

SuccessClap

UPSC MATHEMATICS STUDY MATERIAL

BOOK- 15 Flow Charts

Updated Version 2.0 (Dated 25-09-2020) (Download from Website)

SuccessClap Android App

STUDY, DISCUSS & CLARIFY YOUR
DOUBTS directly with US



Easy to Read and Take Notes

Complete FREE Study Material , Question Bank in our App

Download Android App: [Click Here](#)

VIDEO COURSES for UPSC MATHEMATICS

[Crash Course for UPSC Mathematics 2020](#)

[22 Weeks Course for Working Professional](#)

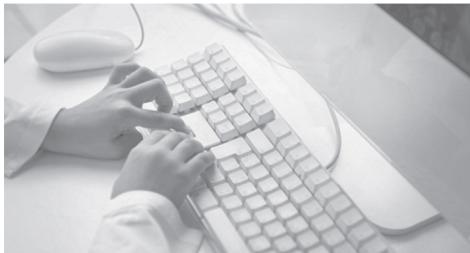
[PREVIOUS YEAR VIDEO & TEXT SOLUTIONS 1992-2019](#)

[Question Bank Solutions](#)

[Full Length Tests, Topic wise Tests](#)

For Exam Related Info and Guidance WhatsApp: 9346856874

Table of Contents



COMPUTATIONAL TECHNIQUE LAB

1. ERRORS CALCULATION

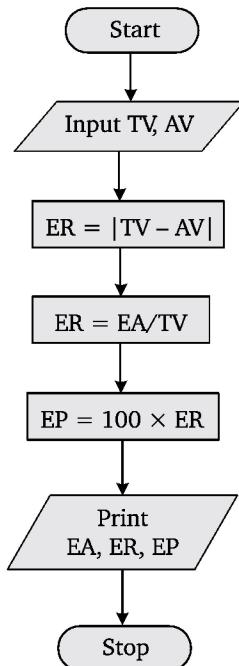
SYMBOLS USED

TV = True Value
AV = Approximate Value
EA = Absolute Error
ER = Relative Error
EP = Percentage Error

ALGORITHM : CALCULATION OF ERRORS

- Step 1.** START
- Step 2.** Input TV, AV
- Step 3.** EA = $|TV - AV|$
- Step 4.** ER = EA/TA
- Step 5.** EP = ER * 100
- Step 6.** Print EA, ER, EP
- Step 7.** STOP

FLOW CHART : ERRORS CALCULATION



PROGRAM : Write a program to calculate the errors.

```
//error calculation
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
    float tv,av,er,ep,ea;
    clrscr();
    printf("\n enter true value \n");
    scanf( "%f",&tv);
    printf("\n enter approximate value\n");
    scanf( "%f",&av);
    ea=fabs(tv-av);
    er=ea/tv;
    ep=100*er;
    printf("\n\n absolute error= %e ",ea);
    printf("\n\n relative error= %e ",er);
    printf("\n\n percentage error= %e ",ep);
    getch();
}
```

Output: ERROR CALCULATION

```
enter true value
37.46235
enter approximate value
37.46
absolute error= 2.349854e-03
relative error= 6.272574e-05
percentage error= 6.272574e-03
```

2. CONVERSION OF DECIMAL TO BINARY NUMBER

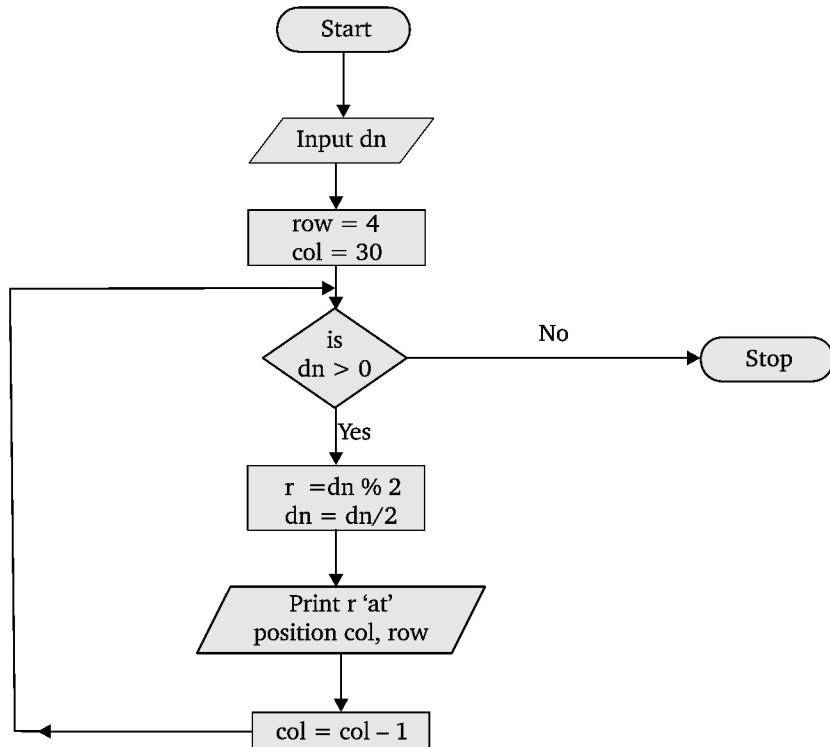
SYMBOLS USED

bn = binary number
dn = decimal number
r = remainder

ALGORITHM : DECIMAL TO BINARY CONVERSION

- Step 1.** START
- Step 2.** Input dn
- Step 3.** row = 4, col = 30
- Step 4.** Perform steps 5 to 8 while (dn > 0)
- Step 5.** r = dn%2
- Step 6.** dn = dn/2
- Step 7.** Print r 'at' col, row
- Step 8.** col = col - 1
- Step 9.** STOP

FLOW CHART : DECIMAL TO BINARY CONVERSION



PROGRAM. Write a C program to convert a given decimal number into binary number.

```

// Decimal to binary conversion
#include<stdio.h>
#include<conio.h>
void main()
{
    int dn,r,row=4,col=30;
    clrscr();
    printf("\n enter decimal number ");
    scanf("%d",&dn);
    printf("\n binary no. is = ");
    while(dn>0)
    {
        r=dn%2;
        dn=dn/2;
        gotoxy(col,row);
        printf("%d",r);
        col--;
    }
    getch();
}
  
```

Output: DECIMAL TO BINARY CONVERSION

enter decimal number 99

binary no. is = 1100011

3. CONVERSION OF BINARY TO DECIMAL NUMBER

SYMBOLS USED

bn = binary number

dn = decimal number

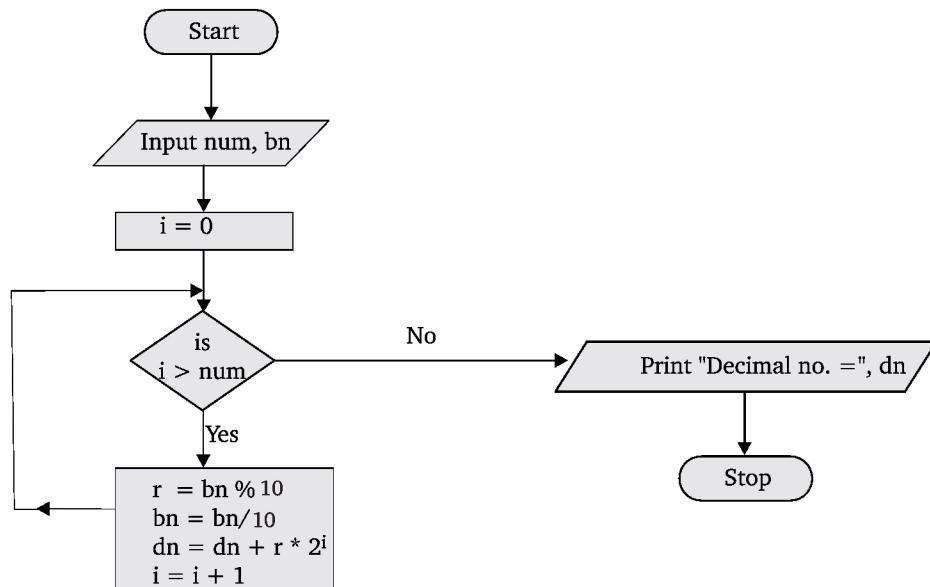
num = number of digits in a binary number

r = remainder

ALGORITHM : BINARY TO DECIMAL CONVERSION

- Step 1.** Start
- Step 2.** Input num, bn
- Step 3.** i = 0
- Step 4.** Perform step 5 to 8 while ($i < num$)
- Step 5.** r = bn % 10
- Step 6.** bn = bn/10
- Step 7.** dn = dn + r * (2^i)
- Step 8.** i = i + 1
- Step 9.** Print "decimal no = ", dn
- Step 10.** Stop

FLOW CHART : BINARY TO DECIMAL NUMBER



PROGRAM. Write a C-program to convert a given binary number to decimal number.

```
// Binary to decimal conversion
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
    int r,i,num;
    long int bn,dn=0;
    clrscr();
    printf("\n enter number of digits\n ");
    scanf("%d",&num);
    printf("\n enter binary number   ");
    scanf("%ld",&bn);
    printf("\n decimal no. is = ");
    for(i=0;i<num;i++)
    {
        r=bn%10;
        bn=bn/10;
        dn=dn +r*pow(2,i);
    }
    printf("%ld",dn);
    getch();
}
```

Output : BINARY TO DECIMAL CONVERSION

enter number of digits

7

enter binary number 1100011

decimal no. is = 99

3. CONVERSION OF DECIMAL NUMBER TO OCTAL NUMBER

SYMBOLS USED

dn = decimal number

on = octal number

ALGORITHM : CONVERT DECIMAL NUMBER INTO EQUIVALENT OCTAL NUMBER

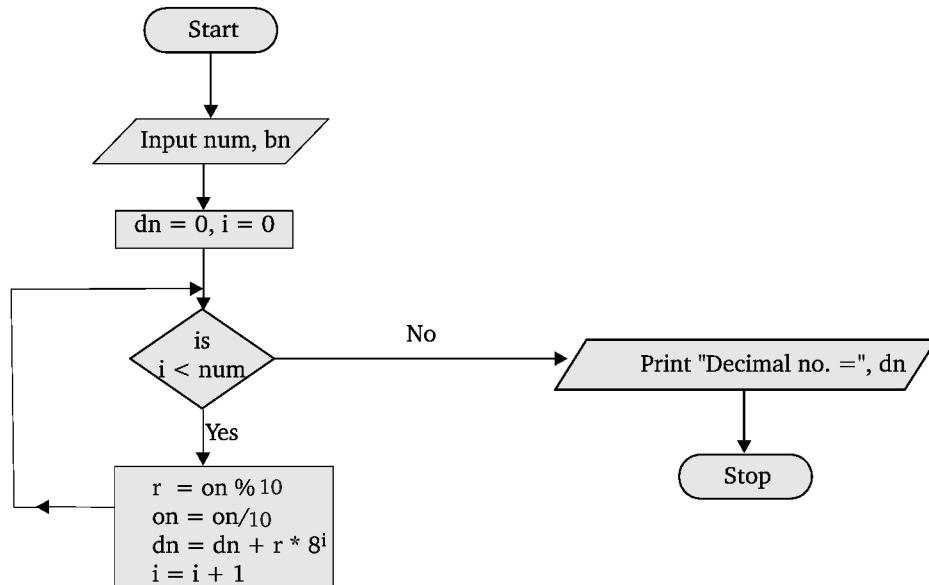
Step 1. Start

Step 2. Input dn

Step 3. on = 0, i = 1

- Step 4.** Perform steps 5 to 8 while ($dn > 0$)
Step 5. $r = dn \% 8$
Step 6. $dn = dn/8$
Step 7. $on = on + r * i$
Step 8. $i = i * 10$
Step 9. Print “Equivalent octal number” on,
Step 10. Stop

FLOW CHART : CONVERSION OF DECIMAL NUMBER TO OCTAL NUMBER



PROGRAM : Write a C program to convert a decimal number into octal number.

// Decimal to octal conversion

```

#include<stdio.h>
#include<conio.h>
void main()
{
    int dn,r,on=0,i=1;
    clrscr();
    printf("\n enter decimal number    ");
    scanf("%d",&dn);
    printf("\n octal no. is = ");
    while(dn>0)
    {
        r=dn%8;
        dn=dn/8;
        on=on+r*i;
        i=i*10;
    }
    printf("%d",on);
  
```

```
    getch();  
}
```

Output : DECIMAL TO OCTAL CONVERSION

enter decimal number 9876
Octal no. is = 23072

4. CONVERSION OF GIVEN OCTAL NUMBER INTO AN EQUIVALENT DECIMAL NUMBER

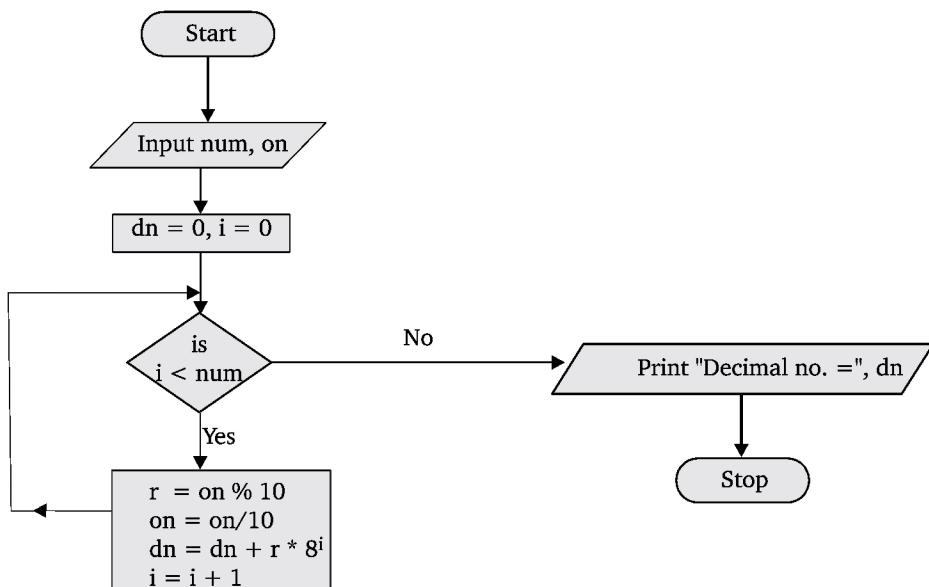
SYMBOLS USED

dn = decimal number
on = octal number
num = number of digits in Octal number

ALGORITHM : CONVERT OCTAL NUMBER INTO DECIMAL NUMBER

- Step 1.** START
- Step 2.** Input “Enter number of digits”, num
- Step 3.** Input “Enter Octal number”, on
- Step 4.** i = 0; dn = 0
- Step 5.** Perform steps 6 to 8 while ($i < num$)
- Step 6.** r = on % 10
- Step 7.** on = on/10
- Step 8.** dn = dn + r * 8^i
- Step 9.** Print “decimal number = ”, dn
- Step 10.** STOP

FLOW CHART : CONVERSION OF OCTAL NUMBER INTO DECIMAL NUMBER



PROGRAM : Write a C program to convert a given octal number into a decimal number.

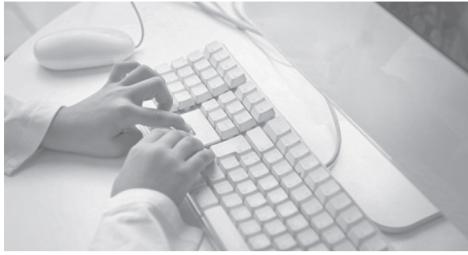
// Octal to decimal conversion

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
    int r,i,num;
    int on,dn=0;
    clrscr();
    printf("\n enter number of digits\n ");
    scanf("%d",&num);
    printf("\n enter octal number   ");
    scanf("%d",&on);
    printf("\n decimal no. is = ");
    for(i=0;i<num;i++)
    {
        r=on%10;
        on=on/10;
        dn=dn+r*pow(8,i);
    }
    printf("%d",dn);
    getch();
}
```

Output : OCTAL TO DECIMAL CONVERSION

