Contents

Unit-I

Fundamentals of "C" C Program, branching in C, if Statement, if-else statement, nested if, go to statement, else-if statement, switch break statements, loops, while, do-while, for, nesting of loops.

1.1 HISTORY OF DEVELOPMENT OF C

By 1960 a number of computer languages had come into existence, almost each for a specific purpose. For example, COBOL was being used for commercial applications, FORTRAN for Engineering and Scientific Applications and so on. Instead of learning and using this many languages the international committee came out with a language called ALGOL 60. However, ALGOL 60 never became popular because it seemed too abstract, too general. To reduce this abstractness and generality, a new language called combined programming language (CPL) was developed at Cambridge University. CPL was an attempt to bring ALGOL 60 down to earth. However, CPL turned out to be so big, having so many features, that it was hard to learn and difficult to implement.

Basic Combined Programming Language (BCPL), developed by Martin Richards at Cambridge University aimed to solve this problem by bringing CPL down to its basic good features. But unfortunately it turned out to be too less powerful and too specific. Around same time a language called B was written by Ken Thompson T's Bell Labs. In 1972 a person named Dennis Ritchie inherited the features of B and BCPL and developed a new Language called C.

1.2 ADVANTAGES & DISADVANTAGES OF C PROGRAMMING LANGUAGE

(OR)

Features of C

C is the most popular programming language, C has many advantages:

1. Procedural Oriented language.

C Language is procedure oriented language, Here user creates procedures or functions to execute their task. Procedure oriented language is very much easy to learn because it follows algorithm to execute your statements. To develop program using procedure oriented language, you need to draw/prepare algorithm and then start converting it into procedure or functions.

2. Lots of Libraries

C Language provides lots of functions which consist of system generated functions and user defined functions. C Compiler comes with list of header files which consist of many general functions which can be used to develop program. while programmer can also create function as per their requirements that is called as user generated/defined function.

3. Speed Compilation

C compiler produces machine code very fast compared to other language compiler. C compiler can compile around 1000 lines of code in a second or two. One more benefit of the C Compiler is that it also optimizes the code for faster execution.

4. Easy to Learn (Syntax is near to English Language)

C Language syntax is very easy to understand. It uses keyword like if, else, goto, switch, goto, main, etc. This kind of keyword we all are using in our day to day life to convey meaning or to get some decisions.

5. Middle level language

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

6. General purpose programming language:

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

7. Portability

We can compile or execute C program in any operating system (unix,dos,windows)

8. Powerful programming language

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

C Language Disadvantages:

Every coin has two sides, as C Language has also some disadvantages. C Language has not any major disadvantages but some features is missing in the C Language, obviously that's why C Language is very much powerful now.

1. Object Oriented Programming Features (OOPS)

Object Oriented Programming Features is missing in C Language, You have to develop your program using procedure oriented language only.

2. Run Time Type Checking is Not Available

In C Language there is no provision for run time type checking, for example i am passing float value while receiving parameter is of integer type then value will be changed, it will not give any kind of error message.

3. Namespace Feature

C does not provide namespace features, so you can't able to use the same variable name again in one scope. If namespace features is available then you can able to reuse the same variable name

1.3 INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)

An **integrated development environment** (IDE) also known as *integrated design environment* or *integrated debugging environment* is a software application that provides facilities to computer programmers for software development. An IDE normally consists of a:

- Source code editor
- Compiler and/or interpreter
- Build automation tools
- Debugger

1.4 PROCESS OF CREATING, COMPILING AND RUNNING A C PROGRAM

The steps involved in building a C program are:

1. First program is created by using any text editor and the file is stored with extension as .c we call this file as source file.
2. Next the program is compiled. There are many compilers available like Sun compiler, Borland compiler. In TurboC environment we press **Alt+F9** for compilation.
3. At this stage errors like typing mistake, mistake in key words, syntax usage errors will be detected and will displayed by the compiler. The programmer should correct the errors. But a programmer must note that compiler cannot detect logical errors.
4. After this process the compiled code will be stored in an object file.
5. Press **Ctrl+F9** to link object code with system library and to execute the program by creating an executable file.
6. Give the required input on console screen and press enter key. To view the results of the program press **Alt+F5**

1.5 CHARACTER SET OF C LANGUAGE:

C compiler allows to use many no. of characters :

Lower case letters : a b c … z

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

1.6 BASIC STRUCTURE OF C PROGRAM

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

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

function2(sub program)
{
local declarations:
 .
 code.
 .
}
```

Example:

```
#include<stdio.h> /* pre processor */

void fun1( ) /* function pro.type */

int x=10; /* global declaration */

main ( )
{
int y=20;
printf( " this is main block");
printf ("\n sum=%d",x+y); /* prints 30*/
fun1( ) /* calling of function*/
getch( );
}
void fun1( )
{
int z=30;
printf( " this is fun2 block");
printf ("\n sum=%d",x+z); /* prints 40*/
}
```

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

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

Global declarations: It allows to declare variables as global whose scope is valid throughout the program.

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

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

Local declarations: Every function at its beginning may have local variables whose scope is valid within the function only.

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

1.7 TOKENS OF C PROGRAM

The smallest individual unit in c program is called token. The tokens in c program are listed below:

- Keywords
- Variables
- Constants
- Special characters
- Operators

Keywords

The C keywords are reserved words and have fixed meanings. There are 32 keywords in C. the list is given below

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

Variable

A variable is used to store values. Every variable has memory location. The memory locations are used to store the values of the variables. The variables can be of any type. The variable can hold single value at a time. The program can change value of variable.

<u>Rules for naming a variable</u>

- Variable name contain a sequence of letters and digits, with a letter as a first character.
- A variable name should not be a reserved word
- Lower case letters are preferred. However, the upper case letters are also permitted. But C compiler treats lower and upper case letters differently.
- The _ underscore symbol can be used in variable name. In general, underscore is used to link two words when the length of variable name is long.
- The latest C compilers allow the _ underscore character as first character of variable name.

Examples for variable names

Valid variable names	**Invalid variable names**
sum pieceFlag i	Sum$value ($ is not a valid character) Piece Flag (space not allowed) 3Spencer (it should not begin

J5x7 Number_of_moves	with digit) int (int is a reserved word)

Variable declaration

Syntax:

data type variable_name;

Or

data type variable_name1, variable_name2 …….. variable_ name_n;

Example

int a,b,c;

float x;

Constants

The constants in c are applicable to the values which do not change during execution of program. C supports various types of constants including integers, characters, floating and string constants.

C has two types of constants: literal and symbolic.

Literal constant:

A literal constant is directly assigned to a variable. Consider the following example

int x=5;

where x is a variable of type int and 5 is a literal constant.

Symbolic constant:

The symbolic constant can be created in the following ways

1. #define
2. the const keyword

The #define preprocessor directive can be used for defining constants.

Example: # define price 152

We can also define a constant using const keyword

Example: const int price=152;

Special symbols

In c program we use different special symbols such as #, &, ?, :, % etc.

operators

C supports wide variety of operators such as

- Arithmetic operators
- Logical operators
- Relational operators
- Bit wise operators
- Special purpose operators

1.8 IDENTIFIERS:

- Identifiers are names of variables, functions and arrays etc. they are user-defined names.
- Identifier contains a sequence of letters and digits, with a letter as a first character. Lower case letters are preferred. However, the upper case letters are also permitted. But C compiler treats lower and upper case letters differently.
- The _ underscore symbol can be used in an identifier. In general, underscore is used to link two words in long identifiers.
- *Identifiers* provide names for the following language elements:
    - Functions
    - Function parameters
    - Arrays
    - Pointers
    - Strings
    - Macros and macro parameters
    - Type definitions
    - Enumerated types and enumerators
    - Structure and union names

1.9 OPERATORS AVAILABLE IN C

OPERATORS IN C

An operator is a symbol that operates on a certain data type and produces the output as the result of the operation.

Category of operators

Unary Operators
A unary operator is an operator, which operates on one operand.

Binary
A binary operator is an operator, which operates on two operands

Ternary
A ternary operator is an operator, which operates on three operands.

C has a rich set of operators which can be classified as

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

Arithmetic Operators
The arithmetic operator is a binary operator, which requires two operands to perform its operation of arithmetic. Following are the arithmetic operators that are available

Operator	Description
+	Addition
-	Subtraction
/	Division
*	Multiplication

%	Modulo or remainder

At the time of using the '/' division operator on two integers, the result will also be an integer. E.g. 100/8 will result in 12 and not 12.50.

Relational Operators

Relational operators compare two operands and return true or false i.e. 1 or 0.

Following are the various relational operators

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

the equality operators is related to the relational operators is as follows:

==	Equal to
!=	Not equal to

Logical Operators

A logical operator is used to compare or evaluate logical and relational expressions. There are three logical operators available in the C language.

Operator	**Meaning**
&&	Logical AND
| |	Logical OR
!	Logical NOT

Truth table for Logical OPERATORS

Operand 1	Operand 2	operand 1 && operand 2	operand 1| | operand 2	!operand1
1	1	1	1	0

1	0	0	1	0
0	1	0	1	0
0	0	0	0	1

Increment and Decrement Operators

These operators are unary operators. The ++ operator increments the value of the variable by one, whereas the – operator decrements the value of the variable by one.

These operators can be used in either the postfix or prefix notation as follows:

Postfix: a++ or a--
Prefix: ++a or --a

Postfix notation

In the postfix notation, the value of the operand is used first and then the operation of increment or decrement takes place, e.g. consider the statements below:

```
int a,b;

a = 10;

b = a++;

printf("%d\n",a);

printf("%d\n",b);
```

Program Output

```
10
11
```

Prefix notation

In the prefix notation, the operation of increment or decrement takes place first after that the new value of the variable is used.

```
int a,b;

a = 10;

b = ++a;

printf("%d\n",a);

printf("%d\n",b);
```

Program Output

```
11
11
```

Assignment Operator

An assignment operator (=) is used to assign a constant or a value of one variable to another.

Example:

a = 5;

b = a;

interestrate = 10.5;

result = (a/b) * 100;

Important Note:

Remember that there is a remarkable difference between the equality operator (==) and the assignment operator (=). The equality operator is used to compare the two operands for equality (same value), whereas the assignment operator is used for the purposes of assignment.

Multiple assignments:

You can use the assignment for multiple assignments as follows:

a = b= c = 10;

At the end of this expression all variables a, b and c will have the value 10. Here the order of evaluation is from right to left. First 10 is assigned to c, hence the value of c now becomes 10. After that, the value of c (which is now 10) is assigned to b, hence the value of b becomes 10. Finally, the value of b (which is now 10) is assigned to a, hence the value of a becomes 10.

Arithmetic Assignment Operators / compound assignment statements

Arithmetic Assignment operators are a combination of arithmetic and the assignment operator. With this operator, the arithmetic operation happens first and then the result of the operation is assigned.

Operators	**Example**	**Expands as**
+=	sum+=3	sum = sum + 3

-=	count-=4	count = count - 4
*=	factorial*=num	factorial = factorial * num
/=	num/=10	num = num /10
%=	a%=3	a = a % 3

Conditional Operator

A conditional operator checks for an expression, which returns either a true or a false value. If the condition evaluated is true, it returns the value of the true section of the operator, otherwise it returns the value of the false section of the operator.

Its general structure is as follows:

```
expression1 ? expression 2 (True Section): expression3 (False Section)
```

Example:

a=3,b=5,c;

c = (a>b) ? a+b : b-a;

The variable c will have the value 2, because when the expression (a>b) is checked, it is evaluated as false. Now because the evaluation is false, the expression b-a is executed and the result is returned to c using the assignment operator.

Bit wise operators:

The bitwise operators operate on sequences of binary bits. Bitwise operations are necessary for much low-level programming, such as writing device drivers, low-level graphics.

Bitwise operators include:

&	AND
|	OR
^	XOR

~	one's compliment
<<	Shift Left
>>	Shift Right

Special operators

The operators such as sizeof and comma are treated as special operators.

sizeof Operator:

The **sizeof** operator gives the amount of storage, in bytes, required to store an operand.

syntax

sizeof unary-expression

sizeof (type-name)

Example : sizeof(char) is equal to 1.

Comma operator

We can combine multiple expressions in a single expression using the comma operator

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

{
  int i,j,k;
  k = (i = 4, j = 5);
  printf("k = %d",k);
}
```

output k=5

Comma operator returns the value of the rightmost operand.

```
#include<stdio.h>
main(){
   int i, j;
   printf("%d",(i = 0, j = 10));
}
```

output : 10

1.10 BIT WISE OPERATORS IN C

The bitwise operators operate on sequences of binary bits. Bitwise operations are necessary for much low-level programming, such as writing device drivers, low-level graphics.

Bitwise operators include:

&	AND
|	OR
^	XOR
~	one's compliment
<<	Shift Left
>>	Shift Right

Bitwise AND (&):

The & operator performs a bitwise AND on two integers. Each bit in the result is 1 only if both corresponding bits in the two input operands are 1.

Example:

```
  main()
 {
  unsigned int a = 60; /* 60 = 0011 1100 */
  unsigned int b = 13; /* 13 = 0000 1101 */
  unsigned int c = 0;

  c = a & b;              /* 12 = 0000 1100 */
 }
```

Bitwise OR (|)

The | (vertical bar) operator performs a bitwise OR on two integers. Each bit in the result is 1 if either of the corresponding bits in the two input operands is 1.

Example:

```
main()
 {
   unsigned int a = 60; /* 60 = 0011 1100 */
   unsigned int b = 13; /* 13 = 0000 1101 */
   unsigned int c = 0;

   c = a | b;              /* 61 = 0011 1101 */
 }
```

Bitwise XOR (^)

The ^ (caret) operator performs a bitwise exclusive-OR on two integers. Each bit in the result is 1 if one, but not both, of the corresponding bits in the two input operands is 1.

Example :

```
main()
 {
   unsigned int a = 60; /* 60 = 0011 1100 */
   unsigned int b = 13; /* 13 = 0000 1101 */
   unsigned int c = 0;

   c = a ^ b;              /* 49 = 0011 0001 */
 }
```

Left shift (<<)

The << operator shifts its first operand left by a number of bits given by its second operand, filling in new 0 bits at the right

Example

```
main()
```

```
{
  unsigned int Value=4;          /*  4 = 0000 0100 */
  unsigned int Shift=2;

  Value = Value << Shift;        /* 16 = 0001 0000 */

}
```

Right shift (>>)

The >> operator shifts its first operand right by number of bits given by its second operand filling in new 0 bits at the left.

```
main()
{
  unsigned int bytes=256; /* 00000000 00000000 00000000 10000000 */

     printf("%3d \n", bytes);
     bytes >>= 1;         /*128= 00000000 00000000 00000000 01000000 */
  }
```

Complement (~)

The ~ (tilde) operator performs a bitwise complement on its single integer operand. Complementing a number means to change all the 0 bits to 1 and all the 1s to 0s.

1.11 PRECEDENCE AND ASSOCIATIVITY OF OPERATORS IN C

Each Operator in C has a precedence associated with it. This precedence is used to determine how an expression involving more than one operator is evaluated. There are distinct levels of precedence and an operator may belong to one of the levels. The operators at the higher level of precedence are evaluated first. The operators of the same precedence are evaluated either from left to right or from right to left, depending on the level. The direction in which the operators is evaluated is known as the associativity property of an operator.

Operator	**Operation**	**ASSOCIATIVITY**	**RANK**
() []	Function call Array subscript	Left to right	1
! ~ + - ++ -- & * sizeof	Logical negation (NOT) Bit-wise 1's complement Unary plus Unary minus Pre increment or post Increment Pre decrement or post decrement Address Indirection Returns size of operand in	Right to left	2
* / %	Multiply Divide Remainder	Left to right	3
+ -	Binary plus Binary minus	Left to right	4
<< >>	Shift left Shift right	Left to right	5
< <=	Less than Less than or equal	Left to right	6

> >=	Greater than Greater than or equal		
= = !=	Equality Not equal to	Left to right	7
& ^ |	Bit wise AND Bit wise XOR Bit wise OR	Left to right Left to right Left to right	8 9 10
&& | |	Logical AND Logical OR	Left to right Left to right	11 12
? :	Work like simple if – else	Right to left	13
= *= /= %= += -=	Simple assignment Assign product Assign quotient Assign modulus Assign sum Assign difference	Right to left	14
,	Evaluate	Left to right	15

1.12 INPUT AND OUTPUT STATEMENTS AVAILABLE IN C

In C the input/output statements can be divided into two types

- Formatted input/output statements
- Unformatted input/output statements.

