OM VINAYAKA OM SAI RAM

INTRODUCTION TO C-LANGUAGE

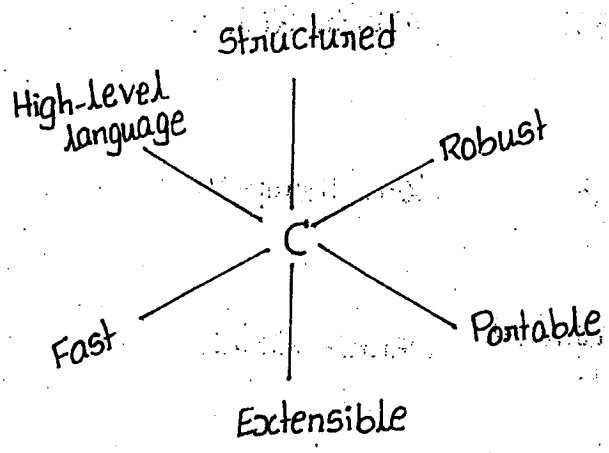
INTRODUCTION:-

⇒C is a powerful, portable and structured programming language.

⇒It combines the features of a high-level programming language with the elements of an assembly language, therefore it is suitable for writing both system software and application software.

⇒C has been used for implementing systems such as 1. Operating Systems
2. Compilers
3. Linkers
4. Word Processors
5. Utility Packages.

FEATURES OF C:-



⇒C is a robust language. It contains a rich set of built-in functions and operators, which can be used to write any complex programs

⇒C is highly portable. This means that C programs written for one computer can be run on another with little or no modification.

⇒Programs written in C are efficient and fast. This is due to variety of data types and powerful operators.

⇒ There are only 32 keywords in C.

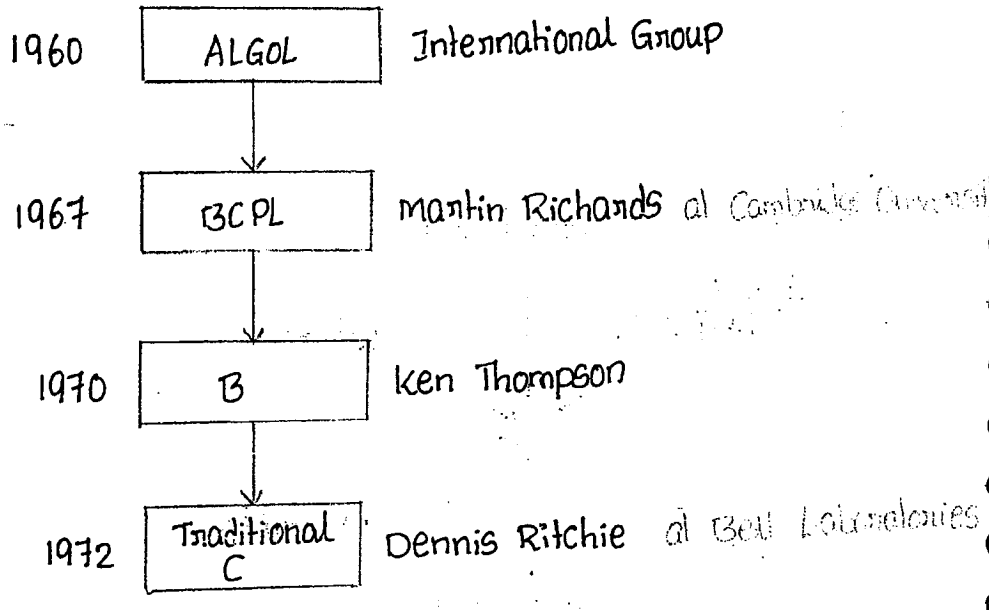
⇒ In C, several standard functions are available which can be used for developing programs.

⇒ C-language is well suited for structured programming.

⇒ C is extensible. Basically a 'C' program is a collection of functions that are supported by the C library. We can continuously add our own functions to C library.

HISTORY OF C-LANGUAGE:-

⇒ C is one of the most popular computer language today, because it is a structured, high-level, machine-independent language.



→ The root of all modern languages is ALGOL, introduced in 1960.

→ In 1967, Martin Richards developed a language called BCPL [Basic Combined Programming Language] for writing system software.

→ In 1970, Ken Thompson developed a language using many features of BCPL and named it as B. B was used to create early versions of UNIX operating system.

→ Finally in 1972, Dennis Ritchie developed C, which uses many concepts from ALGOL, BCPL, B and added the concept of data types and other powerful features.

→ C language was developed at Bell Laboratories in 1972.

→ The C language was developed along with the UNIX operating system. UNIX was coded entirely in C.

BASIC STRUCTURE OF C PROGRAMS:-

⇒ A C program can be viewed as a group of building blocks called functions.

⇒ A function is a set of statements designed to perform a specific task.

⇒ A C program may contain one or more sections as shown in the following figure.

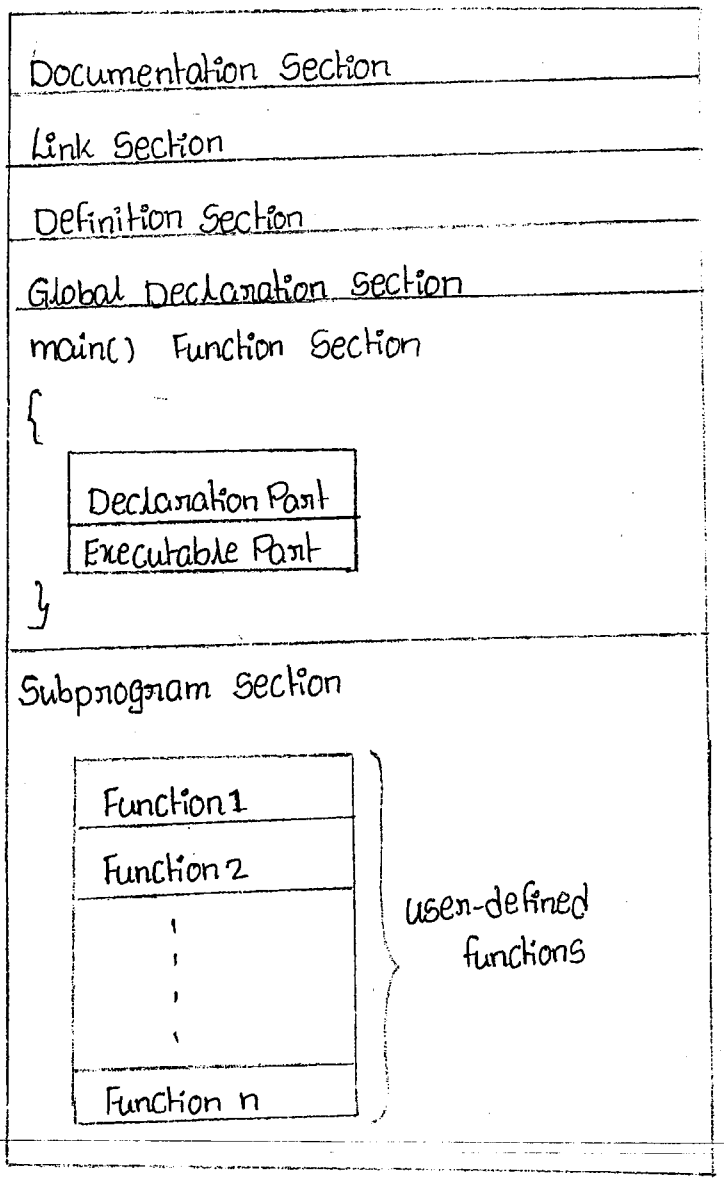


Figure: Structure of a C program

Documentation Section:-

⇒ The documentation section consists of a set of comment lines giving the name of the program, the author and other details.

⇒ The lines beginning with /* and ending with */ are known as comment lines.

⇒ These are used in a program to enhance its readability and understanding.

Example: /* Program to print Welcome message */

⇒ Comment lines are not executable statements, therefore anything between /* and */ is ignored by the compiler.

Link Section:-

⇒ The link section provides instructions to the compiler to link functions from C library.

⇒ C library consists of a set of functions, which are already predefined.

⇒ If we want to access the functions stored in the library, it is necessary to tell the compiler about the files to be accessed.

⇒ This is achieved by using the preprocessor directive #include.

Ex: #include<stdio.h>
#include<conio.h>

Definition Section:-

⇒ The definition section defines all symbolic constants.

⇒ #define directive is used to define symbolic constants.