```
enter number of digits 4
enter octal number 1727
decimal no. is = 983
```

#####



COMPUTATIONAL TECHNIQUE LAB

1. BISECTION METHOD

SYMBOL USED

x_1, x_2 = Initial approximations in which root lies.

err = allow error

x_3 = New approximation of the root in each iteration

itr = a counter which keeps track of the no. of iterations performed.

ALGORITHM : BISECTION METHOD

Step 1. Start

Step 2. Define $f(x)$

Step 3. Input x_1, x_2 , err

Step 4. If $(f(x_1) * f(x_2)) < 0$

print “Initial approximations are correct”

else

print “Initial approximations are not correct”

go to step 3

Step 5. itr = 1

Step 6. Perform steps 7 to 10 while ($|x_2 - x_1| > \text{err}$)

Step 7. $x_3 = (x_1 + x_2)/2$

Step 8. if $f(x_3) = 0$

Print “Solution converges in iteration”, i

Print “Root =”, x_3

go to step 13

Step 9. If $(f(x_1) * f(x_3)) < 0$

$x_2 = x_3$

else

$x_1 = x_3$

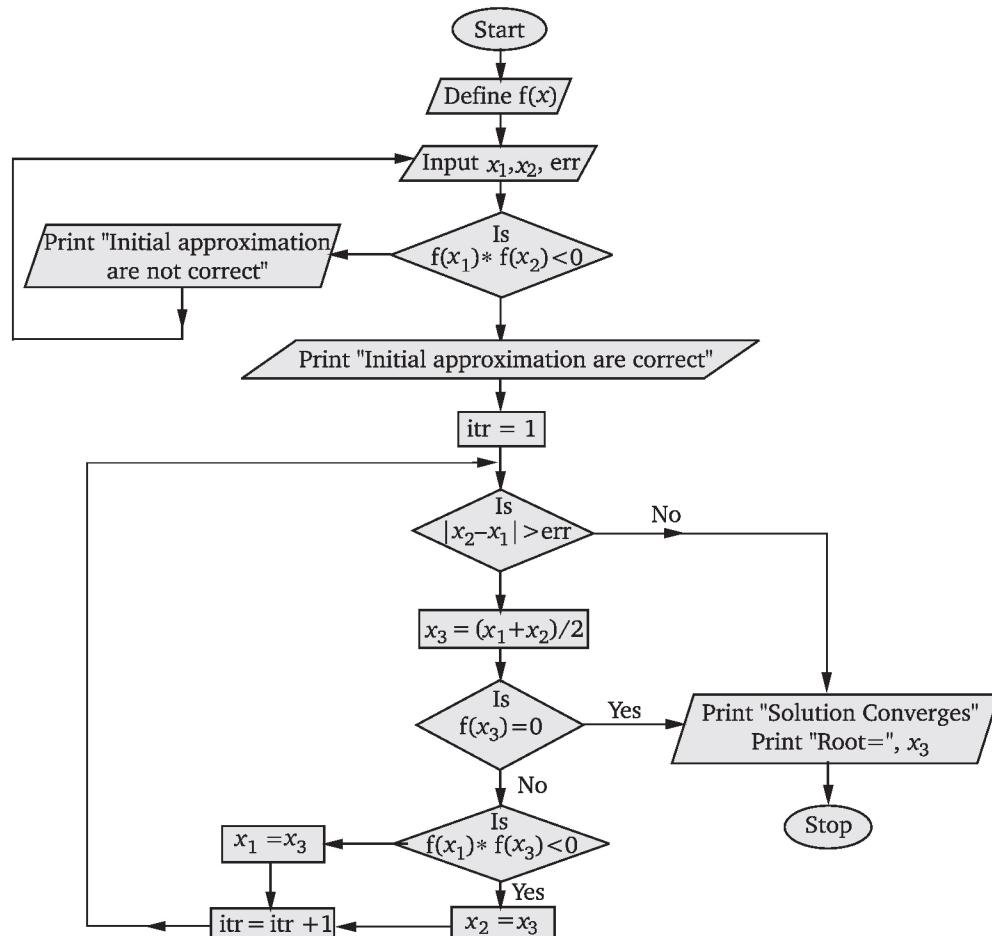
Step 10. itr = itr + 1

Step 11. Print “Solution converges in iteration”, i

Step 12. Print “Root=”, x_3

Step 13. STOP

FLOW CHART : BISECTION METHOD



PROGRAM . Following is a C program to find the root of the equation $f(x)=x^3-4 * x-9$ by Bisection method.
 (UPTU B.Tech. 2003)

```

// Bisection Method
/*Bisection Method to find root of x*x*x-4*x-9=0 */
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<process.h>

float f(float x)
{
    return (x*x*x-4*x-9) ;
}
  
```

```

void main()
{
    clrscr();
    int itr=1;
    float x1,x2,x3, err;

    start: printf("Enter the value of x1, x2,"
                  "allowed error\n");
    scanf("%f%f%f", &x1,&x2,&err);

    if(f(x1)*f(x2)<0)
        printf("\n initial approximations are correct\n");
    else
        { printf("\n initial approximations are not correct\n");
          goto start;
        }
    while(fabs(x2-x1)>err)
    {
        x3=(x1+x2)/2;
        if(f(x3)==0)
        {
            printf("\nsolution converges in iteration=%d",itr);
            printf("\n Root=%f",x3);
            exit(0);
        }
        if(f(x1)*f(x3)<0)
            x2=x3;
        else
            x1=x3;
        itr++;
    }
    printf("\n solution converges in iteration=%d",itr);
    printf("\n Root=%f",x3);
    getch();
}

```

Output: BISECTION METHOD TO FIND ROOT OF $x^3+x^2-4*x-9=0$

Enter the value of x1, x2, allowed error

3 2 .0005

initial approximations are correct

solution converges in iteration =12

Root=2.7065

LAB ASSIGNMENT : BISECTION METHOD

1. Write a C program to find the root of the equation by using Bisection method correct to two places of decimal.

$$f(x) = x^3 - x - 11 = 0$$

Hint: Define a function $f(x) = x * x * x - x - 11$

Input: $x_1 = 2, x_2 = 3, \text{err} = 0.5 \times 10^{-2} = 0.005$

Output: Root = 2.38

2. Write C program to find root for the given function by Bisection method correct to two decimal places.

$$f(x) = 3x - \sqrt{1 + \sin x}$$

Hint: Define $f(x) = 3 * x - \text{sqrt}(1 + \sin(x))$

Input : $x_1 = 0, x_2 = 1, \text{err} = 0.005$

Output: Root = 0.39

3. Write a C program to find root using Bisection method correct to two decimal places for following function.

$$f(x) = x^2 - 4x - 10 = 0$$

Hint: Define $f(x) = x * x - 4 * x - 10$

Input $x_1 = 5, x_2 = 6$ and $\text{err} = 0.005$

Output Root = 5.738

4. Write a C program to find root using Bisection method correct to two decimal places for following function.

$$x \log_{10} x = 1.2$$

Hint: Define $f(x) = x * \log_{10}(x) - 1.2$

Input $x_1 = 2, x_2 = 3, \text{err} = 0.005$

Output Root = 2.74.

2. ITERATION METHOD

SYMBOLS USED

x_1 = Initial approximations of the root

x_2 = New approximation of root in each iteration

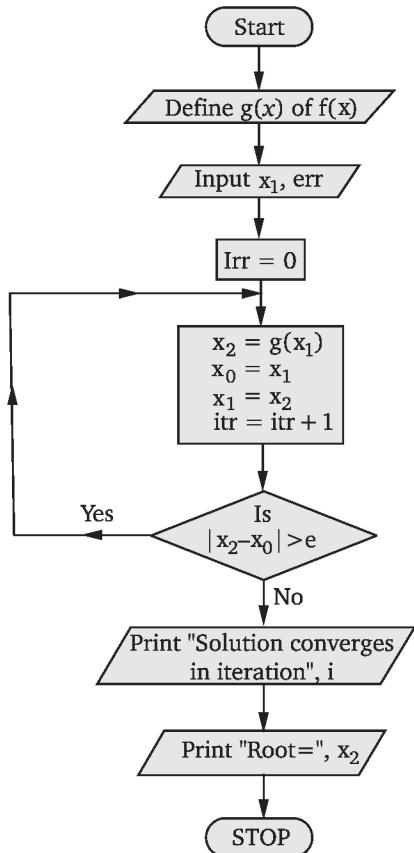
x_0 = Used to store the previous value of x_1

ALGORITHM : ITERATION METHOD

- Step 1.** Start
- Step 2.** For given function $f(x)$ defined $g(x)$
- Step 3.** Input x_1, err
- Step 4.** $\text{itr} = 0$
- Step 5.** Perform steps 6 to 9 while ($|x_2 - x_0| > \text{err}$)
- Step 6.** $x_2 = g(x_1)$

- Step 7.** $x_0 = x_1$
Step 8. $x_1 = x_2$
Step 9. $\text{itr} = \text{itr} + 1$
Step 10. Print “Solution converges in iteration”, i
Step 11. Print “Root=”, x_2
Step 12. STOP

FLOW CHART : ITERATION METHOD



PROGRAM. Following is a C program to find the root of the equation $f(x)=2x-\log_{10} x -7 = 0$ by using iteration method correct to 4 decimal places.

We have $f(x)=2x-\log_{10} x -7 = 0$
 It can be written as

$$x = \frac{1}{2}(\log_{10} x + 7)$$

Define $g(x) = (\log_{10}(x)+7)/2$

Program for iteration method

```
/*Iteration Method to find root of 2x-log10x-7=0 */
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<process.h>

float g(float x)
{
    return((log10(x)+7)/2);
}

void main()
{
    clrscr();
    int itr=0;
    float x1,x2,xo, err;

    printf("Enter the value of x1, allowed error\n");
    scanf("%f%f", &x1,&err);
    do
    {
        x2=g(x1);
        xo=x1;
        x1=x2;
        itr++;
    } while(fabs((x2-xo)>err));

    printf("\n solution converges in iteration =%d",itr);
    printf("\n Root=%f",x2);
    getch();
}
```

Output: ITERATION METHOD TO FIND ROOT OF 2x-log₁₀x-7=0

```
Enter the value of x1, allowed error
3      .00005

solution converges in iteration = 5

Root= 3.789278
```

LAB ASSIGNMENTS : ITERATION METHOD

1. Write a C program to find root of the equation $f(x) = x^3 - 2x + 1 = 0$ using iteration method correct to 3 decimal places.

Hint : $f(x) = x^3 - 2x + 1 = 0$

$$g(x) = x = \frac{x^3 + 1}{2}$$

Define $g(x) = (x * x * x + 1)/2$
Input $x_1 = 0, \text{ err} = 0.0005$
Output Root = 0.6175

2. Write C program to write real root of the equation $\cos x = 3x - 1$ correct to three decimal places using iteration method.

Hint : $f(x) = \cos x = 3x - 1$
 $x = \frac{\cos x + 1}{3} = g(x)$

Define $g(x) = (\cos(x) + 1)/3$
Input : $x_1 = 0, \text{ err} = 0.0005$
Output : Root = 0.607

3. Write C program to find real root of the equation by iteration method upto 4 decimal places

$$f(x) = x^3 + x^2 - 1 = 0$$

Hint : $x = \frac{1}{\sqrt[3]{1+x}}$

Define $x = (1/\sqrt[3]{1+x})$
Input $x_1 = 0.5, \text{ err} = 0.00005$
Output Root = 0.7548

3. REGULA-FALSI METHOD

SYMBOLS USED

x_1, x_2 = Initial approximations near to the root

x_3 = New approximation in each iteration

err = allowed error

max itr = Maximum no. of iterations

itr = iteration counter

ALGORITHM : REGULA-FALSI METHOD

Step 1. Start

Step 2. Define function $f(x)$

Step 3. Input $x_1, x_2, \text{err}, \text{maxitr}$

Step 4. If $(f(x_1) * f(x_2)) \leq 0$

Print "Initial approximation are correct"

else

Print "Initial approximation are not correct".

goto step 3

Step 5. $\text{itr} = 1$

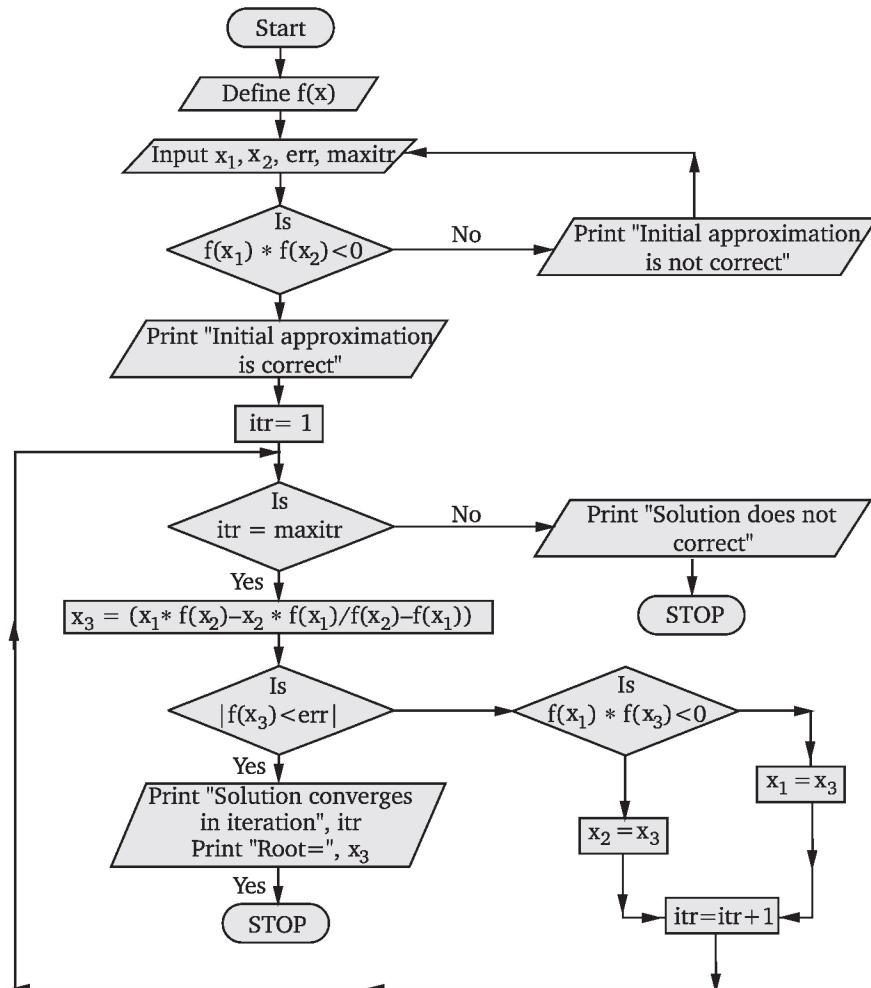
Step 6. Repeat steps 7 to 9 while ($\text{ite} \leq \text{maxitr}$)

Step 7. $x_3 = (x_1 * f(x_2) - x_2 * f(x_1)) / (f(x_2) - f(x_1))$

```

if (|f(x3) < err |)
    Print "Solution converges in iteration", itr
    Print "Root=", x3
    goto step 11
Step 8. if (f(x1)*f(x3)) < 0
    x2 = x3
else
    x1 = x3
Step 9. itr = itr + 1
Step 10. Print "Solution does not converges, iterations not sufficient"
Step 11. STOP
    
```

FLOW CHART : REGULA-FALSI METHOD



PROGRAM.

Following is a C program to find the root of the equation $f(x) = \cos x - xe^x$ by using Regula-Falsi method.

(UPTU B.Tech. 2002)

REGULA FALSI METHOD

```
/* REGULA FALSI METHOD FOR cos(x)-xex */
#include<stdio.h>
#include<conio.h>
#include<math.h>
float f(float x)
{
    return cos(x)-x*exp(x);
}
void main()
{
    int itr, maxitr;
    float x1,x2,x3,err;
    clrscr();
    start: printf("enter the value of x1,x2, allowed error,
maximum iteration\n");
    scanf("%f%f%d", &x1, &x2, &err, &maxitr);

    if(f(x1)*f(x2)<0)
        printf("\n initial approximation are correct");
    else
    {
        printf("\n initial approximation are not correct\n");
        goto start;
    }
    for(itr=1;itr<=maxitr;itr++)
    {
        x3= (x1*f(x2)-x2*f(x1))/(f(x2)-f(x1));
        if(fabs(f(x3))<=err)
        {
            printf("\n Solutions converges in iterations =%d",
                   itr);
            printf("\n Root=%f",x3);
            getch();
            exit(0);
        }
        if (f(x1)*f(x3)<0)
            x2=x3;
        else
            x1=x3;
    }
    printf("\n Solution does not converge,"
           "iterations not Sufficient\n");
    getch();
}
```

Output: REGULA FALSI METHOD FOR COS(X)-X*eX

enter the value of x_1, x_2 , allowed error, maximum iteration

0 1 .00005 20

Solutions converges in iterations = 10

Root = 0.517748

LAB ASSIGNMENTS

1. Write C program to find root of the equation $x^3 - 9x + 1 = 0$ by Regula-Falsi method.

Hint : Define $f(x) = x * x * x - 9 * x + 1$

Input $x_1 = 2, x_2 = 3, \text{err} = 0.00005, \text{maxitr} = 8$

Output Root = 2.9428

2. Write C program to find root of the equation $xe^x - 3 = 0$ by Regula-Falsi method correct to three decimal places.

Input : $x_1 = 1, x_2 = 1.5, \text{err} = 0.0005, \text{maxitr} = 10$

Output : Root = 1.049

3. Write C program to find real root of the equation by Regula-Falsi method

$$f(x) = x^2 - \log_e x - 12$$

Hint : Define $f(x) = x * x - \log(x) - 12$

Input : $x_1 = 3, x_2 = 4, \text{err} = 0.0005, \text{maxitr} = 8$

4. NEWTON-RAPHSON METHOD

SYMBOLS USED

x_1 = Initial approximation of the root

x_2 = New approximation of the root in each iteration

itr = iteration counter

err = allowed error

x_0 = old value of x_1 and x_1 is changed in each iteration

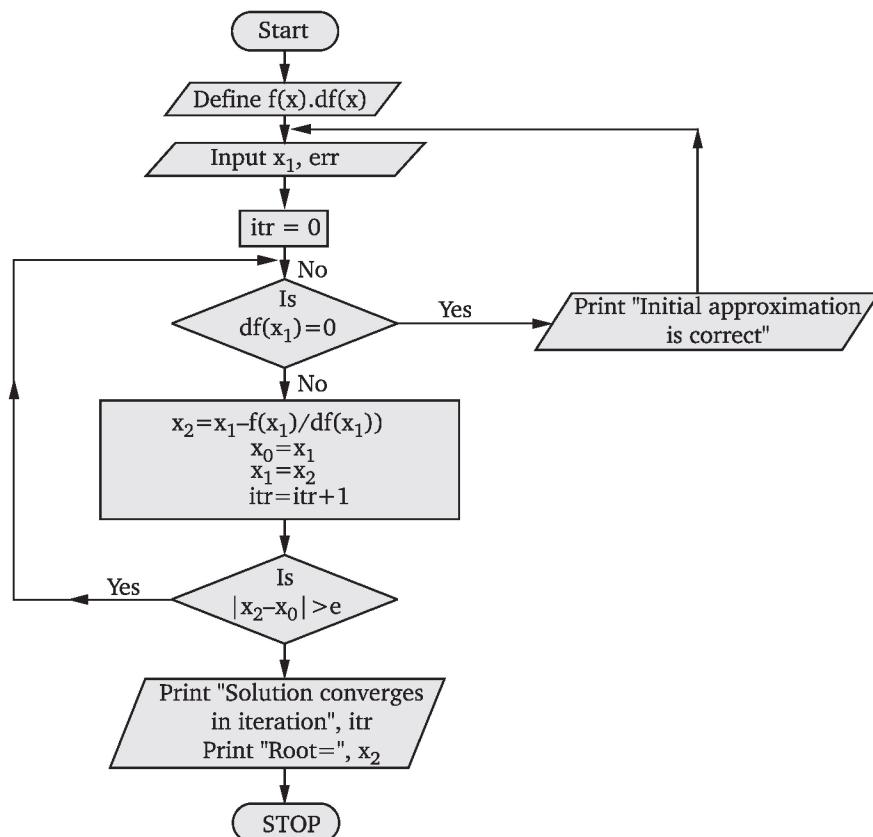
$df(x)$ = first derivative of $f(x)$

ALGORITHM : NEWTON-RAPHSON METHOD

- Step 1.** Start
- Step 2.** Define function $f(x)$ and $df(x)$
- Step 3.** Input x_1, err
- Step 4.** $itr = 0$
- Step 5.** Repeat steps 6 to 10 until ($|x_2 - x_0| < e$)

- Step 6.** if $df(x_1) = 0$
 Print "Initial approximation is not correct"
 goto step 3
- Step 7.** $x_2 = x_1 - (f(x_1)/ df(x_1))$
- Step 8.** $x_0 = x_1$
- Step 9.** $x_1 = x_2$
- Step 10.** $itr = itr + 1$
- Step 11.** Print "Solution converges in iterations", itr
- Step 12.** Print "Root=", x_2
- Step 13.** STOP

FLOW CHART : NEWTON-RAPHSON METHOD



PROGRAM. Following is a C program to find the root of the equation $f(x) = x^4 - x - 10$ by using Newton-Raphson method.

$$\begin{aligned} \text{We have } f(x) &= x^4 - x - 10 \\ \Rightarrow df(x) &= 4x^3 - 1 \end{aligned}$$

NEWTON RAPHSON METHOD

