Steps in development of a program

A computer program is a step-wise set of instructions written in any computer language to solve a particular problem. In order to write a computer program some steps are used to write a program. It is also called Program Development Life Cycle (PDLC). These steps are as follows:

1. Problem Identification/Definition.
2. Algorithm
3. Flowchart
4. Coding
5. Compilation
6. Testing and Debugging
7. Documentation
8. Maintenance

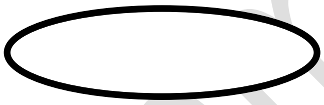
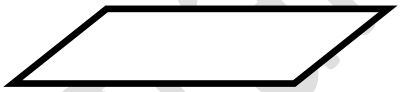
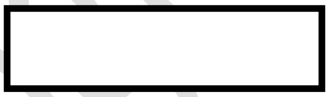
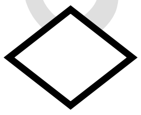
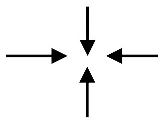
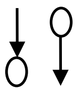
1. **Problem Identification/ Definition:** It is the first and foremost step for a development of a program. In this step program must have to understand the problem. If a programmer is not able to understand the problem he will not be able to solve the problem. So, he must be able to clearly understand the problem, what type of inputs will be supplied to the program and how will be the output of the problem.
2. **Algorithm:** It is the step by step procedure, of finite steps, to solve a particular problem written in any language. In this face instructions/steps are designed in such a way so that desired result is obtained. Although algorithm can be written in any language but English language should be preferred and an algorithm written in English language is called Pseudo code.

   An Algorithm must have following characteristics:
   a. Each step of algorithm must be definite and feasible.
   b. Each step must be clear and unambiguous.
   c. Each step must be performed in finite time and their inputs (if any) must be clearly defined.
   d. One or more steps must not be repeated infinitely i.e. an algorithm must terminate after a finite number of steps.
   e. On termination of algorithm the desired output must be obtained i.e. the algorithm must be effective.

   Few Examples of Algorithm
   1. Write a algorithm to find the sum of two given numbers
      ```
      Step1. START
      Step2. INPUT A, B
      Step3. SUM = A + B
      Step4. PRINT “Sum of two numbers”, SUM
      Step5. END
      ```
   2. Write a algorithm to Convert temperature from Fahrenheit to Celsius.
      ```
      Step1. START
      Step2. READ F
      Step3. C = 5/9 * (F – 32)
      Step4. PRINT “Temperature in Celsius is”, SUM
      Step5. END
      ```
   3. Write a algorithm to find the largest of three numbers.

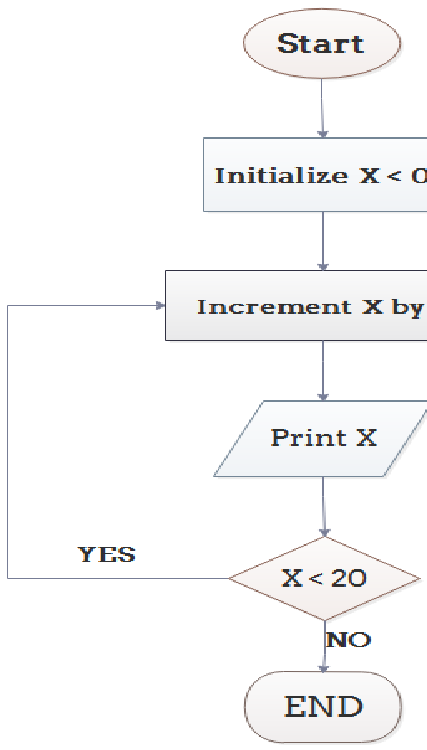
```
Step1. START
Step2. INPUT A, B, C
Step3. IF (A>B) THEN
       Begin
              If (A>C) THEN
                     Write "The largest number is ", A
              ELSE
       Write "The largest number is ", C
       END
       Else
       Begin
              IF (B>C) THEN
                     Write "The largest number is ", B
              ELSE
                     Write "The largest number is ", C
       END
Step4. END
```

3. <u>Flowchart:</u> Flowchart is a pictorial representation of algorithm. In other words, it is a graphical tool. In this various symbols like Rectangle, Parallelogram etc. are used. The flowchart of a program can be linked to the blue print of a building. As a designer draws a blue print before starting of a building in the same way a programmer draws a flowchart of a program/problem before writing a computer program. Some rules are follow with the help of various symbols which are as follows:

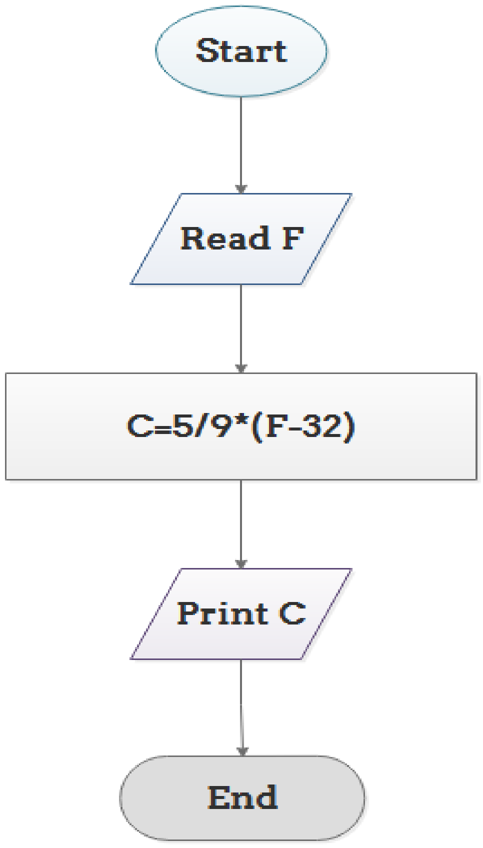
Symbols	Meaning
1. 	START / END :- It indicates the start /end of the flowchart.
2. 	INPUT/OUTPUT:- It indicates the input/output of the flowchart .
3. 	PROCESSING: It indicates are process/ computation
4. 	DECISION/ CONDITION BOX :- It is a decision point with one input and two output.
5. 	**DATA/ CONTROL FLOW :-** These lines indicates the direction of flow of data/control .
6. 	**CONNECTORS :-** It indicates that the flow continues and a matching symbol (same letter) is placed.

A few Example of Flowcharts:

Example 1: Flowchart to Print 1 to 20:



Example 2: Flowchart to Convert Temperature from Fahrenheit (°F) to Celsius (°C)



ADVANTAGES OF USING FLOWCHARTS:

The advantages of flowcharts are as follows:

1. Communication: Flowcharts are better way of communicating the logic of a program as it holds old saying that “a picture is worth a thousand words”. As, a flowchart is a pictorial representation of a program, it is easier for a programmer to explain the logic of a program to some other programmer.

2. Effective Analysis: With the help of a flowchart, problem can be analysed in more effective way.

3. Efficient Coding: The flowchart acts as a guide or blueprint during the system analysis and program development phase.

4. Proper Debugging: With the help of flowchart debugging is easy.

5. Efficient program maintenance: The maintenance of operating program becomes easy with the help of flowchart. It helps the programmer to put efforts more efficiently on that part.

LIMITATION OF USING FLOWCHARTS:

The limitation of flowcharts is as follows:

1. Complex Logic: Sometimes, the program logic become quite complicated/lengthy that makes flowchart complex and clumsy.

2. Alterations and modifications: For any alteration/ modification in any flowchart the whole flowchart is required to be redrawn.

DIFFRENCE BETWEEN FLOWCHART AND ALGORITHM

S. NO.	FLOWCHART	ALGORITHM
1	A flowchart is a graphical representation algorithm.	It is a step by step procedure (Set of instructions) to solve a particular problem.
2	It is very easy to create and understand.	It is sometimes very difficult to create and understand.
3	It uses various kind of Symbols /Diagrams.	It does not use any kind of Symbols/Diagrams.

4.Coding: After the algorithm and flowchart is developed and tested for any problem it (algorithm/flowchart) must be written in any computer language. This process is called program coding. Thus process of converting an algorithm into a computer program through any computer language is known as program coding.

5. Compilation: Compilation is a process of converting source program into object program.

Source program is a program written in any computer language (high level language) which is understandable to human being.

Object program is a program written in any computer language which is understandable to computer.

The programmer writes the program Acc. to the syntax of language .These lines are called source code and compiler converts these lines from high level language to the language which is understandable to the computer. The program after compilation is called object program.

6. Testing and Debugging: All the computer programs are designed /programmed by human beings, which subjects to error and these errors are called Bugs. The process of finding errors is done by the process of testing and removal of errors is called debugging. They are basically three types of errors.

1. Syntax Errors.
2. Logical Errors.
3. Execution Errors.

1. Syntax Errors: Syntax relates to grammar of programming language. If any grammatical rules are not followed then C compiler will display syntax errors. Syntax error can be detected by the process of compilation.

Example of syntax errors:-

Semicolon (;) not placed after statement
i.e. c=a+b is incorrect.
 c=a+b; is correct.
Commas (,) not placed before statement
i.e. Int a b is incorrect.
 Int a,b is correct.

2. Logical Errors:- Logical errors occur due to wrong way writing of program. Sometimes problem is not clearly understand by programmer, so he may write wrong code. Logical Errors are tough to locate. The logical errors can be detected by the compiler as these errors can be detected by the way of testing the program i.e. entering the right input and getting the wrong output.

Example of Logical Errors:-

If we want to add two numbers then statement will be c=a+b, but if we write c=a-b then it will be incorrect logic.

3. Runtime Errors:- The errors which occurs during running of program, are called Runtime Errors. These are also called as Execution errors.

Example of Runtime errors:-

1. Divide by zero.
2. Overflow of memory.
3. Underflow of memory.

7. Documentation:- In this phase the important information regarding program is entered. It is of two types

1. Programmer level

 In programmer level documentation all the information related to programming which is useful for programmer is entered.

2. User level

 In User level documentation all the information related to the program which is useful for the user is entered.

8. Maintenance:- This is the last step of system development life cycle which is never ending process. Maintenance is always an ongoing process due to some reasons like errors detected by user during its use, some addition/modification required in program , backup of the data , change in technology.

Qualities of a Good program

It will be a good program only if it has following characteristics:

1. Efficient: A good program must be efficient means it must be able to perform the said tasks in an efficient manner.

2. Reliable: A computer program must have proper reliability means it must provide correct output in each and every case.

3. Flexible: A program is called flexible if it can be used for more than one purpose.

4. Portable: A good program is always portable. Portability means that it can be executed on different types of machines and different platforms or operating systems.

5. Integrity: Integrity is that property of a program that supports the changes at any place. It ensures correct operations, correct results and correct changes.

6. Clarity: Clarity is that property of a program which gives the opportunity to other programmers to understand the logic of the program.

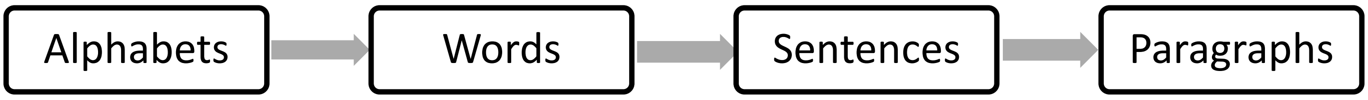
What is C?

C is a programming language developed at AT & T's Bell Laboratories of USA in 1972. It was designed and written by a man named Dennis Ritchie.

