C PROGRAMMING AND DATA STRUCTURES

UNIT-I

INTRODUCTION TO C LANGUAGE

Topics:-

1. C Language elements
2. Variable declarations and data types
3. Operators and expressions
4. Decision statements - If and switch statements
5. Loop control statements - while, for, do-while statements
6. Arrays

Topic 1 : C Language elements

C Character Set:-

C character set consists of

1. Alphabets – A to Z(upper case charecters), a to z(lower case charecters)
2. Digits- 0 to 9
3. Special Symbols - special charecters (: ; [ ] { } ( ) # ? ‘ “ $ % _ etc)
   white spaces(blank space, newline,horizontal tab, etc)

C Language elements(C Tokens)

C tokens are the basic buildings blocks in C language which are constructed together to write a C program.Each and every smallest individual units in a C program are known as C tokens. (or) The smallest individual unit in a C program is called token.
There are 6 tokens in C language.

1. Keywords
2. Identifiers
3. Constants
4. Operators
5. Strings
6. Special symbols

Keywords:- All predefined names or words which cannot be changed are called keywords. There are a total of 32 keywords in 'C'. Keywords are written in lowercase letters.

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

Identifiers:- All user defined names or words are called identifiers.

Ex:- variable name, array name, function name, etc

Following rules must be followed for identifiers:

1. The first character must always be an alphabet or an underscore.
2. It should be formed using only letters, numbers, or underscore.
3. A keyword cannot be used as an identifier.
4. It should not contain any whitespace character.

Constants:- It is a value which cannot be changed during execution of the program.
Constants are categorized into two basic types. They are:-

1. Numeric Constants
   - Integer Constants
   - Real Constants
2. Character Constants
   - Single Character Constants
   - String Constants
   - Backslash Character Constants

Integer constants:- Integer constants are whole numbers without any fractional part. An integer constant is a sequence of digits.It must not have a decimal point.It can either be positive or negative.
Ex:- 19, -203, etc
There are three types of integer constants:

1. Decimal Integer Constants

Integer constants consisting of a set of digits, 0 through 9, preceded by an optional – or + sign.
Example of valid decimal integer constants
341, -341, 0, 8972

2. Octal Integer Constants

Integer constants consisting of sequence of digits from the set 0 through 7 starting with 0 is said to be octal integer constants.
Example of valid octal integer constants
010, 0424, 0, 0540

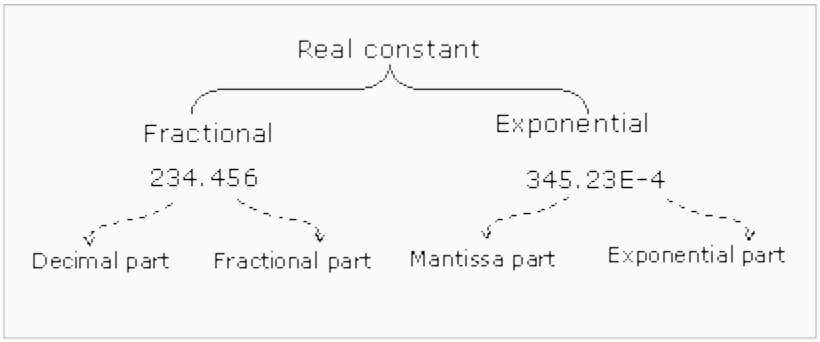
3. Hexadecimal Integer Constants

Hexadecimal integer constants are integer constants having sequence of digits preceded by 0x or 0X. They may also include alphabets from A to F representing numbers 10 to 15.
Example of valid hexadecimal integer constants
0xD, 0X8d, 0X, 0xbD

Real Constants:- A real constant is combination of a whole number followed by a decimal point and the fractional part. The numbers containing fractional parts like 99.25 are called real or floating points constant. It can either be positive or negative.
Ex:- 19.63, -56.142,etc
The Real or Floating-point constants can be written in two notations:

1. Fractional or Normal or decimal notation
2. Exponential or Scientific notation

Fractional form is the general form of representing a floating point value.
Exponential or scientific form is used when the value is so big or so small. Exponential form has two parts. The part before appearing to E is called mantissa and the part following E is called exponent. Here the mantissa must be multiplied by 10^exponent for example 345.25E-4 is equal to $345.25*10^{-4}$ and 4.56E4 is equal to $4.56*10^{4}$.



Note:- C language accepts floating point constants either in fractional or exponential form. The printf() prints in fractional form when %f is used as formatting character, prints in exponential form when %e is used as formatting character.

Example Program 1

```
#include<stdio.h>
int main()
{
printf("%f\n%e",56.78E-2,45.786);
return 0;
}
```

Output:-

```
0.567800
4.578600e+001
```

Example Program 2

```
#include<stdio.h>
int main()
{
float x,y,z;
printf("Enter the first real number:");
scanf("%f",&x);
printf("Enter the second real number:");
scanf("%f",&y);
z=x+y;
printf("Sum in fractional: %f",z);
printf("\nSum in exponential: %e",z);
return 0;
}
```

Input & Output:-

```
Enter the first real number:12675.34
Enter the second real number:145.658E-2
Sum in fractional: 12676.796875
Sum in exponential: 1.267680e+004
```

Single character constants:- It simply contains a single character enclosed within ' and ' (a pair of single quote). Character constants have a specific set of integer values known as ASCII values (American Standard Code for Information Interchange).
Ex:- 'X', '5', ';'

String constants:- These are a sequence of characters enclosed in double quotes, and they may include letters, digits, special characters, and blank spaces.
Ex:- “hello”, “8”, “123”, etc

Backslash Character Constants:-

C supports some character constants having a backslash in front of it. The lists of backslash characters have a specific meaning which is known to the compiler. They are also termed as "Escape Sequence".

Backslash_character	Meaning
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\"	Double quote
\'	Single quote
\\	Backslash
\v	Vertical tab
\a	Alert or bell
\?	Question mark

Operators:- An operator is a symbol (+, -, *, /) that directs the computer to perform certain mathematical or logical manipulations. (or) An operator is a symbol which is used to perform arithmetic or logical operation on given input data.

Ex:- addition(+), multiplication(*), subtraction(-), etc

String:- A string is an group of characters always enclosed with double quotes(" ").

Ex:- "Hello", "GIST", "1234" , etc

Special symbols:- Symbols other than the Alphabets and Digits are called Special symbols.

Special symbols are of two types:-

Special charecters:- The following are special charecters used in C have some special meaning and thus, cannot be used for some other purpose.

[ ] ( ) { } ; : # ' " are some special charecters used in the C program.

Curly Braces{ }: These opening and ending curly braces marks the start and end of a block of code containing more than one executable statement.

Paranthesis( ): These special symbols are used to indicate function calls and function definitions.

Square Brackets[ ]: Opening and closing brackets are used as array element reference. These indicate single and multidimensional subscripts.

