www.gdcnarasannapeta.ac.in
www.narasannapeta.jkc@gmail.com

GOVERNMENT DEGREE COLLEGE

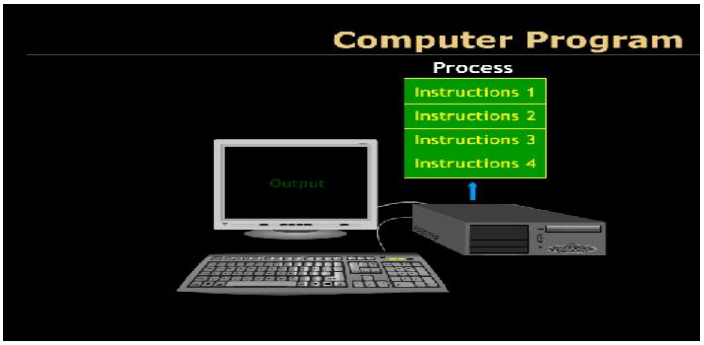
NARASANNAPETA-SRIKAKULAM DIST.-532421.
Accredited by NAAC 'B' Grade
(Affiliated to DR.B.R.Ambedkar University)

STUDY MATERIAL

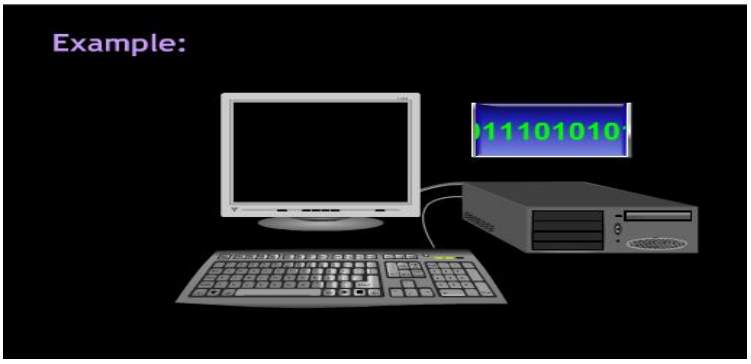
Detartment of Computer Science

Introduction To 'C' Programming

Program: A set of instructions used by computer to perform a particular task in a predefined manner.



Language: A language is a tool which facilitates communication. But a computer understands only binary code which consists of only 2 digits 0's & 1's.

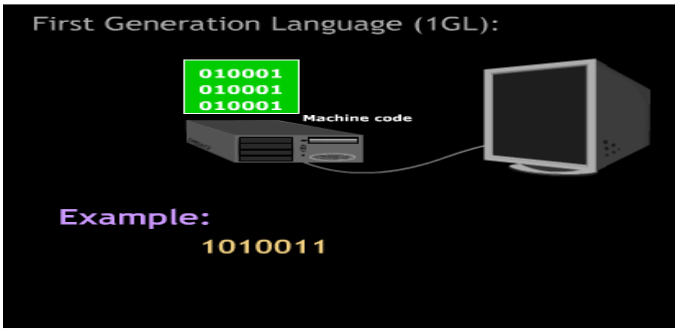


A language is used to write instructions that can be translated into machine understandable form and then executed by computer are called language.

Generations of Languages: Languages are categorized into 5 generations. They are

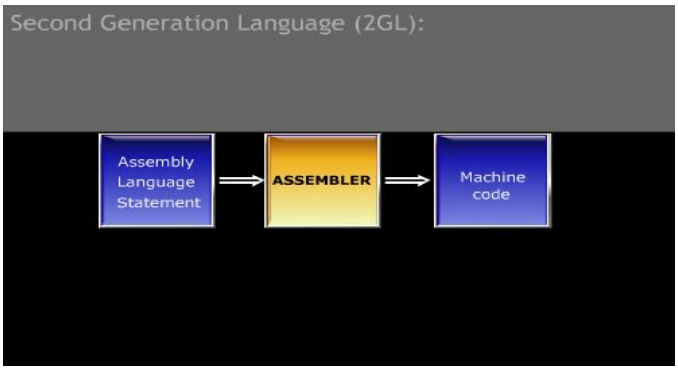
- **First generation language(1GL)**
- **Second generation language(2GL)**
- **Third generation language(3GL)**
- **Fourth generation language(4GL)**
- **Fifth generation language(5GL)**

First generation language (1GL): It is a machine level language or low level language. It consists of binary code i.e. 0's & 1's.

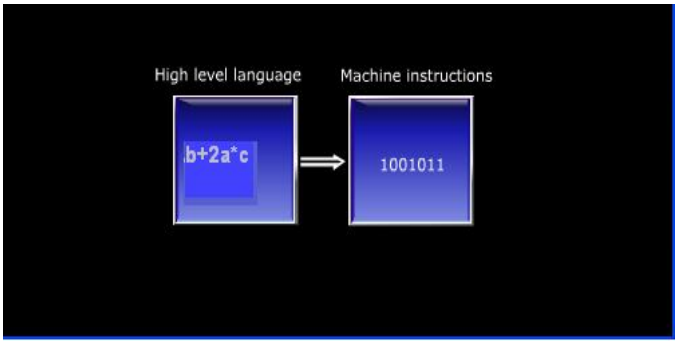


Second generation language (2GL): It is assemble level language which uses mnemonics instead of 0's & 1's. It can be read easy and fairly by a human. It can convert to machine level in order to run on a computer.
E.g.: Add 12, 8

This instruction means add 12 and 8.



Assemble performs more or isomorphic translation from mnemonics to machine statements.



Third generation language (3GL): It is a "high level" programming language designed to be easier for human to understand and write code.

A program developed using 3Gl is highly procedural in that each and every step constituting a program has to be specified clearly by the programmer using constructs of language and conforming to the syntax of language

For example b=c+2*d

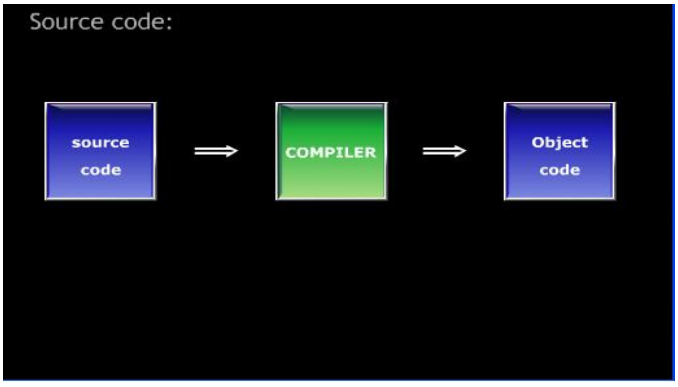
E.g. c, c++, java …

It is important to understand that a computer language defines the nature of a program and not the way that the program will be executed. To generate methods by which a program can be executed by 2 methods

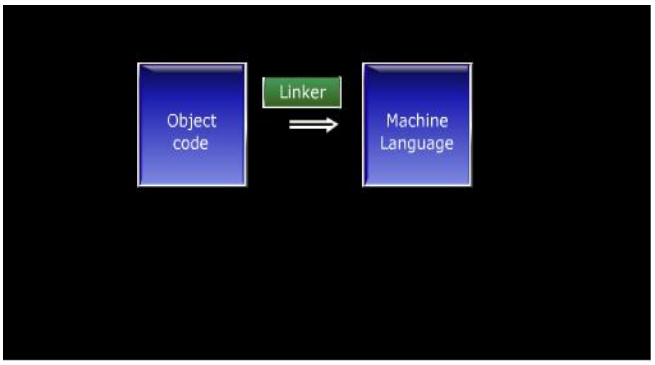
- **→ It can be compiled**
- **→ It can be interpreted**

Before that we should know about source code and object code.

Source Code: Program written in particular program language in its original form. Word source differentiate code from various other forms
E.g. Object code and Executable code

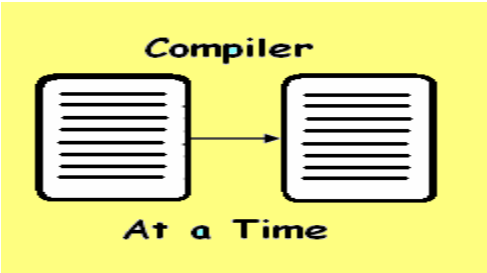


Object Code: The code produced by a compiler.



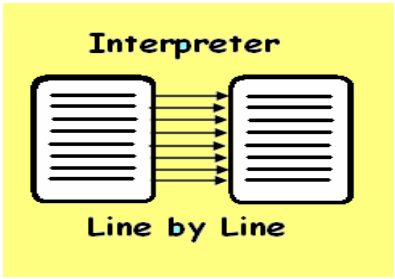
Compiler:-

A compiler reads the entire program and converts it to the object code. It provides errors not of one line but errors of the entire program. Only error-free programs are executed. It consumes little time for converting a source program to an object program.



Interpreter:-

An Interpreter reads only one line of a source program at a time and converts it to the object codes. In case there are errors, the same will be indicated instantly. The disadvantage is that it consumes more time for converting a source program to an object program.



Fourth generation language (4GL): It is a non-procedural language such as

e.g. Select * from EMP;

- → **SQL(Structured query Language)**
- → **Query and manipulate a relational database in a good example of 4GL.**

Fifth generation language (5GL):

- → **Solving problems using constraints given to the program.**
- → **Constraint based and logic programming languages.**
- → **Some declarative languages**
- → **These are used mainly artificial intelligence research such as**
  - o **Prolog**
  - o **Ops5**
  - o **Mercury**
- → **These are built upon LISP**
  - o **ICAD**
  - o **From languages such as KL-One.**

Evolution of 'C':

Dennis Ritchie develops C in 1970's at AT&T Bell labs; Murrey Hill, New Jersey USA.It is developed from older language BCPL.

- **BCPL is developed by Martin Richards.**
- **Ken Thomson develops B.**

C's place among the generations of programming language

- **Powerful low-level features of second generation languages like pointers, bit manipulations etc...**
- **High-level languages belonging to the category of 3GLs.The language become available on a very wide range of platforms, from embedded microcontrollers to super computers.**
- **C exhibits all the features of 3GL such as**
  - **Block structured code**
    - **A unit code in written as a function block**
    - **A function in turn comprises various constructs such as conditional constraints iterative constructs etc... which in turn are written as a block of code.**

High Level Language: - A high level programming language that is more users friendly and memory access in platform independent.

Low Level Language: - A low level programming language is more machine specific and processor operations such as memory access is directly process dependent.

Why Use 'C':

C is success based on purely practically considerations

- **Practicality**
- **Operation system,compiler,embedded software application software(financial accounting, inventory control)**
- **It is a industry standard**

Programming constructs that is provides to the programmer (such as pointer) are essential to know

Examples of the use of C:

→ O/S
→ Language compiler
→ Assemblers
→ Text editor
→ Print spoolers
→ Network driver
→ Modern programs
→ Data bases
→ Language interpreter

Features of C: -

- 'C' is a portable study software.
- 'C' is case sensitive language.
- 'C' is function oriented structured language.
- 'C' supports recursion.
- 'C' has rich set of operator.
- 'C' facilitator to create new user defined data types.
- 'C' supports preprocessing.
- 'C' supports Dynamic memory allocation.
- We can perform low-level operations very easy.
- Very easy to handle pointers.

Character Set: -

- Alphabets: A to Z & a to z.
- Digits: 0 to 9.
- Special Characters.

General Structure of a 'C' programming: -

- /* Documentation: Describe about Programming */.
- Preprocessing directives.
- Global variable declarations.
- Main function.
- Local variable declaration.
- Valid 'C' instructions.
- User defined function definition.

```
main ()
{Begin
------------
------------Statements
End}
```

Every C program contains a number of building blocks known as functions. Each function of it performs a task independently.

1. Include header file section:-
   A C program depends upon some header files for function definition that are used in the program. Each header file by default has an extension ".h". This file should be using #include directive. #include <stdio.h) or #include "stdio.h"

2. Global Declaration:-
   This section declares some variables that are used in more than one function. These variables are known as Global Variables. This section must be used outside of all the functions.

3. Function Main:-
   Every program written in C language must contain the main( ) function. The Execution of the program always begins with the function main( ). The program execution starts from the opening brace({) and closing brace (}). Between these two braces, the programmer should declare declaration and executable parts.

4. Declaration Part:-
   This part declares all variables that are used in the executable part .Initialization means providing initial value to the variables.

5. Executable Part:-
   This part contains the statements following the declaration of the variables.

6. User defined Functions:-
   The functions defined by the user.

7. Comments:-

Comments are not necessary in the program. To understand the flow of the program. The compiler does not execute comments.

- ✓ **Identifiers: -The name given to the program element is called identifier (I).**
- ✓ **Variable: -A variable is an identifier whose value alters during program execution.**
- ✓ **Constant: - A variable is an identifier whose value does not alter during program execution.**
- ✓ **Keyword: - These are reserve words, which are predefined meaning in 'C' language. There are 32 keywords in 'C'.**

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Rules for naming a variable: -

- → **It should be a combination of alphabets & digits.**
- → **The first letter should be an alphabet.**
- → **The compiler recognizes only eight characters for a variable even though we can specify up to 40 characters.**
- → **No special characters except '_' are allowed.**
- → **Keywords cannot be used as variable name.**

Instructions:- It is a combination of variables, keywords, constants & operators to achieve a task. These are classified into

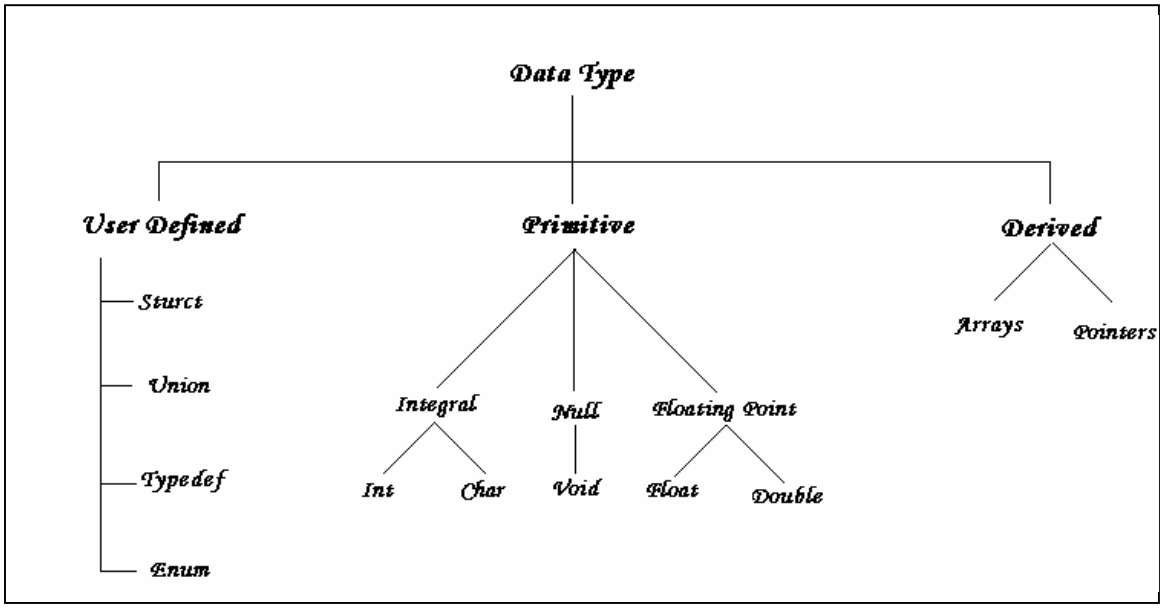
- Type declaration instructions
- IO instructions
- Arithmetic instruction
- Control instruction

Type Declaration Instructions: -These are the instructions used to declare variables & constants.

