UNIT-1

INTRODUCTION TO ALGORITHMS AND PROGRAMMING LANGUAGE

ALOGORITHM:-

An algorithm means developed by programmer to represent the basic logic of selected solution. All an algorithm written in simple English language. The solution in logical steps. An algorithm is represented by flowcharts, decision tables, pseudo codes and algorithm makes the program clear and they help coding.

An algorithm defined as a finite number of steps of instructions. Then produces an output for a set of input values. Algorithm cannot be compiled and executed by a computer.

Properties of algorithm:-

The properties of algorithm as follows

1. The algorithm must have presized instructions
2. The algorithm must have unambiguous instructions(clear instructions)
3. The algorithm should not have any uncertainity about execution of next instruction
4. It must be finite and cannot be open ended.
5. The algorithms must be terminate after finite number of steps.
6. The algorithm should be universal leads to a unique solution of the problem.

1. Write an algorithm for addition of two numbers

Step-1:- start

Step-2:- read a,b values

Step-3:- add a,b and store result in sum(sum=a+b)

Step-4:- display sum

Step-5:- stop

2. Find the biggest of three numbers

Step-1:- start

Step-2:- read a,b,c values

Step-3:- find the big value between a,b and store result max

Step-4:- find the big value between max & c & in max

Step-5:- display max

Step-6:- stop

3. Write an algorithm find the average of three numbers

Step-1:- start

Step-2:- read a,b,c values

Step-3:- find the average of a,b,c values and restore the max value

Step-4:- display the average value of the three

Step-5:- stop

Benfits of algorithm:-

the major benfits of implementing algorithm.

1. It makes the algorithm may be understandable and ambiguous.
2. It makes the program efficient, effective and easy
3. It references the program is easy step by step direction
4. It makes easy to modify and update the existing program.
5. It promotes effective testing of program at developing stage
6. It helps to determine the designed output by just giving input
7. It helps to solve complex programming.

Algorithm:-

Algorithm is a step wise process to solve a particular problem.

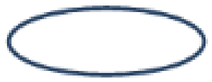
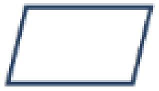
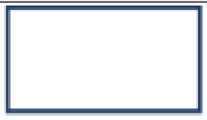
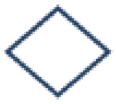
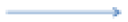
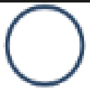
Flow chart:-

"a flowchart is a graphical representation of algorithm".

The program in which different types of instructions are drawn in the form of different shapes of boxes and the logical flow is indicated by interconnecting arrows. The main objective of creating flowchart is to help the programmer in understanding the logic of programmer and to trap any kind of logical errors present in algorithm.

Flow chart symbols:-

Flow chart use different symbols to represent different operations to the program. A flow chart is drawn according to define rules using standard flowchart symbols prescribed by ("ANSI") American national standard institute. The standard symbols that are frequently used in flow chart are

Symbol	Name	Function
	Start /End	Oval shape shows start or end.
	Input / output	Parallelogram shows input or output
	Action or Process	Rectangle represents process
	Decision	A diamond indicates a decision
	Arrows	Arrows shows flow direction
	Circle	Connector symbol

Flow chart symbols:-

Flow lines are represented by arrow lines. That are used to connect symbols these lines indicate the sequence of steps and the flow of operations.

Terminator:-

Terminator is used to represented by rectangle with rounded ends. These symbol is used to indicate the beginning (start),the termination(end/stop) in the program logic.

Input (or) output:-

Input/output is represented by the parallelogram these symbol is represents an input taken from the user (or) the output that is displayed to user.

Processing:-

The processing is represented by rectangle this symbol is used for representing arithmetic and data movement instructions. It denotes the logical process of moving data from one memory location to another location.

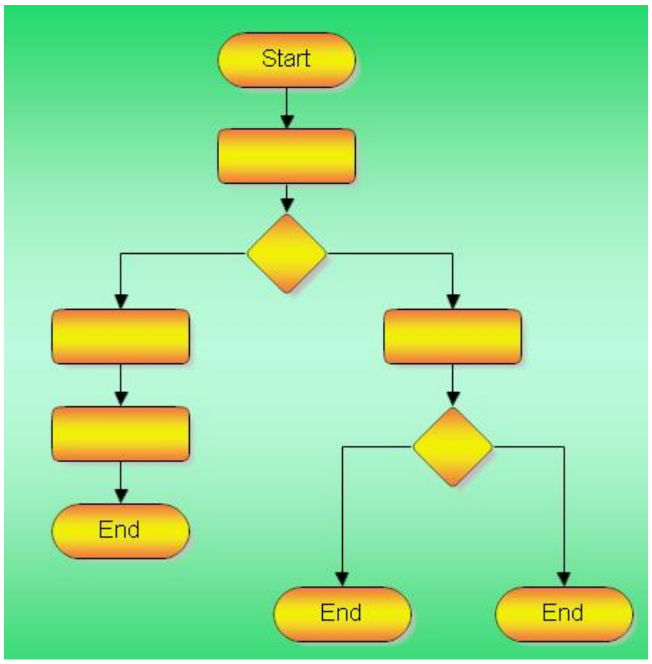
Decision:-

This symbol is represented by diamond. It denotes a decision to be made. This symbol has one entry and exist paths. The path chosen depends on weather the answer to a question is yes (or) no.

Connector:-

This symbol is represented by circle. It is used to join different flow lines.

Basic structure of a flow chart:-



Guidelines for creating a flow chart:-

The following guide line should be used for creating a flow chart

1. The flow chart should be clear, neat and easy to flow.
2. The flow chart should use common words and statements.
3. Each flow chart must had a logical start and logical stop.
4. Intersection of flow lines should be avoided.
5. All necessary requirements should be listed in logical order.
6. Only one flow line should be come out from a process symbol.
7. Only one flow line should enter a decision symbol.
8. Statements should be return briefly within standard symbols

9. Connector symbol should be used in case of complex flow charts, they reduce number of flow lines.

Advantages of flow charts:-

The following are the advantages of flow chart

1. **Makes logic clear:-**

   it is easy follow a graphical representation of the task. The symbols are connected in such a way. That they make the system visible.

2. **Communication:-**
   Since the flow chart is a graphical representation of a problem solving logic. It is enhanced meaning of communicating the logic of a system.
3. **Effective analysis:-**
   A flow chart helps the problem analysis in an effective way.
4. **Proper testing and debugging:-**
   Flow chart helps to identify and correct the errors in the program. It also helps in testing process.
5. **Appropriate document:-**
   Flow chart acts as a high quality program documentation tool.

Disadvantages of flowchart:-

The following are the disadvantages of a flow chart

1. **Complex:-**

For very large programming consisting of thousands of statements. The flow chart consists of many papers (or) pages making them different to flow.

2. **Costly:-**

Flow charts are feasible for sort and straight forward problem solving logic flow chart because a costly flow chart, in huge application**.**

3. **Difficult to modify:-**

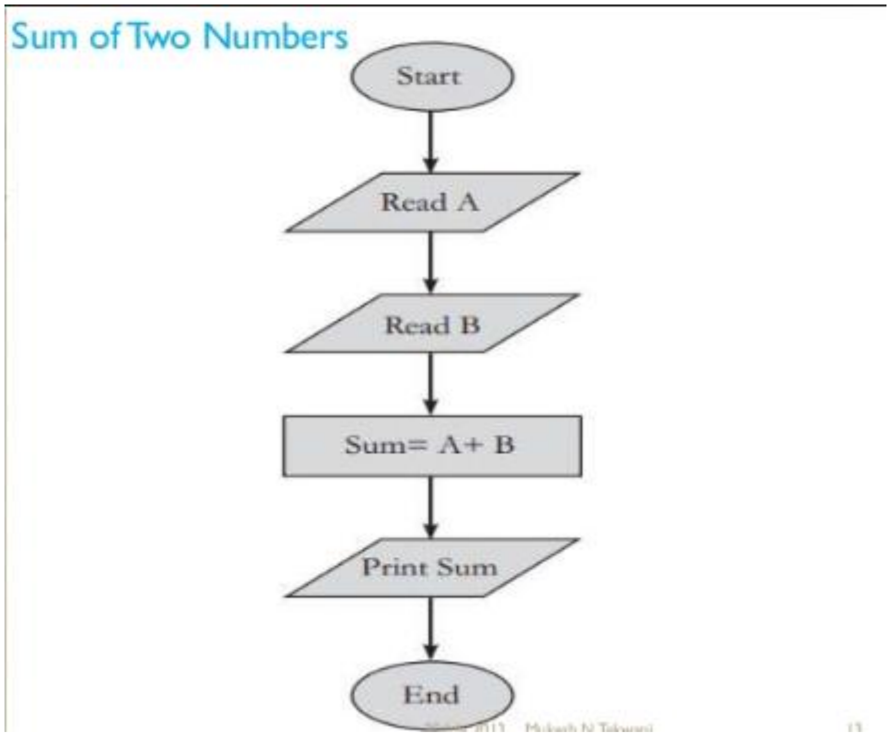
Any modification to flow chart needs to redraw the flowchart. Considering the entire logic again due to its symbolic nature redrawing complex flow chart is a difficult task.

4. **No update:-**

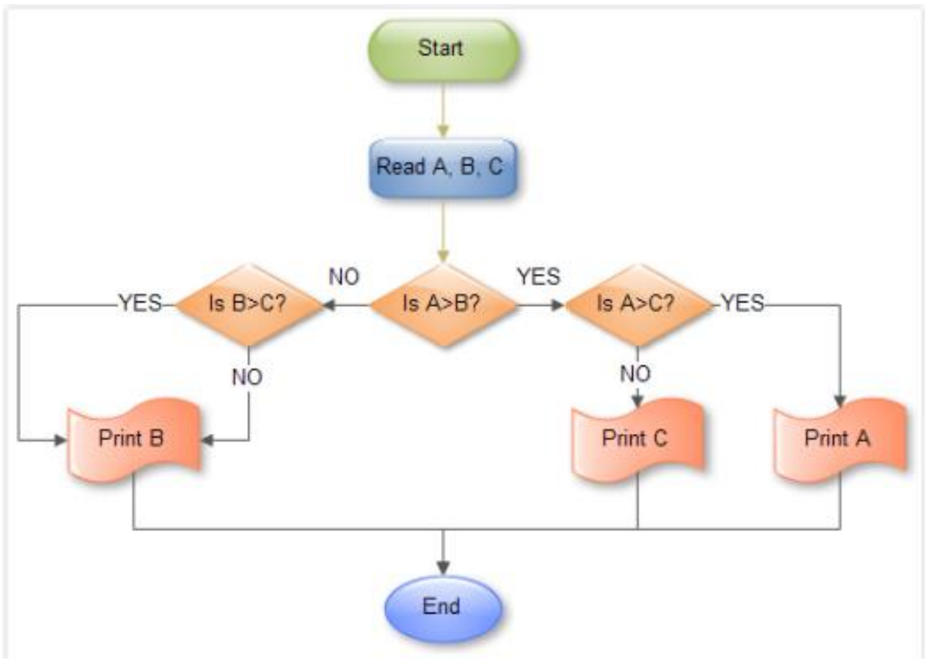
Programs are generally updated regularly but the corresponding flow charts are not updated regularly.

