else if(month == 10)
{
    printf("31 days");
}
else if(month == 11)
{
    printf("30 days");
}
else if(month == 12)
{
    printf("31 days");
}
else
{
    printf("Invalid input! Please enter month number between (1-12).");
}
```

```
}  
  
return 0;  
}
```

Write a C program to input amount from user and print minimum number of notes (Rs. 500, 100, 50, 20, 10, 5, 2, 1) required for the amount. How to the minimum number of notes required for the given amount in C programming. Program to find minimum number of notes required for the given denomination. Logic to find minimum number of denomination for a given amount in C program.

Example

Input

Input amount: 575

Output

Total number of notes:

500: 1

100: 0

50: 1

20: 1

10: 0

5: 1

2: 0

1: 0

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

```
int main()
```

```
{
```

```
    int amount;
```

```
    int note500, note100, note50, note20, note10, note5, note2, note1;
```

```
    /* Initialize all notes to 0 */
```

```
    note500 = note100 = note50 = note20 = note10 = note5 = note2 = note1 = 0;
```

```
    /* Input amount from user */
```

```
printf("Enter amount: ");  
scanf("%d", &amount);
```

```
if(amount >= 500)  
{  
    note500 = amount/500;  
    amount -= note500 * 500;  
}  
if(amount >= 100)  
{  
    note100 = amount/100;  
    amount -= note100 * 100;  
}  
if(amount >= 50)  
{  
    note50 = amount/50;  
    amount -= note50 * 50;  
}  
if(amount >= 20)  
{  
    note20 = amount/20;  
    amount -= note20 * 20;  
}  
if(amount >= 10)  
{  
    note10 = amount/10;  
    amount -= note10 * 10;  
}  
if(amount >= 5)  
{  
    note5 = amount/5;  
    amount -= note5 * 5;  
}  
if(amount >= 2)  
{  
    note2 = amount /2;
```

```

        amount -= note2 * 2;
    }
    if(amount >= 1)
    {
        note1 = amount;
    }

    /* Print required notes */
    printf("Total number of notes = \n");
    printf("500 = %d\n", note500);
    printf("100 = %d\n", note100);
    printf("50 = %d\n", note50);
    printf("20 = %d\n", note20);
    printf("10 = %d\n", note10);
    printf("5 = %d\n", note5);
    printf("2 = %d\n", note2);
    printf("1 = %d\n", note1);

    return 0;
}

```

Write a C program to check whether a triangle is valid or not if angles are given using if else. How to check whether a triangle can be formed or not, if its angles are given using if else in C programming. Logic to check triangle validity if angles are given in C program.

Example

Input

Input first angle: 60

Input second angle: 30

Input third angle: 90

Output

The triangle is valid

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

```
int main()
```

```
{
```

```

int angle1, angle2, angle3, sum;

/* Input all three angles of triangle */
printf("Enter three angles of triangle: \n");
scanf("%d%d%d", &angle1, &angle2, &angle3);

/* Calculate sum of angles */
sum = angle1 + angle2 + angle3;

/*
 * If sum of angles is 180 and
 * angle1, angle2, angle3 is not 0 then
 * triangle is valid.
 */
if(sum == 180 && angle1 > 0 && angle2 > 0 && angle3 > 0)
{
    printf("Triangle is valid.");
}
else
{
    printf("Triangle is not valid.");
}

return 0;
}

```

Write a C program to input side of a triangle and check whether triangle is valid or not using if else. How to check whether a triangle can be formed or not if sides of triangle is given using if else in C programming. Logic to check triangle validity if sides are given in C program.

Example

Input

Input first side: 7

Input second side: 10

Input third side: 5

Output

Triangle is valid

```
#include <stdio.h>

int main()
{
    int side1, side2, side3;

    /* Input three sides of a triangle */
    printf("Enter three sides of triangle: \n");
    scanf("%d%d%d", &side1, &side2, &side3);

    if((side1 + side2) > side3)
    {
        if((side2 + side3) > side1)
        {
            if((side1 + side3) > side2)
            {
                /*
                 * If side1 + side2 > side3 and
                 *   side2 + side3 > side1 and
                 *   side1 + side3 > side2 then
                 * the triangle is valid.
                 */
                printf("Triangle is valid.");
            }
            else
            {
                printf("Triangle is not valid.");
            }
        }
        else
        {
            printf("Triangle is not valid.");
        }
    }
    else
    {
        printf("Triangle is not valid.");
    }
}
```



```
    printf("Triangle is not valid.");  
}
```

```
    return 0;  
}
```

Write a C program to input sides of a triangle and check whether a triangle is equilateral, scalene or isosceles triangle using if else. How to check whether a triangle is equilateral, scalene or isosceles triangle in C programming. Logic to classify triangles as equilateral, scalene or isosceles triangle if sides are given in C program.

Example

Input

Input first side: 30

Input second side: 30

Input third side: 30

Output

Triangle is equilateral triangle

```
#include <stdio.h>  
int main()  
{  
    int side1, side2, side3;  
    /* Input sides of a triangle */  
    printf("Enter three sides of triangle: ");  
    scanf("%d%d%d", &side1, &side2, &side3);  
    if(side1==side2 && side2==side3)  
    {  
        /* If all sides are equal */  
        printf("Equilateral triangle.");  
    }  
    else if(side1==side2 || side1==side3 || side2==side3)  
    {  
        /* If any two sides are equal */  
        printf("Isosceles triangle.");  
    }  
    else
```

```

{
    /* If none sides are equal */
    printf("Scalene triangle.");
}
return 0;
}

```

Write a C program to find all roots of a quadratic equation using if else. How to find all roots of a quadratic equation using if else in C programming. Logic to find roots of quadratic equation in C programming.

Example

Input

Input a: 8

Input b: -4

Input c: -2

Output

Root1: 0.80

Root2: -0.30

```

#include <stdio.h>
#include <math.h> /* Used for sqrt() */
int main()
{
    float a, b, c;
    float root1, root2, imaginary;
    float discriminant;
    printf("Enter values of a, b, c of quadratic equation (aX^2 + bX + c): ");
    scanf("%f%f%f", &a, &b, &c);

    /* Find discriminant of the equation */
    discriminant = (b * b) - (4 * a * c);
    /* Find the nature of discriminant */
    if(discriminant > 0)
    {
        root1 = (-b + sqrt(discriminant)) / (2*a);
        root2 = (-b - sqrt(discriminant)) / (2*a);
    }
}

```

```

    printf("Two distinct and real roots exists: %.2f and %.2f", root1, root2);
}
else if(discriminant == 0)
{
    root1 = root2 = -b / (2 * a);
    printf("Two equal and real roots exists: %.2f and %.2f", root1, root2);
}
else if(discriminant < 0)
{
    root1 = root2 = -b / (2 * a);
    imaginary = sqrt(-discriminant) / (2 * a);

    printf("Two distinct complex roots exists: %.2f + i%.2f and %.2f - i%.2f",
        root1, imaginary, root2, imaginary);
}

return 0;
}

```

Write a C program to input cost price and selling price of a product and check profit or loss. Also calculate total profit or loss using if else. How to calculate profit or loss on any product using if else in C programming. Program to calculate profit and loss of any product in C.

Example

Input

Input cost price: 1000

Input selling price: 1500

Output

Profit: 500

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

```
int main()
```

```
{
```

```
    int cp,sp, amt;
```

```

/* Input cost price and selling price of a product */
printf("Enter cost price: ");
scanf("%d", &cp);
printf("Enter selling price: ");
scanf("%d", &sp);

if(sp > cp)
{
    /* Calculate Profit */
    amt = sp - cp;
    printf("Profit = %d", amt);
}
else if(cp > sp)
{
    /* Calculate Loss */
    amt = cp - sp;
    printf("Loss = %d", amt);
}
else
{
    /* Neither profit nor loss */
    printf("No Profit No Loss.");
}

return 0;
}

```

Write a C program to input marks of five subjects Physics, Chemistry, Biology, Mathematics and Computer, calculate percentage and grade according to given conditions:

If	percentage	>=	90%	:	Grade	A
If	percentage	>=	80%	:	Grade	B
If	percentage	>=	70%	:	Grade	C
If	percentage	>=	60%	:	Grade	D
If	percentage	>=	40%	:	Grade	E

If percentage < 40% : Grade F

Example

Input

Input marks of five subjects: 95

95

97

98

90

Output

Percentage = 95.00

Grade A

```
#include <stdio.h>
int main()
{
    int phy, chem, bio, math, comp;
    float per;
    /* Input marks of five subjects from user */
    printf("Enter five subjects marks: ");
    scanf("%d%d%d%d%d", &phy, &chem, &bio, &math, &comp);
    /* Calculate percentage */
    per = (phy + chem + bio + math + comp) / 5.0;

    printf("Percentage = %.2f\n", per);

    /* Find grade according to the percentage */
    if(per >= 90)
    {
        printf("Grade A");
    }
    else if(per >= 80)
    {
        printf("Grade B");
    }
    else if(per >= 70)
    {
        printf("Grade C");
    }
}
```

```

else if(per >= 60)
{
    printf("Grade D");
}
else if(per >= 40)
{
    printf("Grade E");
}
else
{
    printf("Grade F");
}

```

```

return 0;
}

```

Write a C program to input basic salary of an employee and calculate gross salary according to given conditions.

Basic Salary	<=	10000	:	HRA	=	20%,	DA	=	80%
Basic Salary is between		10001 to 20000	:	HRA	=	25%,	DA	=	90%
Basic Salary	>=	20001	:	HRA	=	30%,	DA	=	95%

How to calculate gross salary of an employee using if else in C programming. Program to calculate gross salary of an employee using if else in C program. Logic to find gross salary of employee in C program.

Example

Input

Input basic salary of an employee: 22000

Output

Gross salary = 44000

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

```
int main()
```

```
{
```

```
    float basic, gross, da, hra;
```

```

/* Input basic salary of employee */
printf("Enter basic salary of an employee: ");
scanf("%f", &basic);

```

```

/* Calculate D.A and H.R.A according to specified conditions */
if(basic <= 10000)
{
    da = basic * 0.8;
    hra = basic * 0.2;
}
else if(basic <= 20000)
{
    da = basic * 0.9;
    hra = basic * 0.25;
}
else
{
    da = basic * 0.95;
    hra = basic * 0.3;
}

```

```

/* Calculate gross salary */
gross = basic + hra + da;

```

```

printf("GROSS SALARY OF EMPLOYEE = %.2f", gross);

```

```

return 0;

```

Write a C program to input electricity unit charge and calculate the total electricity bill according to the given condition:

For	first	50	units	Rs.	0.50/unit
For	next	100	units	Rs.	0.75/unit
For	next	100	units	Rs.	1.20/unit
For	unit	above	250	Rs.	1.50/unit

An additional surcharge of 20% is added to the bill.

How to calculate electricity bill using if else in C programming. Program to find electricity bill using if else in C. Logic to find net electricity bill in C program.

```
#include <stdio.h>

int main()
{
    int unit;
    float amt, total_amt, sur_charge;
    /* Input unit consumed from user */
    printf("Enter total units consumed: ");
    scanf("%d", &unit);
    /* Calculate electricity bill according to given conditions */
    if(unit <= 50)
    {
        amt = unit * 0.50;
    }
    else if(unit <= 150)
    {
        amt = 25 + ((unit-50) * 0.75);
    }
    else if(unit <= 250)
    {
        amt = 100 + ((unit-150) * 1.20);
    }
    else
    {
        amt = 220 + ((unit-250) * 1.50);
    }
    /*
    * Calculate total electricity bill
    * after adding surcharge
    */
    sur_charge = amt * 0.20;
    total_amt = amt + sur_charge;
    printf("Electricity Bill = Rs. %.2f", total_amt);
    return 0;
}
```


5. switch...case statement

➤ Switch case breck

C Switch statement is fall-through

In C language, the switch statement is fall through; it means if you don't use a break statement in the switch case, all the cases after the matching case will be executed.

Let's try to understand the fall through state of switch statement by the example given below.

```
#include<stdio.h>

int main(){
int number=0;

printf("enter a number:");
scanf("%d",&number);

switch(number){
case 10:
printf("number is equal to 10\n");
case 50:
printf("number is equal to 50\n");
case 100:
printf("number is equal to 100\n");
default:
printf("number is not equal to 10, 50 or 100");
}
```

```
return 0;
```

```
}
```

Output

enter a number:10

number is equal to 10

➤ Nested switch case statement

We can use as many switch statement as we want inside a switch statement. Such type of statements is called nested switch case statements. Consider the following example.

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

```
int main () {
```

```
    int i = 10;
```

```
    int j = 20;
```

```
    switch(i) {
```

```
        case 10:
```

```
            printf("the value of i evaluated in outer switch: %d\n",i);
```

```
        case 20:
```

```
            switch(j) {
```

```
                case 20:
```

```
                    printf("The value of j evaluated in nested switch: %d\n",j);
```

```
            }
```

```

    }

    printf("Exact value of i is : %d\n", i );
    printf("Exact value of j is : %d\n", j );

    return 0;
}

```

Output

the value of i evaluated in outer switch: 10

The value of j evaluated in nested switch: 20

Exact value of i is : 10

Exact value of j is : 20

➤ Switch case default

Functioning of switch case statement

First, the integer expression specified in the switch statement is evaluated. This value is then matched one by one with the constant values given in the different cases. If a match is found, then all the statements specified in that case are executed along with the all the cases present after that case including the default statement. No two cases can have similar values. If the matched case contains a break statement, then all the cases present after that will be skipped, and the control comes out of the switch. Otherwise, all the cases following the matched case will be executed.

Let's see a simple example of c language switch statement.

```

#include<stdio.h>

int main(){
    int number=0;

```

```
printf("enter a number:");
scanf("%d",&number);
switch(number){
case 10:
printf("number is equals to 10");
break;
case 50:
printf("number is equal to 50");
break;
case 100:
printf("number is equal to 100");
break;
default:
printf("number is not equal to 10, 50 or 100");
}
return 0;
}
```

Output

enter a number:4

number is not equal to 10, 50 or 100

Switch case example 2

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

```
int main()
```

```
{
```

```

int x = 10, y = 5;
switch(x>y && x+y>0)
{
    case 1:
        printf("hi");
        break;
    case 0:
        printf("bye");
        break;
    default:
        printf(" Hello bye ");
}
}

```

Output

hi

Write a C program to input week number(1-7) and print day of week name using switch case. C program to find week day name using switch case. How to find day name of week using switch case in C programming.

Example

Input

Input week number(1-7): 2

Output

Tuesday

```
#include <stdio.h>

int main()
{
    int week;

    /* Input week number from user */
    printf("Enter week number(1-7): ");
    scanf("%d", &week);

    switch(week)
    {
        case 1:
            printf("Monday");
            break;
        case 2:
            printf("Tuesday");
            break;
        case 3:
            printf("Wednesday");
            break;
        case 4:
            printf("Thursday");
            break;
```

```

    case 5:
        printf("Friday");
        break;
    case 6:
        printf("Saturday");
        break;
    case 7:
        printf("Sunday");
        break;
    default:
        printf("Invalid input! Please enter week number between 1-7.");
}

return 0;
}

```

Write a C program to input month number and print total number of days in month using switch...case. C program to find total number of days in a month using switch...case. Logic to print number of days in a month using switch...case in C programming.

Example

Input

Input month number: 3

Output

Total number of days = 31

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

```
int main()
{
    int month;

    /* Input month number from user */
    printf("Enter month number(1-12): ");
    scanf("%d", &month);

    switch(month)
    {
        case 1:
            printf("31 days");
            break;
        case 2:
            printf("28/29 days");
            break;
        case 3:
            printf("31 days");
            break;
        case 4:
            printf("30 days");
            break;
        case 5:
```



```
printf("31 days");
```

```
break;
```

```
case 6:
```

```
printf("30 days");
```

```
break;
```

```
case 7:
```

```
printf("31 days");
```

```
break;
```

```
case 8:
```

```
printf("31 days");
```

```
break;
```

```
case 9:
```

```
printf("30 days");
```

```
break;
```

```
case 10:
```

```
printf("31 days");
```

```
break;
```

```
case 11:
```

```
printf("30 days");
```

```
break;
```

```
case 12:
```

```
printf("31 days");
```

```
break;
```

```
default:
```

```
printf("Invalid input! Please enter month number between 1-12");
```

```
}
```

```
return 0;
```

```
}
```

Write a C program to input an alphabet and check whether it is vowel or consonant using switch case. C program to check vowel or consonant using switch case. Logic to check vowel or consonant using switch case.

Example

Input

Input alphabet: c

Output

'c' is consonant

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

```
int main()
```

```
{
```

```
char ch;
```

```
/* Input an alphabet from user */
```

```
printf("Enter any alphabet: ");
```

```
scanf("%c", &ch);
```

```
/* Switch value of ch */
```

```
switch(ch)
{
    case 'a':
        printf("Vowel");
        break;
    case 'e':
        printf("Vowel");
        break;
    case 'i':
        printf("Vowel");
        break;
    case 'o':
        printf("Vowel");
        break;
    case 'u':
        printf("Vowel");
        break;
    case 'A':
        printf("Vowel");
        break;
    case 'E':
        printf("Vowel");
        break;
    case 'I':
```

```

    printf("Vowel");
    break;
case 'O':
    printf("Vowel");
    break;
case 'U':
    printf("Vowel");
    break;
default:
    printf("Consonant");
}

return 0;
}

```

Write a C program to input two numbers from user and find maximum between two numbers using switch case. How to find maximum or minimum between two numbers using switch case. Logic to find maximum between two numbers using switch case in C programming.

Example

Input

Input first number: 12

Input second number: 40

Output

Maximum: 40

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

```

int main()
{
    int num1, num2;

    /* Input two numbers from user */
    printf("Enter two numbers to find maximum: ");
    scanf("%d%d", &num1, &num2);

    /* Expression (num1 > num2) will return either 0 or 1 */
    switch(num1 > num2)
    {
        /* If condition (num1>num2) is false */
        case 0:
            printf("%d is maximum", num2);
            break;

        /* If condition (num1>num2) is true */
        case 1:
            printf("%d is maximum", num1);
            break;
    }

    return 0;
}

```

Write a C program to input number from user and check whether the number is even or odd using switch case. How to check even or odd using switch case in C programming. Logic to check whether a number is even or odd using switch case in C program.

Example

Input

Input number: 12

Output

Even number

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

```
int main()
```

```
{
```

```
    int num;
```

```
    /* Input a number from user */
```

```
    printf("Enter any number to check even or odd: ");
```

```
    scanf("%d", &num);
```

```
    switch(num % 2)
```

```
    {
```

```
        /* If n%2 == 0 */
```

```
        case 0:
```

```
            printf("Number is Even");
```

```
            break;
```

```

    /* Else if n%2 == 1 */
    case 1:
        printf("Number is Odd");
        break;
}

return 0;
}

```

Write a C program to input a number and check positive negative or zero using switch case. Checking negative, positive or zero using switch case is little tricky. In this example, I will explain how to check positive negative or zero using switch case. However, it is not recommended way, it's just for learning.

Example

Input

Input number: 23

Output

23 is positive

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

```
int main()
```

```
{
```

```
    int num;
```

```
    printf("Enter any number: ");
```

```
    scanf("%d", &num);
```

```

switch (num > 0)
{
    // Num is positive
    case 1:
        printf("%d is positive.", num);
        break;

    // Num is either negative or zero
    case 0:
        switch (num < 0)
        {
            case 1:
                printf("%d is negative.", num);
                break;
            case 0:
                printf("%d is zero.", num);
                break;
        }
        break;
    }
}

return 0;
}

```


Write a C program to find all roots of a Quadratic equation using switch case. How to find all roots of a quadratic equation using switch case in C programming. Logic to calculate roots of quadratic equation in C program.

Example

Input

Input a: 4

Input b: -2

Input c: -10

Output

Root1: 1.85

Root2: -1.35

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

```
#include <math.h> /* Used for sqrt() */
```

```
int main()
```

```
{
```

```
    float a, b, c;
```

```
    float root1, root2, imaginary;
```

```
    float discriminant;
```

```
    printf("Enter values of a, b, c of quadratic equation (aX^2 + bX + c): ");
```

```
    scanf("%f%f%f", &a, &b, &c);
```

```
    /* Calculate discriminant */
```

```
    discriminant = (b * b) - (4 * a * c);
```

```

/* Compute roots of quadratic equation based on the nature of discriminant */
switch(discriminant > 0)
{
    case 1:
        /* If discriminant is positive */
        root1 = (-b + sqrt(discriminant)) / (2 * a);
        root2 = (-b - sqrt(discriminant)) / (2 * a);

        printf("Two distinct and real roots exists: %.2f and %.2f",
            root1, root2);

        break;
    case 0:
        /* If discriminant is not positive */
        switch(discriminant < 0)
        {
            case 1:
                /* If discriminant is negative */
                root1 = root2 = -b / (2 * a);
                imaginary = sqrt(-discriminant) / (2 * a);

                printf("Two distinct complex roots exists: %.2f + i%.2f and %.2f - i%.2f",
                    root1, imaginary, root2, imaginary);

```

```

        break;

    case 0:
        /* If discriminant is zero */
        root1 = root2 = -b / (2 * a);

        printf("Two equal and real roots exists: %.2f and %.2f", root1, root2);

        break;
    }
}

return 0;
}

```

Write a C program to create menu driven calculator that performs basic arithmetic operations (add, subtract, multiply and divide) using switch case and functions. The calculator should input two numbers and an operator from user. It should perform operation according to the operator entered and must take input in given format.

<number 1> <operator> <number 2>

Example

Input

5.2 - 3

Output

2.2

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

```

int main()
{
    char op;

    float num1, num2, result=0.0f;


    /* Print welcome message */
    printf("WELCOME TO SIMPLE CALCULATOR\n");
    printf("-----\n");
    printf("Enter [number 1] [+ - * /] [number 2]\n");


    /* Input two number and operator from user */
    scanf("%f %c %f", &num1, &op, &num2);


    /* Switch the value and perform action based on operator*/
    switch(op)
    {
        case '+':
            result = num1 + num2;
            break;


        case '-':
            result = num1 - num2;
            break;
    }
}

```

```
    case '*':  
        result = num1 * num2;  
        break;  
    case '/':  
        result = num1 / num2;  
        break;  
  
    default:  
        printf("Invalid operator");  
    }  
  
    /* Prints the result */  
    printf("%.2f %c %.2f = %.2f", num1, op, num2, result);  
  
    return 0;  
}
```

6. While loops

➤ While loop

Example of the while loop in C language

Let's see the simple program of while loop that prints table of 1.

```
#include<stdio.h>

int main(){
    int i=1;
    while(i<=10){
        printf("%d \n",i);
        i++;
    }
    return 0;
}
```

Output

```
1
2
3
4
5
6
7
8
9
```

10

Program to print table for the given number using while loop in C

```
#include<stdio.h>

int main(){
int i=1,number=0,b=9;
printf("Enter a number: ");
scanf("%d",&number);
while(i<=10){
printf("%d \n",(number*i));
i++;
}
return 0;
}
```

Output

Enter a number: 50

50

100

150

200

250

300

350

400

450

500

Write a C program to print all natural numbers in reverse from n to 1 using while loop.
C program for natural natural numbers in reverse. How to print natural numbers from n to 1 using while loop in C programming.

Example

Input

Input n: 10

Output

Natural numbers from 10 to 1: 10, 9, 8, 7, 6, 5, 4, 3, 2, 1

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

```
int main()
```

```
{
```

```
    int n;
```

```
    /*
```

```
     * Input a number from user
```

```
    */
```

```
    printf("Enter value of n: ");
```

```
    scanf("%d", &n);
```

```
    while(n>=1)
```

```
    {
```



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

```
n--;
```

```
}
```

```
return 0;
```

```
}
```

Write a C program to print all alphabets from a to z using while loop. How to display alphabets from a to z using while loop in C programming.

Example

Input

Output

Alphabets: a, b, c, d, e, ... , z

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

```
int main()
```

```
{
```

```
char ch = 'a';
```

```
printf("Alphabets from a - z are: \n");
```

```
while(ch<='z')
```

```
{
```

```
printf("%c\n", ch);
```

```
ch++;
```

```
}
```

```

    return 0;
}
Or
#include <stdio.h>

int main()
{
    int ch = 97;

    printf("Alphabets from a - z are: \n");
    while(ch<=122)
    {
        printf("%c\n", ch);
        ch++;
    }

    return 0;
}

```

Write a C program to enter any number from user and print all even numbers between 1 to n using while loop. C program to display even number in a given range using while loop. How to generate even numbers from 1 to n using while loop in C programming.

Example

Input

Input value of n: 10

Output

Even numbers between 1 to 10: 2, 4, 6, 8, 10

```
#include <stdio.h>

int main()
{
    int i, n;

    // Input upper limit of even number from user
    printf("Print all even numbers till: ");
    scanf("%d", &n);

    printf("All even numbers from 1 to %d are: \n", n);

    /*
     * Starts loop counter from 1, increments by 1 till i<=n
     */
    i=1;
    while(i<=n)
    {
        /* Check even condition before printing */
        if(i%2==0)
        {
            printf("%d\n", i);
```

```
}
```

```
i++;
```

```
}
```

```
return 0;
```

```
}
```

➤ Do while loop

There is given the simple program of c language do while loop where we are printing the table of 1.

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

```
int main(){
```

```
int i=1;
```

```
do{
```

```
printf("%d \n",i);
```

```
i++;
```

```
}while(i<=10);
```

```
return 0;
```

```
}
```

Output

1

2

3
4
5
6
7
8
9
10

Program to print table for the given number using do while loop

```
#include<stdio.h>

int main(){
    int i=1,number=0;
    printf("Enter a number: ");
    scanf("%d",&number);
    do{
        printf("%d \n",(number*i));
        i++;
    }while(i<=10);
    return 0;
}
```

Output

Enter a number: 5

5
10

15
20
25
30
35
40
45
50

➤ **Nested while loop**

The nested while loop means any type of loop which is defined inside the 'while' loop.

```
while(condition)
{
    while(condition)
    {
        // inner loop statements.
    }
    // outer loop statements.
}
```

Example of nested while loop

```
#include <stdio.h>

int main()
{
    int rows; // variable declaration
    int columns; // variable declaration
```

```

int k=1; // variable initialization
printf("Enter the number of rows :"); // input the number of rows.
scanf("%d",&rows);
printf("\nEnter the number of columns :"); // input the number of columns.
scanf("%d",&columns);

int a[rows][columns]; //2d array declaration

int i=1;
while(i<=rows) // outer loop
{
    int j=1;
    while(j<=columns) // inner loop
    {
        printf("%d\t",k); // printing the value of k.
        k++; // increment counter
        j++;
    }
    i++;
    printf("\n");
}
}

```

➤ Nested do..while loop

The nested do..while loop means any type of loop which is defined inside the 'do..while' loop.

```

do
{
    do
    {
        // inner loop statements.
    }while(condition);
// outer loop statements.
}while(condition);

```

Example of nested do..while loop.

```

#include <stdio.h>

int main()
{
    /*printing the pattern
    *****
    *****
    *****
    ***** */

    int i=1;

    do    // outer loop
    {
        int j=1;

        do    // inner loop
        {
            printf("*");

```



```
    j++;  
}while(j<=8);  
printf("\n");  
i++;  
}while(i<=4);  
}
```

7. For loops

➤ Single for loop

Let's see the simple program of for loop that prints table of 1.

```
#include<stdio.h>

int main(){
int i=0;
for(i=1;i<=10;i++){
printf("%d \n",i);
}
return 0;
}
```

Output

```
1
2
3
4
5
6
7
8
9
10
```

C Program: Print table for the given number using C for loop

```
#include<stdio.h>

int main(){
int i=1,number=0;
printf("Enter a number: ");
scanf("%d",&number);
for(i=1;i<=10;i++){
printf("%d \n",(number*i));
}
return 0;
}
```

Output

Enter a number: 2

2

4

6

8

10

12

14

16

18

20

Properties of Expression 2

Expression 2 is a conditional expression. It checks for a specific condition to be satisfied. If it is not, the loop is terminated.

Expression 2 can have more than one condition. However, the loop will iterate until the last condition becomes false. Other conditions will be treated as statements.

Expression 2 is optional.

Expression 2 can perform the task of expression 1 and expression 3. That is, we can initialize the variable as well as update the loop variable in expression 2 itself.

We can pass zero or non-zero value in expression 2. However, in C, any non-zero value is true, and zero is false by default.

Example 1

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

```
int main()
```

```
{
```

```
    int i;
```

```
    for(i=0;i<=4;i++)
```

```
    {
```

```
        printf("%d ",i);
```

```
    }
```

```
}
```

output

0 1 2 3 4

Example 2

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

```
int main()
```

```
{
```

```
    int i,j,k;
```

```
for(i=0,j=0,k=0;i<4,k<8,j<10;i++)  
{  
    printf("%d %d %d\n",i,j,k);  
    j+=2;  
    k+=3;  
}  
}
```

Output

```
0 0 0  
1 2 3  
2 4 6  
3 6 9  
4 8 12
```

Example 3

```
#include <stdio.h>  
  
int main()  
{  
    int i;  
    for(i=0;;i++)  
    {  
        printf("%d",i);  
    }  
}
```

Output

infinite loop

Properties of Expression 3

Expression 3 is used to update the loop variable.

We can update more than one variable at the same time.

Expression 3 is optional.

Example 1

```
#include<stdio.h>

void main ()
{
    int i=0,j=2;
    for(i = 0;i<5;i++,j=j+2)
    {
        printf("%d %d\n",i,j);
    }
}
```

Output

0 2

1 4

2 6

3 8

4 10

Loop body

The braces {} are used to define the scope of the loop. However, if the loop contains only one statement, then we don't need to use braces. A loop without a body is

possible. The braces work as a block separator, i.e., the value variable declared inside for loop is valid only for that block and not outside. Consider the following example.

