Advantages (or) features of C Language:

C is the most popular programming language, C has many advantages:
Modularity: modularity is one of the important characteristics of C. we can split the C program into no. of modules instead of repeating the same logic statements (sequentially). It allows reusability of modules.

Middle level language: as a middle level language C combines both the advantages of low level and high level languages. (arrays,pointers etc).

General purpose programming language: C can be used to implement any kind of applications such as math's oriented, graphics, business oriented applications.

Portability: we can compile or execute C program in any operating system(unix,dos,windows).

Powerful programming language: C is very efficient and powerful programming language, it is best used for data structures and designing system software.
C is case sensitive language.

Character set of C Language:

C compiler allows to use many no. of characters :

Lower case letters	: a b c … z
Upper case letters	: A B C… .Z
Digits	: 0 1 2 3 … 9
Other characters	: + - * ( ) & % $ # { } [ ]' " : ; etc.
White space characters	: blank, new line, tab space etc.

Conversion Characters(format specifiers):

Conversion characters or format specifiers are used to provide the format for the value to be print.. it has a prefix ' % ' and followed by a specifier.

%d	: prints integer
%f	: prints float and double
%c	: prints character
%s	: prints string
%o	: prints octal value
%u	: prints unsigned integer
%x	: prints hexa decimal value

Structure of C Program :

Structure describes the way in which a program can be written , c structure includes :

Pre processor statements :
.
function proto types:
.
global declarations:
.
function1(main):
{
    local declarations:
    .
    code.
    .
}
function2(sub program)
{
local declarations:
    .
    code.
    .
}

```
#include<stdio.h> /* pre processor */
void fun1( )        /* function pro.type */
int x=10;           /* global declaration */
main ( )
{
int y=20;
 printf( " this is main block");
printf ("\n sum=%d",x+y);  /* prints 30*/
fun1( )    /* calling of function*/
getch( )
}
void fun1( )
{
int z=30;
printf( " this is fun2 block");
printf ("\n sum=%d",x+z);  /* prints 40*/
}
```

pre processor statements: these must be at the beginning of the program. It includes include statements, user defined macros, to get processed before the actual program. These can be followed by #(hash) character.

Function prototypes: function prototypes explain the nature of the function which includes return type, function name and list of the arguments type.

Global declarations: it allows to declare variables as global whose scope is valid through out the program.

Main function : every C program consists one function is main(), the statements in main only can be executed.

Functions (sub program): a reusable block of statements that gets executed upon calling, a program have any no. of function which contain local declarations and code.

Local declarations: every function at it's beginning may have local variables whose scope is valid with in the function only.

Code: code is some logical statements written according to the syntax of C to solve a problem.

C Tokens:

C tokens includes identifiers, keywords, variables, constants and operators.

Identifiers : the names of variables, functions, labels and various other user defined items are called as identifiers. Identifiers contains alphabets, digits and underscore. It can't include spaces. It starts with alphabet.
Ex: lcm( ), ab_cd, pi,r etc.

Keywords: keywords(reserved words) are defined by C compiler and each word has different meaning, these can not used as names of variables, functions or labels.
Ex: auto, for, switch, do, while, case, else, goto etc.

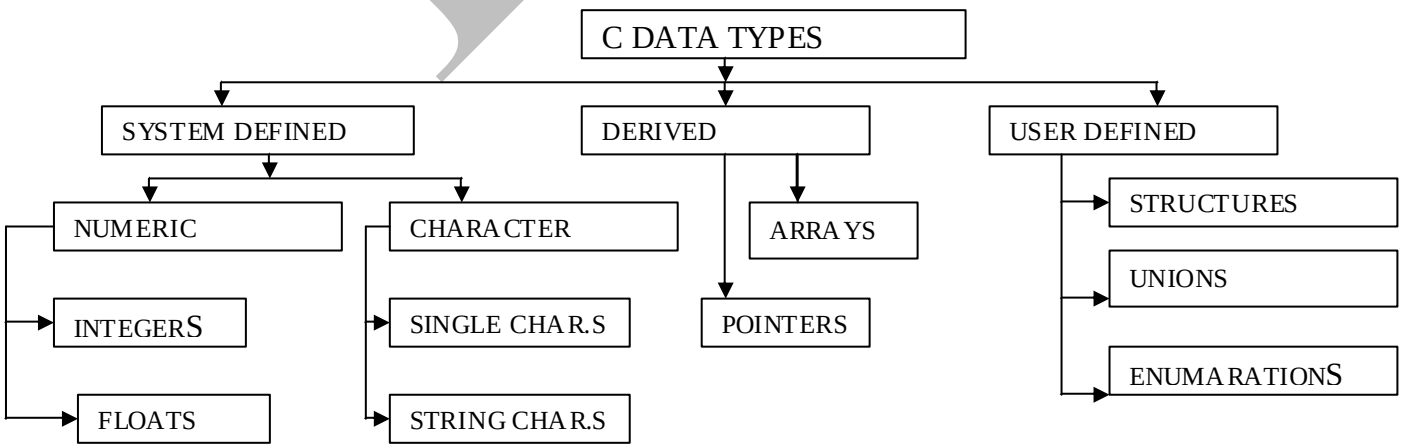
Variables: these are defined by user and used in program. It is composed of letters, digits and underscore. These are instances of data types.
Ex: int a,x1,a_b ; float pi,r; char x1;

Constants : constants means which never changes it's value.
Ex: int x=10,
float a=2.35;
char c='a';
char name[ ]="vijay";

operators: C allows several types of operators such as arithmetic(+,-.*,/,%) and other operators.

Data types in C :

Data type determines the type of data a variable will hold. If a variable 'x' is declared as 'int'. it means x can hold only integer values.
Every variable which is used in the program must be declared as what data-type it is.



System defined data types: these data types can be defined by C compiler, these are two types:
Numeric data types : these are two types: integers and characters.
Character data types: these are two types single characters and string characters.

Derived data types: these data types can be declared by user by deriving the system defined data type concepts. These are two types.
Arrays : set of similar data elements are called as arrays.
Ex: int a[10], char name[20];
Pointers: pointer is a variable which holds the address of similar data variable.
Ex: int a=5,*p; -→ p is a pointer variable declared with * character.

User defined data types: these data types can be defined by user, these are three types.
Structures : group of deferent data items combined together are called as structures. All the data items can stored in individual memory locations.
Unions : group of deferent data items combined together are called as unions. All the data items can share unique memory location.
Enumerations: these data types allows t declare string constants with integer equalant values.

Data type	Memory	Format specifier
int	2 bytes	%d, %i
char	1 bytes	%c
float	4 bytes	%f
double	8 bytes	%f
string	Size of char array	%s
Unsigned int	2 bytes	%u
Long unsingned	8 bytes	%lu
long	4 bytes	%ld

Escape Sequences:

Escape sequences are used in input and output functions such as printf and scanf. Which has '\' and a following character.
\n → new line
\t → horizental tab
\v → vertical tab
\b → back space
\? →question mark
\" → double quote

Differences between variables and identifiers :

Variables are tokens defined by user and used in programming. It must start with a letter or underscore(_), it can't include spaces, it can contain digits.

