UNIT-I

BASICS OF C PROGRAMMING

Introduction to Programming Paradigms – Applications of C Languages – Structure of C program – C programming: Data types – Constants – Enumeration constants – Keywords – Operators: Precedence and Associativity – Expressions – Input/output statements, Assignment statements – Decision Making Statements – Switch Statements – Looping statements – Preprocessor directives – Compilation Process.

Introduction to Programming Paradigms:-

Programming paradigm refers to how a program is written in order to solve a problem. It is a way to think about programs and programming.

Programming can be classified into three categories

* Unstructured programming
* Structured programming
* Object Oriented programming

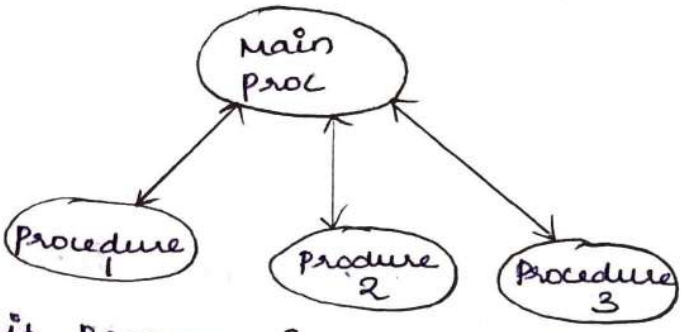
Unstructured Programming:-

Unstructured programming refers to writing small and simple programs consisting of only one main program. This style of programming is generally restricted for developing small applications, but if the application becomes large then it poses real difficulties in terms of clarity of the code, modifiability, and ease of use.

Structured Programming:-

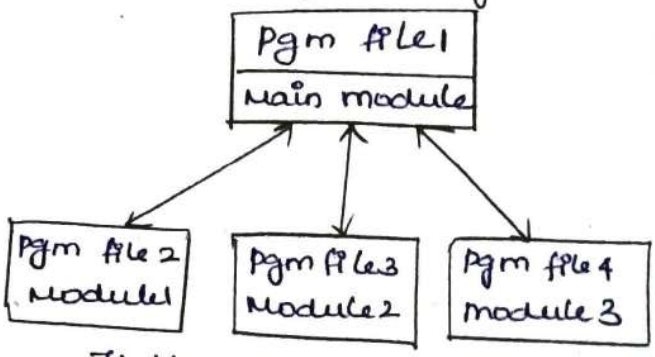
Structured programming, a program is broken down into small independent task, that are small enough to be understood easily, without having to understand the whole program at once. It can be performed into two ways

* Procedural Programming



This programming has a single program that is divided into small piece called procedure.

* Modular Programming



If the program gets larger, then modular way of programming is recommended. Each module is designed to perform a specific function.

Object Oriented Programming :-

It is a type of programming in which programmers define not only the data type of a data structure, but also the types of operations that can be applied to the data structure.

One of the Principle advantages of Object Oriented Programming techniques over procedural programming techniques is that they enable programmers to create Modules that do not need to be changed when a new type of object is added.

The basic concepts of OOP are as follows:

* Objects
* classes
* Data Abstraction & Encapsulation
* Inheritance
* Polymorphism
* Dynamic Binding
* Message passing

Applications of C Programming

* Operating Systems
* Embedded systems
* GUI
* New Programming platforms
* Google
* Mozilla firefox and Thunderbird
* Compiler Design
* Gaming and Animation.

Operating Systems:-

* The first Operating system to be developed using a high level programming language was UNIX, which was designed in the C programming language.
* Microsoft windows and various Android applications were scripted in C.

Embedded Systems:-

* The C programming language is considered an optimum choice when it comes to scripting applications and drivers of embedded systems, as it is closely related to machine hardware.

GUI

*GUI stands for Graphical User Interface. Adobe Photoshop, one of the most popularly used photo editors was created with the help of C.

New Programming platforms:

*A Programming language including all the features of C in addition to the concept of object-oriented programming but, various other programming languages like matlab and mathematica was written in 'C'.

* It facilitates the faster computation of programs.

Google :-

* Google file system and Google chromium browser were developed by using c/c++.

* The Google open source community has a large number of projects being handled using c/c++.

Mozilla firefox and Thunderbird :-

* Mozilla firefox and Thunderbird were open source email client projects, they were written in c/c++.

Compiler design :-

* One of the most popular uses of the c language was the creation of compilers.

* Compilers for several other programming languages were designed in association of c with low level languages making it easier to be comprehensible by the machine.

Example :-

Clang C, MINGW and Apple C

Gaming and Animation :-

C programming language is relatively faster than Java or python, as it is compiler based, it finds several applications in the gaming sector.

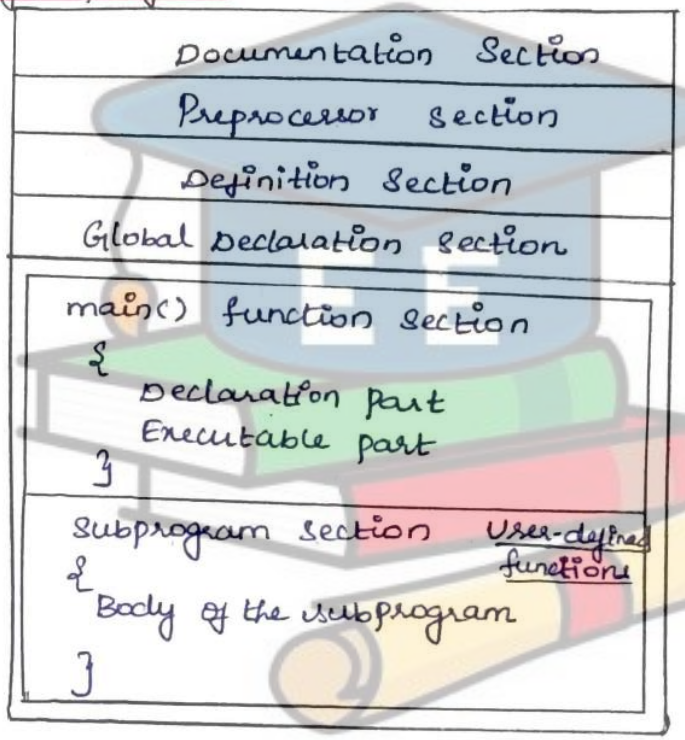
Example :-

Tic-Tac-Toe
The Dino game
Snake game.

Introduction to C :-

* C is a general purpose highlevel language developed by Dennis Ritchie at AT+T Bell laboratories in 1972.
* C is a general purpose, structured programming language.
* C is a middle level language.
* C language allows dynamic memory allocation.
* It is highly portable.

Structure of c program



Documentation Section:-

The documentation section consists of a set of comment lines giving the name of the program and other details. comment line is a non-executable statement. The comments are placed between /* and */

Preprocessor Section :-

Preprocessor Section which direct the compiler to link functions from the system library.

(e.g) #include <stdio.h>
#define PI 3.14

Global declaration section:-

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

Function main:-

Every program written in C language must contain main() function. This section contains two parts,

* Declaration part
* Executable part

Declaration part:-

It declares all the variables used in the executable part.

Executable part:-

This part contains the statements following the declaration of the variable. Here the logic of the program can be implemented. They provide instructions to the computer to perform a specific task.

These two parts must appear between the opening and the closing braces.

Subprogram section:-

It contains all the user defined functions that are called in the main function. User defined functions are generally placed immediately after the main function, although they may appear in any order.

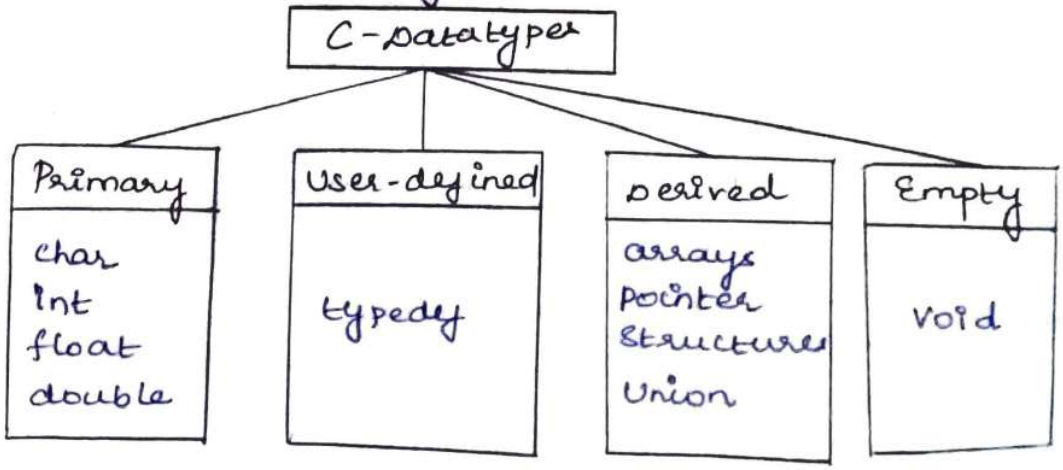
(e.g)

```
/* Sum of two numbers */
#include <stdio.h>
#define A 5
int c;
int add (int);
main()
{
  int b;
  Printf ("Enter the value of b");
  scanf ("%d", &b);
  c= add (b);
  Printf ("sum = %d", c);
}
int add (int b)
{
  c = A+b;
  return (c);
}
```

→ Documentation Section
→ Preprocessor section
→ Definition Section
→ Global variable declaration section
→ main() function
→ Declaration part
→ Executable part
→ Subprogram.

DATATYPES

Datatype is the type of the data, that are going to access within the program. 'C' supports different data types, each datatype may have predefined memory requirement and storage representation.



Primary Datatypes:-

Primary datatypes or fundamental datatypes.

* int * float * double * char

Integer (int):-

* Integers are the numbers with the supported range. Integers occupy one word of storage, and since the word sizes of machines vary from 16 or 32 bits.

* If we use 16 bit word length, the size can be limited upto -32768 to +32767

(e.g) int a;

Float (float):-

Floating point numbers are stored in 32 bits, with a 6 digits of precision. It is used to store numeric values with decimal point.

(e.g) float x;

Double (double):-

Double datatype number uses 64 bits giving a precision of 14 digits, These are known as double precision numbers.

(e.g) double y;

Character (char):-

A single character can be defined as Char datatype. Characters are usually stored in 8 bits

(e.g) char a;

Type Qualifiers:-

If we use a 16 bit word length, size of data type is limited to the specific range. Inorder to provide some control over the range of numbers and storage space of the datatype we use type qualifiers. They are

* Short
* Long
* Signed
* Unsigned.

The declaration of long and unsigned integer permits us to increase the range of values.

User defined Datatypes :-

User defined datatype is used to create new data types, 'type declaration' allows users to define an identifier that would represent an existing datatypes.

Syntax:-

```
typedef type identifier
```

where type refers to an existing datatype.

identifier refers to the newname given to the datatype

(e.g).; typedef int mark;
mark a, b;

Another user-defined datatype is enumerated datatype

Syntax:-

```
enum identifier {val1, val2, val3 ... valn};
```

* The identifier is a user defined enumerated datatype which can be used to declare variables that can have one of the values enclosed within the braces, known as enumeration constants.

* The compiler automatically assigns integer digits beginning with 0 to all the enumeration constants.

(e.g).;
enum day {Monday, Tue, wed}
enum color {blue = 5, black, green, yellow}

Derived Datatype :-

Datatypes that are derived from fundamental datatypes are called derived datatypes. Derived datatype add some functionality to the basic datatypes

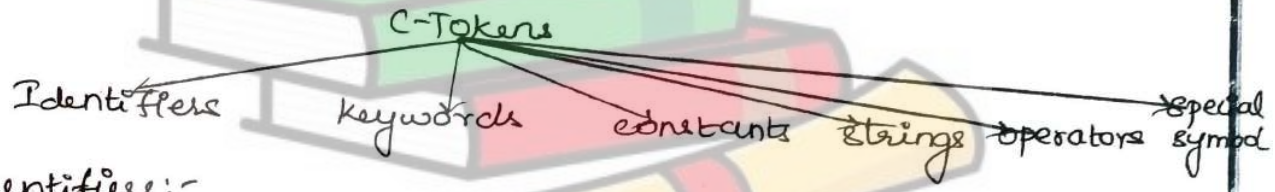
* Array is a collection of variables of same type.
* A pointer is a special variable that holds a memory address of another variable.
* A Structure is to multiple values of same or different datatypes under a single name. The entire structure variable is stored in sequence.
* A Union is also same as structure, but in memory union variables are stored in a common memory location.

Void :-

Void datatype is used to represent an empty or Null value. It is used as a return type if a function does not return any value.

C-Tokens

The smallest individual units in a program is called C-Tokens.



Identifiers :-

Identifiers are names given to various program elements, such as variables, functions and array etc.,

Rules :-

* Identifier consist of letter and digits in any order
* The first character must begin with char or _ (underscore)
* Both uppercase and lowercase are permitted.
* No space or special symbols are allowed between the identifier
* The identifier cannot be keyword.

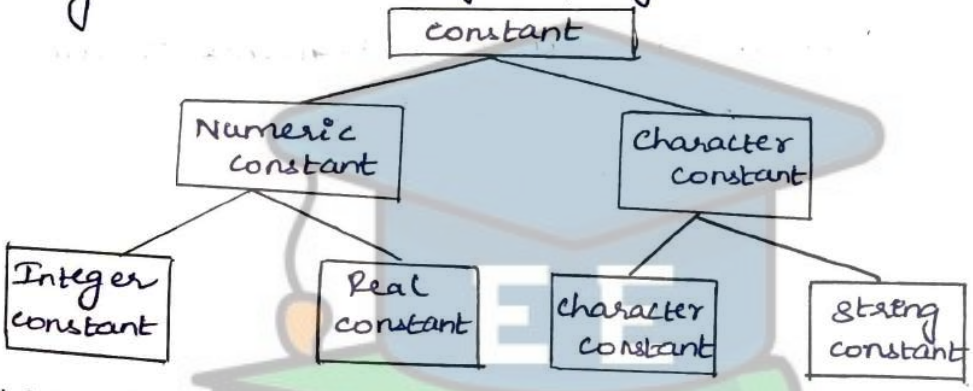
Keywords:-

There are certain reserved words called keywords, that have standard and predefined meaning in 'C' language.

(e.g.)., auto if
break switch
case extern
char while etc....

Constants:-

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



Integer constant:-

* An integer constant formed with the sequence of digits. There are 3 types of integer constant namely decimal integer, octal integer and hexadecimal integer.

* Decimal integers consist of a set of digits, 0 through 9

(e.g.)., 123, -438

* Octal integer consist of any combination of digits from the set 0 through 7 with a leading 0

(e.g.)., 0265 , 054

* Hexadecimal integers consist of a set of digits 0-9 and also include alphabets A through F with a leading 0x

Rules for defining an Integer constant.

* It must have atleast one digit.
* Decimal point is not allowed.
* It can be either positive or negative.
* No special characters or blankspaces are allowed.

Real constant

A real constant is madeup of a sequence of numeric digits with presence of a decimal point.

Fractional form

A real constant must have one digit.
It can be either positive or negative
No commas or blank spaces are allowed.
(e.g)., 175.283, -1.78 , .2683

Exponential form:-

A real number may also be expressed in exponential (or) scientific notation.

mantissa e exponent

Mantissa is either a real number expressed in decimal notation or an integer. The exponent is an integer number with an optional + (or) - sign.

(e.g)., 7.5 e3 (or) 75e2 , 12e-2

Character Constant

A single character constant contains a single character enclosed within a pair of single quotes.''

(e.g)., 'a' , 'z' , '1'

String Constant

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

(e.g). "Program" , "welcome" , "Hello@".

Backslash character constants:

C supports some special backslash character constants that are used in output functions. An escape sequence consists of a backward slash (\) followed by a character and both enclosed with in single quotes.

(e.g);

'\n' - newline
'\t' - tab (horizontal)
'\v' - verticaltab
'\0' - Null
'\\' - backslash.....

Variable.

A variable is an identifier that is used to represent some specified type of information within a designated portion of the program.

Variable declaration

datatype a, b, c;

(e.g)

int a;
float b;
char c;

Initializing variable

variable = constant.

(or)

datatype varname = const

(e.g)

a = 10
int x = 100
char c = '0'

Rules for Declaring variable

* A variable name can be any combination of 1-8 alphabets, digits or underscore.
* The first character must be an alphabet or an underscore.
* No commas or blankspaces are allowed
* No special symbols are allowed.

Operators

An Operator is a symbol that specifies an operation to be performed on the operands. The data items that operators acts upon are called operands.

Binary Operator:-

Operators require two operands called Binary Operator

Unary Operator:-

Operator acts upon only one operand called unary operator.

(e.g) a+b + → operator a,b → operands.

Ternary Operator:-

Operator operates on three operands (? :)

Types of Operator:-

* Arithmetic Operators
* Relational operators
* Logical Operators
* Assignment Operators
* Increment and decrement Operators.
* Conditional operators.
* Bitwise operators.
* Special Operators.

Arithmetic Operators:-

C allows us to carryout basic arithmetic operations like addition, subtraction, multiplication and division.

operator	Meaning	Example
+	Addition	5+3 = 8
-	Subtraction	9-3 = 6
*	Multiplication	3*3 = 9
/	Division	6/2 = 3
%	Modulo Division	5%2 = 1

Example:-

```
#include<stdio.h>
void main()
{
  int a=5, b=2;
  Printf("sum = %d", a+b);
  Printf ("Dyf = %d", a-b);
  Printf ("mul = %d", a*b);
  Printf ("Div = %d", a/b);
  Print f ("moddiv = %d", a%b);
}
```

Output:

```
sum= 7
Dyf = 3
Mul = 10
Div = 2
Moddiv = 1
```

Relational operator:-

Relational operators are used to compare two or more operands. Operands may be variables, constants or expressions. The result of relational expression is either one or zero.

Operator	Meaning	Example	Retival
<	is less than	2<9	1
<=	is less than or equal to	5<=20	1
>	is greater than	10>5	1
>=	is greatter than or equal to	10>=20	0
==	is equal to	5==15	0
!=	is not equal to	6!=5	1

Example:-

```
#include<stdio.h>
void main()
{ int a=5, b=10;
  Printf ("a>b : %d", a>b);
  Printf ("a!=b : %d", a!=b);
  Printf ("a<=b : %d", a<=b);
}
```

output:-

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

Logical operators:-

Logical operators are used to combine the results of two or more conditions.

Operator	Meaning	Example	Retival
&&	Logical AND	(9>2) && (5<2)	0
||	Logical OR	(5<2) || (5==4)	0
!	Logical NOT	!(5<2)	1

Example:-

```
#include<stdio.h>
void main()
{ int a=9, b=2, c=5;
  Printf ("%d", (a>2) && (c>b));
  Printf ("%d", (a>2 || (b>c));
  Printf ("%d", !(a>2));
}
```

output:-

1 1 0

Assignment Operator :-

Assignment Operators are used to assign a value or an expression or a value of a variable to another variable

Syntax:-

```
Var = exp
```

(i) compound assignment :-

Apart from assignment operator, C provides compound assignment operators to assign a value to a variable in order to assign a newvalue to a variable after performing a specified operation.

operation	Example	Meaning
+=	x+=y	x=x+y
-=	x-=y	x=x-y
*=	x*=y	x=x*y
/=	x/=y	x=x/y
%=	x%=y	x=x%y

Example:-

```
#include<stdio.h>
void main()
{
  int a=50, b=5;
  b+=a
  Printf("%d", b);
}
```

output:-

55

(ii) Nested or Multiple Assignment :

We can assign a single value or an expression to multiple variables.

Syntax:-

```
var1 = var2 = var3 = exp (or) var
```

(e.g)

```
a=b=c=10;
```

Increment and Decrement Operators (unary operators)

The Increment operator ++ adds one to the variables and -- operator (decrement) subtracts one from the variables.

operator	Meaning
++a	Pre-Increment
a++	Post-Increment
--a	Pre-Decrement
a--	Post-Decrement

Example:-

```
#include<stdio.h>
Void main()
{
Int a=5;
Printf("a++ =%d", a++);
Printf("++a=%d", ++a);
Printf("a-- =%d", a--);
Printf("--a =%d", --a);
```

output:-

a++ = 5
++a = 7
a-- = 7
--a = 5

Conditional Operator (or) Ternary operator:-

Conditional operator itself checks the condition and executes the statement depending on the condition.

Syntax:-

```
condition ? expr1 : expr2;
```

The "? :" operator acts as a ternary operator, it first evaluates the condition, if it is true then expr1 is evaluated, if the condition is false then expr2 is evaluated.

Example:-

```
#include<stdio.h>
void main()
{
int a=5, b=3, big;
big = a>b ? a : b;
Printf("Big :%d", big);
}
```

output:-

Big :5

Bitwise Operators:

Bitwise operators are used to manipulate the data at bit level. It operates on integers only.

operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
<<	Shift left
>>	Shift right
~	one's complement

a	b	a|b	a&b	a^b	~a
0	0	0	0	0	1
0	1	1	0	1	1
1	0	1	0	1	0
1	1	1	1	0	0

Example:-

```
#include<stdio.h>
Void main()
{
int x=7, y=8;
Printf("x&y :%d", x&y);
Printf("x|y :%d", x|y);
Printf("x^y :%d", x^y);
Printf("x<<2 :%d", x<<2);
}
```

output:-

x&y : 0 x^y : 5
x|y : 15 x<<2 : 14

Special operator :-

C language supports some special operator.

Operator	Meaning
,	comma operator
sizeof	Size of operators
& and *	Pointer operators
. and -->	member selection operator

member selection operator

•, → used to access elements from a structure

Comma operator:

The comma operator is used to separate the statement elements such as variables, constants or exp etc.,

Size of() operator :-

The sizeof() is a unary operator, that returns the length in bytes of the specified variable.

Pointer operator :-

& : This symbol specifies the address of the variable.

* : This symbol specifies the value of the variable.

Operator Precedence and Associativity of operators.

Precedence is used to determine the order in which different operators in a complex expression are evaluated.

Associativity is used to determine the order in which operators with same precedence are evaluated in a complex expressions.

Arithmetic Operator precedence:

The arithmetic operators are evaluated from left to right using the precedence of operators, when the exp is written without the parameters.

