Unit - I

Chapter I: Introduction to Algorithms and Programming Languages

1. What is Algorithm? Give its characteristics and example.

An algorithm is a step-by-step procedure of solving the given problem statement. Algorithms are designed by using *pseudo code.* Pseudo code is a language independent code. All algorithms must satisfy the following characteristics.

Input	: Zero or more quantities are externally supplied
Output	: At least one quantity is produced
Definiteness	: Each instruction is in clear format
Finiteness	: The algorithm must terminates after a finite sequence of steps
Effectiveness	: Every instruction must be in basic format.

Basic rules followed while designing algorithms are:

- ☞ START and STOP statements are used for beginning and ending of the algorithm.
- ☞ READ and WRITE statements are used for input and output statements
- ☞ ← Symbol is used to assign values to the variables.

Control structures used in algorithms:

An algorithm has a finite number of steps and some steps may involve decision making and repetition. There are mainly three control structures. They are (1) Sequence (2) Decision and (3) Repetition.

1) Sequence:

Sequence means that each step of an algorithm is executed in the specified order i.e in sequential order.

Example: Algorithm to add two numbers

```
Step 1   : Start
Step 2   : Read x, y
Step 3   : Sum ← x + y
Step 4   : Write sum
Step 5   : Stop
```

2) Decision:

Decision means execution of a process based on the result of a condition. The process may contain one or more instructions. If the condition is true it executes one process. If the condition false it executes another process.

The word "**if**" is used to specify decision in an algorithm. There are various forms of "**if**" constructs used in algorithms.

a. **Simple 'if' construct:** In this method if the condition is true then it executes a process. Otherwise it will not execute the process.

   Syntax: **If condition then process**

   Example: Algorithm to check the equality of two numbers

```
Step 1:   Start
Step 2:   Input the first number as A
Step 3:   Input the second number as B
Step 4:   If A=B Then Print "Equal"
Step 5:   End
```

b. **IF.. Else construct:** In this method if the condition is true then it executes process-1. If the condition is false then it executes the process-2.
Syntax: **If condition then**

```
            Process-1
        Else
            Process-2
```

Example: Algorithm to check the equality of two numbers

```
Step 1:    Start
Step 2:    Input the first number as A
Step 3:    Input the second number as B
Step 4:    If A=B Then
                Print "Equal"
           Else
                Print "Not Equal"
Step 5:    End
```

3) Repetition:

Repetition means executing one or more steps for a number of times. Repetition is also called Iteration or Loop.

The words **"**While, Do..While, For**"** are used to specify repetition in an algorithm. These loops execute one or more steps as long as the condition is true.

Example: Algorithm to print first 10 natural numbers

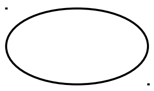
```
Step 1:    Start
Step 2:    Let $x = 1$
Step 3:    Repeat steps 4 and 5 while $x \leq 10$
Step 4:    Print $x$
Step 5:    Let $x = x + 1$
Step 6:    End
```

2. What is flowchart? Which symbols you are using while designing flowcharts.

Pictorial or Graphical representation of an algorithm is called a flowchart. Flowcharts are designed by using some specific symbols. The most import symbols used for designing flowcharts are:

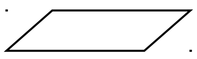
Start / Stop Statements: The symbol used for to show START / STOP statements is "Oval".

Symbol :



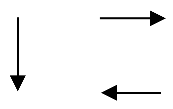
Input / Output Statements: The symbol used to represent input statements and output statements is "Parallelogram".

Symbol :



Flow Lines: The symbol used to represent data flow from one place to another place is "Arrow".

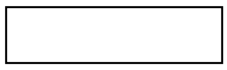
Symbol :



Arrow symbol actually connects every two symbols in the flowchart.

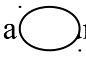
<u>Process Statements</u>: The symbol used to represent processing instructions is "Rectangle".

Symbol :



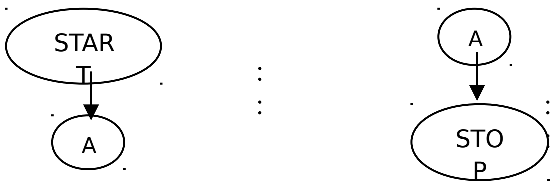
<u>Connector Symbol</u>: The symbol used to connect two parts of the program flow is "Circle".

Symbol :



When we reach the end of a column or a page, but total chart is not finished. In this case, at the bottom of flow we use a connector to show that the flow continues at the top of the next column or page.

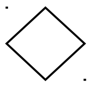
Example :



Additional symbols that used for designing flowcharts are:

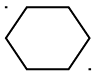
<u>Decision or Condition symbol</u>: Decision diamonds create two data flow from one. A decision diamond always has True / False or YES/NO written on the left and right sides.

Symbol:



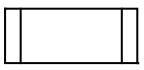
<u>Loop or Iterative symbol</u>: This symbol has hexagon shape. It is used when a problem has repetitive steps. i.e. a problem has solution with steps repeating again and again till a condition or definite number of times.

Symbol:



<u>Function or procedure or sub- program symbol</u>: This symbol has a rectangle shape with two cuts or lines inside. This can be labeled with the function or procedure name. Every program has ends with a terminator shape i.e. with an oval.

Function:

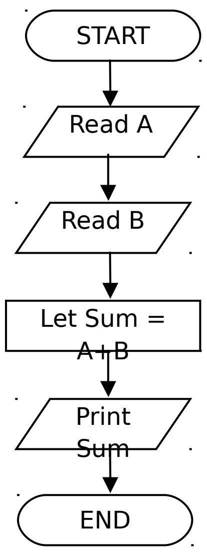
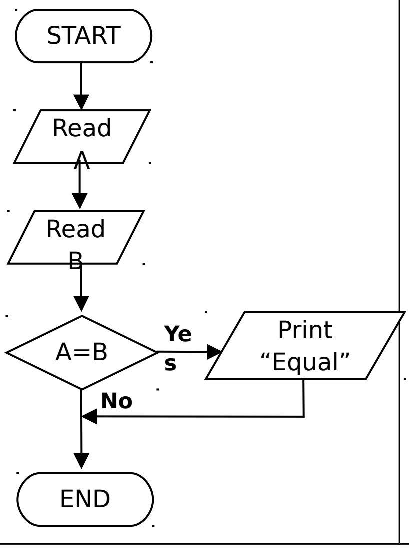
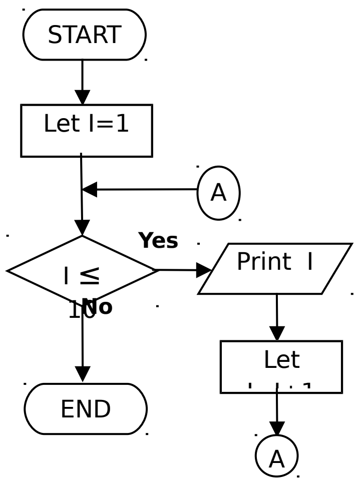


Advantages of Flow Charts:

1. Flow charts are very good communication tools to explain the logic of a process.
2. They are more helpful for complex programs.
3. They are used of program documentation.
4. They can be used to debug programs easily to detect and remove mistakes.
5. They act a guide for the programmers to write program in any programming language.

Disadvantages of Flow Charts

1. Drawing flow charts is time consuming process. It takes more time to draw
2. A simple Modification may require complete re-drawing of the flow chart.

Eg-1: A flow chart to add two numbers	**Eg-2:** A flow chart to find equality of two numbers	**Eg-3:** A Flow Chart for Repetition (To print first 10 natural numbers)
 START Read A Read B Let Sum = A+B Print Sum END	 START Read A Read B A=B Yes Print "Equal" No END	 START Let I=1 A I ≤ 10 Yes Print I Let A No END

3. What is programming language? Write a brief note the importance of programming languages.

A programming language is a language specifically designed to express computations that can be performed by the computer. Programming languages are used to create program that control the behavior of a system, or as a mode of human – computer communication.

Usually, programming languages have a vocabulary of syntax and semantics for instructing a computer to perform specific tasks. The term programming language usually refers to high – level languages such as BASIC, C, C++, COBOL, FORTRAN, Ada, and Pascal etc.

High level programming languages are easy for humans to read and understand, the computer actually understands the machine language that consists of numbers only. Each type of CPU has its own unique machine language.

In between the machine language and high level languages, there is another type of language known as assembly language. Assembly languages are similar to machine languages, but they are much easier to program because they allow programmer to substitute names for numbers.

4. Explain in detail different types of programming language generations.

The concept of generations of programming languages is closely connected to the advances in technology that brought about computer generations. There is five generation of programming languages available.

1. Machine language (1GL)
2. Assembly language (2GL)
3. High - level language (3GL)
4. Very high - level Language (4 GL)
5. Fifth generation Languages (5GL)

First Generation: Machine Language

- This is lowest level of programming language.
- Only two symbols are used in writing the program using machine language.
- They are 0’s and 1’s. Writing the program in machine language is tedious.
- The programmer should have the inner details of computer.
- The main advantage of machine language is that the code can run very fast and efficiently.

Advantages

- Code can be directly executed by the computer.
- Execution is fast and efficient
- Program can be written to efficiently utilize memory.

Disadvantages

- Code is difficult to write.
- Code is difficult to understand by other people.
- Code is difficult to maintain.
- It is difficult to detect and correct errors.
- Code is machine dependent and thus non- portable.

SECOND GENERATION: ASSEMBLY LANGUAGE

- The second generation programming language includes the assembly language.
- Assembly languages are formed with the combination of mnemonic codes.
- These codes are easy to remember abbreviations rather than numbers. Which contain simple English words like ADD, SUB and MUL etc.,
- Program written in assembly language and convert to machine language need a translator that is known as assembler. I-e the computer will understand only the language of 1’s and 0’s and will not understand mnemonics like ADD and SUB.

Advantages

It is easy to understand.
It is easier to write programs in assembly language than in machine language.
It is easy to detect and correct errors.
It is easy to modify

Disadvantages:

Code is machine dependent and thus non- portable.
Programmers must have knowledge of the hardware and internal architecture of the CPU.
The code cannot be directly executed by the computer.

THIRD GENERATION: 3GL

- Third – generation languages are high - level languages.
- Here instructions are written in statements like English language statements.

- ☞ Each instruction in this language expands into several machine language instructions. Third generation programming languages have made it easier to write and debug programs.
- ☞ Third generation programming languages include languages such as BASIC (Beginners' All – Purpose Symbolic Code), FORTRAN (Formula Translator) and COBOL (Common Business Oriented Language), C++ and Java.

Advantages:

- ☞ The code is machine dependent.
- ☞ It is easy to learn and use the language.
- ☞ It is easy to document and understand the code.
- ☞ It is easy to detect and correct errors.

Disadvantages:

- ☞ Code may not be optimized.
- ☞ The code is less efficient.
- ☞ It is difficult to write a code that controls the CPU, memory, and registers.

FOURTH GENERATION: VERY HIGH LEVEL LANGUAGE

- ☞ Fourth generation languages are non- procedural languages.
- ☞ Here programmers define only what they want the computer to do, without supplying all the details of how it has to be done.
- ☞ Fourth generation languages need approximately one tenth the number of statements that a high level language needs to achieve the same results.
- ☞ The main **objectives** of fourth generation language is
  - Increasing the Speed of developing programs.
  - Minimize the user efforts to obtain the information from computer.
  - Decreasing the skill level required of users so that they can concentrate application rather than coding.
- ☞ Five basic types of **language tools** fall into the fourth generation language category.
  - Query language, Report generators, Application generator
  - Decision support systems and financial planning languages.
  - Some micro computer application software.

FIFTH GENERATION LANGUAGE:

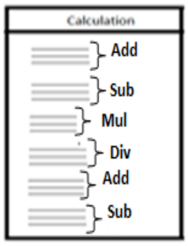
- ☞ Fifth generation languages concentrate on Natural languages representation.
- ☞ 5 GLs are designed to make the computer solve a given problem without the programmer.
- ☞ While working with 4Gl, programmers have to write a specific code to do work, but with a 5GL, they only have to worry about what problems need to be solved and what conditions need to be met, without worrying how to implement a routine or an algorithm to solve them.
- ☞ In general, 5Gls were generally built upon LISP, many originating on the LISP machine, such as ICAD.
- ☞ These languages are also designed to make the computer "smarter". Natural languages already available for microcomputers include Clout, Q&A, and Savvy Retriever (for use with data bases) and HAL (Human Access Language).
- ☞ Fifth generation programming languages are used in artificial intelligence research. Some examples of 5GLs include Prolog, OPS5, and Mercury. A good example of fifth generation language is Visual Basic.

LIST OF PROGRAMMING PRINCIPLES

1. Unstructured programming (USP)
2. Procedure oriented programming (POP)

3. Modular (or) structured oriented programming (SOP)
4. Object oriented programming (OOP)

1.Unstructured programming (USP): According to unstructured programming concept, program is developed by organizing data and instructions in sequential order.



Draw backs of unstructured programming.

- Redundant data, Because of redundancy the size of program increases.
- Occupy more space, which reduce speed.
- There is no proper organization of data and operations.

Ex: Assembly languages, Machine Language are USP Programming Languages.

2. Procedural oriented programming (POP)

Operations provided by a program are divided into small pieces and each piece of a program is called subroutine.

Ex: COBAL, and PASCAL are called procedural oriented languages.

Advantages of procedural oriented programming:

Modularity: It is a concept of developing an application in sub modules i.e. procedure oriented approach.

Reusability: Write once and use many times.

Simplicity: It is easy to understand operations of a program.

Efficiency: By reducing the size of a program efficiency of procedure oriented programs increases.

Characteristics of procedural oriented programming:

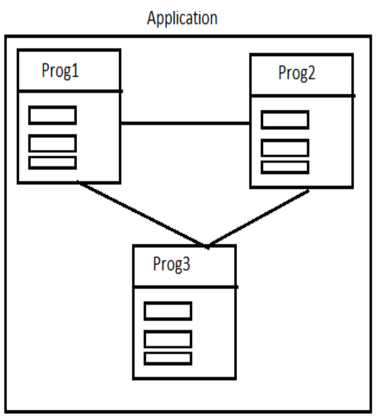
- Large programs are divided into small programs.
- Most of the functions share global data.
- Functions transform data from one another.

Drawbacks of procedural oriented programming:

- Concentrated on development of functions.
- In large program it is very difficult to identify what data is used by which function.
- Debugging application is complex.

5.Write about structured programming language. Give its advantages.

☞ Structured program is a top down approach in which the overall program structure is broken down into separate modules.
☞ It allows the code to be efficiently loaded into the memory and to be reused in other programs.
☞ Structure oriented program is subset of procedural oriented programming approach.
☞ In this approach reusability of subroutines are between the programs.
☞ C is a structured oriented (or) procedure oriented programming language.



Advantages:

☞ The goal of structured programming is to write correct programs that are easy to understand and modify.
☞ Modules enhance the programmer's productivity.
☞ With modules, many programmers can work on a single large program, with each working on different module.
☞ A structured program can be written in less time than an unstructured program.
☞ Modules or procedures written for one program can be reused in both programs as well.

- ☞ A structured program is easy to debug.
- ☞ Individual procedures are easy to change as well as understand.

Chapter II: Introduction to C:

Introduction

The programming language C was developed in the early 1972s by Dennis Ritchie at Bell Laboratories to be used by the UNIX operating system. It was named 'C' because many of its features were derived from an earlier language called 'B'.

History of C

1. The programming language term is started in the year of 1950's with the language called **FORTRAN**.
2. From **FORTRAN** language one more programming language is evaluated **ALGOL.**
3. The beginning of **C** is started in the year of 1967 with the language called **BCPL**. Basic Combined Programming Language which is evaluated by **Martin Richards.**
4. In the year of 1970's from BCPL one more programming language is evaluated by **ken Thomson** called **B language.**
5. From ALGOL, BCPL and B; in the year of 1972 **Dennis Ritchie** was evaluated one more programming language called **C language**. At Bell Labs for developing system software.
6. C was documented and popularized in the book **'The C Programming Language'** by Brian **W. Kernighan and Dennis Ritchie** in 1978. This book also popular that the language came to be known as **'K& RC'**.
7. In the year of 1989 C programming language is standardized by ANSI that version is called ANSI C.
8. In 1990 the international standards organization (ISO) adopted the ANSI standard. This version of C came to be known as C89.
9. In 1995, some minor changes were made to C89, the new modified version was known as C95.
10. In 1999, some significant changes were made to C95; the modified version came to be known as C99.
11. In the year of 2000, C99 was adopted as an ANSI standard.

1.What are the Uses or characteristics of C language:

1. **General Programming Language:** It is a general purpose programming language. Hence, it is used for writing system and application programs.
2. **Maintainability:** It is reliable, simple and easy to use.
3. **System Programming:** C language is used to develop compilers, operating systems and other utility programs for system software.
4. **Portability:** C language is highly portable language i.e. a C program written on one computer can be compiled and executed on a different computer.
5. **Structured Programming Language:** It supports many control structures such as "if, switch, while, do…while, for" etc. to develop well defined and easy to maintain programs.
6. **Modularity:** Through modularisation, we can divide a large program into small modules called functions.
7. **Limited Keywords:** It has only 32 keywords and several standard functions are available with 'C' are used for developing error free programs.

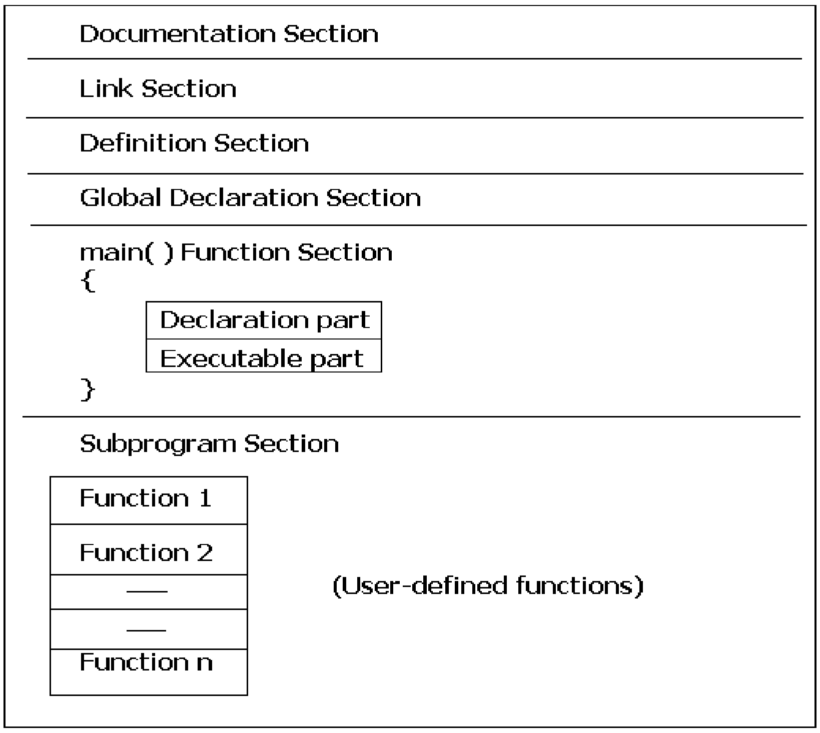
Applications of C

- ☞ To develop the system software like operating system and compilers.
- ☞ To develop the application software's like spread sheets and databases.
- ☞ To develop the graphic related applications that s gaming   application.
- ☞ To evaluate any mathematical problem C program language can be use.

2. Write about the structure of a C program?

A 'C' program can be viewed as a group of building blocks called functions. A function is a self contained block of statements which perform a particular task. In order to write a 'C' program, first create the functions and then put them together.

BASIC STRUCTURE OF C PROGRAMS:



1) Document Section:

- ☞ This is an optional section
- ☞ This section consist information about the program
- ☞ It is not understand by compiler
- ☞ This documenttation is provided by using comments.
- ☞ C provides two types of comments.
    1. Single line comments. (//)
    2. Multi line comments. (/*…. */)

  Ex: /* this is a c-program to find addition of 2 numbers */

Link section or preprocessor section

- ☞ Preprocessor statements are used to import different predefined functions into the program.
- ☞ This section is executed , before compiling the program and it is used for linking other program with current program.
- ☞ The functions are included form the system library by the means of preprocessor directives.
- ☞ The system library consists of a set of header files names 'stdio.h', 'math.h', 'conio.h' etc., these are included using '#include' statement.
- ☞ This statement is not terminated by a semicolon (;).