```
/*NEWTON RAPHSON METHOD TO FIND ROOT OF x4-x-10 */

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

float f(float x)
{
    return x*x*x*x-x-10;
}
float df (float x)
{
    return 4*x*x*x-1;
}
void main ()
{
    int itr=0;
    float x1,x2,err,xo;
    clrscr();

    start:     printf("\n Enter the value of x1,allowed error\
n\n");
    scanf("%f%f",&x1,&err);

    do
    {
        if(df(x1)==0)
        {
            printf("\n Initial approximation is not correct");
            goto start;
        }
        x2=x1-f(x1)/df(x1);
        xo=x1;
        x1=x2;
        itr++ ;
    } while(fabs(x2-xo)>err);
```

```

        printf("\n Solution converges in iteration = %d\n",itr);
        printf("\nRoot =%f",x2);
        getch();
    }
}

```

Output : NEWTON RAPHSON METHOD TO FIND ROOT OF x⁴-x-10

```

Enter the value of x1,allowed error
1.6 .0005
Solution converges in iteration = 4
Root=1.855585

```

LAB ASSIGNMENTS : NEWTON-RAPHSON METHOD

- 1.** Write C program to find root of the equation $f(x) = x^3 - 2x - 5$ correct to 3 decimal places by Newton-Raphson method.

Hint : Define $f(x) = x * x * x - 2 * x - 5$

and $f(x) = 3 * x * x - 2$

Input $x_1 = 2$, err = 0.0005

Output Root = 2.09455

- 2.** Write C program to find root of the equation $x \log_{10} x = 1.2$ by Newton-Raphson method correct to four decimal places.

Hint : Define $f(x) = x * \log_{10}(x) - 1.2$

$df(x) = \log_{10}(x) + 0.4343$

Input $x_1 = 2$, err = 0.00005

Output Root = 2.7408

- 3.** Write C program to find square root of 12 correct to three decimal places by Newton-Raphson method.

Hint : Define $f(x) = x^2 - 12 = 0$

$f(x) = x * x - 12$

$df(x) = 2 * x$

Input $x_1 = 3.4$, err = 0.0005

Output Root = 3.464

5. MULLER'S METHOD

SYMBOLS USED

x_1, x_2, x_3 = Initial approximation

x_4 = New iteration in each step

err = allowed error

itr = iteration counter

ALGORITHM : MULLER'S METHOD

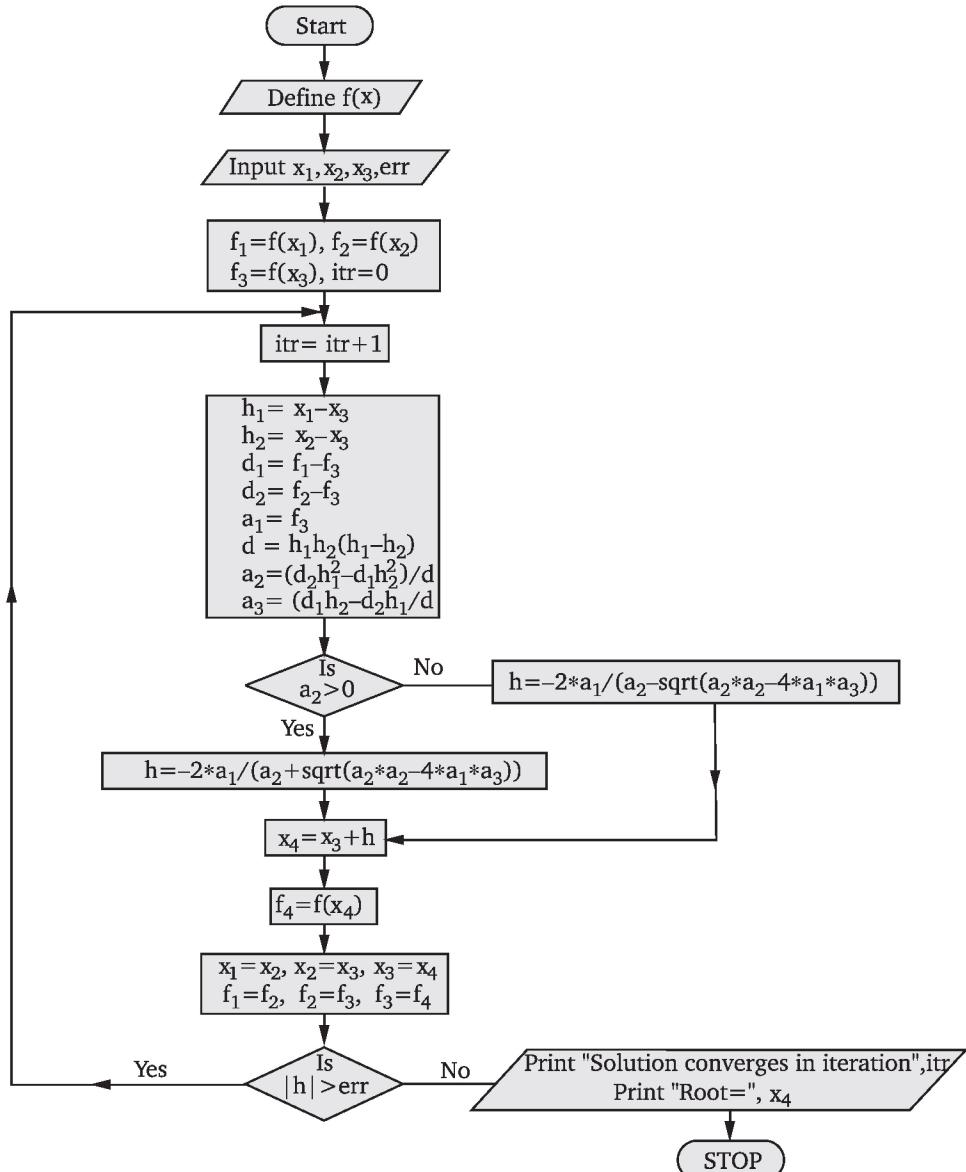
- Step 1.** Start
- Step 2.** Define $f(x)$
- Step 3.** Input x_1, x_2, x_3 , err
- Step 4.** $f_1 = f(x_1), f_2 = f(x_2), f_3 = f(x_3)$, itr = 0
- Step 5.** Repeat steps 6 to 17 until ($|h| < e$)
- Step 6.** itr = itr + 1
- Step 7.** $h_1 = x_1 - x_3, h_2 = x_2 - x_3$
- Step 8.** $d_1 = f_1 - f_3, d_2 = f_2 - f_3$
- Step 9.** $a_1 = f_1$
- Step 10.** $d = h_1 * h_2 * (h_1 - h_2)$
- Step 11.** $a_2 = (d_2 * h_1^2 - d_1 * h_2^2) / d$
- Step 12.** $a_3 = (d_1 * h_2 - d_2 * h_1) / d$
- Step 13.** if ($a_2 > 0$)

$$h = -2 * a_1 / (a_2 + \sqrt{a_2^2 - 4 * a_1 * a_3})$$

 else

$$h = -2 * a_1 / (a_2 - \sqrt{a_2^2 - 4 * a_1 * a_3})$$
- Step 14.** $x_4 = x_3 + h$
- Step 15.** $f_4 = f(x_4)$
- Step 16.** $x_1 = x_2, x_2 = x_3, x_3 = x_4$
- Step 17.** $f_1 = f_2, f_2 = f_3, f_3 = f_4$
- Step 18.** Print "Solution converges in iteration", itr
- Step 19.** Print "Root=", x_4
- Step 20.** STOP

FLOW CHART : MULLER METHOD



PROGRAM. Following is a C program to find the root of the equation $f(x) = \cos x - xe^x$ by Muller's method.

MULLER'S METHOD

```
/* MULLER'S METHOD cos(x)-x*ex */
```

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

```
#include<math.h>

float f(float x)
{
    return cos(x)-x*exp(x);
}

void main()
{
    int itr=0;
    float x1,x2,x3,x4,err,d,d1,d2,h1,h2,a1,a2,a3,h;
    float f1,f2,f3,f4;
    clrscr();

    printf("\n Enter initial approximation x1,x2,x3 \n");
    scanf("%f%f%f",&x1,&x2,&x3);
    printf("\n Enter the allowed error\n");
    scanf("%f", &err);
    f1=f(x1), f2=f(x2), f3=f(x3);
    do
    {
        itr++;
        h1=x1-x3;
        h2=x2-x3;
        d1=f1-f3;
        d2=f2-f3;
        a1=f3;
        d=h1*h2*(h1-h2);
        a2=(d2*h1*h1-d1*h2*h2)/d;
        a3=(d1*h2-d2*h1)/d;
        if(a2>0)
            h=-2*a1/(a2+sqrt(a2*a2-4*a1*a3));
        else
            h=-2*a1/(a2-sqrt(a2*a2-4*a1*a3));
        x4=x3+h;
        f4=f(x4);
        x1=x2, x2=x3, x3=x4 ;
        f1=f2, f2=f3, f3=f4 ;
    } while(fabs(h)>err);

    printf("\n Solution converges in iterations=%d",itr);
    printf("\n\n Root =%f",x4);
    getch();
}
```

Output: MULLER'S METHOD FOR cos(x)-x*ex

Enter initial approximation x1,x2,x3

-1 0 1

Enter the allowed error

.0005

Solution converges in iterations =4

Root = 0.517757

LAB ASSIGNMENTS : MULLER's METHOD

1. Write C program to find the root of the equation $x^3 - x - 4 = 0$ by Muller method correct to 4 decimal places.

Hint : Define $f(x) = x * x * x - x - 4$ **Input :** $x_1 = 0, x_2 = 1, x_3 = 2$

err = 0.00005

Output Root = 1.7963

2. Write a C program to find root of the equation $3x + \sin x - e^x$ by Muller's method.

Hint : Define $f(x) = 3 * x - \sin(x) - \exp(x)$ **Input :** $x_1 = 0.5, x_2 = 1.0, x_3 = 0.0$ **Output :** Root = 0.36042

3. Write C program to find the root of the equation

$$f(x) = x^3 + 2x^2 + 10x - 20 = 0$$
 by Muller method.**Hint :** Define $f(x) = x * x * x + 2 * x * x + 10 * x - 20$ **Input :** $x_1 = 0, x_2 = 1, x_3 = 2, \text{err} = 0.0005$ **Output :** Root = 1.3688

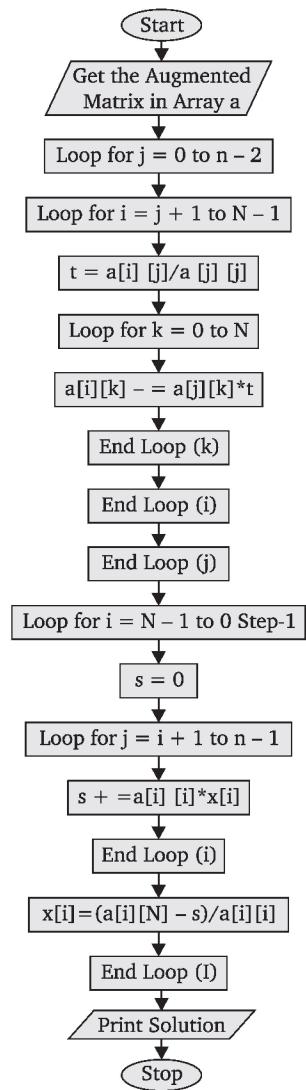
#



COMPUTATIONAL TECHNIQUE LAB

1. GAUSS ELIMINATION METHOD

FLOW CHART FOR GAUSS-ELIMINATION METHOD

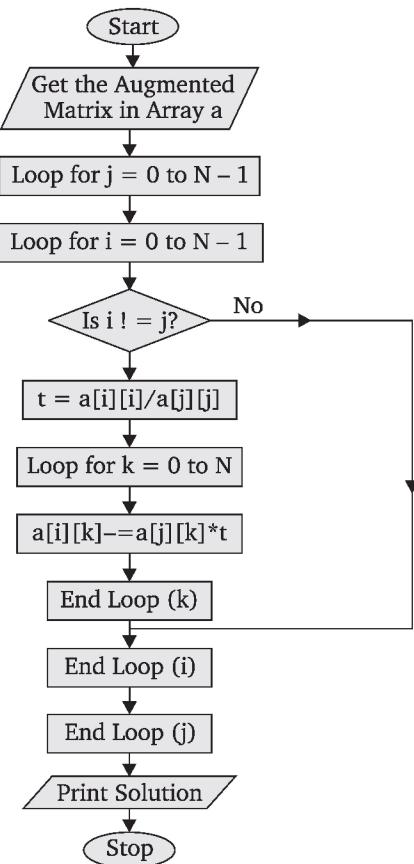


PROGRAM

```
/* Gauss elimination method */
# include <studio .h>
# define N 4
main ( )
{
    float a [N] [N + 1], x [N], t, s;
    int i, j, k;
    printf ("Enter the elements of the"
            augmented matrix rowwise/n");
    for (i = 0; i < N; i++)
        for (j = 0; j < N + 1; J++)
            scanf (" %", &a [i] [j]);
    for (j = 0 ; j < N - 1; j++)
        for i = j + 1; i < N; i++)
    {
        t = a [i] [j] / a [j] [j] ;
        for (k = 0 ; k < n + 1 ; k++)
            a [i] [k] -= a [j] [k] * t;
    }
    printf ("The upper triangular matrix"
            is :-\n") ;
    for ( i = 0; i < N; j++)
    {
        for (j = 0 ; j < N; i++)
            printf (" %8 . 4f ", a [i] [j] ;
        printf (" \n ");
    }
    for (i = N - 1 ; i > = 0; i - -)
    {
        s = 0;
        for ( j = i + 1; j < N; j++)
            s += a [i] [j] *x [j];
        x [i] = (a [i] (N) - S) / a [i] [i] ;
    }
    printf ( "The solution is : - \n") ;
    for (i = 0; i < N; i++)
        printf ("x [% 3d] = % 7.4f\n", i + 1,
x [i]);
    }
```

2. GAUSS-JORDAN METHOD

FLOW CHART



PROGRAM

```

/* Gauss jordan method */

#include <stdio.h>
#define N 3
main ( )
{
    float a [N] [N + 1], t ;
    int i, j, k;
    printf (Enter the elements of the"
            "augmented matrix rowwise\n") ;
    for (i = 0; i < N; i++)
        for (j = 0 ; j < N + 1 ; j ++)

```

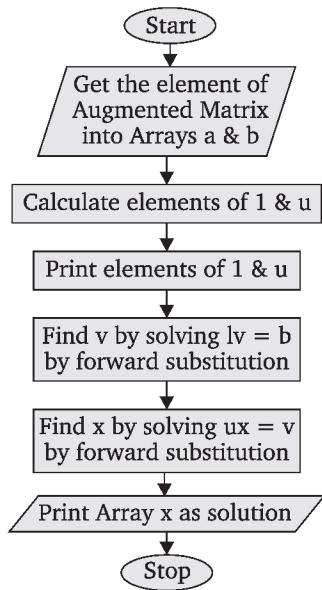
```

        scanf (" % f " , &a [i] [j]) ;
for (j = 0; j < N ; j++)
    for (i = 0; i < N; i++)
        if (i != j)
        {
            t = a [i] [j]/ a [j] [j];
            for (k = 0; k < N + 1; k++)
                a [i] [k] -= a [j] [k] * t ;
        }
printf ("The diagonal matrix is :-\n") ;
for (i = 0; i < N; i++)
{
    for (j = 0; j < N + 1 ; j++)
        print f (% 9.4f ", a [i] [j] ;
    printf (" \n") ;
}
printf (" The solution is : - \n") ;
for (i = 0; i < N; i++)
    printf ("x [% 3d] = % 7.4 f \ n" , i + 1, a [i]
[N] a [i] [i] ;
}

```

3. CROUT'S TRI-ANGULARISATION METHOD

(1) FLOW CHART



PROGRAM

```
/* Crout triangularization method */
#include <studio .h>
#define N 4
typedef float matrix [N] [N];
matrix l, u, a;
float b [N], x [N], v [N];
void urow (int i)
{
    float s ;
    int j, k ;
    for (j = 1; j < N; j++)
    {
        s = 0;
        for (k = 0 ; k < N - 1 ; k++)
            s += u [k] [j] * l [i] [k] ;
        u [i] [j] = a [i] [j] - s ;
    }
}
void lcol (int j)
{
    float s ;
    int i, k ;
    for (i = j + 1 ; i < N; i++)
    {
        s = 0 ;
        for (k = 0; k <= j - 1 ; k++)
            s += u [k] [j] * l [i] [k];
        l [i] [j] = (a [i] [j] - s) / u [j] [j] ;
    }
}
void printmat (matrix x)
{
    int i, j ;
    for (i = 0 ; i < N ; i++)
    {
        for (j = 0; j < N; j++)
            printf (" % 8 . 4f", x [i] [j] );
        printf ("\n") ;
    }
}
main ( )
{
    int i, j, m;
    float s;
    printf ("Enter the elements of augmented"
           "matrix rowwise \n") ;
```