```
#include<stdio.h>

void main ()
{
    int i;
    for(i=0;i<10;i++)
    {
        int i = 20;
        printf("%d ",i);
    }
}
```

Output

20 20 20 20 20 20 20 20 20 20



Nested for loop

Logic to print square star pattern

```
*****
*****
*****
*****
*****
```

```

#include <stdio.h>

int main()
{
    int i, j, N;

    /* Input number of rows from user */
    printf("Enter number of rows: ");
    scanf("%d", &N);

    /* Iterate through N rows */
    for(i=1; i<=N; i++)
    {
        /* Iterate over columns */
        for(j=1; j<=N; j++)
        {
            /* Print star for each column */
            printf("*");
        }

        /* Move to the next line/row */
        printf("\n");
    }
}

```



```
return 0;  
}
```

Write a C program to print hollow square or rectangle star(*) pattern series using for loop. How to print hollow square or rectangle star pattern of N rows using for loop in C programming. Logic to print empty square or rectangle star pattern in C program.

Example

Input

Enter number of rows: 5

Output

```
*****  
*      *  
*      *  
*      *  
*      *  
*****
```

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

```
int main()
```

```
{
```

```
int i, j, N;
```

```
/* Input number of rows from user */
```

```
printf("Enter number of rows: ");
```

```
scanf("%d", &N);
```

```
/* Iterate over each row */
```

```
for(i=1; i<=N; i++)
```

```
{
```

```
/* Iterate over each column */
```

```
for(j=1; j<=N; j++)
```

```
{
```

```
if(i==1 || i==N || j==1 || j==N)
```

```
{
```

```
/* Print star for 1st, Nth row and column */
```

```
printf("*");
```

```
}
```

```
else
```

```
{
```

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

```
}
```

```
}
```

```
/* Move to the next line/row */
```

```
printf("\n");
```

```
}
```

```
return 0;
```

```
}
```

Write a C program to print hollow square star pattern with diagonal using loops. How to print hollow square star pattern with diagonals in C program. Logic to print hollow square star pattern with diagonal in C programming.

Example

Input

Enter number of rows: 5

Output

```
*****
*   *   *
* * * *
* * * *
*****
```

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

```
int main()
```

```
{
```

```
    int i, j, N;
```

```
    /* Input number of rows from user */
```

```
    printf("Enter number of rows: ");
```

```
scanf("%d", &N);
```

```
/* Iterate over each row */
```

```
for(i=1; i<=N; i++)
```

```
{
```

```
/* Iterate over each column */
```

```
for(j=1; j<=N; j++)
```

```
{
```

```
/*
```

```
 * Print star for,
```

```
 * first row (i==1) or
```

```
 * last row (i==N) or
```

```
 * first column (j==1) or
```

```
 * last column (j==N) or
```

```
 * row equal to column (i==j) or
```

```
 * column equal to N-row (j==N-i+1)
```

```
 */
```

```
if(i==1 || i==N || j==1 || j==N || i==j || j==(N - i + 1))
```

```
{
```

```
    printf("*");
```

```
}
```

```
else
```

```
{
```

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

```

    }
}

/* Move to the next line */
printf("\n");
}

return 0;
}

```

Write a C program to print rhombus star pattern of N rows using for loop. How to print rhombus or parallelogram star pattern using for loop in C programming. Logic to print rhombus or parallelogram star pattern series in C program.

Example

Input

Enter number of rows: 5

Output

```

    *****
  *****
*****
*****
*****

```

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

```

int main()
{
    int i, j, rows;

    /* Input number of rows from user */
    printf("Enter rows: ");
    scanf("%d", &rows);

    for(i=1; i<=rows; i++)
    {
        /* Print trailing spaces */
        for(j=1; j<=rows - i; j++)
        {
            printf(" ");
        }

        /* Print stars after spaces */
        for(j=1; j<=rows; j++)
        {
            printf("*");
        }

        /* Move to the next line */

```

```
printf("\n");  
}
```

```
return 0;  
}
```

Write a C program to print hollow rhombus star series pattern using for loop. How to print hollow rhombus or parallelogram star pattern in C programming. Logic to print empty rhombus or parallelogram star pattern series in C programming.

Example

Input

Input number of rows: 5

Output

```
*****  
  
*      *  
  
*      *  
  
*      *  
  
*****
```

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

```
int main()  
{  
    int i, j, rows;
```

```
/* Input number of rows from user */
```

```
printf("Enter rows : ");
```

```
scanf("%d", &rows);
```

```
for(i=1; i<=rows; i++)
```

```
{
```

```
/* Print trailing spaces */
```

```
for(j=1; j<=rows-i; j++)
```

```
{
```

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

```
}
```

```
/* Print stars and center spaces */
```

```
for(j=1; j<=rows; j++)
```

```
{
```

```
if(i==1 || i==rows || j==1 || j==rows)
```

```
printf("*");
```

```
else
```

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

```
}
```

```
printf("\n");
```



```
}
```

```
return 0;
```

```
}
```

Write a C program to print inverted/mirrored rhombus star pattern series using for loop. How to print inverted/mirrored rhombus or parallelogram star pattern of N rows in C program. Logic to print mirrored rhombus or parallelogram star pattern series in C programming.

Example

Input

Input rows: 5

Output

```
*****
```

```
*****
```

```
*****
```

```
*****
```

```
*****
```

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

```
int main()
```

```
{
```

```
int i, j, N;
```

```

/* Input number of rows from user */
printf("Enter rows: ");
scanf("%d", &N);


for(i=1; i<=N; i++)
{
    /* Print trailing spaces */
    for(j=1; j<i; j++)
    {
        printf(" ");
    }

    for(j=1; j<=N; j++)
    {
        printf("*");
    }

    printf("\n");
}

return 0;
}

```

Write a C program to print hollow mirrored rhombus star pattern series of N rows using for loop. How to print hollow mirrored rhombus or parallelogram star pattern

series in C program. Logic to print hollow inverted rhombus or parallelogram star pattern using loop in C programming.

Example

Input

Input rows: 5

Output

```
*****
*      *
*      *
*      *
*      *
*****
```

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

```
int main()
```

```
{
```

```
    int i, j, N;
```

```
    /* Input number of rows from user */
```

```
    printf("Enter number of rows: ");
```

```
    scanf("%d", &N);
```

```

    for(i=1; i<=N; i++)
    {
        /* Print trailing spaces */
        for(j=1; j<i; j++)
        {
            printf(" ");
        }

        /* Print hollow rhombus */
        for(j=1; j<=N; j++)
        {
            if(i==1 || i==N || j==1 || j==N)
                printf("*");
            else
                printf(" ");
        }

        printf("\n");
    }

    return 0;
}

```

Write a C program to print hollow right triangle star pattern series using for loop. How to print hollow right triangle star pattern series of n rows using for loop in C

programming. Logic to print hollow right triangle star pattern series in C programming.

Example

Input

Input rows: 5

Output

```
*
* *
*  *
*   *
*****
```

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

```
int main()
```

```
{
```

```
    int i, j, rows;
```

```
    /* Input rows from user */
```

```
    printf("Enter number of rows: ");
```

```
    scanf("%d", &rows);
```

```

for(i=1; i<=rows; i++)
{
    for(j=1; j<=i; j++)
    {
        /*
        * Print star for first column(j==1),
        * last column(j==i) or last row(i==rows).
        */
        if(j==1 || j==i || i==rows)
        {
            printf("*");
        }
        else
        {
            printf(" ");
        }
    }

    printf("\n");
}

return 0;
}

```

Write a C program to print inverted right triangle star pattern series using for loop.
How to print inverted right triangle star pattern series of n rows in C programming.
Logic to print inverted right triangle star pattern in C programming.

Example

Input

Input rows: 5

Output

**

*

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

```
int main()
```

```
{
```

```
    int i, j, rows;
```

```
    /* Input number of rows from user */
```

```
    printf("Enter number of rows : ");
```

```
    scanf("%d", &rows);
```

```

/* Iterate through rows */
for(i=1; i<=rows; i++)
{
    /* Iterate through columns */
    for(j=i; j<=rows; j++)
    {
        printf("*");
    }

    /* Move to the next line */
    printf("\n");
}

return 0;
}

```

Write a C program to print hollow inverted right triangle star pattern of n rows using for loop. How to print hollow inverted right triangle star pattern series of n rows in C program. Logic to print hollow inverted right triangle star pattern in C programming.

Example

Input

Input rows: 5

Output

```

*****
*   *

```


* *

* *

*

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

```
int main()
```

```
{
```

```
    int i, j, rows;
```

```
    /* Input number of rows from user */
```

```
    printf("Enter number of rows: ");
```

```
    scanf("%d", &rows);
```

```
    /* Iterate through rows */
```

```
    for(i=1; i<=rows; i++)
```

```
    {
```

```
        /* Iterate through columns */
```

```
        for(j=i; j<=rows; j++)
```

```
        {
```

```
            /*
```

```
                * Print stars for first row(i==1),
```

```
                * first column(j==1) and
```

```
                * last column(j=rows).
```

```

        */
        if(i==1 || j==i || j==rows)
        {
            printf("*");
        }
        else
        {
            printf(" ");
        }
    }

    /* Move to next line */
    printf("\n");
}

return 0;
}

```

Write a C program to print hollow mirrored inverted right triangle star pattern series of n rows using for loop. How to print hollow mirrored inverted right triangle star pattern in C program. Logic to print hollow mirrored inverted right triangle star pattern in C programming.

Example

Input

Input rows: 5

Output

* *

* *

* *

*

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

```
int main()
```

```
{
```

```
    int i, j, rows;
```

```
    /* Input rows to print from user */
```

```
    printf("Enter number of rows : ");
```

```
    scanf("%d", &rows);
```

```
    for(i=1; i<=rows; i++)
```

```
    {
```

```
        /* Print leading spaces */
```

```
        for(j=1; j<i; j++)
```

```
        {
```

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

```
        }
```

```

    /* Print hollow inverted right triangle */
    for(j=i; j<=rows; j++)
    {
        /*
         * Print star for ith column(j==i),
         * last column(j==rows) and for
         * first row(i==1).
         */
        if(j==i || j==rows || i==1)
        {
            printf("*");
        }
        else
        {
            printf(" ");
        }
    }

    printf("\n");
}

return 0;
}

```

Write a C program to print hollow pyramid or hollow equilateral triangle star pattern series using for loop. How to print hollow pyramid star pattern series in C program. Logic to print hollow pyramid star pattern series in C programming.

Example

Input

Input rows: 5

Output

```

      *
    * *
  *   *
*     *
*****
```

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

```
int main()
```

```
{
```

```
    int i, j, rows;
```

```
    /* Input rows to print from user */
```

```
    printf("Enter number of rows : ");
```

```
    scanf("%d", &rows);
```

```
for(i=1; i<=rows; i++)
```

```
{
```

```
    /* Print trailing spaces */
```

```
    for(j=i; j<rows; j++)
```

```
    {
```

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

```
    }
```

```
    /* Print hollow pyramid */
```

```
    for(j=1; j<=(2*i-1); j++)
```

```
    {
```

```
        /*
```

```
        * Print star for last row (i==rows),
```

```
        * first column(j==1) and for
```

```
        * last column (j==(2*i-1)).
```

```
        */
```

```
        if(i==rows || j==1 || j==(2*i-1))
```

```
        {
```

```
            printf("*");
```

```
        }
```

```
        else
```

```
        {
```

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

```
        }
```

```
}
```

```
/* Move to next line */
```

```
printf("\n");
```

```
}
```

```
return 0;
```

```
}
```

Write a C program to print reverse Pyramid or reverse equilateral triangle star pattern series using for loop. How to print inverted pyramid or inverted equilateral triangle star pattern series in C program. Logic to print reverse pyramid star pattern series in C programming.

Example

Input

Input rows: 5

Output

```
*****
```

```
*****
```

```
*****
```

```
***
```

```
*
```

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

```

int main()
{
    int i, j, rows;

    /* Input rows to print from user */
    printf("Enter number of rows : ");
    scanf("%d", &rows);

    for(i=1; i<=rows; i++)
    {
        /* Print leading spaces */
        for(j=1; j<i; j++)
        {
            printf(" ");
        }

        /* Print stars */
        for(j=1; j<=(rows*2 -(2*i-1)); j++)
        {
            printf("*");
        }

        /* Move to next line */
        printf("\n");
    }
}

```



```
}
```

```
return 0;
```

```
}
```

Write a C program print hollow inverted pyramid star pattern series of n rows using for loop. How to print hollow inverted equilateral triangle star pattern series of n rows in C program. Logic to print hollow inverted pyramid star pattern in C programming.

Example

Input

Input rows: 5

Output

```
*****
```

```
*      *
```

```
*      *
```

```
*    *
```

```
  *
```

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

```
int main()
```

```
{
```

```
int i, j, rows;
```

```
/* Input rows to print from user */
```

```
printf("Enter number of rows: ");
```

```
scanf("%d", &rows);
```

```
/* Iterate through rows */
```

```
for(i=1; i<=rows; i++)
```

```
{
```

```
/* Print leading spaces */
```

```
for(j=1; j<i; j++)
```

```
{
```

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

```
}
```

```
/* Print hollow pyramid */
```

```
for(j=1; j<=(rows*2 - (2*i-1)); j++)
```

```
{
```

```
/*
```

```
* Print star for first row(i==1),
```

```
* ith column (j==1) and for
```

```
* last column (rows*2-(2*i-1))
```

```
*/
```

```
if(i==1 || j==1 || j==(rows*2 - (2*i - 1)))
```

```
{
```

```
printf("*");
```

```

    }
else
{
    printf(" ");
}
}

/* Move to next line */
printf("\n");
}

return 0;
}

```

Write a C program to print the half diamond star pattern using for loop. How to print half diamond star pattern structure using for loop in C program. Logic to print half diamond star pattern series in C programming.

Example

Input

Input rows: 5

Output

```

*
* *

```

**

*

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

```
int main()
```

```
{
```

```
    int i, j, N, columns;
```

```
    /* Input number of columns from user */
```

```
    printf("Enter number of columns:");
```

```
    scanf("%d",&N);
```

```
    columns=1;
```

```
    for(i=1;i<N*2;i++)
```

```
    {
```

```
        for(j=1; j<=columns; j++)
```

```
        {
```

```

    printf("*");
}

if(i < N)
{
    /* Increment number of columns per row for upper part */
    columns++;
}
else
{
    /* Decrement number of columns per row for lower part */
    columns--;
}

/* Move to next line */
printf("\n");
}

return 0;
}

```

Write a C program to print the mirrored half diamond star pattern using for loop. How to print mirrored half diamond star pattern in C program. Logic to print mirrored half diamond star pattern in C programming.

Example

Input

Input rows: 5

Output

```
  *  
 * *  
* * *  
* * * *  
* * * * *  
 * * * *  
  * * *  
   * *  
    *
```

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

```
int main()
```

```
{
```

```
    int i, j, N;
```

```
    int star, spaces;
```

```
    /* Input number of columns to print from user */
```

```
    printf("Enter number of columns : ");
```

```
    scanf("%d", &N);
```

```

spaces = N-1;
star = 1;

/* Iterate through rows */
for(i=1; i<N*2; i++)
{
    /* Print leading spaces */
    for(j=1; j<=spaces; j++)
        printf(" ");

    /* Print stars */
    for(j=1; j<=star; j++)
        printf("*");

    /* Move to next line */
    printf("\n");

    if(i < N)
    {
        star++;
        spaces--;
    }
    else

```

```

    {
        star--;
        spaces++;
    }
}
return 0;
}

```

Write a C program to print diamond star pattern series using for loop. How to print diamond star pattern structure in C program. Logic to print diamond star pattern series in C programming.

Example

Input

Input rows: 5

Output

```

    *
  * * *
* * * * *
* * * * * *
* * * * * * *
  * * * * *
    * * * *
      * * *
        *

```

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



```

int main()
{
    int i, j, rows;
    int stars, spaces;

    printf("Enter rows to print : ");
    scanf("%d", &rows);

    stars = 1;
    spaces = rows - 1;

    /* Iterate through rows */
    for(i=1; i<rows*2; i++)
    {
        /* Print spaces */
        for(j=1; j<=spaces; j++)
            printf(" ");

        /* Print stars */
        for(j=1; j<stars*2; j++)
            printf("*");
    }
}

```

```

    /* Move to next line */
    printf("\n");

    if(i<rows)
    {
        spaces--;
        stars++;
    }
    else
    {
        spaces++;
        stars--;
    }
}

return 0;
}

```

Write a C program to print hollow diamond star pattern series of n rows using for loop. How to print hollow diamond star pattern structure in C program. Logic to print hollow diamond star pattern in C programming.

Example

Input

Input N: 5

Output

```

*****

***   ***

**    **

*      *

*      *

**     **

***    ***

*****

*****

```

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

```
int main()
```

```
{
```

```
    int i, j, n;
```

```
    printf("Enter value of n : ");
```

```
    scanf("%d", &n);
```

```
    // Loop to print upper half of the pattern
```

```
    for(i=1; i<=n; i++)
```

```
    {
```

```
        for(j=i; j<=n; j++)
```

```
{
```

```
    printf("*");
```

```
}
```

```
    for(j=1; j<=(2*i-2); j++)
```

```
    {
```

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

```
    }
```

```
    for(j=i; j<=n; j++)
```

```
    {
```

```
        printf("*");
```

```
    }
```

```
    printf("\n");
```

```
}
```

```
// Loop to print lower half of the pattern
```

```
for(i=1; i<=n; i++)
```

```
{
```

```
    for(j=1; j<=i; j++)
```

```
    {
```

```
        printf("*");
```

```
    }
```

```

    for(j=(2*i-2); j<(2*n-2); j++)
    {
        printf(" ");
    }

    for(j=1; j<=i; j++)
    {
        printf("*");
    }

    printf("\n");
}

return 0;
}

```

Write a C program to print right arrow star pattern series using for loop. How to print right arrow star pattern structure in C program. Logic to print right arrow star pattern in C programming.

Example

Input

Input N: 5

Output

```
***  
  **  
   *  
  **  
 ***  
****  
*****
```

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

```
int main()
```

```
{
```

```
    int i, j, n;
```

```
    // Input number of rows from user
```

```
    printf("Enter value of n : ");
```

```
    scanf("%d", &n);
```

```
    // Print the upper part of the arrow
```

```
    for(i=1; i<n; i++)
```

```
    {
```

```
        // Print trailing (2*rownumber-2) spaces
```

```
        for(j=1; j<=(2*i-2); j++)
```

```
        {
```

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

```
}
```

```
// Print inverted right triangle star pattern
```

```
for(j=i; j<=n; j++)
```

```
{
```

```
printf("*");
```

```
}
```

```
printf("\n");
```

```
}
```

```
// Print lower part of the arrow
```

```
for(i=1; i<=n; i++)
```

```
{
```

```
// Print trailing (2*n - 2*rownumber) spaces
```

```
for(j=1; j<=(2*n - 2*i); j++)
```

```
{
```

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

```
}
```

```
// Print simple right triangle star pattern
```

```
for(j=1; j<=i; j++)
```

```
{
```

```
printf("*");
```

```

    }
    printf("\n");
}
return 0;
}

```

Write a C program to print left arrow star pattern series using for loop. How to print left arrow (mirrored arrow) star pattern structure in C program. Logic to print left arrow star pattern in C programming.

Example

Input

Input N: 5

Output

```

    * * * * *
  * * * *
* * *
* *
*
* *
* * *
* * * *
* * * * *

```

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

```
int main()
```



```

{
    int i, j, n;

    // Input number of rows from user
    printf("Enter value of n : ");
    scanf("%d", &n);

    // Print upper part of the arrow
    for(i=1; i<n; i++)
    {
        // Print trailing (n-rownumber) spaces
        for(j=1; j<=(n-i); j++)
        {
            printf(" ");
        }

        // Print inverted right triangle
        for(j=i; j<=n; j++)
        {
            printf("*");
        }

        printf("\n");
    }
}

```

```

// Print bottom part of the arrow
for(i=1; i<=n; i++)
{
    // Print trailing (rownumber-1) spaces
    for(j=1; j<i; j++)
    {
        printf(" ");
    }

    // Print the right triangle
    for(j=1; j<=i; j++)
    {
        printf("*");
    }

    printf("\n");
}

return 0;
}

```

Write a C program to print plus star pattern series using for loop. How to print plus star pattern series using loop in C program. Logic to print plus star pattern in C programming.

Example

Input

Input N: 5

Output

```
      +
      +
      +
      +
+++++++
      +
      +
      +
      +
```

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

```
int main()
```

```
{
```

```
    int i, j, N;
```

```
    printf("Enter N: ");
```

```
    scanf("%d", &N);
```

```
    // Run an outer loop from 1 to N*2-1
```

```

for(i=1; i<=(N * 2 - 1); i++)
{
    // For the center horizontal plus
    if(i == N)
    {
        for(j=1; j<=(N * 2 - 1); j++)
        {
            printf("+");
        }
    }
    else
    {
        // For spaces before single plus sign
        for(j=1; j<=N-1; j++)
        {
            printf(" ");
        }
        printf("+");
    }

    printf("\n");
}

return 0;

```

```
}
```

Write a C program to print X star pattern series using loop. How to print the X star pattern series using for loop in C program. Logic to print X using stars in C programming.

Example

Input

Input N: 5

Output

```
*      *  
  
*      *  
  
*      *  
  
*      *  
  
*      *  
  
*      *  
  
*      *  
  
*      *  
  
*      *  
  
*      *
```

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

```
int main()
```

```
{
```

```
    int i, j, N;
```

```
    int count;
```

```
printf("Enter N: ");
```

```
scanf("%d", &N);
```

```
count = N * 2 - 1;
```

```
for(i=1; i<=count; i++)
```

```
{
```

```
    for(j=1; j<=count; j++)
```

```
    {
```

```
        if(j==i || (j==count - i + 1))
```

```
        {
```

```
            printf("*");
```

```
        }
```

```
    else
```

```
    {
```

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

```
    }
```

```
}
```

```
printf("\n");
```

```
}
```

```
return 0;
```

```
}
```

Write a C program to print 8 star pattern series using loop. How to print 8 star pattern series using for loop in C program. Logic to print 8 star pattern series of N columns in C programming.

Example

Input

Input N: 5

Output

```
***
*  *
*  *
*  *
*  *
***
*  *
*  *
*  *
***
```

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

```
int main()
```

```
{
```

```
    int i, j, size;
```

```
printf("Enter size: ");
```

```
scanf("%d", &size);
```

```
for(i=1; i<size*2; i++)
```

```
{
```

```
    for(j=1; j<=size; j++)
```

```
    {
```

```
        // Condition for corner and center intersection space
```

```
        if((i==1 && (j==1 || j==size)) ||
```

```
           (i==size && (j==1 || j==size)) ||
```

```
           (i==size*2-1 && (j==1 || j==size)))
```

```
        {
```

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

```
        }
```

```
        else if(i==1 || i==size || i==(size*2)-1 || j==1 || j==size)
```

```
        {
```

```
            printf("*");
```

```
        }
```

```
        else
```

```
        {
```

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

```
        }
```

```
    }
```

```
printf("\n");
```


}

Output

*

153

```

int main()
{
    int i, j, n;

    printf("Enter value of n : ");
    scanf("%d", &n);

    for(i=n/2; i<=n; i+=2)
    {
        for(j=1; j<n-i; j+=2)
        {
            printf(" ");
        }

        for(j=1; j<=i; j++)
        {
            printf("*");
        }

        for(j=1; j<=n-i; j++)
        {
            printf(" ");
        }
    }
}

```

```
    for(j=1; j<=i; j++)
```

```
    {
```

```
        printf("*");
```

```
    }
```

```
    printf("\n");
```

```
}
```

```
for(i=n; i>=1; i--)
```

```
{
```

```
    for(j=i; j<n; j++)
```

```
    {
```

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

```
    }
```

```
    for(j=1; j<=(i*2)-1; j++)
```

```
    {
```

```
        printf("*");
```

```
    }
```

```
    printf("\n");
```

```
}
```

```
return 0;
```



```

int main()
{
    int i, j, n;
    char name[50];
    int len;

    printf("Enter your name: ");
    gets(name);

    printf("Enter value of n : ");
    scanf("%d", &n);

    len = strlen(name);

    // Print upper part of the heart shape
    for(i=n/2; i<=n; i+=2)
    {
        for(j=1; j<n-i; j+=2)
        {
            printf(" ");
        }

        for(j=1; j<=i; j++)

```

```

    {
        printf("*");
    }

    for(j=1; j<=n-i; j++)
    {
        printf(" ");
    }
    for(j=1; j<=i; j++)
    {
        printf("*");
    }
    printf("\n");
}

// Prints lower triangular part of the pattern
for(i=n; i>=1; i--)
{
    for(j=i; j<n; j++)
    {
        printf(" ");
    }

    // Print the name
    if(i == n)

```

```

{
    for(j=1; j<=(n * 2-len)/2; j++)
    {
        printf("*");
    }
    printf("%s", name);

    for(j=1; j<(n*2-len)/2; j++)
    {
        printf("*");
    }
}
else
{
    for(j=1; j<=(i*2)-1; j++)
    {
        printf("*");
    }
}

printf("\n");
}
return 0;
}

```

8. C – Arrays(Very important)

➤ 1D array

C array example

```
#include<stdio.h>

int main(){
    int i=0;
    int marks[5];//declaration of array
    marks[0]=80;//initialization of array
    marks[1]=60;
    marks[2]=70;
    marks[3]=85;
    marks[4]=75;
    //traversal of array
    for(i=0;i<5;i++){
        printf("%d \n",marks[i]);
    }//end of for loop
    return 0;
}
```

Output

80

60

70

85

75

Let's see the C program to declare and initialize the array in C.

```
#include<stdio.h>

int main(){

int i=0;

int marks[5]={20,30,40,50,60};//declaration and initialization of array

//traversal of array
for(i=0;i<5;i++){
printf("%d \n",marks[i]);
}

return 0;
}
```

Output

20

30

40

50

60

Write a C program to declare, initialize, input elements in array and print array. How to input and display elements in an array using for loop in C programming. C program to input and print array elements using loop.

Example

Input

Input size: 10

Input elements: 1

2

3

4

5

6

7

8

9

10

Output

Output: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

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

```
#define MAX_SIZE 1000 // Maximum array size
```

```
int main()
```

```
{
```

```
    int arr[MAX_SIZE]; // Declare an array of MAX_SIZE
```

```
    int i, N;
```

```
    /* Input array size */
```

```
    printf("Enter size of array: ");
```

```
    scanf("%d", &N);
```

```
    /* Input elements in array */
```

```

printf("Enter %d elements in the array : ", N);
for(i=0; i<N; i++)
{
    scanf("%d", &arr[i]);
}

/*
 * Print all elements of array
 */
printf("\nElements in array are: ");
for(i=0; i<N; i++)
{
    printf("%d, ", arr[i]);
}

return 0;
}

```

Write a C program to input elements in array and print all negative elements. How to display all negative elements in array using loop in C program. Logic to display all negative elements in a given array in C programming.

Example

Input

Input array:

-1

-10

100

5

61

-2

-23

8

-90

51

Output

Output: -1, -10, -2, -23, -90

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

```
#define MAX_SIZE 100 // Maximum array size
```

```
int main()
```

```
{
```

```
    int arr[MAX_SIZE]; // Declare array of MAX_SIZE
```

```
    int i, N;
```

```
    /* Input size of the array */
```

```
    printf("Enter size of the array : ");
```

```
    scanf("%d", &N);
```

```

/* Input elements in the array */
printf("Enter elements in array : ");
for(i=0; i<N; i++)
{
    scanf("%d", &arr[i]);
}

printf("\nAll negative elements in array are : ");
for(i=0; i<N; i++)
{
    /* If current array element is negative */
    if(arr[i] < 0)
    {
        printf("%d\t", arr[i]);
    }
}

return 0;
}

```

Write a C program to read elements in an array and find the sum of array elements. C program to find sum of elements of the array. How to add elements of an array using for loop in C programming. Logic to find sum of array elements in C programming.

Example

Input

Input elements: 10, 20, 30, 40, 50

Output

Sum of all elements = 150

```
#include <stdio.h>

#define MAX_SIZE 100

int main()
{
    int arr[MAX_SIZE];
    int i, n, sum=0;

    /* Input size of the array */
    printf("Enter size of the array: ");
    scanf("%d", &n);

    /* Input elements in array */
    printf("Enter %d elements in the array: ", n);
    for(i=0; i<n; i++)
    {
        scanf("%d", &arr[i]);
    }

    /*
    * Add each array element to sum
    */
```

```

    for(i=0; i<n; i++)
    {
        sum = sum + arr[i];
    }

    printf("Sum of all elements of array = %d", sum);

    return 0;
}

```

Write a C program to input elements in an array from user, find maximum and minimum element in array. C program to find biggest and smallest elements in an array. Logic to find maximum and minimum element in array in C programming.

Example

Input

Input array elements: 10, 50, 12, 16, 2

Output

Maximum = 50

Minimum = 2

```

#include <stdio.h>

#define MAX_SIZE 100 // Maximum array size

int main()
{

```

```
int arr[MAX_SIZE];
```

```
int i, max, min, size;
```

```
/* Input size of the array */
```

```
printf("Enter size of the array: ");
```

```
scanf("%d", &size);
```

```
/* Input array elements */
```

```
printf("Enter elements in the array: ");
```

```
for(i=0; i<size; i++)
```

```
{
```

```
    scanf("%d", &arr[i]);
```

```
}
```

```
/* Assume first element as maximum and minimum */
```

```
max = arr[0];
```

```
min = arr[0];
```

```
/*
```

```
 * Find maximum and minimum in all array elements.
```

```
*/
```

```
for(i=1; i<size; i++)
```

```
{
```



```

    /* If current element is greater than max */
    if(arr[i] > max)
    {
        max = arr[i];
    }

    /* If current element is smaller than min */
    if(arr[i] < min)
    {
        min = arr[i];
    }
}

/* Print maximum and minimum element */
printf("Maximum element = %d\n", max);
printf("Minimum element = %d", min);

return 0;
}

```

Write a C program to input elements in array and copy all elements of first array into second array. How to copy array elements to another array in C programming. Logic to copy array elements in C program using loop.

Example

Input

Input array1 elements: 10 1 95 30 45 12 60 89 40 -4

Output

Array1: 10 1 95 30 45 12 60 89 40 -4

Array2: 10 1 95 30 45 12 60 89 40 -4

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

```
#define MAX_SIZE 100
```

```
int main()
```

```
{
```

```
    int source[MAX_SIZE], dest[MAX_SIZE];
```

```
    int i, size;
```

```
    /* Input size of the array */
```

```
    printf("Enter the size of the array : ");
```

```
    scanf("%d", &size);
```

```
    /* Input array elements */
```

```
    printf("Enter elements of source array : ");
```

```
    for(i=0; i<size; i++)
```

```
    {
```

```
        scanf("%d", &source[i]);
```

```
    }
```

```
    /*
```

```
    * Copy all elements from source array to dest array
```

```

    */
    for(i=0; i<size; i++)
    {
        dest[i] = source[i];
    }

    /*
    * Print all elements of source array
    */
    printf("\nElements of source array are : ");
    for(i=0; i<size; i++)
    {
        printf("%d\t", source[i]);
    }

    /*
    * Print all elements of dest array
    */
    printf("\nElements of dest array are : ");
    for(i=0; i<size; i++)
    {
        printf("%d\t", dest[i]);
    }

```

```
    return 0;
}
```

Write a C program to input elements in array and count negative elements in array. C program to find all negative elements in array. How to count negative elements in array using loop in C programming. Logic to count total negative elements in an array in C program.

Example

Input

Input array elements : 10, -2, 5, -20, 1, 50, 60, -50, -12, -9

Output

Total number of negative elements: 5

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

```
#define MAX_SIZE 100 // Maximum array size
```

```
int main()
```

```
{
```

```
    int arr[MAX_SIZE]; // Declares array of size 100
```

```
    int i, size, count = 0;
```

```
    /* Input size of array */
```

```
    printf("Enter size of the array : ");
```

```
    scanf("%d", &size);
```

```

/* Input array elements */
printf("Enter elements in array : ");
for(i=0; i<size; i++)
{
    scanf("%d", &arr[i]);
}

/*
 * Count total negative elements in array
 */
for(i=0; i<size; i++)
{
    /* Increment count if current array element is negative */
    if(arr[i] < 0)
    {
        count++;
    }
}

printf("\nTotal negative elements in array = %d", count);

return 0;
}

```

Write a C program to input elements in array from user and count even and odd elements in array. How to find total number of even and odd elements in a given array using C programming. Logic to count even and odd elements in array using loops.

Example

Input

Input array: 1 2 3 4 5 6 7 8 9

Output

Total even elements: 4

Total odd elements: 5

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

```
#define MAX_SIZE 100 //Maximum size of the array
```

```
int main()
```

```
{
```

```
    int arr[MAX_SIZE];
```

```
    int i, size, even, odd;
```

```
    /* Input size of the array */
```

```
    printf("Enter size of the array: ");
```

```
    scanf("%d", &size);
```

```
    /* Input array elements */
```

```
    printf("Enter %d elements in array: ", size);
```

```
for(i=0; i<size; i++)
```

```
{
```

```
    scanf("%d", &arr[i]);
```

```
}
```

```
/* Assuming that there are 0 even and odd elements */
```

```
even = 0;
```

```
odd = 0;
```

```
for(i=0; i<size; i++)
```

```
{
```

```
    /* If the current element of array is even then increment even count */
```

```
    if(arr[i]%2 == 0)
```

```
    {
```

```
        even++;
```

```
    }
```

```
    else
```

```
    {
```

```
        odd++;
```

```
    }
```

```
}
```

```
printf("Total even elements: %d\n", even);
```

```
printf("Total odd elements: %d", odd);
```

```
    return 0;
}
```

C Array Example: Sorting an array

In the following program, we are using bubble sort method to sort the array in ascending order.