Ex: #define PI 3.14
#define MAX 100

Global Declaration Section:-

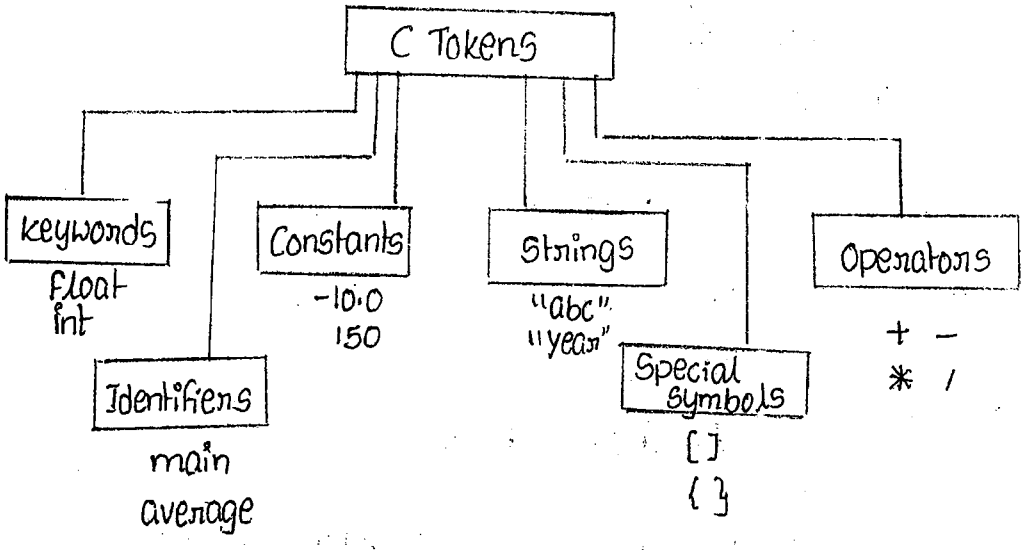
⇒ There are some variables that are used in more than one function. Such variables are called global variables and are declared in the global declaration section.
i.e. outside of all functions.

C TOKENS:-

⇒ In a passage of text, individual words and punctuation marks are called tokens.

⇒ In a C program the smallest individual units are known as C tokens.

⇒ C has six types of tokens.



⇒ C programs are written using these tokens and the syntax of the language.

KEYWORDS AND IDENTIFIERS:-

keywords:-

⇒ All keywords have fixed meanings and these meanings can't be changed.

⇒ C has 32 keywords.

Note: All keywords must be written in lowercase.

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	size of	volatile
do	if	static	while

Identifiers:-

⇒ Identifiers refer to the names of variables, functions and arrays.

⇒ These are user-defined names and consists of a sequence of letters and digits, with a letter as first character.

⇒ Both uppercase and lowercase letters are permitted, but lowercase letters are commonly used.

⇒ The underscore(_) character is also permitted in identifier.

⇒ Rules for identifiers are as follows:

1. The first character must be an alphabet (or) underscore
2. It must consists only letters, digits or underscore.
3. Only first 31 characters are significant in an identifier.
4. It can't be same as keyword.
5. It must not contain white space.

Ex: average, max_marks, letter_1, etc.

⇒ Invalid Identifier — Reason

1Letter	— Begins with a digit
double	— keyword
TWO*FOUR	— Character * not allowed
Joe's	— Character ' not allowed.

CONSTANTS:-

⇒ Constants in C refers to the fixed values that do not change during the execution of a program.

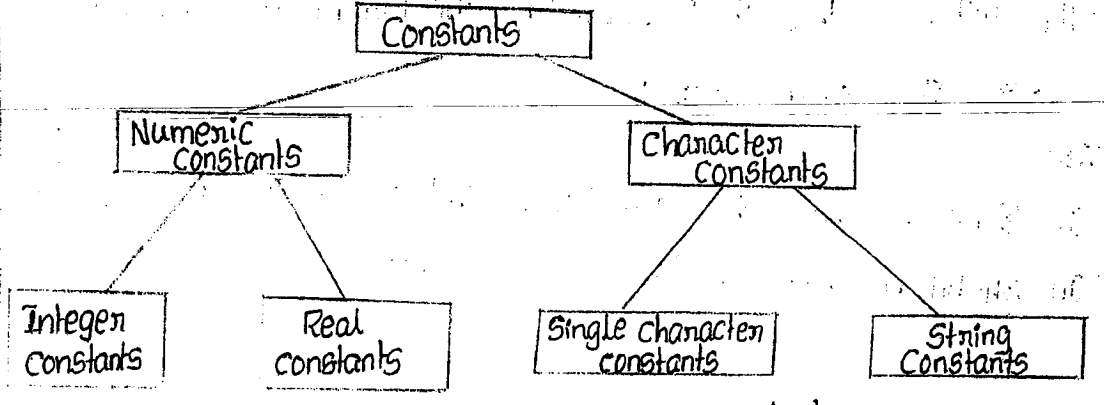


Figure: Basic types of C constants

Integer Constants:-

⇒ An integer constant refers to a sequence of digits.

⇒ There are 3 types of integers:

1. Decimal Integers
2. Octal Integers
3. Hexadecimal Integers.

1. Decimal Integers:-

⇒ Decimal integers consists of a set of digits, 0 through 9, preceded by an optional - (or) + sign.

Ex: 123 -321 0 654321 +78

Note: Embedded spaces, commas & special characters are not permitted between digits. [Ex: 20,000 $1000 are invalid].

2. Octal Integers:-

⇒ An octal integer constants consists any combination of digits from the set 0 through 7, with a leading 0.

Ex: 037 0 0435 0551

3. Hexadecimal Integers:-

⇒ A sequence of digits preceded by 0x (or) 0X are called as hexadecimal integers.

⇒ Hexadecimal integer constants consists of a set of digits 0 through 9 and alphabets A through F.

⇒ The letters A through F represents the numbers 10 through 15.

Ex: 0X2 0x9F 0X13CD

NOTE:-

⇒ In 16-bit machines the largest value that can be stored is 32767.

⇒ In 34-bit machines, it is 2,147,483,647.

Real Constants:-

⇒ Integer constants are inadequate to represent quantities that vary continuously, such as distances, heights, temparatures etc.

⇒ These quantities are represented by real numbers.

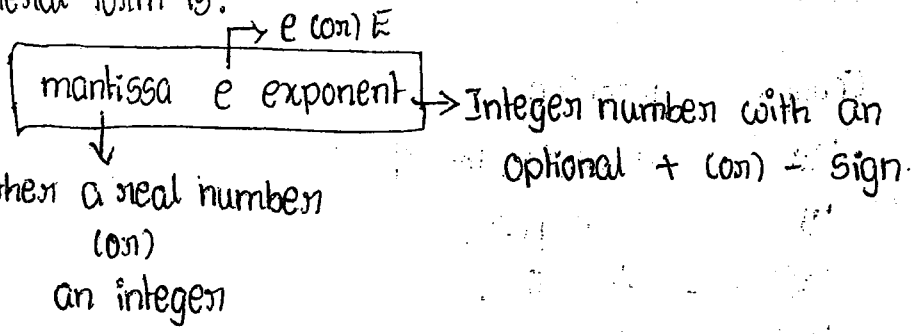
⇒ The numbers containing fractional part are called real constants.

Ex: 0.00083 -0.75 435.23 +247.0

⇒ These numbers are shown in decimal notation, having a whole number followed by a decimal point and the fractional part.

⇒ A real number may also be expressed in exponential [scientific] notation.

The general form is:



Example: The value 215.65 may be written as 2.1565e2 in exponential notation. e2 means multiple by 10^2.

⇒ Exponential notation is useful for representing numbers that are either very large or very small.

Ex: 750000000 may be written as 7.5E8 or 75E7.

-0.000000368 may be written as -3.68E-7

Single character constants:-

⇒ A single character constant contains a single character enclosed within a pair of single quote marks.

Ex: '5', 'x', 'P' etc.

⇒ Character constants have integer values known as ASCII values.

ASCII → American Standard Code for Information Interchanging.

Letters		ASCII values
0-9	⇔	48-57
A-Z	⇔	65-90
a-z	⇔	97-122

⇒ printf("%c", 'a'); → prints the letter a.

⇒ printf("%d", 'a'); → prints the number 97.

Backslash Character Constants:-

⇒ C supports some special backslash character constants that are used in output functions.

Constant	meaning
'\a'	audible alert (bell)
'\b'	back space
'\f'	form feed
'\n'	New line
'\r'	Carriage return
'\t'	Horizontal tab
'\v'	Vertical tab
'\''	Single quote
'\"'	Double quote
'\?'	Question mark
'\\'	Backslash
'\0'	Null

String Constants:-

⇒ A string constant is a sequence of characters enclosed in double quotes. The characters may be letters, numbers, special characters, and blank space.

Ex: "Hello!" "1987" "Well Done" "?--1" "X" "3+9" etc.

Note: 'x' ≠ "x"

VARIABLES:-

⇒ A variable is a data name that may be used to store a data value.

⇒ A variable value may change during program execution. i.e. A variable may take different values at different times during execution.

⇒ A variable name can be chosen by the programmer in a meaningful way so as to reflect its function.

Ex: Average, total, height, max_marks etc.

⇒ A variable name may consist letters, digits and underscore(_).

Rules:-

1. They must begin with a letter (or) Underscore(_).
2. ANSI standard recognizes a length of 31 characters. However, many compilers treat first eight characters are significant.
3. C compiler treats uppercase and lowercase letters as different. i.e. Total is not same as total (or) TOTAL
4. It shouldn't be a keyword.
5. White space is not allowed.

