KURIAKOSE ELIAS COLLEGE
MANNANAM

RECORD OF PRACTICAL WORK
M.Sc. PHYSICS

SEMESTER -III

PH010402-COMPUTATIONAL PHYSICS PRACTICAL

NAME:…………………………………………………….

REG.NO:.……………………………………………….

YEAR:……………………………………………………..

Certified that this is a bonafide record of work done by…………………………………
with Reg.No:………………………………during the year…………………………….

Faculty in charge   Head of the Department

Examiners: 1.

2.

Date:………………...

Index

sl no	Date	Name of the experiment	Page no
1		Bisection Method	3
2		Euler’s Method	8
3		Monte Carlo Method	14
4		Projectile Motion	19
5		Simpson’s Rules	26
6		Young’s double slit experiment	36
7		R-K _Method	43
8		Gauss elimination method	48
9		Standing Wave	54
10		Series LCR circuit	58
11		Charging a Capacitor Through Inductance and Resistance	71
12		Trapezoidal rule	80

Experiment No:1
Date:

SOLUTIONS OF NON-LINEAR EQUATIONS - BISECTION METHOD

AIM:

To write a C++ program for finding the root using bisection method.

THEORY:

The bisection method is one of the simplest and most reliable iterative methods for the solutions of non linear equations. This method is also known as binary chopping or half interval method, relies on the fact that if f(x) is real and continuous in the interval *a* <x< *b* and f(a) and f(b) are of opposite signs i.e. f(a)f(b)<0, then there is at least one real root in the interval between *a* and *b.*

Let x_1=*a* and x_2=*b*. Let x_0 be the midpoint between *a* and *b*.

$$\text{i.e. } x_0 = \frac{x_1+x_2}{2}$$

Now there exists the following three conditions

1. If f(x_0)=0, we have a root at x_0
2. If f(x_0)f(x_1)<0, there is a root between x_0 and x_1
3. If f(x_0)f(x_2)<0, there is a root between x_0 and x_2

It follows that by testing the sign of the function at the midpoint , we can deduce which part of interval contains the root.

CONVERGENCE OF BISECTION METHOD:

In the bisection method, we choose a midpoint x_0 in the interval between x_1 and x_2 .Depending on the sign of functions f(x_0),f(x_1) and f(x_2), x_1 or x_2 is set equal to x_0 such that the new interval contains the root .In either case ,the interval containing the root is reduced by a factor of 2 .the same procedure is repeated for the new interval. If the procedure is repeated n times, then the interval containing the root is reduced to the size.

$$\frac{x_2 - x_1}{2^n} = \frac{\Delta x}{2^n}$$

After n iterations, the root must lie within $\pm \Delta x / 2^n$ of our estimate. This means that the error bound at the n^{th} iteration is

$$E_n = \left| \Delta x / 2^n \right|$$

Similarly

$$E_{n+1} = \left| \Delta x / 2^{n+1} \right| = \left| E_n / 2 \right|$$

That is the error decrease linearly with each step by a factor of 0.5.

The bisection method is therefore linearly convergent. Since the convergence is slow to achieve a high degree of accuracy, a large number of iterations may be needed. However, the bisection algorithm is guaranteed to converge.

ALGORITHM:

```
Step 1: Start.
Step 2: Decide initial values for $x_1$, $x_2$,$f(x_1)$ and N.
Step 3: Compute $f_1=f(x_1)$ and $f_2=f(x_2)$ and set i=0.
Step 4: If $f_1f_2>0$, $x_1$ and $x_2$ do not bracket any root and go to step 8.Otherwise continue.
Step 5: Compute $x_0 = \frac{x_1+x_2}{2}$ and $f_0=f(x_0)$.
Step 6: If $f_1f_0<0$, set $x_2=x_0$ , $f_2=f_0$ else set $x_1=x_0$ , $f_1=f_0$ .
Step 7: If i>N, then root $\left(\frac{x_1+x_2}{2}\right)$ .Write the value of root and go to step 8, else i=i+1, go to step 5
Step 8: Stop.
```

PROGRAM:

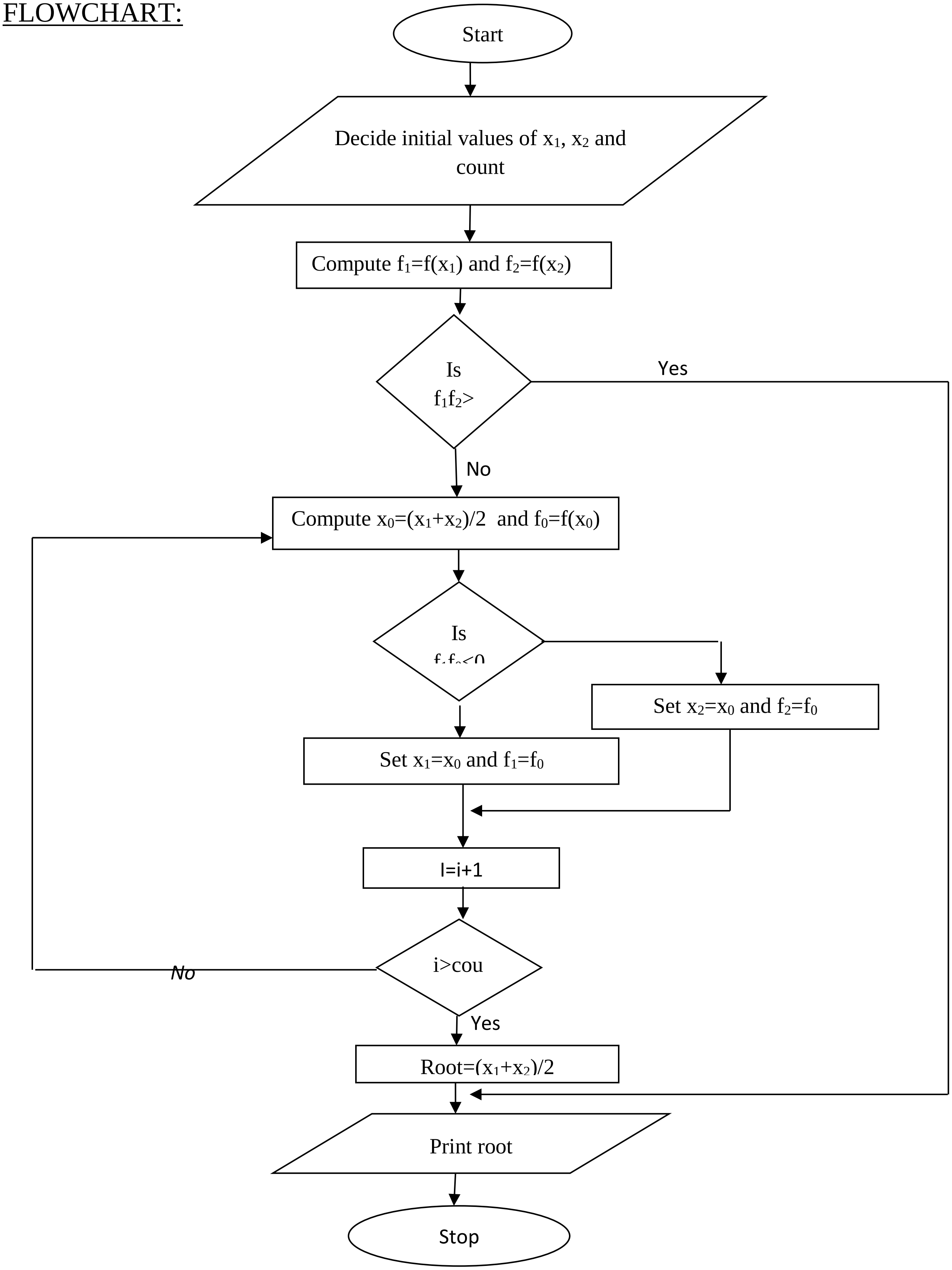
```
#include<iostream.h>
#include<conio.h>
#include<math.h>
//#define F(x)(2*x*x*)+6*x-5
#define F(x)(x*x-25)
class bisect
{
private:
int s,count;
float a,b,root;
public:
bisect();
void display();
void solution();
};
bisect::bisect()
{
cout<<"Enter the initial values"<<endl;
cout<<"a= ";
cin>>a;
cout<<"b= ";
cin>>b;
cout<<"Enter the number of iterations: ";
cin>>count;
```

```
}
void bisect::display()
{
if(s==0)
{
cout<<"the initial values do not bracket any root";
}
else
{
cout<<"root= "<<root;
cout<<"\n F(root)= "<<F(root);
}
}
void bisect::solution()
{
float x1,x2,x0,f0,f1,f2;
x1=a;
x2=b;
f1=F(x1);
f2=F(x2);
if(f1*f2>0)
{
s=0;
}
else
{
for(int i=0;i<count;i++)
{
x0=(x1+x2)/2;
f0=F(x0);
if(f0==0)
{
s=1;
root=x0;
}
if(f1*f0<0)
{
x2=x0;
f2=f0;
}
else
{
x1=x0;
f1=f0;
}
root=(x1+x2)/2;
}
}
}
void main()
{
clrscr();
bisect bis;
bis.solution();
bis.display();
getch();
}
```

RESULT:

A C++ program to find the root of an Non linear equation using bisection method is entered, compiled and executed.

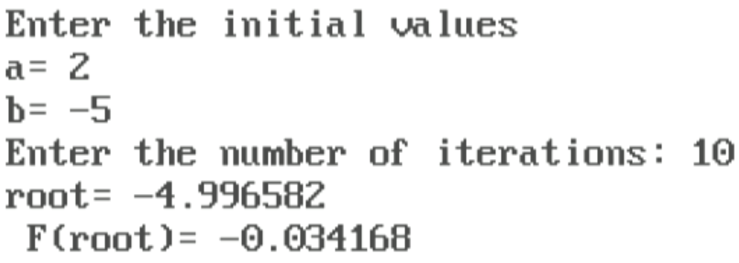
FLOWCHART:



Start
Decide initial values of x_1, x_2 and count
Compute $f_1=f(x_1)$ and $f_2=f(x_2)$
Is $f_1f_2>$
Yes
No
Compute $x_0=(x_1+x_2)/2$ and $f_0=f(x_0)$
Is $f_1f_0<0$
Set $x_2=x_0$ and $f_2=f_0$
Set $x_1=x_0$ and $f_1=f_0$
I=i+1
i>cou
No
Yes
Root=$(x_1+x_2)/2$
Print root
Stop

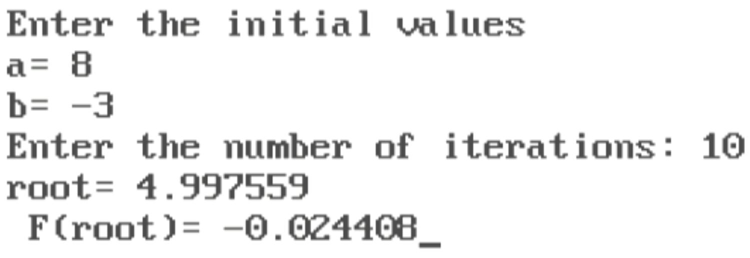
OUTPUT:

Case1:



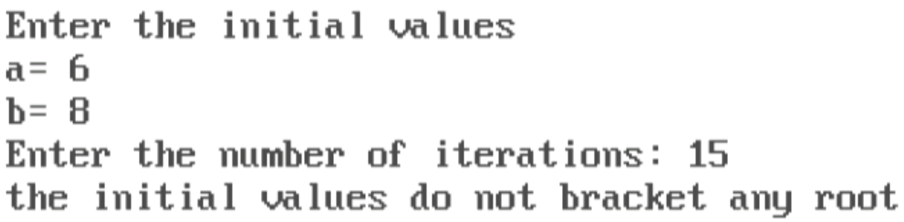
```
Enter the initial values
a= 2
b= -5
Enter the number of iterations: 10
root= -4.996582
 F(root)= -0.034168
```

Case2:



```
Enter the initial values
a= 8
b= -3
Enter the number of iterations: 10
root= 4.997559
 F(root)= -0.024408_
```

Case3:



```
Enter the initial values
a= 6
b= 8
Enter the number of iterations: 15
the initial values do not bracket any root
```

Experiment No:2
Date:

EULER'S METHOD

AIM:

To find the solution of a first order differential equation using Euler's method.

THEORY:

In solving a first order differential equation by numerical methods , we come across two types of solutions:

a) a series solution of y in terms of x, which will yield the value of y at a particular value of x by direct subscription in the series solution.

b) values of y at specified values of x.

The two methods due to Taylor Picord studied earlier belong to the first category and following methods due to Euler, Runga-Kutta come under second category.

The methods of second category called step-by-step methods because the values of y are calculated by short steps ahead of equal interval h of the independent variable x.

To solve dy/dx= f(x, y) with initial condition y(x_0)=y_0………………………..(1)

Let us take the points x=$x_{0,}x_1$,x_2,etc. where x_i -x_{i-1}=h

That is x_i=x_0+ih , i=0,1,2…………

Let the actual solution of the differential equation be denoted by the graph{continues line graph}$P_0(x_0, y_0)$ lies on the curve. we require the value of y of the curve at x=x_1

The equation of tangent at (x_0, y_0) to the curve is

$$\text{y-}y_{0=}y^{'}(x_0, y_0)(x - x_0)$$

$$=\text{f}\ (x_0, y_0).(x - x_0)$$

$$\text{Y=}y_0{+}f(x_0 y_0).(x - x_0)$$

This is the value of y on the tangent corresponding to x=x_0.In the interval (x_0, y_0),the curve is approximately bisected by the tangent. Therefore, the value of y on the curve is approximately equal to the value of y on the tangent at (x_0, y_0) corresponding to x=x_1.

$$y_1 = y_0 = f(x_0, y_0)(x_1, x_0)$$

That is $\quad y_1 = y_0 + hy_0 \quad$ where h=$x_1 - x_0$

$$(m_1 p_1 = m_1 q_1)$$

Again , we approximate curve by the line through $(x_{1,}y_1)$ and whose slope is f(x_1, y_1) we get

$$y_2 = y_1 + hf(x_1, y_1) = y_1 + hy_1$$

Thus $\quad \boldsymbol{Y_{n+1} = Y_n + hf(x_n, y_n)}$**; n=0,1,2,3…………**

This formula is called **EULER'S ALGORITHAM**

ALGORITHM:

```
Step 1:  Enter initial values of x and y.
Step 2:  Enter the value of $x_n$.
Step 3:  Enter the variation, h.
Step 4:  Calculate the number of iteration  n= ($x_n$-x)/h+0.5.
Step 5:  Compute dy by using dy =h*Func(x,y)
Step 6:  Increment x by h and y by dy.
Step 7:  Display the values of x and y
Step 8:  Repeat step 5 until number of iterations become n.
Step 9:  Stop the process.
```

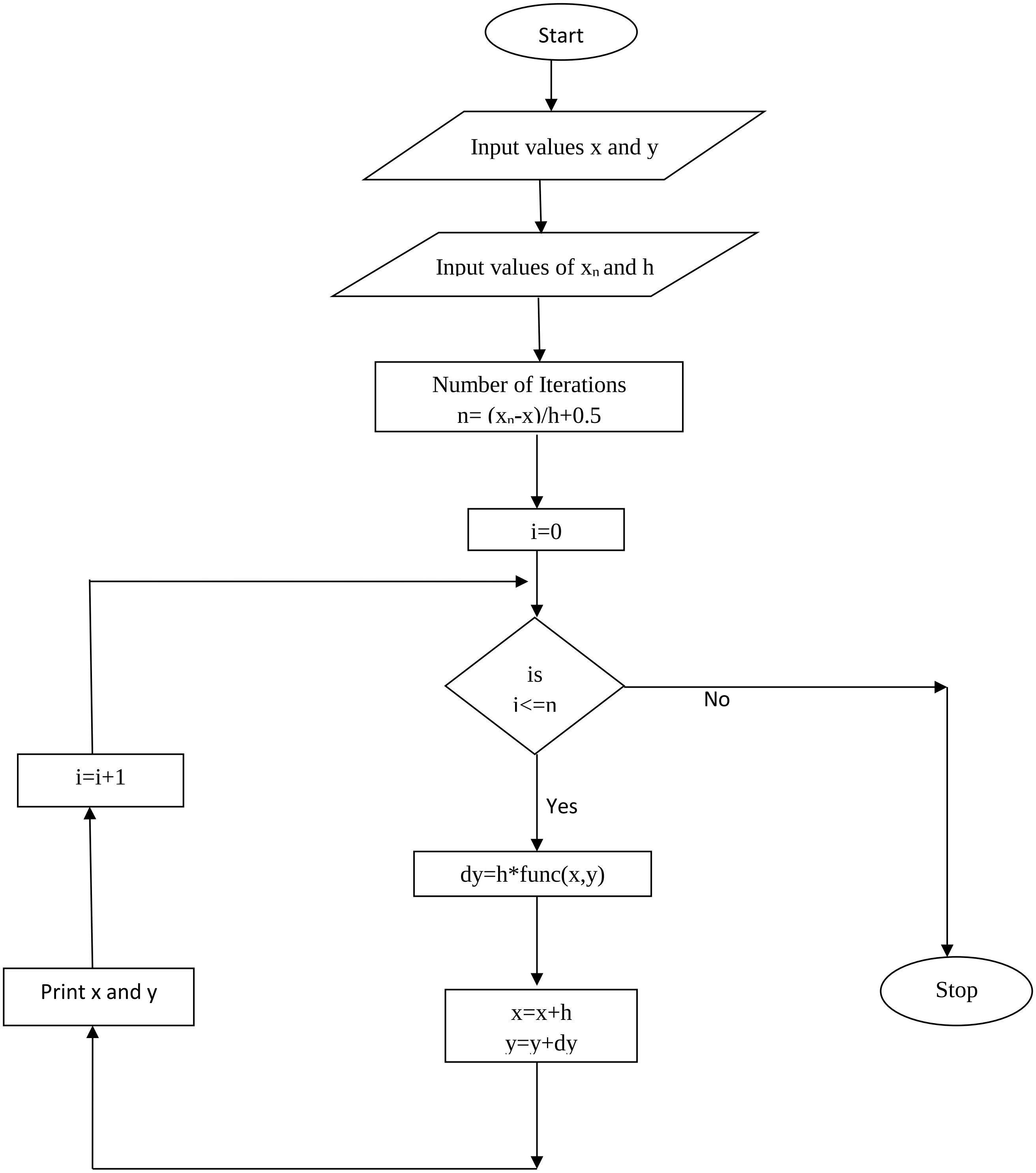
PROGRAM:

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
class ODiffEQ
{
double x,y,xn,h;
public:
void getvalues();
double func(double,double);
void calc();
};
void ODiffEQ::getvalues()
{
cout<<"\n Enter the initial value of x:";
cin>>x;
cout<<"\n Enter the initial values of y:";
cin>>y;
cout<<"\n Enter the value of xn:";
cin>>xn;
cout<<"\n Enter the variation of h:";
cin>>h;
}
double ODiffEQ::func(double x,double y)
{
return(x+y);
}
void ODiffEQ::calc()
{
int n,i;
double dy;
n=(int)(xn-x)/h+0.5;
cout<<"\n Iteration\tx\ty";
cout<<"\n-------------------------\n";
for(i=1;i<=n;i++)
{
dy=h*func(x,y);
x=x+h;
y=y+dy;
cout<<"\n\t"<<i<<"\t"<<x<<"\t"<<y;
}
cout<<"\n\n Values of y at"<<xn<<"="<<y;
}
void main()
{
clrscr();
ODiffEQ diff;
diff.getvalues();
diff.calc();
getch();
}
```

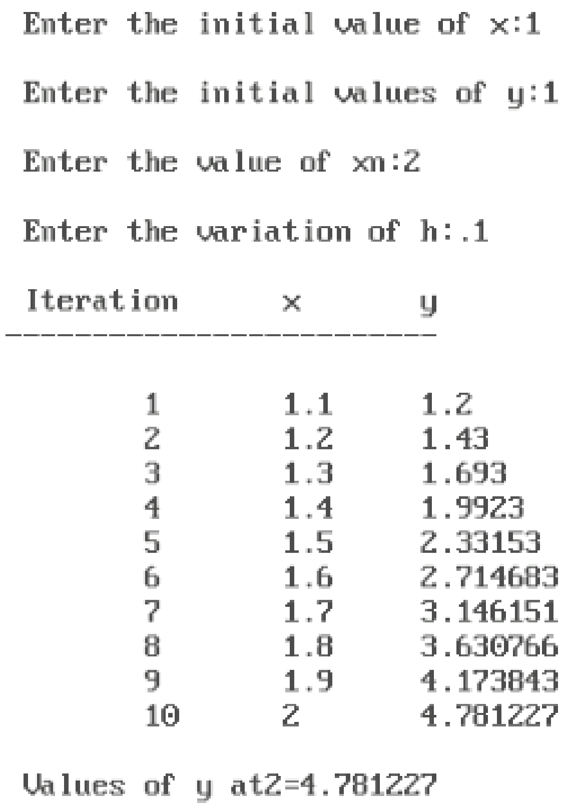
RESULT:

A C++ program to find the root of a first order differential equation using Euler’s method is entered, compiled and executed.

FLOW CHART:



OUTPUT:



```
Enter the initial value of x:1

Enter the initial values of y:1

Enter the value of xn:2

Enter the variation of h:.1

 Iteration      x       y
-------------------------


       1       1.1     1.2
       2       1.2     1.43
       3       1.3     1.693
       4       1.4     1.9923
       5       1.5     2.33153
       6       1.6     2.714683
       7       1.7     3.146151
       8       1.8     3.630766
       9       1.9     4.173843
       10      2       4.781227

Values of y at2=4.781227
```

Experiment No:3
Date:

MONTE CARLO METHOD

AIM:

To write and execute a C++ program to calculate the volume of an ellipsoid by Monte Carlo method.

THEORY:

Monte Carlo method is the most convenient method to calculate higher dimension integrals. This method predicts results based on random sampling and is statistical in nature. To, understand it imagine we have to find the area of an irregularly shaped plane, enclosing it within a square of known area. Hang it in a board and ask a shooter to shoot at it. Randomly count the total number of shots lying within the irregular figure. Assuming the unbiased probability of hitting, we obtain the following relation,

$$\frac{area\ of\ figure}{area\ of\ squre} = \frac{Number\ of\ shots\ falling\ in\ figure}{total\ no\ of\ shots}$$

The greater the no. of shots, the higher the accuracy of the shots. Such probabilistic method can be generated through random numbers. The method is statistical in nature, we may calculate the statistical measure of error in one computation. One such a measure of error is called standard deviation. The basic idea is to repeat the measurement many times keeping the number of shots fixed , instead of measuring a value once. If Mi is the i^{th} measurement and N is the number of times such measurements are made, then the calculation of the mean of the measurement and the mean of their square variance is given by,

$$\sigma^2 = (\Sigma Mi/N)^2 - (\Sigma Mi^2/N)$$

Square root of this quantity is called standard deviation. It is a measure of how much a given measurement can deviate for a fixed number of shots in an interval(M-σ ,M+σ),where M is the mean of measurement. M=ΣMi/N. We see that $\sigma\ \alpha\ 1/\sqrt{N}$ that is, the greater the number of measurements, the more precisely one can locate the true value. We enclose ellipsoid in a cuboids of volume abc and calculate the number of shots falling within it.

ALGORITHM:

```
Step 1:   Start
Step 2:   Input the number of measurements (m), number of trials (N), semi axes (a, b, c).
Step 3:   Initialize sum Vo=0, sum Vo Sa =0, count =0, i=0.
Step 4:   If  I <= N, continue, otherwise go to the step 10.
Step 5:   Calculate x, y,  z.
Step 6:   If   $x^2/a^2 + y^2/b^2 + z^2/c^2 <= 1$, continue otherwise, calculate I =I+1 &  go to  step4
Step7:    Calculate count =count +1,  Vol  = (8abc*count)/n, sum Vo=  sum Vo+ Vol,
           sum $V_{Sa}$= Vo Sa + $(Vol)^2$
```

Step 8: Calculate I=I+1 and got o step 4.

Step 9: Calculate avg. volume = Sumv_o/m,

$\sigma = sqrt(m * sum\ v_0 sa - sumv_0\ sumv_0)/m$.

Exact volume =$4\pi abc/3$

Step10: Display calculated volume, standard deviation and exact volume.

Step11: Stop.

PROGRAM:

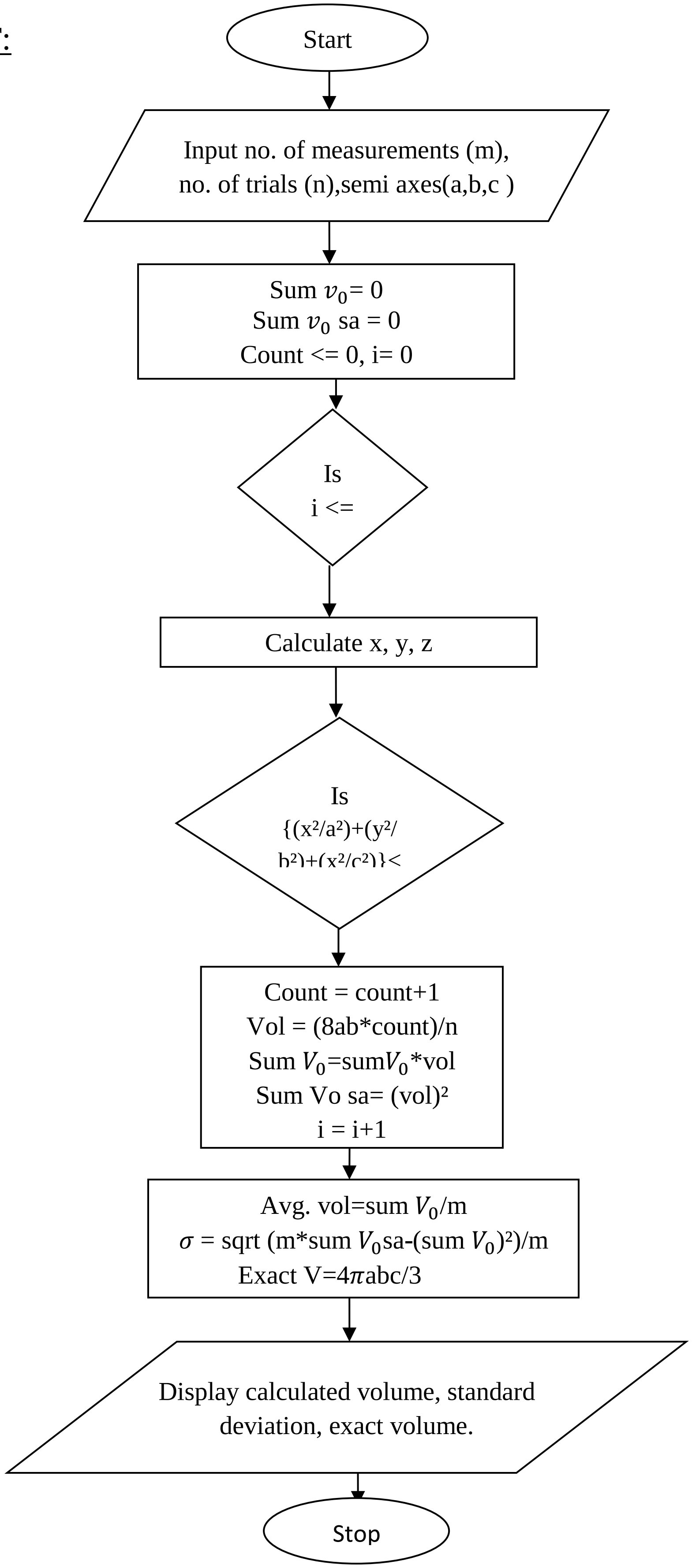
```
#include<iostream.h>
#include<conio.h>
#include<iomanip.h>
#include<stdlib.h>
#include<time.h>
#include<math.h>
const pi=3.1415;
class mont
{
private:
int m,n;
float a,b,c;
public:
mont();
void calculation();
};
mont::mont()
{
cout<<"number of measurements\n";
cin>>m;
cout<<"number of trials\n";
cin>>n;
cout<<"three semi axes\n";
cin>>a>>b>>c;
}
void mont::calculation()
{
randomize();
float sumv0=0,sumv0sq=0,x,y,z,count,avvol,exactv0,sigma,vol;
for(int j=1;j<=m;j++)
{
count=0;
for(int i=1;i<=n;i++)
{
```

```
x=a*rand()/(float)RAND_MAX;
y=b*rand()/(float)RAND_MAX;
z=c*rand()/(float)RAND_MAX;
if((x/a)*(x/a)+(y/b)*(y/b)+(z/c)*(z/c)<=1)
count=count+1;
}
vol=8*a*b*c*count/n;
sumv0+=vol;
sumv0sq+=(vol)*(vol);
}
avvol=sumv0/m;
exactv0=4*pi*a*b*c/3;
sigma=sqrt(m*sumv0sq-sumv0*sumv0)/m;
cout<<"calculated volume\n"<<avvol;
cout<<" standard deviation\n"<<sigma;
cout<<" exact volume\n"<<exactv0;
getch();
}
void main()
{
clrscr();
mont amount;
amount.calculation();
getch();
}
```

RESULT:

The volume of an ellipsoid was found out using Monte Carlo method by executing a C++ program.

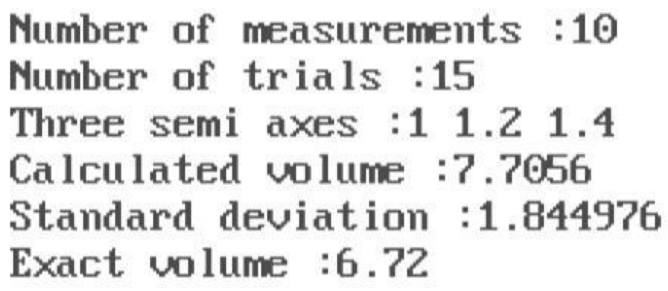
FLOWCHART:



Start
Input no. of measurements (m),
no. of trials (n),semi axes(a,b,c)
Sum v_0= 0
Sum v_0 sa = 0
Count <= 0, i= 0
Is
i <=
Calculate x, y, z
Is
$\{(x^2/a^2)+(y^2/$
$b^2)+(x^2/c^2)\}<$
Count = count+1
Vol = (8ab*count)/n
Sum V_0=sumV_0*vol
Sum Vo sa= $(vol)^2$
i = i+1
Avg. vol=sum V_0/m
σ = sqrt (m*sum V_0sa-(sum $V_0)^2$)/m
Exact V=4πabc/3
Display calculated volume, standard
deviation, exact volume.
Stop

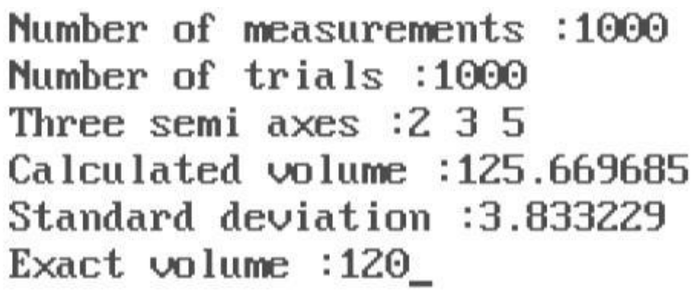
OUTPUT:

Case 1:



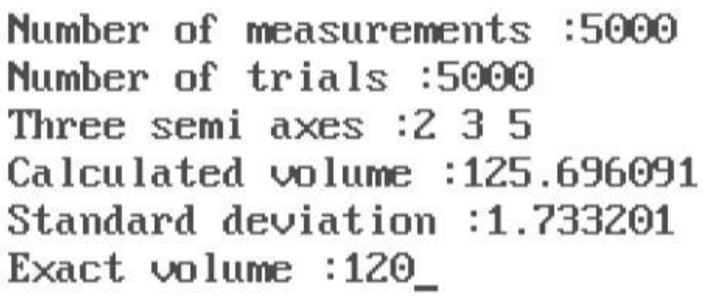
```
Number of measurements :10
Number of trials :15
Three semi axes :1 1.2 1.4
Calculated volume :7.7056
Standard deviation :1.844976
Exact volume :6.72
```

Case 2:



```
Number of measurements :1000
Number of trials :1000
Three semi axes :2 3 5
Calculated volume :125.669685
Standard deviation :3.833229
Exact volume :120_
```

Case 3:



```
Number of measurements :5000
Number of trials :5000
Three semi axes :2 3 5
Calculated volume :125.696091
Standard deviation :1.733201
Exact volume :120_
```

Experiment No:4
Date:

PROJECTILE MOTION

AIM:

To write and execute a C++ program to describe the projectile motion.

THEORY:

If a freely falling body is given an initial velocity in a direction different form the vertical, then it does not follow a straight line path, rather it follows a projectile path. Consider a body thrown at some an angle θ with an initial velocity v_0.The body will go up first and eventually fall back.

When a projectile moving through air, a resistance medium, height and horizontal range are reduced. Air drag decreases the magnitudes of both the components of velocity. The equation of motion is,

$$\text{m}\frac{dv_x}{dt} = -F_d \cos\emptyset$$

and

$$\text{m}\frac{dv_y}{dt} = -mg - F_d \cos\emptyset$$

Where F_d is the damping force due to the air drag. The angle $\emptyset$ is determined as

$$\cos\emptyset = \frac{v_x}{v}, \quad \sin\emptyset = \frac{v_y}{v}.$$

$$\text{Where } v^2 = v_x^2 + v_y^2.$$

Resistive force F_d is

$$F_d = \frac{1}{2} D\rho A v^2$$

Where D is the drag coefficient , A is the area of cross section of the body and ρ is density of air.

ALGORITHM:

Step 1: Start the program.

Step 2: Give the value of p_i (Π), d, DC, and g.

Step 3: Define a class Projectile.

Step 4: create an object P in the above class.

Step 5: Initialize the time as zero.

Step 6: Give the initial value of the mass m, radius r, maximum time T, initial value of velocity v_0, angle of projectile on θ and step h.

Step 7: Compute the no. of steps n=$^{T}/_{h}$.

Step 8: Compute the x and y component of velocity.

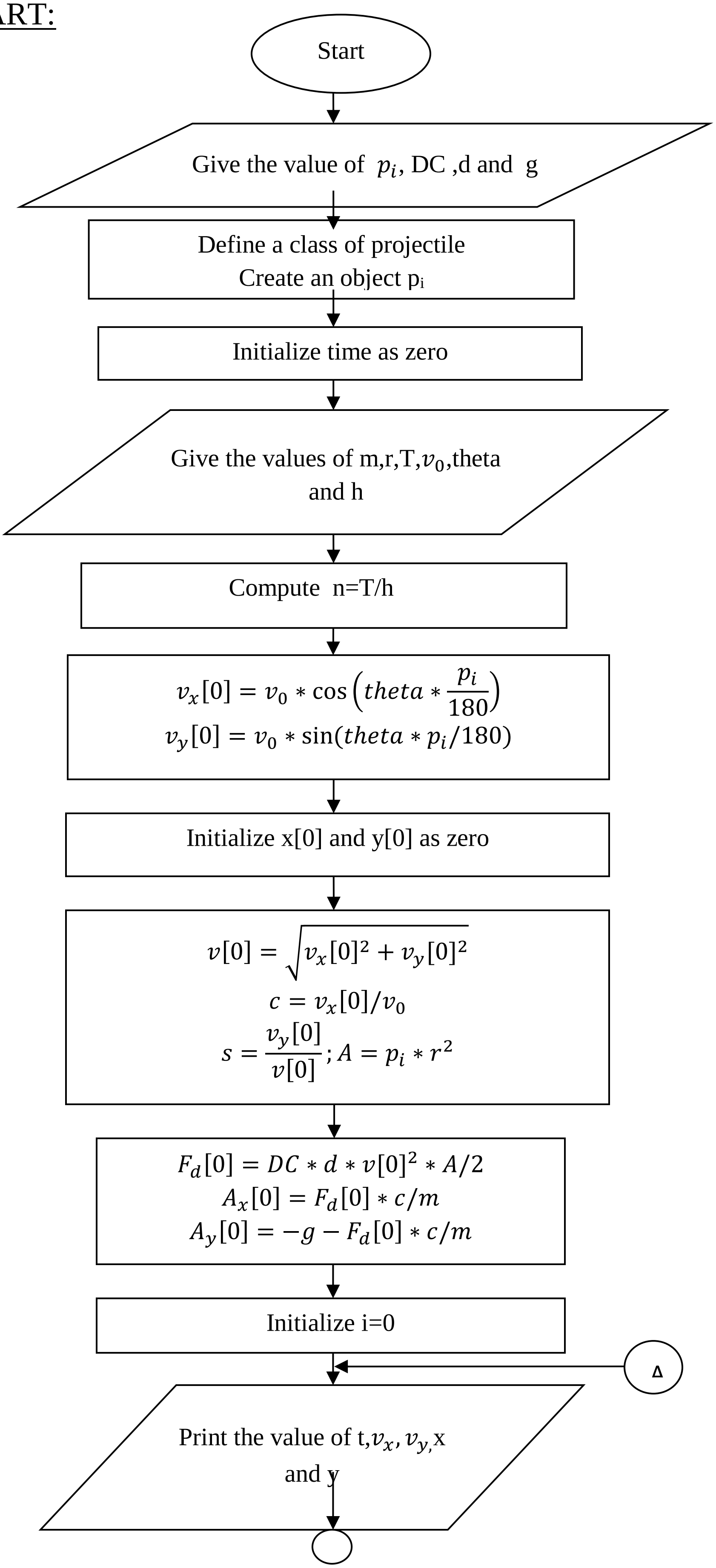
$$v_x[0] = v_0 * \text{Cos}(theta * p_i/180).$$
$$v_y[0] = v_0 * \text{Sin}(theta * p_i/180)$$

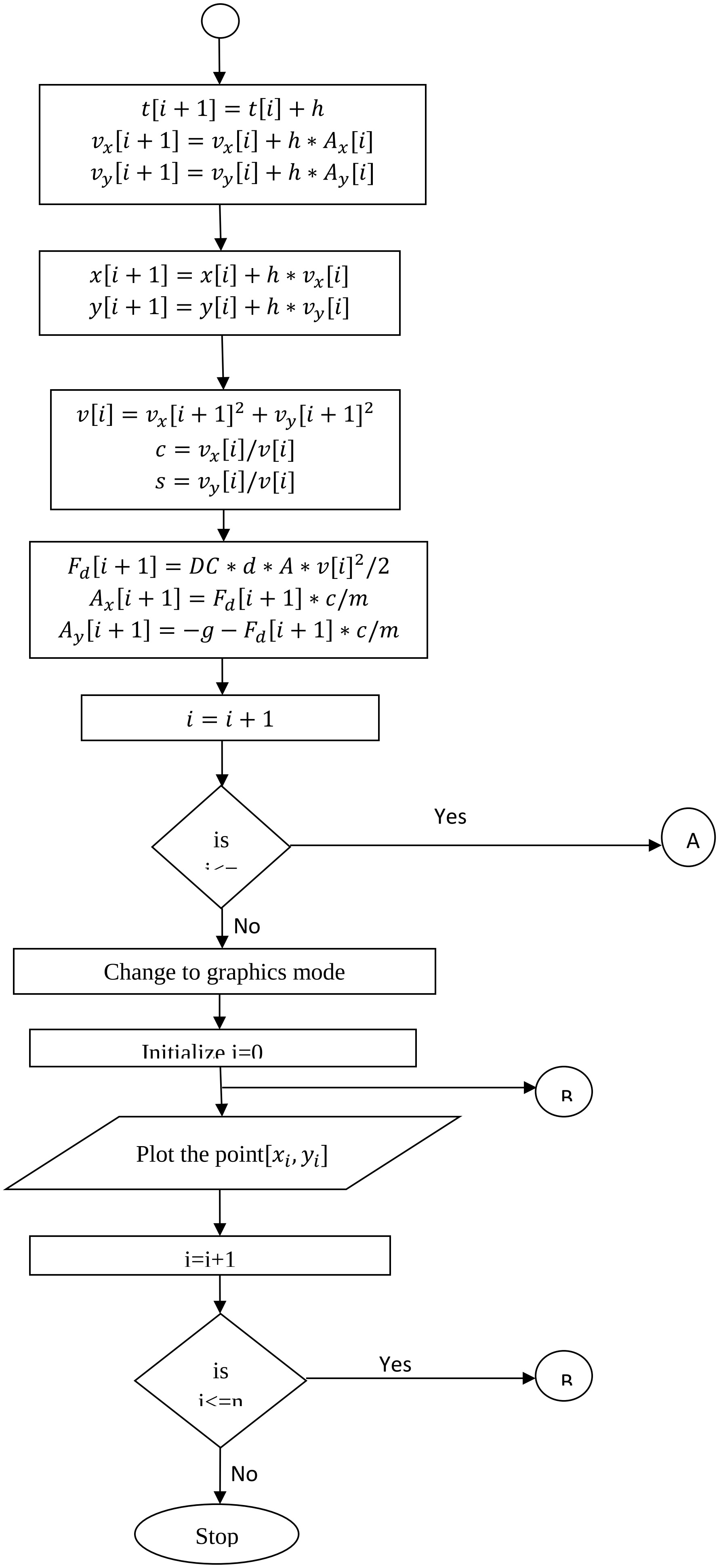
```
Step 9:   Initialize x [0] and y [0] as zero.
Step 10:  Compute v[0]= $\sqrt{v_x[0]^2 + v_y[0]^2}$.
              $c = {}^{v_x[0]}\!/_{v[0]}$.
          $s = {}^{v_y[0]}\!/_{v[0]}$
area A=$p_i * r^2$