```
#include<stdio.h>

void main ()
{
    int i, j,temp;
    int a[10] = { 10, 9, 7, 101, 23, 44, 12, 78, 34, 23};
    for(i = 0; i<10; i++)
    {
        for(j = i+1; j<10; j++)
        {
            if(a[j] > a[i])
            {
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
        }
    }

    printf("Printing Sorted Element List ...\n");
    for(i = 0; i<10; i++)
```



```

{
    printf("%d\n",a[i]);
}
}

```

Program to print the largest and second largest element of the array.

```

#include<stdio.h>

void main ()
{
    int arr[100],i,n,largest,sec_largest;
    printf("Enter the size of the array?");
    scanf("%d",&n);
    printf("Enter the elements of the array?");
    for(i = 0; i<n; i++)
    {
        scanf("%d",&arr[i]);
    }
    largest = arr[0];
    sec_largest = arr[1];
    for(i=0;i<n;i++)
    {
        if(arr[i]>largest)
        {
            sec_largest = largest;
            largest = arr[i];
        }
    }
}

```

```

    }
    else if (arr[i]>sec_largest && arr[i]!=largest)
    {
        sec_largest=arr[i];
    }
}

printf("largest = %d, second largest = %d",largest,sec_largest);

}

```

➤ 2D array

In the 1D array, we don't need to specify the size of the array if the declaration and initialization are being done simultaneously. However, this will not work with 2D arrays. We will have to define at least the second dimension of the array. The two-dimensional array can be declared and defined in the following way.

```
int arr[4][3]={1,2,3},{2,3,4},{3,4,5},{4,5,6}};
```

Two-dimensional array example in C

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

```
int main(){
```

```
int i=0,j=0;
```

```
int arr[4][3]={1,2,3},{2,3,4},{3,4,5},{4,5,6}};
```

```
//traversing 2D array
```

```
for(i=0;i<4;i++){
```

```
    for(j=0;j<3;j++){
```

```
    printf("arr[%d] [%d] = %d \n",i,j,arr[i][j]);  
  }//end of j  
} //end of i  
return 0;  
}
```

Output

```
arr[0][0] = 1  
arr[0][1] = 2  
arr[0][2] = 3  
arr[1][0] = 2  
arr[1][1] = 3  
arr[1][2] = 4  
arr[2][0] = 3  
arr[2][1] = 4  
arr[2][2] = 5  
arr[3][0] = 4  
arr[3][1] = 5  
arr[3][2] = 6
```

C 2D array example: Storing elements in a matrix and printing it.

```
#include <stdio.h>  
  
void main ()  
{  
    int arr[3][3],i,j;  
    for (i=0;i<3;i++)
```

```

{
    for (j=0;j<3;j++)
    {
        printf("Enter a[%d][%d]: ",i,j);
        scanf("%d",&arr[i][j]);
    }
}
printf("\n printing the elements ....\n");
for(i=0;i<3;i++)
{
    printf("\n");
    for (j=0;j<3;j++)
    {
        printf("%d\t",arr[i][j]);
    }
}
}

```

Output

Enter a[0][0]: 56

Enter a[0][1]: 10

Enter a[0][2]: 30

Enter a[1][0]: 34

Enter a[1][1]: 21

Enter a[1][2]: 34

Enter a[2][0]: 45

Enter a[2][1]: 56

Enter a[2][2]: 78

printing the elements

56 10 30

34 21 34

45 56 78

Write a C program to read elements in two matrices and add elements of both matrices. C program for addition of two matrix. Matrix addition program in C. Logic to add two matrix in C programming.

Example

Input

Input elements in 3x3 matrix1:

1 2 3

4 5 6

7 8 9

Input elements in 3x3 matrix2:

9 8 7

6 5 4

3 2 1

Output

Sum of both matrix =

10 10 10

10 10 10

10 10 10

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

```
#define SIZE 3 // Size of the matrix
```

```
int main()
```

```
{
```

```
    int A[SIZE][SIZE]; // Matrix 1
```

```
    int B[SIZE][SIZE]; // Matrix 2
```

```
    int C[SIZE][SIZE]; // Resultant matrix
```

```
    int row, col;
```

```
    /* Input elements in first matrix*/
```

```
    printf("Enter elements in matrix A of size 3x3: \n");
```

```
    for(row=0; row<SIZE; row++)
```

```
    {
```

```
        for(col=0; col<SIZE; col++)
```

```
        {
```

```
            scanf("%d", &A[row][col]);
```

```
        }
```

```
    }
```

```

/* Input elements in second matrix */

printf("\nEnter elements in matrix B of size 3x3: \n");

for(row=0; row<SIZE; row++)
{
    for(col=0; col<SIZE; col++)
    {
        scanf("%d", &B[row][col]);
    }
}

/*
 * Add both matrices A and B entry wise or element wise
 * and stores result in matrix C
 */

for(row=0; row<SIZE; row++)
{
    for(col=0; col<SIZE; col++)
    {
        /* Cij = Aij + Bij */
        C[row][col] = A[row][col] + B[row][col];
    }
}

```

```

/* Print the value of resultant matrix C */
printf("\nSum of matrices A+B = \n");
for(row=0; row<SIZE; row++)
{
    for(col=0; col<SIZE; col++)
    {
        printf("%d ", C[row][col]);
    }
    printf("\n");
}

return 0;
}

```

Write a C program to read elements in two matrices and find the difference of two matrices. Program to subtract two matrices in C. Logic to subtract two matrices in C programming.

Example

Input

Input elements in 3x3 matrix1:

1 2 3

4 5 6

7 8 9

Input elements in 3x3 matrix2:

9 8 7

6 5 4

3 2 1

Output

Difference of both matrices =

-8 -6 -4

-2 0 2

4 6 8

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

```
#define SIZE 3 // Size of the matrix
```

```
int main()
```

```
{
```

```
    int A[SIZE][SIZE]; // Matrix 1
```

```
    int B[SIZE][SIZE]; // Matrix 2
```

```
    int C[SIZE][SIZE]; // Resultant matrix
```

```
    int row, col;
```

```
    /* Input elements in first matrix */
```

```
    printf("Enter elements in matrix A of size 3x3: \n");
```

```
    for(row=0; row<SIZE; row++)
```

```
    {
```

```

    for(col=0; col<SIZE; col++)
    {
        scanf("%d", &A[row][col]);
    }
}

/* Input elements in second matrix */
printf("\nEnter elements in matrix B of size 3x3: \n");
for(row=0; row<SIZE; row++)
{
    for(col=0; col<SIZE; col++)
    {
        scanf("%d", &B[row][col]);
    }
}

/*
* Subtract both matrices and store the result in matrix C
*/
for(row=0; row<SIZE; row++)
{
    for(col=0; col<SIZE; col++)
    {
        /* Cij = Aij - Bij */

```

```

        C[row][col] = A[row][col] - B[row][col];
    }
}

/* Print difference of both matrices A and B */
printf("\nDifference of two matrices A-B = \n");
for(row=0; row<SIZE; row++)
{
    for(col=0; col<SIZE; col++)
    {
        printf("%d ", C[row][col]);
    }
    printf("\n");
}

return 0;
}

```

Write a C program to read elements in a matrix and perform scalar multiplication of matrix. C program for scalar multiplication of matrix. How to perform scalar matrix multiplication in C programming. Logic to perform scalar matrix multiplication in C program.

Example

Input

Input elements of matrix A:

1 2 3

4 5 6

7 8 9

Input multiplier: 2

Output

2 4 6

8 10 12

14 16 18

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

```
#define SIZE 3 // Maximum size of the array
```

```
int main()
```

```
{
```

```
    int A[SIZE][SIZE];
```

```
    int num, row, col;
```

```
    /* Input elements in matrix from user */
```

```
    printf("Enter elements in matrix of size %dx%d: \n", SIZE, SIZE);
```

```
    for(row=0; row<SIZE; row++)
```

```
    {
```

```
        for(col=0; col<SIZE; col++)
```

```
        {
```

```
            scanf("%d", &A[row][col]);
```

```
        }
```

```
}
```

```
/* Input multiplier from user */
```

```
printf("Enter any number to multiply with matrix A: ");
```

```
scanf("%d", &num);
```

```
/* Perform scalar multiplication of matrix */
```

```
for(row=0; row<SIZE; row++)
```

```
{
```

```
    for(col=0; col<SIZE; col++)
```

```
    {
```

```
        /* (cAij) = c . Aij */
```

```
        A[row][col] = num * A[row][col];
```

```
    }
```

```
}
```

```
/* Print result of scalar multiplication of matrix */
```

```
printf("\nResultant matrix c.A = \n");
```

```
for(row=0; row<SIZE; row++)
```

```
{
```

```
    for(col=0; col<SIZE; col++)
```

```
    {
```

```
        printf("%d ", A[row][col]);
```

```
    }
```

```
printf("\n");  
}
```

```
return 0;  
}
```

Write a C program to read elements in two matrices and multiply them. Matrix multiplication program in C. How to multiply matrices in C. Logic to multiply two matrices in C programming.

Example

Input

Input elements of matrix1:

1 2 3

4 5 6

7 8 9

Input elements of matrix2:

9 8 7

6 5 4

3 2 1

Output

Product of matrices =

30 24 18

84 69 54

138 114 90

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

```
#define SIZE 3 // Size of the matrix
```

```
int main()
```

```
{
```

```
    int A[SIZE][SIZE]; // Matrix 1
```

```
    int B[SIZE][SIZE]; // Matrix 2
```

```
    int C[SIZE][SIZE]; // Resultant matrix
```

```
    int row, col, i, sum;
```

```
    /* Input elements in first matrix from user */
```

```
    printf("Enter elements in matrix A of size %dx%d: \n", SIZE, SIZE);
```

```
    for(row=0; row<SIZE; row++)
```

```
    {
```

```
        for(col=0; col<SIZE; col++)
```

```
        {
```

```
            scanf("%d", &A[row][col]);
```

```
        }
```

```
    }
```

```
    /* Input elements in second matrix from user */
```

```
    printf("\nEnter elements in matrix B of size %dx%d: \n", SIZE, SIZE);
```

```
    for(row=0; row<SIZE; row++)
```

```

{
    for(col=0; col<SIZE; col++)
    {
        scanf("%d", &B[row][col]);
    }
}

/*
 * Multiply both matrices A*B
 */
for(row=0; row<SIZE; row++)
{
    for(col=0; col<SIZE; col++)
    {
        sum = 0;
        /*
         * Multiply row of first matrix to column of second matrix
         * and store sum of product of elements in sum.
         */
        for(i=0; i<SIZE; i++)
        {
            sum += A[row][i] * B[i][col];
        }
    }
}

```



```

        C[row][col] = sum;
    }
}

/* Print product of the matrices */
printf("\nProduct of matrix A * B = \n");
for(row=0; row<SIZE; row++)
{
    for(col=0; col<SIZE; col++)
    {
        printf("%d ", C[row][col]);
    }
    printf("\n");
}

return 0;
}

```

Write a C program to enter elements in two matrices and check whether both matrices are equal or not. C program to check whether elements of two matrices are equal or not. Logic to check if two matrices are equal or not in C programming.

Example

Input

Input elements of matrix1:

1 2 3

4 5 6

7 8 9

Input elements of matrix2:

1 2 3

4 5 6

7 8 9

Output

Both matrices are equal

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

```
#define SIZE 3 // Matrix size
```

```
int main()
```

```
{
```

```
    int A[SIZE][SIZE];
```

```
    int B[SIZE][SIZE];
```

```
    int row, col, isEqual;
```

```
    /* Input elements in first matrix from user */
```

```
    printf("Enter elements in matrix A of size %dx%d: \n", SIZE, SIZE);
```

```
    for(row=0; row<SIZE; row++)
```

```
    {
```

```
        for(col=0; col<SIZE; col++)
```

```

    {
        scanf("%d", &A[row][col]);
    }
}

/* Input elements in second matrix from user */
printf("\nEnter elements in matrix B of size %dx%d: \n");
for(row=0; row<SIZE; row++)
{
    for(col=0; col<SIZE; col++)
    {
        scanf("%d", &B[row][col]);
    }
}

/* Assumes that the matrices are equal */
isEqual = 1;

for(row=0; row<SIZE; row++)
{
    for(col=0; col<SIZE; col++)
    {
        /*
         * If the corresponding entries of matrices are not equal

```

```

        */
        if(A[row][col] != B[row][col])
        {
            isEqual = 0;
            break;
        }
    }
}

/*
 * Checks the value of isEqual
 * As per our assumption if isEqual contains 1 means both are equal
 * If it contains 0 means both are not equal
 */
if(isEqual == 1)
{
    printf("\nMatrix A is equal to Matrix B");
}
else
{
    printf("\nMatrix A is not equal to Matrix B");
}

return 0;

```

```
}
```

Write a C program to read elements in a matrix and find the sum of main diagonal (major diagonal) elements of matrix. Find sum of all elements of main diagonal of a matrix. Logic to find sum of main diagonal elements of a matrix in C programming.

Example

Input

Input array elements:

1 2 3

4 5 6

7 8 9

Output

Sum of main diagonal elements = 15

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

```
#define SIZE 3 // Matrix size
```

```
int main()
```

```
{
```

```
    int A[SIZE][SIZE];
```

```
    int row, col, sum = 0;
```

```
    /* Input elements in matrix from user */
```

```
    printf("Enter elements in matrix of size %dx%d: \n", SIZE, SIZE);
```

```
    for(row=0; row<SIZE; row++)
```

```
    {
```

```

    for(col=0; col<SIZE; col++)
    {
        scanf("%d", &A[row][col]);
    }
}

/* Find sum of main diagonal elements */
for(row=0; row<SIZE; row++)
{
    sum = sum + A[row][row];
}

printf("\nSum of main diagonal elements = %d", sum);

return 0;
}

```

Write a C program to read elements in a matrix and find the sum of elements of each row and columns of matrix. C program to calculate sum of rows and columns of matrix. Logic to find sum of each row and columns of a matrix in C programming.

Example

Input

Input elements in array:

1 2 3

4 5 6

7 8 9

Output

Sum of row 1 = 6

Sum of row 2 = 15

...

...

Sum of column 3 = 18

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

```
#define SIZE 3 // Matrix size
```

```
int main()
```

```
{
```

```
    int A[SIZE][SIZE];
```

```
    int row, col, sum = 0;
```

```
    /* Input elements in matrix from user */
```

```
    printf("Enter elements in matrix of size %dx%d: \n", SIZE, SIZE);
```

```
    for(row=0; row<SIZE; row++)
```

```
    {
```

```
        for(col=0; col<SIZE; col++)
```

```
        {
```

```
            scanf("%d", &A[row][col]);
```

```
        }
```

```
    }
```

```
/* Calculate sum of elements of each row of matrix */
```

```
for(row=0; row<SIZE; row++)
```

```
{
```

```
    sum = 0;
```

```
    for(col=0; col<SIZE; col++)
```

```
    {
```

```
        sum += A[row][col];
```

```
    }
```

```
    printf("Sum of elements of Row %d = %d\n", row+1, sum);
```

```
}
```

```
/* Find sum of elements of each columns of matrix */
```

```
for(row=0; row<SIZE; row++)
```

```
{
```

```
    sum = 0;
```

```
    for(col=0; col<SIZE; col++)
```

```
    {
```

```
        sum += A[col][row];
```

```
    }
```

```
    printf("Sum of elements of Column %d = %d\n", row+1, sum);
```

```
}
```

```
return 0;
```



```
}
```

Write a C program to read elements in a matrix and interchange elements of primary(major) diagonal with secondary(minor) diagonal. C program for interchanging diagonals of a matrix. Logic to interchange diagonals of a matrix in C programming.

Example

Input

Input matrix elements:

1 2 3

4 5 6

7 8 9

Output

Matrix after interchanging its diagonal:

3 2 1

4 5 6

9 8 7

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

```
#define MAX_ROWS 3
```

```
#define MAX_COLS 3
```

```
int main()
```

```
{
```

```
    int A[MAX_ROWS][MAX_COLS];
```

```
    int row, col, size, temp;
```

```
    /* Input elements in matrix from user */
```

```
printf("Enter elements in matrix of size %dx%d: \n", MAX_ROWS, MAX_COLS);
```

```
for(row=0; row<MAX_ROWS; row++)
```

```
{
```

```
    for(col=0; col<MAX_COLS; col++)
```

```
    {
```

```
        scanf("%d", &A[row][col]);
```

```
    }
```

```
}
```

```
size = (MAX_ROWS < MAX_COLS) ? MAX_ROWS : MAX_COLS;
```

```
/*
```

```
 * Interchange diagonal of the matrix
```

```
*/
```

```
for(row=0; row<size; row++)
```

```
{
```

```
    col = row;
```

```
    temp = A[row][col];
```

```
    A[row][col] = A[row][(size-col) - 1];
```

```
    A[row][(size-col) - 1] = temp;
```

```
}
```

```
/*
```

```

    * Print the interchanged diagonals matrix
    */

    printf("\nMatrix after diagonals interchanged: \n");
    for(row=0; row<MAX_ROWS; row++)
    {
        for(col=0; col<MAX_COLS; col++)
        {
            printf("%d ", A[row][col]);

        }

        printf("\n");
    }

    return 0;
}

```

9. Functions in C(Very important)

➤ Return Type function

Example for Function without argument and return value

Example 1

```
#include<stdio.h>
void printName();
void main ()
{
    printf("Hello ");
    printName();
}
void printName()
{
    printf("Javatpoint");
}
```

Output

Hello Javatpoint

Example 2

```
#include<stdio.h>
void sum();
void main()
{
    printf("\nGoing to calculate the sum of two numbers:");
    sum();
}
void sum()
{
    int a,b;
```

```

    printf("\nEnter two numbers");
    scanf("%d %d",&a,&b);
    printf("The sum is %d",a+b);
}

```

Output

Going to calculate the sum of two numbers:

Enter two numbers 10

24

The sum is 34

Example for Function without argument and with return value

Example 1

```

#include<stdio.h>
int sum();
void main()
{
    int result;
    printf("\nGoing to calculate the sum of two numbers:");
    result = sum();
    printf("%d",result);
}
int sum()
{
    int a,b;
    printf("\nEnter two numbers");
    scanf("%d %d",&a,&b);
    return a+b;
}

```

Output

Going to calculate the sum of two numbers:

Enter two numbers 10

24

The sum is 34

Example 2: program to calculate the area of the square

```
#include<stdio.h>
int sum();
void main()
{
    printf("Going to calculate the area of the square\n");
    float area = square();
    printf("The area of the square: %f\n",area);
}
int square()
{
    float side;
    printf("Enter the length of the side in meters: ");
    scanf("%f",&side);
    return side * side;
}
```

Output

Going to calculate the area of the square

Enter the length of the side in meters: 10

The area of the square: 100.000000

Write a C program to input any number from user and find cube of the given number using function. How to find cube of a given number using function in C programming. Write a C function to find cube of a given number.

Example

Input

Input any number: 5

Output

Cube of 5 = 125

```
#include <stdio.h>

/* Function declaration */
double cube(double num);

int main()
{
    int num;
    double c;

    /* Input number to find cube from user */
    printf("Enter any number: ");
    scanf("%d", &num);

    c = cube(num);

    printf("Cube of %d is %.2f", num, c);

    return 0;
}

/**
 * Function to find cube of any number
 */
double cube(double num)
{
    return (num * num * num);
}
```

Write a C program to input radius of circle from user and find diameter, circumference and area of the given circle using function. How to find diameter, circumference and area of a circle using function in C programming.

Example

Input

Input radius: 10

Output

Diameter = 20 units

Circumference = 62.83 units

Area = 314.16 sq. units

```
#include <stdio.h>
#include <math.h> // Used for constant PI referred as M_PI

/* Function declaration */
double getDiameter(double radius);
double getCircumference(double radius);
double getArea(double radius);

int main()
{
    float radius, dia, circ, area;

    /* Input radius of circle from user */
    printf("Enter radius of circle: ");
    scanf("%f", &radius);

    dia = getDiameter(radius);    // Call getDiameter function
    circ = getCircumference(radius); // Call getCircumference function
    area = getArea(radius);      // Call getArea function

    printf("Diameter of the circle = %.2f units\n", dia);
    printf("Circumference of the circle = %.2f units\n", circ);
    printf("Area of the circle = %.2f sq. units", area);

    return 0;
```



```
}
```

```
/**  
 * Calculate diameter of circle whose radius is given  
 */  
double getDiameter(double radius)  
{  
    return (2 * radius);  
}
```

```
/**  
 * Calculate circumference of circle whose radius is given  
 */  
double getCircumference(double radius)  
{  
    return (2 * M_PI * radius); // M_PI = PI = 3.14 ...  
}
```

```
/**  
 * Find area of circle whose radius is given  
 */  
double getArea(double radius)  
{  
    return (M_PI * radius * radius); // M_PI = PI = 3.14 ...  
}
```

Write a C program to input two or more numbers from user and find maximum and minimum of the given numbers using functions. How to find maximum and minimum of two or more numbers using functions in C programming.

Example

Input

Input two numbers: 10

20

Output

Maximum = 20

Minimum = 10

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

```
/* Function declarations */  
int max(int num1, int num2);  
int min(int num1, int num2);
```

```
int main()  
{  
    int num1, num2, maximum, minimum;
```

```
    /* Input two numbers from user */  
    printf("Enter any two numbers: ");  
    scanf("%d%d", &num1, &num2);
```

```
    maximum = max(num1, num2); // Call maximum function  
    minimum = min(num1, num2); // Call minimum function
```

```
    printf("\nMaximum = %d\n", maximum);  
    printf("Minimum = %d", minimum);
```

```
    return 0;  
}
```

```
/**  
 * Find maximum between two numbers.  
 */  
int max(int num1, int num2)  
{
```

```

    return (num1 > num2 ) ? num1 : num2;
}

/**
 * Find minimum between two numbers.
 */
int min(int num1, int num2)
{
    return (num1 > num2 ) ? num2 : num1;
}

```

Write a function in C programming to find prime numbers using function. How to find all prime numbers between two intervals using functions. Display all prime numbers between a given range using function in C programming.

Example

Input

Input lower limit: 10

Input upper limit: 50

Output

Prime numbers between 10-50 are: 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47

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

```

/* Function declarations */
int isPrime(int num);
void printPrimes(int lowerLimit, int upperLimit);

```

```

int main()
{
    int lowerLimit, upperLimit;

```

```

printf("Enter the lower and upper limit to list primes: ");
scanf("%d%d", &lowerLimit, &upperLimit);

// Call function to print all primes between the given range.
printPrimes(lowerLimit, upperLimit);

return 0;
}

/**
 * Print all prime numbers between lower limit and upper limit.
 */
void printPrimes(int lowerLimit, int upperLimit)
{
    printf("All prime number between %d to %d are: ", lowerLimit, upperLimit);

    while(lowerLimit <= upperLimit)
    {
        // Print if current number is prime.
        if(isPrime(lowerLimit))
        {
            printf("%d, ", lowerLimit);
        }

        lowerLimit++;
    }
}

/**
 * Check whether a number is prime or not.
 * Returns 1 if the number is prime otherwise 0.
 */
int isPrime(int num)

```

```

{
    int i;

    for(i=2; i<=num/2; i++)
    {
        /*
         * If the number is divisible by any number
         * other than 1 and self then it is not prime
         */
        if(num % i == 0)
        {
            return 0;
        }
    }

    return 1;
}

```

Write a function to print all Armstrong numbers between given interval in C programming. How to print all Armstrong numbers in given range using functions in C programming. C function to print all Armstrong numbers from 1 to n.

Example

Input

Input lower limit: 1

Input upper limit: 1000

Output

Armstrong numbers between 1 to 1000 are: 1, 153, 370, 371, 407,

```

#include <stdio.h>

/* Function declarations */
int isArmstrong(int num);
void printArmstrong(int start, int end);

```

```

int main()
{
    int start, end;

    /* Input lower and upper limit to of armstrong numbers */
    printf("Enter lower limit to print armstrong numbers: ");
    scanf("%d", &start);
    printf("Enter upper limit to print armstrong numbers: ");
    scanf("%d", &end);

    printf("All armstrong numbers between %d to %d are: \n", start, end);
    printArmstrong(start, end);

    return 0;
}

/**
 * Check whether the given number is armstrong number or not.
 * Returns 1 if the number is armstrong otherwise 0.
 */
int isArmstrong(int num)
{
    int temp, lastDigit, sum;

    temp = num;
    sum = 0;

    /* Calculate sum of cube of digits */
    while(temp != 0)
    {
        lastDigit = temp % 10;
        sum += lastDigit * lastDigit * lastDigit;
        temp /= 10;
    }
}

```

```

/*
 * Check if sum of cube of digits equals
 * to original number.
 */
if(num == sum)
    return 1;
else
    return 0;
}

/**
 * Print all armstrong numbers between start and end.
 */
void printArmstrong(int start, int end)
{
    /*
     * Iterates from start to end and print the current number
     * if it is armstrong
     */
    while(start <= end)
    {
        if(isArmstrong(start))
        {
            printf("%d, ", start);
        }

        start++;
    }
}

```

➤ **Printf function**

Example for Function with argument and without return value

Example 1

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

```

void sum(int, int);
void main()
{
    int a,b,result;
    printf("\nGoing to calculate the sum of two numbers:");
    printf("\nEnter two numbers:");
    scanf("%d %d",&a,&b);
    sum(a,b);
}
void sum(int a, int b)
{
    printf("\nThe sum is %d",a+b);
}

```

Output

Going to calculate the sum of two numbers:

Enter two numbers 10

24

The sum is 34

Example 2: program to calculate the average of five numbers.

1. #include<stdio.h>
2. void average(int, int, int, int, int);
3. void main()
4. {
5. int a,b,c,d,e;
6. printf("\nGoing to calculate the average of five numbers:");
7. printf("\nEnter five numbers:");
8. scanf("%d %d %d %d %d",&a,&b,&c,&d,&e);
9. average(a,b,c,d,e);
10. }
11. void average(int a, int b, int c, int d, int e)


```

12.{
13. float avg;
14. avg = (a+b+c+d+e)/5;
15. printf("The average of given five numbers : %f",avg);
16.}

```

Output

Going to calculate the average of five numbers:

Enter five numbers:10

20

30

40

50

The average of given five numbers : 30.000000

Example for Function with argument and with return value

Example 1

```

1. #include<stdio.h>
2. int sum(int, int);
3. void main()
4. {
5.     int a,b,result;
6.     printf("\nGoing to calculate the sum of two numbers:");
7.     printf("\nEnter two numbers:");
8.     scanf("%d %d",&a,&b);
9.     result = sum(a,b);
10.    printf("\nThe sum is : %d",result);
11.}
12.int sum(int a, int b)
13.{
14.    return a+b;
15.}

```

Output

Going to calculate the sum of two numbers:

Enter two numbers:10

20

The sum is : 30

Example 2: Program to check whether a number is even or odd

```
1. #include<stdio.h>
2. int even_odd(int);
3. void main()
4. {
5.   int n,flag=0;
6.   printf("\nGoing to check whether a number is even or odd");
7.   printf("\nEnter the number: ");
8.   scanf("%d",&n);
9.   flag = even_odd(n);
10.  if(flag == 0)
11.  {
12.    printf("\nThe number is odd");
13.  }
14.  else
15.  {
16.    printf("\nThe number is even");
17.  }
18.}
19.int even_odd(int n)
20.{
21.  if(n%2 == 0)
22.  {
23.    return 1;
24.  }
25.  else
26.  {
27.    return 0;
28.  }
```

29.}

Output

```
Going to check whether a number is even or odd
Enter the number: 100
The number is even
```

➤ Call by value

Call by value in C

- In call by value method, the value of the actual parameters is copied into the formal parameters. In other words, we can say that the value of the variable is used in the function call in the call by value method.
- In call by value method, we can not modify the value of the actual parameter by the formal parameter.
- In call by value, different memory is allocated for actual and formal parameters since the value of the actual parameter is copied into the formal parameter.
- The actual parameter is the argument which is used in the function call whereas formal parameter is the argument which is used in the function definition.

Let's try to understand the concept of call by value in c language by the example given below:

```
1. #include<stdio.h>
2. void change(int num) {
3.     printf("Before adding value inside function num=%d \n",num);
4.     num=num+100;
5.     printf("After adding value inside function num=%d \n", num);
6. }
7. int main() {
8.     int x=100;
9.     printf("Before function call x=%d \n", x);
10.    change(x); //passing value in function
```

```
11. printf("After function call x=%d \n", x);
12. return 0;
13.}
```

Output

```
Before function call x=100
Before adding value inside function num=100
After adding value inside function num=200
After function call x=100
```

call by Value Example: Swapping the values of the two variables

```
1. #include <stdio.h>
2. void swap(int , int); //prototype of the function
3. int main()
4. {
5.     int a = 10;
6.     int b = 20;
7.     printf("Before swapping the values in main a = %d, b = %d\n",a,b); // printing
    the value of a and b in main
8.     swap(a,b);
9.     printf("After swapping values in main a = %d, b = %d\n",a,b); // The value of a
    ctual parameters do not change by changing the formal parameters in call by va
    lue, a = 10, b = 20
10.}
11. void swap (int a, int b)
12. {
13.     int temp;
14.     temp = a;
15.     a=b;
16.     b=temp;
17.     printf("After swapping values in function a = %d, b = %d\n",a,b); // Formal pa
    rameters, a = 20, b = 10
18.}
```

Output

```
Before swapping the values in main a = 10, b = 20
After swapping values in function a = 20, b = 10
```

After swapping values in main a = 10, b = 20

➤ Call by reference

Call by reference in C

- In call by reference, the address of the variable is passed into the function call as the actual parameter.
- The value of the actual parameters can be modified by changing the formal parameters since the address of the actual parameters is passed.
- In call by reference, the memory allocation is similar for both formal parameters and actual parameters. All the operations in the function are performed on the value stored at the address of the actual parameters, and the modified value gets stored at the same address.

Consider the following example for the call by reference.