Special symbols:

Special symbols includes the following symbols

Symbol	Meaning		Symbol	Meaning
1. ~	Tilde	8.	"	Double quote
2. ,	Comma	9.	!	Exclamatory mark
3. .	period	10.	(	left parenthesis
4. ;	Semicolon	11.	)	Right "
5. :	colon	12.	[	left bracket
6. ?	Questionmark	13	]	Right "
7. '	single quote	14	{	left brace
		15	}	Right "

/, \, <, >, =, +, -, &, #, $, *, %, ^, _ ...

DATA TYPES:-

⇒ Data type refers to the type of data used in the program

⇒ C language is rich in its data types

⇒ The variety of data types available in C, allow the programmer to select the appropriate data type as per the need of the application.

⇒ Data types are mainly divided into 4 types:

1. Primary (or) Fundamental data types
2. Derived data types
3. User-defined data types
4. Empty data set

PRIMARY DATA TYPES:-

⇒ All C compilers support 4 fundamental data types:

1. Integer data type (int)
2. Character data type (char)
3. Floating point data type (float)
4. Double-precision floating point (double)

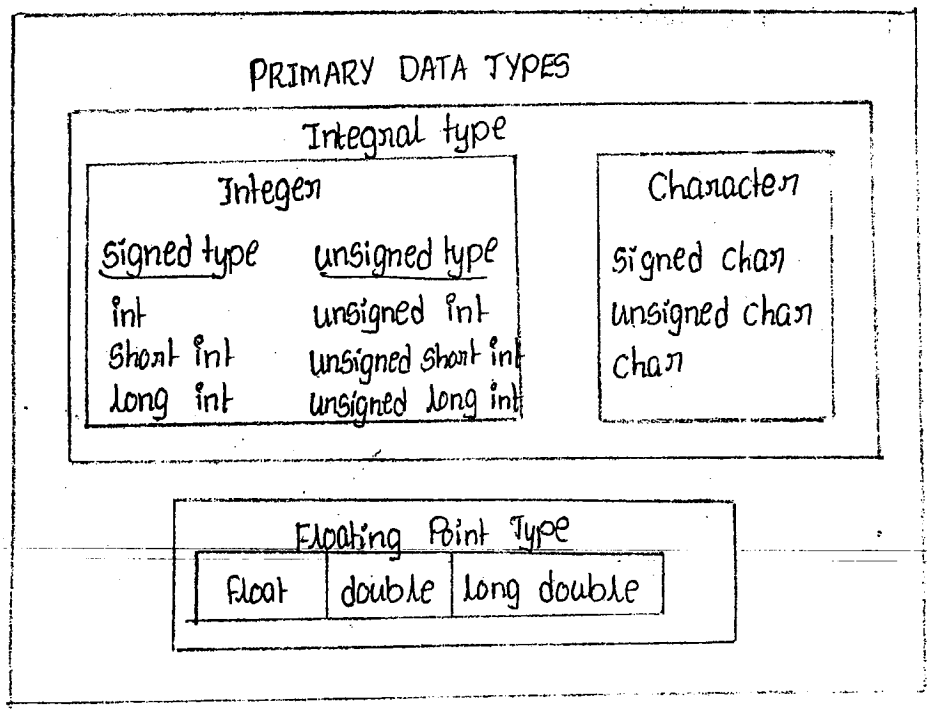


Figure: Primary data types in C.

INTEGER DATA TYPE:-

⇒ The int type is used to represent integers in C.

⇒ The keyword is int.

⇒ The size of int is 2 bytes.

⇒ The control string is %d

⇒ The range of int data type is: -32,768 to 32,767

⇒ In order to provide some control over the range of numbers, storage space, C has 3 classes of integers in both signed and unsigned forms.

1. short int → represents fairly small integer values
2. int
3. long int → large values, increases the range of values

TYPE	Size (bytes)	Range	Control String
int (or) signed int	2	-32,768 to 32,767	%d
unsigned int	2	0 to 65535	%u
short int (or) signed short int	1	-128 to 127	%d
unsigned short int	1	0 to 255	%u
long int (or) signed long int	4	-2,147,483,648 to 2,147,483,647	%ld
unsigned long int	4	0 to 4,294,967,295	%ud

⇒ short int represents fairly small integer values.

⇒ We declare long and unsigned integers to increase the range of values.

⇒ The use of qualifier 'signed' on integers is optional because by default declaration assumes a signed number.

Character TYPES:-

⇒ A single character can be defined as a character (char) type data.

⇒ Char occupies 1 byte of memory. i.e. characters are usually stored in 8 bits of internal storage.

⇒ Control string is: %c

⇒ Range is: -128 to 127

⇒ The qualifier signed (or) unsigned may be explicitly applied to char.

Type	Size	Range
Char (or) signed char	1 byte	-128 to 127
unsigned char	1 byte	0 to 255

FLOATING POINT TYPES:-

⇒ A floating point (or) real number has an integral part and a fractional part, that are seperated by a decimal point.

⇒ Floating point numbers are defined in C by:

1. float data type
2. double data type

float:

→ Real numbers are stored in 32 bits, with 6 digits precision.

⇒ The size is 4 bytes, control string is: %f

→ When accuracy provided by float is not enough, the type double can be used to define a real number.

double:

→ A double type uses 64 bits giving a precision of 14 digits.

→ To extend the precision further we may use long double which uses 80 bits.

TYPE	Size	Range
float	4 bytes	3.4E-38 to 3.4E+38
double	8 bytes	1.7E-308 to 1.7E+308
long double	10 bytes	3.4E-4932 to 1.1E+4932

DERIVED DATA TYPES:-

⇒ The derived data types are: Arrays
Structures
Unions
Pointers

USER-DEFINED DATA TYPES:

⇒ C supports 2 user-defined data types

1. Type definition
2. Enumerated data type

1. Type definition [typedef]:-

⇒ Type definition allows users to define an identifier that would represent an existing data type.

⇒ The user-defined data type identifier can later be used to declare variables.

Syntax:- typedef type identifier → new name given to the data type
↓ keyword ↳ Existing data type

Examples: typedef int units;
typedef float marks;

→ Here unit symbolises int and marks symbolises float.
→ They can be used to declare the variables as follows:

units a, b, c;
marks sub1, sub2, sub3;

⇒ The main advantage of typedef is that we can create meaningful datatype names for increasing the readability of the program.

2. Enumerated data type [enum]:-

⇒ The enumerated data type can be defined as follows:

enum identifier {value1, value2, ------value n};
→ Enumeration constants

⇒ Here 'identifier' is a user-defined enumerated data type, which can be used to declare the variables, that can have one of the values enclosed within braces.

⇒ We can declare the variables as follows:

enum identifier V1, V2, ---Vn;

→ V1, V2, ----Vn can only have one of the values enclosed in braces.

Example: enum day {monday, Tuesday, ----, sunday};

enum day week-st, week-end;

week-st = monday;

week-end = sunday;

⇒ By default the compiler automatically assigns integer digits beginning with 0 to all enumeration constants.

i.e. monday is assigned with 0,

Tuesday is assigned with 1 and so on.

⇒ We can override the automatic assignments by assigning values explicitly to the enumeration constants.

Ex: enum day {monday=1, Tuesday, ----sunday};

EMPTY DATA SET:-

⇒ The void type has no values.

⇒ This is usually used to specify the type of functions.

⇒ The type of a function is said to be void when it doesn't return any value.

Ex: void main()

{

}

DECLARATION OF VARIABLES:-

Variable:- A variable is a data name which is used to store data values.

⇒ The declaration of variables must be done before they are used in the program.

⇒ Variable declaration does two things:

1. It tells the Compiler what the variable name is.
2. It specifies what type of data the variable will hold.

Primary Type Declaration:-

⇒ A variable can be used to store a value of any data type.

⇒ The syntax for declaring variables is as follows:

```
data-type variable1, variable2, ------variableN;
```

→ Variables are seperated by commas.

→ A declaration statement must end with a semicolon.

Example:
```
int count;
double ratio;
int num1, num2, total;
char a,b,C;
```

⇒ The following program illustrates declaration of variables:

```
main
{
  /*------Declaration ----*/
  float x,y;
  int code;
  short int count;
  long int amount;
  unsigned int n;
  char C;
  double ratio;
  /* Computation */
  ----------
  ----------
  ----------
}
```

NOTE:
→ Declaration of variables is usually done immediately after the opening brace of the program.

User-defined type declaration:-

⇒ C supports 2 user-defined data types:

1. Type Definition
2. Enumerated Data type.

Type Definition:-

⇒ The general form is:

```
typedef type identifier;
```

→ The user-defined identifier can be used to declare the variables as follows:

```
identifier v1, v2, v3, --- vn;
```

Example:
```
typedef int units;
typedef float marks;
```

→ units, marks can be used to declare the variables as follows:

```
units batch1, batch2;
marks sub1, sub2;
```

→ Here batch1, batch2 are declared as int and sub1, sub2 are declared as float variables.