Valid	invalid
int x	int 3x
float pi	float pi+var
int a_b	inta.b

the names of variables, functions, labels and other user defined functions are called as identifiers.

IDENTIFIER	VARIABLE
All identifiers are not variables	All variables are identifiers
Identifier may not have any memory unless it is a variable	All variables have memory
Mentioning the type of an identifier is not needed unless it is a variable.	Type of the variable must be defined.

Input output functions of C language(statements):

All these input and output functions are declared in <stdio.h>

Function	Type	meaning
printf	Output	Writes formatted text on the screen
scanf	Input	Reads formatted text from keyboard
getc	Input	Reads a character from stream
putc	Output	Writes a character into a stream
gets	Input	Reads a string from keyboard
puts	output	Writes string on to the screen

Commenting the C statements(wrting the comments in C)

Comment is a non executable statement in C. when a programmer includes comments the program can be easily understandable. Those comments must be enclosed in '/* and */'.

```
Ex: main( )
    {
    printf("hello world");  /* printf prints hello on the monitor*/
    }
```

Different types of errors:

Error is a mistake or bug, which can be associated with programmer in program, common programming errors in C can be classified into 3 types:

1. compilation errors.(syntax errors)
2. linker errors.
3. logical errors.

Compilation(syntax) errors: these are raised when we compile the program and can be located and corrected easily. Most common compilation or syntax errors are.

- Undefined variable
- Redeclaration of variable
- Unterminated string character
- Missing semicolon ;
- Function call missing (,;")
- Function should have prototype.

Linker errors: the program must be linked to the 'C' library. If it fails in such case these errors are raised . most common errors are:

- Unable to link cos.obj
- Undefined symbol

Logical (run time) errors: these are raised because of the due to logical inefficiency. We can know them when program gets executed. It is very difficult to locate them, most common logical errors are:

- Divide by zero
- Floating point error
- Null value
- Garbage result in printing.

Operators in C

C supports many kinds of operators, operators can be applied on variables or consonants. Classification of operators can be done in two ways based on number of operands and their nature.

I number of operands involved:

Unary operators: can be applied on only one operator(++,--,size of)
Binary operator: can be applied on two operands(+,*)
Ternary operator: can be applied on three operands(? :)

II their nature:

Type	Operators	Meaning
Arithmetic operators	+ - * / %	Addition subtraction multiplication division modulus
Relational operators	>> >= < <= = = !=	Greater than greater than or equal to less than less than or equal to equal to(comparison) not equal to
Logical operators	&& | | !	AND OR NOT
Assignment operators	=, +=, -=, / =, %=	assignment
Bit wise operators	& | ^ << >> ~	Bit wise AND Bit wise OR Bit wise XOR left shift right shift compliment
Other operators	++ -- sizeof ? :	Increment decrement sizeof operator ternary conditional

Bit wise operators in C

When bit wise operators are used on operands they convert the numbers into binary system and then apply bit wise operators. The bit wise operators are:

Bit wise operators	& | ^ << >> ~	Bit wise AND Bit wise OR Bit wise XOR left shift right shift compliment

AND : the result of applying AND is 1 if both are 1 other wise 0.

Ex:

```
   Y=1101          13
   X=1110          14
  ----------       -----
X & Y=1100         12
-----------------------------------
```

OR: the result of applying OR is 1 if at least one of bit value is 1 other wise 0.

Ex:

```
   Y=1101          13
   X=1110          14
  ----------       -----
X |Y=1111          15
-----------------------------------
```

XOR:

the result of applying XOR is 1 if only one of bit value is 1 other wise 0.

Ex:

```
   Y=1101          13
   X=1110          14
  ----------       -----
X ^Y=0011          03
-----------------------------------
```

example program to demonstrate bit wise operators:

```
main( )
{
int x=13,y=14;
clrscr( );
printf(" \nand= %d",x&y);
printf(" \n or=%d",x|y);
printf(" \n xor=%d",x^y);
}
```

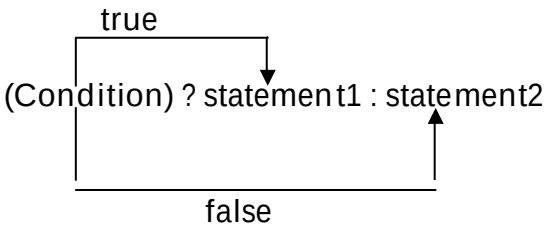
output :

```
and=12
or=15
xor=3
```

Ternary (Conditional) operator:

- Conditional operator is an equivalent form of if –else loop.
- It is called as ternary operator because used on three operands.

Syntax:



This expression evaluates statement 1 or statement2 depends on the condition. If the condition is true statement1 gets executed other wise statement2 gets executed.

Ex:

```
 main( )
{
 int x;
clrscr( );
printf( " enter a no");
scanf("%d",&x);
(x%2==0) ? printf("even no") : printf("odd no");
getch();
}
```

assignment operators : C provides different types of assignment operators are used to assign values.

=,+=,-=,*=,/=,%=

```
main ( )
{
int  a=5;
clrscr( );
printf( " \n %d", a+=5);     /* prints 10 */
printf( " \n %d", a-=2);     /* prints 8 */
printf( " \n %d", a*=3);     /* prints 24 */
printf( " \n %d", a/=4);     /* prints 6 */
getch( );
}
```

size of operator :

C provides a unary operator named *sizeof* to find number of bytes needed to store an object. An xpression of the form sizeof(object) returns an integer value that represents the number of bytes needed to store that object in memory. It can be used with data types or variables or constants, we can use this operator in dynamic memory allocation to calculate memory.

Ex:

```
main( )
{
int  x=10;
clrscr( );
printf("%d",sizeof(x));          /* prints 2 */
printf("%d",sizeof(int));        /* prints 2 */
printf("%d",sizeof(char));       /* prints 1 */
printf("%d",sizeof(float));      /* prints 4 */
gettch( );
}
```

Boolean expression :

Boolean expression is one, which contains "true" or "false" these are result of expressions. i.e (< ,<=,>,>=,!=) . in C all integer values also can be Boolean expressions because a non-zero value is treated as "true" and a zero value treated as "false".

Examples:

1)
```
main( )
{
clrscr( );
if(1)
printf("hello");
else
printf("bye");
}
```

It prints hello, because 1 contains positive Boolean (true) expression

2)
```
main( )
{
int x=3;
clrscr( );
if(x>0)
printf("hello");
else
printf("bye");
}
```

It prints hello, because (x>0) contains positive Boolean (true) expression

Control Structures in C:

Normally statements in C program are executed sequentially i.e in the order in which they are written. This is called sequential execution. We can transfer the control point to a desired location is possible with control structures. C allows many kinds of control statements.

