Saathi

UNIT-1. Date 22 05 23

① # Introduction to computing:-

•) Computer System:-

Hardware (Phy. equipment)

Software: collection of prog. (inst.) to allow the hardware to do its job.

5 parts:-
input devices
CPU
Primary storage
output devices
Auxiliary storage devices

•) Computer Software:-

System software
- Operating System
- System support
- System Development

Application software.
- General purpose
- Application Specific

•) Computer languages:-

1940's → Machine level languages → only lang. understood by computer hardware

1950's → Symbolic languages → uses symbols/mnemonics to represent various machine language instr. + Assembler is used.

1960's → high-level languages → used for System software & new applic. code.
eg. C

•) Algorithm:-

→ Step-by-step approach to solve a given problem.
→ represented in English like lang. & has some math. symbols like >, <, =, etc.
→ planned strategy (to write) to find a solution.

Ex.- Algorithm / pseudo code to add 2 numbers.

Step1:- Start
Step2:- Read the two numbers in to a, b
Step3:- c = a+b
Step4:- Write / print c
Step 5:- Stop.

•) Flow chart:-

→ Graphical representation of an Algorithm / or portion of an algorithm.
→ Drawn using certain Sp. purpose Symbols
→ Diagrammatic representation of way to solve the given prob

Page No.

Saathi

Date __/__/____

⬭	Oval	Terminal	Start / Stop / begin / end
▱	parallelogram	input/output	making data available for processing (input) or recording of process infor. (output)
▭	Rectangle	Process	Any processing to be done. A process changes or moves data. An assignment operation.
◇	Diamond	Decision	Decision / switching type of operations
○	circle	Connector	used to connect diff. parts of flowchart
→	Arrow	flow	joins 2 symbols & also represents flow of execution.

•) C-Language :- Dennis Ritchie for use with UNX on DEC PDP-11 computers.

Basic Structure of C-language

1. Documentation Section → /* and */ whatever written btw these 2 are called comments.
2. Linking Section → #include<stdio.h> tells compiler to link certain occurance of keywords or func°. in your prog. to header files specified in this sec°.
3. Definition Section → #define MAX 25 used to declare some const. & assign them some value.

#define is a compiler directive which tells the compiler whenever MAX is found in the program replace it with 25.

4. Global declaration Section : int i; (before main())
variable used throughout program are declared so as to make them accessible to all parts of program.
5. Main function Section :→ ① Declaration section → variables & their data type are declared
② Execution section → actually performs the task we need

```
main()
{
  point from where execution starts
}
```

6. Sub program or function Section : all functions which ours program needs.

Simple 'C' program

```
/* Simple program in C */
#include<stdio.h>
main()
{
  printf("welcome");
}
/* End of main */
```

Page No.

Saathi

Date ___/___/___

•) C - tokens: Tokens are individual words & punctuations marks in Eng. lang. Sentence. Smallest individual unit → C tokens

- C tokens
  - Operators — eg. +, -, *
  - Special Symbols — [ ], { }
  - Strings — "asifia"
  - Constants — 15.4, 'a', 200
  - Identifiers — rate, no_of_hours
  - Keywords — int, printf.

•) Keywords - reserved words
predefined meaning in 'C' lang.

Auto	Break	Case	Char	Const	Continue
Default	Do	Double	Else	Enum	Extern
Float	For	Goto	if	int	Long
Register	Return	Short	Signed	Sizeof	Static
Struct	Switch	Typedef	Union	unsigned	Void
volatile	while				

•) Identifiers: Name of the variables & other program elements such as functions, array, etc.

Few rules:-
→ Identifiers can be named from combination of A-Z, a-z, 0-9, _
→ 1st alphabet should be either alphabet or underscore, digit are not allowed.
→ Should not be a keyword eg name, ptr, sum

•) Constants - A quantity that doesn't vary during the execution of a prog.
↳ Numeric constant & Character constants.

Escape sequence	description
\a	Alert
\b	backspace
\f	form feed
\n	New line
\r	carriage return
\t	horizontal Tab
\v	vertical Tab

•) Variables: A quantity that can vary during execution of a program

•) Data type:- essential to identify the storage representation and the type of operations that can be performed on that data.