White spaces or escape sequences or back slash character constants:- whitespace is any character or series of characters that represent horizontal or vertical space.
Ex:- \t, \n, \0, \r, \?, \', \" ,etc

C Tokens Example Program

```
main( )
{
  int x, y, total;
  x = 10, y = 20;
  total = x + y;
  printf ("Total = %d \n", total);
}
```

where,
{,}, (,),",%,; – special charecters (or) delimiters
int – keyword
x, y, total – identifiers
+,=, and comma(,) – operators
10,20 – Integer constants
\n- backslash character constant

Output:-
Total = 30

---***---

Topic 2 : Variable declarations and data types

Variable:- It is a name given to memory cell which is used to store value.

Syntax for declaring variable:-

data_type variable1, variable2,…,variablen;

Example:-
int i,j,k;
char c,ch;
float f,salary;
double d;

Intitialization of variables:- Assigning values or constants to a variables is called Initialization. It can be done in 2 ways.

i) At the time of compilation
ii) At the time of execution

At the time of compilation:-If we initialise a variable at the time of declaration, It is referred to as Compile time initialization.C variables declared can be initialized with the help of assignment operator '='.

Syntax:-variable_name=constant/expression;
At the time of execution:-If Initialization is done during execution of program , it is referred to as run time initialization. Scanf( ) function is used to initialize variable at run time.

Example:-

Compile time:-

int i = 10;

i is initialized as 10, Now value of i is 10

Run time:-

int i; /* No value is assigned to i */

scanf ("%d", & i); /* at execution time the value of i is assigned as the value we enter.*/

Note:- C variables must be declared before they are used in the c program.

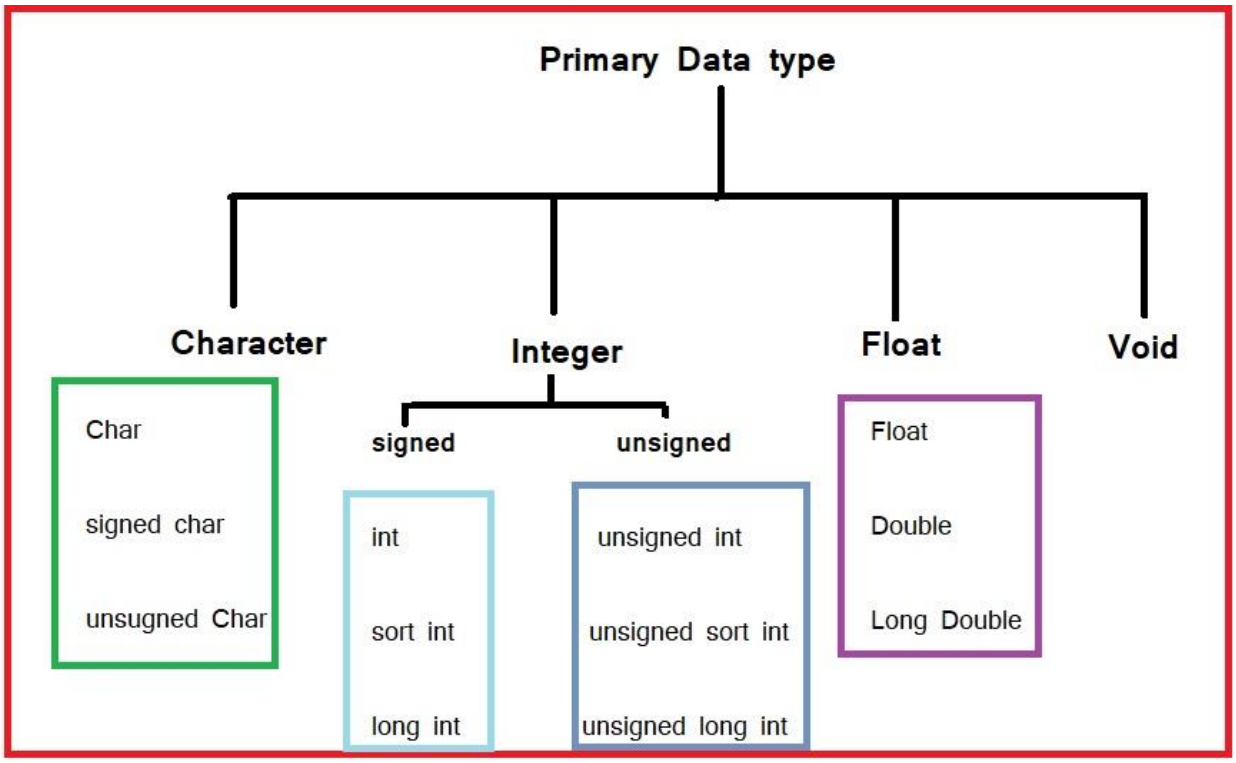
Data types

Data type enables the user to specify what type of data to be entered as input for program to do manipulation or operation. There are four data types in C language. They are:-

Data Types	
Basic (or) fundamental (or) primary data types-	integer, character, float, double
User defined data types -	type definition, enumerated
Derived data types-	pointer, array, structure, union
Void data type -	void

Most of the time, for small programs, we use the basic fundamental data types in C – int, char, float, and double.

For more complex and huge amounts of data, we use derived types – array, structure, union, and pointer.



Integer DataType:-
Integer data type allows a variable to store Integer constants.
Signed Integer :-

Short Integer	Long Integer
1. Occupies 2 bytes of memory	1. Occupies 4 bytes of memory
2. Control string - %d	2. Control string - %ld
3. Keyword – int or short int	3. Keyword – long int
4. Range:- -32768 to 32767	4. Range:- -2147483648 to 2147483647

UnSigned Integer:-

Short Integer	Long Integer
1. Occupies 2 bytes of memory	1.Occupies 2 bytes of memory
2. Control string - %u	2.Control string - %lu
3. Keyword – unsigned int	3. Keyword – unsigned long int
4. Range:- 0 to 65535	4. Range:- 0 to 4294967295

Float and double data types:- float and double data types allows a variable to store real constants.

Float	double
1. Occupies 4 bytes of memory	1.Occupies 8 bytes of memory
2. Control string - %f	2.Control string - %lf
3. Keyword – float	3. Keyword – double
4. Range:- 3.4e-38 to 3.4e+38	4. Range:- 1.7e-308 to 1.7e+308

Note:- C also supports long double data type.

Long double data type:- long double data type allows a variable to store real constants.

1.Occupies 10 bytes of memory
2.Control string - %Lf
3. Keyword – long double
4. Range:- 1.7e-4932 to 1.7e+4932

Character data type:- character data type allows variable to store only one character. charecter can be alphabet or digit or special symbol.