Step 11:  Compute
          $F_d[0] = (DC * d * v[0]^2 * A)/2$
Step 12:  Compute $A_x[0] = F_d[0] * {}^{c}\!/_{m}$, $A_y[0] = -g - F_d[0] * {}^{c}\!/_{m}$.
Step 13:  Repeat the steps 14-18 varying i from zero to n incrementing by 1.
Step 14:  Print the value of t[i],x[i],y[i],$v_x[i], v_y[i]$.
Step 15:  Compute t[i+1] = t[i]+h.
              $v_x[i+1] = v_x[i] + h * A_x[i]$
          $v_y[i+1] = v_y[i] + h * A_y[i]$
Step 16:  Compute $x[i+1] = x[i] + h * v_x[i]; y[i+1] = y[i] + h * v_y[i]$
Step 17:  Compute $v[i] = (v_x[i+1])^2 + (v_y[i+1])^2$
              $c = {}^{v_x[i+1]}\!/_{v[i]}$
          $s = {}^{v_y[i+1]}\!/_{v[i]}$
Step 18:  Compute $F_d[i+1] = (DC * d * A * v[i]^2)/2$
              $A_x[i+1] = F_d[i+1] * c/m$
          $A_y[i+1] = -g - F_d[i+1] * c/m$
Step 19:  Change to graphics mode.
Step 20:  Plot the point (x[i],y[i]) which i<=n.
Step 21:  Program ends.
```

PROGRAM:

```
#include<iostream.h>
#include<iomanip.h>
#include<math.h>
#include<conio.h>
#include<graphics.h>
#include<dos.h>
#include<fstream.h>
const float pi=3.1415926,d=1.2,DC=.46,g=9.8;
class projectile
{
private:
float m,r,T,h,Fd[1000],v0,teta,s,n;
float x[1000],y[1000],vx[1000],vy[1000],ax[1000],ay[1000],v[1000],t[1000];
public:
projectile()
{
t[0]=0;
}
void getdata();
void calculate();
void graphics();
};
void projectile::getdata()
{
clrscr();
cout<<"Enter the value of mass,radius,tmax"<<endl;
cin>>m>>r>>T;
cout<<"Enter the initial value of velocity,Angle of projectile and time step:\n";
cin>>v0>>teta>>h;
}
void projectile::calculate()
{
clrscr();
n=T/h;
vx[0]=v0*cos(teta*pi/180);
vy[0]=v0*sin(teta*pi/180);
x[0]=0;
y[0]=0;
v[0]=sqrt((vx[0]*vx[0])+(vy[0]*vy[0]));
float c=vx[0]/v[0];
s=vy[0]/v[0];
float a=pi*r*r;
Fd[0]=(DC*d*v[0]*v[0]*a)/2;
ax[0]=Fd[0]*c/m;
ay[0]=-g-(Fd[0]*c/m);
ofstream outfile("seena.txt");
outfile<<setw(10)<<"t"<<setw(16)<<"x"<<setw(16)<<"y"<<setw(10)<<"vx";
outfile<<setw(16)<<"vy";
for(int i=0;i<n;i++)
{
outfile<<setw(10)<<t[i]<<setw(16)<<x[i]<<setw(16)<<y[i]<<setw(10)<<vx[i];
outfile<<setw(16)<<vy[i]<<endl;
t[i+1]=t[i]+h;
vx[i+1]=vx[i]+(h*ax[i]);
vy[i+1]=vy[i]+(h*ay[i]);
x[i+1]=x[i]+(h*vx[i]);
y[i+1]=y[i]+(h*vy[i]);
v[i]=sqrt((vx[i+1]*v[i+1])+(vy[i+1]*vy[i+1]));
c=vx[i+1]/v[i];
s=vy[i+1]/v[i];
Fd[i+1]=(DC*d*a*v[i]*v[i])/2;
ax[i+1]=-(Fd[i+1]*c)/m;
ay[i+1]=-g-((Fd[i+1]*c)/m);
}
getch();
}
void projectile::graphics()
{
int driver,mode;
driver=DETECT;
initgraph(&driver,&mode,"c:\\tc\\bgi");
setbkcolor(BLACK);
setcolor(WHITE);
line(10,400,701,400);
line(320,10,320,400);
settextstyle(4,0,3);
outtextxy(500,420,"x------>");
settextstyle(4,1,3);
outtextxy(290,20,"y------>");
for(int i=0;i<n;i++)
{
putpixel(320+x[i]*30,400-y[i]*90,WHITE);
putpixel(320-x[i]*30,400-y[i]*90,WHITE);
delay(30);
}
getch();
}
void main()
{
projectile p;
clrscr();
p.getdata();
p.calculate();
p.graphics();
getch();
closegraph();
}
```

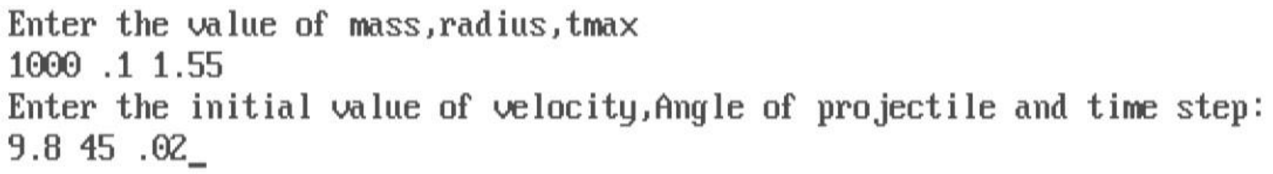
FLOW CHART:


Start
Give the value of p_i, DC ,d and g
Define a class of projectile
Create an object p$_i$
Initialize time as zero
Give the values of m,r,T,v_0,theta and h
Compute n=T/h
$v_x[0] = v_0 * \cos\left(theta * \frac{p_i}{180}\right)$
$v_y[0] = v_0 * \sin(theta * p_i/180)$
Initialize x[0] and y[0] as zero
$v[0] = \sqrt{v_x[0]^2 + v_y[0]^2}$
$c = v_x[0]/v_0$
$s = \frac{v_y[0]}{v[0]}; A = p_i * r^2$
$F_d[0] = DC * d * v[0]^2 * A/2$
$A_x[0] = F_d[0] * c/m$
$A_y[0] = -g - F_d[0] * c/m$
Initialize i=0
Δ
Print the value of t,$v_x, v_{y,}$x and y

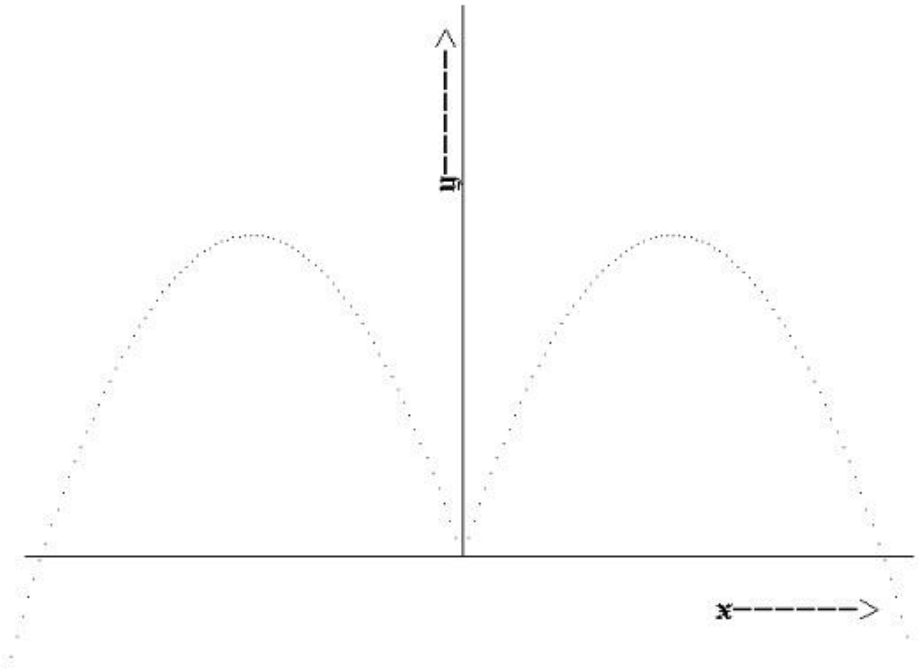


$t[i+1] = t[i] + h$
$v_x[i+1] = v_x[i] + h * A_x[i]$
$v_y[i+1] = v_y[i] + h * A_y[i]$
$x[i+1] = x[i] + h * v_x[i]$
$y[i+1] = y[i] + h * v_y[i]$
$v[i] = v_x[i+1]^2 + v_y[i+1]^2$
$c = v_x[i]/v[i]$
$s = v_y[i]/v[i]$
$F_d[i+1] = DC * d * A * v[i]^2/2$
$A_x[i+1] = F_d[i+1] * c/m$
$A_y[i+1] = -g - F_d[i+1] * c/m$
$i = i + 1$
is
Yes
A
No
Change to graphics mode
Initialize i=0
B
Plot the point$[x_i, y_i]$
i=i+1
is
i<=n
Yes
B
No
Stop

OUTPUT:



```
Enter the value of mass,radius,tmax
1000 .1 1.55
Enter the initial value of velocity,Angle of projectile and time step:
9.8 45 .02_
```



RESULT:

A C++ program for describing the projectile motion of a body is written and executed. The path of the projectile is plotted.

Experiment No:5
Date:

NUMERICAL INTEGRATION USING SIMPSONS 1/3 AND 3/8 RULE

AIM:

To write and execute a C++ program for numerical integration of a given function using Simpson's $^1/_3$ and $^3/_8$ rule.

THEORY:

In numerical analysis, Simpson's rule is a method for numerical integration , the numerical approximation of definite integrals. Simpsons rule is a Newton-cotes formula for approximating the integral of a function and using quadratic polynomials i.e. Parabolic arcs instead of the straight line segments.

Assume the integration of function f(x) as follows

$$\text{I} = \int_a^b f(x)dx \qquad \longrightarrow (1)$$

Let the interval [a,b] be divide in to n equal subintervals such that

$$\text{a}=x_0 < x_1 < x_2 < \cdots \ldots \ldots \ldots \ldots .. < x_n = b$$

Clearly $x_n = x_0 + nh$ .Hence the integral becomes

$$\text{I} = \int_{x_0}^{x_n} f(x)dx \qquad \longrightarrow (2)$$

Approximating $f(x)$ by Newton's forward difference formula we get the general formula

$$\int_{x_0}^{x_n} f(x)dx = nh[f(x_0) + \frac{n}{2}\Delta f(x_0) + \frac{n(2n-3)}{12}\Delta^2 y_0 + \cdots \ldots \ldots \ldots \ldots \ldots] \qquad \longrightarrow (3)$$

Simpsons 1/3 Rule.

This rule is obtained by putting n=2 in equation (3) i.e. by replacing the curve by $\frac{n}{2}$ arcs of second degree polynomials or parabolas. We have then

$$\int_{x_0}^{x_2} f(x)dx = 2h\left[f(x_0) + \Delta f(x_0) + \frac{1}{6}\Delta^2 f(x_0)\right]$$

$$=\frac{h}{3}(f(x_0) + 4f(x_1) + f(x_2))$$

Similarly,

$$\int_{x_2}^{x_4} f(x)dx = \frac{h}{3}[f(x_2) + 4f(x_3) + f(x_4)]$$

And finally

$$\int_{x_{n-2}}^{x_n} f(x)dx = \frac{h}{3}[f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)]$$

Summing up we obtain

$$\int_{x_0}^{x_n} f(x)dx = \frac{h}{3}[f(x_0) + 4\big(f(x_1) + f(x_3) + \cdots \ldots \ldots \ldots \ldots + f(x_{n-1})\big)$$
$$+2[f(x_2) + f(x_4) + \cdots \ldots \ldots \ldots \ldots . f(x_{n-2})] + f(x_n)]$$

which is known as Simpson’s $\frac{1}{3}$ rule or simply Simpsons rule . It should be noted that this rule requires the division of the whole range of even numbers of subintervals of width h.

Simpsons 3/8 Rule.

Setting n=3 in equation (3). We get,

$$\int_{x_0}^{x_3} f(x)dx = \frac{3h}{8}[f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)]$$

Similarly,

$$\int_{x_3}^{x_6} f(x)dx = \frac{3h}{8}[f(x_3) + 3f(x_4) + 3f(x_5) + f(x_6)]$$

Summing up all these we obtain

$$\int_{x_0}^{x_n} f(x)dx = \frac{3h}{8}[f(x_0) + 3f(x_1) + 3f(x_2) + 2f(x_3) + 3f(x_4) + 3f(x_5) + 2f(x_6)$$
$$+ \cdots \ldots \ldots \ldots + f(x_n)]$$

This rile is called Simpsons $^3/_8$ rule and is not so accurate as Simpsons rule.

ALGORITHM: Simpsons 1/3 rule.

```
Step 1:   Start the program.
Step 2:   Input the values of upper limit $x_f$,lower limit $x_i$, number of the interval n and the
          function.
Step 3:   Calculate the step size h=$\frac{x_i - x_f}{n}$ and sum 1 =f($x_i$)+4f($x_i + h) + f\left(x_f\right)$.
Step 4:   Initialize sum 2 = 0.
Step 5:   If j<=n, continue, otherwise go to step 7.
Step 6:   Calculate.
          Term = $2f(x_i + jh) + 4f(x_i + (j+1)h)$
          Sum 2 = sum 2 + term.
          J = J+2, go to STEP:5.
Step 7:   Calculate S = sum 1 + sum 2; Integral = $\frac{hs}{3}$.
Step 8:   Print the value of the integral.
Step 9:   Stop.
```

ALGORITHM : Simpsons 3/8 Rule.

```
Step 1:    Start the program.
Step 2:    Input the value of lower limit $x_i$,upper limit $x_f$, no. of intervals n and the function.
Step 3:    Calculate step size, h=$\frac{x_f-x_i}{n}$ and
                   Sum 1 = $f(x_i) + 3f(x_i + h) + 3f(x_i + 2h) + f(x_f)$.
Step 4:    Initialize Sum 2 = 0.
Step 5:    If j<=n, Continue. Otherwise go to step 7.
Step 6:    Calculate
             Term = $2f(x_i + jh) + 3f(x_i + (j + 1)h) + 3f(x_i + (j + 2)h)$
           Sum 2 = sum2+term
           J = J+3, go to Step: 5 .
Step 7:    Calculate S = sum 1 + sum 2 ; Integral = $\frac{3hs}{8}$.
Step 8:    Print the value of the integral.
Step 9:    Stop.
```

PROGRAM:

SIMPSON 1/3 RULE

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
#include<iomanip.h>
class simpson
{
float xi,xf,h,n,s,sum1,sum2,fun(float),term,i;
public:
void getdata();
void calculation();
};
void simpson::getdata()
{
cout<<"\nINTEGRATION USING SIMPON 1/3 RULE\n";
cout<<"\nEnter the no.of intervel:";
cin>>n;
cout<<"\nEnter the lower limit:";
cin>>xi;
cout<<"\nEnter the upper limit:";
cin>>xf;
}
float simpson::fun(float x)
```

```
{
return(1/(1+x));//return(2*x*x+1);
}
void simpson::calculation()
{
h=(xf-xi)/n;
sum1=fun(xi)+4*fun(xi+h)+fun(xf);
int q=1;
sum2=0;
for(int j=2;j<=n;j+=2)
{
term=2*fun(xi+(j*h))+4*fun(xi+(j+1)*h);
sum2=sum2+term;
q=q+1;
}
s=sum1+sum2;
i=h*s/3;
cout<<"\nThe value fo the integral is :"<<i<<"\n";
cout<<"\nThe no.of iteration is :"<<q<<"\n";
}
void main()
{
clrscr();
simpson s;
s.getdata();
s.calculation();
getch();
}
```

SIMPSON 3/8 RULE

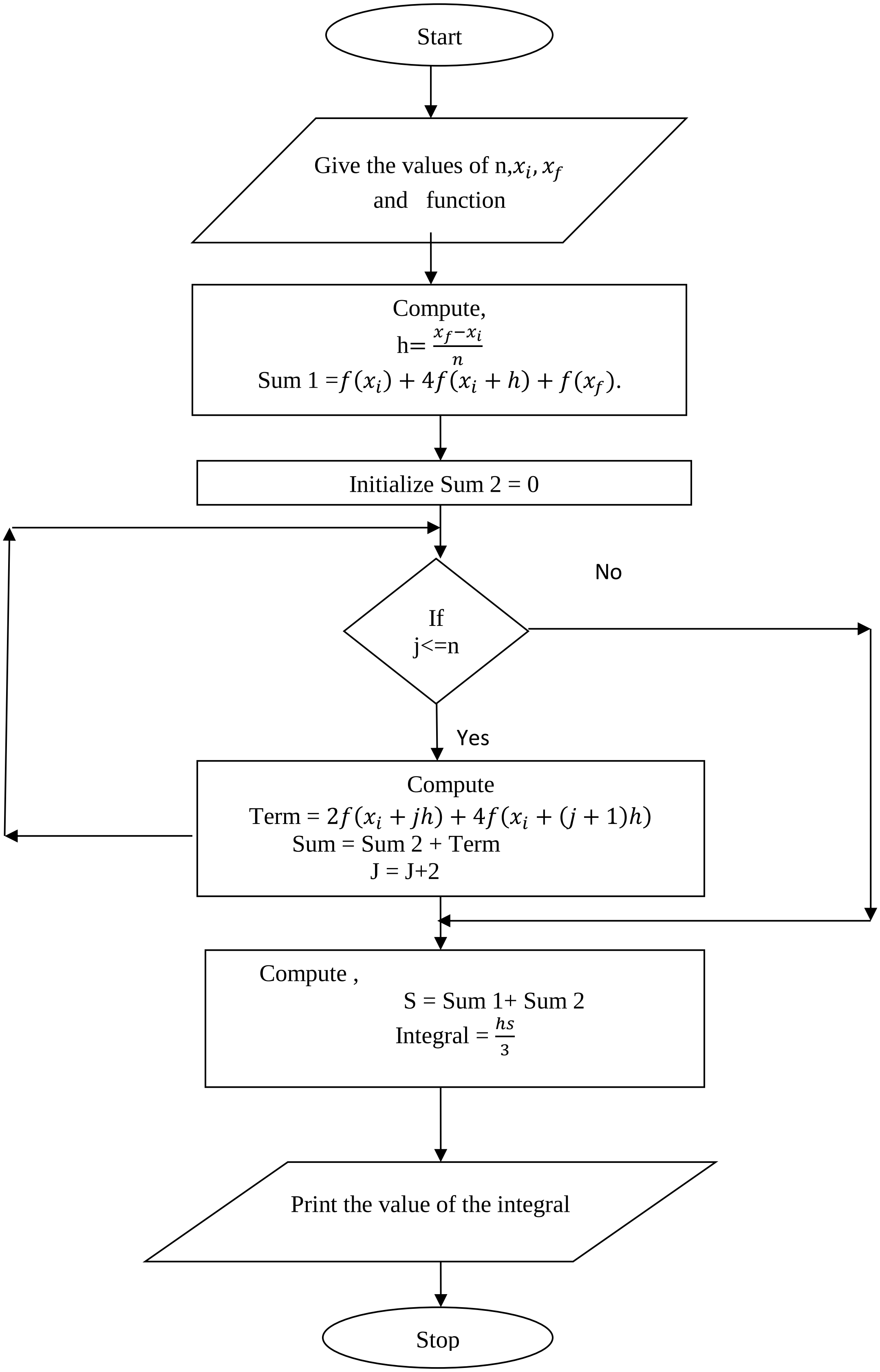
```
#include<iostream.h>
#include<conio.h>
#include<math.h>
#include<iomanip.h>
class simpson
{
float xi,xf,h,n,s,sum1,sum2,fun(float),term,i;
public:
void getdata();
void calculation();
};
```

```
void simpson::getdata()
{
cout<<"\nINTEGRATION USING SIMPSON 3/8 RULE";
cout<<"\n Enter the no.of intervels :";
cin>>n;
cout<<"\nEnter the lower limit :";
cin>>xi;
cout<<"\nEnter the upper limit :";
cin>>xf;
}
float simpson::fun(float x)
{
return(1/(1+x));//return(2*x*x*+1);
}
void simpson::calculation()
{
h=(xf-xi)/n;
sum1=fun(xi)+3*fun(xi+h)+3*fun(xi+(2*h))+fun(xf);
int q=1;
sum2=0;
for(int j=3;j<=n;j+=3)
{
term=2*fun(xi+(j*h))+3*fun(xi+(j+1)*h)+3*fun(xi+(j+2)*h);
sum2=sum2+term;
q=q+1;
}
s=sum1+sum2;
i=3*h*s/8;
cout<<"\nThe value of the integral is :"<<i<<"\n";
cout<<"\nThe no of iteration is :"<<q<<"\n";
}
void main()
{
clrscr();
simpson s;
s.getdata();
s.calculation();
getch();
}
```