→ **Constant Declaration:** - const data type variable_name=value;

→ **Variable Declaration:** -data type variable_name;

Data type: -Data type specifies the data that the variable can hold. It specifies the size of memory of that variable.



Sno	Data type	size in bytes	Range
1	int	2	-32768 to 32767
2	char	1	-128 to 127
3	float	4	$-3.4e^{38}$ to $3.4e^{38}$
4	double	8	$-1.7e^{308}$ to $1.7e^{308}$

Void:- If the function is returning nothing we will specify it 'void'. To declare generic pointers we use void.

Note:

→ All the variables must be declared before processing.
→ Variable declaration must be placed at the beginning of the function.

Qualifiers: - These are used to

- Change the range of a variable.
- Change the memory size of a variable.

Input-output instructions: -These are used to perform input output operations. These are categorized into

- ✓ **Console IO: -**These are used to handle IO with console devices.
- ✓ **Disk IO: -** These are used to handle IO with HDD & FDD.
- ✓ **Port IO: -** These are used to handle hardware ports.

Console IO Functions: - These are used to handle IO with console devices. These are classified into two types

- Formatted console input function
- Formatted console output function

1→ **Formatted console input function: -** (scanf ())

- These are used to accept the data for different variables through keyboard
- This function is defined in the header file <stdio.h>

General Form: - scanf ("formate specifier", list of variable addresses);

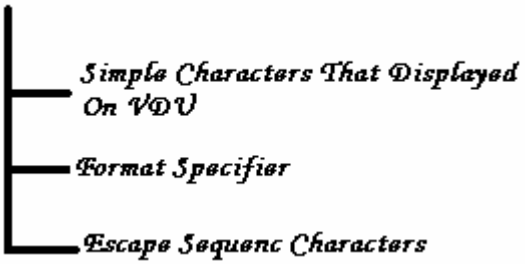
Formate Specifier: - It is used to specify what kind of data we are inputting through keyboard.

Format specifier	Excepted Input
%d	integers (which has no fractional part)
%f	float (which has fractional part)
%c	characters
%s	string
%i	a decimal integer
%l	A long decimal integer
%o	An octal integer
%X	a hexa decimal integer
%u	an unsigned integer
%D or %ld	long integer
%e, %E	floating point exponential
%X	long hexa decimal integer
%O	Long octal integer
%p	pointer
% [...]	Scan set format specifier
%lf	double
%Lf	long double

2→Formatted console output function :-(printf ())

- These are used to perform output operation to console output device VDU.
- This function is defined in the header file stdio.h

General form: -printf ("console string", list of variables)



Escape sequence characters: -These are used to escape from the normal sequence in the output to create a new sequence.

Escape sequence characters	Purpose
\n	the cursor jumps to its new line.
\t	the cursor lefts 6 / 8 spaces in the same line
\v	vertical tab

\a	alarm or beep
\f	form feed
\\	Prints a
\"	Prints a"
\r	carriage return
\'	Prints a'
\b	backspace

Arithmetic Instructions: -An expression is a combination of data (constants, variables) interconnected with operators.

Operator: - Which can operate its associated operands.
Operator is classified into the following.

- → Arithmetical operators.
- → Relational operators.
- → Logical operators.
- → Assignment operators.
- → Increment /Decrement operators.
- → Conditional operators.
- → Bitwise operators.
- → Special operators

OPERATOR	PURPOSE	OTHER INFORMATION
+	Addition	both binary and unary.
-	Subtraction	both binary and unary.
*	Multiplication	binary operator.
/	Division	binary operator.
%	Modulo division (Or) Remainder operator	binary operator.
<, >, <=, >=	These four are binary operator	
!=	Not equal operator	binary operator
==	Equivalence operator	binary operator
&&	Logical and	binary operator
||	Logical or	unary operator
!	Logical not	binary operator
=	Assignment	binary operator
+=	Shorthand addition	binary operator
-=, *=, /=, %=	All these shorthand	binary operator

++	Increment operator	unary operator Increments the Operand by 1
--	Decrement operator	unary operator Decrements the operand by 1

Conditional operator: -This is also called as ternary operator.

General form: (expression)? True Exp: False Exp;

Bitwise operators: -

These are used to perform bit manipulations

&	Bitwise and	binary operator
|	Bitwise or	binary operator
~ (Tilda)	One's compliment	unary operator
^ (Circumflex)	Bitwise xor	binary operator
<<	Shift left	binary operator
>>	Shift right	binary operator

Special Operators: -

- ✓ **Sizeof () Operator: - It returns the number of bytes required for a specified a data type or variable.**

  General Form: - sizeof (int); or int a; sizeof (a);

- ✓ **Address operator (&): - it is used to access the address of a memory variable. This is a unary operator.**

  General Form: - printf ("%p", &i);

- ✓ **Memory reference Operator (*): - It is used to access the data in the specified memory location. This is a unary operator.**

  General Form: - printf ("%d",*(&i));

- ✓ **Comma Operator (,): - This used to combine related statements into a single statement.**

  General Form: - datatype var1,var2…var n;

  e.g.
  ```
  int a, b, c;
  a=10, b=20;
  c=a+b;
  ```

- ✓ **Data Typecasting: - It is used to convert one type of data type into another specified data type element.**

General Form: - (datatype) data item;
 # -Stringizing operator, ##-Token packing operator,
[ ]-Array subscription operator, ( )-Function call operator.

Operator Precedence: -

Operator	Description	Associativity
.	Structure operator	left to right
.→	Pointer to structure operator	"
[]	Function call operator	"
()	Array operator	"
+	Unary operator	"
-	Unary operator	"
++	Increment operator	"
--	Decrement operator	"
!	Not	"
~	One's complement operator	"
*	Memory reference operator	"
&	Address operator	"
,	Comma operator	"
Sizeof ()	sizeof operator	right to left
(Type)	type casting	"
*	Multiplication	left to right
/	Division	"
%	modulo division	"
+	Addition	"
-	Subtraction	"
<<	Shift left	"
>>	Shift right	"

<	----------	"
>	----------	"
< =	----------	"
> =	----------	"
=	Equivalence to	"
!	Not equal to	"
&	Bitwise And	"
|	Bitwise Or	"
^	Circumflex	"
&&	Logical And	"
||	Logical or	"
?:	Conditional operator	"
+ =	Shorthand operator	right to left
- =	"	"
/ =	"	"
% =	Shorthand operator	right to left
* =	"	"
< < =	"	"
> > =	"	"
& =	"	"
| =	"	"

Exercise

1) Write a sample 'C' Program

```
#include<stdio.h>
main()
{
clrscr();/*clears the screen*/
printf("This is my sample 'C' program");
getch();
}
```

2) Write a program for addition of 2 no's

```
#include<stdio.h>
main()
{
int a,b,c;
clrscr();
printf("enter a & b values");
```

```
scanf("%d%d",&a,&b);
c=a+b;
printf("\n The value of A:%d",a);
printf("\n The value of B:%d",b);
printf("\n The Sum Of %d+%d=%d",a,b,c);
getch();
}
```

3) Write a program for the demonstration of constant variables

```
#include<stdio.h>
main()
{
int m,h;
float pe;
const float g=9.8;
clrscr();
printf("Enter mass & height");
scanf("%d%d",&m,&h);
pe=g*m*h;
printf("\n Potential Energy=%.2f",pe);
getch();
}
```

4) Write a program to demonstrate global variable declaration

```
const float pi=3.14;
float peri(int);
main()
{
int r;
float area,per;
clrscr();
printf("Enter the radius:");
scanf("%d",&r);
area=pi*r*r;
per=peri(r);
printf("\n Area of circle=%.2f",area);
printf("\n Perimeter of circle=%.2f",per);
getch();
}
float peri(int x)
```

```
{
return 2*x*pi;
}
```

5) Write a program to demonstrate swapping of 2 No's

```
#include<stdio.h>
#include<conio.h>
main()
{
int a,b,c;
clrscr();
printf("Enter a & b values");
scanf("%d%d",&a,&b);
printf("\n Before Swapping");
printf("\n A=%d \t B=%d",a,b);
c=a;
a=b;
b=c;
printf("\n After Swapping");
printf("\n A=%d \t B=%d",a,b);
getch();
}
```

(Or)

```
#include<stdio.h>
#include<conio.h>
main()
{
int a,b;
clrscr();
printf("Enter a & b values");
scanf("%d%d",&a,&b);
printf("\n Before Swapping");
printf("\n A=%d \t B=%d",a,b);
a=a+b;
b=a-b;
a=a-b;
printf("\n After Swapping");
printf("\n A=%d \t B=%d",a,b);
getch();
}
```

6) Write a program to demonstrate Increment & Decrement operators

```
main()
{
int i=1;
clrscr();
printf("\n++I=%d",++i);
printf("\nI++=%d",i++);
printf("\n--I=%d",--i);
printf("\nI--=%d",i--);
printf("\nI=%d",i);
getch();
}
```

7) Write a program to demonstrate Conditional operator

```
main()
{
int a,b;
clrscr();
printf("Enter A & B values");
scanf("%d%d",&a,&b);
(a>b)?printf("A(%d) is big",a):printf("B(%d) is big",b);
getch();
}
```

Control Instructions

These are used to maintain the flow of control of a program. These are categorized into 3 types. They are

1→Decision Control Statements: - These are the statements depend upon conditions.

Simple if statement: -

General Form: -if (expression)
Statement/compound Statement

- **→ The expression should be in parenthesis.**
- **→ The expression returns 0(false) or non-zero (true).**
- **→ If the expression is zero non-zero (true) then statement under "If" are executed.**
- **→ If the expression is zero (false) then the statement under "If" are ignored.**
- **→ If more than one statement present under "If" statements then they have to written within {-----}**

If …Else Statement: -

General Form: -if (expression)
Statements/Compound Statements;
Else
Statements/compound Statements;

- **→ If the expression is non-zero then the statements under "If" are executed.**
- **→ If the expression is zero then the statements in "Else" part are executed.**

Nested If Statement: -We can place any number of if-else statements with in an "If" statement. This is called nested-if statement.

General Form: -if (expression)
{
if (expression)
{

```
                    if (expression)
                        Statements;
                    Else
                        Statements;
                }
            }
```

2→Repetitive (or) Loop Control statements:-Many programs require a group of statements have to be executed until a condition is satisfied. This is called looping.

The loop control statements are classified into: -

1. While statements: -

General form:

```
While (exp)
{
statement/compound statement;
}
```

- **The statement under while executed repeatedly until the expression returns non-zero (true).**
- **Once the expression is zero (true) the next statement of while.**
- **This is also called "Entry controlled loop" statement.**

2. Do-While statement: -

General form:

```
Do
{
Statement;
} While (exp);
```

- **The statements under do are executed first, later it checks for the validity of the expression.**
- **If the expression is non-zero, again the statements under do are executed.**
- **This process is continued till the expression remains true.**
- **Once it becomes falls the control goes to the next statement of the while statement.**
- **This is exit controlled loop structure.**

3. **For statement: -**

General form:

for(Initialization; Test condition; update statement)

- **The statement in initialization part is executed first, later it checks the validity of test expression.**
- **If test expression is true the statements under "For" are executed. And the control goes to pupation part.**
- **After that, the text expression is verified.**
- **This process will be continued to the test condition is non-zero. Once the test condition is z is non-zero. Once the test condition is zero control goes to the next statement of "For" statement.**
- **This is also called "Entry Controlled Loop" statement.**

3→**Sequential Control Instructions: - These statements are used to alter the normal sequence of execution. In general it is used along with the "If" statement.**

A→Goto Statement: -

General Form: - goto label;

Here label is a valid identifier. Label specifies to which location the control to be transferred. As the target label should appear as:

Label: statements;

Eg:

```
Top:
If (expression)
Goto top;
```

B→**Break Statement: - It is used to terminate the loop through the4 expression in loop structure is true**

General Form: - break;

C→ **Continue statement: -**

→ **Continue statement steps the current iteration and process with next iteration of a loop statement.**

General form: - Continue;

D→ Case control structure or statement: -

→ If we want to take one decision among several decisions, we may use 'Switch case statement'.

General form: -

```
Switch (integral expression)
{
Case case constant 1: statement/compound
statement;
                Break;
Case case constant 2: statement/compound
statement;
                Break;

Case case constant 3: statement/compound
statement;
                Break;
 .              .
 .              .
 .              .
Case case constant n: statement/compound
statement;
                Break;
Default: statement/compound statement;
}
```

→ Hence, the integral expression must return either int or char.

→ The result of integral expression is matched for equality with the case constant if match is found; the statement block corresponding to the case constant is executed.

→ There should not be more than one case constant with in a single switch statement.

→ Once the match is found, the statements will be executed till it reaches either 'break' or end of the switch statement.

→ If match is not found then the statement under 'Default' are executed.

→ "C" allows totally 257(256+1 default) case constant.