Page No.

Saathi

Date ___/___/___

Storage size & range of basic data types:-

Type	Length	Range
unsigned char	1byte = 8 bits	0 to 255
char	1byte = 8 bits	-128 to 127
short int	2 bytes = 16 bits	-32768 to 32767
unsigned int	4 bytes = 32 bits	

char	**1 byte = 8 bits**
Signed char	1 byte
Unsigned char	1 byte
Short	**2 bytes**
Signed short	2 bytes
unsigned short	2 bytes
int	**2 bytes**
Signed int	2 bytes
Unsigned int	2 bytes
Short int	2 bytes
Signed short int	2 bytes
Unsigned short int	2 bytes
long int	**4 bytes**
Signed long int	4 bytes
Unsigned long int	4 bytes
float	**4 bytes**
double	**8 bytes**
long double	**10 bytes**

① **Basic data type** :-
All arithmetic operations
eg. int a, b;
char c;

② **Derived data type** :-
Store a set of data values
eg. Arrays & Structures.
int a[10];
char name[20];

③ **user defined data type** :-
ex- Declaration,
typedef int integer;
ex- integer num1, num2;

④ **Pointer data type**
Pointer data type is necessary to store the address of a variable.

•) **Operators** :- A symbol that tells the computer to perform certain mathematical or logical manipulation. They form expressions.

i) **Arithmetic operator**:- real operands (integer arithmetic).
Mixed mode arithmetic - combination of arithmetic & real.

+	add	/	division
-	subtract		
*	multiplication	%	modulo division (remainder)

ii) **Relational Operator** :- Compare 2 quantities.

< is less than	>= is greater than or equal to
> is greater than	== is equal to
<= is less than or equal to	!= is not equal to.

Saathi

Date ___/___/___

iii) Logical operators:

&& Logical And Operator: if both the operands are non-zero, then the condition is true. (A && B) is false.

|| Logical OR Operator: if any of the 2 operands is non-zero, then condition is true. (A || B) is true.

! Logical NOT Operator: used to reverse the logical state of its operand. If condition is true, then Logical NOT operator will make it false. ! (A && B) is true.

XOR → ek value same or ek value short, none change.

iv) Bitwise operator :-

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

Other bitwise operators:- (for manipulation of data at bit level) (not applied to float or double)

~ → binary one's complement operator

<< → binary left shift operator

>> → binary right shift operator

v) Assignment operators :-

i) = simple assignment operator → assign value from right side operands to left side operand. int a = 2

ii) += Add And assignment operator: int a += 1 => a = a + 1

iii) -= Subtract And " " :- int a -= 1 => a = a - 1

vi) Miscellaneous operators

sizeof( ) → returns size of variable

& → returns address of variable

* → pointer to a variable

?: → conditional expression if cond" is true ? then value X : otherwise value Y.

Date ___/___/___

Saathi

vii) Increament and Decreament operators :-

++ and -- are increment & decrement operators used to add or subtract. Both are unary

Eg:- if m=5; y=++m then, it is equal to m=5; m++; y=m

if m=5; y=m++ then, it's eq. to m=5; y=m; m++

```
/* programs to exhibit use of operators */
#include <stdio.h>
int main()
{
  int sum, mul, modu;
  float sub, divi;
  int i, j;
  float d, m;
  printf("Enter 2 integers");
  scanf("%d %d", &i, &j);
  printf("Enter 2 real numbers");
  scanf("%f %f", &d, &m);
  sum = i+j;
  mul = i*j;
  modu = i%j;
  sub = d-m;
  divi = d/m;
  printf("sum is %d", sum);
  printf("mul is %d", mul);
  printf("Remainder is %d", modu);
  printf("Subtraction of float is %f", sub);
  printf("division of float is %f", divi);
  return 0;
}

/* program to implement relational & logical */
#include<stdio.h>
main(){
  int i,j,k;
  printf("Enter any 3 no.");
  scanf("%d %d %d", &i, &j, &k);
  if ((i<j) && (j<k))
    printf("k is largest");
  else if i<j || j>k
  { printf("j is largest");
  else
    printf("j is not largest of all");
  }
  return 0;
}
```