I. Conditional(decision) control structures
   - if
   - if –else
   - if-elseif – else
   - nested if

II. Jumping(branching) control structures
   - goto
   - break
   - return
   - continue

III. Loop(iterative) control structures
   - for loop
   - while loop
   - do - while loop

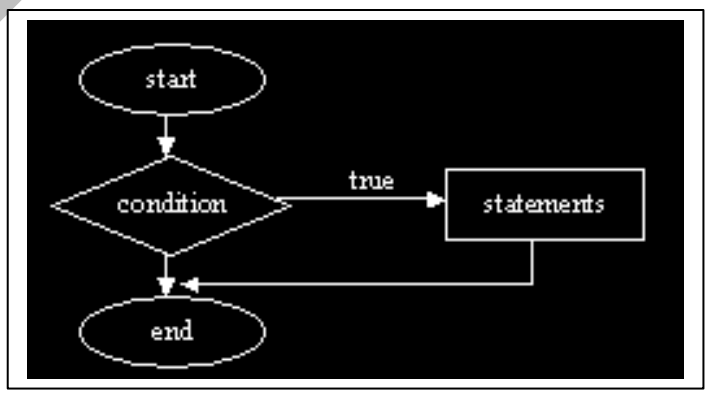
IV. Multi way conditional(case) control structures
   - switch- case

Conditional Control Structures:

If: if is also called as conditional statement in if statements get executed only when the condition is true, it can terminate when the condition becomes false.

Syntax:

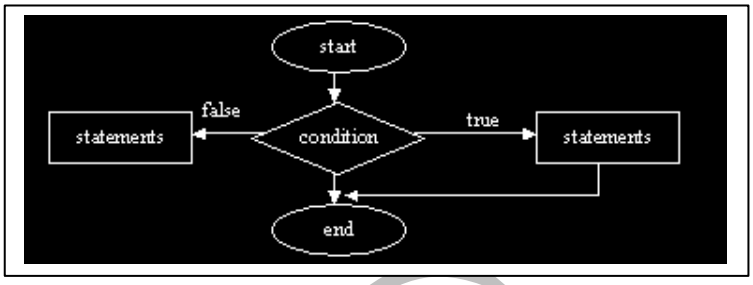
```
if (condition)
{
……..
…statements…
……….
}
```



if –else: in if-else conditional control statement, statements in if block gets executed only when the condition is true and statements in else block gets executed only when the condition is false.

Syntax:

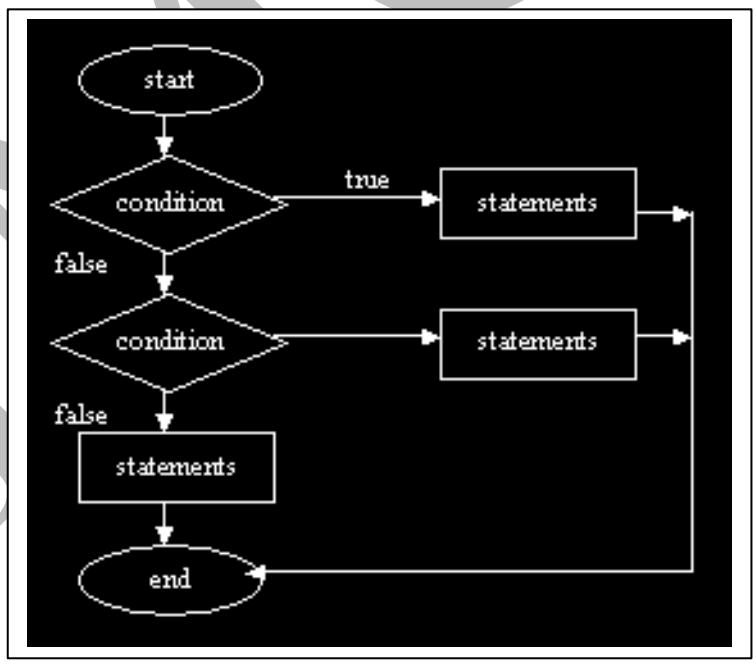
```
if (condition)
{
statements
}
else
{
statements
}
```



if –else if-else: in this statement statements in loops gets executed when the corresponding conditions are true. Statements in final else block gets executed when all other conditions are false. We can use any no. of else-if blocks between if and else.

Syntax:

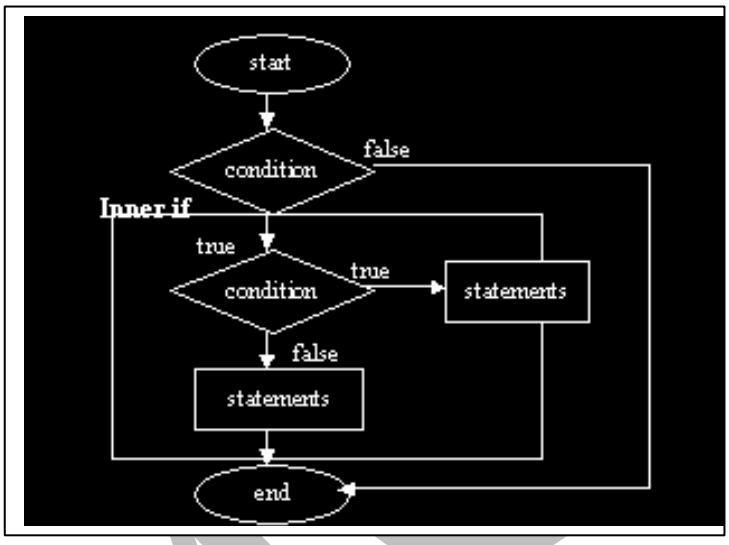
```
if (condition1)
{
statements
}
else if(condition2)
{
statements
}
else if(condition3)
{
statements
}
…
….
else
{
statements
}
```



nested if : writing the if statement with in another if is called as nested if. Inner if is processed only when outer if condition is true. Hence statements of inner if gets executed when outer if and inner if conditions are true.

Syntax:

```
if (condition1)
{
 if(condition2)
  {
   statements
  }
 else
  {
  statements
  }
}
else
 statements
```



II. Jumping Control Structures:

goto :

goto os a kind of branching structure. It moves the control (moves forward or back)to label. goto is condition dependent.

In above goto forward statements2 gets omitted because after executing statements1 goto moves the control to label statements.

In goto back after executing statement1, statements2 repeated more times depends on condition(goto transfers to label, if condition is true only).

