① Define an algorithm and describe its fundamental properties. Also, construct an algorithm to find the roots of a quadratic equation.

A:- Algorithm:- The word Algorithm is a step by step procedure that is helpful in solving a problem.

Fundamental properties of Algorithms:-

1. Well defined inputs:- Algorithms should have clearly defined inputs that are understood and can be processed.
2. Well defined outputs:- Algorithms should produce a predictable and meaningful output for a given input.
3. Finiteness:- Algorithms must terminate after a finite number of steps.
4. Effectiveness:- Complete the task effectively (or) Algorithms should be able to solve the problem.
5. Feasibility:- Algorithms should be practical and implementable with available resources.

Algorithm:

Step 1: Start the program

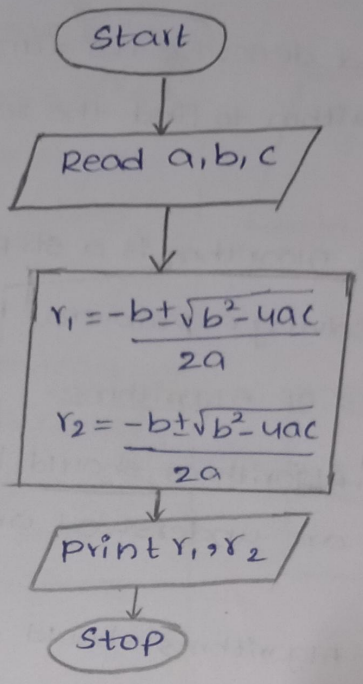
Step 2: Read co-efficient values of a, b, and c.

Step 3: compute roots $r_1 = (-b + sqrt(d))/(2.0*a);$

$r_2 = (-b - sqrt(d))/(2.0*a);$

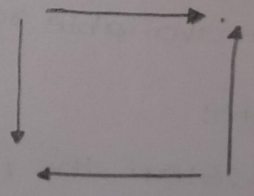
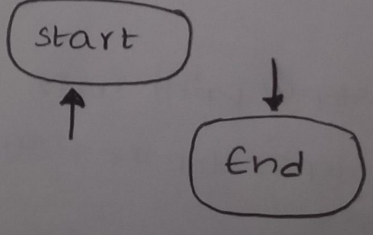
Step 4: print r_1, r_2

Step 5: Stop the program.



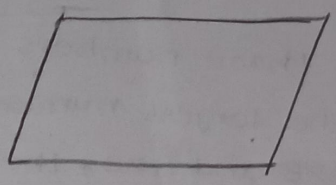
② List and explain various flowchart symbols. Also, design a flowchart to demonstrate the largest and smallest of three numbers.

A:-

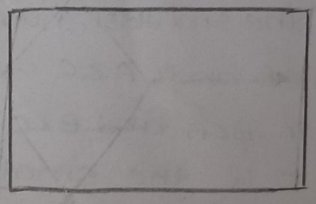
S.NO	Description	Symbols
1	Flowlines: These are the left to right or top to bottom lines connection symbols. These lines shows the flow of control through the program.	
2	Terminal symbol: The oval shaped symbol always begins and ends the flowchart. Every flow chart starting and ending symbol is terminal	

Symbol.

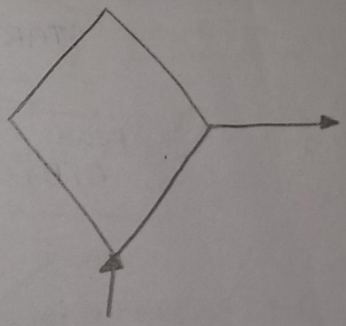
3 Input/Output symbol:
The parallelogram is used for both input (read) and output (write) is called I/O symbol.



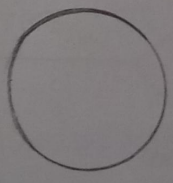
4 process symbol: The rectangle shaped symbol is called process symbol. It is used for calculations and initialization of memory locations



5. Decision symbol: The diamond shaped symbol is called decision symbol. This box is used for decision making. There will be always two exists from a decision symbol One is labeled YES and other labeled NO.