Saathi

Date ___/___/___

```
/* program to implement increment and decrement operators */
#include <stdio.h>
main()
{
  int i;
  printf("enter a number");
  scanf("%d", &i);
  i++;
  printf("after incrementing %d", i);
  i--;
  printf("after decrement %d", i);
}
```

Output

```
enter a number: 9
after incrementing 10:
after decrement 9:
```

```
/* program using ternary operator and assignment */
#include <stdio.h>
main()
{
  int i, j, large;
  printf("Enter 2 numbers");
  scanf("%d %d", &i, &j);
  large = (i>j)? i:j;
  printf("largest of two is %d", large);
  return 0;
}
```

Output

```
enter 2 numbers
8
3
largest of two is 8
```

Date ___/___/___

Saathi

2) **Arrays:** Group of related data items that share a common name.
Ex- Students.
The complete set of students are represented using an array name students. A particular value is indicated by writing a no. called index no. or subscript in brackets after array name.
Complete set of value is referred to as an array, the individual values are called elements.

1-D Arrays:- A list of items can be given one variable index is called single subscripted variable or a 1-D array.
The subscript value starts from 0. If we want 5 elements the declaration will be
int number [5];
elements will be number[0], number[1], number[2], number[3], number[4]
There will not be number[5]
Declaration of 1-D arrays:- Type variable-name [Sizes];
Type - data type of all elements. Ex:- int, float etc.
Variable-name - is an identifier
Size - is the max. no. of elements that can be stored.
Ex:- float avg[50]
This array is of type float. Its name is avg. and it can contains 50 elements only. The range starting from 0-49 elements.

```
/* Program showing 1-D array */
#include <stdio.h>
main()
{
   int i;
   float x[10], value, total;
   printf("Enter 10 real numbers\n");
      for(i=0; i<10; i++)
      {
         scanf("%f", &value);
         x[i] = value;
      }
      total = 0;
      for (i=0; i<10; i++)
         total = total + x[i];

      for(i=0; i<10; i++)
         printf("X[%2d] = %5.2f\n", i+1, x[i]);
         printf("total = %0.2f", total);

}
```

Output
Enter 10 real numbers
1
2
3
4
5
6
7
8
9
0
X[1] = 1.00
X[2] = 2.00
X[3] = 3.00
X[4] = 4.00
X[5] = 5.00
X[6] = 6.00
X[7] = 7.00
X[8] = 8.00
X[9] = 9.00
X[0] = 0.00
total =

Saathi

Date ___/___/_____

2-D Arrays: To store tables we need 2-D arrays. Each table consists of rows & columns. 2-D arrays are declare as -

type array name [row-size][col-size];

/* WAP showing 2-D Array */
/* Showing Multiplication Table */

```
#include<stdio.h>
#include<math.h>
#define ROWS 5
#define COLS 5
main()
{
    int row, cols, prod[ROWS][COLS];
    int i, j;
    printf("Multiplication table");
    for(j=1; j<=COLS; j++)
        printf("%d", j);
    for(i=0; i<ROWS; i++)
    {
        row = i+1;
        printf("%2d|", row);
        for(j=1; j<=COLS; j++)
        {
            COLS=j;
            prod[i][j]=row*cols;
            printf("%4d", prod[i][j]);
        }
    }
return 0;
}
```

Date ___/___/___

Saathi

```
#include<stdio.h>
int main(){
    int disp[2][3];
    int i,j;
    for(i=0;i<2;i++){
        for(j=0;j<3;j++){
            printf("Enter value for disp[%d][%d]:",i,j);
            scanf("%d",&disp[i][j]);
        }
}
printf("Two Dimensional array elements:\n");
for(i=0;i<2;i++){
    for(j=0;j<3;j++){
        printf("%d",disp[i][j]);
        if(j==2){
            printf("\n");
        }
    }
}
return 0;
}
```

Output

```
Enter value for disp[0][0]:1
               [0][1]:2
               [0][2]:3
               [1][0]:4
               [1][1]:5
               [1][2]:6
Two Dimensional array elements:
1 2 3
4 5 6
```

(3) Strings (character Arrays):

A String is an array of characters. Any group of characters (except double quote sign) defined b/w double quotes is a constant string.