Examples:-

Draw a flow chart to find sum of two numbers



2. Draw a flow chart to find the largest of three numbers



Generations of programming language:-
Program:- program is a set of instructions
Programming:-
1. Process of creating programs.
2. A programming language is a computer language used to write instructions for computer in the well defined format.
3. The set of instructions is called is called a program and the process of creating programs is called programming.

Types of programming language:-

Programming languages are classified into major languages.

1. Machine language (or) binary language(1st generation language)
2. Assembly language(second generation language)
3. High level language(third generation language)
4. Very high level language(4th generation language)
5. Fifth generation language

1. Machine language (or) binary language:-
1. machine level language is the lowest level programming understand by computers
2. to perform various input and output operations
3. the set of instructions which are directly understand by the c.p.u of computer is called machine language.

4. The programming languages which contains machine code is called machine language.
5. All the information in the computer is handle using integrated circuits, semiconductors.
6. The machine language uses two binary digits 0 and 1 to handle input and outputs. So it is also known as binary language.
7. Machine language different from machine to machine because the internal structure of every computer different from one computer to another.
8. Machine language can be easily used by the computer, but it is difficult to read and understand by the user.
9. The machine language program runs fastly because no translations is require for the c.p.u.

2. Assembly language(second generation language):-

1. Assembly language was the next high level programming language which is categorized as the second generation language
2. Assembly language is easy to understand compare to machine language. It uses English words it performs specific operations for example "add" is used for addition, "sub" is used for subtraction etc.
3. Assembly language is machine depended language
4. Assembler is used to translate assembly language instructions in to machine language and reverse also
5. Assembly language statements are return in one per line where each statements contains operations
6. Programming in assembly language requires extensive knowledge of computer design.

3. High level language (or) third generation language:-

1. High level programming languages includes: FORTRAN, COBOL, PASCAL, BASIC, C, C++, JAVA which enables the programs to develop the software applications.
2. The high level language use syntax which is very easy to understand.
3. Programs written in high level language (or) shorter in length.
4. Translates like compilers & interpreters are used to translate programs written in high level languages into machine languages and reverse also
5. High level languages are machine independent
6. High level languages are easy to learn because they use common English words as, they (or) their keyword
7. It is easy to modify and maintain the programs certain in high level language.

4. Very high level language:-

1. Fourth generation languages are the easiest programming languages available today. They are very easy to learn because these syntax and grammar is very easy to learn.

2. Programming in this languages does not require any programming experience in other language.
3. Fourth generation languages are machine independent.
4. Fourth generation languages are also known as very high level languages.
5. Most fourth generation languages are used to occur data bases
6. Sql(structured query language) is the best example fourth generation language. It is used to create and modify information in dbms(database management system).

5. Fifth generation language:-

1. Fifth generation programming language design a make a computer solve a given problem without the programmer(user)
2. The fifth generation programming language is also known as natural language.
3. Fifth generation programming is based on solving problems using constraints given to the program.
4. Prolog &mercury are the best known fifth generation language.

Pseudo code:-

- ➢ Pseudo code is a form of structured English, that describes algorithms.
- ➢ An ideal pseudo code must be complete, describing the entire logic of the algorithm. So, that it can be translated straight away in to a programming language.
- ➢ It is basically meant for human reading rather than machine reading.
- ➢ Pseudo codes are an outline of a program that can be easily converted into programming statements.
- ➢ Purpose of pseudo code is to enhance human understanding of the solution. They are commonly used in textbooks & scientific publications for documenting algorithms
- ➢ Flow charts can be considered as graphical alternatives to pseudo codes.

Parts of pseudo code:-

- Consider the following the parts of pseudo code
- if condition then

```
        Sequence 1
        else
        Sequence 2
        end if
```

- Here, the else keywords & sequence 2 are optional if, the condition is true sequence 1 is performed otherwise sequence 2 is performed.

Example:-

```
if age>=18 then
display eligible to vote
else
display not eligible
end if
```

structured programming language:-

1. Structured programming also referred as modular programming
2. It is basically a subset of procedural programming which enforces a logical structure on the program to make it. More efficient an easier to understand & modify.
3. Structured programming is a top down approach in which the overall program structure is breakdown into separate modules.
4. This allows code to be loaded in to memory more efficiently and also be refused in other programs.
5. Structured programming is based on modulations.
6. Module procedural languages supports the concepts of structured programming.
7. In structured programming the program flow follows a simple sequence & usually avoids the use of goto statements.
8. Structured programming also supports, selection of repitations

Advantages of structured programming language:-

1. The Goal of structured programs is to write correct programs that are easy to understand & modify.
2. Structured programs takes less time to write then other programs
3. A structured program is easy to debug because each procedure in its specialized to perform just one task.
4. Individual procedures are easy to change as well as to understand.

Example of structured programming:-

1. To create a program to manage the names & addresses of a list of students for these you would need to breakdown the program in the following modules.
   - Enter new name and address
   - Modify existing entries
   - Sort entries
   - Print the list
2. Now, each of the proceeding modules can bee furtherly breakdown into small modules.

3. For example, the first module can be
    - Prompt the user to enter new data
    - Read the existing list from the disk
    - Add the name & address to the existing list
    - Save the updated list to the disk.
4. Simply modify existing entries can be furtherly divided into modules such as
    - Read the existing list from the disk
    - Modify one or more entries
    - Save the updated list to the disk.

Introduction:-

'C' is a general purpose programming language. It is a procedure – oriented, Structured programming language.

The structured programming language allows you divided into small modules by using functions. It is a middle level language that means it has good high-level language programming skills and it also has low level programming features.

Dennis Ritchie developed C language in the year 1972 at AT&T Bell laboratories in U.S.A. The C-language was developed from the language B.C.P.L (Basic combined programming language). The C-language is well suited for developing system software and application software.

Features of C-Language:-

C-Language is very powerful and popular because of its features. The Main features are:

1.Portability:- C-Language programs are highly portable. Portability means a c-program written in one environment can be executed in another environment. For example you write a program in DOS environment you can run in windows environment.

2.Structured Programming language:- C-language is a structured programming language the structured programming basically consists of writing a list of instructions for the computer to follow, and organizing these instructions into groups known as functions.

3.Extendibility: - C-language has an important facility called extendibility. It means you can write your own file or functions and include in another programs in other words a user can write No. of functions, sub-programs according to the requirement.

4.Reliability: - A ‘C’ compiler gives and accurate results it has a facility of warning which guides for better and efficient programming.

5.Middle level language: - ‘C’ language is also called middle level language. Because it has both types of features i.e. high-level languages as well as low-level languages.

Uses of c:-

1. ‘c’ is very simple language i.e., used by software professionals
2. The uses of ‘c’ language are:-
    - C-language is mainly used for system programming.
    - It is widely accepted by professionals
    - For portability and convenience is some times used as an intermediate language for implementing other languages.
    - ‘c’ language is widely used to implement user applications.

structure of the ‘C’ program.

The ‘C’ Program structure contains the following sections.

1. Documentation Section.
2. Header file section
3. Definition section
4. Global declaration section
5. Main ()
   {

   Declaration part

   Execution part

   }
6. Function definitions section

1. Documentation Section: - In the documentation section we can give the comments. here comment statements are non-executable statements. Comment can be started by using ‘/*’ and ends with the ‘*/’.

Syntax: /* Line 1 */

You can give the comments more than 1 line.

```
/* Line 1
   Line 2
   Line 3 */
```

2.Header file section: - This section provides instructions to the compiler to link functions from the Library each header file by default contains with the extension of 'h' the file should be included by using # include.

E.g.: # include <stdio.h>

The stdio.h is a file it is included all the definitions of input, output functions.

3.Definition Section: - We can define a variable with its value in the definition section.

Syntax : #define variable name value

e.g. 1. #define a 10

2. # define name "   "

4.Global declaration section: - Some variables are used in more than one function such variables are called global variables and that variables are declared in the global declaration section that is out side of the main ().

5.Main ( ): - Every 'C' program must contain main ( ) with empty parenthesis after main is necessary. The function main is the starting point of every 'C' program. The program execution starts with the opening braces ({) and ends with the closing braces (}) between these braces the programmer should give the program statements. The main has two parts

They are

Declaration Part: - The declaration part declared the entire variables that are used in executable part the initialization of variables are also done in this part.

Execution Part: - This part has reading, writing and processing statements having input or output functions, formulas, conditional statements, looping statement and function calling statements.

Function definitions section: - in this section function definitions are defined by the user. These functions are generally defined after the main function or before the main function. This section is optional.

Ex:-

Write the first 'c' program

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

```
        void main()

        {

        clrscr();

        printf("hello mscs");

        return 0;

        getch();

        }
```

Output:-

```
        Hello mscs
```

1. **#include<stdio.h>:-**
   - ➢ It is the first statement in our code. All pre-process commands starts with #symbol(hash)
   - ➢ The # include statements tells the compiler to include the standard library (input/output) or header file (stdio.h) in the program
2. **Main():-**
   - ➢ Main() function after all the statements in the program have been written. The last statements in the program have been written. The last statements in the program return will an integral value to the operating system.

     The two {} curly brashes are used to grap of all the related statements of main() function.

Printf("hello mscs"):-

The printf function is defined in stdio.h file and is used to print text on the screen. The message to be displayed on the screen to the enclosed with in double quotes(" ") and put inside brackets (or) parenthesis.

Example:-

```
/* print DNR college statement */

#include<stdio.h>

void main()

{

clrscr();

printf("\n DNR college");

getch();

}
```

Example:-

Sequence	**purpose**
\n	new line
\t	tab space
\b	back space
\v	vertical tab
\f	new page/clear screen

Return 0:-

This is a return command that is used to write the value "0" to the operating system give an indication. That there are no errors during the execution of program.

FILES used in c program:-

Files used in 'c' program is divided into 4 types

1.Source file

2.Header file

3.Object file

4.Executive file

Source File:- The source code file contains a source of the programs. The file extension of any c source code that defines the main functions' and other functions. The main function is the starting point of execution when you successfully compile and run the program in the files of c program the source file is always file extension is".c".

Header file:-Header files provide instructions to the compiler to link functions from the library. Each header file by default contains.'h'extension and its name can use only leters,digits,dashes and under scores some standard header files are automatically available to 'c' programmers

Examples of standard header files includes

*****string.h:-for string handling function

*stdio.h:-for standard input/output functions *math.h:-for mathematical functions

*alloc.h:-for dynamic memory allocation

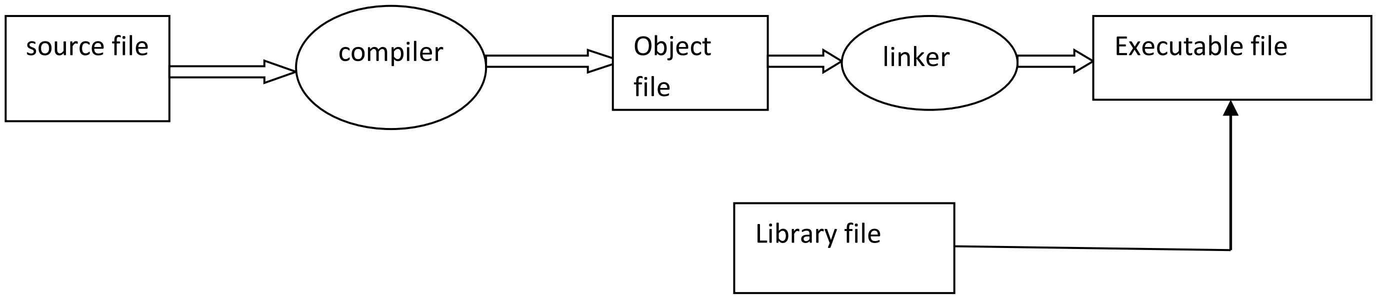
*conio.h:-for clearing the screen

Object file:-Object files are generated by the compiler as a result of processing the source code file.thy contain binary code of the function definitions (.o).