Syntax1 : goto back

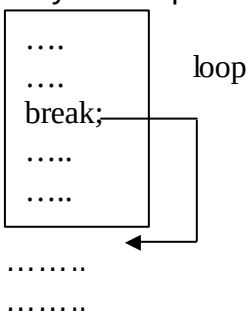
```
……..
…statements1..
..
label:
..statements2
..
if(condition)
goto label;
..
EX:
i=0;.
abc:
 printf("hello");
i++;
if(i<10)
goto abc;
```

syntax: goto forward

```
……statemnts1..
if(condition)
goto label;
..
.. statemnts2..
label:
…….
…..
EX:
i= -3;
if(x>0)
goto abc;
x=-x;
abc:
printf("%d",x);
```

Break :

Break quits the corresponding iterative loop statements, break is valid only for loop statements and switch case .



…….
…….

```
Ex:
for(i=1;i<10;i++)
{
printf("hello");
if(i==5)
break;
}
```

→ Prints 5 times only

return :

return is used in functions. It moves the control to calling function from the called function module. Return is also return values to the point where the function is called.

```
Ex: main( )
    {
     int a=10,b=20;
     printf("%d",sum(x,y);
    getch( );
    }
    int sum(int x, int y)
    {
    return  x+y;
    }
```

Iterative (loop) control statements :

for loop :

in for loop statements get executed as long as condition is true. For loop contains three expressions, expression1 includes initializing the variables and expression 2 includes increment or decrement of initialized variable.

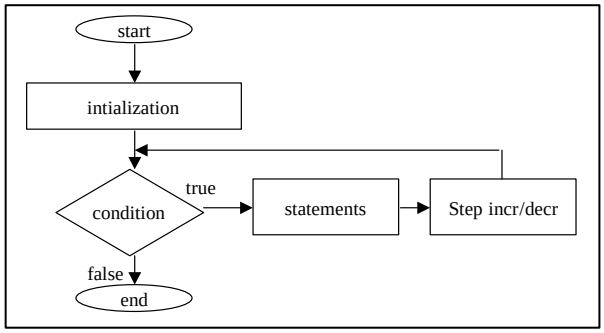
Syntax:

```
    for( expression1; condition; expression2)
   {
      statements;
   }
```

example :

```
 for(i=1;i<=10;i++)
printf("hello");
```

→ prints hello 10 times.

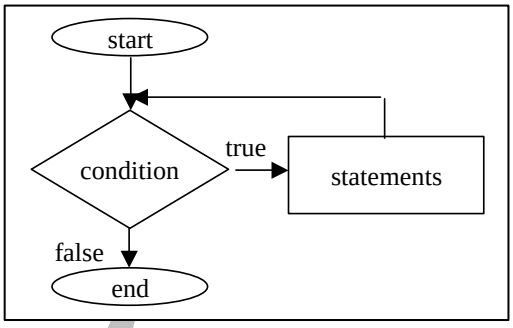


While loop :

In while loop statements get executed as long as condition is true. In while it checks the condition first and executes the statements later. So minimum no. of execution times of statements is 0.

Syntx:

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



example :

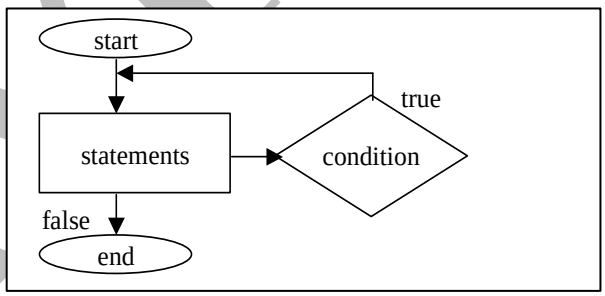
```
i=1;
while(i<=10)
{
printf("hello");⟶ prints hello 10 times.
i++;
}
```

do while loop:

In do while loop statements get executed as long as condition is true. In do while it executes the statements first and checks the condition later. So minimum no. of execution times of statements is 1.

Syntx:

```
do
{
 ....
  statements;
 ....
}while(condition);
```



example :

```
i=1;
do
{
printf("hello");⟶ prints hello 10 times.
i++;
} while(i<=10);
```

multi-way conditional control statement:

switch case:

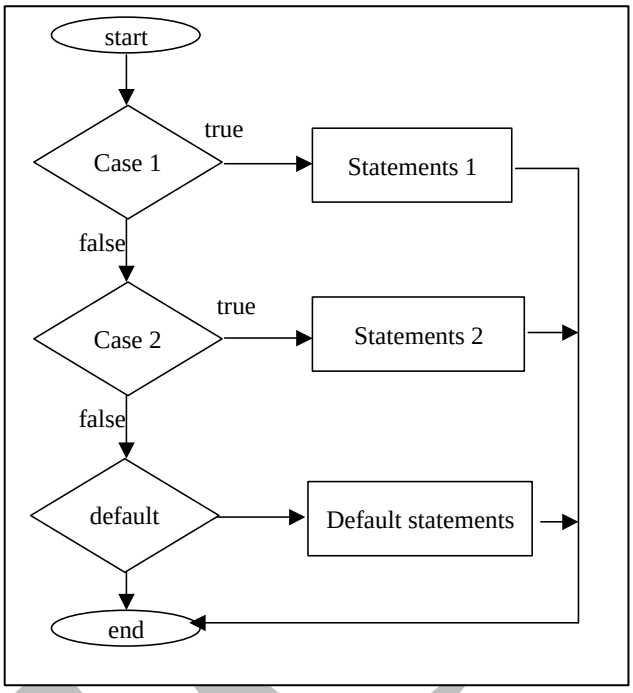
switch case is similar ti else if. In switch case statements get executed when corresponding case constants are true only(which have been mentioned match with the value of variable in switch). Control point comes to *default* when all the cases are false.

Syntax:

```
Switch(variable/expression)
{
case constant1: statements1;
                break:
case constant2: statements2;
                break:

case constant3: statements3;
                break:

default: statements;
                break:
}
```



ex:

program to perform arithmetic operators using switch case.

```
main( )
{
char op;
int a=10,b=5;
clrscr( );
printf("enter your operator +,-,*,/,%");
scanf("%c",&op);
switch(op)
{
case  '+': printf("sum=%d",a+b);
           break;
case  '-':  printf("difference=%d",a-b);
           break;
case  '*':  printf("mul=%d",a*b);
           break;
case  '/': printf("div=%d",a/b);
           break;
case  '%': printf("rim=%d",a%b);
           break;
default:  printf("your option is wrong";
           break;
}
getch( );
}
```

FUNCTIONS :

Function : a function is a reusable block of statements that gets executed on calling. It can be treated as sub program.

Advantages :

1. It allows reusability(a function can be called any no. of times).
2. it reduces the program code.
3. functions allows calls of other functions with in.

Types of Functions:

1). System defined functions : these are also called as built in or library functions. They are defined by C compiler and included in header files.

Ex: pow( ) → power function
Sqrt( )→square root function.
Printf( ), scanf( ), and string functions etc.

