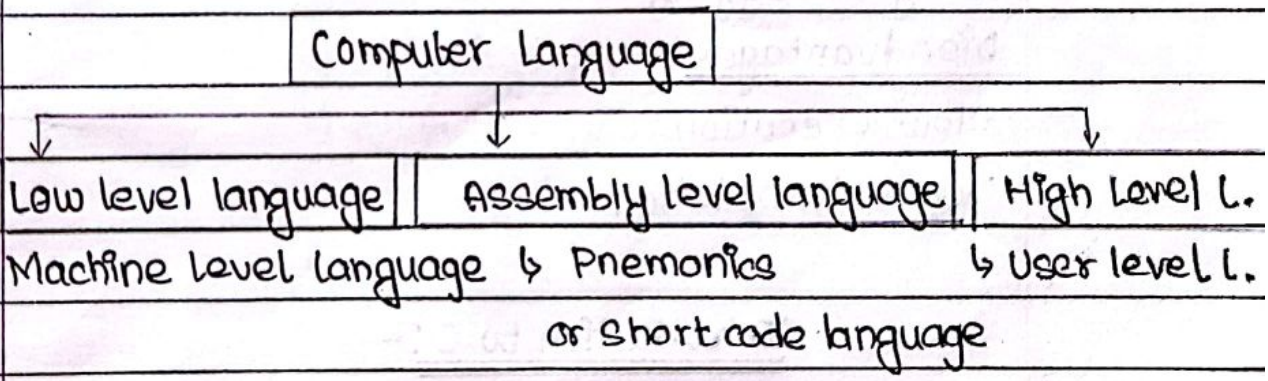
Unit-1

Problem solving with Computer

* Software:- It is a collection of programme.
* Programme:- It is a collection of instruction.
* Programming language:- The language which are used to instruct the computer to perform certain task are called programming language.
  → Programming language is the native language of the computer.
  → Eg: C, C++, Java, Python



1. Low level language:-

Advantages

→ Translation free
→ High speed / Fast execution

Disadvantages

→ Difficult to write program
→ Error prone

2. Assembly level language :-

Advantages

→ Easy control on hardware

→ Fast execution

Disadvantages

→ Difficult to modify & trouble shoot program.

→ User needs to remember machine codes.

3. High level language :-

Advantages

→ Machine independent

→ Easy Debugging

Disadvantages

→ Slow execution

→ Needs translation

Introduction to C :-

→ C is the general-purpose, structured and procedural middle level programming languages i.e extremely popular, simple & flexible to use.

→ C is developed by Denis Ritchie at Bell Laboratory in 1970 A.D.

1972

Date
Page

Features of C- language :-

1. **Simple :-** C is the simple & easy programming language. It makes the language easily comprehensible & enables a programmer to redesign or create a new application.

2. **Structure :-** C is a general-purpose structured language. which allows you to break a code into different parts using functions which can be stored for future use.

3. **Portable :-** C language is portable which provides the functionality of using a single code on multiple systems depending on the requirement.

4. **Fast execution :-** C is a statically typed programming language which are faster than dynamic ones & gives it an edge over other dynamic languages

5. **Middle level Programming Language :-**

Although C was initially developed to do only low level programming, it now also supports the features & functionalities of high level programming making it a middle level language.

6. Rich library sets :- C comes with an extensive set of libraries with several built in function that make the life of a programmer easy.

7. Modular programming language :- Modularity is storing C-programming language code in the form of libraries for future use.

8. Better memory management :- C-language supports using pointer for dynamic memory management. It means you can utilize & manage the size of data structure in C.

Date
Page

* History of C-language :-

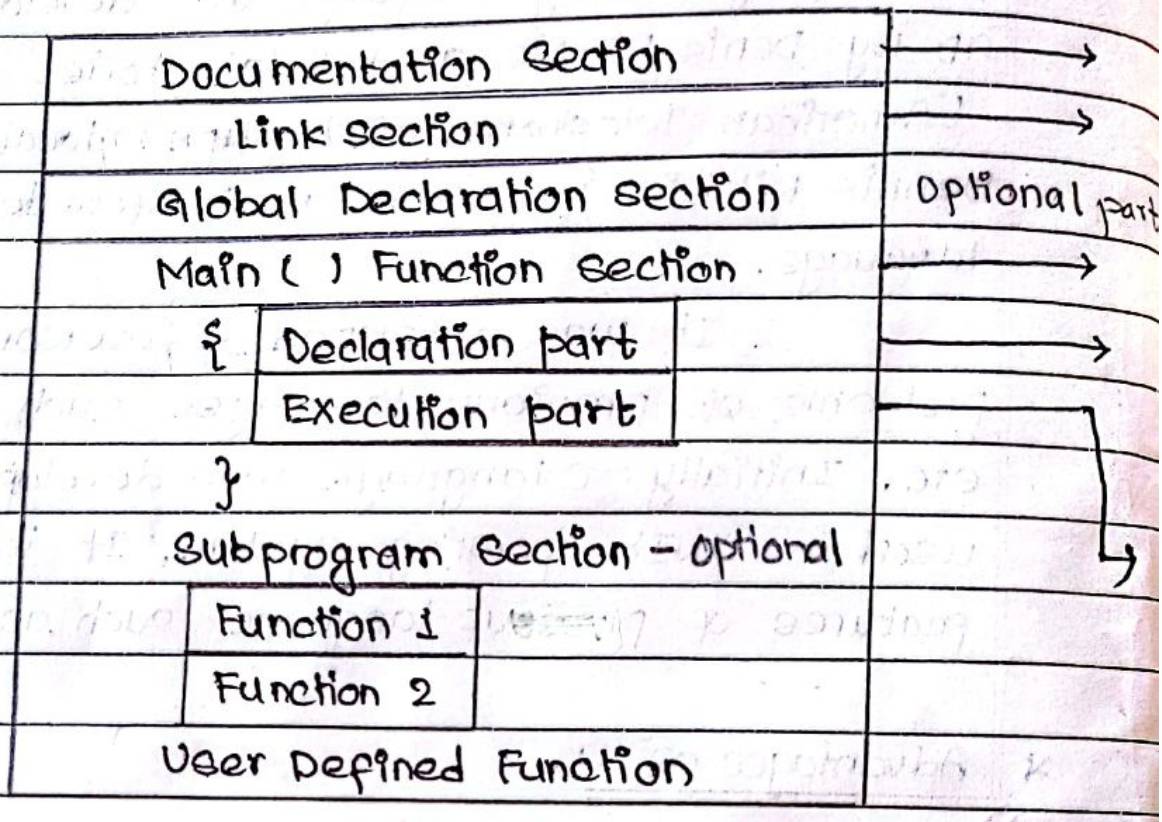
C programming language was developed in 1972 AD by Denis Ritchie at bell labratories of AT&T (American Telephone & Telegraph) located in USA. Dennis Ritchie is known as the founder of C-language.

It was developed to overcome the problems of previous languages such as B, BCPL etc. Initially, C language was developed to be used in UNIX operating system. It inherits many features of previous language such as B & BCPL.

* Advantages of C :-

→ Easy to learn
→ Low level language support
→ Presence of many libraries
→ Produce Portable program
→ Powerful programming language.
→ Easy debbuging
→ Procedure Orentea Language

* Structure of C-language :-



Date
Page

// Program to add two numbers

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

std- standard io - input output

```
void main ( )
{ int a, b, sum;
  printf ("Enter first number");
  scanf (" %d", &a);
  print f ("Enter second number");
  scan f ("%d", &b);
  sum = a+b
  Print f ("sum is %d", sum); }
```

int - integer

To put →

To show →

Output

Enter first no. 2
Enter sec. no. 3
Sum is: 5

* Write a program to calculate area of rectangle.

Area LxB

```
// Program to calculate area of rectangle
# include <stdio.h>
void main ( )
{ int l, b area ;
  printf (" Enter the length");
  scanf (" %d" &l);
  print f (" Enter the breadth");
  scanf (" %d" &b);
  Area = l * b
  Print f (" Area is %d", Area); }
```

* WAP to calculate simple interest:

· SI = (P * T * R) / 100

```
// Program to calculate simple interest
# include <stdio.h>
void main( )
{ int P, T, R SI ;
printf ("enter two numbers");
scanf ("%d %d, &P, &T");
printf ("enter third number");
Scanf ("%d, & R");
 SI =(P * T * R) / 100;
Printf ("SI is %d", SI ); }
```

Date
Page

* Description:-

→ The documentation section consists of sets of comment lines giving the name of the program.

→ The link section provides instructions to the complier to link function from the system library.

→ The global declaration section is used to define global variables.

→ The main function section is used to combine all the user-defined functions. It consists of two parts :-

1. Declaration parts declares all the variables used in the executable parts.
2. Execution part is the main part of the program that defines the execution process of the program.

→ Sub program section contains all the user-defined functions.