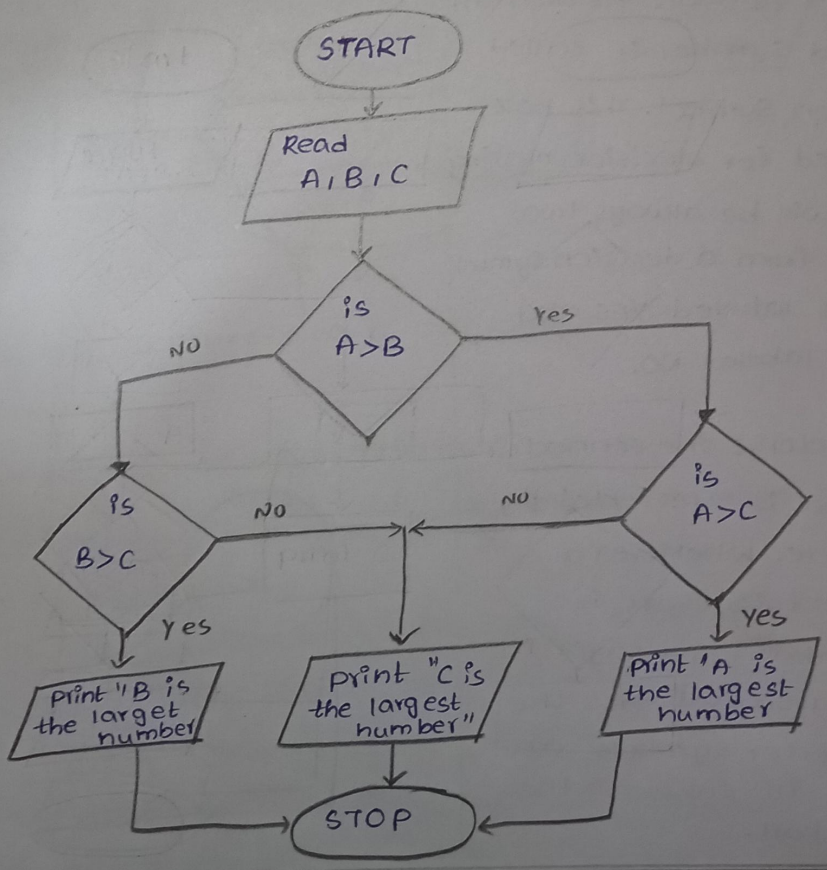


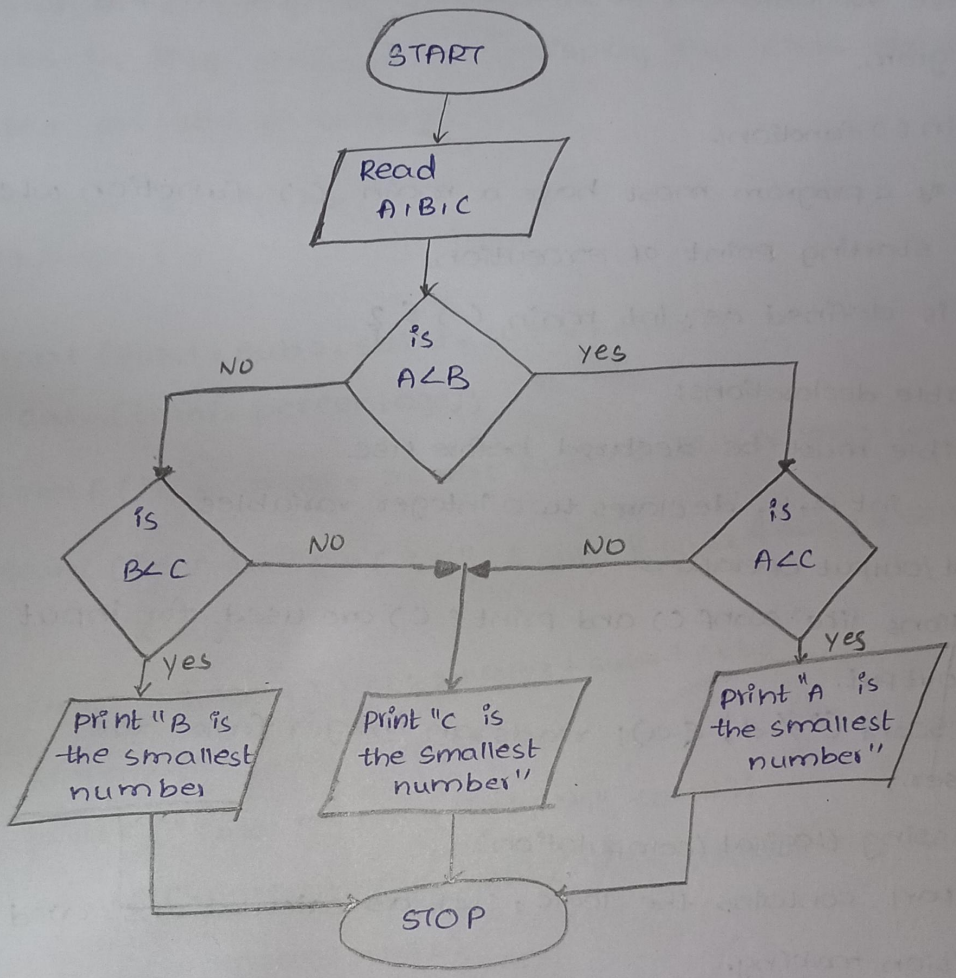
6 Connectors: The connector symbol is represented by a circle. whenever a complex flowchart is morethan one page, in such a situation, the connector symbols are used to connect the flowchart.