Enumerated Data type:-

→ The syntax is:

```
enum identifier {value1, value2, ---- value n};
enum identifier v1, v2, v3, --- vn;
```

Example:
```
enum day{monday, Tuesday, ---- sunday};
enum day week_st, week_end;
week_st = monday;
week_end = sunday;
```

MANAGING INPUT AND OUTPUT OPERATIONS:

⇒ There exist several functions for input and output operations in C.
⇒ These functions are collectively known as the standard I/O library.

1. Reading a character:

⇒ Reading a single character can be done by using the function getchar.
⇒ The getchar takes the following form:

```
variable_name = getchar();
```

→ Here, variable_name is a valid C name that has been declared as char type.
→ When this statement is encountered, the computer waits until a key is pressed and then assigns that character as a value to getchar function.

Example: char section;
section = getchar();

Example program:

```
/* The following example shows the use of getchar function */
#include<stdio.h>
main()
{
    char section;
    printf("Enter your section name:");
    section = getchar(); /* Reading a character */
    printf("your section is: %c", section);
    getch();
}
```

Character Test Functions:-

⇒ C supports several character functions, which are defined in the file ctype.h.

⇒ Each function takes single character as an argument.

Function	Test
isalnum(c)	— Is c an alphanumeric character?
isalpha(c)	— Is c an alphabetic character?
isdigit(c)	— Is c a digit?
islower(c)	— Is c lower case letter?
isupper(c)	— Is c an upper case letter?
isprint(c)	— Is c printable character?
ispunct(c)	— Is c a punctuation mark?
isspace(c)	— Is c an upper white space character.

Note:
We should include ctype.h in a program before using these functions

⇒ For example, isalpha(c) returns TRUE (non-zero value) if 'c' is an alphabet. Otherwise it returns FALSE (Zero).

Example program:-

→ Write a program that request the user to enter a character and displays a message telling that whether the entered character is an alphabet, or digit or any other special symbols.

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
void main()
{
    char ch;
    clrscr();
    printf("Enter a character:");
    ch=getchar();
```

```
    if (isalpha(ch))
        printf("The entered character is an alphabet");
    else if (isdigit(ch))
        printf("The entered character is a digit");
    else
        printf("The entered character is a special symbol");
    getch();
}
```

Writing a character:-

⇒ putchar function is used for writing characters one at a time to the screen. i.e. putchar function displays one character at a time.

⇒ Syntax is:

```
putchar(variable-name);
```

→ Here variable-name is a char variable containing a character. This function displays the character contained in variable-name at the screen.

Example:
```
char answer;
answer = 'y';
putchar(answer); ⇒ Display the character Y on the screen.
```

Example Program:-

```
/* Write a program to read and display a character using getchar & putchar functions */

#include<stdio.h>
void main()
{
    char c;
    printf("Enter a character:");
    c=getchar(); /* Reading a character */
    printf("The entered character is:");
    putchar(c); /* Writing a character */
    getch();
}
```

Formatted Input:

⇒ Formatted input refers to the input data that have been arranged in a particular format.

Example: 15.75 123 John.

⇒ The scanf function is used to read formatted input.

⇒ The general form of scanf is:

```
scanf("Control String", &variable1, &variable2, ----);
```

→ The control string contains the format of data being received.

→ variable1, variable2, ---- are the variable names.

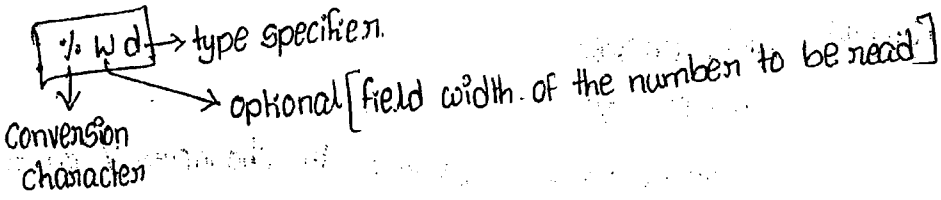
→ The & symbol before each variable name is an operator that specifies the variable address.

Note:

* We must always use &. Otherwise unexpected results may occur.

⇒ Control string contains:

→ Field (or format) specifications, conversion character %, type specifier and an optional number specifying the field width.



1. Inputting Integer numbers:-

⇒ The field specification for reading an integer number is %wd.

Example: scanf("%d", &number); ⇒ Reads an integer value from the keyboard.

Example 2:-

scanf("%2d", &num);

→ Suppose the following data are entered at the console:

31426

Now, the variable num will be assigned to 31 because of %2d.

Example 3:-

⇒ scanf("%2d %5d", &a, &b);

Suppose the input data are as follows:

31426 50

Now, the variable a will be assigned to 31 (because of %2d) and b will be assigned to 426 (Unread part of 31426). The value 50 that is unread will be assigned to the first variable in the next scanf call.

Example 4:-

⇒ What happens if we enter a floating point number instead of an integer?

Ans) The fractional part may be stripped away and also the scanf may skip reading further input.

Ex: scanf("%d", &a);

Suppose if we enter input data 34.56, the a value will be assigned to 34 [i.e. the scanf skips the fractional part].

Ex: scanf("%d %d", &a, &b);

→ suppose if we enter input data as follows:

23.567 6

→ The variable a will be assigned to 23. The scanf skips reading further input i.e. it doesn't read b value.

Example 5:-

⇒ An input field may be skipped by specifying * in the place of field width.

scanf("%d %*d %d", &a, &b);

→ Suppose we enter the following data as input:

123 456 789

In this case a=123, b=789. The value 456 will be skipped (because of *).

2. Inputting Real Numbers:-

⇒ scanf reads real numbers using the specification %f.

Ex: scanf("%f %f %f", &x, &y, &z);

⇒ If we want to read double type numbers, then specification should be %lf

3. Inputting character Strings:-

⇒ scanf reads a single character using the specification %c

⇒ scanf can also read strings containing more than one character by using %s specification

4. Inputting Mixed data types:-

⇒ It is possible to use one scanf statement to input a data line containing mixed mode data.

Ex: scanf("%d %c %f %s", &count, &code, &ratio, &name);

Important Note:-

⇒ When a scanf function completes reading its list, it returns the value of number of items that are successfully read.

Ex: a = scanf("%d %c %f", &a, &b, &c);

→ Suppose the following data is entered at the console:

20 A 10.15

The a value will be assigned to 3. Since, the scanf reads 3 input values, and returns 3.

→ Considered the following data is entered:

20 10.15 A

scanf returns 1 and a value will be assigned to 1. Because the function encounter a floating point, when it was expecting a character. Therefore scanf terminates its scan after reading the first value.

FORMATTED OUTPUT:-

⇒ Output can be displayed by using printf function

⇒ The general form of printf statement is:

```
printf("Control String", arg1, arg2, ....., argn);
```

→ Where the control string may contain:

1. Characters that will be printed on the screen as they appear.

   Ex: printf("Welcome");

   The above printf function displays the message Welcome on the screen.

2. Format specifications that define the output format for display of each item.

   Ex: printf("%d", a);
   ↳ Format specifier

3. Escape sequence characters such as \n, \t and so on.

   Ex: printf("\n SITAMS \n CHITTOOR");

→ The arguments arg1, arg2, ... argn are the variables whose values are printed according to the format specifications.

Examples:-

1. printf("Programming in C"); → Displays a message Programming in C
2. printf(" "); → Displays one space
3. printf("\n"); → New line
4. printf("%d", x); → Display the value of x
5. printf("a=%f \n b=%f", a, b); → Display the values of a, b
6. printf("\n \n");
7. printf("sum = %d", 2+3); → Display sum = 5

Output of Integer Numbers:-

⇒ The format specification for printing an integer number is %d

Ex: int a=10;
printf("%d", a); → Display 10
printf("a=%d", a); → Display a=10

Ex: int a,b;
a=10;
b=20;
printf("a+b=%d", a+b); → Display a+b=30
printf("Sum=%d", a+b); → Display Sum=30

Outputting Real Numbers:-

⇒ The format specifier for printing a real number is %f

Ex: float a=5.7;
printf("%f", a); → Display 5.7

Outputting a Single Character:-

⇒ A single character can be displayed by using the %c format specifier.

Ex: char a='x';
printf("%c", a); → Display x

Mixed Data Output:-

⇒ It is possible to mix data types in one printf statement.

Ex: int a=100;
char b='z';
float c=10.57;
printf("%d IE %C IE %f", a,b,c); → Display 100 z 10.57

OPERATORS & EXPRESSIONS

⇒ In order to perform different kinds of operations, C uses different operators.

⇒ An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations.

⇒ An operand is a data item on which operators perform the operations

Ex: 5 + 3 → operands
↳ operator

⇒ Operators are used in program to manipulate data and variables.

⇒ C operators are classified into:
1. Arithmetic operators (+, -, *, /, %)
2. Relational operators (<, >, <=, >=, ==, !=)
3. Logical operators (&&, ||, !)
4. Assignment operators (=)
5. Increment and decrement (++, --) operators
6. Conditional operators (? :)
7. Bitwise operators (&, |, ^)
8. Special operators (,, sizeof, &)

Expression:

An expression is a sequence of operands and operators that reduces to a single value.

Ex: x = 5*4 + 8/2;

ARITHMETIC OPERATORS :-

⇒ C provides all the basic arithmetic operators.

Operator	Meaning	Example
+	Addition (or) Unary plus	2+2 = 4
−	Subtraction (or) Unary minus	5-3 = 2, -3, -4
*	Multiplication	5*2 = 10
/	Division	10/2 = 5
%	Modulo division	11%3 = 2 (Remainder)

⇒ The above operators can operate on any built-in data type allowed in C.

⇒ The unary minus operator, multiplies its sigle operand by -1.

⇒ Integer division truncates any fractional part.

Ex: 14/4 = 3 (decimal part truncated).

⇒ Modulo division produces the remainder of an integer division.

Ex: 14%4 = 2 (Remainder of division)

