Assignment - 1.

## Chapter - 1

Q.NS1. Explain Generation of computers with their charecterstics.

→ There are five generation of computers which are explained below:

i) 1st Generation

This was from the period of 1940s to 1955. They used vaccum tubes for circuitry for the purpose of memory. These machines were complicated, large and expensive. They were mostly reliant on batch operating systems and punch cards. for example: ENIAC, EDVAC etc.

ii) 2nd Generation.

This was from period of 1957-1963. In this generation COBOL and FORTRAN are employed as assembly language and programming languages. They advanced from vaccum tubes to transistors. This made the computers smaller, faster and more-energy-efficient. And they advanced from binary to assembly language.
Example: IBM 1602, IBM 7094 etc.

iii) 3rd Generation

This was from 1964 to 1971. This period was the development of the integrated circuit. A single integrated circuit is made up of many transistors, which increases the power of a computer while simultaneously lowering its cost. These computers were quicker, smaller, more reliable and less expensive. High level programming languages were utilized. Example: IBM-360 series, the Honeywell-6000 series, etc.

iv) 4th Generation

The invention of the microprocessors brought along this generation. This year 1971-1980 were dominated by fourth generation computers. C, C++ and Java were the programming languages utilized in 4th Generations.
Example: the stor 1000, PDP11, etc.

v) 5th Generation

These computers have been utilized since 1980 and continued to be used now. The defining aspect of this generation is artificial intelligence (AI). 5th generation use ULSI (Ultra large Scale Integration) technology. C, C++, Java, .Net and more programming languages are used. Example: IBM, laptop, Desktop, etc.

Q.NS2. Explain Programming languages and its types.

→ Programming language is a means of communicating with computers, enabling developers to write software and instruct machines to perform specific task. They are the tools used to create this software, allowing developers to write instructions that computers can understand and execute.

There are two types of programming languages, which can be categories into following ways:

a. Low level language
   i. Machine language (1GL)
   ii. Assembly language (2GL)
b. High level language
   i. Procedural-Oriented language (3GL)
   ii. Problem-Oriented language (4GL)
   iii. Natural languge (5GL).

a. Low-level language
low level languages are closer to the hardware and provide little abstraction. They allow for direct manipulation of hardware resources.
i) Machine language. (1GL)
Machine language consists of strings of binary numbers (0s and 1s) and it is the only one language, the processor directly understands. Machine language has an Merits of very fast execution speed and efficient use of primary memory. It is very difficult to programs using 1GL since all the instructions are to be represented by 0s and 1s.

ii) Assembly language (2GL)
Assembly language is also known as low-level language because to design a program programmer requires detailed knowledge of hardware specification. This language uses mnemonics code in place of 0s and 1s. The program is converted into machine code by assembler. The resulting program is referred to as an object code.

b High-level language
These language are designed to be human-readable and provide a high level of abstraction from the underlying hardware. They simplify programming tasks and make code more portable across different platforms. Example : Python, Java, C++, Ruby and JavaScript.
i) Procedural-Oriented language.
Procedural languages focus on organizing code into procedure or functions. They use a linear flow of control and emphasize procedures and subroutines for code reusability. Because of their flexibility, procedural languages are able to solve a variety of problems. It is easier but needs higher processor and large memory. Example : C, Pascal, etc.

ii) Problem / Object-Oriented language (4GL)
OOP languages organize code around objects, which are instances of classes. They promote encapsulation, inheritance, and polymorphism as core concepts. It needs to be translated therefore its execution time is more.

iii) Natural language (5GL).
Natural language is still in developing stage where we could write statements that would look like normal sentences. Since, the program uses normal sentences, they are easy to understand. The programs designed using 5GL will have artificial inteligence. It is slower than previous generation language as it should be completely translated into binary code which is a tedious task.

Q.N.S.3. Define computer software with its types.
→ Computer software refers to a set of program's, instructions, and data that control and enable the functioning of a computer system.
Its types are :
i) System Software
ii) Programming languages
iii) Application software.

i> System software
This software manages and controls the hardware component of a computer and provides a platform for running ~~approac~~ application software. System softer is closer to the system-

ii) Application Software

This software is designed to perform specific tasks or provide services to users. It perform more specialized tasks like word processing, spreadsheets, emails, photo, etc. It needs more storage space as it is bigger in size.

iii) Programming languages.

Programming languages are the tools used to create this software, allowing developers to write instructions that computers can understand and execute.

Q.N.S.4 Explain language processor. Difference Between Compiler and Interpreter.

→ A language processor is a broad term used to describe any software or hardware component that aids in the compilation or interpretation of programming or scripting languages. These processor are essential in the development and execution of software.

| S.No. | Compiler | Interpreter |
|---|---|---|
| i. | Compiler transforms code written in high level programming language into the machine code, at once before program runs. | Interpreter converts each and every high-level program statement, one by one, into the machine code, during program run. |
| ii. | Compiled code runs faster. | Interpreted code runs slower. |
| iii. | Compiler displays all errors after compilation. | Interpreter displays errors of each line one by one. |
| iv. | Compiler is based on translation linking loading model. | Interpreter is based on interpretation method. |
| v. | Compiler takes an entire program. | Interpreter takes a single line of code |

Q.N.S.5. Difference between System and Application software.

→ The difference between system and application software are :-

| S.No. | System Software | Application software. |
|---|---|---|
| i. | They are designed to manage the resources of the system, like memory and process management, security etc. | They are designed to fulfill the requirements of the user for performing specific tasks. |
| ii. | It is written in a low level language like a machine or assembly language. | A high level language is used to write application softwares. |
| iii. | The system software starts running when the system is powered on and runs until the system is powered off. | The application software starts when the user begins and it ends when the user stops it. |
| iv. | The system software is a general purpose software. | The application software is specific purpose software. |
| v. | It is classified as a package program or customized program. | It is classified as time-sharing resource sharing, client server. |
| vi | Capable of running independently. | Can't run independently. |
| vii. | System software are independent of application software. | Application software needs system software to run. |
| viii. | System software is crucial for the effective functioning of a system. | Application software is not extremly important for the functioning of syste |

ii) Application Software
This software is designed to perform specific tasks or provide services to users. It perform more specialized tasks like word processing, spreadsheets, emails, photo, etc. It needs more storage space as it is bigger in size.

iii) Programming languages:
Programming languages are the tools used to create this software, allowing developers to write instructions that computers can understand and execute.

Q.N.S.4 Explain language processor. Difference Between Compiler and Interpreter.

→ A language processor is a broad term used to describe any software or hardware component that aids in the compilation or interpretation of programming or scripting languages. These processor are essential in the development and execution of software.

| S.No | Compiler | Interpreter |
|---|---|---|
| i. | Compiler transforms code written in high level programming language into the machine code, at once before program runs. | Interpreter converts each and every high-level program statement, one by one, into the machine code during program run. |
| ii. | Compiled code runs faster. | Interpreted code runs slower. |
| iii. | Compiler displays all errors after compilation. | Interpreter displays errors of each line one by one. |
| iv. | Compiler is based on translation linking loading model. | Interpreter is based on interpretation method. |
| v. | Compiler takes an entire program. | Interpreter takes a single line of co |

Q.N.S.5. Difference between System and Application software.

→ The difference between system and application software are :-

| S.No. | System Software | Application software. |
|---|---|---|
| i. | They are designed to manage the resources of the system, like memory and process management, security etc. | They are designed to fulfill the requirements of the user for performing specific tasks. |
| ii. | It is written in a low level language like a machine or assembly language. | A high level language is used to write application softwares. |
| iii. | The system software starts running when the system is powered on and runs until the system is powered off. | The application software starts when the user begins and it ends when the user stops it. |
| iv. | The system software is a general purpose software. | The application software is specific purpose software. |
| v. | It is classified as a package program or customized program. | It is classified as time-sharing, resource sharing, client server. |
| vi | Capable of running independently. | Can't run independently. |
| vii. | System software are independent of application software. | Application software needs system software to run. |
| viii. | System software is crucial for the effective functioning of a system. | Application software is not extremly important for the functioning of syster |

Q.No.6: Write down some features of good computer program.

→ Some of the features of good computer program are:-

i. The program should compile and run smoothly on different platforms. A program is said to be more portable, if it is easily adapted in different computer systems.
ii. Program should prevent unwanted works, so that the maintenance cost in future will be low.
iii. It should take less space and easily converted to machine language; and should be more effective.
iv. It should give some performance in all simple to complex conditions.
v. Program should be machine independent.
vi. Cost must be measured over the life of the program and must include both cost and human cost of producing these programs.
vii. Program should be written in such a manner that it allows to add new features without changing the existing module.

Q.No.7 List out the problem solving techniques.

→ Problem is defined as the difference between an existing situation and a desired solution, that is, in accordance with calculation; a problem is numerical ~~solution~~ situation and has simplest form.

To develop the solution for the given problem, the following programming techniques are used:

a) Program Analysis
b) Algorithm Development.
c) Flowchart
d) Coding ~~and Execution~~
e) Compilation and Execution
f) Debugging and Testing.
g) Documentation.

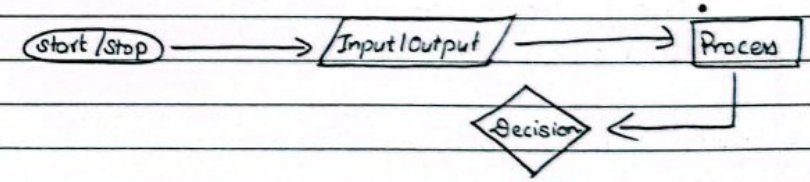
Q.No.8: Explain algorithm. Define term flowchart? Discuss about different symbol used in flowchart.