Flow chart for finding the largest and smallest of three numbers.

1. start
2. Input three numbers A, B and C
3. Find the largest number:
   * If A > B and A > C, then A is the largest
   * else if B > A and B > C then B is the largest
   * else c is the largest.
4. Find the smallest number:
   * If A < B and A < C then A is the smallest
   * else if B < A and B < C, then B is the smallest
   * else C is the smallest.
5. Output: the largest and smallest numbers
6. end.





(3) Explain the structure of a c program, including its essential components.

A:- 1. preprocessor Directives (#include)

⇒ these are commands given to the preprocessor before compilation begins.

e.g.. #include <stdio.h> ; includes the standard input output library.

2. Global Declarations :- (optional).

⇒ Global variables and function protypes (if needed) are declared here.

⇒ These variable can be accessed anywhere in the Program.

3. Main ( ) function:-

⇒ Every a program most have a main ( ) function which is starting point of execution.

⇒ It is defined as: int main ( ) { }

4. Variable declarations:-

⇒ varible must be declared before use.

⇒ E.g. .. int a, b, declares two integer variables.

5. Input/Output operations:-

⇒ Functions like scanf ( ) and printf ( ) are used for input and output.

e.g.. Scanf ("%d", &a); reads an integer from the user.

6. processing (Logical (computation).

⇒ This part contains the logic, such as calculations and decision making.

e.g:- sum = a+b;

Return statement:-

⇒ The return 0; statement indicates sucessful execution of the program.

⇒ If main ( ) is of type int, if must return an integer value

ex:- #include <stdio.h>

```
int main ( )
{
printf (" Hello world :\n");
return 0;
}
```

(4) Develop a C program to calculate the percentage of marks in five subjects and display the class obtained based on the percentage.

A:-

```
#include <stdio.h>
int main ()
{
float (sub1, sub2, sub3, sub4, sub5);
float (total, percentage);
printf ("enter marks of five subjects (out of 100): \n");
scanf ("%f %f %f %f %f ", &sub1 &sub2 &sub3 &sub4
        &sub5);
total = subject 1 + sub2 + sub3 + sub4 + sub5 ;
percentage = (total /500)* 100;
printf (" total marks = %.2f \n", total);
printf (" percentage = %.2f %.%\n", percentage);
if (percentage >=70)
{
printf ("class: distinction\n" );
}
else if (percentage >=60)
{
printf ("class: First class\n");
}
else if (percentage>= 50) {
printf ("class: second class\n");
}
else if (percentage >=40) {
printf ("class: pass class\n");
}
```

```
else
{
printf("class: fail \n")
}
return 0;
}
```

(5) Write short notes on the following operators in C:
a) Arithmetic operators b) Logical operators c) Bitwise operators d) conditional operators e) Assignment operators

A:- 1. Arithmetic operators:

⇒ Arithmetic operators perform mathematical operations:- ( )

* Addition (+) = Add two operands.
* Subtraction (-) = subtract the second operand from the first ( )
* Multiplication (*) = multiply the two operands.
* Division (/) : Divides the numerator by the denominator
* modulus (%); returns the remainder of division.

e.g:- int a=10; b=3;
int sum=a+b
int difference =a-b
int product =a* b
int division =a\b
int remainter =a% b.