Definition section

- ☞ The definition section defines all the symbolic constants.
- ☞ The symbolic constants are defined by the statement '#define'.
- ☞ This statement is not terminated by a semicolon (;).

Ex:
```
#define  pi  3.14
#define   c  120
```

Global declaration section

- ☞ Global variables are used to share the data along all functions of the program.
- ☞ There are some variable that are used in more than one function. Such variable are known as global variables.
- ☞ These variables are declared in the global declaration section, which is outside of all the functions.
- ☞ These variables are accessed by entire program.

Main ( ) function section

Every 'C' program must have one main() function section. It consists of two parts namely:

1. declaration part
2. executable part

- ☞ The declaration part declares all the variables that are used in executable part.
- ☞ The executable part must contain at least one statement.
- ☞ Every statement in declaration and executable parts should ends with semicolon (;) symbol.

Sub-program section

- The sub-program section contains all the user-defined functions that are called in the main() function section.
- These functions can appeared in any order. These functions are placed immediately after main() function section.
- Every 'C' program must contain main function while other sections are optional.

FIRST C PROGRAM

C programming language is a case sensitive language. I.e. upper case and lower case both are treated as different.
When we are working with C language existing keywords and functions should be used in same case only.

```
void main()
{
  printf("Welcome to GMinformatics");
}
```

OUTPUT: welcome to GMinformatics

In the above program:

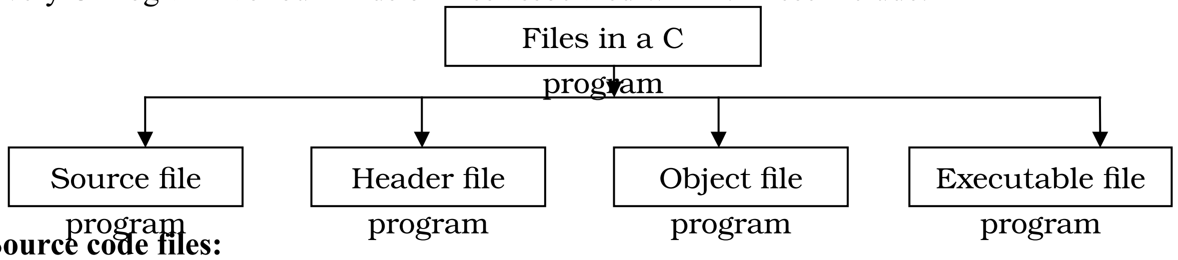
- void is a keyword which indicates return type of the function.
- main() is a identifier which indicates starting point of the program.
- Opening curly brace indicates instruction block is started, closing curly brace is indicates instruction block is ended.

- **printf():-** printf () is a predefine function, functionality of printf function is it prints number of arguments must be within the double quotes and every argument should separated with comma.

Within double quotation whatever we leave it prints as these. If any format specifier copy that type of value.

3. Write about the files used in a C program? (Or)

Every C Program has four kinds of files associated with it. These include:



Source code files:

The source code file contains the source code of the program. The file extension of any C source code file is '.c'.

Header files:

When we are working with large projects, it is often desirable to separate out certain subroutines from the main function of the program. The advantage of header files can be realized in the following cases:

☞ The programmer wants to use the same subroutines in different programs.

☞ The programmer wants to change or add subroutines.

☞ Header files names ends with a 'dot h' (.h).

☞ **Standard header files** in the program we have used printf( ) function that has not been written by us. We do not know the details of how this function works. Such functions that are provided by all c compilers are included in standard header files.

Examples of standard header files:
String.h: for string handling functions.
Stdlib.h: for some miscellaneous functions.
Stdio.h; for standardized input and output functions.
Math.h : for mathematical functions
Alloc.h : for dynamic memory allocation
Conio.h: for clearing the screen.

Object file:

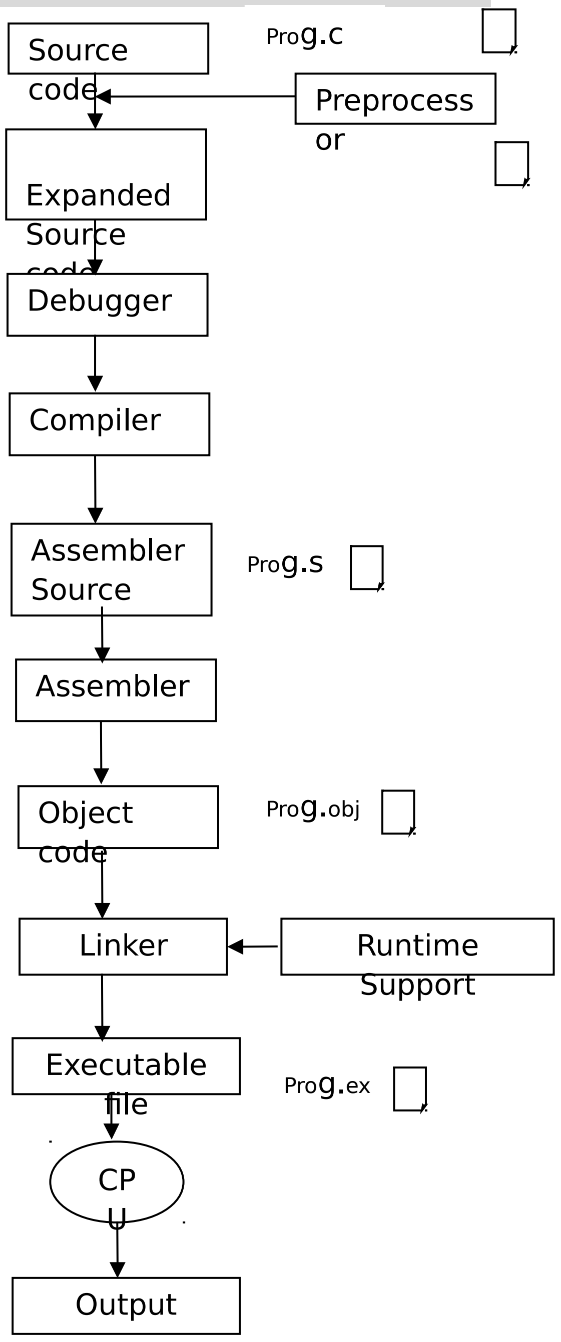
Object files are generated by the compiler as a result of processing the source code file. Object files contain compact binary code of the function definitions. Object files have '.o' extension, and some of the operating systems include Windows and MS- DOS have a '.obj' extension for the object files.

Binary executable files:

The binary executable file is generated by the linker. The linker links the various object files to produce binary file that can be directly executed. On windows operating system, the executable files have an '.exe' extension.

4. Write about the execution process of a C program? (Or)
Write about the build process?

- The code written in C language is called the source code. It is saved with .c extension (prog.c).
- A program called preprocessor process the source code before compilation of the program. The preprocessor generates the expanded source code or intermediate code, which is saved with .i extension (prog.i). The expanded source code is then sent to the compiler.
- Debugger is a part of compiler, which identifies the errors and warnings if any. If the expanded code is error-free then the compiler generates the assembler source , which is saved with .asm extension (prog.asm)
- The assembler then converts the assembler source into object code, which is saved with the either .obj extensions (Prog.obj).
- As the final step, the linker links the runtime support library to the object code and generates executable, which is saved with .exe extension (prog.exe). It is the actual build, which can be copied on to any other machine and used as software.



5. What is Comments in C? How can you use comments Give an example.

Many a time the meaning or the purpose of the program code is not clear to the reader. Therefore, it is a good programming practice to place some comments in the code to help the reader understand the code clearly. Comments are just a way of explaining what a program does. Comments can be used anywhere in the program. It is not understand by compiler

C provides two types of comments.

Single line comments. (//)

Multi line comments. (/*…. */)

```
/* author: Reema Thareja
     Description: To print ‘welcome to the world of C’ on the screen */
   #include<stdio.h>
   int main
   {
      printf( “ welcome to the world of C”); //print message
      return 0; // return a value 0 to the operating system
   }
```

Output: Welcome to the world of C.

6. Write about the concept of Tokens in C.

- Smallest unit of a program or an individual unit is called Token.
- When we are working with C program or an individual unit is called Token.
- Tokens can be keyword, operators, separators, constant and any other identifiers.
- When we are working with token we can’t split the token or we can’t break the token but beginning the tokens we can use n number of spaces, Tabs and new lines.

‘C’ tokens are divided into 6 types. They are:

1. keywords
2. Variables
3. constants
4. strings
5. special characters

6. operators

Keywords:
It is a reserved word some meaning is already allocated to this word, and that meaning already knows by compiler. In c programming language total number of keywords is 32. these keywords cannot be used as an identifier.

ex:- if, else, break, while, case, goto.

Variable:
a variable is defined as a meaningful name given to a data storage location in computer memory. or a variable is a quantity that does change during the program execution.

Constants:
It is a fixed never be changed during the execution of the program. In c programming language constants are classified into two types.

- Alpha numeric constants.
- numeric constants

Strings:
A string is a sequence of character enclosed in double quotes. the character may be an alphabet, number, special symbol and blank spaces.
Example: "MAHESH", "MCA", "gminformatics"
Special characters:
'C' supports some special operators of interest such as [ ], ( ), -->, * these are used as a separation, pointer and some other special purposes.

Operators:

It is a special kind of symbol which performs a particular task. In c programming language total number of operators 44.

7. What is identifier and write the rules to follow while selecting an identifier?

An identifier is a name given to the program elements such as variables, arrays, and functions. Identifiers may consist of sequence of letters, numerals, or underscores.

Rules for forming identifier names:

- ☞ Identifiers cannot include any special characters or punctuation marks (like #, $, ^,?,., etc) except the underscore.
- ☞ There cannot be two successive underscores.
- ☞ Keywords cannot be used as identifiers.
- ☞ Identifiers can be of any reasonable length. They should not contain more than 31 characters.
- ☞ In identifier declarations name of the variable must starts with alphabet or underscore only.

Examples: roll_number, marks, name, emp_number, basic_pay, HRA, DA, dept_code

8. What is variable and write the rules to follow while selecting a variable?

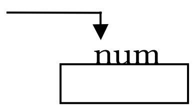
Variable:
A variable is defined as a meaningful name given to a data storage location in computer memory.
Or
A Variable is a quantity that does change during the program execution.
Rules for forming identifier names:

- ☞ Name of the memory location is called variable.
- ☞ Before using any variable in the program it must be declare first.
- ☞ Declaration of variable means need to mention data type, name of the variable followed by semicolon.
- ☞ In C programming language variable declarations must be exist top of the program after opening the curly braces.
- ☞ In variable declarations name of the variable must starts with alphabet or underscore only.
- ☞ In variable declarations maximum length of a variable is 32 characters, after 32 characters complier will not consider remaining characters.
- ☞ In variable declaration existing keywords operators, separators, constants and any other special characters will be not allowed.

Syntax:

<Data type> <Name/list of the variable>

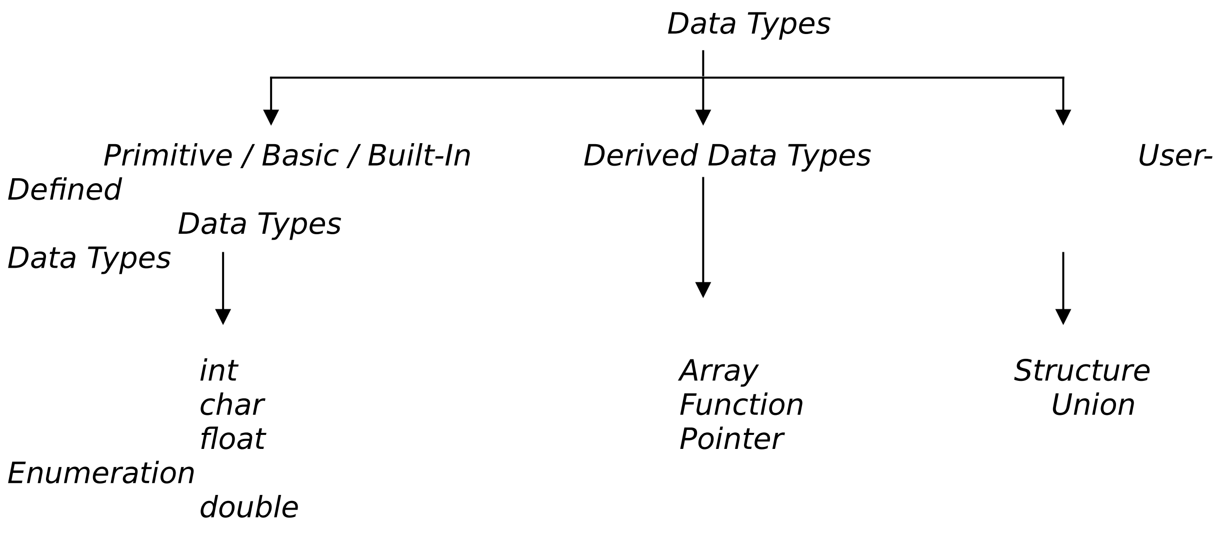
Example: int num;



9. Define data type? Explain different data types in C language.

Data types are the keywords and tell the compiler which type of data (information) is maintained in memory.

- ☞ Data types will decide what type of values need to be hold into variables.
- ☞ In C programming language there are three types of basic data types are exist. i.e. char, int, float.In C programming language there are 9 types of predefine data types are available.
- ☞ The advantage of classifying this many types is utilizing memory more efficiently and increase the performance.



1. BASIC DATA TYPES:

C compilers support five fundamental data types. They are integer (int), character (char), floating point (float), double-precision floating point (double) and void.

Integer types:

It represents without fractional part. Negative values are also allowed. C has three classes of integer storage. They are short int, int and long int, in both signed and unsigned forms. The size and range of these types are shown in the following table:

SNO	TYPE	SIZE	RANGE	FORMAT SPECIFIER
1	int	2	-32678 to +32767	%d
2	short int	2	-32678 to +32767	%hi
3	signed short int	2	-32678 to +32767	%hi
4	unsigned int	2	0 t0 65535	%u
5	unsigned short int	2	0 t0 65535	%hu
6	long int	4	-2147483648 to 2147483647	%li
7	unsigned long int	4	0 to 4294967295	%lu

Character Type:

A single character can be defined as a character (char) type data. Characters are usually stored in 8 bits (One byte) of internal storage. The qualifier signed or unsigned may be explicitly applied to char.

SNO	TYPE	SIZE	RANGE	FORMAT SPECIFIER
1	char	1	-128 to 127	%c
2	Signed char	1	-128 to 127	%c
3	unsigned char	1	0 to 255	%c

Floating Point Types:

Floating point numbers represents with fraction values. These are stored in 32 bits, with 6 digits precision. Floating point numbers are defined by C by the keyword float. When the accuracy provided by a float number is not sufficient, the type double can be used to define the number. A double data type number uses 64 bits giving a precision of 14 digits. These are known as double precision numbers. To extend the precision further, we may use long double which uses 80 bits.

SNO	TYPE	SIZE	RANGE	FORMAT SPECIFIER
1	float	4	3.4E- 308 to 3.4E+38	%f
2	double	8	1.7E-308 to 1.7E+308	%lf
3	long double	10	3.4E-4932 to 1.1E + 4932	%lf

Void Types:

The void type has no values. This is usually used to specify the type of functions. The type of a function is said to be void when it does not return any value to the calling function.

2. DERIVED DATA TYPES:

In implementation whenever the basic data types are not supports user requirement then go derived data types of C language.derived data types are created from basic data types only. But extending the size and ranges of basic data type.

Array:

An array is a collection of memory locations which can share same data name and same data type values.

Function:

Self contain block of one or more statements which is designed for a particular task is called functions or sub program in an application called function.

Pointer:

A pointer is a variable which holds address of another variable. (Or)

3.USER DEFINED DATA TYPES:

All predefine data types are designed for basic operations only. i.e. it can work for basic data types. In implementation whenever the primitive data types are not supporting user requirement then go for structures.

Structure:

Structure is a collection of different data type variables in a single entity.Structure is a collection of primitive and derived data type variables.By using structures we can create user defined data types.

Union:

Union is a user defined type. It looks similar to structure, but the functionality differs. When a variable belongs to union is created, it allocates the common memory allocation to all the members.

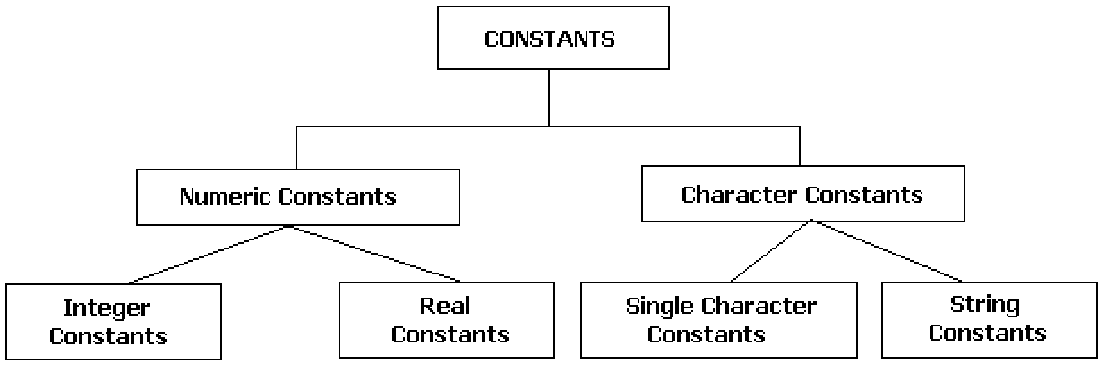
Enumeration:

enum is a keyword. By using enum we can create sequence of integer content values.By using enum we can create userdefined datatype of integer.

10. Write about the Constants or Literals used in C language?

It is a fixed never changed during the execution of the program. Constants are used to define fixed values like pi or electron charge.

The literal values inserted in the code are called constants because we can't change these values. C supports several types of constants (literals) as follows:



Integer Constants: An Integer constant refers to a sequence of digits. There are three types of integer constants. They are: Decimal Integer, Octal Integer and Hexadecimal integer.

Decimal Integers: Decimal integers consists of a set of digits, 0 to 9, preceded by an optional – or + sign.

Ex: 123 -321 0 43564 +765

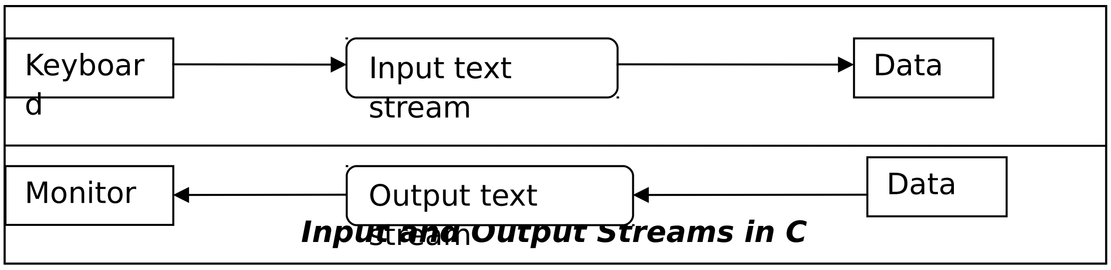
Octal Integers: An Octal integer constant consists of any combination of digits form 0 to 7, with a leading 0.

Ex: 035 0 0435 0776

Hexadecimal Integers: A sequence of digits preceded by 0x or 0X is considered as hexadecimal integer. They may also include alphabets A to F or a to f. The letters A to F represent the numbers 10 to 15.
Ex: 0x2 0x9f oXbcd 0x
Real Constants or floating point constants:
Numbers with fractional part are called real constants. These are often called as floating point constants. Real constants are generally represented in two forms are fractional form and exponential form.
Ex: 0.0083 -0.75 456.76 +32.3
A Real number may also be expressed in a exponential notation
 Ex : Mantissa e exponent
 215.65 may be written as 2.1565 e + 2
 2.1565 x 10^2
Single Character Constants: A single character constant (or character constant) contains a single character enclosed within a pair of single quote marks.
Ex: '5' 'X' ';' ' '
String Constants: A String constant is a sequence of characters enclosed in double quotes. The character may be letters, numbers, special characters and blank spaces.
Ex: "Hello!" "MAHESH", "Well Done" "gminformatics" "vsu updates"

11.Write about Input/ Output statements in C language.