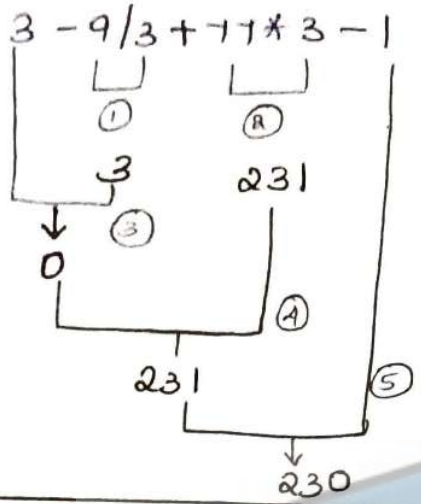
The arithmetic operator precedence

Precedence	Operators
High	* / %
Low	+ -

Example:-

$x - y/3 + z*3 - 1$

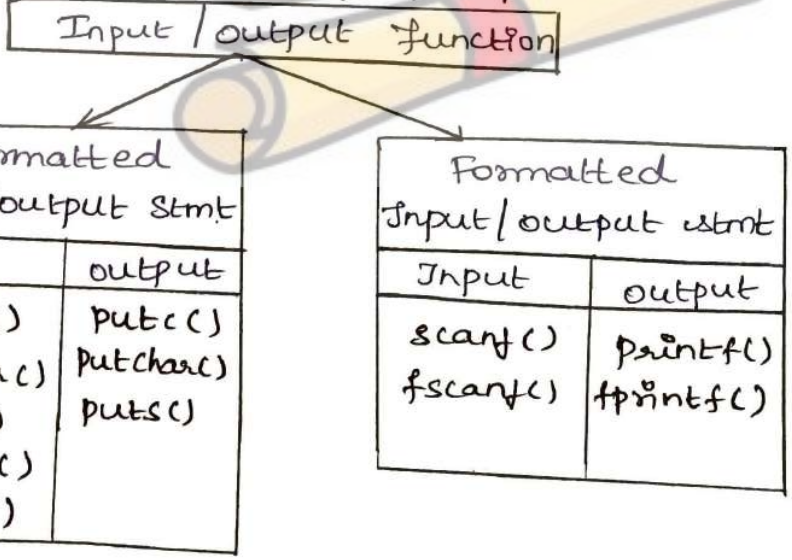
$x = 3$
$y = 9$
$z = 77$



Input/output Statements

The input/output functions permit the transfer of information between the computer and the standard input/output devices. In c input/output functions are classified into two types:

1. Formatted Input/output statements
2. Unformatted Input/output statements.



Unformatted Input/output statement:

The simplest of all input/output operations are reading a character from the standard input unit and writing it to the standard output unit.

getchar()

Reading a single character can be done by using the function getchar. It reads a single character from a standard input device (keyboard) till the user press the enter key.

Syntax:-

```
Varname = getchar()
```

getche()

The getche() function reads a single character data from the standard input unit.

Syntax:-

```
Varname = getche()
```

When this statement is executed the entered character is displayed to the computer screen, without waiting for the enter key to be hit.

getch()

The getch function reads a single character data from the standard input unit.

putchar()

Single character can be displayed using the library function putchar.

Syntax:-

```
putchar (varname)
```

Input/output of string data.

gets()

The gets function accepts the name of the string as a parameter, till a newline character is encountered.

Syntax:-

```
gets(str);
```

str → is a string variable.

Puts:-

The puts function is used to display the string to the standard output device.

Syntax:-

```
puts(str);
```

Example:-

```
#include<stdio.h>
#include<conio.h>
void main()
{
 char str[30];
 puts("Enter the name");
 gets(str);
 puts("Programming in C");
 puts(str);
 getch();
}
```

output:

```
Enter the name
Aruna
Programming in C
Aruna.
```

Formatted input & output function

Scanf()

It is used to read formatted data from the keyboard. The scanf function takes a text stream from the keyboard. extracts and formats data from the stream according to a format control string and then stores the data in specified program variables.

Syntax:-

```
scanf("controlstring", &arg1, &arg2 .... argn);
```

where control string constains the required formatting specification information. This begin with a percent sign(%) followed by a conversion character.

arg1, arg2 .... argn are arguments that represent the address of individual input data items.

%c → Single character
%d → decimal integer
%e, %f, %g → floating point value
%h → Short integer
%o → Octal integer
%s → String.

Printf()

It is used to display information required by the user and also prints the values of the variables.

Syntax:-

```
Printf ("controlstring", arg1, arg2, ..... argn);
```

Example:-

```
#include<stdio.h>
#include<conio.h>
void main()
{ int rollno;
  float avg;
  char name[30];
  Printf ("Enter name, rollno, avg");
  Scanf ("%s%d%f", &name, &rollno, &avg);
  Printf ("Name = %s\n Rollno = %d\n Avg = %f\n", name, rollno, avg);
  getch();
}
```

Output:-

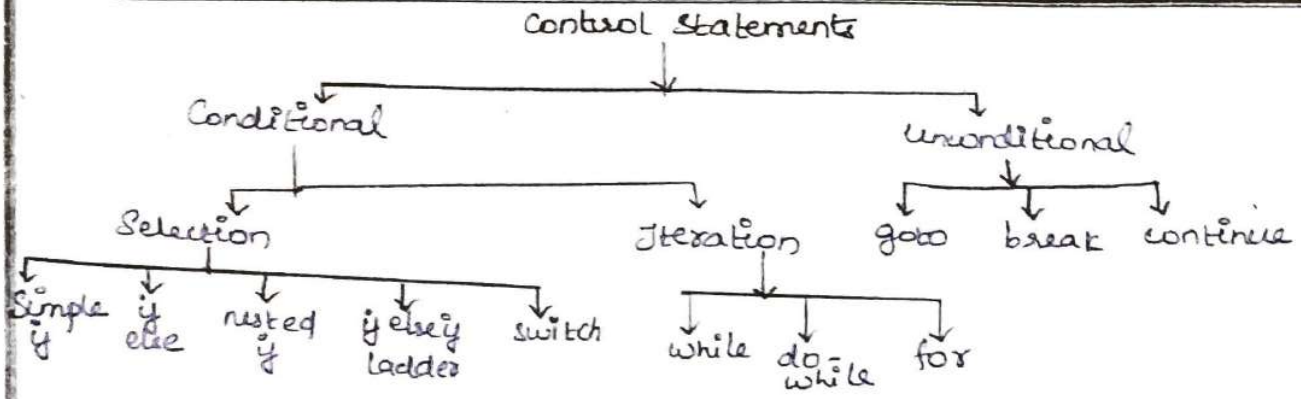
```
Enter name, rollno, Avg
Name = kumar
Rollno = 123
Avg = 85.18
```

Control statements

C has somekind of statements that permit the execution of a single statement or a block of statements. In some conditional executions one group of statement is selected from several available groups. This is known as selection. In some cases, a group of instructions be executed repeatedly, until some logical condition has been satisfied. This is known as looping.

All of these operations can be carried out using the various control statements.

* Conditional statements
* Unconditional statements.



Decision Making Statement / Selection statement

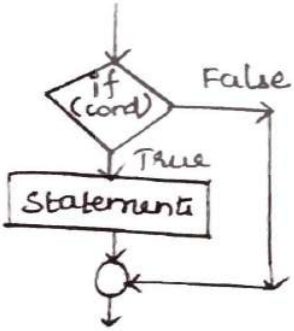
In selection or conditional branching, the program control is transferred from one point to another based on the outcome of a certain condition.

Simple-if Statement:

* The if statement is the simplest form of selection statement, that is frequently used in decision making.

Syntax:-

```
if (condition)
{
  statement
}
```



* The if condition is evaluated first, if it is true then the statements following if condition is executed, otherwise it skipped the condition and continues the next statement.

Example:-

```
#include <stdio.h>
void main()
{
  int n;
  Printf ("Enter the number");
  Scanf ("%d", &n);
  if (n>0)
    Printf ("The number is positive number");
  }
```

output:-

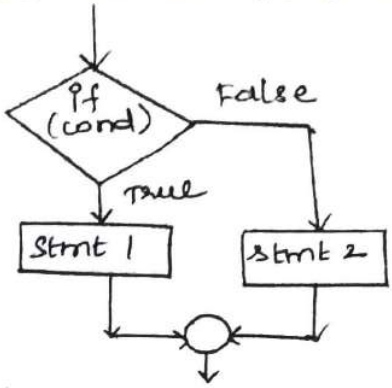
Enter the number
100
The number is positive number.

if-else Statement:-

if else statement evaluates the condition, if it is true the true block will be executed, otherwise the false block will be executed.

Syntax:-

```
if (cond)
    stmt1;
else
    stmt 2;
```




```
#include<stdio.h>
#include<conio.h>
void main()
{ int n;
  Printf ("Enter the number");
  Scanf ("%d", &n);
  If(n>0)
     Printf ("The number is positive");
  else
     Printf ("The number is Negative");
}
```

Output:-

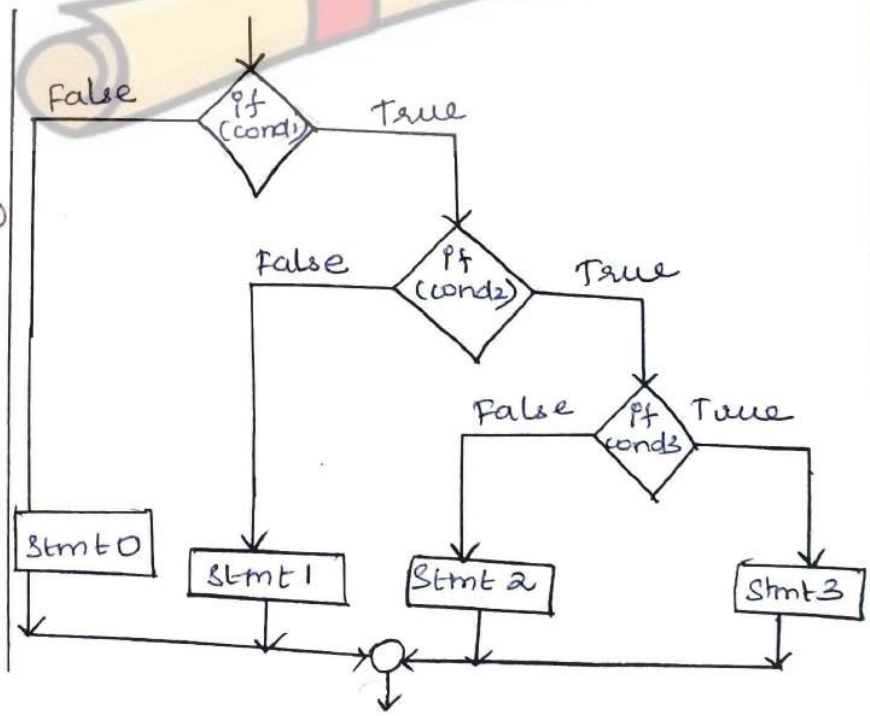
Enter the number
-5
The number is Negative

Nested-if:-

If one or more if statements are embedded within the if statement is called Nested if statement.

Syntax:-

```
if (condition1)
{ if (condition2)
   { if (condition3)
      { stmt 3;
      }
     else
      } stm 2;
  else
   } stmt 1;
else
  stmt 0;
```



Example:-

```
#include<stdio.h>
Void main()
{
 int a,b,c;
 Printf("Enter threenumbers");
 Scanf("%d %d %d",&a,&b,&c);
 if(a>b)
  {
    if(a>c)
      {printf("A is big");
      }
    else
      Printf("c is big");
  }
 else
   {
    if(b>c)
        Printf("B is big");
    else
        Printf("c is big");
   }
}
```

Output:-

Enter three numbers

5 25 38

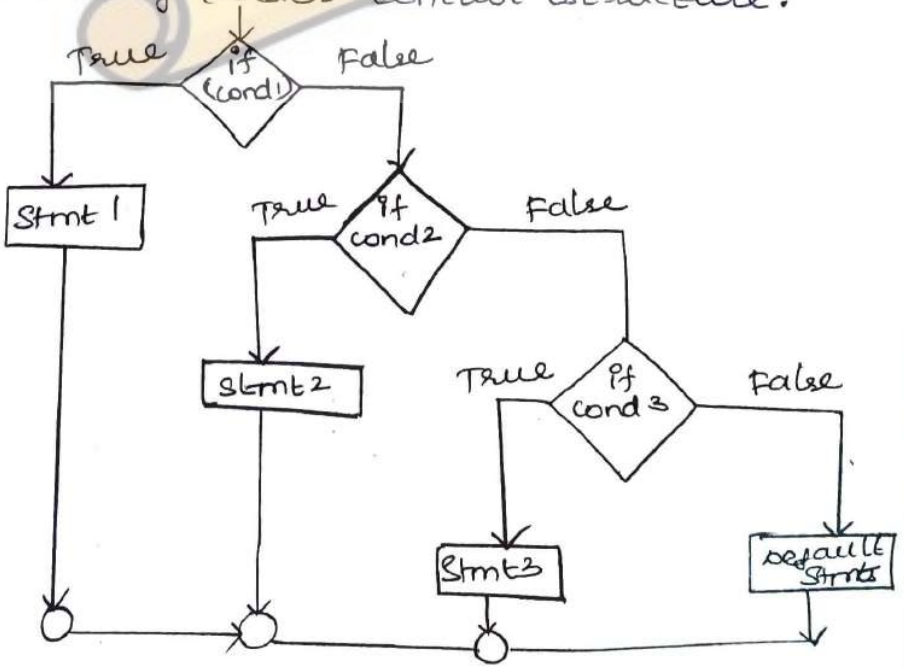
c is big

If-else-if-ladder:

The nested if statement becomes complex when there are more than three conditions. In that situation it can be represented in if-else-if ladder control structure.

Syntax:-

```
if (cond 1)
     Stmt 1;
elseif (cond 2)
     Stmt 2;
elseif (cond 3)
     Stmt 3;
Else
   default Stmt
```



Example:-

```
#include <stdio.h>
void main()
{
int avg;
Printf ("Enter the avg mark");
if(avg>90)
      printf ("O Grade");
else if(avg>80)
      Printf ("A+ Grade")
else if(avg>70)
      Printf ("A Grade")
else if (avg>60)
      Printf ("B+ Grade")
else if (avg>=50)
      Printf ("B Grade")
else
    Printf ("Fail")
}
```

Output:-

Enter the avg mark

65

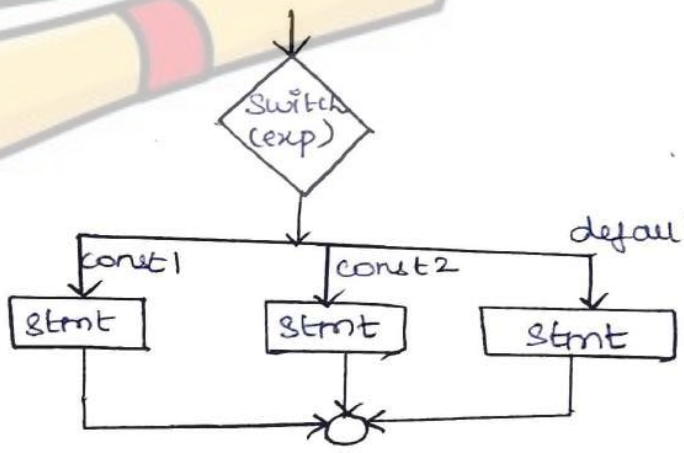
B+ Grade

Switch case statement

The switch statement is a multiple branching (multiway decision making) statement. The switch statement selects a particular group of statement from several available groups.

Syntax.

```
switch (exp)
{
case const1 : Stmt;
              break;
Case const2 : Stmt;
              break;
      .
      .
default : Stmt;
}
```



Example:-

```
#include<stdio.h>
void main()
{int a,b,c;
 char op;
 Prent f("+ADD \n-SUB\n *-MUL\n");
 Print f("Enter the case value");
```

```
Scanf ("%c", &op);
Printf ("Enter the values");
Scanf ("%d %d", &a, &b);
switch(op)
{
case '+' : c = a+b;
           break;
case '-' : c = a-b;
           break;
case '*' : c = a*b
           break;
}
Printf ("Result = %d", c);
}
```

Output:

+ Add
- Sub
* mul

Enter the case value
+
Enter the values
5 10
Result = 15

Rules for writing Switch() Stmt:-

* The expression in switch statement must be an integer value or a character constant.
* No real numbers are used in expression.
* Each case block and default blocks must be terminated with break statements.
* The default is optional.
* No two case constants are identical.
* The switch can be nested.
* The value of switch expression is compared with the case constant expression in the order specified.
* In the absence of break statement, all statements that are followed by matched cases are executed.

Looping Statement (Iteration)

Looping is the process of repeating the same set of statements again and again until the specified condition holds true.

Loops are classified as,

1) Counter controlled loops
2) Sentinal controlled loops.

Counter Controlled loop:-

In counter controlled loop the number of iterations to be performed is known in advance. It is also known as definite repetition loop.

Sentinal controlled loop:-

In Sentinal controlled loop the number of times the iteration is to be performed is not known before. It is also known as indefinite repetition loop.

Three types of looping statements,

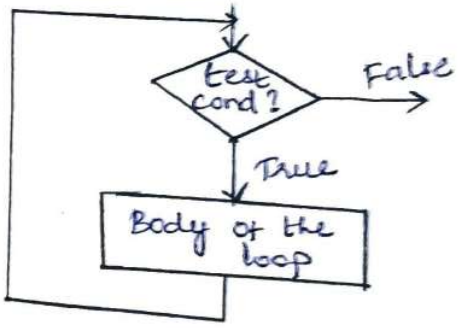
* while loop * do while loop * for loop

while loop:-

The while loop is an entry controlled loop, because the control condition is placed at the first line of the code. If the control condition evaluates to false, then the statements enclosed in the loop are never executed.

Syntax:-

```
initialization
while (condition)
{
  body of the loop
  inc/dec
}
```



Example:-

```
#include<stdio.h>
void main()
{ int n=0
 while(n<=5):
  {
   printf("%d",n);
   n++;
  }
}
```

output:-

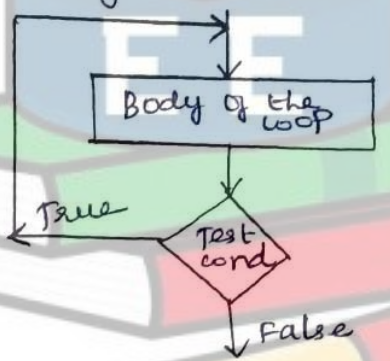
0 1 2 3 4 5

do-while loop:-

The do-while loop is an exit controlled loop because the test condition is evaluated at the end of the loop. The body of the loop gets executed atleast one time. The test condition must be terminated by (;) semicolon in do-while loop.

Syntax:-

```
initialization
do
{
 body of the loop
 inc/dec
}while(cond);
```



Example:-

```
#include<stdio.h>
void main()
{
 int n=0;
 do
  {
   printf("%d",n);
   n++;
  }while(n<=5);
}
```

o/p:-

0 1 2 3 4 5

for loop:-

It is a definite loop. It is used to execute a set of instructions repeatedly, until the condition becomes false.

Syntax:-

```
for(initialization ; condition ; increment/decrement)
{
   body of the loop
}
```

Initialization:-

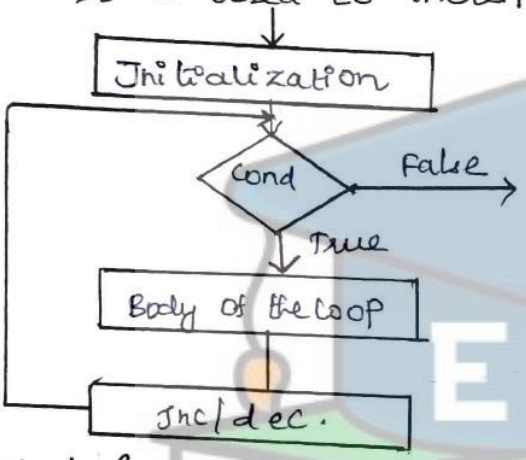
It is used to initialize or assign a starting value to the loop counter. If the loop counter has already been initialized the initialization expression can be skipped, but a semicolon is necessary and must be placed.

Condition:-

It is used to test the condition.

Increment/decrement:-

It is used to increment/decrement a countervariable



Example:-

```
#include<stdio.h>
Void main()
{
 int i;
for(i=1;i<=5;i++)
 Printf("%d", i);
}
```

output:-

1 2 3 4 5

Nested for loop:-

The loop within the loop is called nested for loop. The number of iterations in this type of structure will be equal to the number of iterations in the outerloop multiplied by the number of iterations in the inner loop.

Example:-

```
#include<stdio.h>
Void main()
{
 int i,j;
 for(i=1;i<=3;i++)
  {printf("\n");
   for(j=1;j<=3;j++)
        Printf("%d", j);
  }
}
```

output:-

1 2 3
1 2 3
1 2 3

Unconditional statement

The unconditional statement transfer the control from one point to another without checking any condition.

* goto
* break
* Continue

goto statement:

The goto statement is used to branch unconditionally from one point to another point in the program. The goto statement requires a label in order to identify the place where the control to be transferred.

Syntax:-

```
goto label;
- - - - -
label:
```

Example:-

```
#include<stdio.h>
void main()
{ int a=50, b=20;
 if(a>b):
    goto big;
 else
    printf("B is big");
 big:
    printf("A is big");
```

output:-
A is big

Break statement:-

It is used to terminate the loop, when the keyword break is used inside any 'c' loop, control is automatically transferred to the first statement after the loop.

Syntax:-

```
break
```

Example:-

```
#include<stdio.h>
void main()
{
 int i;
 for(i=1;i<=5;i++)
 { if(i==3)
     break;
   printf("%d",i);
 }}
```

o/p:
1
2

Continue Statement

The continue statement is used to continue next iteration of loop statement, when it occurs in the loop, it does not terminate but it skip the statement after this statement.

Syntax:

```
Continue;
```

Example :-

```
#include <stdio.h>
void main()
{
 int i;
 for(i=1;i<=5;i++)
 {
   if(i==3)
     continue;
   Printf("%d", i);
 }
}
```

output:

1
2
4
5

Preprocessor - Directives:

The preprocessor as the name implies, is a program that processes the source code before it passes through the compiler.

The Preprocessing language consist of directives to be executed and macros to be expanded.

* Inclusion of header files
* Macro expansion
* Conditional compilation
* Other directives

Inclusion of Header files:-

A header file is a file containing c declarations and macro definitions to be shared between several source files.

This is used to include an external file, which contain functions or some other macro definitions to our source program.

Syntax:

```
#include <filename>
#include filname
```

when the file is included in angle brackets (< >) the included file is searched only in standard directories.

Macros:

A macro is a fragment of code which has been given a name. Whenever the name is used it is replaced by the contents of the macro.

(i) macro substitutions:-

This is used to define Symbolic constants in the source Program.