Binary executable file:-The binary executable file is generated by linker the linker links various object files to produce a binary file that can be directly executed (.exe).

Compiling and executing C-program:- C is a compiler language every c program must be run through a 'C' compiler that's creates an executable file to be run by the computer

The programming process starts with creating a source file. That consists of a program written in 'C'langauge



***Source file**:-The source file usually contains ASCII characters and can be produced with a text editor the source file is produced by a special program called compiler

***compiler**:-The compiler translates the source code into a object code the object code contains machine instructions for the CPU .

***Object file:**The object file is processed with another special program called linker.

***Linker:** The output of the linker is an executable file**.**

In C language program, there are two kinds of source file, .c source file,.h source file

Every C program uses standard headers files

***Keywords**:-C languages have some reserved words which cannot be used as variables the reserved words are called keywords. There are mainly 40keywords among which 32 keywords are used by many ‘c’ compilers these keywords are called standard keywords where as the remaining keywords are called optional keywords

- Auto
- Break
- Case
- Char
- Const
- Continue
- Default
- Do
- Double
- Else
- Enum
- Extern
- Float
- For
- Go to
- If
- Int
- Long
- Register

- Return
- Short
- Signed
- Size of
- Static
- Struct
- Switch
- Type def
- Union
- Unsigned
- Void
- Volatile
- while

Optional keywords:-

Ado

FORTRAN

Sum

Huge

Entry

Near

Far

Pascal

Identifiers: Identifier is a name that is given to variables, functions and arrays.

Rules for constructing Identifiers:

1. Identifiers must be from the character set.
2. The first character of an idenfier should be an alphabet. It should not be a digit or special character.
3. Identifiers should not be a keyword.
4. Special characters are not accepted in the identifiers except '-' (under score)
5. The length of an identifier should not be exceeding eight characters.

BASIC DATA TYPES IN "C":-

A Data type is a set of values along with a set of rules for allowed operations. 'C' supports several data types of data each of which is stored differently in the computers memory mainly data types are divided into three types.

1.Primary data types

2.Derived data type.

3. User defined data type

1.Primary data type: - 'C' supports mainly four primary data types.

a. Character data type
b.Integer data type
c.float data type

d.Double data type.

a. Character data type: - The character data type accepts single character only. Characters are either signed or unsigned. But mostly characters are used an unsigned type. The size of the character data type is 1 byte in the memory. The range of unsigned character is 0to 255.

The range of signed are character is –128 to +127. Char is the keyword of the character data type.

Syntax: char list of variables

Ex. Char ch1, ch2, ch3;

b. **<u>Integer data type: -</u>** An integer type accepts integer values only. It does not contains any real or float values. The range of an integer variable is -32, 768 to +327,67. Int is the keyword for integer data type.

In generally 2 bytes of memory is required to store an integer value.

Syntax: int list of variables.

Ex: int a, b, c;

c. **<u>Float data type: -</u>** The float data type accepts real values it can contains any floating point values. The range of the floating variable is 3.4E – 38 to 3.4E + 38. Float is the keyword for

In generally 4 bytes of memory is required to store an float value with 6 digits of precision.
Syntax: float list of variable.

Ex: float f1, f2, f3;

d. **<u>Double data type : -</u>** The double data type accepts large floating value. The range of the double variable is 1.7E –308 to 1.7E + 308. Double is the key word for double data type. In generally 8 bytes of memory is required to store double value.
Syntax: Double list of variables.
E.g. Double d, e, f;

Sl.No.	Data types	Size	Range	Format string
1.	Char	1	-2^7 to 2^7-1 -128 to + 127	%c
2.	Signed char	1	- 128 to +127	%c
3	Unsigned char	1	0 to 255	%c

4	Int	2	-32768 to +32767	%d
5	Signed int	2	-32768 to +32767	%d
6	Unsigned int	2	0 to 65535	%d
7	Short int	2	-32768 to +32767	%d
8	Signed short int	2	-32768 to +32767	%d
9	Unsigned short int	2	0 to 65535	%u
10	Long int	4	-214,74,83,648 to +214,74,83,648	%ld
11	Signed long int	4	-214,74,83,648 to +214,74,83,648	%ld
12	Unsigned long int	4	429,496,72,95	%ld
13	Float	4	3.4E –38 to 3.4E +38	%f
14	Double	8	1.7E – 308 to 1.7E +308	%f
15	Long Double	10	3.4E –4,939 to 1.1E + 4,932	%lf

2.Derived data types: - Derived data types are derived from the primary data types. The derived data types may be used for representing a single or multiple values. These are called secondary data type. The derived data types are arrays, strings, etc.

3.Userdefined data types: - The data types are defined by the user is called user defined data types. The user defined data types are structures unions etc.

A) "Enumerated data types :- It allows the user to define a variable or an identifier, which is used for representation of existing data types. In other words, it provides us a way to define our own data type and also can define the value for a variable or an identifier stores into the main memory.

Syntax : enum identifier { v1,v2,v3…….,vn};

Here enum is the reserve word and v1,v2,v3…,vn all are the values which is also called enumeration constants.

Ex : enum month{jan,feb,mar,….,dec};

B) Typedef :- This is used to represent the existing data type .i.e. by using this the new type can be used in place of the old type anywhere in a C program. Also we can create the typedef variables for improving the readability of the program.

Syntax : typedef data-type identifier;

Here data-type may be int, float ,double and char. Identifier gives us the information of new name given to the data type. Note that type def cannot create a new type.

Ex : typedef int pay;

Here pay tells us the link with int can used for the declaration of the variable as

Pay p1,p2

VARIABLES :

1. Variable is an identifier .It is used for storing value .The variable can be changed during the execution of the program .

2.variable refers to an address of memory where the data is stored

3.'C' supports two basic kinds of variables

*.Numeric and *.character variables

***Numeric variables**:-

1.Numeric variables can be used to store either integer values (or)floating point values

2.Numeric variables may be also associated with modifier such as short,long,signed and unsigned

3.The difference between signed and unsigned variables are can only be positive

***Character variable**:-

1.character variables ca include any letter from the alphabet or from the ASCII code

2.char and numbers 0 to 9that are given with in single quotes

***Declaring variables**:-

*Each variable used in the program must be declared. to declare a variable specify the data type of the variable followed by its name type of the variable followed by its name

*The data type indicates the kind of data that the variable will store *variable names should always be meaningful and must reflect the purpose of they are used in the program

*in 'c'variable declaration always ends with a semi column(;)

For examples:-

- int a;
- int emp-num;
- Char grade;
- double balance-amount;
- float b;

*In 'c' variables are declared at three basic as follows

1.When a variable is declared inside a function it is known as local variables

2.when a variable is declared in the definition of function parameter then it is known as formal

3.when the variable is declared outside of all functions it is known as global variable

Initializing variables:-

*while declaring the variables we can also initialize them with some values

For example:- int a=10;
int emp-num=7;
Char grade='A';
double balance-amount=100000;
float salary=5000;

<u>Constants:</u> - The value was not changed during the execution of the program is called constants. Mainly they are two types of constants.

They are

1. Numeric constants
2. Character constants.

1. <u>Numeric constants: -</u> These have numeric data with or without decimal points having positive or negative sign. These are further sub divided into two categories that are

a. Integer constant

b. Real or float constant

a. Integer constant: - Integer constant has integer data with out any decimal points are with any positive or negative sign. These are further sub divided into three types. Those are

i. Decimal integer constant

ii. Octal integer constant

iii. Hexa decimal integer constant

i) Decimal integer constant: - These have no decimal points it is a combination of 0to9 digits. These have either positive or negative sign. For example: 23, -36

ii) Octal integer constant: - These consist of combination of numbers from 0to7 with positive or negative sign. It has leading with 'O' (Upper case or lower case). 'O' means octal for eg: O37, O-35

iii) Hexa decimal integer constant: - These have Hexa decimal data leading with OX or X or H (capital or small). These have combination of 0to9 and AtoF (capital or small). These letters represents the numbers 10to15.

For eg: OX 32, Oxef

b. Real or float constant:- Some constants which have decimal point value with in it is having any positive or negative sign.

For eg: 22.34, 2.4 E 38 that means 2.4 X $10^{38.}$

Real constants are further divided into two categories.

i. With exponent part

ii. Without exponent part.

i.Without exponent part: - Without exponent 'E' and having a decimal part mantissa.

E.g.: 30.6, -30.6

ii. With exponent part: - . It is also called a scientific representation. Here 'C' has base

value of 10. It computes the power.

E.g.: 3.5 X 10^5 = 3.5 e5.

2.character constants: - character constants have either a single character or a group of characters or a character with back slash (\) used for special purpose. These are further divided into three types.

a. Single character constant.

b. String character constant.

c. Back slash character constant.

a. **<u>Single character constant: -</u>** These have a single character with in single quotes. So, These are called single character constant.

E.g.: ‘a’,’5’

b. **<u>String character constant: -</u>** A string is a combination of character or group of Characters, a string constant or a string is enclosed with in double. So it is called String constant.

E.g. : “Rama Krishna”

c. **<u>Back slash: -</u>** These are used for special purpose in ‘C’ language. These are used in output statement like print (). Puts (), another name of Back slash character constant is escaping sequences.

The escape sequences are.

‘\a’	- bell
‘\b’	- back space
‘\t’	- Horizontal tab
‘\v’	- Vertical tab
‘\r’	- Carriage returns
‘\n’	- New line
‘\o’	- Null character.

<u>OPERATORS</u>

Operators are used for compute a formula or compare two variable values or create logical relationship between two operands or low-level programming we can say operators are used for processing.

Operators are divided into:

1. Arithmetic operators

2. Relational operators.
3. Logical operators
4. Assignment operators
5. Conditional operators
6. Bit wise operators
7. Increment / Decrement operators.
8. Comma operator.
9. Equality operators
10. Size of operators

1. Arithmetic Operators: - 'C' provides all the basic Arithmetic operators in an Arithmetic expression like x + y x and y are the operands. '+' is the operator. 'C' used the precedence rules to decide which operator is used first these are listed in the following table.

Operator	**Descriptions**	**Example**
+	Addition	x+y
-	Subtraction	x-y
*	Multiplication	x*y
/	Division	x/y
%	Modulus (remainder after division)	x%y

2.Relational Operators: - The comparison can be done with the help of relational operators. An expression such as containing a relational operator is termed as a relational expression. The value of relational expression is either 1 or 0. It is '1' if the specified relation is true. And '0' if the relation is false.

Operator	**Description**	**Example**
<	less than	x<y

<=	Less than or equal to	x<=y
>	grater than	x>y
>=	grater than or equal to	x>=y
= =	Equal to	x= =y
! =	not equal to	x!=y

3.Logical Operators : - A statement contains more than one relational operators they must be separated by a logical operator an expression which combines two or more relational expressions is termed as a logical expression compound relational expression.