Declaring:- char String name [size];

Initializing:- 1. Char city[10] = "NEW YORK";

2. Char city[10] = {'N','E','W','Y','O','R','K','\0'};

C also permits us to initializing a String without specifying size.

Ex:- Char Strings[] = {'G','O','O','D','\0'};

Saathi

Date ___ / ___ / ___

```
/* Program to read string using scanf & getchar */
#include<stdio.h>
main(){
  char line[80], ano_line[80], character;
  int c;
  c=0;
  printf("Enter string using scanf to read \n");
  scanf("using getchar enter new line \n");
  do
  {
    character = getchar();
    ano_line[c] = character;
    c++;
  }
  while(character != '\n');
  c=c-1;
  ano_line[c] = '\0';
  return 0;
}
```

(4) Pointers:- is a variable that can store an address of a variable
A pointer itself usually occupies 4 bytes of memory.

Advantages-

1. access a variable that is defined outside the function.
2. more efficient in handling the data tables.
3. reduce the length & complexity of a program.
4. ↑ the execution speed

Declaration:- Datatype * variable-name;

```
eg.  int *ad;      /* pointer to int */
     char *s;      /* pointer to char */
     float *fp;    /* pointer to float */
     char **s;     /* pointer to variable that is pointer to char */
```

Saathi

Date ___/___/___

5) Structures:- Collection of elements of dissimilar data types. Provide the ability to create user defined data types & also to represent real world data.

Declaration:-

```
Struct <Structure name>
{
    Structure element 1;
    Structure element 2;
    =
    =
    Structure element n;
};
```

Then, the Structure variables are declared as

```
Struct <Structure name> <var 1, var 2>;
```

Eg:-

```
Struct emp
{
    int empno;
    char ename[20];
    float sal;
};
Struct emp e1,e2,e3;
```

/* WAP to implement structure */

```
#include <stdio.h>
Struct StudentData {
    char *stu_name;
    int Stu_id;
    int Stu_age;
};
Int main(){
    Struct StudentData Student;
    Student.Stu_name = "Steve";
    Student.Stu_id = 1234;
    Student.Stu_age = 30;
    printf("Student name is: %s", Student.Stu_name);
    printf("\n Student Id is: %d", Student.Stu_id);
    printf("\n Student Age is: %d", Student.Stu_age);
    return 0;
}
```

Output:- Student Name is : Steve
Student Id is : 1234
Student Age is: 30

(6) UNIONS:-

Union, similar to structures, are collection of elements of diff. data types. However, the members within a union all share the same storage area within the computer's memory, whereas each member within a structure is assigned its own unique storage area.

```
struct cx
{
   int i;
   char ch[2];
};
   struct cx s1;
```

```
union ex
{
   int i;
   char ch[2];
};
union ex u;
```

```
/* use of union in C */
#include <stdio.h>
union unionJob
{
   char name[32];
   float salary;
   int workerNO;
} uJob;
struct structJob
{
   char name[32];
   float salary;
   int workerNO;
} sJob;
int main()
{
   printf("Size of union = %d bytes", sizeof(uJob));
   printf("\n Size of structure = %d bytes", sizeof(sJob));
   return 0;
}
```

Output

Size of union = 32
Size of structure = 40.

Date ___/___/___

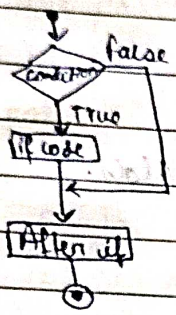
Saathi

7) **If Statement** → used to check the condition given by the user and perform some operation depending on the accuracy of the condition. We use if statement when we need to perform diff. operations in different scenarios.