```

for (i = 0; i < N; i++)
{
    for (j = 0; j < N; j++)
        scanf ("% f ", &a [i] [j]) ;
        scanf (" % f ", &b [i]) ;
}
/* 1 and u */
for (i = 0; i < N; i++)
    l [i] [i] = 1 . 0 ;
for (m = 0 ; m < N; m++)
{
    uprow (m);
    if (m < N - 1) lcol (m) ;
}
printf ("\t\tU\n") ; printmat (u);
printf ("\t\tL\n") ; printmat (l);
for (i = 0; i < N; i++)
{
    s = 0;
    for (j = 0; j <= i - 1; j++)
        s += l [i] [j] * v [j];
    v [i] = b [i] - s ;
}
for (i = N - 1 ; i >= 0 ; i - )
{
    s = 0 ;
    for (j = i + 1 ; j < N; j++)
        s += u [i] [j] * x [j] ;
    x [i] = (v [i] - s/u [i] [i]) ;
}
printf , ( "The solution is : - \n") ;
for (i = 0 ; i < N ; i++)
    printf (" X [% 3d] = % 6 . 4f/n" , i + 1, x [i]) ;

```

4. /* LU DECOMPOSITION METHOD */

```

# include <conio .h>
# include <studio. h>
# include <math .h.>
main ( )
{
    float a [15] [15], al [15] [15], b [15], X [15], au [15] [15], z [15];
    float sum, t, big, ab, p ;
    int n, m, li, lj, k, j, i, lk, 13, m2, jj, kpl, kk, l ;
    clrscr ( ) ;
    printf ("enter the value of n/n") ;
    scanf (" % d", &n) ;
    printf ("enter the matrix row wise /n") ;
    for (i = 1 ; j <= n ; i++)
        for (j = 1 ; j <= n; j++)

```

```
scanf ("% f " , & a [i] [j] ) ;
printf ("enter the matrix B\n") ;
for (j = 1 ; j < = n; j++)
    scanf ( " % f" , & b [j] ) ;
for (i = 1 ; i < = n; i++)
for (j = 1 ; j < = n ; j++)
{
    a1 [i] [j] = 0 ;
    au [i] [j] = 0 ;
}
for (i = 1 ; i < = n ; i++)
{
    au [i] [i] = 1 ;
    a1 [i] [1] = a [i] [1] ;
    au [1] [i] = a [1] [i] / a1 [1] [1];
}
for (j = 2 ; j < = n; j++)
{
for (i = j ; i < = n ; i++)
{
    sum = 0 ;
        for (k = 1 ; k ≤ j - 1 ; k++)
            sum=sum+a1 [i] [k] * au [k] [j] ;
        a1 [i] [j] = a [i] [j] - sum ;
}
if ( j ! =n)
(
    for (jj = j + 1; jj < = n; jj++)
    {
        sum=0;
        for (kk = 1; kk < j - 1 ; jj++)
            sum=sum + a1 [j] [kk] * au [kk] [jj];
        au [j] [jj] = (a [j][jj] - sum) / a1 [j][j];
    }
}
z [1] = b [1] / a1 [1] [1] ;
for (i = 2; i < = n; i++)
{
    sum=0;
    for (k = 1; k < i - 1; k++)
        sum=sum+a1 [i] [k] * z [k];
    z [i] = (b [i] - sum)/a1 [i] [i] ;
}
```

```

x [n] = z [n] ;
for (i = 2 ; i <= n ; i++)
{
    l = n - i + 1;
    sum=0;
    for (k = l + 1; k <= n; k++)
        sum= sum+au [l] [k] * x [k];
    x [l] = z [l] - sum;
}
printf ("\n") ;
printf ( "lower triangular matrix \n") ;
for (i = 1; i <= n ; i++)
{
    for (j = 1 ; j <= n; j++)
        printf ("% 5. 2f ", a1 [i] [j]) ;
    printf ("\n") ;
}
printf ( "upper triangular matrix \n") ;
for ( i = 1 ; i <= n ; i++)
{
    for (j = 1; j <= n; j++)
        printf ( "% 5. 3f " , au [i] [j]) ;
    printf ( '\n' ) ;
}
printf ( "solution vector\n") ;
for (i = 1 ; i <= n ; i++)
printf ( "% 5. 2f " , x [i] ) ;
printf ( "\n") ;
getch ( ) ;
}

```

5. SOLUTION OF SYSTEM OF EQUATIONS USING GAUSS SEIDAL METHOD

```

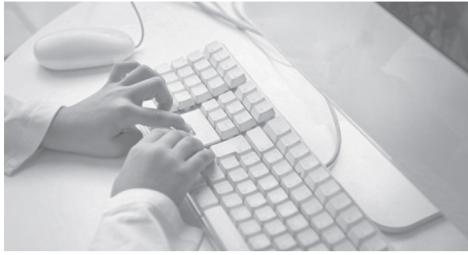
*/
#include <conio .h>
#include <studio .h>
#include <math .h>
main ( )

{
    float a [15] [15] , b [15] , x [15] , oldx [15], eps,
c , big, sum;
    int n, niter1, i, j, k, l, ii ;
    clrscr ( ) ,
    printf ( "enter the value of N, NITHER, ESP\n") ;
    scanf ( "% d% d% f" , &n, &niteral, &eps) ;

```

```
        printf ("enter the matrix A\n") ;
        for (i = 1 ; i <= n; i++)
            for (j = 1; j <= n; j++)
                scanf ("%f", a[i][j]);
            scanf ("enter the array B\n") ;
            for (i = 1; i <= n; i++)
                scanf (" % f ", & b [i] );
            printf ("enter the array 01dx1n"
            for (i = 1; i <= n; i +);
            {
                for (j = 1; j <= n, j + 1)
                {
                    printf (" % f ", a [i] , [j] ) ;
                }
                print (" % / n") ;
            printf (" the array B/n") ;
            for (i = 1, i <= n; i +)
            {
                x (i) = 0/dx [i] ;
                printf ("% f ", b [i]) ;
            }
            printf ("\n") ;
            for (i = 1 ; i <= n; i +) ;
            {
                for (i = 1 ; i <= n iter ; i +)
                {
                    sum = 0
                    for (j = 1 : j <= n ; j + j)
                    if ([i - j] ; = 0)
                        sum = sum + a [i] [j] *x [j] ;
                    x [i] = b [i] - sum/ a [i] [i] ;
                }
            printf (n iter = % d, i) ;
            for (i = 1 ; i <= n; i +)
            printf (" % 12.6 f" , x [i]) ;
            printf ("\n") ;
            big = abb (x [i] - old x [i]) ;
            for (k = 2; k <= n, k++)
            {
                c = abb (xck) - old x [k] ;
                if (big <= c)
                    big = c;
            }
            For (i = 1; i <= n ; i++)
            old x [i] = x [i] ;
        }
    }
```

##/#/#/#



COMPUTATIONAL TECHNIQUE LAB

1. NEWTON'S FORWARD INTERPOLATION FORMULA

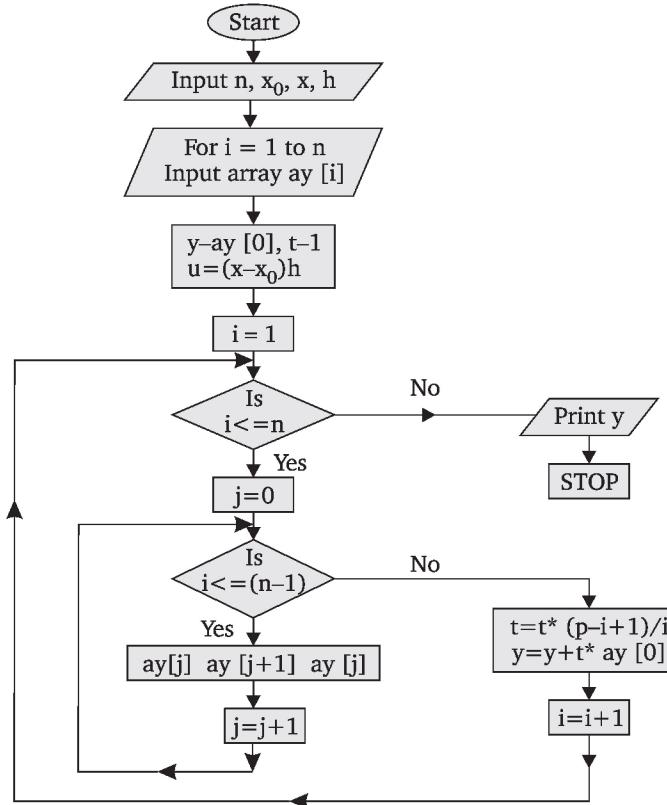
SYMBOLS USED

x_0 = initial value of x
 h = length of interval
 n = number of subintervals
 x = value of x at which we have to find the value of y
 y = value of y at x
 ay = array to store the different values of y
 t = temporary variable

ALGORITHM : NEWTON'S FORWARD INTERPOLATION FORMULA

- Step 1.** Start
- Step 2.** Input n , x_0 , h , x
- Step 3.** Input values of ay
- Step 4.** $y = ay[0]$, $t = 1$
- Step 5.** $u = (x - x_0)/h$
- Step 6.** For $i = 1$ to n
- Step 7.** For $j = 0$ to $(n-i)$
- Step 8.** $y[j] = y[j+1] - y[j]$
- Step 9.** End of j loop.
- Step 10.** $t = t * (p - i + 1)/i$
- Step 11.** $y = y + t * ay[0]$
- Step 12.** End of i loop
- Step 13.** Print y
- Step 14.** STOP

FLOW CHART : NEWTON'S FORWARD INTERPOLATION FORMULA



PROGRAM .

Following is a program to show the Newton's forward interpolation formula.

NEWTON'S FORWARD INTERPOLATION FORMULA (UPTUMCA-2004)

```

#include<stdio.h>
#include<conio.h>
void main ( )
{
    clrscr ( );
    float ay [30], x0, h, x, y, t = 1, u;
    int n, i, j;
    printf ("Enter the value of n\n");
    scanf ("%d", &n);
    printf ("Enter the initial value of x\n");
    scanf ("%d", &x0);
    printf ("\nEnter length of each interval\n");
    scanf ("%f", &h);
    for (i = 0; i < n; i++)
        
```

```

    {      print f ("Enter the value of y (%d) = " , i);
           scan f ( "%f" , & ay [i]);
    }
    printf("Enter the value of x for which value of y is wanted \n");
    scanf ("%f" , &x)';
    y=ay [0];
    u=(x-x0)/h;
    for (i=1; i<=n; i++)
    {
        for (j=0; j<=n-1; j++)
            ay [j] = ay [j+1]-ay [j];
        t=t* (u-i+1)/i;
        y=y+t*ay [0]
    }
    printf ("\n Value of y at x=% .2f " , x, y); getch ();
}

```

Output : NEWTON'S FORWARD INTERPOLATION FORMULA

```

Enter the value of n
6
Enter the initial value of x
0
enter length of each interval
1
Enter the value of y (0) = 1
Enter the value of y (1) = 3
Enter the value of y (2) = 11
Enter the value of y (3) = 31
Enter the value of y (4) = 69
Enter the value of y (5) = 131
Enter the value of y (6) = 223
Enter the value of x for which value of y is wanted
3.4
Value of y at x = 3.40 is 43.70

```

LAB ASSIGNMENT : NEWTON'S FORWARD INTERPOLATION FORMULA :

1. Write a C program for the Newton's forward interpolation formula to find the value of y at $x = 2.7$ from the following data

x	1	2	3	4	5	6	7	8
$f(x)$	1	8	27	64	125	216	343	512

Hint: Input $x_0 = 1, h = 1, n = 7$

Output $f(2.7) = 50.65$

2. Write a C program to find $f(3.4)$ using the following values by Newton's forward interpolation formula

x	0	1	2	3	4	5	6
f(x)	1	3	11	31	69	131	223

Hint: Input $x_0 = 0, h = 1, n = 6$

Output $f(3.4) = 43.704$

2. NEWTON'S BACKWARD INTERPOLATION FORMULA

SYMBOLS USED

n = number of sub-intervals

h = length of interval

x_n = last value of x_i

x = values of x at which we have to find the value of y

$a[y]$ = an array to store different values of y

y = value of y at x

ALGORITHM : NEWTON'S BACKWARD INTERPOLATION FORMULA

Step 1 :

Start

Step 2 :

Input n, x_n, h, x

Step 3 :

For $i = 0$ to n

Step 4 :

Input $ay[i]$

Step 5 :

$ay = ay[n], t = 1$

Step 6 :

$u = (x - x_n)/h$

Step 7 :

For $i = 1$ to n

Step 8 :

For $j = 0$ to $(n-i)$

Step 9 :

$ay[j] = ay[j+1] - ay[j]$

Step 10 :

End of j loop

Step 11 :

$t = t * (u + i - 1) / i$

Step 12 :

$y = y + t * ay[n - i]$

Step 13 :

End of j loop

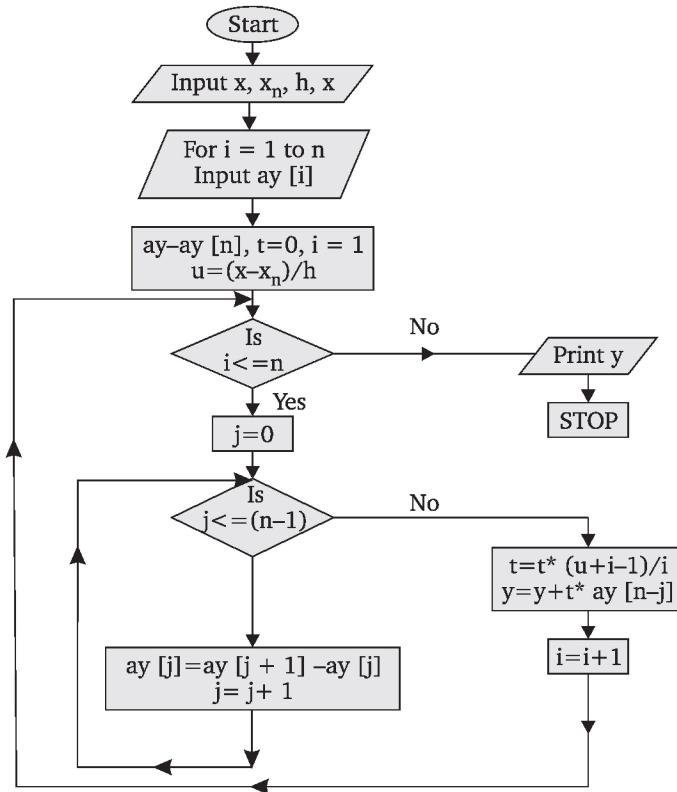
Step 14 :

Print y

Step 15 :

STOP.

FLOW CHART : NEWTON's BACKWARD INTERPOLATION FORMULA



PROGRAM : Following is a program to demonstrate the Newton's Backward Interpolation Formula.

NEWTON'S BACKWARD INTERPOLATION FORMULA

```

#include<stdio.h>
#include<conio .h>
void main ( )
{
    clrscr ( );
    float ay [30], xn, h, x, y, t=1, u;
    int n, i, j;
    printf ("Enter the value of n\n");
    scanf ("%d", &n);
    printf ("Enter the last value of x\n");
    scanf ("%f ", &xn);
    printf ("\nEnter length of each interval\n");
    scanf ("%f ", &h);
    for (i=0; i<=n; i++)
  
```