→ Algorithm is a set of logical procedure to solve the problem. The word 'Algorithm' is the name of one Persian author meaning rules of restoration and reduction. Once the problem is analyzed, its solution is broken into a number of sample steps. A problem in a finite sequence is called an algorithm.

Flowchart is the pictorial representation of an algorith using standard symbol to denote the different types of instruction. Here each step is represented by a symbol and also contains a short description of the process steps within the symbol.

Symbols used in flowchart.

The flowchart being symbolic representation standard symbols is used for each specific operation. The most commonly used symbols are shown below:




Q.No.9 What is debugging and explain types of error.

→ The process of finding and removing errors from a program is known as debugging. One simple method of debugging is to place print statements throughout the program to display the values of variables. There are three types of errors:

a) Syntax Error
b) Logic Error
c) Run-time Error.

a) Syntax Error

Syntax errors are those errors which are arise from violating the rules of programming language. On encountering these errors, a computer display errors message. It is easy to debug.

b) Logic error

Logic errors are those which arises when programmers proceed the logic process in wrong way or miss some statements. It is difficult to debug such errors because the computer does not display them.

c) Run-time Error:

Run-time errors are those which occur when programmers attempt to run ambiguous instructions. They occur due to infinite loop statement, device errors, software errors etc. The computer will print the error message. Some of the run-time errors are:

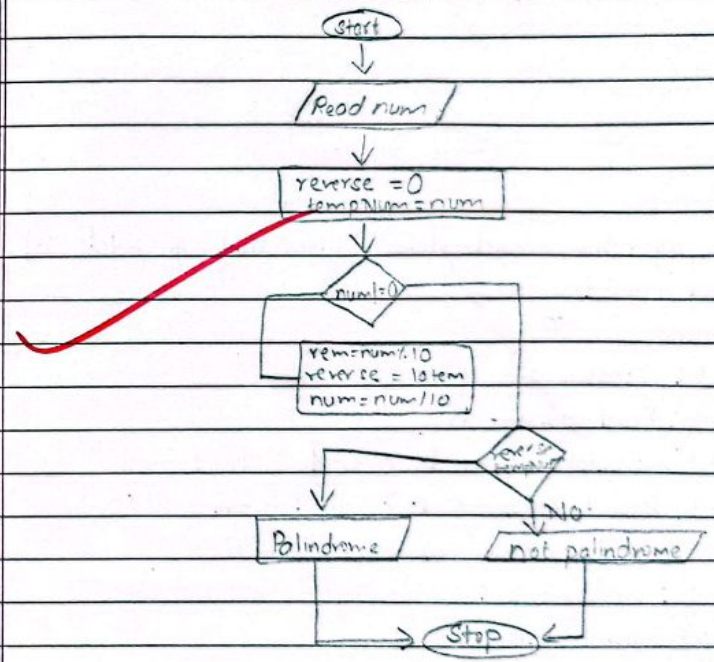
- Divide by zero
- Null pointer assignment
- Data over flow.

* Algorith and flowchart

1. Write an algorithm and flowchart to determine whether a number is palindrome or not.

Step 1 : Start
Step 2 : Read the input number from the user.
Step 3 : Declare and initialize the variable reverse and assign input to a temp variable temperature num =num.
Step 4: Start the while loop until num != 0 become false.
- rem = num % 10
- reverse *= 10 + rem
- num = num / 10

Step 5 : Check if reverse = tempNum
Step 6 : If its true then the number is a plaindrome.
Step 7 : If not, the number is NOT a palindrome.
Step 8 : Stop.

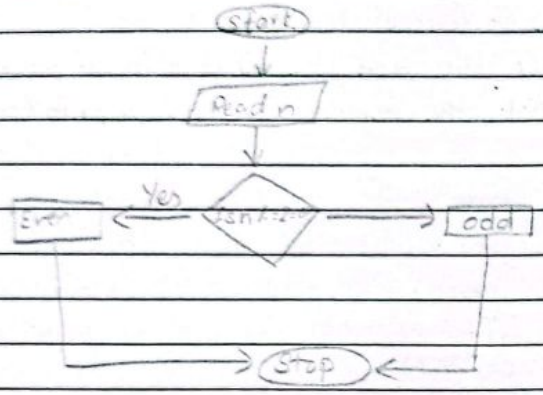



2. Write an algorithm and flowchart to determine whether a number is odd or even.

Algorithm :

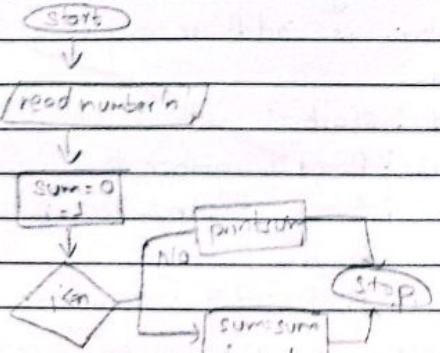
Step 1 : Start
Step 2 : Read a number to n
Step 3 : Divide the number by 2 and store the remainder in R.
Step 4 : If R=0 then print an even number

Step 5: Else, print an add number.
Step 6: Stop.

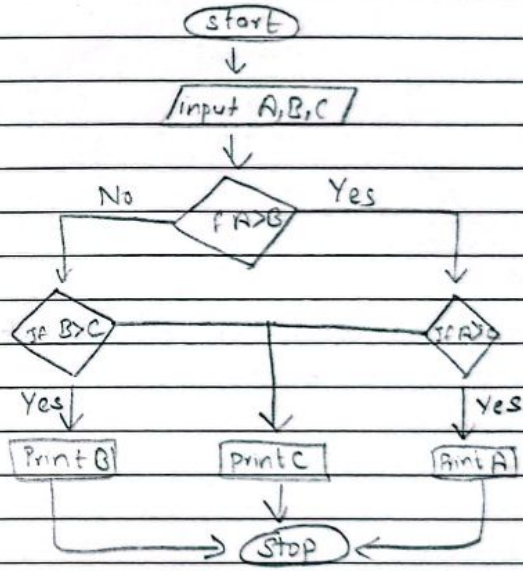



3. Write algorithm and draw flowchart to add N natural numbers.

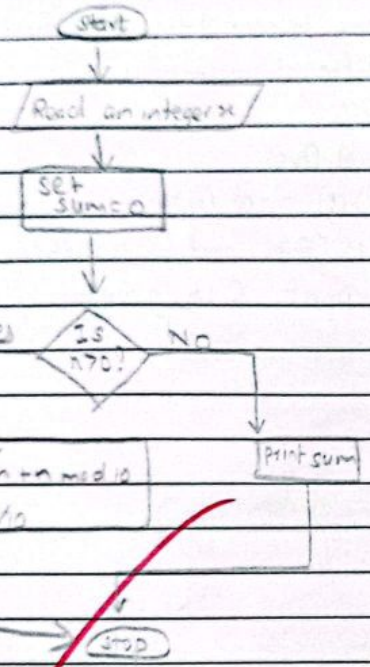
→ Algorithm
Step 1: Start
Step 2: Read number n
Step 3: Declare sum to 0 and i to 1
Step 4: Repeat steps 5 to 7 until i<=n
Step 5: Update sum as sum = sum + i
Step 6: increament i
Step 7: print sum
Step 8: Stop.




5. Write algorithm and draw flowchart to find the largest of three different numbers.
Step 1: Start
Step 2: Input A, B, C
Step 3: If (A>B) and (A>C) then print "A is greater"
Else if (B>A) and (B>C) then print "B is greater"
Else print C is greater".
Step 4: Stop.

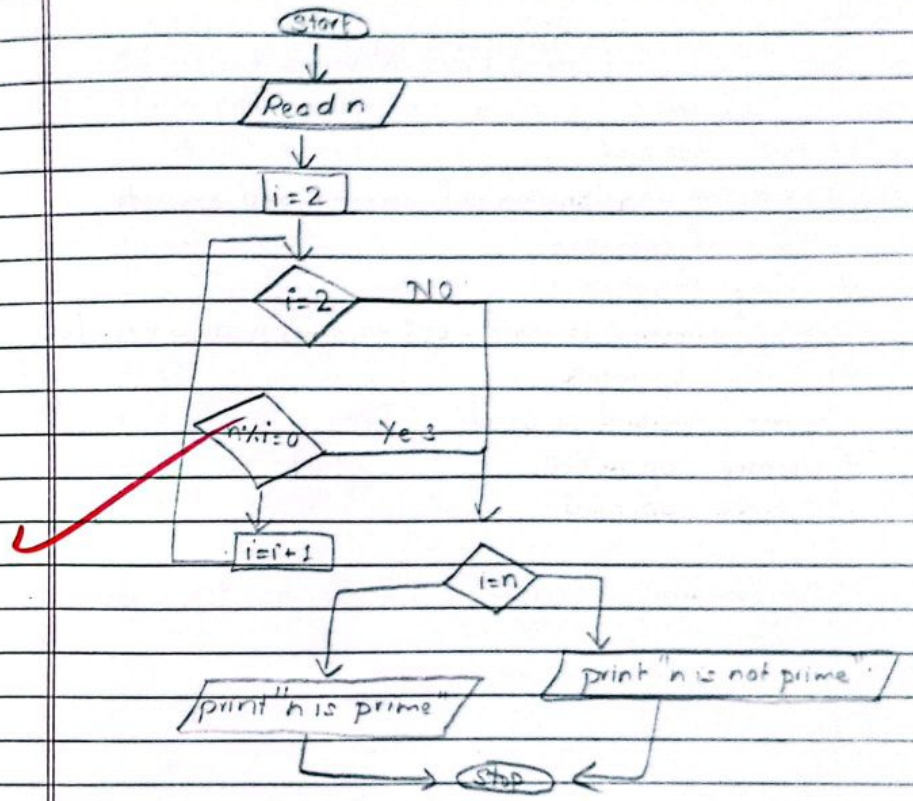



6. Write algorithm and draw flowchart for a program that calculates sum of digits of an integer number.
Step 1: Start
Step 2: Take a number x as input.
Step 3: Initialize a variable to 0 say sum = 0.
Step 4: Perform r = x % 10 and add r to sum i.e sum = sum + r
Step 5: Perform x = x/10
Step 6: If x = 0 stop the process and display sum as output else go to step 3.
Step 7: stop.