→ User-defined functions are generally placed immedia after the main function.

Date
Page

* Compilation and Execution Process :-

The code of the program written in high level programming language is called source code. The machine level code i.e. set after compilation is called object code. The compilation is the process of converting source code into object code. It is done with the help of compiler.

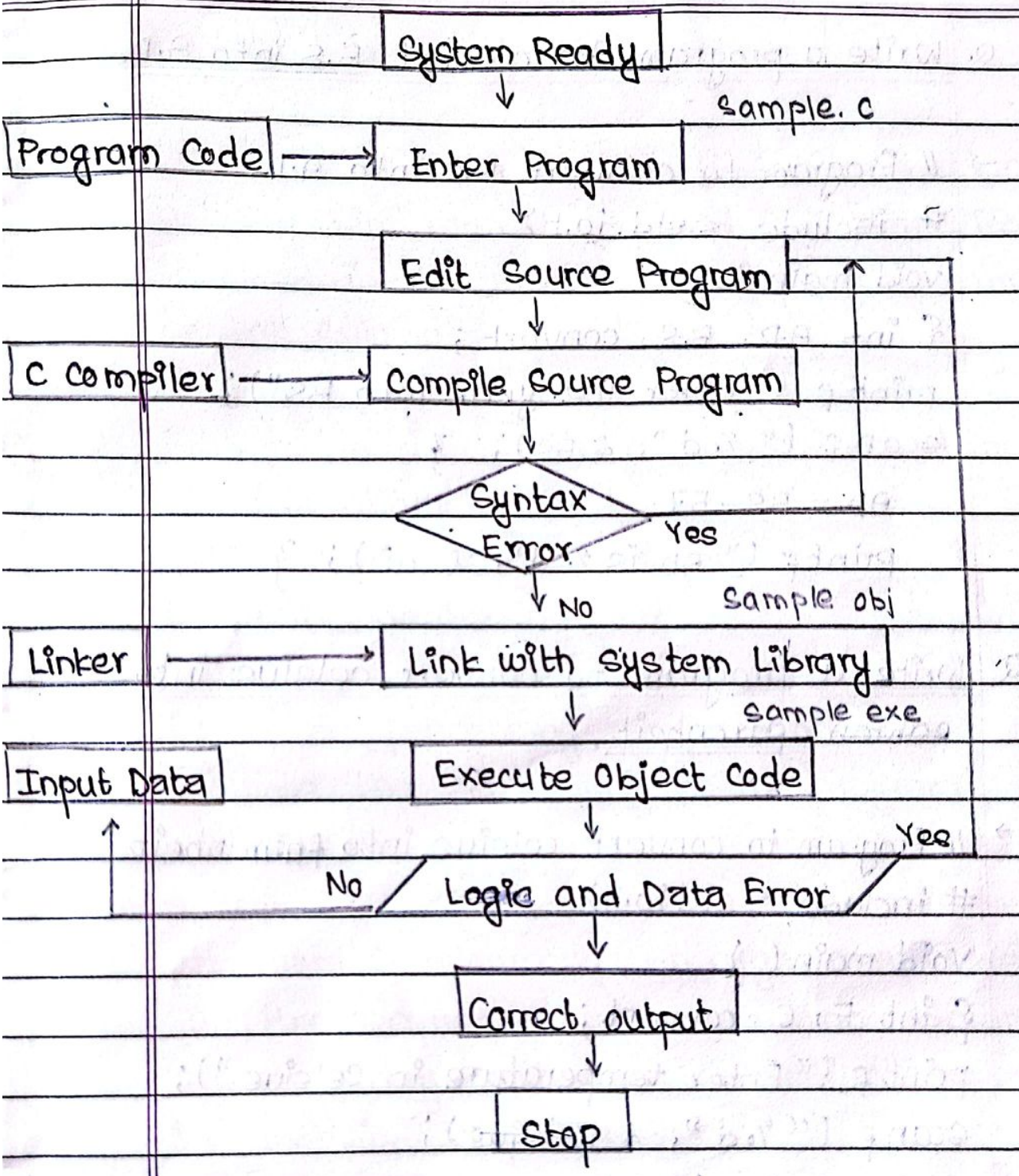


Fig. Process of compilation

Q. Write a program to convert B.S into A.D.

→
```
// Program to convert B.S into A.D.
# include <std.io.h>
void main ( )
{ int A.D, B.S, convert ;
printf (" Enter the year into BS");
scanf (" %.d", & BS);
    AD = BS-57
    printf (" AD is %.d", & AD); }
```

Q. Write a program to convert celsius into farenheit.

→
```
// Program to convert celsius into fahrenheit
# include < std io.h>
void main ( )
{ int F, C, convert;
print f (" Enter temperature in celsius");
scanf ("%d", & C );
        F =(C  * 9/5)+ 32;
print f (" Temp in fahrenheit is %d, F );
}
```

* Problem Solving Using Computer :-

Problem is defined as the difference betn an existing situation & desired situation. The steps involved in the problem solving using computer are as follows:-

1. Problem Analysis :- The activities involved in the problem analysis are :-
- Define objectives of a program.
- Determine desire outputs.
- Determine desire inputs.
- Determine processing requirements.
- Evaluate the feasability of the program.
- Document the analysis

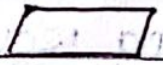
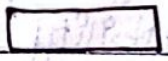
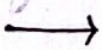
2. Algorithm Development:- An algorithm is defined as the sequence of instructions designed in such a way that the instructions are execut in specified sequence.

Q. Write a algorithm to find sum of numbers.

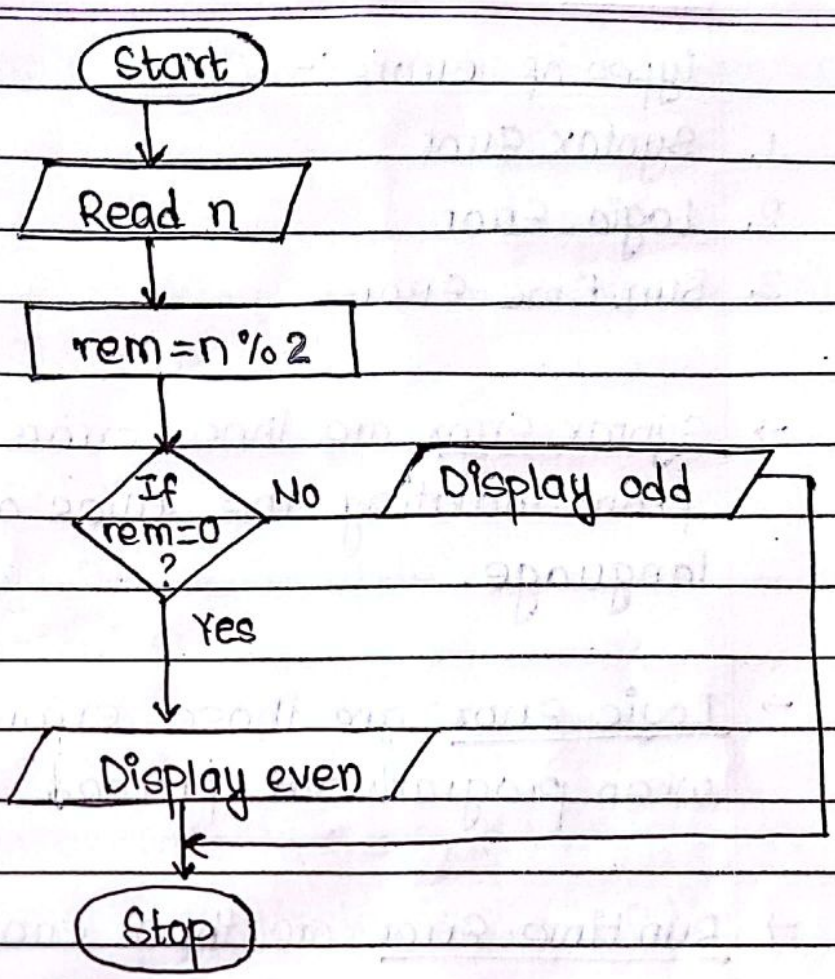
Step 1: Start
Step 2: Read two numbers a, b
Step 3: sum = a+b
Step 4: Display sum (Print)
Step 5: Stop

3. Flowchart Development:- Flowchart is a pictorial (picture) representation of the Algorithm.

Flowchart Symbols:-

Name	Symbol	Use in flowchart
Oval		Start / stop
Parallelogram		Input / Output
Rectangle		Processing
Flowline		Direction of Flow of Logic
Diamond		Decision making
Connector		Connects parts of flowchart

Q. Make a flowchart to determine whether the number is odd or even.



4. **Program Coding**:- Coding is the process of transforming program logic into a computer language format.

5. **Compilation & Execution**:- It is the process of converting source code (high level) into object code (machine level).