```
1. #include<stdio.h>
2. void change(int *num) {
3.     printf("Before adding value inside function num=%d \n",*num);
4.     (*num) += 100;
5.     printf("After adding value inside function num=%d \n", *num);
6. }
7. int main() {
8.     int x=100;
9.     printf("Before function call x=%d \n", x);
10.    change(&x);//passing reference in function
11.    printf("After function call x=%d \n", x);
12. return 0;
13.}
```

Output

```
Before function call x=100
Before adding value inside function num=100
After adding value inside function num=200
After function call x=200
```

Call by reference Example: Swapping the values of the two variables

```
1. #include <stdio.h>
2. void swap(int *, int *); //prototype of the function
3. int main()
4. {
5.     int a = 10;
6.     int b = 20;
7.     printf("Before swapping the values in main a = %d, b = %d\n",a,b); // printing
    the value of a and b in main
8.     swap(&a,&b);
9.     printf("After swapping values in main a = %d, b = %d\n",a,b); // The values of
    actual parameters do change in call by reference, a = 10, b = 20
10.}
11.void swap (int *a, int *b)
12.{
13.    int temp;
14.    temp = *a;
15.    *a=*b;
16.    *b=temp;
17.    printf("After swapping values in function a = %d, b = %d\n",*a,*b); // Formal
    parameters, a = 20, b = 10
18.}
```

Output

Before swapping the values in main a = 10, b = 20

After swapping values in function a = 20, b = 10

After swapping values in main a = 20, b = 10

Write a C program to input a number from user and find power of given number using recursion. How to find power of a number using recursive function in C programming. Logic to find power of a number using recursion in C programming.

Example

Input

Input base number: 5

Input power: 2

Output

$2^5 = 25$

```
#include <stdio.h>

/* Power function declaration */
double pow(double base, int expo);

int main()
{
    double base, power;
    int expo;

    /* Input base and exponent from user */
    printf("Enter base: ");
    scanf("%lf", &base);
    printf("Enter exponent: ");
    scanf("%d", &expo);

    // Call pow function
    power = pow(base, expo);

    printf("%.2lf ^ %d = %f", base, expo, power);

    return 0;
}

/**
 * Calculate power of any number.
 * Returns base ^ expo
 */
```

```

*/
double pow(double base, int expo)
{
    /* Base condition */
    if(expo == 0)
        return 1;
    else if(expo > 0)
        return base * pow(base, expo - 1);
    else
        return 1 / pow(base, -expo);
}

```

Write a recursive function in C programming to print all natural numbers between 1 to n. How to print all natural numbers from 1 to n using recursion in C program. Logic to print all natural numbers in given range using recursion in C programming.

Example

Input

Input lower limit: 1

Input upper limit: 10

Output

Natural numbers between 1 to 10: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10,

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

```

/* Function declaration */
void printNaturalNumbers(int lowerLimit, int upperLimit);

```

```

int main()
{

```



```

    int lowerLimit, upperLimit;

    /* Input lower and upper limit from user */
    printf("Enter lower limit: ");
    scanf("%d", &lowerLimit);
    printf("Enter upper limit: ");
    scanf("%d", &upperLimit);

    printf("All natural numbers from %d to %d are: ", lowerLimit, upperLimit);
    printNaturalNumbers(lowerLimit, upperLimit);

    return 0;
}

/**
 * Recursively prints all natural number between the given range.
 */
void printNaturalNumbers(int lowerLimit, int upperLimit)
{
    if(lowerLimit > upperLimit)
        return;

    printf("%d, ", lowerLimit);

    // Recursively call the function to print next number
    printNaturalNumbers(lowerLimit + 1, upperLimit);

}

```

Write a recursive function in C programming to print all even or odd numbers between 1 to n. How to print all even numbers in given range using recursion in C programming. Logic to print even/odd numbers in given range using recursion.

Example

Input

Input lower limit: 1
Input upper limit: 10

Output

Even numbers between 1 to 10: 2, 4, 6, 8, 10
Odd numbers between 1 to 10: 1, 3, 5, 7, 9

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

```
/* Function declaration */  
void printEvenOdd(int cur, int limit);
```

```
int main()  
{  
    int lowerLimit, upperLimit;
```

```
    // Input lower and upper limit from user  
    printf("Enter lower limit: ");  
    scanf("%d", &lowerLimit);  
    printf("Enter upper limit: ");  
    scanf("%d", &upperLimit);
```

```
    printf("Even/odd Numbers from %d to %d are: ", lowerLimit, upperLimit);  
    printEvenOdd(lowerLimit, upperLimit);
```

```
    return 0;  
}
```

```
/**  
 * Recursive function to print even or odd numbers in a given range.  
 */  
void printEvenOdd(int cur, int limit)  
{
```

```
if(cur > limit)
    return;
```

```
printf("%d, ", cur);
```

```
// Recursively call to printEvenOdd to get next value
printEvenOdd(cur + 2, limit);
```

```
}
```

Write a recursive function in C programming to find sum of all natural numbers between 1 to n. How to find sum of all natural numbers using recursion in C program. Logic to find sum of natural numbers in given range using recursion.

Example

Input

Input lower limit: 1

Input upper limit: 10

Output

Sum of natural numbers from 1 to 10 = 55

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

```
/* Function declaration */
int sumOfNaturalNumbers(int start, int end);
```

```
int main()
{
    int start, end, sum;
```

```
/* Input lower and upper limit from user */
printf("Enter lower limit: ");
```

```

scanf("%d", &start);
printf("Enter upper limit: ");
scanf("%d", &end);

sum = sumOfNaturalNumbers(start, end);

printf("Sum of natural numbers from %d to %d = %d", start, end, sum);

return 0;
}

/**
 * Recursively find the sum of natural number
 */
int sumOfNaturalNumbers(int start, int end)
{
    if(start == end)
        return start;
    else
        return start + sumOfNaturalNumbers(start + 1, end);
}

```

Write a recursive function in C to find sum of all even or odd numbers in a given range.
How to find sum of all even numbers between 1 to n using recursion in C programming.

Example

Input

Input lower limit: 1
Input upper limit: 100

Output

Sum of even numbers between 1 to 100 = 2550

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

```
int sumOfEvenOdd(int start, int end);
```

```
int main()
```

```
{
```

```
    int start, end, sum;
```

```
    /* Input lower and upper limit from user */
```

```
    printf("Enter lower limit: ");
```

```
    scanf("%d", &start);
```

```
    printf("Enter upper limit: ");
```

```
    scanf("%d", &end);
```

```
    printf("Sum of even/odd numbers between %d to %d = %d\n", start, end,  
sumOfEvenOdd(start, end));
```

```
    return 0;
```

```
}
```

```
/**
```

```
 * Find sum of all even or odd numbers recursively.
```

```
 */
```

```
int sumOfEvenOdd(int start, int end)
```

```
{
```

```
    /* Base condition */
```

```
    if(start > end)
```

```
        return 0;
```

```
    else
```

```
        return (start + sumOfEvenOdd(start + 2, end));
```

```
}
```

Write a recursive function in C programming to find reverse of a number. How to find reverse of a number in C programming using recursion. Logic to find reverse of a number using recursion in C programming.

Example

Input

Input number: 12345

Output

Reverse: 54321

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

```
/* Fuction declaration */
int reverse(int num);
```

```
int main()
{
    int num, rev;
```

```
    /* Input number from user */
    printf("Enter any number: ");
    scanf("%d", &num);
```

```
    /* Call the function to reverse number */
    rev = reverse(num);
```

```
    printf("Reverse of %d = %d", num, rev);
```

```
    return 0;
}
```

```

/**
 * Recursive function to find reverse of any number
 */
int reverse(int num)
{
    // Find total digits in num
    int digit = (int) log10(num);

    // Base condition
    if(num == 0)
        return 0;

    return ((num%10 * pow(10, digit)) + reverse(num/10));

}

```

Write a recursive function in C to check palindrome number. How to check whether a number is palindrome or not using recursion in C program. Logic to check palindrome number using recursion in C programming.

Example

Input

Input number: 121

Output

121 is palindrome

```

**
 * C program to check palindrome number using recursion
 */

#include <stdio.h>
#include <math.h>

```

```
/* Function declarations */  
int reverse(int num);  
int isPalindrome(int num);
```

```
int main()  
{  
    int num;
```

```
    /* Input any number from user */  
    printf("Enter any number: ");  
    scanf("%d", &num);
```

```
    if(isPalindrome(num) == 1)  
    {  
        printf("%d is palindrome number.\n", num);  
    }  
    else  
    {  
        printf("%d is NOT palindrome number.\n", num);  
    }
```

```
    return 0;  
}
```

```
/**  
 * Function to check whether a number is palindrome or not.  
 * This function returns 1 if the number is palindrome otherwise 0.  
 */  
int isPalindrome(int num)  
{  
    /*  
     * Check if the given number is equal to  
     * its reverse.
```



```

    */
    if(num == reverse(num))
    {
        return 1;
    }

    return 0;
}

/**
 * Recursive function to find reverse of any number
 */
int reverse(int num)
{
    /* Find number of digits in num */
    int digit = (int)log10(num);

    /* Recursion base condition */
    if(num == 0)
        return 0;

    return ((num%10 * pow(10, digit)) + reverse(num/10));

}

```

Write a recursive function in C programming to calculate sum of digits of a number. How to calculate sum of digits of a given number using recursion in C program. Logic to find sum of digits using recursion in C programming.

Example

Input

Input number: 1234

Output

Sum of digits: 10

```
* C program to calculate sum of digits using recursion
*/
```

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

```
/* Function declaration */
int sumOfDigits(int num);
```

```
int main()
{
    int num, sum;
```

```
    printf("Enter any number to find sum of digits: ");
    scanf("%d", &num);
```

```
    sum = sumOfDigits(num);
```

```
    printf("Sum of digits of %d = %d", num, sum);
```

```
    return 0;
}
```

```
/**
 * Recursive function to find sum of digits of a number
 */
```

```
int sumOfDigits(int num)
```

```
{
    // Base condition
    if(num == 0)
        return 0;
```

```
    return ((num % 10) + sumOfDigits(num / 10));
```

```
}
```

Write a recursive function in C to find factorial of a number. How to find factorial of a number using recursion in C program. Logic to find factorial of a number using recursion in C programming.

Example

Input

Input any number: 5

Output

Factorial of 5 = 120

```
#include <stdio.h>

/* Function declaration */
unsigned long long fact(int num);

int main()
{
    int num;
    unsigned long long factorial;

    /* Input an integer from user */
    printf("Enter any number: ");
    scanf("%d", &num);

    factorial = fact(num); // Call factorial function

    printf("Factorial of %d is %llu", num, factorial);

    return 0;
}
```

```

/**
 * Function to compute and return factorial of any number recursively.
 */
unsigned long long fact(int num)
{
    // Base condition
    if(num == 0)
        return 1;
    else
        return num * fact(num - 1);
}

```

Write a recursive function to generate n^{th} fibonacci term in C programming. How to generate n^{th} fibonacci term in C programming using recursion. Logic to find n^{th} Fibonacci term using recursion in C programming.

Example

Input

Input any number: 10

Output

10th Fibonacci term: 55

```

*
* C program to find nth Fibonacci term using recursion
*/

```

```

#include <stdio.h>

```

```

/* Function declaration */
unsigned long long fibo(int num);

```

```

int main()
{
    int num;
    unsigned long long fibonacci;

    /* Input a number from user */
    printf("Enter any number to find nth fibonacci term: ");
    scanf("%d", &num);

    fibonacci = fibo(num);

    printf("%d fibonacci term is %llu", num, fibonacci);

    return 0;
}

/**
 * Recursive function to find nth Fibonacci term
 */
unsigned long long fibo(int num)
{
    if(num == 0) //Base condition
        return 0;
    else if(num == 1) //Base condition
        return 1;
    else
        return fibo(num-1) + fibo(num-2);
}

```

10. library functions(inportant)

➤ library functions

C – Library functions

C Library functions:

Library functions in C language are inbuilt functions which are grouped together and placed in a common place called library.

Each library function in C performs specific operation.

We can make use of these library functions to get the pre-defined output instead of writing our own code to get those outputs.

These library functions are created by the persons who designed and created C compilers.

All C standard library functions are declared in many header files which are saved as file_name.h.

Actually, function declaration, definition for macros are given in all header files.

We are including these header files in our C program using “#include<file_name.h>” command to make use of the functions those are declared in the header files.

When we include header files in our C program using “#include<filename.h>” command, all C code of the header files are included in C program. Then, this C program is compiled by compiler and executed.

Please check the below links for actual C source code for the respective C header files.

[1. C – stdio.h source code](#)

[2. C – conio.h source code](#)

[3. C – string.h source code](#)

[4. C – stdlib.h source code](#)

[5. C – math.h source code](#)

6. C – time.h source code

7. C – ctype.h source code

If you want to check source code for all header files, you can check inside “include” directory after C compiler is installed in your machine.

For example, if you install DevC++ compiler in C directory in your machine, “C:\Dev-Cpp\include” is the path where all header files will be available.

LIST OF MOST USED HEADER FILES IN C PROGRAMMING LANGUAGE:

Check the below table to know all the C library functions and header files in which they are declared.

Click on the each header file name below to know the list of inbuilt functions declared inside them.

Header file	Description
<u>stdio.h</u>	This is standard input/output header file in which Input/Output functions are declared
<u>conio.h</u>	This is console input/output header file
<u>string.h</u>	All string related functions are defined in this header file
<u>stdlib.h</u>	This header file contains general functions used in C programs
<u>math.h</u>	All maths related functions are defined in this header file

<u>time.h</u>	This header file contains time and clock related functions
<u>ctype.h</u>	All character handling functions are defined in this header file
<u>stdarg.h</u>	Variable argument functions are declared in this header file
<u>signal.h</u>	Signal handling functions are declared in this file
<u>setjmp.h</u>	This file contains all jump functions
<u>locale.h</u>	This file contains locale functions
<u>errno.h</u>	Error handling functions are given in this file
<u>assert.h</u>	This contains diagnostics functions

Continue on C – User defined functions & adding them in C library....

Continue on C – Command line arguments....

Continue on C – Variable length arguments....

C – stdio.h library functions

C stdio.h library functions:

All C inbuilt functions which are declared in stdio.h header file are given below. The source code for stdio.h header file is also given below for your reference.

LIST OF INBUILT C FUNCTIONS IN STDIO.H FILE:

Function	Description
----------	-------------

printf()	This function is used to print the character, string, float, integer, octal and hexadecimal values onto the output screen
scanf()	This function is used to read a character, string, numeric data from keyboard.
getc()	It reads character from file
gets()	It reads line from keyboard
getchar()	It reads character from keyboard
puts()	It writes line to o/p screen
putchar()	It writes a character to screen
clearerr()	This function clears the error indicators
f open()	All file handling functions are defined in stdio.h header file
f close()	closes an opened file
getw()	reads an integer from file
putw()	writes an integer to file
f getc()	reads a character from file
fputc()	writes a character to file
f putc()	writes a character to file

f gets()	reads string from a file, one line at a time
f puts()	writes string to a file
f eof()	finds end of file
f getchar	reads a character from keyboard
f getc()	reads a character from file
f printf()	writes formatted data to a file
f scanf()	reads formatted data from a file
f getchar	reads a character from keyboard
f putchar	writes a character from keyboard
f seek()	moves file pointer position to given location
SEEK_SET	moves file pointer position to the beginning of the file
SEEK_CUR	moves file pointer position to given location
SEEK_END	moves file pointer position to the end of file.
f tell()	gives current position of file pointer
rewind()	moves file pointer position to the beginning of the file
putc()	writes a character to file
sprint()	writes formatted output to string

sscanf()	Reads formatted input from a string
remove()	deletes a file
fflush()	flushes a file

SOURCE CODE FOR STDIO.H HEADER FILE:

Please find below the source code for stdio.h header file. This code is taken from DevC++ compiler files for your reference.

C – conio.h library functions

C conio.h library functions:

All C inbuilt functions which are declared in conio.h header file are given below. The source code for conio.h header file is also given below for your reference.

LIST OF INBUILT C FUNCTIONS IN CONIO.H FILE:

Functions	Description
clrscr()	This function is used to clear the output screen.
getch()	It reads character from keyboard
getche()	It reads character from keyboard and echoes to o/p screen
textcolor()	This function is used to change the text color
textbackground()	This function is used to change text background

SOURCE CODE FOR CONIO.H HEADER FILE:

Please find the source code for conio.h header file below. This code is taken from DevC++ compiler files for your reference.

C – string.h library functions

C string.h library functions:

All C inbuilt functions which are declared in string.h header file are given below. The source code for string.h header file is also given below for your reference.

LIST OF INBUILT C FUNCTIONS IN STRING.H FILE:

String functions	Description
<u>strcat ()</u>	Concatenates str2 at the end of str1
<u>strncat ()</u>	Appends a portion of string to another
<u>strcpy ()</u>	Copies str2 into str1
<u>strncpy ()</u>	Copies given number of characters of one string to another
<u>strlen ()</u>	Gives the length of str1
<u>strcmp ()</u>	Returns 0 if str1 is same as str2. Returns <0 if str1 < str2. Returns >0 if str1 > str2
<u>strcmpi ()</u>	Same as strcmp() function. But, this function negotiates case. “A” and “a” are treated as same.

<u>strchr ()</u>	Returns pointer to first occurrence of char in str1
<u>strrchr ()</u>	last occurrence of given character in a string is found
<u>strstr ()</u>	Returns pointer to first occurrence of str2 in str1
<u>strrstr ()</u>	Returns pointer to last occurrence of str2 in str1
<u>strdup ()</u>	Duplicates the string
<u>strlwr ()</u>	Converts string to lowercase
<u>strupr ()</u>	Converts string to uppercase
<u>strrev ()</u>	Reverses the given string
<u>strset ()</u>	Sets all character in a string to given character
<u>strnset ()</u>	It sets the portion of characters in a string to given character
<u>strtok ()</u>	Tokenizing given string using delimiter
string functions	Description
<u>memset()</u>	It is used to initialize a specified number of bytes to null or any other value in the buffer
<u>memcpy()</u>	It is used to copy a specified number of bytes from one memory to another

<u>memmove()</u>	It is used to copy a specified number of bytes from one memory to another or to overlap on same memory.
<u>memcmp()</u>	It is used to compare specified number of characters from two buffers
<u>memicmp()</u>	It is used to compare specified number of characters from two buffers regardless of the case of the characters
<u>memchr()</u>	It is used to locate the first occurrence of the character in the specified string

SOURCE CODE FOR STRING.H HEADER FILE:

Please find the source code for string.h header file below. This code is taken from DevC++ compiler files for your reference.

C – stdlib.h library functions

C stdlib.h library functions:

All C inbuilt functions which are declared in stdlib.h header file are given below. The source code for stdlib.h header file is also given below for your reference.

LIST OF INBUILT C FUNCTIONS IN STDLIB.H FILE:

Function	Description
----------	-------------

malloc()	This function is used to allocate space in memory during the execution of the program.
calloc()	This function is also like malloc () function. But calloc () initializes the allocated memory to zero. But, malloc() doesn't
realloc()	This function modifies the allocated memory size by malloc () and calloc () functions to new size
free()	This function frees the allocated memory by malloc (), calloc (), realloc () functions and returns the memory to the system.
abs()	This function returns the absolute value of an integer . The absolute value of a number is always positive. Only integer values are supported in C.
div()	This function performs division operation
abort()	It terminates the C program
exit()	This function terminates the program and does not return any value
system()	This function is used to execute commands outside the C program.
atoi()	Converts string to int

atol()	Converts string to long
atof()	Converts string to float
strtod()	Converts string to double
strtol()	Converts string to long
getenv()	This function gets the current value of the environment variable
setenv()	This function sets the value for environment variable
putenv()	This function modifies the value for environment variable
perror()	This function displays most recent error that happened during library function call.
rand()	This function returns the random integer numbers
delay()	This function Suspends the execution of the program for particular time

SOURCE CODE FOR STDLIB.H HEADER FILE:

Please find the source code for stdlib.h header file below. This code is taken from DevC++ compiler files for your reference.

C – math.h library functions

C math.h library functions:

All C inbuilt functions which are declared in math.h header file are given below. The source code for math.h header file is also given below for your reference.

LIST OF INBUILT C FUNCTIONS IN MATH.H FILE:

“math.h” header file supports all the mathematical related functions in C language. All the arithmetic functions used in C language are given below.

Click on each function name below for detail description and example programs.

Function	Description
<u>floor ()</u>	This function returns the nearest integer which is less than or equal to the argument passed to this function.
<u>round ()</u>	This function returns the nearest integer value of the float/double/long double argument passed to this function. If decimal value is from “.1 to .5”, it returns integer value less than the argument. If decimal value is from “.6 to .9”, it returns the integer value greater than the argument.
<u>ceil ()</u>	This function returns nearest integer value which is greater than or equal to the argument passed to this function.
<u>sin ()</u>	This function is used to calculate sine value.
<u>cos ()</u>	This function is used to calculate cosine.
<u>cosh ()</u>	This function is used to calculate hyperbolic cosine.
<u>exp ()</u>	This function is used to calculate the exponential “e” to the x th power.

<u>tan ()</u>	This function is used to calculate tangent.
<u>tanh ()</u>	This function is used to calculate hyperbolic tangent.
<u>sinh ()</u>	This function is used to calculate hyperbolic sine.
<u>log ()</u>	This function is used to calculates natural logarithm.
<u>log10 ()</u>	This function is used to calculates base 10 logarithm.
<u>sqrt ()</u>	This function is used to find square root of the argument passed to this function.
<u>pow ()</u>	This is used to find the power of the given number.
<u>trunc.(.)</u>	This function truncates the decimal value from floating point value and returns integer value.

SOURCE CODE FOR MATH.H HEADER FILE:

Please find the source code for math.h header file below. This code is taken from DevC++ compiler files for your reference.

C – time.h library functions

C time.h library functions:

All C inbuilt functions which are declared in time.h header file are given below. The source code for time.h header file is also given below for your reference.

LIST OF INBUILT C FUNCTIONS IN TIME.H FILE:

Below are the list of time related fuctions in C programming language. Please refer C – time related functions page for sample program and output for each below functions.

Functions	Description
setdate()	This function used to modify the system date
getdate()	This function is used to get the CPU time
clock()	This function is used to get current system time
time()	This function is used to get current system time as structure
difftime()	This function is used to get the difference between two given times
strftime()	This function is used to modify the actual time format
mktime()	This function interprets tm structure as calendar time
localtime()	This function shares the tm structure that contains date and time informations
gmtime()	This function shares the tm structure that contains date and time informations
ctime()	This function is used to return string that contains date and time informations
asctime()	Tm structure contents are interpreted by this function as calendar time. This time is converted into string.

SOURCE CODE FOR TIME.H HEADER FILE:

Please find the source code for time.h header file below. This code is taken from DevC++ compiler files for your reference.

C – ctype.h library functions

C ctype.h library functions:

All C inbuilt functions which are declared in ctype.h header file are given below. The source code for ctype.h header file is also given below for your reference.

LIST OF INBUILT C FUNCTIONS IN CTYPE.H FILE:

“ctype.h” header file support all the below functions in C language. Click on each function name below for detail description and example programs.

Functions	Description
<u>isalpha()</u>	checks whether character is alphabetic
<u>isdigit()</u>	checks whether character is digit
<u>isalnum()</u>	Checks whether character is alphanumeric
<u>isspace()</u>	Checks whether character is space
<u>islower()</u>	Checks whether character is lower case
<u>isupper()</u>	Checks whether character is upper case
<u>isxdigit()</u>	Checks whether character is hexadecimal

<u>iscntrl()</u>	Checks whether character is a control character
<u>isprint()</u>	Checks whether character is a printable character
<u>ispunct()</u>	Checks whether character is a punctuation
<u>isgraph()</u>	Checks whether character is a graphical character
<u>tolower()</u>	Checks whether character is alphabetic & converts to lower case
<u>toupper()</u>	Checks whether character is alphabetic & converts to upper case

SOURCE CODE FOR CTYPE.H HEADER FILE:

Please find the source code for ctype.h header file below. This code is taken from DevC++ compiler files for your reference.

11. Recursion(inportant)

➤ Recursion

Recursion in C

Recursion is the process which comes into existence when a function calls a copy of itself to work on a smaller problem. Any function which calls itself is called recursive function, and such function calls are called recursive calls. Recursion involves several numbers of recursive calls. However, it is important to impose a termination condition of recursion. Recursion code is shorter than iterative code however it is difficult to understand.

Recursion cannot be applied to all the problem, but it is more useful for the tasks that can be defined in terms of similar subtasks. For Example, recursion may be applied to sorting, searching, and traversal problems.

Generally, iterative solutions are more efficient than recursion since function call is always overhead. Any problem that can be solved recursively, can also be solved iteratively. However, some problems are best suited to be solved by the recursion, for example, tower of Hanoi, Fibonacci series, factorial finding, etc.

In the following example, recursion is used to calculate the factorial of a number.

```
#include <stdio.h>

int fact (int);

int main()
{
    int n,f;

    printf("Enter the number whose factorial you want to calculate?");

    scanf("%d",&n);

    f = fact(n);

    printf("factorial = %d",f);
```

```

}
int fact(int n)
{
    if (n==0)
    {
        return 0;
    }
    else if ( n == 1)
    {
        return 1;
    }
    else
    {
        return n*fact(n-1);
    }
}

```

Output

Enter the number whose factorial you want to calculate?5

factorial = 120

We can understand the above program of the recursive method call by the figure given below:

```

return 5 * factorial(4) = 120
└─ return 4 * factorial(3) = 24
    └─ return 3 * factorial(2) = 6
        └─ return 2 * factorial(1) = 2
            └─ return 1 * factorial(0) = 1

```

javaTpoint.com

$1 * 2 * 3 * 4 * 5 = 120$

Fig: Recursion

Recursive Function

A recursive function performs the tasks by dividing it into the subtasks. There is a termination condition defined in the function which is satisfied by some specific subtask. After this, the recursion stops and the final result is returned from the function.

The case at which the function doesn't recur is called the base case whereas the instances where the function keeps calling itself to perform a subtask, is called the recursive case. All the recursive functions can be written using this format.

Pseudocode for writing any recursive function is given below.

```

if (test_for_base)
{
    return some_value;
}
else if (test_for_another_base)
{
    return some_another_value;
}

```



```
}  
else  
{  
    // Statements;  
    recursive call;  
}
```

Example of recursion in C

Let's see an example to find the nth term of the Fibonacci series.

```
#include<stdio.h>  
  
int fibonacci(int);  
  
void main ()  
{  
    int n,f;  
    printf("Enter the value of n?");  
    scanf("%d",&n);  
    f = fibonacci(n);  
    printf("%d",f);  
}  
  
int fibonacci (int n)  
{  
    if (n==0)  
    {  
        return 0;  
    }  
}
```

```

else if (n == 1)
{
    return 1;
}
else
{
    return fibonacci(n-1)+fibonacci(n-2);
}
}

```

Output

Enter the value of n?12

144

Memory allocation of Recursive method

Each recursive call creates a new copy of that method in the memory. Once some data is returned by the method, the copy is removed from the memory. Since all the variables and other stuff declared inside function get stored in the stack, therefore a separate stack is maintained at each recursive call. Once the value is returned from the corresponding function, the stack gets destroyed. Recursion involves so much complexity in resolving and tracking the values at each recursive call. Therefore we need to maintain the stack and track the values of the variables defined in the stack.

Let us consider the following example to understand the memory allocation of the recursive functions.

```

int display (int n)
{
    if(n == 0)
        return 0; // terminating condition
}

```

```

else
{
    printf("%d",n);
    return display(n-1); // recursive call
}
}

```

Example: Sum of Natural Numbers Using Recursion

```

#include <stdio.h>

int sum(int n);

int main() {
    int number, result;

    printf("Enter a positive integer: ");
    scanf("%d", &number);

    result = sum(number);

    printf("sum = %d", result);
    return 0;
}

int sum(int n) {
    if (n != 0)

```

```

        // sum() function calls itself
        return n + sum(n-1);
    else
        return n;
}

```

Output

Enter a positive integer:3

sum = 6

Number Factorial

The following example calculates the factorial of a given number using a recursive function –

```

#include <stdio.h>

unsigned long long int factorial(unsigned int i) {

    if(i <= 1) {
        return 1;
    }
    return i * factorial(i - 1);
}

int main() {
    int i = 12;
    printf("Factorial of %d is %d\n", i, factorial(i));
    return 0;
}

```

```
}
```

When the above code is compiled and executed, it produces the following result –

Factorial of 12 is 479001600

Fibonacci Series

The following example generates the Fibonacci series for a given number using a recursive function –

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

```
int fibonacci(int i) {
```

```
    if(i == 0) {
```

```
        return 0;
```

```
    }
```

```
    if(i == 1) {
```

```
        return 1;
```

```
    }
```

```
    return fibonacci(i-1) + fibonacci(i-2);
```

```
}
```

```
int main() {
```

```
    int i;
```

```
    for (i = 0; i < 10; i++) {
```

```
    printf("%d\t\n", fibonacci(i));  
}
```

```
    return 0;  
}
```

When the above code is compiled and executed, it produces the following result –

```
0  
1  
1  
2  
3  
5  
8  
13  
21  
34
```

12. Pointers

➤ Pointers

Pointers in C are easy and fun to learn. Some C programming tasks are performed more easily with pointers, and other tasks, such as dynamic memory allocation, cannot be performed without using pointers. So it becomes necessary to learn pointers to become a perfect C programmer. Let's start learning them in simple and easy steps.

As you know, every variable is a memory location and every memory location has its address defined which can be accessed using ampersand (&) operator, which denotes an address in memory. Consider the following example, which prints the address of the variables defined –

```
#include <stdio.h>

int main () {

    int var1;
    char var2[10];

    printf("Address of var1 variable: %x\n", &var1 );
    printf("Address of var2 variable: %x\n", &var2 );

    return 0;
}
```

When the above code is compiled and executed, it produces the following result –

Address of var1 variable: bff5a400

Address of var2 variable: bff5a3f6

What are Pointers?

A pointer is a variable whose value is the address of another variable, i.e., direct address of the memory location. Like any variable or constant, you must declare a pointer before using it to store any variable address. The general form of a pointer variable declaration is –

type *var-name;

Here, type is the pointer's base type; it must be a valid C data type and var-name is the name of the pointer variable. The asterisk * used to declare a pointer is the same asterisk used for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer. Take a look at some of the valid pointer declarations –

```
int *ip; /* pointer to an integer */
double *dp; /* pointer to a double */
float *fp; /* pointer to a float */
char *ch /* pointer to a character */
```

The actual data type of the value of all pointers, whether integer, float, character, or otherwise, is the same, a long hexadecimal number that represents a memory address. The only difference between pointers of different data types is the data type of the variable or constant that the pointer points to.

How to Use Pointers?

There are a few important operations, which we will do with the help of pointers very frequently. (a) We define a pointer variable, (b) assign the address of a variable to a pointer and (c) finally access the value at the address available in the pointer variable. This is done by using unary operator * that returns the value of the variable located at the address specified by its operand. The following example makes use of these operations –

```
#include <stdio.h>

int main () {

    int var = 20; /* actual variable declaration */
    int *ip;      /* pointer variable declaration */

    ip = &var; /* store address of var in pointer variable*/

    printf("Address of var variable: %x\n", &var );

    /* address stored in pointer variable */
    printf("Address stored in ip variable: %x\n", ip );
```



```
/* access the value using the pointer */
printf("Value of *ip variable: %d\n", *ip );

return 0;
}
```

When the above code is compiled and executed, it produces the following result –

Address of var variable: bffd8b3c

Address stored in ip variable: bffd8b3c

Value of *ip variable: 20

NULL Pointers

It is always a good practice to assign a NULL value to a pointer variable in case you do not have an exact address to be assigned. This is done at the time of variable declaration. A pointer that is assigned NULL is called a **null** pointer.

The NULL pointer is a constant with a value of zero defined in several standard libraries. Consider the following program –

```
#include <stdio.h>

int main () {

    int *ptr = NULL;

    printf("The value of ptr is : %x\n", ptr );

    return 0;
}
```

When the above code is compiled and executed, it produces the following result –

The value of ptr is 0

In most of the operating systems, programs are not permitted to access memory at address 0 because that memory is reserved by the operating system. However, the memory address 0 has special significance; it signals that the pointer is not intended

to point to an accessible memory location. But by convention, if a pointer contains the null (zero) value, it is assumed to point to nothing.

To check for a null pointer, you can use an 'if' statement as follows –

```
if(ptr) /* succeeds if p is not null */  
if(!ptr) /* succeeds if p is null */
```

A pointer in c is an address, which is a numeric value. Therefore, you can perform arithmetic operations on a pointer just as you can on a numeric value. There are four arithmetic operators that can be used on pointers: ++, --, +, and -

To understand pointer arithmetic, let us consider that **ptr** is an integer pointer which points to the address 1000. Assuming 32-bit integers, let us perform the following arithmetic operation on the pointer –