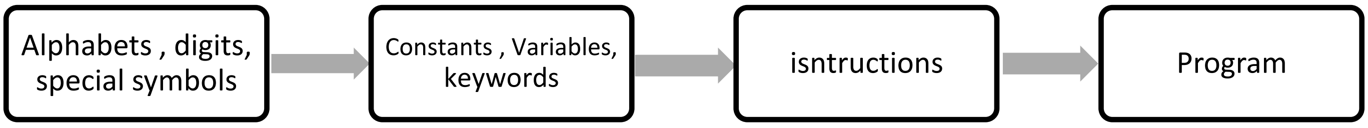
Getting Started with C

Communicate with a computer involves speaking the language the computer understands, which immediately rules out English of communication with computer. However, there is a close relationship between English language and learning in C language. The classical method of learning English is to first learn the alphabets used in the language, then learn to combine these alphabets to form words, which in turn are combined to form sentences are combined to form paragraphs. Learning in C is similar and easier. Instead of straight-away learning how to write programs, we must first know what alphabets, numbers and special symbols are used in C, then how using them constants, variables and keywords are constructed, and finally how are these combined to form an instruction. A group of instruction would be combined alter on to form a program. This is illustrated in the Figure 1.1

Steps in Learning in English language



Steps in Learning in C


Alphabets , digits, special symbols
Constants , Variables, keywords
isntructions
Program

Introduction To C

The C programming language is a popular and widely used programming language for creating computer programs in. C is a general purpose structured programming language that is powerful, efficient and compact. It was developed at AT & T's Bell Laboratories of USA in 1972. It was designed and written by Dennis Ritchie. C was an offspring of the Basic Cambridge Programming Language (BCPL) called B .

The Main Characteristics of C language are:

1. It is reliable, simple and easy to use.
2. Program written in C are efficient and fast. This is due to its variety of data types and powerful operators.
3. There are only 32 keywords and its strength lies in its built-in functions.
4. C helps in developing structured programs.
5. C is highly suitable for recursive programming.
6. It is also suitable for graphics programming.
7. C is highly portable.
8. It is highly suitable for writing both system software and business packages.

9. We can add our own functions to the C library. Therefore, important features of C is its ability to extend itself.
10. C is format free language. No line numbers needed. Need not place statements on a specified location on a line.
11. Standard libraries for I/O, string manipulation, Arithmetic operations etc.
12. Pointer implementation is available.
13. Bitwise manipulation is also possible.
14. C stands in between the assembly languages and High Level Languages. C has all the advantages of assembly language and it has all the significant features of modern HLLs (High Level Languages). That's why it is often called a Middle Level Language.

Character Set

A character denotes any alphabet, digit or special symbol used to represent information Figuare 1.2 shows the valid alphabets, numbers and special symbols allowed in C.

Alphabets	A, B………..Y,Z. a , b……………….y , z
Digits	0, 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9
Special Symbols	+ - * / % # , . ; ' " ( ) { } [ ] < = > & ! # ^ ? ~ -- \ |
White Spaces	Blank space, tab, Carriage return, formfeed and newline

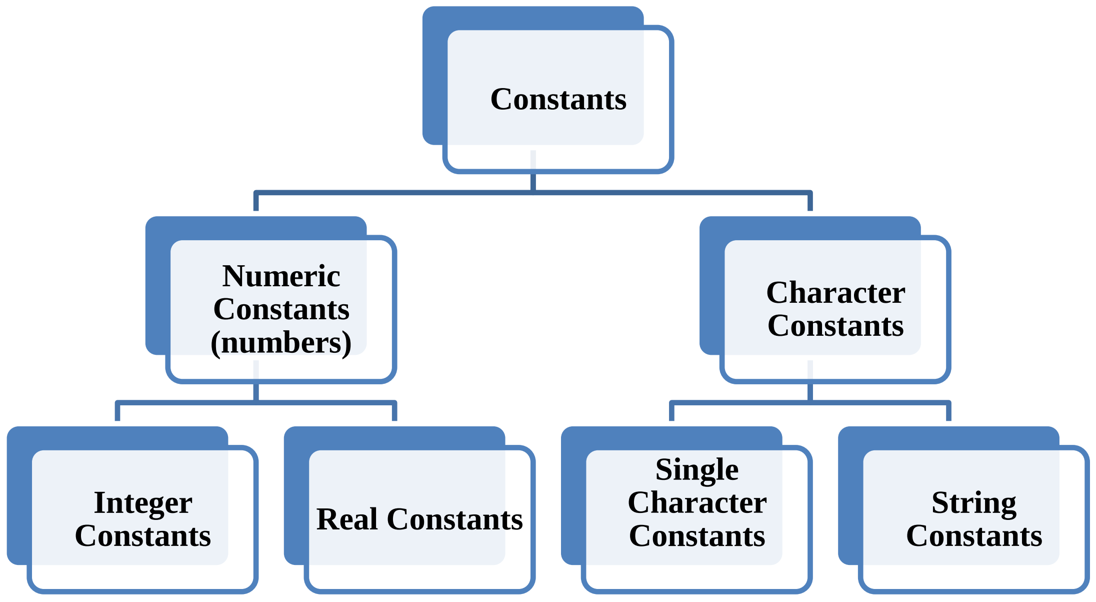
Keywords

C contains 32 keywords that have a standard, predefined meanings. These keywords can be used only for their indended purpose, they cannot be redefined by the programmer. All keywords must be written in lowercase. The keywords are also known as Reserved words. The list of all keywords are given in table.

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

Constants

A Constant is a quality that remain unchanged during the execution of a program. C supports several types of constant as shown in fig 1.1



Integer Constant:

An integer constant refers to a sequence of digits without decimal point.

Rules for constructing integer constants are :

1. An integer constant must have at least one digit.
2. An integer constant contains neither a decimal point nor an exponent.
3. Sign (+ or -) must be precede the numbers.
4. Commas and blank spaces cannot be included within the numbers.
5. Default sign is positive.
6. The allowable range for integer constants for a 16 bit computer is -32768 to +32767

For example:

The following are valid decimal integer constants

15 -12 +56 0 5672

On the other hand, the following are not valid integer constants.

15.0	contains a decimal point
14,500	contains a comma.
6 -	sign does not precede
50 41	Blank spaces are not allowed

Real (floating point) Constants :

Real constants are also called Floating Point Constant. A real constant is a number that contains either a decimal point or an exponent (or both). The real constants could be written in two forms – fractional form and Exponent form.

(a) Fractional form real constants:

Rules for real constants in fractional forms are:

1. A real constant must have at least one digit.
2. A real constant must contain a decimal point.
3. Commas and blank spaces cannot be included within the number.
4. Sign (+ or -) must be precede the number.
5. It could be either (+ve) or (-ve).
6. Default Sign is positive.

Example : The following are valid real constants.

39.9	0.027	0.0	245.9863

(b) Exponent form real constants:

The general form of real constants expressed in exponent form is.

```
Mantissa    e   exponent
```

Rules:

1. The mantissa is either a real number expression in decimal notation or an integer.
2. The exponent is always an integer number with an optional plus or minus sign.
3. The mantissa part may have a (+) or (-) sign. Default sign is (+) (positive).
4. The mantissa part and the exponential part would be separated by a letter ‘e’.
5. No commas or blank spaces are allowed.
6. Range of real constants expressed in exponential form is -3.4 e -38 to 3.4 e 38.

Example: The following are the valid real constants expressed in exponential form.

0.5e3	-2.3e-6	1.7e+8	2.4e-3

Single Character Constant:

A single character constant is a single character enclosed in apostrophes (single quotation makrs). The character may be letter, numbers, or special character.

Example:

‘a’	‘b’	‘c’	‘d’	‘:’	‘2’	‘&’

String Constant:

A string constant is a sequence of character enclosed in double quotes. Total numbers of characters enclosed within the quotes will be called the length of sting constant.

Example:

“A.B.C”	“2019”	“4+5+6”	“D”

Variable:

A variable is a data name that may be used to store a data value. A variable may take different values at different times during the execution of a program. Variable must be defined before they are used in a program.

The rules for variable name are:

1. A variable may consists of letters, digits, and underscore (__).
2. The first character must be letter.
3. Special symbols and blanks are not allowed.
4. Keywords or Reserved words are not allowed.
5. Both uppercase and lowercase letters are allowed and are considered to be distinguishable. E.g. the variable SUM is not same as sum or Sum.
6. A variable name may be as long as you wish, but the first 8 characters of a variable are recognized by C compilers. The variable name accountsbranch and accountsoffice would be regarded as identical variable in C.

Example:

The following are valid variable names

Area	sum	average	A2
Distance	table1	total	john

DATA TYPES

Various quantities such as constants, variables etc. occurring in a C program must have a type associated with them. There can be one and only one type associated with an entity. C supports a very large number of data types. The type of an entity establishes the following information about it.

(a) its meaning.
(b) Constraints applicable to it.
(c) Possible value of the entity.
(d) Possible operations applicable on the entity.
(e) Function that can be used with it.

A data type is defined as a finite set of values along with set of rules for permissible operations. Data in C belongs to one of following types.

1. Basic (or Simple) Data Types.

(a) Integer (int.)
(b) Floating point (float)
(c) Character (char)

2. Structured Data Types.

(a) Array and Strings.

(b) Structure.

(c) Unions

3. Enumerated Data Types.

4. Pointer Data Types.

5. The Void Data Types

Structured data types, Enumerated Data types and pointer data types are discussed as end when they are encountered. The void data set is discussed in the chapter on functions.

1. Basic (or Simple Data Types:

Each types of data may be represented differently in the computer's memory. Consequently, memory requirement of these data types will also be different. Various data types with the size according to 16-bits machine are given in table.

Integer type (int)	Character Type (char)
Int (16 bits)	Char (8 bits)
Short int (16 bits)	Unsigned Char (8 bits)
Long int (16 bits)	
Unsigned int (16 bits)	**Floating point Type (float)**
Unsigned Short int (16 bits)	Float (32 bits)
Unsigned long int (32 bits)	Double (64 bits)
	Long Double (80 bits)

As shown in table , C supports a number of qualifiers that can be applied to the basic data types. They are

- Short
- Long
- Signed
- Unsigned

Also double stores double precision floating point number. An unsigned number is always +ve or zero. With signed modifier, first bit is reserved for the sign (+ve) or (-ve) of the variable.

Range of a signed data types can be calculated according to the following formula.

For example, size of the int data type is 16. So range is to i.e. -32768 to +32767. Range of unsigned data type can be calculated according to the following formula

Declaration of variables:

A variable can be used to store a value of any data type. The declaration of variable must be done before they are used in the program. The syntax for it is as follows

```
Data type    v1,  v2,  ……………,  vn ;
```

Where v1, v2, ………………, vn are the names of variables, separated by commas.
Example:

```
Int      a;
Float    sum;
Char     name;
```

Here int is integer variable; sum is real and name is character variable.

Structure of C program:

The structure of C programs is shown in figure.

```
1.  Link section
2.  Definition section
3.  Global declaration section
4.  Main ()
    {
        Declaration part
        Executable part
    }
5.  Subprogram section
    Function 1 ⎫
    Function 2 ⎪
    ……………… ⎬  (user Defined functions)
    ……………… ⎪
    Function n ⎭
```

1. The link section provides instructions to the compiler to link functions from the system library.
2. The destination section defines various symbolic constants.
3. Global declaration section consists of all global variables. Global variables are the variables that are used in more than one function.
4. Every C program must have one and only one main () Function section. The declaration part of this section declares all the variables used in the executable part. There is atleast one statement in the executable part.
5. The subprogram section contains all the user-defined functions that are called in the main function.

All sections, except the main function section are optional. Comments can be included between the section, before the section or after the section for increase the readability.