Input output deals with the basic understanding of the streams involved in accepting input and printing output in C program.
Streams: A stream acts in two ways. It is the source of data as well as the destination of data. Streams are associated with a physical device such as a monitor or with a file stored on the secondary memory. C uses two forms of streams text – and binary.
In text stream, sequence of characters is divided into lines with each line being terminated with a new line character (\n). And binary stream contains data values using their memory representation.



Input and Output Streams in C

Formatting Input /Output
C language supports two formatting functions printf and scanf.
printf():
printf( ) is a predefine function, functionality of printf function is prints every argument on console. printf function its 'n' number of arguments but first argument must be within the double quotes and every argument should separate with comma.

Within the double quotation whatever we leave it prints as these, if any format specifier copies that type of value.

syntax:

printf (“control string”, variable list);

scanf( ) :

scanf is a predefine function by using scanf we can read the data at runtime from user at run time. Scanf function its ‘n’ number of arguments but first argument must be within the double quotes and every argument should separated with comma.

Within the double quotation we need to pass proper format specifiers only. i.e. one type of values we need to scan same type format specifier should be provided.

When we are working with scanf function argument list should be provided with ‘&’ symbol or else values will be not store in proper variables.

syntax

scanf(“control sting”, arg1, arg2, arg3 …., argn)

Format specifiers:-it indicates what type of values needs to be printed on console.

- ☞ %d -------------------------> int
- ☞ %f---------------------------> float
- ☞ %c----------------------------> char

12. Write about the operators used in C language?

C is having very rich built-in operators. An operator is a leading body that acts on specific operands to result the task. These operators are classified into various types based on their role played at different streams upon these.

Operator: - It is a special kind of symbol which performs in specific task.

Operand: - The objects on which the operators are acts upon are known as operands.

C operators can be classified into a number of categories. They are

1. Arithmetic operators
2. Relational operators
3. Logical operators
4. Equality operators
5. Unary operators
6. Conditional operators
7. Bitwise operators
8. Assignment operators
9. Comma operator
10. Sizeof operator

1. Arithmetic operators:

Arithmetic operators are used to build arithmetic Expressions in “C” program. Arithmetic operators are used to perform both unary and binary arithmetic operations. The arithmetic operators in “C” Language are

+, -, *, /, %

Operator	Meaning	Syntax
+	Addition	a+b
-	Subtraction	a-b
*	Multiplication	a*b
/	Division	a/b
%	Modulo division	a%b

2. Relational operators:

Relational operators are used in decision statements like if, while and for etc., these are the binary operators used between any two operands. Any relational expression returns either true or false that is 1 or 0. The relational operators in "C" language are <, >, <=, >=, ==, !=.

Operator	Meaning
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to

3. Logical operators:

The logical operators are used when we want to test more than one condition and make decisions. Logical operators are also used to concatenate multiple relational expressions. Hence a logical expression is called compound relational expressions. The result of any logical expression is either true or false that is either 1 or 0.

Operator	Meaning
&&	logical AND
||	logical OR
!	logical NOT

4. Equality operators: C language supports two kinds of equality operators to compare their operands for strict equality or inequality.

Operator	Meaning
==	Return 1 if both operands are equal, 0 otherwise.
!=	Returns 1 if operands do not have the same value, 0 otherwise.

5. Unary operators: Unary operands act on single operands. C language supports three unary operators: Unary minus, increment, and decrement operators.

Unary minus: Unary minus operator is different from the binary arithmetic operator that operator. When we are working with unary operator an operand is preceded by a a minus sign, the unary operator negates its value. i.e if the number is positive then it become negative and if the number is negative then it becomes positive.

Increment and decrement operators:

++, – – are the increment and decrement operators used to increment and decrement the value of a variable by 1. These can be used as a postfix or prefix operators according to the requirement.

***6. Conditional operators*:**

Conditional operators are ternary category operators (? :).

- Ternary category means it require three arguments.
    - i.e. left, middle and right side arguments.

- If the expression is true then it returns with middle argument, if the expression is false then it returns with right side argument and left side argument is expression or condition.

$$\text{Value} = \frac{EXP1}{L} \quad ? \quad \frac{EXP2}{M} : \frac{EXP\ 3}{R};$$

Note: Number of question marks and colons should be equal.
Every colon should match with just before the question mark.
Every colon should follow by question mark only.

Example: large = (a>b) ? a:b

7. Bit wise operators:

These are the special operators used for manipulation of data at bit level. These operators are used for testing the bits or shifting them right or left. These may be applied to the float or double. The bit wise logical operators are. &, |, ^, ~. Bitwise shift operators are >>, <<.

8. Assignment operator:

It is a binary category operator which is used to assign the right side value to left side. When we are working with binary operators it should require and among those two operands.

Syntax: **L=R;**

9. Comma Operator:

The comma operator C takes two operands. It works by evaluating the first and discarding its value, and then evaluates the second and returns the value as the result of the expression. Among all the operators the comma operator has the lowest precedence.

int a=2, b=3, x=0;

10. The sizeof operator:

It is a unary operator cum keyword, always sizeof operator returns an intefger i.e. sizeof the even argument. The size of operator is used to determine the amount of memory space that the variable, or expression.

m=size(sum);

13. Write about type conversion and typecasting

It is a concept of converting one data type values into another data type.
Type casting can be process by using type operator.
In implementation when we are expecting another format data from expression then go for type casting.
Syntax:
Syntax
Data Type1 var1 = value;
Data Type2 var2 ;
Var2 = (Data Type) var1;
Ex: float f=12.9;
int i;
i= (int)f;
In C Programming language there are two types are type casting process exist.

- Implicit Type casting
- Explicit Type casting

***Implicit Typecasting*:** It is a concept of converting the lower data type values into higher data types.Implicit type casting is under control of compiler. i.e. as a programmer when implicit type casting is occurred there is no need to be handle explicitly.

```
// An example of implicit conversion
#include<stdio.h>
int main()
{
   int x = 10;    // integer x
   char y = 'a';  // character c

   // y implicitly converted to int. ASCII
   // value of 'a' is 97
   x = x + y;

   // x is implicitly converted to float
   float z = x + 1.0;

   printf("x = %d, z = %f", x, z);
   return 0;
}
```

Explicit Type casting: It is a concept of converting higher data type values into lower data types. i.e. as a programmer when we are expecting explicit type casting then mandatory to use type casting process or else data over flow will be occurred. It helps us to compute expressions containing variables of different data types.

```
// C program to demonstrate explicit type casting
#include<stdio.h>

int main()
{
   double x = 1.2;

   // Explicit conversion from double to int
   int sum = (int)x + 1;

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

   return 0;
}
```

PROGRAMMING EXAMPLES:

STATEMENTS

A statement is a syntactic construction that performs some action when the program is executed. In C language, statements are classified into three types as:

1. Simple (or) Expression Statements
2. Compound (or) Block Statements
3. Control Statements

1. Simple (or) Expression Statements

A simple statement is a single statement that ended with a semicolon.

Example: int x,y;

2. Compound (or) Block Statements

Any sequence of simple statements can be grouped together and enclosed within a pair of braces termed as compound (or) block statements.

Example: {

```
        int x=4, y=2, z;
        z=x+y;
        printf("\nResult :%d",z);
}
```

UNIT II

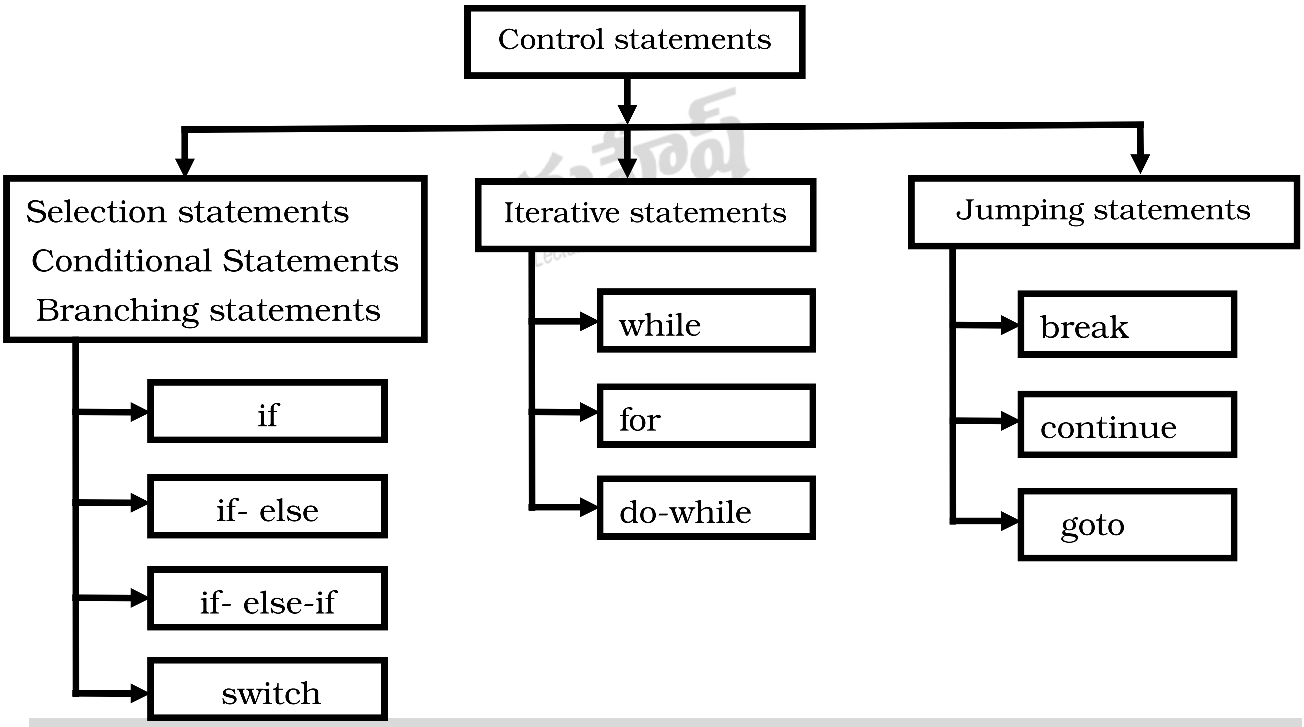
CHAPTER I: DECISION CONTROL AND LOOPING STATEMENTS

INTRODUCTION TO DECISION CONTROL STATEMENTS:

Generally, C program is a set of statements which are normally executed in sequential order from top to bottom. But sometimes it is necessary to change the order of executing statements based on certain conditions, and repeat a group of statements until certain conditions. Such statements are called control statements. In C language, control statements are classified into three categories as:

a) Selection (or) Decision Control Statements
b) Loop (or) Iterative Control Statements
c) Branch (or) Jump Control Statements

The control structures are used to control the flow of program execution. In C language there are there are three types of flow statements are exist.



1. Write about the conditional branching or conditional selection statements used in C language?

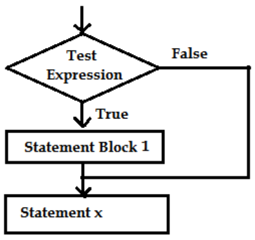
The conditional branching statements help to jump from one part of the program to another depending on whether a particular condition is satisfied or not. These decision statements include:

1. If statement
2. if-else statement
3. if-else-if statement
4. switch statement

1. If statement: The simplest form of decision control statement that is generally used in decision making. The general syntax of simple 'if statement' follows.

Syntax of if statement:

```
if (condition)
{
     statement1;
     ................
     statementn;
}
```



If the condition is logically true (i.e. non-zero), the statement following the ‘if’ executed.When there are multiple statements then they must be enclosed within curly braces ‘{ }’.

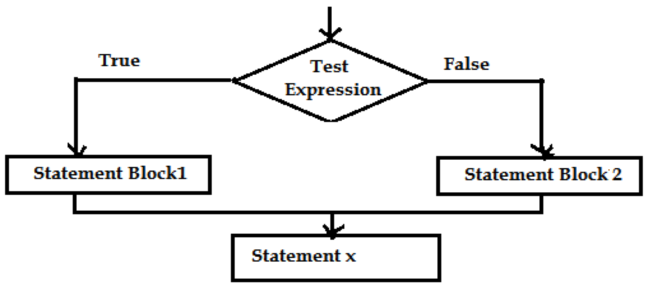
w.a.p Program to demonstrate simple if

```
#include<stdio.h>
#include<conio.h>
void main()
 {
   int age;
   clrscr();
   printf(" Enter age of the person :  ");
   scanf("%d",&age);
   if(age>=18)
   printf(" The person is eligible to vote");
   printf(" \nThank you");
   getch();
 } O/P: Enter the age of the person:  30
        The person is eligible to vote.
        Thank you
```

2. If-else statement: Using else is always optional.In implementation we having alternate block of condition then go for else part.When we are constructing the ‘else’ part mandatory to place ‘if’ part also. Because without if there is no else among those if and else only one block will be executed. i-e. When if condition false then only else part will be executed.

Syntax:

```
if(condition)
  {
       statement1;
       statement2;
   }
 else
  {
       statement3;
   }
```



w.a.p to find biggest of two numbers

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

```
        }
```

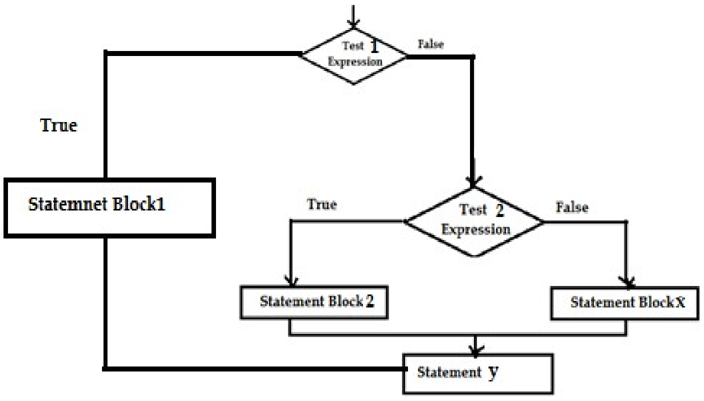
O/P: Enter two numbers: 20 30

 30 is biggest number.

3. If-else-If statement (Nested if else): If it is a concept of placing a condition part within another condition. We can construct 'n' number of nested blocks but any type of editor can support up to 255 blocks.

Syntax:

```
        if(condition1)
             {
                block 1;
              }
        else if(condition2)
           {
              block2;
            }
        else
             {
                block3;
             }
        }
```



w.a.p to find biggest of two numbers

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int a,b;
   clrscr();
   printf("enter 2 values:");
   scanf("%d%d",&a,&b);
   if(a==b)
   printf(" Two numbers are equal");
   else if( a>b)
       printf("%d is greater number",a);
   else
   printf("%d is smaller number",b);
   getch();
}
```

4) <u>Switch case (case control structures)</u>

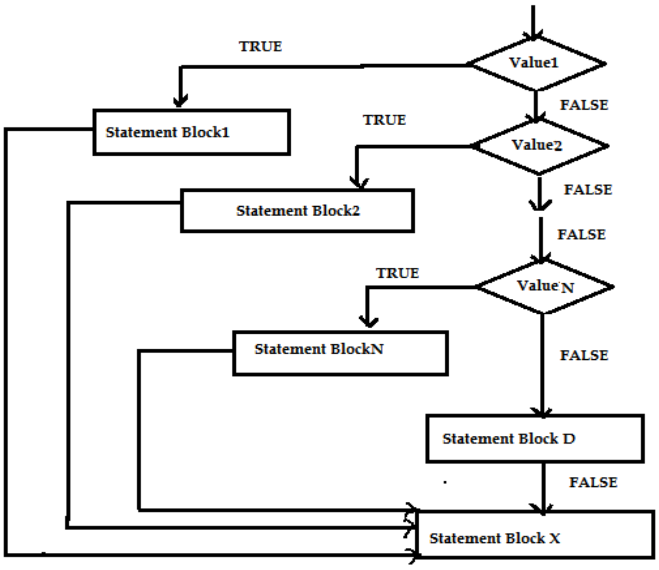
By using series of 'if' statements we can make a selection in a number of alternatives, but it is too complicated to develop. For this 'C' provides a special control statement that allows us to execute one in a number of cases electively, that is called '**<u>switch</u> statement' or case control statement.**

- ☞ Switch is a keyword. By using switch case we can create selection statements with multiple choices.
- ☞ Multiple choices can be created by using case keyword.
- ☞ When we are working with switch it require condition or expression of type integer only
- ☞ When we are working with switch all case constant values will be mapping switch condition return value.

- ☞ At the time of execution if condition return values is match with anyone of the cases then from matching is up to break, everything will be executed, if the break is not exist including default all cases will be executed.
- ☞ A default is a special kind of case which will execute automatically when the matching case is not occurred.
- ☞ Using default is always optional, in implementation when we are not handling all cases of the switch block then recommended go for default.

Syntax to switch case

```
switch(condition)
{
   case const1: Block1;
             break;
   case const2: Block 2
            break;
   case const3: Block 3;
           break;
    default:  block n;
}
```



Advantages of using switch statement:

- ☞ Easy to debug.
- ☞ Easy to read and understand.
- ☞ Ease of maintenance as compared with its equivalent if-else statement.
- ☞ Like if else statement switch statement also nested.
- ☞ Executes faster than its equivalent if-else construct.

w.a.p to perform arithmetic operations using switch case

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int a,b,c,opt;
 clrscr();
 printf("Enter two numbers: ");
scanf("%d%d",&a,&b);
printf("Enter your choice 1 to 4:  ");
scanf("%d",&opt);
 switch(opt)
 {
   case 1:  c=a+b;
   printf("Addition of two numbers: %d",c);
   break;
   case 2: c=a-b;
   printf("Subtraction of two numbers: %d",c);
   break;
   case 3: c=a*b;
   printf("Multiplication of two numbers:%d",c);
   break;
   case 4: c=a/b;
   printf("Division of two numbers: %d",c);
   default: printf("Invalid option");
   }
```

```
            getch();
          }
```

2. Write about the iterative or looping control statements in C language?

Set of instructions give to the compilers to execute set of statements until condition became false that is **loop**.

The way of repetitions are forms the circle, that's why iteration statements are called loops. i.e. loops means forming a circle. The basic purpose loop is ***code repetition.***

In implementation whenever the repetition are occurred when instance of writing statement go for loops.

In c programming language loops are classified into **three** types.

1. while loop
2. do-while loop
3. for loop

When we are working with iteration statements first it checks the condition and if the condition is evaluated as true then loop block will be executed.

After execution of the statement block once again it checks the condition, if the condition is true once again block will be executed.

While loop: When we are working with while loop always pre-checking process will occur and it repeats in clock direction.

<u>Syntax to while</u>

```
 while (condition)
 {
    statement 1;
    statement 2;
    ............
    ............
    statement n;
    inc /dec;
 }
```

Working: It executes all the statements in the statement-block as long as the condition true. When the condition false then the loop is terminated.

w.ap to demonstrate while loop.

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

O/P: *1 2 3 4 5 6 7 8 9 10*

<u>do-while:</u>

In implementation if at least need to be executed the statement block once. Then go for do-while. When we are working with do-while always post checking process will occur i.e. after execution of the statement block only condition will be evaluated.

Syntax to do-while

```
do
{
   statement 1;
   statement 2;
   statement 3;
   inc/dec;
}while (condition);
```

Working: It executes all the statements in the statement-block first and then tests the condition. When the condition true then the loop is repeated otherwise the loop is terminated.

w.a.p to demonstrate do- while loop

```
#include<stdio.h>
#include<conio.h>
void main()
{
  int i;
  i=1;
  do
  {
    printf("\t%d",i);
    i++;
  }  while(i<=10);
  getch();
} O/P: 1     2     3     4     5     6     7     8     9     10
```

Note:- In do-while loop mandatory to place semicolons after while.

For loop: When we are working with for loop it contains three parts.