6. **Debugging and Testing**:- The process of finding & removing errors from the program is called Debugging. Generally, programmers commit three

types of errors:-

1. Syntax Error
2. Logic Error
3. Run time Error

→ Syntax Error are those errors which are arrived from violating the rules of programming language.

→ Logic Error are those errors which arrised when programmar proceed the wrong logic.

→ Run time Error are those errors which attempt to run ambigous instructions.

7. Program Documentation:- Program documentation refers to the details that describes a program. There are two types of documentation :-

1. Internal Documentation
2. External Documentation

* Write a program to find area & ~~circumference~~ of circle.

```
// Program to find area & circumference of a circle
# include <stdio.h>
void main ( )
{ int    .    .r, A, P ;
printf (" Enter radius of circle");
scanf (" %d", &r );
  A= 3.1415 * r * r ;
  P= 2* 3.1415 * r ;
printf (" Area of circle is %d", A );
printf (" Perimeter of circle is %d", P);
 }
```

Unit-2

Elements of C

1. C Tokens

- In C program, the smallest individual units are known as C Tokens.
- C has 6 types of tokens:-

i. Identifiers
ii. Keywords
iii. Constants
iv. Strings
v. Operators
vi. Special symbols

i. **Identifiers :-** Identifiers are the names that are given to various program elements such as variables, functions and arrays.

Rules for identifiers:-

- First character must be an alphabet (or underscore).
- Must contain only letters, digits, or underscore.
- Cannot use a keyword.
- Mustn't contain white space.

ii. **Keywords :-** There are certain reserved words called keywords that have standard, pre-defined meanings

in C. Eg. auto, break, case, float, for, go to, if, int. etc. Keywords are lowercase.

* **Datatypes**:- C supports several data types, each of which may be represented differently within the computer memory.

C data types

Primary Data Type (Basic)	Derived Data Type	User-defined Data Type
- int	- array	- structure
- float	- function	- union
- char	- pointer	- Enumeration
- double		

1. **Primary Data Types**:- The primary datatypes are also known as built-in data types.

Data types	Description	Memory Requirement
int	Integer quantity	2 bytes
float	Floating point no.	4 bytes
char	single character	1 bytes
double	Double precision Floating point no.	8 bytes

2. Derived DataTypes :- The data types which are derived from the fundamental data types are called Derived data types. Eg: function, array etc.

Q. Write a program to find the average exam scores of a 5 student.

```
// Program to find the average exam score of 5 studen
# include <stdio.h>
void main()
{  int A, B, C, D, E, avg;
   printf ("Enter the score of 5 students");
   scanf ("%d %d %d %d %d", &A, &B, &C, &D,
                &E);
   Avg=(A+B+C+D+E)/5;
   printf (" The average score of five student
            is %d", avg);
}
```

Q. Write a program to find the net price of a computer.

[Net price = Basic price - 30% discount + 5% VAT]

[Discount = B.P - 30% of B.P]

[VAT = B.P + 5% of B.P]

```
# include <stdio.h>
void main ()
{ int BP, discount, VAT, NP;
  printf (" Enter B.P of a computer ");
  scanf (" %d", & B.P );
  discount = B.P - (30/100) * B.P;
  VAT = BP + (5/100) * BP;
  NP = BP - discount + VAT;
  printf (" The net price of a computer is %d", NP);
}
```

Q Write a program to find the compound interest.

$$CI = P\left[\left(1+\frac{R}{100}\right)^{T}\right]$$

```
#include <stdio.h>
void main ()
{ float P, R, T, CI;
  printf (" Enter principal, rate, time ");
  scanf (" %f %f %f", & P, & R, & T);
  CI = P (1+R/100)      CI = P * pow ((1+R/100), T);
printf (" The compound interest is %f ", CI);
}
```

3. User-defined DataTypes :- The datatypes which are defined by the users are called User-defined data types. User-defined datatypes are structure, union, enumeration.

iii. Constants / Literal :- A value that doesn't change during the execution of the program. There are four types of constant in C.

1. Integer constant
2. Real constant
3. Character constant
4. String constant

(1.) Integer constant :- A constant which consists of whole number that may be positive or negative including 0 are called integer constant.

(2.) Real constant :-
(Floating point constant)
This type of constant must contain decimal as well as integers part. Eg: 3.1415 where 3 is integer & .1415 is decimal.

(3.) Character constant :- The character constant is a

printable alpha-numeric character, digits and special character that must be written within a pair of single quotes.

4. String constant:- It is the combination of different characters enclosed with double quotes.

* Variables:-

- Variables is the identifiers that is used to represent a single data items i.e. numerical quantity or character constant.
- Variables may take different values during different times during execution.

Rules for writing variables:- (Same as 'identifiers)

- They must begin with letters.

* Declaration of variables:-

Eg:

```
int length, breadth;
float marks;
char name [20];
char a;
```

Q. Write a program to calculate bonus if the company provides 30% bonus of basic salary to the employ

```
#include <stdio.h>
void main()
{ int BS, B;
  printf("Enter the basic salary");
  scanf("%d", &BS);
  B = (30/100) * BS;
  printf("Bonus is %d", B);
  }
```

Q. Write a program to find the age of father if the age is 3 times of the age of son

```
#include <stdio.h>
void main()
{ int S, F;
  printf("Enter the age of son");
  scanf("%d", &S);
  Age of Father = 3*S ;
  printf("The age of father is %d", F);
```

* Escape sequences :- Escape sequence is a non-printing character used in C. It is a character combination consisting of back slace (\). eg:
\n - next line
\t - Horizontal tab
\v - Vertical tab

* Expression :- Expression represents a single data item, such as number or a character. It may consists of some combinations of such entities interconnected by one or more operators.

* Write a program to compute ~~Quotient~~ by displaying remainder & Quotient.

```
# include <stdio.h>
 void main ()
{ int divi, div, quo, rem;
   printf ("Enter dividend & divisor");
   scanf ("%d %d", & divi & div);
   quo = divi / div;
   rem = divi % div;
   printf ("Quotient is %d and Remainder is %d
            quo, rem);
   }
```

* Symbolic constant:- A symbolic constant is a name that substitute for a sequence of characters. The characters may be numberic, or string character.

```
# include <stdio.h>
# define Var 56
 main()
{
  int b;
   b = 2+var;
  printf = ("%d", b);
   }
```

Q. Program to input character.

```
#include <stdio.h>
 void main()
{ char a;
 printf ("Enter any character");
 scanf ("%c", &a);
 printf ("character is %c", a);
 }
```

Q. Write a program to find ASCII value of input character.

```
#includo
void main ( )
{   char a;
  printf (" Enter a character");
  scanf (" %c", &a);
  printf (" ASCII value of %c is %d", a, a);
}
```

```
{ char a;                          [Output]
  printf (" Enter char");          (A)
  scanf (" %c", &a);
  printf (" %c ", a);      // A     [output = A
  printf (" %d ", a);      // 65    output = 65]
}
```

Q. Write a program to enter your name & display it.

```
void main()
{
  Char name [50];
  printf ("Enter your name");
  scanf ("%s", name);
  printf ("Name is %s", name);
}
```

[There should not be '&' in string]

```
scanf ("%[^\n]s", name);
```

Input and Output

→ A computer program takes input generally from keyboard (standard input device), then process it and sends the result to output device.

→ There are various input & output functions in C.

→ The input functions take the data from the input devices & transfers into the program. The standard input functions are: scanf(), getchar(), getche(), gets().

→ The output function send the result from the program to the output devices. The standard output functions are: printf(), putchar(), putch(), puts() etc.

→ The input/output functions are classified into two types:

1. Formatted I/O functions
2. Unformatted I/O functions

1. Formatted I/O:-

The function which allows input or output to format according to users requirement is known as formatted I/O function.
For example: they can be used to specify the number of digits to be displayed after the decimal point, number of spaces before data items and position where the output is to be displayed. The examples for formatted i/o are scanf(), printf().

* Formatted Input:-

The built-in functions scanf() can be used to input data into the computer from a standard input device. The general form of scanf is:

(syntax)

- scanf(" control string ", arg 1, arg 2, ..., arg n)

- The control string refers to the field format in which the data is entered. The arguments arg1, arg 2, ..., arg n specify the address of the locations where the data is stored.

The control string again contains multiple parts. The general form is:

[whitespace_character] [ordinary_character] % [field width] conversion_character.

1. Whitespace characters [optional] :-
It reads the data without storing the consecutive whitespace characters till next whitespace characters.