8. Write an algorithm and draw flowchart to determine whether number is prime or not.

Step 1 : ~~Read n~~. Start
Step 2 : Read n
Step 3 : set f=1.
Step 4 : if n==1 then
print "n is not prime number"
go to step 8.
Step 5 : for i=2 to n-1
step 6 : if n%i = 0 then
set f=1 and break else
go to step 5.
Step 7 : if f=1 then
print "n is not prime number"
else
print "n is prime number"
Step 8 : Stop.




Jandesh
10/28

Chapter 3

# 'OPERATORS'

A + B — operator, operand

Expression - combination of operand and operator.

Types of operators

* Uniary operators
→ single operand is used. eg: -a, ++a, --a, a++, etc.
* Binary operators
→ two operand is used.
* Ternary operators
→ three operand is used.

Post increment / Decrement
i++ , i--

Pre increment / Decrement
++i, --i

① Arithmetic operator
+ - Addition
- - Subtraction
* - Multiplication
/ - Quotient
% - Remainder

② Logical operator
&& → And operator
|| → OR operator
! → not operator

③ Relational operator
>
<
>=
<=
== comparing
!=

④ Assignment operator.
+=
-=
/=
*=
%=
=

⑤ Conditional operator
Exp^n 1 ? Exp^n 2 : Exp^n 3 / Eg: a>0 ? printf(a is +ve):(a is-

⑥ Bitwise operator
& - and / Eg: printf("Bitwise and operator = %d", 4&3);
| - OR
^ - X-OR
<< left shift
>> Right shift.

Precedence and Associativity.

| | | |
|---|---|---|
| 1 | $ | left to Right. |
| 2 | /, %, * | |
| 3 | +, - | |

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

a$b = a^b

✻ ++, i--

int i = 1.

| | for i++ | for, i-- |
|---|---|---|
| printf (i = %d \n, i) | 1 | 1 |
| printf (i = %d \n, i++) | 1 | 1 |
| printf (i = %d \n, i) | 2 | 0 |

for ++i

1

2

2

✻ a=5, b=2, c = (a++) + (++b)

a = (c--) + (b++)

a++.

| | |
|---|---|
| printf (a = %d \n, a) | 25 |
| printf (a = %d \n, a++) | 25 |
| printf (a = %d \n, a) | 26 |

c = 5 + 3 = 8

a=6, b=3, c=8

a = 8 + 3 = 11

b=4, c=7

# Chapter - 4

## INPUT AND OUTPUT

formatted I/O
→ Use format specifier

Unformatted I/O
→ Don't use format specifier (only for string & character).

✻ Input

scanf ( ):

Syntax

scanf ("format string", list of variable);

Eg

```
scanf ("%d", &a);
scanf (%f, &b);
scanf (%c, &c);
scanf (%s, &s);
```

✻ Output:

printf ( ):

syntax.

printf ("format string", list of variable);

Eg.

```
printf ("Welcome to class");
printf ("Sum = %d", sum);
printf ("a %d \t b = %d \n", a, b);
```

Character

Input :

variable = getch ( );

variable = getchar ( );

Eg: char ch = getch ( );

Output :

putchar (variable)

Eg: putchar (ch);

putchar (A);

## String

Input
gets (string-variable);
Eg: char s[10];
gets (s);

Output
puts (string-variable);
Eg: chars[10];
puts (s);
→ puts ("welcome to C class")

✳ Minimum field width.

a = 1, 2, 3, 4

printf ("%d \n", a); ⇒ 1234

printf ("%.10d \n", a); ⇒ | | | | | | 1 | 2 | 3 | 4 |
↳ minimum 10.

printf ("%-10d \n", a); ⇒ | 1 | 2 | 3 | 4 | . | | | | | |

printf ("%010d \n", a); ⇒ 0000001234

✳ precision specifier.

%a.b datatype

%2.3f

f = 12.52.

printf ("%.4.3f ", f);

Row → for ( )
{
Colum { for ( ) { } printf("\n") }
}

16-feb.

(1)
```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

```
for (i=1; i<=5, i++)
{
  for (j=1; j<i ; j++)
  {
    printf ("%d\t", j);
  }
  printf ("\n")
}
return 0;
}
```

(2)
```
5 4 3 2 1
5 4 3 2
5 4 3
5 4
5.
```

```
int main ( )
{
int i,j;
for (i=5; i>=1; i--)
{
  for (j=5, j>=i, j--)
  {
    printf ("%d\t", j);
  }
  printf("\n");
}
return 0;
}
```

3. N
N E
N E P
N E P A
N E P A L

```
int main ()
{
  int i,j; ⟶.        char s[] = "Nepal";
for (i=0; i<4; i++)
  {
    for (j=0; j<=i ; j++)
      {
        printf ("%c\t", s[j]);
      }
      printf ("\n");
  }
  return 0;
}
```

4. char s[ ] = programming";

```
  0 1 2 3 4 5 6 7 8 9 10
  p r o g r a m m i n g
  r o g r a m m i n
  o g r o m m i
  g r a m m
  r o m
  a.
int main ()
{
  int. i,j;
  char s[] = "Nepal";
```

```
Row⟶ for (i=0; i<=5; i++)
  {
    for (j=i; j<=10-1; j++)
      {
        printf ("%c\t", s[j]);
      }
    printf ("\n");
  }
  return 0;
}
```

5.

```
                N
              E E E
            P P P P P
          A A A A A A A
        L L L L L L L L L
```

```
int main ().
{
  int i,j,k;
  char s[] = "NEPAL";
for (i=0; i<=4; i++)
  {
    for (k=4-i; k>=1; k--)
      printf (" ");
    for (j=1  j<=2*i+1 ; j++)
      {
        print f ("%c\t", s[i]);
      }
      print f ("\n");
    }
    return 0;
}
```

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

```
int main ()
{
  int i,j,
 for (i=1 , i<=5 , i++)
 {
 for (j=1 ; j<=i ,j++)
  {
   printf ("%d\t,j")
 }
  printf ("\n")
 for (k= i-1 ; k>=1 ; k--)
  {
    print f ("%d\t", k);
  }
  print f ("\n")
 }
   return 0;
}
```

7. 0 1 1 2 3 5

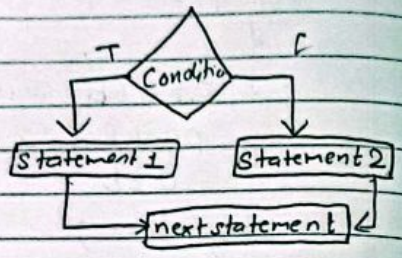
```
int main ()
{
  int a=0, b=1 , c , i, n;
  printf ("Enter no. of terms");
  scanf ("%d",&n);
  printf ("The fibonacci series are : \n");
  printf ("%d \t %d \t",a,b);
  for (i=3; i<=n; i++)
   {
     c=a+b;
     printf ("%d \t", c);
      a=b
      b=c;
   }
 return 0;
}
```

- add even
- HCF LCM.
- leap year

(i) if_else

syntax

```
if (condition)
{
    statement 1;
}
else
{
    statement
}
```




(iii) else if

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



ii. else if

```
if (condition)
{
    statement 1;
else if (condition)
{
    stament 2;
}
.
.
.
else
{
    statementN;
}
```

(iv) switch (Menu driven program).

```
switch (expression)
{
  case constant 1:
        statement;
        break;
  case constant 2:
        statement;
        break;
  .
  .
  .
  Case constant N:
        statement;
        break;
  default :
        statement;

}
```

=) break

=) Exist (0);

=) goto statement :

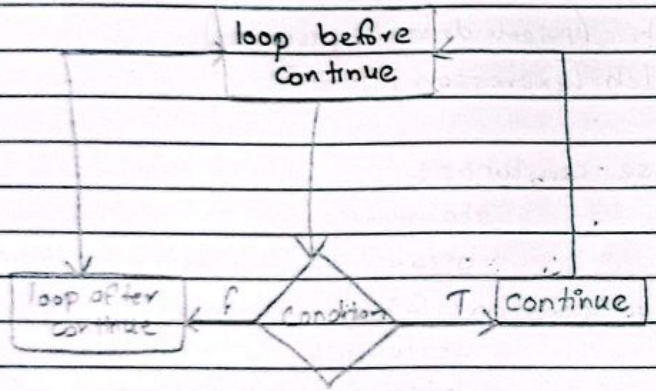
Syntax

```
goto label name;
. . . . . .
. . . . . .
label name :
    statements;
```

=) Continue:

```
continue ;
```

flow chart




Write a programme to find the HCF and LCM of two given number

```
int main ()
{
  int a, b, lc, HCF, LCM, rem;
  printf ("Enter two no:");
  scanf ("%d %d", &a, &b);
    lc = a*b;

  do
   {
     rem = a%b
     if (rem == 0)
      {
        HCF = b;
      }
     else
      {
        a = b;
        b = rem;
      }
  } while (rem != 0);
  LCM = lc/HCF
  printf ("LCM = %d \n", LCM);
  printf ("HCF = %d \n", HCF);
  return 0;
}
```

leap year

(i) divisible by 4 but not by 100
OR
divisible by 400.

```
if ((y%4==0 && y%100 != 0) || (y%400 == 0))
{
    printf ("leap year");
}
else
{
    printf ("Not leap year");
}
```

(ii) Quadratic equation.