Syntax:

```
#define identifier (string/integer)
```

The preprocessor accomplishes the task specified in #define statement

Simple macros

This is commonly used to define symbolic constants.

```
#define age 18
#define pi 3.14
```

Argumented macros

The argumented macros are used to define more complex and useful form of replacement in the source program.

```
#define identifier (v1,v2,vs) str/int
```

Nested macro:

The macros defined within another macro called nested macro

```
#define A 5
#define B A+5
```

Function like macro

Function like macros are more complex than object like macro. A function like macro definition declares the names of formal parameters within Parenthesis, separated by commas.

Syntax:-

```
#define macroname(parameterlist) replacei sting
```

Example:-

```
#include <stdio.h>
#define sqr(x) (x*x)
void main()
{
 int a=5;
 Printf("Area = %d", sqr(a));
}
```

Conditional compilation:

A conditional is a directive that instructs the preprocessor to select whether or not to include a chunk of code in the final token stream passed to the compiler.

#ifdef directive:

The #ifdef is the simplest sort of conditional preprocessor directive and is used to check for the existence of macro definition.

Syntax:

```
#if def macro
    controlled text
#endif
```

#ifndef directive

The #ifndef directive is the opposite of #ifdef directive. It checks whether the macro has not been defined or if its definition has been removed with #undef,

Syntax:

```
#ifndef macro
    controlled text
#endif.
```

#if directive:

The #if directive is used to control the compilation of portions of a source file.

Syntax:-

```
# if condition
    controlled text
#endif.
```

#else directive:

The #else directive can be used within the controlled text of a #if directive to provide alternative text to be used if the condition is false.

```
#if condition
    controlled text
#else
    controlled text
#endif
```

#elif directive:

The #elif directive is used when there are morethan two possible alternatives. The #elif directive like the #else directive is embedded with the #if directive.

Syntax:

```
#if condition
    controlled text
#elif cond
    controlled text
#else
    controlled text
#endif
```

#endif directive:

It is used to end the conditional compilation directive.

Other directives.

#error:

It is used to produce compiler-time error message.

Syntax:-

```
#error string
```

Predefined Macro Names:

- __LINE__
- __FILE__
- __DATE__
- __TIME__
- __STDC__
- __TIMESTAMP__

Compilation Process:-

The Process of creating a Program and linking with c library to perform the tasks is called execution of the Program.

c Program executes in four steps.

1) Creating the program
2) compiling the program
3) linking the program
4) execution of program

Creating the Program:-

Creating the program means entering and editing the Program in standard c editor and save the program with .c extension.

Compiling the Program:-

It is the Process of converting the highlevel language program into machine understandable. The source program statements should be translated into object programs which is suitable for, execution by the computer, compiler converts executable code to binary code or object code.

Highlevel lang (Sourcecode) → Translator (Compiler) → Machine language (object code)

Linking the program:

* The object code of a Program is linked with libraries that are needed for execution of a program.
* It Creates a file with '*.exe' extension.
* It links the seperately compiled functions together into one program.
* It combines the functions in the standard c library with the objcode (*.Obj) extension.

Execution of Program:

After the compilation the executable object code will be loaded in the main memory and the program is executed.

TWO MARKS

1) What is the importance of keyword in C. (AU Apr/may 2015)

Keywords are reserved words whose meaning has already been explained to the compiler. The keywords are also called reserved words.

2) What do you mean by C Tokens? (AU may/June 2012)

The Smallest individual units of C programs are known as C Tokens. (e.g) keywords, identifiers, constants, String, operators, Special characters.

3) List out the rules to be followed by identifiers?

* An identifier must begin with a letter (or) underscore character.
* Both uppercase and lowercase letters are permitted
* An identifier cannot be any one of the keywords.
* No space and special symbols are allowed between the identifier.

4) What are Variables? Give examples? (AU may/June 2016)

A variable is an identifier that is used to represent some specified type of information within a designated portion of the program.

Example: int sum;

5) Write a note on register storage class? (AU Apr/may 2015, Jan 14)

* The register storage class suggests that the access to the declared object should be as fast as possible.
* The object of an identifier for which the register storage class has been specified in stored in CPU register.
* The register storage class object will have automatic.

6) Define Implicit type conversion (AU May/June 2016)

Implicit type conversion also known as automatic type conversion. It is done by the compiler on its own, without any external trigger from the user.

Example:-

```
int a=5, b=20;
float c;
c=a+b;
```

The resultant value will be float.

7) Give the use of preprocessor (AU Apr/May 2015)

Preprocessor is a translator that converts a program written in one high level language into an equivalent program written in another high level language.

8) What is the use of #define preprocessor? (AU NOV/DEC 2015, 2014)

#define is used to define constant value and can be any of the basic datatypes.

(e.g) #define pi 3.14

9) Distinguish between while and do-while statement?

while	do-while
*In while loop a condition is evaluated at the beginning of the loop.	*In do while loop a condition is evaluated at the end of the loop.
*The body of the loop is not executed when the value of the condition is false.	*The body of the loop will be executed at least once even the condition is false.

10) Write a for loop statement to print numbers from 10 to 1 (AU Jan 2014)

```
#include<stdio.h>
void main()
{
int i;
for(i=10;i>=1;i--)
printf("%d",i);
}
```

UNIT- II

ARRAYS AND STRINGS

Introduction to Arrays : Declaration, Initialization - One dimensional array - Two dimensional Arrays - String operations : Length, compare, Concatenate, copy - Selection sort, linear and binary search

Introduction to Arrays

An array is a collection of similar data elements. These data elements have the same data type. The elements of the array are stored in consecutive memory locations and are referred by an index is also known as subscript.

Declaration of Arrays

An array must be declared before being used.

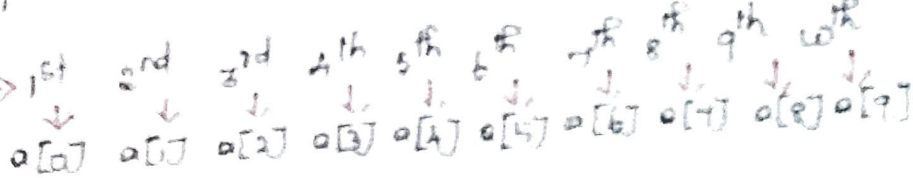
Syntax: datatype arrayname [size];

datatype - what kind of values it can store (int, char, float, double)

Name - to identify the array

size - the maximum number of values that the array can hold

Example: int a[10] - the statement declares a to be an array containing 10 numbers. In C the array index starts from 0 and ends with n-1



Accessing the elements of an array:

For accessing an individual element of the array, the array subscript must be used. To access all elements of the array we must use a loop.

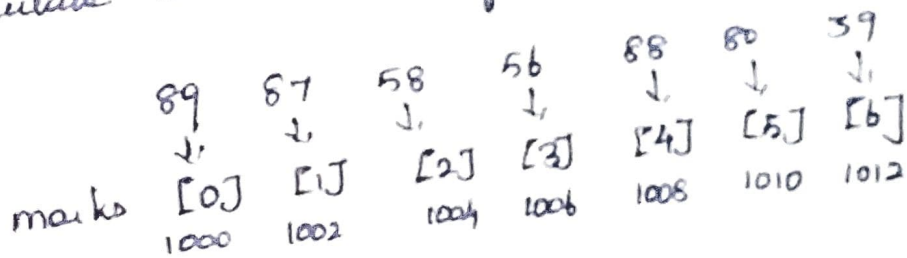
Code to initialize each element of the array to 100.

```
int i, num[10];
for (i=0; i<10; i++)
    num[i] = 100;
```

Calculating the address of Array elements

An array stores all its data elements in Consecutive memory locations, storing just the base address (i.e) the address of the first element in the array is sufficient. The address of other data elements can simply be calculated using the base address.

eg: Given an array int marks[] = {89, 87, 58, 56, 88, 80, 39} calculate the address of marks[5] if base address = 1000



address (marks[5]) = 1000 + 2(5-0)
= 1000 + 10
= 1010.

We know that storing an integer value requires 2 bytes, therefore, word size is 2 bytes.

Calculating the length of an array

Length of the array is given by the number of elements stored in it.

length = upper bound - lower bound + 1

Storing values in arrays:

When we declare an array, we are just allocating space for the elements. no values are stored in the array. To store values in the array, there are three ways.

* Initialize the array element at the time of declaration
* Inputting values from the keyboard.
* Assigning values to individual elements.

* Initializing the arrays during declaration

Elements of the array can also be initialized at the time of declaration as other variables. When an array is initialized, we need to provide a value for every element in the array.

type arrayname [size] = { list of values }.

Eg int marks [5] = {80, 85, 90, 95, 98}.

an array with name marks is declared, that has enough space to store 5 elements.

While initializing the array at the time of declaration, the programmer may omit the size of the array.

Eg int marks[] = {90, 95};

80	85	90	95	75	int marks[5] = {80, 85, 90, 95, 75}
90	95	0	0	0	int marks[5] = {90, 95}
0	0	0	0	0	int marks[5] = {0};
[0]	[1]	[2]	[3]	[4]	

* Inputting values from the keyboard:

In this method, a while / do while or a for loop is executed to input the value for each element of the array.

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

* Assigning values to individual elements.

Any value that evaluates to the datatype of the array can be assigned to the individual array Elements. A simple assignment statement can be written as.

```
marks[3] = 98;
```

We cannot assign one array to another array, even if the two arrays have the same type and size.

copy an array at the individual element level

```
int i, a[5], arr1[5];
arr[5] = {1, 2, 3, 4, 5}
for(i=0; i<5; i++)
    arr1[i] = arr[i];
```

Two Dimensional Arrays

A two dimensional array is specified using two subscripts where one subscript denotes row and the other denotes column.

Declaring two dimensional Arrays:

The two dimensional arrays must be declared before being used. The declaration statement tells the Compiler the name of the array, the data type of each element in the array, and the size of each dimension.

Syntax:

data type arrayname [rowsize] [column size];

Eg a[3][3].

a[0][0]	a[0][1]	a[0][2]
a[1][0]	a[1][1]	a[1][2]
a[2][0]	a[2][1]	a[2][2].

Initializing two dimensional array:

int a[3][3] = {1, 2, 3, 4, 5, 6, 7, 8, 9}

The initialization of a two dimensional array is done by row by row. The above statement can also be written as int a[3][3] = {{1,2,3}, {4,5,6}, {7,8,9}}.

Accessing the elements of 2 dimensional array:

A two dimensional array contains two subscripts. we will use two for loops to scan the elements. The first for loop will scan each row in the 2D array and the second for loop will scan individual column for every row in the array.

Program to print the elements of a 2D array:

```
# include <stdio.h>
int main()
{ int a[2][2] = {75, 80, 85, 98};
  int i,j;
  for (i=0; i<2; i++)
  { printf("\n");
  for (j=0; j<2; j++)
   printf ("%d \t", a[i][j]);
  }
}
```

output:

75 80

85 98

Strings

In c language, a string is a null terminated character array. This means after the last character, a null character ('\0') is stored to signify the end of the character array.

eg char str[] = "program";

p	r	o	g	r	a	m	\0

char str[] = ""; \0 → Empty string.

When the compiler encounters a sequence of characters enclosed in the double quotation marks, it appends a null character '\0' at the end by default.

Operations on strings

* Length of a string
* compares two string
* concatenates two string
* copying

Length of a string :

The number of character in the string constitutes the length of the string. To find the length all blank spaces are counted as characters in the string.

eg

```
# include <stdio.h>
void main ()
{
char str[50], i=0, len;
printf ("Enter the string:");
gets (str);
while (str[i] != '\0')
  i++;
len = i;
printf ("Length of string %d", len);
};
```

output:

Enter the string: programming

Length of string : 11

Comparing two strings

If s1 and s2 are two strings then comparing two strings will give either of those results.

* s1 and s2 are equal.
* s1 > s2, when in dictionary order s1 will come after s2.
* s1 < s2, when in dictionary order s1 precedes s2.

→ To compare the two strings, each and every character is compared from both the strings. If all the characters are same than the two strings are said to be equal.

→ We first check whether the two strings are of same length. If not, then there is no point in moving ahead as it. Straightway means that the two strings are not same.

→ If the two strings are of the same length, then we compare character by character to check if all the characters are same, if yes, then the variable equal is set to 1. else same = 0, then we check which string precedes the other in dictionary order and print the corresponding message.

```
# include <stdio.h>
# include <string.h>
void main ()
{
char str1[50], str2[50];
int i=0; len1=0, len2=0, equal=0;
Print f (" Enter String 1 ");
gets (Str1);
puts (" Enter string 2 ");
gets (Str2);
len1 = Strlen (str1);
len2 = strlen (str2);
if (len1 == len2)
{ while (i< len1)
  { if str1[i] == str2[i].
    i++;
  else
    break
  if (i == len1)
  { equal = 1;
    printf (" 2strings are Equal")
if(len1 != len2)
  printf (" two strings are not Equal")
if equal == 0.
{ if str1[i] > str2[j])
  printf (" str1 is big ");
else
  printf (" str2 is big ");
```

output: Enter str1 : flower
Enter str2 : flower
The strings are eq

Concatenating two strings

If s1 and s2 are two strings, then concatenation operation produces a string which contains characters of s1 followed by the characters of s2.

Eg

```
# include <stdio.h>
# include <conio.h>
int main()
{ char str1[100], str2[100], str3[100];
  int i=0, j=0;
  printf ("Enter the first string");
  gets (str1);
  printf ("Enter the second string");
  gets (str2);
  while (str1[i] != '\0')
  { str3[j] = str1[i];
    i++;
    j++;
  }
  i=0;
  while (str2[i] != '\0')
  {
  str3[j] = str2[i];
   i++;
   j++;
  }
str3[j] = '\0';
printf ("The concatenated string is :");
puts (str3);
getch();
return 0;
}
```

output:

```
Enter the first string : python
Enter the second string : Program
the concatenated string is
python program
```

Copying string

String copy is used to copy the contents of one string into another string

```
#include <stdio.h>
void main()
{ char s1[50], s2[50];
  int i;
  printf("Enter the string \n");
  gets(s1);
  i=0;
  while(s1[i]!='\0')
  {
    s2[i]=s1[i];
    i++;
  }
  s2[i]='\0';
  printf("copied string : %s", s2);
}
```

Output:

```
Enter the string : welcome
copied string : welcome.
```

Searching techniques

Searching is the process of determining whether an element is present in a given list of elements or not.

Linear search.

It is the simplest technique. The search begins at one end of the list & searches for the required element one by one until the element is found or till the end of the list is reached.

```
#include <stdio.h>
void main()
{ int a[50], x, i, n;
  printf("Enter the size of array\n");
  scanf("%d", &n);
  printf("Enter the array elements");
  for(i=0; i<n; i++)
  scanf("%d", &a[i]);
  printf("Enter the element to search");
Scanf("%d", &x);
for(i=0; i<n; i++)
{ if (a[i]==x)
  { printf("element found");
    break;
  }
if (i==n);
    printf("element is not found");
}
```

output:

Enter the size of array 5
Enter the elements 10 15 35 14 26
Enter the element to search 14
Element is found.

Binary search.

Binary search requires the list to be sorted in ascending order. The Binary search algorithm begins by comparing the element that is present at the middle of the list.

If there is a match then the search ends and the location of the middle element is returned. If there is a mismatch with the middle element and the search element is less than the middle element then first part of the list is searched, otherwise second part of the list is searched.

This process continued until the search element is equal to the middle element or the list contains only one element that is not equal to the search element.

```
#include <stdio.h>
#include <conio.h>
void main()
{
int i, first, last, mid, n, x, a[50];
printf("Enter the elements");
scanf("%d", &n);
printf("Enter the elmt in ascending
                    order");
for(i=0; i<n; i++)
scanf("%d", &a[i]);
printf("Enter the elmt to search");
Scanf("%d", &x);
```

```
first = 0
last = n-1
mid = (first + last)/2
while (first <= last)
{ if (a[mid] < x)
      first = mid + 1
else if (a[mid] == x)
{ printf ("Elmt is found");
  break;
}
else
  last = mid - 1
mid = (first + last)/2
if (first > last)
printf ("Elmt is not found in list");
}
```

Output:

```
Enter number of elements
5
Enter the elements in sorted order
12 14 35 56 67
Enter the element to seach 56
Element is found.
```

Selection sort

The selection sort determines the minimum of the list and swaps it with the element at the assumed minimum index.

If the array contains n elements to be sorted in their correct sequence, the first elmt is compared with the remaining elements and which is the lower element, place that element in first position. Then the second elmt from the array is taken and compared with the remaining n-2 elements. If an element with a value less than that of the second element is found in the n-2 elements, it is swapped with the second element of the array and so on. Continue this process until all the elmts in an array are sorted.

```
#include <stdio.h>
void main()
{
int a[50], i, n, j, temp;
printf ("Enter the no. of Elmts");
scanf ("%d", &n);
for (i=0; i<n; i++)
  scanf ("%d", &a[i]);
for (i=0; i<n-1; i++)
```

```
{
 min = i;
for (j = i+1 ; j<n ; j++)
{ if (a[min]>a[j])
     min = j;
}
if (min != i)
{ temp = a[i];
  a[i] = a[min];
  a[min] = temp;
 }
}
printf ("Sorted Elmts \n");
for (i=0 , i<n; i++)
printf ("%d", a[i]);
}
```

output:

Enter the no of Elements : 5

6 3 9 4 1

Sorted elements

1 3 4 6 9.

Extra programs

Matrix addition:

```
#include <stdio.h>
void main()
{
int a[5][5], b[5][5], c[5][5];
int i, j, m, n;
Printf(" Enter the row & col size");
scanf ("%d%d", &m, &n);
printf (" Enter Matrix A Elmts");
for (i=0; i<m ; i++)
{for (j=0; j<n; j++)
 {
 scanf ("%d", &a[i][j]);
 }
}
Printf ("Enter the B Matrix Elements");
for (i=0; i<m; i++)
{
for (j=0 ; j<n ; j++)
{
scanf ("%d", &b[i][j]);
 }
}
Printf (" Matrix addition ")
for (i=0 ; i<m ; i++)
{ for (j=0 ; j<n ; j++)
{
c[i][j] = a[i][j] + b[i][j];
 }
}
```

```
Printf("Resultant matrix ");
for (i=0; i<m; i++)
{
  Printf("\n");
for (j=0; j<n; j++)
{
printf ("%d\t", c[i][j]);
 }
}
}
```

Scaling Matrix

```
#include <stdio.h>
#include <conio.h>
void main()
{
int a[10][10], i, j, m, n, sf;
Printf("Enter the size of a matrix");
scanf ("%d %d", &m, &n);
printf("Enter the values of a matrix")
for (i=0; i<m, i++)
{
for (j=0; j<n; j++)
{scanf ("%d", &a[i][j]);
 }
}
printf("Enter the scaling factor\n");
scanf ("%d", &sf);
for (i=0, i<m; i++)
 {for (j=0; j<n, j++)
 {  a[i][j] *= sf
 }}
```

```
Printf("Scaled matrix is \n");
for (i=0, i<m; i++)
{ printf ("\n");
for (j=0; j<n; j++)
{printf ("%d", a[i][j]);
 }
}
}
```

Output

```
Enter the size of a matrix
3 3
Enter the value of a Matrix
2 4 6 3 6 9 4 8 12
Enter the scaling factor
2
Scaled Matrix is
4 8 12
6 12 18
8 16 24
```

Compute mean value of N numbers

```
# include <stdio.h>
# include <conio.h>
void main()
{ int n, i, a[20];
  float mean, sum = 0.0;
 Printf ["Enter the no of elmts");
 Scanf ("%d", &n);
 Printf (" Enter the elements");
for [i=0; i<n; i++)
scanf (" %d ", &a[i]);
for [i=0; i<n; i++)
  sum = sum + a [i];
  mean = sum/n;
Printf (" mean = %f \n", mean);
```

output

Enter the no. of elements 6
Enter the elements : 8 3 9 4 5 6
mean = 5.8333

Compute median value of n numbers

```
# include <stdio.h>
void main()
{ int a[10], i, j, t, n;
  float median;
 Printf (" Enter the elements");
 Scanf ("%d", &n);
 Printf (" Enter the elements");
for (i=0; i<n; i++)
scanf ("%d", &a[i]);
for (i=0; i<n; i++)
for (j=j+1; j<n ; j++)
{ if (a[i]>a[j])
{   t=a[i]; a[i]=a[j]; a[j]=t;
}
} if (n%2) ==0
 median = (a[n/2]+a[n/2 -1])/2 ;
else
 median = a[(n-1)/2];
Printf (" Median = %f ", median);
```

output

Enter the no of Elmts 6
Enter the elements 2 5 4 8 6 9
Median = 5.000

Compute mode value of n No's.

```
# include <stdio.h>
Void main()
{ int i, j, n, k=0, c=1, max=0
 int mode a[20]={0}, b[20]={0};
Printf ["Enter the limit");
Scanf ["%d", &n);
Printf (" Enter the numbers" );
for (i=0; i<n; i++)
scanf ["%d", &a[i]),
for (i=0; i<n-1 ; i++)
{ mode =0;
 for (j=j+1; j<n, j++)
{ if(a[i]==a[j])
  mode++
} if ((mode>max) && (mode !=0))
{ k=0; max =mode; b[k]=a[i]; k++;
}
else if (mode ==max)
{ b[k]=a[i]; k++ ; }
} for (i=0 ; i<n; i++)
{ if (a[i]==b[i])
    c++;
}}
```

```
if (c==n)
 printf("there is no mode")
else
{
printf("Mode ");
for(i=0; i<k; i++)
printf("%d", b[i]);
}
}
```

output.

Enter the limit 5

Enter the numbers

1 2 1 2 1

Mode = 1

Two marks.

1. Define Array?

Array is a collection of similar type of values. All values are stored in continuous memory locations. All values share a common name. The elements are organized in sequential order.

Eg int a[10]

2. How can create an array in c?

General syntax to create an array.

Syntax : <data type> <arrayname> [size];

Example : int a[10];

* Arrays are referenced with the help of the index values.
* [] notation to represent the variable as an array.
* Assume that the array contains n integers, then the first element are indexed with the 0 value and the last element are indexed with n-1 value.

3. Write example code to declare two dimensional array.

char name [b][w]

* Here we have two indexes / subscripts. These two indexes are referred as rows and columns. [b] refers to rows and [w] refers to columns.

4. List the String function available in C.

* getchar() } single character functions
* putchar()
* gets() } Multi character functions.
* puts()

5. Define String with example?

* array of characters is called a string. A string is terminated by a null character \0.

* For example, consider: "c programming". Here "c programming" is a string. When, compiler encounters strings, it automatically appends a null character \0 at the end of string.