Signed charecter	Unsigned charecter
1. Occupies 1 byte of memory	1.Occupies 1 byte of memory
2. Control string - %c	2.Control string - %c
3. Keyword – char	3. Keyword – char
4. Range:- -128 to 127	4. Range:- 0 to 255

-----****-----

Topic 3 : Operators and Expressions

An operator is a symbol that performs operations on given data.
Types of Operators

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Comma operator
- Assignment operator
- Conditional or ternary operator

Arithmetic Operators:-An arithmetic operator performs mathematical operations such as addition, subtraction, multiplication, division etc on numerical values
There are two types of arithmetic operators.

- Unary arithmetic operators
- Binary arithmetic operators

Unary arithmetic operators:-unary operators operates on only single operand. The following are unary operators:

Unary minus(-):-unary minus operator produces the negative of its operand

Increment(++):-Increment operator increases the value of operand by 1.

There are two types of increment operators.

- Pre increment(++a):- In the Pre-Increment, value is first incremented and then used inside the expression.

```
#include <stdio.h>
main()
{
int data = 10;
int result = 0;
result = ++data;
printf("data = %d\n",data);
printf("result = %d\n",result);
}
```

Output:

```
data = 11
result = 11
```

- Post increment(a++):-In the Post-Increment, value is first used inside the expression and then incremented.

```
main()
{
int data = 10;
int result = 0;
result = data++;
printf("data = %d\n",data);
printf("result = %d\n",result);
}
```

Output:

```
data = 11
result = 10
```

Decrement:-Decrement operator decreases the value of operand by 1.
There are two types of decrement operators.

- Pre decrement(--a):- In the Pre-Increment, value is first incremented and then used inside the expression.

```
#include < stdio.h >
main()
{
   int a = 10, b;
    b = --a;
     printf("b = %d\n\n", b);
   printf("a = %d\n", a);
}
```

Output:

```
b = 9
a = 9
```

- Post decrement(a--):- In the Post-Increment, value is first used inside the expression and then incremented.

```
#include < stdio.h >
 main()
{
   int a = 10, b;
    b = a--;
     printf("b = %d\n\n", b);
   printf("a = %d\n", a);
 }
```

Output:

```
b = 10
a = 9
```

Sizeof :-The sizeof is a unary operator that returns the size of data (constants, variables, array, structure, etc) in form of bytes.

```
#include<stdio.h>
main()
{
int a;
float b;
double c;
char d;
printf("Size of int=%lu bytes\n",sizeof(a));
printf("Size of float=%lu bytes\n",sizeof(b));
printf("Size of double=%lu bytes\n",sizeof(c));
printf("Size of char=%lu byte\n",sizeof(d));
}
```

Output

Size of int = 4 bytes
Size of float = 4 bytes
Size of double = 8 bytes
Size of char = 1 byte

Address(&):-Every variable is a memory location and every memory location has its address defined which can be accessed using address (&) operator, which denotes an address in memory.

```
#include<stdio.h>
main ()
{
int  var1=10;
  printf("Address of var1 variable: %u\n",&var1);
  printf("Value of var1 variable: %d",var1);
}
```

Output:-
Address of var1 variable: bff5a400
Value of var1 variable: 10

Binary arithmetic operators:-There are five *binary arithmetic operators*: addition, subtraction, multiplication, division, and modular division(%).

Operator	Meaning of operator
+	addition
-	subtraction
*	multiplication
/	Division(quotient)
%	remainder after division (modulo division)

```
#include<stdio.h>
main( )
{
int a = 9,b = 4, c;

   c = a+b;
printf("a+b = %d \n",c);
   c = a-b;
printf("a-b = %d \n",c);
   c = a*b;
printf("a*b = %d \n",c);
   c = a/b;
printf("a/b = %d \n",c);
   c = a%b;
printf("Remainder when a divided by b = %d \n",c);
}
```

Output:-

```
a+b = 13
a-b = 5
a*b = 36
a/b = 2
Remainder when a divided by b=1
```

Relational Operators:- Relational operators are used to compare values of two variables. There are six relational operators. If the relation is true, then it will return value 1. Otherwise, it returns value 0.

RELATIONAL OPERATORS IN C	**USAGE**	**DESCRIPTION**	**EXAMPLE**
>	a > b	a is greater than b	7 > 3 returns true (1)
<	a < b	a is less than b	7 < 3 returns false (0)
>=	a >= b	a is greater than or equal to b	7 >= 3 returns true (1)
<=	a <= b	a is less than or equal to b	7 <= 3 return false (0)
==	a == b	a is equal to b	7 == 3 returns false (0)
!=	a != b	a is not equal to b	7 != 3 returns true(1)

```
#include <stdio.h>
main( )
{
 int a = 9;
 int b = 4;
 printf(" a >  b: %d \n", a > b);
 printf("a >= b: %d \n", a >= b);
 printf("a <= b: %d \n", a <= b);
 printf("a <  b: %d \n", a < b);
 printf("a == b: %d \n", a == b);
 printf("a != b: %d \n", a != b);
}
```

Output:-

```
a >  b: 1
a >= b: 1
a <= b: 0
a <  b: 0
a == b: 0
a != b: 1
```

Logical Operators:- Logical operators are used to compare two expressions. If comparision is true produces 1 as output otherwise 0 as output. There are 3 logical operators.

Operator	Meaning	Example
&&	Logical AND. True only if all operands are true	If c = 5 and d = 2 then, expression ((c==5) && (d>5)) equals to 0.
||	Logical OR. True only if either one operand is true	If c = 5 and d = 2 then, expression ((c==5) || (d>5)) equals to 1.
!	Logical NOT. True only if the operand is 0	If c = 5 then, expression !(c==5) equals to 0.

```
#include<stdio.h>
main( )
{
int a = 5, b = 10 ,ret;
   ret = ( (a <= b) || (a != b) );
   printf("Return value of above expression is %d\n",ret);
   ret = ( ( a < b) && (a == b ) );
   printf("Return value of above expression is %d\n",ret);
   ret = ! ( ( a < b) && (a == b ) );
   printf("Return value of above expression is %d\n",ret);
}
```

Output:-
Return value of above expression is 1
Return value of above expression is 0
Return value of above expression is 1

Bitwise Operators:- Bitwise Operators are used for manipulating data at the bit level. Bitwise operators cannot be directly applied to primitive data types such as float, double, etc. Always remember one thing that bitwise operators are mostly used with the integer data type.
There are six bitwise operators.

Operator	Meaning
&	Bitwise AND operator(Binary)
|	Bitwise OR operator(Binary)
^	Bitwise exclusive OR operator(Binary)
~	Binary One's Complement(unary)
<<	Left shift operator(unary)
>>	Right shift operator(unary)

Left Shift(<<) and Right shift(>>):- The bitwise shift operators move the bit values of a binary object. The left operand specifies the value to be shifted. The right operand specifies the number of positions that the bits in the value are to be shifted.