```
   3
4) 14
   12
   ---
   (2) → Remainder
```

> **Note:** The modulo division operator % can't be operated on floating point data.

Integer Arithmetic :-

⇒ An arithmetic operation involving only integer operands is called integer arithmetic and the expression is called integer expression.

⇒ Integer arithmetic always yields an integer value.

Ex: a = 14, b = 4

a-b = 10
a+b = 18
a*b = 56
a/b = 3
a%b = 2

⇒ During modulo division, the sign of the result is always the sign of the first operand.

Ex: -14 % 3 = -2
-14 % -3 = -2
14 % -3 = 2

Real Arithmetic:-

⇒ An arithmetic operation involving only real operands is called real arithmetic.

⇒ The result is always floating point value.

Ex: x = 6.0/7.0 = 0.857143
z = -2.0/3.0 = -0.666667

Mixed-mode Arithmetic:-

⇒ When one of the operands is real and other is integer, then it is called mixed-mode arithmetic.

⇒ If either operand is of the real type, then the result is always a real number.

Ex: 15/10.0 = 1.5

RELATIONAL OPERATORS:-

⇒ Relational operators are used to compare two quantities, and depending on their relation, we take certain decisions

⇒ An expression containing relational operator is known as relational expression.

Ex: a<b (or) 1<20

⇒ If the relation is true, it returns 1. Otherwise 0 for false relation.

⇒ C supports six relational operators.

Operator	meaning	Example	Return value
>	Greater than	5>4	1
>=	Greater than or equal to	11>=5	1
<	Less than	10<9	0
<=	Less than or equal to	10<=10	1
==	equal to	2==3	0
!=	Not equal to	3!=3	0

⇒ Relational expressions are used in decision statements such as if, while, do-while, for.

LOGICAL OPERATORS:-

⇒ The logical operators are used when we want to test more than one condition and make decisions.

⇒ C supports 3 logical operators.

Operator	meaning	Example	Return value
&&	– Logical AND	5>3 && 5<10	0 or 1
||	– Logical OR	2==3 || 4==4	1
!	– Logical NOT	!(2==3)	1

⇒ An expression which combines two or more relational expressions is called logical expression or a compound expression

Ex: 5>3 && 5<10

⇒ The logical expression yields 1 or 0 according to the truth table.

⇒ AND Truth table:

Condition1	Condition2	Return value
1	1	1
1	0	0
0	1	0
0	0	0

⇒ The logical AND (&&) operator provides true only when both the expressions are true, otherwise 0.

⇒ Logical OR Truth table:-

Condition1	Condition2	Return value
1	1	1
1	0	1
0	1	1
1	1	0

⇒ The logical OR (||) provides true when one of the expression is true, Otherwise 0

⇒ The logical NOT (!) provides 0 if the condition is true, otherwise 1.

Examples	Return value
5>3 && 5<10	1
8>5 || 8<2	1
!(8==8)	0

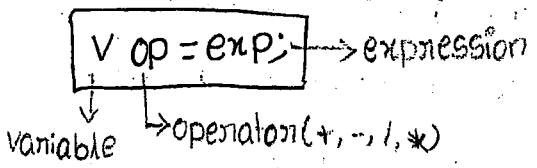
ASSIGNMENT OPERATORS:-

⇒ Assignment operators are used to assign the value or result of an expression to a variable.

Syntax: `Variable_name = constant (OR) expression`

Example: a=10 , c=20
b=10*5

⇒ In addition, C has set of 'shorthand' operators of the form



→ The operator op= is known as shorthand assignment operator.

⇒ The assignment statement V op=exp is equivalent to:

`V = V op (exp);`

⇒ For example x+=3 is equivalent to x=x+3

x+=y+1 is equivalent to x=x+(y+1)

⇒ Some of the commonly used shorthand assignment operators are:

Statement with simple assignment operator	Statement with shorthand operator
a=a+1	a+=1
a=a-1	a-=1
a=a*(n+1)	a*=(n+1)
a=a/(n+1)	a/=n+1
a=a%b	a%=b

⇒The use of shorthand assignment operators has 3 advantages:

1. What appears on the left-hand side need not be repeated
2. The statement is easier to read.
3. The statement is more efficient.

INCREMENT AND DECREMENT OPERATORS:-

⇒ C has two very useful operators: increment and decrement operators

operator	meaning
++	Increment
--	Decrement

⇒The operator ++ adds 1 to the operand.

i.e. x++ is equivalent to x=x+1

⇒ The operator -- subtracts 1 from its operand.

i.e. x-- is equivalent to x=x-1.

⇒Both these operators may either follow or precede the operand.

i.e. x=x+1 can be represented as x++ (or) ++x

Post Increment/Decrement:-

⇒ If ++ or -- are used as a suffix to the variables name then post increased/decreased operations take place.

Ex: m=5;
y=m++;

→ After executing these statements y would be 5 and m=6.

→ i.e. The postfix operator first assigns the value to the variable on lefthand side and then increments the operand.

Ex: x=20;
y=x--;

Here after executing y=20, x=19

Pre Increment/Decrement :-

⇒ If "++" or "--" are used as a prefix to the variable name then pre increment/decrement operations take place.

Ex: m=5;
y=++m;

→ After executing these statements, y=6, m=6. i.e. A prefix operator first adds 1 to the operand and then the result is assigned to the variable on left.

Ex: m=5;
y=--m;

→ In this case the value of m and y would be 4.

⇒ Note: We use increment and decrement statements in for and while loops extensively.

Examples:

x=10, y=20	x=10, y=20
z=x*y++ ⇒ z=200	z=x*++y ⇒ z=210
a=x*y ⇒ a=210	a=x*y ⇒ a=210

NOTE:-

⇒ When postfix ++ (or --) is used with a variable in an expression, the expression is evaluated using the original value of the variable and then the variable is incremented (or decremented) by one.

⇒ When prefix ++ (or --) is used in an expression, the variable is incremented (or decremented) first and then the expression is evaluated using the new value of the variable.

CONDITIONAL OPERATOR:-

⇒ The operator '?:' is known as conditional operator.

⇒ The conditional operator has the following form:

```
condition ? statement 1 : statement 2;
```

Working:-

1. First the condition is evaluated.
2. If the condition is true, then statement1 will be executed.
3. If the condition is false, then statement 2 will be executed.

⇒ The operator '?:' is also known as ternary operator. Because it takes 3 operands.

Ex:
```
a=10;
b=15;
x=(a>b)?a:b;
```

Here x will be assigned to 15. Because first it checks the condition a>b (i.e. 10>15), condition is false. So statement2 will be executed. So b is assigned to x.

Ex:
```
a=25;
b=10;
x=(a>b)?a:b;
```

Here x will be assigned to 25 (∵ a>b is true).