1. Initialization
2. Condition
3. Iteration

Syntax to for loop:

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

    statement block:

}
```

Working

- ☞ The execution process of the for loop always starts from initialization
- ☞ Initialization will be executed only once when we are entering into the loop first time.
- ☞ After execution of the initialization block control will pass to conditional block, if the condition is evaluated as true then control will pass to statement block.
- ☞ After execution of the statement block control will pass to iteration block, after execution of the iteration, once again the control will pass to the condition.
- ☞ Always the repetitions will happen between condition, statement block and iteration only.
- ☞ When we are working with for loop everything is optional, but mandatory to place two semicolons.

☞ When we are working with for loop it repeats in anti clock direction.

w.a..p to demonstrate for loop

```
#include<stdio.h>
#include<conio.h>
void main()
{
  int n,i;
  clrscr();
  printf("Enter a range: ");
  scanf("%d",&n);
  for(i=1; i<=n; i++)  //i<=n/2
  {
     printf("\t%d",i);
  }
  getch();
}
```

3. Write about the concept of nested loops in C.

Loop control statements can be placed within another loop control statement. Such representation is known as nested loops. In these situations, inner loop is completely embedded within outer loop.

Example:

```
for( i=1; i<=n; i++)                outer loop
{
        ----
        for( j=1; j<=i; j++)        inner loop
        {
                ---
        }
        ----
}
```

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int r,i,j;
 printf("enter number of rows");
 scanf("&d",&r);
 for(i=1;i<=r;i++)
 {
  printf("\n");
  for(j=1;j<=i;j++)
  printf("*");
  }
 getch();
}
```

3. Write about the break and continue (Jumping or Skipping) statements used in C language? Or unconditional branching statements.

Branch control statements are used to transfer the control from one place to another place. C language provides three branch control statements. Those are

i) break statement ii) continue statement iii) goto statement

break statements:

- ‘break’ is a keyword by using break we can terminate loop body of switch body.
- Using break is always optional. But it should be exist within the loop body or switch body only.
- In implementation where we know maximum number of repetitions but certain condition is there we need to stop the repetition process, and then go for break. A break is usually associated with an ‘if’.

  Syntax: break;

w.a.p to demonstrate break statement

```
#include<stdio.h>
#include<conio.h>
void main( )
{
    int i=1;
    while(i<=10)
    {

      if(i==6)
      break;
      printf("\t Mahesh");
      i=i+1;
    }
  getch();
}  O/P: Mahesh     Mahesh      Mahesh      Mahesh      Mahesh
```

The ‘continue’ statement:

- Continue is a keyword by using ‘continue’ we can skip some of statements from loop body.
- Using continue keys always optional. But it should be within the loop body only.
- In implementation where we knows maximum number of repetitions but certain conditions is there where we need to skip the statement block of loop body then go for continue.
- A ‘continue’ is usually associated with an’if’.

  Syntax: continue;

Write a C++ program to demonstrate continue statement.

```
#include<stdio.h>
#include<conio.h>
void main()
{
 int i;
 for(i=1;i<=10;i++)
 {
  if(i==6)
    continue;
  printf("\t %d",i);
  }
  getch(); } O/P: 1        2     3     4     5     7     8     9     10
```

The ‘goto’ statement

goto is a keyword by using goto we can pass the control anywhere in the program within the local scope.

- ‘goto’ always refers a identifier followed by colon is called label.
- Any valid identifier followed by colon is called label.
- When we are working with ‘goto’ it makes the program in unstructured manner.

Syntax:

statement 1;

```
statement 2;
goto label;
statement 3;
Label:
statement 4;
statement 5;
```

/*demonstrating 'goto' statement*/

```
#include<stdio.h>
#include<conio.h>
void main()
{

 clrscr();
 printf("Mahesh");
 goto ABC;
 printf("X");
 printf("Y");
 printf("Hello");
 ABC:
 printf("\t MCA");
 getch();
}
```

O/P: Mahesh   MCA

CHAPTER II: FUNCTIONS

What is function? Give its advantages.

Function: Self contain block of one or more statements which is designed for a particular task is called functions or sub program in an application called function.

Syntax:

```
return type function name (parameters)
    {
       statement body;
       -----------------
       -----------------
       return statement;
    }
```

Advantages:

- ☞ By using functions we can develop an application in module format.
- ☞ When we are applying the application in module format then easily we can develop easily we can debug and trace the program.
- ☞ By using functions large applications can be design like a smaller one.
- ☞ By using functions we can keep track of what they are doing.
- ☞ The basic purpose of function is code reuse.
- ☞ A C program is a collection of functions.
- ☞ From any function we can invoke (call) any another function.
- ☞ Always compilation process will starts from top to bottom.

☞ Always execution process will starts from main to ends only.

Distinguish between Library functions and User defined functions in C and Explain with examples.

In ‘C’ functions can be divided into two types:

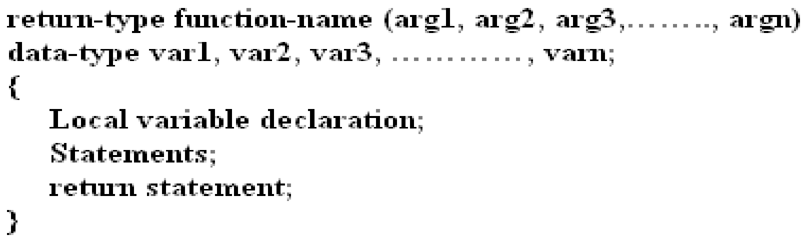
1. Library Functions
2. User-defined Functions

Library Functions: C provides library functions for performing some operations. These functions are present in the c library and they are predefined. Prototypes and Function definitions for library functions are available in some header files like math.h, stdio.h, conio.h etc.(scanf and printf are also predefined functions whose prototypes and definitions are available in stdio.h header file). Every compiler provides list of header files and their pototypes.

Examples: printf( ) scanf ( ) clrscr ( ) getch ( ) sqrt ( ) pow ( )

User-Defined Functions: User-defined functions are defined by the user according to their requirements.

Syntax 1:



```
return-type function-name (arg1, arg2, arg3,........, argn)
data-type var1, var2, var3, ............, varn;
{
    Local variable declaration;
    Statements;
    return statement;
}
```

To create and useuser define function we have to know these 3 elements.

1. Function Declaration

2. Function Definition

3. Function Call

1. Function declaration

The program or a function that calls a function is reffered to as the calling program or calling function. The calling program should declare any function that is to be used later in the program this is known as the function declaration or function prototype.

2. Function Definition

The function definition consists of the whole description and code of a function. It tells that what the function is doing and what are the input outputs for that. A function is called by simply writing the name of the function followed by the argument list inside the parenthesis. Function definitions have two parts:

Function Header

The first line of code is called Function Header.

int sum(int x, int y)

It has three parts

(i). The name of the function i.e. sum

(ii). The parameters of the function enclosed in parenthesis

(iii). Return value type i.e. int

Function Body

Whatever is written with in { } is the body of the function.

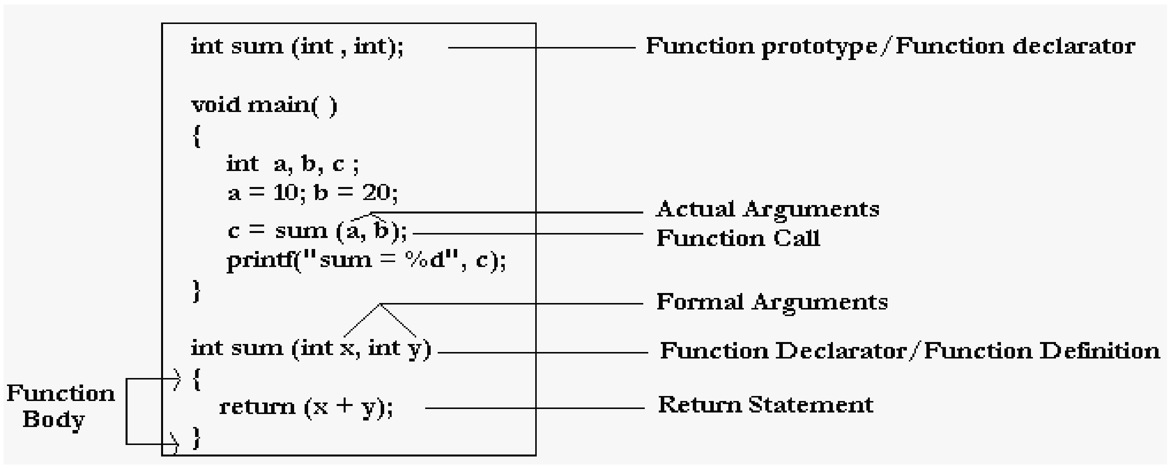
3. Function Call

In order to use the function we need to invoke it at a required place in the program. This is known as the function call.

1. What is a function and explain different parts of a function?

A complex problem can be divided into small and easily manageable parts. Each part can be called as a module. In a program a module is defined using a function. Self

contain block of one or more statements which is designed for a particular task is called functions.
A function accepts arguments from other functions, returns maximum a single value. A C program has any number of functions. But execution starts from the main (). All the functions other than the main are called sub function. Sub functions are executed when they are called directly or indirectly from the main ().



***Parts of function*:**
A function has the following parts:
1. Function prototype/declaration.
2. Definition of a function (function declarator)
3. Function call
4. Actual and Formal Arguments
5. The return statement.

1. Function Prototypes: A function prototype declaration consists of the function return type, name, and arguments list. It tells the compiler the name of the function, the type of value returned and the type and number of arguments. When the programmer defines the function, the definition of function must be like its prototype declaration. If the programmer makes a mistake, the compiler generates an error message. The function prototype declaration statement is always terminated with semi-colon.
Syntax:
<return type> function name (list of arguments type);
Example:
float adding(float,float);
2. Function Definition: The first line is called function declarator and is followed by function body. The block of statements followed by function declarator is called as function definition. The declarator and function prototype declaration should match each other. The function body is enclosed with curly braces. The function can be defined anywhere.
Syntax:
```
<return type> function name (List of formal arguments)
{
-------------------------
-------------------------
return <exp>;
}
```

Example:

```
float adding(float x,float y)
{
return x+y;
}
```

3. Function Call: A function call is a latent body. It gets activated only when a call to function is invoked. A function must be called by its name, followed by argument list enclosed in parenthesis and terminated by semi-colon.

Syntax:

```
<variable>=function name (List of actual arguments);
```

Example:

```
c=adding(a,b);
```

4. Actual and Formal Arguments: In function header whatever the variables we are creating those are called formal arguments or parameters.In fnction calling statement whatever the variables/data we are passing those variable are called actual arguments.

5. The return statement: The return statement is used to return value to other caller function. The return statement returns only one value at a time.

Syntax: return (variable-name); or return variable-name;

4. Explain the various categories of user defined functions in C with examples?

A function depending on whether arguments are present or not and whether a value is returned or not may belong to any one of the following categories:

(i) Functions with no arguments and no return values.

(ii) Functions with arguments and no return values.

(iii) Functions with arguments and return values.

(iv) Functions with no arguments and return values.

(i) Functions with no arguments and no return values:-

When a function has no arguments, it does not return any data from calling function. When a function does not return a value, the calling function does not receive any data from the called function. That is there is no data transfer between the calling function and the called function.

```
#include<stdio.h>
#include<conio.h>
void add();     //function declaration
void main()       //functions example program.
{
        clrscr();
        printf("welcome to functions..!");
        add();              //function calling.
        getch();
}
void add()                //function definition.
{
        int a,b;
        printf("\nenter a & b values: ");
        scanf("%d%d",&a,&b);
        printf("\naddition  : %d",a+b);
}
```

OUTPUT:

```
welcome to functions..!
enter a & b values: 30  20
addition  : 50
```

(ii) Functions with arguments and no return values:-

When a function has arguments data is transferred from calling function to called function. The called function receives data from calling function and does not send back any values to calling function. Because it doesn't have return value.

```
#include<stdio.h>
#include<conio.h>
void add(int,int);    //function declaration
void main()            //functions example program.
{
        int a,b;
        clrscr();
        printf("welcome to functions..!");
        printf("\nenter a & b values: ");
        scanf("%d%d",&a,&b);
        add(a,b);    //function calling.  call by reference.
        getch();
}
void add(int a,int b)        //function definition.
{

        printf("\naddition  : %d",a+b);
}
```

OUTPUT:

```
welcome to functions..!
enter a & b values: 30  20
addition  : 50
```

(iii) Functions with arguments and return values:-

In this data is transferred between calling and called function. That means called function receives data from calling function and called function also sends the return value to the calling function.

```
#include<stdio.h>
void main()
{
 int add_function(int x, int y);
 int a,b,sum;
 clrscr();
 printf("enter any two numbers: ");
 scanf("%d %d",&a,&b);
 sum=add_function(a,b);
 printf("the sum of %d and %d is %d: ",a,b,sum);
 getch();
}
int add_function(int x,int y)
{
 return(x+y);
}
```

OUTPUT

```
enter any two numbers: 50   30
the sum of 50 and 30 is 80:
```

(iv) Function With no Argument And Return Type:-

When function has no arguments data can not be transferred to called function. But the called function can send some return value to the calling function.

```
#include<stdio.h>
#include<conio.h>
int sub();
void main()  //functions example program.
{
        clrscr();
        printf("welcome to functions..!");
        printf("\n subtraction  : %d",sub());  //function calling
        getch();
}
int sub()
{
        int m,n;
        printf("\nenter m & n values: ");
        scanf("%d%d",&m,&n);
        return m-n;
}
```

```
welcome to functions..!
enter m & n values: 30  20
 subtraction  : 10
```

5. Write about different methods of sending parameters to the function?
Write how to send parameters to the function?
Write about pass by value and pass by address (reference).

Need of sending arguments:

The visibility or accessibility of local variables is limited to the function in which they are declared. It is mandatory to promote function to function communication in order to develop complex applications. Message passing in functions is done through sending parameters, arguments or parametric values.

Arguments passing techniques:

In C programming language are two types of parameters passing techniques are available i.e.

1. Pass by value or call by value.
2. Pass by references or call by reference.

CALL BY VALUE:

- ☞ It is a concept of calling a function by sending value type data.
- ☞ In call by value actual arguments and formal arguments both are value type variable.
- ☞ In call by value any modifications are happen on formal arguments then those changes will be not effected on actual arguments.

Ex: printf(), pow(), sqrt(),cos()

Example:

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

```
void fun1(int,int);
void main( )
{
int a=10, b=15;
fun1(a,b);
printf("a=%d,b=%d", a,b);
getch();
}
void fun1(int x, int y)
{
x=x+10;
y= y+20;
}
```

Output: a=10 b=15
The result clearly shown that the called function does not reflect the original values in main function.

CALL BY ADDRESS:

- ☞ It is a concept of calling a function by sending address type arguments.
- ☞ In call by address actual arguments are address type and formal arguments are pointer type.
- ☞ In call by address if any modifications are happen on formal arguments then those changes will be effected on actual arguments
- ☞ Calling function needs to pass '&' operator along with actual arguments and called function need to use '*' operator along with formal arguments. Changing data through an address variable is known as indirect access and '*' is represented as indirection operator.

Ex: scanf(), strcpy(), strlen()

```
#include<stdio.h>
#include<conio.h>
void fun1(int *,int *);
void main( )
{
int a=10,b=15;
fun1(&a,&b);
printf("a=%d,b=%d",a,b);
getch();
}
void fun1(int *x, int *y)
{
*x = *x + 10;
*y = *y + 20;
}
```

6. Define scope? Explain local and global variables with examples.

Scope is how far a variable can be accessed. The scope of variables is depends on location where a variable is declared.
There are 3 types of scopes in which a variable can fall:
1. Block scope
2. Function scope or local scope
3. Global scope or file scope

1. Block Scope:
A variable is said to have block scope, if it is recognised only with in the block where it is declared.The following example demonstrates this concept:

```
#include<stdio.h>
main()
{
{/*Block-1*/
int a;
a=10;
printf("%d\t%d",a,b);
}
{/*Block-2*/
int b;
b=20;
printf("%d\t%d",a,b);
}
}
```

2.***Scope of local variables:***
Variables declared within a function are called local variables. The visibility of these variables is limited to the home function in which they are declared, local variables belongs to one function can't be accessed from another function.
Say for example, variables declared within main() are local to main(), can't be accessed directly from other function like display(). In the same way, variables belongs to display() can't be directly accessed from the main().

```
/* scope of automatic variable */
#include<stdio.h>
int main()
{
int x=10;
display();
return 0;
}
void display()
{
printf("x=%d",x);
}
```

Output:
```
Error: undefined symbol "x" in function display()
```

3.Scope of global variables:
Variables declared outside the functions are called global variables. These variables are available along down to the program from their declaration. Global variables are visible only to the functions, which are down to their definition, can't be accessed from the functions above to their definition.

```
/* Scope of global variables */
#include<stdio.h>
int x=10; /* global variable definition */
void display();
int main()
{
```

```
printf("x=%d",x); /* printing global variable */
display();
printf("\nx=%d",x); /* printing global variable */
return 0;
}
void display()
{
x=x+50; /* changing global variable */
}
```

Output: x=10 x=60

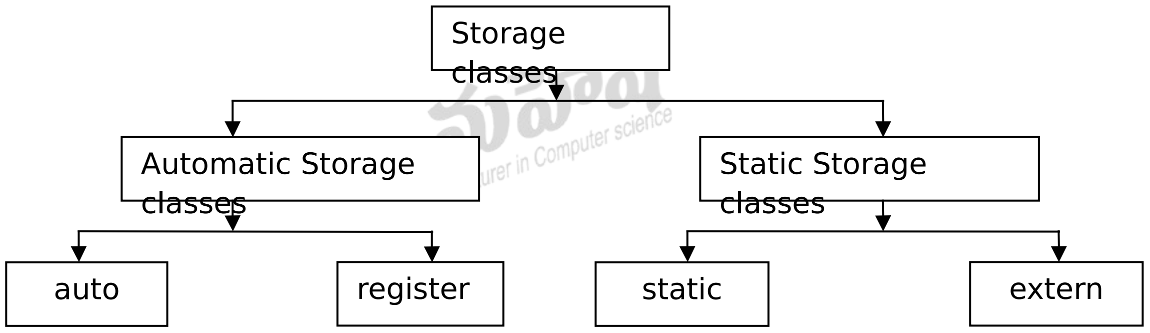
7. Write about Storage classes in C language.

Storage class provides information about their location and visibility. The storage class decides the portion of the program within which the variables are recognized.
Storage classes are provides following information to compiler. i. e.

- Storage area of a variable (Where the variables would be stored.
- Scope of a variable i.e. in which block the variable will be visible.
- Life time of a variable i.e. how long the variable will be here in active mode.
- Default value of a variable, if it is not initialized initial value.

In C programming language storage classs are classified into two types. i.e.

- Automatic storage classes.
- Static storage classes.



Automatic storage classes:

- This storage class variable will be created automatically and destroyed automatically.
- Automatic storage class variable will be hold in stack area of data segment or register of CPU.
- Under automatic storage class we having two types of storage class specifiers.

1. auto
2. register

Auto:

Variable declared inside the function or block without a storage class specification, by default compiler assigns 'auto' keyword and treated as automatic variable.

Ex : auto int x;

/* Example program for auto variables */

```
#include<stdio.h>
#include<conio.h>
void Fun();
void main()
```

```
{
        auto int x=100;
        clrscr();
  //++x;
        printf("\nValue 1:%d",x);
        Fun();
  getch();
}
void Fun()
{
        int y=20;
        printf("\nValue 2:%d",y);
}
```

Register:

It is a special kind of variable which stored in CPU register. Register variables are declared with the keyword ‘register’. The main advantage is that accessing is very fast when compared memory unit. In implementation we are using a variable throughout the program,then go for register variables.

/* Example program for register variables */

```
#include<stdio.h>
#include<conio.h>
void main()
{
        register int x;
        clrscr();
        x=10;
        printf("\nValue :%d",x);
  getch();
}
```

Static storage class:

- ☞ This storage class variable will be created only once and throughout the program it will there be there in active mode only
- ☞ Static storage class variable s will be hold in static area of data segment.
- ☞ Under static storage class we having two types of storage class specifiers.
  1. static
  2. extern

Static: Static variables are declared with the keyword ‘static’ either within the function or outside the function. The advantage of static variables to retain updated values between function calls.

Ex: static int x;

/* Example program for external static variables */

```
#include<stdio.h>
#include<conio.h>
void Fun();
static int x=10;
void main()
```

```
{
        int i;
        clrscr();
        for(i=1;i<=3;i++)
        {
        printf("\nValue :%d",x);
        Fun();
        }
   getch();
}
void Fun()
{
        x=x+5;
}
```

extern:

External storage class variables are declared outside the function with a keyword ‘extern’. Variable declared outside the function without a storage class specification, by default compiler assign ‘extern’ keyword and treated as external variables.