2). User defined functions: these are defined by the user. User can include any number of functions in the program, for which declaration and definition must be provided.

Structure of a function :

Return-type function-name(arguments list….)
{
local declarations
……..
…..code…
……..
}

example :

```
void sum( int,int);  /* function declaration(prototype) */
main ( )
{
 sum(10,20); → /* call of function */
}
void sum(int x, int y) /* function definition */
{
 printf (" sum=%d",x+y);
}
```

Q) :What is function prototype? Function delimitation? Argument?
Ans :

Function prototype : declaring of a function can be called as function prototype. This gives the clear picture of the function i.e which type of arguments it takes and hoe many arguments and what will be return value with the function name.

Syntax: *Return-type function-name(arguments list….);*

Function definition : it defines the behavior of function in terms of instructions. It may include list of the arguments, return variable and local declarations.

Argument : (parameter) : argument is a value that is sent from calling function to called function(definition). It is possible to send any no. of arguments to a function.

Q : Explain different ways of writing functions(user defined functions)?
Ans :

In C user defined functions can be written in four different ways.

1. No Argument , No Return
2. Argument , No Return
3. No Argument , Return
4. Argument, Return

No Argument , No Return : a function which does not take any arguments and does not return any value that function can be said as No Argument , No Return type function.

Example :

```
void show( );
main( )
{
show( );
}
void show( )
{
printf(" no argument no return");
}
```

Argument , No Return : a function which takes arguments but do not return any value that function can be said as Argument , No Return type function.

Example :

```
void sum( int,int);
main( )
{
int a=10,b=20;
sum(a,b);
sum(100,200)
}
void sum(int x, int y)
{
printf(" sum=%d", x+y);
}
```

No Argument ,Return : a function which does not take any arguments but return a value that function can be said as No Argument , Return type function.

Example :

```
int sum( );
main( )
{
int n;
n=sum( );
printf("sum=%d",n);
}
int sum( )
{
int a=10,b=20;
return a+b;
}
```

Argument ,Return : a function which takes arguments and returns a value that function can be said as Argument , Return type function.

Example :

```
int sum(int, int );
main( )
{
int n;
n=sum(10,20 );
printf("sum=%d",n);
}
int sum(int x, int y)
{
return x+y;
}
```

Q : Explain different types of storage classes in C?

Ans :

C has four types of storage classes. They explain the behavior of variables in terms of their scope and life. Storage class of a variable determines how, when and where storage will be allocated for the variable.

1. auto 2. extern 3.static 4.register

auto :

for all local variables, auto storage class is the default storage class. These variables get initialized or recreated each time a function is called. The scope of an auto variable is limited to the function. they get initialized to some garbage values if no data is give.

```
Ex: main( )
    {
    auto int x=10;
    printf("%d", x);
    }
```

extern :

for all global variables this is the storage class. The scope of this external variable is through out the program(in main, all modules). Extern variables get initialized "zero" automatically.

```
Ex:
   int x=30;
  show( );
   main( )
   {
    printf("%d",x); →30
    show( );
   }
   show( )
   {
   printf("%d",x); →30
   }
```

static :

static variable is local but life as a global variable. A static variable can be initialized only once. A static variable also gets initialized to "zero" automatically.

Ex:

```
show( );
main( )
{
 int  i;
for(i=1;i<=5;i++)
show( );
}
show( )
{
static int j;
j++;
printf(" \n function calling no%d",j);
}
```

Out put:
function calling no 1
function calling no 2
function calling no 3
function calling no 4
function calling no 5

register :

for register storage class variables, system allocates memory in CPU registers. Generally, this is done for fast accessing because the speed of the registers is high. Register storage class can be applied to local variables only. And register variables do not have any address.

Ex:

```
 main ( )
{
register int i=30;
printf(" %d ", i); → 30
printf(" %d ", &i); → /* error  i is a register do not have any address*/
}
```

Q. what are recursive functions?

Ans:

If a function is called within the same function is said to be recursion. A recursive function is a condition dependent.

Ex: to find the factorial value of given no(5) the logic is:

Fact(5) = 5 * fact(4)
= 5 * 4 * fact(3)
= 5 * 4 * 3 * fact(2)
= 5 * 4 * 3 * 2 * fact(1)
= 5 * 4 * 3 * 2 * 1

Logic is = n * fact(n-1)

Program : to find out the factorial value of a given no. using recursive method.

```
 int fcat(int);
main( )
{
int  n;
clrscr( );
printf(" enter a no");
scanf(" %d",&n);
printf("fact=%d",fact(n));
getch( );
}
int  fact( int x)
{
if((x==1))||(x==0))
return 1;
else
return  x*fact(x-1)
}
```

ARRAYS :

Array : array is set of elements of similar type, array elements ahare common name and array elements are stored in sequential memory locations. To refer to the elements of array we use index , array of index starts from zero(0).

Advantages of arrays:
It is capable of storing many elements at a time.
It allows random access of elements.
Disadvantages of arrays:
Predetermining the size of array is must
Memory wastage will be there.
To delete an element of an array we need to traverse through out array.

Types of arrays:
Arrays can be mainly three types single dimensional, double dimensional and multi dimensional
1. single dimensional arrays:
it can represent either one row or one column.

Syntax: data type array name[size];
data type array name[ ]={ values list};

ex: int a[10]→here array a can hold maximum 10 integer elements ata time 20 bytes of memory allocated for the array.

Ex: int a[ ]={10,20,30,40 }; → here array size is 4.

To read one dimensional array:

```
For(i=0;i< size;i++)
Scanf("%d",&a[I]);
```

To print one dimensional array

```
For(i=0;i< size;i++)
printf("%d",a[I]);
```

2. double (two) dimensional arrys:
double dimensional arrays can represent the data in the form of rows and columns means taking two indexes.
Syntax: data type array name[row size][col. Size]

int a[2][3]

here array a can hold maximum of (row x col) 6 elements and 12 bytes of memory allocated.

int a[ ][ ]={(10,20,30),(40,50,60)}; → here the size is 2 x 3.

To read two dimensional arrays:

```
For(i=0;i<row;i++)
For(j=0;j<col;j++)
Scanf('"d",&a[i][j]);
```

To print two dimensional arrays:

```
For(i=0;i<row;i++)
{
printf('\n');
For(j=0;j<col;j++)
Scanf("\t%d",&a[i][j]);
}
```

3. multi (three) dimensional arrays:

multi dimensional arrays can represent the data in the form of rows and columns by taking two or more indexes(two and three dimensional arrays).
Three dimensional arrays uses three indexes.
Ex: int a[2][2][3];
int a[ ][ ][ ]= {(10,20,30),(40,50,60),(70,80,90),(11,22,33)};
here the array size is 2x2x3 means(2x2) 4 rows and 3 columns, maximum 12 elements with 24 bytes of memory space.