Operator	Usage
<<	Indicates the specified no. of bits are to be shifted to the left.
>>	Indicates the specified no. of bits are to be shifted to the right.

Left shift:-
The << operator fills vacated bits with zeros.
For example, if variable x has the value 8, the bit pattern (in 16-bit format) of x is:
0000000000001000
The expression x << 3 produces:
0000000001000000 (decimal value=64)

Right Shift:-
For example, if variable y has the value 12, the bit pattern(in 16-bit format) of y is:
0000000000001100
The expression y >> 2 produces:
0000000000000011 (decimal value=3)

Example Program:-

```
#include<stdio.h>
main( )
{
   int x=8,y=12,c,d;
   c=x<<3;
   printf("value of c is %d ",c);
   d=y>>2;
   printf("\nvalue of d is %d ",d);
}
```

Output:-
value of c is 64
value of d is 3

Bitwise AND,OR and XOR operators:-

x	y	x & y	x | y	x ^ y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Bitwise AND:-

12 = 00001100 (In Binary)

25 = 00011001 (In Binary)

Bit Operation of 12 and 25

 00001100

& 00011001

 _____

 00001000 = 8 (In decimal)

Bitwise OR:-
12 = 00001100 (In Binary)

25 = 00011001 (In Binary)

Bitwise OR Operation of 12 and 25

 00001100

| 00011001

 _____

 00011101 = 29 (In decimal)

Bitwise XOR:-

12 = 00001100 (In Binary)

25 = 00011001 (In Binary)

Bitwise XOR Operation of 12 and 25

 00001100

^ 00011001

 _____

 00010101 = 21 (In decimal)

Example Program:-

```
#include<stdio.h>
main()
{
int a = 12, b = 25;
printf("Output = %d", a&b);
   printf("Output = %d", a|b);
   printf("Output = %d", a^b);
}
```

Output:-

Output = 8

Output = 29

Output = 21

Assignment operator(=):-Assignment operator is used to assign value or result of expression to a variable.

Short hand assignment operator:- Shorthand assignment operator combines one of the arithmetic or bitwise operators with assignment operator. It is also called as compound assignment.

Some of the shorthand assignment operators.

1. Addition assignment operator(+=)

Ex: a+= 2 is equivalent to a=a+2.

2. Subtraction assignment operator(-=)

Ex: a-=2 is equivalent to a=a-2.

3. Division assignment operator(/=)

Ex: a/=2 is equivalent to a=a/2.

4. Multiplication assignment operator(*=)

Ex: a*=2 is equivalent to a=a*2.

5. Remainder assignment operator(%=)

Ex:a%=2 is equivalent to a=a%2

6. Left shift assignment operator (<<=)

Ex: a<<=2 is equivalent to a=a<<2.

7. Right shift assignment operator(>>=)

Ex: a>>2 is equivalent to a=a>>>2.

8. Bitwise AND assignment operator(&=)

Ex: a&=2 is equivalent to a=a&2.

9. Bitwise XOR assignment operator(^=)

Ex: a^=2 is equivalent to a=a^2.

10. Bitwise OR assignment(|=)

Ex: a|=2 is equivalent to a=a|2.

Comma Operator(,):- comma operator can be used as separator and also as operator.

1) Comma (,) as separator

While declaration multiple variables and providing multiple arguments in a function, comma works as a separator.

Example:

```
int a,b,c;
```

In this statement, comma is a separator and tells to the compiler that these (a, b, and c) are three different variables.

2) Comma (,) as an operator

Sometimes we assign multiple values to a variable using comma, in that case comma is known as operator.

Example:

```
a = 10,20,30;

b = (10,20,30);
```

In the first statement, value of a will be 10, because assignment operator (=) has more priority more than comma (,), thus 10 will be assigned to the variable a.

In the second statement, value of b will be 30, because 10, 20, 30 are enclosed in braces, and braces has more priority than assignment (=) operator. When multiple values are given with

comma operator within the braces, then right most value is considered as result of the expression. Thus, 30 will be assigned to the variable b.

Conditional Operator(?:):-The conditional operator (? and :) is a special operator which requires three operands. Its syntax is as follows:

Syntax: expression1 ? expression2 : expression3

The first expression1 is evaluated, if it is true then the value of expression2 becomes the result of the overall expression. On the other hand, if expression1 is false, then the value of expression3 becomes the result of the overall expression.

```
#include<stdio.h>
 main( )
{
   int a, b, max;
   printf("Enter a and b: ");
   scanf("%d%d", &a, &b);
   max = a > b ? a : b;
   printf("Largest of the two numbers = %d\n", max);
 }
```

Output:-
Enter a and b: 1993 1534
Largest of the two numbers = 1993

Expressions

An expression is a combination of operators, constants and variables(operands).

Ex:-10 + 4 * 3 / 2

10,4,3,2 are constants and +,*,/ are operators.

Expression evaluation in C:-

In the C programming language, an expression is evaluated based on the operator precedence and associativity. When there are multiple operators in an expression, they are evaluated according to their precedence and associativity.

An expression is evaluated based on the precedence and associativity of the operators in that expression.

What is Operator Precedence?

Operator precedence is used to determine the order of operators evaluated in an expression. In c programming language every operator has precedence (priority). When there is more than one operator in an expression the operator with higher precedence is evaluated first and the operator with the least precedence is evaluated last.

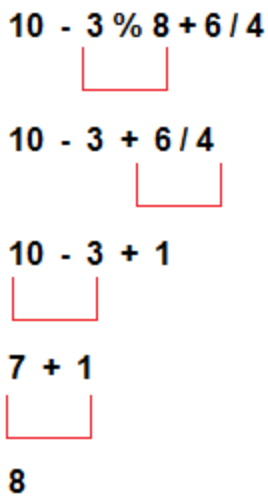
What is Operator Associativity?

Operators Associativity is used when two operators of same precedence appear in an expression. Associativity can be either **L**eft **t**o **R**ight or **R**ight **t**o **L**eft.