Exercise

1. write a sample simple if program

```
#include<stdio.h>
#include<conio.h>
main()
{
      int a,b,big;
      clrscr();
      printf("ENTER 2 NO : ");
      scanf("%d%d",&a,&b);
      big = a;

      if(b>big)
        {
         big = b;
        }
      printf("BIG = %d",big);
      getch();
}
```

2. write a program to demonstrate Even No or not

```
#include<stdio.h>
#include<conio.h>
main()
{
      int n;
      clrscr();
      printf("ENTER A NO : ");
      scanf("%d",&n);
      if(n%2==0)
      printf("%d IS EVEN",n);
      else
      printf("%d IS ODD",n);
      getch();
}
```

3. write a program to demonstrate nested if-else statement

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

```
main()
{
      int x,y;
      clrscr();
      printf("ENTER 2 NO : ");
      scanf("%d%d",&x,&y);
      if(x==y)
        {
        printf("BOTH ARE EQUAL ");
        }
      else if(x>y)
         {
         printf("BIG NO = %d",x);
         }
      else
         {
         printf("BIG NO = %d",y);
         }
      getch();
}
```

4. write a program to demonstrate if-else ladder

```
#include<stdio.h>
#include<conio.h>
main()
{
      char gender;
      int age;
      clrscr();
      printf("ENTER GENDER [M/F] : ");
      scanf("%c",&gender);
      gender = tolower(gender);
      printf("ENTER AGE : ");
      scanf("%d",&age);

if(gender == 'f')
 if(age>=21)
   printf("SEARCH FOR BRIDE GROOM");
 else
   printf("Miss wait for %d years",21-age);
else
 if(gender == 'm')
   if(age>=23)
```

```
      printf("SEARCH FOR BRIDE");
    else
      printf("Mr wait for %d years",23-age);
 else
      printf("CHECK GENDER ONCE");

getch();
}
```

5. write a program to demonstrate biggest of 3 No's

```
#include<stdio.h>
#include<conio.h>
void main()
{
      int a,b,c;
      clrscr();
      printf("ENTER 3 NO : ");
      scanf("%d%d%d",&a,&b,&c);
      if(a>b)
            if(a>c)
            printf("%d is big",a);
            else
            printf("%d is big",c);
      else
            if(b>c)
            printf("%d is big",b);
            else
            printf("%d is big",c);
      getch();
}
```

6. write a program to demonstrate if statements using logical operators

```
#include<stdio.h>
#include<conio.h>
main()
{
      int m1,m2,m3,t;
      float avg;
      clrscr();
      printf("ENTER 3 SUBJECTS MARKS : ");
      scanf("%d%d%d",&m1,&m2,&m3);
```

```
        t = m1+m2+m3;
        avg = (float)t/3;
        printf("\n SUBJECT 1 : %d",m1);
        printf("\n SUBJECT 2 : %d",m2);
        printf("\n SUBJECT 3 : %d",m3);
        printf("\n TOTAL    : %d",t);
        printf("\n AVERAGE  : %.2f",avg);
        if(m1>=35 && m2>=35 && m3>=35)
        {
        printf("\n CLASS    : ");
        if(avg>=60) printf("FIRST CLASS");
        else if(avg>=50) printf("SECOND CLASS");
        else if(avg>=35) printf("THIRD CLASS");
        }
        else printf("\n FAIL");
        getch();
}
```

7. write a program to calculate power bill

```
#include<stdio.h>
#include<conio.h>
main()
{
int o,n,u,r,amt;
clrscr();
gotoxy(15,5);
printf("ENTER OLD READING : ");
gotoxy(15,6);
printf("ENTER NEW READING : ");
gotoxy(35,5);
scanf("%d",&o);
gotoxy(35,6);
scanf("%d",&n);
      u = n-o;
      if(u>=500) r = 7;
      else if(u>=300) r = 5;
      else if(u>=100) r = 3;
      else r = 1;

      amt = r*u;

gotoxy(20,8);  printf(" OLD READING : %d",o);
gotoxy(20,9);  printf(" NEW READING : %d",n);
```

```
gotoxy(20,10); printf(" UNITS     : %d",u);
gotoxy(20,11); printf(" RATE      : %d",r);
gotoxy(20,12); printf(" AMOUNT    : %d",amt);
getch();
}
```

8. write a program for to verify the given character is vowel or consonant

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
main()
{
char ch;
clrscr();
printf("Enter a Character : ");
scanf("%c",&ch);
ch = tolower(ch);
if(ch=='a' || ch =='e' || ch=='i' || ch == 'o' ||ch == 'u')
printf("%c is VOWEL",ch);
else
printf("%c is CONSONENT",ch);
getch();
}
```

9. write a program for given year is leap year or not

```
#include<stdio.h>
#include<conio.h>
main()
{
      int year;
      clrscr();
      printf("ENTER YEAR : ");
      scanf("%d",&year);
      if(year%4== 0 && year%100!=0 || year%400 ==0)
      printf("%d LEAP YEAR",year);
      else
      printf("%d NOT LEAP YEAR ",year);
      getch();
}
```

10. write a program to verify the given roots are real or not

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
main()
{
      int a,b,c;
      float d,r1,r2;
      clrscr();
      printf("ENTER a,b,c Values : ");
      scanf("%d%d%d",&a,&b,&c);
      d = b*b - 4*a*c;
      r1 = (-b + sqrt(d))/ 2*a;
      r2 = (-b - sqrt(d))/ 2*a;
      printf("\n ROOT 1 = %f",r1);
      printf("\n ROOT 2 = %f",r2);
      printf("\n D    = %f",d);
if(d==0)  printf("\n ROOTS ARE REAL & EQUAL");
else if(d>0) printf("\n ROOTS ARE REAL & IMAGINARY");
else  printf("\n ROOTS ARE UNEQUAL & IMAGINARY");
getch();
}
```

11. write a program to demonstrate while loop

```
#include<stdio.h>
#include<conio.h>
main()
{
int i,n;
clrscr();
printf("Enter a No:");
scanf("%d",&n);
i=1;
while(i<=n)
{
printf("\t I=%d",i);
i++;
}
getch();
}
```

12. write a program for sum of N-natural No's

```
#include<stdio.h>
#include<conio.h>
main()
{
int i=1,n,s=0;
clrscr();
printf("Enter a No:");
scanf("%d",&n);
while(i<=n)
{
printf("\n I=%d",i);
s=s+i;
i++;
}
printf("\n Sum of %d No's=%d",n,s);
getch();
}
```

13. write a program for reverse a No

```
#include<stdio.h>
#include<conio.h>
main()
{
int r,n,s=0;
clrscr();
printf("Enter a No:");
scanf("%d",&n);
while(n!=0)
{
r=n%10;
s=s*10+r;
n=n/10;
}
printf("\n Reverse No=%d",s);
getch();
}
```

14. write a program for palindrome No

```
#include<stdio.h>
#include<conio.h>
main()
{
int r,n,s=0,m;
clrscr();
printf("Enter a No:");
scanf("%d",&n);
m=n;
while(n!=0)
{
r=n%10;
s=s*10+r;
n=n/10;
}
if(m==s)
printf("\n Given No %d is palindrome",m);
else
printf("\n Given No %d is not a palindrome",m);
getch();
}
```

15. write a program for Armstrong No

```
#include<stdio.h>
#include<conio.h>
main()
{
int r,n,s=0,m;
clrscr();
printf("Enter a No:");
scanf("%d",&n);
m=n;
while(n!=0)
{
r=n%10;
s=s+r*r*r;
n=n/10;
}
if(m==s)
printf("\n Given No %d is Armstrong",m);
else
printf("\n Given No %d is not a Armstrong",m);
getch();
```

```
}
```

16. write a program for strong No

```
#include<stdio.h>
#include<conio.h>
main()
{
int r,n,s=0,m,i,f;
clrscr();
printf("Enter a No:");
scanf("%d",&n);
m=n;
while(n!=0)
{
r=n%10;
f=1;
i=1;
while(i<=r)
{
f=f*i;
i++;
}
s=s+f;
n=n/10;
}
if(m==s)
printf("\n Given No %d is strong",m);
else
printf("\n Given No %d is not a strong",m);
getch();
}
```

17. write a program for sum digits of given No

```
#include<stdio.h>
#include<conio.h>
main()
{
int r,n,s=0;
clrscr();
printf("Enter a No:");
scanf("%d",&n);
while(n!=0)
```

```
{
r=n%10;
s=s+r;
n=n/10;
}
printf("\n Sum of digits of given No=%d",s);
getch();
}
```

18. write a program to demonstrate do-while loop

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

19. write a program for sum of given No until users choice is no

```
#include<stdio.h>
#include<conio.h>
main()
      {
            char choice;
            int n,s=0;
            clrscr();
            do
            {
                  printf("ENTER A NO : ");
                  scanf("%d",&n);
                  s = s + n;
                  fflush(stdin);
                  printf("DO U WANT 2 CONTINUE[Y/N]?");
                  scanf("%c",&choice);
                  choice=tolower(choice);
```

```
        }while(choice=='y');
        printf("\n SUM : %d",s);
        getch();
        }
```

20. write a program for product the given no till the given no is zero

```
#include<stdio.h>
#include<conio.h>
 main()
{
      int n;
      long int p=1;
      clrscr();
      printf("\n\t\t ENTER 0 TO EXIT \n");
      do
      {
            printf("ENTER A NO : ");
            scanf("%d",&n);
            if(n==0) break;
            p=p*n;

      }while(n!=0);
      printf("\n PRODUCT : %ld",p);
      getch();
}
```

21. write a program for the biggest of given no till the given no is zero

```
#include<stdio.h>
#include<conio.h>
main()
{
      int n,big=0;
      clrscr();
      printf("\n\t\t ENTER 0 TO EXIT \n");
      do
      {
            printf("ENTER A NO : ");
            scanf("%d",&n);
```

```
            if(n>=big)
            big = n;

        }while(n!=0);
        printf("\n BIG : %d",big);
        getch();
    }
```

22. write a program for smallest of given no till the given no is zero

```
    #include<stdio.h>
    #include<conio.h>
    main()
    {
        int n,small;
        clrscr();
        printf("\n\t\t ENTER 0 TO EXIT \n");
        printf("ENTER A NO : ");
        scanf("%d",&n);
        small = n;
        do
        {
            printf("ENTER A NO : ");
            scanf("%d",&n);

            if(n!=0)
            if(n<=small)
            small = n;

        }while(n!=0);
        printf("\n SMALL : %d",small);
        getch();
    }
```

23. write a program for print N-natural No's

```
    #include<stdio.h>
    #include<conio.h>
    main()
    {
        int i,n;
        clrscr();
        printf("ENTER NUMBER : ");
```

```
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
              printf("\n I:%d",i);
        }
        getch();
  }
```

24. write a program for loop to print given string for the sepcified times

```
  #include<stdio.h>
  #include<conio.h>
  main()
  {
        int c,n;
        char name[25];
        clrscr();
        printf("ENTER NAME : ");
        gets(name);
        fflush(stdin);
        printf("HOW MANY TIMES ? ");
        scanf("%d",&n);
        for(c=1;c<=n;c=c+1)
        {
              printf("\n %s",name);
        }
        getch();
  }
```

25. write a program for multiplication table

```
  #include<stdio.h>
  main()
  {
        int c,n;
        clrscr();
        printf("ENTER NUMBER : ");
        scanf("%d",&n);
        for(c=1;c<=10;c=c+1)
        {
        printf("\n %3d x %3d = %3d",n,c,c*n);
        }
        getch();
```

```
}
```

26. write a program for factorial of a given No

```
#include<stdio.h>
main()
{
      int c,n,f=1;
      clrscr();
      printf("ENTER A NO : ");
      scanf("%d",&n);
      for(c=1;c<=n;c=c+1)
      {
      f=f*c;
      }
printf("\n FACTORIAL OF %d: %d",n,f);
      getch();
}
```

27. write a program for power of a given No

```
#include<stdio.h>
#include<conio.h>
main()
{
      int c,x,y,p=1;
      clrscr();
      printf("ENTER X,Y VALUES ");
      scanf("%d%d",&x,&y);
      for(c=1;c<=y;c=c+1)
      {
      p=p*x;
      }
      printf("%d POWER % d : %d",x,y,p);
      getch();
}
```

28. write a program for power/factorial

```
#include<stdio.h>
#include<conio.h>
main()
{
      int c,x,y,p=1,f=1;
```

```
        float r;
        clrscr();
        printf("ENTER X,Y VALUES ");
        scanf("%d%d",&x,&y);
        for(c=1;c<=y;c=c+1)
        {
        p = p*x;
        f = f*c;
        }
        r = (float)p/f;
        printf("\n %d POWER % d : %d",x,y,p);
        printf("\n %d FACTORIAL : %d",y,f);
        printf("\n   RESULT   : %.2f",r);
        getch();
}
```

29. write a program for squares of a N-natural No's

```
#include<stdio.h>
#include<conio.h>
main()
{
        int c,n,s=0;
        clrscr();
        printf("ENTER A NUMBER : ");
        scanf("%d",&n);
        for(c=1;c<=n;c=c+1)
        {
        s = s + c*c;
        }
        printf("\n SUM OF SQARES :  %d  ",s);
        getch();
}
```

30. write a program for factors of given No

```
#include<stdio.h>
#include<conio.h>
main()
{
        int n,c;
        clrscr();
        printf("ENTER A NO : ");
        scanf("%d",&n);
```

```
        printf("FACTORS OF %d",n);
        for(c=1;c<=n/2;c++)
        {
              if(n%c==0)
              printf("\n %d",c);
        }
        printf("\n %d",n);
        getch();
   }
```

31. write a program for No.of factors of given No

```
#include<stdio.h>
#include<conio.h>
main()
{
        int n,c,s=1;
        clrscr();
        printf("ENTER A NO : ");
        scanf("%d",&n);
        printf("FACTORS OF %d",n);
        for(c=1;c<=n/2;c++)
        {
              if(n%c==0)
              {
                    printf("\n %d",c);
                    s = s+1;
              }
        }
printf("\n %d",n);
printf("\n NO OF FACTORS FOR %d IS %d",n,s);
        getch();
}
```

32. write a program for given no is prime or not

```
#include<stdio.h>
#include<conio.h>
main()
{
        int n,i,count=0;
        clrscr();
        printf("ENTER A NO : ");
        scanf("%d",&n);
```

```
        printf("FACTORS OF %d",n);
        for(i=1;i<=n;i++)
        {
              if(n%i==0)
              {
                    printf("\n %d",i);
                    count++;
              }
        }
  if(count==2)
  printf("\n %d IS PRIME",n);
  else
  printf("\n %d IS NOT PRIME",n);
        getch();
  }
```

33. write a program for perfect No

```
  #include<stdio.h>
 #include<conio.h>
  main()
  {
        int n,s=0,c;
        clrscr();
        printf("ENTER A NO : ");
        scanf("%d",&n);
        for(c=1;c<=n/2;c=c+1)
        {
              if(n%c==0)
              printf("\nC:%d",c);
              s = s+c;
        }
        printf("\nSum:%d",s);
        if(n==s)
        printf("\n%d is PERFECT NO",n);
        else
        printf("\n%d  is NOT PERFECT NO",n);
        getch();
  }
```

34. write a program for find sum of any given "n" no

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

```
main()
{
      int no,c,s=0,n;
      clrscr();
      printf("HOW MANY NO? ");
      scanf("%d",&n);
      for(c=1;c<=n;c=c+1)
      {
            printf("ENTER A NO : ");
            scanf("%d",&no);
            s = s + no;
      }
      printf("\n SUM : %d",s);
      getch();

}
```

35. Program find the biggest and smallest of any given 'n' no

```
main()
{
      int no,c,small,big,n;
      clrscr();
      printf("\n Enter how many no.s:");
      scanf("%d",&n);
      printf("ENTER A NO : ");
      scanf("%d",&no);
      big = small = no;
      for(c=1;c<n;c++)
      {
            printf("ENTER A NO : ");
            scanf("%d",&no);

            if(no<=small)
            small = no;

            if(no>=big)
            big = no;
      }
      printf("\n SMALL  : %d",small);
      printf("\n BIG    : %d",big);
      getch();
}
```

36. write a program for FEBINOCCI SERIES (FIRST N)

```
#include<stdio.h>
#include<conio.h>
main()
{
      int a,b,c,i,n;
      clrscr();
      printf("\n Enter how many terms:");
      scanf("%d",&n);
      a = 0;
      b = 1;
      printf("\n %d",a);
      printf("\n %d",b);
      for(i=3;i<=n;i++)
      {
            c = a+b;
            printf("\n %d",c);
            a = b;
            b = c;
      }
      getch();
}
```

37. write a program for FEBINOCCI SERIES below given No

```
#include<stdio.h>
#include<conio.h>
main()
{
      int a,b,c,no;
      clrscr();
      printf("ENTER A NO : ");
      scanf("%d",&no);
      a = 0;
      b = 1;
      c = 0;
      for( ;c<=no; )
      {
            printf("\n %d",c);
            a = b;
```

```
                b = c;
                c = a + b;
        }
        getch();
}
```