Q.What is searching ? explain any two techniques?
Ans:
Searching is a process of finding out the existence of an element in a list. There are several methods such as linear search, binary search, fibonacci search etc.
Linear search: this is also called a sequential search. In this technique we compare the search key with the elements of the array sequentially till it is found or till the end of array.
Ex:

```
#include<stdio.h>
#include<conio.h>
main( )
{
 int a[10],i,x,found=0;
 clrscr( );
 printf("enter 10 values to array a");
 for(i=0;i<10;i++)
 scanf(2%d2,&a[i]);
 printf("enter element to be search");
 scanf("%d",&x);
  for(i=0;i<10;i++)
   if(a[i]==x)
   {
     found=1;
```

```
      break;
     }
     if(found==0)
    printf("elemet not found");
  else
  printf('element found');
 getch( );
}
```

binary search : this si also called as logarithmic search. In this technique it involves a method of splitting the set of elements into equal halves and splitting lower set(if elememt falls lower side) or upper set(if element falls upper side) repeatedly till element is found.

Ex:

```
#include<stdio.h>
 #include<conio.h>
 main( )
 {
  int a[10],i,x,found=0,l,h,mid;
  clrscr( );
  printf("enter 10 values to array a in sorted order");
  for(i=0;i<10;i++)
  scanf("%d",&a[i]);
  printf("enter element to be search");
  scanf("%d",&x);
  l=0;
 h=10-1;
 while(l<=h)
 {
  mid=(l+h)/2;
  if(x<a[mid])
  h=mid-1;
  else if(x>a[mid])
  h=mid+1;
 else if(x==a[mid])
   {
     found=1;
     break;
   }
 }
   if(found==1)
   printf("element found");
  else
```

```
   printf("not found");
  getch( );
}
```

STRINGS :

String is a set of characters. It is a one dimensional array type of char. Every string is terminated by null character '\0' whose integer equalant value is zero.

a	m	a	r	\0

String can be initialised in two different ways:

```
   char x[10];
   char x[ ]={'a','m','a','r','\0'};
    (or)
   char x[ ]=''amar'';
```

to print the string we write → printf("%s",x);

to access the individual characters of the string, we may write

```
    printf("%c",x[1]);→it prints 'm'
```

it is also possible to write set of strings:

```
    char x[5][10];
```

here first size 5 indicates no. of strings and second size 10 indicates no. of characters in each string.

```
    char x[ ][ ]={ ''amar'',''vijay'',''kiran'',''ravi''};
```

STRING FUNCTIONS:

String functions (built-in) of C are declared in the header file #include<string.h>. these functions allows us to work with strings.

strcpy( ), strcat( ), strlen( ), strrev( ), strlwr( ), strupr( ) strcmp( ).

strcpy( ): this string function is used to copy the one string characters to another string.(which copies second string to first string).

Syntax : strcpy(string1,string2);

Ex: program:

```
main ( )
{
  char x[10],y[10];
  clrscr();
  printf(''enter a string to x'');
```

```
  scanf(''%s'',&x);
  strcpy(y,x) → here x characters copied to y.
  printf(''\n y string is =%s'',y);
  getch( );
}
```

strcat( ): this string function is used to concatenate(add) two strings. (Second string can be added to first string).
Syntax : strcat(string1,string2);
Ex: program:

```
main ( )
{
  char x[10],y[10];
  clrscr();
  printf(''enter two strings to x,y'');
  scanf(''%s%s'',&x,&y);
  strcat(y,x) → here y characters added to x.
  printf(''\n x string is =%s'',x);
  printf(''\n y string is =%s'',y);
  getch( );
}
```

strlen( ); this string function is used to find the length (no. of characters) of the given string.
Syntax : strlen(string);
Ex: program:

```
main ( )
{
  char x[10];
  clrscr();
  printf(''enter a strings to x'');
  scanf(''%s'',&x);
  printf(''\n x string length is =%d'',strlen(x));
  getch( );
}
```

strrev( ): this string function is used to convert the given string characters into reverse order.
Syntax : strrev(string);
Ex: program:

```
main ( )
{
  char x[10];
  clrscr();
  printf(''enter a strings to x'');
  scanf(''%s'',&x);
  printf(''\n x reverse  string of x is =% s'',strrev(x));
  getch( );
}
```

strlwr( ):- this string function is used to convert the given string characters in to lower case format.

Syntax : strlwr(x)

strupr( ):- this string function is used to convert the given string characters in to uppercase format.

Syntax : strupr(x)

Ex:

```
main( )
{
 char x[10];
 clrscr( );
 printf(''enter a string'');
 scanf(''%s'',&x);
 strupr(x);
 printf(''\n given string in upper case=%s'',x);
 strlwr(x);
 printf(''\n given string in lower case=%s'',x);
 getch( );
}
```

strcmp( ): this string function is used to compare two strings. (which returns the value 0 when both strings are equal, returns 1(positive) when strin1 is greater than string 2, returns negative when the string 1 is less than string 2).

Syntax : strcmp(string1,string2);

Ex: program:

```
main ( )
{
  char x[10],y[10];
  int  I;
  clrscr();
  printf(''enter two strings to x,y'');
```

```
 scanf(''%s%s'',&x,&y);
 l=strcmp(x,y);
if(l>0)
 printf('' x is big'');
else if(l<0)
 printf('' y is big'');
else
printf(''both are equal'');
 getch( );
}
```

POINTERS :

Pointer is a variable which holds the address of simple variable of same type. To declare a pointer variable we use * operator.

Declaration of pointer variables:

```
 int  a=30,*p;
p=&a;

float b=2.58,*p1;
p1=&b;
```

let us consider in int a=30, *p; p=&a;
a value=30
a adrress= &a which is copying (storing) to pointer variable p.
printf(''%d'',a);→prints a value
printf(''%d'',&a)→prints a address value
printf(''%d'',p)→ prints a address value
printf(''%d'',*p)→ prints a value(value at that address)

advantages of pointers:

- it allows dynamic memory allocation.
- It allows call by reference mechanism for function arguments.
- Pointers improve efficiency of program

Disadvantages of pointers:

- Pointer variables requires extra memory.

Array of pointers:

Array of pointers is a set of pointer variables that holds address of simple variable of same type.

Ex:

```
main( )
{
   int  a[ ]={10,20,30,40,50},i;
   int *p[10];
    clrscr( );
  p[0]=&a[0];
  p[1]=&a[1];
  p[2]=&a[2];
  p[3]=&a[3];
  p[4]=&a[4];
  printf(''\n  values='');
  for(i=0;i<5;i++)
  printf(''\t %d'',a[i]);
  printf(''\n  addresses of values='');
  for(i=0;i<5;i++)
  printf(''\t %d'',p[i]);
  getch();
}
```

call by value :

in call by value simple variables are sent to the functions in the form of arguments. In this case only values of the arguments are copied to the corresponding function parameters. When we make some changes to variables in the function, it does not affect the variables of the calling function.