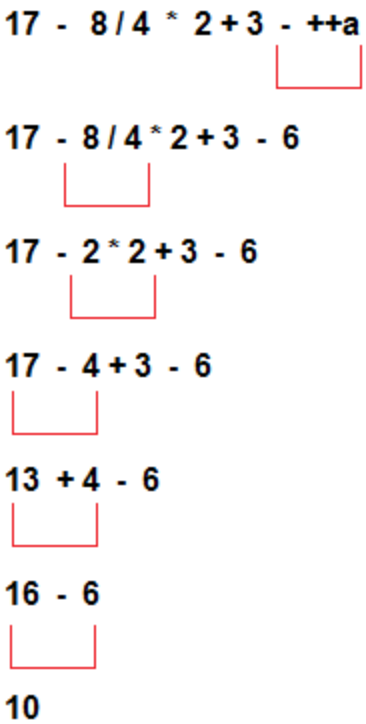
Precedence and Associativity Table

Operator	Description	Associativity
( ) [ ] . -> ++ --	Parentheses (function call) Brackets (array subscript) Member selection via object name Member selection via pointer Postfix increment/decrement	left-to-right
++ -- + - ! ~ * & sizeof	Prefix increment/decrement Unary plus/minus Logical negation/bitwise complement Dereference Address (of operand) Determine size in bytes on this implementation	right-to-left
* / %	Multiplication/division/modulus	left-to-right
+ -	Addition/subtraction	left-to-right
<< >>	Bitwise shift left, Bitwise shift right	left-to-right
< <= > >=	Relational less than/less than or equal to Relational greater than/greater than or equal to	left-to-right
== !=	Relational is equal to/is not equal to	left-to-right
&	Bitwise AND	left-to-right
^	Bitwise exclusive OR	left-to-right
|	Bitwise inclusive OR	left-to-right
&&	Logical AND	left-to-right
||	Logical OR	left-to-right
? :	Ternary conditional	right-to-left
= += -= *= /= %= &= ^= |= <<= >>=	Assignment Addition/subtraction assignment Multiplication/division assignment Modulus/bitwise AND assignment Bitwise exclusive/inclusive OR assignment Bitwise shift left/right assignment	right-to-left
,	Comma (separate expressions)	left-to-right

Example 1:



Example 2:



Example Program:-

```
#include<stdio.h>
main()
{
  float a,b,c,x,y,z;
  a=9;
  b=12;
  c=3;
  x= a-b/3+c*2-1;
  y= a-b/(3+c)*(2-1);
  z=a-(b/(3+c)*2)-1;
  printf("x = %f\n",x);
  printf("y = %f\n",y);
  printf("z = %f\n",z);
}
```

Output:-

```
x = 10.000000
y = 7.000000
z = 4.000000
```

---*---**

Topic 4 : Decision statements - If and switch statements

In C programming, Control structures defines the flow of execution of the program.

Types of control structures

1. Sequence

2. Selection

3. Repetition

1. *Sequence*: Statements are executed in a specified order. No statement is skipped and no statement is executed more than once.

2. *Selection:*

It selects a statement to execute on the basis of condition. Statement is executed when the condition is true and ignored when it is false.

Selection statements :- if, if else, nested if else, else if ladder, switch statements.

3. *Repetition:*

In this structure the statements are executed more than one time. It is also known as iteration or loop

Loop statements:- while loop, for loop, do-while loop.

<u>if statement</u> :- When user need to execute a block of statements only when a given condition is true then we use if statement.

Syntax:-

```
if(condition)
{
statement 1;

statement 2;

...
 }
```

Example program:-

```
#include<stdio.h>
 main( )
{
int number=0;
printf("Enter a number:");
scanf("%d",&number);
if(number%2==0)
{
printf("%d is even number",number);
}
}
```

Output

```
Enter a number:4
4 is even number
```

if-else statement:- The *if* statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it won't. But what if we want to do something else if the condition is false. Here comes the C *else* statement. We can use the *else* statement with *if* statement to execute a block of code when the condition is false.

Syntax:-

```
if(condition)
{
//code to be executed if condition is true
}
else
{
//code to be executed if condition is false
}
```

Example Program 1:-

```
#include<stdio.h>
main( )
{
int number;
printf("enter a number:");
scanf("%d",&number);
if(number%2==0)
{
printf("%d is even number",number);
}
else
{
printf("%d is odd number",number);
}
}
```

Output

```
enter a number:4
4 is even number
enter a number:5
5 is odd number
```

Example Program 2:-

```
#include <stdio.h>
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 1

Enter your age?18
You are eligible to vote...

Output 2

Enter your age?13
Sorry ... you can't vote

else-if ladder statement:- The else..if statement is useful when user need to check multiple conditions within the program.

Note:- nesting of if-else blocks can be avoided using else..if statement.

Syntax:-

```
if(condition1)
{
//code to be executed if condition1 is true
}
else if(condition2)
{
//code to be executed if condition2 is true
}
```

```
else if(condition3)
{
//code to be executed if condition3 is true
}
...
else
{
//code to be executed if all the conditions are false
}
```

Example Program:-

```
#include<stdio.h>
#include<conio.h>
voidmain()
{
   int a;
   printf("Enter a Number: ");
   scanf("%d",&a);
   if(a >0)
   {
      printf("Given Number is Positive");
   }
   elseif(a == 0)
   {
      printf("Given Number is Zero");
   }
   elseif(a <0)
   {
      printf("Given Number is Negative");
   }
   getch();
}
```

Output:

```
Enter a Number: -5
Given Number is Negative
```

Nested if-else statement:- When an if else statement is present inside the body of another "if" or "else" then this is called nested if else.

Syntax:-

```
if(expression)
{
   if(expression1)
   {
      statement-block1;
   }
   else
   {
      statement-block2;
   }
}
else
{
   statement-block3;
}
```

Example Program:-

```
#include<stdio.h>
main( )
{
   int a,b,c;
   printf("Enter 3 number to compare: ");
   scanf("%d%d%d",&a,&b,&c);
   if(a > b)
   {
      if(a > c)
      {
         printf("a is greatest");
      }
      else
      {
         printf("c is greatest");
      }
   }
```

```
else
  {
    if(b > c)
    {
      printf("b is greatest");
    }
    else
    {
      printf("c is greatest");
    }
  }
}
```

Output:

```
Enter 3 Number to compare: 5
7
9
c is greatest
```

Switch statement:- The switch statement allows us to execute one code block among many alternatives.

Syntax:-

```
switch( expression )
{
        case constant-1:
                        Block-1;
                        Break;
        case constant-2:
                        Block-2;
                        Break;
        case constant-n:
                        Block-n;
                        Break;
        default:
                        Block-1;
                        Break;
}
Statement-x;
```

How does the switch statement work?

The `expression` is evaluated once and compared with the values of each `case` label.

- If there is a match, the corresponding statements after the matching label are executed. For example, if the value of the expression is equal to `constant2`, statements after `case constant2:` are executed until `break` is encountered.
- If there is no match, the default statements are executed.

> If we do not use `break`, all statements after the matching label are executed.
> By the way, the `default` clause inside the `switch` statement is optional.

Example program:-

```
// Program to create a simple calculator
#include <stdio.h>
int main()
{
   char operator;
   double n1, n2;
   printf("Enter an operator (+, -, *, /): ");
   scanf("%c", &operator);
   printf("Enter two operands: ");
   scanf("%lf %lf",&n1, &n2);
   switch(operator)
   {
      case '+':
         printf("%.1lf + %.1lf = %.1lf",n1, n2, n1+n2);
         break;

      case '-':
         printf("%.1lf - %.1lf = %.1lf",n1, n2, n1-n2);
         break;

      case '*':
         printf("%.1lf * %.1lf = %.1lf",n1, n2, n1*n2);
         break;

      case '/':
         printf("%.1lf / %.1lf = %.1lf",n1, n2, n1/n2);
         break;
```