```
int main()
{
    float a,b,c,d,x,y,real,img;
    printf ("Enter value of a,b,c");
    scanf ("%f %f %f", &a, &b, &c);
    d = b*b - 4*a*c;
    if (d>0)
    {
        d = sqrt (d);
        x = (-b+d) / (2*a);
        y = (-b-d) / (2*a);
        printf ("The roots are : \n");
        printf ("x1 = %.2f \n", x);
        printf ("x2 = %.2f \n", y);
    }
    else if (d<0
    {
        d = sqrt [fabs (d)];
        real = (-b) / (2*a);
        img = d / (2*a);
        printf ("The roots are : \n");
        printf ("x1 = %.2f+i%.2f \n", real, img);
        printf ("x2 = %.2f-i%.2f \n", real, img);
    }
    else
    {x = -b/(2*a);
     printf ("The root is: \n");
     printf ("x = %.2f", x);
    }
    retur 0;
}
```

(iii) Odd / even.

```
int main ()
{
    int n;
    printf("Enter any no.");
    scanf ("%d", &n);
    if (n%2 == 0)
        goto even;
    else
        goto odd;
    even:
        printf ("The no is even");
    odd:
        printf ("The no is odd");
    return 0;
}
```

continue.

```
int main()
{
   int n, i;
   printf("Enter any no");
   scanf("%d", &n);
   printf("The ~~odd~~ even no from 1 to %d are: \n", n);
   for(i=1; i<=n; i++)
   {
      if(i%2!=0)
            continue;
      printf("%d \t", i);
   }
   return 0;
}
```

Write a menu driven programme to perform mathematical calculation.

```
int main()
{
    int a, b;
    char choice;
    printf("Enter two no:");
    scanf("%d %d", &a &b);
    printf("......Menu ......\n");
    printf("Enter + for addition \n");
    printf("Enter - for subtraction \n");
    printf("Enter * for multiplication \n");
    printf("Enter / for division \n");
    printf("Enter your choice:");
    scanf("  %c", &choice);
    //OR
    choice = getchar();
    switch(chose)
    {
       case '+':
            printf("Sum = %d", a+b;
            break;
       case '-':
            printf("Sum = %d", a-b);
            break;
       case(*):
            printf("Mul = %d", a*b);
            break;
       case '/':
            printf("Div = %d" a/b);
            break;
       defaut:
            printf("wrong choice");
    }
    return 0;
}
```

Assignment - 2

Chapter 2,3,4

1. C language is a middle level language. Explain this statement? Write down the importance of C.

→ C is known as a middle level language because it supports the features of both low and high level languages. A low level language is specific to one machine, it is fast to run but very hard to understand. A high level language is not specific to one machine i.e machine independent. C language can be converted into assembly code, it supports pointer arithmetic, but it is machine independent.

The importance of C are listed below:

i. C is a system programming language which provides flexibility for writing, compilers, operating systems etc
ii. C allows for direct memory manipulation and low level access to system resources. This results in highly efficient code executions.
iii. C gives programmers fine-grained control over memory management and system resources.
iv. Code can be easily integrated with code written in other languages like C++, Java, python.

2. Explain structure of C program with appropriate examples.

| Basic structure of C-program | Example |
|---|---|
| Documentation section [used for comments] | // Sample program |
| Link section | #include <stdio.h> |
| Definition section | void func(); |
| Global declaration section | int A = 54 |
| main() | void main |
| { | { |
| Declaration section | printf("Value of A=%d", |
| Execution section | func(); |
| } | } |
| Sub program section [user defined function] | |
| function 1 | void func() |
| function 2 | { |
| function n | printf("A=%d", A) |
| | } |

3. Give classification of c character set. State that rules for identifiers.

→ The classification of the language c character set are:

i. letter
Uppercase A..... Z
lowercase a..... z

ii. Digits
Decimal digit 0.... 9

iii. Special character
Comma, period, semicolon ; colon : question mark ?, apostrophe ', quotion mark " ", Exclamation mark ! , slash / , underscore _ , dollar $ , percent %. , hastag # , ampersand & , caret ^, asterisk *, minus - , plus + .

The rules of identifiers are:

- first character must be an alphabet (or underscore).
- Must consists of any letters, digits & underscore.
- Only first 31 characters are significant.
- Cannot use a keyword.
- Shouldn't contain white space.

4. What are keywords in C? Explain basic data types along with their qualifiers.

→ Keywords are predefined or reserve words that have special meanings to the compiler and is already explained to the C language compiler. Keywords cannot be used as identifiers or variable names. They should be used only to carry pre-defined meaning.

The basic data types along with their qualifiers are:

a. int:

Int refers to integer numbers. It requires 2 Bytes memory. It can either be signed or unsigned. Derived data types from int are short int, long int.

b. float

float represents real numbers. It requires 4 bytes memory space. It can also be signed or unsigned. Derived data types from float are double float, long float.

c. Char

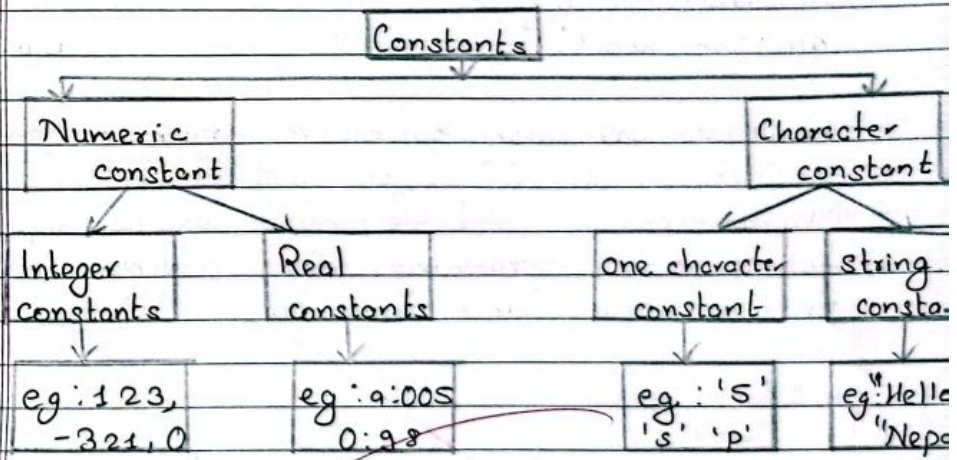
Char represents single alphabet or symbol (eg: 'A', 'f') or multiple alphabets or symbols for the string. It requires 1 Byte memory space. It can also be either signed or unsigned.

d. void.

Void data type has no values. It is used as the return type for functions that donot return a value.

5. Give classification of constant and explain them.

→ There are 4 basic types of constants. They are integer constants, floating constant, character constant and string constants. Integer and floating point constants represents numbers. They are also called numeric type constant.




6. What is Escape sequence? Write any four escape with meaning and symbols.

→ The backslash symbol "\" is considered as escape character. It causes an escape from the normal interpretation of a string so the the next character is recognized as one having especial character meaning. All escap

sequence is represented by back slash (\), followed by a character.
Certain non printing characters, as well as the blackslash (\) and the apostrophe ('), can be expressed in terms of escape sequence. It represents a single character for a specific purpose.

Any four escape sequence with meaning and symbol:

| Character | Escape sequence | Ascii value |
|---|---|---|
| bell (alert) | \a | 007 |
| back space | \b | 008 |
| newline (linefeed) | \n | 010 |
| question mark (?) | \? | 063. |

7. what do you mean by macro expansion and file inclusion in C? explain with example.

→ Macro expansion and file inclusion are two important fectures in C programming that provides mechanism for code reuse and organization.

i) Macro Expansion

In C, macros are a way to define constants or code snippets that can be reused throughout a program. Macro expansion refers to the proces of replacing occurences of macro names with their respective definition of in the code.

Example:

```
#include <stdio.h>
//Define a macro for calculating the square of a number.
#define SQUARE (x) ((x)*(x))

int main()
    int num = 5;
    int result = SQUARE (num) // Macro expansion.
                 SQUARE (5) // becomes ((5) *(5))
    printf("square of %d is % \n", num,
                                   result);