2. Ordinary characters [optional]:-
They are included to take input in certain pattern.

3. Field width [optional]:-
It specifies the maximum character to be read.

4. conversion character :-
It is used on the basic of data type. eg: %d for integer, %f for floating type, %c for character.

```
// Program to show the usage of whitespace
 character in scanf() function.
void main()
{ int n1;
 char ch;
 printf (" Enter a number ");
 scanf (" %d", &n1);
```

```
printf("Enter a character");
scanf(" %c", &ch); // Note a space before %c.
printf("Number: %d \t character: %c", n1, ch);
}
```

output:-

Enter a number : 10

Enter a character : Y

Number : 10 character : Y

Q. Program to find the area of triangle if measurement of three sides are given:

$$\text{Area} = \sqrt{s(s-a)(s-b)(s-c)}$$

$$s = \frac{a+b+c}{2}$$

[sqrt(a) ⇒ $\sqrt{a}$]

```
include <stdio.h>
void main()
{ int a, b, c, A, s;
 printf("Enter three sides of triangle\n");
 scanf("%d %d %d", &a, &b, &c);
  s= (a+b+c) / 2;
  A= sqrt (s* (s-a) * (s-b) * (s-c));
 printf("The area of triangle is %d", A);
 }
```

Q. Program to show use of ordinary character.

```
// program to show the use of ordinary character
void main()
{
 int day, month , year;
 printf ("Enter your date of birth in sequence");
 scanf ("%d/%d/%d", &day, &month, &year);
 printf ("Your date of birth is : %d day %d month
          %d year", day, month , year);
 }
```

```
// program to read marks of student. The input marks
 should be less than 100. (only two digits)
void main()
{
 int mark;
 printf ("Enter marks (<100):");
 scanf ("%2d", &mark);
 printf ("Your marks is %d", mark);
 }
```

```
// program to input string from user & display it
   on screen
void main ()
{
   char n[50];
   printf ("Enter any string");
   scanf ("%s", n);
   printf ("The string is %s", n);
}
```

→ The scanf() function supports the following specification for strings:
   %[characters]
   %[^characters]

→ The specification %[characters] mean that the characters specified within the brackets are allowed in input string.

→ In specification %[^characters], the characters specified after caret (^) are not allowed in the string.

```
void main()
{
 char n[50];
 printf("Enter any string");
 scanf("%[ABCDEFGH]s", n);
 printf("The string is %s", n);
}
```

{ Output comes only the letter which are in [ ]

```
 scanf("%[^A-Z]s", n);
```

{ output comes only the letter which are not in [^]

To support space:-

```
void main()
{
 char n[50];
 printf["Enter any string"];
 scanf("%(^\n)s", n);
 prinf("The string is %s", n);
}
```

* Formatted output:-

Formatted function are used to display or store in a particular specified format.

printf() is the example of formatted output function.

The general form of printf is:

printf("control string", arg1, arg2, ... argn);

The control string may consist any of the following:

% [flag] [field width] [.precision] conversion character

// Program to display number in different format.

```
void main()
{
 int n=1234;
 printf("n=%d", n);
 printf("n=%6d", n);
 printf("n=%2d", n);
 printf("n=%-6d", n);   [(-) from left align]
 printf("n=%0d", n);
}
```

```
#include <stdio.h>
 void main ()
{
  float n= 12.3456
  printf ("n = %7.4f \n", n);
  printf ("n = %7.2f \n", n);
  printf ("n = %-7.2f \n", n);
  printf ("n = %.6.4f \n", n);
```

1	2	.	3	4	5	6
		1	2	.	3	4
1	2	.	3	4		
1	2	.	3	4	5	6

Q. WAP to display the following pattern.

```
#include <stdio.h>                    NEPAL
void main()                           NEPA
{                                     NEP
  int char n[20] = "NEPAL";           NE
  printf ("%s\n", n);                 N
  printf ("%5.4s\n", n);
  printf ("%5.3s\n", n);
  printf ("%5.2s\n", n);
  printf ("%5.1s\n", n);
}
```

2. Unformatted I/O:-

Unformatted I/O functions do not allow to read or display data in desired format.

1. getchar() & putchar()

syntax: character_variable = getchar();

syntax: putchar (character_variable)

// Program to read character from character & display it on string.

```
void main()
{
 char gender;
 printf("Enter gender M or F");
 gender = getchar();
 printf("Your gender is: ");
 putchar(gender);
}
```

11. getch(), getche() and putch()

syntax: character_variable = getch();

syntax: character_variable = getche();

syntax: putch(character_variable);

// program to read two character: one using getch() another using getche() and display the character using putch().

```
void main()
{
 char c1, c2;
 printf("Enter 1st character");
 c1 = getch();
 printf("Enter second character");
 c2 = getche();
```

```
printf("\n 1st character:");
putch(c1);
printf("\n 2nd character:");
putch(c2);
```

iii. gets() and puts()

syntax:

```
gets(string_variable);
puts(string_variable);
```

// program to input string along with space, display it.

```
void main()
{
char str[30];
printf("Enter any string");
gets(str);
printf("The entered string is:");
puts(str);
}
```

* Operators :-

An operator is a symbol or sign that operates on single or multiple data items and tells computer to perform certain operations on one or more data items (operands)

For e.g. In expression x+y, -x, where x and y are operands and + and - operators.

* Operator Classification :-

i. According to number of operands :-
- Unary
- Binary
- Ternary

ii. According to utility and action :-
- Relational Operators
- Arithmetic Operators
- Logical Operators
- Assignment Operators
- Increment & Decrement Operators
- Conditional operators (Ternary operators)
- Bitwise operators
- Special operators (comma operator & size of operator)

1. Arithmetic operators :-

Q. Program to convert seconds in hour, minute, secs.

```
#include <stdio.h>
void main()
{
  int sec, min, hr, rsec;
  printf("Enter time in second");
  scanf("%d", &sec);
  hr= sec/3600;
  rsec = sec %. 3600;
  min = rsec/60;
  sec= rsec %. 60;
printf("Time in hour, minute, secs is %d %d %d",
                hr, min, sec);
}
```

time in sec - 8563

$hr = \frac{8563}{3600} = 2$

rsec = 8563 %. 3600
= 1363

Q. Program to display a 3 digit number in reverse order.

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

```
printf ("Enter any three digit number");
scanf ("%d", &num);
num=n;      n=num;
d1= n%10;
n=n/10;
d2= n%10;
n= n/10;
printf ("%d %d %d", d1, d2, n);
}
```

(10) Relational operators :-

- Relational operators are used to compare two data items.
- There are four relational operators;

True = 1 ← (2<=10)

False = 0 ← (3>=10)

III. Logical operators:-

			And	OR
0	0	→	0	0
0	1	→	0	1
1	0	→	0	1
1	1	→	1	1

→ Logical operators are the combination of relational operators which represent conditions that are either true or false.

IV. Assignment operators:-

Assignment operators assigns the values of an expression to a variable.

P = P+1 → P+=1 P = P%5 → P%=5

P = P-1 → P-=1

P = P*(q+5) → P*=(q+5)

P = P/(q+5) → P/=q+5

V. Increment & Decrement:-

```
void main()
{ int y;
  int x=10
  y=++x;
  printf("%d", x);
  printf("%d", y);
}
```

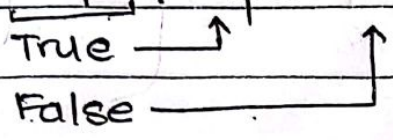
++x
(x=x+1)
= 10+1
= 11
y = 11

y=x++
10
x = x+1 = 11

VI. Conditional operators :-

It is aleo known as ternary operator. The conditional

form: exp1 ? exp2 : exp3



[n1>n2 : n1 : n2]

```
void main()
{int n1, n2, n3, n4, larg1, larg2, largest;
 printf ("\n Enter 4 numbers: ");
 scanf ("%d %d %d %d", &n1, &n2, &n3, n4 );
 larg1 = n1>n2 ? n1 : n2;
 Larg2 = larg1>n3 ? larg1 : n3;
 largest= larg2>n4 ? larg2 : n4;
 printf("\n largest among %d %d %d & %d is %d",
        n1, n2, n3, n4, largest);
}
```

```
ints int x=50 , y=10
   s=x>y ? x++ : y--
           ++x : --y
   printf ("%d", s);
```

VII. Bitwise Operators :-

1 1 0 1 0	0 0 1 0 1
& 1 0 1 0 0	| 1 0 1 1 0 1
1 0 0 0 0	0 1 1 0 1

Precedence & Associativity :-

i = 2*3/ 4+4/ 4+8-2+5/8

= 6/4+4/4+8-2+5/8

= 1+4/4+8-2+5/8

= 1+1+8-2+5/8

= 1+1+8-2+0

= 2+8-2+0

= 10-2+0

= 8+0

= 8 //

⇒ c+=(a>0 && a<=10)?++a:a/b

= (1>0 && 1<=10)

1. If Statement:

Syntax:

```
if (text_expression)
{
  statements if true...
}
```

*

```
// Program to check whether a entered number
is negative
void main()
{
  int num;
print f ("Enter a number");
scanf ("%d", & num);
if (num<0)
{
  printf ("The number is negative");
}
 else
{ print f ("The number is positive");
}
}
```

2. if...else statement:

syntax:

```
if (condition)
{
  Statement if true.
}
else
{ Statement if false
}
```