OPERATOR AND EXPRESSION

Introduction:

An operator is a symbol or letter used to indicate a specific operation on variables in a program. For example, the symbol '+' is an add operator that adds two data item called **operands.** An expression is a combination of operands (i.e. constants, variables, numbers) connected by operators and parenthesis. For instance, in the expression given below, A and B are operands and '+' is an operator

A + B

C supports many types of operators such as arithmetic, relational, logical etc. An expression involves arithmetic operators is known as arithmetic expressions. The computed result of an arithmetic expression is always a numerical value. The expression which involves relational and/or logical operators is called as Boolean expression. The computed result of a Boolean expression is always a logical value i.e. either true or false.

Arithmetic Operators:

Arithmetic Operators are summarized in table :

Arithmetic Operators	Purpose	Example
+	Addition	4+2 = 6
-	Subtraction or unary minus	4-2 = 2
*	Multiplication	4 *2= 8
/	Division	4/2 = 2
%	Remainder after Division	4/2 = 0

The unary minus (-) operator multiplies its single operand by -1. Therefore, a number preceded by a minus sign changes its sign. When both the operands are integers (or real) then the result produces an integer value (or real value). If one operand is of the real type, and another is integer type, then the result is always a real number. The modulo division operator % cannot be used on floating data.

Integer Expressions:

The integer expressions are formed using integer variables and/or integer constants connected by integer operators.

Example:

1 * 2 + 3 / 2 – 9 % 2

For evaluation of integer expressions the following rules are applicable.

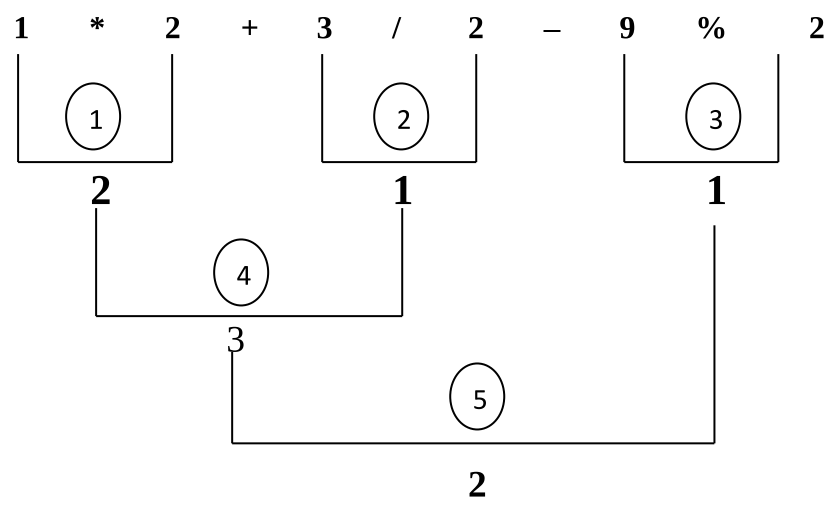
1. / , * and % operator have highest level of precedence and so these operators will be operated first.
2. + and – operators have lower precedence and these operators will be operated later.
3. When operators have same precedence, the evaluation is carried out from left to right.
4. Parentheses will be used to overcome the precedence. The expression within parentheses will have the highest precedence and will be evaluation first.

Example:

Find the value of the following integer expression .

1 * 2 + 3 / 2 – 9 % 2

Solve:



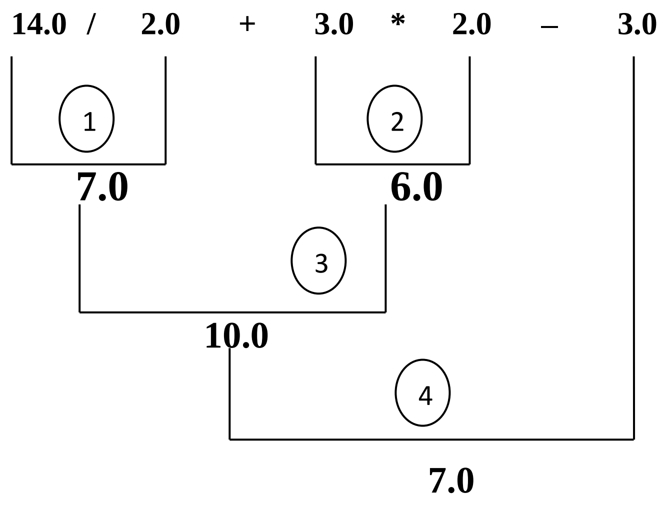
Float Expression:

Float Expression is formed using Real / Integer variable and /or Real/ Integer constant connected by Real operators.

Example: 4.5 * 2.0 + 5.0 / 2.0 + 15.75

The Rule for evaluating Real Expressions is same as that of integer Expressions.

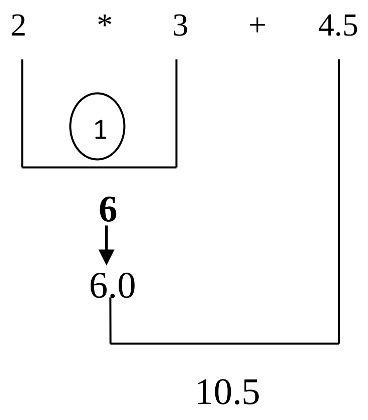
Example:



Mixed mode Expression:

C permits mixing of integer and real quantities in an expression. Such as expression may be called as mixed mode expression. In this mode, integer value is automatically converted into the real value and always yield the real value.

Example: Value of the expression A * B + C



Relational Operator:

Relational operators and their meanings are show in the table.

>	means	**Greater than**
<	means	**Less than**
==	means	**Equal to**
>=	means	**Greater than or Equal to**
<=	means	**Less than or Equal to**
!=	means	**Not equals to**

Relational Expression:
An Expression such as a<b containing a relational operator is termed as a relational expression. The value of a relational expression is either true (one) or False (zero).
Example:

15 > 5 is true

45>20 is false

Precedence of relational operators is

(1) < < = > > = higher precedence.

(2) = = ! = lower precedence.

Logical Operators:

There are three logical operators. Logical operators are shown in table:

&&	Logical END
||	Logical OR
!	Logical NOT

Truth Table of Logical Operators is shown below.

a	b	a&&b	a||b	!(a)	! (b)
0	0	0	0	1	1
0	1	0	1	1	0
1	0	0	1	0	1
1	1	1	1	0	0

Logical NOT operator is a unary operator

Shorthand Assignment Operators:

Form of Assignment Operators is

```
Variable operator = Expression
```

The Assignment statement

```
Variable operator = Expression
```

is equivalent to

```
Variable = Variable  operator (expression)
```

Example:

A += 8 is equivalent to a = a + 8.

Some of the commonly used shorthand assignment operators are given in the following table.

Statement with assignment operator	Statement with shorthand operator
x = x + 8	x + = 8
x = x - 8	x - = 8
x = x * 8	x * = 8
x = x / 8	x / = 8
x = x % 8	x % = 8

Advantages of the shorthand assignment operators:

Advantages of the shorthand assignment operators are;
1. Statement is more efficient.
2. What appear on the left hand side need not be repeated and therefore it becomes easier to write.
3. Statement is more easier to read and concise.

Increment and Decrement Operators:

C has two most useful operators namely increment operator ++ and decrement operator --. These operator transform a variable into a statement expression that abbreviates a special form of assignment. When used as a stand- alone expression, ++a and a++ are both equivalent to the assignment.

m = m+1

They simple increase the value of m by 1. Similarly, the expression statements – m and m – are both equivalent to the assignment.

m = m-1

They simple decrease the value of m by 1

However, when used as expression, the pre-increment operation ++m is different from the post-increment operation m++. The pre-increment increases the value of variable first before using it in the expression, whereas the post-increment increases the value of the variable only after using the prior value of the variable within the expression.

Example:

(a) m = 8;
a = ++m
In this case, the value of a and m would be 8, if we rewrite the above statement as

(b) m = 7;
a = m++;
Then the value of **a** would be 7 and m would be 8.

Conditional operator:

A ternary operator pair " ? : " is available in C to construct conditional expression of the form

Expression 1 ? Expression 2 : Expression 3;

expression 1 is evaluated first. If it is true (nonzero), then the expression2 is evaluated and become the value of the expression otherwise expression3 is evaluated and its value becomes the value of the expression.

Example:

X = 6;
Y = 7;
Z = (X > Y) ? X : Y;

Since value of y is greater than the value of x, so condition is false. Hence Z will be assigned the value of Y (i.e. 7)

Bitwise Operator:

Bitwise operators are used to manipulate the date at bit level. List of bitwise operator is given in the table.

Operators	Meaning
&	Bitwise END
|	Bitwise OR
^	Bitwise Exclusive OR
~	One’s Complement
<<	Shift Left
>>	Shift Right

&, | , and ^ are binary operator. The output produce by these operator is given in the table.

A	B	A & B	A | B	A ^ B
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Precedence in C Operators:

The complete precedence relationship of all the operation discussed in this unit are given in table.

Precedence	Operators	Associatively
1	( )	Left to right
2	++ -- ! ~ -(unary minus)	Right to left
3	* / %	Left to right
4	+ -	Left to right
5	<< >>	Left to right
6	< <= > >=	Left to right
7	= = !=	Left to right
8	&	Left to right
9	^	Left to right
10	|	Left to right
11	&&	Left to right
12	||	Left to right
13	? :	Right to left
14	All assignment operators	Right to left

An operator which appear at a higher level in the list will be given precedence over one which appear lower down in the list.

Unformatted and formatted IOS

Printf ( ) Function:

The general form of printf function is

Printf (“ <control String>”, <list of variables>);

Control string consists of three types of lines

1. Characters that will be printed on the screen as they appear.
2. Escape sequence characters such as \n.
3. Formal specification that define the output format for display of each variable. Format string could be

% d	For	Integer value
% u	For	Unsigned integer value
%f	For	Float in real values
% e	For	Float with an exponent values

%c	For	Character values
%o	For	Octal values
%x	For	Hexadecimal values
% s	For	String values

Scanf ( ) Function:

Scanf ( ) function is used to read the values of variables through keyboard. The format of scanf function is

Scanf (“control string”, & variable1, & variable2,.................);

Control string consists of two types of items

1. Escape sequence characters such as \n.
2. Field (or format specification), consisting of the conversion character %, a data type character (or type specifier), and an optional number, specifying the field width.

The ampersand symbol (&) before each variable name is an operator that specifies the variable name’s address. We must always use this operator, otherwise unexpected results may occur.

Casting of Data:

Sometimes you have to convert one type of data into another type. For example suppose you want to ratio of integer number a and b. The following expression will not give you the desired result

M = a/b

Where m is a real variable. Since a and b are both integer variables, the decimal part of the result of the division would be lost and ratio would represent a wrong figure. For assignment of the desired result to operand m, you will like to convert one of the variables a and b into real numbers. This is known as casting. For casting of an integer variable a into float you have write (float) a.

Expression m = a/b may therefore be written as

M = (float) a/ b

The general form of a cast is :

(Data type) Expression

The expression may be constant, variable or an expression.

Control Structure

Introduction:

In a program made up of only simple statements like input-output and assignment statements, the flow of execution, also called the control flow, is strictly sequential. When the control flow is sequential, the statements are executed in the order in which they appear in the program.
Programming constructs used to alter the flow of control in program from normal sequential execution are called control Structures.
There are three types of control structures.