Formatted input/output statements:

scanf(): reads formatted text form keyboard. The general form is given below.

```
scanf("control string", arg1,arg2,.....argn);
```

The control string specifies the field format in which the data is to be entered and the arguments arg1, arg2….argn specify the address of locations where the data is stored . control string and arguments are separated by commas.

Example : scanf(“%d%d”, &n1, &n2);

Code	Meaning
%c	Read a single character
%d	Read a decimal integer
%f	Read a floating value
%s	Reads a string
%u	Reads unsigned integer
%ld	Reads long integer
%lf	Reads double value
%Lf	Reads long double values

Scanf format codes

printf() : prints formatted text on monitor. The general form is given below.

printf(“control string”, arg1,arg2,…..argn);

Control string consists of three types of items:

1. Characters that will be printed on the screen as they appear.
2. Format specifications that define the output format for display of each item
3. Escape sequence characters such as \n, \t, and \b.

The arguments are the variables whose values are formatted and printed.

fscanf() : the general format of fscanf is

fscanf(fp, “control stirng”, list);

This statement would cause the reading of the items in the list from the file specified by fp, according to the specifications contained in the control string.

Example:

fscanf(f2, “%s %d”, item, &qty);

fprintf(): the general format of fprintf is

fprintf(fp, “control stirng”, list);

where fp is a file pointer associated with a file that has been opened for writing. The control string contains output specifications for the items in the list. The list may include variables, constants and strings.

Example :

fprintf(fp, “%s%d%f”, name,age,7.6);

UNFORMATTED INPUT / OUTPUT STATEMENTS:

Function	Description	Example
getchar()	Reads a character and the console remains until we press return key	Ch=getchar()
getche()	Reads a character and shows on screen but it will not wait for return key	Ch=getche()
getch()	Reads a character and does not show on the screen and also it will not stop wait for return key	Ch=getch()
gets()	Reads a string it also accepts spaces in name, it stops reading when return key pressed	gets(name)
putchar()	Prints a character on the screen	Putch(ch)
puts()	Prints a name on the screen	Puts(name)
getc()	Reads a character from a stream	Ch=getc(fp)
putc()	Prints a character to file	putc(ch,fp);

1.13 CONVERSION CHARACTERS(FORMAT SPECIFIERS):

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

%d : prints integer

%f : prints float

%c : prints character

%s : prints string

%o : prints octal value

%u : prints unsigned integer

%x : prints hexa decimal value

1.14 ESCAPE SEQUENCES

Character combinations consisting of a backslash (\) followed by a letter or by a combination of digits are called "escape sequences." The following table shows various escape sequence character in C

Escape Sequence	**Represents**
\a	Bell (alert)
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\'	Single quotation mark
\ "	Double quotation mark
\\\\	Backslash
\?	Literal question mark

1.15 COMMENTS IN C

A "comment" is a sequence of characters beginning with a forward slash and asterisk combination (**/***). It is treated as a single white-space character by the compiler.. A comment can include any combination of characters. The comment ends with sequence of characters asterisk and forward slash*/

Example :

```
/* Comments can contain keywords such as
   for and while without generating errors. */
```

1.16 ARITHMETIC EXPRESSIONS IN C

An expression is a combination of variables constants and operators written according to the syntax of C language. In C every expression evaluates to a value. Some examples of C expressions are shown in the table given below.

algebraic expression	c expression
$y=x^2+2$	y=(x*x)+2
a=lb	a=l*b

1.17 CONTROL STRUCTURE

One of the statements in a programming language which determines the sequence of execution of other instructions or statements (the control flow) are called control structure.

Different types of control structures are listed below:

- Conditional control structure
- Loop control structure
- Jump control structure
- Multi way conditional control structure / case control structure

1.18 CONDITIONAL CONTROL STATEMENTS IN C (DECISION MAKING)

The conditional control structure includes the following :

- **if**
- **if else**

- **nested if else**
- **else if ladder**

<u>if statement:</u>

if statement checks whether the test expression inside parenthesis ( ) is true or not. If the test expression is true, statement(s) inside the body of if statement is executed but if test is false, statement(s) inside body of if is ignored.

Syntax:

```
if (test expression)
{
statement(s ) to be executed if test expression is true;
}
```

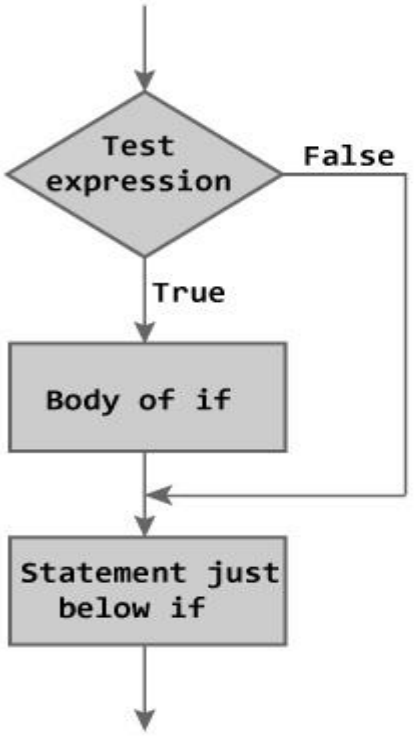


Figure: Flowchart of if Statement

<u>if else statement</u>

The if...else statement is used if the programmer wants to execute some statement/s when the test expression is true and execute some other statement/s if the test expression is false.

Syntax

```
 if (test expression)
  {
     statements to be executed if test expression is true;
  }
 else
  {
     statements to be executed if test expression is false;
 }
```

Flowchart of if...else statement

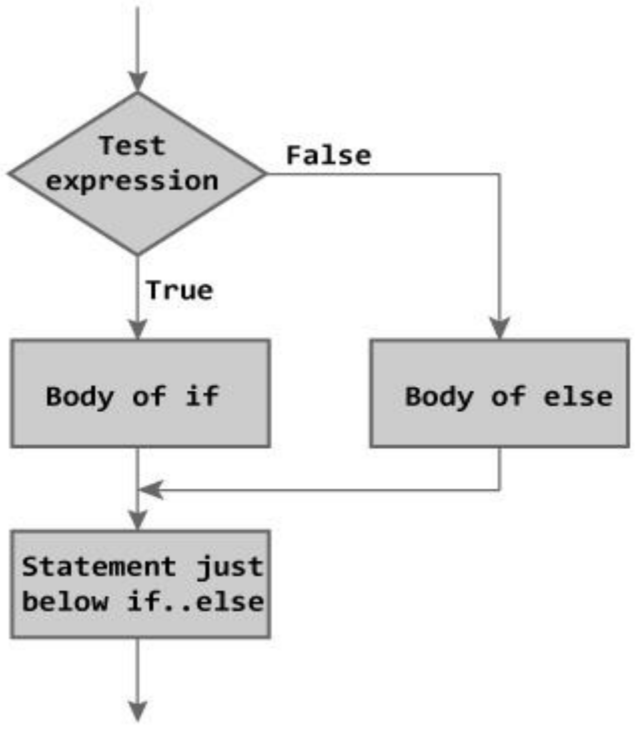
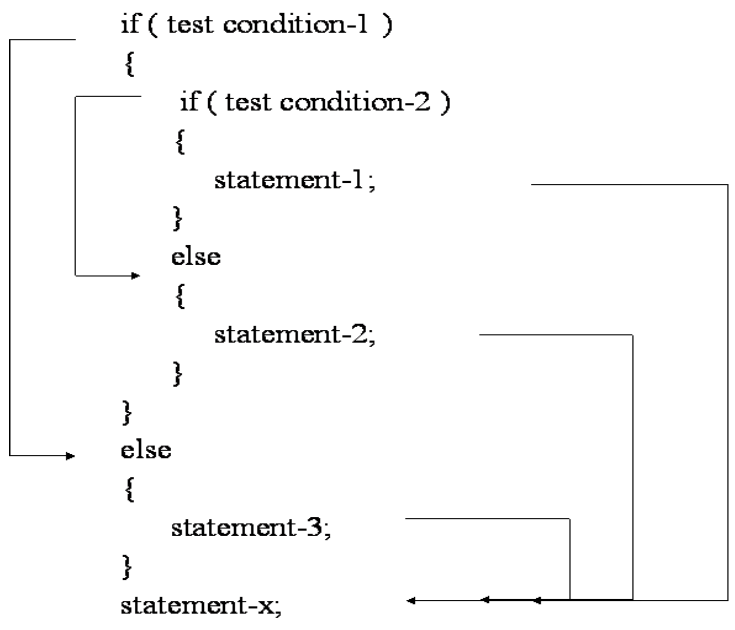


Figure: Flowchart of if...else Statement

nested if else statement

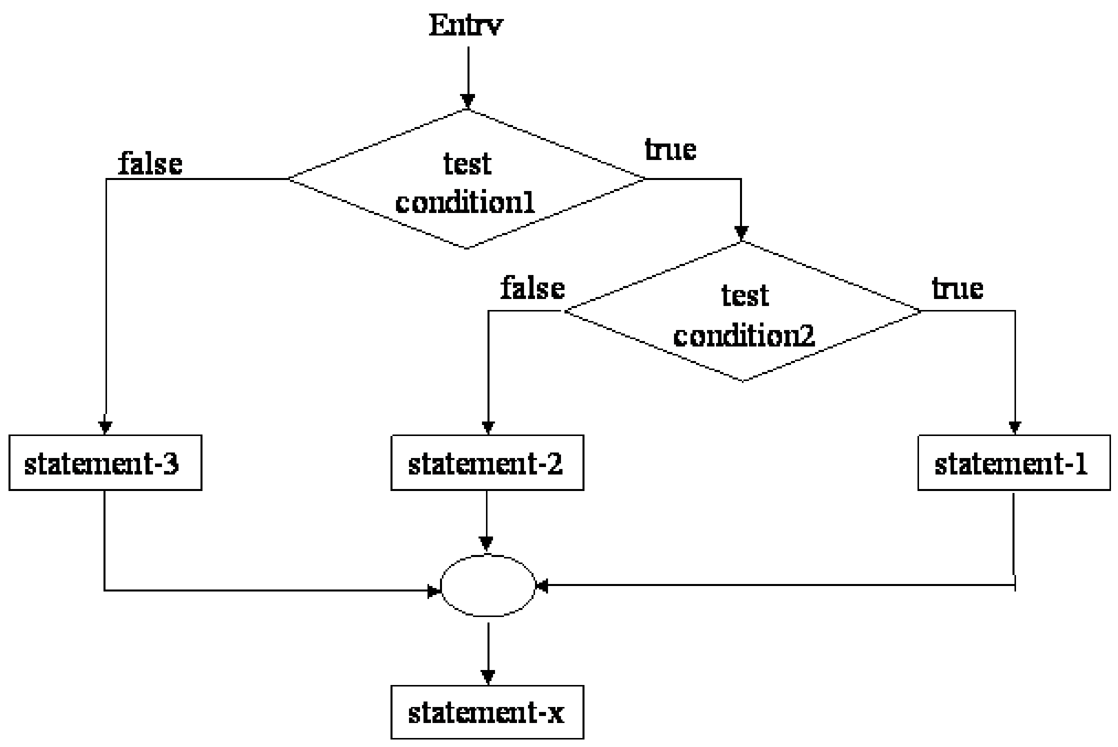
When a series of decisions are involved, we may have to use more than one **if else** statement in nested form.

Syntax:



```
if ( test condition-1 )
{
    if ( test condition-2 )
    {
        statement-1;
    }
    else
    {
        statement-2;
    }
}
else
{
    statement-3;
}
statement-x;
```

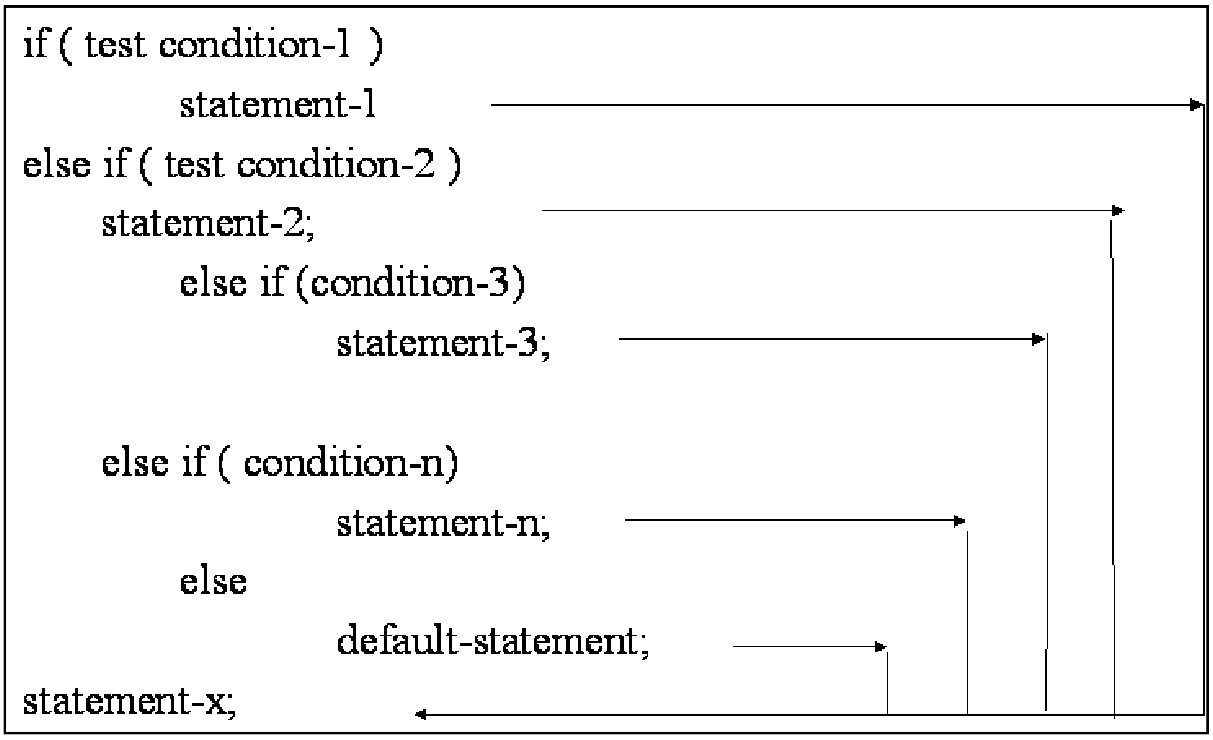
<u>The flow chart for the nested if else statement is given below:</u>



<u>else if ladder</u>

When multi path decisions are involved we use chain of **if else** statements. It forms else if ladder.

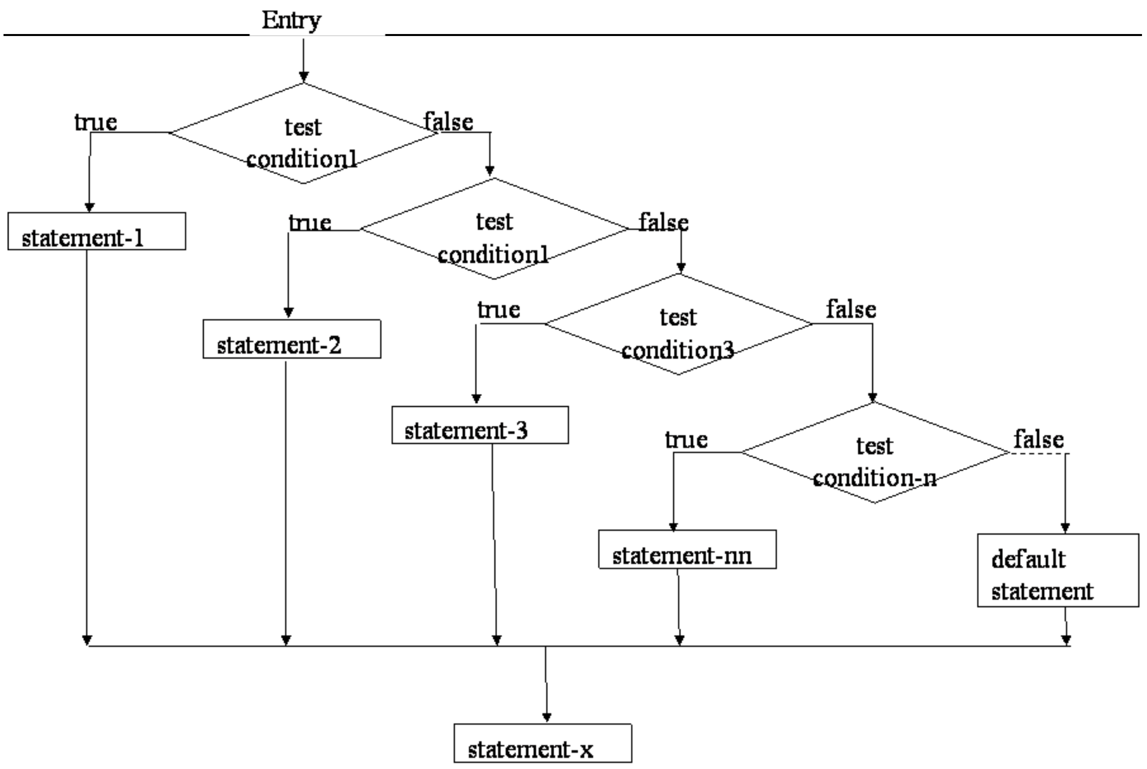
Syntax:



```
if ( test condition-1 )
          statement-1
else if ( test condition-2 )
     statement-2;
          else if (condition-3)
                    statement-3;

     else if ( condition-n)
                    statement-n;
          else
                    default-statement;
statement-x;
```

The flow chart for else if ladder is given below:



1.19 LOOP CONTROL STATEMENT IN C

Loop : A loop is a sequence of statements which is specified once but which may be carried out several times until some condition becomes false.