Operator	**Description**	**Example**
&&(and)	If both condition are true the result will be true other wise false.	(x>y)&&(x<z)
!! (OR)	If at least one condition is true the result will be true other wise false.	(x>y)!!(x<z)
! (not)	If the condition is true, the result will be false. If the condition is false, the result will be true.	!(x>y)

4.Assignment Operators : - Assignment operators are used for assign the result of an expression to a variable the equal to sign ('=') is the assignment operator.

Operator	**Description**	**Example.**
=	Assign the value of operand to the left	x= y
+=	Adds the operand and assigns the result to the left operand	x+= y
- =	Subtract the right operand from the left operand and Stores the result in the left Operand	x- = y
* =	Multiplies the left operand by the right operand and stores the result in the left operand.	x*= y
/=	Divides the left operand by the right operand and stores the result in the left operand	x / = y
% =	Mod the Left operand by the right operand and stores the remainder in the left operand.	x % = y

5.Conditional Operators: In ‘c’ conditional operator ‘? :’ Constructs conditional expression of the form

Exp 1 ? Exp 2 : Exp 3 ;

Where Exp1, Exp2, Exp3 are expression. The operator ‘?:’ works as follows Exp1 is evaluated first. If it is true then the expression Exp2 is evaluated. If the Exp1, is false Exp3 is evaluated.

Operator	**Description**	**Example**
? :	Question mark, Column	(a>b)? a+b : a-b;

6.Bit wise Operators : - A special type of operators known as Bit wise operators for manipulation of data in Bit level. These operators are used for testing the Bits are shifting them right or left.

Operator	**Description**	**Example**
& (and)	Bit wise and	x&y
! (or)	Bit wise or	x ! y
^ (Exclusive or)	Bit wise exclusive	x^y
~	Bit wise not	~x
<<	Bit wise left shift	x<<1
>>	Bit wise right shift	x>>2

7.Increment / Decrement operators: - In some cases it is necessary to modify the value of the variable by adding one to the variable or –1 to the variable until the loop terminates the ‘c’ provides the mechanism increment or decrement operators to accomplish this.

Increment Operators: In ‘c’ the operator ‘+ +’ is used as Increment operator it adds 1 to the variable we can add 1 to the variable in one of the two types.

i. Pre incrementing
ii. Post incrementing.

i.) Pre-incrementing: This operator first increments the value of the variable and then execution proceeding.

Syntax: ‘++’ variable name;

Ex: ‘++’a;

‘++’ a is equivalent to a=1+a

ii).Post-incrementing : This operator continuous with the execution before adding 1 to the variable and then increments the variable by 1

Syntax: variable name ‘++”;

Ex: a’++’;

a ‘++’ is equivalent to a=a+1.

Decrement Operator : In ‘C’ the operator ‘- -‘ is used as decrement operator. It subtracts one from the variable we can subtract one from the variable in one of two ways.

a.Pre – Decrementing

b.Post – Decrementing

a)Pre–Decrementing : This operator first decreases the value of the variable and then execution proceeds.

Syntax : ‘- -‘ variable name;

Ex : ‘- -‘ a;

‘- -‘ a is equivalent to a= -1+a.

b)Post–Decrementing : - This operator continuous with the execution before subtracting one from the variable and then decrements the variable by 1

Syntax : Variable name ‘- -‘ ;

Ex : a’- -‘;

a ‘- -‘ is equivalent to a=a-1.

8.Comma operator: - The comma operator is used to separate more than one variable in variable declaration.

Ex.: Int a, b, c;

The comma operator is also used to separate two or more expressions.

Ex.: x=2,y=5;

9.Equality operators:- C language supports two kinds of equality operators to compare their operands for equality or inequality. They are '==' equal to x==y
'!=' not equal x!=y

10.Size of operators:- The size of operator is a unary operator used to calculate the size of data type this operator can be applied to all data types this operator is used to determine the amount of memory space that the variable/expression/data type will take

```
Ex:-int a =10;
    unsigned int result;
    result=size of (a);
     result= 2 which is the space required to store the variable 'a' in memory
```

Examples of constants:-

Without constant	With constant

#include<stdio.h>
 Main()
{
int m=10;
    m=20;
printf("%d",m);
}
o/p:-20
	#include<stdio.h>
	Main()
	{
	Const int m=10;
	        m=20;
	printf("%d",m);
	}
	o/p:-cannot modify a const object in function name

<u>Input/Output functions</u>

The Input/Output functions are used for reading the data from the keyboard and displaying the result on the screen are the two main tasks of any program to perform these tasks 'C' has no. of Inputs & Output functions. When a program is needs data it takes data through the input functions and send the result through the output function.

<u>Input Functions</u>: Input functions are

1) scanf ( )
2) getch ( )
3) getchar ( )
4) getche ( )

5) gets ( )

1.<u>scanf () : -</u> The scanf function is used to read the data from the keyboard. You can store the given value into the variable through the scanf ( ).

Syntax: scanf (“Control string”, &v1,&v2,&v3,---- &vn”);

Here v1, v2, ----vn are the variables and “&” is used to store the given value into the variables. The control string has some format strings.

Some format strings are

Format String	**Meaning**
%c	To read a single character.
%d	To read the integer value
%ld	To read the long integer value
%f	To read the float value.

<u>2.getchar ( )</u>: - This function is used for reading a single character from the keyboard. The getchar ( ) can be assigned a character into the character type variable.

Syntax: ch=getchar ( )

Where ‘ch’ is the variable of character type.

<u>3.getch ( ) : -</u> The getch( ) is used to get a single character from the keyboard it will not display the character on the screen and it will store the given character on the buffer it is used at the end of the program to terminate the output screen.

Syntax: getch ();

<u>4.getche ( ):</u> - The getche ( ) is used to get a single character from the keyboard it will display the character on the screen and the character will be stored on buffer it is used at the end of the program to terminate the output screen.

Syntax: getche ();

5.**<u>gets ( ) :</u>** - The purpose of the gets function is to read the string it can read a string until you press enter key from the keyboard it will mark null character ('\0') at the end of the string.

Syntax: gets (ch);

Where 'ch' is the string variable.

<u>Output functions:</u> output functions are:

2. printf()
3. putchar ()
4. puts ( )

1. **<u>printf ( ) :</u>** The printf( ) is used to display a text message or a value stored in the variable. It requires conversion symbol and the variable name to print the data.

Syntax: Printf ("control strings", v1,v2,------vn");

(OR)

Printf ("Message line or text line");

Where v1,v2,-------vn are the variable.

The control strings uses some printf ( ) format strings some format strings are:

Format string	**Meaning**
%c	To print a single character
%d	To print the integer value
%ld	To print the long integer value
%f	To print the floating value.

2.**<u>putchar ( ):</u>** - The putchar ( ) is a single character out put function. It can display a single character on the screen at a time.

Syntax: Putchar (ch);

Where 'ch' is the variable of character data type in which a single character data is stored.

3.puts ( ) : The purpose of puts ( ) is to print or display a string inputed by the gets ( ).

Syntax : puts (s);

(OR)

puts ("Text line");

Where 's' is the string variable it will display the string stored in 's'.

Type conversion and Type casting:-

The type conversion is done when the expression has various of different data types the data type is promoted from lower to higher level

Type conversion is automatically done when assign an integer value to a floating point variable consider the following code

```
Floatx;
Int y=3;
X=y;
```

Now x=3.0 as the integer value is automatically converted into

Its equivalent floating point representation

Type casting:- Type casting is also known as forced conversion it tells the compiler to represent the value of expression in certain way.it is done when the value of higher data type is converted into the value of lower data type

Consider the following code

```
Float salary=10000.00,x=y
   Int sal;
   Sal=(int)salary
```

When floating point numbers are converted into integers the digits after the decimal are truncated

Example program:-

```
/*Addition of two numbers*/
      #include<stdio.h>
            Main()
```

```
                {
              Int a,b,c;
              Clrscr();
          Printf("enter a,b values");
         Scanf("%d",&a);
         Scanf("%d",&b);
           C=a+b;
        Printf("addition=%d",c);
           }
```

Unit-2

Decision control and looping statements

Introduction to decision looping control statements:- supports two types of decision control statements that can after the flow of sequence of instructions.

They are 1) conditional branching statements

2) unconditional branching statements.

1) conditional branching statements – the conditional branching helps to jump one part of the program to another depending on weather a particular condition is satisfy or not. the control statement includes

a) If statement

b) If else statement

c) Nested if-else statement

d) if-else-if statement or ladder if statement

e) Switch case statement

a) if statement:- only one statement occurs in "if". It is having only one block.

```
Syntax:- if (condition)
         {
         True statement;
         }

         Statement x;
```

First the condition will be checked. If the condition is true than the true statement block will be executed and after execution of this block, statement x will be executed. If the condition is false then only statement X will be executed

b) if else statements :- This statements also has a single condition with two different blocks (i) is true block and other one is false block

Syntax:- if (condition)

```
{

True statement;

}

Else

{

False statement;

}

Statement x;
```

First the condition will be checked. If the condition is true then true statement block will be executed. Then control goes to statement x.

If the condition is false then the false statement block will be executed then control goes to statement x.

c) nested if else statement:- when an if statement occurs within another if statement is called nested if else statement.

Syntax:- if(condition)

```
{

if(condition 2)

{statement 1;

}

else
```

```
        {
        Statement 2;
        }
        }
        else
        {
        Statement 3;
        }
        Statement x;
```

Condition one will be checked if it is true condition two will be checked. If condition 2 is true then the statement one will be executed. If condition 2 is false the statement two will be executed.

d) if-else-if or ladder if statement:- no. of conditions arise in a sequence, then we can use ladder if statement to solve the problem in the simple manner.

```
Syntax:- if(condition)
    {
    Statement 1;
    }
    Statement 2;
    }
    else if(condition 3)
    {
    statement 3;
    }
    ---------------
    ---------------
    else if(condition)
    {
```

```
Statement n;

}

else

{

Default statement;

}

Statement x;
```

In this statement 1st condition will be checked, if it is true the statement 1 will be executed, if the condition 1 is false the condition 2 will be checked. If it is true statement 2 will be executed. Otherwise further next condition will be checked and this process will be continue till the end of the condition.

e)switch case statement:- the switch case statement is a multiway branch statement. The tests whether expression matches one of the constant values. This switch case statement requires only one arguments which is checked with numbers of cases options.

Syntax:- switch (expression)

```
{

Case value 1: statement 1;

        Break;

Case value 2: statement 2;

        Break;

        ------

        ------

Case value n: statement n;

                Break;

Default: statement;

}
```

Un conditional branching statements:-

Iterative statements(looping statements):- iterative statements are used to repeat their execution of list of statements depends on the value of integer expression

(or)

A single statement (or) group of statements will be executed again and again in a program such type of processing is called loop.

"c" supports three types of iterative statements also known as looping statements.

They are a) while-loop

b) Do-while loop

c) For loop

a) while loop:- in while loop one (or) more statements are repeatedly executed. Until a particular condition is true.

Syntax:- while(condition)

```
{
 Block of statements;
 }
```

In the while loop first the condition is tested. If the condition is true. Then only the statement will be executed. Otherwise if the condition is false. The control will be jump to false statement.

b)do-while loop:- the do-while loop is similar to while loop. The only difference is that in a do-while loop the test condition is tested at the end of the loop.