Example. char c[] = "c programming"

6. Give an example for initialization of string array.

```
# include <stdio.h>
void main()
{ char str[] = "Hello";
  printf("given string : %s", str);
}
```

O/P

given string : Hello

7. Name any two library functions used for String Handling?

* Strcmp()
* Strlen()
* Strcat()
* Strncat()
* Strcpy()
* Strncmp()

8. What is a String library?

A set of simple string manipulation functions are implemented in <String.h> or on some systems in <Strings.h>. The String library has some useful functions for working with strings like Strcpy, Strcat, Strcmp, Strlen, Strcoll etc.

9. What is mean by sorting.

Sorting is the process of arranging a list of elements in a particular order. There are many types of sorting techniques.

* Bubble sort
* Insertion sort
* selection sort
* Quick Sort
* Merge sort
* Heap sort.

Unit III
Functions and Pointers

Introduction to functions.

A function is a set of instructions that are used to perform specified tasks which repeatedly occurs in the main program. Every function interfaces to the outside world in terms of how information is transferred to it and how results generated by it and transmitted back. This interface is specified by function name.

```
main()
{
  ...
  func();
}
```

```
func()
{
  stmts...
}
```

The main() calls a function named func(). Therefore main() is known as the calling function and func() is known as called fun

Functions can be classified into two types
* Userdefined function
* Built-in function.

Userdefined function.

The function defined by the users according to their requirements are called userdefined functions. The user can modify the function according to their requirement.

Elements of user-defined functions.
* Function declaration
* Function definition
* Function call.

Function declaration.

Function declaration is a declaration statement that identifies a function with its name, a list of arguments that it accepts, and the type of data it returns.

Syntax.
```
Returndatatype functionname(datatype var1, datatype var2...);
```

function name is a valid name for the function, naming a function follows the same rules as naming variables. The function name is used to call it for execution in a program.

Return datatype specifies the data type of the value that will be returned to the calling function as a result of the processing performed by the called function.

datatype var1, datatype var2.... is a list of variables of specified datatypes. These variable are passed from the calling function to the called function. They are also known as arguments (or) Parameters. After the declaration of every function there should be a semicolon (;)

Function definition.

When a function is defined, space is allocated for that function in the memory. A function definition comprises two parts * Function header * Function body.

Syntax.

```
returndatatype funname(datatype var1, datatype var2...)
{
    statements
    - - -
    - -
}
return(var);
```

The number of arguments and the order of arguments in the function header must be same as that given in the function declaration statement.

returndatatype funname(datatype var1, datatype var2...) is known as function header. The function header is same as function declaration. The function header is not followed by a semicolon. The list of variables in the function header is known as the formal parameter list.

The parameter list may have zero or more parameters of any data type.

Function call.

The function call statement invokes the function. When a function is invoked the compiler jumps to the called function to execute the statements that are part of that function.

Syntax

```
functionname(var1, var2, - - -);
```

Function Prototype

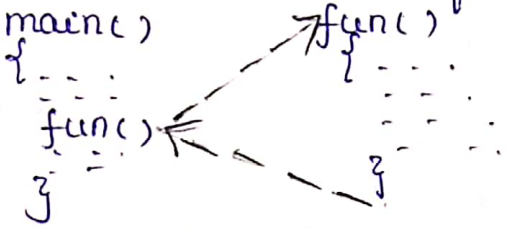
The functions are classified into the following types depending on whether the arguments are presents or not, whether a value is returned or not. These are called function Prototypes.

A function prototype declaration consists of the function's return type, name and arguments itself. It is always terminated with semicolon.

* Function with no arguments and no return value.
* Function with argument and no return value
* Function with arguments and with return value
* Function with no arguments and with return value.

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

In this prototype, no data transfer take place between the calling function and the called function.



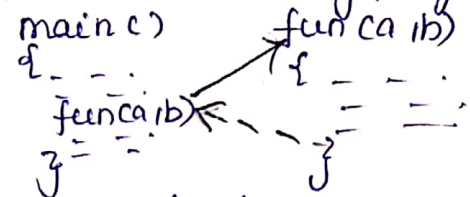

```
(eg) #include<stdio.h>
     void sum();
     void main()
     {
       void sum();
       sum();
     }
     void sum()
     {
       int a,b,c;
       printf("Enter 2 values\n");
       scanf("%d %d",&a,&b);
       c=a+b;
       printf("sum=%d",c);
     }
```

o/p

Enter 2 values
50 60
Sum=110

(ii) Function with arguments and no return values.

In this prototype, data is transferred from calling function to called function. The called program receives some data from the calling program and does not send back any values to calling program.



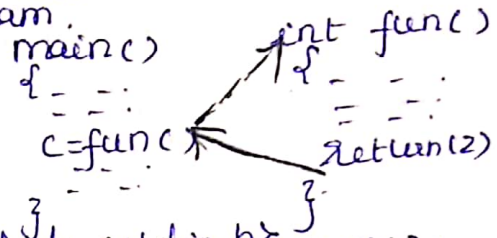

```
#include<stdio.h>
void sum(int,int);
void main()
{
  void sum(int,int);
  int a,b;
  printf("Enter 2 values\n");
  scanf("%d %d",&a,&b);
  sum(a,b);
}
void sum(int a, int b)
{
  int c;
  c=a+b;
  printf("sum=%d",c);
}
```

o/p

Enter 2 values
26 3
sum=29

(iii) Function with no arguments and with return value.

In this prototype one way communication take place (i.e) the calling program cannot pass any arguments to the called pgm, but the called pgm may send some return value to the calling program.



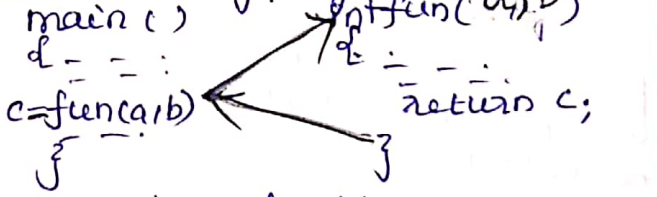

```
#include<stdio.h>
int sum();
void main()
{
  int a=sum();
  int c;
  c=sum();
  printf("sum=%d",c);
}
int sum()
{
  int a,b,c;
  printf("Enter 2 values\n");
  scanf("%d %d",&a,&b);
  c=a+b;
  return c;
}
```

o/p

Enter 2 values
10 20
sum=30

(iv) Function with arguments and with return value.

In this prototype the data is transferred between the calling function and called function. The called program receive some data from the called program and send back a value return to the calling program




```
#include<stdio.h>
int sum(int,int);
void main()
{
  int sum(int,int);
  int a,b,c;
  printf("Enter 2 values");
  scanf("%d %d",&a,&b);
  c=sum(a,b);
  printf("sum=%d",c);
}
int sum(int a,int b)
{
  int c;
  c=a+b;
  return c;
}
```

o/p

Enter 2 values
10 15
sum=25

Recursive Function.

Recursion is the process of calling the same function itself again and again until some condition is satisfied. Every recursive solution has two major cases.

Base case - in which the problem is simple enough to be solved directly without making any further calls to the same function

Recursive case - in which first the problem is divided into simpler sub-parts. Second the function calls itself but with sub-parts of the problem obtained in the first step. Third, the result is obtained by combining the solution of simpler sub parts.

(eg) Calculate the factorial of an integer number.

Base case: when n=1, the result will be 1as 1! = 1.

Recursive case: fact(n) = n * fact(n-1)

(eg)
```
#include<stdio.h>
void main()
{
   int fact(int);
   int n, f;
   Printf("Enter the number");
   Scanf("%d", &n);
   f = fact(n);
   Printf("Factorial of %d is %d", n, f);
}
```

```
int fact(int n)
{
  if (n==1)
    return 1;
  else
    return (n * fact(n-1));
}
```

o/p
Enter the number 5
factorial of 5 is 120

Calculate GCD using recursive function.

```
#include<stdio.h>
int gcd(int, int);
void main()
{
  int num1, num2, res;
  Printf("Enter 2 numbers");
  Scanf("%d %d", &num1, &num2);
  res = gcd(num1, num2);
  Printf("Gcd of %d and %d = %d",
          num1, num2, res);
}
int gcd(int x, int y)
{
  int rem;
  rem = x % y;
  if(rem==0)
    return y;
  else
    return(gcd(y, rem));
}
```

(eg) assume a=44 b=6

gcd(44, 6)
rem = 44 % 6 = 2
gcd(6, 2)
rem = 6 % 2 = 0.
return 2.

Types of Recursion.

Recursion is a technique that breaks a problem into one or more sub-problems that are similar to the original problem.

Any recursive function can be characterized based on

* Direct or indirect recursion
* Tail or Non-tail recursion
* Linear or Tail recursion.

Direct Recursion.

A function is said to be directly recursive if it explicitly call itself

```
(eg) #include<stdio.h>
int fact(int);// Function Declaration
void main()
{
   int n, f;
   Printf("Enter a number\n");
   Scanf("%d",&n);
   f=fact(n);
   Printf("Factorial of %d is %d",n,f);
}
int fact(int n)
{
   if (n==1)
      return 1;
   else return(n*fact(n-1));
}
```

Indirect Recursion

Indirect recursion occurs when a function calls another function, which in turn calls another function, eventually resulting in the original function being called again.

```
(eg) int fun(int n)
     {
        if(n==0)
           return n;
        else return fun1(n);
     }
     int fun1(int x)
     {
        return fun(x-1);
     }
```

Tail Recursion.

A recursive function is said to be tail recursive if no operations are pending to be performed when the recursive function returns to its caller.

```
(eg) int fact(int n)
     {
        return fact1(n,1);
     }
int fact1(int n, int res)
     {
        if (n==1)
           return res;
        else
           return fact1(n-1, n*res);
     }
```

Non-Tail Recursion.

In a non-tail recursion function there are pending operations to be performed on return from a recursive call. The last operation of this function is a recursive function call.

```
(eg) int fact(int n)
     {
        if(n==1)
           return 1;
        else
           return(n*fact(n-1));
     }
```

Linear and Tree Recursion.

A recursive function is said to be linearly recursive when the pending operation does not make another recursive call to the function.

(eg) The factorial function is linearly recursive, as the pending operation involves only multiplication to be performed and does not involve another recursive call to fact.

A recursive function is said to be tree recursive if the pending operations makes another recursive call to the function.

```
int fib(int num)
{
  if(num==0)
    return 0;
  else if(num==1)
    return 1;
  else
    return(fib(num-1)+fib(num-2));
}
```

(eg)

fib(5) = fib(4) + fib(3)
fib(4) = fib(3) + fib(2)
fib(3) = fib(2) + fib(1)
fib(2) = fib(1) + fib(0).
now fib(2) = 1+0 = 1
fib(3) = 1+1 = 2
fib(4) = 2+1 = 3
fib(5) = 3+2 = 5.

(1) Computation of sine series.

$x - x^3/3! + x^5/5! - \dots\dots$

```
#include<stdio.h>
#include<math.h>
void main()
{
  int fact(int x);
  float x, sum=0;
  int n, i, t=1;
  printf("Enter the value of x");
  Scanf("%f",&x);
  Printf("Enter the value of n");
  Scanf("%d",&n);
  for(i=1;i<=n;i++)
  {
  Sum=Sum+pow(-1,(i+1))*(pow(x,t))/
        ((float)fact(t));
  Printf("x^%d/%d!",t,t);
  if(i%2==0)
      peintf("+");
  else
      printf("-");
  t=t+2;
}
```

x=2, n=3

0+1*2/1

(2-1)*8/6.

```
printf("\n sum of series=%.3f",sum);
}
int fact(int num)
{
  int i,f=1;
  for(i=1;i<=num;i++)
    f=f*i;
  return f;
}
```

o/p.

Enter the value of x 3
Enter the value of n 5
x^1/1! -x^3/3! + x^5/5! - x^7/7! + x^9/9! -
Sum of series = 0.145

(2) Binary search - Recursive Function

```
#include<stdio.h>
#include<stdlib.h>
int binary(int [],int,int,int);
void main()
{
  int n,i,x,pos;
  int low,high,a[50];
  printf("Enter the total number of
                elements");
  Scanf("%d",&n);
  printf("Enter the array elements in
                sorted order");
  for(i=0;i<n;i++)
    Scanf("%d",&a[i]);
  low=0;
  high=n-1;
  Printf("Enter element to be search");
  Scanf("%d",&x);
  pos=binary(a,x,low,high);
  if(pos!=-1)
    Printf("Number found at
                Position%d",pos+1);
  else
    Printf("The number is not
                present in the list");
}
```

```
int binary(int a[], int x, int low,
                        int high)
{
  int mid;
  if(low > high)
      return -1;
  mid = (low+high)/2;
  if (x == a[mid])
      return (mid);
  elseif (x < a[mid])
      binary(a, x, low, mid-1);
  else
      binary(a, x, mid+1, high);
}
```

O/P

```
Enter number of elements
5
Enter the array elements in sorted order
5 8 12 16 23
Enter element to be search
16
Number found at position 4.
```

(3) Scientific calculator using Builtin function.

```
#include<stdio.h>
#include<math.h>
void main()
{
 float x, y, result;
 int opt;
 do
 {
 printf("Menu \n");
printf("1. Sin(x)\n 2. cos(x)\n");
printf("3. tan(x)\n 4. Sinh(x)\n");
printf("5. cosh(x)\n 6. tanh(x)\n");
printf("7. log(x)\n 8. Sqrt(x)\n");
printf("9. exp(x)\n 10. pow(x)\n");
printf("11. exit \n");
printf("Enter your choice");
scanf("%d", &opt);
switch(opt)
{
case 1 : Printf("Enter x \n");
 scanf("%f", &x);
  result = sin(x);
  printf("Sin(%f) = %f", x, result);
  break;
case 2 :
   printf("Enter x \n");
   scanf("%f", &x);
    result = cos(x);
   printf("cos(%f) = %f", x, result);
  break;
case 3 :
   printf("Enter x\n");
   scanf("%f", &x);
   result = tan(x);
   printf("tan(%f) = %f", x, result);
  break;
case 4 :
   printf("Enter x\n");
   scanf("%f", &x);
    result = sinh(x);
   printf("Sinh(%f) = %f", x, result);
   break;
case 5 :
    printf("Enter x \n");
    scanf("%f", &x);
    result = cosh(x);
   printf("cosh(%f) = %f", x, result);
   break;
case 6 :
    printf("Enter x \n");
    scanf("%f", &x);
    result = tanh(x);
    printf("tanh(%f) = %f", x, result);
    break;
case 7 :
    printf("Enter x \n");
    scanf("%f", &x);
    result = log(x);
    Printf("log(%f) = %f", x, result);
    break;
case 8 :
    printf("Enter x \n");
    scanf("%f", &x);
    result = sqrt(x);
    printf("Sqrt(%f) = %f", x, result);
    break;
case 9 :
    Printf("Enter x \n");
    scanf("%f", &x);
    result = exp(x);
```

```
Printf("exp(%f) = %f", x, result);
break;
case 10:
        printf("Enter x and y\n");
        scanf("%f %f", &x, &y);
        result = pow(x, y);
        printf("pow(x, y) = %f",
                result);
        break;
}
}while (option != 11);
 if (option == 11)
    Printf("Exit \n");
}
```

Output

```
        Menu
1. sin(x)   2. cos(x)   3. tan(x)
4. sinh(x)  5. cosh(x)  6. tanh(x)
7. log(x)   8. sqrt(x)  9. exp(x)
10. Pow(x)  11. exit
Enter your option : 1
 Enter x : 90.0
 sin(90.0) = 1.000000
        Menu
1. Sin(x)   2. Cos(x)   3. tan(x)
4. Sinh(x)  5. cosh(x)  6. tanh(x)
7. log(x)   8. Sqrt(x)  9. exp(x)
10. pow(x)  11. exit
Enter your option : 11
Exit.
```

(4) Fibonacci series – Recursion.

```
#include<stdio.h>
int fib(int);
int main()
{
  int n, i=0, res;
  Printf("Enter the number of
                terms\n");
  scanf("%d", &n);
  Printf("Fibonacci series\n");
for(i=0; i<n; i++)
 {
  res = fib(i);
  printf("%d \t", res);
 }
}
int fib(int n)
{
 if(n==0)
    return 0;
 else if(n==1)
    return 1;
 else
    return (fib(n-1)+fib(n-2));
}
```

o/p

```
Enter the number of terms
6
0  1  1  2  3  5
```

(5) Tower of hanoi – Recursion.

```
#include<stdio.h>
void move(int, char, char, char);
int main()
{
  int n;
  Printf("Enter the number of rings:");
  scanf("%d", &n);
  move(n, 'A', 'C', 'B');
  return 0;
}
void move(int n, char source, char dest,
                char aux);
{
  if (n == 1)
    Printf("\n Move from %c to %c",
                source, dest);
  else
  {
  move(n-1, source, aux, dest);
  move (1, source, dest, aux);
  move (n-1, aux, dest, source);
  }
}
```

Built-in Functions.

C language provides built-in functions called library functions. The compiler itself evaluates these functions.

The library provides a basic set of mathematical functions, string manipulation, type conversions and file and console base I/O.

Math library functions <math.h>

1. Sqrt(x)
This function evaluates squareroot of x. Note that the argument must be non-negative. Eg) sqrt(25) ⇒ 5

2. abs(x)
The absolute value of 'x' is the value of 'x' without sign. The abs() function evaluates the absolute value of an integer quantity Eg) abs(-82) ⇒ 82

3. exp(x)
This function evaluates e^x. Its general form is exp(). The argument x must be a real quantity exp(7.5)

4. Pow(a,b)
The arguments a and b are may be integer or float values. This evaluates the value of a^b

5. ceil(x)
Ceiling of a number x is the smallest integer greater than or equal to x. ceil(5.6) evaluates 6.

6. rand()
This will return a random positive integer number. Note that no arguments is necessary here.

7. Sin(x)
This evaluates the trigonometric sine value of a real or integer quantity given in radian measure. sin(30)⇒0.5

8. Cos(x)
This evaluates the trigonometric cosine of a real or integer quantity. cos(30) ⇒ 0.866

9. tan(x)
This evaluates the trigonometric tangent of a real or integer quantity given in radian measure. tan(30)⇒0.577

10. floor(x)
Return a value rounded down to the next lower integer. floor(5.6) ⇒ 5

11. acos(x), asin(x), atan(x)
Return the arc cosine, sine and tangent of x.

String Manipulation Functions (string.h)

Function	Description.
Strcat(S1,S2)	Concatenates S2 at the end of S1
Strncat(S1,S2,n)	Appends a portion of string to another.
Strcpy(S1,S2)	Copies S2 into S1
Strncpy(S1,S2,n)	Copies given number of characters of one string to another.
Strlen(S1)	Gives the length of S1
Strcmp(S1,S2)	compares two strings. Returns 0 if S1 is same as S2. Returns <0 if S1<S2, returns >0 if S1>S2.
Strcmpi(S1,S2)	compares two strings, but this function negotiates case. "A" and "a" are treated as same
Strchr(S,'a')	Returns pointer to first occurrence of char 'a' in S1
Strrchr(S,'a')	Returns last occurrence of given character in a string is found.
Strlwr(S1)	Converts string to lowercase.
Strupr(S1)	Converts string to uppercase.
Strrev(S1)	Reverses the given string.
Strtok(S1,delim)	Tokenizing given string using delimiter.
Strpbrk(S1,S2)	Returns a pointer to the first occurrence in S1 of any character in S2
Strspn(S1,S2)	The function returns the index of the first character in S1 doesnot match any character in S2.

String functions - pgm

strncat

```
#include<stdio.h>
#include<string.h>
void main()
{
  char str1[50]="Welcome";
  char str2[15]=" to C programming";
  strncat(str1,str2,4);
  printf("\nStr1 : %s",str1);
}
```

O/P
Str1 : Welcome to C

Strncpy

```
#include<stdio.h>
#include<string.h>
void main()
{
  char str1[50], str2[10]="Good";
  strncpy(str1,str2,2);
  printf("\nStr1 : %s", str1);
}
```

O/P Go

Strcmpi ($)

```
#include<stdio.h>
#include<string.h>
void main()
{
  char str1[20]="Hello", str2[20]="hello";
  if(strcmpi(str1,str2)==0)
    printf("Two strings are equal");
  else
    printf("Two strings are not equal");
}
```

O/P
Two strings are equal

Strupr, strlwr.

```
#include<stdio.h>
#include<string.h>
void main()
{
  char S1[10]="String", S2[15]="Program";
  strupr(S1);
  strlwr(S2);
  puts(S1);
  puts(S2);
}
```

O/P
STRING
program

Strrev(S1)

```
#include<stdio.h>
#include<string.h>
void main()
{
  char str[50];
  puts("Enter string");
  gets(str);
  strrev(str);
  puts(str);
}
```

O/P
Enter String
program
margorp

Strtok ( )

```
#include<stdio.h>
#include<string.h>
void main()
{
char str[]="Function,returns,a,ptr";
char delim=",";
char res[25];
res=strtok(str,delim);
while(res!=NULL)
{printf("\n %s",res);
  res=strtok(NULL,delim);
}}
```

O/P
Function
returns
a
ptr.

Strpbrk.

```
#include<stdio.h>
#include<string.h>
void main()
{
char str1[]="C program"
char str2[]="AB"
char *ptr=strpbrk(str1,str2)
if(ptr==NULL)
  Printf("No characters matches in strings");
else
  printf("characters in string2 matches in string1");
}
```

O/P
No characters matches in strings

Pointers.

A pointer is a variable that contains the memory location of another variable. It is a derived datatype in C language.

Declaring pointer variables.

datatype *ptrname;

eg. int *num;
char *ch;
float *fnum;

Initialization of a pointer variable.

The process of assigning the address of a variable to a pointer variable is known as initialization. A pointer can be assigned or initialized with the address of an object.

A pointer variable cannot hold a non-address value. A pointer can be assigned or initialized with another pointer of the same type. However it is not possible to assign a pointer of one type to a pointer of another type.

(eg) int num = 15;
int *ptr = &num.

```
eg #include<stdio.h>
void main()
{
  int num, *ptr;
  ptr = &num;
  printf("Enter the number\n");
  scanf("%d", &num);
  printf("The number that was
          entered is %d", *ptr);
}
```

o/p
Enter the number
50
The number that was entered is 50

Dereferencing operator (*)

The dereference (*) operator is a unary operator and should appear on the left side of its operand. The object referenced by a pointer can be indirectly accessed by dereferencing the pointer. A dereferencing operation allows a pointer to be followed to the data object to which it points.

The operand of a dereference operator should be of pointer type.

Reference operator (&)

The reference to an object can be created using referencing operator (&). The reference operator is also known as address-of-operator.

Void pointer or Generic pointer.

The pointers are used for pointing to different data types. In C there is a general purpose pointer that can point to any data type and is known as void pointer.