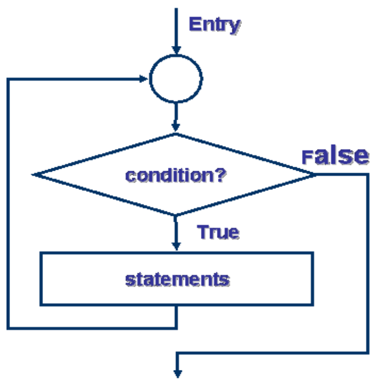
In C the loop control statements are :

- while loop
- do while loop
- for loop

<u>while loop</u>

The while loop executes a simple statement or a block of statements until the conditional expression becomes FLASE.

<u>The flow chart for the while loop is given below:</u>



The general form of while loop is given below

```
while (condition)
{
   statement block
}
```

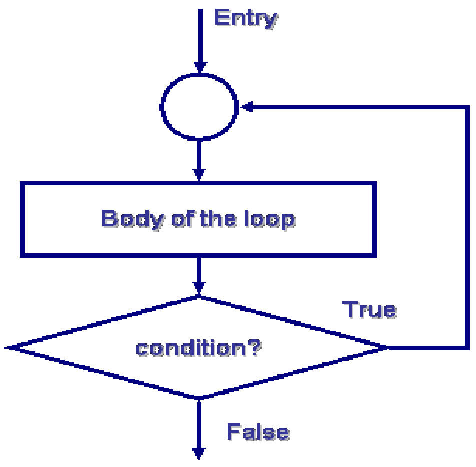
Example :

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

<u>do while loop</u>

On some occasions, it might be necessary to execute the body of the loop before the test is performed. This can be handled by *do…while* statement.

The flow chart for do while is given below:



The general form of the do loop is

```
do
{
    statement block
} while (condition);
```

Example :

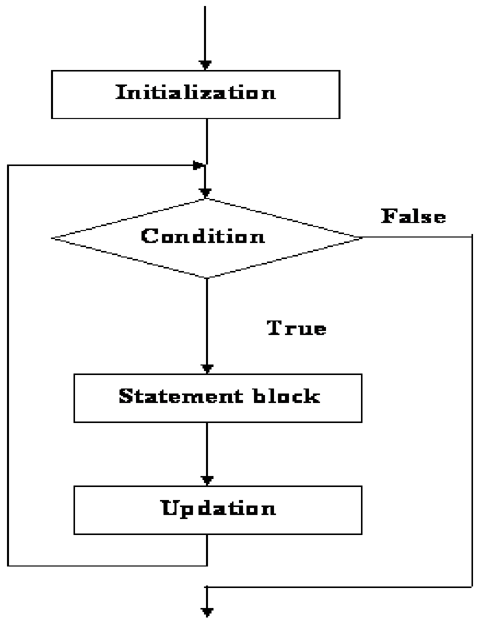
```
#include <stdio.h>
main( ) {
  int i = 0;
  do {
    printf("\t%d", i);
    i++;
  } while (i<10);
  }
```

The above example displays 0 to 9 digits as output

for loop

The for loop is also an entry controlled loop like while loop. In this loop we combine initialization, condition and updation at one location.

The flow chart of for loop is given below:



The general form of for loop is given below

```
for (initialization; condition; updation)
{
   statement block
}
```

- The initialization is used to set a counter variable to its starting value.
- The condition is generally a relational statement that checks the counter variable against a termination value
- the updation increments (or decrements) the counter value. The loop repeats until the condition becomes false.

Example:

The following code will print **hello** ten times:

```
for (t=0; t<10; t++)
printf("Hello\n");
```

1. 20 JUMP CONTROL STATEMENTS IN C

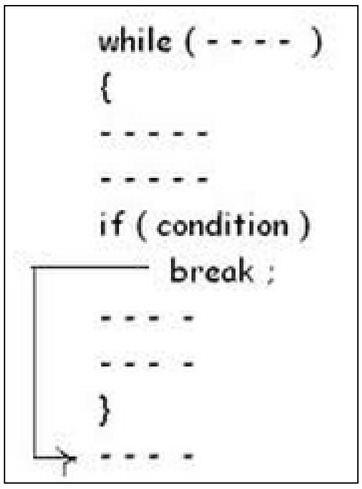
The jump control statements in C include the following

- **break statement**
- **continue statement**
- **goto statement**

break statement

- **break** is used to exit from a do, for, or while loop.
- It is also used to exit from a switch statement.
- When loops are nested, the **break** would only exit from the loop contains it.
- It means the **break** will exit only a single loop.

An example of break statement



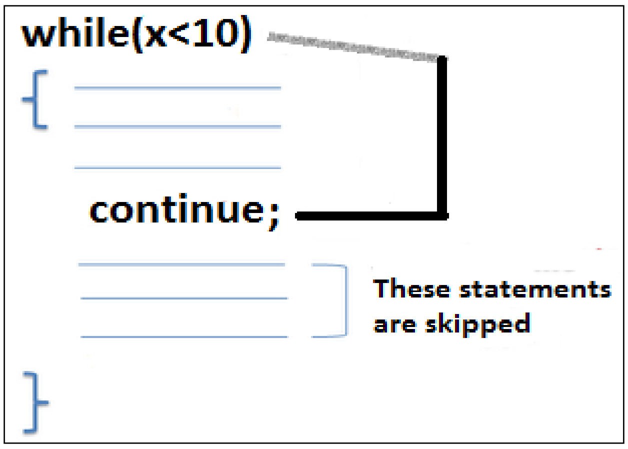
```
while ( - - - - )
{
- - - - -
- - - - -
if ( condition )
      break ;
- - - -
- - - -
}
- - - -
```

:

continue statement:

- The *continue* statement skips the remaining statements in the body of the loop, and continues with the next iteration of the loop.
- The *continue* statement can only be used within the body of a *while*, *do*, or *for* statement.

An example of continue statement is given below:



goto statement

- C supports the goto statement to branch unconditionally from one point to another in the program.
- The goto requires a label in order to identify the place where the branch is to be made.
- The general forms of goto and label statements are shown below:

```
goto label;
……
label:
statement;
```

```
label;
statement;
……
goto label ;
```

1.21 MULTI WAY CONDITIONAL STATEMENT (OR) CASE CONTROL STATEMENT

Multi way conditional statement (or) switch case statement:

switch-case statement is used to select one of many alternatives.

The general form of switch-case statement is given below:

```
switch (e) {
   case c1:
      statements1
      break;
   case c2:
      statements2
      break;
    …more case clauses…
   default:
      statementsdef
      break;
}
```

- *e* is the control expression, which is used to choose what statements are to be executed;
- *ci* is a constant value;
- *statements*$_i$ is a sequence of statements to be executed if *ci* is equal to *e*;
- *statementsdef* is a sequence of statements to be executed if none of the *ci* values match the expression *e*.

Rules for switch-case statements

- The expression *e* must be an integral type.
- Case labels *ci* must be:
    - Constants or constant expressions.
    - Unique. No two labels can have the same value.
    - Must end with colon.
- The *break* statement transfers the control out of the *switch* statement.
- The *default* label is optional.

- The *default* may be placed anywhere but usually placed at the end.
- It is permitted to nest *switch* statements

Example:

```
#include <stdio.h>
 int main ()
{
   /* local variable definition */
   char grade = 'B';
   switch(grade)
   {
   case 'A' :
      printf("Excellent!\n" );
      break;
   case 'B' :
   case 'C' :
      printf("Well done\n" );
      break;
   case 'D' :
      printf("You passed\n" );
      break;
   case 'F' :
      printf("Better try again\n" );
      break;
   default :
      printf("Invalid grade\n" );
   }
   printf("Your grade is  %c\n", grade );
    return 0;
}
```

1.22 DIFFERENCE BETWEEN WHILE LOOP & DO WHILE LOOP

while loop	**do while loop**
1. It is entry controlled loop. It means in this loop the condition is tested at top of the loop.	1. It is the exit control loop. It means in this loop the condition is tested at the bottom of the loop
2. In this loop if the condition is true the body of the loop will be executed. Here the minimum no. of executions may be zero.	2. In this loop, the body of the loop will be executed with the testing the condition for first time. Here the minimum number of executions is one.
3. In this loop the while keyword followed by test condition is not terminated with semi colon.	3. In this loop the while keyword followed by test condition is terminated with a semi colon

4. Syntax:
while (condition)
{
   statement block
}
	4. Syntax:
	do
	{
	   statement block
	} while (condition);

5. Example:
sum = 0;
n=1;
while (n<=10)
{
sum=sum+n;
n=n+1;
}
	5. Example :
	int i = 0;
	do {
	  printf("\t%d", i);
	  i++;
	} while (i<10);

1. 23 NESTED LOOP.

- Using a loop within another loop is called nested loop.
- In general use nested for loops.
- The nested loops are generally used to construct matrix like data
- The outer loop counts the rows and inner loop counts columns

Syntax

```
for (initializing ; test condition ; increment / decrement)
{
statement;
for (initializing ; test condition ; increment / decrement)
{
body of inner loop;
}
statement;
}
```

Example:

```
#include <stdio.h>
int main() {
    int row,column;
      for(row = 1; row <= 4; row++) {
        for(column = 1; column <= 4; column++)
            printf(" * ");
         putchar('\n');
    }
}
```

output :

```
* * * *
* * * *
* * * *
* * * *
```

UNIT - II

Functions in C –global and local variables, parameter passing, standard functions in header files, recursion. Array's in C- one dimensional arrays, multi dimensional arrays, arrays as function arguments, sorting, searching, and merging.

2.1 FUNCTION AND ITS ADVANTAGES

Function:

- Function is a block of statements that solve a task or sub task of other task.
- C supports two types of functions: library functions and user defined functions.
- The library functions can be used in any program by including respective header files. The header files must be included using # include preprocessor directive.
- The programmer can also define and use his/her own functions for performing some specific tasks. Such functions are called as user-defined functions.

The following are the advantages of functions:

1. Functions support modular programming
2. Functions reduce program size
3. Use of functions avoid code duplication
4. Functions provide code reusability.
5. Functions can be called repetitively as per requirement
6. A set of functions can be used to form libraries

2.2 THE PARTS OF A FUNCTION.

A Function has the following parts:

1. Function prototype
2. Definition of a function
3. Function call
4. Actual and formal arguments
5. The return statement

6.

1. Function prototype

- The function prototype specifies the following:
    - Name of the function,
    - Type of value it returns
    - Type and number of arguments it takes.
- When the programmer defines the function, the header of the function must be like its prototype declaration.
- If the programmer makes a mistake, the compiler flags an error message.
- The prototype declaration statement is always terminated with semi-colon.
- The following statements are the examples of function prototypes:
- void show(void);
- float sum(float, int);

2. Function Definition

The first line is called function header and is followed by function body. The block of statements followed by function header is called as function definition. The header and function prototype declaration should match each other. The function body is enclosed with curly braces. The function can be defined anywhere. If the function is defined before its call, then its prototype declaration is optional

3. Function Call

The function gets activated only when a call to function is invoked. A function must be called by its name, followed by argument list enclosed in parenthesis and terminated by semi colon.

4. Actual and formal Arguments

The arguments declared in caller function and mentioned in the function call are called as actual arguments. The arguments declared in the function header are known as formal arguments.

5. The return Statement

The return statement is used to return value to the caller function. The return statement returns only one value at a time. When a return statement is encountered, compiler transfers the control of the program to caller function. The syntax of return statement is as follows:

return (variable name);   or   return variable name;

Example:

```
#include<stdio.h>
float sum(float, int);       //function prototype
void main( )
{
float x,y=2.4;
int z=5;
x=sum(y,z);                  //function call with actual argument in parenthesis
}
float sum(float j, int k)  // function header
{
return (j+k);                       //function body with return statement
}
```

2.3 DIFFERENT TYPES OF FUNCTIONS IN C

Function:

Function is a block of statements that solve a task or sub task of other task.

C supports two types of functions:

1. Library functions / Built in functions / Pre defined functions
2. User defined functions / Programmer defined functions

1.Library functions

C library functions are the built-in functions that are available in C compiler. They are defined and declared in their respective header files. They are the readily available functions which can be used directly.

The library functions can be used in any program by including respective header files. The header files must be included using # include preprocessor directive.

Example : printf() and scanf () are the formatted input and output functions which are declared and defined in the header file <stdio.h>

2.User defined functions

The programmer can also define and use his/her own functions for performing some specific tasks. Such functions are called as user-defined functions.

The general form of the user defined function is as follows:

```
Return type function_name( formal parameters)
{
Body of the function
}
```

```
float sum(float j, int k)  // function header
{
return (j+k);                    //function body with return statement
}
```

2.4 DIFFERENT WAYS OF WRITING A FUNCTIONS.

DIFFERENT CATEGORIES OF FUNCTIONS.

We can divide the functions into four categories based on the arguments presence and return values.

1. Without arguments and return values
2. With arguments but without return values
3. With arguments and return values
4. Without arguments but with return values

1.With out arguments and return values

Calling function	Analysis	Called function

main( )
{
- - - - - -
- - - - - -
abc();
	No arguments are passed	abc( )
		{
		- - - - - -
		- - - - - -
		- - - - - -

- - - - - - - - - - }	No values are sent back	}

- ➢ In this case there is no data transfer between calling and the called functions.
- ➢ These functions may be useful to print some messages, draw a line etc.

Example :

```
#include <stdio.h>
void message();
main( )
{
message( );
}
void message()
{
printf("have a nice
day");
}
```

output:

have a nice day

2. With arguments but with out return values

Calling function	Analysis	Called function

main( )
{
		abc(int y)
		{

- - - - - -
- - - - - -
abc(x);
- - - - -
- - - - -
}
	Argument(s) are passed
	
	No values are sent back
		- - - - - -
		- - - - - -
		- - - - - -
		}

- In this case arguments are passed through the calling function. The called function operates on the values. But no result is sent back.
- These functions are partly dependent on the calling function. The result obtained is utilized by the called function.

Example:

```
#include <stdio.h>
void sqr(int x);
void main( )
{
int x = 5;
sqr(5);
}
void sqr(int y)
{
printf("\n %d",
k*k);
}
```

output

```
25
```

3.With arguments and return values

Calling function	Analysis	Called function

main( )
{
int z;
- - - - - -
z=abc(x);
- - - - -
- - - - -
}
	Argument(s) are passed
	
	values are sent back
		abc(int y)
		{
		- - - - - -
		y++;
		- - - - - -
		return (y);
		}

- In this case the copy of actual arguments is passed to formal arguments.
- The return statement returns the values from called function.
- Here data is transferred between calling and the called functions i.e. communication between functions is made.

Example

```
#include <stdio.h>
int sum(int x, int y);
void main( )
{
int x = 5,y=6,z;
z=sum(x,y);
printf("\n %d", z);
}
int sum(int x, int y)
{
return(x+y);
}
```

output

```
11
```

4.With out arguments but with return values

Calling function	Analysis	Called function

main( )
{
int z;
- - - - - -
z=abc( );
- - - - -
- - - - -
}
	No Argument(s) are passed
	
	values are sent back
		abc( )
		{
		int y=5;
		- - - - - -
		- - - - - -
		return (y);
		}

In the above case called function is independent. It performs the task and returns output.