Syntax:- do

```
{
 Block of statements;
 }
 While(condition);
```

c) for-loop:- for (initialization; test condition; increment/decrement)

```
{
Body of loop;
}
```

The execution for the "for loop" is as follows

1) Initialization of variable is done first
2) The value of variable is tested is tested is using test condition. If the condition is true the body of the loop is executed if the condition is false the loop is terminated.

3) When the body of loop is executed the control transfers back to the 1st statement for the last expression. That is increment/decrement.

After increment/decrement the value of the control again goes to the test condition. If the condition is true, the body of the loop is again executed. The process continuous until the test condition is false.

Ex program:- program to print 1 to N number using for loop.

```
#include <stdio.h>

Main ()

{

int  i, n;

clrscr();

printf (“enter n value”);

scanf (“%d”, &n);

printf (“\n 1 to n numbers are:”);

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

{

Printf (“\n%d”, i);

}

}
```

Nested for loop:- one for statement with in another for statement is called nested for loop.

Syntax:- for (initialization; test condition; increment/decrement)

```
{

for (initialization; test condition; increment/decrement)

{

Statement 1;

Statement 2;

-------

--------
Statement n;
}
Statement 1;
Statement 2;
---------
---------
Statement n;
}
```

break statement:- break is a keyword in "c" statement is used to terminate the loop i.e., the control comes out from the current loop. When over the break keyword is encountered. It is also called as loop terminator.

The break statement is widely used with for loop, while loop & do-while loop.

Syntax:- break;

Continue statement:- the continuous statement is opposite to break statement. It appears in the body of the loop. Whenever a continuous statement appears the rest of the statements in the loop are skipped. And it continuous with the next interation of current loop.

(or)

The continuous statement statement is exactly opposite to break statement. The continuous statement is used for continuing next iteration of next statement, when it occurs in the loop it doesn't terminate, but it skips the statements after this statement. It is useful when you want to continue the program without executing any part of the program.

Syntax:- continue;

goto statement:-

It doesn't require any condition goto is a keyword in 'c' this statement process control any where in the program. i.e., control is transferred to another part of program without testing any condition. The user has to define goto statement as follows.

Syntax:- Goto label;

return statement:-

It is a keyword in 'c' it is commonly used in user defined functions (or) sub functions to return a value, the program control transfers from sub function to main function.

Note:- 'C' language supports 4 statements to perform unconditional control transfers.

They are

1. return
2. goto
3. break
4. continue

these 4 statements are also called “jumping statements”.

FUNCTIONS

Function:- a function is a group of statements that together perform a specific tasks. A large program can be split into smaller segments. So that it can be efficiently solved.

The function are categorized into two types

1. user defined function
2. pre-defined function

the major difference between the user defined functions and pre-defined functions are not required to be written by user. While writing a program. Many program required to a particular group of unstructured accessed repeatedly. From the different place with in the program. This repeated instructions can be placed in a single function and can be accessed whenever necessary.

(or)

Function is a group of statements that carries out to specific task. Every c program starts with atleast one function. Which is main(). The main() calls another to share the work.

The c language supports two ways of functions.

1. Pre-defined functions(or) library functions.
2. User-defined functions

Library functions are pre-defined se of functions they task is limited.

Eg:- printf(), scanf(), sqrt()….etc.

These functions cannot modified.

User defined functions are define by user. According to our requirement are called user defined functions.

Advantages of function:-

1. It provides reusability. Because function calls many times.
2. Generally large programs can be divided into smaller programs.
3. It is to easy to modify.
4. It is easy to debug and find out the errors.

How function work:-

- Function can be define and called it takes some data from the calling function. And return a value to the called function. Whenever function is called control passes to the called function. And working of the calling function is complete control returns back the calling function and executes the next statement.
- The value of actual arguments passed by the calling function are received by the formal arguments of the called function.
- The function operates on formal arguments and sends back the result to the calling function.
- The return function performs the task. Every function consists of three parts

1. Function declaration
2. Function definition
3. Function call (or) calling function

Function declaration:-

Function declaration consists of three components such as return type, function name and aruguments being passed. The function declaration must be ends with a semi column(;)

Syntax:-

```
return type, function name larg1, arg ……arg n);

int sum(int, int);

float multi();
```

function def:- the function definition consists of two components. The first line and the body of the function. The function definition is same as function declaration except that is does not end with a semi column(;)

syntax:- return type function (arg 1,arg 2…..arg n)

```
{

function body;

return(expression);

}
```

The body of the function may contains local variables, statements, expressions and also a return type.

Function cal (or) calling function:- the function call statement invokes the function. When a function is invoked the compiler jumps to the called function to execute the statements that are a part of function. Once the called function is executed, the program control passes back to the calling function.

Syntax:- function name(variable 1, variable 2….);

Example:- main()

```
{
 …..
 …..
 abc (x,y,z);
 ……
 ……
 }
abc(i,j,k)
{
 ……
 ……
  return();
  }
```

Example program:- addition of two numbers

```
#include<stdio.h>
main()
{
int x=2,y=5,z;
z=sum(x,y);
printf(“z=%d”, z);
}
sum(i, j)
{
return(i+j);
}
```

Output:- z+7

Types of functions:-

Depending upon the arguments return value sends back to the calling functions based on this the functions are divided into four types.

1. A function without arguments and without return a value
2. A function with arguments and with return value
3. A function with arguments and without return values
4. A function without aruguments and with return values

1. A function without arguments and without return value:-

Syntax:-

Calling function	**analysis**	Called function

void fname();
statements;
{
f name();
statements();
}
	No arguments are passed
	No values are passed
		void fname()
		{
		Statements;
		}

- ✓ In this function neither the data is passed through the calling function nor the data is sent back from the called function.
- ✓ There is no data transfer between calling & the called function. The function is only executed & nothing is return (or) obtain
- ✓ These functions are independent. They read data values & print results in the same blak.
- ✓ Such functions may be useful to print same messages, draw a line or split the lines etc.

Example:-

```
#include<stdio.h>

void main()

{

void msg();/* calling function*/

msg();

}

void msg()/* called function */

{

Printf("welcome");

}
```

2. **A function with arguments and with return value:-**
   Syntax:-

Calling function	**analysis**	**Called function**

void main()
{
int f name (actual arguments list);
f name(actual arguments list);
}
	Arguments are passed.
	Values are sent back
		int f name(formal arguments list)
		{
		Statements;
		Return value;
		}

- ✓ In such functions the copy of actual arguments is passed to formal arguments.
- ✓ The return value is sent back to the called function.
- ✓ In such functions the data is transferred between calling and the called function i.e., communication between is made.

Example:-

```
/* addition of two numbers */

#include<stdio.h>

void main()

{

int add(int,int);

int a,b,c;

clrscr();

printf("enter two values");

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

c=add(a,b);

printf("sum=%d",c);

getch();

}

int add(int x,int y)

{

int z;

z=x+y;

return z;
```

}

3. A function with arguments and without return value:-
Syntax:-

Calling function	analysis	Called function

void main()
{
void f name(actual arg datatype list1);
statements;
f nam(actual arglist 1);
statements;
}
u
c
	Arguments are passed
	No values are sent back
		void f name(formal arglist)
		{
		Statements;
		}

- ✓ h functions the arguments are passed through the calling function
- ✓ The calling function operates the values but no result is sent back.
- ✓ Such functions are dependent on the calling function there is no gain to the main() function.

Example:-

```
/* write a program to given number is even or odd*/

#include<stdio.h>

main()

{

void even odd(int);

int a;

clrscr();

printf("enter one value:");

scanf("%d",&a);

even odd(a);

getch();

}

void even odd(int n)

{

if(n%2==0)
printf("given number is even:");
else
printf("given number is odd");
}
```

Output:-

```
Enter one value:-4
Given number is even
```

4. A function without arguments and with return value:-

Syntax:-

Calling function	**analysis**	**Called function**

void main()
{
int f name();
statements;
f name;
statements;
}
	No arguments are passed
	Values are sent back
		int f name()
		{
		Statements;
		return value;
		}

1. In such type of function no arguments are passed through the main function but the called function returns the value
2. The called function is independent. It reads the value input from the keyboard or generates from the initialization and returns the value
3. Here both the calling & called function are communicated with each other.

Example program :

```
/* Sum of three numbers*/
  #include<stdio.h.>
   void main()
      {
         int sum();
         int s;
         clrscr();
         s=sum();
          printf("Sum =:%d",s);
     }
         sum()
        {
           int x,y,z;
           printf("enter three values   :");
           scanf("%d%d%d",&x,&y,&z);
           return(x+y+z);
```

```
                  }
            Output:
                    Enter three values    :1 2 3
                     Sum=6
```

Explain parameters passing methods with an example program;

There are two ways to pass arguments or parameters of functions

1). Call by value

Call by value

Call by reference

Call by value :

Call by value in which values of variables are passed by the calling function to the called function.

Example program ;

```
 /* Swapping of two numbers*/

 #include<stdio.h>

main()

     {

         void  swap(int,int);

          int a,b;

          clrscr();

          printf("enter a,b values    :");

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

          printf("\nbefore swapping a=%d\t b=%d\t",a,b);

          swap(a,b);

          getch();

   }

         void swap(int a,int b)

          {

              int temp;
```

```
        temp=a;
        a=b;
        b=temp;
       printf("\n After swapping in function a=%d\t,b=%d\t",a,b);
   }
```

Output :

```
     Enter a,b values    : 2 3
     Before swapping a=2 b=3
     After swapping in function a=3,b=2
```

Call by reference :-

Call by reference in which address of variable passed by the calling function to called function.

Example:-

```
/* swapping of two numbers */
#include<stdio.h>
void swap(int *p, int *q);
main()
{
int a,b;
clrscr();
printf("enter a,b values:");
scanf("%d%d', &a,&b);
printf("\n after swapping a=%d\t b=%d\t",a,b);
getch();
}
void swap(int *m, int *n)
{
```

```
int temp;

temp=*m;

*m=*n;

*n=temp;

printf("\n after swapping in main function a=%d\t b=%d\t",a,b);}
```

recursion:-

- 'c' language supports recursive features i.e., function is called repeatively by itself. The recursion can be directly (or) indirectly.
- The direct recursion function called itself. In indirect recursion function call another function.
  Then the called function calls calling function.

Example:-

```
#include<stdio.h>

void main()

{

int fact(int)

clrscr();

printf("enter n value");

scanf("%d", &n);

f=fact(n);

printf("\n factorial of %d is %d", n, f);

getch();

}

int fact(int n)

if(n==0)

{

return 1;

else

f=n*fact(n-1);
return f;

}
```

Output:-

```
Enter n value:2

The factorial of 2 is 2
```

Storage classes:-

- The storage class determine the part of memory when the variable would be stored.
- Each variable has a storage class which decides scope, visibility and life time of that variable.
- A variable declared inside of the main function is called "local variables".
- A variable declared outside of any function is called "global variable".
- There are 4 types of storage classes in 'c'
  1) Automatic storage class
  2) External storage class
  3) Static storage class
  4) Register storage class

1. Automatic storage class:-

Automatic variables are define inside a function. A variable declared inside the function without use storage class, name by default is an automatic variable. Automatic variables is also called as local variable. It is declared to use "auto keyword".

Ex:-

```
void main()

{

auto int num;

}
```

2. External storage class:-

The variable are declared outside of the function the external variable is also called as "global variables" this variables are used any function in the programme.

1.incase external and auto variables are declared with the same name in the programme. The first priority is given to the auto variables.

2. it is declared to use extern keyword.