(a) Conditional Control Transfer or Decision Statements: This structure selects one of two alternate segments of program code depending upon whether a given condition is true or false. The statements that we are going to discuss in this chapter are:

- **(i)** If statement
- **(ii)** If-else statement
- **(iii)** Switch statement

(b) Unconditional control transfer: This structure transfer control unconditionally. i.e. the control transfer is not based on the value of any condition. Example of this structure is GOTO statement.

(c) Repetitive Statements: Repetitive Statements are used to repeatedly execute a segment of code. These statements are also called loops. The body of the loop is executed till the specified condition is satisfied .

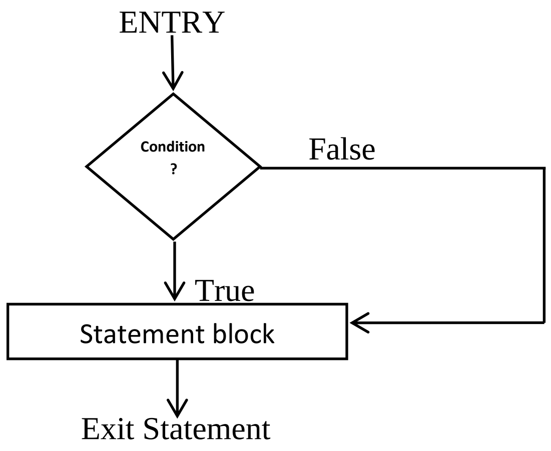
- **(i)** While loop
- **(ii)** Do loop
- **(iii)** For loop

If Statement:

The general form of a simple if statement is :

```
if(condition )
{
    Statement block:
}
```

If the condition is true, the statement-block will be executed; otherwise the statement block will be skipped. Flow chart of this statement is shown in figure.



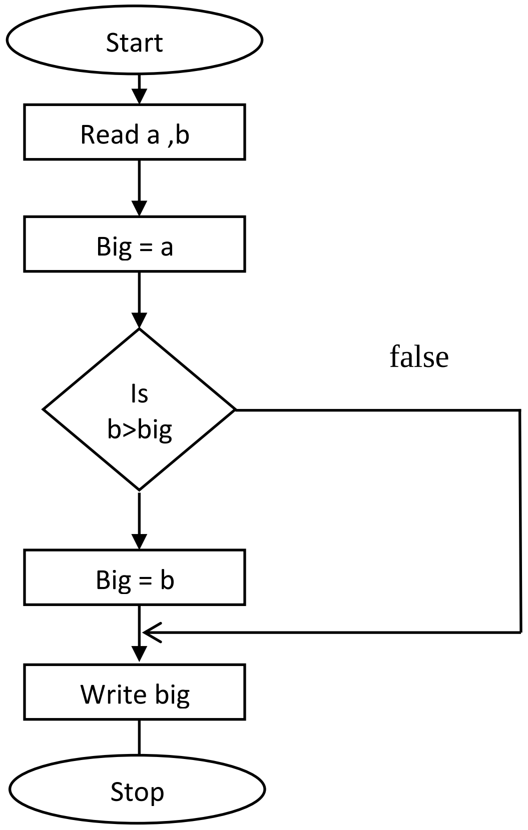
Whenever a condition is evaluated in C, it is given a value 1 if condition is true, and 0 if it is false

Example:
Write a program to find the larger of two given numbers.

Algorithm:

```
1. Input two numbers a and b
2.  Assign big = a;
3.  If ( b > big) then big = b
4.  Output big
5. Stop
```

Flowchart:



Program:

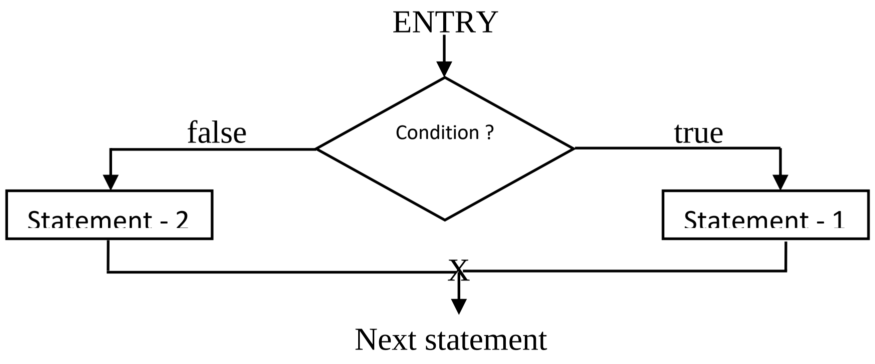
```
#include<stdio.h>
main()
{
    Float a,b,big;
    Printf("enter two numbers\n");
    Scanf("%f%f", &a,&b);
    Big =a ;
    if(b > big)   big = b;
    printf("largest number is : %f", big);
    }
```

If Else Statement:

The general form of this statement is

```
If  (condition)
{
    Statement -1;
}
Else
{
    Statement-2;
}
```

If the given condition is true then statement -1 is executed otherwise statement-2 will be executed. Flowchart of this statement is shown in figure.



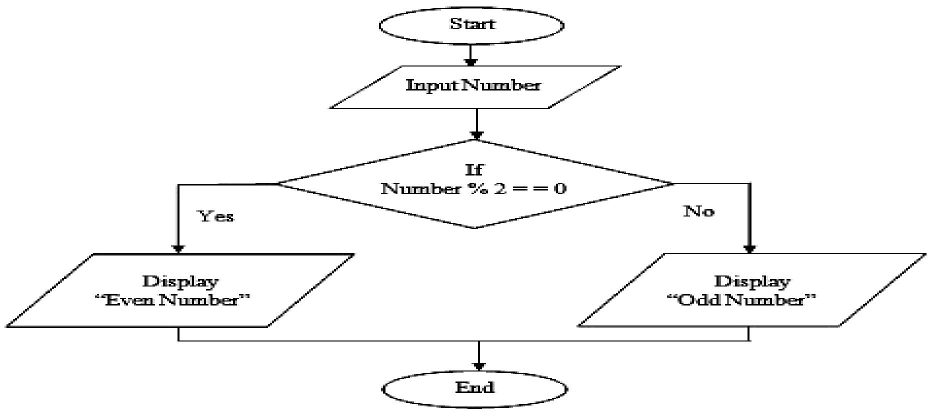
Example:

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

Algorithm:

```
1. input a number, Number
2. If ( Number MOD 2 ) = 0 then
           Output ' number is even'
   Otherwise
           Output ' number is odd '
3. Stop
```

Flowchart:



Program:

```
#include <stdio.h>
main()

{

    int number;

    printf (“enter a number”);

    scanf (“%d” , &number);

    if (number % 2 = =0)

        printf(“number is even”);

    else

        printf(“number is odd”);

}
```

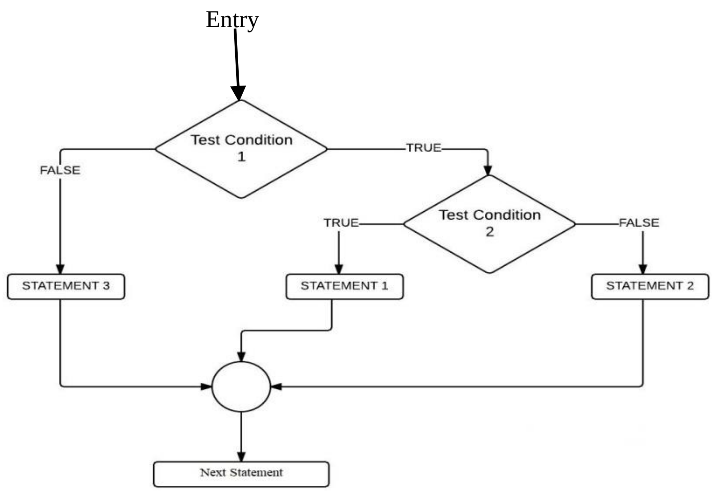
Nested if Statement:

The if statement may contain one or more if statements. In such a case, they are said to be nested. An example of a nested if statement is shown below.

```
if condition -1
{
        if condition -2
        {
            Statement-1;
        }
        else
        {
            Statement -2;
    }
    else
    {
            Statement -3
    }
```

If condition-1 is true then the condition-2 is checked. If condition -2 is true then statement-1 is executed , otherwise statement -2 is executed. If condition -1 is false, statement -3 is executed.

Flowchart:



Program:

```
    #include<stdio.h>
    main()
    {
        Float a,b,c,big;
        printf("enter three numbers\n");
        scanf("%f%f%f, &a,&b,&c);
    if(a>b)
    {
        if(a >b) big =a;
        else
        big=c;
}
else
{
    If(b>c) big=b;
    else
    big=c;
}
Printf("largest number is %f, big);
}
```

Loops

Introduction:

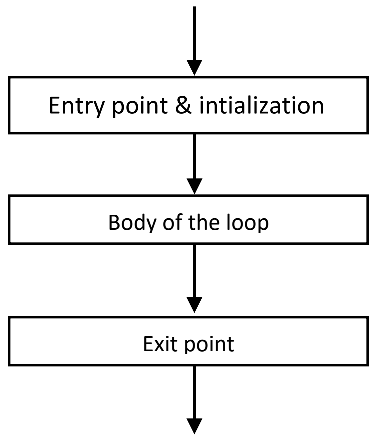
The versatility of the computer lies in its ability to perform a set of instructions repeatedly. This involves repeating some portion of the program either a specified number of times or until a particular condition is being satisfied. This repetitive operation is done through a loop control instruction.

Repetitive structure consists:

(i) Entry point & Initialization: In a repetitive structure, before or at the point of entry into the structure, some variables are initialized which are controlled during execution of repetitive structure.

(ii) Loop: A set of statements are executed using repetitive structure. These statements are executed many times while these statements appear once in program.

(iii) Exit point: There must be an exit point to come out of loop so that loop does not become infinite loop.



Types of repetitive structures:

There are two types of repetitive structures:

1. Condition controlled: In this loop, body is repetitively executed until the given condition become true.

Example of condition controlled loop in C are:

(i) While statement

(ii) do statement

2. Counter Controlled : In the counter controlled method, we know that the number of times the set of statement (body of the loop) is to be executed. In this type of structure we use a counter whose value changes during execution. When counter attains a predetermined value, the repetition is terminated. Counter controlled structure in C is made by using for loop.

While Statement:

The while loop is often used when the number of times the loop has to be executed is not known in advance. The general form of this statement is

```
While (condition)

{

    Statement (s);

}
```

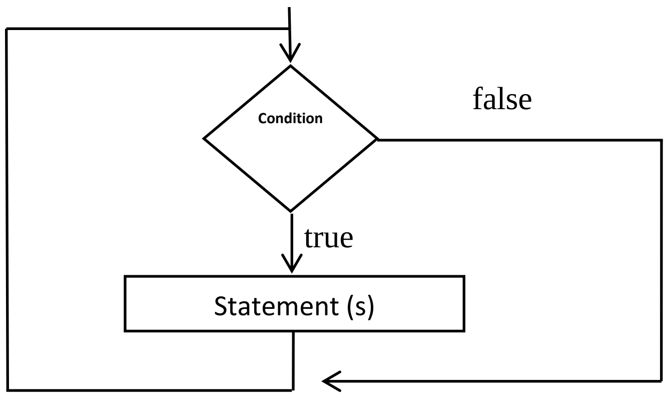
The sequence of operations in a while loop is an follows:

(i) Evaluate the condition

(ii) If the condition is true then execute the statement (s) and repeat step 1.

(iii) If the condition is false then the control is transferred out of the loop.