Example

```
#include <stdio.h>
int sum( );
void main( )
{
int z;
z=sum( );
printf("\n %d", z);
}
int sum( )
{
int x = 5,y=6;
return(x+y);
}
```

output

```
11
```

2.5. DIFFERENT PARAMETER PASSING TECHNIQUES (OR) CALL BY VALUE AND CALL BY REFERENCE (OR) METHODS OF PASSING ARGUMENTS TO A FUNCTION.

The main objective of passing argument to a function is message passing. The message passing is also known as communication between two functions i.e. between caller and callee functions. There are 2 methods by which we can pass values to the function. These methods are:

1. Call by value (pass by value)
2. Call by reference (pass by reference)

Pass by value

In this type, value of actual argument is passed to the formal argument and operation is done on the formal arguments. Any change in the formal argument does not affect the actual argument because formal arguments are photocopy of actual arguments. Hence, when function is called by call by value method, it does not affect the actual contents of actual arguments. The changes made in the formal arguments are local to the block of called function.

The following example illustrates the use of call by value

```
#include<stdio.h>
#include<conio.h>
void change(int x,int y);
main( )
  {
     int x,y;
     clrscr();
        printf("Enter x,y,values:");
        scanf(“%d%d”, &x,&y)

          printf("\n\nThe values of x,y before function call:");
         printf("\nx=%d,\ty=%d”,x,y);
```

```
            change(x,y);


           printf("\n\nThe values of x,y after function call:");
           printf("\nx=%d,\ty=%d",x,y);
}
 void change(int x,int y)
  {
          printf("\n\nThe values of x,y in function before changing:");
         printf("\nx=%d,\ty=%d",x,y);


            x=x+10;
            y=y+10;


         printf("\n\nThe values of x,y in function after changing:");
         printf("\nx=%d,\ty=%d",x,y);
  }
```

```
output:
Enter x,y values 12 10
The values of x,y before function call
x=12  y=10
The values of x,y in function before changing
x=12  y=10
The values of x,y in function after changing
x=22  y=20
The values of x,y after function call
x=12  y=10
```

Pass by reference

In this type, instead of passing values, addresses are passed. Function operates on address rather than values. Here the formal arguments are pointers to the actual arguments. Hence changes made in the arguments are permanent.

The following example illustrates the concept of pass by address

```
#include<iostream.h>
#include<conio.h>
void change(int *p,int *q);
main( )
  {
     int x,y;
     clrscr();
     printf(“Enter x,y,values:");
     scanf(“%d%d”,&x,&y);

        printf("\n\nThe values of x,y before function call:");
          printf("\nx=%d,\ty=%d”,x,y);
             change(&x,&y);

               printf("\n\nThe values of x,y after function call:");
               printf("\nx=%d,\ty=%d”,x,y);
        }
 void change(int *p,int *q)
  {
        printf("\n\nThe values of *p, *q in function before changing:");
        printf("\n*p=%d,\t*q=%d”,*p,*q);

     *p=*p+10;
     *q=*q+10;
```

```
        printf("\n\nThe values of *p, *q in function after changing:");
        printf("\n*p=%d,\t*q=%d",*p,*q);
  }
```

output:

```
Enter x,y values 12 10
The values of x,y before function call
x=12  y=10
The values of  *p  ,*q in function before change
x=12  y=10
The values of  *p  ,*q in function after change
x=22  y=20
The values of x,y after function call
x=22  y=20
```

2.6. RECURSION.

Recursion:

1. A recursive function is a function that calls itself.
2. The speed of a recursive program is slower because of stack overheads.
3. In recursive function we need to specify recursive conditions, terminating conditions, and recursive expressions.

Example :

The below program demonstrate recursion concept.

```
#include <stdio.h>
long factorial(long);
int main(void)
{
 long number = 0;
  printf("\nEnter an integer value: ");
  scanf(" %ld", &number);
  printf("\nThe factorial of %ld is %ld\n", number, factorial(number));
  return 0;
}
```

```
 long factorial(long n)
{
  if(n < 2)                /* terminating condition */
    return n;
  else                   /* recursive condtion */
    return n*factorial(n - 1); /*recursive expression */
}
```

output

The factorial of 5 is 120

2.7 LOCAL AND GLOBAL VARIABLES

Local Variable

- It is declared by auto int or int keywords
- It is declared inside the function
- If it is not initialized **garbage** value is stored
- It is created when function is called and lost when function terminates
- It can be accessed in only **one function**, where it is declared
- It can be accessed by only one function, so data sharing is not possible
- Parameter **passing is required** i.e., we can access local variable in different function by sending it as parameters in the function call
- If value of local variable is modified in function , changes are **not visible** in rest of program

Global variable:

- It is declared outside the function
- If it is not initialized , **0** is stored by default
- It is created before execution starts and lost when program terminates
- It is visible to **entire program**
- It is accessed by **multiple functions** so data **sharing** is possible

- Parameters passing is **not required** since global variable can be accessed throughout the program(without sending it as parameter to any function)
- If value of global variable is modified in a function, the changes are **visible** in rest of program.
- If the value of local and global variable is same, then the compiler considers local variable and ignores the global variable

2.8 FUNCTIONS AVAILABLE IN DIFFERENT HEADER FILES

C Library has several header files. Few of the header files and related functions are listed below:

Header file name	Functions
stdio.h	scanf(), printf(), getch(),getchar(), getc(), putch(),putchar(), fscanf(), fprintf() etc.
math.h	sqrt(), pow(), abs(), ceil(),floor(), sin(), cos(),tan() etc.
string.h	strcat(), strcmp(), strcpy(), strlen(), strlwr(), strupr() etc.
stdlib.h	malloc(), calloc(), realloc(), free() etc.
conio.h	clrscr()
ctype.h	atoi(), isupper(), isdigit(), islower(), isalpha().

2.9 DIFFERENCE BETWEEN RECURSION AND ITERATION.

The difference between recursion and iteration are explained below:

Iteration	**Recursion**
1. Iteration uses repetition control structure	1. Recursion uses selection control structure

2. Iteration explicitly uses a repetition control structure	2. Recursion achieves repetition through repeated function calls
3. Iteration terminates when loop continuation condition fails	3. Recursion terminates when a base case is recognized.
4. Iteration modifies the counter until the counter value reaches to fail the loop repetition condition	4. Recursion produce simpler versions of the original problem until the base condition is reached
5. An infinite loop occurs with iteration when the loop continuation test never becomes false.	5. Infinite recursion occurs if the recursion step does not reduce the problem during each recursive call to reach the base condition
6. As iteration normally occurs with in a function so it may not take more processing time and memory space.	6. Due to the repeated function calls in recursion, it may take more processing time and memory space.

Example :
`long factorial(long n)`
`{`
`  long f=1,i;`
`  for(i=1;i<=n;i++)`
`      f=f*i;`
` return f;`
`}`
	Example :
	`long factorial(long n)`
	`{`
	`  if(n < 2)           return n;`
	`  else`
	`    return n*factorial(n - 1);`
	`}`

2.10 ARRAY

An array is collection of elements of same data type that share a common name. Array elements are stored in adjacent memory locations. To refer the array elements we use index or subscript. The array index starts with zero. So the index of last element is always n-1 where n is the size of array

Syntax for declaring array

```
type array_name[size];
```

The type specifies the data type of element. Size indicates number of elements that an array can hold.

```
Eg.  int marks[6];
     float height[10];
```

Initialization of array

The general form of initializing of an array is given below:

```
type array_name[size] = { list of values };
```

Here the size may be omitted; the values are separated with comma

Examples :

```
int  a[3]={1,2,3};
          (or)
int a[ ]={1,2,3}
```

Advantages of arrays

- Array is capable of storing many elements at a time with a common name
- We can assign the element of an array to any ordinary variable or another array element of same data type.
- Any element of array can be accessed randomly

Disadvantages of array

Predetermination of array size is must

There is a chance of memory wastage

Deleting or inserting element needs shifting of elements

2.11 ARRAY AS ARGUMENT TO A FUNCTION

It is possible to pass the values of an array to a function. To pass an array to a function, it is enough to list the name of array with out any subscript and the size of array as agrument

Example:

To call a function named **largest** with an array we write the following statement

```
        largest(a,n);
```

Here ‘a’ is an array, n indicates no. of elements.

The declaration of the function that takes array as arguments is as follows

return type funtion_name(datatype arrayname[], int size);

int largest(int a[], int n);

the below program demonstrates using single dimentional array argument

```
#include<stdio.h>
void disparray(int a[], int n);
main()
{
int a[10], n,i;
printf(“enter no. of elements”);
scanf(“%d”, &n);
printf(“enter array elements”);
for(i=0;i<n;i++)
scanf(“%d”, &a[i]);
disp(a,n);
}
void disparray(int a[], int n)
```

```
{
int i;
for(i=0;i<n;i++)
printf("%d", a[i]);
}
```

Explanation:

In the above program the function disparray has array argument. It lists the contents of the array.

2.12 DOUBLE DIMENSIONAL ARRAYS:

A double dimensional array can be thought of as a table of elements which contains rows and columns.

Declaring two dimensional array:

Syntax:

```
            datatype  arrayname[rowsize][columnsize];
```

Example: int a[3][3];

The above declaration allocates nine adjacent memory locations. The locations are referred as follows:

a[0][0], a[0][1], a[0][2]

a[1][0], a[1][1], a[1][2]

a[2][0], a[2][1], a[2][2]

Initializing two dimensional array:

We can assign the values to array elements in its declaration as follows:

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

The above declaration is equivalent to the following

```
static  int a[2][3] = { {1,2,3},
                        {4,5,6}};
```

Two dimensional array as function arguments:

We can send the two dimensional array as argument to a function. In the function call we mention the array name and dimensions. In function declaration we write array name along with two subscripts and dimensions. Mentioning the value in the first subscript is optional.

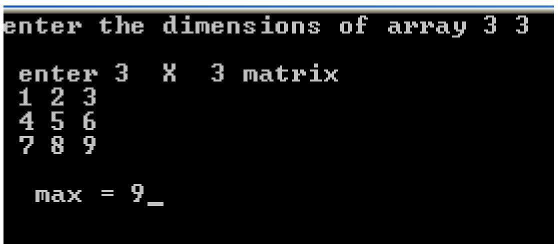
Eg: Declaring a function with two dimensional array as argumnet.

```
void read_array(int a[4][4], int m, int n);
               (or)
void read_array(int a[ ][4], int m, int n);
```

The below program demonstrates sending double dimensional array as argument.

```
/* a program to find max. element in a double dimensional array */
#include <stdio.h>
#include <conio.h>
int find_max(int a[][5], int m, int n); /* function prototype */
void main()
{
int a[5][5],m,n,max,i,j;
clrscr();
printf("enter the dimensions of array");
scanf("%d%d", &m,&n);
printf("\n enter %d  X  %d matrix", m,n);
for(i=0;i<m;i++)
for(j=0;j<n;j++)            /* reading an array */
scanf("%d", &a[i][j]);
max=find_max(a,m,n);          /* function call */
printf("\n  max = %d", max);
}
int find_max(int a[][5], int m, int n)          /* function header */
{
int max,i,j;
max=a[0][0];
for(i=0;i<m;i++)
for(j=0;j<n;j++)
if(max<a[i][j])
max=a[i][j];
return max;
}
```

<u>output</u>



```
enter the dimensions of array 3 3

 enter 3  X  3 matrix
 1 2 3
 4 5 6
 7 8 9

  max = 9_
```

<u>Reading and displaying two dimensional array:</u>

<u>To get the elements into two-dimensional array we write two for loops as follows:</u>

```
for(i=0; i<n;i++)
for(j=0; j<m;j++)
scanf("%d",&a[i][j]);
```

<u>To display the elements of different rows we write the code as follows:</u>

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

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

2.13 MULTIDIMENSIONAL ARRAYS

C allows an array of three or more dimensions. The exact limit is determined by the compiler. The general form of multi-dimensional array is given below.

Type array_name[s1][s2][s3]……..[sn];

Here si is size of ith dimension

Example:

int survey[3][5][12];

Survey is a 3-dimensional array declared to contain 180 integer type elements. The array survey may represent a survey data of rainfall during the last 3 years form jan to dec in 5 cities.

Two marks questions

1.Algorithm:

The approach or method that is used to solve a problem is known as algorithm.

For example if you want to develop a program that tests the given number is even or odd, the set of statements that solves the problem becomes program, the method that is used to test if the number is even or odd is the algorithm.

The algorithm for solving even/odd problem can be expressed as follows:

- First divide the number by two
- If the remainder of the division is zero, the number is even
- Otherwise, the number is odd

2. Assembler

Program that translates programs from assembly language to machine language.

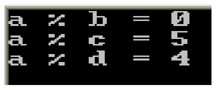
3. Modulus operator

The modulus operator is symbolized by the percent sign (%). It returns the remainder value of a division.

Example :

```
*modulus operator*/
#include <stdio.h>
#include<conio.h>
main()
{
int a=25, b=5,c=10,d=7;
clrscr();
printf("a %% b = %d\n", a % b);
printf("a %% c = %d\n", a % c);
printf("a %% d = %d\n", a % d);
}
```

Output:



```
a % b = 0
a % c = 5
a % d = 4
```

4. Top down programming:

It is a programming methodology, in which each task is divided into number of sub tasks and each sub task is implemented through a procedure or function. Finally all tasks will be combined to complete the task.

C programming supports top down programming.

5. Format string

Format string is a string, which is an argument in printf and scanf functions. The format string specifies the type of data that we are scanning or printing.

Example : %d specifies integer type data

%f specifies float type data etc.

6. Argument types

Arguments are of two types: actual arguments and formal arguments.

The arguments declared in caller function and mentioned in the function call are called as **actual arguments**. The arguments declared in the function header are known as **formal arguments.**

7. Sorting

Sorting is method of arranging numbers / strings in ascending or descending order. There are different sorting techniques such as bubble sort, selection sort, insertion sort, quick sort etc.

8. Searching

Searching is method of locating elements in a list. We have different searching techniques such as linear search, binary search etc.

9. Merging

It is the process of combining two lists. While merging we sort the elements and combine the list. This leads to merge sort.

UNIT - III

Data types, scope and visibility, automatic conversion of variables different types of variables, include directives, define directive, define with arguments

Pointers in C - Arrays and pointers, pointers to functions, pointers and strings, command line arguments

3.1 DATA TYPES AVAILABLE IN C

Data type:

In C each variable is attached with some data type.

The data type defines:

1. The amount of storage allocated to variables.
2. The values that they can accept.
3. The operations that can be performed on variables.

DATATYPES IN C

C data types can be classified into the following categories:

- Basic data type
- Derived type
- User defined type
- Void data type

Basic data type:

The basic data type include the following

- Integer type
- Floating point type
- Character type

Integer Type:

Integers are whole numbers. C has 3 classes of integer storage namely short int, int and long int.

Floating Point Types:

Floating point number represents a real number with 6 digits precision. Floating point numbers are denoted by the keyword float.

Character Type:

A single character can be defined as a character type of data.

The basic data types supported by C are described with their size in bytes and ranges in the following table

Data type	Size in bytes	Range
char	1	-128 to127
unsigned char	1	0 to 255
int	2	-32768 to 32767
unsigned int	2	0 to 65536
long int	4	-2,147,483,648 to 2,147,483,647
float	4	3.4 E -38 to 3.4E+38
double	8	1.7 E -308 to 1.7 E +308
long double	10	3.4 E -4932 to 1.1E +4932

Derived data type

The derived data types are of following types:

1. Pointers
2. Functions
3. Arrays

User defined data type

User defied data types are of the following types:

1. Structure
2. Union
3. Enumeration data type

The void data type

The void type is added in ANSI C. It is also known as empty data type. It can be used in two ways.

- When specified as a function return type. Void means that the function does not return any value.

- The void keyword is also used as argument for function. When found in a function heading, void means that the function does not take any arguments.

Examples:

```
void printline();
int func1(void);
```

3.2 STORAGE CLASSES IN C

In C a variable can have any one of the following four storage classes.