Ex:-

```
void main()

{

extern int a=7;

}
```

3. **Static storage class:-**
   The static variable may be any internal or external type. It is depending upon where it is depending upon where it is declared. If it is declared outside of the function. It will be static variable (or) global variable. Incase it will be declared inside of the function. It will be auto variable or local variable. Optional (when variable is declared as static its garbage value is removed and initialize to null value). It is declared to use static keyword.
   Ex:-
   ```
    main()
   {
   int x;
   static int y;
   printf("x=%d  and y=%d",x,y);
   }
   ```
   Output:-
   ```
   X=1 2 3 9  and   y=0
    ↓                  ↓
   Garbage value   null value
   ```
4. **Register storage class:-**
   A variable is declared to use "register" keyword is called register storage class variable. When a variable is declared using register keyword. Then the value of that variable is directly stored the c.p.u register.
   Ex:-
   ```
   void main()
   {
   register int;
   clrscr();
   for(i=0;i<10;i++)
   printf("%d", i);
   getch();
   }
   ```

Unit-III

ARRAYS

Definition:- Array is a collection of consecutive memory locations of same data type and can be collectively referred with a single name and each location can be accessed with its index member or subscript.

Arrays are used to reduce confused and length of the program. Each element of an array can be accessed with the array name followed by a subscript which includes in square brackets .The subscript provided from zero(0) for the first element ,increased by one to the next element.

Initialization of arrays:- an elements of array can be initialized at the time of declaration. When an array is initialized we need to provide a value for every element in the array

Syntax:-Data type Array-name [size]={list of values};

Example:-int a[5]={1,2,3,4,5};

Types of arrays:- There are three types of ARRAYS, they are

1. one – dimensional array
2. two – dimensional array
3. multi – dimensional array

1.ONE – DIMENSIONAL ARRAY :

A list of items can be given one variable name using only one subscript and such variable is called a single subscripted variable or one dimensional array .

Declaration Of One Dimensional Array :

Like any other variable, array must be declared before they are used .

Syntax :- Data type array name [size] ;

Example:- int a[5];

Float c[10];

Initialization of one dimensional array :

The array can be initialized at the time of declaration .

Syntax: - Data type array name [size] = {list of values};

Example:- int a[5]={10,20,30,40,50};

a[0] stores 10

a[1]stores 20

a [2]stores 30

a [3]stores 40

a[4]stores 50

memory allocation diagram:-

10	20	30	40	50
a[0]	a[1]	a[2]	a[3]	a[4]

Example:-1

```
/*one –dimensional array*/

#include <stdio.h>

main()

{

int  a[5]={10,20,30,40,50};

    clrscr( );

for(int i=0;i<,5;i++)

{

   printf("value of a[%d] is %d\n”,i,a[i] ");

}

}

getch( );

}
```

OUTPUT   :

```
                    Value of a[0] is 10

                    Value of a[1] is 20

                    Value of a[2] is 30

                    Value of a[3] is 40

                    Value of a[4] is 50
```

Example:-2

```
/*write a c-program me to read and print the an array elements*/

#include<stdio.h>
```

```
#include<stdio.h>

Void main()

{

Int a[10],n,I;

Clrscr();

Printf("\n enter the size of an array:");

Scanf("%d",&n);

Printf("\n enter the elements in to an array");

for(i=0;i<n;i++)

{

Scanf("%d"&a[i]);

}

Printf("\n the array elements are:");

for(i=0;i<n-1;i++)

{

Printf("%3d",a[i]);

}

getch();

}
```

2. Two – dimensional array :

These arrays are called double dimensional array. Another name of two - dimensional array is tabular or rectangular array. These array are in row and column form. So these are called row column array or square array .

These arrays have two subscripts. We can the double dimensional array in the matrix form so these are also called matrix array.

Declaration of two-dimensional array:-

Syntax : Data type array name [rows][column];

Ex : int a[2][3];

INITIALIZATION OF TWO-DIMENSIONAL ARRAY :

The two dimensional array can be initialized at the time of declaration.

Syntax:-Data type variable name[row size][column size] = {list of values};

Example:- int a[2][3]={5,7,102,13};

Here the value assigned to the array is as follows.

a[0][0]=5	a[1][0]=12
a[0][1]=7	a[1][1]=1
a[0][2]=10	a[1][2]=3

Example-1:- /* Two-dimensional array*/

```
#include<stdio.h>
main( )
{
  int   i,j;
int a[2][3]={10,20,30,40,50};
  clrscr( );
    for(i=0;i<2;i++)
  {
     for(j=0;j<3;j++)
    {
Printf("value of a[%d][%d] is %d\n",i,j,a[i][j]);
}
}
}
```

Example:2

```
/*write a c programme to read and print a two dimensional array*/
#include<stdio.h>
```

```
#include<stdio.h>

Void main()

{

Int a[5][5],i,j,r,c;

Clrscr();

Printf("enter the order of matrix:");

Scanf("%d%d",&r,&c);

Printf("\n enter the  elements in to the array:");

for(i=0;i<=r-1;i++)

{

for(j=0;j<=c-1;j++)

{

Scanf("%d",&a[i][j]);

}

}

Printf("\n the elements of the matrix are \n\n");

for(i=0;i<r-1;i++)

{

for(j=0;j<=c-1;j++)

{

Printf("%d3d",a[i][j]);

}

Printf("\n\n");

}

getch();

}
```

Operations on two-dimensional array:-

The operation on two-dimensional array is

1. Transpose

2. Addition(or)sum

3.Multiplication(or)product

4.Substraction(or)deference

1.Transpose:- Transpose of a m*n matrix of A is given as a n*m matrix of B

$$A_{ij}=B_{ji}$$

2. Addition(or)sum:- Two matrixes can be added together by storing the result in third matrix two matrixes are said to be compares when they have same number of rows and coloumns

Two elements of matrixes can be added by as follows

$$C_{ij}=A_{ij}+B_{ij}$$

3.Multiplication(or)product:- two matrixes can be multiplied with each other. If the number of columns in first matrix is equal to the number of rows in second matrix

m*n matrix of A can be multiplied with p*q matrix of B ,if n=p

elements of matrixes are multiplied by as follows

$C_{i,j}=C_{i,j}+A_{i,k}*B_{k, j}$

4.Substraction(or)deference:- two matrixes can be substracted from each other by storing the result in third matrix

Two matrixes are said to be compared, when they have same number of rows and columns

Elements of matrixes can be substracted as follows

$$C_{i,j}=A_{i,j}-B_{i,j}$$

3.Multi – dimensional array :

A multi-dimensional array is an array of arrays. It is specified to use indexes a multi-dimensional array

m1*m2*m3*------------mn is a collection of m1*m2*m3*-----------mn elements

A particular element is specified in using subscripts as A[I1][I2][I3]------[In]Mn

String

A string is an array of characters. A string in c defined as any group of characters enclosed between double quotation marks. The string can be of any length, the end of the string is marked with the single character(‘\0’) the null character.

The strings are actually one dimensional array of characters terminated by null character. The character arrays are declared in c in a similar mannar of numeric array

Declaration:-

SYNTAX : char string name[size];

Strings can be initialized at the time of declaration at the same time initializations at the time of declaration the string can be initialized in any one of the following manner.

Ex:- char name[10] = “Krishna”

Char name[10] = {‘k’,’r’,’i’,’s’,’h’,’n’,’a,};

“c” library supports a large number of string handling functions. Following are the most commonly used **STRING HANDLING FUNCTIONS**.

E.g.:- #include <stdio.h>

```
main ()

{

Char greeting[10]={‘v’,’e’,’n’,’k’,’a’,’t’,};

printf(“greeting message=%s\n”,greeting);

}
```

String Handling Functions.

The c library supports a large number of string handling functions that can be used to manipulate the string in many ways you must include the string header file in your program

String handling functions are:

1.strcat()→concatenation

2.strcpy()→copy

3.strlen()→length

4.strcomp()→compare

5.strrev()→reverse

6.strupr()→upper case

7.strlwr()→lower case

1. strcat( ) **:**

The strcat( ) function is used to joins two strings .

SYNTAX : strcat(str1,str2);

str1 and tr2 are two strings. When the function strcat() is executed str2 is appended to str1.

Ex : Write a program to adding 2 strings using string handling function

```
#include<stdio.h>

#include<string.h>

main( )

{

  char    A[10],B[10];

  clrscr( );

  printf("Enter any strings into A :  ");

  scanf("%s",A);

  printf("\n\nEnter any string into B: ");

  scanf("%s",B);

  strcat(A,B);

  printf("\n\nResult : %s",A);

getch( );

}
```

OUTPUT :

```
                Enter  any  string  into  A : sudha

                Enter  any  string  into  B : kar

                 Result :  sudhakar
```

2. strcpy( ) **:**

This function is used to copy one string into another string it accepts two strings as arguments.

SYNTAX : strcpy(target,source);

If first argument is generally an identifier, that represents the strings. The second argument can be a string constant. The function copies the string of source to target.

Ex : Write a program to copy any string from one array to another array by using string handling function

```
#include<stdio.h>
main( )
{
  char   A[50],B[50];
  clrscr( );
  printf("Enter any string for copy:  ");
  gets(A);
  strcpy(B,A);
  puts("\n\nCoping string is  \n\n");
  puts(B);
getch( );
}
```

OUTPUT :

```
                    Enter any string for copy : BHASKAR
                        Coping  string  is  BHASKAR .
```

3.strlen( ) **:**

This function counts and returns the number of characters in a string.

SYNTAX : variable= strlen(string);

Where variable is an integer variable, which receives the value of the length of the string the argument is a string constant.

Ex : Write a program to find string length by using string handling function

```
#include<stdio.h>
main( )
{
   char   a[50];
   int  count;
   clrscr( );
   printf("Enter any string for counting character:");
   gets(a);
   count=strlen(a);
   printf("\n\number  of  characters=%d",count);
getch( );
}
```

OUTPUT :

```
Enter  any  string  for counting  character   :    ram charan
                    Number  of   characters  = 10 .
```

4. strcmp( ) :

This function compares two strings and returns a integer value.

SYNTAX : strcmp(str1,str2);

It returns zero if the strings are equal, greater than zero if string 1 is bigger than string 2 and less than zero if string1 is less than string2.

Ex : Write a program to compare two strings

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

```
{
   char   A[10],B[10];
   int d;
   clrscr( );
   printf("Enter string of A lower case or upper case:");
   gets(A);
   printf("\n\nEnter string of B lower case or upper case:");
   gets(B);
   d=strcmp(A,B);
   if(d==0)
     printf("\n\n2 strings are equal");
   else
     printf("\n\n2 strings are not equal");
getch( );
}
```

OUTPUT :

```
                    Enter  string  of  A in lower case or upper case  : string
                    Enter  string  of  B in lower case or upper case  : string
                            2  strings  are  equal.


                    Enter  string  of  A in lower case or upper case  : string
                    Enter string of B in lower case or upper case : STRING
```

Ex : Write a program to check whether the given string palindrome or not using string handling function

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

```
main( )
{
  char   a[50],b[50];
  int   c;
  clrscr( );
  printf("Enter  any  string  :  ");
  gets(a);
  strcpy(b,a);
  strrev(a);
  c=strcmp(a,b);
  if(c==0)
  {
    printf("\nReverse  string  is  :  %s",a);
    printf("\nTwo  strings  are  equal.");
  }
  else
  {
    printf("\nReverse  string  is  :  %s",a);
    printf("\nTwo  strings  are  not  equal.");
getch( );
}
```

OUTPUT :

```
           Enter  any  string  :   liril
              Reverse  string  is  :  liril
              Two  strings  are  equal.


           Enter  any  string  :   reverse
```

Reverse string is **:** esrever

Two strings are equal.