```
    // operator doesn't match any case constant +, -, *, /
    default:
        printf("Error! operator is not correct");
  }

  return 0;
}
```

Output

```
Enter an operator (+, -, *,): -
Enter two operands: 32.5
12.4
32.5- 12.4 = 20.1
```

---*---**

Topic 5: Loop control statements - while, for, do-while statements

Loop control structures:-

Loops in programming come into use when we need to repeatedly execute a block of statements. For example: Suppose we want to print “Hello World” 10 times. This can be done in two ways as shown below:

Method 1:- to write printf( ) statement 10 times

```
#include <stdio.h>
 main( )
{
   printf( "Hello World\n");
   printf( "Hello World\n");
   printf( "Hello World\n");
   printf( "Hello World\n");
   printf( "Hello World\n");
   printf( "Hello World\n");
   printf( "Hello World\n");
   printf( "Hello World\n");
   printf( "Hello World\n");
   printf( "Hello World\n");
    return0;
}
```

Output:

```
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
```

Method 2:- using loops concept

In Loop, the statement needs to be written only once and the loop will be executed 10 times as shown below.

```
#include <stdio.h>
 int main( )
{
   inti=0;

   for(i = 1; i <= 10; i++)
   {
      printf( "Hello World\n");
   }

   return0;
}
```

Output:-

```
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
Hello World
```

In computer programming, a loop is a sequence of instructions that is repeated until a certain condition is reached. A loop consists of two parts, a body of a loop and a control statement. The control statement is a combination of some conditions that direct the body of the loop to execute until the specified condition becomes false.

Types of Loops in C

Depending upon the position of a control statement in a program, looping in C is classified into two types:

1. Entry controlled loop – for loop, while loop

2. Exit controlled loop – do-while loop

In an **entry controlled loop,** a condition is checked before executing the body of a loop. It is also called as a pre-checking loop.

In an **exit controlled loop**, a condition is checked after executing the body of a loop. It is also called as a post-checking loop.

for loop:-

syntax:-

```
for (initializationStatement; testExpression; updateStatement)
{
   // statements inside the body of loop
}
```

How for loop works?

- The initialization statement is executed only once.
- Then, the test expression is evaluated. If the test expression is evaluated to false, the `for` loop is terminated.
- However, if the test expression is evaluated to true, statements inside the body of `for` loop are executed, and the update expression is updated.
- Again the test expression is evaluated.

This process goes on until the test expression is false. When the test expression is false, the loop terminates.

```
// Print numbers from 1 to 10
#include<stdio.h>
main()
 {
  int i;
  for (i = 1; i < 11; ++i)
  {
    printf("%d ", i);
  }
}
```

Output

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

Explanation:-

1. i is initialized to 1.
2. The test expression i < 11 is evaluated. Since 1 less than 11 is true, the body of for loop is executed. This will print the **1** (value of i) on the screen.

3. The update statement ++i is executed. Now, the value of i will be 2. Again, the test expression is evaluated to true, and the body of for loop is executed. This will print **2** (value of i) on the screen.
4. Again, the update statement ++i is executed and the test expression i < 11 is evaluated. This process goes on until i becomes 11.
5. When i becomes 11, i < 11 will be false, and the for loop terminates.

While loop:-

Syntax:-

```
initializationStatement;
while (testExpression)
{
  // statements inside the body of the loop
        updateStatement;
}
```

How while loop works?

- The `while` loop evaluates the test expression inside the parenthesis `()`.
- If the test expression is true, statements inside the body of `while` loop are executed. Then, the test expression is evaluated again.
- The process goes on until the test expression is evaluated to false.
- If the test expression is false, the loop terminates (ends).

Example Program:-

```
#include<stdio.h>
main( )
{
  int i = 1;

  while (i <= 5)
  {
    printf("%d\n", i);
    ++i;
  }
}
```

Output

```
1
2
3
4
5
```

Here, we have initialized i to 1.

1. When i is 1, the test expression i <= 5 is true. Hence, the body of the while loop is executed. This prints 1 on the screen and the value of i is increased to 2.
2. Now, i is 2, the test expression i <= 5 is again true. The body of the while loop is executed again. This prints 2 on the screen and the value of i is increased to 3.
3. This process goes on until i becomes 6. When i is 6, the test expression i <= 5 will be false and the loop terminates.

do-while loop:- The `do..while` loop is similar to the `while` loop with one important difference. The body of `do...while` loop is executed at least once. Only then, the test expression is evaluated.

syntax:-

```
initialization Statement;
do
{
  // statements inside the body of the loop
   updateStatement;
}
while (testExpression);
```

How do...while loop works?

- The body of do...while loop is executed once. Only then, the test expression is evaluated.

- If the test expression is true, the body of the loop is executed again and the test expression is evaluated.
- This process goes on until the test expression becomes false.
- If the test expression is false, the loop ends.

```
// Program to add numbers until the user enters zero
#include <stdio.h>
main( )
{
   double number, sum = 0;
   // the body of the loop is executed at least once
   do
   {
      printf("Enter a number: ");
      scanf("%lf", &number);
      sum += number;
   }
   while(number != 0.0);
   printf("Sum = %.2lf",sum);
   return 0;
}
```

Output

```
Enter a number: 1.5
Enter a number: 2.4
Enter a number: -3.4
Enter a number: 4.2
Enter a number: 0
Sum = 4.70
```

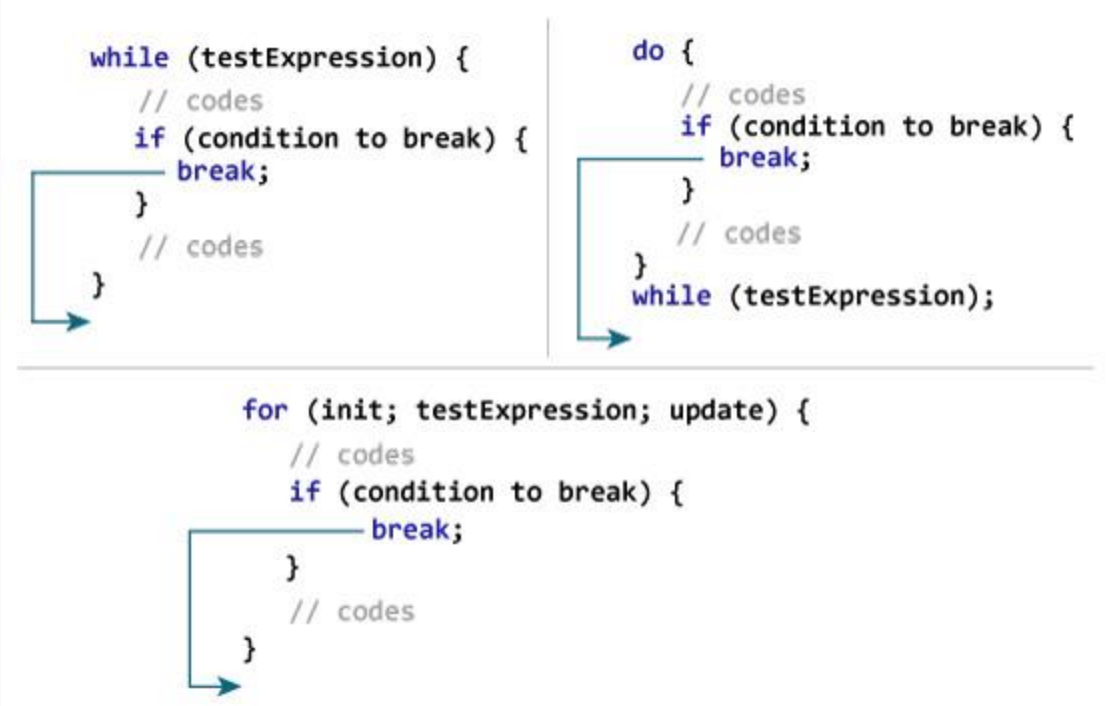
break statement:-

The break statement ends the loop immediately when it is encountered. Its syntax is:

```
break;
```

The break statement is almost always used with `if...else` statement inside the loop.

How break statement works?



Example 1: break statement

```
// Program to calculate the sum of numbers (10 numbers max)
// If the user enters a negative number, the loop terminates

#include <stdio.h>
main( )
{

  int i;
  double number, sum = 0.0;
  for (i = 1; i <= 10; ++i)
  {
    printf("Enter a n%d: ", i);
    scanf("%lf", &number);
    // if the user enters a negative number, break the loop
    if (number < 0.0)
    {
      break;
    }
    sum += number; // sum = sum + number;
```

```
  }
  printf("Sum = %.2lf", sum);
}
```

Output

```
Enter a n1: 2.4
Enter a n2: 4.5
Enter a n3: 3.4
Enter a n4: -3
Sum = 10.30
```

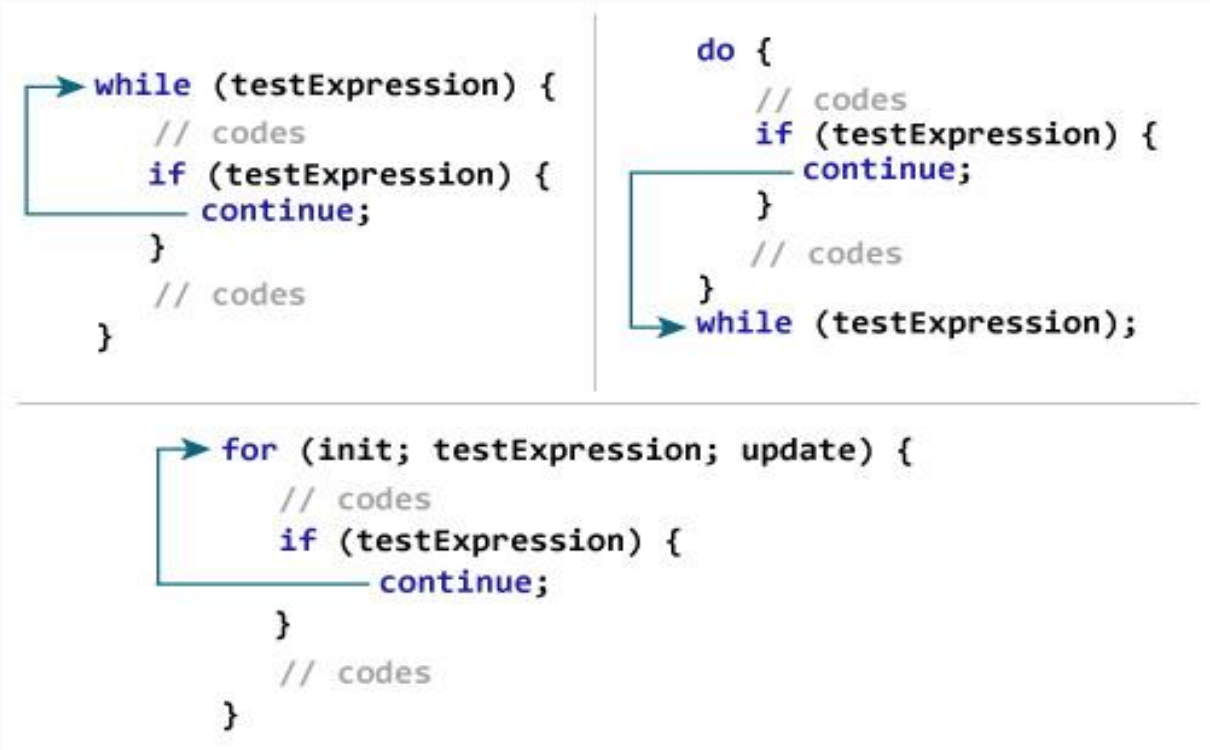
continue statement:-

The `continue` statement skips the current iteration of the loop and continues with the next iteration. Its syntax is:

```
continue;
```

The `continue` statement is almost always used with the `if...else` statement.

How continue statement works?



Example 2: continue statement

```
// Program to calculate the sum of numbers (10 numbers max)
// If the user enters a negative number, it's not added to the result

#include <stdio.h>
main( )
{
  int i;
  double number, sum = 0.0;
  for (i = 1; i <= 10; ++i)
  {
    printf("Enter a n%d: ", i);
    scanf("%lf", &number);
    if (number < 0.0)
    {
      continue;
    }
    sum += number; // sum = sum + number;
  }
  printf("Sum = %.2lf", sum);
}
```

Output

```
Enter a n1: 1.1
Enter a n2: 2.2
Enter a n3: 5.5
Enter a n4: 4.4
Enter a n5: -3.4
Enter a n6: -45.5
Enter a n7: 34.5
Enter a n8: -4.2
Enter a n9: -1000
Enter a n10: 12
Sum = 59.70
```

goto statement:-

The goto statement allows us to transfer control of the program to the specified label.

Syntax of goto Statement:

```
goto label;
... .. ...
... .. ...
label:
statement;
```

The `label` is an identifier. When the `goto` statement is encountered, the control of the program jumps to `label:` and starts executing the code.

Example goto Statement

```
// Program to calculate the sum and average of positive numbers
// If the user enters a negative number, the sum and average are displayed.