    return 0;
}
```

ii) file Inclusion

file inclusion allows you to include the content of another file in your C program. This is typically used to include header files ('*.h') th contain function prototypes, macros and other declaration needed in the program.

Example:

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

8. What are variables and how are they classified?

→ A variable is an identifier that is used to represent some specified type of information within a degi designated portion of the program. In its simplest form, a variable is an identifier that i used to represent a single data item, i.e a numerical quantity or a character constant.

Classification of variables in C:

i) Based on data type
- Integer variables
- floating-point variables
- Character variables
- Void variables.

ii) Based on scope
- Local variable: declared within a block, such as a function and are accessible only within that block.
- Global variables: declared outside of any function and are accesible from any part of the program.

9. What are the various types of operators used in C language.

→ An operator is a symbol that tells the computer to perform mathematical or logical manipulations. The various types of operators used in C language are:-

i) Arithmetic operator
performs mathematical operations such as addition, subtraction, multiplication on numerical value.
Arithmetic operators are: +, -, *, /
Example: value = a+b;

ii) Relational Operators
- checks the relationship between two operands.
- If the relation is true, it returns 1; if the relation is false it returns value 0.

iii) Logical operators
- Used in decision making in C programming.
- Logical operator returns either 0 or 1 depends upon whether expression results true or false.
- Logical operators are: &&, ||, !.

iv) Assignment operators
- Used for a value to a variable.
- Assignment operators are: =, +=, -=, *=, /=
eg: a+=b ⟶ a=a+b
a*=b ⟶ a=a*b

v) Conditional operator.

a. Unary operators: It acts on single operands. The most common unary operator is unary minus (-) that appears before a data item to make it negative.
Example: unary minus (-5, -20, etc), address of operator (&a).

b) Binary operators
It acts on two operands.
Example: +, -, %, /, *, etc.

c Ternary operators
It acts on three operands. The symbol ? : is called ternary operator in C language.
Usage: big = a>b ? a : b; i.e if a>b, then big = a else big = b.

10. What do you mean by precedence and associativity of an operator? Explain with suitable example.

→ If more than one operands operator is involved in an expression, C-language has a predefined rule of priority for the operators. This rule priority of operators is called operator precedence.

If two or more than two operators of same precedence is present in an expression, associativity indicates the order in which they execute.
for example, addition associativity is left to right is present in an expression, as: 2+3+4 is evaluated as (2+3)+4. In contrast, the assign operators associativity is right to left; so the expression x=y=z evaluated as x=(y=z)

Example of precedence operator.
Multiplication is of higher precedence than addition so the expression 2+3*4 is evaluated as

3*4 = 12
2+12 = 14

The evaluation order can be explicity controlled using parenthese, eg: (2+3)*4 is evaluated as

2+3 = 5
5*4 = 20

11) Write the difference between formatted I/O and unformated I/O.

→ The difference between formatted I/O and unformated I/O are:-

| SN. | formatted I/O | Unformatted I/O. |
|---|---|---|
| i) | Supply I/O in user desired format. | Suply I/O as only character steam of characters. |
| ii. | Contain formate specifiers in their syntax. | Donot contain format specifier. |
| iii. | Store data in user friendly manner. | Store data more compactly. |
| iv. | Used in all data type. | Used efficiently for characters of string |
| v. | Examples: printf(), sconf() etc. | Examples: getchar(), putchar(), etc. |

12. Write the various input and output unformatted function and discuss any two.

→ The various input and output unformatted functio
i) get char()
ii) put char()
iii) get ch()
iv) get che()

I) The get char() functions.
- Single characters can be entered into the comput using the c library function get char.
- The get char function is a part of standard C I/O library.
- It returns a single character from a standard input device typically a keyboard.

ii) The putchar() functions.
Single characters can be displayed using

using the C library function putchar. This function is complementary to the character input function getchar. The putchar function, like getchar, is a part of the standard I/O library.

14. What is the purpose of the printf() function? How is it used within a C program? Compare with the putchar().

→ Printf() is used to print or display data on the console in a formatted form. The format of printf() is:

Printf("control string", list of arguments);

Its primary purpose is to display information to user in a structured and readable format. Here's how 'printf()' is typically used in a C program:

```
#include <stdio.h>
void main()
{
    int num = 10
    float pi = 3.14159;
    Cha letter = 'A';

printf("The value of num is : %d\n", num);
printf("The value of pi is :%d", pi);
printf(The letter is %c\n", letter);
```

Comparision between 'printf()' and 'putchar()':

i) Printf() is used for formatted output, while 'putchar()' is used for unformatted output.

ii) 'Printf()' can print strings, variables and expression with various format specifiers, allowing for precies control over the appearance of the output. 'putchar()' can only print single characters.

15. What is the purpose of the scanf() function? How is it used within a C program? Compare with the getchar().

→ The scanf() reads the input data from standard input device i.e keyboard. The general format of scanf is

scanf("format string", list of arguments);

Here's how 'scanf()' is typically used in a C program.

```
#include <stdio.h>
int main()
{
    int num;
    printf("Enter an integer:");
    scanf("%d", &num);
    return 0;
}
```

Comparision between 'scanf()' and 'getchar()'-

i) 'Scanf()' is used for formatted input, while getchar() is used for unformatted input of single character.

ii) Scanf() reads based on format specifiers, also for structured input of different types of data. 'get char()' simply reads a single character at a time from the input stream.

Sandesh
12/4

Types of function

1. Library functions →
2. User defined function →

Function declaration or prototype

```
return_type func_name (Arg 1, Arg 2, .... Arg N);
int main ( )
{
    variable = func name (parameters); // function call / function
    return 0;
}
return_type func_name (Arg 1, Arg 2 ..... Arg N)   // function def
{
    return (expn);
}
```

* Actual and formal parameters
  - used in function call
  - used in function definition

* Types of function

## function without argument and no return

(i) function prototype
```
void func_name ( );
```
(ii) function defination
```
void func_name ( );
{
}
```
(iii) function call
```
func_name ( );
```
Example;
```
#include <stdio.h>
void leapyear ();
int main ()
{
  leap year();
    return 0;
}
void leapyear ()
{
 int year;
 printf("enter any year:");
```

## function with no argument and a return value

(i) function prototype
```
return_type function_name ( );
```
(ii) function defination
```
return-type function_name();
{
 return (expression);
}
```
(iii) function call
```
variable = func_name ( );
```
Example:
```
#include <stdio.h>
  int fact ();
  int main ( )
{ int f=1.
  f= fact ();
 printf ("The factorial is
        %d , f );
  return 0;
}
int fact ()
```

## function with argument and no return value

(i) function prototype
```
void func_name (Arg1.....);
```
(ii) function defination
```
void (func_name (Arg1 )
{
}
```
(iii) function call
```
func_name (parameter);
```
Example:
```
#include <stdio.h>
 void arm (int n);
 int main ( )
{
  int m;
  print f ("enter any no :");
  scanf ("%d", &m);
  arm (m) ;//m is actual parameter
  return 0;
}
```

## function with argument and a return value.

(i) function prototype
```
return-type func_name (Arguments);
```
(ii) function defination.
```
{
}
```
(iii) function call
```
variable = func_name (parameter);
```
Example:
```
#include <stdio.h>
int palin (int n);
 int main ( )
{
  int m, p;
  printf ("enter any no:");
  scanf ("%d", &m);
  p = plain (m);
 if (m == p)
```

```
if(year%.4==0&&year
    %.100!=0)||(year%.400
                ==0)
printf("leapyear");
else
   printf("Not leap
               year");
}
```

```
{ int n,i,f=1;
 printf("enter any no:");
scanf("%d",&n)
for(i=1;i<=n;i++)
{
 f=f*i;
}
 return f;
}
```

```
void arm(int n)
{
 int d=0,x,rem,sum=0;
 while(x!=0)
  {
    d++;
    x=x/10;
  }
 x=n;
 do
 {
 rem=x%10;
 sum=sum+pow(rem,d);
  x=x/10;
 } while(x!=0);
 if(n==sum)
   printf("It is Armstrong");
else
   printf("It is not Armstrong");
}
```

```
else,
printf("It is not palindrome");
 return 0;
}
int plain(int n)
{
 int rem,rev=0;
 do
 {
  rem=n%10;
   rev=rev*10+rem;
    n=n/10;
 }while(n!=0);
  return rev;
}
```

WAP to find LCM and HCF of two number using function.

```
#include <stdio.h>
int hcflcm (int a, int b);
int main ()
{
  int a,b, lcm, hcf;
  printf ("Enter two value:");
  scanf ("%d %d", &a, &b);
  hcf=hcflcm (a,b);
  lcm = (a*b) /hcf;
  printf ("HCF = %d \n", hcf);
  printf ("Lcm %d \n", lcf);
   return 0;
 }
 int hcflcm (int a, int b)
{
  int rem, hcf;
 do
  {
    rem = a % b;
     if (rem == 0)
          hcf = b;
     else
      {
        a = b;
         b = rem;
      }
    } while (rem != 0)
     return hcf;
}
```

Q. Define a function named fact() to calculate factorial of a number n and then write a program that uses this function fact() to calculate combination and permutation.

$$\left[\frac{n!}{(n-r)!\,r!} = {}^{n}C_{r}\right]$$

```
#include <stdio.h>
int fact (int n)
{
  int i, f=1;
  for (i=1; i<=n; i++)
  {
    f=f*i
  }
  return f;
}

int combination (int n, int r)
{
return (fact(n) / fact(r) * fact (n-r));
}

long permutation (int n, int r)
{
 return ( fact (n) /fact (n-r));
}
int main ( )
{
  int n, r;
  printf ("Enter any number : ");
  scanf ("%d", &n);
  printf ("Enter r : ");
  scanf ("%d", &r);
```

```
printf ("Permutation = %ld \n", permutation (n,r));
printf ("Combination = %ld \n", combination (n,r));
printf ("The factorial of number %d is %d \n",
                                  n, fact (n));
return 0;
}
```

# Call by value and call by reference;

| Call by value | Call by reference. |
|---|---|
| ① pass value to function. | pass address to function. |
| ② Actual parameter doesn't change even if formal parameter changes. | Actual parameter changes according to formal parameter. |

```
#include <stdio.h>
void swap (int a, int b)
{
  int c;
  c=a;
  a=b;
  b=c;
  printf ("value of a and b in swap function
        are : %d \t %d \n", a, b);
}
int main ()
{
  int a = 10, b=20;
printf ("value of a and b before swapping
        are : %d \t %d \n", a, b);
swap (a,b);
printf ("values of a and b after swapping function are : %d \n
%d \n", a, b)
```

call by reference.