```
ptr++
```

After the above operation, the **ptr** will point to the location 1004 because each time ptr is incremented, it will point to the next integer location which is 4 bytes next to the current location. This operation will move the pointer to the next memory location without impacting the actual value at the memory location. If **ptr** points to a character whose address is 1000, then the above operation will point to the location 1001 because the next character will be available at 1001.

Incrementing a Pointer

We prefer using a pointer in our program instead of an array because the variable pointer can be incremented, unlike the array name which cannot be incremented because it is a constant pointer. The following program increments the variable pointer to access each succeeding element of the array –

```
#include <stdio.h>  
  
const int MAX = 3;  
  
int main () {  
  
    int var[] = {10, 100, 200};  
    int i, *ptr;  
  
    /* let us have array address in pointer */  
    ptr = var;
```

```

for ( i = 0; i < MAX; i++) {

    printf("Address of var[%d] = %x\n", i, ptr );
    printf("Value of var[%d] = %d\n", i, *ptr );

    /* move to the next location */
    ptr++;
}

return 0;
}

```

When the above code is compiled and executed, it produces the following result –

```

Address of var[0] = bf882b30
Value of var[0] = 10
Address of var[1] = bf882b34
Value of var[1] = 100
Address of var[2] = bf882b38
Value of var[2] = 200
Decrementing a Pointer

```

The same considerations apply to decrementing a pointer, which decreases its value by the number of bytes of its data type as shown below –

```

#include <stdio.h>

const int MAX = 3;

int main () {

    int var[] = {10, 100, 200};
    int i, *ptr;

    /* let us have array address in pointer */
    ptr = &var[MAX-1];

    for ( i = MAX; i > 0; i--) {

```

```

printf("Address of var[%d] = %x\n", i-1, ptr );
printf("Value of var[%d] = %d\n", i-1, *ptr );

/* move to the previous location */
ptr--;
}

return 0;
}

```

When the above code is compiled and executed, it produces the following result –

Address of var[2] = bfeedbcd8

Value of var[2] = 200

Address of var[1] = bfeedbcd4

Value of var[1] = 100

Address of var[0] = bfeedbcd0

Value of var[0] = 10

Pointer Comparisons

Pointers may be compared by using relational operators, such as ==, <, and >. If p1 and p2 point to variables that are related to each other, such as elements of the same array, then p1 and p2 can be meaningfully compared.

The following program modifies the previous example – one by incrementing the variable pointer so long as the address to which it points is either less than or equal to the address of the last element of the array, which is &var[MAX - 1] –

```

#include <stdio.h>

const int MAX = 3;

int main () {

    int var[] = {10, 100, 200};
    int i, *ptr;

    /* let us have address of the first element in pointer */
    ptr = var;

```

```

i = 0;

while ( ptr <= &var[MAX - 1] ) {

    printf("Address of var[%d] = %x\n", i, ptr );
    printf("Value of var[%d] = %d\n", i, *ptr );

    /* point to the next location */
    ptr++;
    i++;
}

return 0;
}

```

When the above code is compiled and executed, it produces the following result –

```

Address of var[0] = bfdbcb20
Value of var[0] = 10
Address of var[1] = bfdbcb24
Value of var[1] = 100
Address of var[2] = bfdbcb28
Value of var[2] = 200

```

Before we understand the concept of arrays of pointers, let us consider the following example, which uses an array of 3 integers –

```

#include <stdio.h>

const int MAX = 3;

int main () {

    int var[] = {10, 100, 200};
    int i;

    for (i = 0; i < MAX; i++) {
        printf("Value of var[%d] = %d\n", i, var[i] );
    }
}

```

```
    return 0;
}
```

When the above code is compiled and executed, it produces the following result –

Value of var[0] = 10

Value of var[1] = 100

Value of var[2] = 200

There may be a situation when we want to maintain an array, which can store pointers to an int or char or any other data type available. Following is the declaration of an array of pointers to an integer –

```
int *ptr[MAX];
```

It declares **ptr** as an array of MAX integer pointers. Thus, each element in ptr, holds a pointer to an int value. The following example uses three integers, which are stored in an array of pointers, as follows –

```
#include <stdio.h>

const int MAX = 3;

int main () {

    int var[] = {10, 100, 200};
    int i, *ptr[MAX];

    for ( i = 0; i < MAX; i++) {
        ptr[i] = &var[i]; /* assign the address of integer. */
    }

    for ( i = 0; i < MAX; i++) {
        printf("Value of var[%d] = %d\n", i, *ptr[i] );
    }

    return 0;
}
```

When the above code is compiled and executed, it produces the following result –

Value of var[0] = 10

Value of var[1] = 100

Value of var[2] = 200

You can also use an array of pointers to character to store a list of strings as follows –

```
#include <stdio.h>

const int MAX = 4;

int main () {

    char *names[] = {
        "Zara Ali",
        "Hina Ali",
        "Nuha Ali",
        "Sara Ali"
    };

    int i = 0;

    for ( i = 0; i < MAX; i++) {
        printf("Value of names[%d] = %s\n", i, names[i] );
    }

    return 0;
}
```

When the above code is compiled and executed, it produces the following result –

Value of names[0] = Zara Ali

Value of names[1] = Hina Ali

Value of names[2] = Nuha Ali

Value of names[3] = Sara Ali

A pointer to a pointer is a form of multiple indirection, or a chain of pointers. Normally, a pointer contains the address of a variable. When we define a pointer to a pointer, the first pointer contains the address of the second pointer, which points to the location that contains the actual value as shown below.



A variable that is a pointer to a pointer must be declared as such. This is done by placing an additional asterisk in front of its name. For example, the following declaration declares a pointer to a pointer of type int –

```
int **var;
```

When a target value is indirectly pointed to by a pointer to a pointer, accessing that value requires that the asterisk operator be applied twice, as is shown below in the example –

```
#include <stdio.h>

int main () {

    int var;
    int *ptr;
    int **pptr;

    var = 3000;

    /* take the address of var */
    ptr = &var;

    /* take the address of ptr using address of operator & */
    pptr = &ptr;

    /* take the value using pptr */
    printf("Value of var = %d\n", var );
    printf("Value available at *ptr = %d\n", *ptr );
    printf("Value available at **pptr = %d\n", **pptr);

    return 0;
}
```

When the above code is compiled and executed, it produces the following result –

Value of var = 3000

Value available at *ptr = 3000

Value available at **pptr = 3000

C programming allows passing a pointer to a function. To do so, simply declare the function parameter as a pointer type.

Following is a simple example where we pass an unsigned long pointer to a function and change the value inside the function which reflects back in the calling function –

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

void getSeconds(unsigned long *par);

int main () {

    unsigned long sec;
    getSeconds( &sec );

    /* print the actual value */
    printf("Number of seconds: %ld\n", sec );

    return 0;
}

void getSeconds(unsigned long *par) {
    /* get the current number of seconds */
    *par = time( NULL );
    return;
}
```

When the above code is compiled and executed, it produces the following result –

Number of seconds :1294450468

The function, which can accept a pointer, can also accept an array as shown in the following example –

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

```

/* function declaration */
double getAverage(int *arr, int size);

int main () {

    /* an int array with 5 elements */
    int balance[5] = {1000, 2, 3, 17, 50};
    double avg;

    /* pass pointer to the array as an argument */
    avg = getAverage( balance, 5 );

    /* output the returned value */
    printf("Average value is: %f\n", avg );
    return 0;
}

double getAverage(int *arr, int size) {

    int i, sum = 0;
    double avg;

    for (i = 0; i < size; ++i) {
        sum += arr[i];
    }
    avg = sum / size;
}

```

```

    }

    avg = (double)sum / size;
    return avg;
}

```

When the above code is compiled together and executed, it produces the following result –

Average value is: 214.40000

We have seen in the last chapter how C programming allows to return an array from a function. Similarly, C also allows to return a pointer from a function. To do so, you would have to declare a function returning a pointer as in the following example –

```

int * myFunction() {
    .
    .
    .
}

```

Second point to remember is that, it is not a good idea to return the address of a local variable outside the function, so you would have to define the local variable as static variable.

Now, consider the following function which will generate 10 random numbers and return them using an array name which represents a pointer, i.e., address of first array element.

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

```
#include <time.h>
```

```
/* function to generate and return random numbers. */
```

```

int * getRandom( ) {

    static int r[10];

    int i;

    /* set the seed */
    srand( (unsigned)time( NULL ) );

    for ( i = 0; i < 10; ++i) {
        r[i] = rand();
        printf("%d\n", r[i] );
    }

    return r;
}

/* main function to call above defined function */
int main () {

    /* a pointer to an int */
    int *p;

    int i;

    p = getRandom();

```

```
for ( i = 0; i < 10; i++ ) {  
    printf("(p + [%d]) : %d\n", i, *(p + i) );  
}  
  
return 0;  
}
```

When the above code is compiled together and executed, it produces the following result –

1523198053

1187214107

1108300978

430494959

1421301276

930971084

123250484

106932140

1604461820

149169022

*(p + [0]) : 1523198053

*(p + [1]) : 1187214107

*(p + [2]) : 1108300978

*(p + [3]) : 430494959

*(p + [4]) : 1421301276

*(p + [5]) : 930971084

*(p + [6]) : 123250484

*(p + [7]) : 106932140

*(p + [8]) : 1604461820

*(p + [9]) : 149169022

Example: Working of Pointers

Let's take a working example.

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

```
int main()
```

```
{
```

```
    int* pc, c;
```

```
    c = 22;
```

```
    printf("Address of c: %p\n", &c);
```

```
    printf("Value of c: %d\n\n", c); // 22
```

```
    pc = &c;
```

```
    printf("Address of pointer pc: %p\n", pc);
```

```
    printf("Content of pointer pc: %d\n\n", *pc); // 22
```

```
    c = 11;
```

```
    printf("Address of pointer pc: %p\n", pc);
```

```

printf("Content of pointer pc: %d\n\n", *pc); // 11

*pc = 2;

printf("Address of c: %p\n", &c);

printf("Value of c: %d\n\n", c); // 2

return 0;

}

```

Output

Address of c: 2686784

Value of c: 22

Address of pointer pc: 2686784

Content of pointer pc: 22

Address of pointer pc: 2686784

Content of pointer pc: 11

Address of c: 2686784

Value of c: 2

Write a C program to read two numbers from user and add them using pointers. How to find sum of two number using pointers in C programming. Program to perform arithmetic operations on number using pointers.

Example

Input

Input num1: 10

Input num2: 20

Output

Sum = 30

Difference = -10

Product = 200

Quotient = 0

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

```
int main()
```

```
{
```

```
    int num1, num2, sum;
```

```
    int *ptr1, *ptr2;
```

```
    ptr1 = &num1; // ptr1 stores the address of num1
```

```
    ptr2 = &num2; // ptr2 stores the address of num2
```

```
    printf("Enter any two numbers: ");
```

```
    scanf("%d%d", ptr1, ptr2);
```

```
    sum = *ptr1 + *ptr2;
```

```
    printf("Sum = %d", sum);
```

```
    return 0;
```

```
}
```

Write a C program to copy one array elements to another array using pointers. How to copy array elements from one array to another array using pointers. Logic to copy one array to another array using pointers in C programming.

Example

Input

Input array1 elements: 10 -1 100 90 87 0 15 10 20 30

Output

Array1: 10 -1 100 90 87 0 15 10 20 30

Array2: 10 -1 100 90 87 0 15 10 20 30

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

```
#define MAX_SIZE 100 // Maximum array size
```

```
/* Function declaration to print array */
```

```
void printArray(int arr[], int size);
```

```
int main()
```

```
{
```

```
    int source_arr[MAX_SIZE], dest_arr[MAX_SIZE];
```

```
    int size, i;
```

```
    int *source_ptr = source_arr; // Pointer to source_arr
```

```
    int *dest_ptr = dest_arr; // Pointer to dest_arr
```

```
    int *end_ptr;
```

```

/*
 * Input size and elements in source array
 */
printf("Enter size of array: ");
scanf("%d", &size);
printf("Enter elements in array: ");
for (i = 0; i < size; i++)
{
    scanf("%d", (source_ptr + i));
}

```

```

// Pointer to last element of source_arr
end_ptr = &source_arr[size - 1];

```

```

/* Print source and destination array before copying */
printf("\nSource array before copying: ");
printArray(source_arr, size);

```

```

printf("\nDestination array before copying: ");
printArray(dest_arr, size);

```

```

/*
 * Run loop till source_ptr exists in source_arr
 * memory range.
 */
while(source_ptr <= end_ptr)
{
    *dest_ptr = *source_ptr;

    // Increment source_ptr and dest_ptr
    source_ptr++;
    dest_ptr++;
}

/* Print source and destination array after copying */
printf("\n\nSource array after copying: ");
printArray(source_arr, size);

printf("\n\nDestination array after copying: ");
printArray(dest_arr, size);

```

```

    return 0;
}

/**
 * Function to print array elements.
 *
 * @arr    Integer array to print.
 * @size    Size of array.
 */
void printArray(int *arr, int size)
{
    int i;

    for (i = 0; i < size; i++)
    {
        printf("%d, ", *(arr + i));
    }
}

```

Write a C program to add two matrix using pointers. C program to input two matrix from user and find sum of both matrices using pointers.

Example

Input

Input matrix1:

1 2 3

4 5 6

7 8 9

Input matrix2:

9 8 7

6 5 4

3 2 1

Output

Sum of both matrices:

10 10 10

10 10 10

10 10 10

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

```
#define ROWS 3
```

```
#define COLS 3
```

```
/* Function declaration to input, add and print matrix */
```

```
void matrixInput(int mat[][COLS]);
```

```
void matrixPrint(int mat[][COLS]);
```

```
void matrixAdd(int mat1[][COLS], int mat2[][COLS], int res[][COLS]);
```

```
int main()
```

```

{
    int mat1[ROWS][COLS], mat2[ROWS][COLS], res[ROWS][COLS];

    // Input elements in first matrix
    printf("Enter elements in first matrix of size %dx%d: \n", ROWS, COLS);
    matrixInput(mat1);

    // Input element in second matrix
    printf("\nEnter elemetns in second matrix of size %dx%d: \n", ROWS, COLS);
    matrixInput(mat2);

    // Finc sum of both matrices and print result
    matrixAdd(mat1, mat2, res);

    printf("\nSum of first and second matrix: \n");
    matrixPrint(res);

    return 0;
}

/**

```

```

* Function to read input from user and store in matrix.
*
* @mat Two dimensional integer array to store input.
*/
void matrixInput(int mat[][COLS])
{
    int i, j;

    for (i = 0; i < ROWS; i++)
    {
        for (j = 0; j < COLS; j++)
        {
            // (*(mat + i) + j) is equal to &mat[i][j]
            scanf("%d", (*(mat + i) + j));
        }
    }
}

/**
* Function to print elements of matrix on console.
*

```

* @mat Two dimensional integer array to print.

*/

```
void matrixPrint(int mat[][COLS])
```

```
{
```

```
    int i, j;
```

```
    for (i = 0; i < ROWS; i++)
```

```
    {
```

```
        for (j = 0; j < COLS; j++)
```

```
        {
```

```
            // (*(mat + i) + j) is equal to mat[i][j]
```

```
            printf("%d ", (*(mat + i) + j));
```

```
        }
```

```
        printf("\n");
```

```
    }
```

```
}
```


13. Structures(inportant)

➤ Structures

C Structure

Why use structure?

In C, there are cases where we need to store multiple attributes of an entity. It is not necessary that an entity has all the information of one type only. It can have different attributes of different data types. For example, an entity **Student** may have its name (string), roll number (int), marks (float). To store such type of information regarding an entity student, we have the following approaches:

- Construct individual arrays for storing names, roll numbers, and marks.
- Use a special data structure to store the collection of different data types.

Let's look at the first approach in detail.

1. `#include<stdio.h>`
2. `void main ()`
3. `{`
4. `char names[2][10],dummy; // 2-`
dimensional character array names is used to store the names of the students
5. `int roll_numbers[2],i;`
6. `float marks[2];`
7. `for (i=0;i<3;i++)`
8. `{`
- 9.
10. `printf("Enter the name, roll number, and marks of the student %d",i+1);`
11. `scanf("%s %d %f",&names[i],&roll_numbers[i],&marks[i]);`
12. `scanf("%c",&dummy); // enter will be stored into dummy character at each it`
eration
13. `}`
14. `printf("Printing the Student details ...\n");`

```
15. for (i=0;i<3;i++)
16. {
17.  printf("%s %d %f\n",names[i],roll_numbers[i],marks[i]);
18. }
19.}
```

Output

Enter the name, roll number, and marks of the student 1Arun 90 91

Enter the name, roll number, and marks of the student 2Varun 91 56

Enter the name, roll number, and marks of the student 3Sham 89 69

Printing the Student details...

Arun 90 91.000000

Varun 91 56.000000

Sham 89 69.000000

The above program may fulfill our requirement of storing the information of an entity student. However, the program is very complex, and the complexity increase with the amount of the input. The elements of each of the array are stored contiguously, but all the arrays may not be stored contiguously in the memory. C provides you with an additional and simpler approach where you can use a special data structure, i.e., structure, in which, you can group all the information of different data type regarding an entity.

What is Structure

Structure in c is a user-defined data type that enables us to store the collection of different data types. Each element of a structure is called a member. Structures can simulate the use of classes and templates as it can store various information

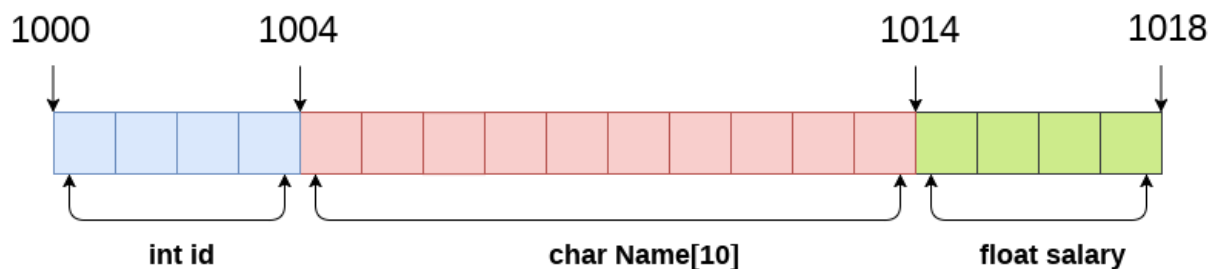
The **,struct** keyword is used to define the structure. Let's see the syntax to define the structure in c.

1. struct structure_name
2. {
3. data_type member1;
4. data_type member2;
5. .
6. .
7. data_type memeberN;
8. };

Let's see the example to define a structure for an entity employee in c.

1. struct employee
2. { int id;
3. char name[20];
4. float salary;
5. };


The following image shows the memory allocation of the structure employee that is defined in the above example.



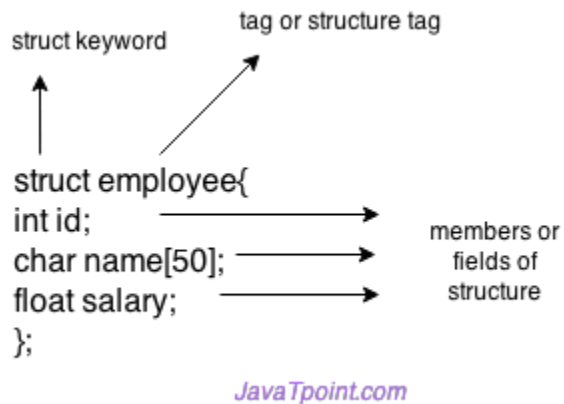
```
struct Employee
{
    int id;
    char Name[10];
    float salary;
} emp;
```

sizeof (emp) = 4 + 10 + 4 = 18 bytes

where;
sizeof (int) = 4 byte
sizeof (char) = 1 byte
sizeof (float) = 4 byte

 → 1 byte

Here, **struct** is the keyword; **employee** is the name of the structure; **id**, **name**, and **salary** are the members or fields of the structure. Let's understand it by the diagram given below:



Declaring structure variable

We can declare a variable for the structure so that we can access the member of the structure easily. There are two ways to declare structure variable:

1. By struct keyword within main() function
2. By declaring a variable at the time of defining the structure.

1st way:

Let's see the example to declare the structure variable by struct keyword. It should be declared within the main function.

1. struct employee
2. { int id;
3. char name[50];
4. float salary;
5. };

Now write given code inside the main() function.

1. struct employee e1, e2;

The variables `e1` and `e2` can be used to access the values stored in the structure. Here, `e1` and `e2` can be treated in the same way as the objects in C++ and Java.

2nd way:

Let's see another way to declare variable at the time of defining the structure.

1. `struct employee`
2. `{ int id;`
3. `char name[50];`
4. `float salary;`
5. `}e1,e2;`

Which approach is good

If number of variables are not fixed, use the 1st approach. It provides you the flexibility to declare the structure variable many times.

If no. of variables are fixed, use 2nd approach. It saves your code to declare a variable in `main()` function.

Accessing members of the structure

There are two ways to access structure members:

1. By `.` (member or dot operator)
2. By `->` (structure pointer operator)

Let's see the code to access the `id` member of `p1` variable by `.` (member) operator.

1. `p1.id`

C Structure example

Let's see a simple example of structure in C language.

1. `#include<stdio.h>`
2. `#include <string.h>`
3. `struct employee`
4. `{ int id;`

```

5.  char name[50];
6. }e1; //declaring e1 variable for structure
7. int main( )
8. {
9.  //store first employee information
10. e1.id=101;
11. strcpy(e1.name, "Sonoo Jaiswal");//copying string into char array
12. //printing first employee information
13. printf( "employee 1 id : %d\n", e1.id);
14. printf( "employee 1 name : %s\n", e1.name);
15. return 0;
16.}

```

Output:

employee 1 id : 101

employee 1 name : Sonoo Jaiswal

Let's see another example of the structure in [C language](#) to store many employees information.

```

1. #include<stdio.h>
2. #include <string.h>
3. struct employee
4. { int id;
5.  char name[50];
6.  float salary;
7. }e1,e2; //declaring e1 and e2 variables for structure
8. int main( )
9. {
10. //store first employee information
11. e1.id=101;
12. strcpy(e1.name, "Sonoo Jaiswal");//copying string into char array
13. e1.salary=56000;
14.

```

```

15. //store second employee information
16. e2.id=102;
17. strcpy(e2.name, "James Bond");
18. e2.salary=126000;
19.
20. //printing first employee information
21. printf( "employee 1 id : %d\n", e1.id);
22. printf( "employee 1 name : %s\n", e1.name);
23. printf( "employee 1 salary : %f\n", e1.salary);
24.
25. //printing second employee information
26. printf( "employee 2 id : %d\n", e2.id);
27. printf( "employee 2 name : %s\n", e2.name);
28. printf( "employee 2 salary : %f\n", e2.salary);
29. return 0;
30.}

```

Output:

employee 1 id : 101

employee 1 name : Sonoo Jaiswal

employee 1 salary : 56000.000000

employee 2 id : 102

employee 2 name : James Bond

employee 2 salary : 126000.000000

Example: Add two distances

// Program to add two distances (feet-inch)

#include <stdio.h>

struct Distance

```

{
    int feet;
    float inch;
} dist1, dist2, sum;

int main()
{
    printf("1st distance\n");
    printf("Enter feet: ");
    scanf("%d", &dist1.feet);

    printf("Enter inch: ");
    scanf("%f", &dist1.inch);
    printf("2nd distance\n");

    printf("Enter feet: ");
    scanf("%d", &dist2.feet);

    printf("Enter inch: ");
    scanf("%f", &dist2.inch);

    // adding feet
    sum.feet = dist1.feet + dist2.feet;
    // adding inches

```



```

sum.inch = dist1.inch + dist2.inch;

// changing to feet if inch is greater than 12
while (sum.inch >= 12)
{
    ++sum.feet;
    sum.inch = sum.inch - 12;
}

printf("Sum of distances = %d\'-%.1f\'", sum.feet, sum.inch);
return 0;
}

```

Output

1st distance

Enter feet: 12

Enter inch: 7.9

2nd distance

Enter feet: 2

Enter inch: 9.8

Sum of distances = 15'-5.7"

C structs and Pointers

In this tutorial, you'll learn to use pointers to access members of structs in C programming. You will also learn to dynamically allocate memory of struct types.

Before you learn about how pointers can be used with structs, be sure to check these tutorials:

C Pointers

C struct

C Pointers to struct

Here's how you can create pointers to structs.

```
struct name {  
    member1;  
    member2;  
    .  
    .  
};
```

```
int main()  
{  
    struct name *ptr, Harry;  
}
```

Here, `ptr` is a pointer to `struct`.

Example: Access members using Pointer

To access members of a structure using pointers, we use the `->` operator.

```
#include <stdio.h>  
  
struct person  
{  
    int age;
```

```

float weight;
};

int main()
{
    struct person *personPtr, person1;
    personPtr = &person1;

    printf("Enter age: ");
    scanf("%d", &personPtr->age);

    printf("Enter weight: ");
    scanf("%f", &personPtr->weight);

    printf("Displaying:\n");
    printf("Age: %d\n", personPtr->age);
    printf("weight: %f", personPtr->weight);

    return 0;
}

```

In this example, the address of `person1` is stored in the `personPtr` pointer using `personPtr = &person1;`.

Now, you can access the members of `person1` using the `personPtr` pointer.

By the way,

`personPtr->age` is equivalent to `(*personPtr).age`

`personPtr->weight` is equivalent to `(*personPtr).weight`

Dynamic memory allocation of structs

Before you proceed this section, we recommend you to check [C dynamic memory allocation](#).

Sometimes, the number of struct variables you declared may be insufficient. You may need to allocate memory during run-time. Here's how you can achieve this in C programming.

Example: Dynamic memory allocation of structs

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

```
#include <stdlib.h>
```

```
struct person {
```

```
    int age;
```

```
    float weight;
```

```
    char name[30];
```

```
};
```

```
int main()
```

```
{
```

```
    struct person *ptr;
```

```
    int i, n;
```

```
    printf("Enter the number of persons: ");
```

```
    scanf("%d", &n);
```

```

// allocating memory for n numbers of struct person
ptr = (struct person*) malloc(n * sizeof(struct person));

for(i = 0; i < n; ++i)
{
    printf("Enter first name and age respectively: ");

    // To access members of 1st struct person,
    // ptr->name and ptr->age is used

    // To access members of 2nd struct person,
    // (ptr+1)->name and (ptr+1)->age is used
    scanf("%s %d", (ptr+i)->name, &(ptr+i)->age);
}

printf("Displaying Information:\n");
for(i = 0; i < n; ++i)
    printf("Name: %s\tAge: %d\n", (ptr+i)->name, (ptr+i)->age);

return 0;
}

```

When you run the program, the output will be:

Enter the number of persons: 2

Enter first name and age respectively: Harry 24

Enter first name and age respectively: Gary 32

Displaying Information:

Name: Harry Age: 24

Name: Gary Age: 32

In the above example, `n` number of struct variables are created where `n` is entered by the user.

To allocate the memory for `n` number of `struct person`, we used,

```
ptr = (struct person*) malloc(n * sizeof(struct person));
```

Accessing Structure Members

To access any member of a structure, we use the member access operator (`.`). The member access operator is coded as a period between the structure variable name and the structure member that we wish to access. You would use the keyword `struct` to define variables of structure type. The following example shows how to use a structure in a program –

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

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

```
struct Books {  
    char title[50];  
    char author[50];  
    char subject[100];  
    int book_id;  
};
```

```

int main( ) {

    struct Books Book1;    /* Declare Book1 of type Book */

    struct Books Book2;    /* Declare Book2 of type Book */

    /* book 1 specification */

    strcpy( Book1.title, "C Programming");

    strcpy( Book1.author, "Nuha Ali");

    strcpy( Book1.subject, "C Programming Tutorial");

    Book1.book_id = 6495407;


    /* book 2 specification */

    strcpy( Book2.title, "Telecom Billing");

    strcpy( Book2.author, "Zara Ali");

    strcpy( Book2.subject, "Telecom Billing Tutorial");

    Book2.book_id = 6495700;


    /* print Book1 info */

    printf( "Book 1 title : %s\n", Book1.title);

    printf( "Book 1 author : %s\n", Book1.author);

    printf( "Book 1 subject : %s\n", Book1.subject);

```

```

printf( "Book 1 book_id : %d\n", Book1.book_id);

/* print Book2 info */

printf( "Book 2 title : %s\n", Book2.title);

printf( "Book 2 author : %s\n", Book2.author);

printf( "Book 2 subject : %s\n", Book2.subject);

printf( "Book 2 book_id : %d\n", Book2.book_id);

return 0;

}

```

When the above code is compiled and executed, it produces the following result –

Book 1 title : C Programming

Book 1 author : Nuha Ali

Book 1 subject : C Programming Tutorial

Book 1 book_id : 6495407

Book 2 title : Telecom Billing

Book 2 author : Zara Ali

Book 2 subject : Telecom Billing Tutorial

Book 2 book_id : 6495700

Structures as Function Arguments

You can pass a structure as a function argument in the same way as you pass any other variable or pointer.

```

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

```



```

struct Books {
    char title[50];
    char author[50];
    char subject[100];
    int book_id;
};

/* function declaration */
void printBook( struct Books book );

int main( ) {

    struct Books Book1;    /* Declare Book1 of type Book */
    struct Books Book2;    /* Declare Book2 of type Book */

    /* book 1 specification */
    strcpy( Book1.title, "C Programming");
    strcpy( Book1.author, "Nuha Ali");
    strcpy( Book1.subject, "C Programming Tutorial");
    Book1.book_id = 6495407;

    /* book 2 specification */
    strcpy( Book2.title, "Telecom Billing");
    strcpy( Book2.author, "Zara Ali");
    strcpy( Book2.subject, "Telecom Billing Tutorial");
    Book2.book_id = 6495700;

    /* print Book1 info */
    printBook( Book1 );

    /* Print Book2 info */
    printBook( Book2 );

    return 0;
}

```

```
void printBook( struct Books book ) {

    printf( "Book title : %s\n", book.title);
    printf( "Book author : %s\n", book.author);
    printf( "Book subject : %s\n", book.subject);
    printf( "Book book_id : %d\n", book.book_id);
}
```

When the above code is compiled and executed, it produces the following result –

```
Book title : C Programming
Book author : Nuha Ali
Book subject : C Programming Tutorial
Book book_id : 6495407
Book title : Telecom Billing
Book author : Zara Ali
Book subject : Telecom Billing Tutorial
Book book_id : 6495700
```

Pointers to Structures

You can define pointers to structures in the same way as you define pointer to any other variable –

```
struct Books *struct_pointer;
```

Now, you can store the address of a structure variable in the above defined pointer variable. To find the address of a structure variable, place the '&' operator before the structure's name as follows –

```
struct_pointer = &Book1;
```

To access the members of a structure using a pointer to that structure, you must use the → operator as follows –

```
struct_pointer->title;
```

Let us re-write the above example using structure pointer.