Syntax:

```
if(expression){
// code to be executed
}
```

Fig: If Statement



// prog. to check largest number using only if Statement in C program

```
#include<stdio.h>
int main(){
    int a,b,c;
    printf("Enter 3 no.s to check largest\n");
    scanf("%d %d %d",&a,&b,&c);
    if(a>b && a>c){
        printf("%d is largest",a);}
    if(b>a && b>c){
        printf("%d is largest",b);}
    if(c>a && c>b){
        printf("%d is largest",c);}
    if(a==b && a==c){
        printf("All no. are equal");}
    return 0;
}
```

Output:- Enter 3 no.s to check largest
10
20
30
30 is largest number.

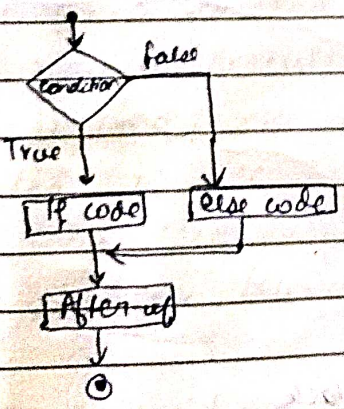
If Else Statement

Used to perform two operations for a single condition in one program. If else condition is an extension of if condition.

Syntax:-

```
if(expression){
// code to be executed if the condition is true
}else{
// code to be executed if the condition is false
}
```

Fig: If Else Statement



Output:

Enter the number=121
121 is not an armstrong number.

// prog to check number Armstrong using if else condition in C.

```
#include<stdio.h>
int main(){
    int n,r,sum=0,temp;
    printf("Enter the number=");
    scanf("%d",&n);
    temp=n;
    while(n>0){
        r=n%10;
        sum=sum+(r*r*r);
        n=n/10;}
    if(temp==sum)
    printf("%d is an armstrong no.");
    else
    printf("%d is not an armstrong no.");
    return 0;
}
```

Saathi

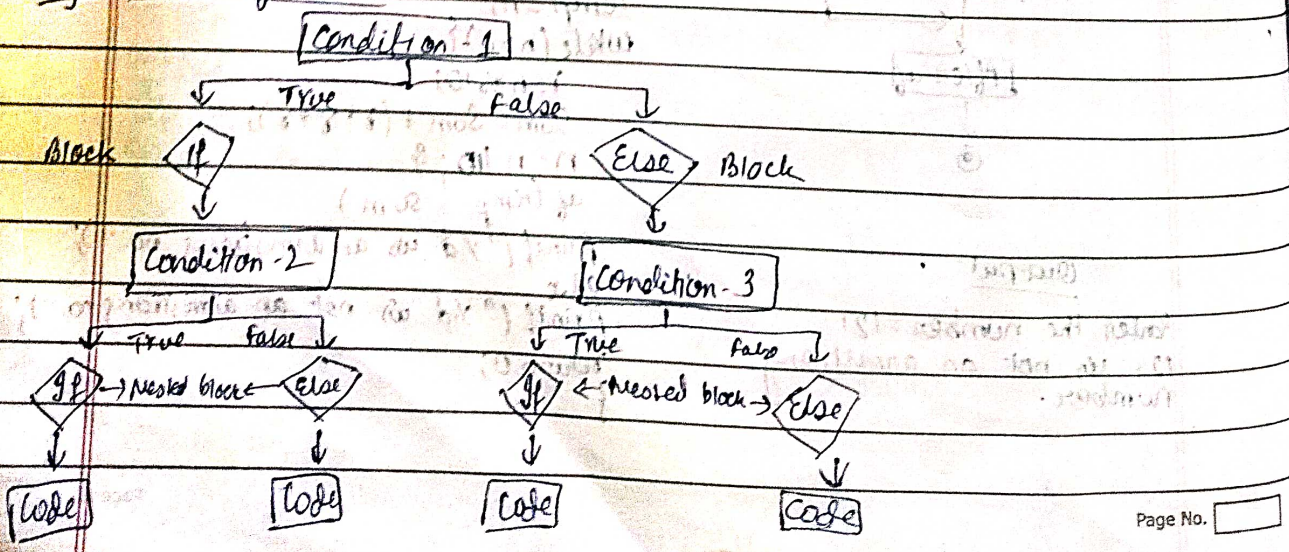
Date ___/___/___

Nested If Else → used to check more than one condition that is already satisfied. We can say that nested if-else statements are an advanced version of if-else statements in C.