* // Program to determine whether the entered number is odd or even

```
void main ()
{
 int num;
 printf ("Enter a number");
 scanf ("%d", & num);
 if (num%.2 ==0)
{
 print f (" The number is even");
}
 else
{
 print f ("The number is odd ");
}
}
```

3. Nested if statement

Syntax:

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

* // Program to find smallest among the three numbers using nested if else statement.

```
void main()
{
 int n1, n2, n3;
 printf ("enter 3 numbers");
 scanf ("%d %d %d", &n1, &n2, &n3);
 if (n1<n2)
{
  if (n1<n3)
{
 printf ("%d is the smallest number", n1);
} else
```

4. if-else-if ladder:

syntax:

```
if (condition 1)
stat-A;
else if (condition2)
stat-B;
else if (condition 3)
stat-C;
...
else
next statement;
```

* // Program to determine the division of a student from entered percentage

```
void main()
{
  int p;
printf ("Enter the percentage");
scanf ("%d", &p);
if (p>80)
{
  printf ("Distinction");
}
else
  if (p>60 && p<80)
```

```
{printf(" first division");
}else
  if (p>50 && p<60)
{
  printf("second division");
}
  else if (p>40 && p<50)
{
  printf(" Third division");
  }
  }
```

```
if (n1>n2 && n1>n3)
{
  printf(" %d is largest", n1);
}
else if (n2>n1 && n2>n3)
  {
    printf (" %d is largest", n2);
  }
else if (n3>n1 && n3>n2)
  {
    printf ("%d is largest", n3);
}
```

* //Program to read marks of four subject of a student

```
void main()
{
  float a, b, c, d, total, per;
  printf ("enter the marks of four subject");
  scanf ("%f %f %f %f", &a, &b, &c, &d);
  total = a+b+c+d;
  per = total / 4;
  if (per >=80)
{
  printf (" %d is grade A ");
}
  else if (per >= )
{
  printf (" %d is grade B ");
}
else if (per >= )
{
  printf (" %d is grade C");
```

5. Switch expression :

syntax:

```
switch (expression)
{
```

```
case constant 1:
statement 1 (s)
break;
case constant 2:
statement 2(s)
break;
...
case constant N:
statement N(s)
break;
default:
default_statements(s)
}
```

* Write a menu driven program to perform following operation on two numbers.

1. Addition
2. Subtraction
3. Multiplication
4. Division

```
void main()
{
int n, a, b;
printf("1. Addition \n 2. Substraction \n 3. Multiplicati
\n 4. Division");
```

```
printf ("Enter two numbers:");
scanf ("%d %d", &a, &b);
printf ("Choose any one number from 1 to 4");
scanf ("%d", &n);
switch (n)
{
 case 1:
printf (" Sum of two numbers is %d", a, b, a+b
 break;
 case 2:
printf (" sub of two numbers is %d", a, b, a-b);
 break;
 case 3:
printf (" Mul of two numbers is %d", a, b, a*b)
 break;
 case 4
 printf ("Div of two numbers is %d", a, b, a/b)
 break;
 default:
 printf (" Invalid choice");
```

* **Looping :- (Repititive Statement)**

There are three types of loops statements in C:

→ The for loop
→ The while loop
→ The do while loop

Syntax (for loop):-

```
for ( [initialisation]; [condition]; [increment/decrement]
{
  [statement body];
}
```

* // Program to print odd number from 1 to 50.

```
void main()
{
  for (i=1; i<50; i+=2)
  {
    printf ("%d", i);
  }
}
```

* // Program to print number that are exactly divisible by 7 from 1 to 100.

```
void main()
{
  for i=7; i<100; i+=7)
```

```
{ printf ("%d", i);
  }
  }
```

* // Program to print number that is exactly divisible by 3 & 7 from 1 to 500

```
void main()
{
  for (i=1; i<500 ; i++)
{
  if ( i%3 == 0 && i%7 == 0)
  {
    printf ("%d", i);
  }
}
}
```

* // Program to find the factorial of a positive number entered by a user

```
void main()
{
  int num, i;
  long fact = 1;
```

```
printf ("\n Enter a number whose factorial is to
        be calaulated);
scanf ("%d",&num);
for (i=1; i<=num; i++)
fact= fact *1
print f ("\n the factorial is: %ld", fact);
}
```

* //Program to find sum of numbers from 1 to 20.

```
void main()
{
 int num,i;
  sum=1;
 for (i=1; i<=20; i++)
 sum = sum+1;
 printf ("\n sum of numbers is %d", sum);
}
```

* The while loop:-

syntax: while (condition)

```
{
 statement body;
}
```

⇒ for (i=1; i<10; i++)
{ print ("HWIC")
}

```
i=1;
while (i<10)
{
  printf ("HWIC");
  i++;
}
```

* // Program to print the multiples of 5 from 1 to 1

```
void main()
  i=5;
 while (i<=100)
 {
   print ("%d", i);
   i= i+5;
 }
```

* The do-while loop:-

syntax:
```
do
  {
    Statement body;
  } while (condition);
```

```
i=1;
do
{ printf ("HWIC");
  i++;
}
```

```
while (i<10);
```

* **Jumping statements:-**

The C language has four statements such as break, continue, return and goto used to perform an unconditional branch are called Jumping statements.

1. **The break statement**

The break statement terminates the execution of the loop & the control is transferred to the statement immediately following the loop.

2. **The continue statement**

The continue statement terminates the current iteration of a while, for or do/while statement & resumes execution back at the beginning of the loop body with the next iteration.

3. **The return statement**

The return statement is used to return from a function.

4. The go to statement

The go to statement is used to after the normal sequence of program execution by unconditionally transferring control to some other part of the program.

* Write a program to reverse a given number.
* Write a program to check whether a number is a palindrome.

```
// Program to reverse a given number
#include <stdio.h>
void main()
{
  long int num, rev=0;
  int digit;
  printf ("\n Enter number to be reversed : ");
  scanf (" %ld ", &num);
  while (num!=0)
  {
   digit = num % 10 ;
   rev = rev * 10 + digit;
   num = num / 10;
  }
printf (" The reversed number is : %ld", rev)
}
```

```
// Program to check whether a number is palindrome
// or not
#include <stdio.h>
void main()
{
 int num, rev=0, digit, temp;
 printf ("Enter the number to be checked");
 scanf ("%d", &num);
 temp = num;
 while (num!=0)
 {
    digit = num % 10;
    rev = rev * 10 + digit;
    num = num / 10;
 }
if (temp == rev)
{
   printf ("The number is palindrome");
 }
 elee
{
 printf ("The number is not palindrome");
}
```

Date
Page

* // Program to input an integer number & check
// whether it is prime number

```
void main()
{
 int n, i;
 printf ("enter a number");
 scanf ("%d", &n);
 for (i=2 ; i<n ; i++)
 {
   if (n%i=0)
 {
   printf (" %d is not prime", n);
 break;
 }
 }
  if (i==n)
 print f (" %d is prime", n);
 }
```

* // Program to check whether a number is
// armstrong or not

```
void main ()
{
  int num, digit, sum=0;
  int temp;
```

```
printf ("Enter number to be checked");
scanf ("%d", &num);
temp = num;
while (num!=0)
{
 digit = num % 10;
 sum = sum + digit * digit * digit;
 num = num / 10;
}
 if (temp == sum)
 printf (" Armstrong number ");
 elee,
 printf (" Not Armstrong number");
}
```

* // Program to find sum of digits of a number

```
#include <stdio.h>
void main()
{
  int n, sum=0, rem;
  printf ("Enter the number");
  scanf ("%d", &n);
  while (n>0)
  {
   rem = n%10;
   sum= sum + rem;
   n = n/10;
  }
  printf ("Sum of digits is %d", sum);
}
```

* WAP to display the following pattern.

```
1
1  1
1  1  1
1  1  1  1
1  1  1  1  1
```

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

*

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

```
void main()
{
  int i,j;
  for (i=1; i>=5; i++)
{
```

```
for( j=1; j<i ; j++)
{
  printf (" %d", j);
{
  print f (" %d", j);
}
  print f (" /n");
}
```

* 1 2 3 4 5
1 2 3 4
1 2 3
1 2
1

```
void main()
{
 Int i, j;
 for (i=5; i>=1; i--)
{
 for ( j= 1; j<= i ; j++)
{
  print f ("%d", j);
}
  print f (" /n");
}
```

* // Program to determine the sum of the harmonic series (1+2/2 + 1/3 + 1/4 + 1/n) for a given value of n

```
void main()
{
  int n, i;
  float sum=0;
  printf ("enter n");
  scanf ("%d", &n);
  for (i=1; i<=n; i=i+1)
  sum = sum + 1.0 / i;
  printf ("sum is %f", sum);
}
```

* // Program to print fibonacci series up to n terms

```
void main()
{
  int a, b, c, i, terms;
  /* input number from user*/
  printf ("Enter number of terms");
  scanf ("%d", & terms);
  /* Fibonacci magic initialization*/
  a=0;
  b=1;
  c=0;
```

```
printf ("Fibonacci term\n");
/* Iterate through n terms */
for (i=1; i<=terms; i++)
{
  printf ("%d", c);
  a=b;
  b=c;
  c= a+b;
  // copy n-1 to n-2
  // copy current to n-1
  // new term
}
}
```

ARRAY

→ An array is a collection of elements of same data types placed in contiguos memery location and can be accesed individually using an index to a unique given name.