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

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

```

struct Books {
    char title[50];
    char author[50];
    char subject[100];
    int book_id;
};

/* function declaration */
void printBook( struct Books *book );

int main( ) {

    struct Books Book1;    /* Declare Book1 of type Book */
    struct Books Book2;    /* Declare Book2 of type Book */

    /* book 1 specification */
    strcpy( Book1.title, "C Programming");
    strcpy( Book1.author, "Nuha Ali");
    strcpy( Book1.subject, "C Programming Tutorial");
    Book1.book_id = 6495407;

    /* book 2 specification */
    strcpy( Book2.title, "Telecom Billing");
    strcpy( Book2.author, "Zara Ali");
    strcpy( Book2.subject, "Telecom Billing Tutorial");

```

```

Book2.book_id = 6495700;

/* print Book1 info by passing address of Book1 */
printBook( &Book1 );

/* print Book2 info by passing address of Book2 */
printBook( &Book2 );

return 0;
}

void printBook( struct Books *book ) {

    printf( "Book title : %s\n", book->title);
    printf( "Book author : %s\n", book->author);
    printf( "Book subject : %s\n", book->subject);
    printf( "Book book_id : %d\n", book->book_id);
}

```

When the above code is compiled and executed, it produces the following result –

Book title : C Programming

Book author : Nuha Ali

Book subject : C Programming Tutorial

Book book_id : 6495407

Book title : Telecom Billing

Book author : Zara Ali

Book subject : Telecom Billing Tutorial

Book book_id : 6495700

Bit Fields

Bit Fields allow the packing of data in a structure. This is especially useful when memory or data storage is at a premium. Typical examples include –

- Packing several objects into a machine word. e.g. 1 bit flags can be compacted.
- Reading external file formats -- non-standard file formats could be read in, e.g., 9-bit integers.

C allows us to do this in a structure definition by putting :bit length after the variable. For example –

```
struct packed_struct {  
    unsigned int f1:1;  
    unsigned int f2:1;  
    unsigned int f3:1;  
    unsigned int f4:1;  
    unsigned int type:4;  
    unsigned int my_int:9;  
} pack;
```

Here, the packed_struct contains 6 members: Four 1 bit flags f1..f3, a 4-bit type and a 9-bit my_int.

C automatically packs the above bit fields as compactly as possible, provided that the maximum length of the field is less than or equal to the integer word length of the computer. If this is not the case, then some compilers may allow memory overlap for the fields while others would store the next field in the next word.

Here's how you can pass structures to a function

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

```
struct student {
    char name[50];
    int age;
};

// function prototype
void display(struct student s);

int main() {
    struct student s1;

    printf("Enter name: ");

    // read string input from the user until \n is entered
    // \n is discarded
    scanf("%[^\n]%*c", s1.name);

    printf("Enter age: ");
    scanf("%d", &s1.age);

    display(s1); // passing struct as an argument

    return 0;
}
```

```
void display(struct student s) {  
    printf("\nDisplaying information\n");  
    printf("Name: %s", s.name);  
    printf("\nAge: %d", s.age);  
}
```

Output

Enter name: Bond

Enter age: 13

Displaying information

Name: Bond

Age: 13

Here, a struct variable s1 of type struct student is created. The variable is passed to the display() function using display(s1); statement.

Return struct from a function

Here's how you can return structure from a function:

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

```
struct student
```

```
{
```

```
    char name[50];
```

```
    int age;
```

```
};
```

```

// function prototype
struct student getInformation();

int main()
{
    struct student s;

    s = getInformation();

    printf("\nDisplaying information\n");
    printf("Name: %s", s.name);
    printf("\nRoll: %d", s.age);

    return 0;
}

struct student getInformation()
{
    struct student s1;

    printf("Enter name: ");
    scanf ("%s", s1.name);

    printf("Enter age: ");
    scanf("%d", &s1.age);
}

```



```
    return s1;
}
```

Here, the `getInformation()` function is called using `s = getInformation();` statement. The function returns a structure of type `struct student`. The returned structure is displayed from the `main()` function.

Notice that, the return type of `getInformation()` is also `struct student`.

Passing struct by reference

You can also pass structs by reference (in a similar way like you pass variables of built-in type by reference). We suggest you to read pass by reference tutorial before you proceed.

During pass by reference, the memory addresses of struct variables are passed to the function.

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

```
typedef struct Complex
```

```
{
    float real;
    float imag;
} complex;
```

```
void addNumbers(complex c1, complex c2, complex *result);
```

```
int main()
```

```
{
    complex c1, c2, result;
```

```

printf("For first number,\n");
printf("Enter real part: ");
scanf("%f", &c1.real);
printf("Enter imaginary part: ");
scanf("%f", &c1.imag);

printf("For second number, \n");
printf("Enter real part: ");
scanf("%f", &c2.real);
printf("Enter imaginary part: ");
scanf("%f", &c2.imag);

addNumbers(c1, c2, &result);
printf("\nresult.real = %.1f\n", result.real);
printf("result.imag = %.1f", result.imag);

return 0;
}

void addNumbers(complex c1, complex c2, complex *result)
{
    result->real = c1.real + c2.real;
    result->imag = c1.imag + c2.imag;
}

```

Output

For first number,

Enter real part: 1.1

Enter imaginary part: -2.4

For second number,

Enter real part: 3.4

Enter imaginary part: -3.2

result.real = 4.5

result.imag = -5.6

14. Union(inportant)

➤ Union

C Union

Like structure, Union in c language is a user-defined data type that is used to store the different type of elements.

At once, only one member of the union can occupy the memory. In other words, we can say that the size of the union in any instance is equal to the size of its largest element.

Advantage of union over structure

It occupies less memory because it occupies the size of the largest member only.

Disadvantage of union over structure

Only the last entered data can be stored in the union. It overwrites the data previously stored in the union.

Defining union

The union keyword is used to define the union. Let's see the syntax to define union in c.

```
1. union union_name
2. {
3.     data_type member1;
4.     data_type member2;
5.     .
6.     .
7.     data_type memeberN;
8. };
```

Let's see the example to define union for an employee in c.

```
1. union employee
2. { int id;
3.   char name[50];
4.   float salary;
5. };
```

C Union example

Let's see a simple example of union in C language.

```
1. #include <stdio.h>
2. #include <string.h>
3. union employee
4. { int id;
5.   char name[50];
6. }e1; //declaring e1 variable for union
7. int main( )
8. {
9.   //store first employee information
10.  e1.id=101;
11.  strcpy(e1.name, "Sonoo Jaiswal");//copying string into char array
12.  //printing first employee information
13.  printf( "employee 1 id : %d\n", e1.id);
14.  printf( "employee 1 name : %s\n", e1.name);
15.  return 0;
16. }
```

Output:

employee 1 id : 1869508435

employee 1 name : Sonoo Jaiswal

Difference between unions and structures

Let's take an example to demonstrate the difference between unions and structures:

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

```
union unionJob
```

```
{
```

```
    //defining a union
```

```
    char name[32];
```

```
    float salary;
```

```
    int workerNo;
```

```
} uJob;
```

```
struct structJob
```

```
{
```

```
    char name[32];
```

```
    float salary;
```

```
    int workerNo;
```

```
} sJob;
```

```
int main()
```

```
{
```

```
printf("size of union = %d bytes", sizeof(uJob));  
printf("\nsize of structure = %d bytes", sizeof(sJob));  
return 0;  
}
```

Output

size of union = 32

size of structure = 40

Why this difference in the size of union and structure variables?

Here, the size of sJob is 40 bytes because

- the size of name[32] is 32 bytes
- the size of salary is 4 bytes
- the size of workerNo is 4 bytes

However, the size of uJob is 32 bytes. It's because the size of a union variable will always be the size of its largest element. In the above example, the size of its largest element, (name[32]), is 32 bytes.

With a union, all members share the same memory.

Example: Accessing Union Members

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

```
union Job {  
    float salary;  
    int workerNo;  
};
```

```
int main() {
```

```
j.salary = 12.3;

// when j.workerNo is assigned a value,
// j.salary will no longer hold 12.3
j.workerNo = 100;

printf("Salary = %.1f\n", j.salary);
printf("Number of workers = %d", j.workerNo);
return 0;
}
```

Output

Salary = 0.0

Number of workers = 100

Defining a Union

To define a union, you must use the union statement in the same way as you did while defining a structure. The union statement defines a new data type with more than one member for your program. The format of the union statement is as follows –

```
union [union tag] {
    member definition;
    member definition;
    ...
    member definition;
} [one or more union variables];
```


The union tag is optional and each member definition is a normal variable definition, such as `int i`; or `float f`; or any other valid variable definition. At the end of the union's definition, before the final semicolon, you can specify one or more union variables but it is optional. Here is the way you would define a union type named `Data` having three members `i`, `f`, and `str` –

```
union Data {  
    int i;  
    float f;  
    char str[20];  
} data;
```

Now, a variable of `Data` type can store an integer, a floating-point number, or a string of characters. It means a single variable, i.e., same memory location, can be used to store multiple types of data. You can use any built-in or user defined data types inside a union based on your requirement.

The memory occupied by a union will be large enough to hold the largest member of the union. For example, in the above example, `Data` type will occupy 20 bytes of memory space because this is the maximum space which can be occupied by a character string. The following example displays the total memory size occupied by the above union –

```
#include <stdio.h>  
  
#include <string.h>  
  
union Data {  
    int i;  
    float f;  
    char str[20];  
};
```

```

int main( ) {

    union Data data;

    printf( "Memory size occupied by data : %d\n", sizeof(data));

    return 0;
}

```

When the above code is compiled and executed, it produces the following result –

Memory size occupied by data : 20

Accessing Union Members

To access any member of a union, we use the member access operator (.). The member access operator is coded as a period between the union variable name and the union member that we wish to access. You would use the keyword union to define variables of union type. The following example shows how to use unions in a program –

```

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

union Data {
    int i;
    float f;
    char str[20];
};

int main( ) {

    union Data data;

```

```

data.i = 10;
data.f = 220.5;
strcpy( data.str, "C Programming");

printf( "data.i : %d\n", data.i);
printf( "data.f : %f\n", data.f);
printf( "data.str : %s\n", data.str);

return 0;
}

```

When the above code is compiled and executed, it produces the following result –

data.i : 1917853763

data.f : 4122360580327794860452759994368.000000

data.str : C Programming

Here, we can see that the values of i and f members of union got corrupted because the final value assigned to the variable has occupied the memory location and this is the reason that the value of str member is getting printed very well.

Now let's look into the same example once again where we will use one variable at a time which is the main purpose of having unions –

```

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

```

```

union Data {

```

```

int i;
float f;
char str[20];
};

int main( ) {

    union Data data;

    data.i = 10;
    printf( "data.i : %d\n", data.i);

    data.f = 220.5;
    printf( "data.f : %f\n", data.f);

    strcpy( data.str, "C Programming");
    printf( "data.str : %s\n", data.str);

    return 0;
}

```

When the above code is compiled and executed, it produces the following result –

data.i : 10

data.f : 220.500000

data.str : C Programming

15. Strings(inportant)

➤ Strings

C Strings

The string can be defined as the one-dimensional array of characters terminated by a null ('\0'). The character array or the string is used to manipulate text such as word or sentences. Each character in the array occupies one byte of memory, and the last character must always be 0. The termination character ('\0') is important in a string since it is the only way to identify where the string ends. When we define a string as `char s[10]`, the character `s[10]` is implicitly initialized with the null in the memory.

There are two ways to declare a string in c language.

By char array

By string literal

Let's see the example of declaring string by char array in C language.

```
char ch[10]={'j', 'a', 'v', 'a', 't', 'p', 'o', 'i', 'n', 't', '\0'};
```

As we know, array index starts from 0, so it will be represented as in the figure given below.

0	1	2	3	4	5	6	7	8	9	10
j	a	v	a	t	p	o	i	n	t	\0

While declaring string, size is not mandatory. So we can write the above code as given below:

```
1. char ch[]={ 'j', 'a', 'v', 'a', 't', 'p', 'o', 'i', 'n', 't', '\0'};
```

We can also define the string by the string literal in C language. For example:

1. `char ch[]="javatpoint";`

In such case, `'\0'` will be appended at the end of the string by the compiler.

Difference between char array and string literal

There are two main differences between char array and literal.

- o We need to add the null character `'\0'` at the end of the array by ourself whereas, it is appended internally by the compiler in the case of the character array.

- o The string literal cannot be reassigned to another set of characters whereas, we can reassign the characters of the array.

String Example in C

Let's see a simple example where a string is declared and being printed. The `'%s'` is used as a format specifier for the string in c language.

```
1.  #include<stdio.h>
2.  #include <string.h>
3.  int main(){
4.      char ch[11]={'j', 'a', 'v', 'a', 't', 'p', 'o', 'i', 'n', 't', '\0'};
5.      char ch2[11]="javatpoint";
6.
7.      printf("Char Array Value is: %s\n", ch);
8.      printf("String Literal Value is: %s\n", ch2);
9.      return 0;
10. }
```

Output

Char Array Value is: javatpoint

String Literal Value is: javatpoint

Traversing String

Traversing the string is one of the most important aspects in any of the programming languages. We may need to manipulate a very large text which can be done by traversing the text. Traversing string is somewhat different from the traversing an integer array. We need to know the length of the array to traverse an integer array, whereas we may use the null character in the case of string to identify the end the string and terminate the loop.

Hence, there are two ways to traverse a string.

- o By using the length of string
- o By using the null character.

Let's discuss each one of them.

Using the length of string

Let's see an example of counting the number of vowels in a string.

```
1.  #include<stdio.h>
2.  void main ()
3.  {
4.      char s[11] = "javatpoint";
5.      int i = 0;
6.      int count = 0;
7.      while(i<11)
8.      {
```

```

9.      if(s[i]=='a' || s[i] == 'e' || s[i] == 'i' || s[i] == 'u' || s[i] == 'o')
10.     {
11.         count ++;
12.     }
13.     i++;
14. }
15. printf("The number of vowels %d",count);
16. }

```

Output

The number of vowels 4

Using the null character

Let's see the same example of counting the number of vowels by using the null character.

```

1.  #include<stdio.h>
2.  void main ()
3.  {
4.      char s[11] = "javatpoint";
5.      int i = 0;
6.      int count = 0;
7.      while(s[i] != NULL)
8.      {

```



```

9.      if(s[i]=='a' || s[i] == 'e' || s[i] == 'i' || s[i] == 'u' || s[i] == 'o')
10.     {
11.         count ++;
12.     }
13.     i++;
14. }
15. printf("The number of vowels %d",count);
16. }

```

Output

The number of vowels 4

Accepting string as the input

Till now, we have used scanf to accept the input from the user. However, it can also be used in the case of strings but with a different scenario. Consider the below code which stores the string while space is encountered.

```

1.  #include<stdio.h>
2.  void main ()
3.  {
4.      char s[20];
5.      printf("Enter the string?");
6.      scanf("%s",s);
7.      printf("You entered %s",s);
8.  }

```

Output

Enter the string?javatpoint is the best

You entered javatpoint

It is clear from the output that, the above code will not work for space separated strings. To make this code working for the space separated strings, the minor change required in the scanf function, i.e., instead of writing `scanf("%s",s)`, we must write: `scanf("%[^\n]s",s)` which instructs the compiler to store the string `s` while the new line (`\n`) is encountered. Let's consider the following example to store the space-separated strings.

```
1.  #include<stdio.h>
2.  void main ()
3.  {
4.      char s[20];
5.      printf("Enter the string?");
6.      scanf("%[^\n]s",s);
7.      printf("You entered %s",s);
8.  }
```

Output

Enter the string?javatpoint is the best

You entered javatpoint is the best

Here we must also notice that we do not need to use address of (&) operator in scanf to store a string since string `s` is an array of characters and the name of the array, i.e., `s` indicates the base address of the string (character array) therefore we need not use & with it.

Some important points

However, there are the following points which must be noticed while entering the strings by using scanf.

- o The compiler doesn't perform bounds checking on the character array. Hence, there can be a case where the length of the string can exceed the dimension of the character array which may always overwrite some important data.
- o Instead of using scanf, we may use gets() which is an inbuilt function defined in a header file string.h. The gets() is capable of receiving only one string at a time.

Pointers with strings

We have used pointers with the array, functions, and primitive data types so far. However, pointers can be used to point to the strings. There are various advantages of using pointers to point strings. Let us consider the following example to access the string via the pointer.

1. `#include<stdio.h>`
2. `void main ()`
3. `{`
4. `char s[11] = "javatpoint";`
5. `char *p = s; // pointer p is pointing to string s.`
6. `printf("%s",p); // the string javatpoint is printed if we print p.`
7. `}`

Output

javatpoint

`char s[11] = "javatpoint"`

Index	0	1	2	3	4	5	6	7	8	9	10
values	j	a	v	a	t	p	o	i	n	t	\0
Address	20	21	22	23	24	25	26	27	28	29	30
Variable	ptr	char *ptr = s									
Value	20										
Address	10										

As we know that string is an array of characters, the pointers can be used in the same way they were used with arrays. In the above example, p is declared as a pointer to the array of characters s. P affects similar to s since s is the base address of the string and treated as a pointer internally. However, we can not change the content of s or copy the content of s into another string directly. For this purpose, we need to use the pointers to store the strings. In the following example, we have shown the use of pointers to copy the content of a string into another.

1. `#include<stdio.h>`
2. `void main ()`
3. `{`
4. `char *p = "hello javatpoint";`
5. `printf("String p: %s\n",p);`
6. `char *q;`
7. `printf("copying the content of p into q...\n");`
8. `q = p;`

9. `printf("String q: %s\n",q);`

10. `}`

Output

String p: hello javatpoint

copying the content of p into q...

String q: hello javatpoint

Once a string is defined, it cannot be reassigned to another set of characters. However, using pointers, we can assign the set of characters to the string. Consider the following example.

1. `#include<stdio.h>`

2. `void main ()`

3. `{`

4. `char *p = "hello javatpoint";`

5. `printf("Before assigning: %s\n",p);`

6. `p = "hello";`

7. `printf("After assigning: %s\n",p);`

8. `}`

Output

Before assigning: hello javatpoint

After assigning: hello

The following example uses some of the above-mentioned functions –

```
#include <stdio.h>

#include <string.h>

int main () {

    char str1[12] = "Hello";

    char str2[12] = "World";

    char str3[12];

    int len ;

    /* copy str1 into str3 */

    strcpy(str3, str1);

    printf("strcpy( str3, str1) : %s\n", str3 );

    /* concatenates str1 and str2 */

    strcat( str1, str2);

    printf("strcat( str1, str2): %s\n", str1 );

    /* total length of str1 after concatenation */

    len = strlen(str1);

    printf("strlen(str1) : %d\n", len );
```

```
    return 0;
}
```

When the above code is compiled and executed, it produces the following result –

```
strcpy( str3, str1) : Hello
```

```
strcat( str1, str2): HelloWorld
```

```
strlen(str1) : 10
```

C strlen()

The strlen() function calculates the length of a given string.

The strlen() function takes a string as an argument and returns its length. The returned value is of type `size_t` (the unsigned integer type).

It is defined in the <string.h> header file.

Example: C strlen() function

```
#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 = %zu \n",strlen(a));
```

```
printf("Length of string b = %zu \n",strlen(b));
```

```
return 0;
```

```
}
```

Output

Length of string a = 7

Length of string b = 7

C strcpy()

In this tutorial, you will learn to use the strcpy() function in C programming to copy strings (with the help of an example).

C strcpy()

The function prototype of strcpy() is:

```
char* strcpy(char* destination, const char* source);
```

- The strcpy() function copies the string pointed by source (including the null character) to the destination.
- The strcpy() function also returns the copied string.

The strcpy() function is defined in the string.h header file.

Example: C strcpy()

```
#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;
}
```

Output

C programming

C strcat()

In C programming, the strcat() function concatenates (joins) two strings.

The function definition of strcat() is:

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

It is defined in the string.h header file.

strcat() arguments

As you can see, the strcat() function takes two arguments:

destination - destination string

source - source string

The strcat() function concatenates the destination string and the source string, and the result is stored in the destination string.

Example: C strcat() function

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

int main() {
    char str1[100] = "This is ", str2[] = "programiz.com";

    // concatenates str1 and str2
    // the resultant string is stored in str1.
    strcat(str1, str2);

    puts(str1);
    puts(str2);

    return 0;
}
```

Output

This is programiz.com

programiz.com

C strcmp()

In this tutorial, you will learn to compare two strings using the strcmp() function.

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

C strcmp() Prototype

The function prototype of strcmp() is:

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

strcmp() Parameters

The function takes two parameters:

- str1 - a string
- str2 - a string

Return Value from strcmp()

Return Value	Remarks
--------------	---------

0	if strings are equal
---	----------------------

non-zero	if strings are not equal
----------	--------------------------

The strcmp() function is defined in the string.h header file.

Example: C strcmp() function

```
#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(str1, str3);
```

```
    printf("strcmp(str1, str3) = %d\n", result);
```

```
    return 0;  
}
```

Output

```
strcmp(str1, str2) = 1
```

```
strcmp(str1, str3) = 0
```

Write a C program to compare two strings using loop character by character. How to compare two strings without using inbuilt library function `strcmp()` in C programming. Comparing two strings lexicographically without using string library functions. How to compare two strings using `strcmp()` library function.

Example

Input

Input string1: Learn at Codeforwin.

Input string2: Learn at Codeforwin.

Output

Both strings are lexographically equal.

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

```
#define MAX_SIZE 100 // Maximum string size
```

```
/* Compare function declaration */
```

```
int compare(char * str1, char * str2);
```

```
int main()
```

```
{
```

```
char str1[MAX_SIZE], str2[MAX_SIZE];

int res;

/* Input two strings from user */
printf("Enter first string: ");
gets(str1);
printf("Enter second string: ");
gets(str2);

/* Call the compare function to compare strings */
res = compare(str1, str2);

if(res == 0)
{
    printf("Both strings are equal.");
}
else if(res < 0)
{
    printf("First string is lexicographically smaller than second.");
}
else
{
    printf("First string is lexicographically greater than second.");
}
```

```

    }

    return 0;
}

/**
 * Compares two strings lexicographically.
 * Returns 0 if both strings are equal,
 *     negative if first string is smaller
 *     otherwise returns a positive value
 */
int compare(char * str1, char * str2)
{
    int i = 0;

    /* Iterate till both strings are equal */
    while(str1[i] == str2[i])
    {
        if(str1[i] == '\0' && str2[i] == '\0')
            break;

        i++;
    }

```

```
// Return the difference of current characters.  
return str1[i] - str2[i];  
}
```

Write a C program to concatenate two strings in single string. How to concatenate two strings to one without using `strcat()` library function. Adding two strings into one without using inbuilt library function. Logic to concatenate two strings in C programming. C program to concatenate two strings using `strcat()` library function.

Example

Input

Input string1: I love

Input string2: Codeforwin

Output

Concatenated string: I love Codeforwin

```
#include <stdio.h>  
  
#define MAX_SIZE 100 // Maximum string size  
  
int main()  
{  
    char str1[MAX_SIZE], str2[MAX_SIZE];  
    int i, j;  
  
    /* Input two strings from user */  
    printf("Enter first string: ");
```

```
gets(str1);  
printf("Enter second string: ");  
gets(str2);
```

```
/* Move till the end of str1 */
```

```
i=0;
```

```
while(str1[i] != '\0')
```

```
{
```

```
    i++;
```

```
}
```

```
/* Copy str2 to str1 */
```

```
j = 0;
```

```
while(str2[j] != '\0')
```

```
{
```

```
    str1[i] = str2[j];
```

```
    i++;
```

```
    j++;
```

```
}
```

```
// Make sure that str1 is NULL terminated
```

```
str1[i] = '\0';
```



```
printf("Concatenated string = %s", str1);
```

```
return 0;
```

```
}
```

Write a C program to convert string from lowercase to uppercase string using loop.
How to convert string from lowercase to uppercase using for loop in C programming.
C program to convert lowercase to uppercase string using `strupr()` string function.

Example

Input

Input string: I love Codeforwin.

Output

I LOVE CODEFORWIN.

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

```
#define MAX_SIZE 100 // Maximum string size
```

```
int main()
```

```
{
```

```
    char str[MAX_SIZE];
```

```
    int i;
```

```
    /* Input string from user */
```

```
    printf("Enter your text : ");
```

```
    gets(str);
```

```

for(i=0; str[i]!='\0'; i++)
{
    /*
     * If current character is lowercase alphabet then
     * convert it to uppercase.
     */
    if(str[i]>='a' && str[i]<='z')
    {
        str[i] = str[i] - 32;
    }
}

printf("Uppercase string : %s",str);

return 0;
}

```

Write a C program to convert uppercase string to lowercase using for loop. How to convert uppercase string to lowercase without using inbuilt library function `strlwr()`. How to convert string to lowercase using `strlwr()` string function.

Example

Input

Input string: I love CODEFORWIN.

Output

Lowercase string: i love codeforwin.

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

```
#define MAX_SIZE 100 // Maximum string size
```

```

int main()
{
    char str[MAX_SIZE];
    int i;

    /* Input string from user */
    printf("Enter any string: ");
    gets(str);

    // Iterate loop till last character of string
    for(i=0; str[i]!='\0'; i++)
    {
        if(str[i]>='A' && str[i]<='Z')
        {
            str[i] = str[i] + 32;
        }
    }

    printf("Lower case string: %s", str);

    return 0;
}

```

Write a C program to toggle case of each characters of a string using loop. How to change case of each characters of a string in C programming. Program to swap case of each characters in a string using loop in C. Logic to reverse the case of each character in a given string in C program.

Example

Input

Input string: Learn at Codeforwin.

Output

Toggled case string: LEARN AT cODEFORWIN.

```
#include <stdio.h>

#define MAX_SIZE 100 // Maximum string size

/* Toggle case function declaration */
void toggleCase(char * str);

int main()
{
    char str[MAX_SIZE];

    /* Input string from user */
    printf("Enter any string: ");
    gets(str);

    printf("String before toggling case: %s", str);
```

```
toggleCase(str);
```

```
printf("String after toggling case: %s", str);
```

```
return 0;
```

```
}
```

```
/**
```

```
 * Toggle case of each character in given string
```

```
 */
```

```
void toggleCase(char * str)
```

```
{
```

```
    int i = 0;
```

```
    while(str[i] != '\0')
```

```
    {
```

```
        if(str[i]>='a' && str[i]<='z')
```

```
        {
```

```
            str[i] = str[i] - 32;
```

```
        }
```

```
        else if(str[i]>='A' && str[i]<='Z')
```

```
        {
```

```
str[i] = str[i] + 32;
```

```
}
```

```
i++;
```

```
}
```

```
}
```

Write a C program to input any string from user and find the first occurrence of a given character in the string. How to find the first occurrence of a given character in a string in C programming. Logic to find first occurrence of a character in a string in C programming.

Example

Input

Input string: I love Codeforwin.

Input character to search: o

Output

'o' is found at index 3

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

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

```
#define MAX_SIZE 100 // Maximum string size
```

```
/* Function declaration */
```

```
int indexOf(const char * str, const char toFind);
```

```
int main()
```

```
{
```

```

char str[MAX_SIZE];
char toFind;
int index;

/* Input string from user and character to be searched */
printf("Enter any string: ");
gets(str);
printf("Enter character to be searched: ");
toFind = getchar();

index = indexOf(str, toFind);

if(index == -1)
    printf("'"%c' not found.", toFind);
else
    printf("'"%c' is found at index %d.", toFind, index);

return 0;
}

/**
 * Returns the first index of the given character toFind in the string.
 * If returns -1 if the given character toFind does not exists in the string.

```

```

*/
int indexOf(const char * str, const char toFind)
{
    int i = 0;

    while(str[i] != '\0')
    {
        if(str[i] == toFind)
            return i;

        i++;
    }

    // Return -1 as character not found
    return -1;
}

```

Write a C program to find maximum occurring character in a string using loop. How to find highest frequency character in a string using loop in C programming. Program to find the highest occurring character in a string in C. Logic to find maximum occurring character in a string in C programming.

Example

Input

Input string: I love Codeforwin.

Output

Maximum occurring character: 'o'

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



```

#define MAX_SIZE 100 // Maximum string size
#define MAX_CHARS 255 // Maximum characters allowed


int main()
{
    char str[MAX_SIZE];
    int freq[MAX_CHARS]; // Store frequency of each character
    int i = 0, max;
    int ascii;

    printf("Enter any string: ");
    gets(str);

    /* Initializes frequency of all characters to 0 */
    for(i=0; i<MAX_CHARS; i++)
    {
        freq[i] = 0;
    }

    /* Finds frequency of each characters */
    i=0;
    while(str[i] != '\0')

```

```

{
    ascii = (int)str[i];
    freq[ascii] += 1;

    i++;
}

/* Finds maximum frequency */
max = 0;
for(i=0; i<MAX_CHARS; i++)
{
    if(freq[i] > freq[max])
        max = i;
}

printf("Maximum occurring character is '%c' = %d times.", max, freq[max]);

return 0;
}

```

Write a C program to read any string from user and remove first occurrence of a given character from the string. The program should also use the concept of functions to remove the given character from string. How to remove first occurrences of a given character from the string.

Example

Input

Input string: I Love programming. I Love Codeforwin. I Love India.

Input character to remove: 'I'

Output

Love Programming. I Love Codeforwin. I Love India.

```
#include <stdio.h>

#include <string.h>

#define MAX_SIZE 100 // Maximum string size

/* Function declaration */
void removeFirst(char *, const char);

int main()
{
    char str[MAX_SIZE];
    char toRemove;

    printf("Enter any string: ");
    gets(str);

    printf("Enter character to remove from string: ");
    toRemove = getchar();
```

```
removeFirst(str, toRemove);
```

```
printf("String after removing first '%c' : %s", toRemove, str);
```

```
return 0;
```

```
}
```

```
/**
```

```
 * Function to remove first occurrence of a character from the string.
```

```
 */
```

```
void removeFirst(char * str, const char toRemove)
```

```
{
```

```
    int i = 0;
```

```
    int len = strlen(str);
```

```
    /* Run loop till the first occurrence of the character is not found */
```

```
    while(i<len && str[i]!=toRemove)
```

```
        i++;
```

```
    /* Shift all characters right to the position found above, to one place left */
```

```
    while(i < len)
```

```
{
```

```

    str[i] = str[i+1];
    i++;
}
}

```

Write a C program to remove all repeated characters in a string using loops. How to remove all duplicate characters from a string using for loop in C programming. Program to find and remove all duplicate characters in a string. Logic to remove all repeated character from string in C program.

Example

Input

Input string: Programming in C.

Output

String after removing duplicate characters: Progamin C.