38. write a program for FEBINOCCI SERIES

```
#include<stdio.h>
#include<conio.h>
main()
{
        int a,b,c,no,i;
        clrscr();
        printf("HOW MANY NO? ");
        scanf("%d",&no);
        a = 0;
        b = 1;
        c = 0;
        for(i=1;i<=no;i++ )
        {
                printf("\n%5d. %5d",i,c);
                a = b;
                b = c;
                c = a + b;
        }
        getch();
}
```

39. write a program for $x + x^2/2! + x^3/3! \ldots x^n/n!$

```
#include<stdio.h>
#include<conio.h>
main()
{
        int i,j,f,p;
        float s=0.0;
        int x,n;
        clrscr();
        printf("ENTER X,N VALUES : ");
        scanf("%d%d",&x,&n);
        for(i=1;i<=n;i++)
        {
                f=1;
                p=1;
```

```
            for(j=1;j<=i;j++)
            {
            p = p * x;
            f = f * j;
            }
            printf("%d/%d +",p,f);
            s = s + (float)p/f;
      }
      printf("\b =  %.2f",s);
      getch();
}
```

40. write a program for $x + x^3/3! + x^5/5+...$

```
      #include<stdio.h>
      #include<conio.h>
      main()
      {
      int i,x,n,f=1,p=1;
      float s=0.0;
      clrscr();
      printf("ENTER X,N VALUES : ");
      scanf("%d%d",&x,&n);
      s = p = x;
      for(i=3;i<=n;i = i+2)
      {
            f = f*i*(i-1);
            p = p*x*x;
            printf("%d/%d+",p,f);
            s = s + (float)p/f;

      }
      printf("\b = %.2f",s);
      getch();
      }
```

41. write a program for

```
/*
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5  */
```

```
#include<stdio.h>
#include<conio.h>
main()
{
      int i,j,n;
      clrscr();
      printf("\n Enter the value of n :");
      scanf("%d",&n);
      for(i=1;i<=n;i++)
      {
            for(j=1;j<=i;j++)
            printf("%d ",j);
            printf("\n");
      }
      getch();
}
```

42. write a program for

```
/*
      5 4 3 2 1
        4 3 2 1
          3 2 1
            2 1
              1
            2 1
          3 2 1
        4 3 2 1
      5 4 3 2 1
*/
#include<stdio.h>
#include<conio.h>
main()
{
      int i,j,n;
      clrscr();
      printf("Enter a No:");
      scanf("%d",&n);
      for(i=n;i>=1;i--)
      {
            for(j=1;j<=n-i;j++)
            printf(" ");
            for(j=i;j>=1;j--)
            printf(" %d",j);
```

```
            printf("\n");
        }
        for(i=2;i<=n;i++)
        {
            for(j=1;j<=n-i;j++)
            printf(" ");
            for(j=i;j>=1;j--)
            printf(" %d",j);
            printf("\n");
        }
        getch();
    }
```

43. write a program for

```
          1
        1 2 1
      1 2 3 2 1
    1 2 3 4 3 2 1
  1 2 3 4 5 4 3 2 1
  #include<stdio.h>
  #include<conio.h>
  main()
  {
        int i,j,n;
        clrscr();
        printf("Enter a No:");
        scanf("%d",&n);
        for(i=1;i<=n;i++)
        {
            for(j=1;j<=n-i;j++)
            printf(" ");
            for(j=1;j<=i;j++)
            printf("%d ",j);
            for(j=i-1;j>=1;j--)
            printf("%d ",j);
            printf("\n");
        }
        getch();
  }
```

44. write a program for

```
1                   1
1 2               2 1
1 2 3           3 2 1
1 2 3 4       4 3 2 1
1 2 3 4 5 5 4 3 2 1
```

```
#include<stdio.h>
#include<conio.h>
main()
{
      int i,j,n;
      clrscr();
      printf("Enter a No:");
      scanf("%d",&n);
      for(i=1;i<=n;i++)
      {
      for(j=1;j<=i;j++)
      printf(" %d",j);
      for(j=1;j<=(n-i)*2;j++)
      printf("  ");
      for(j=i;j>=1;j--)
      printf(" %d",j);
      printf("\n");
      }
      getch();
}
```

45. write a program for

```
A B C D E E D C B A
A B C D       D C B A
A B C           C B A
A B               B A
A                   A
```

```
main()
{
```

```
        int i,j,n;
        clrscr();
        printf("Enter a No:");
        scanf("%d",&n);
        for(i=n;i>=1;i--)
        {
        for(j=1;j<=i;j++)
        printf(" %c",j+64);
        for(j=1;j<=(n-i)*2;j++)
        printf("  ");
        for(j=i;j>=1;j--)
        printf(" %c",j+64);
        printf("\n");
        }
        getch();
}
```

46. write a program for

```
A B C D E E D C B A
A B C D     D C B A
A B C         C B  A
A B             B A
A                 A
A B             B A
A B C         C B A
A B C D     D C B A
A B C D E E D C B A
```

```
#include<stdio.h>
#include<conio.h>
main()
{
        int i,j,n;
        clrscr();
        printf("Enter a No:");
        scanf("%d",&n);
        for(i=n;i>=1;i--)
        {
```

```
            for(j=1;j<=i;j++)
            printf(" %c",j+64);
            for(j=1;j<=(n-i)*2;j++)
            printf("  ");
            for(j=i;j>=1;j--)
            printf(" %c",j+64);
            printf("\n");
            }
            for(i=2;i<=n;i++)
            {
                  for(j=1;j<=i;j++)
                  printf(" %c",j+64);
                  for(j=1;j<=(n-i)*2;j++)
                  printf("  ");
                  for(j=i;j>=1;j--)
                  printf(" %c",j+64);
                  printf("\n");
            }
            getch();
      }
```

47. write a program to demonstrate goto statements

```
#include<stdio.h>
#include<conio.h>
main()
{
      int c=1;
      clrscr();
      z:
      if(c<=5)
      {
      printf("\n Hello ");
      c = c+1;

      goto z;
      }
      getch();
}
```

48. write a program to demonstrate break statement

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

```
#include<conio.h>
main()
{
int i;
clrscr();
for(i=1;i<=10;i++)
{
printf("\n I:%d",i);
if(i==5)
break;
}
getch();
}
```

49. write a program to demonstrate continue statement

```
#include<stdio.h>
#include<conio.h>
main()
{
int i;
clrscr();
for(i=1;i<=10;i++)
{
if(i==5)
continue;
printf("\n I:%d",i);
}
getch();
}
```

50. write a program to demonstrate switch statement

```
#include<stdio.h>
#include<conio.h>
main()
{
int ch;
clrscr();
printf("Enter ur choice");
scanf("%d",&ch);
switch(ch)
```

```
{
case 1:printf("\n Sunday");
       break;
case 2:printf("\n Monday");
       break;
case 3:printf("\n Tuesday");
       break;
case 4:printf("\n Wednesday");
       break;
case 5:printf("\n Thursday");
       break;
case 6:printf("\n Friday");
       break;
case 7:printf("\n Saturday");
       break;
default:printf("\n wrong choice");
}
getch();
}
```

51. write a program for arithmetic calculator using switch statement

```
#include<stdio.h>
#include<conio.h>
main()
{
       int ch,a,b,res;
       do
       {
       clrscr();
printf("\n\t\t 1. ADDITION ");
printf("\n\t\t 2. SUBSTRACTION ");
printf("\n\t\t 3. MULTIPLICATION ");
printf("\n\t\t 4. DIVISION ");
printf("\n\t\t 5. EXIT ");
printf("\n\n\t\t ENTER UR CHOICE [1,2,3,4,5] : ");
       scanf("%d",&ch);

       if(ch==5)
       exit(0);

       if(ch>5)
```

```
        printf("\n \t\t\t WRONG CHOICE ");
        else
        {
              printf("Enter 2 no : ");
              scanf("%d%d",&a,&b);
              switch(ch)
              {
                    case 1:
                    res = a + b;
                    break;
                    case 2:
                    res = a - b;
                    break;
                    case 3:
                    res = a * b;
                    break;
                    case 4:
                    res = a / b;
                    break;
                    default:
                    res = 0;
              printf("\n \t\t\tUR CHOICE IS WRONG");
              }
        printf("\n \t\t\t RESULT = %d",res);
        }
  printf("\n\n\n \t\t\tPRESS ANY KEY 2 CONT..");
        getch();
        }while(ch!=5);
  }
```

Unformatted Console IO Functions

These are used to perform on character data.

Input Functions: -

1→**getchar ():** - It is defined in <stdio.h>

General Form: -variable_name=getchar ();

- It is used to accept a character without auto return
- It returns the character data that is inputted

```
Char c;
C=getchar ();
```

2→**getch ():** -It is defined in <conio.h>

General Form: - variable_name=getch ();

- It is used to accept a character with auto return with echo. It returns the inputted character.
-

3→**getche ():** - it is used accept a character with auto return with echo. It returns the inputted character.

Output Functions: -

1→**putchar ():** - It is defined in <studio.h>

General Form: - putchar (char variable/const);

- It will display the content of specified character variable or character constant. It is a macro.

```
putchar ('A'); //char c='x';
putchar(c);
```

2→**putch ():** - It is defined in <conio.h>

- It is used to display contents of character variable or character constant.

ARRAYS

An array is a collection of data items or variables. Which are of homogeneous data type, referred with a common name and stored sequentially in the memory.

General function: - data type array name [size];

- **Here size specifies the number of variables in that array**
- **By default, the index starts from 0 to size-1.**
- **Subscripting along with the index can access the individual variables of the array.**
- **The number of bytes required for the array of variables is considered as (size * data types memory).**

Usage of array:

- **To store more than one value in a single variable.**
- **To decrease the completing of a program.**
- **To increase the readability of a program.**
- **Easy to searching and sorting.**

Searching: - Searching is a technique of finding the element in the list or not.

Sorting: - Array the elements in some logical order either ascending order.

Initializing an array: - We can initialize the linear array at the time of declaration itself.

General function:
data type array name [size] = {item1, item2… item n};

- **Size can be omitted.**

Multi dimensional arrays:

- **If we can specify, more than one dimensions for an array then it is called multidimensional array.**
- **If we are specifying two subscripts, then it is called two-dimensional arrays.**

General function:
data type array name [row size][column size];

The row size and column size must be positive integers.

Initializing two-dimensional array: We can initialize a two-dimensional array like a linear array. Here the items are initialized in two wise.

General function:

```
Data type array name [row size][column size];
        Int Tlist [3] [2] = {1, 2, 3, 4, 5, 6};
```

The row size can be omitted but column size is must be specified. (Not optional).

Exercise

1) write a program to demonstrate array

```
#include<stdio.h>
#include<conio.h>
main()
{
   int a[5]={10,20,30,40,50};
   clrscr();
   printf("\n Array is:");
   printf("\n a[0]:%d",a[0]);
   printf("\n a[1]:%d",a[1]);
   printf("\n a[2]:%d",a[2]);
   printf("\n a[3]:%d",a[3]);
   printf("\n a[4]:%d",a[4]);
   getch();
}
```

2) write a program for reading values for an array

```
main()
{
   int a[5],i;
   clrscr();
   for(i=0;i<5;i++)
   {
         printf("ENTER A NO : ");
         scanf("%d",&a[i]);
   }
```

```
   printf("\n ARRAY ELEMENTS :\n ");
   for(i=0;i<5;i++)
   {
        printf("\na[%d]= %d",i,a[i]);
   }
   getch();
}
```

3) write a program for sum of elements in an array

```
main()
{
   int a[20],i,n,s=0;
   clrscr();
   printf("\n Enter no. of elements:");
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
        printf("ENTER A NO : ");
        scanf("%d",&a[i]);
        s = s + a[i];
   }
   printf("\n ARRAY ELEMENTS :\n ");
   for(i=0;i<n;i++)
   {
        printf("\n  a[%d]:%d",i,a[i]);
   }
   printf("\n  SUM = %d",s);
   getch();
}
```

4) write a program to find the biggest and least elements in an array of n elements

```
main()
{
int a[20],i,n,big,small;
clrscr();
printf("\n Enter no. of elements:");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
printf("ENTER A NO : ");
scanf("%d",&a[i]);
```

```
}

big = small = a[1];
for(i=2;i<=n;i++)
{
if(a[i]>=big)  big = a[i];
if(a[i]<=small) small = a[i];
}
printf("\n Given Elements : ");

for(i=1;i<=n;i++)
{
printf("\na[%d]: %d",i,a[i]);
}
printf("\n BIG   = %d",big);
printf("\n SMALL = %d",small);
getch();
}
```

5) write a program to find an element in an array

```
main()
{
      int a[15],no,i,n,found = 0;
      clrscr();
      printf("Enter no.of elements");
      scanf("%d",&no);
      for(i=0;i<no;i++)
      {
            printf("ENTER A NO : ");
            scanf("%d",&a[i]);
      }

      printf("ENTER NO TO SEARCH : ");
      scanf("%d",&n);

      for(i=0;i<no;i++)
      {
            if(a[i]==n)
            {
                  found = 1;
                  break;
            }
      }
```

```
        if(found==1)
          {
          printf("\n SUCCESSFUL SEARCH");
          printf("\n %d FOUND at position : %d",n,i);
          }
        else
        printf("\n UNSUCCESSFUL SEARCH",n);
        getch();
   }
```

6) write a program insert an element in an array at specified position

```
main()
{
   int a[30],no,p,i,n;
   clrscr();
   printf("\n Enter no. of elements:");
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
        printf("ENTER A NO : ");
        scanf("%d",&a[i]);
   }

   printf("\nENTER NO AND POSITION TO INSERT ");
   scanf("%d%d",&no,&p);

   if(p>=0 && p<=n)
   {
    for(i=n-1;i>=p;i--)
     {
      a[i+1]= a[i];
     }
    a[p] = no;
    n++;
   }
   else printf("\n Invalid position");
   printf("\n ARRAY AFTER INSERT \n");
   for(i=0;i<n;i++)
   printf("   %d",a[i]);
   getch();
}
```

7) write a program deleting an element from an array

```
main()
{
int a[30],i,n,p;
clrscr();
printf("\n Enter no. of elements:");
scanf("%d",&n);
for(i=1;i<=n;i++)
  {
  printf("\n ENTER ELEMENT :");
  scanf("%d",&a[i]);
  }
printf("\n Enter position to  delete the element in that
position:");
scanf("%d",&p);
if(p>=1 && p<=n)
   {
   for(i=p;i<=n;i++)
       {
        a[i]=a[i+1];
        }
   n--;
   }
else printf("\n Invalid position");
printf("\n ARRAY AFTER DELETION:");
for(i=1;i<=n;i++) printf("%d  ",a[i]);
getch();
}
```

8) write a program for sort the array in ascending order

```
   main()
   {
         int a[5],i,j,temp;
         clrscr();
         for(i=0;i<5;i++)
         {
               printf("ENTER A NO : ");
               scanf("%d",&a[i]);
         }
         for(i=0;i<5;i++)
         {
```

```
            for(j=i+1;j<5;j++)
            {
                if(a[i]>=a[j])
                {
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
                }
            }
        }
        printf("ARRAY IN ASCENDING ORDER :\n");
        for(i=0;i<5;i++)
        printf("\n a[%d]:%d",i,a[i]);
        getch();
    }
```