- automatic
- static
- register
- extern

A storage class of a variable tells us the following

- The location where the variable is stored
- The initial value of the variable when declared
- Scope of the variable
- Longevity or life time of variable

Here the scope of variable refers in what parts of the program a variable is actually available for use (active). The longevity refers to the period during which a variable retains a given value during the execution of program (alive)

Automatic storage class

The automatic variables are declared with the keyword auto. In general using the keyword auto is optional.

Features of automatic variable

Location :	RAM
Default value:	an unpredictable value, or garbage value
Scope :	local to the block in which variable is declared.

Life time : till the control remains in the block in which variable is defined

Example :

The below program shows the working of auto variable.

```
#include <stdio.h>
void call1( );
void call2( );
main( )
{
int v = 10;
call2( );
printf("\n V=%d",v);
}
void call1( )
{
int v = 20;
printf("\n V=%d",v);
}
void call2( )
{
int v = 30;
call1( );
printf("\n V=%d",v);
}
```

output:

```
V=20
V=30
V=10
```

REGISTER STORAGE CLASS

FEATURES OF REGISTER VARIABLES

Location	:	CPU Registers.
Default value:		an unpredictable value, or garbage value
Scope	:	local to the block in which variable is declared.
Life time	:	till the control remains in the block in which variable is defined

A value stored in cpu registers can always be accessed faster than the one which is stored in memory. Therefore if a variable is used at many places in a program it is better to declare its storage class as register.

Example :

The below program shows the working of register variable

```
#include<stdio.h>
main( )
{
   register int m=1;
    for( ; m<=5;m++)
    printf("\t %d", m);
}
```

output:

```
1    2    3    4    5
```

STATIC STORAGE CLASS

Features of static variable

Location	:	RAM
Default value:		0
Scope	:	local to the block in which variable is declared.
Life time	:	value of the variable persists in different function calls

Example:

The below program shows the working of static variable.

```
#include <stdio.h>
void increment( );
main( )
{
        increment( );
        increment( );
        increment( );
}
void increment( )
{
        int static m;
        m++;
        printf("\n m=%d",m);
}
```

output

```
m=1
m=2
m=3
```

EXTERN STORAGE CLASS

Features of extern variable

Location	:	RAM
Default value:		0
Scope	:	global
Life time	:	value of the variable will be available as long as the program execution does not come to end

Example :

The below program shows the working of extern variable

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

```
int v=10;
void call1( );
void call2( );
void main( )
{
        call1( );
        call2( );
        printf("\n In main( ) v = %d",v);
}
void call1( )
{
        printf("\n In call1( ) v = %d",v);
}
void call2( )
{
        printf("\n In call2( ) v = %d",v);
}
```

output:

```
In call1( ) v = 10
In call2( ) v = 10
In main( ) v = 10
```

Situations of using various storage classes.

- Use static storage class only if you want the value of a variable to persist between different function calls. Generally static variables are used in recursive functions.
- Use register storage class for the variables which are being used very often in the program

- Use extern storage class for the variables which are being used by almost all functions in the program. If you do not have express needs mentioned above use automatic storage class.

3.3 TYPECASTING (TYPE CONVERSION)

Typecasting refer conversion of type. Typecasting is essential when the values of variables are to be converted from one type to another type. C allows implicit as well as explicit conversion.

Explicit typecasting

Sometimes errors may be encountered in the program with implicit typecasting. The desired type can be achieved by typecasting a value of particular type. The following is the syntax of typecasting in C.

(data-type name) expression

Example

x=(float) 5/2;

Implicit Type Conversion

The type conversion is carried out when the expression contains different types of data items. When the compiler carries such type conversion itself by using in built data types then it is called implicit type conversion. The variable of lower type (small range) type when converted to higher type (large range) is known as promotion. When the variable of higher type is converted to lower type, it is called demotion.

The below table describes various implicit type conversion rules that a compiler follows.

Argument 1	Argument 2	Argument 3
char	int	int
int	float	float
int	long	long
double	float	double
int	double	double
long	double	double
int	unsigned	unsigned

3.4 PREPROCESSOR:

Preprocessor is a program that processes the code before it passes through the compiler. Preprocessor directives are placed in the source program before the main line. Before the source code passes through the compiler it is examined by the preprocessor. If it is appropriate then the source program is handed over to the compiler.

Preprocessor directives follow the special syntax rules and begin with the symbol # symbol and do not require any semicolon at the end. A set of commonly used preprocessor directives

Preprocessor directives:

Directive	**Function**
#define	Defines a macro substitution
#undef	Undefines a macro
#include	Specifies a file to be included
#ifdef	Tests for macro definition
#endif	Specifies the end of #if
#ifndef	Tests whether the macro is not def
#if	Tests a compile time condition
#else	Specifies alternatives when # if test fails

The preprocessor directives can be divided into three categories

1. Macro substitution division
2. File inclusion division
3. Compiler control division

Macros:

Macro substitution is a process where an identifier in a program is replaced by a pre defined string composed of one or more tokens. we can use the #define statement for that task.

It has the following form

#define identifier string

The preprocessor replaces every occurrence of the identifier in the source code by a string. The definition should start with #define and should follow an identifier and a string

There are different forms of macro substitution. The most common form is
1. Simple macro substitution
2. Argument macro substitution
3. Nested macro substitution

Simple macro substitution:

Simple string replacement is commonly used to define constants

examples:

#define pi 3.1415926

#define AREA 12.36

Writing macro definition in capitals is a convention and it is not a rule.

Macros as arguments:

The preprocessor permits us to define more complex and more useful form of replacements. It takes the following form.

# define identifier(f1,f2,f3…..fn) string.

A simple example of a macro with arguments is

# define CUBE (x) (x*x*x)

If the following statements appears later in the program,

volume=CUBE(side);

The preprocessor would expand the statement to

volume =(side*side*side)

Nesting of macros:

We can also use one macro in the definition of another macro. That is macro definitions may be nested. Consider the following macro definitions

# define SQUARE(x)((x)*(x))

# define CUBE (x) (SQUARE(x) *(x))

Undefining a macro:

A defined macro can be undefined using the statement

# undef identifier.

This is useful when we want to restrict the definition only to a particular part of the program.

File inclusion:

The preprocessor directive "#include file name” can be used to include any file in to your program

Eg.

#include< filename>

Without double quotation marks. In this format the file will be searched in only standard directories**.**

3.5. POINTERS

Pointer variable is a variable that holds address of simple variable of same type.

Advantages (importance) of pointers:

- Pointers save the memory space
- Execution time with pointer is faster because data is manipulated with address i.e. direct access to memory location
- Memory is accessed efficiently with the pointers. i.e. memory is dynamically allocated and released.
- Pointers are used with data structures.
- Pointer are useful for representing two-dimensional and multi dimensional arrays.

Disadvantages:

- Pointer variable require extra memory.
- Use of pointers increases complexity.

Pointer arithmetic:

Pointer arithmetic is one of the powerful features of C. if the variable p as pointer to a type 'p+1' gives the address of next variable of same type. This mechanism is extensively used in accessing array elements using pointers.

If p1,p2 are the pointer the following statements are valid.

```
P1+1;
P2+3;
P1-2;
```

But the following are the invalid statements

```
p1+p2;
p1-p2;
p1/p2;
```

3.6 DYNAMIC MEMORY ALLOCATION:

- Allocating memory at runtime is called dynamic memory allocation.
- It eliminates predetermining of number of variables to be used in the program.
- This mechanism is extensively used in data structures like linked lists and binary trees.
- In dynamic memory allocation, after our purpose is served, we can release the memory to avoid blocking of the memory.
- Dynamic memory allocation is done using malloc or calloc functions
- Memory release is done using free function.

Example :

```
main()
{
int *p;
*p=(int *) malloc(sizeof(int));
*p=30;
printf("%d",*p);
free(p);
}
```

In the example, for the pointer variable p two bytes of memory has been allocated dynamically. And memory has been released by the function free.

<u>malloc , calloc , realloc , free functions</u>

- malloc, calloc, realloc functions are related to dynamic memory allocation
- free function releases the memory
- all these functions are declared in malloc.h

<u>malloc(k):</u>

- malloc allocates k bytes of memory dynamically.
- It returns the base address on success; otherwise it returns NULL
- The space is not initialized.

<u>calloc(n,k)</u>

- malloc allocates k bytes of memory dynamically for n elements of array.
- It returns the base address on success; otherwise it returns NULL
- The space is initialized to zero.

realloc(p,k)

- changes the size of the block pointed by p to k bytes
- contents in old spaces will not be changed

- it returns the base address of the resized spaces on success; otherwise it returns NULL.
- New spaces are not initialized.

free(p)

- It de allocates the memory pointed by the pointer p
- to be de-allocated the memory must be allocated by malloc, calloc, or realloc functions.

3.7 STRINGS

A string is an array of characters that is terminated with null character '\0'. Any group of characters defined between double quotation marks is a constant string.

Eg: "Man is obviously made to think"

Declaring and initializing string variables:

A string variable is any valid c variable name and is always declared as an array. The general form of declaration of string variable is

char string-name [size];

When the compiler assigns a character string to a character array it automatically supplies a null character at the end of string. Therefore, the size should be equal to maximum number of characters in the string+1.

Character arrays may be initialized when they are declared c-permits a char array to be initialized in any of the following forms:

static char x[9] = "New York"

(or)

static char x[9] = { 'N','e','w', ' ' 'y','o','r','k', '\0'};

(or)

static char x[]="New york";

3.8 STRING FUNCTIONS

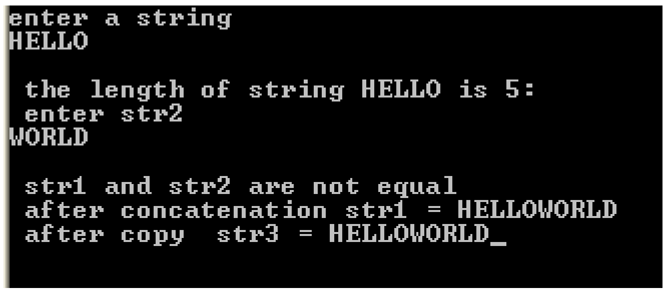
Function	Description
strcpy(s1, s2)	Copies the value of s2 into s1
strcat(s1, s2)	Appends s2 to the end of s1

strcmp(s1, s2)	Compared s1 and s2 alphabetically; returns a negative value if s1 should be first, a zero if they are equal, or a positive value if s2 sbould be first
strlen(s1)	Returns the number of characters in s1 not counting the null
strlwr(s1)	Converts upper case characters of a string s1 to lowercase
strupr(s1)	Converts lower case characters of a string s1 to upper case
Strrev(s1)	Reverses all the characters of a string
strncpy(s1, s2, n)	Copies n characters of s2 into s1. Does not add a null.
strncmp(s1, s2)	Compares the first n characters of s1 and s2 in the same manner as strcmp
strncat(s1, s2, n)	Appends source string to destination string upto specified length

The below program demonstrates use of string functions in c

```
/* a program to compare strings,concatenate strings, copy strings
   and to find length of a string
    using string functins*/
#include<stdio.h>
#include<conio.h>
#include<string.h>
main()
{
char str1[20],str2[20],str3[40];
int n,p;
clrscr();
puts("enter a string");
gets(str1);
n=strlen(str1);
printf("\n the length of string %s is %d:",str1,n);
puts("\n enter str2");
gets(str2);
p=strcmp(str1,str2);
if(p)
printf("\n str1 and str2 are not equal");
else
printf("\n str1 and str2 are equal");
strcat(str1,str2);
printf("\n after concatenation str1 = %s",str1);
strcpy(str3,str1);
printf("\n after copy  str3 = %s",str3);
}
```

Output:



```
enter a string
HELLO

 the length of string HELLO is 5:
 enter str2
WORLD

 str1 and str2 are not equal
 after concatenation str1 = HELLOWORLD
 after copy  str3 = HELLOWORLD_
```

3. 9 COMMAND LINE ARGUMENTS

⇒ Command line arguments are the arguments send to the executable program

⇒ The main function holds two arguments one is argument counter and the other is argument vector, it is an array of pointers to character type.

⇒ File name is included first in the list of arguments

⇒ The concept of command line arguments is useful to implement the commands

⇒ For example if we want to execute a program to copy the contents of a file named x.dat to another file named y.dat then we may use a command line as follows

c:\program x.dat y.dat

here the program is file name where the executable code of the program is stored.

The following program demonstrate the command line arguments:

```
main(int argc, char *argv[])
{
int i;
printf("number of arguments=%d",argc);
for(i=0;i<argc;i++)
printf("\n %s",argv[i]);
}
```

suppose that the name of the program is ex.c and after creating the executable file, we may type the following command.

c:\ex aa bb cc

the above command produces the following output.

```
Number of arguments=4
ex
aa
bb
cc
```

UNIT - IV

Structures and Unions: arrays as structures members, nested structure array of structures, structures as function arguments, pointer to a structure, input and output-elementary functions

4.1 STRUCTURE

Structure is data type that combines logically related data items of different data type. In C structure combines the data members of different types.

<u>The syntax of structure declaration is as follows:</u>

```
struct <struct name>
{
data type variable1;
data type variable2:
- - - - -- -- - ---- - -
 - - - - - - - -- --- --
data type variable n;
};
```

example:

```
struct item
{
int codeno;
float price;
int qty;
};
```

the object of structure item can be declared as follows:

```
 struct item a, *b;
```

The operator (.) dot and (→) arrow are used to access the member variables of struct. The dot operator is used when simple object is declared and arrow operator is used when object is pointer to structure. The access of members can be accomplished as per the syntax given below:

[Object name][Operator][Member variable name]

When an object is a simple variable, access to members are accessed as below:

a.codeno

a.price

a.qty

when an object is pointer to structure then members are accessed as below:

a->codeno

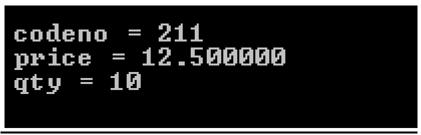
a->price

a->qty

Example program

```
#include<stdio.h>
#include<conio.h>
struct item
{
int codeno;
float price;
int qty;
};
main()
{
        struct item a = { 211, 12.50,10}; /* initializing a structure
        variable */
        clrscr();
        /* printing item a details */
        printf("\n codeno = %d", a.codeno);
        printf("\n price = %f", a.price);
        printf("\n qty = %d", a.qty);
}
```

Output:



```
codeno = 211
price = 12.500000
qty = 10
```

4.2. FEATURES OF STRUCTURES:

1. The structure variables can be initialized.

Example

sturct item x={11, “abc”, 2.7};

2. The structure variables can be assigned to other variable of same structure.

Example:

struct y;

y=x; /* values of members of x are assigned to y members*/

3. Nesting of structures

Structures can contain other structures as members; in other words, structures can *nest*. Consider the following two structure types:

```
struct first_structure_type
{
  int integer_member;
  float float_member;
};
struct second_structure_type
{
  double double_member;
  struct first_structure_type struct_member;
};
```

example :

```
struct date
{
```

```
int day;
int month;
int year;
};
struct student
{
int rno;
char name[20];
date dob;     /* nesting of structure */
};
```

4.Structure variable can be send as argument and return from the function.

A function may take structure variable as argument. We can send the structure variable as argument to a function in two ways:

1.using call by value.

2.using call by reference.

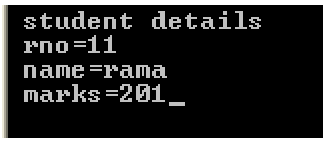
In call by value technique we pass structure variable as an argument and in call by reference we pass the address of the structure variable as argument.

The below program demonstrates structure variable as argument to function.

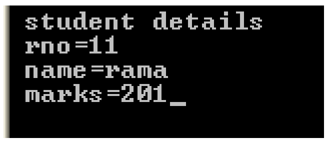
```
/*a program to pass structure to a function */
#include<stdio.h>
#include<conio.h>
struct student
{
   int rno;
   char name[10];
   int marks;
};
```

```
/* structure variable as argument  using call by value*/
void show_data(struct student s);
/* structure variable as argument  using call by reference*/
void show_data1(struct student *p);
 main()
{
struct student s = {11, "rama",201}; /*structure variable initialization
*/
clrscr();
show_data(s);
show_data1(&s);
}
void show_data(struct student s)
{
printf("\n student details");
printf("\n rno=%d",s.rno);
printf("\n name=%s",s.name);
printf("\n marks=%d",s.marks);
}
void show_data1(struct student *p)
{
printf("\n student details");
printf("\n rno=%d",p→rno);
printf("\n name=%s", p→name);
printf("\n marks=%d", p→marks);
}
```

Output:



```
student details
rno=11
name=rama
marks=201_
```



```
student details
rno=11
name=rama
marks=201_
```

5. A structure can contain an array as member

In structure declaration we can declare an array as structure member.