5.Strrev():- The purpose of this function is two reverse a string this function takes string variable as a single argument here the first character becomes last and the last character becomes first in the string

syntax:-strrev(string);

ex:- char str1[10]=”sri”;

char str2[10]=”ramya”;

Printf(“reverse string of str1=%s”,strrev(s1));

o/p:- reverse string of s1=irs

6.Strupr():- The purpose of this function is to convertinto upper case this function converts all the character of a string from lower case to upper case

syntax:-strupr(string);

ex:- char str1[10]=”sai”;

char str2[10]=”suma”;

Printf(“s1 is converted into upper case :%s”,strupr(s1));

o/p:- s1 is converted into upper case :SAI

7.Strlwr:- The purpose of this function is to convert string into lower case this function converts all the character of a string from upper case to lower case

syntax:-strlwr(string);

ex:- char str1[10]=”SAI”;

char str2[10]=”suma”;

Printf(“s1 is converted into lower case :%s”,strupr(s1));

o/p:- s1 is converted into lower case :sai

operations of string:- the different operations that are performed on strings are

***length**:- the number of characters in the string constitutes the length of the string

Ex:- LENGTH(“c programming is fun”);

Will return “c” programme to “20” (including blank spaces)

***converting characters of a string into uppercase**:-

The ASCII code for capital ‘A’ to’Z’ is from ‘65’ to ‘91’ and the ASCII code for ‘a’ to’z’ ranges from ‘97’ to ‘123’

To convert a lower case character to upper case we need to subtract 32 from the ASCII value of the character

Ex:-hello

HELLO

***converting characters of a string into lowercase**:-

The ASCII code for capital ‘A’ to’Z’ is from ‘65’ to ‘91’ and the ASCII code for ‘a’ to’z’ ranges from ‘97’ to ‘123’

To convert upper case character to lower case we need to add 32 from the ASCII value of the character

Ex:- The ASCII code for capital ‘A’ to’Z’ is from ‘65’ to ‘91’ and the ASCII code for ‘a’ to’z’ ranges from ‘97’ to ‘123’

To convert a lower case character to upper case we need to subtract 32 from the ASCII value of the character

Ex:-HELLO

hello

***concatenating two strings two form a new string**:-

It s1 and s2 are two strings then concatenation operation produces a string which contains characters of s1 followed by s1 characters of s2

Ex:- s1-hai,s2-hoe are you

→hai how are you

***comparing two strings:-** To compare two strings each and every character is compared from both the strings if all the character are same then the two strings are equal

Ex:-S1-hello

S2-hello→strings are equal

***Reverse a string:-** if s1=”hello” the reverse of s1=”olleh”. To reverse a string we need to swap the first character with the last, second character with the last second character and so on----

Ex:-how r u

U r who

POINTER

A pointer is a variable, which contains the address of another variable. A pointer provides an indirect way of accessing the value of a data item.

Declaring a pointer variable: Declaration of pointer variable is similar to the declaration of a common variable. A pointer is a variable should be declared before they are used. The general syntax used for the declaration of pointer is as

Syntax:-Data-type *pointer-variable;

Where data type may be integer (int), real(float),character(char) or double. Also here (asteric sign) means it is pointer operator and pointer variable is any variable linked with'*' sign.

it is pointer operator and pointer variable is any variable linked with '*' sign.

Write a program to print address and value of variable

```
#include  <stdio.h>

Main()

{

Clrscr ();

Printf("enter x value:");

Scanf("%d",&x);

P=&x;

Printf("\n address of x=%u",&x);

Printf("\n value of x=%d",x);

Printf("\n address of x=%d,*p);

getch();

}
```

```
o/p:- enter x value=6

address of x=65500

value of x=6
```

```
address of x=65500

value of x=6
```

ADVANTAGES AND DISADVANTAGES OF POINTERS:-

ADVANTAGES:-

1. Pointers increase the execution speed of the C-Program and are more efficient.

2. Pointer access the memory elements very easily

1. 3. Pointer reduces the length and complexity of the program.

4. By using pointer, we can declare lesser number of variables in memory.

5.Pointers access the memory elements very easily

6.we can pass arguments to functions by reference

Disadvantages:-

1.pointers are slower than normal variables

2.if used incorrectly pointer leads to bugs(errors)

3.an erroneous input leads to erroneous output

Passing arguments to functions using pointers:-

Using call-by value method it is impossible to modify the actual parameters in the call when you pass then to a function

Pointers provide a mechanism to modify data declared in one function using the code written in another function we must pass the address of variables,we want to change

To use pointers for passing arguments to a function we must do the following

1.declare the function variables as pointers

2.use the de-reference pointer in the function body

3.pass the address as the actual arguments when the function is called

Ex:- swapping of two numbers by using call-by reference

```
#include<stdio.h>

Main()
```

```
{

Void swap(int *,int*)

Int a,b;

Clrscr();

Printf(“enter a,b values:”);

Scanf(“%d%d”,&a,&b);

Swap(&a,&b);

Printf(“\n after swapping in main a=%d\t b=%d\t,a,b);

getch();

}

Void swap(int *P,int *q)

{

int temp;

temp=*p;

*p=*q;

*q=temp;

Printf(“\n after swapping in function a=%d\t b=%d\t,*p,*q);
```

```
o/p:-enter  a,b values:10 20

before swapping a=10 b=20

after swapping in function a=20 b=10

after swapping in main a=10 b=20
```

array of pointers:- an array of pointers can be declared as int*p[10]

the above statement can be declared have an array of 10 pointers to an integer variable.for example int p=1,q=2,r=3,s=4,t=5

ptr[0]=&p;

```
ptr[1]=&q;

ptr[2]=&r;

ptr[3]=&s;

ptr[4]=&t;
```

Ex:- #include<stdio.h>

```
main()

{

Int *p,i=0;

Int a[5]={1,2,3,4,5};

Clrscr();

P=&a[0];

Printf("pointer of array elements are");

While(i<5)

{

Printf("%d",&p);

P++;

i++;

}

getch();

}
```

```
o/p:-pointer of array elements  are 1,2,3,4,5;
```

Function pointers:-every function has an address function pointers are pointer variables that point to the address of a that point function pointers can be declare assign values and used to access the functions

In order to declare a pointer to a function the name of function must be enclosed between paranthesis and an satiric(*) symbol is inserted before the name

Syntax:- the sysntax of declaring a function pointer is retuen type(*function pointer name)(arg list);

Ex:-int (*func)(int a,float b);

Address Operator (&): - The address of the memory variable can be accused by using address operator (&). Now suppose q is a variable having value 280 and this variable is stored in the memory address at 6000. Then to represent address of the memory, we apply a variable p, which is pointer or memory variable. We can store address of q variable in p pointer variable as by using & operator.

Ex: write a program to print address and value of variable

```
# include<stdio.h>
Main()
{
Int x,*p;
Clrscr();
Printf("enter x values:");
Scanf("%d",&x);
P=&x;
 Printf("\n address of x=%u",&x);
 Printf("\n value of x=%d",x);
 Printf("\n address of x=%d,*p);
 getch();
 }
 o/p:- enter x value=6
```

```
address of x=65500

value of x=6

address of x=65500
```

void pointer:- void pointer(or)generic pointer is a special type of poiner that can be pointed to object to any data type it is declared like a normal pointer using void keyword

Null pointer:- Null pointer are used in situations where one of the pointer in the program points to different location at different types

Wild pointer:-A pointer which is not initialized with any address is called wild pointer

Dangling Pointer:- Dangling pointers are arrised when an object is deleted without modifying the value of pointers

<u>STRUCTURES</u>

In c a structure is a collection of variables that are referenced under one name, providing a convenient means of keeping related information together.

Structure declaration:- In this declaration struct is a keyword structure name is the name of the structure and member 1 through members are individual member declarations. The individual members can be variables pointers arrays or other structures. The structure is terminated with a semi colon. The structure name can be used to declare structure variables of its type.

The structure variable and dot operator accesses the members of structure.

<u>SYNTAX</u> :

```
struct   structure_name

{

       data type var-name;

        data type var-name;

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

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

       };
```

For example;- to declare a structure for a student with related information roll no,name,fees etc--,is declared as exactly in the example

```
Ex:- struct student

{

int rno;

Char name[20];

float fees;

};
```

Initialization of structure:- A structure can be initialized in some way as other data types are initialized initializing a structure means assigning some constants to the member of structure

The initializes are enclosed in braces() and are separated by commas(,).

Syntax:- struct struct-name

```
{

data type member-name1;

data type member-name1;

data type member-name1;

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

};

Struct struct-name

Struct var={constant1,constant2--------};
```

```
Ex:- struct student

{

Int rno;

Char name[20];

Char course[20];

Float fees;
```

};

Struct student stud1={01,”ravi”,”bsc”,4500};

Accessing the members of a structure:-Each member of a structure can be used just like a normal variable. A structure member variable is generally accessed to using a “.” Dot operator.

syntax:-struct-var.member-name

the dot(.)operator is used to select a particular member of the structure

eg:- std1.rno=01;

std1.name=”rahul”;

std1.course=”bsc”;

std1.fees=4500;

Ex:-**Write a c program to read and display student information using structures**

#include<stdio.h>

#include<stdio.h>

Int main()

{

Struct student

{

Int roll-no;

Char name[30];

float fees;

char dob[30];

};

Struct student std1;

Clrscr();

Printf(“\n enter roll-no:”);

```
Scanf("%d",&std1.roll-no);

Printf("\n enter name:");

Scanf("%d",&std1.name);

Printf("\n enter fees:");

Scanf("%f",&std1.fees);

Printf("\n enter date of birth:");

Scanf("%d",&std1.dob);

Printf("\n*********student details*********");

printf("\n rollno=%d",std1.rollno);

printf("\n name=%s",std1.name);

printf("\n fees=%f,std1.fees);

printf("\n dob=%d,std1.dob);

getch();

return 0;
}
```

o/p:-

```
enter roll-no:1

enter name:ravi

enter fees:4500

enter Dob:17-11-1998

******student details*********

Roll-no=1

Name=ravi

Fees=4500

Dob=17-11-1998
```

Nested structure:-A structure can be placed with in another structure. A structure that contains another structure as its member is called nested structure

The easier and clear way is to declare a nested structure separately and than group them in a high level structure (from lower level to higher level)the nesting must be done from inside out.