The void pointer is a generic pointer that can represent any pointer type.

Syntax

```
void *vptr;
```

Eg)
```
#include<stdio.h>
void main()
{
  int x=10;
  char ch='a';
  void *vptr;
  vptr=&x;
  Printf("Generic pointer with int value=%d", *vptr);
   vptr= &ch;
  Printf("Generic pointer with char value=%c", *vptr);
}
```

NULL pointers.

NULL pointer is a special pointer that does not point to any value. This means that a null pointer does not point to any valid memory address.

Syntax.

```
int *nptr=NULL;
```

we may also initialize a pointer as a nullpointer by using a constant 0

Eg) `int *nptr=0.`

Null pointers are used in situations where one of the pointers in the program points to different locations at different times.

Keypoints - Null pointer.

* When a NULL pointer is compared with a pointer to any object or a function the result of comparision is always false.
* Two NULL pointers are always equal.
* Dereferencing a null pointer leads to a runtime error.

Pointer Arithmetic

Arithmetic operations can be applied to pointers in restricted form. When arithmetic operators are applied on pointers, the outcome of the operation is a pointer arithmetic

Addition Operation

An expression of integer type can be added to an expression of pointer type.

If ptr is a pointer to an object, then adding 1 to Pointer points to the next object.

(eg) int *ptr;

ptr = ptr + 5 → it determines the initial value of pointer + integer operand * size of (the ref type).

(ie) = 2000 + 5 * 2 = 2010.

The addition of a pointer and an integer is commutative. (ie) ptr+1 is same as 1+ptr.

Increment Operation.

The increment operator can be applied to an operand of pointer type.

ptr++

++ptr.

Subtraction operation.

A pointer and an integer can be subtracted. Subtraction of integer and pointer is not commutative (ie) ptr-1 is not the same as 1-ptr. The operation 1-ptr is illegal.

Two pointers can also be subtracted.

Decrement operation.

The decrement operator can be applied to an operand of pointer type

Relational Operations.

A pointer can be compared with a pointer of the same type or with zero.

C allows to compare pointers by using relational operators in the expressions.

P1 > P2

P1 == P2

P1 != P2.

```
(eg) #include<stdio.h>
     void main()
     {
     int a=5, b=10, c=0;
     int *P1=&a, *P2=&b, *P3;
     Printf("Value of P1=%d P2=%d", P1, P2);
     c = *P1 + *P2
     Printf("*P1+*P2=%d", c)
     P3 = P1 - P2;
     Printf("P1-P2=%d", P3);
     P1++;
     Printf("P1++=%d", P1);
     P2--;
     Printf("P2--=%d", P2);
     }
```

O/P.

Value of P1 = -18 P2 = -14

*P1 + *P2 = 15

P1 - P2 = 2

P1++ = -10.

P2-- = -16.

Rules for Pointer Operations.

* Only integers can be added to pointers. It is not valid to add a float or a double value to a pointer.
* Multiplication and division operators cannot be applied on pointers.
* Bitwise operators cannot be applied on pointers.
* A pointer of one type cannot be assigned to a pointer of another type
* A pointer variable cannot be assigned a non-address value (except zero).

Arrays and Pointers.

Array is a collection of similar data type elements stored under common name. Array elements are always stored in consecutive memory locations according to the size of the array.

eg) If we have declare an array as

```
int arr[] = {21,32,8,24,5};
```

then in memory it would be stored as

21	32	8	24	5
arr[0]	arr[1]	arr[2]	arr[3]	arr[4]
500	502	504	506	508

Array notation is a form of pointer notation. The name of the array is the starting address of the array in memory. It is also known as the base address

eg)
```
int *ptr; int arr[] = {21,32,8,24,5};
ptr = &arr[0];
```

here ptr is made to point to the first element of the array.

If pointer variable ptr holds the address of the first element in the array, then the address of successive elements can be calculated by writing ptr++

eg)
```
int arr[] = {10,20,30,40,50}
int *ptr = &arr[0];
ptr++;
Printf("The value of element in array: %d", *ptr);
```

Output is 20 because after initialization the ptr value is incremented by 1 it indicates the next value.

When we using pointers, an expression like arr[i] is equivalent to writing *(arr+i). If arr is the array name, then the compiler implicitly takes

arr = &arr[0]

To print the value of the second element of the array, we use the expression *(arr+1)

We can write ptr=arr; is equivalent to ptr=&arr[0] but we cannot write arr=ptr; because while ptr is a variable, arr is a constant. The location at which the first element of arr will be stored cannot be changed once arr[] has been declared. Therefore an array name is often known to be a constant pointer.

arr[i], i[arr], *(arr+i), *(i+arr) gives the same value,

(eg)
```
#include<stdio.h>
void main()
{
 int arr[5]={10,20,30,40,50};
 int *ptr, i;
 ptr = arr;
 for(i=0; i<5; i++)
 {
  printf("Element = %d", arr[i]);
  printf("Element = %d", *(arr+i));
  printf("Element = %d", ptr[i]);
  printf("Element = %d", *(ptr+i));
 }
}
```

(eg) // To read and display an array of integers.

```
#include<stdio.h>
void main()
{
 int i, n;
 int arr[10], *ptr;
 ptr = arr
 Printf("Enter the number of elements:");
 scanf("%d", &n);
 printf("Enter the elements:");
 for(i=0; i<n; i++)
    scanf("%d", ptr+i);
 printf("The entered elements are \n");
 for(i=0; i<n; i++)
    printf("%d", *(ptr+i));
}
```

O/P

Enter the number of elements
5
Enter the Elements
5 6 8 9 10
The entered elements are
5 6 8 9 10

Arrays of Pointers.

An array of pointers can be declared as

```
int *ptr[10];
```

It declares an array of 10 pointers where each of the pointer points to an integer variable.

An array of pointers is a collection of addresses. The addresses in an array of pointers could be the addresses of isolated variables or the addresses of array elements or any other addresses. The only constraint is that all the pointers in an array must be of the same type.

Eg)
```
#include<stdio.h>
void main()
{
  int a=10,b=20,c=30;
  int *arr[3]={&a,&b,&c};
  Printf("The values of variables are:\n");
  Printf("%d %d %d\n",a,b,c);
  Printf("%d %d %d\n",*arr[0],*arr[1],*arr[2]);
}
```

O/P

The values of variables are
10 20 30
10 20 30.

Array of character pointers.

Array of character pointers that points to strings.

```
char *ptr[3];
```

In the ptr array, each element is a character pointer. Therefore, we can assign character pointers to the elements of the array.

Eg) `ptr[0]=str;` // `char *str;`

Another way to initialize an array of characters with three strings can be given as,

```
char *ptr[3]={"Meena","Anu","Mani"};
```

here ptr[0] is Meena, ptr[1] is Anu, ptr[2] is Mani

It requires only 15 bytes to store the three strings.

ptr[0]	M	e	e	n	a	\0
ptr[1]	A	n	u	\0		
ptr[2]	M	a	n	i	\0	

However char[3][10]={"Meena", "Anu", "Mani"} will behave the same way as an array of characters. The only difference is the memory layout while the array of pointers needs only 15 bytes of storage, but str will reserve 30 bytes in memory but 15 bytes only utilized other spaces are not utilized.

Sort the Names in alphabetical order.

```
#include<stdio.h>
#include<string.h>
void main()
{
  char name[50][20], temp[20];
  int i, j, n;
  printf("Enter how many names\n");
  scanf("%d", &n);
  printf("Enter the names \n");
  for(i=0; i<n; i++)
    scanf("%d", name[i]);
  for(i=0; i<n-1; i++)
  {
    for(j=i+1; j<n; j++)
    {
      if(strcmp(name[i], name[j])>0)
      {
        strcpy(temp, name[i]);
        strcpy(name[i], name[j]);
        strcpy(name[j], temp);
      }
    }
  }
Printf("The names after sort \n");
for(i=0; i<n; i++)
printf("%s", name[i]);
}
```

```
%p
Enter how many names
5
Enter the names
vijay sanjay Rahul kumar Naveen
The names after sort
kumar Naveen Rahul sanjay vijay
```

Parameter Passing Methods.

When a function is called, the calling function may have to pass some values to the called function. There are two ways in which arguments or parameters can be passed to the called function.

* call by value * call by reference.

call by value.
In which values of variables are passed by the calling function to the called function.

Call by reference.
In which address of variables are passed by the calling function to the called function.

Call by value

The process of passing the actual value of variables is known as call by value. When a function is called in program the values to the arguments in the function are taken by the calling program, the value taken can be used inside the function.

```
/* swapping of two numbers */
#include<stdio.h>
void swap(int ,int);
void main()
{
   int a,b;
   Printf("Enter the values of a and b\n");
   Scanf("%d%d", &a,&b);
   Printf("Before swap a=%d,b=%d\n", a,b);
    Swap(a,b);
}
void swap(int a,int b)
{
    int temp;
    temp=a;
      a=b;
      b=temp;
    Printf("After swap a=%d,b=%d\n",a,b);
}
```

O/P
Enter the values of a and b
28 42
Before swap a=28, b=42
After swap a=42, b=28

Call by reference.

The process of calling a function using pointers to pass the address of variable is known as call by reference. A function can be declared with pointers as its arguments. Such functions are called by calling program with the address of a variable as argument from it.

/* Program to interchange or swap value - call by Reference */

```
#include <stdio.h>
void main()
{
  int a,b;
  void swap(int *, int *);
  Printf("Enter the values of a and b");
  Scanf("%d %d",&a,&b);
  printf("Before swap a=%d, b=%d \n", a,b);
   swap(&a,&b);
}
void swap(int *a, int *b)
{
   int temp;
   temp=*a;
    *a=*b;
    *b=temp;
   Printf("After swap * a=%d, b=%d", *a,*b);
}
```

o/p

Enter the values of a and b
75 36
Before swap a=75, b=36
After swap a=36, b=75

/* Program to change the value of a variable - Pass by reference */

```
#include <stdio.h>
void add(int *n);
void main()
{
  int num;
  Printf("Enter a number \n");
  Scanf("%d", &num);
  Prinf("The value of number before calling the fun=%d", num);
  add(&num);
  printf("The value of number after calling the fun=%d", num);
}
void add(int * n);
{
   *n = *n+10;
   Printf("\n The value of num in the Called fun=%d", *n);
}
```

o/p

Enter a number 81
The value of number before calling the fun= 81.
The value of num in the called fun = 91
The value of number after calling the fun = 91

w.a.p to find the biggest among three numbers - Pointers

```
#include<stdio.h>
void main()
{
 int a,b,c;
 int *pa=&a,*pb=&b,*pc=&c;
 printf("Enter three numbers");
 scanf("%d%d%d",pa,pb,pc);
 if((*pa>*pb) && (*pa>*pc))
   printf("%d is largest\n",*pa);
 if((*pb>*pa) && (*pb>*pc))
   printf("%d is largest\n",*pb);
 else
   printf("%d is largest\n",*pc);
}
```

o/p
Enter three numbers
10 70 20
70 is largest

w.a.p to display whether the number is prime or composite.

```
#include<stdio.h>
void main()
{
 int num,*pnum=&num;
 int i,flag=0;
 printf("Enter -1 to Exit\n");
 printf("Enter a number\n");
 scanf("%d",pnum);
 while(*pnum!=-1)
 {
 if(*pnum==1)
   printf("%d is neither prime nor
                composite",*pnum);
 else if(*Pnum==2)
   printf("%d is prime\n",*pnum);
 else
 {
 for(i=2;i<*pnum/2;i++)
 {
  if(*pnum/i==0)
    flag=1;
 }
 if(flag==0)
   printf("%d is prime\n",*pnum);
 else
   printf("%d is composite\n",*pnum);
 }
 printf("Enter any number:");
 scanf("%d",pnum);
}
```

o/p.

```
Enter any number:7
   7 is prime.
Enter any number:4
   4 is composite
Enter any number:-1.
```

w.a.pgm to find the largest of n numbers using arrays and pointers.

```
#include<stdio.h>
void main()
{
int i,n,a[20],large=0,pos=0;
int *pn=&n,*pa=a,*plarge=&large,
        *ppos=&pos;
printf("Enter the no.of.elements");
scanf("%d",pn);
for(i=0;i<*pn;i++)
 scanf("%d",*pa+i);
for(i=0;i<*pn;i++)
{
 if(*(pa+i)>(*plarge))
  {*plarge=*(pa+i);
   *ppos=i;
  }
}
printf("The elements are\n");
for(i=0;i<*pn;i++)
 printf("%d",*(pa+i));
Printf("Largest number=%d",
                *plarge);
printf("position=%d",*Ppos);
}
```

o/p

```
Enter the no.of.elements 5
96 45 92 91 87
Largest number=96
position=0
```

III unit (2 Marks)

1. What is the use of pointers? (Advantages of pointer)

A pointer is a variable is used to hold the address of another variable.

Benefits:

* Pointers are efficient in handling data and associated with array.
* Pointers are used for saving memory space.
* Pointers reduce length and complexity of the program.
* The two dimensional and multidimensional array representation is easy in pointers.

2. What is meant by recursion?

Recursion is the process of calling the same function itself again and again until some condition is satisfied.

Conditions:

* The problem must be analysed and written in recursive form
* The problem must have the stopping condition

3. Differentiate pass by value and pass by address in C

Pass by value	Pass by reference
1. In pass by value, the value of actual arguments are passed to the formal arguments and the operation is done on formal arguments	In pass by reference, the address of actual arguments is passed to the formal arguments.
2. Changes made in formal argument values do not affect the actual argument values.	Since address is passed, the changes made in both argument values are permanent.

4. What is the need for functions?

Functions are self contained block of one or more statements that performs a specific task. It increases the modularity, reusability of a program.

5. How is a pointer variable initialized?

We can initialize a pointer variable while declaring the ptr variable. eg) int a;
int *ptr = &a;

Note that variable a is first declared and then its address is stored in pointer variable ptr.

6. Compare actual parameter and formal parameter.

Actual parameters: The parameters in the calling program or the parameters in the function call are known as actual parameters.

Formal parameter: The parameters in the called program or the parameters in the function header is known as formal parameter.

(eg)
```
int main()
{
  ---
  add(a,b)   <- Actual parameter
  ---
}

void add(int x, int y)
{
  ---
}
// x,y formal parameter
```

7. What are the operators exclusively used with pointers?

Indirection operator (*) – when a pointer is dereferenced, the value stored at that address by the pointer is retrieved.

Address operator (&) - It represents the address of the variable.

8. Distinguish between library function and user defined function:

Library function	user defined function.
The library functions are predefined set of functions.	The user defined functions are defined by the user.
Their task is limited.	Their task is based on user requirement.
The user can use the functions but cannot change or modify them.	The user can modify the function according to the requirement.

9. Define generic pointer (or) void pointer?

The void pointer or the generic pointer is a special type of pointer that can be used to point to variables of any datatype.

syntax `void *ptr;`

10. What is function prototyping? why it is necessary.

Function declaration is also known as function prototype. It identifies a function with its name, a list of arguments that it accepts, and the type of data it returns.

Unit: IV

Structures.

A Structure is a user defined data type. It is a collection of variables under a single name. The variables within a structure are of different datatypes and each has a name that is used to select it from the structure.

Structure Declaration:

A structure is declared using the keyword struct followed by a structure name. All the variables of the structure are declared within the structure.

Syntax.

```
Struct structname
{
  datatype var1;
  datatype var2;
  - - - - -
};
```

The structure declaration, does not allocate any memory, it gives details of the member names. once the structure variable is declared then the memory is allotted for the structure. It can be created as follows.

```
Struct structname varname;
```

eg

```
Struct student
{
  char name[30];
  int regno;
  int marks,total;
  float avg;
};
Struct student S;
```

S → structure variable.

```
Struct student
{
  char name[30];
  int regno;
  int marks,total;
  float avg;
} S;
```

The structure variable can be declared at the time of structure declaration.

```
Struct student
{
  char name[30];
  int regno;
  int marks,total;
  float avg;
} S1,S2;
```

Declares two structure variable S1, S2

Typedef declarations.

The typedef keyword enables the programmer to create a new data type name from an existing datatype. By using typedef, no new data is created, rather an alternate name is given to a known data type.

Syntax

```
typedef existingdatatype newdatatype;
```

typedef statement doesnot occupy any memory, it simply defines a new type.

eg) typedef int num; //now num is new datatype
num a=10;

when we precede a struct name with typedef keyword, then the struct becomes a new type.

eg) typedef struct student
{
int regno;
char name[20];
int marks, total;
};

From this declaration student becomes a new data type. The structure variable can be created as student stu; we have not written struct student stu;

Initialization of structures.

A structure can be initialized in the same way as other data types are initialized. Initializing a structure means assigning some constants to the member of the structure. If the variable is int or float c automatically initialized by 0 and for char and string members are initialized to '\0' by default, in the absence of any initialization done by the user.

syntax

```
struct structname
{
datatype var1;
datatype var2;
-
-
-
}struct var={1,"abc"};
```

(or)

```
struct structname
{
datatype var1;
datatype var2;
};
struct structname var={1,"abc"};
```

eg:1)
```
struct student
{
int regno;
char name[20];
int marks;
}stu{123,"veena",98};
```

eg:2)
```
struct student
{
int regno;
char name[20];
int marks;
};
struct student stu={100,"kumar"};
```

When all the members of a structure are not initialized it is called partial initialization. In partial initialization, first few members of the structure are initialized and those that are uninitialized are assigned default values. In eg:2 marks can be assigned as zero.

Accessing the members of a structure

A structure member variable is generally accessed using a '.' (dot) operator.

syntax: `structvar.membername;`

The '.' (dot) operator is used to select a particular member of the structure. To assign value to the individual data members of the structure variable stu

(eg)
```
stu.regno = 523;
stu.name = "Aravind";
stu.marks = 98;
```

The input values for data members of the structure variable stu, we may write

```
scanf("%d", &stu.regno);
```

To print the values of structure variable stu, we may write

```
printf("%d", stu.regno);
```

//To read and display the information about a student.

```
#include<stdio.h>
Struct student
{
   int regno;
   char name[20];
   float marks;
}stud;
void main()
{
  printf("Enter regno:");
  scanf("%d", &stud.regno);
  printf("Enter name of the student");
  scanf("%s", stud.name);
  printf("Enter marks");
  scanf("%f", &stud.marks);
  printf("Student details \n");
  Printf("Regno = %d", stud.regno);
  printf("Name = %s", stud.name);
  Printf("marks = %f", stud.marks);
}
```

O/P

```
Enter regno 23
Enter name of the student
Kumar
Enter marks 89.0
Student details
Regno = 23
Name = Kumar
marks = 89.0
```

Program to read, display add & subtract two complex numbers.

```
#include<stdio.h>
typedef struct
{
  int real, imag;
}comp;
void main()
{
comp c1, c2, cadd, csub;
int choice;
do
{
 printf("Main menu\n");
 printf("\n 1. Read the complex no");
 printf("\n 2. Display the complex no");
 printf("\n 3. Add two complex no");
 printf("\n 4. Subtract two complex no");
 printf("\n 5. Exit");
 printf("Enter your choice");
 scanf("%d", &choice);
switch(choice)
{
case 1: printf("Enter the real & imaginary parts of num1 \n");
        scanf("%d %d", &c1.real, &c1.imag);
        printf("Enter the real & imag parts of num2");
        scanf("%d %d", &c2.real, &c2.imag);
        break;
case 2: Printf("The first complex number is \n");
        Printf("%d+%di", c1.real, c1.imag);
        Printf("The second complex number is\n");
        Printf("%d+%di", c2.real, c2.imag);
        break;
case 3: printf("complex no. Addition");
        cadd.real = c1.real+c2.real;
        cadd.imag = c1.imag+c2.imag;
        printf("%d+%di", cadd.real, cadd.imag);
        break;
case 4:
        csub.real = c1.real - c2.real;
        csub.imag = c1.imag - c2.imag;
        Printf("%d+%di", csub.real, csub.imag);
        break;
}
}while(choice != 5);
}
```

O/P

```
Main menu
1. Read the complex no.
2. Display the complex no
3. Add two complex no
4. Subtract two complex no
5. Exit
Enter your choice
1
Enter the real & imaginary part of num1
5 2
Enter the real & imag part of num2
3 5
Main menu
1. Read the complex no
2. Display the complex no
3. Add two complex no
4. Subtract two complex no
5. Exit.
Enter your choice 3
8+7i
Main menu.
1. Read the complex no
2. Display the complex no
3. Add two complex no
4. Subtract two complex no
5. Exit
Enter your choice 5.
```

Nested Structure.

. A structure that contains another structure as its member is called as Nested structure. (ii) A structure can be placed within another structure.

Syntax.

```
Struct structname1
{
   data members;
   - - -
};
Struct structname2
 {
   - - - -
   - - -
   data members;
  struct structname1 var1;
   - - -
   - - -
 }var2;
```

// Program To read & display information of a student using a structure within a structure (or) Nested structure.

```
#include<stdio.h>
Struct DOB
{
  int date,month,year;
};
Struct student
{
   int rollno;
   char name[50];
   float avg;
   Struct DOB d;
}stud;
void main()
{
printf("Enter rollno and Name\n");
scanf("%d %s", &stud.rollno,
                 stud.name);
printf("Enter the avg mark\n");
scanf("%f", &stud.avg);
Printf("Enter the date of birth\n");
scanf("%d %d %d", &stud.d.date,
          &stud.d.month,
          &stud.d.year);
Printf("Student details\n");
printf("Name = %s\n., Rollno=%d\n",
          stud.name, stud.rollno);
Printf("Avg = %f\n", stud.avg);
printf("DOB = %d-%d-%d\n",
          stud.d.date, stud.d.month,
          stud.d.year);
}
```

O/P

```
Enter rollno and name:
   B32  Ravi
Enter the avgmark
   85.6
Enter the date of birth.
   23  10  2000
 Student details
Name = Ravi
Rollno = 832
Avg  = 85.6.
DOB  = 23-10-2000.
```

Array of Structure.