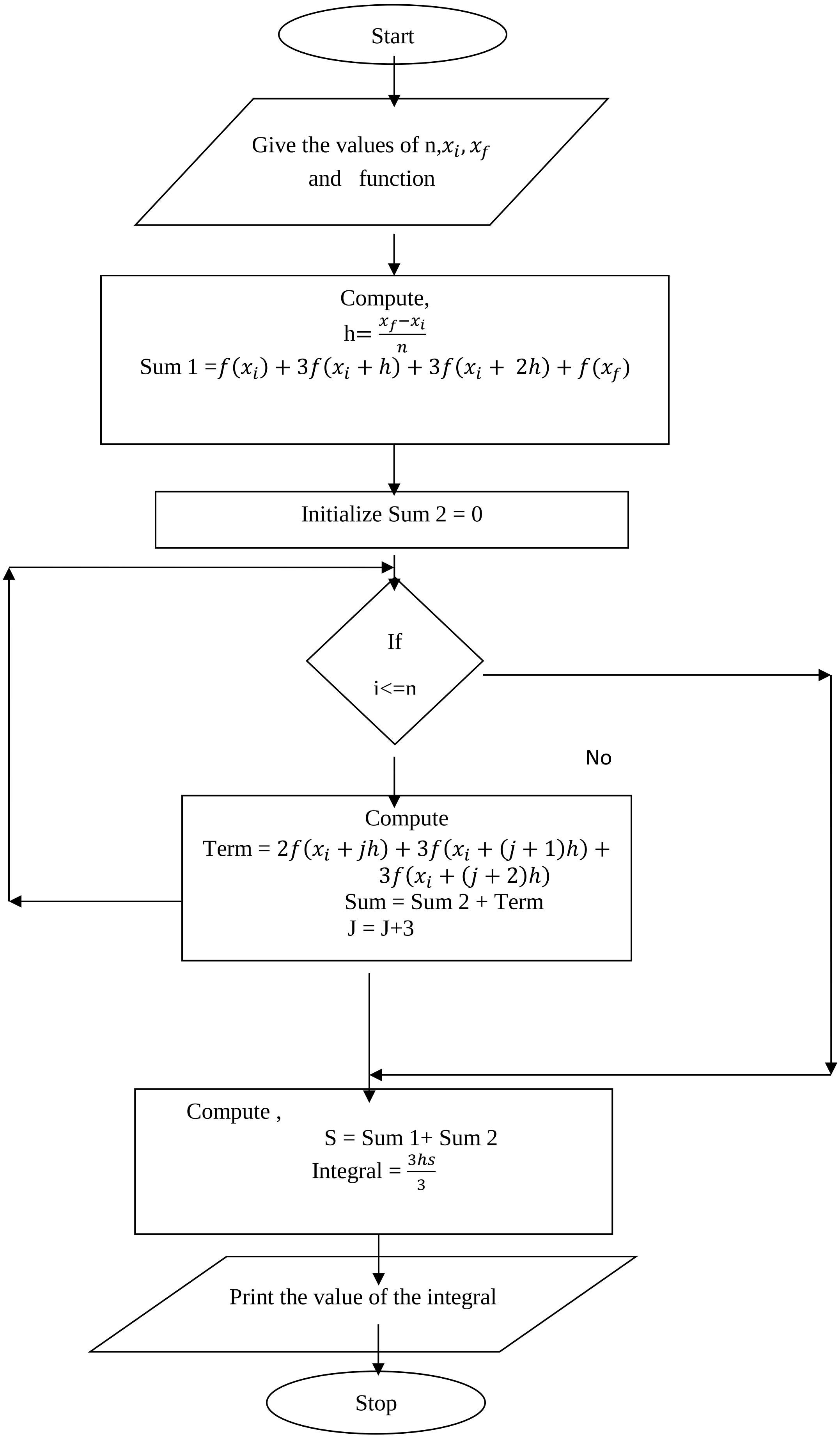
RESULT:

A C++ program for numerical integration of a given function using Simpsons ${}^{1}/_{3}$ and ${}^{3}/_{8}$ rule is written and executed.

Flow Chart: Simpson’s 1/3 Rule:

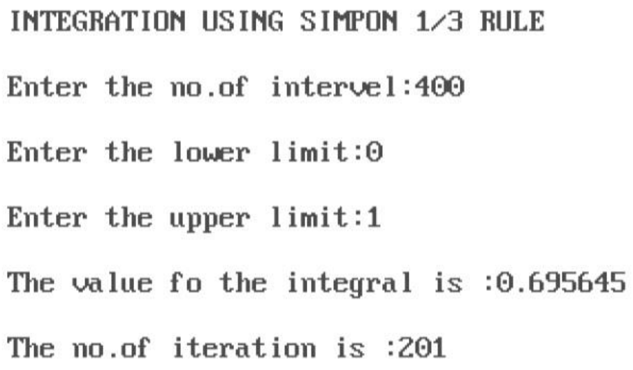

Start
Give the values of n,x_i, x_f and function
Compute,
h= $\frac{x_f - x_i}{n}$
Sum 1 =$f(x_i) + 4f(x_i + h) + f(x_f)$.
Initialize Sum 2 = 0
No
If j<=n
Yes
Compute
Term = $2f(x_i + jh) + 4f(x_i + (j + 1)h)$
Sum = Sum 2 + Term
J = J+2
Compute ,
S = Sum 1+ Sum 2
Integral = $\frac{hs}{3}$
Print the value of the integral
Stop

3/8 rule


Start
Give the values of n,x_i, x_f
and function
Compute,
h= $\frac{x_f - x_i}{n}$
Sum 1 =$f(x_i) + 3f(x_i + h) + 3f(x_i + 2h) + f(x_f)$
Initialize Sum 2 = 0
If
i<=n
No
Compute
Term = $2f(x_i + jh) + 3f(x_i + (j+1)h) +$
$3f(x_i + (j+2)h)$
Sum = Sum 2 + Term
J = J+3
Compute ,
S = Sum 1+ Sum 2
Integral = $\frac{3hs}{3}$
Print the value of the integral
Stop

OUT PUT:

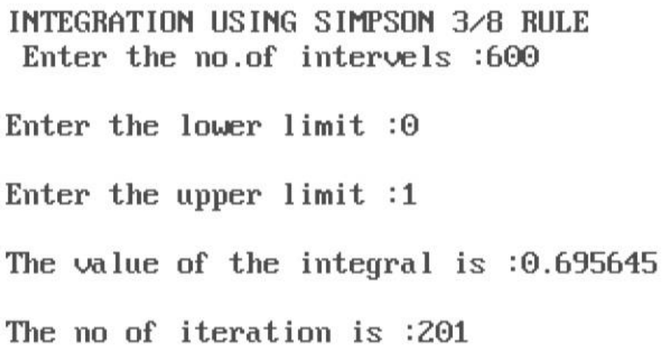
SIMPSON 1/3 RULE



```
INTEGRATION USING SIMPON 1/3 RULE

Enter the no.of intervel:400

Enter the lower limit:0

Enter the upper limit:1

The value fo the integral is :0.695645

The no.of iteration is :201
```

OUT PUT:

SIMPSON 3/8 RULE



```
INTEGRATION USING SIMPSON 3/8 RULE
 Enter the no.of intervels :600

Enter the lower limit :0

Enter the upper limit :1

The value of the integral is :0.695645

The no of iteration is :201
```

Experiment No:6
Date:

YOUNGS DOUBLE SLIT EXPERIMENT

AIM:

To write and execute a C++ program demonstrating the young’s double slit experiment for the generation of interference pattern and also change the distance of the screen from the slit and Study the variation of intensity with distance.

THEORY:

In young’s double slit experiment a monochromatic point source of light emitting spherical wave front is placed in front of the screen with two pin holes. These two holes act as two coherent source of light. At a distance D a screen is placed. Since the holes are point sources, light emitted from each source is spherical wave.

$$E(r,t) = \frac{A}{r}\cos(kr-wt+\phi) \quad \text{-------------(1)}$$

At any point P on the screen the total electric field is got in accordance with the principle of superposition.

$$E = E(r1,t)+E(r2,t)$$

$$=\frac{A}{r1}\cos(kr1-wt+\phi)+\frac{A}{r2}\cos(kr2-wt) \quad \text{------------(2)}$$

These two points on the screen where the phase of the two waves are identical, the two amplitudes get added along loading the maximum intensity points called maxima. When the phase difference is an odd integral multiple of , the two amplitudes get substracted. These points where the intensity is reduced to minimum are known as minima. These maxima and minima all together is called interference fringes.

For finding the time averaged intensity we must consider each point on the screen. Let r1,r2 be the distance over which light travels from the slit s1 and s2 to reach a point.

$$r1=(D^2+(y-y1)^2)^{\wedge}½$$

$$r2=(D^2+(y-y2)^2)^{\wedge}½$$

Where (x1,y1),(x,y) and (x2,y2) are co ordinates of the point the equation (2) gives the resultant amplitude at the point P. Repeating it over time steps in a cycle. We can calculate the average of the intensity at various points on the screen.

$$I_{avg} = \int_0^t \sum_i \frac{|Ei|^2 dt}{\int_0^t dt} = \frac{\sum_i \sum_n |Ei|^2 dt}{\sum_i t} = \frac{1}{n}\sum_i \sum_n |\,Ei\,|^2$$

Where n is the number of steps taken in one cycle.

ALGORITHM:

```
Step 1:   Start
Step 2:   Give the values of π, wavelength,amplitude,screen length,slit spacing and time
Step 3:   Define function (amp* cos(vx-wt)/x+ cos (vy-wy)/y
Step 4:   Compute h,Iam,v,w,e1,y,x2,y2,q,x
Step 5:   Initialize values of x,ymin,zmin,ys1,ys2.
Step 6:   Compute zmax,xscale,yscale.
Step 7:   Repeat steps 8-15 varying I from 0 to n incrementing by 1.
Step 8:   Compute y,r1,r2
Step 9:   Initialize values of int s.
Step10:  Repeat step n varying t from 1 to n time incrementing by1
Step 11: Compute amp=fn & (r1,r2t) and increment the int s=int s+amp*amp
Step 12: Compute int s=int s/n time
Step 13: Compute y2 and z for drawing the pattern
Step 14: Draw the line from (x1,ys1) to (x2,y2) and (x,ys2) to (x2,y2)
Step 15: Draw line from (x2,y2) to (z,y2)
Step 16: Stop
```

PROGRAM:

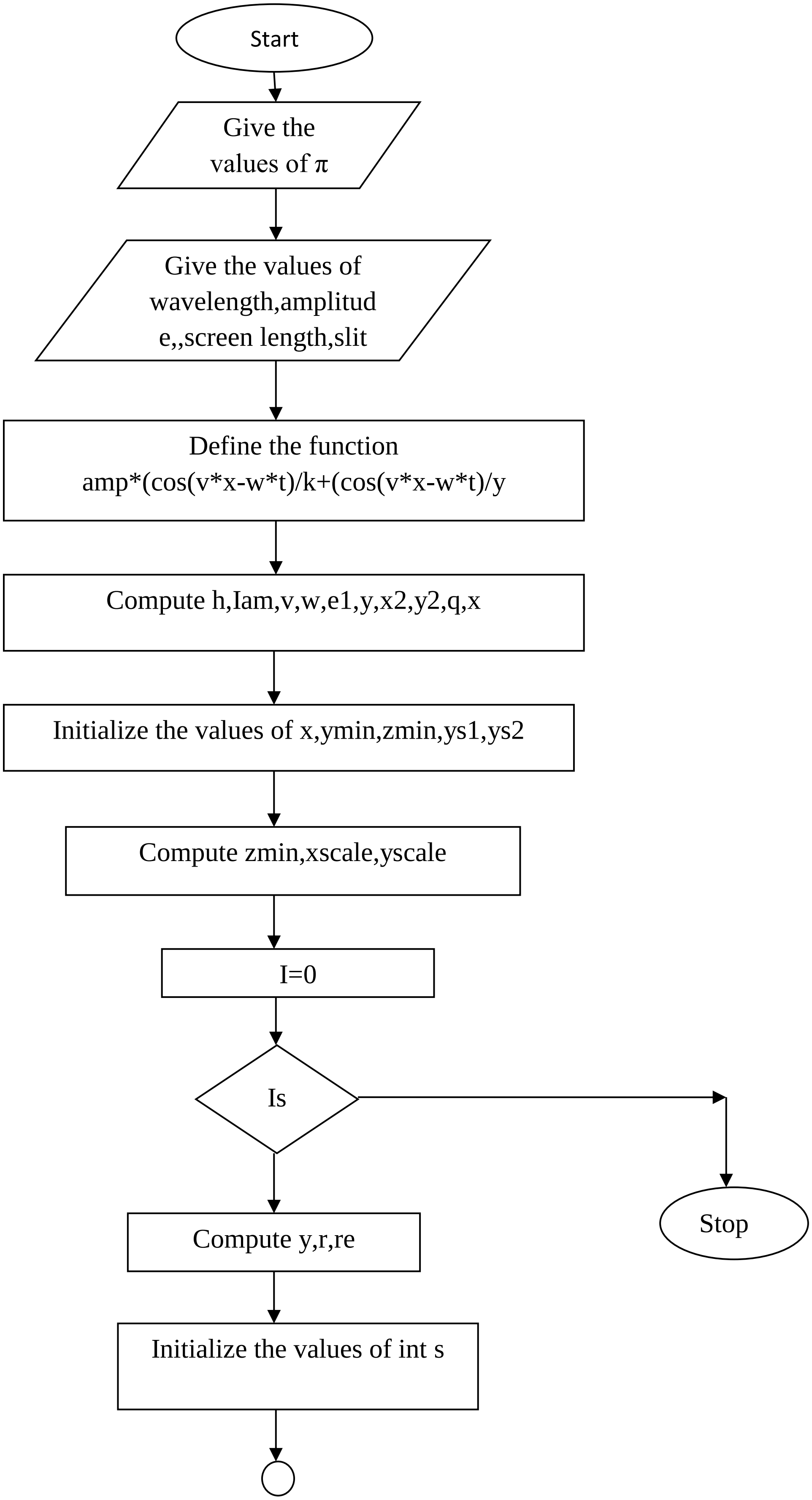
```
 #include<fstream.h>
#include<conio.h>
#include<math.h>
#include<iomanip.h>
#include<graphics.h>
#include<dos.h>
const pi=3.141592;
class slit
{
double iam;
float amp,ymax,d,s1,n,ntime,h;
public:
float v,w;
void getdata();
double fnf(float,flat,float);
void show();
};
void slit::getdata()
{
cout<<"enter wavelength,amplitude,screen length\n";
cin>>iam>>amp>>ymax;
cout<<"enter distance from slit and screen\n";
cin>>d;
cout<<"slit spacing, no of points,time step\n";
```

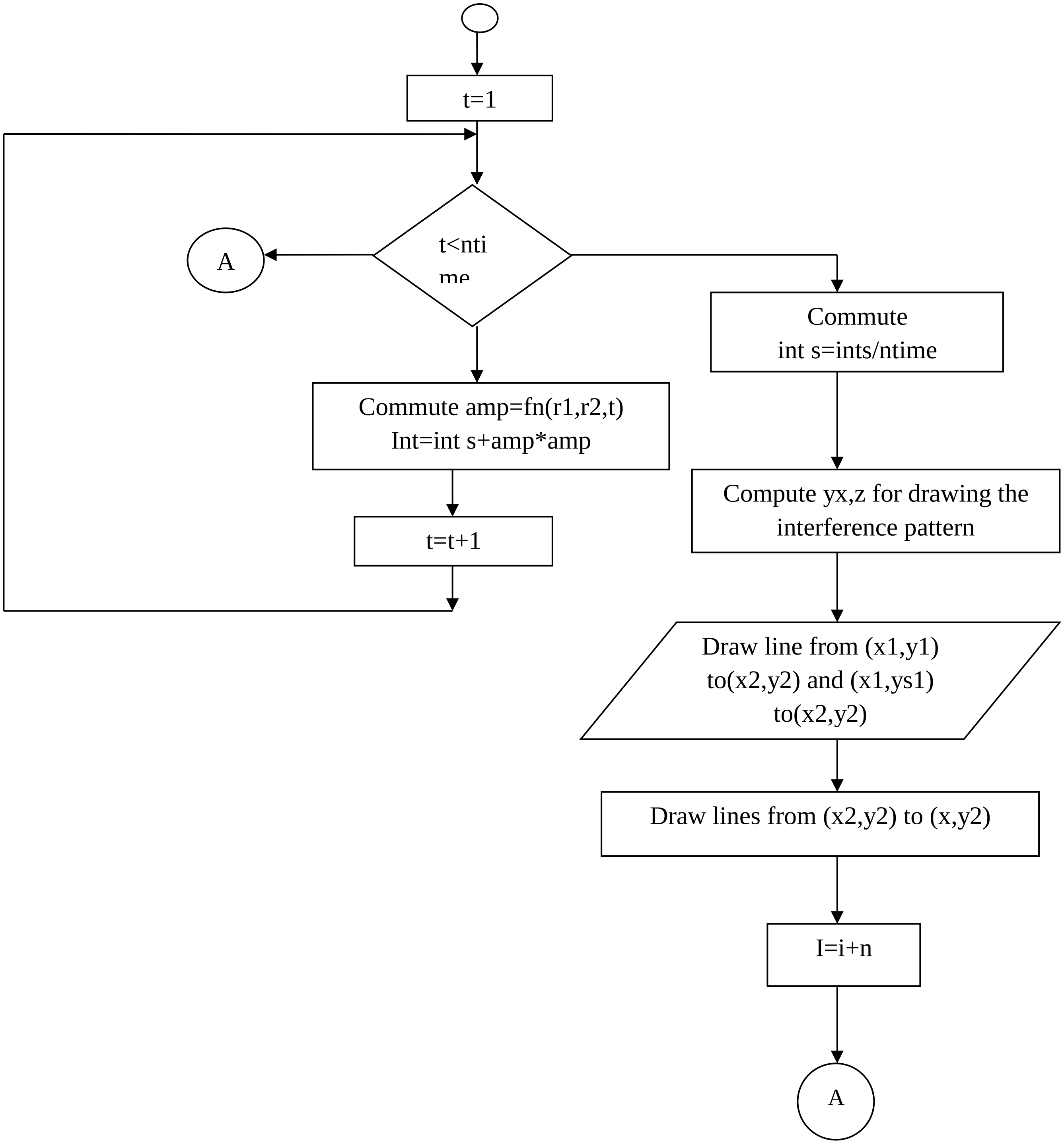
```
cin>>s1>>n>>ntime;
}
double slit::fnf(float x,float y,float t)
{
return amp*((cos(v*x-w*t))/x+(cos(v*y-w*t))/y);
}
void slit::show()
{
clrscr();
int driver,mode;
driver=DETECT;
initgraph(&driver,&mode,"\\tc\\bgi");
float h=ymax/n;
iam=iam*.0000001;
double c1=30;
v=2*pi/iam;
w-c1*v;
float y1,r1,r2,t,z,x2,q,y2;
float x1=0,x;
y1=0.5*(ymax-s1);
x2=x1;
y2=.5*(ymax+s1);
q=amp*2/d;
x=d+x1;
float y,zmax,zscale,yscale,ymin=0,zmin=0,ys1=93,ys2=118,amp;
zmax=pow(q,2);
zmin=0;
ymin=0;
yscale=160/(ymax-ymin);
zscale=200/(zmax-zmin);
setcolor(GREEN);
outtext("INTERFERENCE PATTERN");
for(int i=0;i<=n;i++)
{
y=ymin+(i*h);
r1=(x-x1)*(x-x1)+(y-y1)*(y-y1);
r2=(x-x2)*(x-x2)+(y-y2)*(y-y2);
r1=sqrt(r1);
r2=sqrt(r2);
float ints=0;
for(int t=1;t<+ntime;t++)
{
amp=fnf(r1,r2,t);
ints=ints+amp*amp;
}
ints=ints/ntime;
float x2=400,y2,z,x1=100;
y2=(ymax-y)*yscale+26;
z=(ints-zmin)*zscale+400;
rectangle(101,25,399,185);
delay(25);
setcolor(GREEN);
line(x1,ys1,x2,y2);
line(x1,ys2,x2,y2);
setcolor(GREEN);
line(x2,y2,z,y2);
}
}
void main()
{
slit d;
clrscr();
d.getdata();
d.show();
getch();
closegraph();
}
```

RESULT

A C++ program to generate the interference pattern is written and executed. As the distance from the screen to slit increases, the intensity of maxima decreases and number of maxima and minima decreases