2. Logical operators:-

Logical operators are used to combine conditional statements : ( )

* Logical AND(&&):- returns true if both operators are true ( )
* Logical (OR)(||):- returns true if at least one

Operand is true ( )

* Logical NOT(!) : reverse the logical state of it's operand ( ).

ex-

```
int a = 5 b=10;
if (a>0 && b>0){
}
if (a>0 || b>0){
}
if (!(a>0)){
}
```

c) Bitwise operators:-

⇒ Bitwise operators perform operations on individual bits of integer data types: ( )

* AND (&) - set each bit to 1 if both bits are 1 ( )
* OR (|) - set each bit to 1 if at least one of the bits is 1 ( )
* NOT (~) - Inverts all the bits ( )
* Left shift (<<) - shifts bits to the left by a specified number of positions ( )

ex:-
```
int a=5;
int b=9;
int c= a & b;
int d =a / b;
int e =a ^ b
int f = ~a;
int g=a<<;
int h=b>>;
```

d) Conditional operator:- (ternary)

⇒ The conditional operator (? :) evaluates a condition and returns one of two values based on the result ( )

Ex:- int a=10; b=20;

int max=(a>b) a:b;

e) Assignment operator:-

Assignment operators assign values to variables ( )

* simple Assignment (=): Assigns the right hand operand to the left hand variable

* compound Assignment:- combines an arthemetic bitwise Operation with assignment.

```
int a=10;
a+=5;
a-=3;
a*=2
a/=4;
a%=3;
a&=2
a|=1;
a^=3;
a<<=1;
a>>=2;
```

⑥ Explain the various data types in C with examples.

A:- The various data types in C are:-

1) premitive data types.
2) Derived data types
3) User-Defined data types

1) premitive Basic data type:-

These are the fundamental data types provided by.

int: represents integer numbers ( )

eg:- int age=25;

float:- represent single precision floating point numbers

Eg:- float numbers = 36.6 ; ( )

double:- represents double-precision floating point numbers (shirsha)

ex:- double distance = 12345.6789;

char:- represent, a single character.

ex:- char grade = 'A';

each of these types can have modifies to after their storage size and range ( ).

* short and long:- modify the size of integer types. ( )

ex:- short int small numbers; ( )

ex:- long int large numbers;

* signed and unsigned: Determine whether can variable can hold negative values.

ex:- unsigned int positive number; ( )

2) Derived data types:-

Arrays:- collection of elements of the same type ( )

ex:- int number (10); ( )

pointer:- variables that store memory addresses.

ex:- int* ptr

functions:- Block of code that perform specific tasks and can return values ( )

ex:- int add (int a, int b);

3) User-defined data types:-

these allow programmes to define their own data types.

(structures): Group different data types under a single name

```
structure person {
        char name (50);
```

```
    int age;
    float height;
};
```

Union:- similar to structure but share memory among members.

Ex:- C

```
union data {
    int i;
    float f;
    Char str[20];
};
```

*enumerations:- Define a set of named integer constants.

Ex:- C

```
enum days {sun, mon, tue, wed, thu, fri, sat};
```

(7) Describe the different types of conditional statements in C. Demonstrate their use with examples.

A:- A conditional statement is a programming construct that allows code to make decision based on conditions.

* The most common conditional statements are:-

1. If statement
2. If-else statement
3. If-else-elseif statement
4. Nested if statement
5. switch statement

If statement:-

Excuetes a block of code only if a specific condition is true.

Ex:- Syntax:-

```
#include<stdio.h>
int main(){
int age=20
{
if(age>=20)
{
printf("elegiable for vote");
}
return 0;
}
```

If else statement:-

Excute one block of code if a condition is true and another if the condition is false

Ex:-

```
#include<stdio.h>
int main()
{
int num=5;
if(num % 2==0)
{
printf("even");
}
else
{
print f("odd");
}
return 0;
}
```

If else If statement:-

else if:- Allow for evaluating multiple conditions

Ex:-

```
#include<stdio.h>
int main()
{
int score=75
```

```
if (score >=90)
{
printf ("GradeA");
}
else if (score>=80)
{
print ("Grade B");
}
else if (score>=70)
{
Printf("Grade c");
}
else
{
Print ("Grade f");
}
```

Nested if statement:-

An if else block inside another.

ex:-

```
# include <stdio.h>
int main ()
{
int a=10;
int b=5;
if (a>b)
{
if (a %. 2 == 0)
{
printf ("%d is even", a);
}
else
{
printf ("%d is odd", a);
}
}
else
{
printf ("%d is not greater then %d", a, b);
```

```
}
return 0;
}
```

Switch-case statement:

Alternative for multiple if-else statement based on a single variable

Ex:-
```
#include <stdio.h>
int main()
{
int day = 2;
Switch (day)
{
case 1:
printf("monday\n");
break;
case 2:
printf("tuesday\n");
break;
case 3:
print f("wednesday\n");
break;
case 4;
print f ("thursday\n");
break;
case 5;
printf ("Friday\n");
break;
case 6;
Printf ("saturday\n");
break;
default:
break;
```

```
}
return 0;
}
```

(8) Explain the differences between while and do-while loops, including their syntax and use cases.