```

    {
        print f ("Enter the value of y(%d) = ", i) ;
        scanf ("%f", &ay [i]) ;
    }
    printf ("\n Enter the value of x for which value of y is
wanted\n");
    scanf ("%f", &x) ;
    y=ay [n] ;
    u=(x-xn)/h;
    for (i=1; i = n;i++)
    {
        for (j=0; j<=n-i; j++)
            ay [j]=ay[j+1]-ay [j];
        t =t* (u+i-1)/i;
        y=y+t*ay [n-i];
    }
    printf ("\n value of y at x = % .2f is% .2f", x, y) ; getch
( ) ;
}

```

Output : NEWTON'S BACKWARD INTERPOLATION FORMULA

Enter the value of n

4

Enter the last value of x

60

Enter length of each interval

10

Enter the value of y (0) = 42

Enter the value of y (1) = 87

Enter the value of y (2) = 126

Enter the value of y (3) = 174

Enter the value of y (4) = 193

Enter the value of x for which value of y is wanted

44

value of y at x=44.00 is 145.06

LAB ASSIGNMENT : NEWTON'S BACKWARD INTERPOLATION FORMULA

1. Write a C program for Newton's Backward Interpolation formula to find the value of f (7.5) for the data given below :

x	1	2	3	4	5	6	7	8
f(x)	1	8	27	64	125	216	343	512

Hint: Input $x_n = 5, h = 1, n = 7. x = 7.5$

Output $f(7.5) = 421.875$

2. Write a C program to find $f(4.4)$ by the Newton backward interpolation formula from the given data.

x	0	1	2	3	4	5
y	5	20	81	224	485	900

Hint: Input : $x_n = 5, h = 1, n = 5, x = 4.4$

Output: $y(4.4) = 630.5041$

3. GAUSS FORWARD INTERPOLATION FORMULA

SYMBOLS USED

n = number of subintervals

h = length of interval

x = value of x at which we have to find the value of y

k = location of x_0 . x_0 is that value which is closed to x

y = value of y at x

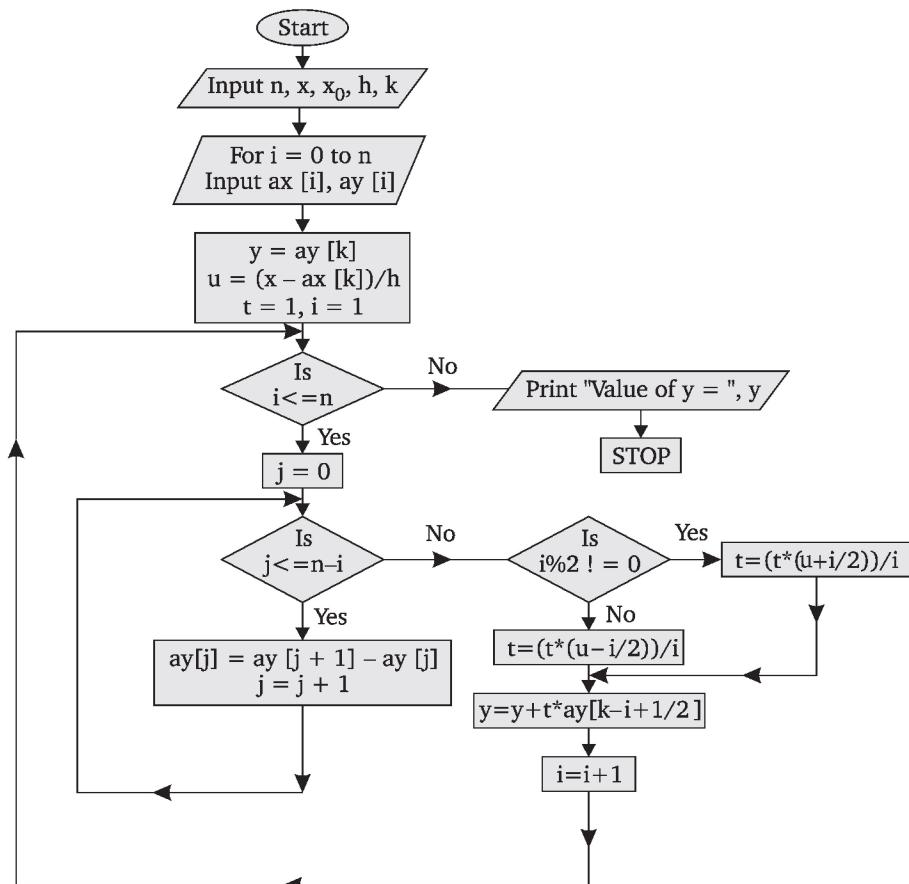
$a[x]$ = an array to store different values of x

$a[y]$ = an array to store the different values of y

ALGORITHM : GAUSS FORWARD INTERPOLATION FORMULA

- Step 1 :** Start
- Step 2 :** Input n, h, x, k
- Step 3 :** For $i = 0$ to n
- Step 4 :** Input $ay[i], ax[i]$
- Step 5 :** $y = ay[k], t = 1$
- Step 6 :** $u = (x - ax[k])/h$
- Step 7 :** For $i = 1$ to n
- Step 8 :** For $j = 0$ to $n - i$
- Step 9 :** $ay[j] = ay[j + 1] - ay[j]$
- Step 10 :** End of j loop
- Step 11 :** If ($i \% 2 == 0$)
 - $t = (t * (u + i/2))/i$
 - else
 - $t = (t * (u - i/2))/i ;$
- Step 12 :** $y = y + t * ay[k - i/2]$
- Step 13 :** End of i loop
- Step 14 :** Print "Value of y at x is" . y
- Step 15 :** STOP

FLOW CHART : GAUSS FORWARD INTERPOLATION METHOD



PROGRAM : Following is a C program to show the working of Gauss Forward Interpolation formula.

```

//GAUSS FORWARD INTERPOLATION FORMULA
#include<studio .h>
#include<conio ,h>
void main ( )
{
    clrscr ( ) ;
    float ax [30], ay [30] , h, x, y, t=1, u;
    int n, i, j, k;
    printf ("Enter the value n\n");
    scanf ("%d", &n) ;
    printf ("\n enter length of each interval\n");
    scanf ("%f", &h) ;
    printf ("Enter the value of x and y/n") ;
    
```

```

for (i=0; i<=n; i++)
    scanf ("%f %f ", & ax [i], & ay [i] );
printf ("\n Enter the value of x for which value of y
is wanted\n");
scanf ("%f", &x);
printf ("\n enter the location of x0 i.e. k/n");
scanf ("%d", &k);
y=ay [k];
u=(x-ax [k]) /h;
for (i=1; i<=n; i++)
{
    for (j=0; j<=n-i; j++)
        ay [j]=ay[j+1]-ay [j];
    if (i%2 != 0)
        t=(t* (u+i/2))/i;
    else
        t=(t* (u-i/2))/i;
    y=y+t*ay[k-i/2];
}
printf ("\n value of y at x=% .2f is % .2f ", x,
y) ; getch ( ) ;
}

```

Output : GAUSS FORWARD INTERPOLATION FORMULA

```

Enter the value of n
5
      enter length of each interval
.5
Enter the value of x and y
2.5  24.145
3     22.043
3.5   20.225
4     18.644
4.5   17.262
5     16.047
Enter the value of x for which value of y is wanted
3.75
      enter the location of x0 i.e. k
2
value of y at x=3.75 is 19.41

```

LAB ASSIGNMENT : GAUSS FORWARD INTERPOLATION FORMULA

1. Write a C program to find value of y (30) by Gauss Forward Interpolation formula from the data given below :

x	21	25	29	33	37
f(x)	18.4708	17.8144	17.1070	16.3432	15.5154

Hint : Input : n = 4, h = 4, x = 30, k = 2

Output : Value of y = 16.92

2. Write a C program for Gauss Forward interpolation formula to find value of f (2.3) from the given data

x	1	2	3	4	5
f(x)	1	-1	1	-1	1

Hint: Input : n = 4, h = 1, x = 2.3, k = 1

Output: Value of y = - 0.146600.

4. GAUSS BACKWARD INTERPOLATION FORMULA

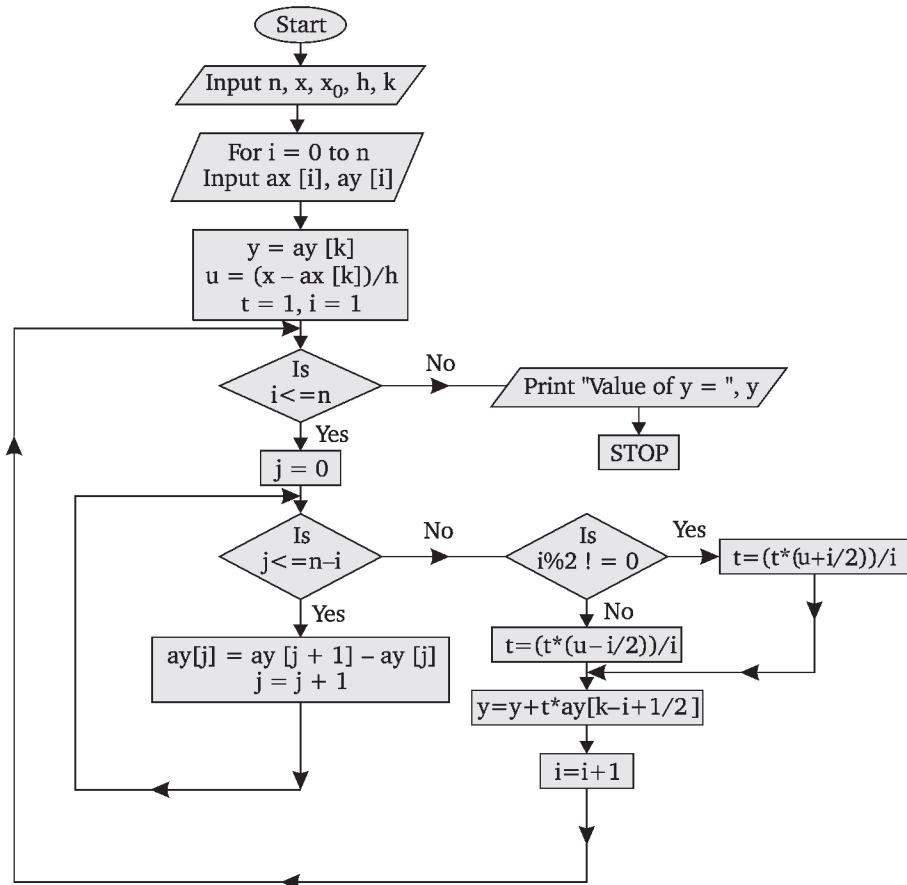
SYMBOLS USED

(Note : same as used in Gauss forward Interpolation formula).

ALGORITHM: GAUSS BACKWARD INTERPOLATION FORMULA

- Step 1 :** Start
- Step 2 :** Input x, h, x_0 , k
- Step 3 :** For i = 0 to n
- Step 4 :** Input $ax[i]$, $ay[i]$
- Step 5 :** $y = ay[k], t = 1$
- Step 6 :** $u = (x - ax[k])/h$
- Step 7 :** For i = 1 to n
- Step 8 :** For j = 0 to n - i
- Step 9 :** $ay[j] = ay[j + 1] - ay[j]$
- Step 10 :** End of j loop
- Step 11 :** if ($1 \% 2 != 0$)
 - $t = (t * (u - i/2))/i$
 - else
 - $t = (t * (u + i/2))/i$
- Step 12 :** $y = y + t * ay[k - (i + 1)/2]$
- Step 13 :** End of i loop.
- Step 14 :** Print “value of y=”, y
- Step 15 :** Stop.

FLOW CHART : GAUSS BACKWARD INTERPOLATION METHOD



PROGRAM : Following is a C program to show the working of Gauss Backward Interpolation Method.

GAUSS BACKWARD INTERPOLATION FORMULA

```

#include<studio .h>
#include<conio .h>
void main ( )
{
    clrscr ( );
    float  ax [30], ay [30] , h, x, y, t = 1, u;
    int n, i, j, k;
    printf ("Enter the value of n\n");
    scanf ("%d", &n);
    printf ("\n enter length of each interval\n");
    scanf ("%f", &h);
    printf ("Enter the value of x and y\n");
    for (i=0; i<=n;i++)
    {
        scanf ("%f %f", &ax [i], &ay [i]);
    }
  
```

```

        printf ("\n Enter the value of x for which value of y
is wanted\n") ;
        scanf ("%f", &x) ;
        printf ("\n enter the location of x0 i.e., k\n") ;
        scanf ("%d", &k) ;
        y=ay [k] ;
        u=(x-ax [k])/h ;
        for (i=1;i<= n; i++)
        {
            for (j=0; j<=n-i; j++)
                ay [j] = ay [j+1] - ay [j];
            if (i % 2 != 0)
                t=(t* (u-i/2))/i;
            else
                t=(t* (u+i/2))/i;
            y=y+t*ay [k- (i+1)/2];
        }
        printf ("\n Value of y at x=% .2f is % .2f", x, y) ;
        getch ( ) ;
    }
}

```

Output : GAUSS BACKWARD INTERPOLATION FORMULA

```

Enter the value of n
3
        enter length of each interval
1
Enter the value of x and y
4      270
5      648
6      1330
7      2448
        Enter the value of x for which value of y is wanted
5.8
        enter the location of x0 i.e. k
1
Value of y at x=5.80 is 1169.28

```

LAB ASSIGNMENT : GAUSS BACKWARD INTERPOLATION METHOD

1. Write a C program for Gauss Backward interpolation method to find y (1.15) form data given below :

x	1	1.10	1.20	1.30
y	1.0	1.04881	1.09544	1.14017

Hint : Input : n = 3, h = 0.10, k = 1, x = 1.15

Output : Value of y = 1.072397

2. Write a C program for Gauss Backward Interpolation Method to find the value for 1936 from the given data.

x	1901	1911	1921	1931	1941	1951
f(x)	12	15	20	27	39	52

Hint : Input : n = 5, h = 10, x = 1936, k = 3

output : Value of y = 32.3437.

5. STIRLING'S DIFFERENCE FORMULA

SYMBOLS USED

n = number of subintervals

h = length of interval

x = value of x at which we have to find the value of y

k = location of x_0 . x_0 is that value which is closed to x

y = value of y at x

a [x] = an array to store the different values of x

a [y] = an array to store the different values of y

ALGORITHM : STERLING'S DIFFERENCE FORMULA

Step 1 : Start

Step 2 : Input n, h

Step 3 : For i = 1 to n

Step 4 : Input $a_x[i]$, $a_y[i]$

Step 5 : Input x, k

Step 6 : $u = (x - x_k)/h$

Step 7 : $y = a_y[k]$, m = n

Step 8 : if ($k <= n/2$)

 n = 2k

 else

 n = 2 (n - k)

Step 9 : For i = 1 to n

Step 10 : For j = 0 to (m - i)

Step 11 : $a_y[j] = a_y[j + 1] - a_y[j]$

Step 12 : End of j loop

Step 13 : if ($i \% 2 \neq 0$)

$t_1 = (t_1 * (u - i/2))/i$

$t_2 = (t_2 * (u + i/2))/i$

 else

$t_1 = (t_1 * (u + i/2))/i$

$t_2 = (t_2 * (u - i/2))/i$

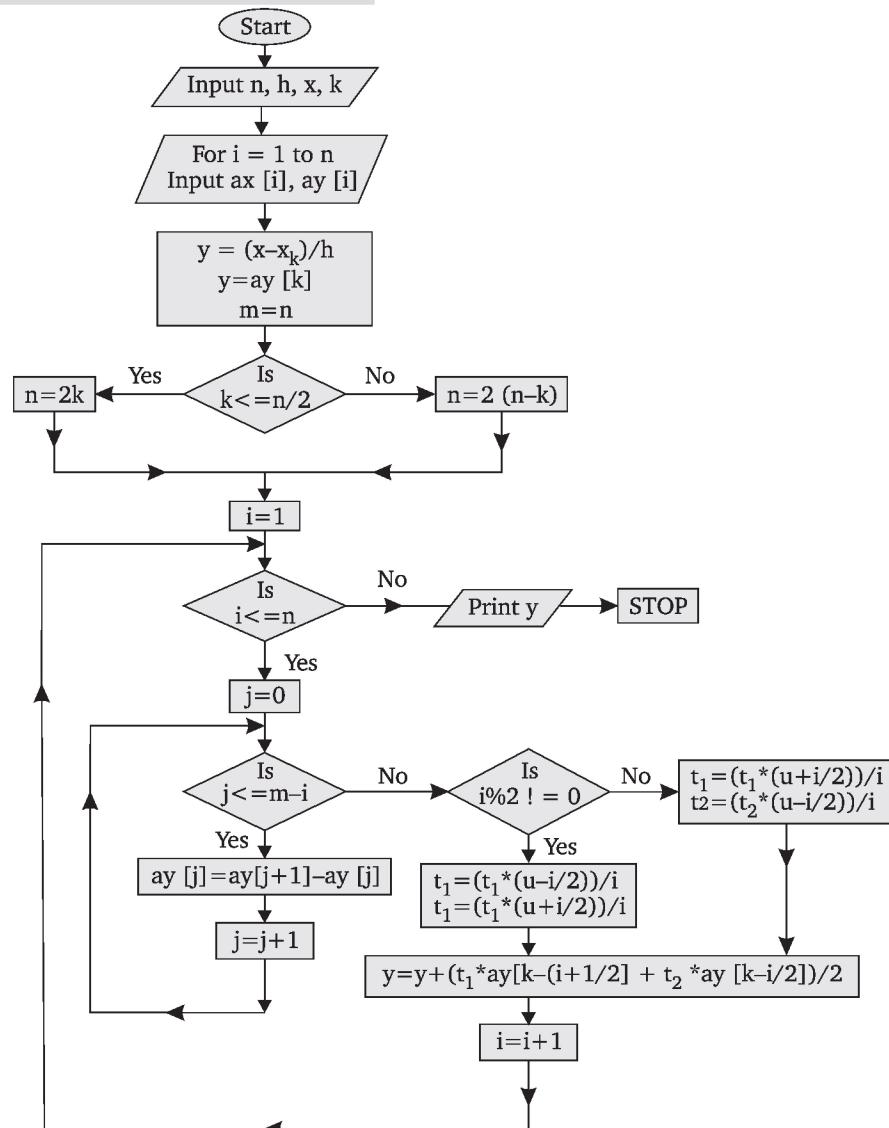
Step 14 : $y = y + (t_1 * a_y[k - (i + 1)/2] + t_2 * a_y[k - i/2])/2$

Step 15 : End of i loop

Step 16 : Print y

Step 17: Stop

FLOW CHART : STERLING DIFFERENCE FORMULA



PROGRAM : Following is a C program showing the working of Stirling Difference Formula for interpolation.

//STIRLING'S INTERPOLATION FORMULA

```

#include<stdio.h>
#include<conio.h>
void main ( )
{
    clrscr ( );
    float ax [30], ay [30], h, x, y, t1=1, t2=1, u;
    int n, i, j, m, k;
    
```

```

printf ("Enter the value of n\n");
Scanf ("%f", & h);
printf ("\n enter length of each interval\n");
scanf ("%f", &n);
printf ("Enter the value of x and y\n ");
for (i=0, i< = n++)
{
    scanf ("%f %f ", &ax [i], &ay [i]) ;
}
printf ("Enter the value of x for which value of y is
wanted\n");
scanf ("%f", &x);
printf ("\n enter the location of x0 i.e. k\n");
scanf ("%d", & k);
printf ("\n enter the location of x0 i.e. k\n");
scanf ("%d", &k);
y=ay [k];
u=(x-ax [k])/h;
m=n;
if (k< = n/2)
n=2*k;
else
n = 2* (n-k);
for (i=1; i<=n; i++)
{
    for (j=0; j<=m-i; j++)
ay[j]=ay [j+1]-ay [j];
    if (i%2 != 0)
    {
        t1=(t1*(u-i/2)) /i;
        t2=(t2*(u+i/2)) /i;
    }
    else
    {
        t1=(t1*(u+i/2))/i;
        t2=(t2*(u-i/2))/i;
    }
    y=y+(t1*ay[k-(i+1) / 2] + t2*ay[k-i/2]) /2 ;
}
print f ("\n Value of y at x=% .2f is % .2f ", x, y);
getch ();
}

```

Output : STIRLING'S INTERPOLATION FORMULA

```

Enter the value of n
4
enter length of each interval
5
Enter the value of x and y
10 492
15 483
20 472
25 459

```