→ Types of Array :-

1. one dimensional (1D) array
2. Multi dimensional array

* Declaration of one-dimensional array

The general form of 1D array :-

syntax : storage-class data-types array-name [size];

eg: int a [10];

float marks [50];

char string [100];

Initializing 1D array :-

syntax:

data-type array-name [size] = [value1, value2, value ....]

eg: int a [6] = [21, 31, 87, 94, 78, 74]
float marks [3] = [21.7, 54.97, 84.3]
char colour [3] = ['R', 'E', 'D'];

printf ("%d", a[0] + a[3]);

* Reading the elements of Array

```
int i;
int mark [5];
for (i=0; i<5; i++)
{
 printf (" enter marks");
 scanf ("%d", &mark (i));
}
 printf ("The entered number are ");
 for (i=0; i<5; i++)
{
 printf ("%d", mark [i]);
}
```

Date
Page

* // Program to accept marks for 5 subjects and find
//the sum & average.

```
void main()
{
int i;
int mark[5], sum=0, ave;
for (i=0; i<5; i++)
{
printf ("enter marks");
scanf ("%d", &mark[i]);
}
for (i=0; i<5; i++)
{
sum= sum + marks[i];
}
printf ("The sum is %d", sum);
ave = sum /5;
print f ("The average is %d", ave);
}
```

* // Program to find the largest element of an array
//with 10 elements

```
void main()
```

```
{
 int i;
 int element [10];
 for (i=0; i<10; i++)
{
 print f ("enter elements");
 scan f ("%d", & element [i]);
}

 large = element [0];
 for (i=1 ; i<10 ; i++)
{
 if (large < element [i] )
 large = element [i]
}
```

Imp.

* **Sorting :-**

It is the process of arranging the element in ascending or descending order.

Q: // write a program to sort the one-dimensional order in ascending order

```
void main()
{
  int num [100], i, j, n, terms;
  printf (" Enter the number of elements");
  scanf ("%d", &n);
  for (i=0; i<n; i++)
{
  printf (" Enter element ");
  scanf ("%d", &num [i]);
}
  printf("The elements are ");
  for (i=0; i<n; i++)
{
  printf (" %d", num [i]);
  for (i=0; i<n-1; i++)
{
  for (j=i+1; j<n; j++)
{
```

```
if (num[i]> num[j])
{
  temp=num[i];
  num[i]= num[j];
  num[j]=temp;
 }
 }
 }
 printf(" The elements in ascending order are");
 for (i=0; i<n; i++)
{
 print f (" %d", num");
 }
```

2. Multi-Dimensional Array:

Multi-dimensional array are the arrays with more than one dimension.

Two Dimensional Array

* Declaration of 2D Array

Syntax:

data-types array-name [row-size] [column_size]

* Reading 2D array

eg:

```
int num [3] [4];
for (i=0; i<3 ; i++)
{
  for (j=0; j<4 ; j++)
{
   printf (" enter element");
   scanf ("%d", & num [i] [j]);
  }
  }
```

* Printing 2D Array

```
for (i=0; i<3; i++)
{
for (j=0; j<4; j++)
{
   printf ("%d", num [i] [j] );
}
```

* WAP to read two M*N matrices and display their sum.

```
void main()
{
 int mat1 [M][N], mat2 [M][N], sum [M][N], i, j;
 printf ("Enter the value of M");
 scanf ("%d", &M);
 print f("Enter the value of N");
 scanf ("%d", &N);
 print f ("Enter the elements of first matrix");
 for (i=0; i<M; i++)
{
 for (j=0; j<N; j++)
 {
  print f ("Enter element");
  scanf ("%d", &mat1 [i] [j]);
 }
}
  print f ("The elements of matrix 1");
  for (i=0 ; i<M; i++)
{
  for(j=0 ; j<N ; j++)
{
 print f ("%d ; mat [i] [j]);
}
}
```

```
printf ("Enter the elements of second matrix");
for (i=0; i<M ; i++)
{ for(j=0; j<N; j++)
{ printf ("Enter element");
  scanf ("%d", &mat2 [i][j]); } }
  printf ("The elements of matrix 2");
  for (i=0; i<M; i++)
{ for (j=0; j<N ; j++)
i printf ("%d", mat 2 [i] [j]); } }
  for (i=0; i<M; i++)
{ for (j=0; j<N ; j++)
  sum [i] [j] = mat 1 [i][j] + mat2 [i] [j]; } }
print f (" The sum of matrix is ");
for (i=0; i<M; i++)
  for (j=0; j<N; j++ )
print f ("%d", sum [i] [j];
```

```
* //Program to find transpose of a matrix (2X3)
#include <stdio.h>
void main()
{
  int mat[2][3], i,j;
  printf ("\n Enter the elements of matrix");
  for (i=0; i<2; i++)
{ for (j=0; j<3; j++
{ printf (" Enter element");
  scanf (" %d", &mat1[i][j]); } }
  printf ("The elements of matrix are ");
  for (i=0; i<2; i++)
{ for (j=0; j<3; j++)
{ printf (" %d", mat1[i][j]); } }
  printf (" The transpose of matrix is ");
  for (i=0; i<3; i++)
{
  for (j=0; j<2; j++)
{  printf (" %d \t", mat1[j][i]); }
  printf ("\n");
}
```

* //Program to find sum of diagonal elements of 3x3

```
#include <stdio.h>
void main()
{
int mat[3][3], i, j, m, n, sum=0;
printf("Enter the
```

classmate
Date
Page

* **String:-**

Strings are array of characters.
A string is always terminated by a null character (i.e slash zero \0).

Imp * **String Manipulation Functions:-**

The <string.h> header file defines some string handling functions such as strcpy(), strcat(), strrev(), strcmp(), strlen() etc.

1. **strcpy()** - copying string

It is used to copy one string into another string.

strcpy()

```
#include <stdio.h>
#include <string.h>
void main()
{
  char a[]="Ram";
  char b[50], c[50];
  printf("original string is %s", a);
  strcpy(b, a);
  printf("Copied string is %s", b);
}
```

2. strlen() - It is used to find the length of the string (number of characters)

```
void main()
{
char str[20]; int len;
printf ("Enter any string");
scanf ("%s", str);
len = strlen (str);
printf ("The length of string is %d", len);
}
```

3. strrev() - It is used to reverse a string

```
void main()
{
char str [50];
printf ("Enter string");
gets (str);
strrev (str);
printf ("The reversed string is %s", str);
}
```

4. strcat() - It is used to concatenate two string.

```
void main()
{
char str1[] = "Happy";
char str2 [] = "New Year";
```

```
    printf ("The concatenated string is %s",
                strcat (str1, str2));
    }
```

5. strcmp()- (comparing string)

→ It is used to compare the two strings.

→ The functions returns 0, if first string is alphabetically less than second string

```
void main()
{
  char str1[100], str2[100];
  printf ("Enter first string");
  gets (str1);
  printf ("Enter second string");
  gets (str2);
  if (strcmp (str1, str2)==0)
{
  printf ("The string are identical");
}
else
{ printf ("The string are not identical");
}
}
```

6. strupr() - It converts the string into uppercase.

```
void main()
{
  str[]="apple";
  printf("The string in uppercase is %s", strupr(str));
}
```

7. strlwr() - It converts the string into lowercase.

```
void main()
{
  str[]="APPLE";
  printf("The string in lowercase is %s", strlwr(str));
}
```

Q. WAP to convert the string into lowercase (if the user input string in uppercase & vice-versa)

* Function:-

A function is a self content block of statements that perform a task or job.

* Advantages of Function:-

- Manageability
- Non-redundant (non-repeated) programming
- Code Reusability
- Logical clarity
- Easy to divide the work to many different programmars.

* Types of Function:-

a) Library function
b) User-defined function

* Elements or components of function:-

1. Function prototype or Declaration

syntax:

return_type function_name (type 1, type 2, --- type

eg: int add (int a, int b);

float area (float radius);

2. Function defination:-

- Program statement that describes the specific tasks to be done by the function.

syntax:

data_type function_name (datatypes var1, datatypes var2 ...)