/* Example program for external variables */

```
#include<stdio.h>
#include<conio.h>
void Fun();
extern int x=100;
int y=200;
void main()
{
        clrscr();
        printf("\nValue
1:%d",x);
        Fun();
   getch();
}
void Fun()
{
        printf("\nValue 2:%d",y);
}
```

Type	**Scope**	**Life**	**Default value**
auto	Body	Body	Garbage value
register	Body	Body	Garbage value
static	Function	Program	0
extern	Program/function	Program	0

7. What is recursion and explain with any suitable example? (Or) Write program to accept any integer and print its factorial?

A function, which calls itself directly or indirectly again and again, is known as the recursive function. Recursive functions are very useful while constructing the data structures like linked lists, double linked lists and trees.

Iteration is the process of executing a statement or multiple statements repeatedly. Iterations can be constructed in “C” language by two ways that is by using the predefined iterative control structures and recursion. Recursive function will be involved by itself directly or indirectly as long as the given condition is satisfied. Some of the complex problems can be easily solved by implementing recursion.

Recursions can be classified into two types

Direct recursion: Function itself is clled direct recursion.

```
void sum()
{
 ………
 ………
 sum()
}
```

Indirect recursion: Function calls another function, which initiates to call the initial function is called indirect recursion.

```
void sum()
{
  ………
  ………
}
void call()
{
  ……….
  ……….
  sum()
}
```

/* program to accept any integer and print its factorial */

```
#include<stdio.h>
long factorial(int);
int main()
{
int n;
long fact;
printf("Enter an integer:");
scanf("%d",&n);
fact=factorial(n);
printf("Factorial of the number %ld",fact);
return 0;
}
long factorial(int n)
{
int f;
if(n==1)
return 1;
f=n*factorial(n-1);
return f;
}
```

UNIT – III

CHAPTER I: ARRAYS

1. What is array? Give its properties.

Definition: An array is a collection of memory locations which can share same data name and same data type values."

Syntax:

Data type array name [size];

Properties:

- ☞ An array is a collection of similar data type values in a single variable.
- ☞ An array is a derived data type in C which is constructed from fundamental data type of C language.
- ☞ In implementation when we require 'n' number of similar data type values then go for array.
- ☞ The array variables are created at the time of compilation.
- ☞ When we are working with arrays all elements will allocated in continuous memory location only.
- ☞ When we are working with arrays we can access the elements randomly. Because continuous memory locations are created.
- ☞ When we are working with arrays all elements will share same name with unique identification value called index. Which starts from **zero** and ends with 'size-1'?
- ☞ When we are working with arrays we need to use array subscript operator i.e. [ ]

2. Define array? write how to create,access and store elements into an array. Write about one dimensional array with example.

Array:

An array is a collection of memory locations which can share same data name and same data type values.

Every element in an array is identified with an index. The index starts from 0. The index of the last element is n-1, where n is the size of array. Every element in an array is free to participate in arithmetic, relational and logical operations.

Declaration of a single dimensional array*:* Declaring an array means specifying three things.

Data type: What kind of values it can store. For example int, char, float and double.

Array name: To identify the array

Size: The maximum number of values than the array can hold.

Syntax:

<Data type> <name of array> [<size>]

Example:

int m [5];

20	30	50	96	100
m[0]	m[1]	m[2]	m[3]	m[4]

Define an array by the name 'm' that can hold a maximum of 5 elements. The individual elements of an array are accessed and manipulated using the array name followed by their index. The marks stored in the first subject is accessed as m[0] and the marks scored in the 5th subject m[4].

Accessing data from an array:

To access elements from an array, index need to be changed from 0 to n-1 that can be done by using a common loop.

```
for(i=0;i<size;i++)
a[i]
```

Array initilization:

Initializing while declaration of an array:

We can assign a set of values of similar type to an array while its declaration. Assigning while declaration is called initializing. Elements of same type separated with comma (,) and placed in a pair of parenthesis { } can be directly assigned using assigning operator (=) to the array declaration statement. Values are assigned to the array in the same order as in the order of set.

To access elements from the keyboard and store into an array, index need to be changed from 0 to n-1 that can be done by using a common loop.

```
int i, int marks[5]
for(i=0;i<size;i++)
scanf("%d",&marks[i]);
```

Example:

```
int a[ ]={10,20,40,45,30,12}; /* size is optional */
#include<stdio.h>
int main()
{
int a[ ]={10,20,40,45,30,12};
int i;
printf("Elements of array:\n");
for(i=0;i<6;i++)
printf("%d\t",a[i]);
return 0;
}
```

3. Define array? Explain different methods to store values in an array?

Array:

An array is a collection of memory locations which can share same data name and same data type values.Every element in an array is identified with an index. The index starts from 0. The index of the last element is n-1, where n is the size of array.

Storing data into an array:

We can initialize the elements of the array in the same way as the ordinary variables when they are declared.Three ways to Store values in the arry . The are 1) initilize the elements during declaration 2)input values for the elements from keyboard 3) Assign values to individual elements.

Initilize values during declaration: Elements of an array can also be initilized at the time of declaration. Assigning while declaration is called initializing.In implementation when an array initilized we need to provide a value for every element in the array.

Syntax: datatype ArrayName[size] = {List of Values};

Here, List of Values is separated by comma operator.

```
int marks[5]={90,45,67,85,36};
int marks[4] = { 78,65};
int marks[ ] = {90,45,69,85};
```

Example:

```
int a[ ]={10,20,40,45,30,12}; /* size is optional */
#include<stdio.h>
int main()
{
int a[ ]={10,20,40,45,30,12};
int i;
printf("Elements of array:\n");
for(i=0;i<6;i++)
printf("%d\t",a[i]);
return 0;
}
```

Inputtng values from th ekey board: An array can be fillled by inputting values from the keyboard.

To access elements from the keyboard and store into an array, index need to be changed from 0 to n-1 that can be done by using a common loop.

```
int i, int marks[5]
for(i=0;i<size;i++)
scanf("%d",&marks[i]);
```

Assign values to individual elements: The third way is to assign values to individual elements of the array by using assignmnet operator.

Marks[3]=100;

Here 100 is assigned to the fourth element of the array which is specified as marks[3].

/* Program to read and write array elements */

```
#include<stdio.h>
int main()
{
int a[5];
int i;
printf("Enter 5 integers:\n");
for(i=0;i<5;i++)
scanf("%d",&a[i]);
printf("The elements are:\n");
for(i=0;i<5;i++)
printf("%d\t",a[i]);
return 0;
}
```

4. How do you calculate the length of an array in C language?

An array is a collection of memory locations which can share same data name and same data type values.

Every element in an array is identified with an index. The index starts from 0. The index of the last element is n-1, where n is the size of array. Every element in an array is free to participate in arithmetic, relational and logical operations.

In C language, there is no direct way to get how many elements are there in the array (length), because all the elements of array have the same size, dividing the total size of the array by the size of any one of the elements gives the number of elements in the array. Here we use element 0 because it is the only element guaranteed to exist, as arrays must have at least one element.

```
#include<stdio.h>
int main()
{
int a[5],len;
len=sizeof(a)/sizeof(a[0]);
printf("Length of array %d",len);
return 0;
}
```

5. Write about two dimensional arrays?

Two Dimensional Array

Collection of single dimensional array in a single variable is called two dimensional array or array (array).

- ☞ In 2D array elements are arranged in rows and columns.
- ☞ When we are working with 2D array we need to use two subscript operators which indicate **row size** and **column size**.
- ☞ The main memory of 2D array is row and sub memory is columns.
- ☞ In '**2D array**' array name always gives base address of the array i.e. first row base address, arr+1 is next main memory of the array. i.e. second row base address.

Declaration of matrix:

Syntax:

Data type array name [row size] [column size];

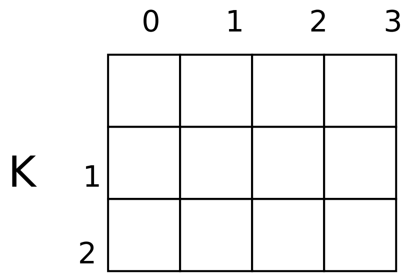
Example:

int a[3][5];

Here, every single dimensional array is considered as a separate row and elements are considered as columns.

For two dimensional arrays, memory is allocated in terms of table format that contains collection of rows and columns. So, that double dimensional arrays are useful for matrix representation of given elements.

For the above example, memory allocation will be:



Let 'm' is the row size and 'n' is the column size, then a double dimensional array can be defined as – "Double dimensional array is a collection of m x n homogeneous data elements that are stored in m x n successive memory locations".

Initializing a two dimensional array:

A set of sets of elements can be initialized while declaration of a two dimensional array. Here it is mandatory to specify the dimension while initializing a two dimensional array.

Example

```
        int num[5][2]= {        {10, 20},
                                {30, 40},
                                {50, 60},
                                {70, 80},
                                {90, 100},
                        };
```

Program for 2 Dimensional array

```
#include<stdio.h>
int main()
{
int a[ ][5]={{10,6,7,12,11},{23,32,14,52,22},{33,17,18,54,28}};
int i,j;
printf("Elements of matrix are:\n");
for(i=0;i<3;i++) /* selecting rows */
{
for(j=0;j<5;j++) /* traversing through the selected row */
printf("%5d",a[i][j]);
printf("\n\n"); /* Blank line between the rows */
}
return 0;
}
```

Output:

```
Elements of matrix are:
10 6  7  12 11
23 32 14 52 22
3317 18 54 28
```

6. How to send an array as an argument to the function?(Or)
How to send a single dimensional array as an argument to the function? (Or)
How to send a two dimensional array as an argument to the function?

One dimensional array for inter-function communication

Like a normal variable even an array can be send as an argument into the sub function. Here the name of array is given as an actual argument with the calling statement. An array is defined as a formal argument with the signature of function definition. Elements of actual argument (original array) can be directly accessed by the formal argument from the sub function.

In case of single dimensional array, it is optional to specify the dimension (size) with the formal argument and prototype and mandatory in case of two or multidimensional arrays.

While sending an array as an argument, even the size 'n' need to be send as an argument to control the loop in sub function.

/* sending a single dimensional array as argument */

```
#include<stdio.h>
void display(int[],int)
int main()
{
 int a[ ] = {14,41,32,76,77,89}
 display(a,6);
 return 0;
}
void display(int p[], int n)
{
  int i;
  printf("elements of array : \n");
 for( i=0;i<n;i++)
printf("\t %d",p[i]);
}
```

20	30	50	96	100

/* sending a two dimensional array as argument */

```
#include<stdio.h>
void display(int[][],int,int);
int main()
{
int a[3][5]={{10,6,7,12,11},{23,32,14,52,22},{33,17,18,54,28}};
display(a,3,5); /* sending matrix as argument */
return 0;
}
void display(int b[3][5],int n,int m)
{
        int i,j;
        printf("Elements of matrix..\n");
        for(i=0;i<n;i++)
 {
   for(j=0;j<m;j++)
   printf("%5d",b[i][j]);
  printf("\n");
 }
}
```

7.Discuss about the operations performed on arrays

There are a number of operations that can performed on arrays. These operations include:

- ☞ Traversing an array

☞ Inserting an element in an array
☞ Deleting an element from an array
☞ Merging two arrays
☞ Searching an element in an array
☞ Sorting an array in ascending or descending order

Traversing an array: Traversing an array means accessing each and every element of the array for a specific purpose.

Inserting an element in an array: Inserting an element in an array means adding a new data elements to an already existing array.

Deleting an element from an array: Deleting an element from an array means removing a data element from an already existing array.

Merging two arrays: Merging two arrays in a third array means first copying the elements of the first array into the third array and then copying the contents of the second array into the third array.

Searching an element in an array: searching means to find whether a particular value is present in the array or not. If the value is present in the array then searching is said to be successful. And if the value is not present in the array searching is said to be unsuccessful.

There are two popular methods for searching the array elements. i-e linear search and binary search.

Sorting an array in ascending or descending order: to arrange the elements in ascending or descending order.

CHAPTER – II STRINGS

7. What is string? How to declare and initilize strings?

☞ Group of characters or collection of characters or character array is called string.
☞ Within the single quotation of characters is called character constant. i.e ‘G’, ‘M’ etc.
☞ Always character constant returns an integer value i.e. ASCII value of the character.
☞ Within the double quotation any content is called string constant. i.e. “Mahesh”, “MCA” , “Mahesh soft solutions” etc.
☞ String constant always ends with null character. And null character representation is ‘\0’ and ASCII value is 0.

☞ A string is a one dimensional array of characters terminated by a null (\0) character. The null character is stored at the end of array of characters to mark the end of the string.

Declaring a String Variable:

To manipulate strings, an array must be declared with character data type. A string can be declared as follows:

Syntax: char arrayname[size];

Here the 'size' is the number of characters in the array.

Example: char name[40]; char city[15]; etc.

Initializing a String Variable:

1. A string of characters can be stored in an array as follows:

Ex: 1. char name[7] = {'M','A','H','E','S,'H",'\0'};

2. char city[7] = "NELLORE"; **(OR)** char city[] = "NELLORE";

In the example-2, the null character is not needed. The compiler automatically inserts '\0' character at the end. So, the conceptual view for the above examples is as shown below:

N	E	L	L	O	R	E	\0

Array of Strings (or) Two-dimensional character type array:

To store more than one string in the array, then the array can be declared as follows:

char arrayname[size1][size2];

Here, '**size1**' specifies the number of strings and '**size2**' refers the number of characters in each string.

Example: char city[3][10] = {"NELLORE","VIJAYAWADA","AMARAVATHI"};

City[0]	N	E	L	L	O	R	E	\0			
City[1]	V	I	J	A	Y	A	W	A	D	A	\0
	A	M	A	R	A	V	A	T	H	I	\0

City[2]

This example reserves three memory locations to store three strings. Each string may contain ten characters (10 bytes). The conceptual view for the above declaration is above table.

8. Explain how string is accepted and accessed (Reading/Writing) with an example?

As C language has no special type to handle string, it is mandatory to stored in a character array but, it is a difficult job to read and write a string character by character using a loop every time. C language simplifies the process of reading and writing of string through printf() and scanf().

Reading a string:

It is enough to specify the address of character array with its name and a format specifier %s to the scanf() to read and store the string into the character array. The scanf() automatically reads the

string character by character, stores into the character array and a terminating character '\0' is automatically added at the end.
Example:

```
char x[50];
scanf("%s",x);
```

Writing a string:
Similar to scanf(), if we give the address of character array with its name and a format specifier %s to the printf(), the printf() automatically prints the string character by character until the terminating character '\0'.
Example:

```
printf("%s",x);
```

/* program to read and write a string */

```
#include<stdio.h>
int main()
{
char x[50];
int i;
printf("Enter any string:");
scanf("%s",x);
printf("The given string:");
printf("%s",x);
return 0;
}
```

Execution:

```
Enter any string: Mahesh
The given string: Mahesh
```

9. Write about different operations performed on a string? (Or) Write about any 5 string handling functions in C language?

In C language, there are several function are used to manipulate strings. These functions are included in "string.h" file. The following are some of the most commonly used string functions:
1. Finding the length of a string:
Strlen():

- ***strlen()*** is a predefined function in the header file "string.h"
- By using this function we can find the length of the string.
- Strlen() function required one argument of type const char* and from given address it returns length of the string.
- Length of the string is total number of characters including null.

Syntax: strlen(string)

Program for Finding the length of a string

```
#include<stdio.h>
#include<string.h>
int main()
{
        char x[50];
        int l;
        printf("Enter any string:");
        scanf("%s",x);
        l=strlen(x);
        printf("The length of string %d",l);
        return 0;
}
```

Execution:

```
Enter any string: Mahesh
The length of string 6
```

2. Reversing a string:

Strrev():

- ***strrev()*** is a predefined function defined within the header file “string.h”.
- By using strrev function we reverse the given string.
- Strrev function require one argument of type char*.
- from given address up to null entire content will be arrange in reverse order.

Syntax: strrev(str)

Program for Reversing the length of a string

```
#include<stdio.h>
#include<string.h>
int main()
{
        char x[50];
        int i;
        printf("Enter any string:");
        scanf("%s",x);
        strrev(x);
        printf("The reverse string %s",x);
        printf("%s",x);
        return 0;
}
```

Execution:

```
Enter any string: Mahesh
The reverse string hsehaM
```

3. Copying a string:

Strcpy ():

- ***strcpy()*** is a predefined function defined with in the header file “string.h”
- By using ‘strcpy’ function you can copy a string into another string.
- ‘strcpy’ function require two arguments of type char* i.e an address of a string.
- When we are working with strcpy function from given address all source string content will be copied to destination string position.

Syntax: strcpy(string1, string2);

Here ‘string1’ is destination string and ‘string2’ is source string. The contents of string2 are copied to string1.

Program for copying a string

```
#include<stdio.h>
#include<string.h>
int main()
{
        char x[50],y[50];
        printf("Enter any string:");
        scanf("%s",x);
        strcpy(y,x); /* "x" is copied onto "y" */
        printf("%s",y);
        return 0;
}
```

Execution:

```
 Enter any string: Mahesh
 Mahesh
```

4. Concatenating a string to another:

Strcat( ) :

- ☞ ***strcat()*** is a predefined function defined within the header file "string.h"
- ☞ This function appends the contents of one string (source) to another string (destination).
- ☞ This is called concatenation of two strings.
- ☞ The contents of the source string are unchanged.

Syntax: strcat(string1, string2);

Where 'string1' is destination string and 'string2' is source string. The contents of string2 are appended to string1.

Program for concatenating a string

```
#include<stdio.h>
#include<string.h>
int main()
{
        char x[50],y[50];
        printf("Enter the first string:");
        scanf("%s",x);
        printf("Enter the second string:");
        scanf("%s",y);
        if(strcmp(x,y)==0)
        printf("Equal");
        else if(strcmp(x,y)>0)
        printf("Biggest string %s",x);
        else
        printf("Biggest string %s",y);
        return 0;
}
```

Execution:

```
Enter the 1st string: hello
Enter the 2nd string: sir
The resultant string is hellosir
```

5. Comparing any two strings:

Strcmp( ) Function:

- ☞ ***strcmp()*** is a predefined function defined within the header file "string.h"

☞ This function is used to compare two strings.

Syntax: strcat(string1, string2); Here,

1. If string1 is equal to string2 then it returns value 0
2. If string1 is less than string2 then it returns a negative (less than 0) value.
3. If string1 is greater than string2 then it returns positive (greater than 0) value.

Program for comparing any two strings

```
#include<stdio.h>
#include<string.h>
int main()
{
char x[50],y[50];
printf("Enter the first string:");
scanf("%s",x);
printf("Enter the second string:");
scanf("%s",y);
if(strcmp(x,y)==0)
printf("Equal");
else if(strcmp(x,y)>0)
printf("Biggest string %s",x);
else
printf("Biggest string %s",y);
return 0;
}
```

Execution:

```
Enter the first string: hello
Enter the second string: world
Biggest string: world
```

10. Write a program to convert a character of a string to upper case?

```
#include<stdio.h>
#include<string.h>
int main()
{
char s[50];
int pos;
printf("Enter any string in lower case..\n");
scanf("%s",s);
printf("Enter the position:");
scanf("%d",&pos);
if(pos<1||pos>strlen(s))
printf("Invalid position");
else
{
pos--;
s[pos]=s[pos]-32; /* converting a character to upper case */
printf("The resultant string is %s\n",s);
}
return 0;
}
```

Execution:

```
Enter any string in lower case..
america
Enter the position:3
The resultant string is amErica
```

11. Write a program to convert a character of a string to lower case?

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

```
#include<string.h>
int main()
{
char s[50];
int pos;
printf("Enter any string in upper case..\n");
scanf("%s",s);
printf("Enter the position:");
scanf("%d",&pos);
if(pos<1||pos>strlen(s))
printf("Invalid position");
else
{
pos--;
s[pos]=s[pos]+32; /* converting a character to lower case */
printf("The resultant string is %s\n",s);
}
return 0;
}
```

Execution:

```
Enter any string in lower case..
AMERICA
Enter the position:3
The resultant string is AMeRICA
```

Discuss about string and character functions

String manipulation functions that are part of ctype.h, string.h and stdlib.h. some charcter functions contained in ctype.h. "ctype.h" header file support all the below functions in C language.

isalpha( ) function: it checks whether given character is alphabetic or not.