Flowchart :



Program:

Write a program to print 1 to 10 number.

```
#include<stdio.h>
main()
{
            int i=1;
             while(i <= 10)
            {
                    printf("%d\n",i);
                    i++;
            }

}
```

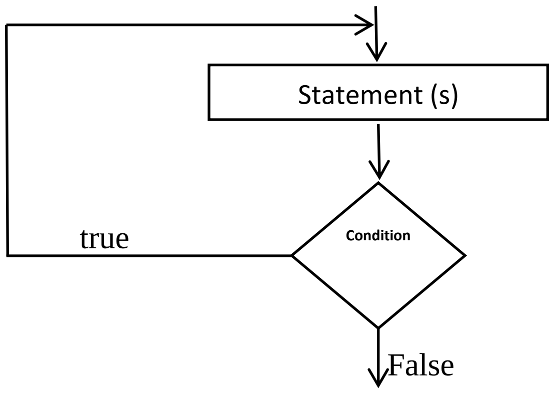
Do Statement:

In while statement, condition is evaluated first. Therefore, the body of the loop may not be executed at all if the condition is not satisfied at the very first attempt. But , in do loop, condition is evaluated at the end. Therefore, the body of the loop is executed at least once in this statement.

The general form of this statement is

```
Do
{
    Statement(s);
}
While (condition);
```

The statement(s) is executed first. Then the condition is tested. If condition is true, the statement is executed again. This process of repeated execution of the statement (s) continues until the condition finally becomes false. Flowchart of this statement is shown in figure.



Program:

Write program to print 1 to 10 number using do while loop

```
#include<stdio.h>
 main()
{
        int p=1;
do
        {
            printf("%d\t",p);
            p++;
        }
while(p<=10);
}
```

Difference between do loop and while loop:

The difference between while and do statement is as:

1. In while the condition is tested before executing the body of the loop. In do the condition is tested after executing the body of the loop.
2. Body of do loop is executed at least once, but the body of while loop may not be executed at all. In while, if initially the condition is not satisfied the body of the lop will not get executed even once.

The for loop:

There are situations where you want to have statement or group of statement to be executed number of times and the number of repetitions does not depend on the condition but it is simply a repetition upto a certain numbers. The best way for this type of repetition is a for loop. The general form of the for loop for single statement is :

```
for ( initialization ; condition; increment )
{
        Statement (s);
}
```

The following is a for loop example

```
for ( i=2;i<=10; i++)
{
        statement(s);
}
```

Here variable i has been initialized to 1. The statements in the loop will be executed as long as the condition (i<=10) is true and i will be increased by 1 every time the compiler comes to the end of the loop.

The execution of the for statements is as follows:

(1) Initialization of the control variable is done first, using assignment statements such as i =1

(2) Evaluate the condition next such as i <=10, if the condition is true, the body of the loop (statement (s)) is executed, otherwise the loop is terminated.

(3) Now the control variable is incremented using as assignment statement such as i = i+1 and repeat step 2.

Program:

Write a program of find 1 to 10 number using for loop.

```
#include <stdio.h>

void main(void)
{
int count;
// Display the numbers 1 through 10
for(count = 1; count <= 10; count++)
printf("%d ", count);
printf("\n");
}
```

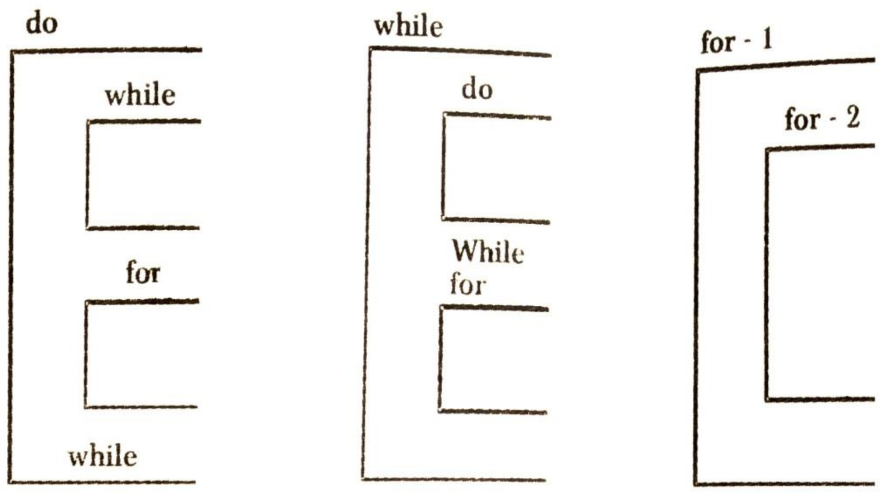
Output example:

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

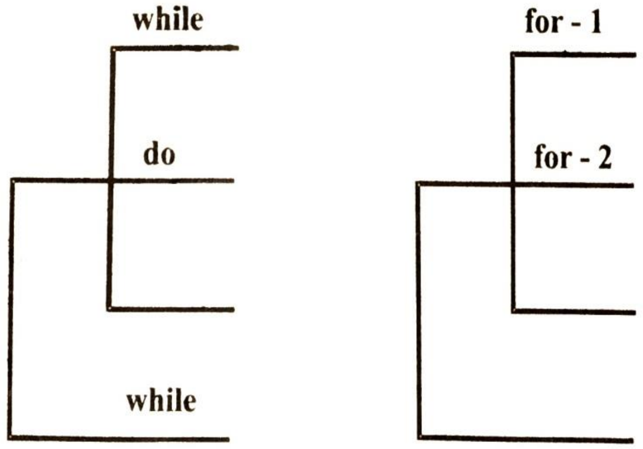
Nested Loops:

Loop constructs can be nested or embedded within one another. The inner loop must be completely embedded within the outer loop. There should be no overlapping of loops. The following rules apply to the use of range of while , do and for in a program.

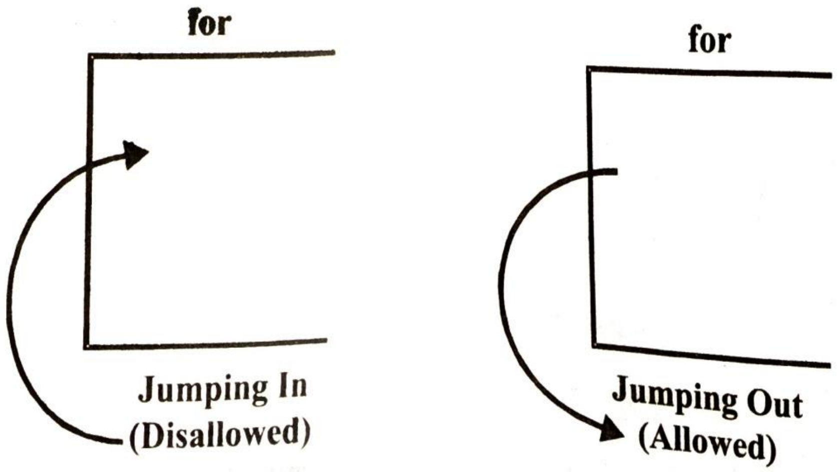
1. loops mav be nested as



2. loops should not be overlap



3. Jumping out of the loop is allowed but jumping in is not allowed.



Program:

```
#include <stdio.h>
int main()
{
   int i=1,j;
   while (i <= 5)
   {
          j=1;
         while (j <= i )
         {
             printf("%d ",j);
             j++;
         }
              printf("\n");
             i++;
   }
   return 0;
}
```

Output:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

Switch statement:

The switch statement tests the value of a given expression against a list of case values and when a match is found, a statement or statements associated with the case is executed.

The general for of this statement is

```
Switch ( expression)
{
        Case val-1 :
                    Statement -1;
                    Break;
        Case val-2 :
                    Statement -2;
                    Break;
        ……………..
        ……………..
        Default:
                    Default – statement;
                    Break;
   }
```

The expression can be an integer or character data type. The expression is evaluated and the statement whose value matches with the value of the expression is selected for execution. If the value of expression does not match any of the case values then default-statement will be executed. The break statement at the end of each statement signals the end of a particular case and causes an exit from the switch statement, transferring the control to the statement following the switch.

Program:

```
main( )
{
        int   i = 2 ;
        switch ( i )
            {
                case 1 :
                        printf ( "I am in case 1 \n" ) ;
                        break ;
                case 2 :
                        printf ( "I am in case 2 \n" ) ;
                        break ;
                case 3 :
                        printf ( "I am in case 3 \n" ) ;
                        break ;
                default :
                        printf ( "I am in default \n" ) ;
        }
}
```

The output of this program would be:

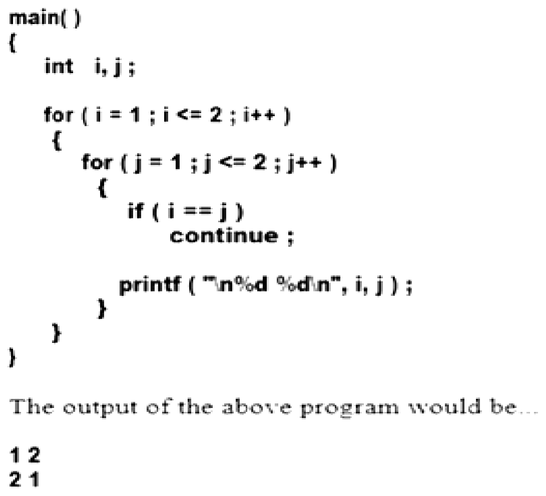
```
I am in case 2
```

Continue Statement:

In some programming situations we want to take the control to the beginning of the loop, bypassing the statements inside the loop, which have not yet been executed. The keyword continue allows us to do this. When continue is encountered inside any loop, control automatically passes to the beginning of the loop.

A continue is usually associated with an if.

As an example, let's consider the following program.



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

    for ( i = 1 ; i <= 2 ; i++ )
     {
        for ( j = 1 ; j <= 2 ; j++ )
         {
            if ( i == j )
                continue ;

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

The output of the above program would be...

```
1 2
2 1
```

GoTo Statement:

This statement transfer control unconditionally to the statement specified by the label. The general form of this statement is .

Goto label;

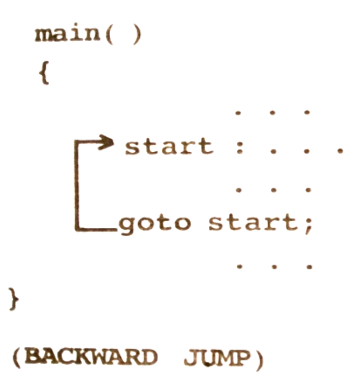
A label is any valid variable name. only one statement may be prefixed by a particular label. A statement is prefixed by a label in the following manner,

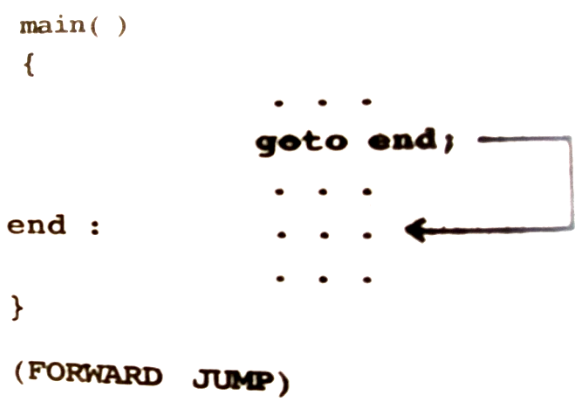
Label : Statement ;

A colon separates the label and the labelled statement.

Example:

The following programs skelton, shows the use of goto and the labelling of statements.