```
(a>b)? printf("%d", a) : printf("%d", b);
printf("%d", a>b ? a : b);
```

BITWISE OPERATORS:-

⇒C supports bitwise operators for manipulation of data at bit level.

⇒These operators are used for testing the bits, on shifting them right on left.

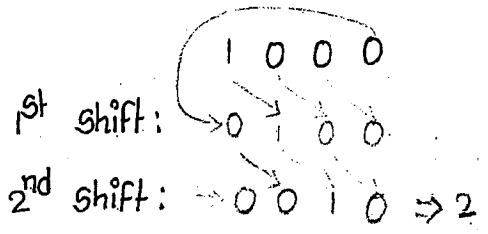
⇒These operators can operate only on integer data.

Operator	Meaning
>>	Right shift
<<	Left shift
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	One's complement.

Right Shift (>>):

Example: Shift the number 8 by 2 bits right.

Sol: 8 binary equivalent 1000



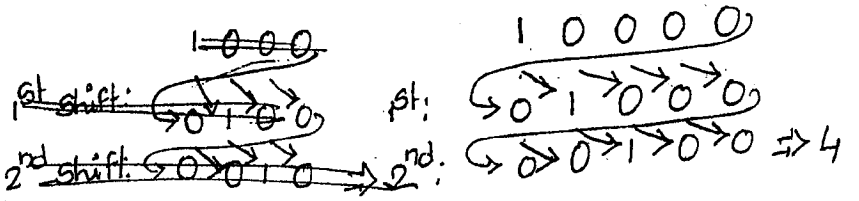
⇒ Shifting two bits right means, the inputted number is to be divided by 2^s, where s is no.of shifts.

i.e. $y = n/2^s$

where n = Number
s = Number of positions to be shifted.

Example2: Shift 16 by 2 bits right

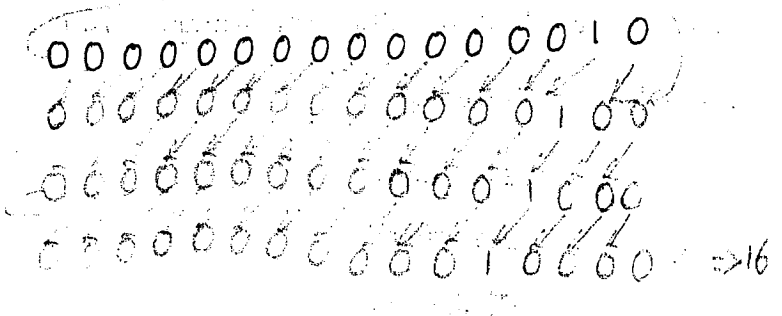
16 binary equivalent 10000



Left Shift (<<):-

Example: Shift the number 2 by 3 bits left

2 binary equivalent



⇒ Shifting 3 bits left means, multiply the number by 2^3.

i.e. $y = n * 2^3$.

Bitwise AND:-

Example: consider a=8 and b=4

c = a & b

a=8 : 1 0 0 0

b=4 : 0 1 0 0

c=a&b : 0 0 0 0

∴ c=0

a=8, b=8

c = a & b

a : 1 0 0 0

b : 1 0 0 0

c : 1 0 0 0

∴ c=8.

Bitwise OR:-

Example: Consider a=8 and b=4

c = a | b

a : 1 0 0 0

b : 0 1 0 0

c=a|b : 1 1 0 0

∴ c=12

Bitwise Exclusive OR:-

Consider a=8 and b=2

```
c = a^b;

a:   1 0 0 0
b:   0 0 1 0
c=a^b: 1 0 1 0

∴ c = 10
```

TRUTH Table of XOR

X	Y	Z
1	1	0
0	1	1
1	0	1
0	0	0

One's Complement:-

Input	OUTPUT
1	0
0	1

Example: consider a=8

```
c = ~a

a:   1 0 0 0
~a:  0 1 1 1  ⇒ 7
```

SPECIAL OPERATORS:-

⇒ C supports the following special operators:

1. Comma Operator
2. Sizeof operator
3. & operator

1. Comma operator:-

⇒ The comma operator is used to seperate two or more expressions.

⇒ A comma-linked list of expressions are evaluated left to right and the value of the right-most expression is the value of the combined expression.

Ex: a=2,4,5 ⇒ a=5

value = (x=10, y=5, x+y) ⇒ value = 15

(x=x+y, y+1)

⇒ Some applications of comma operator:

In for loops: for(n=1, m=10; n<=m; n++, m++)

Exchanging values: t=x, x=y, y=t;

2. sizeof operator:-

⇒ The sizeof operator gives the number of bytes occupied by a operand.

→ The operand may be a variable, a constant or a data type qualifier.

Ex: int sum;
float avg;

sizeof(sum) → Returns 2
sizeof(avg) → Returns 4

Ex: m = sizeof(sum);
n = sizeof(double);
k = sizeof(235L);

3. & operator:-

⇒ The & operator prints the address of the variable in the memory.

Ex:
```
#include<stdio.h>
void main()
{
    int a;
    a=20;
    printf("%d", a); /* Displays value of a */
    printf("%u", &a); /* Displays address of variable a */
}
```

⇒ Write a program to display number of bytes occupied by different data types

```
#include<stdio.h>
void main()
{
    printf("Number of bytes occupied by int is: %d", sizeof(int));
    printf("Number of bytes occupied by char is: %d", sizeof(char));
    printf("Number of bytes occupied by float is: %d", sizeof(float));
    printf("Number of bytes occupied by double is: %d", sizeof(double));
}
```

OPERATOR PRECEDENCE AND ASSOCIATIVITY:-

⇒ Every operator has a precedence value.

⇒ Precedence of operators refers to the order in which they are operated in a program (or) expression.

⇒ Associativity specifies the order in which the operators are evaluated when they have same precedence.

→ Associativity is of two types:

1. Left-to-Right Associativity: Evaluates an expression starting from left and moving towards right.

2. Right-to-Left Associativity: Evaluates from right to left.

⇒ The precedence and associativity of various operators in C are as shown in the following table:

Operator	Description	Associativity	Precedence
( )	Function call	Left to Right	1
[ ]	Array element Reference		
+	Unary plus	Right to Left	2
−	Unary minus		
++	Increment		
−−	Decrement		
!	Logical negation		
~	One's complement		
*	Pointer reference		
&	Address		
Sizeof	Size of an object		
(type)	Type cast (conversion)		
*	Multiplication	Left to Right	3
/	Division		
%	Modulo division		
+	Addition	Left to Right	4
−	Subtraction		
<<	Left Shift	Left to Right	5
>>	Right Shift		

EXPRESSIONS:-

An expression is a sequence of operands and operators that reduces to a single value.

Ex: x = 5*4+8/2;

c = a+b;

⇒ Based on the position and number of operands, expressions are classified into 6 types.

1. PRIMARY EXPRESSION:-

⇒ An ordinary mathematical expression is called infix expression.

⇒ In infix, operators are placed between the operands in an expression.

Ex: (A/B)+(C-D)*E

2. POSTFIX EXPRESSION:-

⇒ In postfix expressions the operators are placed after the operands

Ex: Function call → Clrscr()

a++

a--

3. PREFIX EXPRESSION:-

⇒ In prefix expressions, the operators are placed before the operands.

Ex: ++a

--a

4. UNARY EXPRESSION:-

⇒ An unary expression is like a prefix expression consists of one operand and one operator and the operand comes after the operator.

Ex: ++a, -b, +d

5. BINARY EXPRESSION:-

⇒ It is a combination of 2 operands and an operator.

Ex: a+b, c*d

6. TERNARY EXPRESSION:-

⇒ It is an expression which consists of a ternary operator pair "?:"

Ex: condition ? exp1 : exp2;

TYPE CONVERSIONS IN EXPRESSIONS:-

⇒ C permits mixing of constants and variables of different types in an expression.

⇒ C performs conversions between different data types. i.e. it converts one data type into another data type.

⇒ There are two types: 1. Implicit-type Conversion
2. Explicit-type conversion.

Implicit type Conversion:-

→ Automatic type conversion by the compiler.

→ C automatically converts any intermediate values to the proper type. This automatic conversion is known as implicit type conversion.

RULES:-

→ During the expression evaluation, if the operands are of different data types, the 'lower' type is automatically converted to the 'higher' type before the operation proceeds.

Ex: 3+1.5 = 4.5

a+10 = 107 ⇒ Here a is char. It is converted into integer. Ascci equivalent of a is 97. so 97+10=107.

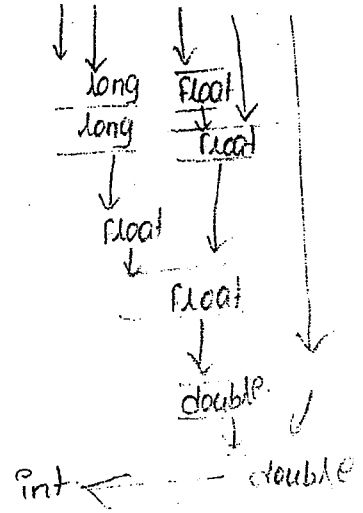
→ In general

char < int < unsigned int < long int < unsigned long int < float < double < long double

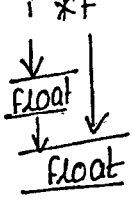
⇒ For assignment statements, C converts the value at the right side of the assignment statement to the type of the variable on the left side.

Ex:
```
int a;
Char ch;
Float f;
Ch=a; /* Here a is integer that is converted to char */
a=f; → float is converted into int
f=ch; → Char is converted into float.
f=a; → int is converted into float.
```

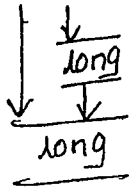
Ex:
```
int i,x;
float f;
double d;
long int l;
x = l/i + i*f - d
```



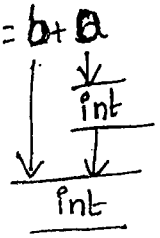
Ex:
```
int i;
float f;
i*f
```
Here i is integer, f is float. During compilation lower type is converted into higher type. i.e. int is converted into float.



Ex:
```
int i;
long int x,l;
x=l/i;
```



Ex:
```
Char a='A';
int b=10;
c=b+a
```



c = 10+65
= 75

Explicit Type Conversion:-

→ The type of an expression can also be converted explicitly by type casting.

Syntax: `(type-name) expression` → Here expression is converted into the data type specified in paranthesis.

Ex: `x = (int) 7.5` → 7.5 is converted to integer by truncation.

`a = (int) 21.3 / (int) 4.5` → Evaluated as 21/4

`b = (double) sum/n` → Division is done in floating point mode.

`y = (int) (a+b)` → The result of a+b is converted to integer.

`z = (int) a+b` → a is converted to integer and then added to b.

DECISION MAKING AND BRANCHING / SELECTION

⇒ Flow of control:- Flow of control shows the order in which the program is executing.

⇒ Normally a C program is a set of statements executed in sequence. i.e. All the statements are executed sequentially in the order in which they appear.

⇒ But in pratical, there are many situations where we have to change the order of execution of statements based on certain conditions.

⇒ This involves decision-making to see whether a particular condition is satisfied or not. Based on the condition the order of execution of statements will be changed.

⇒ C language supports the following decision-making statements.

1. if statement
2. switch statement
3. goto statement.

→ These statements control the flow of execution, so they are also called as control statements.

Decision making with if statement:-

⇒ The if statement is a powerful decision-making statement and is used to control the flow of execution of statements.

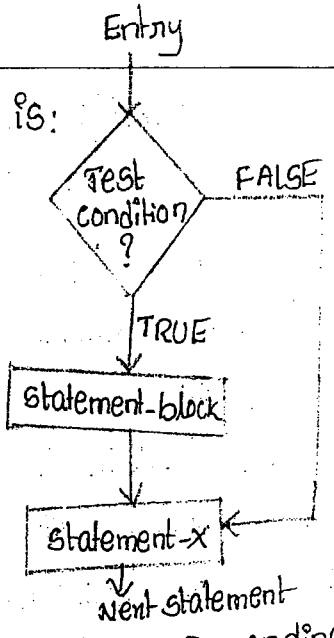
⇒ The if statement has the following forms:

1. Simple if statement
2. If-else statement
3. Nested if...else statement
4. else if ladder.

Simple if statement:-

⇒ The general form of simple if statement is:

```
if( test-condition)
{
    statement-block;
}
 statement-x;
```



WORKING:-

⇒ First the computer will evaluate the test condition. Depending on whether the condition is true or false, it transfers the control to a particular point.

⇒ The statement-block may be a single statement or a group of statements.

⇒ If the condition is true, the statement-block will be executed. If the condition is false, it will skip the statement-block and control will jump to statement-x.

Example Programs:-

① Write a program to check the equivalence of two numbers using simple if statement.

```
#include<stdio.h>
void main()
{
    int a,b;
    clrscr();
    printf("Enter two numbers:");
    scanf("%d %d",&a,&b);
    if(a==b)
    {
        printf("Two numbers are equal");
    }
    getch();
}
```

Explanation:-

⇒ In this program two numbers a & b are entered.

⇒ Their equivalence is checked by using the condition if(a==b)

⇒ If condition is true it will display the message "both numbers are equal"

⇒ If the condition is false, it will not do any thing.

(2) Write a program to perform the ratio of a/b and print the result when b is not equal to zero.

```
#include<stdio.h>
Void main()
{
   int a,b;
   float ratio;
   printf("Enter a,b values:");
   scanf("%d %d",&a,&b);
   if(b!=0)
   {
      ratio = a/b;
      printf("Ratio = %f", ration);
   }
}
```

Sample output:-

Enter a,b values 3 2

Ratio = 1.5

Sample output:-

Enter a,b values 5 0

/* Nothing will be displayed */

Explanation:-

⇒ In the above program, the values of a,b are read from keyboard.

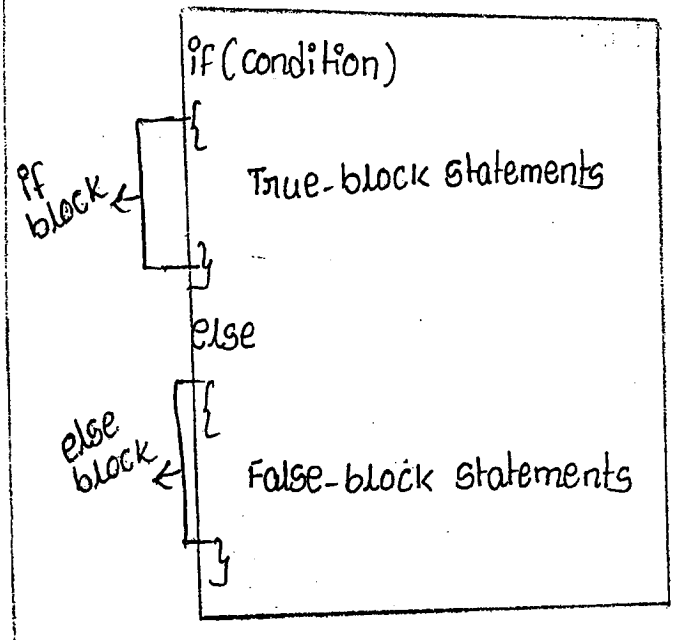
⇒ It will check the condition whether the value of b is equal to 0 or not.

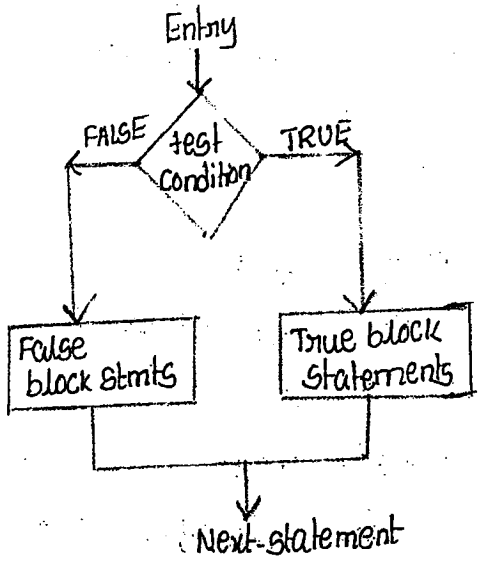
⇒ If b!=0 then it will calculate the ratio and print the result.

⇒ If b value is 0, then the program will not do anything.

The if-else statement:-

⇒ The if-else statement is an extension of simple if statement.
⇒ The simple if statement does nothing when the condition is false.
⇒ The if-else statements takes care of both true as well as false conditions.
⇒ It has two blocks.
→ One block is for "if" and is executed when the condition is true.
→ Another block is executed when the condition is false.
⇒ The general form of if-else statement is as follows:





WORKING:-
→ First it will check the condition.
→ If the condition is true, if block statements (i.e. true-block) statements will be executed.
→ If the condition is false, then else block statements (i.e. false-block) statements will be executed.
→ In either case, either true or false block statements will be executed. But not both.

Important Note:-
⇒ The 'else' statement can't be used without 'if' statement.
⇒ No multiple else statements are allowed with one if statement.

Example program:-

① Write a program to find biggest of 3 numbers using nested if-else.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a,b,c;
    clrscr();
    printf("Enter a,b,c values:");
    scanf("%d %d %d",&a,&b,&c);
    if(a>b)
    {
        if(a>c)
        {
          printf("%d is biggest number",a);
        }
        else
        {
          printf("%d is biggest number",c);
        }
    }
    else
    {
        if(c>b)
        {
          printf("%d is biggest number",c);
        }
        else
        {
          printf("%d is the biggest number",b);
        }
    }
    getch();
}
```

The else-if ladder:-

⇒ When multiple decisions are involved, then else-if ladder is used.

⇒ The general syntax of else-if ladder is as follows:-

```
if(condition-1)
{
   statement-1;
}
else if(condition-2)
{
   statement-2;
}
else if(condition-3)
{
   statement-3;
}
.
.
.
else
{
   default-statement;
}
statement-x;
```

⇒ The conditions are checked from top to bottom.

⇒ If condition 1 is true then statement-1 will be executed.

⇒ If condition 1 is false, then the control is transferred to check the next condition.

⇒ If anyone of the condition is true, then the statements associated with it will be executed.

⇒ This process continues till the last.

WORKING:-

⇒ The conditions are evaluated from top to bottom.

⇒ As soon as a true condition is found, the statements associated with it is executed and the control is transferred to statement-x.

⇒ When all the n conditions becomes false, then the final else containing default-statement will be executed.

Example program:-

⇒Write a program to calculate commission based on the sales amount.

Sales amount	Commission
< 5000	NIL
>=5000 && <10000	2% of sales amount
>=10000	5% of sales amount

```
#include<stdio.h>
#include<conio.h>
void main()
{
    float sa, cm;
    clrscr();
    printf("Enter sales amount:");
    scanf("%f", &sa);
    if(sa<5000)
    {
      printf("Commission is NIL");
    }
    else if(sa>=5000 && sa<10000)
    {
        cm = 0.02*sa;
        printf("Commission is: %f", cm);
    }
    else
    {
        cm=0.05*sa;
        printf("Commission is: %f", cm);
    }
    getch();
}
```

The Switch Statement:-

⇒ The switch statement is a multi-way branch statement.

⇒ If there is a possibility to make a choice from a number of options, then switch statement is used.

⇒ The switch statement requires only one argument of any data type.

⇒ The switch statement checks the value of a given argument against a list of case values and when a match is found, a block of statements associated with that case is executed. If no match then the default statement will be executed.

⇒ The general form of switch statement is as follows:

```
switch(variable or expression)
{
   case value1 : block1;
                 break;

   case value2 : block2;
                 break;

   case value3 : block3;
                 break;

   --------
   --------

   default : default-block;
             break;
}
```

⇒ switch, case, default and break are keywords.