```

30      453
Enter the value of x for which value of y is wanted
19
    enter the location of x0 i.e. k
2
Value of y at x=19.00 is 474.49

```

LAB ASSIGNMENT : STIRLING DIFFERENCE FORMULA

1. Write a C program to find the value of y [28] from the given data using Stirling Difference Formula.

x	20	25	30	35	40
y	49225	48316	47236	45926	44306

Hint : Input : $x = 28, n = 4, h = 5, x_0 = 30, k = 2$

Output : $y(28) = 47692$.

2. Using Stirling difference formula, write a C program to find $y(25)$ by given data :

x	20	24	28	32
y	24	32	34	40

Hint : Input : $h = 4, n = 3, x = 25, k = 1$

Output : $y(25) = 32.945287$

6. LAGRANGE'S INTERPOLATION FORMULA FOR UNEQUAL INTERVAL

SYMBOLS USED

n = number of subintervals

x = value of x at which we have to find the value of y

nr = numerator of each term of Lagrange's formula

dr = denominator of each term of Lagrange's formula

y = value of y at x

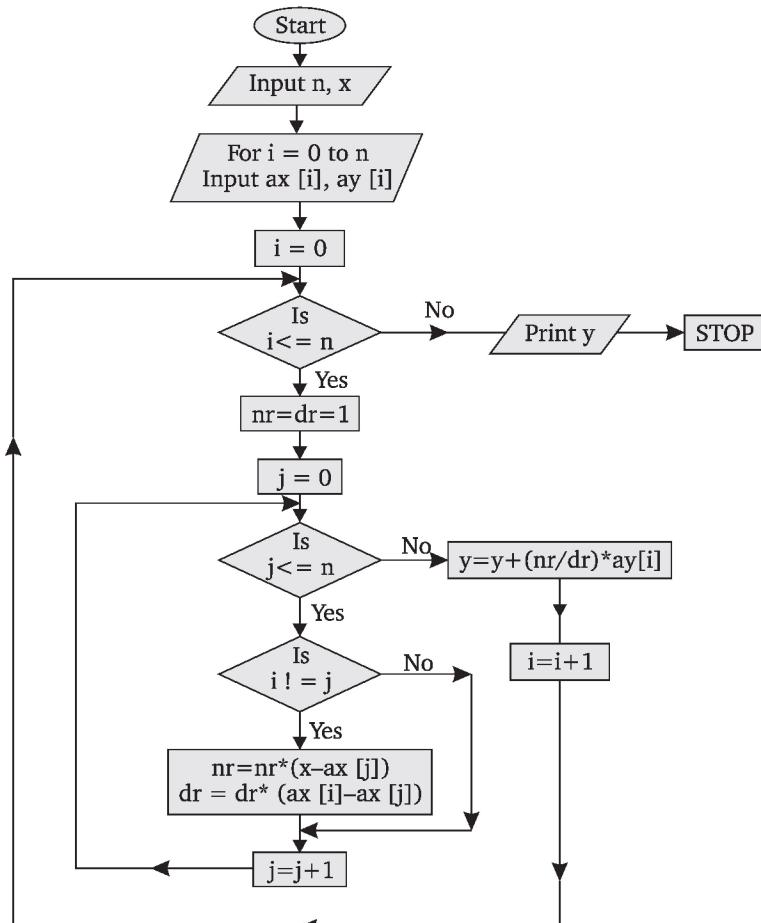
$a[x]$ = an array to store different values of x

$a[y]$ = an array to store the difference values of y'

ALGORITHM : LAGRANGE'S INTERPOLATION FORMULA

- Step 1 :** Start
- Step 2 :** Input n, x
- Step 3 :** For $i = 0$ to n
- Step 4 :** Input $ax[i], ay[i]$
- Step 5 :** For $i = 0$ to n
- Step 6 :** $nr=dr=1$
- Step 7 :** For $j = 0$ to n
- Step 8 :** if ($i \neq j$)
 - $nr = nr * (x - ax[j])$
 - $dr = dr * (ax[i] - ax[j])$
- Step 9 :** End of j loop.
- Step 10 :** $y = y + (nr/dr) * ay[i]$
- Step 11 :** End of i loop
- Step 12 :** Print y
- Step 13 :** Stop.

FLOW CHART : LAGRANGE'S INTERPOLATION FORMULA



PROGRAM :

Following is a program for the Lagrange's interpolation formula.

//Lagrange's Interpolation Method

```

#include<<conio.h>>
#include<stdio.h>
#define MAX 100

void main ( )
{
    float ax [MAX+1], ay [MAX+1], nr, dr, x, y=0;
    int i, j, n ;
    clrscr ( ) ;
    printf ("\n\nEnter the value of n\n\n") ;
    scanf ("%d", &n) ;
    printf ("Enter the set of values of x and y\n") ;
    for (i=0; i<= n =; i++)
        scanf ("%f%f", &ax [i], &ay [i]);
    puts ("\nEnter the value of x for which value of y is required");
  
```

```

scanf ("%f", &x);
for (i=0; i<=n; i++)
{
    nr=dr=1;
    for (j=0; j<= n; j++)
        if (i != j)
        {
            nr*=x-ax [j];
            dr*=ax [i] -ax[j] ;
        }
    y=(nr/dr) *ay [i] ;
}
printf ("\n when x=%f,      y=%f", x, y) ;
getch () ;
}

OUTPUT : LAGRANGE'S INTERPOLATION METHOD
Enter the value of n
4
Enter the set of values of x and y
1      8
2      15
4      19
8      32
10     40
Enter the value of x for which value of y is required
5
when x = 5, y = 22.7460

```

LAB ASSIGNMENT : LAGRANGE'S INTERPOLATION FORMULA

1. Write a C program using Lagrange's formula to find f (4) by the given data

x	0	1	2	5
f(x)	2	5	7	8

Hint : Input : n=3, x=4

Output : f (4)=8.4

2. Write a C program to find y (5) using Langrange's interpolation formula for the data given below :

x	1	3	4	8	10
f(x)	8	15	19	32	40

Hint : Input : n = 4, x=5

Output : y (5)= 11.318

7. NEWTON'S DIVIDED DIFFERENCE FORMULA

SYMBOLS USED :

n = number of subintervals

x = value of x at which we have to find the value of y

y = value of y at x

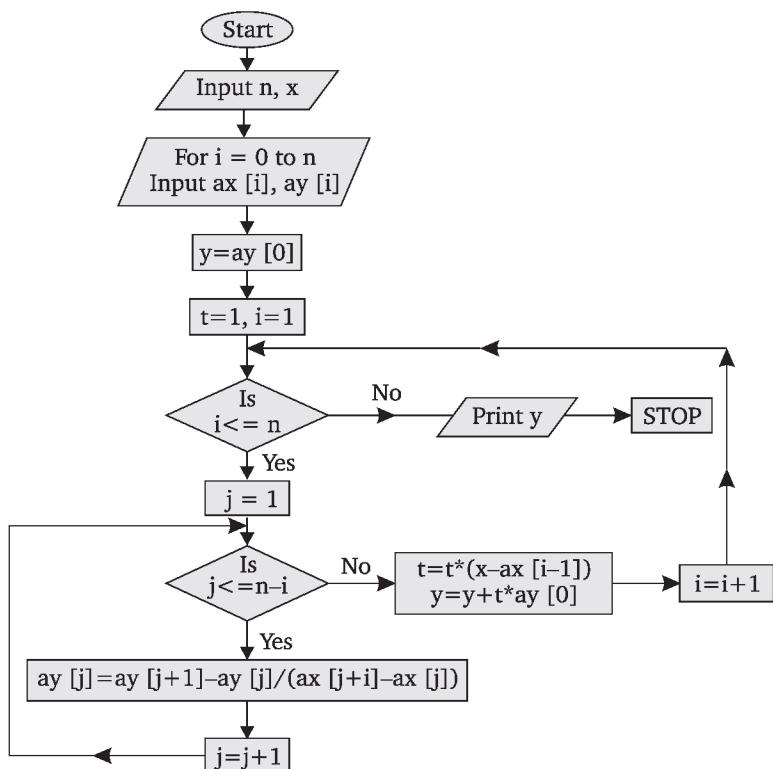
a [x] = an array to store the different values of x

a [y] = an array to store the different values of y.

ALGORITHM : NEWTON'S DIVIDED DIFFERENCE FORMULA

Step 1 : Start
Step 2 : Input n, x
Step 3 : For i = 0 to n
Step 4 : Input ax [i], ay [i]
Step 5 : y = ay [0], t = 1
Step 6 : For i = 1 to n
Step 7 : For j = 0 to (n-i)
Step 8 : ay [j] = (ay [j + 1] - ay [j]) / (ax [j + 1] - ax [j])
Step 9 : End of j loop
Step 10 : t = t * (x - ax [i - 1])
Step 11 : y = y + t * ay [0]
Step 12 : End of i loop
Step 13 : Print "value of y=", y
Step 14 : Stop.

FLOW CHART : NEWTON'S DIVIDED DIFFERENCE FORMULA



PROGRAM : Flowing program shows the working of Newton's Divided Difference formula for interpolation.

//NEWTON'S DIVIDED DIFFERENCE INTERPOLATION FORMULA

```

#include<studio.h>
#include<conio.h>
  
```

```

void main ( )
{
    clrscr ( ) ;
    float ax [30], ay [30], x, y, t=1;
    printf ("Enter the value of n\n") ;
    scanf ("%d", &n) ;
    printf ("\n enter value of x \n") ;
    scanf (" %f ", &x) ;
    printf ("Enter the value of x and y\n ") ;
    for (i=0; i<=n;i++)
    {
        scanf ("%f %f", &ax [i], &ay [i]) ;
    }
    y=ay [0] ;
    for (i=1; i<=n; i++)
    {
        for (j=0;j<=n;j++)
            ay [j]=(ay [j+1] - ay [j])/ (ax [j+i] - ax [j]) ;
        t = t* (x-ax [i-1]) ;
        y=y+t*ay [0] ;
    }
    printf ("\n Value of y at x=% .2f is % .2f ", x, y) ;
    getch ( ) ;
}

```

Output : NEWTON'S DIVIDED DIFFERENCE INTERPOLATION FORMULA

```

Enter the value of n
3
enter value of x
1.6
Enter the value of x and y
1      3.49
1.4    4.82
1.8    5.96
2.2    6.5
Value of y at x=1.60 is 5.44

```

LAB ASSIGNMENT: NEWTON'S DIVIDED DIFFERENCE FORMULA

1. Write a computer program in C language to find value of $f(15)$ by using Newtons divided difference formula from the following table :

x	4	5	7	10	11	13
f(x)	48	100	294	900	1210	2028

Hint : Input : $n = 5, x = 15$

Output : Value of y at $x = 15$ is 3150.

2. Write a C program for the Newton's divided difference formula to find value of $f(10)$ from the following data :

x	5	6	9	11
f(x)	12	13	14	16

Hint : Input : $n = 3, x = 10$

Output : Value of y at $x = 10 = 14.667$.



COMPUTATIONAL TECHNIQUE LAB

1. TRAPEZOIDAL RULE

SYMBOLS USED

n = number of subdivision

x_0 = lower limit of integral

x_n = upper limit of integral

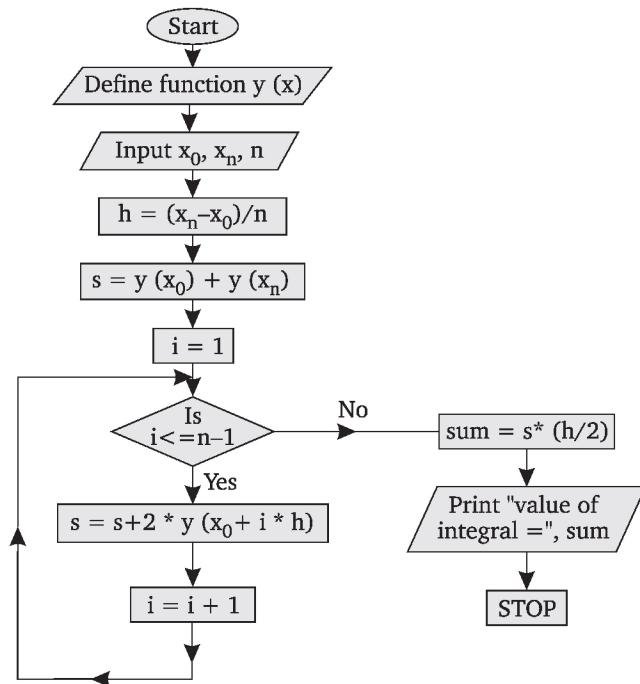
h = length of subinterval

$y(x)$ = function to be integrated

ALGORITHM : TRAPEZOIDAL RULE

- Step 1 :** Start
- Step 2 :** Input x_0 , x_n , n
- Step 3 :** $h = (x_n - x_0) / n$
- Step 4 :** $s = y(x_0) + y(x_n)$
- Step 5 :** For $i = 1$ to $n - 1$
- Step 6 :** $s = s + 2 * y(x_0 + i * h)$
- Step 7 :** End of i loop
- Step 8 :** Sum = $s * (h/2)$
- Step 9 :** Print "Value of integral = ", sum
- Step 10 :** Stop.

FLOW CHART : TRAPEZOIDAL RULE



PROGRAM : Following is a C program to find the value of the integral $\int_0^6 \frac{dx}{1+x^2}$ by trapezoidal rule.

```

Define a function y as 1/(1+x*x)
// Trapezoidal Rule
# include<stdio.h>
#include<conio.h>
float y (float x)
{
    return 1/(1+x*x);
}
void main()
{
    float x0,xn,s,h,sum;
    int i,n;
    clrscr();

    puts("\n Enter number of subdivisions i.e n ");
    scanf(" %d",&n);
    puts("\n Enter lower limit of integral i.e. x0 ");
    scanf(" %f",&x0);
    puts("\n Enter upper limit of integral i.e. xn ");
    
```

```

        scanf(" %f", &xn);

        h=(xn-x0)/n;
        s=y(x0)+y(xn);
        for(i=1;i<=n-1;i++)
            s+=2*y(x0+i*h);
        sum=s*(h/2);
        printf("\n Value of Integral is %.3lf\n",sum);
        getch();

    }

```

Output: TRAPEZOIDAL RULE

```

enter number of subdivisions i.e. n
6
enter lower limit of integral i.e. x0
0
enter upper limit of integral i.e. xn
6
Value of Integral is 1.411

```

LAB ASSIGNMENT : TRAPEZOIDAL RULE

1. Write a C program to solve the following integral by trapezoidal rule

$$\int_{0.2}^{1.4} (\sin x - \log_e x + e^x) dx$$

Hint : Define a function y

$$\sin(x) - \log_e(x) + \exp(x)$$

Input : $x_0 = 0.2$ $x_n = 1.4$ $n = 12$

Output : Value of integral = 4.056174

2. Write a C program to calculate the value of integral $\int_4^{5.2} \log x dx$ by trapezoidal rule.

Hint : Define function $y = \log(x)$

Input : $x_0 = 4$, $x_n = 5.2$ $n = 6$

Output : Value of integral = 1.827648

2. SIMPSON'S 1/3 RULE

SYMBOLS USED

x_0 = lower limit of integral

x_n = upper limit of integral

n = no. of subdivisions

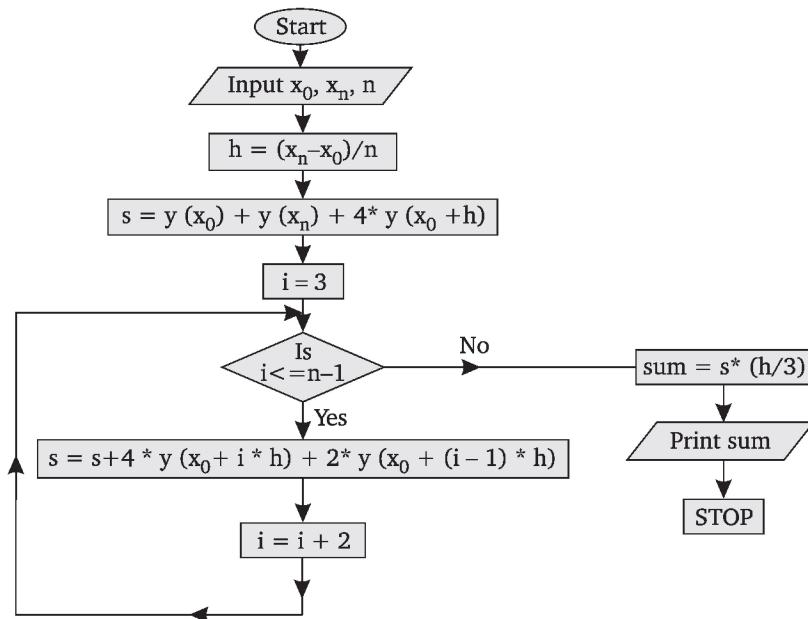
h = length of subinterval

$y(x)$ = function to be integrated

ALGORITHM : SIMPSON's 1/3 RULE

- Step 1 :** Start
- Step 2 :** Input x_0, x_n, n
- Step 3 :** $h = (x_n - x_0)/n$
- Step 4 :** $s = y(x_0) + y(x_n) + 4 * y(x_0 + h)$
- Step 5 :** For $i = 3$ to $(n - 1)$
- Step 6 :** $s = s + 4 * y(x_0 + i * h) + 2 * y(x_0 + (i - 1) * h)$
- Step 7 :** $i = i + 2$
- Step 8 :** End of i loop.
- Step 9 :** Sum = $s * (h/3)$
- Step 10 :** Print "Value of integral = ", sum
- Step 11 :** Stop.

FLOW CHART : SIMPSON's 1/3 RULE



PROGRAM : Following is a C program to the integral $\int_0^6 \frac{dx}{1+x^2}$ by Simpson's 1/3 rule.
 $//\text{SIMPSON 1/3rd RULE}$

```

#include<stdio.h>
#include<conio.h>
float y(float x)
{
    return 1/(1+x*x);
}
void main()

```