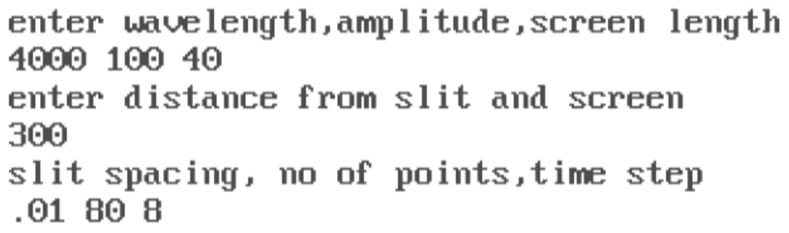
FLOWCHART:


Start
Give the values of π
Give the values of wavelength,amplitude,,screen length,slit
Define the function amp*(cos(v*x-w*t)/k+(cos(v*x-w*t)/y
Compute h,Iam,v,w,e1,y,x2,y2,q,x
Initialize the values of x,ymin,zmin,ys1,ys2
Compute zmin,xscale,yscale
I=0
Is
Stop
Compute y,r,re
Initialize the values of int s


t=1
t<nti
me
A
Commute
int s=ints/ntime
Commute amp=fn(r1,r2,t)
Int=int s+amp*amp
Compute yx,z for drawing the
interference pattern
t=t+1
Draw line from (x1,y1)
to(x2,y2) and (x1,ys1)
to(x2,y2)
Draw lines from (x2,y2) to (x,y2)
I=i+n
A

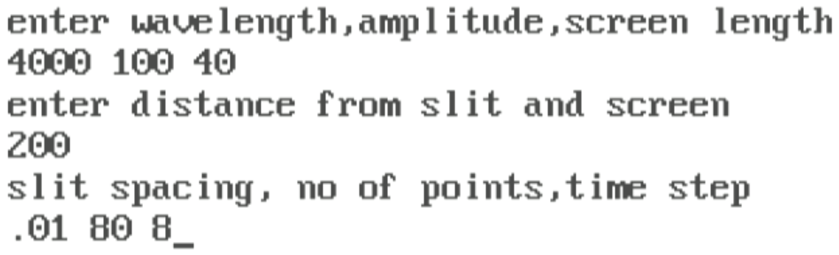
OUTPUT:

Case 1:



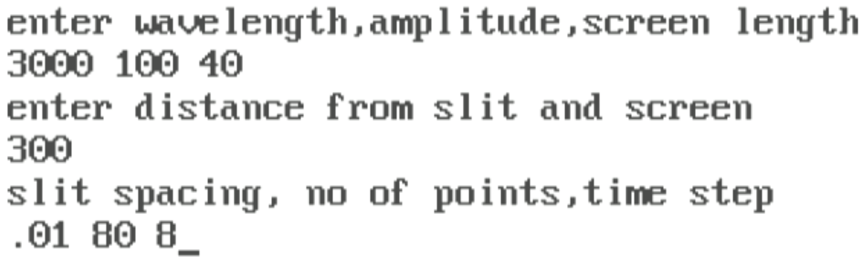
```
enter wavelength,amplitude,screen length
4000 100 40
enter distance from slit and screen
300
slit spacing, no of points,time step
.01 80 8
```

Case 2:



```
enter wavelength,amplitude,screen length
4000 100 40
enter distance from slit and screen
200
slit spacing, no of points,time step
.01 80 8_
```

Case 3:



```
enter wavelength,amplitude,screen length
3000 100 40
enter distance from slit and screen
300
slit spacing, no of points,time step
.01 80 8_
```

Experiment No:7
Date:

SOLUTION OF A DIFFERENTIAL EQUATION USING RUNGA-KUTTA METHOD

AIM:

To write a C++ program to solve an ordinary differential equation using Runga- Kutta method.

THEORY:

R-K method refers to a family of one step method, used for numerical solution of initial value problem. They are all based on the general form of extrapolation equation,

$$Y_{i+1} = Y_i + \text{slope} * \text{interval}$$
$$= Y_i + mh$$

Where *m* represents the slope, which is the weighted average of the slopes at various points in the interval ‘h”.

R-K method is known by their order. R-K method is called the *r*-order R-K method when slopes at *r*-points are used to construct the weighted average slope, *m*. In Euler`s method, we used only one slope at X_i to estimate X_{i+1} and therefore Euler’s method is a first order R-K method.

Similarly Heun’s method is the second order R-K method, because it employs slopes at two end points of the interval.

In Heun’s method ,higher the order better the accuracy of the estimates , therefore selection of order for using R-K method depends on the problem under consideration . However, the commonly used one is the fourth order R-K method. The most popular method is the classical fourth order R-K method.

Consider the differential equation,

$$\frac{dy}{dx} = f(x, y)$$

Take the point (x, y) on the curves Y(x) and draw a straight line from (x_i, y_i) with the slope, $m_1 = f(x_i, y_i)$. Let it cut the vertical line through $x_i + \frac{h}{2}$. Now determine the slope $\frac{dy}{dx}$ of the curve given by,

$$m_2 = f\left(x_i + \frac{h}{2}, y_i + m_1\frac{h}{2}\right)$$

From the point (Xi,Yi), draw a line with slope mr . Cutting the vertical line at (Xi+h) where the slope

$$m_3 = f\left(x_i + \frac{h}{2}, y_i + m_2\frac{h}{2}\right)$$
$$m_4 = f(x_i + h, y_i + m3h)$$

The weighted average of the slope is

$$m = (m_1 + 2m_2 + 2m_3 + m_4)/6$$

so , the solution of the differential equation can be obtained over some interval X, with initial condition $Y=Y_0$ at $X=X_0$. Choosing the slope size h, the solution of the slope are,

$$m_1=f\ (X_i,Y_i)$$
$$m_2=f\ (x_i+h/2,\ Y_i+m_1h/2)$$
$$m_3=f\ (x_i+h/2,y_i+m_2h/2)$$
$$m_4=f\ (x_i+h,y_i+m_3h)$$
$$Y_{i+1}=Y_i+\ (m_1+2m_2+2m_3+m_4)*h/6$$

ALGORITHM:

Step1: start
Step2: Define the initial condition and step size
Step3: Compute $m_{1,}$ m2,m_3 , m_4, and Y_{i+1}
Step4 : If Y is not the estimated value, repeat the process until the estimated
value is obtained. Otherwise go to step 5
Step5: print the result
Step6: stop

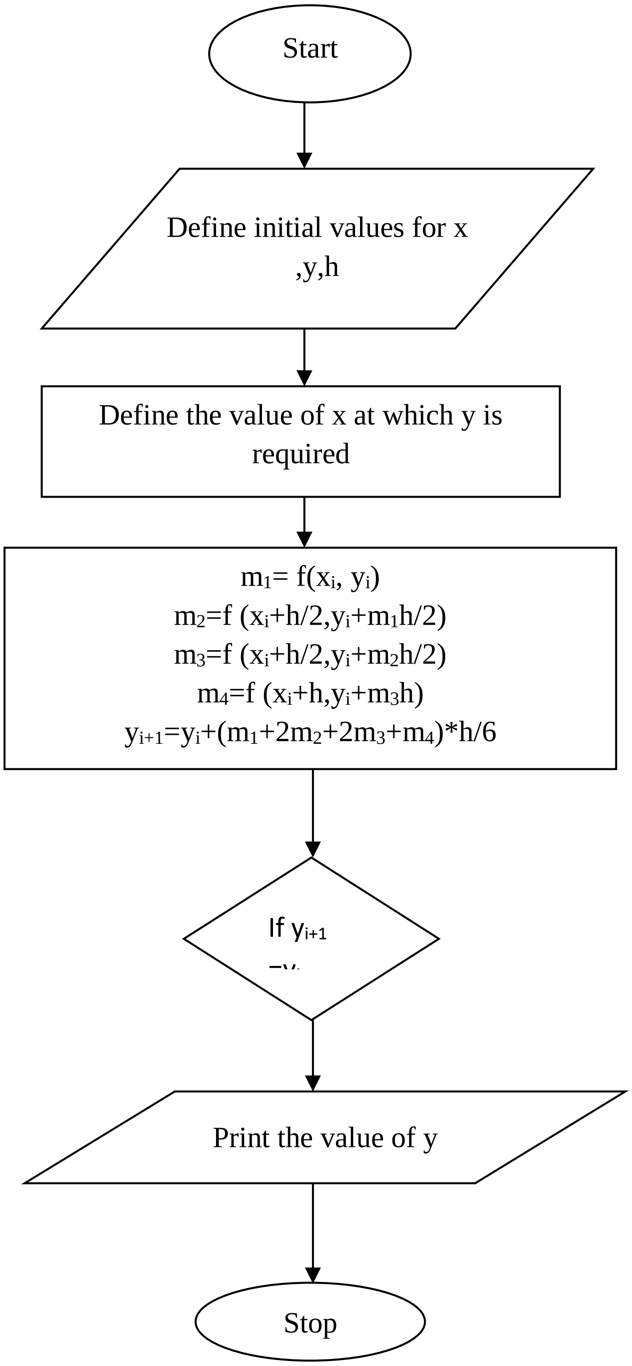
PROGRAM:

```
#include<iostream.h>
#include<conio.h>
#include<iomanip.h>
class  runge
{
private:
int n,i;
float x,y,xp,h,m1,m2,m3,m4,f;
public:
float fna(float,float);
void getdata();
void solution();
};
void runge::getdata()
{
cout<<"solution  by  runge kutta  method"<<endl;
cout<<"input  initial values  of  x  and y"<<endl;
cout<<"x=";
cin>>x;
cout<<"y=";
cin>>y;
cout<<"\ninput x at which y is  required"<<endl;
cin>>xp;
cout<<"Enter the step size h "<<endl;
cin>>h;
}
void runge::solution()
{
n=(int)((xp-x)/h+.5);
for(i=1;i<=n;i++)
{
m1=fna(x,y);
m2=fna(x+.5*h,y+.5*m1*h);
m3=fna(x+.5*h,.5*m2*h);
m4=fna(x+h,y+m3*h);
x=x+h;
y=y+(m1+2.0*m2+2.0*m3+m4)*h/6.0;
cout<<setw(2)<<i<<'\t';
cout<<setw(6)<<x<<'\t';
cout<<setw(6)<<y<<'\t';
}
cout<<"The value of y at  x= "<<x<<" is  "<<y;
}
float runge::fna(float x,float y)
{
return(3*x*x+1);
}
void main()
{
clrscr();
runge r;
r.getdata();
r.solution();
getch();
}
```

RESULT:

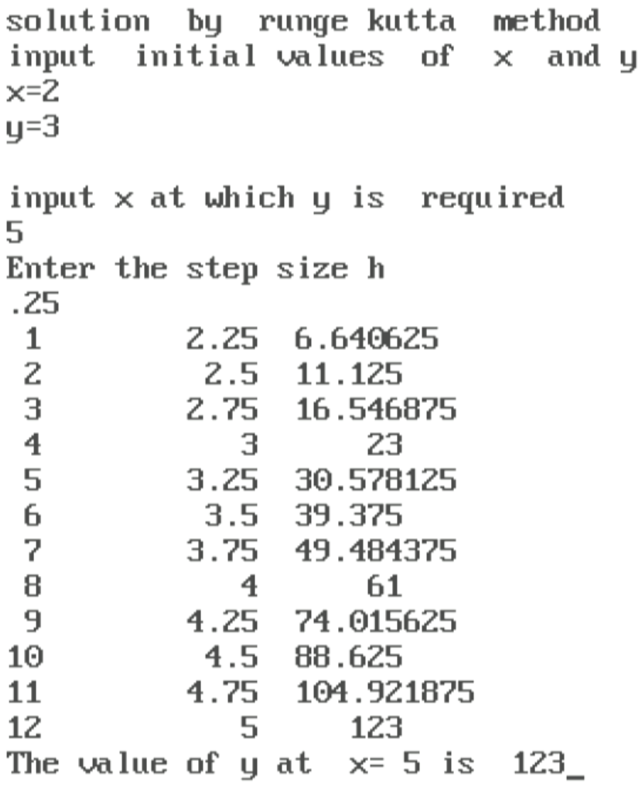
A C++ program for solving an ordinary differential equation using R-K method is entered and executed.

<u>FLOWCHART:</u>


Start
Define initial values for x ,y,h
Define the value of x at which y is required
$m_1= f(x_i, y_i)$
$m_2=f (x_i+h/2,y_i+m_1h/2)$
$m_3=f (x_i+h/2,y_i+m_2h/2)$
$m_4=f (x_i+h,y_i+m_3h)$
$y_{i+1}=y_i+(m_1+2m_2+2m_3+m_4)*h/6$
If y_{i+1}
=y
Print the value of y
Stop

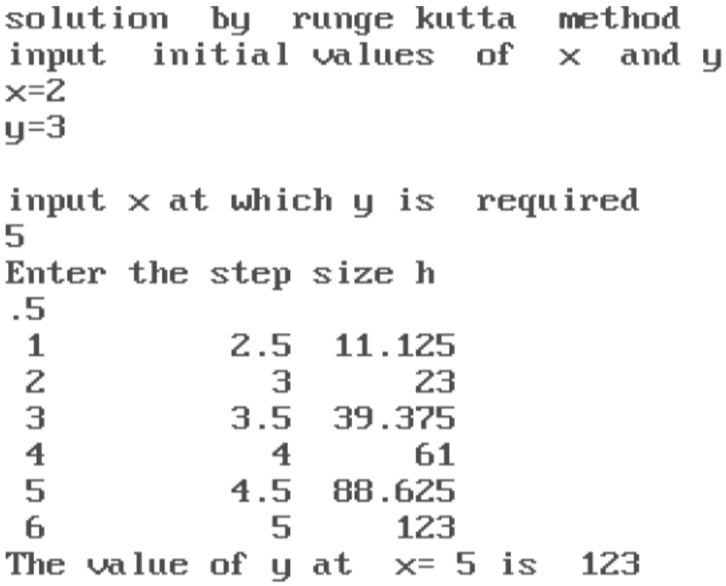
OUTPUT:

CASE 1:



```
solution  by  runge kutta  method
input  initial values  of  x  and y
x=2
y=3

input x at which y is  required
5
Enter the step size h
.25
 1        2.25  6.640625
 2         2.5  11.125
 3        2.75  16.546875
 4           3      23
 5        3.25  30.578125
 6         3.5  39.375
 7        3.75  49.484375
 8           4      61
 9        4.25  74.015625
10         4.5  88.625
11        4.75  104.921875
12           5     123
The value of y at  x= 5 is  123_
```

CASE 2:



```
solution  by  runge kutta  method
input  initial values  of  x  and y
x=2
y=3

input x at which y is  required
5
Enter the step size h
.5
 1         2.5  11.125
 2           3      23
 3         3.5  39.375
 4           4      61
 5         4.5  88.625
 6           5     123
The value of y at  x= 5 is  123
```

Experiment No:8
Date:

SOLUTION BY GAUSS ELIMINATION METHOD

AIM:

To write and execute a program in C++ to solve the given system of linear equation by Gauss elimination method.

THEORY:

Gauss elimination method proposes systematic strategy for reducing the system of equations to the upper triangular from using the forward elimination approach and then obtaining values of the unknown using the back substitution process. The strategy therefore compress of two phases.

1. Forward elimination phase

   This phase is concerned with the manipulation of equation in order to eliminate some unknown from the equation and produce upper triangular system.

2. Back substitution method

   This phase is concerned with the actual solution of the equation and uses the back substitution process on the reduced upper triangular system.

Consider a general set of n equation in n unknowns

$$a_{11}x_1 + a_{12}\,x_2 \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots + a_{1n}x_n = b_1$$

$$a_{21}x_1 + a_{22}\,x_2 \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots + a_{2n}x_n = b_2$$

$$\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots$$

$$\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots$$

$$a_{n1}x_1 + a_{n2}\,x_2 \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots \dots + a_{nn}x_n = b_n$$

The following steps are involved in implementing Gauss elimination strategy for such a general system.

1. Arrange the equation such that a_{11}=0
2. Eliminate x_1 from all except the first equation. This is done as following

   a). Normalize the first equation by dividing it by a_{11}

   b). Subtract from the second equation a_{21} time the normalized equation.

$$\left(a_{21} - a_{21}\left(\frac{a_{11}}{a_{11}}\right)\right)x_1 + \left(a_{22} - a_{21}\left(\frac{a_{12}}{a_{11}}\right)\right)x_2 + \cdots \dots .. \dots \dots \dots$$

$$\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots.. = (b_2 - a_{21}(\frac{b_1}{a_{11}}))$$

i.e. The result equation does not contain x_1 and the new second equation is

$$0 + a_{33}^{|}x_2 + \cdots \dots \dots \dots + a_{3n}^{|}x_n = b_3^{|}$$

If we repeat this procedure till the n^{th} equation is operated on, we will get the following new system of equations.

$$a_{11}x_1 + a_{12}\,x_2 \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots + a_{1n}x_n = b_1$$

$$a_{22}\,x_2 + \cdots \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots + a_{2n}^{|}x_n = b_2^{|}$$

3. Eliminate x_2 from the 3^{rd} to last equation in new set. Again assume that $a_{22}^{|}$ ≠0.
   a) Subtract the 3^{rd} equation $a_{32}^{|}$ times the normalized second equation.
   b) Subtract from the 4^{th} equation $a_{42}^{|}$ times the normalized 2^{th} equation and so on.

This process will continue until the last equation contains only one unknown namely x_n.
The final form of equation will look like this,

$$a_{11}x_1 + a_{12}\,x_2 \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots + a_{1n}x_n = b_1$$

$$a_{22}^{|}x_2 + \cdots \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots + a_{2n}^{|}x_n = b_2$$

$$\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots$$

$$\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots$$

$$\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots + a_{nn}^{n-1}x_n = b_n^{n-1}$$

This process is called triangularisation. The no. of primes indicated the no. of times the coefficient has been modified.

4. Obtain Solution by back substitution

The solution is as follows,

$$x_n = b_n^{n-1}/a_{nn}^{n-1}$$

This can be substituted back in the $(n-1)^{th}$ equation to obtain the solution for x^{n-1} . This back substitution will be continued till we get the solution for x_1.

ALGORITHM:

Step 1: Start the program

Step 2: Array equation such that a_{11}except the first equation.

Step 3: Eliminate x_1 from all except the first equation.

Step 4: Eliminate x_2 from 3^{rd} to last equation and this is repeated till the last equation contain only one unknown namelyx_n.

Step 5: Obtain solution as x_n=b_n^{n-1}/a_{nn}^{n-1} by back substitution.

Step 6: Stop.