9) write a program to demonstrate 2D array

```
main()
{
   int a[3][2]={1,6,3,4,2,5};
   int i,j;
   clrscr();
printf("\n The Given Matrix is:");
for(i=0;i<3;i++)
{
   printf("\n");
   for(j=0;j<2;j++)
   printf("\t %d",a[i][j]);
}
getch();
}
```

10) write a program for sum of 2 given matrices

```
main()
{
   int a[5][5],b[5][5],c[5][5],i,j,m,n,p,q;
   clrscr();
   printf("\n Enter order of first matrix:");
   scanf("%d%d",&m,&n);
   printf("\n Enter order of second matrix:");
   scanf("%d%d",&p,&q);
   if(m==p && n==q)
```

```
   {
   printf("\n ENTER VALUES FOR MATRIX A : \n");
   for(i=0;i<m;i++)
   for(j=0;j<n;j++)
      scanf("%d",&a[i][j]);
   printf("\n ENTER VALUES FOR MATRIX B : \n");
   for(i=0;i<p;i++)
   for(j=0;j<q;j++)
      scanf("%d",&b[i][j]);
   printf("\n THE VALUES FOR MATRIX A : \n");
   for(i=0;i<m;i++)
   {
   for(j=0;j<n;j++)

      printf("\t %d",a[i][j]);
      printf("\n");
      }
   printf("\n THE VALUES FOR MATRIX B : \n");
   for(i=0;i<p;i++)
   {
   for(j=0;j<q;j++)

      printf("\t %d",b[i][j]);
      printf("\n");
   }
   for(i=0;i<m;i++)
   for(j=0;j<n;j++)

   c[i][j] = a[i][j] + b[i][j];

   printf("RESULTANT MATRIX  : \n");
   for(i=0;i<m;i++)
   {
         for(j=0;j<n;j++)
         printf("\t%d",c[i][j]);

   printf("\n");
   }
   }
   else printf("\n MATRIX ADDITION IS NOT POSSIBLE");
getch();
}
```

11) write a program to find the sum of diagnol of a square matrix(trace matrix)

```
main()
{
int a[10][10],r,c,s=0,i,j;
clrscr();
printf("\n How many Rows? ");
scanf("%d",&r);
printf("\n How many Cols? ");
scanf("%d",&c);
if(r!=c)
printf("\n NOT A SQUARE MATRIX ");
else
{
      printf("Enter %d Values \n ",r*c);
       for(i=0;i<r;i++)
       {
            for(j=0;j<c;j++)
            {
                  printf("Enter a no : ");
                  scanf("%d",&a[i][j]);
                  if(i==j) s+=a[i][j];

            }
       }

printf("\n GIVEN MATRIX \n");
for(i=0;i<r;i++)
{
      for(j=0;j<c;j++)
      printf("  %d",a[i][j]);
      printf("\n");
}
}
printf("TRACE : %d",s);
getch();
}
```

12) write a program to find transpose matrix for a given matrix

```
main()
```

```
{
      int a[10][10],r,c,i,j;
      clrscr();
      printf("How many Rows? ");
      scanf("%d",&r);
      printf("How many Cols? ");
      scanf("%d",&c);

      for(i=0;i<r;i++)
      {
            for(j=0;j<c;j++)
            {
                  printf("Enter a No : ");
                  scanf("%d",&a[i][j]);
            }
      }

      printf("Given Matrix \n");
      for(i=0;i<r;i++)
      {
            for(j=0;j<c;j++)
            printf("%3d",a[i][j]);
            printf("\n");
      }
      printf("Transpose Matrix : \n");
      for(i=0;i<c;i++)
      {
            for(j=0;j<r;j++)
            printf("%3d",a[j][i]);
            printf("\n");
      }
getch();
}
```

13) write a program to find given matrix is symetric or not

```
main()
{
      int a[10][10],b[10][10],r,c,i,j,sym = 1;
      clrscr();
      printf("How many Rows? ");
      scanf("%d",&r);
      printf("How many Cols? ");
```

```
        scanf("%d",&c);

        for(i=0;i<r;i++)
        {
              for(j=0;j<c;j++)
              {
                    printf("Enter a No : ");
                    scanf("%d",&a[i][j]);
              }
        }

        for(i=0;i<c;i++)
        for(j=0;j<r;j++)
        b[i][j]=a[j][i];

        printf("Given Matrix \n");
        for(i=0;i<r;i++)
        {
              for(j=0;j<c;j++)
              printf("%3d",a[i][j]);
              printf("\n");
        }
        printf("Transpose Matrix : \n");
        for(i=0;i<c;i++)
        {
              for(j=0;j<r;j++)
              printf("%3d",b[i][j]);
              printf("\n");
        }

        if(r==c)
        {
              for(i=0;i<r;i++)
              for(j=0;j<c;j++)
              {
                    if(a[i][j]!=b[i][j])
                    {
                          sym = 0;
                          break;
                    }
              }
        }
        else
        sym = 0;
```

```
        if(sym==1)
        printf("\n GIVEN MATRIX IS SYMETRIC");
        else
        printf("\n GIVEN MATRIX IS NOT SYMETRIC");
   getch();
   }
```

14) write a program for matrix multiplication

```
main()
{
   int a[10][10],b[10][10],c[10][10];
   int r1,r2,c1,c2,i,j,k;
   clrscr();
   printf("Enter Rows & Cols for Matrix 1 : ");
   scanf("%d%d",&r1,&c1);
   printf("Enter Rows & Cols for Matrix 2 : ");
   scanf("%d%d",&r2,&c2);
   if(c1 != r2)
   printf("MULTIPLICATION IS NOT POSSIBLE ");
   else
   {
   printf("Enter Values for Matrix 1 \n ");
   for(i=0;i<r1;i++)
   for(j=0;j<c1;j++)

      scanf("%d",&a[i][j]);
   printf("The Values for Matrix 1 \n");
   for(i=0;i<r1;i++)
   {
   for(j=0;j<c1;j++)
      printf("%3d",a[i][j]);
      printf("\n");
   }
   printf("Enter  Values for Matrix 2 \n ");
   for(i=0;i<r2;i++)
   for(j=0;j<c2;j++)
      scanf("%d",&b[i][j]);
   printf("Enter Values for Matrix 2 \n");
   for(i=0;i<r2;i++)
   {
   for(j=0;j<c2;j++)
```

```
      printf("%3d",b[i][j]);
      printf("\n");
      }
   for(i=0;i<r1;i++)
   {
    for(j=0;j<c2;j++)
      {
      c[i][j] = 0;
      for(k=0;k<r2;k++)
      c[i][j] = c[i][j] + a[i][k]*b[k][j];
      }
   }

   printf("\n Resultant matrix is : \n");
   for(i=0;i<r1;i++)
   {
   for(j=0;j<c2;j++)
   printf("%3d",c[i][j]);
   printf("\n");
   }
   }
getch();
}
```

STRINGS

A string is a character may terminate with a null character`\0'.

General function: char str name [size];

Input string through keyboard: scanf ("%s", str name);

- The scanf () function with format specifier %s input the string until a white space character is encountered.
- When white space character is inputted that will be treated as termination and scanf () function stored `\0` at the end o the string.

E.g.: char str [10];
Scanf ("%s", str);

Outputting a string to VDU: printf ("%", str name);

- Printf () function directs all the characters to VDU untill`\0` is encountered in the string.

```
Char str [10];
Scanf ("%s", str);
Printf ("%s", str);
```

Initializing of string: We can initialize the strings in two ways.
General forms are:

- Char str name [size]={all character at end \0};

E.g.: char name [15] = {'B','H','A','S','K','A','R','D','E','V'};

- Char string name [size]="size";

E.g.: char str [5] ="BHASKAR DEV";
Here size is optional.

Scanset format specifier:

- It is a new format specifier.
- A scanf is the collection of characters with in "[]".

General function: ("% [characters]", string name);
Here, the scanf () function takes all the characters for string until a character is not in scanset is encountered that will be treated as termination.

```
Char str [10];
Scanf ("% [ABab]", str);
Scanf ("% [A...Z a...z 0...9]", str);
```

Inverted Scanset Format Specifier: -It is a scanset preceded with circumflex (^).

General Form: - scanf ("% [^characters]", string name);
Here the scanf () takes all the characters that are not specified in scanset. If a character that is in inverse scanset is encountered that will be treated as termination.

Unformatted IO Function For Strings: -

1→gets () Function: -

- **It is defined in <stdio.h>.**
- **It is an unformatted Input function to accept a string.**

General Form: - gets (str_name);
→Here gets () accepts the string until returning key is pressed.

```
Char str [10];
Gets (str);
```

2→puts () Function: -

- **It is defined in <stdio.h>**
- **It is an unformatted Output function to display a string.**

General Form: - puts (str_name);
→Here puts () displays all the characters in the string until '\0' reached.

```
Char str [] ="Bhaskar Dev";
Puts (str);
```

String Handling Functions: - All the string-handling functions are defined in <string.h>

→ **Strlen (): - The task of this function is to find the length of the given string.**

Length= strlen (str);

→ **Strcpy (): - The task of this function is to copy the content of one string to another string.**

Strcpy (destination string, source string);

→ **Strrev (): - The task of this string is to reverse the given string and stores it in the same string.**

Strrev (string name);

→ **Strcat (): - The task of this function to concatenate the two strings.**

Strcat (string1, string2);

→ **Strcmp (): - The task of this function is to compare two strings for equality. The equality can be verified with the return value of strcmp**

Strcmp (string1, string2)
↳ =0 when ASCII value of string1 & string2 are equal.
↳ <0 when ASCII value of string1 is less than string2.
↳ >0 when ASCII value of string1 is greater than string2.

→ **Strupr (): - The task of this function is to convert the given string into upper case.**

Strupr (string name);

→ **Strlwr (): - The task of this function is to convert the given string into lower case.**

Strlwr (string name);

→ **Strdup ():** - The task of this function to make a duplicate string for the given string

Strdup (string name);

Array Of Strings: - Like basic data types arrays, we can create array of strings. An array of string is nothing but a two dimensional array of characters, each row indicating a string.

General Form: char string array name [number of strings] [size of each string];

- Specifying the row index along with the string array name can access string.

Initializing Array Of String: -These can be initialized as

General Form:

char str_array name [no of str] [size of each string]={string1, string2... string n};

char strarray [50] [50] = {"bhaskar","dev"};

Exercise

1. write a program read a string

```
#include<stdio.h>
#include<conio.h>
main()
{
      char name[20];
      int i;
      clrscr();
      printf("ENTER UR NAME : ");
      gets(name);
      for(i=0; name[i]!='\0'; i++)
      printf("\n %c",name[i]);
      getch();
}
```

2. write a program to read a string using gets() function & print that strin using puts() function

```
main()
{
char str[20];
clrscr();
printf("\n Enter a string:");
gets(str);
printf("\n The string is :");
puts(str);
getch();
}
```

3. write a program to print a string character wise

```
#include<stdio.h>
#include<conio.h>
main()
{
      char name[20];
      int i,j;
      clrscr();

      printf("ENTER UR NAME : ");
      gets(name);
      for(i=0; name[i]!='\0'; i++)
      {
            for(j=0;j<=i;j++)
            printf("%2c",name[j]);
            printf("\n");
      }
      getch();
}
```

4. write a program to print the before characters of a given string

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
      char name[20];
```

```
        int i,j;
        clrscr();

        printf("ENTER UR NAME : ");
        gets(name);

        strupr(name);
        puts(name);

        for(i=0; name[i]!='\0'; i++)
        {
              printf("%c - ",name[i]);
              for(j='A';j<= name[i];j++)
              printf("%2c",j);
              printf("\n");
        }
        getch();
  }
```

5. write a program to print specified No. of strings

```
#include<stdio.h>
#include<conio.h>
main()
      {
            char name[5][20];
            int i,n;
            clrscr();
            printf("Enter a No:");
            scanf("%d",&n);
            fflush(stdin);
            for(i=1;i<=n;i++)
            {
                  printf("Enter Name : ");
                  gets(name[i]);
            }
            printf("\n Given Name : \n");
            for(i=1;i<=n;i++)
            puts(name[i]);
            getch();
      }
```

6. write a program for String Handling Function strcpy()

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
      char a[20],b[20];
      clrscr();
      printf(" Enter String A: ");
      gets(a);
      strcpy(b,a);
      printf(" String Copied \n B = %s",b);
      getch();
}
```

7. write a program for String Handling Function strcat()

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
      char a[20],b[20];
      clrscr();
      printf(" Enter a String : ");
      gets(a);
      printf(" Enter a String : ");
      gets(b);
      strcat(a,b);
      printf(" String concatinated \n a = %s",a);
      getch();
}
```

8. write a program for String Handling Function strlen().

```
main()
{
      char a[20];
      int len;
      clrscr();
      printf(" Enter a String : ");
      gets(a);
      len = strlen(a);
      printf(" String Length  = %d",len);
```

```
        getch();
}
```

9. write a program for String Handling Function strcmp()

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
        char a[20],b[20];
        int i;
        clrscr();
        printf("Enter A String : ");
        gets(a);
        printf("Enter A String : ");
        gets(b);
        i = strcmp(a,b);
        if(i==0)
        printf("BOTH ARE EQUAL");
        else
                if(i>0)
                printf("BIG = %s (FIRST STRING)",a);
                else
                printf("BIG = %s (SECOND STRING)",b);

        getch();
}
```

10. write a program for String Handling Function strcmpi()

```
#include<string.h>
main()
{
        char a[20],b[20];
        int i;
        clrscr();
        printf("Enter A String : ");
        gets(a);
        printf("Enter A String : ");
        gets(b);
        i = strcmpi(a,b);
        if(i==0)
```

```
        printf("BOTH ARE EQUAL");
        else
              if(i>0)
              printf("BIG = %s (FIRST STRING)",a);
              else
              printf("BIG = %s (SECOND STRING)",b);

        getch();
  }
```

11. write a program for String Handling Function strlwr(),strupr

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
char a[20];
clrscr();
printf("Enter a String : ");
gets(a);
printf("\n Given String in Upper Case : %s",strupr(a));
printf("\n Given String in Lower Case : %s",strlwr(a));
getch();
}
```

12. write a program for String Handling Function strrev()

```
main()
{
      char a[25];
      clrscr();
      printf("Enter A string : ");
      gets(a);
      printf("Reverse String : %s",strrev(a));
      getch();
}
```

13. write a program for String Handling Function strchr():

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
char ch,str[25];
int x;
clrscr();
printf("Enter a String : ");
gets(str);
printf("Enter a Character to search : ");
scanf("%c",&ch);
x = strchr(str,ch);
if(x!=0)
printf("%c Existing in %s",ch,str);
else
printf("%c NOT EXISTING",ch);
getch();
}
```

14. write a program for String Handling Function strstr

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
char ch[25],str[25];
int x;
clrscr();
printf("Enter a String : ");
gets(str);
printf("Enter a String to search : ");
gets(ch);
x = strstr(str,ch);
if(x!=0)
printf("%s Existing in %s",ch,str);
else
printf("%s NOT EXISTING",ch);
getch();
}
```

15. write a program for String Handling Function strdup

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
char s1[20];
char *s2;
clrscr();
printf("\n Enter a string :");
gets(s1);
s2=strdup(s1);
printf("\n The duplicate copy is : %s",s2);
getch();
}
```

16. write a program for Given string is palandrome or not

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
      char a[25],b[25];
      int x;
      clrscr();
      printf("Enter a String : ");
      gets(a);
      printf("\n String in A:%s",a);
      strcpy(b,a);
      printf("\n String in B:%s",b);
      strrev(b);
      printf("\nAfter reversing the string B is:%s",b);
      x = strcmpi(a,b);
      if(x==0)
      printf("\n%s is PALANDROME",a);
      else
      printf("\n%s is NOT PALNDROME",a);
      getch();
}
```

17. write a program to arrange the given strings in alphabetical order

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
char s[10][15],t[15];
int i,j,n;
clrscr();
printf("\n Enter no. of strings:");
scanf("%d",&n);
fflush(stdin);
printf("\n Enter strings:");
for(i=0;i<n;i++)
  {
  scanf("%s",s[i]);
  }
for(i=0;i<n-1;i++)
for(j=i+1;j<n;j++)
   {
   if(strcmpi(s[i],s[j])>0)
        {
        strcpy(t,s[i]);
        strcpy(s[i],s[j]);
        strcpy(s[j],t);
        }
   }
printf("\n Strings in Alphabetical order");
for(i=0;i<n;i++)
  {
  printf("\n %s",s[i]);
  }
getch();
}
```

Functions

Function is a self-contained block of statements to achieve a specific task. A 'C' program is a collection of functions along with the main () function.