If we want to handle more records within one structure, we create athe structure variable as an array. That kind of declaration is called array of structure.

```
eg) Struct student
    {
        int rollno;
        int s1, s2, s3;
        char name[30];
    } stud[5];
```

This create an array of 5 elements (ie) stud[0], stud[1], stud[2], stud[3], stud[4]

Write a program to read and display the information of all the students in the class. Then edit the details of the i^{th} student and redisplay the entire information.

```
#include<stdio.h>
#include<string.h>
#include<conio.h>
struct student
{
  int rollno, fees
  char name[30];
  Char dob[30];
} stud[20];
void main()
{
 int n, i, roll, newroll, nfees;
  char ndob[30], nname[30];
  Printf("Enter the number of students");
 Scanf("%d", &n);
for(i=0; i<n; i++)
{
printf("Enter the rollno and name\n");
Scanf("%d %s", &stud[i].rollno,
                    stud[i].name);
Printf("Enter fees");
Scanf("%.d", &stud[i].fees);
Printf("Enter date of birth\n");
gets(stu[i].dob);
}
printf(" Student details\n");
for(i=0; i<n; i++)
{
Printf("Name = %.s", stud[i].name);
Printf("Rollno = %d", stud[i].rollno);
Printf("Fees = %.d", stud[i].fees);
Printf("DOB = %.s", stud[i].dob);
}
Printf("Enter the rollno whose
        record should be edited");
Scanf("%.d", &roll);
Printf("Enter the newrollno");
Scanf("%.d", &newroll);
Printf("Enter the newname");
Scanf("%.s", nname);
Printf("Enter the new-fees");
Scanf("%.d" &nfees);
Printf("Enter the newdob");
Scanf("%.s", ndob);
Stud[roll].rollno = newroll;
Strcpy(Stud[roll].name, nname);
Stud[roll].fees = nfees;
Strcpy(Stud[roll].dob, ndob);
Printf("updated student details\n");
for(i=0; i<n; i++)
{
Printf("Rollno = %.d", stud[i].rollno);
Printf("Fees = %.d", Stud[i].fees);
Printf("Name = %.s", Stud[i].name);
Printf("DOB = %.s", stud[i].dob);
}
getch();
}
```

output

Enter the number of students
2
Enter the rollno and name
10 meena
Enter Fees 50000
Enter date of birth
18-6-1999
Enter the roll no and name
12 kumar
Enter Fees 75000
Enter date of birth
20-3-2000
Student details
Name = Meena
Rollno = 10
Fees = 50000
DOB = 18-6-1999

Name = kumar
Rollno = 12
Fees = 75000
DOB = 20-3-2000.

Enter the rollno whose record should be edited 12
Enter the new roll no 25
Enter the new name Ashok
Enter the new fees 35000
Enter the new dob 14-12-1999
updated student details.
Rollno = 10
Fees = 50000
Name = Meena
DOB = 18-6-1999.

Rollno = 25
Fees = 35000
Name = Ashok
DOB = 14-12-1999.

Pointers and Structures.

Structures can be created and accessed using pointers. When we have a pointer of structure type we use → to access the structure members.

Syntax

```
struct structname.
{
   - - -
   datamembers
   - - -
}*ptr;
```

next we create one structure variable to assign the address.

```
struct structname s;
ptr = &s;
```

Then we can access the datamember by using pointing-to operator (→)

w.a.c pgm by using arrays of pointers to a structure, to read and display the data of a student.

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
Struct student
{
  int rollno;
  char name[25], dept[10];
} *ptr[10];
void main()
{
  int i,n;
  Printf ("Enter how many std?");
  Scanf("%d", &n);
  for(i=0; i<n; i++)
  {
  Ptr[i] = (Struct student *) malloc
            (Size of (Struct student));

  Printf("Enter the data \n");
  printf("Enter rollno \n");
   Scanf("%d", &ptr[i]->rollno);
  Printf("Enter name \n").
  Scanf("%s", &Ptr[i]->name);
  Printf("Enter dept: \n");
  Scanf("%s", Ptr[i]->dept);
  }
Printf("Details of student\n").
for(i=0; i<n; i++)
{
printf("Rollno=%d", ptr[i]->rollno);
Printf("Name=%s", Ptr[i]->name);
printf("Dept=%s", Ptr[i]->dept);
}
}.
```

o/p
Enter how many students?
o/p. 2.
Enter the data
Enter rollno 12
Enter name. Sudha
Enter dept EEE
Enter the data
Enter rollno 18
Enter name karya.
Enter dept CSE
Details of student
Roll no = 18
Name = Sudha
Dept = EEE
Rollno = 18
Name = karya
Dept = CSE.

Self referential structures.

Self referential structures are those structures that contain a reference to dat of its same type (ie) in addition to other data, a self referential structure contains a pointer to a data that is of the same type as that of the structure.

(eg)

```
Struct node
  {
    int num;
    Struct node *next;
  };
```

Here the structure node will contain two types of data. an integer num and a pointer to a node next.

Self referential structure is the foundation of other data structures.

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
  char *str;
  str=(char *)malloc(6);
  strcpy(str,"Hello");
  printf("memory contains %s\n",str);
  str=(char*)realloc(str,10);
  strcpy(str,"Program");
  printf("After reallocation %s",str);
  free(str);
}
```

o/p.

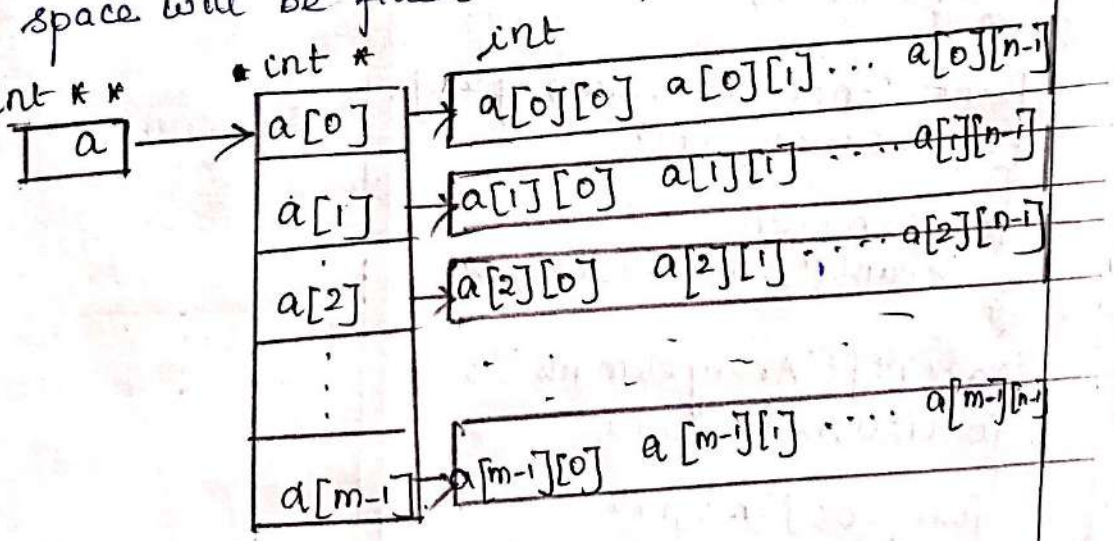
memory contains Hello
After reallocation Program.

Dynamically allocating a 2D Array.

Dynamically allocating a 2D Array array is a pointer-to-pointer to int (ie) int **a.

at the first level as it points to a block of pointers, one for each row. we first allocate space for rows in the array. The space allocated to each row is big enough to hold a pointer to int (or) int *. If we successfully allocate it, then this space will be filled with pointers to columns.

(eg).



Write a C pgm to multiply 2 matrices - dynamic memory allocation.

Read an 2D array using dynamic memory allocation.

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
  int **a,i,j,m,n;
  printf("Enter the number of
        rows & col");
  scanf("%d%d",&m,&n);
  a=(int **)malloc(m*sizeof(int*));
  if(a==NULL)
  {
  printf("memory not allocated");
  exit(-1);
  }
  for(i=0;i<m;i++)
  {
  a[i]=(int *)malloc(n*sizeof(int));
   if(a[i]==NULL)
    {
     printf("Memory not allocation");
     exit(-1);
   }
  }
printf("Enter the values of array");
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
    scanf("%d",&a[i][j]);
}
for printf("Array elements");
 for(i=0;i<m;i++)
 {
  for(j=0;j<n;j++)
     printf("%d",a[i][j]);
 }
for(i=0;i<m;i++)
    free(a[i]);
free(a);
}
```

o/p.

```
Enter the number of rows & col
2 2
Enter the values of array
55 85 28 36.
Array elements
55 85 28 36.
```

Dynamic Memory Allocation.

The process of allocating memory to the variables during execution of the program or at run time is known as dynamic memory allocation.

C provides four library routines to automatically allocate memory at the run time.

*malloc()
*calloc()
*free()
*realloc()

malloc()

The malloc() function reserves a block of memory of specified size and returns a pointer of type void.

It allocates memory and returns a pointer to the first byte of allocated space.

syntax:

```
ptr = (cast-type*)malloc(bytesize);
```

cast-type → is the datatype.

bytesize → is the size of the memory to be allocated.

(eg) int *a;
a = (int *)malloc(10 * sizeof(int));

This statement is used to dynamically allocate memory equivalent to 10 times size of of an int (ie) 20 bytes is allocated.

Malloc() is declared in <stdlib.h>. so we include this header file in any program that calls malloc

(eg) // Program to read and display values of an integer array.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
  int i,n;
  int *arr;
  Printf("Enter no.of.elements\n");
  Scanf("%d",&n);
  arr=(int *)malloc(n*sizeof(int));
  for(i=0;i<n;i++)
    Scanf("%d",&arr[i]);
  Printf("Array elements\n");
  for(i=0;i<n;i++)
    Printf("%d",arr[i]);
}
```

o/p

```
Enter no.of.elements
5
10 15 18 12 55
Array elements
10 15 18 12 55.
```

Calloc()

The function calloc() is another function that reserves memory at the run time. calloc() stands for contiguous memory allocation and is primarily used to allocate memory for arrays.

Syntax

```
ptr = (cast-type*)calloc(n, elesize)
```

n → no. of blocks

elesize → size in bytes for each elements.

calloc() allocates multiple block of memory with same size and initialized them with zero.

The only difference between malloc() and calloc() is that when we use calloc(), all bytes are initialized to zero.

eg)

```
#include<stdio.h>
#include<stdlib.h>
void main()
{
  int i, n;
  int *arr; scanf("%d", &n);
  arr = (int*)calloc(n, sizeof(int));
  printf("Enter the elements \n");
  for(i=0; i<n; i++)
    scanf("%d", &arr[i]);
  printf("Array elements \n");
  for(i=0; i<n; i++)
    printf("%d", arr[i]);
  free(arr);
}
```

o/p

```
4
Enter the elements
10 15 20 70
Array elements
10 15 20 70.
```

free()

The free() function is used to release the previously allocated memory space using malloc() (or) calloc().

When we dynamically allocate memory then it is our responsibility to release the space when it is not required.

Syntax

```
free(ptr);
```

realloc()

The function realloc() is used to change the memory size already allocated by calloc() and malloc(). This process is called reallocation of memory.

Syntax

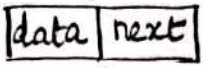
```
ptrvar = realloc(ptrvar, newsize)
```

newsize → is the newsize that is going to be altered in previously allocated memory.

realloc can allocate more bytes without losing our data.

Singly linked List.

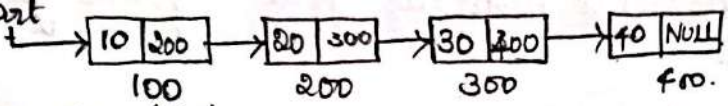
A linked list is a linear collection of data elements. These data elements are called nodes. Each node contains datafield and a pointer field



node

A singly linked list is the simplest type of linked list in which every node contains some data and a pointer to the next node of the same type.

(eg)



Operations on list.

* Insertion
* Deletion
* Searching.
* Traversing

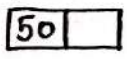
Inserting a newnode in a Linked list.

Insert a newnode to an already existing linked list. It will take four cases.

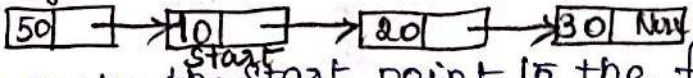
* The newnode is inserted at the beginning of a linked list
* The newnode is inserted at the end of a linked list
* The newnode is inserted after a given node
* The newnode is inserted before a given node.

newnode is inserted at the beginning of a linked list

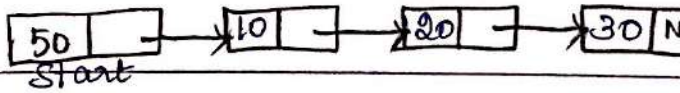
Suppose we want to add a newnode with data 50 as the first node of the list, then the following changes will be done in the linked list.

(i) Allocate memory for the new node and initialize its data part to 50 

(ii) Add the newnode as the first node of the list. next part of the new node contains the address of start.



(iii) Now make the start point to the first node of the list



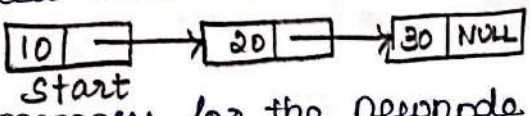
Algorithm to insert a newnode at the beginning of a Linked list.

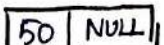
Step1 : newnode = malloc (sizeof(struct node))
Step2 : newnode → data = num;
Step3 : newnode → next = Start;
Step4 : ReStart = newnode

Suppose the newnode is added as the first node of the list, then it will be known as the start node.
(ie) Start = newnode.

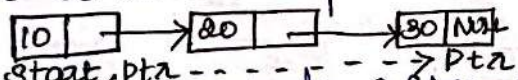
Add a newnode at the end of the list.

Suppose we want to add a newnode with data 50 as the last node of the list. Then the following changes will be done in the linked list.



• Allocate memory for the newnode and initializes its data part as 50 and next part to NULL 

• Take a pointer variable ptr which points to start. move ptr so that it points to the last node of the list.



• Add the new node after the node pointed by ptr. This is done by storing the address of the newnode in the next part of ptr

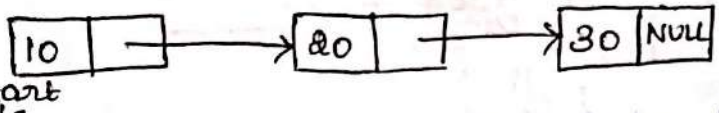
Algorithm to insert a newnode at the end of the list.

Step1 : newnode = malloc (sizeof (struct node)).
Step2 : newnode → data = num;
Step3 : newnode → next = NULL;
Step4 : Set ptr = start.
Step5 : while (ptr → next != NULL)
ptr = ptr → next
Step6 : Set ptr → next = newnode.

Add a new node after a given node.

* Allocate memory for the newnode and initialize its data part as 50 

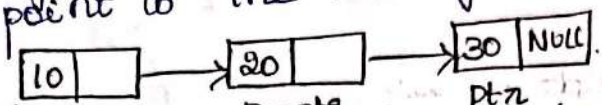
* Take two pointer variables ptr and preptr and initialize them with start. so that start, ptr and preptr all point to the first node of the list.



Start
Ptr
Preptr.

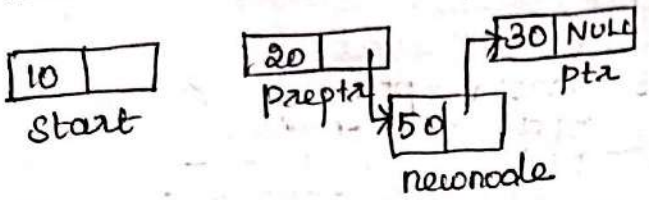
num (01) = Value

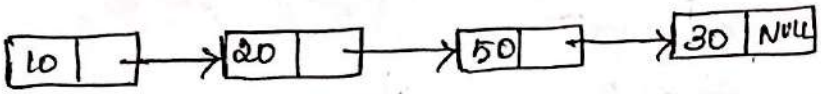
Move ptr and preptr until the data part of preptr = Value of the node after which insertion has to be done. preptr will always point to the node just before ptr.



Start
Preptr
Ptr

Add the newnode in between the nodes pointed by preptr and ptr.





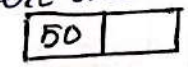
Start.

Algorithm to insert a newnode after a node that has Value num

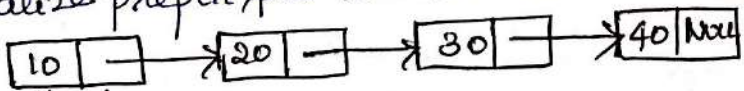
Step1 : newnode = malloc (sizeof (struct node))
Step2 : Set newnode → data = num
Step 3 : Set ptr = start, preptr = start.
Step 4 : while preptr → data != val
Set preptr = ptr
Set ptr = ptr → next
Step 5 : preptr → next = newnode
Step 6 : newnode → next = ptr

insert a newnode before a given node.

* Allocate memory for the newnode and initialize its data part to 50

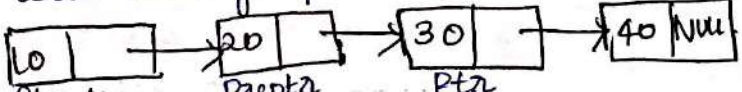


* Initialize preptr, ptr to the start node



Start
Ptr
Preptr.

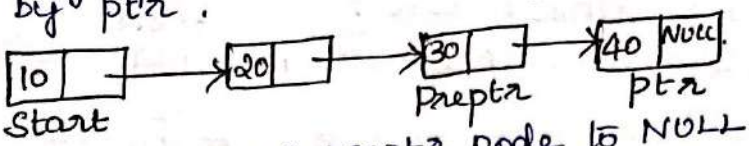
* Move ptr and preptr until the data part of ptr = Value of the node before which insertion has to be done. preptr will always point to the node just before ptr.



Start
Preptr
Ptr

* Insert the newnode in between the nodes pointed by preptr and ptr.

* Move ptr and preptr such that next part of ptr=NULL preptr always points to the node just before the node pointed by ptr.



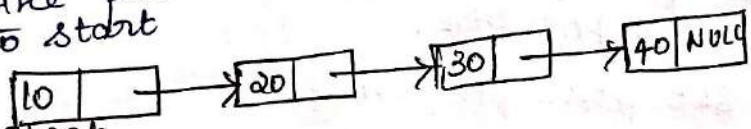
* Set the next part of preptr node to NULL

Algorithm check Start = NULL otherwise do the steps.

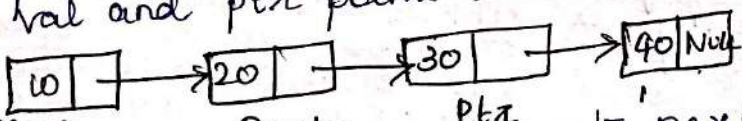
Step1 : set ptr = Start
Step2 : while ptr → next != NULL
set preptr = ptr
set ptr = ptr → next
Step3 : Set preptr → next = NULL
Step4 : free ptr.

Delete the node after a given node from a linked list.

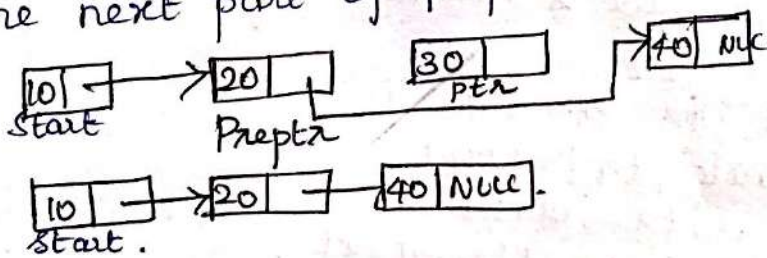
* Take pointer variables ptr and preptr which initially Point to start



move preptr and ptr such that preptr points to the node containing val and ptr points to the succeeding node.



Set the next part of preptr to next part of ptr.



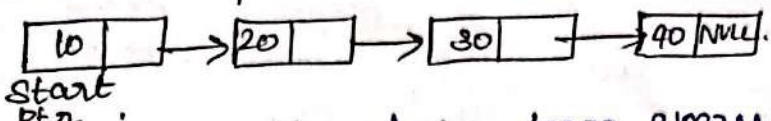
Algorithm.

Step1: if Start = NULL then no node in the list
Step2: Otherwise Set Ptr = Start, preptr = ptr.
Step3 : while preptr → data != val
set preptr = ptr
set ptr = Ptr → next.
Step4 : Set temp = Ptr
Step5 : Set preptr → next = Ptr → next
Step6 : free temp

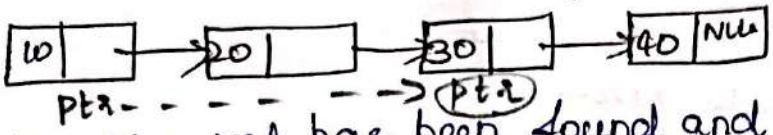
Searching for a value in a linked list.

Searching a linked list means to find a particular element in the linked list.

* Initialize a pointer variable ptr with start.



* Then compare the data from every node with val for which the search is being made.



* finally the val has been found and the address of that node is stored in pos and the control jumps to the last statement otherwise pos set to NULL.

Algorithm.

Step1: Set ptr = start
Step2: while ptr != NULL.
Step3: if val = ptr→data
set pos = ptr.
else
set ptr = ptr→next

Step 5: set pos = NULL.

Traversing a Singly linked list.

Traversing a linked list means accessing the nodes of the list in order to perform some operations on them. A linked list always contains a pointer variable Start which stores the address of the first node of the list. The end of the list is marked by NULL or -1

Algorithm.

step1: set ptr = start.
step2: while ptr != NULL.
Ptr→data.
set ptr = ptr→next.

Algorithm to print the information stored in each node of a linked list.

step1: set ptr = start.
step2: while ptr != NULL.
Print ptr→data.
set ptr = ptr→next

Algorithm to print the number of nodes.

```
Step1 : Initialize count = 0
Step2 : Set ptr = start
Step3 : While ptr != NULL
            count = count + 1
            ptr = ptr -> next.
Step4 : Print count.
```

2 Marks.

1. What is a structure?

It is a user defined datatype. It contains one or more data items of different (heterogeneous) datatype in which the individual elements can differ in type. The individual structure elements are called members.

2. Write the rules for declaring a structure?
   * A structure must end with a semicolon.
   * struct keyword is mandatory.
   * Each structure member must be terminated.
   * The structure variable must be accessed with (.) dot operator.

3. Differentiate array and structure.

Array	Structure.
1. An array is a collection of similar data items	A structure is a collection of variables of different data types.
2. An array is a derived datatype	It is a user defined datatype.
3. The individual data members of an array can be initialized	The individual data members of the structure cannot be initialized.
4. Array elements can be accessed by indexing the array name.	Structure members can be accessed using dot/arrow operator.

4. Define dynamic memory allocation functions and list its types?

The process of allocating memory during program execution is called dynamic memory allocation. C language offers 4 dynamic memory allocation functions. They are
   * malloc()
   * calloc()
   * realloc()
   * free()

5. Define singly linked list.

It is a linear data structure. It is a type of list. In SLL each node in the list stores the contents of the node and a pointer or reference to the next node in the list.

```
struct node
{
    int data;
    struct node *next;
};
```

6. what are self referential structures?