Syntax :-

```
// check if the 1st condition holds
if (condition 1) {
    // if 2nd condition holds
if (condition 2) {
do something
    }
    // if 2nd condition doesn't holds.
else {
    do something else
    }
}
// if 1st condition doesn't hold
else {
    // if the 3rd condition holds
if (condition 3) {
do something
    }
    // if 3rd condition doesn't hold
else {
do something else
    }
}
```

fig. Nested if else



Saathi

Date ___/___/___

// Program to check Grade of a student:

```
#include<stdio.h>
int main() {
int marks;
printf("enter marks to check the grade\n");
scanf("%d", &marks);
if (marks >= 35) {
    printf("You are pass\n"); }
if marks (>=90) {
   printf("You got A");
  } else if (marks >=80) {
  printf("You got B");
   } else if (marks >=70) {
  printf("You got C");
   } else if (marks >=60) {
   printf("You got D");
  } else {
   printf("You got a Fail");
  }
  return 0;
}
```

Output

Enter marks to check the grade
99
You are pass
You got A.

(8)

While loop	do-while loop
i) 1st condition is checked, then loop's body is executed.	i) 1st body of loop is executed then the condition is checked.
ii) Statement may not be executed at all.	ii) Statement must be executed at least once.
iii) While loop terminates when condition becomes false.	iii) As long as condition is true, the compiler keeps executing the loop in do-while loop.
iv) Syntax:- while (condition) no semicolon	iv) Syntax:- while (condition); Semicolon

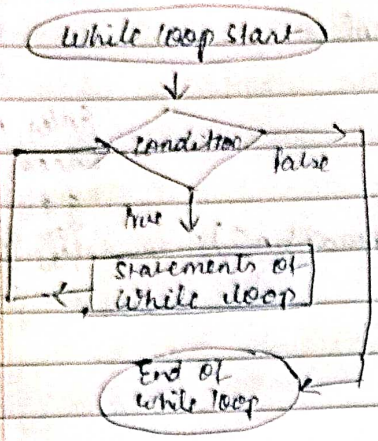
v) Syntax of while loop:-
 while (Condition)
 {
 Block of Statements;
 }
 Statement-x;
	v) Syntax of do-while loops
	 do {
	 Statements; }
	 while (condition);
	 Statement-x;

Page No.

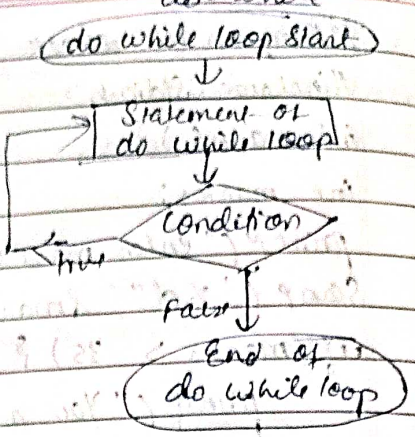
Saathi

Date / /

vi) While



do while




```
/* program to print table
for given no. using while loop */
#include <stdio.h>
int main()
int i=1, number=0, b=9;
printf("Enter a number: ");
scanf("%d", &number);
while (i<=10){
printf("%d\n", (number*i));
i++;
}
return 0;
}
```

Output

```
Enter a number: 50
50
100
150
200
250
300
350
400
450
500
```

```
/* Program to print table for
given no. using do while loop */
#include<stdio.h>
int main(){
int i=1, number=0;
printf("Enter a number: ");
scanf("%d", &number);
do{
printf("%d\n", (number*i));
i++;
}while(i<=10);
return 0;
}
```

output

```
Enter a number: 5
5
10
15
20
25
30
35
40
45
50
```

(9) Define modulus operator. WAP in C to find whether given no. is prime or not.

→ Modulus operator (remainder operator) is an operator that returns the remainder after doing an integer division.