Uses of Functions: -

→ To achieve modularity (Replication vs Repetition).
→ To achieve readability.
→ Errors can be easily identified (logically).
→ Code can be reduced whenever there is replication.

Types of functions: -

1→Library Functions: - All the predefined functions are library functions, which all come along with 'C' software. These functions are called standard functions.

E.g.: - scanf (), printf (), strlen () …

2→User Defined Functions: - The following steps can implement –They are

- ✓ **Function Declaration or Prototype: -**
  - It can be used to verify whether the function call made by the user is valid or not.
  - It tells information about the function, like Number of arguments about the function along with data.Also tells the return type.

General Form: - return type function name (data type of arguments);

e.g.: - void println(void);

✓ **Function Definition: - it is nothing but composing statements for the specific task by means of some logic.**

General Form: - return type function name (Arguments with data type)

```
{
Local variable declaration;
/ / Valid statements / /
Return statements;
}
```

- ♣ **The arguments that are specified in the function definition are called formal or dummy arguments.**
- ♣ **'Return' is the statement, which is used to send at most one value to the calling function.**
- ♣ **The 'return' statement & considered as termination the function. These may be more than one return statement in the function in the function. In that case the first occurrence of 'return' is considered as termination.**

3→ Function calling: - Once the function declaration and definition is completed, we can utilize that function by specifying the function name with the actual arguments. When a function is called, corresponding prototype is verified, for the validity of function call. If the function call is valid, then the control transferred to the function definition.

E.g.: - printline ();

Categories of functions: -The functions are categorized into 3 types

→ **Function with no arguments & no return values: - In this case there will not be any data transfers between the calling function and called function. When this type of function is called, the control transferred to the function definition, executes all the statements and returns backs to the calling function.**

```
Void message (void)
{
……….
……….
}
```

→ **Function with arguments & no return values: -** In this case there will be one-way data transfers from the calling function to the called function. When this function is called, the control transferred along with the values of actual arguments, performs execution and will not return any value to the calling function.

→ **Function with arguments & with return values: -** In this case there will be a two-way data transfer between the calling function and called function. Calling function may send any number of arguments where as the called function will return only one value.

Function Call Mechanisms: - These are categorized into 2 types

Call By Value: - In this method the function is called by sending the value of actual arguments from calling function. In this called function, the dummy arguments are created & the actual arguments are copied to the them at the time of function call. As the dummy arguments made changes, they don't affect the Actual arguments.

Call By Reference: - In this method the function is called by passing the addresses (Reference) of actual arguments instated of the values. In function definition, the addresses of actual arguments are received by the respective pointer variables.

As we are pasting the address, the updations perform on the formal arguments will effect the actual arguments.

This method is used whose it is required to update the contents of actual arguments in the called function.

Passing Arrays to function:

→Like basic data items, we are specify or send an array as arguments to the function.

→ In the function call, we have to specify the name of the array.

→ **As we are specifying the values of the array (starting address) the array is passed as reference to the function.**

General form:

Prototype: Return type function name (data type [], other items);

Function call: Function name (array name, other arguments)

Function definition: Return type function name (data type formal args)
{……..}

String as arguments to function: **we can pass entire string to a function .it can be illustrated with the following example.**

Recursion: Recursion is the technique of achieving the task of a function by means of itself. It is known as recursive function.

Library function:

Function	prototype	description
Clrscr	void clrscr ();	clear the screen, moves cursor to top left Corner.
Gotoxy ()	void gotoxy (int, int);	moves the cursor to the desired position.
Kbhit ()	char kbhit ();	verified that any key is pressed or not, if The key is pressed returns its ASCII code Else 0

<ctype.h>

isdigit ()	int isdigit (char)	→non zero (if digit) →Zero (if non digit)
isspace ()	int isspace (char)	→non zero (if white space) →Zero (else)

isctrl ()	int isctrl (char)	→non-zero if control char. (ASCII value 0 to 32 char)
isalpha ()	int isalpha (char)	
isupper ()	int isupper (char)	
islower ()	int islower(char)	
toupper ()	char toupper (char)	
tolower ()	char tolower (char)	

<Math.h>

abs ()	int abs (int);	absolute value
ceil ()	int ceil (double);	nearest integer>value
floor ()	int floor (double);	nearest integer<value
Pow ()	int pow (int, int);	
Log ()	double log (double);	with base e
log10 ()	double log10(double);	with base e
exp ()	double exp(double)	exponential value
sqr()	double sqr(double)	
sqrt()	double sqrt(int)	
cos	double cos(double)	returns cosine value for the given radians
acos()	double acos(double)	returns inverse of cosine of given radians

<stdlib.h>

atol()	int atoi(char[])	returns integer equivalent of given string. it returns zero if the string contains non numeric.
Atof()	float atof(char[])	converts the given string into float Equivalent.
Atol()	long int atol(char[]);	returns long int equivalent of given string.
Exit()	void exit(int);	normal termination of the program
Abort()	void abort(void)	abnormal termination.
System ()	void system (char [])	executes the dos command.

Exercise

1) write a program to demonstrate Functions

```
int add(int a,int b)  /* function definition */
{                     /* a,b are formal arguments */
int c;
c = a+b;
return c;
}

main()
{
int x,y,sum;
clrscr();
printf("\n ENTER 2 NO : ");
scanf("%d%d",&x,&y);
sum = add(x,y);   /*  function call  x,y are actual
arguments */
printf("\n SUM = %d",sum);
getch();
}
```

2) write a program to calculate ncr value using functions

```
float fact(float n)
{
   int i;
   float f=1.0;
   for(i=1;i<=n;i++)
   f = f*i;
   return f;
}
main()
{
   float n,r,fn,fr,fnr,f;
   clrscr();
   printf("ENTER N,R VALUES : ");
   scanf("%f%f",&n,&r);
   fn = fact(n);
   fr = fact(r);
   fnr = fact(n-r);
   f = fn/(fr*fnr);
   printf("\n RESULT : %.0f",f);
```

```
    getch();
}
```

3) write a program for reverse No

```
int reverse(int x)
{
   int r,s=0;
   while(x!=0)
   {
   r = x % 10;
   s = s*10+r;
   x = x/10;
   }
   return s;
}
main()
{
   int n,r;
   clrscr();
   printf("ENTER A NO : ");
   scanf("%d",&n);
   r = reverse(n);
   if(r==n)
   printf("%d PALANDROME",n);
   else
   printf("%d NOT PALANDROME",n);
   getch();
}
```

4) write a program for call by value

```
void change(int x)
{
   x = x*10;
printf("\n VALUE IN FUNCTION : %d",x);
}

main()
{
   int a=10;
   clrscr();
   printf("\n A = %d",a);
```

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

5) write a program for call by reference

```
void change(int *x)
{
*x=*x * 10;
printf("\n value in function=%d",*x);
}
main()
{
int y=10;
clrscr();
printf("\n Y = %d",y);
change(&y);
printf("\n Y = %d",y);
getch();
}
```

6) write a program funtion to find given no is prime or not

```
int isPrime(int n)
{
      int i,k=0;
      for(i=1;i<=n;i++)
      {
      if(n%i==0) k++;
      }
      if(k==2) return 1;
      else return 0;
}
main()
{
      int n;
      clrscr();
      printf("Enter a no : ");
      scanf("%d",&n);
      if(isPrime(n))
```

```
            printf("%d IS PRIME",n);
            else
            printf("%d NOT PRIME",n);
            getch();
      }
```

7) write a program to pass array as arguments to a function

```
printarray(int ar[])
{
  int i;
  printf("Enter an array values");
  for(i=0;i<10;i++)
  scanf("%d",&ar[i]);

  printf("\n The Array elements are:");
  for (i=0;i<10;i++)
    printf("%5d",ar[i]);
}
main()
{
   int  a[10];
   clrscr();
   printarray(a);
   getch();
}
```

8) write a program for recursive function

```
long int fact(int n);
main()
{
   int n;
   long int f;
   clrscr();
   printf("ENTER N : ");
   scanf("%d",&n);
   f=fact(n);
   printf("Factorial is :%ld",f);
   getch();
}
long int fact(int n)
{
```

```
        int i;
        long int f=1;
        for(i=n;i>=1;i--)
        f = f*i;
        return f;
    }
```

Storage classes

We can specify the storage class along with the data type, at the time of variable declaration.

The storage class tells the following:

→ where the variable is stored(ram/cpu registers)
→ where is the scope of the variable
→ what is the life of variable.

In "c" the storage classes are classified into

a) Automatic storage class:

→ The keyword 'auto' is used to declare automatic storage class variables.
→ The automatic storage class variables with have the following properties:

Storage: Main memory (RAM)

Initial default value: Unpredictable (garbage)

Scope: Local to the block/function in which the variable is declared.

Life: Till the control remains in the function/block in which the variable is declared.

Eg: main()

```
Auto int a=0;
{
auto int a=1;
{
auto int a=2;
printf("%d",a)
}
printf("%d",a);
}
printf("a")
}
```

b) Register storage class:

- → The key word is used to declare register storage class variables.
- → Register storage class variable will have following properties.

Storage: Cpu register

Initial default value: Unpredictable (garbage)

Scope: Local to the block /function, in which the variable is declared.

Life: Till the control remains in the function /block, in which variable is declared.

Note: The variable can be accessed by very quickly when it is allocated in the cpu

Registers. Generally these are used for counter variables.

- → In general, cpu registers are the bit length is 2 bytes .so they can not allocate for float , double and long variables.
- → As the number of cpu registers are limited, and they may be busy with some other tasks , the variables are allocated in main memory, though the variables are declared as register storage class variables.

c) Static storage class:

- → The keyword static is used to declare the static storage variable.
- → The static storage class variables will have the following properties.

Storage: Main memory

Initial default value: 0

Scope: Global to the block/function/entire program.

Life: Through-out the program.

Static storage class variables are very use full in recursive functions.

d) External storage class:

The keyword "extern" is used to define external storage class variables.

The external storage class variables will have the following properties.

Storage: Main memory.

Initial default value: 0

Scope : Through out the program

Life: Till the program comes to end

The extern storage class variables cannot be initialized at the time of definition.

Exercise

1. write a program for automatic storage class

```
void display()
{
auto int a;
printf("\n a = %d",a);

}

main()
{
clrscr();
display();
getch();
}
```

2. write a program for static variable

```
void increment()
{
static int a;
a++;
printf("\n a= %d",a);
}
```

```
main()
{
clrscr();
increment();
increment();
increment();
getch();
}
```

3. write a program for register storage class

```
void put()
{
register int i;
for(i=1;i<=100;i++)
    {
    printf("\n INDIA IS GREAT");
    }
}

main()
{
clrscr();
put();
getch();
}
```

4. write a program for global variables

```
int a,b=20;
void demo()
{
a++;
printf("\n a = %d",a);
}
main()
{
clrscr();
demo();
b++;
printf("\n b = %d",b);
getch();
}
```

User Defined Data Types

Structures:

"C" supports a constructed user defined data type "structure" which is a method of packing data of different data types.

"Structure" is useful to combine logically related information.

→ The purpose of using the "structures" is to increase the readability of a program.

General form of defining a structure:

```
Struct structure name
{
data type member 1;
data type member 2;
      :
      :
data type member n;
};
```

here structure is a key word to define a structure with name "structure name" having the field's member1;……member n; the fields of structure also called as attributes, members, and properties.

Once the structure definition is completed, then we can declare the structure variables.

General form: struct struct name variable1,variable2,……,variable n;

The size of a structure variable is the sum of size of all structure members.

The numbers of a structure variables can be accessed by means of a structure operator (.). The structure operator is used to superate the structure variable and its member.

Initialization structures:

We can initialize the structure , variables at the time of declaration .

General form: Structure name variable_name = { values for the member in the order of definitions }

The name of the structure is optional when all the variables of that structure are declared at structure definition.

Structures within structures:

Nesting of structures is possible in "c" language. It means that we can define a structure variable as a member in another structure. At the same time we can define a structure with in another structure.

Syntax: -

```
Struct structure name1
    {
    data type member 1;
    data type member 2;
        :
        :
    data type member n;
    };
    Struct structure name2
    {
    data type member 1;
    data type member 2;
        :
```

```
        :
    data type member n;
    Struct structure name1 variable1,variable2…;
    };
```

Eg:

```
struct date
{
int day;
int month;
int year;
};
struct employee
{
int eno;
char ename[30]
struct date dob:
struct date doj;
float e basic;
};
```

POINTERS TO STRUCTURES:

We can create pointers to a structure variable.

The general form of defining pointer variable is

```
Struct structure name * pointer variable;
```

→ Here the pointer variable ,holds the address of the structure variable.

→ The folder of the structure can be accessed indirectly by means of a pointer variable using pointer to structure operator. (→)

STRUCTURE AS ARGUMENTS TO FUNCTIONS:

We can pass structure variables to functions in two ways

- ✓ **call by value.**
- ✓ **call by reference.**

→ sending the values of structure variable:

when we are sending structure variable to a function, a dummy structure variable will be created in the called function and the content of actual argument is copy to it.

Syntax: -

```
Struct structure name
        {
        data type member 1;
        data type member 2;
              :
              :
        data type member n;
        };
Return type functionname(struct structure name
variable_name)
      {
      --------------
      --------------
      --------------
      }
```

Eg:

```
Struct employee
{
int eno;
char ename[40];
float ebasic;
};
void display(struct employee);
```

```
void display(struct employee)
{  }
display(e);
```

→ **sending references of a structure to a function:**

whenever we want to update a structure variable, we have to past the structure variable by reference to that function.