3. Calling of function:-

syntax:

function_name();

Q. WAP to find a sum of two numbers by using function.

```
#include<stdio.h>
int sum(int, int)    // Function declaration
void main()
{
 int a, b, sum;
 printf ("Enter two numbers");
 scanf (" %d %d", &a, &b);
 add = sum(a, b)      // Calling function
 printf ("%d", add);
}
int sum(int a, int b)
```

```
{ int s;                    // Function defination
  s = a+b
  return s;
}
```

Q. WAP to find simple interest by using function.

```
#include <stdio.h>
int SI (int, int, int)
void main()
{
int P, T, R, X;
printf ("Enter principal, time, rate");
scanf ("%d %d %d", &P, &T, &R);
X= SI (P, T, R);
printf ("%d", X);
}
 int SI (int P, int T, int R)
{
  int s;
    s = (P*T*R)/100
  return s;
}
```

* Different forms of function:-

1. Function with no return value and no arguments.

eg:

```
#include<stdio.h>
void sum();
void main()
{
  sum();
}
void sum()
{
  int a,b,s;
  printf("enter two numbers");
  scanf("%d %d", &a&b);
  s=a+b
  printf("sum is %d", s);
}
```

2. Function with arguments but not return type.

eg:

```
void sum(int, int);
int main()
{
  int a,b;
```

```
printf("Enter two numbers");
scanf("%d %d", &a, &b);
sum(a, b);
return 0;
}
void sum(int x, int y)
{
  int s;
  s=x+y;
  printf("The sum is %d", s);
}
```

3. Function with Return value & no arguments.

Eg:

```
int sum();
int main()
{
  int s;
  s=sum();
  printf("sum is %d", s);
  return 0;
}
int sum()
{
```

```
int s,a,b;
printf("Enter two numbers");
scanf("%d %d", &a, &b);
s= a+b;
return s;
}
```

WAP to find addition, subtraction, multiplication & division by using function.

```
#include<stdio.h>
void sum()
void main()
{
sum();
}
void sub();
void main()
{
sub();
};
void mul();
void main()
{
mul();
}
```

```
void div();
void main()
{
  div();
}
void sum()
{
  int a, b, sum;
  printf("Enter two numbers");
  scanf("%d %d", &a, &b);
  sum = a+b;
  printf("%d", sum);
}
void sub()
{
  int a, b, sub;
  printf("enter two numbers");
  scanf("%d %d", &a, &b);
  sub = a-b;
  printf("The subtraction is %d", sub);
}
void mul()
{
int a, b, mul;
```

```
printf("Enter two numbers");
scanf("%d%d", &a, &b);
mul = a*b;
printf("The multiplication is %d", mul);
}
void div()
{
int a, b, div;
printf("Enter two numbers");
scanf("%d%d", &a, &b);
div = a/b;
printf("Division is %d", div);
}
```

* **Recursion & Recursive function :-**

→ Recursion is a technique for defining a problem in terms of one or more smaller versions of same problem.

→ Recursive is a function that calls itself directly or indirectly to solve a smaller version of task until a final call which doesn't require a self call.

```
// Program to find the factorial of a number
using recursion
#include <stdio.h>
int fact (int);
void main()
{
  int a;
  printf ("Enter any number");
  scanf ("%d", &a);
  printf ("The factorial of number is %d", fact(a
}
  int fact (int b)
{
  int (b==0)
  return 1;
  else
  return (b * fact (b-1));
}
```

* Passing Array to function:-

Eg:-

```
// Program to read 10 numbers in an array & find
its sum.
#include <stdio.h>
```

```
int sum (int a[]);
void output (int a[]);
int main()
{
  int a[10], s, i;
  printf ("Enter the elements");
  for (i=0; i<10; i++)
  {
    scanf ("%d", &a[i]);
  }
  output(a);
  s=sum(a);
  printf (" The sum of array elements is %d", s);
  return 0;
}
void output (int a[])
{
  int i;
  for (i=0; i<10; i++)
  {
  printf ("%d", a[i]);
  }
  int sum (int a[])
  {
    int i;
    for (i=0; i<10; i++)
```

```
{
  sum= sum+a[i];
}
return sum;
}
```

* // Program to find smallest element in array using function

classmate
Date
Page

* //Program to add two 3×4 matrices & print the result in matrix form, use seperate functions to take input & to add & display result.

```
#include<stdio.h>
int i,j;
int main()
{
 int a[3][4], b[3][4]
 void input(int a[][4]);
 void add (int a[][4], int b[][4]);
 printf("input first matrix:\n");
 input(a);
 printf("Addition matrix is:\n");
 add(a,b);
 return 0;
}
void input[int a[][4]);
{
 for(i=0; i<3; i++)
{
 for(j=
```

* Function Calling :-

→ The arguments that are passed to the call function are called actual arguments or parameters.

→ The arguments that are received by the called function from the calling function are called formal parameters.

* Types of function calls :

There are two types of function calls :

1. call by value / Passing arguments by value
2. Call by reference / Passing arguments by reference or address.

1. Call by value :-

→ When the values of actual parameters are passed to a function as arguments, it is known as function call by value.

→ In this call, the value of each actual arguments are copied into corresponding formal argument.

→ The content of the arguments in the calling function are not altered, even if they are changed in the called function.

→ Examples of called function :

Output:-

```
#include <stdio.h>
void func(int, int);
void main()
{
int x=10, y=20;
printf("Before calling :- x=%d y=%d", x, y);
func(x, y);
printf("After calling :- x=%d y=%d", x, y);
}
void func(int x, int y)
{
int x=10; y=10;
printf("within function, x=%d y=%d", x, y);
}
```

```
Before calling :- x=10 y=20
within function :- x=20 y=10
After calling :- x=10 y=20
```

2. Call by reference :- (address)

→ In call by address method, the address of the actual argument in calling function are copied to the formal arguments in called function.

→ Since the reference of arguments are used the values of actual arguments change with change in values of corresponding formal arguments.

```
#include <stdio.h>
void func(int, *int);
void main()
{
 int x=10, y=20;
 printf(" Before calling : x= %d  y= %d", x, y);
 func (&x, &y);
 printf("After calling: x= %d  y= %d", x, y);
}
 void func (int*x, int*y)
{
  *x=20;
  *y = 10;
 printf (" within function, x= %d  y= %d", *x, *y);
}
```

* Types of variables / Storage class:-

1. Automatic variables / Storage class:-

→ They are also called local variables.

→ The variables that are declared with in the function & their scope is within the function are called local variables or automatic variables

```
#include <stdio.h>
void func(int, *int);
void main()
{
 int x=10, y=20;
 printf (" Before calling : x=%d y=%d", x, y);
 func (&x, &y);
 printf ("After calling: x=%d y=%d", x, y);
}
void func (int*x, int*y)
{
 *x=20;
 *y = 10;
 printf (" within function, x= %d y=%d", *x, *y);
}
```

* Types of variables / storage class:-

1. Automatic variables / storage class:-

→ They are also called local variables.

→ The variables that are declared with in the function & their scope is within the function are called local variables or automatic variables

```
void fun();
void main()
{
  func();
}
void func()
{
  int a,b;
  a=10, b=20;
  printf("a=%d, b=%d", a,b);
}
```

Here, a & b are local variables.

2. Global variables:

→ The variables that are declared outside any function are called global variables.

→ All functions in the program can access & modify the global variables.

```
#include <stdio.h>
int a=4, b=5, c=6;
int sum();
int prod();
int main()
{
```

```
printf ("sum %d", sum());
printf ("Product is %d", prod());
return 0;
}
int sum()
{
return (a+b+c);
}
int prod()
{
return (a*b*c);
}
```

3. Static variables:-

→ A static variable is a local variable that retains the latest value when function is called.

```
void test()
{
static int i=10;
printf ("%d", i);
i++;
}
void main()
{
test();
test();
```

4. Register variable:-

→ The variable that are stored in the processor register are called register variables.

→ They can be accessed & manipulated faster than other variables.

```
int main
{
register int x=10;
--------
--------
```

5. Pointer variable:-

→ Pointer is a variable that holds / stores the address rather than value.

→ It store the address of another variable in a memory.

* Declaration & Initialization of pointer variable:-

To declare a pointer variable we must use following syntax:

Data types * pointer_name;

eg: int * ptr;

Valid example:-	Invalid example:-
int *p;	int *p;
int num;	float num;
p = #	p = #

* //Program to illustrate the use of pointer

```
#include <stdio.h>
void main()
{
 int x=10, *p;
 p=&x;
 printf("%d", &x); ← // 10173
 printf("%d", x);      //10
 printf("%d", p);      // 10173
 printf("%d", *p);     //10
 printf("%d", x);      //10
```

x p(pointer variable)

10 | 10173

10173 address

[P(address) ≠ address(value)]

= Passing pointer to function

declare int func (int *, int*)

call func (&a, &b)

define int func (int *x, int *y)

{

}

* Pointer and Array :-

1. 1D Array :-

• In a 1D array the address at first element can be expressed as & a [0] or simply a.