```
#include <stdio.h>
void swap (int *a, int *b)
{
    c = *a;
    *a = *b;
    *b = c;
    printf(" value of a and b in swap function
          are : %d \t %d \n", *a, *b);
}
int main ()
{
   int a = 10, b = 20;
```

Static variables:

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

```
increment ();
increment ();
}
```

Output : 1 2 3 4

Recursive function:
factorial using recursion.

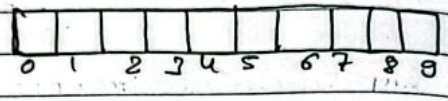
```
#include <stdio.h>
int fact (int n);
int main ()
{
   int n;
   printf ("enter any no:");
   scanf ("%d", &n);
   f = fact (n);
   printf ("the factorial is %d", f);
   return 0;
}
int fact (int n)
{
   if (n == 0)
        return 1;
   else
        return (n * fact (n-1));
}
```

Array, string and pointer. → is static memory allocation [can't be resized so pointer is used].

Array : It is the collection of data item of same datatype under common name.
: array itself is a pointer.

Syntax : datatype array-name [size];

Eg :
```
int A[10];
float B[20];
char s[10];
```
↳ string.




Key points about array in C:
1) fixed size.
2) Homogeneous elements
3) Zero-based indexing.
4) Contiguous memory allocation
5) Declaration syntax : datatype array-Name [arraySize]
6) Initialization.
7) Accessing elements.
8) Usage

size = size of(A) / size of(A[0])
of array.

Advantages of C array.
i) Efficient memory usage.
ii) Random access
iii) Simple syntax
iv) Versatility
v) Efficient iteration.

Disadvantages
i) fixed size.
ii) Memory wastage.
iii) No Bound checking
iv) Inflexible.
v) Passing by reference.

Types :-
1. One Dimension array. (1D)

Declaration : datatype array-name [size];
eg : int A[10];

Int Input : for (i=0; i<n; i++)
{
scanf ("%d", &A[i]);
}

processing : Sorting < Asce / Desc
Maximum, Second largest
Minimum, Second smallest.

Output : for (i=0; i<n; i++)
{
printf ("%d \t", A[i]);
}

* WAP to read and display on array of n element.

```
int main()
{
  int n,i;
  print f("Enter no. of elements: ");
  scanf("%d", &n);
  int A[n];
```

```
printf ("Enter %d elements: ", n);
for (i=0; i<n ; i++)
scanf ("%d", &A[i]);
printf ("Displaying an array : \n");
for (i=0; i<n; i++)
    printf ("%d \t", A[i]);
return 0;
}
```

* WAP to short (a.k.a bubble sort.) an array of element and display them along with maximum element, minimum element, second largest and second minimum.

```
int main()
{
int A, i, j, temp;
printf ("Enter no. of elements :");
scanf ("%d", &n);
int A [n];
printf ("Enter %d elements: ", n);
for (i=0; i<n ;i++)
scanf ("%d", &A [i]);
printf ("The array before sorting :\n");
    for (i=0; i<n ; i++)
    printf ("%d \t", A [i]);
for (i=0; i <n ; i++)
{
 for (j= i+1; j<n ; j++)
 {
  if (A [i] >A [j])
    {
      temp = A [i];
      A[i] = A [j];
```

```
    }
  }
}

printf ("The array after sorting :\n");
    for (i=0; i<n ; i++)
printf ("%d \t", A [i]);
printf ("\n maximum = %d \n", a [n-1]);
printf ("\n minimum = %d \n", a [0]);
printf ("\n second largest = %d \n", a [n-2]);
printf ("\n second smallest = %d \n", a=1);

return 0;
}
```

* WAP to sort an array of n element and disply them along with maximum element, minimum element, second largest and second minimum using function.

```
#include <stdio.h>
void sort (int a[], int n);
void input (int x[], int n);
void display (int x [], int n);
int main()
{
int n, i, j, temp;
printf ("enter any no: ");
scanf ("%d", &n);
int a [n];
printf ("enter element : ");
 input (a, n);
```

```
sort (a,n);
output (a,n);
return 0;
}
ret void sort int a[], int n)
{
  int i,j, temp;
for (i=0; i<n; i++)
{
  for j=i+1; j<n; j++)
  {
    if (a[i]>a[j])
    {
      temp=a[i];
      a[i]=a[j];
      a[j] = temp;
    }
  }
}
void input (int x[], int n)
{
  int i;
  for (i=0; i<n; i++)
  scanf ("%d", &x[i]);
}
void output (int x[], int n)
{
  int i;
  printf ("\n The elements after sorting are:\n");
for (i=0; i<n; i++)
  printf ("%d\t", x[i]);
printf ("\n Maximum = %d \n", x[n-1]);
printf ("\n Minimum = %d \n", x[0]);
printf ("\n Second largest = %d \n", x[n-2]);
printf ("\n Second smallest = %d \n", x[1]);
}
```

15th March.

## Two dimensional

Syntax : Array.

datatype array_name[size][size2];

Eg

```
int A[2][3];
char S[5][7];
```

Initidization

```
int A[2][2] = {{1,2}, {3,4}}
          OR {1,2,3,4}

int B[][2] = {1,2,3,4,5,6};
int C[2][] = {1,2,3}; X (// Invalid)
int D[2][3] = {1,2,3,4};
```

Input :

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

Output :

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

Processing :

- Matrix
- Sum of Two matrix
- Multiply of two matrix
- Transpose
- Diagonal sum
- Sum of all elements

} with / without function.

Q. WAP to read and display the matrix of $n \times m$ order.

```
#include <stdio.h>
int main ()
{
  int m, n, i, j;
  printf ("Enter row and column");
  scanf ("%d %d", &m, &n);
  int A [m] [n];
  printf ("Enter elements of materials");
  for (i=0; i<m; i++)
   for (j=0; j<n; j++)
    scanf ("%d", &A [i] [j]);
  printf (" The matrix is : \n");
  for (i=0; i<m; i++)
  {
    for (j=0; j<n; j++)
    {
      printf ("%d\t", A[i] [j]);
    }
    printf ("\n");
  }
  return 0;
}
```

Formulae :

(a) Sum of matrix

$Sum[i][j] = A[i][j] + B[i][j]$

(b) Multiply : $M_{m \times q} = M_{m \times q} + A_{m \times n} * B_{p \times q}$

Uncommon part : m, q, i, j

$M_{ixj} = M_{ixj} \times A_{ixk} \times B_{kxj}$

Common part : n, p (k).

$M[i][j] = M[i][j] + A[i][k] * B[k][j]$

(c) Transverse : $T[i][j] = A[j][i]$

$$A \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}_{m \times n} \qquad T \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}_{n \times m}$$

(d) Sum of diagonal

```
if (i==j)
  sum = sum + A[i][j];
```

$$\begin{bmatrix} 1_{00} & 2_{01} \\ 3_{10} & 4_{11} \end{bmatrix}$$

Sum of all element
sum = sum + A[i][j];

Q. WAP to read two matrixes of order mxn and display their sum with/without their function.

*) Without using function

```
#include <stdio.h>
int main ()
{
  int m,n,i,j ;
  printf ("Enter row and column");
  scanf (" %d %d", &m ,&n);
  int A[m][n], B[m,n] , sum[m][n];
  printf("Enter element of first matrix");
  for (i=0; j<m ; i++)
  for (j=0 ; j<n ; j++)
  scanf ("%d", & A[i][j]);

  printf ("Enter elements of second matrix");
  for (i=0; i<m; i++)
  for (j=0; j<n ; j++)
  scanf ("%d", & B[i][j]);
  for (i=0 ; j<m ; i++)
  for (j=0; j<n ; j++]
  sum[i][j] = A[i][j] + B[i][j]

  printf ("The sum of two matrix:\n");
  for (i=0; i<m ; i++)
  {
  for (j=0; j<n ; j++)
  {
  printf ("%d\t", sum[i][j]);
  }
  printf ("\n");
  }
  return 0;
}
```

*) Using function.

```
int m,n,i,j;
void input (int X[m][n]);
void sum (int A[m][n], int B[m][n], int S[m][n]);
void output (int sum[m][n]);
int main ()
{
  printf ("Enter row and column");
  scanf ("%d %d", &m &n);
  int A[m][n], B[m][n], sum[m][n];
  printf ("Enter elements of first matrix:");
  input (A);
  printf ("Enter element of second matrix:");
  input (B);
  sum (A, B, S);
  output (S);
  return 0;
}

void sum (int A[m][n], int B[m][n], int S[m][n])
{
```

```
for (i=0 ; i<m; i++)
for (j=0 ; j<n ; j++)
sum[i] [j] = A[i] [j] + B[i] [j] ;
}
void input (int X[m] [n])
{
  for (i=0; i<m; i++)
  for (j=0; j<n; j++)
  scanf ("%d", &X[i] [j]);
}

void input (int s[m] [n])
{
  printf ("The sum of two matrix: \n");
  for (i=0; i<m ; i++)
  {
    for (j=0 ; j<n ; j++)
    {
      printf ("%d\t", sum [i] [j] );
    }
    printf ("\n");
  }
}
```

Q. WAP to read two matrixes of order mxn and pxq respectively and perform multiplication between them if possible and display their result.

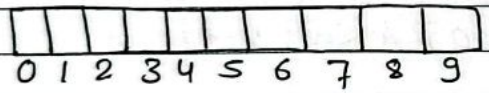
```
#include <stdio.h>
int main()
{
  int m,n,p,q,i,j,k;
  printf ("Enter rows and columns");
  scanf ("%d %d %d %d", &m, &n, &p, &q);
  int A[m] [n], B[p] [q], M[m] [q];
  if (n==p)
  {
    printf ("Enter first matrix");
    for (i=0; i<m ; i++)
    for (j=0 ; j<n ; j++)
    scanf ("%d," &A[i] [j]);

  printf ("Enter second matrix");
  for (i=0; i<p ; i++)
  for (j=0; j<q ; j++)
  scanf ("%d," &B[i] [j]);

  for (i=0 ; i<m; i++)
  {
    for (j=0 ; j<q ; j++)
    {
      M[i] [j] = 0;
      for (k=0; k<n; k++)
      {
        M[i] [j] = M[i] [j] + A[i] [k] * B[k] [j];
      }
    }
  }
  printf ("Multiplication of two matrices: \n");
  for (i=0; i<m; i++)
  {
    for (j=0 ; j<q ; j++)
    {
      printf ("%d\t", M[i] [j]);
```