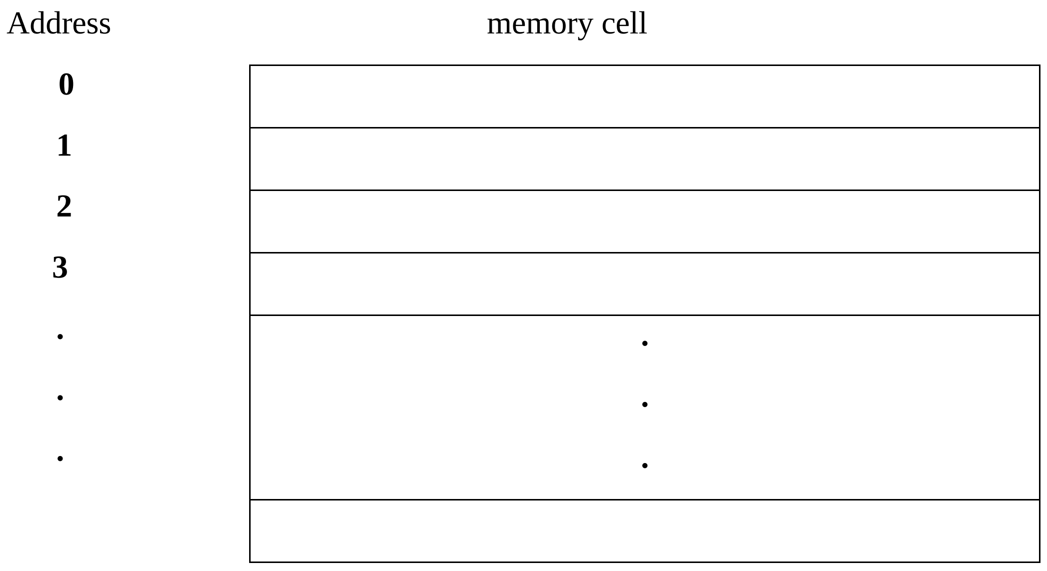
Finally, it should be noted that the use of the goto statement is discouraged in C since it alters the clear and sequential flow of logic that is characteristics of the language. Some computer scientists recommended a total ban on this statement, through this may be a bit extreme.

Program:

```
/*To print numbers from 1 to 10 using goto statement*/
#include <stdio.h>
int main()
{
        int number;
        number=1;
        repeat:
            printf("%d\n",number);
        number++;
          if(number<=10)
                goto repeat;
    return 0;
}
```

INTRODUCTION:

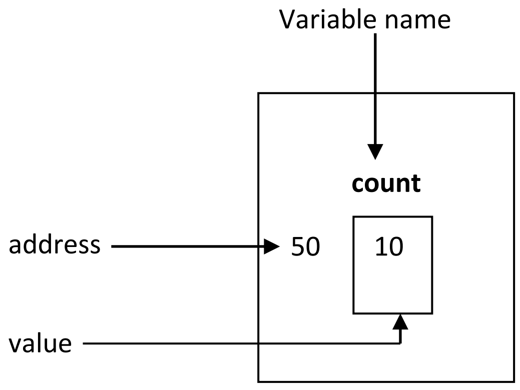
As we know, computer use their memory for storing the instruction of a program, as well as the values of the variables that are associated with it. The computer's memory is a sequence collection of storage cells as shown in figure:



Whenever we declare a variable, the system allocates an appropriate location or cell to hold the value of the variable. For example, consider a integer variable count and consider the following assignment statement.

```
Count = 10;
```

This statement instructs the system to find a location or cell for integer variable count and puts the value 10 in that location. Let us assume that the system has selected the memory location 50 for count as shown in figure.



Now, we may have access to the value 10 by using either the name count or the address 50. We can define a variable which stores the memory address. Such variables that hold memory addresses are called **Pointer.**

Pointer Variable:

A pointer is a variable that stores the address of some other variable. In some situations, it is easy to access the variable through its storage address in the main memory. The pointer variables can also access the values stored in the variable whose address they contain. In this way, the pointer is much more powerful than an ordinary variable.

Advantages of pointer variable:

The advantages of the pointer variables are:

(i) The storage size can be adjusted dynamically as desired by the program.

(ii) Storage may be shared among several variables.

(iii) Complex data structures, like linked list, stack, queue etc, may be designed and manipulated.

(iv) Pointer increase the execution speed.

(v) Pointer reduce the length and complexity of a program.

(vi) With the use of pointer, one can return multiple value from a function.

(vii) Pointer are used to efficiently handle the array or data types.

(viii) By using pointers, one can pass an array or string as a parameter to a function.

(ix) A pointer enables us to access a variable that is defined outside the function.

(x) The use of pointer array to character string results in saving of data storage space in memory.

Pointer Operators

Let us consider two main pointer operations:

(a) The 'addess of' operators:

The declaration statement

Int alpha = 1;

tells the compiler to

- Reserve just enough space in memory to hold an integer value.
- Associate the name alpha with this space.
- Store the value 1 there.

(b) The 'Value of address' operations:

The other pointer operator available in C is '*' called 'value of address' operator. It gives the value stored at a particular address. The value of address operator is also called indirection operator.

Since the operand of the indirection operator is the address of a variable stored in memory. And it returns the value stored at that address.

Declaration of a pointer data type:

Since addresses themselves are value they can be given name and stored in memory just like any other value. To do this , we have to make a declaration of the form.

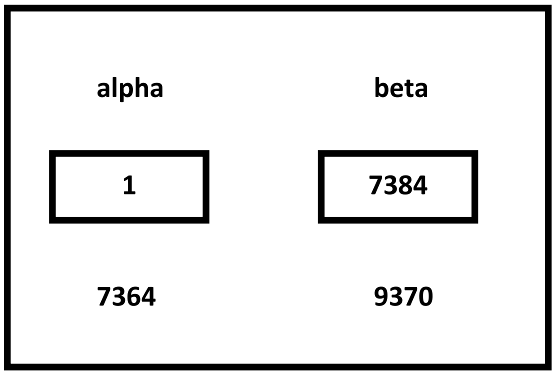
Int * beta;

This is pointer declaration. This tells the compiler that beta is a variable which will be used to store the address of an integer. We can assign to beta the address of alpha using the assignment statement as follows:

Beta = & alpha

Now, the pointer variable beta points to the integer variable alpha.

Since beta is also a variable, the compiler allocates space for it in the memory like for any other variable. The following memory diagram would illustrate the contents of alpha and beta.



It is clear form the diagram that beta's value is alpha's address.

Example:

To illustrate the above, consider the following program.

```
#include<stdio.h>

Main ()

{

Int alpha;

Int *beta;

Alpha=1;

Beta= &alpha;

Printf (“the value %d is stored at address %d\n”, alpha, &alpha);

Printf (“the value %d is stored at address %d\n”, beta, &beta);

Printf (“the value %d is stored at address %d\n”, beta, beta);

}
```

Function

Introduction of function:

A function is a self-contained block of statements that perform a coherent task of some kind. Every C program can be thought of as a collection of these functions. As we noted earlier, using a function is something like hiring a person to do a specific job for you. Sometimes the interaction with this person is very simple; sometimes it's complex.

Suppose you have a task that is always performed exactly in the same way—say a bimonthly servicing of your motorbike. When you want it to be done, you go to the service station and say, "It's time, do it now". You don't need to give instructions, because the mechanic knows his job. You don't need to be told when the job is done. You assume the bike would be serviced in the usual way, the mechanic does it and that's that.

Types of functions

1) **Predefined standard library functions** – such as puts(), gets(), printf(), scanf() etc – These are the functions which already have a definition in header files (.h files like stdio.h), so we just call them whenever there is a need to use them.

2) **User Defined functions –** The functions that we create in a program are known as user defined functions.

In this guide, we will learn how to create user defined functions and how to use them in C Programming

Why we need functions in C

Functions are used because of following reasons –
a) To improve the readability of code.
b) Improves the reusability of the code, same function can be used in any program rather than writing the same code from scratch.
c) Debugging of the code would be easier if you use functions, as errors are easy to be traced.
d) Reduces the size of the code, duplicate set of statements are replaced by function calls.

Syntax of a function

```
return_type function_name (argument list)
{
   Set of statements – Block of code
}
```

return_type: Return type can be of any data type such as int, double, char, void, short etc. Don't worry you will understand these terms better once you go through the examples below.

function_name: It can be anything, however it is advised to have a meaningful name for the functions so that it would be easy to understand the purpose of function just by seeing it's name.

argument list: Argument list contains variables names along with their data types. These arguments are kind of inputs for the function. For example – A function which is used to add two integer variables, will be having two integer argument.

Block of code: Set of C statements, which will be executed whenever a call will be made to the function.

Do you find above terms confusing? – Do not worry I'm not gonna end this guide until you learn all of them :)
Lets take an example – Suppose you want to create a function to add two integer variables.

Let's split the problem so that it would be easy to understand –
Function will add the two numbers so it should have some meaningful name like sum, addition, etc. For example lets take the name addition for this function.

```
return_type addition(argument list)
```

This function addition adds two integer variables, which means I need two integer variable as input, lets provide two integer parameters in the function signature. The function signature would be –

```
return_type addition(int num1, int num2)
```

The result of the sum of two integers would be integer only. Hence function should return an integer value – **I got my return type** – It would be integer –

```
int addition(int num1, int num2);
```

So you got your function prototype or signature. Now you can implement the logic in C program like this:

How to call a function in C?

Consider the following C program

Example1: Creating a user defined function addition()

```
#include <stdio.h>
int addition(int num1, int num2)
{
    int sum;
    /* Arguments are used here*/
    sum = num1+num2;

    /* Function return type is integer so we are returning
     * an integer value, the sum of the passed numbers.
     */
    return sum;
}

int main()
{
    int var1, var2;
    printf("Enter number 1: ");
    scanf("%d",&var1);
    printf("Enter number 2: ");
    scanf("%d",&var2);

    /* Calling the function here, the function return type
     * is integer so we need an integer variable to hold the
     * returned value of this function.
     */
    int res = addition(var1, var2);
    printf ("Output: %d", res);

    return 0;
}
```
Output:

```
Enter number 1: 100
Enter number 2: 120
Output: 220
```

Global and Local Variable:

A local variable is a variable that is declared inside a function. A global variable is a variable that is declared outside all functions. A local variable can only be used in the function where it is declared. A global variable can be used in all functions.

See the following example:

```
        #include<stdio.h>
        // Global variables//
                int A;
                int B;
                int Add()
        {
                return A + B;
        }
                int main()
        {
                int answer; // Local variable
                A = 5;
                B = 7;
                answer = Add();
                printf("%d\n",answer);
                return 0;
        }
```

As you can see two global variables are declared, A and B. These variables can be used in main() and Add(). The local variable answer can only be used in main().

Function Declaration:

A function (method) is a block of code that can be called from another location in the program or class. It is used to reduce the repetition of multiple lines of code.

Syntax

returnType functionName (parameterTypes); //function prototype

//main code

```
returnType functionName (functionParameters)

{

    //function implementation

    //statements that execute when called return value;

}
```

Example

```
int findMaximum(int, int); //prototype

void main()
{
   int maxNumber = findMaximum(5, 7); //calling a function
}

int findMaximum(int number1, int number2)
{ //implementation
int maximum = number2;
  If (number1 > number2) maximum = number1;
  return maximum;
}
```

Standard Function:

The standard functions are built-in functions. In C programming language, the standard functions are declared in header files and defined in .dll files. In simple words, the standard functions can be defined as "the ready made functions defined by the system to make coding more easy". The standard functions are also called as **library functions** or **pre-defined functions**.