```

{
    float x0,xn,h,s,sum;
    int i,n;
    clrscr();
    puts("\n enter number of subdivisions i.e. n ");
    scanf("%d",&n);
    puts("\n enter lower limit of integral i.e. x0 ");
    scanf("%f",&x0);
    puts("\n enter upper limit of integral i.e. xn ");
    scanf("%f",&xn);

    h=(xn-x0)/n;
    s=y(x0)+y(xn)+4*y(x0+h);

    for(i=3;i<=n-1;i+=2)
        s+=4*y(x0+i*h)+2*y(x0+(i-1)*h);
    sum=s*(h/3);
    printf("\n Value of Integral is %.3lf\n",sum);
    getch();
}

```

Output: SIMPSON 1/3 rd RULE

```

enter number of subdivisions i.e. n
6
enter lower limit of integral i.e. x0
0
enter upper limit of integral i.e. xn
6
Value of Integral is 1.366

```

LAB ASSIGNMENT : SIMPSON's 1/3 RULE

1. Write a C program to solve the integral $\int_0^4 e^x dx$ using Simpson's 1/3 rule.

Hint : Define function $y = \exp(x)$

Input : $x_0 = 0$, $x_n = 4$, $n = 4$

Output : Value of integral = 53.87

2. Write a C program to solve the integral $\int_0^7 \frac{1}{x} dx$ using Simpson's 1/3 rule.

Hint : Define function $y = (1/x)$

Input : $x_0 = 1$, $x_n = 7$, $n = 6$

Output : Value of integral = 1.9587

3. Write a C program to solve the integral $\int_{0.5}^{0.7} x^{1/2} e^{-x} dx$ by Simpson's 1/3 rule.

Hint : Define function $y = \sqrt{x} * \exp(-x)$

Input : $x_0 = 0.5$, $x_n = 0.7$, $n = 4$

Output : Value of integral = 0.08483

3. SIMPSON'S 3/8 RULE

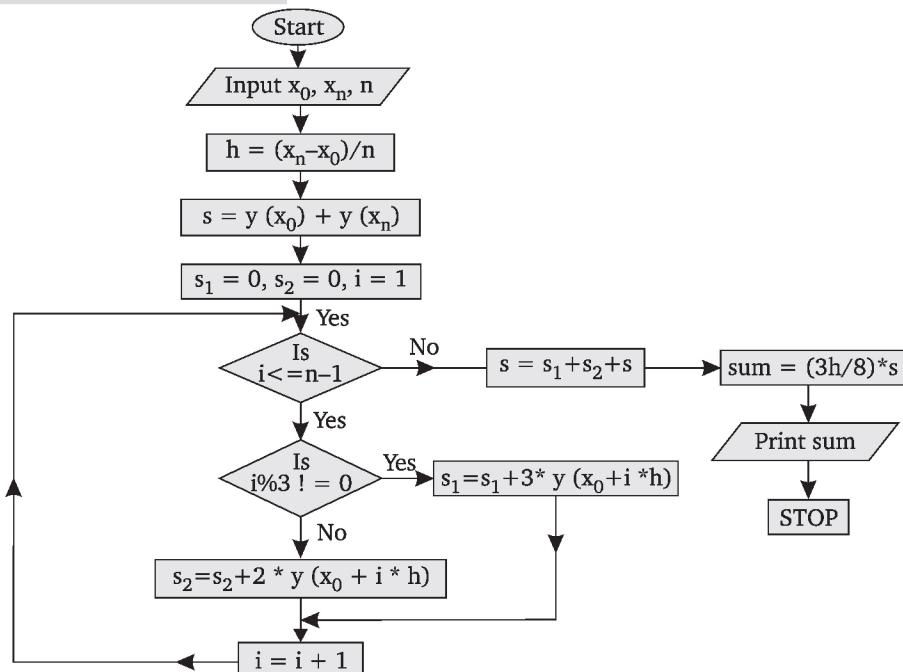
SYMBOLS USED

n = number of subdivisions
 x_0 = lower limit of integral
 x_n = upper limit of integral
 h = length of each subinterval
 $y(x)$ = function to be integrated

ALGORITHM : SIMPSON's 3/8 RULE

- Step 1 :** Start
- Step 2 :** Input x_0, x_n, n
- Step 3 :** $h = (x_n - x_0)/n, s_1 = 0, s_2 = 0$
- Step 4 :** $s = y(x_0) + y(x_n)$
- Step 5 :** For $i = 1$ to $(n - 1)$
- Step 6 :** If $(i \% 3 \neq 0)$
 - $s_1 = s_1 + 3 * y(x_0 + i * h)$
 - else
 - $s_2 = s_2 + 2 * y(x_0 + i * h)$
- Step 7 :** End of i loop
- Step 8 :** $s = s + s_1 + s_2$
- Step 9 :** Sum = $s * (3 * h)/8$
- Step 10 :** Print "Value of integral = ", sum
- Step 11 :** STOP.

FLOW CHART : SIMPSON's 3/8 RULE.



PROGRAM : Following is a C program to the integral $\int_0^1 \frac{dx}{1+x}$ by Simpson's 3/8 rule.
//SIMPSON 3/8th RULE

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

float y(float x)
{
    return 1/(1+x);
}

void main()
{
    float x0,xn,h,s,s1=0,s2=0,sum;
    int i,n;
    clrscr();
    puts("\n enter number of subdivisions i.e. n ");
    scanf("%d",&n);
    puts("\n enter lower limit of integral i.e. x0 ");
    scanf("%f",&x0);
    puts("\n enter upper limit of integral i.e. xn ");
    scanf("%f",&xn);
    h=(xn-x0)/n;
    s=y(x0)+y(xn);
    for(i=1;i<=n-1;i++)
        if(i%3!=0)
            s1+=3*y(x0+i*h);
        else
            s2+= 2*y(x0+i*h);
    s=s+s1+s2;
    sum=s*(3*h/8);
    printf("\n Value of Integral is %.3lf\n",sum);
    getch();
}

Output: SIMPSON 3/8th RULE
enter number of subdivisions i.e. n
6
enter lower limit of integral i.e. x0
0
enter upper limit of integral i.e. xn
1
Value of Integral is 0.693
```

LAB ASSIGNMENT : SIMPSON's 3/8TH RULE

1. Write a C program to solve the integral $\int_0^1 \frac{dx}{1+x^2}$ using Simpson 3/8 rule.

Hint : Define function $y = 1/(1 + x * x)$

Input : $x_0 = 0$, $x_1 = 1$, $n = 6$

Output : Value of integral = 0.785396

2. Write a C program to solve the integral $\int_0^6 (1+x^2)dx$ using Simpson 3/8th rule.

Hint : Define function $y(x) = (1 + x * x)$

Input : $x_0 = 0$, $x_n = 6$, $n = 6$

Output : Value of integral = 78

4. BOOLE'S RULE

SYMBOLS USED

n = number of subdivision

x_0 = lower limit of integral

x_n = upper limit of integral

h = length of subinterval

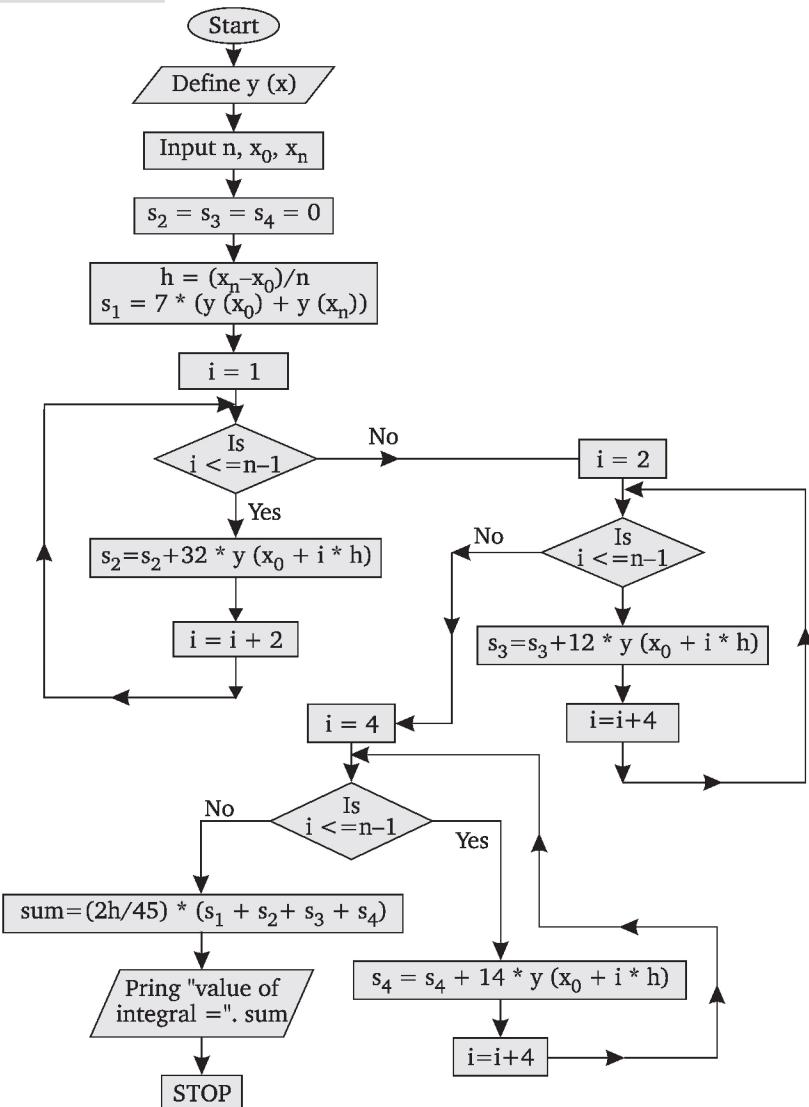
$y(x)$ = function to be integrated

ALGORITHM : BOOLE'S RULE

- Step 1 :** Start
- Step 2 :** Define function $y(x)$
- Step 3 :** Input n , x_0 , x_n
- Step 4 :** $s_2 = s_3 = s_4 = 0$
- Step 5 :** $h = (x_n - x_0)/n$
- Step 6 :** $s_1 = 7 * y(x_0) + y(x_n)$
- Step 7 :** For $i = 1$ to $(n - 1)$
- Step 8 :** $s_2 = s_2 + 32 * y(x_0 + i * h)$
- Step 9 :** $i = i + 2$
- Step 10 :** End of i loop
- Step 11 :** For $i = 2$ to $(n - 1)$
- Step 12 :** $s_3 = s_3 + 12 * y(x_0 + i * h)$
- Step 13 :** $i = i + 4$
- Step 14 :** End of i loop
- Step 15 :** For $o = 4$ to $(n - 1)$
- Step 16 :** $s_4 = s_4 + 14 * y(x_0 + i * h)$
- Step 17 :** $i = i + 4$

- Step 18 :** End of i loop
- Step 19 :** Sum = $(s_1 + s_2 + s_3 + s_4) * (2 * h) / 45$
- Step 20 :** Print "Value of integral = ", sum
- Step 21 :** STOP

FLOW CHART : BOOGLE'S RULE



PROGRAM : Following is a C program to solve the integral $\int_0^6 (1+x^2)dx$ by the Boole's rule.

```

// BOOLE'S RULE
#include<stdio.h>
#include<conio.h>

```

```
float y(float x)
{
    return (1+x*x);
}
void main()
{
    float x0,xn,h,s1,s2=0,s3=0,s4=0,s,sum;
    int i,n;
    clrscr();
    puts("\n enter number of subdivisions i.e. n ");
    scanf("%d",&n);
    puts("\n enter lower limit of integral i.e. x0 ");
    scanf("%f",&x0);
    puts("\n enter upper limit of integral i.e. xn ");
    scanf("%f",&xn);

    h=(xn-x0)/n;
    s1=7*y(x0)+y(xn);

    for(i=1;i<=n-1;i+=2)
        s2+=32*y(x0+i*h);
    for(i=1;i<=n-1;i+=4)
        s3+= 12*y(x0+i*h);

    for(i=4;i<=n-1;i+=4)
        s4+=14*y(x0+i*h);

    s=s1+s2+s3+s4;
    sum=s*(2*h/45);
    printf("\n Value of Integral is %.3lf\n",sum);
    getch();
}
```

Output: BOOLE's RULE

```
enter number of subdivisions i.e. n
8
enter lower limit of integral i.e. x0
0
enter upper limit of integral i.e. xn
6
Value of Integral is 67.450
```

LAB ASSIGNMENT : BOOLE'S RULE

1. Write a C program to solve the integral $\int_0^4 \frac{dx}{1+x^2}$ by Boole's rule. Divide the interval into 4 equal parts.

Hint : Define a function $y(x) = 1/(1 + x * x)$

Input : $x_0 = 0, x_n = 4, n = 4$

Output : Value of integral = 1.28941

2. Write a C program to solve the integral $\int_1^7 \frac{1}{x} dx$ by Boole's rule.

Hint : Define a function $y(x) = (1/x)$

Input : $x_0 = 1, x_n = 7, n = 8$

Output : Value of integral = 1.949018

5. WEDDLE'S RULE

SYMBOLS USED

n = number of subdivision

x_0 = lower limit of integral

x_n = upper limit of integral

h = length of subinterval

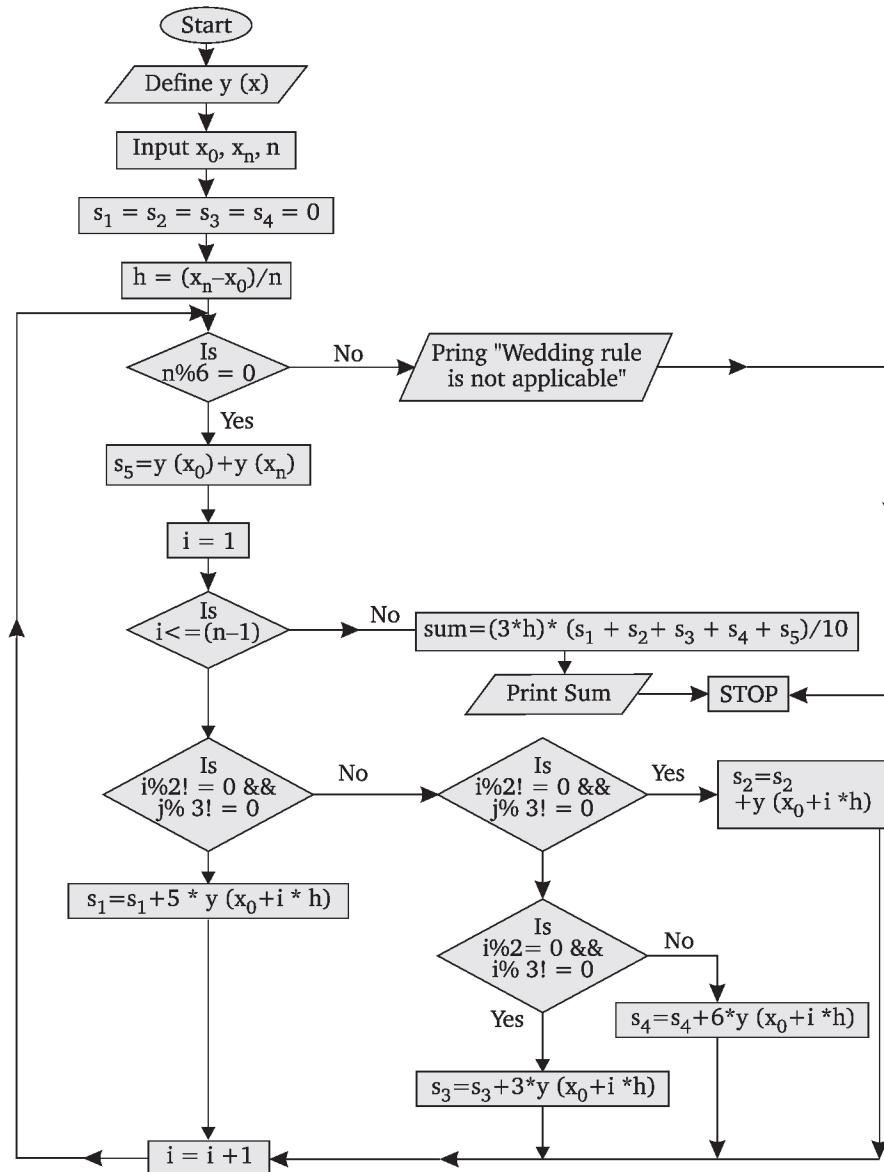
$y(x)$ = function to be integrated

ALGORITHM : WEDDLE'S RULE

```
Step 1 :      Start
Step 2 :      Define function y (x)
Step 3 :      Input x0, xn, n
Step 4 :      s1 = s2 = s3 = s4 = 0
Step 5 :      h = (xn - x0)/n
Step 6 :      If (n %6 = 0)
Step 7 :          s5 = y (x0) + y (xn)
Step 8 :      For i = 1 to n - 1
Step 9 :          If (i%2! = 0 and i%3! = 0)
                  s1 = s1 + 5 * y (x0 + i * h)
          else
              if (i%2 = 0 and i%3 != 0)
                  s2 = s2 + y (x0 + i * h)
              else
                  if (i%2 = 0 and i%3 = 0)
                      s3 = s3 + 3 * y (x0 + i * h)
                  else
                      s4 = s4 + 6 * y (x0 + i * h)
Step 10 :     End of i loop
Step 11 :     Sum = 3 * h * (s1 + s2 + s3 + s4 + s5)/10
Step 12 :     print "Value of integral = ", sum
```

- Step 13 :** else
Step 14 : Print "Weddle's rule is not applicable"
Step 15 : Stop.