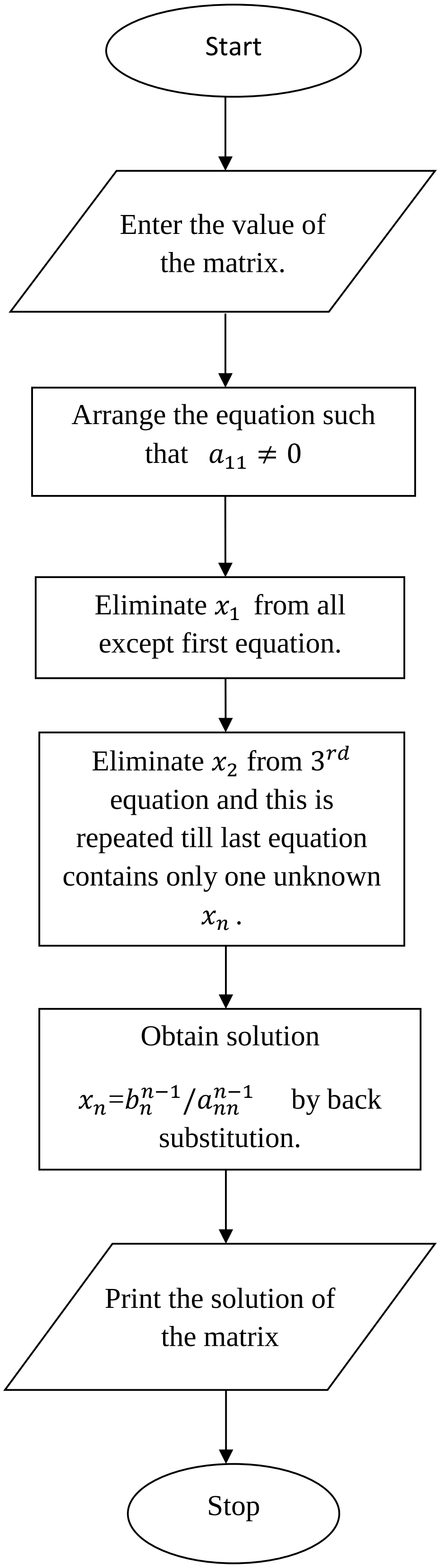
PROGRAM :

```
#include<iostream.h>
#include<conio.h>
class gauss
{
 int status,n,i,j;
 float a[10][10],b[10],x[10];
 public:
 void getdata();
 void elimination();
 void display();
 };
 void gauss::getdata()
 {
 cout<<"Solution by Gauss Elimination method \n";
 cout<<"Dimension of matrix \n";
 cin>>n;
 cout<<"Enter the matrix with one row in each line \n";
 for(i=1;i<=n;i++)
 for(j=1;j<=n;j++)
 cin>>a[i][j];
 cout<<"Enter the vector b \n";
 for(i=1;i<=n;i++)
 cin>>b[i];
 }
 void gauss::display()
 {
 if(status!=0)
 {
 cout<<"Solution \n";
 for(i=1;i<=n;i++)
 cout<<x[i]<<"\n";
 }
 else
 {
 cout<<"Singular matrix \n";
 }
 }
 void gauss::elimination()
 {
 for(int k=1;k<=n-1;k++)
 {
 float pivot=a[k][k];
 if(pivot<0.000001)
 {
 status=0;
 }
 else
 {
 status=1;
 }
 for(i=k+1;i<=n;i++)
 {
 float factor=a[i][k]/pivot;
 for(j=k+1;j<=n;j++)
 {
 a[i][j]=a[i][j]-factor*a[k][j];
 }
 b[i]=b[i]-factor*b[k];
 }
 }
 x[n]=b[n]/a[n][n];
 for(k=n-1;k>=1;k--)
 {
 float sum=0.0;
 for(j=k+1;j<=n;j++)
 {
 sum=sum+a[k][j]*x[j];
 x[k]=(b[k]-sum)/a[k][k];
 }
 }
 }
 void main()
 {
 clrscr();
 gauss g;
 g.getdata();
 g.elimination();
 g.display();
 getch();
}
```

RESULT:

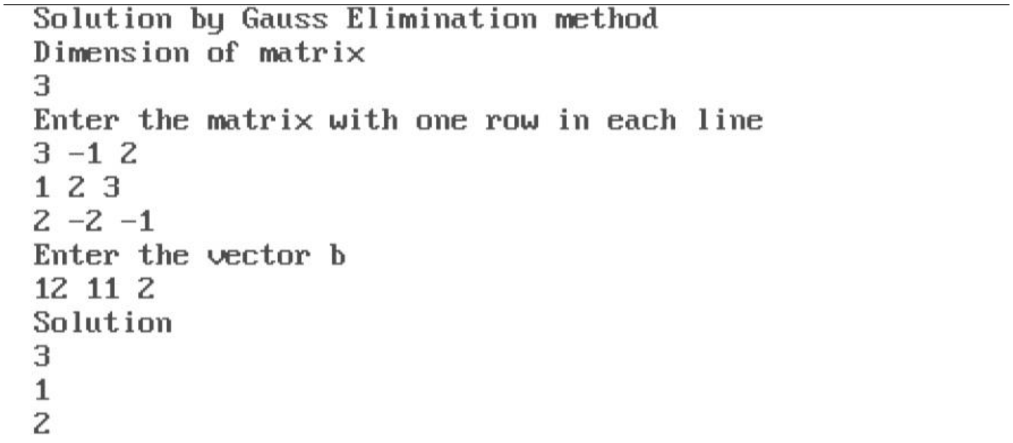
A C++ program to solve given system of linear equations by Gauss Elimination method is entered, executed and solution is obtained.

FLOW CHART:


Start
Enter the value of the matrix.
Arrange the equation such that $a_{11} \neq 0$
Eliminate x_1 from all except first equation.
Eliminate x_2 from 3^{rd} equation and this is repeated till last equation contains only one unknown x_n .
Obtain solution x_n=b_n^{n-1}/a_{nn}^{n-1} by back substitution.
Print the solution of the matrix
Stop

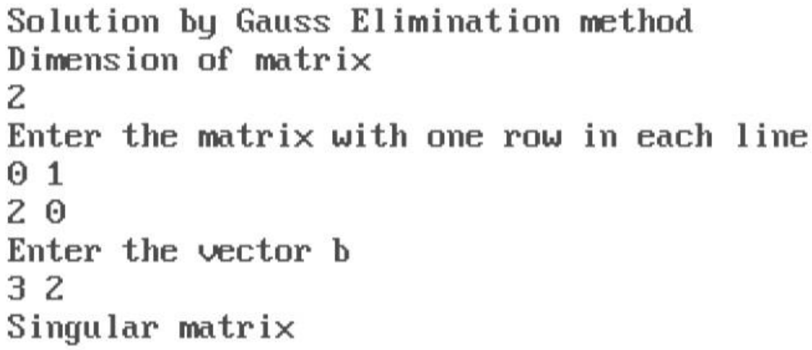
OUT PUT:

Case 1:



```
Solution by Gauss Elimination method
Dimension of matrix
3
Enter the matrix with one row in each line
3 -1 2
1 2 3
2 -2 -1
Enter the vector b
12 11 2
Solution
3
1
2
```

Case 2:



```
Solution by Gauss Elimination method
Dimension of matrix
2
Enter the matrix with one row in each line
0 1
2 0
Enter the vector b
3 2
Singular matrix
```

Experiment No:9
Date:

FORMATION OF STANDING WAVES λ

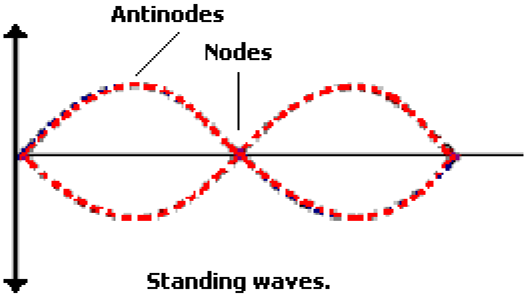
AIM :

To write & execute a C++ program to generate a pattern of standing wave. Run this program with different values of wavelength and velocity.

THEORY :

Standing wave is a wave in a medium in which each point on the axis of the wave has associated constant amplitude. It is also known as stationary wave. The most common cause of standing wave is the phenomenon of resonance. If two waves have same amplitude, wavelength & velocity travel in the opposite direction then they superimpose each other. The resultant is known as standing wave. Such a situation occurs when the wave propagate in a medium, the wave get reflected. Resulting wave is described by the superposition of incident and reflected waves. The wave pattern thus obtained is called a standing wave and is given by

y= A sin $\left[\frac{2\pi(x-vt)}{\lambda}\right] + A\,\sin[\frac{2\pi(x+vt)}{\lambda}]$



With the boundary condition, amplitude vanishes at the end of the medium. In addition to the end points there are other points which are always at rest. These stationary points are called nodes & are separated by one half of the wavelength of the wave. In the standing wave, energy is not transport beyond nodes as it gets trapped between them. These are points called antinodes, which have maximum amplitude. For instance standing electromagnetic waves are generated between two mirrors in a laser. Wave mechanics was developed by Schrodinger by assuming that the wave representing the electron forms a standing wave in an atom.

ALGORITHM:

```
Step 1: Start
Step 2: Input values of wavelength, velocity , $I_{max}$ ,$x_{max}$
Step 3: Initialize x=0; t=0
Step 4: Calculate $t_p$=λ/v
Step 5: If t < $t_p$ continue, otherwise go to step 10
Step 6: If x < $x_{max}$ , continue otherwise go to step 9
Step 7: Calculate
        $F(x, t) = A \sin[\frac{2\pi(x-vt)}{\lambda}] + A \sin[\frac{2\pi(x+vt)}{\lambda}]$
Step 8: Plot F(x ,t), x= x+ h
Step 9: Calculate t = t+ ht , go to step 5
Step 10: Stop
```

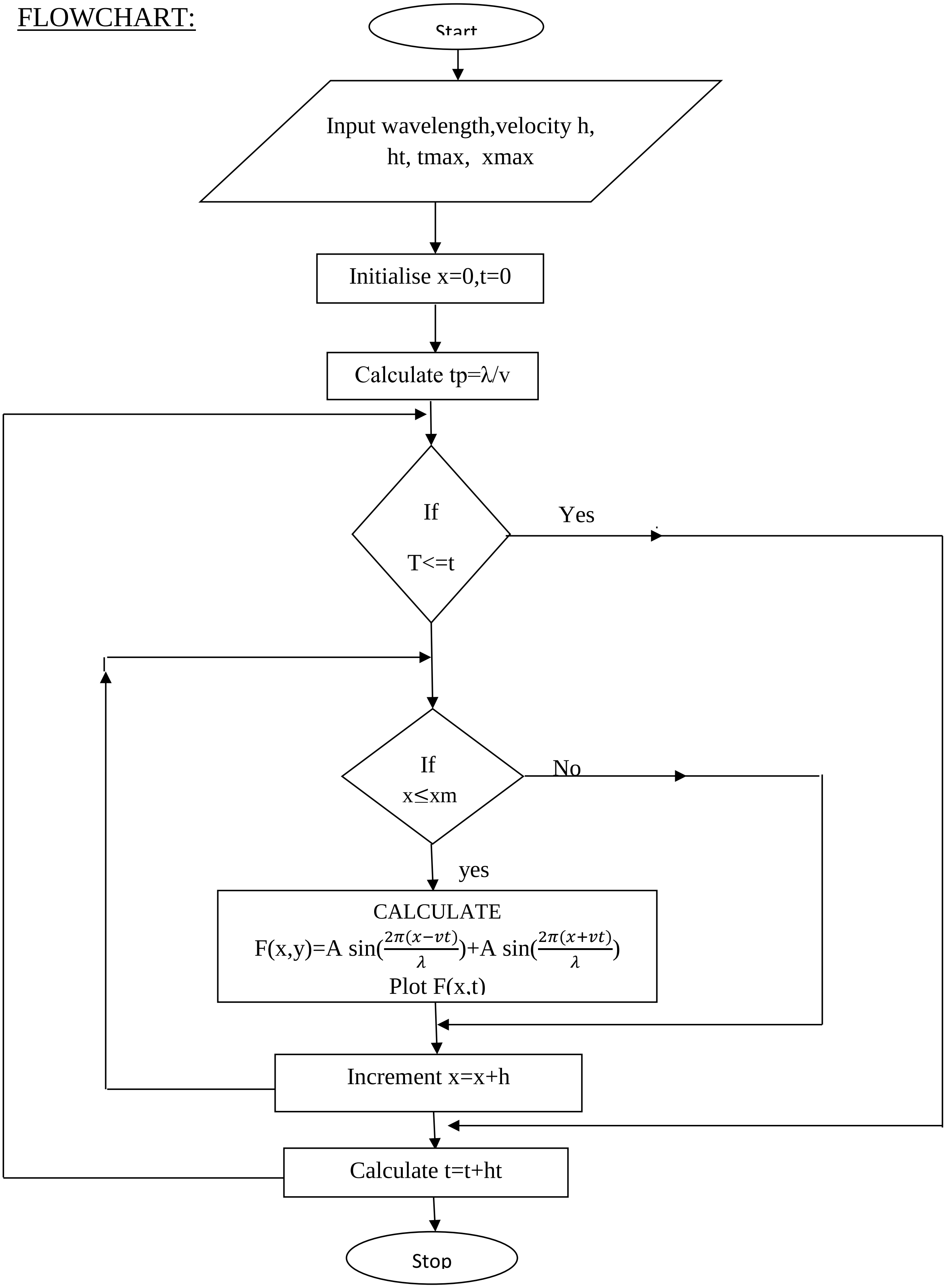
PROGRAM

```
 #include<iostream.h>
#include<conio.h>
#include<math.h>
#include<graphics.h>
#include<dos.h>
const float pi=3.141592;
class resultant
{
public:
void getdata();
float a,lam,v,xmax,tmax;
void calc();
float fnf(float,float);
};
void resultant::getdata()
{
cout<<"Enter the values of a,lam,v,xmax,tmax\n";
cin>>a>>lam>>v>>xmax>>tmax;
}
float resultant::fnf(float x,float t)
{
return(a*sin(2*pi*(x-v*t)/lam)+a*sin(2*pi*(x+v*t)/lam));
}
void resultant::calc()
{
float tp=lam/v,xmin=0.0,ht=0.05,h=0.05,t=0.05;
int driver,mode;
driver=DETECT;
initgraph(&driver,&mode,"C:\\tc\\bgi");
float xscale,yscale,ymin,ymax,x,y;
ymin=-2;
ymax=+2;
xscale=610/(xmax-xmin);
yscale=440/(ymax-ymin);
while(t<=tp)
{
float x=xmin;
float X=xmin*xscale+10;
float Y=(ymax-fnf(x,t))*yscale+8;
moveto(X,Y);
while(x<=xmax)
{
float y=fnf(x,t);
float X=x*xscale+10;
 float Y=(ymax-y)*yscale+8;
 lineto(X,Y);
 x=x+h;
 }
 t=t+ht;
 }
 getch();
 }
 void main()
 {
 clrscr();
 resultant data;
 data.getdata();
 data.calc();
 getch();
 closegraph();
 }
```

RESULT :

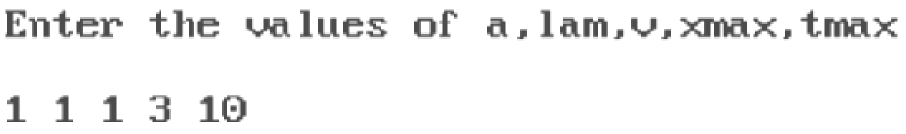
The C++ program to generate standing wave pattern is written and executed. As the velocity of wave decreases number of loops increases, the number of nodes and antinodes remains the same. As the wavelength decreases, number of nodes and antinodes increases.

FLOWCHART:


Start
Input wavelength,velocity h,
ht, tmax, xmax
Initialise x=0,t=0
Calculate tp=λ/v
If
T<=t
Yes
If
x≤xm
No
yes
CALCULATE
F(x,y)=A sin($\frac{2\pi(x-vt)}{\lambda}$)+A sin($\frac{2\pi(x+vt)}{\lambda}$)
Plot F(x,t)
Increment x=x+h
Calculate t=t+ht
Stop

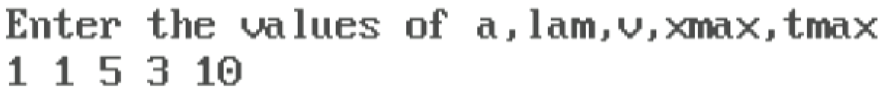
OUTPUT:

Case 1:



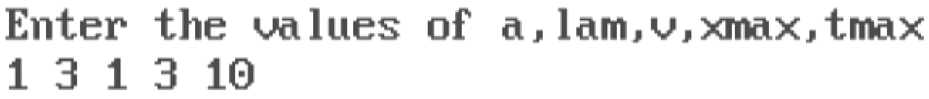
```
Enter the values of a,lam,v,xmax,tmax

1 1 1 3 10
```

Case 2:



```
Enter the values of a,lam,v,xmax,tmax
1 1 5 3 10
```

Case 3:



```
Enter the values of a,lam,v,xmax,tmax
1 3 1 3 10
```

Experiment:10

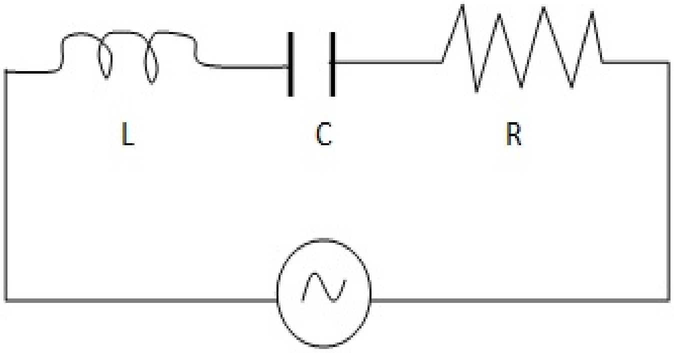
Date ;

SERIES LCR CIRCUIT

Aim

Study the variation in phase relation between applied voltage and current of a series LCR circuit with three different set of values of L and C. Find the resonant frequencies and values of maximum current in each case.

Theory



An LCR is an electrical circuit consisting of a resistor(R), an inductor(L) and a capacitor(C) connected in series or in parallel.

The circuit forms a harmonic oscillator for current and resonance in a similar way as an LC circuit. Introducing the resistor increases the decay of these oscillations, which is also known as damping.

. Let an alternating voltage $V = V_m sin\omega t$ be applied to a circuit containing an inductance L, capacitor C and resistor R connected in series as in figure. Kirchoff 's loop rule,

$$V_m \sin\omega t = L\frac{dI}{dt} + \frac{q}{C} + IR \qquad (1)$$

The current through the circuit has the same frequency as applied voltage and can be represented as:

$$I = A \sin\omega t + B \cos\omega t \qquad (2)$$

Differentiating, $\frac{dI}{dt}$ = Aωcos ωt – Bωsinωt = ω[Acosω t-Bsinωt] (3)

If dq is the charge acquired by the plate of the capacitor in a time dt , then dq = I dt

Or q=∫ I dt

=∫ [A sin ωt +B cosω t]dt

= -$\frac{A}{\omega}$ cosωt + $\frac{B}{\omega}$ sinωt

ie q=$\frac{1}{\omega}$[-A cosω t+ B sinω t] (4)

substituting for I, $\frac{dI}{dt}$ and q in eqn (1)

V_m sinωt= ωL[A cosωt -B sinωt] + $\frac{1}{\omega C}$[-Acosωt + B sinωt]

+R[A sinωt+B cosωt] (5)

When t=0; sinωt=0; cosωt=1

ie eqn(5) becomes,

0 = ωLA- $\frac{A}{\omega C}$ + RB

RB =-A [Lω - $\frac{1}{\omega C}$] = -Am

Where m = [ωL - $\frac{1}{\omega C}$]

Thus, B = -$\frac{Am}{R}$ (6)

At $t=\frac{T}{4}$, ωt=$\frac{2\pi}{T}$*$\frac{T}{4}$=$\frac{\pi}{2}$

Therefore , sinωt = sin$\frac{\pi}{2}$ = 1; cosωt=0

Therefore equation (5) becomes v_m= -ωLB+$\frac{B}{\omega c}$+RA

That is v_m= -B [ωL- $\frac{1}{\omega C}$]+ RA (7)

OR v_m= - Bm + RA

Substituting for B in equation (7)

$$v_m=\frac{Am^2}{R} + AR \quad =A[R+\frac{m^2}{R}]$$

$$=A\left[\frac{R^2+m^2}{R}\right]$$

Therefore A= [$\frac{V_m R}{R^2+m^2}$] (8)

Substituting for A in equation (7)

$$B = \frac{V_m R_m}{(R^2+m^2)R}$$

That is B =$\frac{V_m m}{(R^2+m^2)}$ (9)

Substituting the values of A and B from equation (8) and (9) in equation (2)

$$I = \frac{V_m R}{R^2+m^2}\text{sinωt} - \frac{V_m m}{(R^2+m^2)}\text{cosωt}$$

$$=\frac{V_m}{\sqrt{R^2+m^2}}\left[\frac{R}{\sqrt{R^2+m^2}}\text{sinωt}-\frac{m}{\sqrt{R^2+m^2}}\text{cosωt}\right] \quad (10)$$

Put $\frac{R}{\sqrt{R^2+m^2}}$ = cos φ and $\frac{m}{\sqrt{R^2+m^2}}$ = sin φ

Therefore, tan φ = $\frac{m}{R}$ (11)

Thus eqn (10) becomes

$$I = \frac{V_m}{\sqrt{R^2+m^2}}[\text{ cos φ sinωt - sin φ cosωt }]$$

$= \frac{V_m}{\sqrt{R^2+m^2}}$ sin (ωt – φ)

i.e, I = I_m sin (ωt – φ) (12)

where $I_m = \frac{V_m}{\sqrt{R^2+m^2}}$ (13)

This equation represents the peak value of the current through the circuit.

From eqn (12), it is clear that the current through the circuit lags behind the applied emf by φ, given by,

$$\tan \varphi = \frac{m}{R}$$

or

$$\tan \varphi = \frac{\omega L - \frac{1}{\omega C}}{R}$$

(14)

The term, $\cos \varphi = \frac{R}{\sqrt{R^2+m^2}} = \frac{R}{\sqrt{R+\left(\omega L-\frac{1}{\omega C}\right)^2}}$

Is known as the power factor of the LCR circuit. The term $\sqrt{R^2 + m^2}$ in eqn (13) represents the effective resistance offered by the circuit to the flow of AC and is called the impedance of the LCR circuit denoted by, Z = $\sqrt{R^2 + m^2}$

Therefore eqn (13) becomes $I_m = \frac{V_m}{Z}$ (15)

$$Z = \sqrt{R^2 + m^2}$$

$$Z = \frac{R}{\sqrt{R+\left(\omega L-\frac{1}{\omega C}\right)^2}} \quad (16)$$

Or

$$Z = \sqrt{R^2 + (X_L - X_C)^2} \quad (17)$$

Where X_L = Lω and $X_C = \frac{1}{\omega C}$

Special Cases:

1. If $\omega_L > \frac{1}{\omega C}$; (ωL - $\frac{1}{\omega C}$) is a positive quantity. In this case, φ becomes positive or current lags behind the applied voltage.
2. If $\omega_L <$ $\frac{1}{\omega C}$; (ωL - $\frac{1}{\omega C}$) is negative so that φ is negative. In this case, current leads the applied voltage.
3. If $\omega_L =$ $\frac{1}{\omega C}$; the current and the emf are in phase. This is the condition for resonance.