Eg.

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

in the above declaration name[10] is a character array and it is member of student structure.

4.3. ARRAY OF STRUCTURES.

We can use the structures variable as an array. Declaring and accessing an array of structure is similar to an array built in type.

Eg:

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

for the above declaration, we can declare an array of student structure as follows:

sturct student s[10];

for the above declaration, it creates 10 structure variables in sequential order and allocates 140 bytes of memory.

We access the member with dot operator as follows

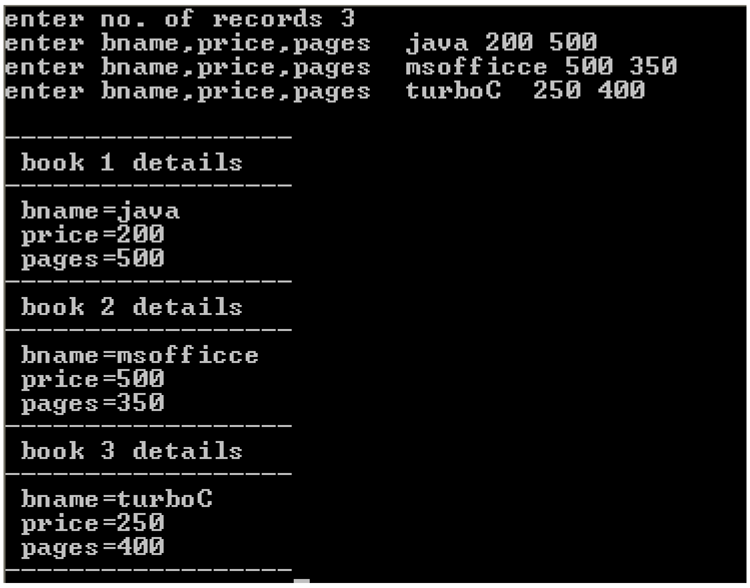
s[i].rno , s[i].name. s[i].marks. where i is the index of array s.

the below program demonstrates use of array of sturcutes

```
/* a program to demonstrate array of structure */
#include<stdio.h>
#include<conio.h>
struct book
{
char bname[10];
int price;
int pages;
};
main()
{
struct book b[10];
int i,n;
clrscr();
printf("enter no. of records");
scanf("%d",&n);
for(i=0;i<n;i++)
{
printf("enter bname,price,pages");
scanf("%s%d%d",&b[i].bname,&b[i].price,&b[i].pages);
}
printf("\n-----------------");
for(i=0;i<n;i++)
{
printf("\n book %d details ",i+1);
printf("\n------------------");
printf("\n bname=%s",b[i].bname);
printf("\n price=%d",b[i].price);
printf("\n pages=%d",b[i].pages);
printf("\n------------------");

}
}
```

Out put:



```
enter no. of records 3
enter bname,price,pages  java 200 500
enter bname,price,pages  msofficce 500 350
enter bname,price,pages  turboC  250 400

------------------
 book 1 details
------------------
 bname=java
 price=200
 pages=500
------------------
 book 2 details
------------------
 bname=msofficce
 price=500
 pages=350
------------------
 book 3 details
------------------
 bname=turboC
 price=250
 pages=400
------------------_
```

4. 4 STRUCTURE VARIANTS

We can declare the structures in different ways as shown below:

1) sturct date

```
{
   int month;
   int day;
   int year;
} todaysdate, purchasedate;
```

In the above declaration we declared structure variables at structure declaration

2)
```
sturct date
{
   int month;
   int day;
   int year;
} todaysdate = {1,11,2005};
```

In the above declaration we initialized structure variable at structure declaration

3)
```
sturct date
{
   int month;
   int day;
   int year;
} dates[100];
```

In the above declaration we declared an array of structure variables at structure declaration

4)
```
sturct
{
   int month;
   int day;
   int year;
} dates[100];
```

The above declaration denotes anonymous structure.

5)
```
struct student
{
int rno;
char name[20];
date dob;     /* nesting of structure */
};
```

The above declaration denotes nesting of structure

4. 5. WRITE ABOUT UNIONS IN C?

Unions

Union is a user defined data type that encapsulates different data types. All the members of the union share common memory area.

Example.

```
main()
{
        union alpha
        {
         int a;
        float b;
        };
union alpha a1;
a1.a=10;
printf("%d",a1.a);
a1.b=2.9;
printf("%f",a1.b);
}
```

In the above example the size of the alpha would be 4 bytes.

11

4.6 COMPARE STRUCTURE AND UNION

Structure:

- A user defined type that combines elements of different datatype
- It can hold set of different values grouped together.
- Members of the structure variable are accessed with dot (.) extension.
- Allocates memory for all the members as a block of memory locations.

Union

- A user defined type that combines elements of different datatype
- It can hold single value at a time
- Members of the union variable are accessed with dot (.) extension.
- Members of the union share common memory area, whose size is equivalent to the highest of memory bytes required by individual members.

4.7 EXPLAIN THE CONCEPT OF POINTERS WITH STRUCTURES.

In C it is possible assign the address of structure variable to a pointer of the same structure. Accessing the structure member is done with arrow operator.

The below program demonstrates the concept of structures with pointers.

```
/*a program to access array elements and
  structure elements using  pointers*/
#include<stdio.h>
#include<conio.h>
struct student
{
int rno;
char name[10];
int marks;
};
main()
```

```
{
struct student s = {11, "rama",201}; /*structure variable initialization
*/
struct student *q;
clrscr();
/*accessing structure elements with pointer */
q=&s;
printf("\n student details");
printf("\n rno=%d",q->rno);
printf("\n name=%s",q->name);
printf("\n marks=%d",q->marks);
}
```

Output:

```
rno = 11
name= rama
marks=201
```

4.8 FUNCTION WITH POINTERS

In C using the pointers with functions is done in two ways:

1. Pointers as function arguments
2. Pointers to functions

1.Pointers as function arguments:

In this case, instead of passing values, addresses are passed. Function operates on address rather than values. Here the formal arguments are pointers to the actual arguments. Hence changes made in the arguments are permanent.

Example function is given below:

```
void change(int *p,int *q)
  {
 printf("\n\nThe values of *p, *q in function before changing:");
 printf("\n*p=%d,\t*q=%d”,*p,*q);
    *p=*p+10;
    *q=*q+10;
 printf("\n\nThe values of *p, *q in function after changing:");
 printf("\n*p=%d,\t*q=%d”,*p,*q);
  }
```

2. Pointers to functions

Similar to a variable, a function also holds an address in the memory. So we can declare a pointer to a function, which can then be used as an argument in another function. A pointer to function is declared as follows:

type (*fptr) ( )

This tells the compiler that *fptr is a pointer to function which returns type value. The parentheses around *fptr are necessary.

The below program demonstrates a pointer to function concept:

```
#include <stdio.h>
void my_int_func(int x)
{
   printf( "%d\n", x );
}


 main()

{

   void (*foo)(int);

   foo = &my_int_func;


   /* call my_int_func (note that you do not need to write (*foo)(2) ) */

   foo( 2 );
```

```
    /* but if you want to, you may */
    (*foo)( 2 );
}
```

4.9 POINTER WITH ARRAYS.

In C we can use the concept of pointers with arrays in two ways:

- A pointer works similar to an array.
- A pointer can be used to create a dynamic array

A pointer works similar to an array

In C the name of the array gives the base address of that array. If we assign the base address of array to a pointer variable then that pointer works similar to an array.

The below program demonstrates the concept of pointer works similar to an array:

```
/* program to demonstrate pointer works similar to an array */
#include<stdio.h>
#include<conio.h>
main()
{
int a[]={1,2,3,4,5};
int *p,i;
clrscr();
p=a; /* p holds address of a */
for(i=0;i<5;i++)
printf("\t%d",p[i]);
}
```

Output

```
1 2 3 4 5
```

A pointer can be used to create a dynamic array

Pointers can be used to create arrays dynamically. This avoids predetermination of size of array and also avoids the wastage of storage space. Creating dynamic array is done with malloc( ) function.

The process of creating a dynamic array is shown in below program:

```
main()
{
int *p;
int n,i;
printf(“ enter the size of array”);
scanf(“%d”, &n);
p=(int *) malloc(n*sizeof(int));   /* allocating memory dynamically */
printf(“enter %d elements into an array”,n);
for(i=0;i<n;i++)
scanf(“%d”,&p[i]);
for(i=0;i<n;i++)
printf("\t%d",p[i]);
free(p);
}
```

UNIT V

5.1 DEFINING, OPENING AND CLOSING A FILE

Defining and opening a file: To store the data in a file in secondary memory we must specify certain things about the file. They include the following.

- File name
- Data structure
- Purpose

File name is a string of characters that make up a valid name for the operating system. It may contain two parts, primary name and an optional period with the extension.

Examples s.dat ,store,student.c

Data structure of a file is defined as FILE in the library of standard I/O function definitions, therefore all files should be declared as type FILE before they are used. FILE is a defined data type.

When we open a file, we must specify what we want to do with the file. For example, we may write data into the file or read the existing data.

The following is the general form for declaring and opening file.

FILE *fp;

fp=fopen("filename", "mode");

The first statement declares the variable fp as a pointer to the datatype FILE. The second statement specifies purpose of opening the file. The different modes of opening the file are listed below:

r	it opens the file in read only mode. The physical file must be Available on disk
w	it opens the file for write purpose. If the file is already available its contents will be erased. If the file is not available it creates new file.
a	it opens the file in append mode. The file is opened for write purpose. Already available information in the file will not be erased, it will extend the data of file
r+	it opens the file in read and write mode. The physical file must be Available on disk

w+	it opens the file in write and read mode. The file is opened for write purpose but reading the data is also possible
a+	the file is opened in append and read mode. The file is Opened for adding data as well as reading data is possible. The physical file must be available in disk.

Example : fp=fopen (“x.dat”, “w+”)

Closing a file : a file must be closed as soon as all the operations on it have been completed. This ensures that all associated information with the file is destroyed. It takes the following from:

fclose(file-pointer);

fclose (fp);

fp is a file pointer.

5. 2 FUNCTIONS TO ACCESS THE FILE IN RANDOM ACCESS METHOD.

To access the file randomly C provides the following functions.

ftell() function

fseek() function

rewind() function

ftell() function: this function takes the file pointer as argument and returns a number of type long. The number specifies the current position of file. The general form is given below:

n=ftell(fp);

fseek() function : this function is used to move the file pointer position to the desired location with in the file. It takes the following form.

fseek(filepointer, offset, position);

EXAMPLE:

fseek(fp,m,0) moves the file pointer m bytes from start of file

fseek(fp,m,1) moves the file pointer m bytes from current location

fseek(fp,-m,2) moves the file pointer m bytes back from end of file

rewind() function : this function moves the file pointer to the start of the file.

5.3. WRITE ABOUT DIRECT ACCESS AND INDEXED SEQUENTIAL FILES?

Direct access file:

The direct file organization allows the program to have immediate access to the record required. The computer searches the record using a record key. It can directly locate the desired record with the help of key. With out having to searching through any other records.

The direct file access is better to have immediate processing of records.

Indexed sequential file access:

This is the process of storing or retrieving the information directly, but only after reading an index to locate the address of the item of information

The records in this type of file are organized in sequence for the efficient processing of large amount of data.

5.4 . BINARY FILES

Binary file:

It stores data in files using the computer's internal code. In binary file there is no need of number –to-character conversion when writing a number to a file and no character-to-number conversion when a value is read form the file. The accessing speed is more Binary file than text file.

Example

```
#include<stdio.h>
#include<conio.h>
struct emp
{
int eno;
char ename[10];
int deptno;
};
```

```
main()
{
FILE *fp;
struct emp e;
int i=1;
fp=fopen("emp.dat","wb+");
clrscr();
/* storing employee data */
while (i<=5)
{
printf("\nenter eno,ename,deptno:\n");
scanf("%d%s%d”,&e.eno,e.ename,&e.deptno);
fwrite(&e,sizeof(e),1,fp);
i++;
}
rewind(fp);
/* displaying employee data */

while(fread(&e,sizeof(e),1,fp)==1)
{
printf("\n%5d%10s %5d",e.eno,e.ename,e.deptno);
}
fclose(fp);
}
```

5.5 EXPLAIN THE PROCEDURE TO CREATE WINDOW?

The function window() is used to create window on the screen. The function is not defined in conio.h. The rows in the window can be numbered from 1 to 25 starting from top to bottom. The columns can be numbered from 1 to 80 starting from left to right.

The prototype of the function is given below.

void window(x1,y1,x2,y2);.

The following are the functions to perform i/o operations in the window

- cprintf() : exactly similar to printf(). It sends the output to the active window.
- cscanf() : it is similar to scanf(). It reads the values from the window.
- cputs(): it is similar to puts(). It prints a string to active window.
- clrscr(): clears the window
- gotoxy(): moves the cursor to column x row y. x and y are measured from the top left corner of the window.

Example:

```
#include<stdio.h>
#include<conio.h>
main()
{
int i,j;
clrscr();
window(30,3,60,15);
for(i=0;i<11;i++)
{
for(j=0;j<=10;j++)
{
cprintf("%d X %d=%d\n",i,j,i*j);
}
}
```

SOLUTIONS FOR PRACTICAL PROGRAMS

1. **Write a program to determine the largest of three numbers?**

```
#include <stdio.h>
int main(){
     float a, b, c;
     printf("Enter three numbers: ");
     scanf("%f %f %f", &a, &b, &c);
     if(a>=b && a>=c)
        printf("Largest number = %.2f", a);
     if(b>=a && b>=c)
        printf("Largest number = %.2f", b);
     if(c>=a && c>=b)
        printf("Largest number = %.2f", c);
     return 0;
}
```

2. **Write a program to print Fibonacci series on N numbers?**

```
#include <stdio.h>
int main()
{
int count, n, t1=0, t2=1, display=0;
printf("Enter number of terms: ");
scanf("%d",&n);
printf("Fibonacci Series: %d+%d+", t1, t2); /* Displaying first two
terms */
count=2;    /* count=2 because first two terms are already displayed.
*/
while (count<n)
{
display=t1+t2;
t1=t2;
t2=display;
++count;
printf("%d+",display);
}
return 0;
}
```

3. **Write a program to check weather given number is palindrome or not?**

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

```
#include<conio.h>
main()
{
int n,rev,r,x;
clrscr();
printf("enter a number :");
scanf("%d",&n);
rev=0;
x=n;
while(n!=0)
{
r=n%10;
rev=rev*10+r;
n=n/10;
}
if(rev==x)
printf("\ngiven number is polyndrome ");
else
printf("\ngiven number is not polyndrome");
}
```

4. **Write a program to find the area of a triangle?**

```
#include<math.h>
main()
{
        int a,b,c;
        float s,area;
        clrscr();
        printf("Enter a, b and c values\n");
        scanf("%d %d %d",&a,&b,&c);
        s=(a+b+c)/2;
        area=sqrt(s*(s-a)*(s-b)*(s-c));
        printf("area=%6.2f",area);
        getch();
}
```

5. **Write a program to determine the roots of a quadratic equation?**

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
main()
{
float a,b,c,d,r1,r2,real,img;
clrscr();
printf("\n enter a,b,c values:");
scanf("%f%f%f", &a,&b,&c);
d=(b*b)-4*a*c;
if(d>0)
```

```
        {
        printf("\n roots are real and unequal");
        r1=(-b+sqrt(d))/(2*a);
        r2=(-b-sqrt(d))/(2*a);
        printf("\n root1= %f, \t root2= %f",r1,r2);
        }
        else if(d==0)
        {
        printf("\n roots are real and equal");
        r1=-b/(2*a);
        printf("\n root1= root2= %f",r1);
        }
        else
        {
        d=abs(d);
        printf("\n roots are imaginary");
        real=-b/(2*a);
        img=sqrt(d)/(2*a);
        printf("\n root1= %f+i%f",real,img);
        printf("\n root2= %f-i%f",real,img);
        }
}
```

6. Write a program to convert binary number to decimal number?

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

void bin_dec(long int num)   // Function Definition
{
long int rem,sum=0,power=0;
while(num>0)
 {
 rem = num%10;
 num = num/10;
 sum = sum + rem * pow(2,power);
 power++;
 }

printf("Decimal number : %d",sum);
}
//-----------------------------------
void main()
{
```

```
long int num;
clrscr();

printf("Enter the Binary number (0 and 1): ");
scanf("%ld",&num);

bin_dec(num);

getch();
}
```