In C when we use standard functions, we must include the respective header file using **#include** statement. For example, the function **printf()** is defined in header file **stdio.h** (Standard Input Output header file). When we use **printf()** in our program, we must include **stdio.h** header file using **#include<stdio.h>** statement.

C Programming Language provides the following header files with standard functions.

Header File	Purpose	Example Functions
stdio.h	Provides functions to perform standard I/O operations	printf(), scanf()
conio.h	Provides functions to perform console I/O operations	clrscr(), getch()
math.h	Provides functions to perform mathematical operations	sqrt(), pow()
string.h	Provides functions to handle string data values	strlen(), strcpy()
stdlib.h	Provides functions to perform general functions/td>	calloc(), malloc()
time.h	Provides functions to perform operations on time and date	time(), localtime()

ctype.h	Provides functions to perform - testing and mapping of character data values	isalpha(), islower()
setjmp.h	Provides functions that are used in function calls	setjump(), longjump()
signal.h	Provides functions to handle signals during program execution	signal(), raise()
assert.h	Provides Macro that is used to verify assumptions made by the program	assert()
locale.h	Defines the location specific settings such as date formats and currency symbols	setlocale()
stdarg.h	Used to get the arguments in a function if the arguments are not specified by the function	va_start(), va_end(), va_arg()
errno.h	Provides macros to handle the system calls	Error, errno
graphics.h	Provides functions to draw graphics.	circle(), rectangle()
float.h	Provides constants related to floating point data values	
stddef.h	Defines various variable types	
limits.h	Defines the maximum and minimum values of various variable types like char, int and long	

Parameter Passing

When a function gets executed in the program, the execution control is transferred from calling-function to called function and executes function definition, and finally comes back to the calling function. When the execution control is transferred from calling-function to called-function it may carry one or number of data values. These data values are called as **parameters**.

Parameters are the data values that are passed from calling function to called function.

In C, there are two types of parameters and they are as follows...

- **Actual Parameters**
- **Formal Parameters**

The **actual parameters** are the parameters that are speficified in calling function. The **formal parameters** are the parameters that are declared at called function. When a function gets executed, the copy of actual parameter values are copied into formal parameters.

In C Programming Language, there are two methods to pass parameters from calling function to called function and they are as follows...

- **Call by Value**
- **Call by Reference**

Call by Value

In call by value parameter passing method, the copy of actual parameter values are copied to formal parameters and these formal parameters are used in called function. The changes made on the formal parameters do not effect the values of actual parameters. That means, after the execution control comes back to the calling function, the actual parameter values remains same. For example consider the following program...

Example Program

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

void main(){
  int num1, num2 ;
  void swap(int,int) ; // function declaration
  clrscr() ;
  num1 = 10 ;
  num2 = 20 ;

  printf("\nBefore swap: num1 = %d, num2 = %d", num1, num2) ;

  swap(num1, num2) ; // calling function

  printf("\nAfter swap: num1 = %d\nnum2 = %d", num1, num2);
  getch() ;
}
void swap(int a, int b)  // called function
{
  int temp ;
  temp = a ;
  a = b ;
  b = temp ;
}
```

In the above example program, the variables num1 and num2 are called actual parameters and the variables a and b are called formal parameters. The value of num1 is copied into a and the value of num2 is copied into b. The changes made on variables a and b does not affect the values of num1 and num2.

Call by Reference

In Call by Reference parameter passing method, the memory location address of the actual parameters is copied to formal parameters. This address is used to access the memory locations of the actual parameters in called function. In this method of parameter passing, the formal parameters must be pointer variables.

That means in call by reference parameter passing method, the address of the actual parameters is passed to the called function and is received by the formal parameters (pointers). Whenever we use these formal parameters in called function, they directly access the memory locations of actual parameters. So the changes made on the formal parameters effects the values of actual parameters. For example consider the following program...

Example Program

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

void main(){
   int num1, num2 ;
   void swap(int *,int *) ; // function declaration
   clrscr() ;
   num1 = 10 ;
   num2 = 20 ;

   printf("\nBefore swap: num1 = %d, num2 = %d", num1, num2) ;
   swap(&num1, &num2) ; // calling function

   printf("\nAfter swap: num1 = %d, num2 = %d", num1, num2);
   getch() ;
}
void swap(int *a, int *b)  // called function
{
   int temp ;
   temp = *a ;
   *a = *b ;
   *b = temp ;
}
```

Array

In mathematics, we are familiar with the notaion : x1, x2, ………………, x100 are the subscripts of variable x. We can say that variable x has 100 elements. Thus we have defined 100 variables in the program to use them. C provides another way to represent such type of variables. This is done by writing the subscript between two square brackets as : X[1], x[2], ……………..x[100]. Such variable are called subscripted variables. The general form may be specified as:

name [s1, s2,………………];

where

name is the name of subscripted variable and S1, S2 ……………..are the subscripts

The subscripts of a variable may be

(a) integer constant

(b) integer variable

(c) integer expressions

Array Declaration:

Array store elements of the same data types. Like any other data element, an array has to be declared before it is used. The general form of declaration of one dimensional array is

Data type variable-name [size]

Data type indicated the type of values that are going to be stored in array elements, that is, it specifies the type of array element. Size indicates the maximum number of elements that can be stored inside the array.

The array declaration provides the following information to the compiler.

- Name of the array
- Types of the elements to be stored in the array
- Number of subscript it has and
- Array size

Example:

Float a[10];

The declaration defines **a** to be an array consisting of 10 real elements that are designed as a[0], a[1],…….a[9].

Initialization of Arrays:

An array can be initialized along with the declaration by placing the elements of the array, separated by comas within braces. The general form of initialization of array is:

Static Data-type variable –name [size] = {list of values};

Example:

The statement

Static int digits [5] = {1, 2, 3, 4, 5};

Will declare the variable digits as an array of size 5 and will assign the value of individual array element as follows:

Digit [0] = 0;
Digit [1] = 1;
Digit [2] = 2;
Digit [3] = 3;
Digit [4] = 4;

All individual array elements, that are not assigned explicit initial values, will automatically be set by zero.

Input and output of arrays:

The entire array can not be input/output using a single input/output statement. To read/write an array, the element of the array have to be read/ write individually. Therefore, an array can be read (or write) by reading (or writing) the element inside a loop that iterates as many times as there are element in the array. The for-do loop is generally used to input/ output array elements.

Example:

```
/ Program to Read and Display Array Elements

#include <stdio.h>
int main()
{
    int subjects_marks[5];
    int index=0;

    printf("\nENTER MARKS FOR 5 SUBJECTS: \n");
    for(index=0; index < 5; index=index+1)
    {
        printf("\n\tENTER  MARKS FOR SUBJECT %d: ",index+1);
        scanf("%d", &subjects_marks[index]);
    }

    printf("\nSUBJECT MARKS ARE : \n");
    for(index=0; index < 5; index=index+1)
    {
        printf("\n\t SUBJECT %d  MARKS ARE %d \n",index+1 , subjects_marks[index]);
    }

    printf("\n----------------------DIGITALPADM.COM");
    return 0;
}
```

Types of array:

(a) One (Single) Dimensional Array.

(b) Two Dimensional Array.

(a) One dimensional Array: An array is a set of elements of the same data type represented by a single variable name. Each individual array elements can be referred to by specifying the array name, followed by a subscript, enclosed in square bracket.

Example:

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

```
int main()
{
   int i;

   // declaring and Initialising array in C
   int value[6] = {5,10,15,20,25,30};

   for (i=0;i<6;i++)
   {
      // Accessing each variable using for loop
      printf("Position : [%d] , Value : %d \n", i, value[i]);
   }

   // Wait For Output Screen
   getch();

   //Main Function return Statement
    return 0;
}
```

Sample Output:

```
Position : [0] , Value : 5
Position : [1] , Value : 10
Position : [2] , Value : 15
Position : [3] , Value : 20
Position : [4] , Value : 25
Position : [5] , Value : 30
```

(b) Two Dimensional Array:

An array of arrays is known as 2D array. The two dimensional (2D) array in C programming is also known as matrix. A matrix can be represented as a table of rows and columns. Before we discuss more about two Dimensional array lets have a look at the following C program.

Initialization of 2D Array:

There are two ways to initialize a two Dimensional arrays during declaration.

```
int disp[2][4] = {
   {10, 11, 12, 13},
   {14, 15, 16, 17}
};
```

OR

```
int disp[2][4] = { 10, 11, 12, 13, 14, 15, 16, 17};
```

Although both the above declarations are valid, I recommend you to use the first method as it is more readable, because you can visualize the rows and columns of 2d array in this method.

Example:

Following C program ask to the user to enter row and column size of the array, then ask to the user to enter the array elements to initialize the array element in the array, and the program will display the two dimensional array:

```
/* C Program - Two Dimensional Array */

#include<stdio.h>
#include<conio.h>
void main()
{
		clrscr();
		int arr[10][10], row, col, i, j;
		printf("Enter number of row for Array (max 10) : ");
		scanf("%d",&row);
		printf("Enter number of column for Array (max 10) : ");
		scanf("%d",&col);
		printf("Now Enter %d*%d Array Elements : ",row, col);
		for(i=0; i<row; i++)
		{
				for(j=0; j<col; j++)
				{
						scanf("%d",&arr[i][j]);
				}
		}
				printf("The Array is :\n");
		for(i=0; i<row; i++)
		{
				for(j=0; j<col; j++)
				{
						printf("%d  ",arr[i][j]);
				}
				printf("\n");
		}
```

```
        getch();
}
```

String: Strings are defined as an array of characters. The difference between a character array and a string is the string is terminated with a special character '\0'.

Declaration of strings: Declaring a string is as simple as declaring a one dimensional array. Below is the basic syntax for declaring a string.

char str_name[size];

In the above syntax str_name is any name given to the string variable and size is used define the length of the string, i.e the number of characters strings will store. Please keep in mind that there is an extra terminating character which is the Null character ('\0') used to indicate termination of string which differs strings from normal character arrays.

String variable: A string variable is an array of characters that has a defined length. It can be used to store string values. In C language, a special character, called the Null Character ("\0"), is insterted at the end of each string variable. The ASCII value of this character is zero. It can be used to mark the end of data in an array of defined length. In C, we don't have a separate data type to store strings. We use character arrays to store strings. We generally take an array of static length that is large enough to hold data. For example, and array of 20 character can be taken to store names of student. But , in case of student name "Dheeraj", only 7 characters will be utilized. So , C will automatically put a null character after the last character of a string variable.

String Function:

The more commonly-used string functions

The nine most commonly used functions in the string library are:

- strcat - concatenate two strings.
- strchr - string scanning operation.
- strcmp - compare two strings.
- strcpy - copy a string.
- strlen - get string length.
- strncat - concatenate one string with part of another.
- strncmp - compare parts of two strings.
- strncpy - copy part of a string.

StrCpy Function:
This function is used to copy one string to another.
Form is

Strcpy (string1, String2);

It assigns to contents of string 2 to string 1. Note that string 2 may be a string constant or string variable.

Example:

```
#include<stdio.h>

#include<conio.h>

#include<string.h>
```

```
void main ( )

{

clrscr ( );

char s1[21]=”sarv”;

char s2 [20]=”institute”;

printf(“value of s1=%s”,s1);

printf (“\n value of s2=%s”,s2);

strcpy(s1,s2);

printf (“\n value of s1=%s”,s1);

getch( );

}
```

Output show:-

```
Value of s1= Sarv

Value of s2= Institute

Value of s1= Institute
```

StringCmp:

This function compares two strings and returns 0 if they are equal. It returns a negative number if the first string is alphabetically less than the second string or a positive number if the first string is greater than the second string. Form is

Strcmp (string1, string2);

Where string1 and string2 are the string constants or string variables.