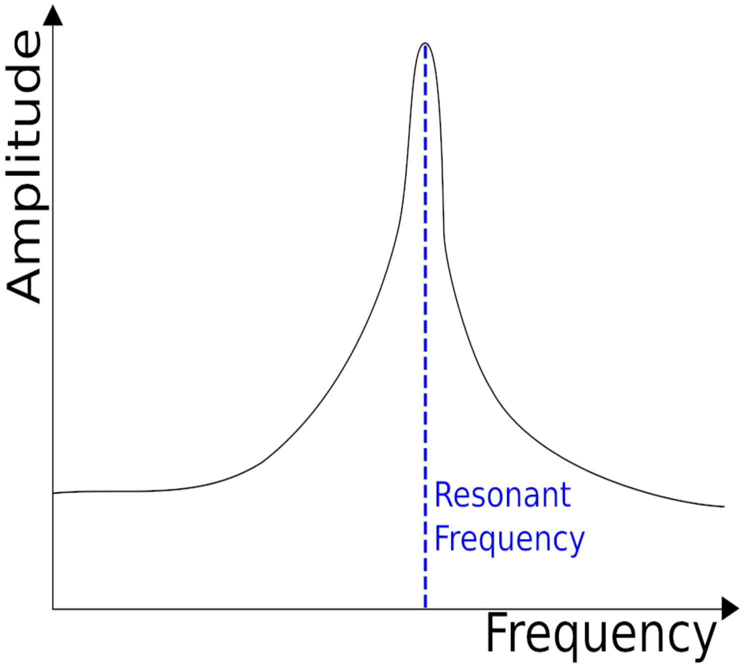
$$2\Pi f L = \frac{1}{2\Pi f C}$$

$$f^2 = \frac{1}{4\Pi^2 LC}$$

$$f_R = \frac{1}{2\Pi\sqrt{LC}}$$

This is the frequency at resonance.

Resonance is a phenomenon in which a vibrating system or external force drives another system to oscillate with greater amplitude at specific frequencies. Frequencies at which the response amplitude is a relative maximum are known as the system’s resonant frequencies or resonance frequencies. At resonance frequency, small periodic driving forces have the ability to produce large amplitude oscillations due to the storage of vibrational energy.


Amplitude
Resonant
Frequency
Frequency

Algorithm

Step 1: Start

Step 2: Enter the frequency(f), amplitude(e0), capacitance(C), inductance(L), resistance(R) values.

Step 3: Find angular frequency $\omega = 2\P f$, phase, resonant frequency of the LCR circuit.

Step 4: initialize time t= 0.

Step 5: Loop starts over t= 0 upto t= 1/f.

Step 6: Calculate current and amplitude using $i = i_0 \sin(\omega t - \varphi)$ and $e = e_0 \sin\omega t$.

Step 7: Increment t.

Step 8: Loop started at 5 ends.

Step 9: Stop.

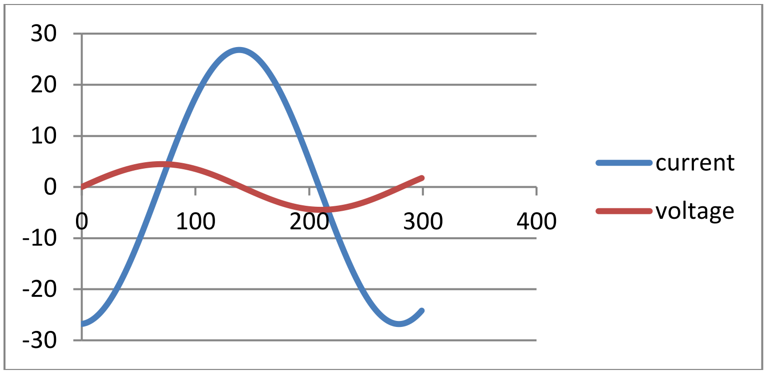
Program

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
#include<fstream.h>
```

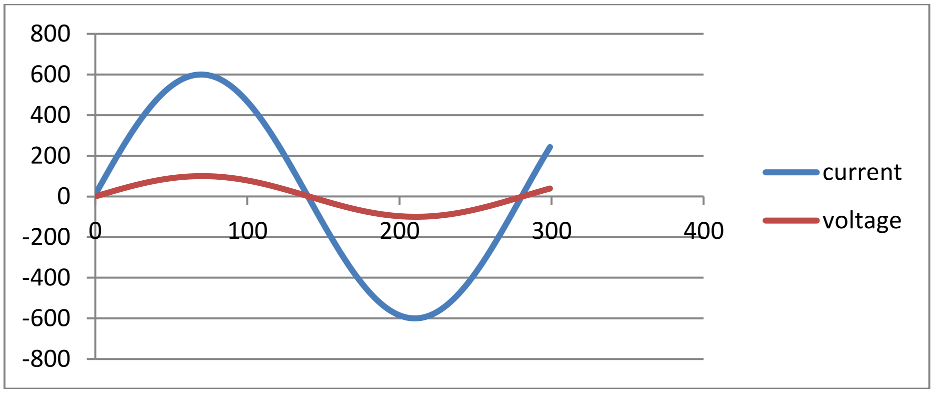
```
#include<iomanip.h>
class lcrphase
{
public:
float f,f0,e0,c,l,r,w,m,i0,tph,phi,i,sq,tl,il,t,e;
void getdata()
{
cout<<"Enter freq,ampl,capac,indu,resis"<<"\n";
cin>>f>>e0>>c>>l>>r;
};
void phase()
{
fstream outfile("lcr",ios::out);
outfile<<"max current"<<setw(20)<<"res freq"<<setw(20)<<"\n";
w=2*3.14*f;
m=(w*l)-(1/(c*w));
sq=sqrt((r*r)+(m*m));
i0=e0/sq;
cout<<"Maximum current"<<i0<<"\n";
f0=1/(2*3.14*sqrt(l*c));
cout<<"Resonant frequency"<<f0<<"\n";
tph=m/r;
phi=atan(tph);
cout<<"phase"<<phi<<"\n";
outfile<<i0<<setw(20)<<f0<<setw(20)<<"\n";
outfile.close();
};
void calculate()
{
fstream outfile("lcr6",ios::out);
cout<<"time"<<setw (15)<<"current"<<setw (15)<<"voltage"<<"\n";
outfile<<"time"<<setw(15)<<"current"<<setw(15)<<"voltage"<<"\n";
for(t=0;t<=0.003;t=t+0.00001)
{
tl=t*100000;
i=i0*sin((w*t)-phi);
il=i*60;
e=i0*(e0*sin(w*t));
cout<<tl<<setw(15)<<il<<setw(15)<<e<<"\n";
outfile<<tl<<setw(15)<<il<<setw(15)<<e<<"\n";
}
outfile.close();
}};
int main()
{
clrscr();
lcrphase va;
va.getdata();
va.phase();
va.calculate();
getch();
}
```

Output

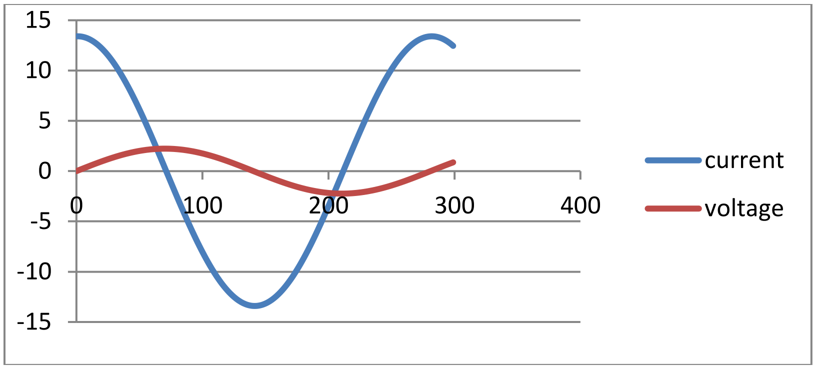
```
Enter frequency, amplitude, capacitance, inductance and resistance
356
10
.00001
.03
1
```



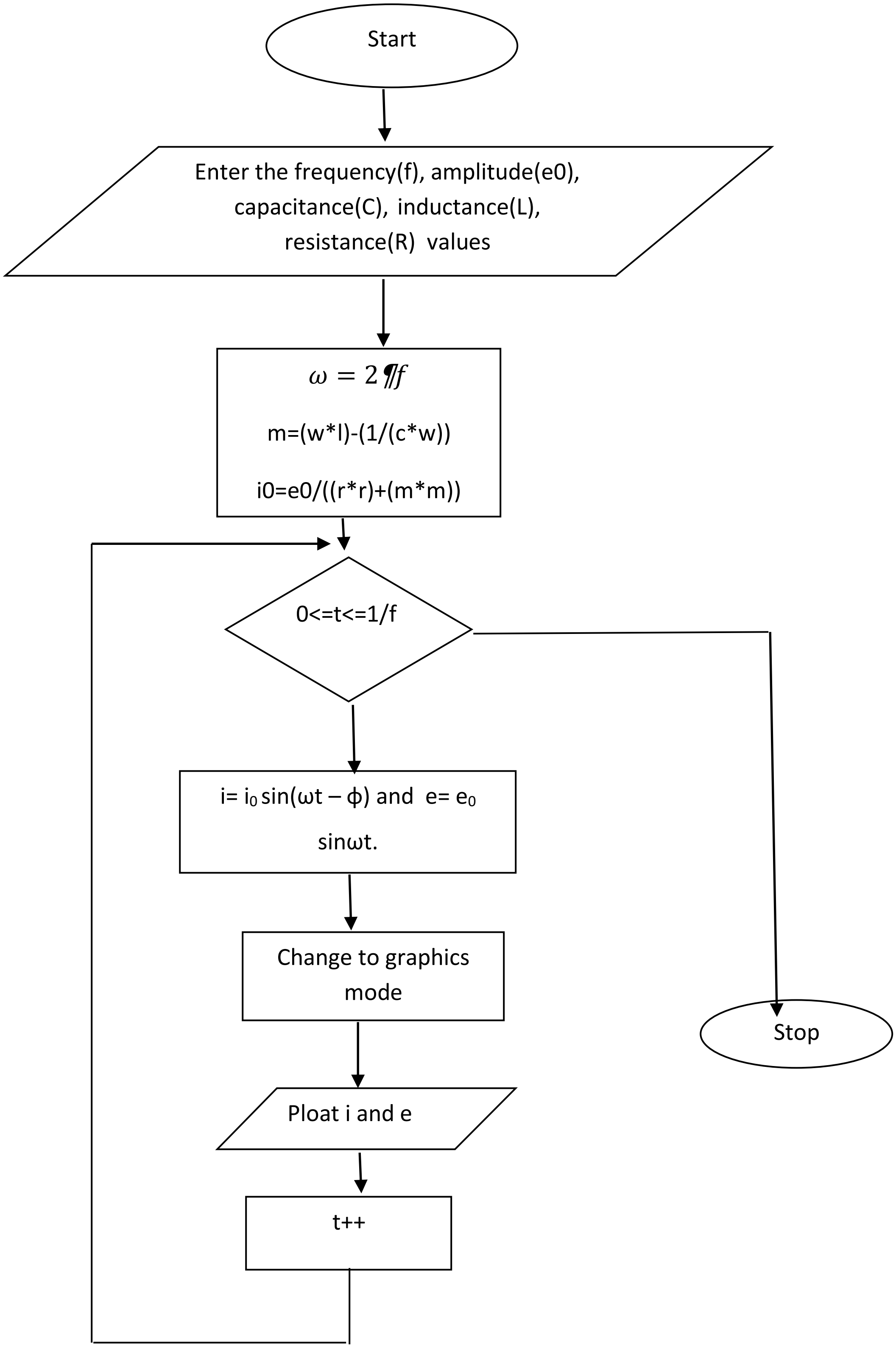
Enter frequency, amplitude, capacitance ,inductance and resistance
356
10
.00001
.02
1



Enter frequency, amplitude, capacitance, inductance and resistance
356
10
.000005
.02
1


15
10
5
0
-5
-10
-15
0
100
200
300
400
current
voltage

FLOWCHART


Start
Enter the frequency(f), amplitude(e0), capacitance(C), inductance(L), resistance(R) values
$\omega = 2¶f$
m=(w*l)-(1/(c*w))
i0=e0/((r*r)+(m*m))
0<=t<=1/f
i= i_0 sin(ωt – ϕ) and e= e_0 sinωt.
Change to graphics mode
Ploat i and e
t++
Stop

EXPERIMENT NO:11

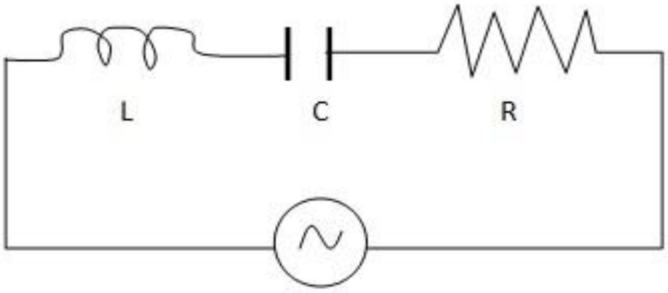
DATE:

CHARGING OF A CAPACITOR THROUGH AN INDUCTANCE AND RESISTANCE

AIM:

Write and execute a CPP program for to study the charging of a capacitor through an inductor and resistance connected in series with a dc source. Also plot the graph for charging of the capacitor for three different conditions: (a) non-oscillatory (b) critical and (c) oscillatory charging.

THEORY:



V_0sinωt

Let the circuit shown in the figure. Let **I**be the current in the circuit and **q** is the charge on the plate of the capacitor at any instant t,

The equation of emf is

$$L\frac{dI}{dt} + IR + \frac{q}{c} = E \qquad \text{..............................(1)}$$

Substituting I = $\frac{dq}{dt}$and rearranging equation (1), we get

$$\frac{d^2q}{dt^2} + \frac{R}{L}\frac{dq}{dt} + \frac{q}{CL} = \frac{E}{L} \quad \text{........................(2)}$$

Put $\frac{R}{L} = 2b$ and $\frac{1}{CL} = k^2$

$$\frac{d^2q}{dt^2} + 2b\frac{dq}{dt} + \left(k^2 - \frac{E}{L}\right) = 0$$

On solving the above equation we get the solution as, q=$q_0\left[1-\frac{e^{-bt}}{2}\left[\left(1+\frac{b}{m}\right)e^{mt}+\left(1-\frac{b}{m}\right)e^{-mt}\right]\right]$.......................... (3)

where m=$\sqrt{b^2-k^2}$represents the general expression for the charge on the plate of the capacitor at any instant. Depending on the values of b and k we have three cases.

(i)Non-oscillatory (damping)

For non-oscillatory charging m=$\sqrt{b^2-k^2}$ be real

i.e,b^2>k^2

q=$q_0\left[1-\frac{e^{-bt}}{2}\left[\left(1+\frac{b}{m}\right)e^{mt}+\left(1-\frac{b}{m}\right)e^{-mt}\right]\right]$ ....(4)

(ii)Critical charging

Here m=0

i.e, b^2=k^2

then equation (3) becomes

$q=q_0\,[1-e^{-bt}]$........................(5)

(iii) Oscillatory charging

For oscillatory charging m=$\sqrt{b^2-k^2}$ be imaginary,

thenb^2-k^2 is negative or $b^2<k^2$

then eqn.(3) becomes,

q=$q_0\left[1-\frac{ke^{-bt}}{w}\sin(wt+\phi)\right]$........................(6)

wherew=$\sqrt{k^2-b^2}$ and ϕ =$\tan^{-1}\left(\frac{w}{b}\right)$

ALGORITHM

Step 1:start

Step 2:input values of values of voltage,inductance,resistance and capacitance

Step 3:calculate the value of $b^2 - k^2$

Step 4:loop starts

Step 5:if$b^2 - k^2$ is positive ,corresponds to condition for damped charging and charge on the plate of the capacitor is,

$$q = q_0 \left[1 - \frac{e^{-bt}}{2}\left[\left(1 + \frac{b}{m}\right)e^{mt} + \left(1 - \frac{b}{m}\right)e^{-mt}\right]\right]$$

or;

if $b^2 - k^2$=0,then corresponds to condition for critical charging and charge on the plate of the capacitor is,

$$q = q_0\,[1 - e^{-bt}]$$

or,

if$b^2 - k^2$ is negative, corresponds to condition for oscillatory charging and charge on the plate of the capacitor is,

$$q = q_0 \left[1 - \frac{ke^{-bt}}{w}\sin(wt + \phi)\right]$$

step 6: loop ends

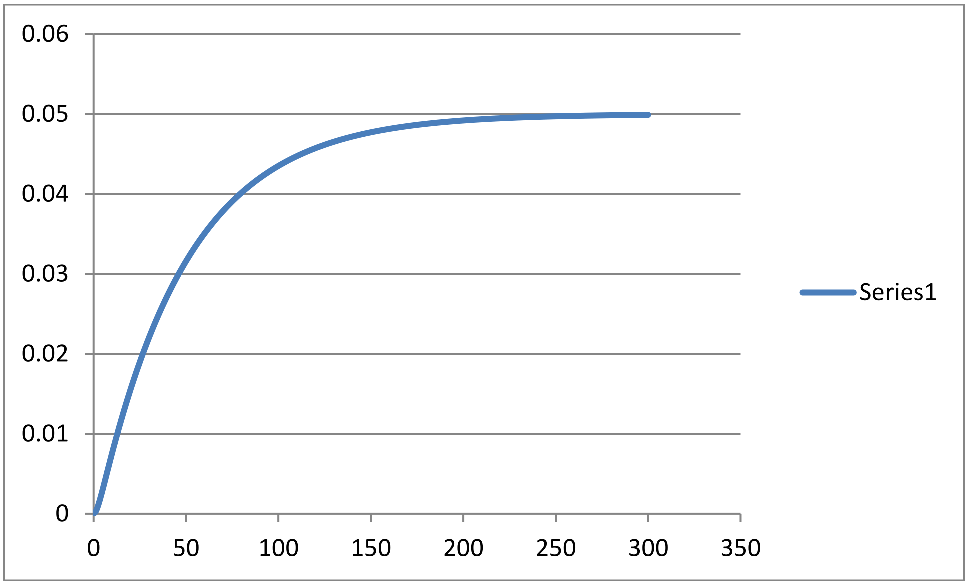
step 7: stop

PROGRAMME

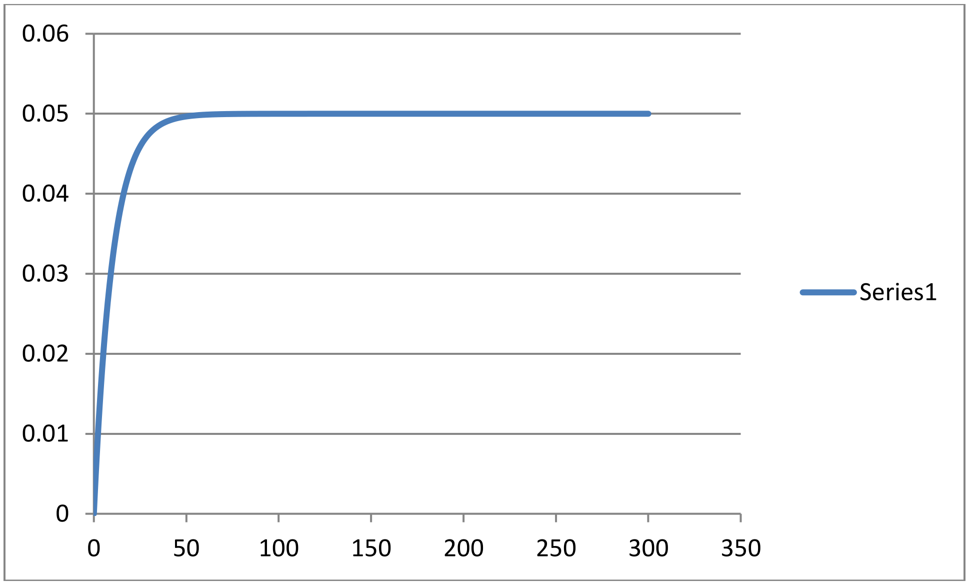
```
#include<iostream.h>
#include<math.h>
#include<iomanip.h>
#include<fstream.h>
#include<conio.h>
class lcrcharging
{
public:
float f,e,c,l,r,w,m,m2,q,q0,q1,q2,q3,ph,b,k,k2,a1,a2,a3,c1,t,tt,s,ch,ex,s1,s2;
void getdata()
{
cout<<"enter voltage,inductance,capacitance,resistance"<<"\n";
cin>>e>>l>>c>>r;
};
void condition()
{
q0=c*e;
b=r/(2*l);
k2=1/(l*c);
k=sqrt(k2);
m2=(b*b-k2);
if(m2>0)
{
m=sqrt(m2);
cout<<"m="<<m<<"\n";
}
else if(m2==0)
{
cout<<"m2=0";
}
else
{
cout<<"m is negative"<<"\n";
}
ch=(b*b-k2);
}
void damped()
{
fstream outfile("damped",ios::out);
cout<<"t"<<setw(15)<<"charge\n";
for(tt=0;tt<=300;tt++)
{
t=tt/1000;
a1=(1+b/m)*exp(m*t);
a2=(1-b/m)*exp(-m*t);
a3=a1+a2;
ex=exp(-1*b*t);
q=q0*(1-((ex/2)*a3));
cout<<tt<<setw(15)<<q<<"\n";
outfile<<tt<<setw(15)<<q<<"\n";
}
outfile.close();
}
void critical()
{
cout<<"t"<<setw(15)<<"q"<<"\n";
 fstream outfile("critical",ios::out);
 for(tt=0;tt<=300;tt++)
 {
 t=tt/1000;
 q=q0*(1-exp(-1*b*t));
 cout<<tt<<setw(15)<<q<<"\n";
 outfile<<tt<<setw(15)<<q<<"\n";
 }
 outfile.close();
 }
 void oscillatory()
 {
 fstream outfile("oscillatory",ios::out);
 cout<<"t"<<setw(15)<<"q"<<"\n";
 for(tt=0;tt<=300;tt++)
 {
 t=tt/1000;
 w=sqrt(k2-b*b);
 ph=atan(w/b);
 s1=(k/w);
 s2=sin((w*t)+ph);
 s=s1*s2;
 q1=exp(-1*b*t);
 q2=(s*q1);
 q3=(1-q2);
 q=q0*q3;
 cout<<tt<<setw(15)<<q<<"\n";
 outfile<<tt<<setw(15)<<q<<"\n";
 }
 outfile.close();
 }
 };
 void main()
 {
 lcrcharging charge;
 charge.getdata();
 charge.condition();
 if(charge.ch>0)
 {
 cout<<"damped"<<"\n";
 charge.damped();
 }
 else if(charge.ch==0)
 {
 cout<<"critical"<<"\n";
 charge.critical();
 }
 else
 {
 cout<<"oscillatory"<<"\n";
 charge.oscillatory();
 }
 getch();
 clrscr();
 }
```

OUTPUT

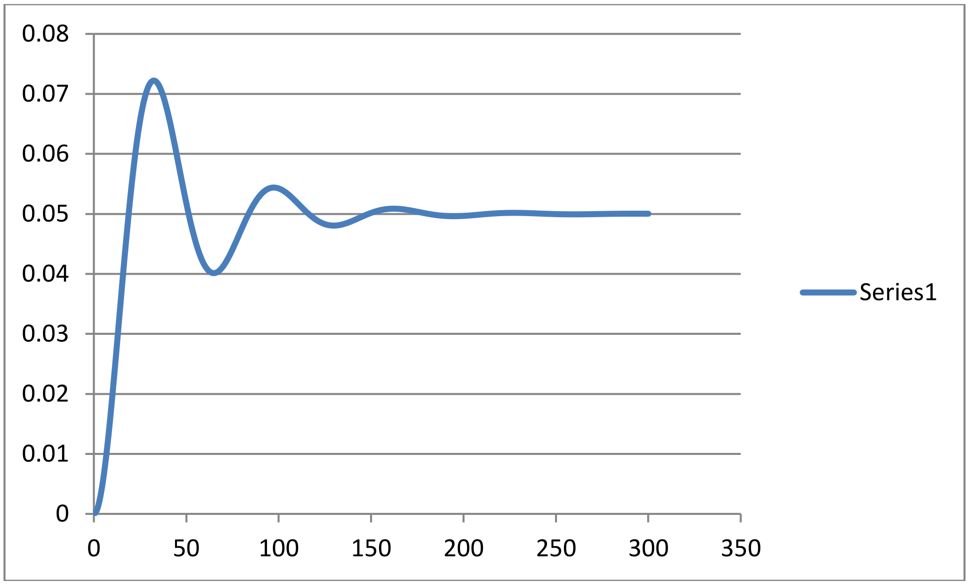
```
Enter voltage,inductance,capacitance,resistance
500
1
.0001
500
damped
```



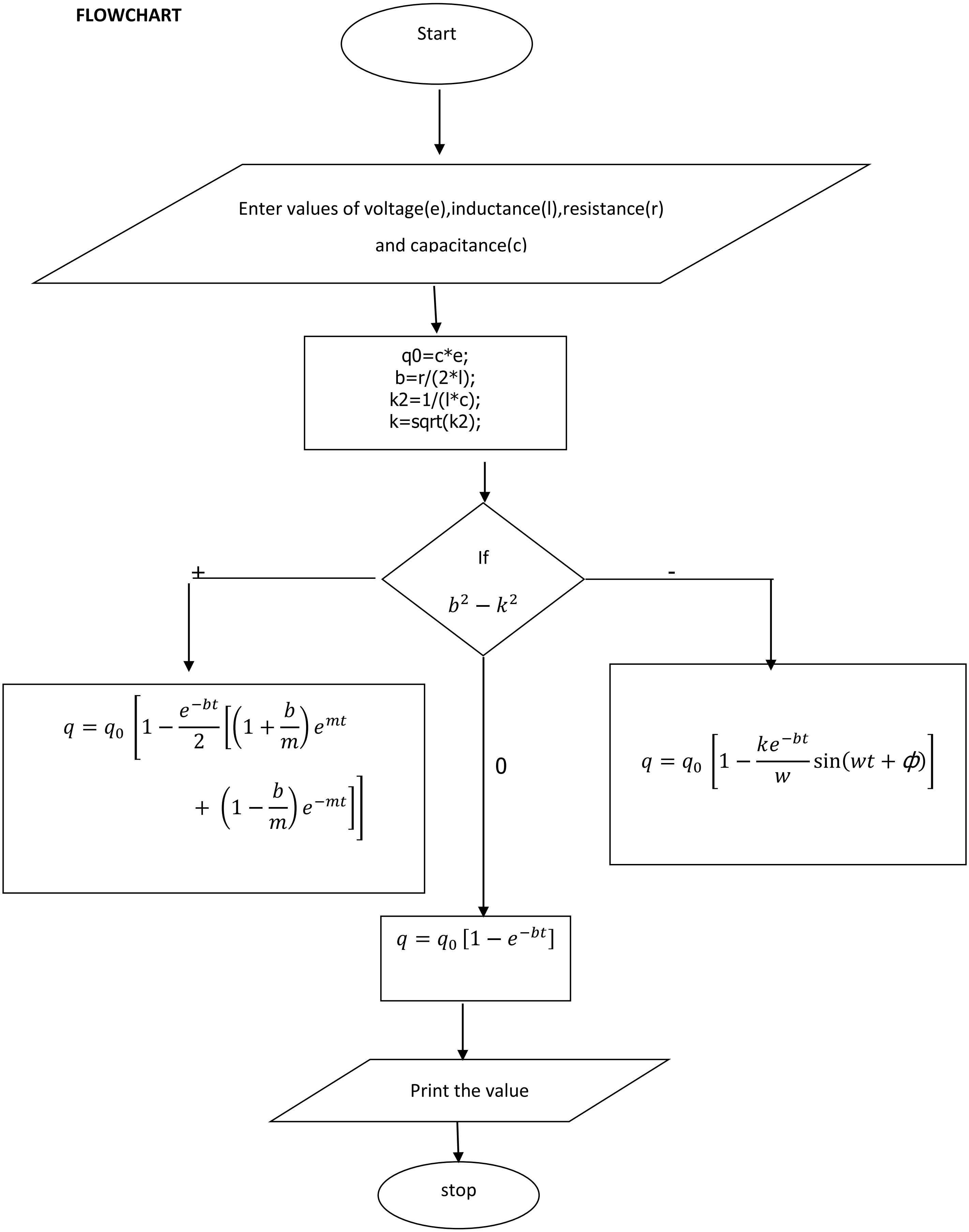
```
Enter voltage,inductance,capacitance,resistance
500
1
.00001
200
Critical
```