A structure consisting of atleast a pointer member pointing to the same structure is known as self referential structure.

Eg:

```
struct book
{
    int bookid
    struct book *ptr;
};
```

7. Explain typedef with syntax

typedef is a keyword used in clanguage to assign alternative names to existing types.

Syntax

```
typedef existingname aliasname;
typedef int integer;
```

8. How structure elements can be accessed?

1. Direct member access/dot operator.

   strctname.struct membername.

2. Indirect member access operator/arrowoperator.

   Pointer to struct → structmembername.

Singly Linked List

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
struct node
{
int data;
struct node*next;
};
struct node*start=NULL;
struct node*create_11(struct node*);
struct node*display(struct node*);
struct node*insert_beg(struct node*);
struct node*insert_end(struct node*);
struct node*insert_before(struct node*);
struct node*insert_after(struct node*);
struct node*delete_beg(struct node*);
struct node*delete_end(struct node*);
struct node*delete_node(struct node*);
struct node*delete_after(struct node*);
struct node*delete_list(struct node*);
struct node*sort_list(struct node*);
int main()
{
int option;
clrscr();
do
{
printf("\n\n****MAIN MENU****");
printf("\n 1.Create a list");
printf("\n 2.Display the list");
printf("\n 3.Add a node at the beginning");
printf("\n 4.Add a node at the end");
printf("\n 5.Add a node before a given mode");
printf("\n 6.Add a node after a given mode");
printf("\n 7.Delete from the beginning");
printf("\n 8.Delete from the end");
printf("\n 9.Delete a given mode");
printf("\n 10.Delete after a given mode");
printf("\n 11.Delete the entire list");
printf("\n 12.Sort the list");
printf("\n 13.EXIT");
printf("\n*************************");
printf("\n\nEnter your option");
scanf("%d",&option);
switch(option)
{
case 1:
    start=create_11(start);
    printf("\nLINKED LIST CREATED");
    break;
case 2:
    start=display(start);
    break;
case 3:
    start=insert_beg(start);
    break;
case 4:
    start=insert_end(start);
    break;
case 5:
    start=insert_before(start);
    break;
case 6:
    start=insert_after(start);
    break;
case 7:
    start=delete_beg(start);
    break;
case 8:
    start=delete_end(start);
    break;
case 9:
    start=delete_node(start);
    break;
case 10:
    start=delete_after(start);
    break;
case 11:
    start=delete_list(start);
    printf("\nLINKED LAST DELETED");
    break;
case 12:
    start=sort_list(start);
    break;
}
}
while(option!=13);
getch();
return 0;
}
struct node*create_11(struct node *start)
{
struct node *new_mode,*ptr;
int num;
printf("\nEnter -1 to end");
printf("\nEnter the data: ");
scanf("%d",&num);
while(num!=-1)
{
new_node=(struct node*)
malloc(sizeof(struct node));
new_node->data=num;
if(start==NULL)
{
```

```
new_node->next=NULL;
start=new_node;
}
else
{
ptr=start;
while(ptr->next!=NULL)
ptr=ptr->next;
new_node->next=NULL;
}
printf("\nEnter the data:");
scanf("%d",&num);
}
return start;
}
struct node*display(struct node*start)
{
struct node*ptr;
ptr=start;
while(ptr!=NULL)
{
printf("\t%d",ptr->data);
ptr=ptr->next;
}
return start;
}
struct node*insert_beg(struct node*start)
{
struct node*new_node;
int num;
printf("\nEnter the data");
scanf("%d",&num);
new_node=(struct node*)
malloc(sizeof(struct node));
new_node->data=num;
new_node->next=start;
start=new_node;
return start;
}
struct node*insert_end(struct node*start)
{
struct node*ptr, *new_node;
int num;
printf("\nEnter the data:");
scanf("%d",&num);
new_node=(struct node*)
malloc(sizeof(struct node));
new_node->data=num;
new_node->next=NULL;
ptr=start;
while(ptr->next!=NULL)
ptr=ptr->next;
ptr->next=new_node;
return start;
}
struct node*insert_before(struct node*start)
{
struct node*new_node,*ptr,*preptr;
int num,val;
printf("\n Enter the data");
scanf("%d",&num);
printf("\nEnter the value before which the data
has to be inserted");
scanf("%d",&val);
new_node=(struct node*)
malloc(sizeof(struct node));
new_node->data=num;
ptr=start;
while(ptr->data!=val)
{
preptr=ptr;
ptr=ptr->next;
}
preptr->next=new_node;
new_node->next=ptr;
return start;
}
struct node*insert_after(struct node*start)
{
struct node*new_node,*ptr,*preptr;
int num,val;
printf("\nEnter the data");
scanf("%d",&num);
printf("\nEnter the value after which the data has
to be inserted");
scanf("%d",&val);
new_node=(struct node*)
malloc(sizeof(struct node));
new_node->data=num;
ptr=start;
preptr=ptr;
while(preptr->data!=val)
{
preptr=ptr;
ptr=ptr->next;
}
preptr->next=new_node;
new_node->next=ptr;
return start;
}
struct node*delete_beg(struct node*start)
{
struct node*ptr;
ptr=start;
start=start->next;
free(ptr);
```

```
return start;
}
struct node*delete_end(struct node*start)
{
struct node*ptr,*preptr;
ptr=start;
while(ptr->next!=NULL)
{
preptr=ptr;
ptr=ptr->next;
}
preptr->next=NULL;
free(ptr);
return start;
}
struct node*delete_node(struct node*start)
{
struct node*ptr,*preptr;
int val;
printf("\nEnter the value of the node which has to
be deleted");
scanf("%d",&val);
ptr=start;
if(ptr->data==val)
{
start=delete_beg(start);
return start;
}
else
{
while(ptr->data!=val)
{
preptr=ptr;
ptr=ptr->next;
}
preptr->next=ptr->next;
free(ptr);
return start;
}
}
struct node*delete_after(struct node*start)
{
struct node*ptr,*preptr;

int val;
printf("\nEnter the value after which the node has
to be deleted");
scanf("%d",&val);
ptr=start;
while(preptr->data!=val)
{
preptr=ptr;
ptr=ptr->next;
}
preptr->next=ptr->next;
free(ptr);
return start;
}
struct node*delete_list(struct node*start)
{
struct node*ptr;
ptr=start;
while(ptr->next!=NULL)
{
printf("\n%d is to be deleted next",ptr->data);
start=delete_beg(ptr);
ptr=ptr->next;
}
return start;
}
struct node*sort_list(struct node*start)
{
struct node *ptr1,*ptr2;
int temp;
ptr1=start;
while(ptr1->next!=NULL)
{
ptr2=ptr1->next;
while(ptr2!=NULL)
{
if(ptr1->data>ptr2->data)
{
temp=ptr1->data;
ptr1->data=ptr2->data;
ptr2->data=temp;
}
ptr2=ptr2->next;
}
ptr1=ptr1->next;
}
return start;
}
```

Student Performance using Structure

```
#include<stdio.h>
struct student
{
int roll,marks[5];
char name[25];
}s;
void main()
{
char grade;
int i,tot=0;
float avg;
printf("Enter rollno and name");
```

```
scanf("%d%s",&s.roll,s.name);
printf("ent4er marks");
for(i=0;i<5;i++)
{
scanf("%d",&s.marks[i]);
tot+=s.marks[i];
}
avg=tot/5.0;
printf("Tot=%d",tot);
if(avg>=80.0)
printf("A+ grade");
else if(avg>=70.0)
printf("A grade");
else if(avg>=60.0)
printf("B+ grade");
else if(avg>=50.0)
printf("B grade")  ;
else
printf("Fail");
}
```

Complex number-Manipulation

```
#include <stdio.h>
#include <conio.h>
typedef struct
{
   float real, imag;
}complex;
complex a, b, c;
   complex read(void);
   void display(complex);
   complex add(complex, complex);
   complex sub(complex, complex);
   complex mul(complex, complex);
   complex div(complex, complex);
void main ()
{
   clrscr();
   printf("Enter the 1st complex number\n ");
   a = read();
   display(a);
   printf("Enter  the 2nd complex
 number\n");
   b = read();
   display (b);
   printf("Addition\n ");
   c = add(a, b);
   display (c);
   printf("Substraction\n ");
   c = sub(a, b);
   display (c);
   printf("Multiplication\n");
   c = mul(a, b);
  display (c);
   printf("Division\n");
   c = div(a, b);
  display (c);
  getch();
}
complex read(void)
{
   complex t;
   printf("Enter the real part\n");
   scanf("%f", &t.real);
   printf("Enter the imaginary part\n");
   scanf("%f", &t.imag);
   return t;
}
void display(complex a)
{
   printf("Complex number  is\n");
   printf(" %.1f + i %.1f", a.real, a.imag);
   printf("\n");
}
 complex add(complex p,complex q)
{
   complex t;
   t.real = (p.real + q.real);
   t.imag = (p.imag + q.imag);
   return t;
}
 complex sub(complex p,complex q)
{
    complex t;
      t.real = (p.real - q.real);
   t.imag = (p.imag - q.imag);
   return t;
}
complex mul(complex p,complex q)
{
   complex t;
   t.real=(p.real * q.real) - (p.imag * q.imag);
   t.imag=(p.real * q.imag) + (p.imag * q.real);
   return t;
}
 complex div(complex p, complex q)
{
   complex t;
     t.real = ((p.real * q.real) + (p.imag *
q.imag)) / ((q.real * q.real) + (q.imag * q.imag));
 t.imag = ((p.imag * q.real) - (p.real * q.imag)) /
((q.real * q.real) + (q.imag * q.imag));
   return(t);
}
```

Union

Union can be defined as a user-defined data type which is a collection of different variables of different data types in the same memory location. The union can also be defined as many members, but only one member can contain a value at a particular point in time.

Union is a user-defined data type, but unlike structures, they share the same memory location.

Syntax

```
union union name
{
        member definition;
        member definition;
        ...
        member definition;
} [one or more union variables];
```

Example

```
union Data
{
        int i;
        float f;
        char str[20];
} data;
```

Now, a variable of Data type can store an integer, a floating-point number, or a string of characters. It means a single variable, i.e., same memory location, can be used to store multiple types of data. You can use any built-in or user defined data types inside a union based on your requirement.

Accessing Union Members

To access any member of a union, we use the member access operator (.). The member access operator is coded as a period between the union variable name and the union member that we wish to access

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

union Data {
  int i;
  float f;
  char str[20];
};

int main( ) {

  union Data data;
```

```
   data.i = 10;
   data.f = 220.5;
   strcpy( data.str, "C Programming");

   printf( "data.i : %d\n", data.i);
   printf( "data.f : %f\n", data.f);
   printf( "data.str : %s\n", data.str);

   return 0;
}
```

Output :

```
data.i : 10
data.f : 220.500000
data.str : C Programming
```

Accessing members of union using pointers

We can access the members of the union through pointers by using the (->) arrow operator.

```
#include <stdio.h>
union abc
{
   int a;
   char b;
};
int main()
{
 union abc *ptr; // pointer variable declaration
   union abc var;
   var.a= 90;
   ptr = &var;
   printf("The value of a is : %d", ptr->a);
   return 0;
}
```

Output : 90

Storage classes

Variables in C have not only datatype but also storage class that provides information about their location and visibility. The storage class decides the portion of the program within which the variables are recognized

C language supports 4 types of storage classes
* auto * static * extern
* register

1. Automatic Variables

The variables without any storage class specification are considered as automatic variable. They are called as automatic, because their memory space is automatically allocated as the variable is declared.

Syntax

```
storageclastype datatype varname;
```

eg) auto int x; // same as int x

2. Static Variable.

The static variable may be an internal or external type, depending upon where it is declared.

These variables have a property of preserving their value even after they are out of their scope.

Syntax

```
static datatype varname;
```

eg)
```
static int a;
#include <stdio.h>
int func()
{
  static int count=0;
  count++;
  return count;
}
int main()
{
  printf("%d", func());
  printf("%d", func());
  return 0;
}
```

o/p
1 2

3. External Variables

The external variables are declared out of the main() function. The availability of these variables are through out the program and that is both in main program and inside the user defined fun.

Syntax:
```
extern datatype varname;
```

eg)
```
#include<stdio.h>
int a=50;
void main()
{
  fun();
  printf("a=%d", a);
}
void fun()
{
  a=25;
  printf("a=%d", a);
}
```

o/p
a=25
a=50.

4. Register Variable

Registers are special storage areas within a computer's CPU. The register access is faster than the memory access. CPU registers are limited in numbers. Hence we cannot declare more variables with register variable.

Syntax
```
register datatype varname;
```

eg)
```
#include<stdio.h>
void main()
{
  register int i=0;
  for(i=0; i<2; i++)
  {
    printf("%d", i);
  }
}
```

o/p
0
1

Unit V
File Processing.

File.

File is a collection of data stored on a secondary storage device.

Streams in C.

Stream is a logical interface to the devices that are connected to the computer.

The three standard streams in C language are as follows..

* Standard input (stdin)
* Standard output (stdout)
* Standard error (stderr)

Standard input is the stream from which the program receives its data.

Standard output is the stream where a program writes its output data.

Standard error is basically an output stream used by programs to report error messages.

Types of files

In C, the types of files used can be broadly classified into two categories

* Text files
* Binary files.

Text files.

A text file is a stream of characters that can be sequentially processed by a computer in a forward direction. A textfile is usually opened for only one kind of operation (reading, writing (or) appending) at any given time. In a text file, each line contains zero or more characters and ends with one or more characters that specify the end of line.

Binary files.

A binary file may contain any type of data, encoded in binary form for computer storage and processing purposes. While text files can be processed sequentially, binary files on the other hand, can be either processed sequentially (or) randomly depending on the needs of the Application.

Files in C

To use files in C, we must follow the steps.

* Declare a file pointer variable
* Open the file
* Process the file.
* Close the file.

Declaring a file pointer variable.

In order to access a file we must specify the name of file that has to be used. This is accomplished by using a file pointer variable that points to a structure FILE.

Syntax. FILE *fpvar; (eg) FILE *fp.

Opening a file.

A file must first be opened before data can be read from it or written to it. In order to open a file and associate it with a stream, the fopen() function is used.

Syntax. FILE *fopen(const char * filename, const char *mode);

Every file on the disk has a name known as the filename. In C, the fopen() may contain the path information instead of specifying the filename. The path gives information about the location of the file on the disk. If a filename is specified without a path, it is assumed that the file is located in the current working directory.

filemode conveys to C the type of processing that will be done with the file.

File modes.

Mode	Description.
r	Open a text file for reading. If the file does not exist then an error will be reported.
w	open a text file for writing. If the file does not exist then it is created, if exists means contents would be deleted
a	Append to a text file. If the file does not exist it is created.
rb	open a binary file for reading
wb	open a binary file for writing.
ab	Append to a binary file.
r+	open a text file for both reading and writing. When we specify r+ indicate that we want to read the file befor to write it.
w+	open a text file for both reading & writing
a+	opens a text file for both reading & writing
r+b / rb+	open a binaray file for read / write
w+b / wb+	create a binary file for read / write
a+b / ab+	Append a binary file for read / write

(eg)
```
File *fp
fp = fopen("student.txt","r");
if (fp==NULL)
{
 printf("The file could not be
                opened");
 exit(1);
}
```

```
File *fp;
char fname[15];
gets(fname);
fp = fopen(fname,"r");
if (fp==NULL)
{ printf("File not to be
                opened");
 exit(-1);
}
```

closing a file.

The fclose() function is used to disconnects a file pointer from a file. The fclose() function not only closes the file but also flushes all the buffers that are maintained for that file.

syntax: `fclose(fp);`

Read data from files.

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

* fscanf	* fgetsc()
* fgetc()	* fread()

fscanf:

It is used to read formatted data from the stream.

syntax: `fscanf(FILE *stream, const char *format, ...);`

*Stream → filepointer

*format specifiers in a ~~o~~ starts with %. sign.

The fscanf is similar to scanf function except that the first argument of fscanf specifies a stream from which to read, whereas scanf can only read from standard input.

(eg)
```
#include<stdio.h>
void main()
{
 FILE *fp;
 char name[50];
 int rollno;
 fp = fopen("Student.Dat","r");
 if (fp==NULL)
 {printf("The file couldnot be opened");
  exit(1);
 }
 Printf("Enter the name &rollno");
 fscanf(stdin,"%s%d",name,
                &rollno);
 Printf("name = %s\t Rollno=%d",
                name, rollno);
 fclose(fp);
}
```

o/p

Enter the name &rollno
Kavin 45

fgets()

The function fgets() is used to get a string from a file.

Syntax: `char *fgets(char *str, int size, FILE *stream);`

The fgets() function reads at most one less than the number of characters specified by size, from the given stream and stores them in the string str. The fgets() terminates as soon as it encounters either a newline character, EOF or any other error.

```
#include <stdio.h>
void main()
{
  FILE *fp;
  char str[80];
  fp=fopen("sample.txt","r");
  if(fp==NULL)
  { printf("The file could not be opened");
    exit(1);
  }
while(fgets(str,80,fp)!=NULL)
  printf("\n%s",str);
printf("close the file");
fclose(fp);
}
```

o/p

```
abcdef.....
close the file.
```

fgetc()

The fgetc() function returns the next character from stream.

Syntax: `int fgetc(FILE *stream);`

fgetc() returns the character read as an int or return EOF to indicate an error or end of file.

fgetc() reads a single character from the current position of a file. After reading the chacter, the function increments the associated file pointer to point to the next character.

(eg)

```
#include<stdio.h>
void main()
{
  FILE *fp;
  char str[80];
  int i,ch;
  fp=fopen("add.c","r");
  if(fp==NULL)
  { printf("The file could not be opened");
    exit(1);
  }
  ch=fgetc(fp);
  for(i=0;(i<79)&&(feof(fp)==0);i++)
  {
    str[i]=(char)ch;
    ch=fgetc(fp);
  }
  str[i]='\0';
  printf("%s",str);
  fclose(fp);
}
```

o/p

```
#include<stdio.h>
- - -
displays either first 79 char(s) or
lesser char the file contains.
```

fread()

The fread() function is used to read data from a file.

Syntax: `int fread(void *str, size_t size, size_t num, FILE *stream);`

The function fread() reads num(10) number of objects and places them into the array pointed to by str. After successful completion, fread() returns the number of bytes successfully read. The number of objects will be less than num if an read error or end of file is encountered.

```
#include<stdio.h>
void main()
{
 FILE *fp;
 char str[15];
 fp=fopen("sample.txt","r+");
 if(fp==NULL)
 {
  printf("The file could not be opened");
  exit(1);
 }
 fread(str,1,10,fp);
 str[10]='\0';
 printf("First 9 characters of the ffleare:%s",str);
 fclose(fp);
}
```

o/p
First 9 character of the file
hello how

Writing Data to files.

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

* fprintf()
* fputs()
* fputc()
* fwrite()

fprintf()

The fprintf() is used to write formatted output to stream.

Syntax: `fprintf(FILE *stream, const char *format, ....);`

The function writes data that is formatted as specified by the format argument to the specified stream. The fprintf() each optionally contain format tags, that are replaced by the values specified in subsequent additional arguments and are formatted as requested.

```
(eg)#include<stdio.h>
void main()
{
 FILE *fp;
 int i;
 char name[20];
 float salary;
 fp=fopen("sample.txt","w");
 if(fp==NULL)
 {
  printf("The file could not be opened");
  exit(1);
 }
 for(i=0;i<2;i++)
 {
  puts("Enter name");
  gets(name);
  fflush(stdin);
  puts("Enter sal");
  scanf("%f",&salary);
  fprintf(fp,name:%s\t Salary:%f",name,salary);
 }fclose(fp);
}
```

Enter name: suba Enter name: Rama
Enter salary: 10150.0 Enter sala: 15000

fputs()

The fputs() is used to write a line to a file.

syntax.

```
int fputs(const char *str,
          FILE *stream);
```

The fputs() writes the string pointed to by str to the stream pointed to by stream.

```
#include<stdio.h>
void main()
{
 FILE *fp;
 char str[100];
 fp=fopen("samp.txt","w");
 if(fp==NULL)
 {printf("The file could not be opened\n");
  exit(1);
 }
 printf("Enter data");
 gets(str);
 fflush(stdin);
 fputs(str,fp);
 fclose(fp);
}
```

o/p.

Enter data
File programs

fputc()

The fputc() is just the opposite of fgetc(). It is used to write a character to the stream.

syntax.

```
int fputc(int c, File *stream);
```

The fputc() function will write the byte specified by c to the output stream pointed to by stream.

```
#include<stdio.h>
void main()
{ FILE *fp;
  char str[100];
  int i;
  fp=fopen("samp.txt","w");
  if(fp==NULL)
  {printf("The file could not be opened");
   exit(1);
  }
  printf("Enter data");
  gets(str);
  for(i=0;i<str[i];i++)
   fputc(str[i],fp);
  fclose fp()
}
```

o/p

Enter data
file program.

fwrite()

The fwrite() is used to write data to a file.

syntax.

```
int fwrite(const void *str,size_t size,
           size_t ocent,FILE *stream)
```

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

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
void main()
{ FILE *fp; size_t count;
  char str[]="Good morning";
  fp=fopen("samp.txt","wb");
  if(fp==NULL)
  {printf("file could not be opened");
   exit(1);
  }
 count=fwrite(str,1,strlen(str),fp);
 printf("%d bytes were written",count);
 fclose(fp);
}
```

o/p

13 bytes were written.

Detecting the end of file.

When reading or writing data from files, we often do not know exactly how long the file is. In C there are two ways to detect EOF while reading the file in text mode, character by character the programmer can compare the character that has been read with the EOF which is a symbolic constant defined in stdio.h (with a value of -1.

eg)
```
while(1)
 {
   c=fgetc(fp);
   if(c==EOF)
      break;
   Printf("%c",c);
 }
```

*The otherway is to used the standard library function feof() which is defined in stdio.h.

eg)
```
while(!feof(fp))
 {
   fgets(str,79,fp);
 }
```

① W.a.pgm to read a file character by character and display it on the screen.

```
#include<stdio.h>
void main()
{ FILE *fp; int ch; char fname[20];
  printf("Enter the file name \n");
  gets(fname);
  fp=fopen(fname,"r");
  if(fp==NULL)
  { printf("Error opening the file\n");
    exit(1);
  }
  ch=fgetc(fp);
  while(ch!=EOF)
  { putchar(ch);
    ch=fgetc(fp);
  }
  fclose(fp);
}
```

O/P

```
Enter the file name
File1.txt
To read characters
```

② Pgm to count the no. of characters and number of lines in the file.