Syntax: -

```
Function name(&structure variable);
Returntype function name(struct structure name *pointer variable)
{
----------
----------
----------
}
```

Eg:

```
Increment (&e);
Void increment (struct employee *pe)
{
pe→ e basic = pi→ e basic + 4000-00;
return;
}
```

→ **self reference structures:**

whenever the member of a structure is a pointer is referencing to a variable of same structure type then that structure is called self referencing structure.

Synatax: -

```
Struct structure name
      {
```

```
Datatype member1;
Datatype member2;
……..
struct structure name *var_name;
};
```

Eg:

```
Struct node
{
int info;
struct mode * next;
};
```

UNIONS:

It is a method of packing data of different data types and all the union members will share the same storage area with in the main member.

- → Unions are used to define user de find datatypes.
- → Unions are use full for applications involving multiple members at a given time.
- → The union variable may be defined as the variable that is shared by fields of that union be assigned to all the members at a given time.
- → The union variable must be defined as the variable that is shared by fields of that union, which are generally of different data type at different times.

General form:

```
Union unionname
{
data type member1;
data type member2;
      :
      :
data type member n;
```

};

→ Once the union is defined, we can declare the union variables.
→ The size of union variable is the size of its largest member.

General form:

Union union name variabled1,variable2,----variablen ;

→ To access the members of the union we have to use the union operator (.).
→ We can use pointer to union operator (→)when we are accessing the members through a union pointer.

Type definition:

The new data type name can be created using "typedef" statement the "typedef" statement will create a new name for an existing data type other than creating a new datatype.

General form: typedef existing data type new name;

Eg: Typedef int number;

Here number is an additional data type name, but not the replacement data type name int.

→ It can be used to achieve machine independency is portability.
→ It is for self documentation .

Enumerations:

An enumeration is a set of named integers constants. the fields of enumeration called as enumerators.

→ The key word "enum" used to define new user defined data type along with renaming the integer constants.
→ The enumeration values are treated as integer variables.

General form:

enum [enum name] {enumerators separated by [variables]};

Eg1: enum week {sun,mon,tue,wed,thr,fri,sat};

0 1 2 3 4 5 6

enum week day;

Eg2: enum {false, true};

0 1

Eg3: enum{ten=10,twenty=20,thirty=30};

Note: The enumerators can not be used a long with console IO functions; for (inputting)making IO through enumerators.

→ Enumerations can be used to increase readability of the code.

Exercise

1) write a program to demonstrate structure

```
struct employee
{
   int no;
   char name[20];
   float sal;
};

main()
{
struct employee e;
clrscr();
printf("\n Enter employee no,name,sal:");
scanf("%d%s%f",&e.no,e.name,&e.sal);
printf("EMPLOYEE DETAILS : \n");
printf("\n NO     : %d",e.no);
printf("\n NAME   : %s",e.name);
printf("\n SALARY  : %f",e.sal);
getch();
}
```

2) write a program to demonstrate array of structures

```
#include<stdio.h>
struct Employee
{
   int no,sal;
   char name[20],job[20];
};
void line()
{
int i;
for(i=1;i<=80;i++) printf("-");
}
main()
{
   struct Employee e[20];    /* array of structures */
   int i,n;
   clrscr();
   printf("\n Enter no. of employee:");
   scanf("%d",&n);
   for(i=1;i<=n;i++)
     {
      printf("\n");
      printf("Enter no : ");
      scanf("%d",&e[i].no);
      printf("Enter Name : ");
      fflush(stdin);
      gets(e[i].name);
      printf("Enter Job  : ");
      gets(e[i].job);
      printf("Enter Salary : ");
      scanf("%d",&e[i].sal);
      }
    for(i=1;i<=n;i++)
      {
         clrscr();
         printf("\n  \t\t\t EMPLOYEE DETAILS \n");
         line();
         printf("\n No      : %d",e[i].no);
         printf("\n Name    : %s",e[i].name);
         printf("\n Job     : %s",e[i].job);
         printf("\n Salary   : %d",e[i].sal);
         printf("\n Press key to  next record..");
         getch();
      }
```

```
getch();
}
```

3) write a program to demonstrate Pointer to Array of structure variables

```
#include<stdio.h>
struct student
{
   int no;
   char name[25],qual[25];
};

void main()
{
   struct student s[50],*p,*t;
   int i,n;
   p=s;
   t=p;
   clrscr();
   printf("How many Students? ");
   scanf("%d",&n);
   for(i=0;i<n;i++)
   {
   printf("Enter No : ");
   scanf("%d",&p->no);
   fflush(stdin);
   printf("Enter Name : ");
   gets(p->name);
   printf("Enter Qualification :");
   gets(p->qual);
   fflush(stdin);
   p++;
   }
printf("\n\t\t Student Details ");
printf("\n No   Name        \t\t Qual");

for(i=0;i<n;i++)
{
printf("\n %d  %-25s %-25s",t->no,t->name,t->qual);
t++;
}
getch();
```

```
}
```

4) write a program to demonstrate initialize a structure

```
struct Item
{
   char name[20];
   int price;
};

void main()
{
   struct Item i ={"Soap",10};
   struct Item *p;
   p=&i;
   clrscr();
   printf(" ITEM  : %s",p->name);
   printf("\n PRICE : %d",p->price);
getch();
}
```

5) write a program to demonstrate structure with functions

```
   struct student
   {
   int rno;
   char name[10];
   float per;
   };

   void display(int n,char *s,float p)
   {
   printf("\n Roll no: %d",n);
   printf("\n Name  : %s",s);
   printf("\n Percentage : %f",p);
   }

   main()
   {
   struct student s={10,"ravi",70.33};
   clrscr();
   display(s.rno,s.name,s.per);
   getch();
```

```
}
```

6) write a program to demonstrate structure with pointers

```
struct point
{
int x;
int y;
};

void increment(struct point *m)
{
m->x++;
m->y++;
}

main()
{
struct point p;
clrscr();
printf("\n Enter x,y coordinates:");
scanf("%d%d",&p.x,&p.y);
printf("\n Before call: (%d,%d)",p.x,p.y);
increment(&p);
printf("\n After call: (%d,%d)",p.x,p.y);
getch();
}
```

7) write a program to demonstrate structure with in a structure

```
struct date
{
int dd;
int mm;
int yy;
};

struct student
{
char name[15];
char class[10];
struct date d;
```

```
};

main()
{
struct student s;
clrscr();
printf("\n Enter name:");
gets(s.name);
printf("\n Enter class:");
gets(s.class);
printf("\n Enter joining date(d-m-y) :");
scanf("%d%d%d",&s.d.dd,&s.d.mm,&s.d.yy);
printf("\n\n STUDENT DETAILS");
printf("\n Name  : %s", s.name);
printf("\n Class : %s",s.class);
printf("\n DOJ   : %d-%d-
%d",s.d.dd,s.d.mm,s.d.yy);
getch();
}
```

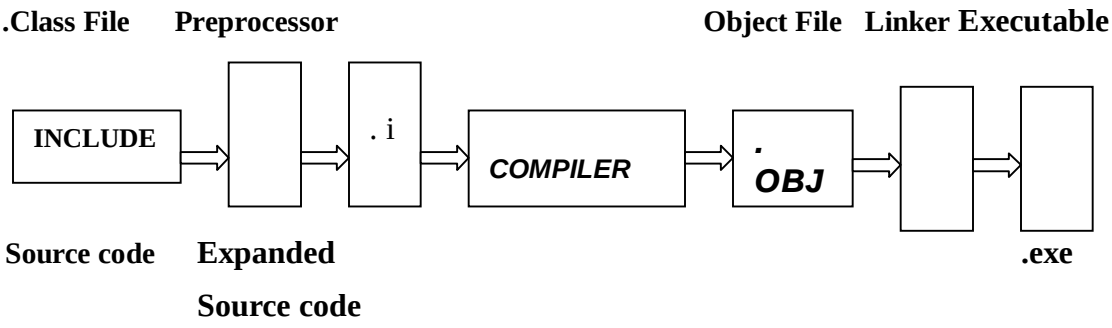
8) write a program to demonstrate union

```
union item
{
int code;
char name[10];
};

main()
{
union item i;
clrscr();
printf("\n Enter item code:");
scanf("%d",&i.code);
printf("\n Item code : %d",i.code);
printf("\n Enter item name:");
scanf("%s",i.name);
printf("\n Item name : %s",i.name);
printf("\n Size of union = %d bytes",sizeof(i));
getch();
}
```

9) write a program to demonstrate typedef

```
typedef struct
{
   char name[25];
   int age;
}person;

main()
{
   person p;
   clrscr();
   printf("Enter Name : ");
   gets(p.name);
   printf("Enter Age : ");
   scanf("%d",&p.age);
   printf("\n Person Details \n");
   printf("\n Name : %s",p.name);
   printf("\n Age  : %d",p.age);
   getch();
 }
```

PREPROCESSORS



Software: The set of programs and closely related with data and documentation.

Preprocessor: A preprocessor is a program that processes of source code before it passed to compilation.

- → The preprocessing features are achieved using preprocessing directives in "C".
- → All the preprocessing directives are generally starts with "#".
- → The preprocessor directives can be placed any where in the program, but in general these are placed before main function.

The preprocessing directives can be classified in to four categories

- ➢ Macro expansion (define, undefined)
- ➢ File inclusion directives (include)
- ➢ Conditional compilation directives.
- ➢ Miscellaneous directives.

- ✓ **The macro expansion directives:** "# define","# undefined" is used to define macros and undefined macros.

General form: #define template definition.

→ "#define" is used to define figurative constants.

Eg: #define pi 3.141

→ #define can be used to define names to operate.

Eg: #define begin {

#define end }

#define and &&

→ #define can be used to define a statement as normally .
#define found printf("item is found")

→ #define can be used to define macro with arguments.
#define max (a,b)(a>b) a:b

main(){ t=max(2,3);.....}

General form: # undifine max*10

```
Main()
{
  .....
.....
#undefine fax
#undifine max 20
.....
.....
```

✓ **File inclusion directives:** (# include) is used to insert the specified file in the source program. In general this directive is placed before the main function.

General function: #include<filename>

→.searches for file in "c" library directory.

#include "filename"

→searches for file in the current directories.

3.conditional compilation directives: used to define which part of our source code is to be compiled.

1.#if #else #elif #endif.

General Form: #if constant condition

```
        Sequence of statement(s);
    #elif constant condition
        sequence of statement(s);
    #else
         sequence of statements(s);
    #endif.
```

Eg:

```
#define country 1
Main()
{
 #if country ==1
        printf ("India");
 #elif country==22
        printf("us");
  #else printf("uk");
   #endif
  }
```

2. #ifdef #else #endif.

General function:

```
 # ifdef macro
        Sequence of statement(s);
 #else
        sequence of statement(s);
 #endif.
```

Eg:

```
#define INDIA
      main()
      {
#ifndef INDIA
      printf("India is my country");
#else
      printf("India is not my country");
#endif
      {
```

3. #ifndef #else #endif

General form:

```
#ifndef macro
      sequence of statement(s);
#else
      sequence of statement(s);
#endif
```

eg:

```
#define INDIA
main()
{
#ifndef INDIA
      printf("India is not my country");
#else
      printf("India is my country");
#endif.
```

4.Miscellenenus directives:

1.#error: can be used to generate fatal errors, which can be helpful in program tracking.

General form:

```
#error message
```

eg:

```
main()
{
int I,n;
printf("%d",n);
#error after two statements
....
....
}
```

2.#pragma: This can be used to define new preprocessing directive.

Pointers

Pointer is a variables which holds the address of another variable. The uses of pointers are:

→ it provides the mechanism of to access a variables out side function in which it is defined.

→ Pointers increase the execution speed.

→ Pointers reduce the length and complexity of program.

→ Pointers provide mechanism to maintain data tables more effectively.

→ A pointer helps to support dynamic memory allocation.

Operators used in pointers:

1. Address operator (&): It is a unary operator to obtain the address of the specified variable.

2. Memory reference operator (*): It is a unary operator to obtain the value at a given address.

3. Pointer to structure operator (→): It is a binary operator to access the members of a given structure variable Address.

Declaration of pointers:

General form: data type *ptr variablename;

Pointer Arithmetic: we can add an integer to a pointer variable .Similarly subtraction also possible .Multiplication and division is not possible.

Void Pointer: (Generic pointers) whenever a pointer is declared as void that it means if can store address of any data type.

Void pointer cannot be dereferencing without explicit type casting because the compiler cannot determine the size of the variable, the pointer point to.

Pointer to Arrays: when an array is declared the name of the array refers to the base address, which is address of starting element in the array.

We can store address of array in another pointer, which is of same type.

Eg: scanf("%d",&list[i]);

Printf("%d",list[i]);

Two dimentional arrays:

list->0th row Address

mlist+i->ith row address

*(mlist+I)+j->ith row0th columm address

*(*mlist+i)+j)->ith row jth column address

Arrays of pointer: Array of pointer is collection of address, which refers to a specific datatype.

General form: datatype *ptrvariable[size];

It can be used to compile non linear data items.

Eg: int a,b,c,d,e,f;

```
int *parray[10];
parray[0]=&a;
parray[1]=&b;
parray[2]=&c;
parray[3]=&d;
parray[4]=&e;
parray[5]=&f

for (i=0;i<6;i++)
    printf("%d",*parray[i]);
```

Note: c supports non-linear and linear data elements and also supports homogeneous and non homogeneous data elements.

Homogeneous-arrays

Non homogeneous-structures

Linear-arrays

Non linear-pointers

Dynamic memory allocation

The memory allocations for the variables are of two types.

1. Static memory allocation:

In this, the memory for the variables are allocated at compile time

2. Dynamic memory allocation:

In dynamic memory allocation, the variable allocated memory at run time.

→ There will be some situation, when the storage requirements of a program can not be known ahead (in advance),in such situations DMA techniques are permits us to allocate memory or release memory at run time.

To achieve dynamic memory allocation,"c" supports the following functions.

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

1. malloc (): <alloc.h> and <stdlib.h>

→ Used to allocate a block of memory.
→ The malloc () function returns the base address of allocated memory block as one.

General form:

Ptrvariable=(cast type*) malloc (size);

Ptr=(int*)malloc(100*sizeof (int);

Note:

1. We cannot give names to the memory that are allocated dynamically.

2. These functions returns null, if these failed to allocate sufficient amount of storage specified in malloc().

3. #define NULL (void*) 0;

2. calloc(): Used allocate memory as block.

→The allocated memory blocks initialized to zero at the time of allocation itself.