```

#include <stdio.h>

#define MAX_SIZE 100 // Maximum string size

/* Function declarations */
void removeDuplicates(char * str);
void removeAll(char * str, const char toRemove, int index);

int main()
{
    char str[MAX_SIZE];

```

```

/* Input string from user */
printf("Enter any string: ");
gets(str);

printf("String before removing duplicates: %s\n", str);

removeDuplicates(str);

printf("String after removing duplicates: %s\n", str);

return 0;
}

/**
 * Remove all duplicate characters from the given string
 */
void removeDuplicates(char * str)
{
    int i = 0;

    while(str[i] != '\0')
    {
        /* Remove all duplicate of character string[i] */

```

```

        removeAll(str, str[i], i + 1);
    i++;
}
}

/**
 * Remove all occurrences of a given character from string.
 */
void removeAll(char * str, const char toRemove, int index)
{
    int i;

    while(str[index] != '\0')
    {
        /* If duplicate character is found */
        if(str[index] == toRemove)
        {
            /* Shift all characters from current position to one place left */
            i = index;
            while(str[i] != '\0')
            {
                str[i] = str[i + 1];
                i++;
            }

```

```

    }
else
{
    index++;
}
}
}
}

```

Write a C program to replace first occurrence of a character with another character in a string. How to replace first occurrence of a character with another character in a string using loop in C programming. Logic to replace first occurrence of a character in given string.

Example

Input

Input string: I love programming.

Input character to replace: .

Input character to replace with: !

Output

String after replacing '.' with '!': I love programming!

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

```
#define MAX_SIZE 100 // Maximum string size
```

```
/* Function declaration */
```

```
void replaceFirst(char * str, char oldChar, char newChar);
```



```
int main()
{
    char str[MAX_SIZE], oldChar, newChar;

    printf("Enter any string: ");
    gets(str);

    printf("Enter character to replace: ");
    oldChar = getchar();

    // Used to skip extra ENTER character
    getchar();

    printf("Enter character to replace '%c' with: ", oldChar);
    newChar = getchar();

    printf("\nString before replacing: %s\n", str);

    replaceFirst(str, oldChar, newChar);

    printf("String after replacing first '%c' with '%c' : %s", oldChar, newChar, str);

    return 0;
```

```

}

/**
 * Replace first occurrence of a character with
 * another in given string.
 */
void replaceFirst(char * str, char oldChar, char newChar)
{
    int i=0;

    /* Run till end of string */
    while(str[i] != '\0')
    {
        /* If an occurrence of character is found */
        if(str[i] == oldChar)
        {
            str[i] = newChar;
            break;
        }

        i++;
    }
}

```

Write a C program to replace all occurrence of a character with another in a string using function. How to replace all occurrences of a character with another in a string using functions in C programming. Logic to replace all occurrences of a character in given string.

Example

Input

Input string: I_love_learning_at_Codeforwin.

Input character to replace: _

Input character to replace with: -

Output

String after replacing '_' with '-': I-love-learning-at-Codeforwin

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

```
#define MAX_SIZE 100 // Maximum string size
```

```
/* Function declaration */
```

```
void replaceAll(char * str, char oldChar, char newChar);
```

```
int main()
```

```
{
```

```
    char str[MAX_SIZE], oldChar, newChar;
```

```
    printf("Enter any string: ");
```

```
    gets(str);
```

```

printf("Enter character to replace: ");

oldChar = getchar();


// Dummy getchar() to eliminate extra ENTER character
getchar();


printf("Enter character to replace '%c' with: ", oldChar);

newChar = getchar();


printf("\nString before replacing: %s", str);


replaceAll(str, oldChar, newChar);


printf("\n\nString after replacing '%c' with '%c' : %s", oldChar, newChar, str);


return 0;
}


/**
 * Replace all occurrence of a character in given string.
 */
void replaceAll(char * str, char oldChar, char newChar)

```

```

{
    int i = 0;

    /* Run till end of string */
    while(str[i] != '\0')
    {
        /* If occurrence of character is found */
        if(str[i] == oldChar)
        {
            str[i] = newChar;
        }

        i++;
    }
}

```

Write a C program to find the first occurrence of word in a string using loop. How to find the first occurrence of any word in a given string in C programming. Logic to search a word in a given string in C programming.

Example

Input

Input string: I love programming!

Input word to search: love

Output

'love' is found at index 2.

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

```

#include <string.h>

#define MAX_SIZE 100 // Maximum string size

int main()
{
    char str[MAX_SIZE], word[MAX_SIZE];
    int i, index, found = 0;

    /* Input string and word from user */
    printf("Enter any string: ");
    gets(str);
    printf("Enter word to be searched: ");
    gets(word);

    /* Run loop from start to end of string */
    index = 0;
    while(str[index] != '\0')
    {

        /* If first character of word matches with the given string */
        if(str[index] == word[0])
        {
            /* Match entire word with current found index */

```

```
i=0;
```

```
found = 1;
```

```
while(word[i] != '\0')
```

```
{
```

```
    if(str[index + i] != word[i])
```

```
    {
```

```
        found = 0;
```

```
        break;
```

```
    }
```

```
    i++;
```

```
}
```

```
}
```

```
/* If the word is found then get out of loop */
```

```
if(found == 1)
```

```
{
```

```
    break;
```

```
}
```

```
index++;
```

```
}
```

```
/* Print success message if the word is found */
```

```

    if(found == 1)
    {
        printf("\n'%s' is found at index %d.", word, index);
    }
    else
    {
        printf("\n'%s' is not found.", word);
    }

    return 0;
}

```

Write a C program to trim leading white space characters from a given string using loop. How to remove leading white space characters from a given string using loop in C programming. Logic to remove leading/starting blank spaces from a given string in C programming.

Example

Input

Input string: Lots of leading space.

Output

String after removing leading white spaces:

Lots of leading space.

```

#include <stdio.h>

#define MAX_SIZE 100 // Maximum string size

/* Function declaration */

```



```

void trimLeading(char * str);

int main()
{
    char str[MAX_SIZE];

    /* Input string from user */
    printf("Enter any string: ");
    gets(str);

    printf("\nString before trimming leading whitespace: \n%s", str);

    trimLeading(str);

    printf("\n\nString after trimming leading whitespace: \n%s", str);

    return 0;
}

/**
 * Remove leading whitespace characters from string
 */

```

```

void trimLeading(char * str)
{
    int index, i, j;

    index = 0;

    /* Find last index of whitespace character */
    while(str[index] == ' ' || str[index] == '\t' || str[index] == '\n')
    {
        index++;
    }

    if(index != 0)
    {
        /* Shift all trailing characters to its left */
        i = 0;
        while(str[i + index] != '\0')
        {
            str[i] = str[i + index];
            i++;
        }
        str[i] = '\0'; // Make sure that string is NULL terminated
    }

```

```
}
```

Write a C program to trim both leading and trailing white space characters in a string using loop. How to remove both leading and trailing white space characters in a string using loop in C programming. Logic to delete all leading and trailing white space characters from a given string in C.

Example

Input

Input string: " Lots of leading and trailing spaces. "

Output

String after removing leading and trailing white spaces:

"Lots of leading and trailing spaces."

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

```
#define MAX_SIZE 100 // Maximum string size
```

```
/* Function declaration */
```

```
void trim(char * str);
```

```
int main()
```

```
{
```

```
    char str[MAX_SIZE];
```

```
    /* Input string from user */
```

```
    printf("Enter any string: ");
```

```
    gets(str);
```

```
printf("\nString before trimming white space: \n'%s'", str);
```

```
trim(str);
```

```
printf("\n\nString after trimming white space: \n'%s'", str);
```

```
return 0;
```

```
}
```

```
/**
```

```
 * Remove leading and trailing white space characters
```

```
 */
```

```
void trim(char * str)
```

```
{
```

```
    int index, i;
```

```
    /*
```

```
     * Trim leading white spaces
```

```
     */
```

```
    index = 0;
```

```
    while(str[index] == ' ' || str[index] == '\t' || str[index] == '\n')
```

```
    {
```

```

    index++;
}

/* Shift all trailing characters to its left */
i = 0;
while(str[i + index] != '\0')
{
    str[i] = str[i + index];
    i++;
}
str[i] = '\0'; // Terminate string with NULL

```

```

/*
 * Trim trailing white spaces
 */
i = 0;
index = -1;
while(str[i] != '\0')
{
    if(str[i] != ' ' && str[i] != '\t' && str[i] != '\n')
    {
        index = i;
    }
}

```

```
i++;
```

```
}
```

```
/* Mark the next character to last non white space character as NULL */
```

```
str[index + 1] = '\0';
```

```
}
```

Write a C program to remove extra spaces, blanks from a string. How to remove extra blank spaces, blanks from a given string using functions in C programming. Logic to remove extra white space characters from a string in C.

Example

Input

Input string: Learn C programming at Codeforwin.

Output

String after removing extra blanks:

"Learn C programming at Codeforwin"

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

```
#include <stdlib.h>
```

```
#define MAX_SIZE 100 // Maximum string size
```

```
/* Function declaration */
```

```
char * removeBlanks(const char * str);
```

```

int main()
{
    char str[MAX_SIZE];
    char * newString;

    printf("Enter any string: ");
    gets(str);

    printf("\nString before removing blanks: \n'%s'", str);

    newString = removeBlanks(str);

    printf("\n\nString after removing blanks: \n'%s'", newString);

    return 0;
}

/**
 * Removes extra blank spaces from the given string
 * and returns a new string with single blank spaces
 */
char * removeBlanks(const char * str)

```

```

{
    int i, j;

    char * newString;

    newString = (char *)malloc(MAX_SIZE);

    i = 0;
    j = 0;

    while(str[i] != '\0')
    {
        /* If blank space is found */
        if(str[i] == ' ')
        {
            newString[j] = ' ';

            j++;

            /* Skip all consecutive spaces */
            while(str[i] == ' ')
                i++;
        }

        newString[j] = str[i];
    }
}

```



```
    i++;  
    j++;  
}  
// NULL terminate the new string  
newString[j] = '\0';  
  
return newString;  
}
```

16. Files handling exercises

➤ Files handling

File Handling in C

In programming, we may require some specific input data to be generated several numbers of times. Sometimes, it is not enough to only display the data on the console. The data to be displayed may be very large, and only a limited amount of data can be displayed on the console, and since the memory is volatile, it is impossible to recover the programmatically generated data again and again. However, if we need to do so, we may store it onto the local file system which is volatile and can be accessed every time. Here, comes the need of file handling in C.

File handling in C enables us to create, update, read, and delete the files stored on the local file system through our C program. The following operations can be performed on a file.

- o Creation of the new file
- o Opening an existing file
- o Reading from the file
- o Writing to the file
- o Deleting the file

Functions for file handling

There are many functions in the C library to open, read, write, search and close the file. A list of file functions are given below:

No.	Function	Description
1	fopen()	opens new or existing file
2	fprintf()	write data into the file
3	fscanf()	reads data from the file

4	fputc()	writes a character into the file
5	fgetc()	reads a character from file
6	fclose()	closes the file
7	fseek()	sets the file pointer to given position
8	fputw()	writes an integer to file
9	fgetw()	reads an integer from file
10	ftell()	returns current position
11	rewind()	sets the file pointer to the beginning of the file

Opening File: fopen()

We must open a file before it can be read, write, or update. The fopen() function is used to open a file. The syntax of the fopen() is given below.

```
FILE *fopen( const char * filename, const char * mode );
```

The fopen() function accepts two parameters:

The file name (string). If the file is stored at some specific location, then we must mention the path at which the file is stored. For example, a file name can be like "c://some_folder/some_file.ext".

The mode in which the file is to be opened. It is a string.

We can use one of the following modes in the fopen() function.

Mode Description

r	opens a text file in read mode
w	opens a text file in write mode
a	opens a text file in append mode
r+	opens a text file in read and write mode
w+	opens a text file in read and write mode

a+ opens a text file in read and write mode
rb opens a binary file in read mode
wb opens a binary file in write mode
ab opens a binary file in append mode
rb+ opens a binary file in read and write mode
wb+ opens a binary file in read and write mode
ab+ opens a binary file in read and write mode

The fopen function works in the following way.

Firstly, It searches the file to be opened.

Then, it loads the file from the disk and place it into the buffer. The buffer is used to provide efficiency for the read operations.

It sets up a character pointer which points to the first character of the file.

Consider the following example which opens a file in write mode.

```
#include<stdio.h>

void main( )
{
FILE *fp ;
char ch ;
fp = fopen("file_handle.c","r") ;
while ( 1 )
{
ch = fgetc ( fp ) ;
if ( ch == EOF )
break ;
```

```
printf("%c",ch) ;
```

```
}
```

```
fclose (fp ) ;
```

```
}
```

Output

The content of the file will be printed.

```
#include;
```

```
void main( )
```

```
{
```

```
FILE *fp; // file pointer
```

```
char ch;
```

```
fp = fopen("file_handle.c","r");
```

```
while ( 1 )
```

```
{
```

```
ch = fgetc ( fp ); //Each character of the file is read and stored in the character file.
```

```
if ( ch == EOF )
```

```
break;
```

```
printf("%c",ch);
```

```
}
```

```
fclose (fp );
```

```
}
```

C fprintf() and fscanf()

Writing File : fprintf() function

The fprintf() function is used to write set of characters into file. It sends formatted output to a stream.

Syntax:

1. int fprintf(FILE *stream, const char *format [, argument, ...])

Example:

1. #include <stdio.h>
2. main(){
3. FILE *fp;
4. fp = fopen("file.txt", "w");//opening file
5. fprintf(fp, "Hello file by fprintf...\n");//writing data into file
6. fclose(fp);//closing file
7. }

Reading File : fscanf() function

The fscanf() function is used to read set of characters from file. It reads a word from the file and returns EOF at the end of file.

Syntax:

1. int fscanf(FILE *stream, const char *format [, argument, ...])

Example:

1. #include <stdio.h>
2. main(){
3. FILE *fp;
4. char buff[255];//creating char array to store data of file
5. fp = fopen("file.txt", "r");

```
6.    while(fscanf(fp, "%s", buff)!=EOF){
7.        printf("%s ", buff );
8.    }
9.    fclose(fp);
10. }
```

Output:

Hello file by fprintf...

C File Example: Storing employee information

Let's see a file handling example to store employee information as entered by user from console. We are going to store id, name and salary of the employee.

```
1.    #include <stdio.h>
2.    void main()
3.    {
4.        FILE *fptr;
5.        int id;
6.        char name[30];
7.        float salary;
8.        fptr = fopen("emp.txt", "w+");/* open for writing */
9.        if (fptr == NULL)
10.       {
11.           printf("File does not exists \n");
12.           return;
13.       }
14.       printf("Enter the id\n");
```

```
15.    scanf("%d", &id);
16.    fprintf(fp, "Id= %d\n", id);
17.    printf("Enter the name \n");
18.    scanf("%s", name);
19.    fprintf(fp, "Name= %s\n", name);
20.    printf("Enter the salary\n");
21.    scanf("%f", &salary);
22.    fprintf(fp, "Salary= %.2f\n", salary);
23.    fclose(fp);
24. }
```

Output:

Enter the id

1

Enter the name

sonoo

Enter the salary

120000

Now open file from current directory. For windows operating system, go to TC\bin directory, you will see emp.txt file. It will have following information.

emp.txt

Id= 1

Name= sonoo

Salary= 120000

C fputc() and fgetc()

Writing File : fputc() function

The fputc() function is used to write a single character into file. It outputs a character to a stream.

Syntax:

1. int fputc(int c, FILE *stream)

Example:

1. #include <stdio.h>
2. main(){
3. FILE *fp;
4. fp = fopen("file1.txt", "w");//opening file
5. fputc('a',fp);//writing single character into file
6. fclose(fp);//closing file
7. }

file1.txt

a

Reading File : fgetc() function

The fgetc() function returns a single character from the file. It gets a character from the stream. It returns EOF at the end of file.

Syntax:

1. int fgetc(FILE *stream)

Example:

1. #include<stdio.h>
2. #include<conio.h>
3. void main(){

```
4.  FILE *fp;
5.  char c;
6.  clrscr();
7.  fp=fopen("myfile.txt","r");
8.
9.  while((c=fgetc(fp))!=EOF){
10.  printf("%c",c);
11.  }
12.  fclose(fp);
13.  getch();
14.  }
```

myfile.txt

this is simple text message

C fputs() and fgets()

The fputs() and fgets() in C programming are used to write and read string from stream. Let's see examples of writing and reading file using fgets() and fputs() functions.

Writing File : fputs() function

The fputs() function writes a line of characters into file. It outputs string to a stream.

Syntax:

```
1.  int fputs(const char *s, FILE *stream)
```

Example:

```
1.  #include<stdio.h>
2.  #include<conio.h>
```

```
3. void main(){
4. FILE *fp;
5. clrscr();
6.
7. fp=fopen("myfile2.txt","w");
8. fputs("hello c programming",fp);
9.
10. fclose(fp);
11. getch();
12. }
```

myfile2.txt

hello c programming

Reading File : fgets() function

The fgets() function reads a line of characters from file. It gets string from a stream.

Syntax:

```
1. char* fgets(char *s, int n, FILE *stream)
```

Example:

```
1. #include<stdio.h>
2. #include<conio.h>
3. void main(){
4. FILE *fp;
5. char text[300];
6. clrscr();
```

```
7.  
8.   fp=fopen("myfile2.txt","r");  
9.   printf("%s",fgets(text,200,fp));  
10.  
11.  fclose(fp);  
12.  getch();  
13.  }
```

Output:

hello c programming

C rewind() function

The `rewind()` function sets the file pointer at the beginning of the stream. It is useful if you have to use stream many times.

Syntax:

```
1.   void rewind(FILE *stream)
```

Example:

File: file.txt

```
1.   this is a simple text
```

File: rewind.c

```
1.   #include<stdio.h>  
2.   #include<conio.h>  
3.   void main(){  
4.       FILE *fp;  
5.       char c;  
6.       clrscr();  
7.       fp=fopen("file.txt","r");
```

```

8.
9.   while((c=fgetc(fp))!=EOF){
10.   printf("%c",c);
11.   }
12.
13.   rewind(fp); //moves the file pointer at beginning of the file
14.
15.   while((c=fgetc(fp))!=EOF){
16.   printf("%c",c);
17.   }
18.
19.   fclose(fp);
20.   getch();
21.   }

```

Output:

this is a simple textthis is a simple text

C ftell() function

The ftell() function returns the current file position of the specified stream. We can use ftell() function to get the total size of a file after moving file pointer at the end of file. We can use SEEK_END constant to move the file pointer at the end of file.

Syntax:

```
1.   long int ftell(FILE *stream)
```

Example:

File: ftell.c

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

```
2.    #include <conio.h>
3.    void main (){
4.        FILE *fp;
5.        int length;
6.        clrscr();
7.        fp = fopen("file.txt", "r");
8.        fseek(fp, 0, SEEK_END);
9.
10.       length = ftell(fp);
11.
12.       fclose(fp);
13.       printf("Size of file: %d bytes", length);
14.       getch();
15.   }
```

Output:

Size of file: 21 bytes

Write a C program to read data from user and append data into a file. How to append data at end of a file in C programming. In this post I will explain append mode in file handling. I will cover how to append data into a file in C using append file mode.

Example

Source file content

I love programming.

Programming with files is fun.

String to append

Learning C programming at Codeforwin is simple and easy.

Output file content after append

I love programming.

Programming with files is fun.

Learning C programming at Codeforwin is simple and easy.

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

```
#include <stdlib.h>
```

```
#define BUFFER_SIZE 1000
```

```
void readFile(FILE * fPtr);
```

```
int main()
```

```
{
```

```
    /* File pointer to hold reference of input file */
```

```
    FILE *fPtr;
```

```
    char filePath[100];
```

```
    char dataToAppend[BUFFER_SIZE];
```

```
    /* Input file path to remove empty lines from user */
```

```
    printf("Enter file path: ");
```

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

```
/* Open all file in append mode. */
```

```
fPtr = fopen(filePath, "a");
```

```
/* fopen() return NULL if unable to open file in given mode. */
```

```
if (fPtr == NULL)
```

```
{
```

```
/* Unable to open file hence exit */
```

```
printf("\nUnable to open '%s' file.\n", filePath);
```

```
printf("Please check whether file exists and you have write privilege.\n");
```

```
exit(EXIT_FAILURE);
```

```
}
```

```
/* Input data to append from user */
```

```
printf("\nEnter data to append: ");
```

```
fflush(stdin); // To clear extra white space characters in stdin
```

```
fgets(dataToAppend, BUFFER_SIZE, stdin);
```

```
/* Append data to file */
```

```
fputs(dataToAppend, fPtr);
```



```
/* Reopen file in read mode to print file contents */  
fPtr = freopen(filePath, "r", fPtr);  
  
/* Print file contents after appending string */  
printf("\nSuccessfully appended data to file. \n");  
printf("Changed file contents:\n\n");  
readFile(fPtr);  
  
/* Done with file, hence close file. */  
fclose(fPtr);  
  
return 0;  
}
```

```
/**  
 * Reads a file character by character  
 * and prints on console.  
 *  
 * @fPtr Pointer to FILE to read.
```

```

*/
void readFile(FILE * fPtr)
{
    char ch;

    do
    {
        ch = fgetc(fPtr);

        putchar(ch);

    } while (ch != EOF);
}

```

Write a C program to read contents of two files and compare them character by character. How to compare two files character by character and line by line in C programming. Logic to compare two files line by line and print difference line and column number in C program.

Example

File 1

Learn C programming at Codeforwin.

Working with files and directories.

File 2

Learn C programming at Codeforwin.

Working with array and pointers.

Output

File are not equal.

Line: 2, column: 14

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

```
#include <stdlib.h>
```

```
/* Function declaration */
```

```
int compareFile(FILE * fPtr1, FILE * fPtr2, int * line, int * col);
```

```
int main()
```

```
{
```

```
    /* File pointer to hold reference of input file */
```

```
    FILE * fPtr1;
```

```
    FILE * fPtr2;
```

```
    char path1[100];
```

```
    char path2[100];
```

```
    int diff;
```

```
    int line, col;
```

```
    /* Input path of files to compare */
```

```
    printf("Enter path of first file: ");
```

```
    scanf("%s", path1);
```

```
printf("Enter path of second file: ");
```

```
scanf("%s", path2);
```

```
/* Open all files to compare */
```

```
fPtr1 = fopen(path1, "r");
```

```
fPtr2 = fopen(path2, "r");
```

```
/* fopen() return NULL if unable to open file in given mode. */
```

```
if (fPtr1 == NULL || fPtr2 == NULL)
```

```
{
```

```
    /* Unable to open file hence exit */
```

```
    printf("\nUnable to open file.\n");
```

```
    printf("Please check whether file exists and you have read privilege.\n");
```

```
    exit(EXIT_FAILURE);
```

```
}
```

```
/* Call function to compare file */
```

```
diff = compareFile(fPtr1, fPtr2, &line, &col);
```

```
if (diff == 0)
```

```
{
```

```
    printf("\nBoth files are equal.");
```

```

    }
    else
    {
        printf("\nFiles are not equal.\n");
        printf("Line: %d, col: %d\n", line, col);
    }

    /* Finally close files to release resources */
    fclose(fPtr1);
    fclose(fPtr2);

    return 0;
}

/**
 * Function to compare two files.
 * Returns 0 if both files are equivalent, otherwise returns
 * -1 and sets line and col where both file differ.
 */
int compareFile(FILE * fPtr1, FILE * fPtr2, int * line, int * col)
{
    char ch1, ch2;

```

```
*line = 1;
```

```
*col = 0;
```

```
do
```

```
{
```

```
    // Input character from both files
```

```
    ch1 = fgetc(fPtr1);
```

```
    ch2 = fgetc(fPtr2);
```

```
    // Increment line
```

```
    if (ch1 == '\n')
```

```
    {
```

```
        *line += 1;
```

```
        *col = 0;
```

```
    }
```

```
    // If characters are not same then return -1
```

```
    if (ch1 != ch2)
```

```
        return -1;
```

```
    *col += 1;
```

```
} while (ch1 != EOF && ch2 != EOF);
```

```

/* If both files have reached end */
if (ch1 == EOF && ch2 == EOF)
    return 0;
else
    return -1;
}

```

Write a C program to list all files in a directory. How to list all files in a directory recursively. How to use `readdir()` function to list all files in a directory recursively. Logic to list all files and sub-directories of a directory in C programming. How to use `opendir()`, `readdir()` and `closedir()` library functions.

```

#include <stdio.h>
#include <sys/types.h>
#include <dirent.h>

```

```

void listFiles(const char *path);

```

```

int main()
{
    // Directory path to list files
    char path[100];

```

```

    // Input path from user
    printf("Enter path to list files: ");
    scanf("%s", path);

    listFiles(path);

    return 0;
}

/**
 * Lists all files and sub-directories at given path.
 */
void listFiles(const char *path)
{
    struct dirent *dp;
    DIR *dir = opendir(path);

    // Unable to open directory stream
    if (!dir)
        return;

    while ((dp = readdir(dir)) != NULL)
    {

```



```
    printf("%s\n", dp->d_name);  
}
```

```
// Close directory stream  
closedir(dir);  
}
```

Write a C program to rename a file using rename() function. How to rename a file using rename() function in C programming. rename() function in C programming.

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

```
int main()  
{  
    // Path to old and new files  
    char oldName[100], newName[100];
```

```
    // Input old and new file name  
    printf("Enter old file path: ");  
    scanf("%s", oldName);
```

```
    printf("Enter new file path: ");  
    scanf("%s", newName);
```

```
    // rename old file with new name
```

```

    if (rename(oldName, newName) == 0)
    {
        printf("File renamed successfully.\n");
    }
    else
    {
        printf("Unable to rename files. Please check files exist and you have permissions
to modify files.\n");
    }

    return 0;
}

```

Write a C program to check file properties using `stat()` function. How to check file permissions, size, creation and modification date of a file in C programming. How to use `stat()` function to find various file properties.

```

#include <stdio.h>
#include <unistd.h>
#include <sys/stat.h>
#include <time.h>

```

```

void printFileProperties(struct stat stats);

```

```

int main()

```

```
{  
    char path[100];  
    struct stat stats;  
  
    printf("Enter source file path: ");  
    scanf("%s", path);  
  
    // stat() returns 0 on successful operation,  
    // otherwise returns -1 if unable to get file properties.  
    if (stat(path, &stats) == 0)  
    {  
        printFileProperties(stats);  
    }  
    else  
    {  
        printf("Unable to get file properties.\n");  
        printf("Please check whether '%s' file exists.\n", path);  
    }  
  
    return 0;  
}
```

```

/**
 * Function to print file properties.
 */
void printFileProperties(struct stat stats)
{
    struct tm dt;

    // File permissions
    printf("\nFile access: ");
    if (stats.st_mode & R_OK)
        printf("read ");
    if (stats.st_mode & W_OK)
        printf("write ");
    if (stats.st_mode & X_OK)
        printf("execute");

    // File size
    printf("\nFile size: %d", stats.st_size);

    // Get file creation time in seconds and
    // convert seconds to date and time format
    dt = *(gmtime(&stats.st_ctime));
    printf("\nCreated on: %d-%d-%d %d:%d:%d", dt.tm_mday, dt.tm_mon, dt.tm_year
+ 1900,

```

```

                                dt.tm_hour, dt.tm_min, dt.tm_sec);

// File modification time
dt = *(gmtime(&stats.st_mtime));
printf("\nModified on: %d-%d-%d %d:%d:%d", dt.tm_mday, dt.tm_mon, dt.tm_year
+ 1900,
                                dt.tm_hour, dt.tm_min, dt.tm_sec);

}

```

Write a C program to print source code of itself as output. How to print source code of itself as output in C programming.

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

```
#include <stdlib.h>
```

```
int main()
```

```
{
```

```
FILE *fPtr;
```

```
char ch;
```

```
/*
```

```
* __FILE__ is a macro that contains path of current file.
```

```
* Open current program in read mode.
```

```

*/
fPtr = fopen(__FILE__, "r");

/* fopen() return NULL if unable to open file in given mode. */
if (fPtr == NULL)
{
    /* Unable to open file hence exit */
    printf("\nUnable to open file.\n");
    printf("Please check whether file exists and you have read privilege.\n");
    exit(EXIT_SUCCESS);
}

/* Read file character by character */
while ((ch = fgetc(fPtr)) != EOF)
{
    printf("%c", ch);
}

/* Close files to release resources */
fclose(fPtr);

```

```
    return 0;
}
```

Write a C program to input line number and replace specific line with another in text file. How to replace specific line in a text file in C programming. Logic to replace specific line with another in a text file in C program.

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

```
#include <stdlib.h>
```

```
#define BUFFER_SIZE 1000
```

```
int main()
```

```
{
```

```
    /* File pointer to hold reference of input file */
```

```
    FILE * fPtr;
```

```
    FILE * fTemp;
```

```
    char path[100];
```

```
    char buffer[BUFFER_SIZE];
```

```
    char newline[BUFFER_SIZE];
```

```
    int line, count;
```

```
    printf("Enter path of source file: ");
```

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

```
printf("Enter line number to replace: ");
```

```
scanf("%d", &line);
```

```
/* Remove extra new line character from stdin */
```

```
fflush(stdin);
```

```
printf("Replace '%d' line with: ", line);
```

```
fgets(newline, BUFFER_SIZE, stdin);
```

```
/* Open all required files */
```

```
fPtr = fopen(path, "r");
```

```
fTemp = fopen("replace.tmp", "w");
```

```
/* fopen() return NULL if unable to open file in given mode. */
```

```
if (fPtr == NULL || fTemp == NULL)
```

```
{
```

```
/* Unable to open file hence exit */
```

```
printf("\nUnable to open file.\n");
```

```
printf("Please check whether file exists and you have read/write privilege.\n");
```

```
exit(EXIT_SUCCESS);
```

```
}
```



```

/*
 * Read line from source file and write to destination
 * file after replacing given line.
 */
count = 0;
while ((fgets(buffer, BUFFER_SIZE, fPtr)) != NULL)
{
    count++;

    /* If current line is line to replace */
    if (count == line)
        fputs(newline, fTemp);
    else
        fputs(buffer, fTemp);
}

/* Close all files to release resource */
fclose(fPtr);
fclose(fTemp);

/* Delete original source file */

```

```
remove(path);
```

```
/* Rename temporary file as original file */
```

```
rename("replace.tmp", path);
```

```
printf("\nSuccessfully replaced '%d' line with '%s'.", line, newline);
```

```
return 0;
```

```
}
```

Write a C program to count occurrences of all words in a file. Logic to count occurrences of all words in a file in C program. How to count occurrences of all words in a file in C programming. C program to count occurrences of unique words in a file.

In previous post I explained how to [count occurrence of a word in file](#). In this post we will step further and will count occurrence of all words in given file. So let's, get started.

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

```
#include <stdlib.h>
```

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

```
#include <ctype.h>
```

```
#define MAX_WORDS 1000
```

```
int main()
```

```
{
```

```

FILE *fptr;

char path[100];

int i, len, index, isUnique;


// List of distinct words
char words[MAX_WORDS][50];
char word[50];


// Count of distinct words
int count[MAX_WORDS];


/* Input file path */
printf("Enter file path: ");
scanf("%s", path);


/* Try to open file */
fptr = fopen(path, "r");


/* Exit if file not opened successfully */
if (fptr == NULL)
{
    printf("Unable to open file.\n");
}

```

```
printf("Please check you have read previleges.\n");
```

```
exit(EXIT_FAILURE);
```

```
}
```

```
// Initialize words count to 0
```

```
for (i=0; i<MAX_WORDS; i++)
```

```
count[i] = 0;
```

```
index = 0;
```

```
while (fscanf(fp, "%s", word) != EOF)
```

```
{
```

```
    // Convert word to lowercase
```

```
    strlwr(word);
```

```
    // Remove last punctuation character
```

```
    len = strlen(word);
```

```
    if (ispunct(word[len - 1]))
```

```
        word[len - 1] = '\0';
```

```
// Check if word exists in list of all distinct words
```

```
isUnique = 1;
```

```
for (i=0; i<index && isUnique; i++)
```

```
{
```

```
    if (strcmp(words[i], word) == 0)
```

```
        isUnique = 0;
```

```
}
```

```
// If word is unique then add it to distinct words list
```

```
// and increment index. Otherwise increment occurrence
```

```
// count of current word.
```

```
if (isUnique)
```

```
{
```

```
    strcpy(words[index], word);
```

```
    count[index]++;
```

```
    index++;
```

```
}
```

```
else
```

```
{
```

```
    count[i - 1]++;
```

```
}
```

```
}
```

```

// Close file
fclose(fp);

/*
 * Print occurrences of all words in file.
 */
printf("\nOccurrences of all distinct words in file: \n");
for (i=0; i<index; i++)
{
    /*
     * %-15s prints string in 15 character width.
     * - is used to print string left align inside
     * 15 character width space.
     */
    printf("%-15s => %d\n", words[i], count[i]);
}

return 0;
}

```

Write a C program to remove all empty lines from a file. How to remove all empty lines from a given file in C programming. Logic to remove empty lines from file in C program.