```
    }
    printf("\n");
  }
}
else
  printf("Multiplication is not possible");
  return 0;
}
```

16 March

String
Array of character
↳ terminated by Null (\0)

Syntax:
```
char string_variable[size];
```
Eg
```
Char s[10];
```




Initialization:
```
char s[] = "NEPAL";
char s[] = {'N','E','P','A','L'}
```
Input:
```
Sconf("%s", s);
```
OR, `gets(s);`
Output:
```
printf("%s", s);
```
OR, `puts( );`

Array of string
```
char s[5][7];
char s[2][7] = {"Sunday", "Monday"};
```
OR `{{'S','U','n','d','o','y','\0'},`
`{'M','o','n','d','a','y','\0'}};`

Input
```
for(i=0;i<5;i++)
{
scanf("%s",s[i]);
}
```

String handling / Manipulation function:

Strlen(): find length of string
syntax:
```
int_variable = strlen(string)
```
Eg
```
int len = strlen("NEPAL");
```
Strcpy: Copy one string to another
```
Strcpy(Dest_string, source_string);
Strcpy(s, "NEPAL");
```

Strcat → Combine two string as one:
```
Strcat(s1, s2);
```
↳ s1 ⇒ s1+s2;

eg
```
strcat("Computer", "Programming");
```
↳ computer programming

Strrev → reverse the string.
```
strrev(string);
strrev("Hello");
```
↳ olleH

Strcmp → compare two string
```
int_variable = strcmp("Ram", "Raju"); (s1, s2);
```
Eg: `int d = strcmp("Ram", "Raju");`
if d → >0 string is greater than string 2
→ =0 " " equal to " "

$(1100)_2 \longrightarrow (9)_{10}$.

$= 1\times2^3 + 1\times2^2 + 0\times2^1 + 0\times2^0$

$= 8 + 4 + 1$

$= 12$

$(12)_{10} \longrightarrow (9)_2$

$2 \mid 12$

$2 \mid 6$

$3$

```
do {
    rem = bn % 10;
    dn = dn + rem * base;
    base = base * 2;
    bn = bn / 10;
} while (bn != 0)

int main()
{
    int bn, dn=0, rem, base = 1;
    printf ("Enter binary no");
    scanf ("%d", &bn);

    printf ("Decimal no: %d", dn);
    return 0;
}
```

classmate
Date 6th Jan
Page

# HISTORY OF COMPUTERS

A computer is an electrical/electronic machine that collects informations, stores it, process it according to instructions and returns results. Abacus was one of the earliest and most well-known device which was invented by Charles Babbage. The farther of computer Charles Babbage designed an Analytical Engine in 1833 which was a general purpose computer. It contained an arthemetic and logic unit, some basic flow chart principles and concept of integrated memory.

After a century, the first electronic computer ENIINAC (Electronic Numerical Integrator and Computer) was invented by John W. Mauchly and J. Presper Eckert. It was Turing-complete and capable of solving vast numerical problems by reprogramming, earning it the title of 'Grond father of computers'.

- In 1822- Charles Babbage, a mathematician, invented the steam powered calculating machine capable of calculating number tables. The 'Different Engine' idea failed owing to a lack of technology at the time.
- In 1848- The world's first computer program was written by Ada Lovelac, an English mathematician. Lovelac also include a step-by-step tutorial on how to compute Bernoulli numbers using Babbage's machine.
- In 1890, Herman Hollerith, an investor creates the punch card technique used to calculate the 1880 US census. He would go on to start the corporation that would becom IBM.
- In 1945 the ENIAC was invented. After ENIAC in 1946 The UNIVAC I was designed which was the first general

purpose electronic digital computer which was designed in the US for corporate applications.

## GENERATIONS OF COMPUTERS

There are five generations of computers which are:

i. 1st Generation
This was from the period of 1940 to 1955. They used vaccum tubes for circuitry. For the purpose of memory, they used magnetic drums. These machines were complicated, large and expensive. They were mostly reliant on batch operating systems and punch cards. As output and input devices, magnetic tape and paper tape were implemented. For example: ENIAC, EDVAC, UNIVAC-I etc.

ii. 2nd Generations
This was from the period of 1957-1963. In this generation, COBOL and FORTRAN are employed as assembly languages and programming languages. They ~~were~~ advanced from vaccum tubes to transistors. This made the computers smaller, faster and more energy-efficient. And they advanced from binary to assembly language. Example: IBM 1602, IBM 7094, CDC 1604 etc.

iii. 3rd Generation
This was from 1964 to 1971. This period was the development of the integrated circuit. A single integrated circuit is made up of many transistors, which increases the power of a computer while simultaneously lowering its cost. These computers were quicker, smaller, more reliable, and less expensive then their predecessors. High level programming languages such as FORTRON-II to IV, COBOL and PASCAL PL/1 were utilized. For example: IBM-360 series, the Honeywell-6000 series, and the IBM 370/168.

iv. 4th Generation
The invention of the microprocessors brought along this generation. The years 1971-1980 were dominated by fourth generation computers. C, C++ and Java were the progamming languages utilized in 4th Generations. For example: the STAR 1000, PDP 11, CRAY-1 etc.

v. 5th Generation
These computers have been utilized since 1980 and continued to be used now. The defining aspect of this generation is artificial intelligence (AI). Fifth generation computer use ULSI (Ultra Large Scale Integration) technology. C, C++, Java, .Net and more programming languages are used. Example: IBM, Laptop, Desktop, Notebook etc.

## TYPES OF PROGRAMMING LANGUAGE

1. Machine Language
2. Assembly language
3. Low level language
4. High level language

1. Machine Language.

# NUMBER SYSTEM

| | Binary. | Decimal | Octal | Hexadecimal |
|---|---|---|---|---|
| inary | – | Sum of power method ($2^n$) | grouping method (group of 3) | grouping method (Group of 4). |
| ecimal | Divide & Reminder method (÷ by 2) | – | Divide & Reminder method (÷ by 8) | Divide & Reminder method (÷ by 16) |
| tal. | Reverse grouping method (group of 3) | Sum of power ($8^n$) | – | Octal → Binary Hexadecimal ↵ |
| exadecal | Reverse grouping method (group of 4) | Sum of power ($16^n$) | Hex → Binary octal. ↵ | – |

## PROBLEM SOLVING USING A COMPUTER

i. Problem Analysis
radius, pi, area, area = pi X (radius)$^2$

ii. Algorithm and flow chart.
a) start b) input radius c) define pi d) calculate area = pi x r$^2$
e) display area f) end.

iii. Programming.
iv. Compilation, linking and execution
v. Debugging and Testing
vi. Documentation

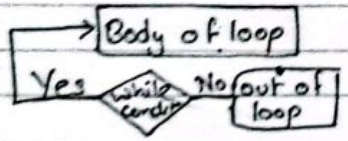
# Assignment-3

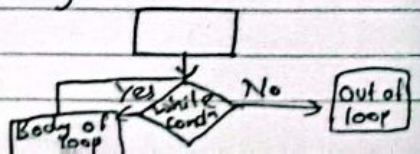
Chapter - 5, 7.

1. What is a control statement ? Differentiate between do while and while statement with the help of flowcharts.

→ The statement which alter the flow of execution of the program are known as control statements. The differentiate between do while and while statement with flowcharts are :-

| S.No. | Do while statement | while statement |
|---|---|---|
| (i) | Do while loop is exist controlled loop i.e the body of the loop is executed 1st without checking condition and at the end of body of loop, the condition is evaluated. | While statement is entry controlled loop i.e test condition is evaluated 1st and body of loop is executed only if this test is true. |
| (ii) | The body of the loop is always executed at least once. | The body of the loop may not be executed at all if the condition is not satisfied at very first attempt. |
| (iii) | Syntax : do { body of the loop } while (condition); | syntax :- while (condition) { body of the loop } |




2. Explain looping control statements with their syntax and flowcharts.

→ Loops are used when we want to execute a part o program or block of statement several times. So, a loop may be defined as a block of statements which are repeatedly executed for a certain number of times or until a particular condition is satisfied. There are three types of loop statements in C.

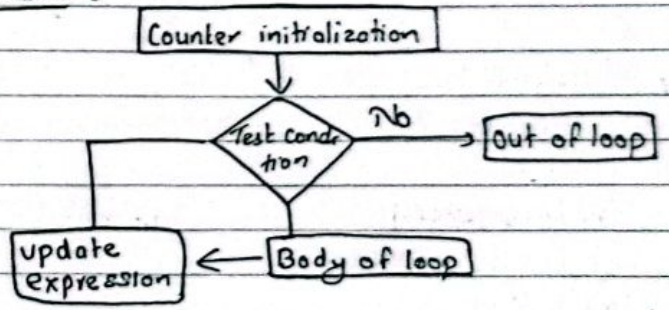
i. for loop
ii. Do while
iii. While

i. for loop

When the number of repetitions is known in advance the use of this loop is also known as determine or definite loop. The use of this loop will be more efficient. The syntax for for (counter initialization; test condition; increment or decrement)

```
{
    /* body of loop */
}
```

Flowchart :



ii. While loop

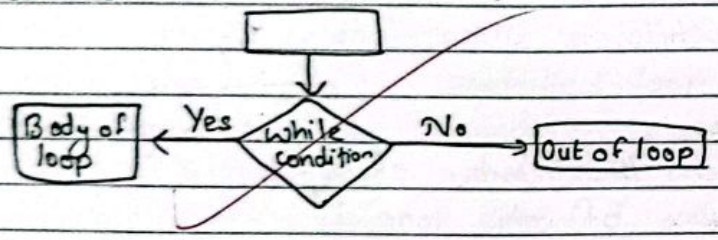
Syntax:

```
while (condition)
statement;
```

OR

```
while (condition)
{
 statement; /* body of loop */
 statement;
 ......
}
```

Flowchart




iii. do while loop

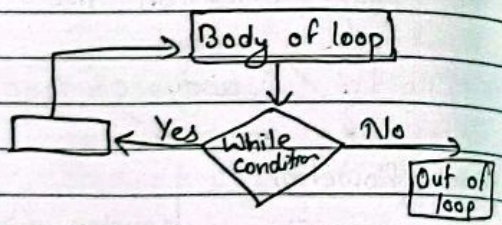
Syntax:

```
do
statement;
while (conditron);
```

OR

```
do
{
 statement 1;
 statement 2;
 ..........
 statementn;
} (while condition);
```

Flow chart:




3. Explain decision control statements with their syntax and flowchart.

→ Decision making statements control the flow of execution, they also fall under the category of control statement. following are decision making statements.

i. If statement

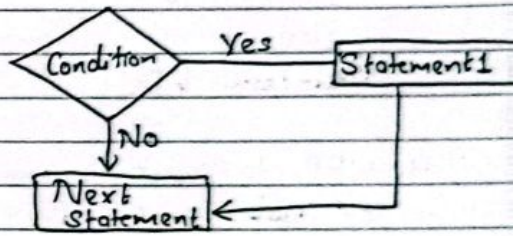
syntax:

```
if (condition)
statement 1;
```

OR

```
if (condition)
{
 statement 1;
 ..... ..... .
 statement n;
}
```

Flowchart:




ii. if else statement

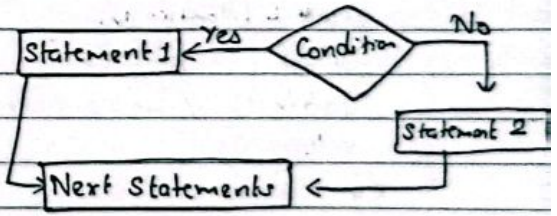
Syntax:

```
if (condition)
 statement 1;
else
 statement 2;
```

OR

```
 if (condition)
 {
  statement;
  ........
 }
 else
 {
  statement;
```

flowchart:



iii. else if statement

Syntax :

```
if (condition 1)
statement A;
else
 if(condition 2)
 statement B;
else
 if (condition 3)
 statement C;
 else
 statement D;
```

4. WAP to input a number to check it is Armstrong or not.

```
#include <stdio.h>
 void arm (int n);
   int main ()
{
  int m;
  print f ("enter any No:");
   scanf ("%d", &m);
   arm(m); // m is actual parameter
   return 0;
}
   void arm (int n)
   {
    int d=0, x=n, rem, sum=0
    while (x!=0)
    {
     d++;
     x=x/10;
    }
     x=n;
     do
     { rem = x%10;
       sum = sum + pow(rem, d);
        x = x/10;
     }
      while (x!=0);
      if (n= sum)
      printf ("It is Armstrong");
      else
      print f ("It is not Armstrong");
   }
```

5. Describe the application of break and continue in C-programming. Explain with examples.

→ In C-programming break and continue are control statement used with loops (like for while and do-while) to alter the flow of execution.

Break : When encounted inside a loop, it terminates the loop immediately, and the program continues executing the code after the loop.

example :

```
#include <stdio.h>
 int main ()
{
  int i;
   for (i=1; i<=10; i++)
  {
      break;
  }
```

```
    printf("%d", i);
}
    printf("\n outside of loop \n");
    return 0;
}
```

Output : 1 2 3 4
outside of loop

Continue : When encountered inside a loop, it skips the rest of the loop's code for the current iteration and proceeds to the next iteration.

example :-

```
#include <stdio.h>
int main().
{
    int i;
    for(i=1, i<=10; i++)
    {
        if (i%2!=0)
        {
            continue;
        }
            printf("%d", i);
    }
        printf("\n outside of loop \n");
        return 0;
}
```

6. Write a menu driven program using switch statement having the following options.

I. Addition
II. Subtraction
III. Multiplication

→ A simple menu-driven program in C using a switch statement to perform addition, substraction, or multiplication.

```
#include <stdio.h>
int main()
{
    int choice;
    float num1, num2, result;
    printf("Menu : \n");
    printf("1. Addition \n");
    printf("2 substraction \n");
    printf("3 multiplication \n");
    printf("Enter your choice (1,2 or 3):");
    scanf("%d", &choice);
    printf("Enter two numbers:");
    scanf("%f %f", &num1, &num2);

    switch (choice)
    {
        case 1:
            result = num1 + num2;
            printf("Result: %.2f \n", result);
        break;
            case 2:
            result = num1 - num2;
            printf("Result : %.2f \n", result);
        break;
            case 3;
            result = num * num2;
            printf("Result: %.2f \n", result);
            break;
```

```
default:
}
 printf("Invalid choice!\n");
 return 0;
}
```

7. Write a program to check the given number is prime or not.

```
#include <stdio.h>
int main()
{
 int num, i=2, c=0;
 printf("Enter number:");
 scanf("%d", &num);
 while (i<=num-1)
 {
  if (num%i==0)
   c++;
  i++;
 }
 if (c==0)
 printf("%d is prime number\n", num);
 else
 printf("%d is not prime number\n", num);
 return 0;
}
```

8. Write a program to check if the number is palindrome or not.

```
#include <stdio.h>
int main()
{
 long n, num, rev=0;
 int rem;
 printf("Enter the number:");
 scanf("%d", &num);
 n=num;
 do
 {
  rem=num%10;
  rev=rev*10+rem;
  num=num/10;
 }
  while (num!=0);
  if (n==rev)
  printf("\n the number %d is palindrome", n);
 else printf("\n the number %d is not palindrome", n);
  return 0;
}
```

10. WAP to print fibonacci series upto $N^{th}$ term:

```
#include <stdio.h>
int main()
{
 int a=0, b=1, c, i, n;
 printf("Enter no. of terms");
 scanf("%d", &n);
 printf("The fibonacci series are:\n");
 printf("%d\t%d\t", a, b);
  for (i=3; i<=n; i++)
```

```
{
  c=a+b
  printf("%d\t",c);
  a=b
  b=c;
}
return 0;
}
```

11. WAP to find all roots of quadratic equations.

→
```
#include <stdio.h>
#include <conio.h>
#include <math.h>
int main()
{
  float a,b,c,d,x,y,real,img;
  printf("Enter value of a,b,c");
  scanf("%f %f %f",&a,&b,&c);
  d=b*b-4*a*c
  if(d>0)
{
  d=sqrt(d);
  x=(-b+d)/(2*a);
  y=(-b-d)/(2*a);
  printf("The roots are:\n");
  printf("x1= %2f\n",x);
  printf("x2 = %2f\n",y);
  else if(d<0)
{
  d=sqrt(fabs(d));
  real=(-b)/(2*a);
  img=d/2*a;
  printf("x1 = %.2f+i%.2f\n",real,img);
  printf("x2 = %.2f-i%.2f\n",real,img);
}
else
{
  x=-b/(2*a);
  printf("The root is:\n");
  printf("x=%.2f",x);
}
  return 0;
}
```

12. WAP to check number calculate HCF and LCM of two number.

→
```
#include <stdio.h>
int main()
{
  int a,b,lc,lcm,HCF,rem;
  printf("Enter two no:");
  scanf("%d %d",&a,&b);
    lc=a*b;
do
{
  rem=a%b
  if(rem==0)
{
  HCF=b;
}
  a=b;
  b=rem;
```

```
    }
  } while (rem!=0);
  LCM = LC/HCF;
  printf ("LCM = %d \n", LCM);
  printf ("HCF = %d \n", HCF);
  return 0;
}
```

14. What are the different types of functions available in C? what do you mean by pass by value and pass by reference.

→ The different types of functions are available in C are as bellow:-

① Standard library function :- These can be in functions provided by the C standard library such as print (), scan (), strcmp (), etc.

② Used-defined functions :- These are functions created by the programmer to perform specific tasks within a program. They help in modularizing the code and improving its readability and maintainability.

→ Pass by value : In this method, a copy of the actual value is passed to the function change made to the progr parameter inside the function do not affect the original value. This is the default method in C.

→ Pass by reference : The address of the actual variable is passed to the function. Any changes made to the parameter inside the function affect the orginal variable. In pass by reference method, you pass the address of the variable using pointers.

15. Difference between iteration and recursive function? Give example.

| | Recursion function | Iteration function |
|---|---|---|
| (i) | A function is called from the defination of the same function to do repeated. | loop is used to do repeated task. |
| (ii) | Recursive is a top-down approch to problem saving. | Iteration is like bottom up approach. |
| (iii) | In recursion, a function calls itself until some condition will be satisfied. | In iteration, a function doesn't call to itself. |
| (iv) | Problem could be defined in terms of its previous result to solve a problem using recursion. | It is not necessary to define a problem in term of its previous result to solve using iteration. |
| (v) | eg: def factorial_recursion(n):<br>if n == 0;<br>return 1<br>else:<br>return n * factorial recursive(n)<br>printf(factorial_recursive(5)). | eg: def factorial_iteration(n)<br>result = 1<br>for i in range(i, n+1);<br>result = 1<br>return result<br>print (factorial_iteration(5)) |

16. What is function with its syntax. Explain actual and formal parameter with example.

→ A function is a group of statements that is executed when it is called from some point of the program. The general syntax is

```
return type function name (parameter 1, prameter 2,.... parameter n)
{
.....statements;
....}
```

Actual and formal parameters:

→ Actual parameters : When a function is called some parameters are written parenthesis, these are known as actual parameter.

Eg :

```
main()
{
.....
convert(c); //c is actual parameter
.....
}
```

→ formal parameters : Those who are written in function header

for example

```
double convert (int a) /* function header */
{
  //a is formal parameter
}
```

Complete

Jaudesh 12/4