Saathi

Date ___/___/_____

```
/* Program to find whether the given number is prime */
#include<stdio.h>
int main()
int isPrime(int num){
    if (num <= 1){
        return 0;
    }
    for (int i=2; i*i <=num; i++){
        if (num % i==0){
            return 0;
        }
    }
    return 1;
}
int main(){
    int number;
    printf("Enter a positive integer: ");
    scanf("%d", &number);
    if (isPrime(number)){
        printf("%d is a prime number.\n", number);
    } else {
        printf("%d is not a prime number.\n", number);
    }
    return 0;
}
```

(10) Define Nested for loop. WAP to find sum of n natural numbers.

⇒ Nested loop has one loop inside of another. These are typically used for working with 2-D structures such as printing stars in rows & columns. When a loop is nested inside another loop, the inner loop runs many times inside the outer loop.

Saathi

Date ___/___/___

```
#include <stdio.h>
int main()
{
    int n, i;
    int sum = 0;
    printf("Enter a number: ");
    scanf("%d", &n);
    for(i=1; i<=n; i++)
    {
        sum = sum + i;
    }
    printf("The sum of first n numbers is %d", sum);
    return 0;
}
```

(11) Compiler

- Translates the entire source code in single run.
- consumes less time.
- more efficient
- not flexible
- localization of error is difficult
- C, C++, C#

Interpreter

- Translates entire source code line by line.
- consumes much more time
- less efficient.
- flexible.
- Localization of error is easier than compiler
- Python, Ruby, Perl, MATLAB.

```
/* Swap 2 numbers without using 3rd variable */
#include <stdio.h>
int main()
{
int a=10, b=20;
printf("Before swap a=%d b=%d", a,b);
a= a+b;
b= a-b;
a= a-b;
printf("\n After swap a=%d b=%d", a,b);
return 0;
}
```

Output

Before swap a=10 b=20
After swap a=20 b=10

Page No.

Saathi

Date ___/___/___

(12) Role of header file.

→ Header file is a file with extension .h which contains C func declarations & macro definitions to be shared between several source files.

```
/* WAP in C to create Multiplication table of 8 */
#include<stdio.h>
int main(){
    int n;
    printf("Enter an integer:");
    scanf("%d", &n);
    for(int i=1; i<=10; ++i){
        printf("%d * %d = %d\n", n, i, n*i);
    }
    return 0;
}
```

Output

```
Enter an integer: 8
8 * 1 = 8
8 * 2 = 16
:
:
8 * 10 = 180
```

(13) Recursion - A method of solving a computational problem where the sol' depends on 'sol's to smaller instances of same problem.

Recursion happens when a functions calls itself.

Sum of natural numbers using Recursion:-

```
#include<stdio.h>
int sum(int n);
int main(){
    int number, result;
    printf("Enter a positive integer:");
    scanf("%d", &number);
    result = sum(number);
    printf("sum = %d", result);
    return 0;
}
int sum(int n){
    if(n!=0)
    // sum() function calls itself
    return n + sum(n-1);
    else
    return n;
}
```

Output

```
Enter a positive integer:
3
sum = 6.
```

Saathi

Date ___/___/___

(14) Function:
A block of statements that can perform a particular task.
eg: main()

```
void main()
{
    int sum(int,int);
    int x=5, y=6;
    total = sum(x,y);
}
```

4 types of functions
i) functions with arguments and return values
ii) functions with arguments and without return values
iii) functions without arguments and with return values
iv) functions without arguments and without return value.

(15)

Call by value	Call by reference
i) Original value is not modified.	i) Original value is modified.
ii) A copy of variable is passed.	ii) A variable itself is passed.
iii) Actual & formal arguments will be created in diff. memory locations.	iii) Actual & formal arguments will be created in same memory location.
iv) Default method in C++, PHP, C#.	iv) Supported only in Java language.
v) Variables is passed using a straightforward method.	v) Pointers are required to store address of variables

C code example of a call by value method:-

```
void main(){
        int a = 10,
        void increment (int);
        cout << " before function calling" << a;
        increment (a);
        cout << " after function calling" << a;
        getch();

        void increment (int x){
            int x = x+1;
            cout << " value is" << x;
        }
```

output

before function calling 10
value is 11
after function calling 10

Page No.