⇒ The break statement is used to exit from the current case structure. The break statement transfer the control out of the switch statement.

⇒ The expression is an integer expression or characters.

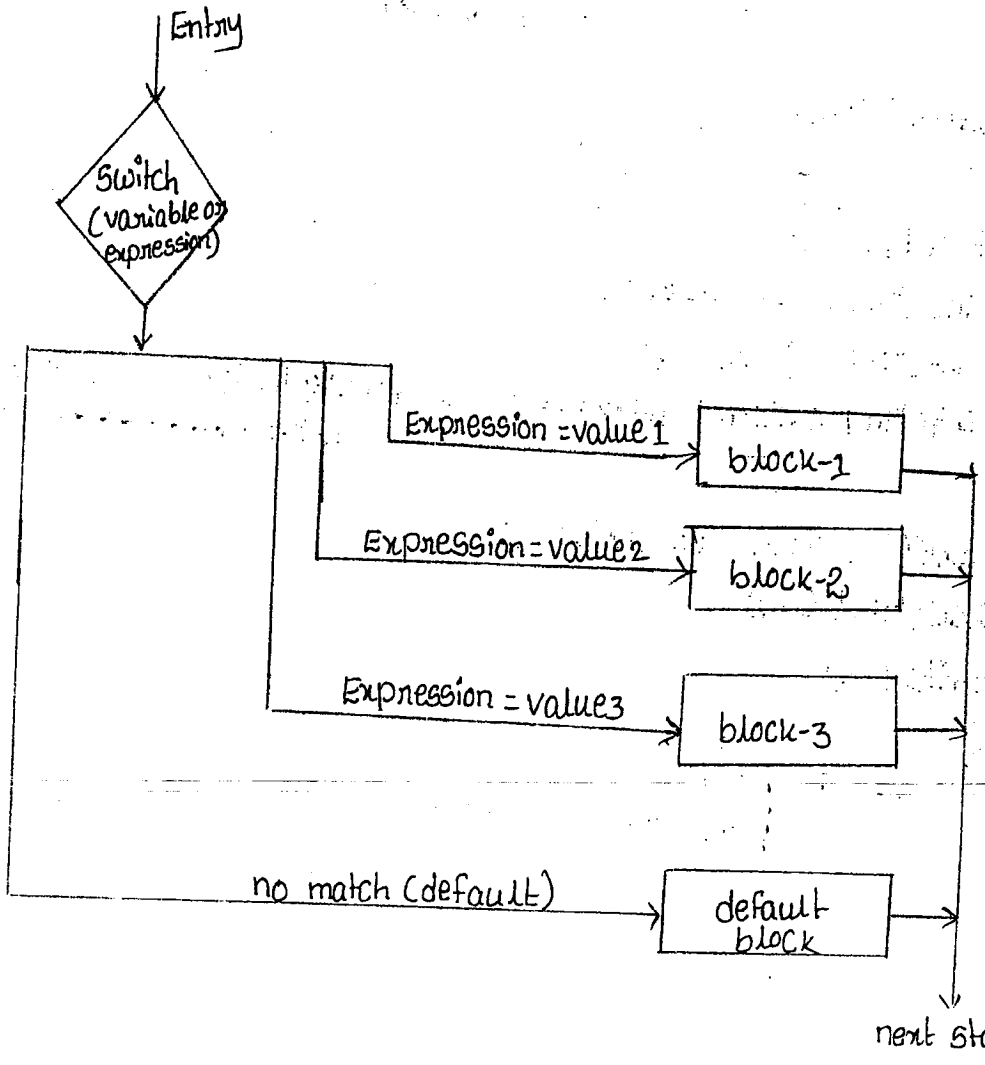
⇒ The values value1, value2, value3, ... are constants and are known as case labels. Each of these values should be unique within a switch statement.

⇒ Each case label should end with colon(:). Case lables may be integer numbers like 1,2,3. --- or it may be character constants like 'A','B','C',....

⇒ block1, block-2. ---- are statement lists and may contain zero or more statements.

⇒ When the switch is executed, the value of the expression is successfully compared against the case values. If a match is found, the block of statements associated with that case are executed.

⇒ The default is an optional case. If present, it will be executed if the value of the expression doesn't match with any of the case values.



Note:

→ The break statement is optional.

If break statement is not present, then all cases are executed.

→ There must be exactly one default statement, and it is optional.

→ case labels must be unique. No two labels can have the same value.

→ case labels must be constants.

Example program:-

→ Write a program to perform the arithematic operations based on the user choice.

Choice	operation
1	Addition
2	Subtraction
3	multiplication
4	Division
5	modulo division

```
#include<stdio.h>
main()
{
    int a,b,c,ch;
    printf("Enter a,b values:");
    scanf("%d %d", &a, &b);
    printf("\n 1. Addition \n 2. Subtraction \n 3. multiplication \n 4. Division
                                                \n 5. modulo division");

    printf("\n Enter your choice:");
    scanf("%d", &ch);
    switch(ch)
    {
        case 1: c=a+b;
                printf("\n Addition: %d", c);
                break;

        case 2: c=a-b;
                printf("\n Subtraction: %d", c);
                break;
```

```
        case 3: c=a*b;
                printf("Multiplication : %d", c);
                break;
        case 4: c=a/b;
                printf("Division : %d", c);
                break;
        case 5: c=a%b;
                printf("Modulo division: %d", c);
                break;
        default: printf("Wrong choice");
                break;
    }
    getch();
}
```

Sample output:-

```
Enter a,b values: 6 3
1. Addition
2. Subtraction
3. Multiplication
4. Division
5. modulo division
Enter your choice : 2
Subtraction : 3
```

Nested switch statement:-

⇒ We can place one switch statement inside another switch statement.

⇒ The inner switch() can be a part of the outer() switch.

⇒ The general form of nested-switch statement is as follows:

```
switch(expression)
{
    case 0:
        -----
        -----
        break;
    case 1: switch(expression)
            {
                case 0: block-1;
                        break;
                case 1: block-2;
                        break;
                  :
                default: staments;
                         break;
            }
        break;
    case 2: ------
            ------
            break;
      :
    default: default-statements;
             break;
}
```

Outer switch

Inner switch

⇒ The inner switch() and outer switch() labels may be same.

Example program:-

⇒ Write a program to determine whether the given number is even or odd. Use nested switch() statements.

```
#include<stdio.h>
#include<conio.h>
Void main()
{
   int x,y;
   Clrscr();
   printf("Enter x value:");
   scanf("%d",&x);
   switch(x)
   {
      case 0: printf("Number is Even");
              break;
      case 1: printf("Number is odd");
              break;
      default : y=x%2;
                switch(y)
                {
                   case 0: printf("Number is Even");
                           break;
                   case 1: printf("Number is odd");
                           break;
                }

   }
   getch();
}
```

Explanation:- In the above program the first switch is used for displaying a message such as even or odd when the entered number is 0 or 1.

→ When the number is other than 0 and 1, its remainder is calculated and stored in y. The variable y is used in the inner switch. If the remainder is 0, it will display the number is even. Otherwise it will display the number is odd.

The goto Statement:-

⇒ The goto statement is an unconditional brach statement. i.e. it doesn't require any condition.

⇒ The goto statement transfers the control from one part of a program to another part without testing any condition.

⇒ The goto statement has the following form:

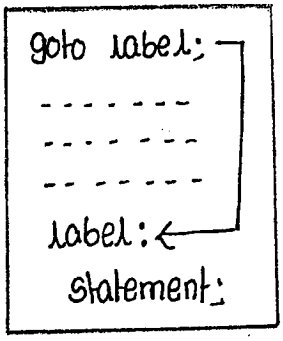
```
goto label;
```

where goto → is a keyword.

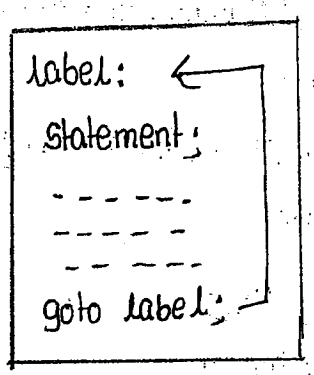
label → is the position where the control is to be transferred.

A label must be a valid variable name.

⇒ The general form of goto and label are as follows:



Forward jump



Backward jump.

⇒ The goto requires a label inorder to identify the place ~~whether~~ where the control is to be transferred.

⇒ The label: can be any where in the program either before on after goto label; statement.

⇒ During program execution, when a statement like goto label; is met, the control is immediately transferred to the place where the label: begins. This happens unconditionally.

Example programs:-

⇒ Write a program to check whether the given number is even or odd using goto statement.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int x;
    clrscr();
    printf("Enter x:");
    scanf("%d",&x);
    if(x%2==0)
      goto even;
    else
      goto odd;
    even: printf("Number is even");
          return;
    odd: printf("Number is odd");

}
```

Explanation:-

⇒ In the above program, if x is divisible by 2, then the control is transferred to the position 'even: and displays number is even.