Example:

```
void swap(int,int);
main( )
{
  int  a=30,b=40;
  swap(a,b);
  printf(''\n after swap a=%d, b=%d'',a,b);
  getch( );
}
void swap(int x, int y)
{
 int temp;
 temp=x;
 x=y;
 y=temp;
}
```

call by reference :

in call by reference address of variables are sent to the functions in the form of arguments. In this case addresses of the argument variables are copied to the corresponding function pointer type parameters. When we make some changes to variables in the function, it affects the variables of the calling function.

Example:

```
void swap(int *,int *);
main( )
{
  int  a=30,b=40;
  swap(&a,&b);
  printf(''\n after swap a=%d, b=%d'',a,b);
  getch( );
}
void swap(int  *x, int  *y)
{
int temp;
temp=*x;
*x=*y;
*y=temp;
}
```

dynamic memory allocation:

- allocating the memory at run time is known as dynamic memory allocation.
- It eliminates predetermining of number of variables too be used in program
- This mechanism used in data structures
- In dynamic memory allocation after our purpose is completed we can release the memory to avoid blocking of the memory.
- Dynamic memory allocation is done using ''malloc'' or ''calloc''functions.
- Memory is done using ''free'' function.

Ex: program:-

```
main( )
{
 int *p;
*p=(int *)malloc(sizeof(int));
*p=30;
printf(''\n %d'',*p);
free(p);
getch();
}
```

Q)explain malloc, calloc, free functions?

Ans:

malloc, calloc, free functions are related to dynamic memory allocation. All these functions are declared in #include<malloc.h>

malloc(n):

- malloc allocates n bytes of memory dynamically.
- It returns the base address or NULL.
- The space is not initialised.

calloc(n,a):

- Calloc allocates a bytes of memory for n(array) elements dynamically.
- It returns the base address or NULL.
- The space is initialised zero.

free(n) :-

- it de-allocates the memory pointed by the pointer n.
- to be de-allocated the memory must be allocated by malloc or calloc functions.

USER DEFINED DATA TYPES
(STRUCTURES, UNIONS, ENUMARATIONS)

Q.what are user defined data types?

Ans:-

Apart from the primary data types int, char, float C allows us to define our own data types. This is done for the purpose of customized programming and to reduce the complexity of programs.

In C user defined data types can be created in three different ways

- structures
- unions
- enumerations

Q.what is structure? Explain with example?

Ans:-

- Structure is a way of defining the user defined data type that encapsulates the different data types. Or group of different data types combined together into an object is known as structure.
- Structure members can store in individual memory locations.
- Structure block can be terminated with semi-colon.
- To access the structure members we can use (.) dot operator with object.

Syntax of structure:

```
struct  < name of the structure>
{
  data-type1   variable 1;
  data-type2   variable2;
};
```

example program:

```
struct student
{
char name[10];
int r, marks;
};
main( )
{
  struct student s1;
  clrscr( );
  printf(''enter student name, rno,marks'');
  scanf(''%s%d%d'',&s1.name,&s1.r,&s1.marks);
  printf(''\n the details are'');
  printf(''\n name=%s\nr=%d\nmarks=%d,s1.name,s1.r,s1.marks);
  getch( );
}
```

Q. what are unions? Explain with example?
Ans:

- Union is a way of defining the user defined data type that encapsulates the different data types.
- All the union members can share single *common memory* location. whose size is equilant to highest of memory required by individual members.
- To access the structure members we can use (.) dot operator with object.

Example:

```
union alpha
{
  int  a,b;
};
main( )
{
  union alpha a1;
  a1.a=30;
  printf(''%d'',a1.b);
}
```

Q. what is enumeration? Explain ?
Ans:

Enumeration is also a way of defining the user defined data types . it allows declaring string constants with integer equilant values.

Example:

```
main( )
{
  enum  colors {red=10,green=20,blue=30};
  enum  colors  c1,c2;
  c1=red;
  c2=blue;
  printf(''%d'',c1);→prints 10
  printf(''%d'',c2);→prints 30
}
```

PROGRAMS FOR PRACTICE AND REFERENECE ALSO.

#1) write a c program to print the prime no's between 50-500.

Solution:

```
#include<stdio.h>
#include<conio.h>
main( )
{
int  x=50,y=500,i,j,c=0;
clrscr( );
for(i=x;i<=y;i++)
{
  for(j=1;j<=i;j++)
   if(j%i==0)
   c++;
  if(c==2)
  printf(" \t %d",i);
}
getch( );
}
```

#2) write a c program to perform matrix multiplication after verifying multiplication conditions.

Solution:

```
#include<stdio.h>
#include<conio.h>
main( )
{
  int  a[5][5],b[5][5],c[5][5],i,j,k,m,n,p,q;
clrscr( );
printf(''enter order of first matrix'');
scanf(''%d%d'',&m,&n);
printf(''enter order of second matrix'');
scanf(''%d%d'',&p,&q);
if(n!=p)
{
  printf(;;multiplication can not possible'');
   getch( );
   exit(1);
}
printf(''enter values for first matrix'');
for(i=0;i<m;i++)
for(j=0;j<n;j++)
```

```
scanf(''%d'',&a[i][j]);
printf(''enter values for second matrix'');
for(i=0;i<p;i++)
for(j=0;j<q;j++)
scanf(''%d'',&a[i][j]);
for(i=0;i<m;i++)
for(j=0;j<q;j++)
{
  c[i][j]=0;
  for(k=0;k<n;k++)
 c[i][j]=c[i][j]+a[i][k]*b[k][j];
}
printf(''\n the result is'');
for(i=0;i<m;i++)
{
printf(''\n'');
for(j=0;j<q;j++)
printf(''\t%d'',c[i][j]);
}
getch( );
}
```

#3) write a c program to count characters, words, lines in a text?

solution:

```
#include<stdio.h>
#include<conio.h>
main( )
 {
  int  c=0,w=0,l=0;
char  ch;
clrscr( );
printf(''enter text(press ctrl + z to stop):');
while(ch=getchar( )!=EOF)
{
if(ch=='\n')
{
l++;
w++;
}
else if(ch==' ')
w++;
else
c++;
```

```
}
printf(''\n no of words=%d'',w);
printf(''\n no of lines=%d'',l);
printf(''\n no of characters=%d'',c);
getch( );
}
```

#4) write a c program to sort the strings?

solution:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main( )
 {
  int  i, n,j;
char  x[10][10];temp[10];
clrscr( );
printf("enter texthow mqny strings ");
scanf("%d",&n);
printf("enter strings");
for(i=0;i<n;i++)
scanf("%s",&x[i]);
for(i=0;i<n-1;i++)
for(j=i+1;j<n;j++)
if(strcmp(x[i],x[j])>0)
{
strcpy(temp,x[i]);
strcpy(x[i],x[j]);
strcpy(x[j],x[i]);
}
for(i=0;i<n;i++)
printf("\n %s",x[i]);
getch( );
}
```

#5) write a c program that uses recursion to solve the towers of Hanoi prob.

Solution :