- Similarly, address of second element can be written as a[1] or, a+1;

a[0]	a[1]	a[2]	a[3]	a[4]
2	3	4	5	6
1013	1016	1017	1018	1019

printf("%d", 0) ⇒ 1015

- Therefore, address of ith element can be written as &a[i], simply, a+i.

```
void main()
{
 int a[] = {10, 20, 30, 40, 50};
 int i, int * ptr;
 ptr = &a[0];
 for (i=0; i<5; i++)
{
printf("%d %d %d %d %d", a[i], &a[i], ptr+i, (a+i),
      *(a+i));
```

i=0

a[0] = 10

&a[0] = 1015

ptr 0 = 1015

a+i = a+0 = a = &a[0] = 1015

*(a+i) = *(a+0) = 10

classmate
Date
Page

* // Program to find average of 5 elements in an array by using pointer

```
#include <stdio.h>
void main()
{
  int a[5];
  int *ptr;
  ptr=a[0];
  printf("enter 5 elements");
  for(i=0;i<5;i++)
  {
  scanf("%d",ptr+i); → // &a[0]=ptr
  }                    &a[1]=ptr+1
  sum=sum+*(ptr+i)     &a[2]=ptr+2
                       &a[i]=ptr+i
                  // a[i]=(*(ptr+i)) value
                  // a[i]=ptr+i [address]

#include <stdio.h>
int main()
{
 int i,sum,*ptr;
 int a[5];
 float average;
 ptr=a[0];
 for(i=0;i<5;i++)
```

```
{ printf ("Enter 5 elements %d", i+1);
  scanf ("%d", ptr+1);
}
  for (i=0; i<=5 ; i++)
  sum = sum + *ptr+i);
  average = sum/5;
  printf ("Average : %.2f", average);
 }
```

Pointer in Multi-Dimensional Array:-

The declaration:-

int a[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}}

is manupulated by the compiler in form:

Data types (*ptr_variable)[size 2];

So, that the 2D array becomes

int (*a)[4] = {{1,2,3,4},{5,6,7,8},{9,10,11,12}}

In general,

Address of ith row & jth column

⇒ &a[i][j] = *(a+i)+j

⇒ a[i][j] = *(*(a+i)+j)

// Program to find sum of two 2x3 matrix by using pointer

→ A pointer can be assigned the address of ordinary variable.

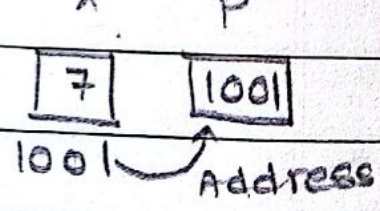
→ Integer data can be added to or subtracted from pointer variables. Example: p1+1

```
int a,b;
float c;
int *p1 *p2;
float *f;
```

→ The content of one pointer can be assigned to another pointer. Example; p1=p2

* //Program to illustrate the pointer arithmetic

```
void main()
{
 int*p, x;
 p=&x;
 printf ("Enter the value of x");
 scanf ("%d", p);
 printf ("%d", *p); //7
 printf ("%d", p); //1001
 printf ("%d", p+1); //1003
 printf ("%d", p-2); //997
```



a[0]	a[1]	a[2]	a[3]	a[4]
1	2	3	4	5
1001	1003	1005	1007	1009

```
void main()
{
int a[] = {1,2,3,4,5};
int *f1, *f2;
f1 = a;                    // f1 = a[0]
f2 = a+3;                  // f1 1001  f2 1007
printf("%d", f2-f1);
printf("%d %d", *f1, *f2);
}
```

* Dynamic Memory Allocation:-

The process of allocating & freeing memory at run time is known as dynamic memory allocation. There are four built in functions for allocating memory dynamically. They are:-

1. Malloc()
2. Calloc()
3. Free()
4. Realloc()

1. Malloc() function:-

It allocates the requested size of bytes & returns pointer to the first byte of an allocated space.

syntax: p=(data-types) malloc (size_of_block)

Here, p is a pointer of type data-type

Eg : x= (int *) malloc (200);

This statement allocates a memory equivalent to 200 bytes.

```
#include <stdio.h>
#include <stdlib.h>
void main()
{
  int i,n;
  printf ("Enter the number of integers");
  scanf ("%d", &n);
  ptr= (int *) malloc (n* sizeof (int));
  for (i=0 ; i<n ;i++)
  {
    print f ("Enter integer");
    scanf ("%d", (ptr+i));
  }
  for (i=0; i<n; i++)
  {
  print f ("%d", *(ptr+i));
  }
}
```

2. Free() function:-

The free() function dynamical deallocates the memory which is allocated previously.

syntax:

```
free(pointer_variable);
```

3. Calloc() Function:-

The calloc function is used to allocate an amount of memory equal to num * size.

It is used to allocate multiple block of memory.

syntax:

```
p=(data_types*) calloc (no_of_blocks, size_of_each_
                                        block);
```

Eg: x=(int*) calloc (5,100 * size of (int));

4. Realloc() function:-

The function is used to modify the size of previously allocated space.

syntax:

```
ptr=realloc (ptr, newsize);
int main()
{
  int;
  int *ptr = (int*) malloc 2 * size of (int));
```

```
for (i=0; i<2; i++)
{
  scanf ("%d", ptr+i);
}
ptr = (int *) realloc (ptr, 4 * sizeof (int));
printf ("enter two more integers");
for (i=0; i<2; i++)
{
  scanf ("%d", ptr+i);
}
  for (i=0; i<4; i++)
{
  printf ("%d", *(ptr+i));
}
  return 0;
}
```

* Structure:-

A structure is a collection of variables referenced under one name, providing convenience to keep related information together.

The general form of structure is;

```
struct structure_name
{
  Type member_name;
  Type member_name;
  -----
};
```

Example:-

```
struct student
{
  int roll;
  char gender;
  char name[50];
  float percent;
};
```

Structure Variables:-

A structure variables is variable of a structure type

classmate
Date
Page

Accessing a structure member :-

A structure member can be accessed by writing :

structure_variable member_name;

Example :

st.id=120;

* // Program that define a structure to store the record of an employee with id, name, sex and salary fields. R
Record of an employee & display the record

```
#include <stdio.h>
struct employee
{ int id;
  char name[50];
  char sex
  float salary;
};
void main()
{ struct employee emp;
  printf("Enter id \n");
  scanf("%d", &emp.id)
  printf("Enter Name \n");
  scanf("%s", emp.name);
  print f("Enter sex \n");
  scanf("%s", emp.sex);
```

```
Struct employee
{ int id;
  char name[50]
  char sex;
  float salary;
} struct employee emp[10];
```

```
printf ("enter salary");
scanf (" %f", & emp.salary );
printf ("The record of employee is: \n");
printf (" Id: %d", emp id);
printf (" Name is : %s", emp. name);
printf ("sex is: %c", emp sex);
printf ("salary is: %f", emp salary);
}
```

Array of structure:-

syntax:

* //create a structure named student// that has name, roll, percentage as its members. Use this structure to read & display records of 10 students.

```
#include <stdio.h>
//struct student
{ int roll;
  char name [50];
  float percent;
};
void main()
{ struct student. st [10];
  int i;
for. (i=0; i<10; i++)
{ printf ("enter the information student %d", i+1);
```

```
scanf("%d %s %f", &st[i].roll, st[i].name, &st[i].percent
}
printf("The information of students are :\n");
for(i=0; i<10; i++)
{ printf("Roll: %d \t Name: %s \t Percent: %f \n", st[
    roll, st[i].name, st[i].percent); } }
```

Nested Structure :- (Structure within another structure)

```
struct personal_record
{
char name[50];
int day;
int month;
int year;
float salary;
```

In the above structure, we can group all the items related to date of birth into one structure.

```
struct personal_record
{
char name[50];
struct dob
{
  int day;
  int month;
  int year;
} birthday;
```

```
float salary;
} person;

* // Program to demonstrate the use of nested structure
#include <stdio.h>
struct student
{
 char name[50];
 float salary;
 struct dob
 {
  int day;
  int month;
  int year;
 } birthday;
} st;
void main()
{
 printf ("Enter Name and salary");
 scanf ("%s %f", st.name, &st.salary);
 printf ("Enter date of birth");
 scanf ("%d%d%d", &st.bd.day, &st.bd.month, &
        st.bd.year);
 printf ("The details of student are:");
 printf ("%s %f %d %d %d", st.name, st.salary, st.bd.day
        st.bd.month, st.bd.year);
```

Page:

Passing structure to the function:

```
// Program to demonstrate passing structure to
function
#include <stdio.h>
void display(struct employee e);
 struct employee
{
 char name[50];
 float salary;
};
void main()
{
struct employee emp;
printf("Enter Name & salary");
scanf("%s %f", emp.name, &emp.salary);
display(emp);
}
void display(struct employee e)
{
printf("Name: %s\nSalary: %.2f", e.name, e.salary);
}
```