Syntax: int isalpha (int x);

isdigit( ) function: it checks whether given character is digit or not.

Syntax : int isdigit (int x);

isalnum() function: it checks whether given character is alphanumeric or not.

Syntax: int isalnum (int x);

isspace( ) function: it checks whether given character is space or not.

Syntax: int isspace(int x);

islower( ) function: it checks whether given character is lower case or not.

Syntax : int islower(int x);

isupper( ) function: It checks whether given character is upper case or not.

Syntax : int isupper (int x);

isxdigit( ) function: It checks whether given character is hexadecimal or not.

Syntax : int isxdigit(int x);

tolower( ) function: It checks whether given character is alphabetic and converts to lowercase.

Syntax : int tolower(int x);

toupper( ) function: It checks whether given character is alphabetic and converts to uppercase.

Syntax : int toupper(int x);

C STRING FUNCTIONS:

String.h header file supports all the string functions in C language. All the string functions are given below.

strcat( ) function: It concatenates two given strings. It concatenates source string at the end of destination string.

Syntax :char * strcat (char * destination, const char * source);

Example:
strcat (str2, str1); – str1 is concatenated at the end of str2.
strcat (str1, str2); – str2 is concatenated at the end of str1.

strncat( ) function: It concatenates (appends) portion of one string at the end of another string.

Syntax :char * strncat (char * destination, const char * source, size_t num);

Example :
strncat (str2, str1, 3); – First 3 characters of str1 is concatenated at the end of str2.
strncat (str1, str2, 3); – First 3 characters of str2 is concatenated at the end of str1.

strcpy( ) function: It copies contents of one string into another string.
Syntax :char * strcpy (char * destination, const char * source);
Example:
strcpy (str1, str2) – It copies contents of str2 into str1.
strcpy (str2, str1) – It copies contents of str1 into str2.

strlen( ) function: It gives the length of the given string.

Syntax :size_t strlen (const char * str);

strlen( ) function counts the number of characters in a given string and returns the integer value.

It stops counting the character when null character is found. Because, null character indicates the end of the string in C.

strcmp( ) function: It compares two given strings and returns zero if they are same.

If length of string1 < string2, it returns < 0 value. If length of string1 > string2, it returns > 0 value.
Syntax :int strcmp (const char * str1, const char * str2);

CHAPTER I: POINTERS

1. What is pointer and write how to define a pointer and list few advantages?

A pointer is a variable which holds address of another variable. (Or)
A pointer is a derived data type in C which is constructed from fundamental data type of C language.

Advantages:

- Normal variable stores the value whereas pointer variable stores the address of the variable.
- Pointers save the memory allocation.
- It promotes direct memory accessing, which improves the performance of the program.
- It allows to directly access the array elements using pointer arithmetic's.
- Pointer makes possible to return multiple values to the calling function.
- Pointer is the key concept in using complex data structures.
- By using pointers we can access a variable which is define outside the function.
- By using pointers we can handle the data structures more effectively.
- When we are working with the pointers it can increase the execution speed.

<u>**Declaring a pointer variable**</u>: A pointer variable should be declared before using it. The pointer variable is declared as follows

Syntax: DtataType *ptr_variable;

<data type of original variable> *<name of pointer>

int *p;

When we are working with pointers, we are using following operators.

& (Address of operator) : & symbol is used to get the address of the variable.
*** (Dereference operator)** : * symbol is used to get the value of the variable that the pointer is pointing to. It is used to access the value of memory allocation indirectly through its address.

```
#include <stdio.h>
#include<conio.h>
int main()
{
int *p; /* declaration of pointer */
int x=345; /* declaration of variable */
p=&x; /* assigning the address to the pointer */
printf("x=%d",x); /* direst accessing */
printf("\nx=%d",*(&x)); /* indirect access through the address */
printf("\nx=%d\n",*p); /* indirect access through the pointer */
return 0;
}
```

2.Define pointer. Discuss about pointer expressions and pointer arithmetics.

A pointer is a variable which holds address of another variable.Like other variables pointer variables can be used in expressions.

We can perform arithmetic operations on the pointers like addition, subtraction, etc. However, as we know that pointer contains the address, the result of an arithmetic operation performed on the pointer will also be a pointer if the other operand is of type integer. In pointer-from-pointer subtraction, the result will be an integer value. Following arithmetic operations are possible on the pointer in C language:

- Increment
- Decrement
- Addition
- Subtraction

Incrementing Pointer in C

If we increment a pointer by 1, the pointer will start pointing to the immediate next location. This is somewhat different from the general arithmetic since the value of the pointer will get increased by the size of the data type to which the pointer is pointing.The Rule to increment the pointer is given below:

new_address= current_address + i * size_of(data type)

```
#include<stdio.h>
int main(){
int number=50;
int *p;//pointer to int
p=&number;//stores the address of number variable
printf("Address of p variable is %u \n",p);
p=p+1;
printf("After increment: Address of p variable is %u \n",p); // in our case, p will get incremented b
y 4 bytes.
return 0;
}
```

Decrementing Pointer in C

Like increment, we can decrement a pointer variable. If we decrement a pointer, it will start pointing to the previous location. The formula of decrementing the pointer is given below:

new_address= current_address - i * size_of(data type)

```
#include <stdio.h>
void main(){
int number=50;
int *p;//pointer to int
p=&number;//stores the address of number variable
printf("Address of p variable is %u \n",p);
p=p-1;
printf("After decrement: Address of p variable is %u \n",p); // P will now point to the immidiate
      previous location.
}
```

C Pointer Addition

We can add a value to the pointer variable. The formula of adding value to pointer is given below:

```
new_address= current_address + (number * size_of(data type))
#include<stdio.h>
int main(){
int number=50;
int *p;//pointer to int
p=&number;//stores the address of number variable
printf("Address of p variable is %u \n",p);
p=p+3;   //adding 3 to pointer variable
printf("After adding 3: Address of p variable is %u \n",p);
return 0;
}
```

C Pointer Subtraction

Like pointer addition, we can subtract a value from the pointer variable. Subtracting any number from a pointer will give an address. The formula of subtracting value from the pointer variable is given below:

```
   new_address= current_address - (number * size_of(data type))
#include<stdio.h>
int main(){
int number=50;
int *p;//pointer to int
p=&number;//stores the address of number variable
printf("Address of p variable is %u \n",p);
p=p-3; //subtracting 3 from pointer variable
printf("After subtracting 3: Address of p variable is %u \n",p);
return 0;
}
```

3.Explain Null pointer and generic pointer.

NULL POINTER

Which pointer variable is initialized with null it is called null pointer. Null pointer will be not any location.

A null pointer is a special type of pointer that cannot points to any where. Null pointer is assigned by using the predefined constant NULL; which is defined by several header files including stdio.h, stdlib.h and alloc.h.

Example: **int *p=NULL;**

/* EXAMPLE PROGRAM FOR NULL POINTER */

```
#include<stdio.h>
main()
{
        int *p;
        clrscr();
        p=NULL;
        printf("\nValue :%d",*p);
}
```

VOID POINTER

- Generic pointer of c and c++ is called void pointer. A void pointer is a special type of pointer that can points to any data type.
- By using void pointer any datatype address can be hold and became access, manipulate the data properly.
- Generic pointer means it can applied on any data type because type specification will be exist at run time only.
- The size of void pointer is 2B
- When we are working with void pointers mandatory to use type casting process for accessing the data.
- On void pointer we can't applied arithmetic operations. i.e. incrementation of the pointer is not possible.

Syntax: void *ptrvariable;

At the time of assigning address of the variable to ptrvariable, type casting must be placed to refer the data item.

/* EXAMPLE PROGRAM FOR VOID POINTER */

```
main()
{
        int a=10;
        double b=3.45678;
        void *p;
        clrscr();
        p=&a;
        printf("\nValue 1:%d",*(int *)p);
        p=&b;
        printf("\nValue 2:%lf",*(double *)p);
}
```

4.Discuss about pointers and arrays
Write about the pointer representation to an array?

The elements of an array can be efficiently accessed by using pointers. 'C' Language provides two methods of accessing array elements. They are pointer arithmetic method and array indexing method. However, pointer arithmetic will be faster.

To access array elements, the memory address of first element (base address) of an array can be assigned to the pointer variable. Using this address we can access the remaining elements of that array quickly.

For example,

```
int a[10], *p;
p = &a;
```

here, it assigns the base address of the array variable ‘a’ to the pointer variable ‘p’. Now to access element of a[4], we can write either a[4] or a+4 or *(p+4).

Example:

```
main( )
{
  int a[20],i , n, *p;
  clrscr( );
  printf(“How many values “);
  scanf(“%d”,&n);
  p = a;                              /* assigns base address of ‘a’ to ‘p’ */
  for(i=0;i<=n-1;i++)
      scanf(“%d”,(p+i));      /* inputs value to array thru pointer    */
  for(i=0;i<=n-1;i++)
      printf(“%d”,*(p+i));   /* displays value of array thru pointer  */
}
```

Array of Pointers:

A pointer can also be declared as an array. As pointer variable always contains an address, an array of pointers contains a collection of addresses.

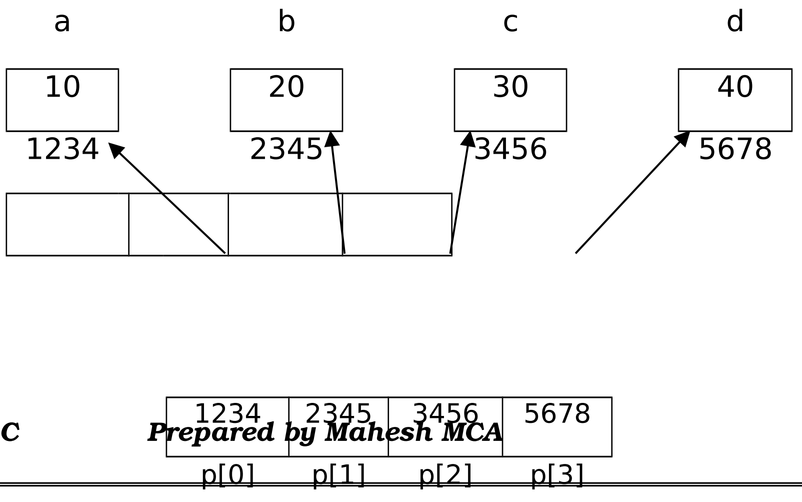
Syntax: **datatype *arrayname[size];**

Example:

```
main( )

{
    int a=10, b=20, c=30, d=40;
    int *p[4];
    p[0] = &a;
    p[1] = &b;
    p[2] = &c;
    p[3] = &d;
}
```

In the above example, the pointer variable ‘p’ declared as an array with 4 elements. Hence, it can hold 4 memory addresses of integer variables i.e. a, b, c and d. This can be as shown below:



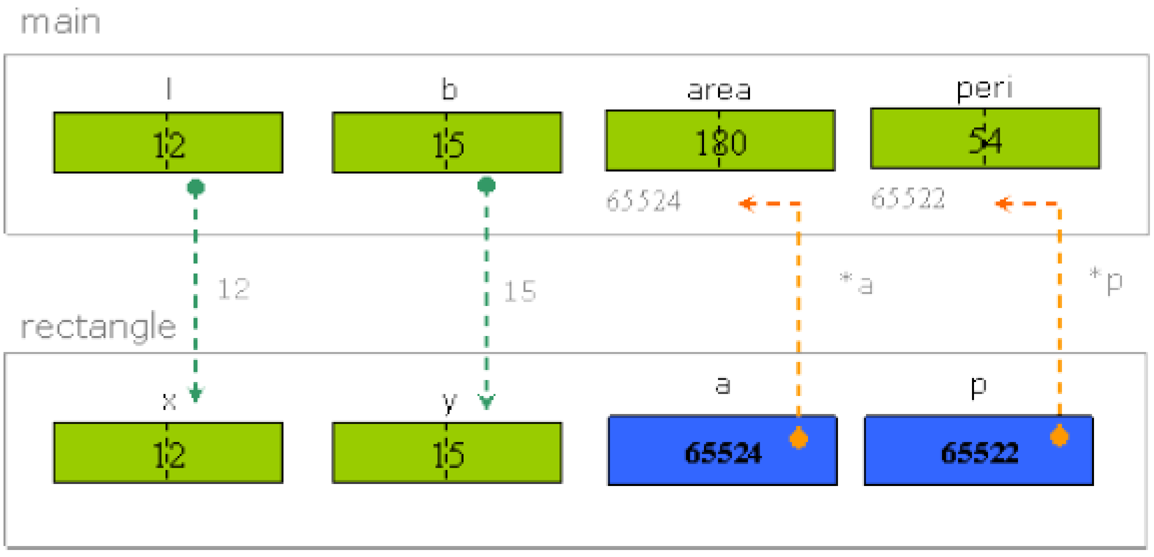
5. How do you pass arguments to the functions using pointers?

When we are working with C language any number of arguments can be send as arguments to a function but only a single value can be returned to the calling function. It limits the flexibility of application development. It can be overcome by pass by address.

Pass by address (Reference)

In this method addresses of actual arguments are sent to the function. Here the formal arguments are the pointers hold addresses of actual arguments. By using indirect operator (*), formal arguments can directly access the actual arguments of calling function. By using this method, it is possible to send any number of values to the calling functions.

```
#include<stdio.h>
#include<conio.h>
void rectangle(int,int,int*,int*);
int main()
{
int l,b,area,peri;
printf("Enter two sides of rectangle:\n");
scanf("%d%d",&l,&b);
rectangle(l,b,&area,&peri);
printf("Area %d",area);
printf("\nPerimeter %d",peri);
return 0;
}
void rectangle(int x,int y,int *a,int *p)
{
*a=x*y;
*p=2*(x+y);
}
```



6. Write how to pass an array as an argument using pointer?

Like a normal variable even an array can be send as an argument into the sub function. Here the name of array is given as an actual argument with the calling statement.

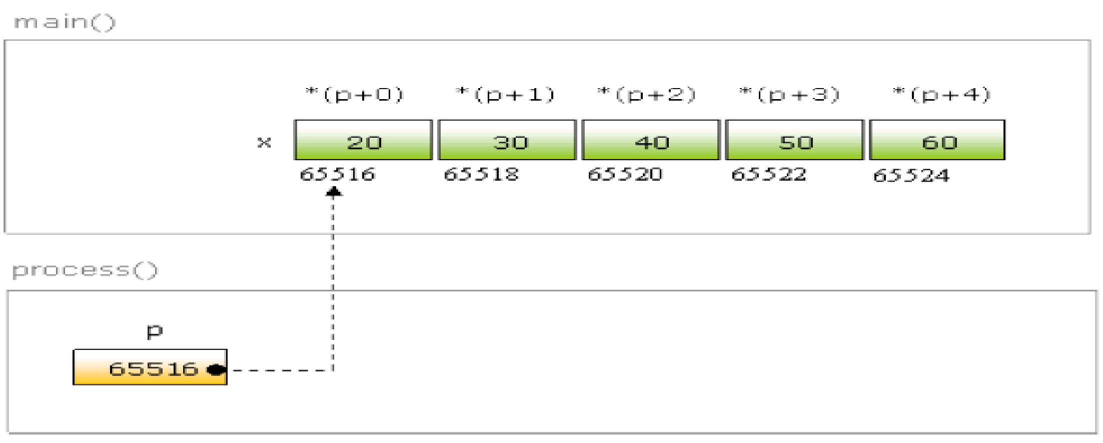
When we write the name of array as actual argument, it is the address of array that is being passed as argument. The formal argument must be a pointer to store the address. Now the pointer could refer every element of array from the called by function using pointer and address arithmetic

It is the reason why in case of array change formal argument results the change in actual argument.

```
#include<stdio.h>
void process(short[],int);
int main()
{
short x[]={12,34,54,55,62,67};
short i;
process(x,6);
printf("Elements of array:\n");
for(i=0;i<6;i++)
printf("%5d",x[i]);
return 0;
}
void process(short *p,int n)
{
short i;
for(i=0;i<n;i++)
*(p+i)=*(p+i)+10;
}
```

Output:

```
Elements of array: 22 44 64 65 72 77
```



8. Dynamic Memory Allocation (DMA)

☞ DMA is a concept of allocation or de-allocating the memory at runtime. i.e. dynamically.By using DMA we can utilize the memory efficiently.By using DMA whenever we want which type, we want how much, we want that time, that type, and that much we can allocate dynamically.

☞ DMA related functions are available in following header files.

<alloc.h>
<stdlib.h>
<malloc.h>
<mem.h>

Memory Management Functions:

malloc() Function:

It allocates memory space to a variable. The space must be specified in the form of bytes. This function returns NULL if the allocation of memory fails. It means, if the memory is not sufficient to allocate then it returns NULL.

The general format of malloc( ) function is as follows:

Syntax: pv = (type *) malloc(size);

Here 'pv' is pointer variable and 'type' is the data type of the variable. The size is a numeric constant or expression that indicates the number of bytes allocated to the variable.

Example-1:

```
int *p;
p = (int *) malloc(4);
```

Here, it allocates 4 bytes of memory to the variable 'p'. hence we can use the variable 'p' as two elements of 'int' type.

Example-2:

```
int *p, n=5;
p = (int *) malloc(n*2);
```

This example allocates memory for 5 integers of two bytes each i.e. 10 bytes.

calloc() Function:

This function is also used to allocate space to a variable. But it initializes all the elements with zero.

Syntax:

pv = (type *) calloc(n, size);

Here 'pv' is pointer variable and 'type' is the data type of the variable. The 'n' is number of elements and the size specifies the number of bytes for each element.

Example-1:

```
int *p;
p = (int *) calloc(5, 2);
```

Example-2:

```
int *p, n=5;
p = (int *) calloc(n, 2);
```

free()function:

This function is used to release (remove) the memory allocated dynamically. When the memory space is not required, then this function can be used.