```
#include <stdio.h>

#include <stdlib.h>


#define BUFFER_SIZE 1000


/* Function declarations */
int isEmpty(const char *str);
void removeEmptyLines(FILE *srcFile, FILE *tempFile);
void printFile(FILE *fptr);


int main()
{
    FILE *srcFile;
    FILE *tempFile;


    char path[100];


    /* Input file path */
    printf("Enter file path: ");
    scanf("%s", path);
```

```
/* Try to open file */
```

```
srcFile = fopen(path, "r");
```

```
tempFile = fopen("remove-blanks.tmp", "w");
```

```
/* Exit if file not opened successfully */
```

```
if (srcFile == NULL || tempFile == NULL)
```

```
{
```

```
    printf("Unable to open file.\n");
```

```
    printf("Please check you have read/write privileges.\n");
```

```
    exit(EXIT_FAILURE);
```

```
}
```

```
printf("\nFile contents before removing all empty lines.\n\n");
```

```
printFile(srcFile);
```

```
// Move src file pointer to beginning
```

```
rewind(srcFile);
```

```
// Remove empty lines from file.
```



```
removeEmptyLines(srcFile, tempFile);
```

```
/* Close all open files */
```

```
fclose(srcFile);
```

```
fclose(tempFile);
```

```
/* Delete src file and rename temp file as src */
```

```
remove(path);
```

```
rename("remove-blanks.tmp", path);
```

```
printf("\n\nFile contents after removing all empty line.\n\n");
```

```
// Open source file and print its contents
```

```
srcFile = fopen(path, "r");
```

```
printFile(srcFile);
```

```
fclose(srcFile);
```

```
return 0;
```

```
}
```

```
/**
 * Print contents of a file.
 */
void printFile(FILE *fptr)
{
    char ch;

    while((ch = fgetc(fptr)) != EOF)
        putchar(ch);
}
```

```
/**
 * Checks, whether a given string is empty or not.
 * A string is empty if it only contains white space
 * characters.
 *
 * Returns 1 if given string is empty otherwise 0.
 */
int isEmpty(const char *str)
{
    char ch;
```

```

do
{
    ch = *(str++);

    // Check non whitespace character
    if(ch != ' ' && ch != '\t' && ch != '\n' && ch != '\r' && ch != '\0')
        return 0;

} while (ch != '\0');

return 1;
}

/**
 * Function to remove empty lines from a file.
 */
void removeEmptyLines(FILE *srcFile, FILE *tempFile)
{
    char buffer[BUFFER_SIZE];

    while ((fgets(buffer, BUFFER_SIZE, srcFile)) != NULL)
    {

```

```
    /* If current line is not empty then write to temporary file */  
    if(!isEmpty(buffer))  
        fputs(buffer, tempFile);  
}  
}
```

Output

Output:

Enter file path: data/blanks.txt

File contents

I love programming.

I am learning C programming at Codeforwin.

Programming with files is fun.

Learning C programming at Codeforwin is simple and easy.

before removing all empty lines.

17. Memory Management

➤ Memory Management

- This chapter explains dynamic memory management in C. The C programming language provides several functions for memory allocation and management. These functions can be found in the <stdlib.h> header file.

- Sr.No. • Function & Description

- 1 • void *calloc(int num, int size);

- This function allocates an array of num elements each of which size in bytes will be size.

- 2 • void free(void *address);

- This function releases a block of memory block specified by address.

- 3 • void *malloc(int num);

- This function allocates an array of num bytes and leave them uninitialized.

- 4 • void *realloc(void *address, int newsize);

- This function re-allocates memory extending it upto newsize.

- Allocating Memory Dynamically

- While programming, if you are aware of the size of an array, then it is easy and you can define it as an array. For example, to store a name of any person, it can go up to a maximum of 100 characters, so you can define something as follows –

- char name[100];

- But now let us consider a situation where you have no idea about the length of the text you need to store, for example, you want to store a detailed description about a topic. Here we need to define a pointer to character without defining how much memory is required and later, based on requirement, we can allocate memory as shown in the below example –

- #include <stdio.h>

- `#include <stdlib.h>`
- `#include <string.h>`
-
- `int main() {`
-
- `char name[100];`
- `char *description;`
-
- `strcpy(name, "Zara Ali");`
-
- `/* allocate memory dynamically */`
- `description = malloc(200 * sizeof(char));`
-
- `if(description == NULL) {`
- `fprintf(stderr, "Error - unable to allocate required memory\n");`
- `} else {`
- `strcpy(description, "Zara ali a DPS student in class 10th");`
- `}`
-
- `printf("Name = %s\n", name);`
- `printf("Description: %s\n", description);`
- `}`
- When the above code is compiled and executed, it produces the following result.
- Name = Zara Ali

- Description: Zara ali a DPS student in class 10th
- Same program can be written using calloc(); only thing is you need to replace malloc with calloc as follows –
- `calloc(200, sizeof(char));`
- So you have complete control and you can pass any size value while allocating memory, unlike arrays where once the size defined, you cannot change it.
- Resizing and Releasing Memory
- When your program comes out, operating system automatically release all the memory allocated by your program but as a good practice when you are not in need of memory anymore then you should release that memory by calling the function `free()`.
- Alternatively, you can increase or decrease the size of an allocated memory block by calling the function `realloc()`. Let us check the above program once again and make use of `realloc()` and `free()` functions –
- `#include <stdio.h>`
- `#include <stdlib.h>`
- `#include <string.h>`
-
- `int main() {`
-
- `char name[100];`
- `char *description;`
-
- `strcpy(name, "Zara Ali");`
-
- `/* allocate memory dynamically */`

- `description = malloc(30 * sizeof(char));`
- `if(description == NULL) {`
- `fprintf(stderr, "Error - unable to allocate required memory\n");`
- `} else {`
- `strcpy(description, "Zara ali a DPS student.");`
- `}`
- `/* suppose you want to store bigger description */`
- `description = realloc(description, 100 * sizeof(char));`
- `if(description == NULL) {`
- `fprintf(stderr, "Error - unable to allocate required memory\n");`
- `} else {`
- `strcat(description, "She is in class 10th");`
- `}`
- `printf("Name = %s\n", name);`
- `printf("Description: %s\n", description);`
- `/* release memory using free() function */`
- `free(description);`
- `}`
- When the above code is compiled and executed, it produces the following result.
- Name = Zara Ali
- Description: Zara ali a DPS student.She is in class 10th
- You can try the above example without re-allocating extra memory, and `strcat()` function will give an error due to lack of available memory in description.

18. C Dynamic Memory

➤ Dynamic Memory

Dynamic memory allocation in C

The concept of dynamic memory allocation in c language enables the C programmer to allocate memory at runtime. Dynamic memory allocation in c language is possible by 4 functions of stdlib.h header file.

1. malloc()
2. calloc()
3. realloc()
4. free()

Before learning above functions, let's understand the difference between static memory allocation and dynamic memory allocation.

static memory allocation dynamic memory allocation

memory is allocated at compile time. memory is allocated at run time.

memory can't be increased while executing program. memory can be increased while executing program.

used in array. used in linked list.

Now let's have a quick look at the methods used for dynamic memory allocation.

malloc() allocates single block of requested memory.

calloc() allocates multiple block of requested memory.

realloc() reallocates the memory occupied by malloc() or calloc() functions.

free() frees the dynamically allocated memory.

malloc() function in C

The malloc() function allocates single block of requested memory.

It doesn't initialize memory at execution time, so it has garbage value initially.

It returns NULL if memory is not sufficient.

The syntax of malloc() function is given below:

1. ptr=(cast-type*)malloc(byte-size)

Let's see the example of malloc() function.

```
1.  #include<stdio.h>
2.  #include<stdlib.h>
3.  int main(){
4.      int n,i,*ptr,sum=0;
5.      printf("Enter number of elements: ");
6.      scanf("%d",&n);
7.      ptr=(int*)malloc(n*sizeof(int)); //memory allocated using malloc
8.      if(ptr==NULL)
9.      {
10.         printf("Sorry! unable to allocate memory");
11.         exit(0);
12.     }
13.     printf("Enter elements of array: ");
14.     for(i=0;i<n;++i)
15.     {
16.         scanf("%d",ptr+i);
17.         sum+=*(ptr+i);
18.     }
19.     printf("Sum=%d",sum);
20.     free(ptr);
```

21. return 0;

22. }

Output

Enter elements of array: 3

Enter elements of array: 10

10

10

Sum=30

calloc() function in C

The calloc() function allocates multiple block of requested memory.

It initially initialize all bytes to zero.

It returns NULL if memory is not sufficient.

The syntax of calloc() function is given below:

1. ptr=(cast-type*)calloc(number, byte-size)

Let's see the example of calloc() function.

1. #include<stdio.h>

2. #include<stdlib.h>

3. int main(){

4. int n,i,*ptr,sum=0;

5. printf("Enter number of elements: ");

6. scanf("%d",&n);

7. ptr=(int*)calloc(n,sizeof(int)); //memory allocated using calloc

8. if(ptr==NULL)

9. {

```

10.     printf("Sorry! unable to allocate memory");
11.     exit(0);
12. }
13.     printf("Enter elements of array: ");
14.     for(i=0;i<n;++i)
15.     {
16.         scanf("%d",ptr+i);
17.         sum+=*(ptr+i);
18.     }
19.     printf("Sum=%d",sum);
20.     free(ptr);
21.     return 0;
22. }

```

Output

Enter elements of array: 3

Enter elements of array: 10

10

10

Sum=30

realloc() function in C

If memory is not sufficient for malloc() or calloc(), you can reallocate the memory by realloc() function. In short, it changes the memory size.

Let's see the syntax of realloc() function.

```
1.    ptr=realloc(ptr, new-size)
```

free() function in C

The memory occupied by `malloc()` or `calloc()` functions must be released by calling `free()` function. Otherwise, it will consume memory until program exit.

Let's see the syntax of `free()` function.

1. `free(ptr)`

19. C - Header Files(inportant)

➤ Header file

A header file is a file with extension .h which contains C function declarations and macro definitions to be shared between several source files. There are two types of header files: the files that the programmer writes and the files that comes with your compiler.

You request to use a header file in your program by including it with the C preprocessing directive `#include`, like you have seen inclusion of `stdio.h` header file, which comes along with your compiler.

Including a header file is equal to copying the content of the header file but we do not do it because it will be error-prone and it is not a good idea to copy the content of a header file in the source files, especially if we have multiple source files in a program.

A simple practice in C or C++ programs is that we keep all the constants, macros, system wide global variables, and function prototypes in the header files and include that header file wherever it is required.

Include Syntax

Both the user and the system header files are included using the preprocessing directive `#include`. It has the following two forms –

`#include <file>`

This form is used for system header files. It searches for a file named 'file' in a standard list of system directories. You can prepend directories to this list with the `-I` option while compiling your source code.

`#include "file"`

This form is used for header files of your own program. It searches for a file named 'file' in the directory containing the current file. You can prepend directories to this list with the `-I` option while compiling your source code.

Include Operation

The `#include` directive works by directing the C preprocessor to scan the specified file as input before continuing with the rest of the current source file. The output from the preprocessor contains the output already generated, followed by the output resulting from the included file, followed by the output that comes from the text after the `#include` directive. For example, if you have a header file `header.h` as follows –

```
char *test (void);
```

and a main program called `program.c` that uses the header file, like this –

```
int x;
```

```
#include "header.h"
```

```
int main (void) {
```

```
    puts (test ());
```

```
}
```

the compiler will see the same token stream as it would if `program.c` read.

```
int x;
```

```
char *test (void);
```

```
int main (void) {
```

```
    puts (test ());
```

```
}
```

Once-Only Headers

If a header file happens to be included twice, the compiler will process its contents twice and it will result in an error. The standard way to prevent this is to enclose the entire real contents of the file in a conditional, like this –

```
#ifndef HEADER_FILE
```

```
#define HEADER_FILE
```

the entire header file file

```
#endif
```

This construct is commonly known as a wrapper `#ifndef`. When the header is included again, the conditional will be false, because `HEADER_FILE` is defined. The preprocessor will skip over the entire contents of the file, and the compiler will not see it twice.

Computed Includes

Sometimes it is necessary to select one of the several different header files to be included into your program. For instance, they might specify configuration parameters to be used on different sorts of operating systems. You could do this with a series of conditionals as follows –

```
#if SYSTEM_1
    #include "system_1.h"
#elif SYSTEM_2
    #include "system_2.h"
#elif SYSTEM_3
    ...
#endif
```

But as it grows, it becomes tedious, instead the preprocessor offers the ability to use a macro for the header name. This is called a computed include. Instead of writing a header name as the direct argument of `#include`, you simply put a macro name there –

```
#define SYSTEM_H "system_1.h"
...
#include SYSTEM_H
```


SYSTEM_H will be expanded, and the preprocessor will look for system_1.h as if the #include had been written that way originally. SYSTEM_H could be defined by your Makefile with a -D option.

C #include

The #include preprocessor directive is used to paste code of given file into current file. It is used include system-defined and user-defined header files. If included file is not found, compiler renders error.

By the use of #include directive, we provide information to the preprocessor where to look for the header files. There are two variants to use #include directive.

1. #include <filename>
2. #include "filename"

The #include <filename> tells the compiler to look for the directory where system header files are held. In UNIX, it is \usr\include directory.

The #include "filename" tells the compiler to look in the current directory from where program is running.

#include directive example

Let's see a simple example of #include directive. In this program, we are including stdio.h file because printf() function is defined in this file.

1. #include<stdio.h>
2. int main(){
3. printf("Hello C");
4. return 0;
5. }

Output:

Hello C

#include notes:

Note 1: In #include directive, comments are not recognized. So in case of #include <a//b>, a//b is treated as filename.

Note 2: In #include directive, backslash is considered as normal text not escape sequence. So in case of #include <a\nb>, a\nb is treated as filename.

Note 3: You can use only comment after filename otherwise it will give error.

C #define

The #define preprocessor directive is used to define constant or micro substitution. It can use any basic data type.

Syntax:

1. #define token value

Let's see an example of #define to define a constant.

1. #include <stdio.h>
2. #define PI 3.14
3. main() {
4. printf("%f",PI);
5. }

Output:

3.140000

Let's see an example of #define to create a macro.

1. #include <stdio.h>
2. #define MIN(a,b) ((a)<(b)?(a):(b))
3. void main() {
4. printf("Minimum between 10 and 20 is: %d\n", MIN(10,20));
5. }

Output:

Minimum between 10 and 20 is: 10

C #undef

The #undef preprocessor directive is used to undefine the constant or macro defined by #define.

Syntax:

1. #undef token

Let's see a simple example to define and undefine a constant.

1. #include <stdio.h>
2. #define PI 3.14
3. #undef PI
4. main() {
5. printf("%f",PI);
6. }

Output:

Compile Time Error: 'PI' undeclared

The #undef directive is used to define the preprocessor constant to a limited scope so that you can declare constant again.

Let's see an example where we are defining and undefining number variable. But before being undefined, it was used by square variable.

1. #include <stdio.h>
2. #define number 15
3. int square=number*number;
4. #undef number
5. main() {
6. printf("%d",square);

7. }

Output:

225

C #ifdef

The #ifdef preprocessor directive checks if macro is defined by #define. If yes, it executes the code otherwise #else code is executed, if present.

Syntax:

1. #ifdef MACRO
2. //code
3. #endif

Syntax with #else:

1. #ifdef MACRO
2. //successful code
3. #else
4. //else code
5. #endif

C #ifdef example

Let's see a simple example to use #ifdef preprocessor directive.

1. #include <stdio.h>
2. #include <conio.h>
3. #define NOINPUT
4. void main() {
5. int a=0;
6. #ifdef NOINPUT
7. a=2;

```
8.  #else
9.  printf("Enter a:");
10. scanf("%d", &a);
11. #endif
12. printf("Value of a: %d\n", a);
13. getch();
14. }
```

Output:

Value of a: 2

But, if you don't define NOINPUT, it will ask user to enter a number.

```
1.  #include <stdio.h>
2.  #include <conio.h>
3.  void main() {
4.  int a=0;
5.  #ifdef NOINPUT
6.  a=2;
7.  #else
8.  printf("Enter a:");
9.  scanf("%d", &a);
10. #endif
11.
12. printf("Value of a: %d\n", a);
13. getch();
14. }
```

Output:

Enter a:5

Value of a: 5

C #ifndef

The #ifndef preprocessor directive checks if macro is not defined by #define. If yes, it executes the code otherwise #else code is executed, if present.

Syntax:

1. #ifndef MACRO
2. //code
3. #endif

Syntax with #else:

1. #ifndef MACRO
2. //successful code
3. #else
4. //else code
5. #endif

C #ifndef example

Let's see a simple example to use #ifndef preprocessor directive.

1. #include <stdio.h>
2. #include <conio.h>
3. #define INPUT
4. void main() {
5. int a=0;
6. #ifndef INPUT
7. a=2;

```
8.  #else
9.  printf("Enter a:");
10. scanf("%d", &a);
11. #endif
12. printf("Value of a: %d\n", a);
13. getch();
14. }
```

Output:

Enter a:5

Value of a: 5

But, if you don't define INPUT, it will execute the code of #ifndef.

```
1.  #include <stdio.h>
2.  #include <conio.h>
3.  void main() {
4.  int a=0;
5.  #ifndef INPUT
6.  a=2;
7.  #else
8.  printf("Enter a:");
9.  scanf("%d", &a);
10. #endif
11. printf("Value of a: %d\n", a);
12. getch();
13. }
```

Output:

Value of a: 2

C #if

The #if preprocessor directive evaluates the expression or condition. If condition is true, it executes the code otherwise #elseif or #else or #endif code is executed.

Syntax:

1. #if expression
2. //code
3. #endif

Syntax with #else:

1. #if expression
2. //if code
3. #else
4. //else code
5. #endif

Syntax with #elif and #else:

1. #if expression
2. //if code
3. #elif expression
4. //elif code
5. #else
6. //else code
7. #endif

C #if example

Let's see a simple example to use #if preprocessor directive.


```
1.  #include <stdio.h>
2.  #include <conio.h>
3.  #define NUMBER 0
4.  void main() {
5.      #if (NUMBER==0)
6.          printf("Value of Number is: %d",NUMBER);
7.      #endif
8.      getch();
9.  }
```

Output:

Value of Number is: 0

Let's see another example to understand the #if directive clearly.

```
1.  #include <stdio.h>
2.  #include <conio.h>
3.  #define NUMBER 1
4.  void main() {
5.      clrscr();
6.      #if (NUMBER==0)
7.          printf("1 Value of Number is: %d",NUMBER);
8.      #endif
9.
10.     #if (NUMBER==1)
11.         printf("2 Value of Number is: %d",NUMBER);
12.     #endif
```

13. getch();

14. }

Output:

2 Value of Number is: 1

C #else

The #else preprocessor directive evaluates the expression or condition if condition of #if is false. It can be used with #if, #elif, #ifdef and #ifndef directives.

Syntax:

1. #if expression
2. //if code
3. #else
4. //else code
5. #endif

Syntax with #elif:

1. #if expression
2. //if code
3. #elif expression
4. //elif code
5. #else
6. //else code
7. #endif

C #else example

Let's see a simple example to use #else preprocessor directive.

1. #include <stdio.h>
2. #include <conio.h>

```

3.  #define NUMBER 1
4.  void main() {
5.  #if NUMBER==0
6.  printf("Value of Number is: %d",NUMBER);
7.  #else
8.  print("Value of Number is non-zero");
9.  #endif
10. getch();
11. }

```

Output:

Value of Number is non-zero

C #error

The #error preprocessor directive indicates error. The compiler gives fatal error if #error directive is found and skips further compilation process.

C #error example

Let's see a simple example to use #error preprocessor directive.

```

1.  #include<stdio.h>
2.  #ifndef __MATH_H
3.  #error First include then compile
4.  #else
5.  void main(){
6.      float a;
7.      a=sqrt(7);
8.      printf("%f",a);
9.  }

```

10. `#endif`

Output:

Compile Time Error: First include then compile

But, if you include `math.h`, it does not gives error.

```
1.  #include<stdio.h>
2.  #include<math.h>
3.  #ifndef __MATH_H
4.  #error First include then compile
5.  #else
6.  void main(){
7.      float a;
8.      a=sqrt(7);
9.      printf("%f",a);
10. }
11. #endif
```

Output:

2.645751

C `#pragma`

The `#pragma` preprocessor directive is used to provide additional information to the compiler. The `#pragma` directive is used by the compiler to offer machine or operating-system feature.

Syntax:

```
1.  #pragma token
```

Different compilers can provide different usage of `#pragma` directive.

The turbo C++ compiler supports following `#pragma` directives.

1. `#pragma argsused`
2. `#pragma exit`
3. `#pragma hdrfile`
4. `#pragma hdrstop`
5. `#pragma inline`
6. `#pragma option`
7. `#pragma saveregs`
8. `#pragma startup`
9. `#pragma warn`

Let's see a simple example to use `#pragma` preprocessor directive.

1. `#include<stdio.h>`
2. `#include<conio.h>`
- 3.
4. `void func() ;`
- 5.
6. `#pragma startup func`
7. `#pragma exit func`
- 8.
9. `void main(){`
10. `printf("\nI am in main");`
11. `getch();`
12. `}`
- 13.
14. `void func(){`

15. `printf("\nI am in func");`

16. `getch();`

17. `}`

Output:

I am in func

I am in main

I am in func

20. C - Error Handling

➤ Error Handling

As such, C programming does not provide direct support for error handling but being a system programming language, it provides you access at lower level in the form of return values. Most of the C or even Unix function calls return -1 or NULL in case of any error and set an error code `errno`. It is set as a global variable and indicates an error occurred during any function call. You can find various error codes defined in `<error.h>` header file.

So a C programmer can check the returned values and can take appropriate action depending on the return value. It is a good practice, to set `errno` to 0 at the time of initializing a program. A value of 0 indicates that there is no error in the program.

`errno`, `perror()`, and `strerror()`

The C programming language provides `perror()` and `strerror()` functions which can be used to display the text message associated with `errno`.

- The `perror()` function displays the string you pass to it, followed by a colon, a space, and then the textual representation of the current `errno` value.
- The `strerror()` function, which returns a pointer to the textual representation of the current `errno` value.

Let's try to simulate an error condition and try to open a file which does not exist. Here I'm using both the functions to show the usage, but you can use one or more ways of printing your errors. Second important point to note is that you should use `stderr` file stream to output all the errors.

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

```
#include <errno.h>
```

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

```
extern int errno ;
```

```

int main () {

    FILE * pf;

    int errnum;

    pf = fopen ("unexist.txt", "rb");

    if (pf == NULL) {

        errnum = errno;
        fprintf(stderr, "Value of errno: %d\n", errno);
        perror("Error printed by perror");
        fprintf(stderr, "Error opening file: %s\n", strerror( errnum ));
    } else {

        fclose (pf);
    }

    return 0;
}

```

When the above code is compiled and executed, it produces the following result –

Value of errno: 2

Error printed by perror: No such file or directory

Error opening file: No such file or directory

Divide by Zero Errors

It is a common problem that at the time of dividing any number, programmers do not check if a divisor is zero and finally it creates a runtime error.

The code below fixes this by checking if the divisor is zero before dividing –

```
#include <stdio.h>

#include <stdlib.h>

main() {

    int dividend = 20;
    int divisor = 0;
    int quotient;

    if( divisor == 0){
        fprintf(stderr, "Division by zero! Exiting...\n");
        exit(-1);
    }

    quotient = dividend / divisor;
    fprintf(stderr, "Value of quotient : %d\n", quotient );

    exit(0);
}
```

When the above code is compiled and executed, it produces the following result –

Division by zero! Exiting...

Program Exit Status

It is a common practice to exit with a value of EXIT_SUCCESS in case of program coming out after a successful operation. Here, EXIT_SUCCESS is a macro and it is defined as 0.

If you have an error condition in your program and you are coming out then you should exit with a status EXIT_FAILURE which is defined as -1. So let's write above program as follows –

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

main() {

    int dividend = 20;
    int divisor = 5;
    int quotient;
    if( divisor == 0) {
        fprintf(stderr, "Division by zero! Exiting...\n");
        exit(EXIT_FAILURE);
    }
    quotient = dividend / divisor;
    fprintf(stderr, "Value of quotient : %d\n", quotient );

    exit(EXIT_SUCCESS);
}
```

When the above code is compiled and executed, it produces the following result –

Value of quotient : 4

21. C Files I/O

➤ File I/O

- The last chapter explained the standard input and output devices handled by C programming language. This chapter cover how C programmers can create, open, close text or binary files for their data storage.
- A file represents a sequence of bytes, regardless of it being a text file or a binary file. C programming language provides access on high level functions as well as low level (OS level) calls to handle file on your storage devices. This chapter will take you through the important calls for file management.
- Opening Files
- You can use the `fopen()` function to create a new file or to open an existing file. This call will initialize an object of the type `FILE`, which contains all the information necessary to control the stream. The prototype of this function call is as follows –
 - `FILE *fopen(const char * filename, const char * mode);`
 - Here, `filename` is a string literal, which you will use to name your file, and access mode can have one of the following values –
- | Sr.No. | Mode | Description |
|--------|-----------------|---|
| 1 | <code>r</code> | Opens an existing text file for reading purpose. |
| 2 | <code>w</code> | Opens a text file for writing. If it does not exist, then a new file is created. Here your program will start writing content from the beginning of the file. |
| 3 | <code>a</code> | Opens a text file for writing in appending mode. If it does not exist, then a new file is created. Here your program will start appending content in the existing file content. |
| 4 | <code>r+</code> | |

- Opens a text file for both reading and writing.
- 5 • w+
- Opens a text file for both reading and writing. It first truncates the file to zero length if it exists, otherwise creates a file if it does not exist.
- 6 • a+
- Opens a text file for both reading and writing. It creates the file if it does not exist. The reading will start from the beginning but writing can only be appended.
- If you are going to handle binary files, then you will use following access modes instead of the above mentioned ones –
- "rb", "wb", "ab", "rb+", "r+b", "wb+", "w+b", "ab+", "a+b"
- Closing a File
- To close a file, use the `fclose()` function. The prototype of this function is –
- `int fclose(FILE *fp);`
- The `fclose(-)` function returns zero on success, or EOF if there is an error in closing the file. This function actually flushes any data still pending in the buffer to the file, closes the file, and releases any memory used for the file. The EOF is a constant defined in the header file `stdio.h`.
- There are various functions provided by C standard library to read and write a file, character by character, or in the form of a fixed length string.
- Writing a File
- Following is the simplest function to write individual characters to a stream –
- `int fputc(int c, FILE *fp);`
- The function `fputc()` writes the character value of the argument `c` to the output stream referenced by `fp`. It returns the written character written on success otherwise EOF if there is an error. You can use the following functions to write a null-terminated string to a stream –
- `int fputs(const char *s, FILE *fp);`

- The function `fputs()` writes the string `s` to the output stream referenced by `fp`. It returns a non-negative value on success, otherwise EOF is returned in case of any error. You can use `int fprintf(FILE *fp, const char *format, ...)` function as well to write a string into a file. Try the following example.

- Make sure you have `/tmp` directory available. If it is not, then before proceeding, you must create this directory on your machine.

- `#include <stdio.h>`

-

- `main() {`

- `FILE *fp;`

-

- `fp = fopen("/tmp/test.txt", "w+");`

- `fprintf(fp, "This is testing for fprintf...\n");`

- `fputs("This is testing for fputs...\n", fp);`

- `fclose(fp);`

- `}`

- When the above code is compiled and executed, it creates a new file `test.txt` in `/tmp` directory and writes two lines using two different functions. Let us read this file in the next section.

- Reading a File

- Given below is the simplest function to read a single character from a file –

- `int fgetc(FILE * fp);`

- The `fgetc()` function reads a character from the input file referenced by `fp`. The return value is the character read, or in case of any error, it returns EOF. The following function allows to read a string from a stream –

- `char *fgets(char *buf, int n, FILE *fp);`

- The functions `fgets()` reads up to `n-1` characters from the input stream referenced by `fp`. It copies the read string into the buffer `buf`, appending a null character to terminate the string.
- If this function encounters a newline character `'\n'` or the end of the file EOF before they have read the maximum number of characters, then it returns only the characters read up to that point including the new line character. You can also use `int fscanf(FILE *fp, const char *format, ...)` function to read strings from a file, but it stops reading after encountering the first space character.
- `#include <stdio.h>`
-
- `main() {`
-
- `FILE *fp;`
- `char buff[255];`
-
- `fp = fopen("/tmp/test.txt", "r");`
- `fscanf(fp, "%s", buff);`
- `printf("1 : %s\n", buff);`
-
- `fgets(buff, 255, (FILE*)fp);`
- `printf("2: %s\n", buff);`
-
- `fgets(buff, 255, (FILE*)fp);`
- `printf("3: %s\n", buff);`
- `fclose(fp);`
-

- }
- When the above code is compiled and executed, it reads the file created in the previous section and produces the following result –
- 1 : This
- 2: is testing for fprintf...
-
- 3: This is testing for fputs...
- Let's see a little more in detail about what happened here. First, fscanf() read just This because after that, it encountered a space, second call is for fgets() which reads the remaining line till it encountered end of line. Finally, the last call fgets() reads the second line completely.
- Binary I/O Functions
- There are two functions, that can be used for binary input and output –
- size_t fread(void *ptr, size_t size_of_elements, size_t number_of_elements, FILE *a_file);
-
- size_t fwrite(const void *ptr, size_t size_of_elements, size_t number_of_elements, FILE *a_file);
- Both of these functions should be used to read or write blocks of memories - usually arrays or structures.

22. Type Casting in C

➤ Type Casting

Converting one datatype into another is known as type casting or, type-conversion. For example, if you want to store a 'long' value into a simple integer then you can type cast 'long' to 'int'. You can convert the values from one type to another explicitly using the cast operator as follows –

(type_name) expression

Consider the following example where the cast operator causes the division of one integer variable by another to be performed as a floating-point operation –

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

```
main() {
```

```
int sum = 17, count = 5;
```

```
double mean;
```

```
mean = (double) sum / count;
```

```
printf("Value of mean : %f\n", mean );
```

```
}
```

When the above code is compiled and executed, it produces the following result –

```
Value of mean : 3.400000
```

It should be noted here that the cast operator has precedence over division, so the value of sum is first converted to type double and finally it gets divided by count yielding a double value.

Type conversions can be implicit which is performed by the compiler automatically, or it can be specified explicitly through the use of the cast operator. It is considered

good programming practice to use the cast operator whenever type conversions are necessary.

Integer Promotion

Integer promotion is the process by which values of integer type "smaller" than int or unsigned int are converted either to int or unsigned int. Consider an example of adding a character with an integer –

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

```
main() {
```

```
int i = 17;
```

```
char c = 'c'; /* ascii value is 99 */
```

```
int sum;
```

```
sum = i + c;
```

```
printf("Value of sum : %d\n", sum );
```

```
}
```

When the above code is compiled and executed, it produces the following result –

Value of sum : 116

Here, the value of sum is 116 because the compiler is doing integer promotion and converting the value of 'c' to ASCII before performing the actual addition operation.

Usual Arithmetic Conversion

The usual arithmetic conversions are implicitly performed to cast their values to a common type. The compiler first performs integer promotion; if the operands still have different types, then they are converted to the type that appears highest in the following hierarchy –

The usual arithmetic conversions are not performed for the assignment operators, nor for the logical operators && and ||. Let us take the following example to understand the concept –

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

```
main() {
```

```
int i = 17;
```

```
char c = 'c'; /* ascii value is 99 */
```

```
float sum;
```

```
sum = i + c;
```

```
printf("Value of sum : %f\n", sum );
```

```
}
```

When the above code is compiled and executed, it produces the following result –

```
Value of sum : 116.000000
```

Here, it is simple to understand that first c gets converted to integer, but as the final value is double, usual arithmetic conversion applies and the compiler converts i and c into 'float' and adds them yielding a 'float' result.