```
Enter voltage,inductance,capacitance,resistance
500
1
.0001
50
Oscillatory
```



FLOWCHART


Start
Enter values of voltage(e),inductance(l),resistance(r)
and capacitance(c)
q0=c*e;
b=r/(2*l);
k2=1/(l*c);
k=sqrt(k2);
If
$b^2 - k^2$
+
-
0
$q = q_0 \left[1 - \frac{e^{-bt}}{2}\left[\left(1 + \frac{b}{m}\right)e^{mt} + \left(1 - \frac{b}{m}\right)e^{-mt}\right]\right]$
$q = q_0 \left[1 - \frac{ke^{-bt}}{w}\sin(wt + \phi)\right]$
$q = q_0\,[1 - e^{-bt}]$
Print the value
stop

EXPERIMENT 12

TRAPEZOIDAL RULE

AIM:

To write and execute a C++ program for the numerical integration of a function using Trapezoidal rule

THEORY

Integral of f(x) over the interval [a,b) is the area under the curve f(x) enclosed between the vertical lines x=a, x=b, y=O, It may not be regular area which can be found by well• known rules givenbelow.

Conventional numerical prescription to determine the area of the region is well known and simple. Divide the region into small rectangular stripes, find the area of each .strip then sum over all such strips lying within the given interval. In the simplest possible approximation, each stripe may be chosen as a rectangle of equal width h where *h=b a,*

N is number of 1trips or *the* sub-intervals but of unequal height of

f(xj).

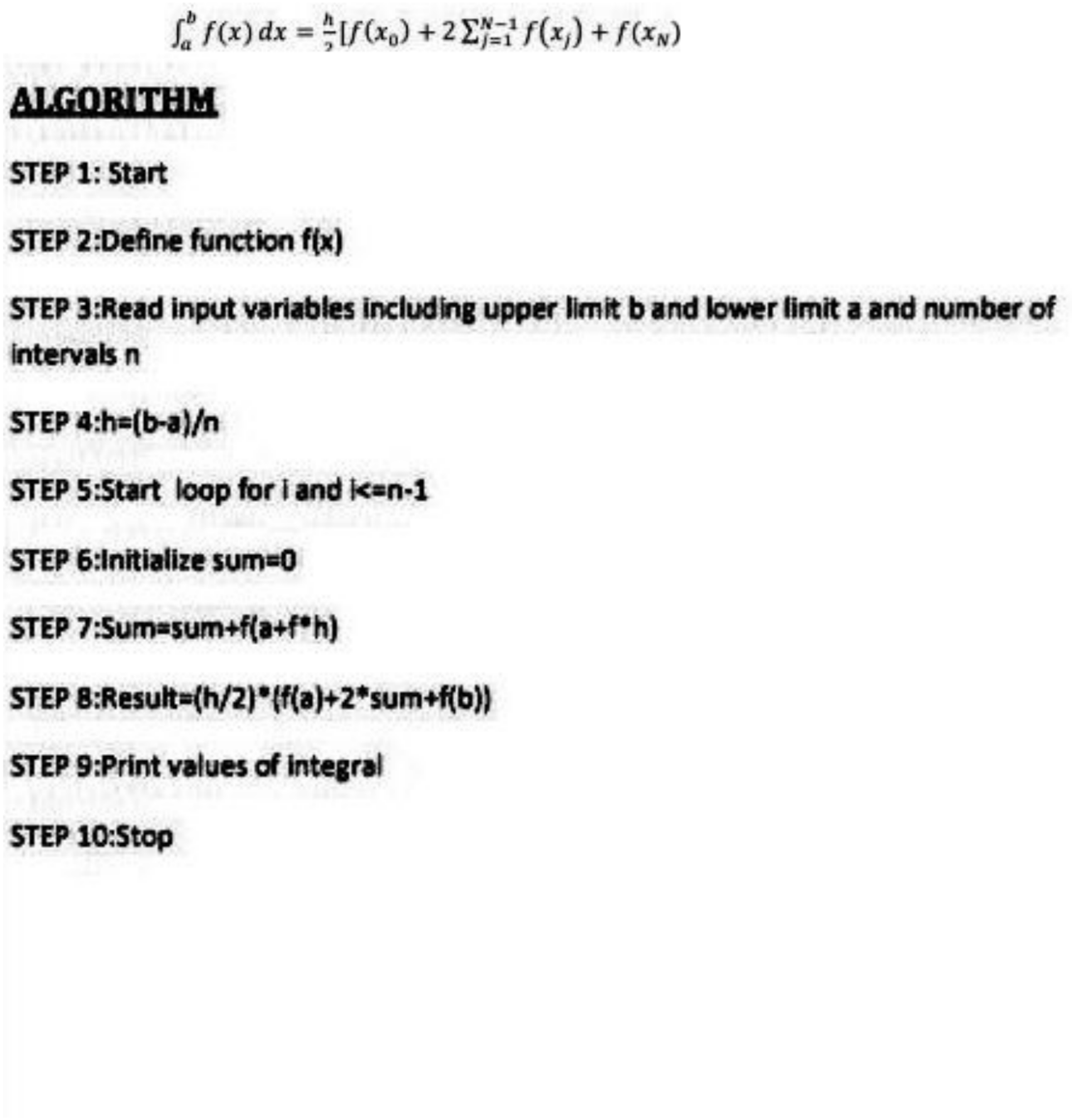
The total area is then given by *f: f(x)dx"' 'I:,f;;if(x1) • h.*

This method obviously represents either under estimation or over estimation.

TRAPEZOIDAL RULE

An improvement over the rectangle rule is to approximate each strip by trapezoidal instead of rectangle. Suppose the interval is divided intoN discrete sub intervals of width h. in each strip the function is approximated by a straight line joining two points at the beginning and at the end of each interval. One the finds the area of each strip as ift(.1)+ f(Xj)

So, the total area is



$$\int_a^b f(x)\,dx = \frac{h}{2}[f(x_0) + 2\sum_{j=1}^{N-1} f(x_j) + f(x_N)$$

ALGORITHM

STEP 1: Start

STEP 2:Define function f(x)

STEP 3:Read input variables including upper limit b and lower limit a and number of intervals n

STEP 4:h=(b-a)/n

STEP 5:Start loop for i and i<=n-1

STEP 6:Initialize sum=0

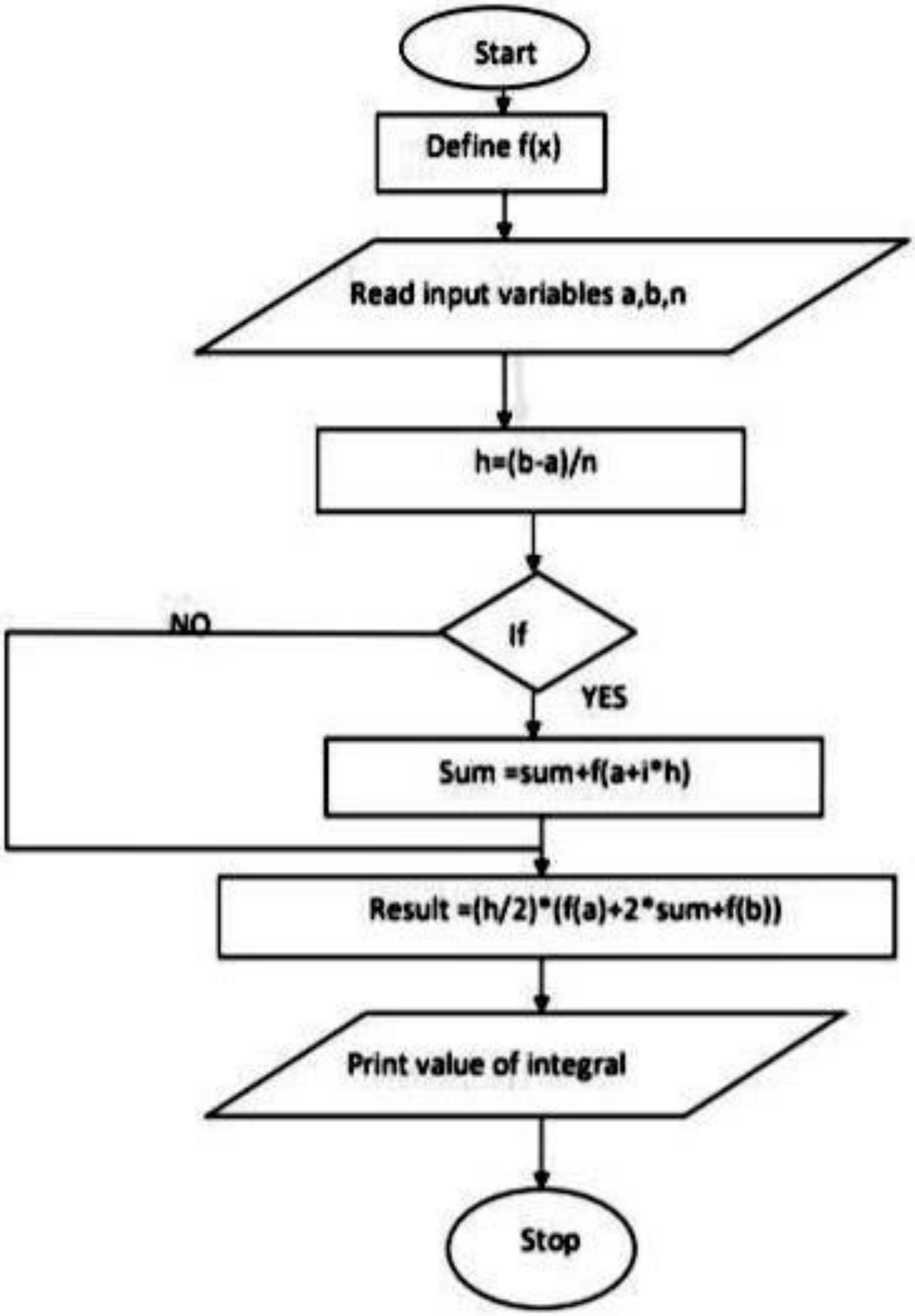
STEP 7:Sum=sum+f(a+f*h)

STEP 8:Result=(h/2)*(f(a)+2*sum+f(b))

STEP 9:Print values of integral

STEP 10:Stop

FLOW CHART

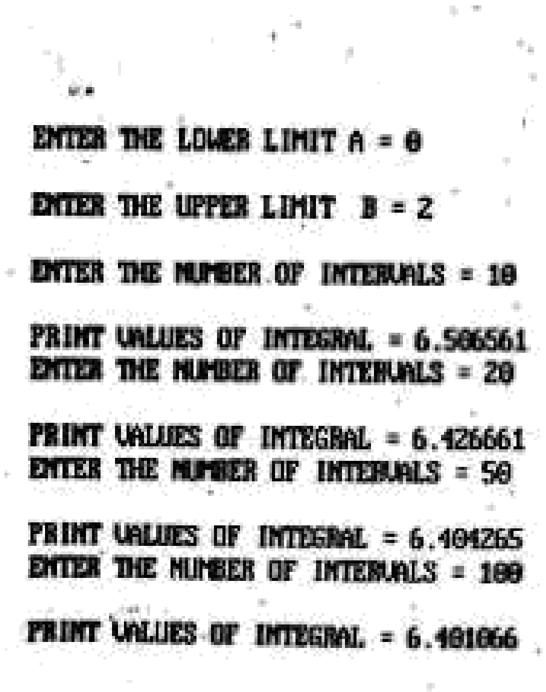

Start
Define f(x)
Read input variables a,b,n
h=(b-a)/n
If
NO
YES
Sum =sum+f(a+i*h)
Result =(h/2)*(f(a)+2*sum+f(b))
Print value of integral
Stop

PROGRAM

```
#include<iostream.h>

#include<conio.h>

#include<math.h>


 float f(float x)

 {

 return ((x)/(1+x));

 }

 void main()

 {

 clrscr();

 int i,n;

 float a,b,h,sum=0,result;

  cout<<"enter lower and upper limit:";
 cin>>a>>b;

 cout<<"enter the number of steps:";

 cin>>n;

 h=(b-a)/n;

 for(i=1;i<=(n-1);i++)

 {
```

```
sum=sum+f(a+i*h);

result=(h/2)*(f(a)+2*sum+f(b));
}

cout<<"\n the result is:"<<result ;
getch();

}
```

OUTPUT



```
ENTER THE LOWER LIMIT A = 0

ENTER THE UPPER LIMIT  B = 2

ENTER THE NUMBER OF INTERVALS = 10

PRINT VALUES OF INTEGRAL = 6.506561
ENTER THE NUMBER OF INTERVALS = 20

PRINT VALUES OF INTEGRAL = 6.426661
ENTER THE NUMBER OF INTERVALS = 50

PRINT VALUES OF INTEGRAL = 6.404265
ENTER THE NUMBER OF INTERVALS = 100

PRINT VALUES OF INTEGRAL = 6.401066
```

RESULT

A C++ program for the numerical integration of a function using Trapezoidal rule is entered, compiled and executed and the value of integral is obtained.