General function:

ptrvariable=(casttype x) calloc(no. of items, size of items);

Eg:ptr=(int x)calloc(10,size of I(int));

3. realloc ():used to compress or expand the memory that is previously allocated through malloc /calloc/realloc

→The data in the previous memory block copied to newly allocated block

General form:
ptrvariable=(casttype*)realloc(prevptr,newsize);

4. Free (): This is the responsibility of the user, to make free the memory that is allocated dynamically through.malloc/calloc/realloc functions

This function is used to free the dynamically allocated memory area

General form: -free (pointer variable);

Exercise

1. write a program to demonstrate pointers

```
main()
{
      int a = 10
      int *ptr = &a;
      clrscr();
      printf("\n a = %d",a);
      printf("\n *ptr = %d",*ptr);
      printf("\n &a = %u",&a);
      printf("\n ptr = %u",ptr);
      getch();
}
```

2. write a program to demonstrate pointers

```
main()
{
      int a,b,*ptr;
      clrscr();
      a = 10;
      b = 20;
      ptr = &a;
      printf("\n *ptr = %d",*ptr);
      *ptr = 100;
      printf("\n a   = %d",a);
      ptr = &b;
      printf("\n *ptr = %d",*ptr);
      *ptr = *ptr * 10;
      printf("\n b   = %d",b);
      getch();
}
```

3. write a program to demonstrate pointers to arrays

```
main()
{
      int a[5]={5,4,6,2,1};
      int *ptr,i;
      clrscr();
      ptr = &a;
      printf("\n Array is:");
```

```
        for(i=0;i<5;i++)
        {
        printf("\n %d",*ptr);
        ptr++;
        }
        getch();
}
```

4. write a program to demonstrate pointers with array variable

```
main()
{
        int a[5]={5,4,6,2,1};
        int *ptr;
        clrscr();
        ptr = a+3;
        printf("\n Element in position 3is:%d",*ptr);
        getch();
}
```

5. write a program to demonstrate pointers with functions

```
void swap(int *,int *);
main()
{
        int x,y;
        clrscr();
        printf("Enter 2 no : ");
        scanf("%d%d",&x,&y);
        printf("\nBefore Swaping: x = %d , y =
%d",x,y);
        swap(&x,&y);
        printf("\nAfter Swaping : x = %d , y =
%d",x,y);
        getch();
}
void swap(int *a,int *b)
{
int c;
c=*a;
*a=*b;
*b=c;
```

```
}
```

6. write a program to demonstrate function with pointers

```
void square(int s,int *a,int *p)
{
      *a = s*s;
      *p = 4*s;
}
main()
{
      int s,area,peri;
      clrscr();
      printf("ENTER SIDE : ");
      scanf("%d",&s);
      square(s,&area,&peri);
      printf("\n AREA = %d",area);
      printf("\n PERI = %d",peri);
      getch();

}
```

7. write a program to demonstrate pointer to pointer

```
      main()
      {
      int a=10,*p,**q;
      clrscr();
      printf("\n a = %d",a);
      p=&a;
      printf("\n *p = %d",*p);
      printf("\n p(address of a) = %u",p);
      q=&p;
      printf("\n **q = %d",**q);
      printf("\n q(address of p) = %u",q);
      printf("\n *q(address of a) = %u",*q);
      printf("\n  address of q = %u",&q);
      getch();
      }
```

8. write a program function with pointers to demonstrate strings

```
int slen(char *p)
{
int len=0;
while(*p!='\0')
  {
  len++;
  p++;
  }
return(len);
}

main()
{
char a[10];
int l;
clrscr();
printf("\nEnter string:");
gets(a);
l=slen(a);
printf("\n Length of %s is: %d",a,l);
getch();
}
```

9. write a program to demonstrate calling max() function through pointer

```
int max(int a,int b)
{
if(a>b) return(a);
else return(b);
}

main()
{
int x,y,big;
int (*p)(int a,int b);
p=&max;
clrscr();
printf("\n Enter two no.s:");
scanf("%d%d",&x,&y);
big=(*p)(x,y);
printf("\n Biggest = %d",big);
getch();
}
```

10. write a program to demonstrate factors

```
void factors(int n)
{
int i;
for(i=1;i<=n/2;i++)
  {
  if(n%i==0) printf("%d ",i);
  }
  printf(" %d",n);
}

main()
{
int n;
void (*p)(int n);
p=&factors;
clrscr();
printf("\n Enter a no:");
scanf("%d",&n);
printf("\n Factors of %d :",n);
(*p)(n);
getch();
}
```

11. write a program to demonstrate largest number

```
int largest(int n)
{
int i,no,big=0;
for(i=1;i<=n;i++)
{
printf("\n Enter a no:");
scanf("%d",&no);
if(no>=big) big=no;
}
return(big);
}

main()
{
```

```
int n,res;
int (*p)(int n);
clrscr();
p=&largest;
printf("\n Enter how many no.s:");
scanf("%d",&n);
res=(*p)(n);
printf("\n Biggest = %d",res);
getch();
}
```

12. write a program to demonstrate void pointers

```
main()
{
int a=100;
char b='s';
float c=45.4;
void *p;
clrscr();
p=&a;
printf("\n *p = %d",*(int *)p);
p=&b;
printf("\n *p = %c",*(char *)p);
p=&c;
printf("\n *p = %f",*(float *)p);
getch();
}
```

13. write a program to demonstrate sum of elements of array through pointer

```
main()
      {
      int a[10],i,n,*p,s=0;
      clrscr();
      printf("\n Enter no. of elements:");
      scanf("%d",&n);
      printf("\n Enter elements:");
      for(i=0;i<n;i++) scanf("%d",&a[i]);
      p=a;
      for(i=0;i<n;i++)
       {
       s=s+(*p);
```

```
        p++;
        }
      printf("\n Sum = %d",s);
      getch();
      }
```

14. write a program to demonstrate string pointers

```
main()
{
char a[10]={"aptech"};
char *s;
clrscr();
s=a;
printf("\n The string is:");
while(*s!='\0')
  {
  printf("%c",*s);
  s++;
  }
getch();

}
```

15. write a program to demonstrate matrix multiplication using pointers

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
void main()
{
int **a,**b,**c,r1,r2,c1,c2,i,j,k;
clrscr();
printf("Enter the No.Of Rows & Coloumns of 1st
matrix");
scanf("%d%d",&r1,&c1);
printf("Enter the No.Of Rows & Coloumns of 2nd
matrix");
scanf("%d%d",&r2,&c2);
if(r2!=c1)
printf("\n Multiplication is not possible");
else
{
```

```
printf("Enter the elements of 1st matrix");
a=(int **)malloc(r1 * sizeof(int *));
for(i=0;i<r1;i++)
{
*(a+i)=(int *)malloc(c1 * sizeof(int));
for(j=0;j<c1;j++)
scanf("%d",*(a+i)+j);
}
printf("Enter the elements of 2nd matrix");
b=(int **)malloc(r2 * sizeof(int *));
for(i=0;i<r2;i++)
{
*(b+i)=(int *)malloc(c2 * sizeof(int));
for(j=0;j<c2;j++)
scanf("%d",*(b+i)+j);
}
c=(int **)malloc(r1 * sizeof(int *));
for(i=0;i<r1;i++)
{
*(c+i)=(int *)malloc(c2 * sizeof(int));
for(j=0;j<c2;j++)
*(*(c+i)+j)=0;
}
for(i=0;i<r1;i++)
for(j=0;j<c2;j++)
for(k=0;k<c1;k++)
*(*(c+i)+j)+=(*(*(a+i)+k))*(*(*(b+k)+j));
printf("\n Resultant Matrix is:");
for(i=0;i<r1;i++)
{
printf("\n");
for(j=0;j<c2;j++)
printf("%3d",*(*(c+i)+j));
}
}
getch();
}
```

Files

A file is a place on the disk, where a group of related data is stored permanently

->c supports a no. of functions that have the ability to perform basic file operations

->The file operations include:

1. Naming a file
2. Opening a file
3. Manipulating a file
4. Closing a file

These functions are divided into two categories

1. High level file IO functions (standard /buffered)
2. Low level file IO functions (system level/unbuffered)

->High level file IO divided into two categories

1. Text files
2. Binary files

Defining and opening a file:- If you want to store data in secondary memory, we must specify contain things about the file, i.e file names file pointer, and mode of operation etc

General form for defining a file pointer:

File*file pointer;

Eg: file *fp;

General form opening a file :-

File pointer=fopen(filename, mode);

The file pointer contains the information about the file and i.e. subsequently used as a communication link. Between the hard disk and the program

Opening the file specifies the purpose of opening the file, which is known as mode of operation

Modes of file opening:

W -The file is open for writing

R - Open for reading

A - Open for appending (extending)

w+-same as 'w' both reading and writing

r+ -same as 'r' both reading and writing

a+ -same as 'a 'both reading and writing

Closing a file: once a file is opened is to be closed as soon as all the operations on text file completed

General function:

```
fclose(file pointer);
```

Ex:

```
fclose(fp);
```

INPUT/OUTPUT FILES:

1.getc() & putc() functions:

Putc (): the task of this function to write a character into a file that is opened for writing.

General function:

```
Putc(char,fileptr);
```

Eg:

```
file*fp;
fp=fopen("file name","w");
putch('a',fp)
fclose(fp);
```

getc (): The task of getc()function is to retrieve a character from an existing file that is opened in read mode.

General function: char variable=getc(fp);

Inputting characters into a file:

```
Fp=fopen("file name","w");
Which(ch=getchar())!=EOF )
{
putc(ch,fp);
}
fclose(fp);
```

Outputting characters from a file:

```
fp=fopen("filename","r");
while=((ch=getc(fp))!=EOF)
printf("%c",ch);
fclose(fp);getw(),putw() functions:
```

putw(): This function is used to store an integer in that is opened in write mode.

General function: Putw(integer,filepointer);

getw(): This function is used to retrieve an integer from a file that is opened in read mode.

General function: int variable=getw(fp);

fgets(),fputs():

fputs(): It is used to store a string in a file that is opened in write/append mode.

General function: fputs(string name,filepointer);

fgets(): It is used to store a string in a file that is opened in read node

General function: get(stringname,length,fp);

->The fgets(stringname,length,fp);

->The fgets() will retrieve a string until either it receives a new line (\n)character or the specifies length is received

fscanf(),fprintf():

These functions are used to perform IO on files with mixed data. These are called as mixedIO functions

fprintf(): The task of this function is to write the mixed data into a file that is opened in write mode

general function: print(fp,"controlstring",list of variables);

fscanf(): The fast of function is used to retrieve mixed data from a file that is opened in read mode

general function:

fscanf(fp,"controlstring",address of variables);

->The fscanf() function will return EOF at the end of the file

Binary Files:(Records files)

Mode for Binary files:

wb-opens a binary file in write mode

rb- opens a binary file in read mode

ab- opens a binary file in appending mode

Wb+or w+b-write as well as read modes

rb+or w+b-both read and write modes

Ab+or a+b-in both appending for read and write

Record IO functions : These functions are used to perform IO on binary files

fwrite(): The task of fwrite()function is to store specified number of bytes from a specified memory location into a binary file that is opened in write/append mode

General function:

write (address of data, size of data, number of items file pointer);

fread(): The task of this function is to retrieve specified number of bytes from a binary file that is opened in read mode

General function:

read(address,size,no of items ,file pointer);

Note: fread,fwrite() functions facilitates to perform IO for arrays, structures and any user defined datatypes

Random access on files:

Random access is the concept of retrieving information from a file, from the specified position in the file

fseek:

fseek(fp,offset,origin)

0-seek-set

1-seek-cur

2-seek-end

->fseek() can be used to move the file pointer to the desired location in the file

->The origin tell base for the reference

->The offset tells no of bytes that the file pointer is shifted from the specified origin

eg:seek (fp,50*il,seek-set);

seek (fp,50*il,seek-end);

ferror() functions :

ferror is used to identify any kind of errors during file operations

General form: error (fp);

The ferror() returns none zero if an error occurs otherwise returns zero

Execise

1) write a program to demonstrate inputting and outputting a file

```
#include<stdio.h>
#include<conio.h>
main()
{
   FILE *fp;
   char c;
   clrscr();
   fp = fopen("Dev.dat","w");
   printf("\n Enter data into the file( Ctrl + z to
stop): ");
   while((c=getchar())!=EOF)
     {
     putc(c,fp);
     }
   fclose(fp);
   fp=fopen("Dev.dat","r");
   printf("DATA IS : ");
   while((c=getc(fp))!=EOF)
   {
   printf("%c",c);
   }
   getch();
}
```

2) write a program to demonstrate to make one file data to another

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

```
{
   FILE *sfp,*dfp;
   char sfile[20],dfile[20],ch;
   clrscr();
   printf("Source File Name With Extension: ");
   gets(sfile);
   printf("Destination File Name With
Extension:");
   gets(dfile);
   sfp = fopen(sfile,"w");
   printf("\n  Enter data into Source File:");
   while((ch=getchar())!=EOF)
     {
     putc(ch,sfp);
     }
   fclose(sfp);
   sfp=fopen(sfile,"r");
   dfp=fopen(dfile,"w");
   while((ch=getc(sfp))!=EOF)
     {
     putc(ch,dfp);
     }
   fclose(sfp);
   fclose(dfp);
   printf("\n DATA TRANSFERED");
   dfp=fopen(dfile,"r");
   printf("\n Data in Destination File:");
   while((ch = getc(dfp))!=EOF)
     {
     printf("%c",ch);
     }
   fclose(dfp);
   getch();
}
```

3) write a program to demonstrate to find even & odd files

```
#include<stdio.h>
#include<conio.h>
void main()
{
int i,n,num;
FILE *f1,*f2,*f3;
```

```
clrscr();
printf("\n Enter  how many no.s:");
scanf("%d",&n);
f1=fopen("main.dat","w");
for(i=1;i<=n;i++)
{
printf("\n Enter a no:");
scanf("%d",&num);

if(num==-1) break;
else putw(num,f1);
}
fclose(f1);
f1=fopen("main.dat","r");
f2=fopen("even.dat","w");
f3=fopen("odd.dat","w");
while((num=getw(f1))!=EOF)
 {
 if(num%2==0) putw(num,f2);
 else putw(num,f3);
 }
fclose(f1);
fclose(f2);
fclose(f3);
printf("\n Even file: ");
f2=fopen("even.dat","r");
while((num=getw(f2))!=EOF)
 {
 printf("%d ",num);
 }
fclose(f2);
printf("\n Odd file :");
f3=fopen("odd.dat","r");
while((num=getw(f3))!=EOF)
 {
 printf("%d ",num);
 }
fclose(f3);
getch();
}
```

4) write a program to demonstrate files using structures

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

```
#include<conio.h>
struct address
{
char name[10],dno[10],street[10];
};

void main()
{
struct address a;
FILE *fp;
clrscr();
fp=fopen("address.dat","w");
printf("\n Enter name:");
scanf("%s",a.name);
printf("\n Enter door no:");
scanf("%s",a.dno);
printf("\n Enter street:");
scanf("%s",a.street);
fprintf(fp,"%s\n%s\n%s",a.name,a.dno,a.street)
;
fclose(fp);
fp=fopen("address.dat","r");
printf("\n Data in the file is :");
while(!feof(fp))
{
fscanf(fp,"%s%s%s",a.name,a.dno,a.street);
printf("%s \n %s \n %s",a.name,a.dno,a.street);
}
fclose(fp);
getch();
}
```

Command line arguments:

C it is possible to pass arguments to main function also. These arguments are called command line arguments

Syntax: main (int *argc char *argr [])

{

.....

}

Eg: main (int argc, char *argr [])

{

int I;

for(I=;I<=args;++)

printf("%s",args[i];

note:args is that the possible argument vector used to store the argument values