#include <stdio.h>
main( )
{
  const int maxInput = 100;
  int i;
  double number, average, sum = 0.0;
  for (i = 1; i <= maxInput; ++i)
  {
    printf("%d. Enter a number: ", i);
    scanf("%lf", &number);
    // go to jump if the user enters a negative number
    if (number < 0.0)
    {
      goto jump;
    }
    sum += number;
  }
```

```
jump:
  average = sum / (i - 1);
  printf("Sum = %.2f\n", sum);
  printf("Average = %.2f", average);
}
```

Output

```
1. Enter a number: 3
2. Enter a number: 4.3
3. Enter a number: 9.3
4. Enter a number: -2.9
Sum = 16.60
Average = 5.53
```

---*---**

Topic 6: Arrays

Consider following example:-

```
main( )
{
int a=4;
int a=9;
int a=12;
Printf(“a=%d”,a);
}
```

Output:-

a=12

Explanation:- ‘a’ is a variable which can store only one data at a time. First 4 is stored in ‘a’ then 9 and then 12 is stored in ‘a’. Therefore finally 12 is stored in memory. Values 4,9 are erased.

Why do we need arrays?

We can use normal variables (v1, v2, v3, ..) when we have a small number of objects, but if we want to store a large number of instances, it becomes difficult to manage them with normal variables. The idea of an array is to represent many instances in one variable.

An array is collection of elements of same data type which are referred with single name.

Types of arrays:-

1. Single dimensional array (or) One dimensional(1D) array
2. Multi dimensional array

1D array:- single dimensional arrays are used to store list of values of same datatype. In other words, single dimensional arrays are used to store a row of values.

Declaration of array:-

Syntax:-

datatype arrayname[size];

Example:- int a[5]; /*integer array*/

float b[3]; /* float array*/

char c[10]; /*character array*/

Initialization of array:- Assigning values to array is called initialization. It can be done in 2 ways.

1. At time of compilation
2. At time of execution

At time of compilation:- assignment (=) operator is used for initialization of array.

Example:-

int n[6] = {2, 3, 15, 8, 48, 13};

Example program

```
include<stdio.h>
main( )
{
   int i;
```

```
   int arr[3] = {2, 3, 4};      // Compile time array initialization
   for(i = 0 ; i < 3 ; i++)
   {
      printf("%d\t",arr[i]);
   }
}
```

Output:-

2 3 4

At time of execution:- scanf( ) is used for initialization of arrays at run time.

```
#include <stdio.h>
main( )
{
 int values[5];
 printf("Enter 5 integers: ");
 // taking input and storing it in an array
 for(int i = 0; i < 5; ++i)
 {
   scanf("%d", &values[i]);
 }
 printf("Displaying integers: ");
 // printing elements of an array
 for(int i = 0; i < 5; ++i)
 {
   printf("%d\n", values[i]);
 }
}
```

Output

```
Enter 5 integers: 1
-3
34
0
3
Displaying integers: 1
-3
34
0
```

```
3
```

Multi Dimensional Array

An array of arrays is called as multi dimensional array. In simple words, an array created with more than one dimension (size) is called as multi dimensional array. Multi dimensional array can be of two dimensional array **or** three dimensional array **or** four dimensional-array **or more.**

Most popular and commonly used multi dimensional array is two dimensional array. The 2-D arrays are used to store data in the form of table. We also use 2-D arrays to create mathematical matrices**.**

Declaration of Two Dimensional Array

We use the following general syntax for declaring a two dimensional array...

```
datatype arrayName [ rowSize ] [ columnSize ] ;
```

Example:-

```
float x[3][4];
```

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

Initialization of Two Dimensional Array

It can be done in 2 ways.

1. At the time of compilation
2. At the time of execution

At time of Compilation:- assignment operator is used for initialization.

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

Example program:-

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

```
main( )
{
int i=0,j=0;
int arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};



        for(i=0;i<4;i++)
        {
                 for(j=0;j<3;j++)
                {
                printf("%d\t",arr[i][j]);
                }         //end of j
      Printf("\n");
        }         //end of i
}
```

Output:-

1 2 3

2 3 4

3 4 5

4 5 6

At the time of execution:- scanf( ) is used for initialization.

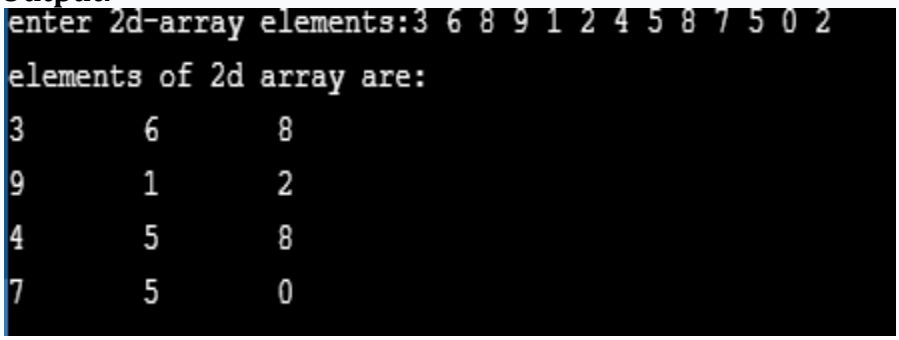
Example program:-

```
#include<stdio.h>
main( )
{
int i,j;
int arr[4][3];
printf("enter 2d-array elements:");
for(i=0;i<4;i++)
{
```

```
 for(j=0;j<3;j++)
 {
   scanf("%d",&arr[i][j]);
 }//end of j
}//end of i


printf("elements of 2d array are:\n");
for(i=0;i<4;i++)
{
 for(j=0;j<3;j++)
 {
   printf("%d\t",arr[i][j]);
 }//end of j
 printf("\n");
 }//end of i
}
```

Output:-



Nested for loops:- Nested loop means a loop statement inside another loop statement. That is why nested loops are also called as "loop inside loop".

Syntax for Nested For loop:-

```
for ( initialization; condition; increment )
 {
```

```
  for ( initialization; condition; increment )
        {
      // statement of inside loop
        }
  // statement of outer loop
}
```

Syntax for Nested While loop:-

```
while(condition)
{
  while(condition)
  {
      // statement of inside loop
  }
  // statement of outer loop
}
```

Syntax for Nested Do-While loop:-

```
do
{
  do
{
    // statement of inside loop
 }while(condition);
  // statement of outer loop
}while(condition);
```

```
/*Program for reading and displaying matrix elements in an array*/

main( )
{
    int i, j;
    // Declare the matrix
   int matrix[ROW][COL] = { { 1, 2, 3 },
                            { 4, 5, 6 },
                            { 7, 8, 9 } };
   printf("Given matrix is \n");
    // Print the matrix using nested loops
   for (i = 0; i < ROW; i++)
   {
        for (j = 0; j < COL; j++)
                    {
         printf("%d ", matrix[i][j]);
                    }
        printf("\n");
   }

}
```

Output:-

Given matrix is

1 2 3

4 5 6

7 8 9

-----*THE END***----**