Syntax: free(fp); where fp is file pointer

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
int main()
{
  int*arr;
  int size,i;
  clrscr();
  printf("\nEnter no of Elements: ");
  scanf("%d",&size);
  arr=(int*)malloc(sizeof(int)*size);
  printf("\nDefault values: ");
  for(i=0; i<size; i++)
   printf("%d ",*(arr+i));
  //printf("%d ",arr[i]);
  printf("\nEnter %d values: ",size);
  for(i=0; i<size; i++)
  {
   scanf("%d",&arr[i]);
  }
  printf("\nArr list is: ");
  for(i=0; i<size;i++)
    printf("%d ",arr[i]);
  getch();
  free(arr);
  return 0;
}
```

```
Enter no of Elements: 5

Default values: 30089 9762 23948 -29916
11780
Enter 5 values: 10      30      50      40      90
30

Arr list is: 10 30 50 40 90
```

In implementation when we need to deallocate more than 64KB data then go for free function.

CHAPTER II: STRUCTURE, UNION, AND ENUMERATED DATA TYPES

1. Explain the concept of Structures in C (or) What is structure and write a program to access and print student details?

- ☞ Structure is a collection of different data type variables in a single entity.
- ☞ Structure is a collection of primitive and derived data type variables.
- ☞ All predefine data types are designed for basic operations only. i.e. it can work for basic data types.
- ☞ In implementation whenever the primitive data types are not supporting user requirement then go for structures.
- ☞ By using structures we can create user defined data types.
- ☞ The size of the structure is sum of all number variables.
- ☞ The least size of structure is 1 byte.
- ☞ By using '**struct**' keyword we can create structures.

<u>Defining a Structure:</u>

The keyword 'struct' is used to define a structure. The structure definition is as follows:

Syntax to create the structure:

```
Struct  Tag_name(identifier)
{
        Data type1 member1;
        Data type2 member 2;
        ………………….
};
```

Note: When we are constructing the structure body should be ended with semicolon. Because semicolon only that structure body or an entity is terminated.

In the above syntax,

- ☞ The " identifier" is the name of the structure
- ☞ Member1, member2, …, member-n are the individual members of the structures. These are nothing but variables.
- ☞ Datatype refers the type of each individual member

Example

```
Struct emp
{
        int Id;
        char name[36];
        int sal;
};
```

<u>Accessing Structure Elements:</u>

The individual structure elements(members) can be referred(accessed) by, specifying the structure type variable name followed by a dot(.) operator and followed by a member name. In the above example

```
e.id;
e.name;
e.sal;
```

Syntax to initialize the structure variables:

Structure tagname var={value1, value2…….};

Example:

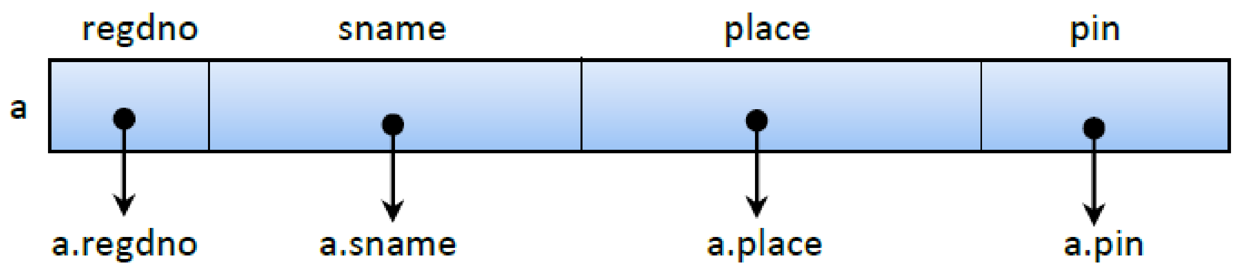
emp e={105053011,Mahesh,25000}

Example program

```
#include<iostream.h>
#include<conio.h>
struct employee
{
  int eno;
  char ename[30];
  float sal;
};
void main()
{
 struct employee e={105053011,"mahesh",25000};
 clrscr();
 printf("Employee Number: ",&e.eno);
 printf("Employee Name: ",&e.ename);
 printf("Employee salary: ",&e.sal);
 getch();
 }
```

2. Write a c program to access and print student details using structures.

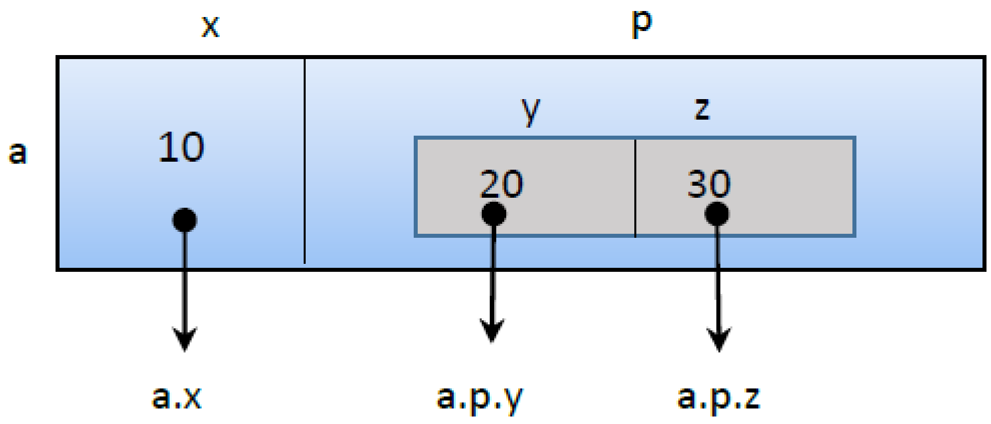
```
struct student
{
int regdno;
char sname[50];
char place[50];
long int pin;
};
int main()
{
struct student a;
printf("Regd Number:");
scanf("%d", &a.regdno);
printf("Student Name:");
scanf("%s", a.sname);
printf("Place:");
scanf("%s", a.place);
printf("PinCode:");
scanf("%ld",&a.pin);
printf("Address:\n");
printf("Regd No:%d\nStudent Name:%s\nPlace:%s\nPin Code:%ld\n",a.regdno,a.sname,a.place,a.pin);
return 0;
}
```



3. Write about the nesting of structures?

Like a variable belongs to any primitive type, array even a variable belongs to another structure can be defined as a member of structure. Variable belongs to structure would have the memory allocation of inner structure members.

```
#include<stdio.h>
struct data
{
int x;
struct
{
int y;
int z;
}p;
};
int main()
{
struct data a;
a.x=10;
a.p.y=20;
a.p.z=30;
printf("x=%d\ny=%d\nz=%d",a.x,a.p.y,a.p.z);
return 0;
}
```



4. Write how to send a structure variable as an argument?

One of the main reasons of using struct type is to send all the details of a single entity as single group rather sending multiple values to a function

While sending a struct variable as argument, we specify the name of struct variable as actual argument and another struct variable is defined as a formal argument, so that, all the members of actual argument will be assigned to the formal argument.

It is important to note that sending struct variable as argument follows pass-by-value rather pass-by-reference. So the change in the members of formal argument will not change in the values of actual argument.

```
struct book
{
char name[50];
char author[50];
char publisher[50];
float price;
};
void display(struct book);
int main()
{
struct book x={"Let us C","Kanithkar","BpB",275};
display(x);
return 0;
}
void display(struct book a) /* members of x would be assigned to a */
{
printf("Book Name:%s",a.name);
printf("\nAuthor: %s",a.author);
printf("\nPublisher: %s",a.publisher);
printf("\nPrice:%f",a.price);
}
```

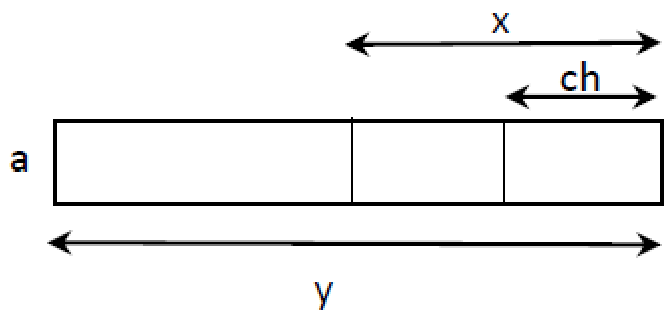
Output: Book Name:Let us c Author: Kanithkar Publisher: BpB Price:275.000000

5. What is union and why it is used?

Union is a user defined type. It looks similar to structure, but the functionality differs. When a variable belongs to union is created, it allocates the common memory allocation to all the members. The size of union variable is equal to the size of member with maximum length. All the members share the same memory. Hence it is not possible to use all the members of a union variable at a time. Union is mostly used as a generic type.

```
#include<stdio.h>
union num
{
char ch;
int x;
float y;
};

int main()
{
union num a;
printf("Size of union variable %d Bytes\n",a);
a.ch='p';
a.x=12356;
a.y=678.65310;
printf("ch=%c\n",a.ch);
printf("x=%d\n",a.x);
printf("y=%f\n",a.y);
return 0;
}
Output:
ch=
x=-239878
y=678.65310
```



By the above output it is proved that last assigned value (678.65310) is overwritten on previous values. a.ch prints last byte binary equal of 678.65310, a.x prints last two bytes binary equal of 678.65310.

Union as a generic type:

A union can have different members of different types. At a time we can store any single value but of any member type. So any variable belongs to union is capable to store the data belongs to any type called generic type.

```
#include<stdio.h>
union num
{
char ch;
int x;
float y;
};
int main()
{
union num a,b,c;
a.y=234.768956;
b.x=615;
c.ch='s';
printf("x=%f",a.y);
printf("\ny=%d",b.x);
printf("\nz=%c",c.ch);
return 0;
}
```

Output: y=234.768951

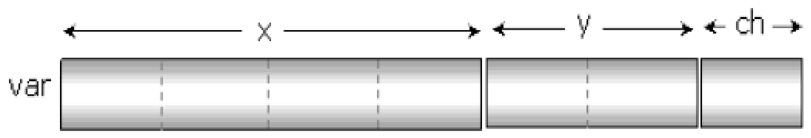
x=615 ch=s

6. Write the differences among the struct and union?

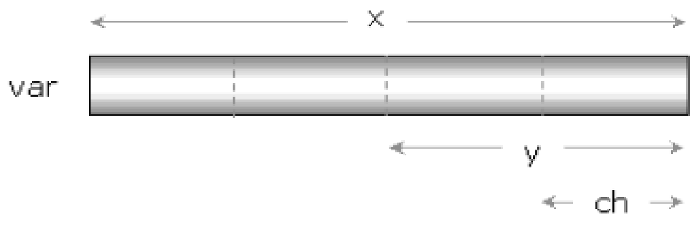
Thought the struct and union are user defined types and looks similar, functionality and usage of both differs.

	Structures	Unions
	It is a user defined type	It is a user defined type
	It is to store multiple values of different types in a common memory	It is used as a generic type
	Different fields are allocated for different members	Common memory is allocated for all the fields
	Different values can be stored in different fields	Only one value can be stored in any union variable
	The size of variable is the sum of sizes of all the fields	The size of variable is the size of field with maximum size
	6 It results no memory wastage	It results memory wastage as we use a part in the total memory allocation

```
struct demo
{
float x;
short y;
char ch;
};
struct demo var;
```



```
union demo
{
float x;
short y;
char ch;
};
union demo var;
```



7. What is Enum explain it? (or) what is enumerated datatype ?

- ☞ enum is a keyword. By using enum we can create sequence of integer content values.
- ☞ By using enum we can create userdefined datatype of integer.
- ☞ When we are working with enum it should be applied for integral data type only.
- ☞ The size of enum constants are 2B.

Syntax: enum tagname { const1=value,const2=value,…….};

Advantages:

1. It prevents the assignment of invalid value to variables.
2. This data type occupies only two bytes of memory.
3. The use of enumeration variables with in a program can increase the logical clarity of the program

Disadvantages:

1. The enumeration constants cannot be read from the key board by using "cout" statement.
2. When we are printing these variables, only the integer values associated with it will be printed on the screen.

Example

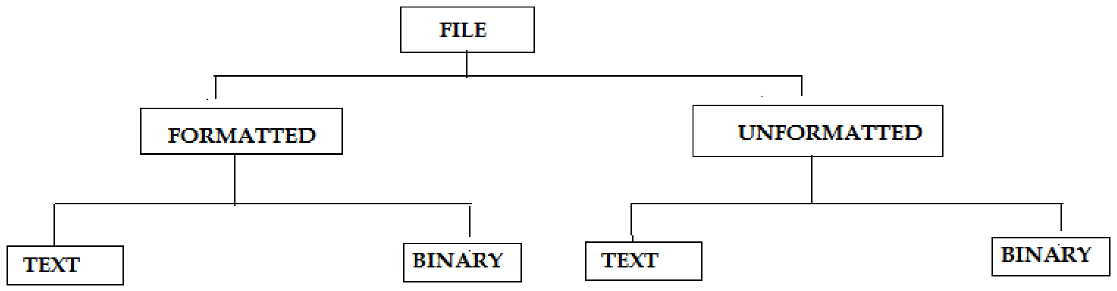
```
#include <stdio.h>
#include<conio.h>
enum ABC{X,Y,Z};
void main()
{
  int a;
  a=X+Y+Z; //a=0+1+2;
  printf("a value is  ",a);
  printf("x values is ",X);
  printf("Y value is  ",Y);
  printf("z value is  ",Z);
  getch();
}
```

UNIT V

FILES

Introduction

- ☞ File is a name of physical memory location in secondary storage area which contains n number of information. i.e. collection of information.
- ☞ In implementation when we need to retrieve the input data from secondary storage area and when we need to redirect the data to secondary storage area then go for file operations.
- ☞ Generally secondary IO operations are called file operations.
- ☞ When we are working with standard I/O devices then it is called standard I/O operations.
- ☞ In implementation when we are interact with secondary I/O operations then it is called file operations.
- ☞ All file operations related functions are defined in stdio.h i.e <stdio.h>
- ☞ The files are generally classified into various type based on their data format and type.



In order to read the data from the file or write the data to the file, first that the file is to be opened. A file internal has to base references such are BOF (beginning of file) and EOF (end of file). The beginning of file can be accessed with built in structure called FILE. It is conglomerate data type, which holds all the attributes of the file, such are permissions, file type, data and time of access and size in bytes.

C language is supported with rich set of library function, which can handle the text, binary, and executable file belongs to various systems.

<u>Definition:</u> A file is a collection of bytes that contains some information permanently. A file is created in secondary storage devices.

Use of Files: We can use files to store data permanently in the memory. A file is created in the secondary memory device such as hard disk. When a file is created, we can modify or delete data present in the file as and when needed.

FILE datatype: To read and write data of files in 'c' language, we must use FILE datatype. It is a structure datatype that is used to create file buffer area.

Syntax: **FILE *fp;**

Here, FILE is the name of datatype and 'fp' is file pointer which will contain all the information about the file.

1.What is file ? how to open and close a file in C language.

Definition: A file is a collection of bytes that contains some information permanently. A file is created in secondary storage devices.

Opening a File (fopen( ) function): A file should be opened before it is used in the program. The fopen( ) function is used to open a file. If the file open operation is success, this function returns a FILE pointer otherwise it returns NULL.

Syntax: fp = fopen(filename,filemode) ;

In the above syntax,

1. The 'fp' is file pointer that is declared earlier with FILE datatype.
2. The 'filename' is a sting that represents name of the file in the secondary memory.
3. The 'filemode' is a string that contains special characters. It represents the purpose for which the file is opened such as reading, writing and so on. The following are different file modes used in 'c' language:

Depending on the operations file modes are classified into six types.

1. **Write(W):-** when we are opening the file in 'W' mode then it opens the file in writing purpose.
   a. In write mode always new file will be created, if the file is already exist with the name then it will deleted and new file will be created.
   b. In 'W' mode file is exist or not always new file will be created.
2. **Read(R):-** open a file for reading purpose.
   a. In 'R' mode if the file is already exist then fopen the file for reading.
   b. If the file is not exist then it returns null.
3. **Append(A):-** open a file for writing purpose at end of the file.
   a. In 'A' mode if the file is already exist then it will open for append, if the file is already exist then open for append.
   b. In 'A'mode new file will be created when the file is not exist.
4. **W+(write and read):-** create a new file for update(writing and reading), if a file by that name is already exists, it will overwritten (new file created).
   a. In w+ mode always new file will be created, even though file is exist or not.
5. **R+(read+write):-** open an existing file for update (reading and writing) .
   a. In 'r+' mode if the file is already exist then it opens for updating, if the file is not exist then fopen function returns null.

6. **A+(w+r or r+w):-** open for append, open for update at the end of the file, or create new file if the file does not exist.In 'a+' mode if the file is not exist then only new file will be created.

Examples:

```
FILE *f1, *f2;
f1 = fopen("student.dat","w");
f2 = fopen("employee.txt","r");
```

Closing a File (fclose( ) function) : The opened file must be closed at the end of all input and output operations done on it. The fclose( ) function is used to close an opened file. Once a file is closed, we cannot perform any operation on that file until it is opened again.

Syntax: **fclose(fp);**

Here, 'fp' is file pointer that holds the opened file.

2. Write about functions used to perform read/write operations on files?

fputc():

It accepts two arguments that are the character that we want to write on to the file and a file pointer. It writes the character onto the file through the file pointer.

fputc('g',fp);

It writes 'g' onto the file through file pointer fp

fgetc():

This function accepts the file pointer as argument and returns ASCII value of last fetched (accessed) character from the file.

ch=fgetc(fp);

It reads character by character from the file through file pointer fp and assigns to ch

fputs()

It accepts two arguments that are the string that we want to write on to the file and a file pointer. It writes string or line of text onto the file through the file pointer.

fputs("india",fp);

It writes 'india' onto the file through file pointer fp

fgets():

It accepts the address of a character array, the length of text has to fetch from the file and the file pointer as arguments. It reads the specified length of text from the file and assigns to the character array whose address has sent as argument.

fgets(str,40,fp);

It reads 40 characters from the file and stores into the string str

fprintf():

It works similar to printf() function but writes on to the file rather console output. It takes three arguments that are the file pointer, format string and the list of variables. The list of values of variables would be printed on to the file as per the format string.

fprintf(p,"%d\t%s\t%f\t%d\t%f\n",pcode,pname,price,qty,tot);

It prints the details of a product onto the file "product" through the file pointer "p"

fscanf():

It is similar to the function scanf() but the difference is that, It fetches records row by row from the formatted text file and stores into the specified variables as per the format specifier.

It accepts the file pointer, format string and the list of addresses of variables as arguments, read the record from the file through the file pointer and stores into the specified variables in a sequence.

fscanf(p,"%d%s%f%d%f",&pcode,pname,&price,&qty,&tot);

It fetches a record from the file "product" through the file pointer "p" and stores into the specified variables.

3. Write programs to write and read the data onto the file?

WRITING DATA TO A FILE

i) fprintf() function: This function is used to store values into the file. It writes values based on formatting characters specified.

Syntax: **fprintf(fp, formatstring, valueslist) ;**

Here, 'fp' is file pointer. The 'formatstring' contains formatting characters. The 'valueslist' contains one or more values separated by commas.

Ex: fprintf(fp, "%d %s %f",rno, snm, avg);

ii) fputc() function: This function is used to store a single character into file.

Syntax: **fputc(ch, fp);**

Here, 'ch' is a character variable or constant and 'fp' is file pointer.

iii) fputs() function: This function is used to store a string into the file.

Syntax: **fputs(string, fp);**

Here, 'string' is a string constant or variable. 'fp' is file pointer.

Program to write text onto the file

```
#include<stdio.h>
int main()
{
FILE *p;
char ch;
p=fopen("igate","w"); /* opeing file */
while(1)
{
ch=getchar(); /* reading from Console Input */
if(ch==-1) /* checking end of file */
break;
fputc(ch,p); /* writing on to file */
}
fclose(p); /* closing the file */
printf("1 file is created..");
return 0;
}
```

READING DATA FROM A FILE

i) fscanf() function:

This function is used to read values from the file and stores them into variables. It reads values based on formatting characters specified.

Syntax: **fscanf(fp, formatstring, variablelist) ;**

Here, ‘fp’ is file pointer. The ‘format-string’ contains formatting characters. The ‘variable-list’ contains one or more variables.

Ex: fscanf(fp, “%d %s %f”,&rno, snm, &avg);

ii) fgetc() function:

This function is used to read a single character from the file and stores it in a variable.

Syntax: **variable = fgetc(fp);**

Here, ‘variable’ is a character type variable and ‘fp’ is file pointer.

iii) fgets() function:

This function is used to read the specified number of characters from the file and stores it in a variable.

Syntax: **fgets(str, n, fp);**

Here, this function reads ‘n’ characters from the file ‘fp’ and stores it in the variable ‘str’.

Ex: fgets(nm, 10, fp);

Program to read text from the file

```
#include<stdio.h>
int main()
{
FILE *p;
char ch;
p=fopen("igate","r"); /* opening the file in read mode */
while(1)
{
ch=fgetc(p); /* reading character from the file */
if(ch==-1) /* checking end of file */
break;
printf("%c",ch); /* printing character on to the console output */
}
fclose(p); /* closing the file */

return 0;
}
```

DETECTING THE END OF FILE

When reading or writing data from files, we do not know exactly how long the file is. In C, there are two ways to detect end of file.

i) EOF symbolic constant:

While reading the file in text mode, we can compare each character with EOF symbolic constant. It is defined in stdio.h with a value -1.

Example:

```
#include <stdio.h>
main()
{
        int c;
        FILE *fp;
        ----
        ----
        While (...)
        {
                c = fgetc(fp);
                if (c = = EOF)
                        break;
                else
                        ----
        }
}
```

ii) feof() function:

This function checks whether end of file has been reached or not. It returns non-zero when end of file is reached otherwise it returns zero.

Syntax: **feof(fp);** where 'fp' is file pointer

Example:

```
#include <stdio.h>
main()
{
        FILE *fp;
        ----
        ----
        While (!feof(fp))
        {
                ----
                ----
        }
}
```

4. Write C programs to write/Read student details onto the file

```
#include<stdio.h>
int main()
{
FILE *fp;
char sname[20];
int m1,m2,m3,tot,n,i;
float avg;
fp=fopen("data","w"); /*Creating file */
printf("How many records?");
scanf("%d",&n);
printf("Enter the records..\n");
for(i=1;i<=n;i++)
{
printf("Student Name:"); /* Reading data from into the program */
scanf("%s",sname);
printf("Enter the marks in 3 subjects:\n");
scanf("%d%d%d",&m1,&m2,&m3);
tot=m1+m2+m3;
avg=(float)tot/3;
fprintf(fp,"%s\t%d\t%d\t%d\t%d\t%f\n",sname,m1,m2,m3,tot,avg); /*Writing onto file */
}
fclose(fp); /* closing file */
return 0;
}
```

Reading student details from the file and writing onto monitor

```
#include<stdio.h>
int main()
{
FILE *fp;
char sname[20];
int m1,m2,m3,tot;
float avg;
fp=fopen("data","r"); /* opening the file in read mode */
printf("Students details are…\n");
while(1)
{
fscanf(fp,"%s%d%d%d%d%f",sname,&m1,&m2,&m3,&tot,&avg); /* reading from file */
if(feof(fp))
break;
printf("%s\t%d\t%d\t%d\t%d\t%f\n",sname,m1,m2,m3,tot,avg); /* writing on monitor */
}
fclose(fp); /* closing file */
return 0;
}
```

Discuss About Error Handling During File Operations

While reading data from or writing data to a file, it is quite common that an error occurs. The following are the reasons that an error may arise.

- ✓ When trying to read a file beyond EOF indicator
- ✓ When trying to read a file that does not exist
- ✓ When trying to use a file that has not been opened
- ✓ When trying to write data to a file that has been opened for reading

If we fail to check for errors, then the program may behave abnormally. So an unchecked error may stop execution of program or gives incorrect output.

ferror() : In C, **ferror()** library function is used to check for errors in the file. It checks for any errors in the program. It returns zero if no errors have occurred. It returns a non-zero value if there is any error.

The error indication will last until the file is closed or it is cleared by the **clearer()** function.

Syntax: **ferror(fp)** where fp is file pointer

Example:

```
#include <stdio.h>
main()
{
        FILE *fp;
        ----
        ----
        if (ferror(fp))
        {
                Printf("there is an error in the file");
        }
        -----
        -----
}
```

Clearerr() function:

This function is used to clear the EOF and error indicators for the file. It is used because error indicators are not automatically cleared.

Syntax: clearerr(fp) where fp is file pointer

Example:

```
#include <stdio.h>
main()
{
        FILE *fp;
        ----
        ----
        if (ferror(fp))
        {
                clearerr(fp);
        }
        -----
        -----
}
```

PROGRAMMING IN C LAB

1. Perfect number

AIM: To write a C program find out whether the given number is perfect or not.
ALGORITHM:

```
Step1: Start
Step2: declare variables n,i,sum←0
Step3: print 'enter a value'
Step4: Read n value.
Step5:  for i←1 to i<=n/2 in steps repeat step6
Step6: if  n%i==0
        6.1: sum←sum+i
        [end for loop]
Step7: if n==sum
        7.1: print 'Given number is perfect number'.
        Else
        7.2: print 'Given number is not perfect number'.
Step8: Stop
```

PROGRAM:

/* 1.PROGRAM TO CHECK WHETHER A GIVEN NUMBER IS PERFECT NUMBER OR NOT*/

```
#include<stdio.h>
#include<conio.h>
void main()
{
  int n,i,sum=0;
  clrscr();
  printf("enter a value:");
  scanf("%d",&n);
  for(i=1;i<=n/2;i++)
```

OUTPUT

```
Case1:  Enter a value: 8
          8 is not perfect number:

…………………………..