Ex:-Write a c program to read and display student information using structure with in a structure

```
#include<stdio.h>

#include<stdio.h>

int main()

{

Struct DOB

{

int day;

int month;

int year;

Struct student

{

Int roll-no;

Char name[30];

float fees;

char dob[30];

};

Struct student std1;

Clrscr();

Printf("\n enter roll-no:");
Scanf("%d",&std1.roll-no);

Printf("\n enter name:");

Scanf("%d",&std1.name);

Printf("\n enter fees:");

Scanf("%f",&std1.fees);

Printf("\n enter date of birth:");

Scanf("%d",&std1.dob);

Printf("\n*********student details*********");

printf("\n rollno=%d",std1.rollno);

printf("\n name=%s",std1.name);

printf("\n fees=%f,std1.fees);

printf("\n dob=%d,std1.dob);

getch();

return 0;
}
```

o/p:-

```
enter roll-no:2

enter name:venkat

enter fees:5000

enter Dob:1-12-1998

******student details*********

Roll-no=2

Name=venkat

Fees=5000

Dob=1-12-1998
```

Self referential structure:- The self referential structures are includes atleat one member as a pointer to the same structure is known as self referential structure it can be linked together to form useful data structures such as lists,queues,stacks and trees. It is terminated with a null pointer(0)

Syntax:- struct struct-name

{

Type member;type2 member2;

Struct struct-name*next;

};

E.g.:-struct node

{

Int data;

Struct node*next;

};

<u>UNIONS</u>

Union: - A union is a special data type available in c.It is a collection of variables of different data types in the same memory location. In structure each member has own storage location. Unions provide an efficient way of using the same memory location for multiple purpose A union is declared using union keyword

Declaring a union:- The syntax for declaring a union is same as that of declaring a structure using the keyboard struct and the union is used union keyword

<u>SYNTAX</u> : Union union-name

{

Data type var-name;

Data type var-name;

```
                ----------------------
                 ----------------------
                };
```

Ex:-write a c program to read and display student information using unions

```
#include<stdio.h>
#include<stdio.h>
int main()
{
Union  student
{
Int roll-no;
Char name[30];
float fees;
char dob[30];
};
Union  student std1;
Clrscr();
Printf("\n enter roll-no:");
Scanf("%d",&std1.roll-no);
Printf ("\n enter name:");
Scanf ("%d",&std1.name);
Printf ("\n enter fees:");
Scanf ("%f",&std1.fees);
Printf ("\n enter date of birth :");
Scanf ("%d",&std1.dob);
Printf("\n*********student details*********");
printf("\n rollno=%d",std1.rollno);
printf("\n name=%s",std1.name);

printf("\n fees=%f,std1.fees);

printf("\n dob=%d,std1.dob);

getch();

return 0;
}
```

o/p:-

```
enter roll-no:1

enter name: Ravi

enter fees:4500

enter DOB: 17-11-1998

******student details*********

Roll-no=1

Name=Ravi

Fees=4500

Dob=17-11-1998
```

Difference between structure and union

STRUCTURE	UNION
1.Every member has its own memory space	1.All member use the same memory space
2.key word struct is used	2. Key word union is used.
3.All members may be initialized	3. Only its first member location is possible.
4. Different Interpretations of the same memory location are not possible.	4. Different Interpretations of the same memory location are possible.
5.Consume memory space compare to union	5.Conservation of memory is possible
6.A structure allocates the total size of all elements in it	6.A union only allocate as much memory as its largest element requires

Difference between array and pointers:-

ARRAYS	POINTERS
1.An array is a collection of similar data types 2.syntax:- Data type array name[size]; 3.example:- int a [10]; 4.Array can be initialized at definition 5.They are static in nature. Once memory is allocated, it can be resized 6.The assemble code of array is different than pointer	1.A pointer is a variable which contains the address of another variable 2. syntax:- Data type*pointer variable; 3. example:-int*x; 4. pointer can be initialized at definition 5.pointer is dynamic in nature the memory allocation can be resized 6. The assemble code of pointer is different than array

Dynamic memory allocation:-

i. The process of allocating memory at runtime is known as dynamic memory allocation
ii. With the static memory allocation some amount of memory is unused. This unused memory is actually empty. But it filled with garbage values
iii. To avoid this problem DMA technique was introduced
iv. DMA allows to allocate memory at run time. Hence there will be no wastage of memory
v. We can achieve DMA with following functions and which are defined as<alloch.h> or <stdio.h>header files

1. malloc()
2. calloc()
3. Realloc()
4. free()

1. malloc ():-

- Malloc allocates requested size of bytes and return void pointer to the first byte of the allocated space
- Malloc() allocates single block of requested memory
- Syntax:- declare a pointer
- Pointer name=(cast type*)malloc(byte size);

2.calloc():-

- Calloc () allocates multiple blocks of requested memory. Each of the same size and returns starting address of the memory to the pointer variable. This is widely used in array.
- Syntax:-declare a pointer
- Ptr=(int*)calloc(n,2);

3.Realloc():-

- This Realloc() is used to make changes in the memory location(i.e.)increase or decrease is already allocated by malloc()or calloc()
- This realloc reallocates the memory
- Syntax:-declare a pointer
- Pointer variable=(cast type*)realloc(pointer name, new size);

4.free():-

- It is essential to clean up memory at the end of the program
- In the memory location filled with the garbage values so it is very much required to clean up
- Syntax:-void tree(void*back);

UNIT-5

FILES

definition:-

a file is a collection of data stored on a secondary storage device such as hand disk.

1. Files are mega data structure in information processing.
2. By using files, we need to perform some input and output operations i.e., how data is read from a file or write a file.

Reading data from files:-

C provides the following set of functions to read a data from a file

1. fscanf()
2. fgets()
3. fgetc()
4. fread()

1. **fscanf():** the fscanf is used to read formatted data from a stream

syntax:-

int fscanf(file *stream, const, char *format….);

the fscanf() function is used to read data from the stream and store them.

According to the parameter format into the locations pointed by additional arguments.

2. **fgets**:- this function stands for 'file gets strings". This function used to get a string from a stream.

Syntax:-

char *fgets(char *str, int size, file *stream);

This function reads at most the number of characters specified by size from the given stream and stores them in string(str)

3. **fgetc():-** this function return the next character from the string and end of the file. If the end of the file reached or if there is an error.

Syntax:-

int fgetc(file *stream);

fgetc() function reads a single character from the current position of a file.

4. **fread():** this function is used to read data from a file

syntax:-

int fread(void 8str,size-t size, size-t num,*file *stream);

this functions reads number of objects are placed them in to the array pointed by the string(str).

Writing data to files:-

'c' provides set of function to write data to a file

1) fprintf()
2) fputs()
3) fputc()
4) fwrite()

1. **fprintf():-** this function is used to write formatted output to stream.

Syntax:-

int fprintf(file *stream, const, char*format….);

this function writes data i.e., formatted as specified by the format argument to the specified stream.

2. **fputs():-** the fputs () is used to write a single line to a file.

Syntax:-

int fputs(const char, file * stream);

this function writes the string pointed to str to the stream pointed by the stream.

3. **fputc**:- the fputc() is the opposite of fget() and used to write a character the stream.

Syntax:-

int fput (int c, FILE *stream);

This function will write in byte specified by ‘C’ to the output stream pointed by them.

4. **fwrite():-** This function is used to write data to a file.

Syntax:-

int fwrite(const void * str, size-t-size, size-t, cons, file * stream);

This function will write objects of size specified by size from the array pointed(ptr) to the stream pointed by the stream.

Types of files:

Files are basically classified into three types:

1) Sequential files (Text files)
2) Random files (direct files)
3) Binary files (indirect level files)

Sequential File(Text files):

1) In this organization, the records will be arranged one after another.
2) It consists of alphabets and digits. So, it is called as text files.
3) For ex: If you want a record from a sequential file then, we will search from beginning to end. It is the time consuming process. Generally it is used to store small amount of data.

Sequential Operations:

1) Defining and naming a file

2) Opening a file
3) Reading a file
4) Writing a file
5) Closing a file

Random Files(Direct Files):

1) It consists of fixed length of records. Each record can be identified by a unique number.
2) It is an efficient method as compared to sequential files.
3) These files are also called as direct files.

Random file Operations:

1) Defining and naming a file
2) Opening a file
3) Reading a file
4) Writing a file
5) Closing a file

Binary files(Indirect Files):

1) It may contain graphical (or) image (or) compiled codes(machine language).
2) This file can be accessed byte by byte. So, data will be stored under 0's and 1's(zero's and one's)
3) It is the fastest and efficient method for accessing data.

Binary File Operations:

1) Defining and naming a file
2) Opening a file
3) Reading a file
4) Writing a file
5) Closing a file

Explain basic file operations :

The following operations perform on files,they are

1). Naming file

2). Defining a file

3). Opening a file

4). Reading a file

5). Writing a file

6). Closing a file

1). Naming a file:

A file name consists of two parts namely

a). Primary name

b). Secondary name

Primary name is mandatory or conform. Secondary name is optional.

Example :

$$\begin{tikzcd} \text{Student.txt} \arrow[r] \arrow[d] & \text{Secondary name} \\ \text{Primary name} & \end{tikzcd}$$

2). Defining a file

While working with files we must create a file pointer to the file.The file pointer provides communication between program and operating system.

Example :

```
FILE *fp;
```

Where “FILE” is a new data type,” fp” is file pointer to the file to file data type.It creates a link between program and operating system.

3). Opening a file

While working with files,the file must be open,the combined fopen() creates a new file or opening and existing file,it is represented by as follows

Example :

```
Variable_name = fopen(“file name”,”file mode”);

FILE *fp;

fp=fopen(“student.txt”,”w”);
```

Here w is used for writing mode

4). Reading a file :

To read data from a file,we use input statements such as fscanf(),fgets(),fgetc(),fread()

5). Writing a file :

To write data from a file,we use output statements such as fprintf(),fputs(),fputc(),fwrite()

6). Closing a file

Once the file processing finished or completed close all the files that have being opened,fclose() is used to close file.

Explain how to open a file

If you want to storee data in the secondary memory,we must specify certain things about the file to the file,they include:

1). File name

2). Data structure

3). Purpose

1). File name

It is a string of characters that makes a valid file name for the operating system,it contains primary name,secondary name.

Example

Student.c

Simple.txt

2). Data structure

Data structure of a file is defined as “FILE” in library of standard input, output function definitions.

FILE is defined as data type

Example

a). FILE *fp;

b). fp=fopen(“file name”, “file mode”);

3). Purpose

It specifies the purpose of opening this file.

```
/*Write a c program for details of student and print it on the screen using files*/
#include<stdio.h>
#include<conio.h>
struct student
  {
     int roll no;
     char name[15];
     int maths, phy,comp;
 };
main()
{
struct student s;
int i,n;
FILE *fp;
clrscr();
fp=fopen(“student.dat”, ”w”);
printf(“enter no of students:”);
scanf(“%d”, &n);
printf(“\n\n*** enter data ***\n\n);
for(i=1;i<=n; i++)
printf(“enter roll no\n:”);
scanf(“%d”, &s.roll no);
printf(“enter name:”);
scanf(“%s”,&s.name);
printf("enter maths marks\n");

scanf("%d",&s.maths);

printf("enter phys marks\n");

scanf("%d",&s.phys);

printf("enter comp marks\n");

scanf("%d",&s.copm);

fwrite(&s,size of(s),i,fp);

}

fclose(fp);

printf("\n\n *** inputted data ***");

fp=fopen("student.dat","r");

while(fread(&s,size of(s),I,fp)

{

printf("\n\n roll number=%d\n",s.rollno);

printf("name=",%s\n",s.name);

printf("maths=",%s\n",s.maths);

printf("phys=",%s\n",s.phys);

printf("comp=",%s\n",s.comp);

}

getch();

}
```

Output:-

```
Enter no.of students

*** enter data ***

Enter roll no:1
Enter name: joshi

Enter maths marks:24

Enter phys maks:23

Enter comp marks:25

*** inputted data ***

Roll no=1

Name=joshi

Maths=24

Phys=23

Comp=25
```