A:- 1. while loops:-

A while loop execute a block of code repeatedly as long as condition remains true.

Syntax

```
c
while (condition){
                }
```

Use cases:-

* Repeating tasks with an unknown number at literations of literations (e.g:- reading input until a user enters "exit").
* Running a loop as long as a condition is true.

2. DO-while loops:-

the do-while loop is an exit controlled loop statement. The body of the loop are executed first and then the condition is evaluated.

Syntax:-

```
do{
  statements;
  variable increment or decrement;
  } while (condition);
```

Uses cases:-

* Input validation: ensuring the user enters valid input before proceeding.

* Menu-Driven programs: Displaying a menu and allowing the user to choose an option until they choose to exit.

(9) Define the various storage classes in C with suitable examples. Then, write a C program that demonstrates the use of arithmetic operators through a switch-case construct.

A:- Storage classes in C

Storage classes define the scope, lifetime, and visibility of a variable in C. There are four main storage classes:

1. Automatic (auto) variable
2. Local variable
3. Global variable
4. static variable
5. External variable
6. Register

Automatic storage class in C:-

⇒ The Auto storage class in C is default storage class for local variable.

```
Ex:#include <stdio.h>
    int main ()
    {
    Auto int x=10;
    Printf ("%d", x);
     return 0;
     }
```

Local variable:-

⇒ A local variable, which is a variable declared with in a function.

Ex:-

```
#include <stdio.h>
int main ()
{
int a;
return 0;
}
```

Local

Global variable:-

⇒ A global variable is declared outside any function, making it accessible to all functions within the same program, and its value persists throughout the program's execution.

Ex:-

```
#include <stdio.h>
int a;
int main ()
{
}
```

Static variable:-

⇒ A variable that retains its value between function calls.

Ex:-

```
#include <stdio.h>
int main ()
{
static int count = 0;
count++;
printf ("static variable: %d \n", count);
return 0;
}
```

External storage class:-

The extern signifies that a variable is defined elsewhere and allows access to that variable's value from the current file.

Ex:-

```
#include <stdio.h>
int global var = 100;
int main ()
{
```

```
extern int globalvar;

printf("extern variable: %d\n", globalvar);
return 0;
}
int main()
{extern example();
return 0;
}
```

Register storage classes in C:-

=) suggests that the variable should be stored in CPU registers for faster access.

ex:-
```
#include <stdio.h>
void fast function()
{
register int counter = 5
printf("counter: %d\n", counter);
}
int main()
{
fast function();
return 0;
}
```
output = 5

Arithmetic operations using switch case.

```
#include <stdio.h>
int main()
{
char;
double num1, num2, result;
printf("enter an operator (+, -, *, /):");
scanf("%c", & operator);
```

```
printf("Enter two numbers:");
scanf("%lf %lf",&num1,&num2);
```

switch case

```
switch
{
case '+':
result=num1+num2;
printf("Result: %.2lf\n",result);
break;
case '-':
resut=num1-num2;
printf("Result: %.2lf\n",result);
break;
case '*':
result=num1*num2;
printf("Result: %.2lf\n",result);
break;
case '/':
if(num2!=0){
result=num1/num2;
printf("Result: %.2lf\n",result);
}
else
{
printf("Error: Division by zero is not allowed\n");
}
break;
default:
printf("Invalid operator!\n");
}
return 0;
}
```

(10) Explain the purpose and functionality of stdin, stdout and stderr streams in c programming.

A:- In computing stdin, stdout and stdout, stderr are standard streams used for input and output in command-line interfaces and programming.

1. stdin:-(standard input)
   This is the default input system. typically used to receive input from the keyboard or another input source.
   Ex:- cat file.txt|grep "hello" (stdin receives file.text content)

2. stdout:- (standard output) - this is the default ouput stream, usually used to display normal program output on the terminal, it can be redirected to a file or another command.
   Ex:- ls > output.txt (stdout is redirected to output.txt)

3. stderr:-(standard error):-
   This is the default stream for error messages it helps seperate errors from normal output, making debugging easier.
   ex:- Non-existence-folder 2> error.txt (stderr is redirected to error.txt).