7. **If a five digit number is input through keyboard Write a program to find the sum of its digits?**

```
#include<stdio.h>
main()
{
        long int n,d,sum,num;
        clrscr();
        printf("Enter any 5 digit number\n");
        scanf("%ld",&n);
        num=n;
        sum=0;
        if(n>9999 && n<100000)
        {
                while(n>0)
                {
                        d=n%10;
                        n=n/10;
                        sum=sum+d;
                }
                printf("the given number=%ld\n",num);
                printf("the sum of 5 digit number=%ld",sum);
        }
                else
                printf("sorry, it is wrong number:");
                getch();
}
```

8 Write a program to print all ASCII values and their equivalent characters using a while loop? The ASCII values vary from 0 to 255?

```
#include<stdio.h>
main()
{
        int i;
        clrscr();
        i=0;
        while(i<=255)
        {
                printf("%d=%c\t",i,i);
                i=i+1;
        }
        getch();
}
```

9 Write a program to print all the Armstrong numbers between 1 and 500?

```
#include<stdio.h>
main()
{
        int n,sum,a,b,i;
        clrscr();
        for(i=1;i<=500;i++)
        {
        b=i;
        sum=0;
        while(b!=0)
        {
                a=b%10;
                sum=sum+a*a*a;
                b=b/10;
        }
        if(sum==i)
                printf("%5d\n",i);
        }
        getch();
}
```

10 Write a program to display the prime factors of a given number?

```
#include<stdio.h>
main()
{
        int n,i,a,p;
        clrscr();
        printf("Enter a number\n");
        scanf("%d",&n);
        i=1;
        while(i<=n)
        {
                a=2;
                p=0;
                while(a<i)
                {
                        if((i%a)==0)
                        p=p+1;
                        a=a+1;
                }
                if(p==0)
                printf("%5d",i);
                i=i+1;
        }
                getch();
}
```

11) Write a function power(a,b)to calculate the value of a raised to b?

```
#include<stdio.h>
#include<math.h>
main()
{
        int a,b;
        long int pv;
        long int power(int,int);
        clrscr();
        printf("Enter a and b values\n");
        scanf("%d%d",&a,&b);
        pv=power(a,b);
        printf("power of a raised b value=%ld",pv);
        getch();
}
long int power(int a,int b)
{
        int i;
        long int pr=1;
        for(i=1;i<=b;i++)
                pr=pr*a;
        return pr;
}
```

12 Write a program to calculate the prime factors of a number ‘n’ entered through keyboard?

```
#include<stdio.h>
main()
{
        int n;
        void primefactors(int);
        clrscr();
        printf("Enter the limit of prime numbers\n");
        scanf("%d",&n);
        primefactors(n);
        getch();
}
void primefactors(int n)
{
        int i,a,p;
        i=1;
        while(i<=n)
        {
                a=2;
                p=0;
                while(a<i)
                {
                        if((i%a)==0)
                                p=p+1;
                                a=a+1;
                }
                if(p==0)
                        printf("%5d",i);
                        i=i+1;
                }
getch();
}
```

13. Write a program to calculate the factorial of a number using recursion?

```
#include<stdio.h>
main()
{
        int n;
        long int f;
        long int fact(int);
        clrscr();
        printf("Enter any number\n");
        scanf("%d",&n);
        f=fact(n);
        printf("factorial of a given number=%ld",f);
        getch();
}

long int fact(int n)
{
        if(n==0)
                 return(1);
        else if(n==1)
                 return(1);
        else
                 return(n*fact(n-1));
}
```

14) **.Illustrate function with no arguments and no return value?**

```
#include <stdio.h>
void message();
main( )
{
message( );
}
void message()
{
printf("have a nice day");
}
```

15) Illustrate function with arguments but does not return any value?

```
#include <stdio.h>
void sqr(int x);
void main( )
{
int x = 5;
sqr(5);
}
void sqr(int y)
{
printf("\n %d", k*k);
}
```

16) llustrate function with arguments and return values?

```
#include <stdio.h>
int sum(int x, int y);
void main( )
{
int x = 5,y=6,z;
z=sum(x,y);
printf("\n %d", z);
}
int sum(int x, int y)
{
return(x+y);
```

```
	}
```

17) Locate an element in an array, using linier search?

```
#include<stdio.h>
main()
{
	int a[20],i,n,ele,found=0;
	clrscr();
	printf("How many numbers do you want\n");
	scanf("%d",&n);
	printf("Enter %d numbers\n",n);
	for(i=0;i<n;i++)
		scanf("%d",&a[i]);
	printf("enter search an element\n");
	scanf("%d",&ele);
	for(i=0;i<n;i++)
	{
		if(a[i]==ele)
		{
			found=1;
			break;
		}
	}
	if(found==1)
		printf("%d element is found",ele);
	else
		printf("%d element is not found",ele);
	getch();
}
```

18) Write a program to locate an element in an array, using binary search?

```
	/* program to search for a element using binary search */
	#include<stdio.h>
	#include<conio.h>
	main()
	{
	int a[10],n,key,i,s,mid,l,p,f=0;
	clrscr();
	printf("enter the no. of elements into an array:");
	scanf("%d",&n);
	printf("\n enter %d elements",n);
	for(i=0;i<n;i++)
```

```
        scanf("%d",&a[i]);
        printf("\n enter the element to search");
        scanf("%d",&key);
        s=0,l=n-1;
        while(s<=l && !f)
        {
        mid=(s+l)/2;
        if(a[mid]==key)
        {
        f=1;
        p=mid;
        break;
        }
        else
        if(key<a[mid])
        l=mid-1;
        else
        s=mid+1;
        }
        if(f)
        printf("\nthe search element %d is found at position %d",key,p);
        else
        printf("\nthe element is not found");
        }
```

19) **Write a program to determine sum and average of elements in an array?**

```
#include<stdio.h>
main()
{
         int a[20],i,n,sum=0;
         float avg;
         clrscr();
         printf("How many elements do you want\n");
         scanf("%d",&n);
         printf("enter %d elements\n",n);
         for(i=0;i<n;i++)
         {
                  scanf("%d",&a[i]);
                  sum=sum+a[i];
         }
         avg=(float)sum/(float)n;
         printf("sum of given elements=%d\n",sum);
         printf("average of given elements=%f",avg);
         getch();
}
```

20) Write a program to delete elements in an array?

```
#include<stdio.h>
main()
{
        int a[20],i,j,n,ele,found=0;
        clrscr();
        printf("How many numbers do you want\n");
        scanf("%d",&n);
        printf("Enter %d numbers\n",n);
        for(i=0;i<n;i++)
              scanf("%d",&a[i]);
        printf("which element do you want to delete\n");
        scanf("%d",&ele);
        for(i=0;i<n;i++)
        {
              if(a[i]==ele)
              {
                      found=1;
                      for(j=i;j<n;j++)
                              a[j]=a[j+1];
                      n=n-1;
              }
        }
        if(found==1)
        {
              printf("%d element is found\n",ele);
              printf("array after deletion:\n");
              for(i=0;i<n;i++)
                      printf("%d\n",a[i]);
        }
        else
              printf("%d element is not found",ele);
        getch();
}
```

21) Write a program to obtain transpose of a matrix?

```
    #include<stdio.h>
    #include<conio.h>
    void read_mat(int a[][5],int m, int n);
    void disp_mat(int a[][5],int m, int n);
    void find_transpose(int a[][5],int b[][5],int m, int n);
    main()
    {
    int a[5][5],b[5][5];
    int m,n,p,q;
```

```
        clrscr();
        printf("enter the dimensions of  matrix");
        scanf("%d%d",&m,&n);
        read_mat(a,m,n);
        find_transpose(a,b,m,n);
        printf("\n the matrix A is : \n");
        disp_mat(a,m,n);
        printf("\n the transpose of A (matrix B) is : \n");
        disp_mat(b,n,m);
        }
        void read_mat(int a[][5], int m, int n)
        {
        int i,j;
        for(i=0;i<m;i++)
        for(j=0;j<n;j++)
        scanf("%d",&a[i][j]);
        }
        void disp_mat(int a[][5], int m, int n)
        {
        int i,j;
        for(i=0;i<m;i++)
        {
        for(j=0;j<n;j++)
        printf("\t%d",a[i][j]);
        printf("\n");
        }
```

22) Write a program to pick up the largest number from any 3 row by 3 column matrix?

```
#include<stdio.h>
main()
{
        int a[3][3],large,i,j;
        clrscr();
        printf("Enter any 3 X 3 column matrix:\n");
        for(i=0;i<3;i++)
              for(j=0;j<3;j++)
                      scanf("%d",&a[i][j]);
        large=a[0][0];
        for(i=0;i<3;i++)
        {
              for(j=0;j<3;j++)
                      if(a[i][j]>large)
                              large=a[i][j];
        }
        printf("given 3 X 3 matrix is \n");
        for(i=0;i<3;i++)
        {
                for(j=0;j<3;j++)
                        printf("%3d",a[i][j]);
                printf("\n");
        }
        printf("largest number in 3 X 3 matrix is= %d",large);
        getch();
}
```

23) Write a program to determine the symmetry of matrix?

```
#include<stdio.h>
main()
{
        int a[10][10],b[10][10],i,j,n,done=0;
        clrscr();
        printf("Enter the order of matrix\n");
        scanf("%d",&n);
        printf("Enter %d X %d matrix:\n",n,n);
        for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                        scanf("%d",&a[i][j]);
        printf("the given matrix is:\n");
        for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                        printf("%3d",a[i][j]);
                printf("\n");
        }
        for(i=0;i<n;i++)
                for(j=0;j<n;j++)
                        b[i][j]=a[j][i];
        printf("the transpose of given %d X %d matrix is \n",n,n);
        for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                        printf("%3d",b[i][j]);
                printf("\n");
        }
        for(i=0;i<n;i++)
        {
                for(j=0;j<n;j++)
                        if(a[i][j]!=b[i][j])
                        {
                        done=1;
                        break;
                        }
```

```
            }
            if(done==1)
                    printf("The given matrix is not symmatric");
            else
                    printf("The given matrix is symmatric");
            getch();
}
```

24 Write a program to perform addition of 3 row by 3 column matrix?

```
/*a program to find matrix addition and subtraction */
#include<stdio.h>
#include<conio.h>
void read_mat(int a[][5],int m, int n);
void disp_mat(int a[][5],int m, int n);
void add_mat(int a[][5],int b[][5],int c[][5],int m, int n);
main()
{
int a[5][5],b[5][5],c[5][5];
int m,n,p,q;
clrscr();
printf("enter the dimensions of first matrix");
scanf("%d%d",&m,&n);
printf("\nenter the dimensions of second matrix");
scanf("%d%d",&p,&q);
if(m==p && n==q)
{
printf("\nenter matrix A of dimensions %d X %d",m,n);
read_mat(a,m,n);
printf("\nenter matrix B of dimensions %d X %d",m,n);
read_mat(b,m,n);
add_mat(a,b,c,m,n);
printf("\n the matrix A is : \n");
disp_mat(a,m,n);
printf("\n the matrix B is : \n");
disp_mat(b,m,n);
printf("\n the result of A+B (matrix C) is : \n");
disp_mat(c,m,n);
}
else
printf("addition and subtraction of matrix  is not possible");
}
void read_mat(int a[][5], int m, int n)
{
int i,j;
for(i=0;i<m;i++)
for(j=0;j<n;j++)
```

```
 scanf("%d",&a[i][j]);
 }
 void disp_mat(int a[][5], int m, int n)
 {
 int i,j;
 for(i=0;i<m;i++)
 {
 for(j=0;j<n;j++)
 printf("\t%d",a[i][j]);
 printf("\n");
 }
 }
 void add_mat(int a[][5], int b[][5],int c[][5],int m, int n)
 {
 int i,j;
 for(i=0;i<m;i++)
 for(j=0;j<n;j++)
 c[i][j]=a[i][j]+b[i][j];
 }

}
```

25 Write a program to perform multiplication of 3 row by 3 column matrix?

```
/*a program to find matrix multiplication */
#include<stdio.h>
#include<conio.h>
void read_mat(int a[][5],int m, int n);
void disp_mat(int a[][5],int m, int n);
void find_mul(int a[][5],int b[][5],int c[][5],int m, int n,int q);
main()
{
int a[5][5],b[5][5],c[5][5];
int m,n,p,q;
clrscr();
printf("enter the dimensions of first matrix");
scanf("%d%d",&m,&n);
printf("\nenter the dimensions of second matrix");
scanf("%d%d",&p,&q);
if(n==p)
{
printf("\nenter matrix A of dimensions %d X %d",m,n);
read_mat(a,m,n);
printf("\nenter matrix B of dimensions %d X %d",p,q);
read_mat(b,p,q);
find_mul(a,b,c,m,n,q);
printf("\n the matrix A is : \n");
disp_mat(a,m,n);
printf("\n the matrix B is : \n");
disp_mat(b,p,q);
printf("\n the result of A*B (matrix C) is : \n");
disp_mat(c,m,q);
}
else
printf("multiplication is not possible");
}
void read_mat(int a[][5], int m, int n)
{
int i,j;
for(i=0;i<m;i++)
for(j=0;j<n;j++)
scanf("%d",&a[i][j]);
}
void disp_mat(int a[][5], int m, int n)
{
int i,j;
for(i=0;i<m;i++)
{
for(j=0;j<n;j++)
printf("\t%d",a[i][j]);
printf("\n");
}
}
void find_mul(int a[][5], int b[][5],int c[][5],int m, int n,int q)
{
int i,j,k;
for(i=0;i<m;i++)
for(j=0;j<q;j++)
{
c[i][j]=0;
for(k=0;k<n;k++)
c[i][j]=c[i][j]+a[i][k]*b[k][j];
}
}
```

26 **Write a program to compute sum of all non diagonal elements of the matrix?**

```
#include<stdio.h>
#include<conio.h>
main()
{
int a[5][5];
int m,n,i,j,sum=0;
clrscr();
```

```
        printf("enter the dimensions of  matrix");
        scanf("%d%d",&m,&n);
        if(m==n)
        {
        printf("enter a square matrix of dimension %d X %d",m,m);
        for(i=0;i<m;i++)
        for(j=0;j<m;j++)
        scanf("%d",&a[i][j]);
        printf("\n the given matrix  is : \n");
        for(i=0;i<m;i++)
        {
        for(j=0;j<n;j++)
        printf("\t%d",a[i][j]);
        printf("\n");
        }
        for(i=0;i<m;i++) /* finding sum of the diagonal elements */
        for(j=0;j<n;j++)
        if(i!=j)
        sum=sum+a[i][j];
        printf("\nthe sum of the non diagonal elements is %d",sum);
        }
        else
        printf("given matrix is not a square matrix");
}
```

27 **Write a program to merge two arrays**

```
#include <stdio.h>

void merge(int [], int, int [], int, int []);

int main() {
  int a[100], b[100], m, n, c, sorted[200];

  printf("Input number of elements in first array\n");
  scanf("%d", &m);

  printf("Input %d integers\n", m);
  for (c = 0; c < m; c++) {
    scanf("%d", &a[c]);
  }

  printf("Input number of elements in second array\n");
  scanf("%d", &n);

  printf("Input %d integers\n", n);
  for (c = 0; c < n; c++) {
    scanf("%d", &b[c]);
  }

  merge(a, m, b, n, sorted);

  printf("Sorted array:\n");

  for (c = 0; c < m + n; c++) {
    printf("%d\n", sorted[c]);
  }

  return 0;
}

void merge(int a[], int m, int b[], int n, int sorted[]) {
  int i, j, k;

  j = k = 0;

  for (i = 0; i < m + n;) {
    if (j < m && k < n) {
      if (a[j] < b[k]) {
        sorted[i] = a[j];
        j++;
      }
      else {
        sorted[i] = b[k];
        k++;
      }
      i++;
    }
    else if (j == m) {
      for (; i < m + n;) {
        sorted[i] = b[k];
        k++;
        i++;
      }
    }
    else {
      for (; i < m + n;) {
        sorted[i] = a[j];
        j++;
        i++;
      }
    }
  }
}
```

28 Write a macro definition to determine weather the character entered is small case letter or not?

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
#include<string.h>
#define LOWER(ch) (ch>=97&&ch<=122 ? printf("\nlower"):printf("\n not
lower"))
{
char ch;
clrscr();
printf("\n enter the character=");
scanf("%c",&ch);
LOWER (ch);
getch();
}
```

29.Write a macro definition with arguments for calculation of the area of a square and circle?