```
#include<stdio.h>
#include<conio.h>
void transfer(int n,char from,char to,char temp);
main( )
{
int n;
clrscr( );
printf(‘’ how many dosks’’);
scanf(‘’%d’’,&n);
transfer(n,’l’,’r’,’c’);
getch( );
}
void transfer(int n,char from, char to,char temp)
{
if(n>0)
{
transfer(n-1,from,temp,to);
printf(‘’\n move disk %d from %c to %c’’,n,from,to);
transfer(n-1,temp,to,from);
}
}
```

#6) write a c program to print the largest and smallest elements from array.

Solution:

```
#include<stdio.h>
#include<conio.h>
main( )
{
int  a[10],i,n,max,min;
clrscr( );
printf(‘’enter how many value’’);
scanf(‘’%d’’,&n);
printf(‘’enter how array value’’);
for(i=0;i<n;i++)
scanf(‘’%d’’,&a[i]);
max=a[0];
min=a[0];
for(i=1;i<n;i++)
{
if(a[i]>max)
max=a[i];
```

```
if(a[i]<min)
min=a[i];
}
printf(''\n max=%d'',max);
printf(''\n min=%d'',min);
getxh( );
}
```

#7) write a c program to transpose the given matrix.

Solution:

```
#include<stdio.h>
#include<conio.h>
main( )
{
  int  a[5][5],b[5][5],i,j,m,n;
clrscr( );
printf(''enter order of  matrix'');
scanf(''%d%d'',&m,&n);
printf(''enter values for mtrix'');
for(i=0;i<m;i++)
for(j=0;j<n;j++)
{
scanf(''%d'',&a[I][j]);
b[j][i]=a[i][j];
}
printf(''\n the transpose result is'');
for(i=0;i<m;i++)
{
printf(''\n'');
for(j=0;j<n;j++)
printf(''\t%d'',c[i][j]);
}
getch( );
}
```

#8) write a c program to print the sum of the digits of given no. using recursive method.

```
#include<stdio.h>
#include<conio.h>
int sum(int);
main( )
{
int  n;
clrscr( );
printf(''enter a no'');
scanf(''%d'',&n);
printf(''\n sum=%d'',sum(n));
getch( );
}
int  sum(int  x)
{
if(x==0)
return 0;
else
return  x%10+sum(x/10);
}
```

#9) write a c program to find the no. of vowels and consonants in a given string.

Solution:

```
#include<stdio.h>
#include<conio.h>
main( )
{
int  v=0,c=0,i;
char  s[200];
clrscr( );
printf('' enter a string'');
scanf(''%s'',&s);
for(I=0;s[i];i++)
{
if(s[i]=='a'||s[i]=='e'||s[i]=='i'||s[i]=='o'||s[i]=='u')
v++;
else
c++;
printf(''\n no of vowels=%d,v);
printf(''\n no of consonants=%d,c);
getch( );
}
```

#10)write a c program to find out area, circumference of circle using macros.
Solution:#

```
#include<stdio.h>
#include<conio.h>
#define pi 3.14
main( )
{
float r;
clrscr( );
printf(‘’enter radius of circle’’);
scanf(‘’%f’’,&r);
printf(“ \n area=%f”,pi*r*r);
printf(“ \ncf=%f”,2*pi*r);
getch( );
}
```

#11)write a c program to swap two values using call by value and call by reference .

solution(call by value):

```
#include<stdio.h>
#include<conio.h>
void swap(int,int)
main( )
{
int a=10,b=20;
clrscr( );
swap(a,b);
printf(“ \n a=%d, b=%d”,a,b);
getch( );
}
void swap(int x, int y)
{
int temp;
temp=x;
x=y;
y=temp;
}
```

using call by reference :

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

#12) write a c program to create a student database using structures read the students name, number and three subject marks and display the student details with total, average and result;

```
#include<stdio.h>
#inxlude<conio.h>
#include<string.h>
struct  student
{
char  name[20],res[20];
int  r,m1,m2,m3,tot,avg;
};
main( )
{
struct student  s[10];
int  i,j,n;
clrscr( );
printf("enter how many student records u want");
scanf("%d",&n);
for(i=0;i<n;i++)
{
```

```
printf(" \nenter name, no of %dstudet",i);
scanf("%s%d",&s[i].name,s[i].r);
printf(" \n enter 3 sub marks");
scanf("%d%d%d",&s[i].m1,&s[i].m2,&s[i].m3);
s[i].tot=s[i].m1+s[i].m2+s[i].m3;
s[i].avg=s[i].tot/3;
if((si].m1>=35)&&([i].m2>=35)&&s[i].m3>=35))
{
if(s[i].avg>=70)
strcpy(s[i].res,"distiction");
else if(([i].avg>=6)&&s[I].avg<70)
strcpy(s[i].res,"first");
else if(([i].avg>=5)&&s[I].avg<60)
strcpy(s[i].res,"second");
else
strcpy(s[i].res,"pass");
}
else
strcpy(s[i].res,"fail");
}
printf(" \n rno \t name \t sub1 \tsub2 \tsub3 \t tot \t avg \t result");
for(i=0;i<n;i++)
printf(" \n%d \t%s \t%d \t%d \t%d \t%d \t%d \t%s",s[i].r,s[i].name,s[i].m1,
s[i].m2,s[i].m3,s[i].tot,s[i].avg,s[i].res);
getch( );
}
```

#13) Write a cprogram to find the GCD of two value?

Solution:

```
#include<stdio.h>
#include<conio.h>
main( )
{
int a,b;
clrscr( );
printf("enter two values");
scanf("%d%d",&a,&b);
while(a!=b)
{
if(a>b)
a=a-b;
else
b=b-a;
}
```

```
printf("\n gcd=%d",a);
getch( );
}
```

#14_write a c program to demonstrate math functions of c?

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
main( )
{
int a=-1;
double x=5,y=3,p=9;
clrscr( );
printf("\n absolute value of a=%d",abs(a));
printf("\n power=%f",pow(x,y));
printf("\n sqrt=%f",sqrt(p));
printf("\n floor=%f",floor(3.4));
printf("\n ceil=%f",ceil(3.4));
getch( );
}
```

#15) write a c program to find the no. of days in a month using enumerated data type?

Solution:

```
#include<stdio.h>
#include<conio.h>
main( )
{
enum  months{jan=31,feb=28,mar=31,apr=30,may=31,jun=30,jul=31,aug=31,
              sep=30,oct=31,nov=30,dec=31};
enum months m1,m2;
m1=apr;
m2=sep;

printf("\n days in apr=%d",m1);
printf("\n days in sep=%d",m2);
getch( );
}
```