Case 2: Enter a value:6
```

```
  {
   if(n%i==0)
   sum=sum+i;
  }
  if(n==sum)
  printf("\n %d is perfect number",n);
  else
  printf("\n %d is not perfect number:",n);
  getch();
}
```

2. Armstrong number

AIM: To write a C program to check whether the given number is Armstrong or not.

ALGORITHM:

```
Step1: start
Step2: Declare the variables n,temp,r, sum←0
Step3: Write ‘enter a number’.
Step4: Read ‘n’ value.
Step5: temp←n
Step6: if(temp>0) then
        6.1: r←temp%10
        6.2: sum←sum+(r*r*r)
        6.3: tem←temp/10
        6.4: got step6
Step7: if(sum==n)
        7.1: Print :”Given number is Amstrong number”.
        Else
        7.2: Print “Given number is not Amstrong number”
Step8: Stop.
```

PROGRAM:

/* PROGRAM TO CHECK WHETHER A GIVEN NUMBER IS ARMSTRONG NUMBER OR NOT*/

```
#include<stdio.h>
#include<conio.h>
void main()
{
  int n,temp,sum=0,r;
  clrscr();
  printf("enter a value: ");
  scanf("%d",&n);
 temp=n;
 while(temp!=0)
  {
   r=temp%10;
   sum=sum+(r*r*r);
```

OUTPUT

```
 Enter a value: 153
 153 is Armstrong number
…………………………….
  Enter a value: 123
 123 is not Armstrong number
```

```
  temp=temp/10;
 }
 if(n==sum)
 printf("\n %d is amstrong number",n);
 else
 printf("\n %d is not amstrng number",n);
}
```

3. Sum of individual digits

AIM: To Write a C program to find the sum of individual digits of a positive integer.

ALGORITHM:

```
Step1: Start
Step2: declare variables n,sd←0
Step3: print 'enter a value'
Step4: Read 'n' value
Step5: while n>0 in steps repeat step6
Step6: sd←sd+n%10
        6.1: n←n/10
        [End while loop]
Step7: print ' sd' value for sum of digits.
Step8: Stop
```

PROGRAM:

```
/* PROGRAM FOR SUM OF DIGITS OF A NUMBER */
#include<stdio.h>
#include<conio.h>
void main()
{
   int n,sd=0;
   clrscr();
   printf("enter a value: ");
   scanf("%d",&n);
   while(n>0)
   {
     sd=sd+n%10;
     n=n/10;
   }
   printf("\n sum of digits  = %d",sd);
   getch();
}
```

OUTPUT:

```
Enter a value: 456

Sum of digits = 15
```

4. FIBONACCI SERIES

AIM: To write a C Program Print Fibonacci series.

ALGORITHM:

Step1: Start

```
Step2: Declare the variables i,j,k,n
Step3: Print 'enter a value'
Step4: read 'n' value
Step5: i←0
Step6: j←1
Step7: print i,j values
Step8: k←i+j
Step9: while k<=n then repeat step10
Step10: print k value
        10.1: i←j
        10.2:j←k

        10.3: k←i+j
        [End while loop]
Step11: stop
```

Program:

/* PROGRAM FOR GENERATING FIBONACCI SERIES*/

```
#include<stdio.h>
#include<conio.h>
void main()
{
  int i,j,k,n;
  clrscr();
  printf("enter a value: ");
  scanf("%d",&n);
  i=0;
  j=1;
  printf("\n %d %d ",i,j);
  k=i+j;
  while(k<=n)
  {
    printf("  %d",k);
    i=j;
    j=k;
    k=i+j;
  }
  getch();
}
```

OUTPUT

```
enter a value: 25
 0 1   1  2  3  5  8  13  21
```

5. Prime Numbers

AIM: To write a C program to generate all the prime numbers between 1 to given range.

ALGORITHM:

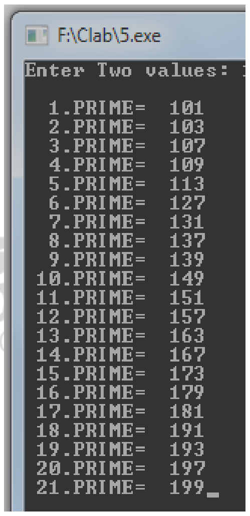
Step1: start

```
Step2: declare the variables n,n1,n2,t, count←0,flag
Step3: Print 'enter two values"
Step3: Read n1, n2 values
Step4: for n←n1 to n<=n2  in steps repeat step5
Step5: flag←0
        5.1: for t←2 to t<=n2 in steps repeat step5.2
        5.2: if n%t==0 then flag=1 and break
        5.3: End for loop
Step6: if flag==1n and n!= 1
Step7: printf n value for prime numbers.
Step8: End for loop
Step9: Stop
```

PROGRAM:

/* PROGRAM FOR TO PRINT PRIME NUMBERS BETWEEN RANGE */

```
#include<stdio.h>
#include<conio.h>
void main()
{
  long int n,n1,n2,t;
  int count=0,flag;
  clrscr();
  printf("Enter Two values: ");
  scanf("%ld%ld",&n1,&n2);
  for(n=n1; n<=n2; n++)
  {
    flag=0;
    for(t=2; t<n; t++) //t<=sqrt(n) <math.h>
    {
      if(n%t==0)
      {
        flag=1;
        break;
      }
    }
    if(flag==0&&n!=1)
      printf("\n%3d.PRIME=%5d",++count,n);
  }
  getch();
}
```



```
Enter Two values: 

  1.PRIME=  101
  2.PRIME=  103
  3.PRIME=  107
  4.PRIME=  109
  5.PRIME=  113
  6.PRIME=  127
  7.PRIME=  131
  8.PRIME=  137
  9.PRIME=  139
 10.PRIME=  149
 11.PRIME=  151
 12.PRIME=  157
 13.PRIME=  163
 14.PRIME=  167
 15.PRIME=  173
 16.PRIME=  179
 17.PRIME=  181
 18.PRIME=  191
 19.PRIME=  193
 20.PRIME=  197
 21.PRIME=  199
```

6. Largest and Smallest Number

Aim: To write a C program to find largest and smallest number in a list of integers.

Algorithm:

```
Step1: Start
Step2: declare the variables max,min, i
Step3: declare the array name 'arr'
Step4: print enter the require values
```

```
Step5: Read the array size using for loop
Step6: max=min=arr[0]
Step7: for i←1 to i<size repeat step8 to 9
Step8: if arr[i]>max
        8.1: max=arr[i]
Step 9: if arr[i]<min
        9.1: min=arr[i]
Step10: print 'max' for maximum value
Step11: print 'min' for minimum value in the list
Step12: Stop
```

PROGRAM

/*6 PROGRAM FOR TO FIND LARGEST/SMALLEST OF N NUMBERS BY USING AN ARRAY*/

```c
#include <stdio.h>
#include <conio.h>
#define size 10
int main()
{
  int arr[size];
  int max,min;
  int i;
  clrscr();
printf("\nEnter %d Values: ",size);
 for(i=0;i<size;i++)
  scanf("%d",&arr[i]);
  max=min=arr[0];
  for(i=1;i<size;i++)
  {
          if(arr[i]>max)
          {
            max=arr[i];
           }
           if(arr[i]<min)
           {
             min=arr[i];
           }
   }
   printf("\nMaximum number in the list =%d",max);
   printf("\nMinimum number in the list=%d",min);
  getch();
  return 0;
}
```

OUTPUT

```
Enter 10 Values: 10  20 30 40 60 80
160 5 69 41

Max=160
```

7. Addition and Subtraction of matrix

AIM: To Write a C program to find out the addition of given two matrices.

ALGORITHM:

```
Step1: start
Step2: Declare the variables i,j,m,n
Step3: take an array a[10][10],b[10],c[10][10]
Step4: Write ‘enter the row and column of the matrix’.
Step5: Red m,n values
Step6: Write ‘Enter the elements of matrix A’.
Step7: for i←0 to i<m
    7.1: for j←0 to j<n
        7.1.1: Read a[i][j]
        7.1.2:  j←i+1
        7.1.3: Repeat Step 7.1
    7.2:i←i+1
    7.3: Repeat step7
Step8: Write ‘Enter the elements of matrix B’.
Step9: for i←0 to i<m
    9.1: for j←0 to j<n
        9.1.1: Read a[i][j]
        9.1.2:  j←j+1
        9.1.3: Repeat Step 9.1
    9.2: i←i+1
    9.3: Repeat step 9
Step10: calculate the addition of matrix and store c.
    10.1: for i←0 to i<m
        10.1.1: for j←0 to j<n
        10.1.2: c[i][j]←a[i][j]+b[i][j]
        10.1.3: j←j+1
        10.1.4: Repeat step 10.1.1
    10.2: i←i+1
    10.3: Repeat step 10.1.
Step11: Write ‘addition of matrix A and B are’.
Step12: for i←0 to i<m
    12.1: for j←0 to j<n
        12.1.1: Read c[i][j]
        12.1.2:  j←j+1
        12.1.3: Repeat Step 12.1
    12.2: i←i+1
    12.3: Repeat step 12.
Step13: Write ‘subtraction of matrix A and B are’.
```

```
Step114: for i←0 to i<m
        14.1: for j←0 to j<n
                14.1.1: Read c[i][j]
                14.1.2:  j←j+1
                14.1.3: Repeat Step 12.1
        14.2: i←i+1
        14.3: Repeat step 12.
Step13: Stop.
```

/*7PROGRAM FOR MATRIX ADDITION AND SUBTRACTION*/

```
#include<stdio.h>
#include<conio.h>
void main()
{
  int i,j,r,c,a[10][10],b[10][10];
  clrscr();
  printf("enter rows of a Matrix   : ");
  scanf("%d",&r);
  printf("enter columns of a Matrix: ");
  scanf("%d",&c);
  printf("\n****************************************\n");
  printf("Enter the elements of Matrix A \n");
  for(i=0;i<r;i++)
  {
    for(j=0;j<c;j++)
    scanf("  %d",&a[i][j]);
  }
  printf("\n Enter elements of Matrix B \n");
  for(i=0;i<r;i++)
  {
   for(j=0;j<c;j++)
   scanf("  %d",&b[i][j]);
  }
  printf("\n**************************\n");
  printf("matrix addition: \n");
  for(i=0;i<r;i++)
  {
    for(j=0;j<c;j++)
    printf(" %5d",a[i][j]+b[i][j]);
    printf(" \n");
  }
  printf("\n******************************");
  printf("\n Matrix subtraction: \n");
  for(i=0;i<r;i++)
  {
```

7. OUTPUT

```
enter rows of a Matrix   : 2
enter columns of a Matrix: 2
**************************
Enter the elements of Matrix A
7       8       9       7
 Enter elements of Matrix B

4       8       3       1
*************************
matrix addition:
    11    16
    12     8
********************
 Matrix subtraction:
    3    0

    6    6
    for(j=0;j<c;j++)
    printf("%5d",a[i][j]-b[i][j]);
    printf("\n");
  }
  getch();
}
```

8. String operations

AIM: To write a C program to perform various string operations:

Algorithm for concatenation of two strings:

```
Step1: Start
Step2: Declare character array s1, s2
Step3: Print 'Enter first string"
Step4: Read s1
Step5: Print 'enter second string'
Step6: Read s2
Step7: use 'strcat(s1,s2)' function
Step8: Print 's1'
Step9: Stop
```

/* CONCATENATION OF TWO STRINGS*/

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
int main()
{
  char s1[30];
  char s2[30];
  clrscr();
  printf("enter  first strinng: ");
  gets(s1);
  printf("Enter second string:  ");
  gets(s2);
  strcat(s1,s2);
  puts(s1);
  getch();
  return 0;
}
```

OUTPUT

```
enter  first strinng: uma

Enter second string:  maheswari

umamaheswari
```

Algorithm for comparison of two strings:

```
Step1: Start
Step2: Declare character array s1, s2
Step3: Declare the integer variable d
```

Step4: Print 'Enter first string"

Step5: Read s1

Step6: Print 'enter second string'

Step7: Read s2

Step8: d←strcmp(s1,s2)

Step9: Print 'd' value for ASCII value difference.

Step10: stop

/* COMPARISON OF TWO STRINGS*/

```
#include <stdio.h>
#include <conio.h>
#include <string.h>
int main()
{
  char s1[30];
  char s2[30];
  int d;
  clrscr();
  printf("Enter First string:  ");
  gets(s1);
  printf("Enter second string: ");
  gets(s2);
  d=strcmp(s1,s2);
  printf("\nASCII Value diff=%d",d);
  getch();
  return 0;
}
```

OUTPUT:

```
Enter First string:  welcome

Enter second string: hello

ASCII Value diff=15
```

Algorithm for Length of a string:

Step1: Start

Step2: Declare character array str

Step3: Declare the integer variable l

Step4: Print 'Enter first string"

Step5: Read str

Step6: l←strlen(str)

Step7: print 'l' value for length of the string

Step8: Stop

/*LENGTH OF A STRING*/

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

```
#include<string.h>
int main()
{
  char str[30];
  int l;
  clrscr();
  printf("enter a string: ");
  gets(str);
  puts(str);
  l=strlen(str);
  printf("\nLenght of str:%d",l);
  getch();
  return 0;
}
```

OUTPUT

```
enter a string: sree chaitanya  degree
college

Sree Chaitanya Degree College

Lenght of str:17
```

9. Searching

Algorithm: To write a C program that implements searching of a given item in a given list.

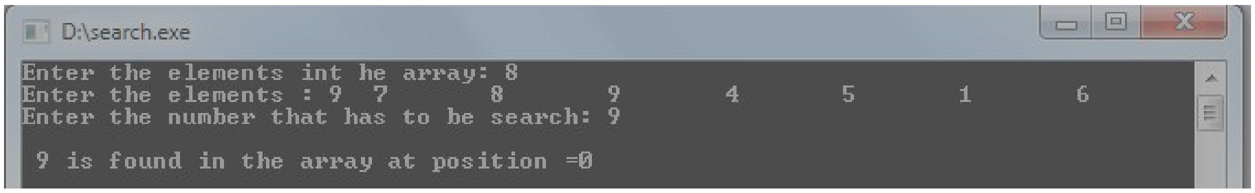
```
Step1: start
Step2: take an array
Step3: declare the variables num, I, n, found=0, pos=-1
Step4: print 'how many elements you want'.
Step5: read n value
Step6: print 'enter the elements'
Step7: read the elements in the array
Step8: print 'enter the number that has to be search'
Step9: read the number
Step10: for i←0 to i<n repeat step 11
Step11: if arr[i]==num
        11.1: found =1
        11.2: POS=I
Step12: print 'given number is found in the array'
        [End for loop]
Step13: if found==0
        13.1: print' given number not in the list'
Step14: Stop
```

PROGRAM

/* PROGRAM TO IMPLEMENT SEARCHING OF GIVEN ITEM IN A GIVEN LIST*/

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

```
#include<conio.h>
int main()
{
 int arr[10], num,i,n,found=0,pos=-1;
 clrscr();
 printf("Enter the elements int he array: ");
 scanf("%d",&n);
 printf("Enter the elements : ");
 for(i=0;i<n;i++)
 scanf("%d",&arr[i]);
 printf("Enter the number that has to be search: ");
 scanf("%d",&num);
 for(i=0;i<n;i++)
 {
   if(arr[i]==num)
   {
     found=1;
     pos=i;
     printf(" \n %d is found in the array at position =%d",num,i);
     break;
   }
 }
 if(found==0)
 printf("\n %d does not exist in the array ",num);
 getch();
 return 0;
 }
```


D:\search.exe

```
Enter the elements int he array: 8
Enter the elements : 9  7       8       9       4       5       1       6
Enter the number that has to be search: 9

 9 is found in the array at position =0
```

10. Sorting numbers

Aim: To Write a C++ program to Sort the given set of number in ascending order.

Algorithm:

Step1: Start

Step2: Declare the variables i,j,n

Step3: take an array a[10]

```
Step4: Print 'enter array size'
Step5: read n value.
Step6: Print 'enter elements into array'.
Step7: for i=0 to n-1 in steps of repeat step8
Step8: Read a[i]
        [End for loop]
Step9: for j=0 to j<n-i-1
Step10: if(a[j]>a[j+1]) then
        10.1: int t←a[j]
        10.2: a[j]←a[j+1]
        10.3: a[j+1]←t
        10.4: j←j+1
        [End for loop]
        10.5: Repeat step10
Step11: Write after sorting
Step12: for i←0 to n-1 insteps of 1 repeat step13
Step13: print a[i]
        [End for loop]
Step14: Stop.
```

/*10.PROGRAM FOR SORTING AN ARRAY*/

```
#include <conio.h>
#define size 10
int main()
{
  int arr[size];
  int i,j,t;
  clrscr();
  printf("\nEnter 10 Values:");
  for(i=0;i<size;i++)
  scanf("%d",&arr[i]);
  for(i=0;i<size;i++)
  {
     for(j=i+1;j<size;j++)
      {
            if(arr[j]<arr[i])
             {
               t=arr[i];
               arr[i]=arr[j];
               arr[j]=t;
             }
      }
   }
   printf("\nAfter sorting....");
   for(i=0;i<size;i++)
    printf("%d ",arr[i]);
```

OUTPUT

```
Enter 10 Values:85  63  74  96 45 65  36
12  1  6

After sorting....1 6 12 36 45 63 65 74 85
```

```
   getch();
   return 0;
}
```

All the best

G. Mahesh MCA

Lecturer in Computers