Example:

```
#include <stdio.h>

#include <string.h>

 int main()

{

              char a[100], b[100];

              printf("Enter a string\n");

             gets(a);


             printf("Enter a string\n");
```

```
            gets(b);
             if (strcmp(a,b) == 0)
            printf("The strings are equal.\n");
            else
            printf("The strings are not equal.\n");
             return 0;
}
```

String len( ) Function: This function counts and returns the number of characters in a string. The null character ‘\0’ is not counted. Form is

N = strlen (string);

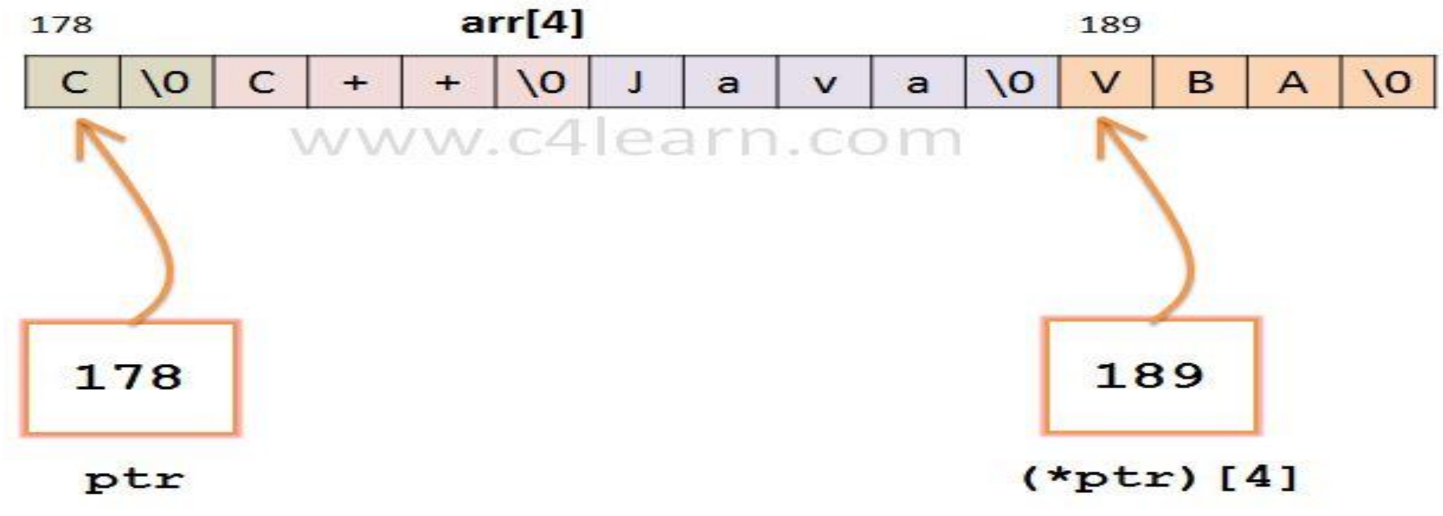
Example:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main ( )
{
clrscr ( );
char str [21];
int a;
printf (“\n enter a string =”);
gets (str);
a=strlen (str);
printf (“the length of given string is:%d”,a);
getch ( );
}
```

Output show :-

```
            Enter a string =123456
            The length of given string is =6
```

Pointer to array of string : A pointer which pointing to an array which content is string, is known as pointer to array of strings.



In this example

1. ptr : It is pointer to array of string of size 4.
2. Array [4] : It is an array and its content are string.

Example:
Printing Address of the Character Array

```
#include<stdio.h>


int main()
{
int i;


char *arr[4] = {"C","C++","Java","VBA"};
char *(*ptr)[4] = &arr;


for(i=0;i<4;i++)
    printf("Address of String %d : %u\n",i+1,(*ptr)[i]);


return 0;
}
```

Output :

```
Address of String 1 = 178
Address of String 2 = 180
Address of String 3 = 184
Address of String 4 = 189
```

Structure:

A structure is a collection of data items, usually of different data types, which are logically combined into a single unit. An entry in a telephone directory is a simple example of a structure. It consists of the name, address and telephone number of an individuals.

Features of Structure:

Features of structure are:

1. The data types of the elements of the structure need not be same.
2. Structures are user-defined data types.
3. Structure is a group of related data items (or elements). For example, an entry in a telephone directory.

Defining a Structure:

The structure is a composed of data items that may be of different data types. The data items of a structure are called fields. Each field has an identifier (ie. Field name) and a data type.

The general format of a structure definition is

```
 struct   sname
{
   //data type member 1
   // data type member 2
   // data type member 3
   ...
  // data type member -n
};
```

Where struct is a keyword also called tag, sname is the name, given to the structures data type. Member-1, member-2,…..member-n are element of the structure being defined.

The portion of the structure definition enclosed in braces is known as a structure template.

Accessing structure members:

1. Array elements are accessed using the Subscript variable , Similarly Structure members are accessed using dot [.] operator.
2. (.) is called as “Structure member Operator”.
3. Use this Operator in between **“Structure name”** & **“member name”**

Example:

```
#include<stdio.h>
struct Vehicle
    {
     int wheels;
    char vname[20];
    char color[10];
     } v1 = {4,"Nano","Red"};

int main()
{
            printf("Vehicle No of Wheels : %d",v1.wheels);
            printf("Vehicle Name             : %s",v1.vname);
            printf("Vehicle Color             : %s",v1.color);
            return(0);
}
```

C Structure Initialization

1. When we declare a structure, memory is not allocated for un-initialized variable.

Declare and Initialize

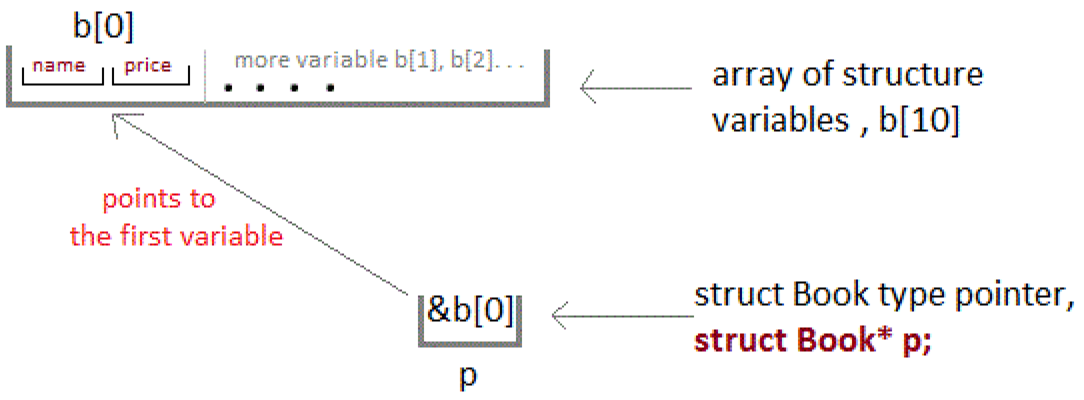
```
struct student
{
   char name[20];
   int roll;
   float marks;
}std1 = { "Pritesh",67,78.3 };
```

Pointer to a structure:

Pointer to Array of Structures in C

Like we have array of integers, array of pointers etc, we can also have array of structure variables. And to use the array of structure variables efficiently, we use pointers of structure type. We can also have pointer to a single structure variable, but it is mostly used when we are dealing with array of structure variables.

```
#include <stdio.h>
struct Book
{
    char name[10];
    int price;
}
int main()
{
    struct Book a;      //Single structure variable
    struct Book* ptr;   //Pointer of Structure type
    ptr = &a;
     struct Book b[10];  //Array of structure variables
    struct Book* p;     //Pointer of Structure type
    p = &b;
    return 0;
}
```


b[0]
name
price
more variable b[1], b[2]. . .
array of structure variables , b[10]
points to the first variable
&b[0]
p
struct Book type pointer,
struct Book* p;

Accessing Structure Members with Pointer

To access members of structure using the structure variable, we used the dot .operator. But when we have a pointer of structure type, we use arrow -> to access structure members.

```
#include <stdio.h>
struct my_structure
{
   char name[20];
   int number;
   int rank;
};
int main()
{
   struct my_structure variable = {"StudyTonight", 35, 1};
   struct my_structure *ptr;
   ptr = &variable;
   printf("NAME: %s\n", ptr->name);
   printf("NUMBER: %d\n", ptr->number);
   printf("RANK: %d", ptr->rank);
   return 0;
}
```

Output:

```
NAME: StudyTonight
NUMBER: 35
RANK: 1
```

UNION: Unions, like structures, contain members whose individual data types may differ from one another. However, the members that compose a union all share the same storage area within the computer's memory, whereas each member within a structure is assigned its own unique storage area. Thus, union are used to conserver memory. They are infact for application involving multiple members, where values need not be assigned to all of the members at any one time.

Difference between a Structure and UNION:

The structure stores individual elements in contiguous memory locations where a union stores all the members in one memory area that is equal to the largest area needed by a member a variable. Therefore, the storage space allocated to a structure variable is equal to the sum of the storage space required by all the data members whereas, in case of union, the storage space allocated to a variable of union is equal to the storage space needed by the largest data member of the union.

Union is also like structure, i.e. collection of different data types which are grouped together. Each element in a union is called member.

Union and structure in C are same in concepts, except allocating memory for their members.

Structure allocates storage space for all its members separately.

Whereas, Union allocates one common storage space for all its members

We can access only one member of union at a time. We can't access all member values at the same time in union. But, structure can access all member values at the same time. This is because, Union allocates one common storage space for all its members. Where as Structure allocates storage space for all its members separately.

Many union variables can be created in a program and memory will be allocated for each union variable separately.

Below table will help you how to form a C union, declare a union, initializing and accessing the members of the union.

Using normal variable	Using pointer variable

Syntax:
union tag_name
{
data type var_name1;
data type var_name2;
data type var_name3;
};
	Syntax:
	union tag_name
	{
	data type var_name1;
	data type var_name2;
	data type var_name3;
	};

Example:
union student
{
int mark;
char name[10];
float average;
};
	Example:
	union student
	{
	int mark;
	char name[10];
	float average;
	};

Declaring union using normal variable: union student report;	**Declaring union using pointer variable:** union student *report, rep;

Initializing union using normal variable:
union student report = {100, “Mani”, 99.5};
	Initializing union using pointer variable:
	union student rep = {100, “Mani”, 99.5};
	report = &rep;

Accessing union members using normal variable:
report.mark;
report.name;
report.average;
	Accessing union members using pointer variable:
	report -> mark;
	report -> name;
	report -> average;

DIFFERENCE BETWEEN STRUCTURE AND UNION IN C:

C Structure	C Union
Structure allocates storage space for all its members separately.	Union allocates one common storage space for all its members. Union finds that which of its member needs high storage space over other members and allocates that much space
Structure occupies higher memory space.	Union occupies lower memory space over structure.
We can access all members of structure at a time.	We can access only one member of union at a time.

Structure example:
struct student
{
int mark;
char name[6];
double average;
};
	Union example:
	union student
	{
	int mark;
	char name[6];
	double average;
	};

For above structure, memory allocation will be like below. int mark – 2B char name[6] – 6B double average – 8B Total memory allocation = 2+6+8 = 16 Bytes	For above union, only 8 bytes of memory will be allocated since double data type will occupy maximum space of memory over other data types. Total memory allocation = 8 Bytes