```
#include<stdio.h>
void main()
{
  FILE *fp;
  int ch, nc=0, nl=1;
  char fname[20];
  printf("Enter the filename:"); gets(fname);
  fp=fopen(fname,"r");
  if(fp==NULL)
  { printf("Error opening the file\n");
    exit(1);
  }
  ch=fgetc(fp);
  while(ch!=EOF)
  { if(ch=='\n')
      nl++;
    nc++;
    ch=fgetc(fp);
  }
  if(nc>0)
    printf("The file contains %d characters and %d lines", nc, nl);
  else
    printf("File is empty");
  fclose(fp)
}
```

O/P

```
Enter the filename
samp.txt
The file contains
15 characters and
3 lines
```

③ W.a.pgm to copy one file into another

```
#include<stdio.h>
void main()
{
  FILE *fp1, *fp2;
  int ch;
  char fname1[20], fname2[20];
  printf("Enter the first filename and 2nd name");
  gets(fname1);
  gets(fname2);
  if((fp1=fopen(fname1,"r"))==0)
  { printf("Error opening the file");
    exit(1);
  }
  if((fp2=fopen(fname2,"w"))==0)
  { printf("Error opening the file");
    exit(1);
  }
  ch=fgetc(fp1);
  while(ch!=EOF)
  { putc(ch,fp2);
    ch=fgetc(fp1);
  }
  printf("File copied");
  fclose(fp1); fclose(fp2);
}
```

O/P

```
Enter the first filename and 2nd name
source.txt
dest.txt
file copied
```

④ Pgm to compare two files.

```
#include<stdio.h>
void main()
{
  FILE *fp1, *fp2; int ch1, ch2;
  char fname1[20], fname2[20];
  printf("Enter the Filenames");
  gets(fname1); gets(fname2);
  if((fp1=fopen(fname1,"r"))==0)
  { printf("Error"); exit(1);
  }
  if((fp2=fopen(fname,"r"))==0)
  { printf("Error"); exit(1);
  }
  ch1=fgetc(fp1);
  ch2=fgetc(fp2);
  while(ch1!=EOF && ch2!=EOF && ch1==ch2)
  { ch1=fgetc(fp1);
    ch2=fgetc(fp2);
  }
  if(ch1==ch2)
    printf("Files are identical");
  fclose(fp1);
  fclose(fp2);
}
```

O/P

```
Enter the filenames
source.txt
dest.txt
Files are identical
```

Types of File Processing

There are two types of files.

Sequential access file: In this type of file, the data are kept sequentially. To read last record of the file, it is expected to read all records before that particular record. It takes more time for accessing the records.

Random access file: In this type, the data can be read and modified randomly. If it is desired to read the last record of a file, directly the same record can be read. Due to random access of data, it takes less access time as compared to the sequential file.

Sequential access file.

① /* Program to find avg of numbers stored in a sequential file */

```
#include<stdio.h>
void main()
{
  FILE *fp;
  int n=0;
  float num,sum,avg;
  sum=0.0
  fp=fopen("data.txt","r");
  while(!feof(fp))
  {
   fscanf(fp,"%f",&num);
    sum = sum+num;
    n=n+1;
  }
  avg=sum/n;
  fclose(fp);
  Print("Avg=%f",avg);
}
```

O/P

Input : data.txt

1 2 3 4 5

avg = 3.000000

② /* Pgm to write record of student to a file using array of structures */

```
#include<stdio.h>
typedef struct
{  int rollno,marks;
   char name[25];}student;
void main()
{
 student stud[5];
 int i;
 fp=fopen("Student.txt","w");
 if(fp==NULL)
 {printf("File opening Error");
  return0;
 }
for(i=0;i<5;i++)
{
printf("Enter rollno,marks,name:");
scanf("%d",&stud[i].rollno);
scanf("%d",&stud[i].marks);
scanf("%s",stud[i].name);
}
for(i=0;i<5;i++)
{
printf("Student details");
printf("Rollno:%d",stud[i].rollno);
printf("Name:%s",stud[i].name);
printf("marks:%d",stud[i].marks);
fprintf(fp,"%d %s %d",stud[i].rollno,
          stud[i].name,stud[i].marks);
}
printf("Data written to the file");
fclose(fp);
}
```

O/P

Enter rollno, marks, name.

1 98 ABC

Enter rollno marks, name

2 86 xyz

(5 student details)

Student details

Rollno=1, Name=ABC, marks=98

Data written to the file.

Random Access file.

- * fseek()
- * ftell()
- * frewind()
- * fgetpos()
- * fsetpos()

functions are used to randomly access a record stored in a binary file.

fseek()

The function fseek() is used to reposition a binary stream.

Syntax: `int fseek(FILE *stream, long offset, int origin);`

fseek() is used to set the file position pointer for the given stream. The variable offset is an integer value that gives the number of bytes to move forward or backward in the file. The offset value may be positive or negative. The origin value should have 0, 1, 2.

* SEEK_SET: To perform input or output on offset bytes from start of the file. (or) Value 0
* SEEK_CUR: To perform input or output on offset bytes from the current position in the file. (or) Value 1
* SEEK_END: To perform input or output on offset bytes from the end of the file.

(1) fseek(fp, 0L, SEEK_SET);	Move to the beginning of the file.
(2) fseek(fp, 0L, SEEK_CUR);	Stay at the current position.
(3) fseek(fp, 0L, SEEK_END);	go to the end of the file.
(4) fseek(fp, m, SEEK_CUR);	move forward by m bytes in the file from the current location.
(5) fseek(fp, -m, SEEK_CUR);	move backwards by m bytes in file from the current location.
(6) fseek(fp, -m, SEEK_END);	move backwards by m bytes from the end of the file.

ftell()

The ftell() function is used to know the current position of file pointer.

Syntax: `long ftell(FILE *stream);`

In successful ftell() function returns the current file position for stream. In case of error, ftell() returns -1

Error can occur because of 2 reasons.

* using ftell() with a device that cannot store data.
* when the position is larger than that can be represented in a long integer.

rewind()

It is used to adjust the position of file pointer. so that the next I/o operation will take place at the beginning of the file.

Syntax. `rewind(FILE *fp);`

rewind() is equivalent to calling fseek() with following parameters. fseek(f, 0L, SEEK_SET);

fgetpos()

It is used to determine the current position of the stream.

Syntax `int fgetpos(FILE *Stream, fpos_t *pos);`

stream is the file whose current file pointer position has to be determined.

pos is used to point to the location where fgetpos() can store the position information.

on success fgetpos() returns zero and in case of an error a non-zero value is returned.

fsetpos()

It is used to move the file position indicator of a stream to the location indicated by the information obtained in pos by making a call to the fgetpos().

Syntax.

`int fsetpos(FILE *stream, const fpos_t pos);`

on success fsetpos() returns a zero and clears the EOF indicator. In case of failure it returns a non-zero value.

eg fsetpos(), fgetpos()

```
#include<stdio.h>
void main()
{ FILE *fp;
  fpos_t pos;
  char text[20];
  fp = fopen("Random.c","rb");
  if (fp==NULL)
  { printf("Error opening file");
    exit(1);
  }
  fread(text,sizeof(char),20,fp);
  if(fgetpos(fp, &pos)!=0)
  { printf("Error in fgetpos()");
    exit(1);
  }
  fread(text, sizeof(char),20,fp);
  printf("In 20 bytes at byte %ld : %s",
                          pos, text);

  pos=90;
  if (fsetpos(fp,&pos)!=0)
  {
    printf("Error in fsetpos");
    exit(1);
  }
  fread(text,sizeof(char),20,fp);
  printf("20 bytes at byte
          %ld : %s",pos,text);
  fclose(fp);
}
```

eg) fseek(), ftell, rewind().

```
#include<stdio.h>
void main()
{
  FILE *fp;
  fp=fopen("input.txt","r");
  if(fp!=NULL)
  {printf("Fp at location %d",ftell(fp));
  fseek(fp,13,0);
  printf("fp at location %d",ftell(fp));
  fseek(fp,4,1);
  printf("fp at location %d",ftell(fp));
  fseek(fp,0,2);
  printf("fp at location %d",ftell(fp));
  fseek(fp,-10,2);
  Printf("fp at location %d",ftell(fp));
  rewind(fp);
  printf("fp at location %d",ftell(fp));
  }
}
```

input.txt

0 1 2 3 4 5 6 7 8
9 10 11 12 13 14
15 16 17 18

```
fp at location 0
fp at location 13
fp at location 17
fp at location 18
fp at location 8
fp at location 0
```

Transaction processing using - Random Access Files

```
#include<stdio.h>
Struct clientdata
{
  int accno;
  char lname[15],fname[15];
  double bal;
};
int readchoice();
vocd textfile(FILE*);
void updaterecord(FILE*);
void newrecord(FILE*);
void delrecord(FILE*);
void main()
{
  FILE *fptr;
  int ch;
  if((fptr=fopen("credit.dat","r+")
                ==NULL)
    printf("File could not be opened");
  else
  { print+("Enter your choice");
    while(choice=readchoice())!=5)
    {switch(choice)
     {
      case1:
          textfile(fptr); break;
      case2:
          updaterecord(fptr); break;
      case3:
          newrecord(fptr); break;
      case4:
          deleterecord(fptr); break;
     }}
  fclose(fptr);
}
```

```
void textfile(FILE * readptr)//Printing purpose
{FILE *writeptr;
 Struct clientdata cli={0," "," ",0.0};
 if(writeptr=fopen("accounts.txt","w")
                ==NULL)
   Printf("File could not be opened");
else
 { rewind(readptr);
  fprintf(writeptr,"%s %s %s %s\n",
     "Acct","Lastname","Firstname","Bal");
 while(!feof(readptr))
 {
  fread(&client,sizeof(Struct clientdata),
                1,readptr);
 if(client.accno!=0)
  fprintf(writeptr,"%d %s %s %f\n",
 cli.accno,cli.lname,cli.fname,cli.bal);
 }
fclose(writeptr);
}
}
void updaterecord(FILE * fptr)
{ int account;
  double trans;
  Struct clientdata client={0,"","",0,0.0};
 Printf("Enter account to update");
 scanf("%d",&account);
 fseek(fptr,(account-1)*sizeof(Struct
   clientdata),SEEK_SET);
 fread(&client,sizeof(Struct clientdata),
                1,fptr);
 if(client.accno==0)
 Printf("Accno %d has no information",
                account);
```

Scanned with CamScanner

```
else
{
printf("%d %s %s %f \n", client.accno,
    client.lname, client.fname, client.bal);
Printf("Enter charge(+) or payment(-)");
scanf("%lf", &trans);
client.bal += trans;
printf("%d %s %s %lf \n", client.accno,
  client.lname, client.fname, client.bal);
fseek(fptr, (account-1)*sizeof(struct
    clientdata), SEEK_SET);
fwrite(&client, sizeof(struct clientdata), 1,
                    fptr);
}}

void deleteRecord(FILE *fptr)
{
struct clientdata client, bclient={0,"","",0.0};
int accno;
printf("Enter acc no to delete (1-100):");
scanf("%d", accno);
fseek(fptr, (accno-1)*sizeof(struct
        clientdata), SEEK_SET);
fread(&client, sizeof(struct clientdata),
                    1, fptr);
if(client.accno == 0)
printf("Account %d does not exist\n",
                    accno);
else
{
fseek(fptr, (accno-1)*sizeof(struct
        clientdata), SEEK_SET);
fwrite(&bclient, sizeof(struct clientdata), 1,
                    fptr);
}}

void newrecord(FILE *fptr)
{
Stuct clientdata client={0,"","",0.0};
int accno1;
printf("Enter new account no");
Scanf("%d", &accno1);
fseek(fptr, (accno-1)*sizeof(Struct
      clientdata), SEEK_SET);
fread(&client, sizeof(Struct clientdata,
                1, fptr);
if(client.accno1 != 0)
    Printf("Account %d already
        Contains information",
        client.accno);
else
{
Printf("Enter Lastname, firstname,
                balance\n");
scanf("%s %s %lf", client.lname,
    client.fname, &client.bal);
client.accno = accno1;
fseek(fptr, (client.accno-1)*
sizeof(struct clientdata), SEEK_SET);
fwrite(&client, sizeof(struct
                client.data), 1, fptr);
}}

int readchoice()
{
  int ch;
  printf("Read the choice");
  printf("1-Store accounts.txt\n"
         "2-update an account\n"
         "3-add a new account\n"
         "4-delete an a ccount\n"
         "5-end program\n");
  scanf("%d", &ch);
  return ch;
}
```

O/P. accounts.txt

Acct	lastname	Firstname	Bal.
45	M	Meena	6000.000
36	K	Aravind	4500.65
23	D	Tivakar	650.85

Enter your choice
1. Store account.txt
2. update an account
3. add a new account
4. delete an account
5. end program.

2
account to update 23
23 D Tivakar 650.850000
charge(+) or payment(-): +500
23 D Tivakar 1150.850000.

Command Line Arguments.

An executable program that perform a specific task for operating system is called a command. Some arguments are to be associated with the commands hence these arguments are called as command line arguments.

The command line arguments supply parameters to the program during its execution.

No arguments were passed to the main(). But in order to understand the full declaration of the main function. The main() can accept two arguments

* The first argument is an integer value that specifies number of command line arguments.
* The second argument is a full list of all the command line arguments.

The full declaration of main() can be given as

```
int main(int argc, char *argv[])
```

The int argc specifies the number of arguments passed into the program from the command line, including the name of the program.

The array of char pointers, argv contains the list of all the arguments. argv[0] is the name of the program, or an empty string if the name is not available. argv[1] to argv[argc-1] specifies the command line argument. In the C program, every element in the argv can be used as a string.

```
#include<stdio.h>
void main(int argc, char *argv[])
{
  int i;
  printf("Number of arguments passed=%d",argc);
  for(i=0; i<argc; i++)
    printf("argv[%d] = %s",i,argv[i]);
  return 0;
}
```

o/p

```
C:\>tc cmd.c command line arguments
  o/p Number of argument passed = 3
      arg[0] = command
      arg[1] = line
      arg[2] = arguments.
```

Unit II - Pgms

1. W.a.pgm to interchange the smallest & largest number in an array.

```
#include<stdio.h>
void main()
{
 int i, n, a[50], temp;
 int small, big, spos, bpos;
 printf("Enter the no. of elements\n");
 scanf("%d", &n);
 printf("Enter the elements \n");
 for(i=0; i<n; i++)
  scanf("%d", &a[i]);
 small = a[0];
 big = a[0];
 spos = 0;
 bpos = 0;
 for(i=0; i<n; i++)
 {
  if (a[i] < small)
  {
   small = a[i];
   spos = i;
  }
  if (a[i] > big)
  {
   big = a[i];
   bpos = i;
  }
 }
printf("Smallest of these no: %d", small);
printf("Biggest of these no: %d", big);
printf("Pos of the biggest no: %d", bpos);
printf("pos of the smallest no: %d", spos);
temp = a[bpos];
a[bpos] = a[spos];
a[spos] = temp;
printf("The new array is \n");
for(i=0; i<n; i++)
  printf("%d", a[i]);
}
```

o/p. Enter the no. of elements 5
Enter the elements
4 8 6 2 7
Smallest of these no: 2
Biggest of these no: 8
pos of the biggest no: 2
pos of the smallest no: 4.
The new array is
4 7 2 6 8

2. matrix multiplication.

```
#include<stdio.h>
void main()
{
int a[10][10], b[10][10], c[10][10];
int i, j, k, m, n, p, q;
printf("Enter the size of a matrix\n");
scanf("%d %d", &m, &n);
printf("Enter the size of b matrix");
scanf("%d %d", &p, &q);
if (n != p)
{
 printf("matrix multiplication is
                    not possible\n");
  exit();
}
printf("Enter the elements of a matrix");
for(i=0; i<m; i++)
{
 for(j=0; j<n; j++)
  scanf("%d", &a[i][j]);
}
printf("Enter the elements of b matrix");
for(i=0; i<p; i++)
{
 for(j=0; j<q; j++)
  scanf("%d", &b[i][j]);
}
printf("matrix multiplication");
for(i=0; i<m; i++)
{
 for(j=0; j<q; j++)
 {
  c[i][j] = 0;
  for(k=0; k<q; k++)
   c[i][j] += a[i][k] * b[k][j];
 }
}

printf("Resultant matrix");
for(i=0; i<m; i++)
{
 for(j=0; j<q; j++)
  printf("%d\t", c[i][j]);
 printf("\n");
}
```

③ In a class there are 10 students. Each student is supposed to appear in 3 tests. W.a.p using 2D array to Print
* The marks obtained by each student in diff subjects
* Total & avg marks of each student
* Calculate the class avg;

```
#include<stdio.h>
void main()
{ int marks[10][3],i,j;
  int totm[10]={0};
  float carg=0.0,targ=0.0,arg[10];
  printf("Enter the data\n");
  for(i=0;i<10;i++)
  { printf("Enter the marks for 3 sub");
    for(j=0;j<3;j++)
      scanf("%d",&marks[i][j]);
  }
  for(i=0;i<10;i++)
  { for(j=0;j<3;j++)
    { totm[i]+=marks[i][j];
    } }
  for(i=0;i<10;i++)
  { for(j=0;j<3;j++)
      arg[i]=(float)totm[i]/3.0;
  }
  for(i=0;i<10;i++)
    targ=targ+arg[i];
  carg=(float)targ/10;
printf("stadno\t marks\t\t\tmarks\t
                arg");
for(i=0;i<10;i++)
{ for(j=0;j<3;j++)
  printf("%d",marks[i][j]);
printf("%d %f",totm[i],arg[i]);
}
printf("Class Avg=%f",carg);
}
```

④ W.a.c pgm to print the Prime number from 1-100 using Recursion.

```
#include<stdio.h>
int prime(int,int);
void main()
{ int i,flag=1;
  printf("The prime no blw 1-100");
  for(i=2;i<=100;i++)
  {
  flag=prime(i,i/2);
  if(flag==1)
    printf("%d",i);
  }
int prime(int num,int max)
{ if(max==1)
    return 1;
  else
  { if(num%max==0)
      return 0;
    else
      prime(num,max-1);
  } }
```

5. Sum of digits - Recursion.

```
#include<stdio.h>
int sod(int n);
int main()
{ int num,res;
  printf("Enter a number:");
  scanf("%d",&num);
  res=sod(num);
  printf("Sum of dig=%d",res);
return 0;
}
int sod(int n)
{ if(n!=0)
  {
  return(n%10+sod(n/10));
  }
  else
    return 0;
}
```

Product details - change data members

Unit V pgms

① Student details - sequential access file.

```
#include <stdio.h>
typedef struct
{
 int sno,m1,m2,m3;
  char name[25];
}stud;
stud s;
void display(FILE *);
int search(FILE *,int);
void main()
{
  int i,n,key,opt;
  FILE *fp;
  printf("Enter how many records");
   scanf("%d",&n);
   fp=fopen("stud.txt","w");
  for(i=0;i<n;i++)
  {
  printf("Enter student information");
  scanf("%d%s%d%d%d",
          &s.sno,s.name,&s.m1,&s.m2,
                                &s.m3);
  fwrite(&s,sizeof(s),1,fp);
  }
  fclose(fp);
  fp=fopen("stud.txt","r");
  if(fp==NULL)
    printf("File not exist");
     exit(1);
  do
  {
  printf("1.Display\n2.search\n
        3.Exit\n Enter your choice");
  scanf("%d",&opt);
  switch(opt)
  {
  case 1: printf("Student records");
          display(fp);
          break;
  case 2: printf("Enter the search key");
          scanf("%d",&key);
  if(search(fp,key))
  {
   printf("Record found");
  printf("%d\t%s\t%d\t%d\t%d",
  s.sno,s.name,s.m1,s.m2,s.m3);
  }
  else
     printf("Record not found");break;
  case 3:
      printf("Exit");
       break;
  }
  }while(opt!=3);
  fclose(fp);
  void display(FILE *fp)
  {
   rewind(fp);
  while(fread(&s,sizeof(s),1,fp))
  printf("%d\t%s\t%d\t%d\t%d",
   s.sno,s.name,s.m1,s.m2,s.m3);
  }
  int search(FILE*fp,int skey)
  {
  rewind(fp);
  while(fread(&s,sizeof(s),1,fp))
  if(s.sno==skey)
      return 1;
  }
```

② Employee details - Random Access File.

```
#include<stdio.h>
#include<conio.h>
struct employee
{
char empname[20];
int age;
float salary;
};
typedef struct employee person;
FILE* fpnew;
void main()
{
person emp;
int i,n,rec,result;
FILE *fp;
```

```
printf("Enter how many records");
scanf("%d", &n);
fp = fopen("Employee.txt", "w");
for(i=0; i<n; i++)
{
printf("Enter employee information");
scanf("%s %d %f", emp.empname,
&emp.age, &emp.salary);
fwrite(&emp, sizeof(emp), 1, fp);
}
fclose(fp);
fpnew = fopen("Employee.txt", "r+b");
printf("Enter the record no, press
        -999 to stop");
scanf("%d", &rec);
while (rec >= 0)
{
fseek(fpnew, rec*sizeof(emp),
                    SEEK_SET);
result = fread(&emp, sizeof(emp), 1,
                    people);
if(result == 1)
{
printf("Record is");
printf("Name : %s\n", emp.empname);
printf("Age : %d\n", emp.age);
printf("Sal = %f\n", emp.salary);
}
else
printf("Record not found");
printf("Enter the record no");
scanf("%d", &rec);
}
fclose(fpnew);
}
```

③. W.a.pgm to open a file inventory and store in the following data item name, number, price, quantity. Extend the program to read this data from the file inventory and display the inventory table with the value of each item

```
#include<stdio.h>
void main()
{
FILE *fp;
int no, quantity, i;
float price, value;
char inve[10];
fp = fopen("inventory.txt", "w");
printf("Enter name, no, price,
                    quantity");
for(i=0; i<3; i++)
{
scanf("%s %d %f %d", inve,
        &no, &price, &quantity);
fprintf(fp, "%s %d %f %d",
        inve, no, price, quantity);
}
fclose(fp);
fp = fopen("inventory.txt", "r");
printf("Invenname number price
        quantity value\n");
for(i=0; i<3; i++)
{
fscanf(fp, "%s %d %f %d", inve,
        &no, &price, &quantity);
value = price * quantity;
printf("%s %d %f %d %f",
        inve, no, price, quantity, value);
}
fclose(fp);
}
```

o/p

input inventory.txt

name	no	price	quantity
aaa	112	25.80	115
bbb	115	79.25	75
ccc	178	98.96	104

o/p

invenname	Number	price	quantity	value
aaa	112	25.80	115	2967.0
bbb	115	79.25	75	5943.75
ccc	178	28.96	104	3011.84