```
#include<stdio.h>
#include<conio.h>
#define PI 3.141
#define AREA(r) PI*r*r
#define SQRAREA(r)  r *r
main()
{
float r;
clrscr();
printf("enter r value :");
scanf("%f",&r);
printf("area of circle = %f",AREA(r));
printf("area of circle = %f", SQRAREA (r));

}
```

30 Write a program to accept your name and then

a. Display it in upper case?

b. Display its length in characters?

c. Display its reverse?

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

```
main()
{
        char name[20];
        int i,len;
        clrscr();
        printf("Enter any name in lowercase\n");
        scanf("%s",name);
        i=0;
        printf("Given name in uppercase:=");
        while(name[i]!='\0')
        {
              printf("%c",toupper(name[i]));
              i=i+1;
        }
        printf("\n");
        for(len=0;name[len]!='\0';len++)
        printf("given string in reverse order:=");
        for(i=len;i>=0;i--)
        printf("%c",name[i]);
        printf("length of given name:=%d\n",len);
        getch();
}
```

31 Write a program to accept a string and then display the number of vowels and consonants contained in it?

```
main()
{
int vow=0;
int cons=0,i;
char x[20];
clrscr();
printf("enter any string\n");
scanf("%s",&x);
for(i=0;x[i];i++)
{
if(x[i]=='a'||x[i]=='e'||x[i]=='i'||x[i]=='o'||x[i]=='u')
vow++;
else
cons++;
}
printf("number of vowels=%d",vow);
printf("\nnumber of consonants=%d",cons);
getch();
}
```

32 Write a program to check weather given string is palindrome or

not?

```
#include<stdio.h>
main()
{
        char str[20];
        int i=0,j,flag;
        clrscr();
        printf("enter the string\n");
        scanf("%s",str);
        j=strlen(str)-1;

        while(i<=j)
        {
              if(str[i]==str[j])
                      flag=1;
              else
              {
                      flag=0;
                      break;
              }
        i++;
        j--;
        }
        if(flag==1)
              printf("given string is palindrome");
        else
              printf("given string is not palindrome");
        getch();
}
```

33 Write a program to concatenate 2 strings?

```
#include<stdio.h>
main()
{
        char s1[20],s2[20],s3[40];
        int i=0,k;
        clrscr();
        printf("Enter first string\n");
        gets(s1);
        printf("Enter second string\n");
        gets(s2);
        while((s3[i]=s1[i])!='\0')
              i=i+1;
        k=0;
        while((s3[i+k]=s2[k])!='\0')
              k=k+1;
        printf("concatenated string is %s\n",s3);
        getch();
}
```

OR

```
#include<stdio.h>
#include<string.h>
main()
{
char x[10],y[10];
clrscr();
printf("enter any two strings\n");
scanf("%s%s",x,y);
strcat(x,y);
printf("\n%s",x);
getch();
}
```

34 Write a program to swap two numbers using pointers?

```
#include<stdio.h>
main()
{
        int a,b;
        void swap();
        clrscr();
        printf("Enter any two numbers:\n");
        scanf("%d%d",&a,&b);
        printf("Before swapping a=%d   b=%d\n",a,b);
        swap(&a,&b);
        printf("After swapping a=%d   b=%d\n",a,b);
        getch();
}
void swap(int *a, int *b)
{
        int temp;
        temp=*a;
        *a=*b;
        *b=temp;
}
```

35 Write a program to find the smallest element in an array using pointers?

```
#include<stdio.h>
main()
{
        int i,n,small,a[20],*ptr;
        clrscr();
        printf("Enter the size of an array\n");
```

```
    scanf("%d",&n);
    printf("Enter %d elements:\n",n);
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);
    ptr=a;
    small=*ptr;
    ptr++;
    for(i=1;i<n;i++)
    {
        if(small>*ptr)
            small=*ptr;
        ptr++;
    }
    printf("the given %d elements are\n",n);
    for(i=0;i<n;i++)
        printf("%3d\n",a[i]);
    printf("the smallest element is: %2d",small);
    getch();
}
```

36 Write a program to process student records using structures?

```
#include<stdio.h>
#include<conio.h>
struct student
{
int rno;
char name[10];
int marks;
};
void show_data(struct student s); /* structure variable as argument */
main()
{
struct student s = {11, "rama",201}; /*structure variable initialization */
clrscr();
show_data(s);
}
void show_data(struct student s)
{
printf("\n student details");
printf("\n rno=%d",s.rno);
printf("\n name=%s",s.name);
printf("\n marks=%d",s.marks);
}
```

37 Write a program to process employee records using structures?

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

```
#include <conio.h>

struct details
{
 char name[30];
 int age;
 char address[500];
 float salary;
};

int main()
{
 struct details detail;
 clrscr();
 printf("\nEnter name:\n");
 gets(detail.name);
 printf("\nEnter age:\n");
 scanf("%d",&detail.age);
 printf("\nEnter Address:\n");
 gets(detail.address);
 printf("\nEnter Salary:\n");
 scanf("%f",&detail.salary);


 printf("\n\n\n");
 printf("Name of the Employee : %s \n",detail.name);
 printf("Age of the Employee : %d \n",detail.age);
 printf("Address of the Employee : %s \n",detail.address);
 printf("Salary of the Employee : %f \n",detail.salary);

 getch();
}
```

38 Write a program to illustrate nesting of structures?

```
program to demonstrate nesting of structure */
#include<stdio.h>
#include<conio.h>
struct time
{
int hour;
int minute;
int second;
};
struct tt
{
int carno;
struct time st;
struct time rt;
};
main()
{
struct tt r1;
clrscr();
printf("\n enter Car no. starting time reaching time:");
scanf("%d",&r1.carno);
scanf("%d%d%d",&r1.st.hour,&r1.st.minute,&r1.st.second);
scanf("%d%d%d",&r1.rt.hour,&r1.rt.minute,&r1.rt.second);
printf("\n\tCar No.\tstarting time  \treaching time:\n");
printf("\t%d",r1.carno);
printf("\t%d:%d:%d",r1.st.hour,r1.st.minute,r1.st.second);
printf("\t%d:%d:%d",r1.rt.hour,r1.rt.minute,r1.rt.second);
}
```

39 Write a program to illustrate any two modes in which a file can be opened?

```
/* a program to perform file operations. the basic file operations include
   naming a file, opening a file, reading data from file
   and writing data to file */
#include<stdio.h>
#include<conio.h>
void main()
{
FILE  *fp;
char c;
clrscr();
printf("\n enter the contents for a file and press F6 to END \n");
fp=fopen("input.dat","w"); /* opening and naming file */
while((c=getchar())!=EOF)
putc(c,fp);             /*  writing data into a file */
fclose(fp);
printf("\n the contents of the file input.dat \n");
fp=fopen("input.dat","r");
while((c=getc(fp))!=EOF)   /* reading data from file */
printf("%c",c);           /* closing file */
fclose (fp);
}
```

40 Write a program to create binary file?

```
#include<stdio.h>
#include<conio.h>
main()
{
        struct std
        {
                int hno;
                char name[15];
                int m1,m2,m3;
        }s;
        FILE *fp;
        char ch='y';
        clrscr();
        fp=fopen("std1.dat","wb");
        do
        {
                printf("enter name\n");
                scanf("%20s",s.name);
                printf("enter hno\n");
                scanf("%4d",&s.hno);
                printf("enter 3 subjects marks\n");
                scanf("%3d%3d%3d",&s.m1,&s.m2,&s.m3);
                fwrite(&s,sizeof(s),1,fp);
                printf("enter your choice y for continue n for stop");
                ch=getchar();
        }
        while(ch=='y');
        fclose(fp);
        getch();
}
```

VIVA QUESTIONS

What is C?
It is a general purpose programming language. It is developed by Dennis Ritche in 1972 from the features of a language called B.

What is IDE?
An **integrated development environment** (IDE) also known as *integrated design environment* or *integrated debugging environment* is a software application that provides facilities to computer programmers for software development.

Program:
It is sequence of instructions with specified syntax in a programming language to perform a task.

Token
The smallest individual unit in c program is called token. The tokens in c program are listed below:

- Keywords, Variables, Constants, Special characters, Operators

Keywords
The C keywords are reserved words and have fixed meanings. There are 32 keywords in C

Variable
A variable is used to store values. Every variable has memory location.
The variable can hold single value at a time. The program can change contents of variable.

Constants
The constants in c are applicable to the values which do not change during execution of program. C supports various types of constants including integers, characters, floating and string constants.

IDENTIFIERS:
- Identifiers are names of variables, functions and arrays etc. they are user-defined names.
- Identifier contains a sequence of letters and digits, with a letter as a first character. Lower case letters are preferred. However, the upper case letters are also permitted. But C compiler treats lower and upper case letters differently.

Data type:
In C each variable is attached with some data type.
The data type defines:
4. The amount of storage allocated to variables.
5. The values that they can accept.
6. The operations that can be performed on variables.

DATATYPES IN C
C data types can be classified into the following categories:
- Basic data type

- Derived type
- User defined type
- Void data type

OPERATOR
An operator is a symbol that operates on a certain data type and produces the output as the result of the operation
Unary Operators
A unary operator is an operator, which operates on one operand.
Binary
A binary operator is an operator, which operates on two operands
Ternary
A ternary operator is an operator, which operates on three operands.

Difference between equality operator (==) and the assignment operator (=) The equality operator is used to compare the two operands for equality (same value), whereas the assignment operator is used for the purposes of assignment.
c = (a>b) ? a+b : b-a;
Conditional Operator
A conditional operator checks for an expression, which returns either a true or a false value. If the condition evaluated is true, it returns the value of the true section of the operator, otherwise it returns the value of the false section of the operator.

Its general structure is as follows:

Expression1 ? expression 2 (True Section): expression3 (False Section)
Bit wise operators:
The bitwise operators operate on sequences of binary bits. Bitwise operations are necessary for much low-level programming, such as writing device drivers, low-level graphics.
Bitwise operators include:

&	AND
|	OR
^	XOR
~	one's compliment
<<	Shift Left
>>	Shift Right

Escape Sequences

Character combinations consisting of a backslash (\) followed by a letter or by a combination of digits are called "escape sequences." To represent a newline character, single quotation mark, or certain other characters in a character constant, you must use escape sequences. An escape sequence is regarded as a single character and is therefore valid as a character constant.
Example : \n, \t, \r

Input Functions in C
Scanf(), getch(), gets(), getchar(), getche().
Output Functions in C
Printf(), putch(), puts(), putchar().
Comment:
A "comment" is a sequence of characters beginning with a forward slash and asterisk combination (**/***). It is treated as a single white-space character by the compiler.
TYPECASTING
Typecasting refer conversion of type. Typecasting is essential when the values of variables are to be converted from one type to another type. C allows implicit as well as explicit conversion
Control structure - One of the statements in a programming language which determines the sequence of execution of other instructions or statements (the control flow) are called control structures.
Different types of control structures are listed below:

- Conditional control structure
- Loop control structure
- Jump control structure
- Multi way conditional control structure / case control structure

Conditional control statements in c:
The conditional control structure include the following

- if , if else, nested if else, else if ladder

Loop : A loop is a sequence of statements which is specified once but which may be carried out several times until some condition becomes false.

Loop control statements in C

- while loop, do while loop, for loop

Jump control statements in C

- break statement, continue statement, goto statement

Nested loop

- Using a loop within another loop is called nested loop.

Function:

- Function is a block of statements that solve a task or sub task of other task.
- C supports two types of functions: library functions and user defined functions.

Function prototype

- The function prototype specifies the following:
  - Name of the function,
  - Type of value it returns
  - Type and number of arguments it takes.

Actual and formal Arguments

The arguments declared in caller function and mentioned in the function call are called as actual arguments. The arguments declared in the function header are known as formal arguments

Different Parameter passing techniques:

There are 2 methods by which we can pass values to the function. These methods are:

3. Call by value (pass by value)
4. Call by reference (pass by reference)

Pass by value

In this type, value of actual argument is passed to the formal argument and operation is done on the formal arguments. Any change in the formal argument does not effect the actual argument

Pass by reference

In this type, instead of passing values, addresses are passed. Function operates on address rather than values. Here the formal arguments are pointers to the actual arguments. Hence changes made in the arguments are permanent.

Recursion:

A recursive function is a function that calls itself

Storage classes in c

Four storage classes in C are :

- automatic , static , register , extern

Local or automatic variable:

- It is declared by auto int or int keywords
- It is declared inside the function
- If it is not initialized **garbage** value is stored
- It is created when function is called and lost when function terminates

Global variable:

- It is declared outside the function
- If it is not initialized , **0** is stored by default
- It is created before execution starts and lost when program terminates
- It is visible to **entire program**

Array :

An array is collection of elements of same data type that share a common name. Array elements are stored in adjacent memory locations. To refer the array elements we use index or subscript. The array index starts with zero. So the index of last element is always n-1 where n is the size of array

Advantages of arrays

- Array is capable of storing many elements at a time with a common name
- We can assign the element of an array to any ordinary variable or another array element of same data type.
- Any element of array can be accessed randomly

STRUCTURE

structure is data type that combines logically related data items of different data type. In C structure combines the data members of different types.

Unions

Union is a user defined data type that encapsulates different data types. All the members of the union share common memory area.

String

A string is an array of characters that is terminated with null character '\0'. Any group of characters defined between double quotation marks is a constant string.

Pointers

Pointer variable is a variable that holds address of simple variable of same type.

Preprocessor:

Preprocessor is a program that processes the code before it passes through the compiler. Preprocessor directives are placed in the source program before the main line. Before the source code passes through the compiler it is examined by the preprocessor. If it is appropriate then the source program is handed over to the compiler.

SATAVAHANA UNIVERSITY
B.COM (COMPUTERS) (I YEAR)
Programming Concept Using C
Practical Paper-II

Note: candidates can strike off ONE question at his choice from the following questions. One question from out of the rest will be allotted to the candidate by the Examiner.

1. Write a program to determine the largest of three numbers?
2. Write a program to print Fibonacci series on N numbers?
3. Write a program to check weather given number is palindrome or not?
4. Write a program to find the area of a triangle?
5. Write a program to determine the roots of a quadratic equation?
6. Write a program to convert binary number to decimal number?
7. If a five digit number is input through keyboard Write a program to find the sum of its digits?
8. Write a program to print all ASCII values and their equivalent characters using a while loop? The ASCII values vary from 0 to 255?
9. Write a program to print all the Armstrong numbers between 1 and 500?
10. Write a program to display the prime factors of a given number?
11. Write a function power(a,b)to calculate the value of a raised to b?
12. Write a program to calculate the prime factors of a number 'n' entered through keyboard?
13. Write a program to calculate the factorial of a number using recursion?
14. Illustrate function with no arguments and no return value?
15. Illustrate function with arguments but does not return any value?
16. Illustrate function with arguments and return values?
17. Write a program to locate an element in an array, using linier search?
18. Write a program to locate an element in an array, using binary search?
19. Write a program to determine sum and average of elements in an array?
20. Write a program to delete elements in an array?
21. Write a program to obtain transpose of a matrix?
22. Write a program to pick up the largest number from any 3row by 3column matrix?
23. Write a program to determine the symmetry of matrix?
24. Write a program to perform addition of 3row by 3column matrix?
25. Write a program to perform multiplication of 3row by 3column matrix?
26. Write a program to compute sum of all non diagonal elements of the matrix?
27. Write a program to merge two matrices?
28. Write a macro definition to determine weather the character entered is small case letter or not?
29. Write a macro definition with arguments for calculation of the area of a square and circle?
30. Write a program to accept your name and then

a) Display it in upper case?

b) Display its length in characters?
c) Display its reverse?

31. Write a program to accept a string and then display the number of vowels and consonants contained in it?
32. Write a program to check weather given string is palindrome or not?
33. Write a program to concatenate 2 strings?
34. Write a program to swap two numbers using pointers?
35. Write a program to find the smallest element in an array using pointers?
36. Write a program to process student records using structures?
37. Write a program to process employee records using structures?
38. Write a program to illustrate nesting of structures?
39. Write a program to illustrate any two modes in which a file can be opened?
40. Write a program to create binary file?