FLOW CHART : WEDDLE'S RULE



PROGRAM : Following is a C program to solve the integral $\int_0^6 (1+x^2)dx$ by the Weddle's rule.

```
// WEDDLE's RULE
#include<stdio.h>
#include<conio.h>
float y(float x)
{
    return (1+x*x);
}
void main()
{
    float x0,xn,h,s1=0,s2=0,s3=0,s4=0,s5,s,sum;
    int i,n;
    clrscr();
    puts("\n enter number of subdivisions i.e. n ");
    scanf("%d",&n);
    puts("\n enter lower limit of integral i.e. x0 ");
    scanf("%f",&x0);
    puts("\n enter upper limit of integral i.e. xn ");
    scanf("%f",&xn);
    h=(xn-x0)/n;
    if(n%6==0)
        s5=y(x0)+y(xn);
    else
        {printf("\n Weddle's rule is not applicable\n");
         goto end;
        }
    for(i=1;i<=n-1;i++)
    {
        if(i%2!=0 && i%3!=0)
            s1+=5*y(x0+i*h);
        else
            if(i%2==0 && i%3==0)
                s2+=y(x0+i*h);
            else
                if(i%2==0 && i%3==0)
                    s3+= 3*y(x0+i*h);
                else
                    s4+=6*y(x0+i*h);
    }
    s=s1+s2+s3+s4+s5;
    sum=s*(3*h/10);
    printf("\n Value of Integral is %.3lf\n",sum);
```

```
end: getch();  
}  
Output: WEDDLE's RULE  
enter number of subdivisions i.e. n  
6  
  
enter lower limit of integral i.e. x0  
0  
  
enter upper limit of integral i.e. xn  
6  
  
Value of Integral is 78.000
```

LAB ASSIGNMENT : WEDDLE'S RULE

1. Write a C program to solve the integral $\int_0^1 \frac{dx}{1+x^2}$ by Weddle's rule. Divide the range into six equal parts.

Hint : Define function $y(x) = 1/(1 + x * x)$

Input : $x_0 = 0$, $x_n = 1$, $n = 6$

Output : Value of integral = 0.7854

2. Write a C program to solve the integral $\int_0^{1.5} \frac{x^3}{e^x - 1} dx$ by dividing the interval into six equal parts using Weddle's rule.

Hint : Define function $y(x) = (x * x * x) / \exp(x) - 1$

Input : $x_0 = 0$, $x_n = 1.5$, $n = 6$

Output : Value of integral = 0.6155

3. Write a C program to solve the integral $\int_0^5 \frac{dx}{4x+5}$ by Weddle's rule.

Hint : Define function $y(x) = 1/(4 * x + 5)$

Input : $x_0 = 0$, $x_n = 5$, $n = 12$

Output : Value of integral = 0.4023.



COMPUTATIONAL TECHNIQUE LAB

1. EULER'S METHOD

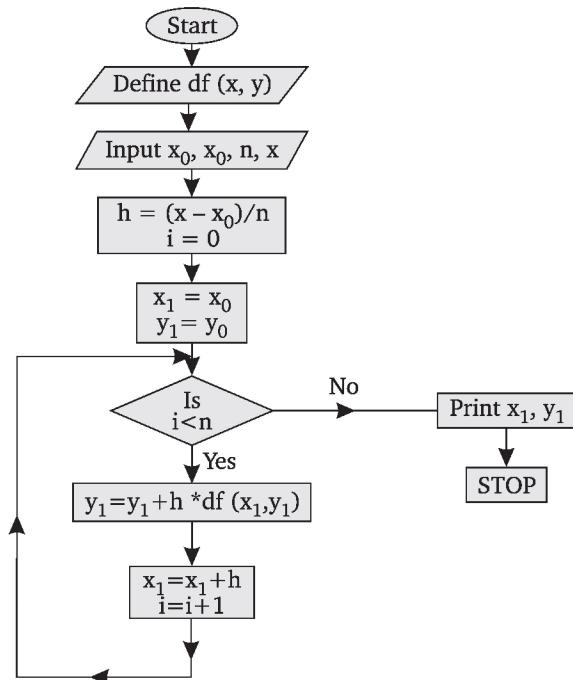
SYMBOLS USED

x_0, y_0 = Initial values of x and y
 h = length of subintervals of step
 x = the value of x at which we have to find y
 n = number of subintervals

ALGORITHM : EULER'S METHOD

- Step 1 :** Start
- Step 2 :** Define $df(x, y)$
- Step 3 :** Input n, x_0, y_0, x
- Step 4 :** $h = (x - x_0)/n$
- Step 5 :** $x_1 = x_0, y_1 = y_0, i = 0$
- Step 6 :** Perform steps 7 to 9 while ($i < n$)
- Step 7 :** $y_1 = y_1 + h * df(x_1, y_1)$
- Step 8 :** $x_1 = x_1 + h$
- Step 9 :** $i = i + 1$
- Step 10 :** Print x_1, y_1
- Step 11 :** Stop.

FLOW CHART : EULER'S METHOD



PROGRAM :

Following is a C program to find the value of y at x for the equation
 $\frac{dy}{dx} = df(x, y) = x + y$ by the Euler's method.

// EULER'S METHOD TO FIND VALUE OF Y AT X=1 FOR EQUATION dy/dx = X+Y

```
#include<stdio.h>
#include<conio.h>
float df(float x, float y)
{
    return x+y;
}

void main()
{
    clrscr();
    float x0,y0,h,x,x1,y1,n,i;

    puts("\n enter value of x0 ");
    scanf("%f", &x0);
    puts("\n enter the value of y0 ");
    scanf("%f", &y0);
    puts("\n enter the value of n ");
    scanf("%f", &n);
    puts("\n enter the value of x ");
    scanf("%f", &x);
    h=(x-x0)/n;
    x1=x0;
    y1=y0;

    for(i=0;i<n;i++)
    {
        y1+=h*df(x1,y1);
        x1+=h;
    }
    printf(" When x =%3.1f y = %4.2f\n",x1,y1);
    getch();
}

Output: EULER'S METHOD TO FIND THE VALUE OF Y(X)=1 FOR
EQUATION X+Y = (dy/dx)
```

enter value of x0
0

enter value of y0
1

```

enter value of n
10
enter value of x
1

```

When x = 1 y = 3.18748

LAB ASSIGNMENT : EULER'S METHOD

1. Write a C program to find y for x = 0.1 for the equation $\frac{dy}{dx} = \frac{y-x}{y+x}$ with initial condition y = 1 at x = 0, dividing the interval into 5 equal parts.

Hint : Define df = (y-x)/(y+x)

Input : n = 5, x₀ = 0, y₀ = 1, x = 0.10

Output : y (0.10) = 1.09283

2. Write a C program for the Euler's method to solve the differential equation $\frac{dy}{dx} = y^2 - x^2$ for y (0.5) with y = 1 when x = 0.

Hint : Define df = y * y - x * x

Input : x₀ = 0, y₀ = 1, x = 0.5, n = 5

Output : y (0.5) = 1.76393

2. MODIFIED EULER'S METHOD

SYMBOLS USED

x₀, y₀ = Initial values of x and y

x= value of x at which we have to find y

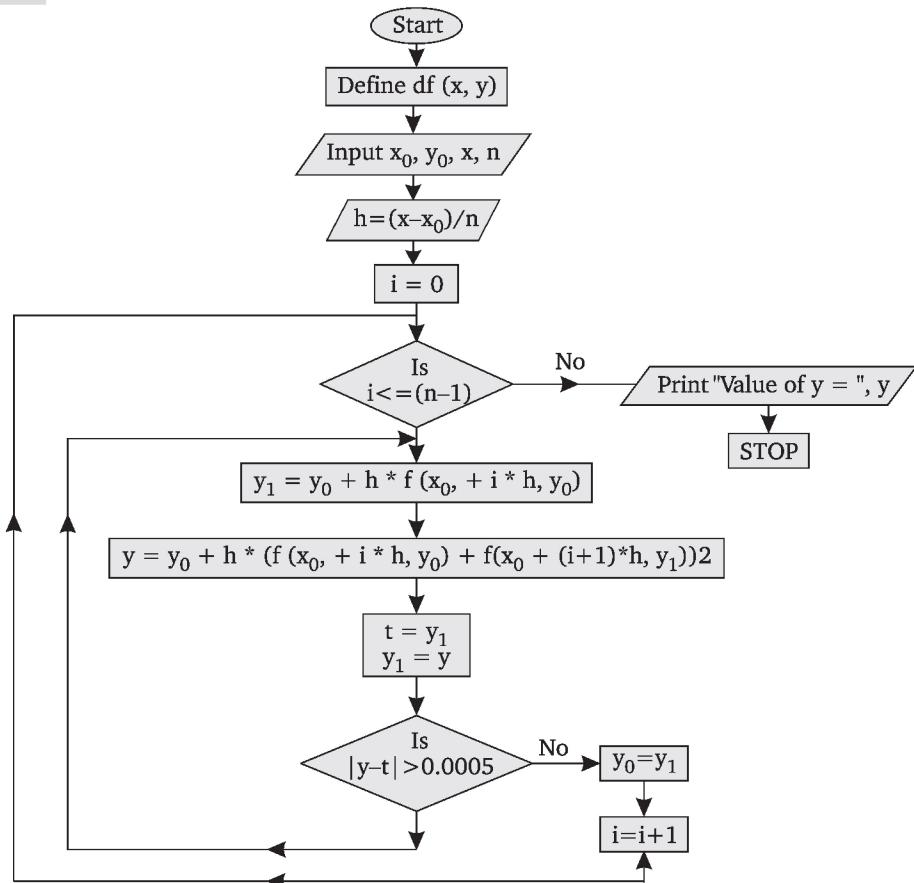
n = number of subintervals

h = step size

ALGORITHM: MODIFIED EULER'S METHOD

- | | |
|------------------|---|
| Step 1 : | Start |
| Step 2 : | define df (x, y) |
| Step 3 : | Input x ₀ , y ₀ , x, n |
| Step 4 : | h = (x - x ₀)/n |
| Step 5 : | for i = 0 to n - 1 |
| Step 6 : | y ₁ = y ₀ + h * f (x ₀ + i * h, y ₀) |
| Step 7 : | y = y ₀ + h * (f (x ₀ + i * h, y ₀) + (f (x ₀ + (i + 1) * h, y ₁))/2 |
| Step 8 : | t = y ₁ |
| Step 9 : | y ₁ = y |
| Step 10 : | Repeat step 7 to 9 until (y - t <=0.0005) |
| Step 11 : | y ₀ = y ₁ |
| Step 12 : | End of for loop |
| Step 13 : | Print y |
| Step 14 : | Stop. |

FLOW CHART



PROGRAM . Following is a C program to find the value of y at $x = 0.05$ for the equation $\frac{dy}{dx} = x + y$ by modified Euler's method with initial condition $y(0) = 1$.

```

// Modified Euler's method for df = x + y.
// EULER'S MODIFIED METHOD TO FIND VALUE OF Y AT X =1 FOR
// EQUATION dy/dx = X+Y
#include<stdio.h>
#include<conio.h>
#include<math.h>
float df(float x, float y)
{
    return  x+y;
}
void main()
{
    clrscr();
    float x0,y0,h,x,x1,y1,y,t;
    int n,i;
    
```

```

    puts("\n enter value of x0 ");
    scanf("%f", &x0);
    puts("\n enter the value of y0 ");
    scanf("%f", &y0);
    puts("\n enter the value of n ");
    scanf("%f", &n);
    puts("\n enter the value of x ");
    scanf("%f", &x);
    h=(x-x0)/n;
    for(i=0;i<n;i++)
    {
        y1=y0+h*df(x0+i*h,y0);

        do
        {   y=y0+h*(df(x0+i*h,y0)+df(x0+(i+1)*h,y1))/2;
            t=y1;
            y1=y;
        } while (fabs(y-t)>0.0005);
        y0=y1;
    }
    printf("\n When x =%3.1f y = %4.2f\n",x,y);
    getch();
}

```

**Output: EULER'S MODIFIED METHOD TO FIND VALUE OF Y(X)=1 FOR
EQUATION X+Y = (dy/dx)**

```

enter value of x0
0
enter value of y0
1
enter value of n
5
enter value of x
.05

```

When x = .05 y = 1.116388

LAB ASSIGNMENT : MODIFIED EULER's METHOD

1. Write a C program to find y (2.2) by Euler's modified method for $\frac{dy}{dx} = -xy^2$ where y (2) = 1.

Hint : Define df = - x * y * y

Input : $x_0 = 2, y_0 = 1, x = 2.2, n = 2$

Output : $y(2.2) = 0.7018$

2. Write a C program to find $y(0.2)$ for $\frac{dy}{dx} = \log_{10}(x + y)$ with initial condition $y = 1$ for $x = 0$ by modified Euler's method.

Hint : Define $df = \log_{10}(x + y)$

Input : $x_0 = 0, y_0 = 1, x = 0.2, n = 1$

Output : $y(0.2) = 1.0082$

3. RUNGE-KUTTA METHOD

SYMBOLS USED

x_0, y_0 = initial values of x and y

h = length of subinterval

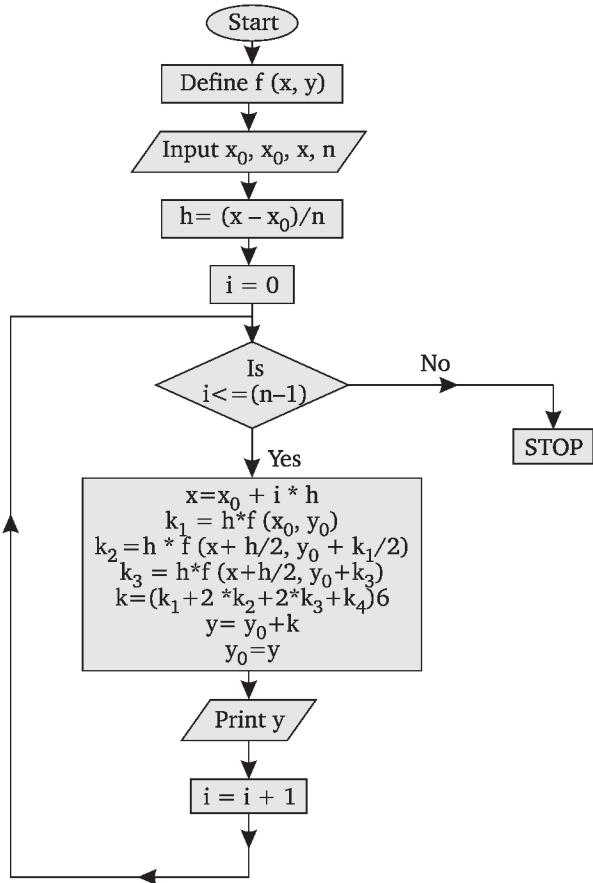
x = value of x at which we have to find y

n = number of subdivision

ALGORITHM : RUNGE-KUTTA 4TH ORDER

- | | |
|------------------|---|
| Step 1 : | Start |
| Step 2 : | Define $f(x, y)$ |
| Step 3 : | Input n, x_0, y_0, x |
| Step 4 : | $h = (x - x_0)/n$ |
| Step 5 : | For $i = 0$ to $(n - 1)$ |
| Step 6 : | $x = x_0 + i * h$ |
| Step 7 : | $k_1 = h * f(x, y_0)$ |
| Step 8 : | $k_2 = h * f(x + h/2, y_0 + k_1/2)$ |
| Step 9 : | $k_3 = h * f(x + h/2, y_0 + k_2/2)$ |
| Step 10 : | $k_4 = h * f(x + h, y_0 + k_3)$ |
| Step 11 : | $k = (k_1 + 2 * k_2 + 2 * k_3 + k_4)/6$ |
| Step 12 : | $y = y_0 + k$ |
| Step 13 : | $y_0 = y$ |
| Step 14 : | Print y |
| Step 15 : | End of For Loop. |
| Step 16 : | Stop. |

FLOW CHART : RUNGE-KUTTA 4TH ORDER



PROGRAM . Following is a C program to find the value of the equation $x+y$ for $x = 0.2$ for initial condition $y = 1$ at $x = 0$ by Runge Kutta 4th order.

```

/* RUNGE KUTTA 4th ORDER METHOD FOR dy/dx = X+y2 */
#include<stdio.h>
#include<conio.h>

float f(float x, float y)
{
    return x+y*y;
}

void main()
{
    clrscr();
    float x0,y0,h,x,y,k1,k2,k3,k4,k;
    int i,n;
    puts("\n enter the value of x0 ");
    scanf("%f", &x0);
    puts("\n enter the value of y0 ");
    scanf("%f", &y0);
}
  
```

```

    puts("\n enter the value of n ");
    scanf("%f", &n);
    puts("\n enter the value of x ");
    scanf("%f", &x);
    h=(x-x0)/n;
    for(i=0;i<=n-1;i++)
    {
        x=x0+i*h;
        k1=h*f(x,y0);
        k2=h*f(x+h/2,y0+k1/2);
        k3=h*f(x+h/2,y0+k2/2);
        k4=h*f(x+h,y0+k3);
        k=(k1+2*(k2+k3)+k4)/6;
        y=y0+k;
        y0 =y;
        printf("\n When x = %8.4f"
               " y = %8.4f\n",x,y);
        getch();
    }
}

```

Output: RUNGE KUTTA 4th ORDER METHOD FOR X+Y2

```

enter the value of x0
0
enter the value of y0
1
enter the value of n
2
enter the value of x
.2
When x = .2 y = 1.27356

```

LAB ASSIGNMENT : RUNGE-KUTTA 4TH ORDER

1. Write a C program to find $y(0.4)$ $\frac{dy}{dx} = -2xy^2$ with $y(0) = 1$ by Runge-Kutta 4th order.

Hint : Define $f(x, y) = -2 * x * y * y$

Input : $x_0 = 0, y_0 = 1, x = 0.4, n = 2$

Output : $y(0.4) = 0.86205$

2. Write a C program to find the solution of the differential equation $\frac{dy}{dx} = 3x + \frac{1}{2}y$ with $y_0 = 1$ at $x = 0.1$ by Runge-Kutta 4th Order.

Hint : Define $f(x, y) = 3 * x + y/2$

Input : $x_0 = 0, y_0 = 1, x = 0.1, n = 1$

Output : $y(0.1) = 1.06665$

3. Write a C program using Runge-Kutta 4th order method to solve $\frac{dy}{dx} = \frac{y^2 - x^2}{y^2 + x^2}$ with $y(0) = 1$ at $x = 0.2, 0.4$.

Hint : Define $f(x, y) = (y * y - x * x)/(y * y + x * x)$

- (a) **Input :** $x_0 = 0, y_0 = 1, x = 0.2, n = 1$

Output : $y(0.2) = 1.1960$

- (b) **Input :** $x_0 = 0, y_0 = 1, x = 0.4, n = 2$

Output : $y(0.4) = 1.37527$

4. MILNE'S PREDICTOR CORRECTOR METHOD

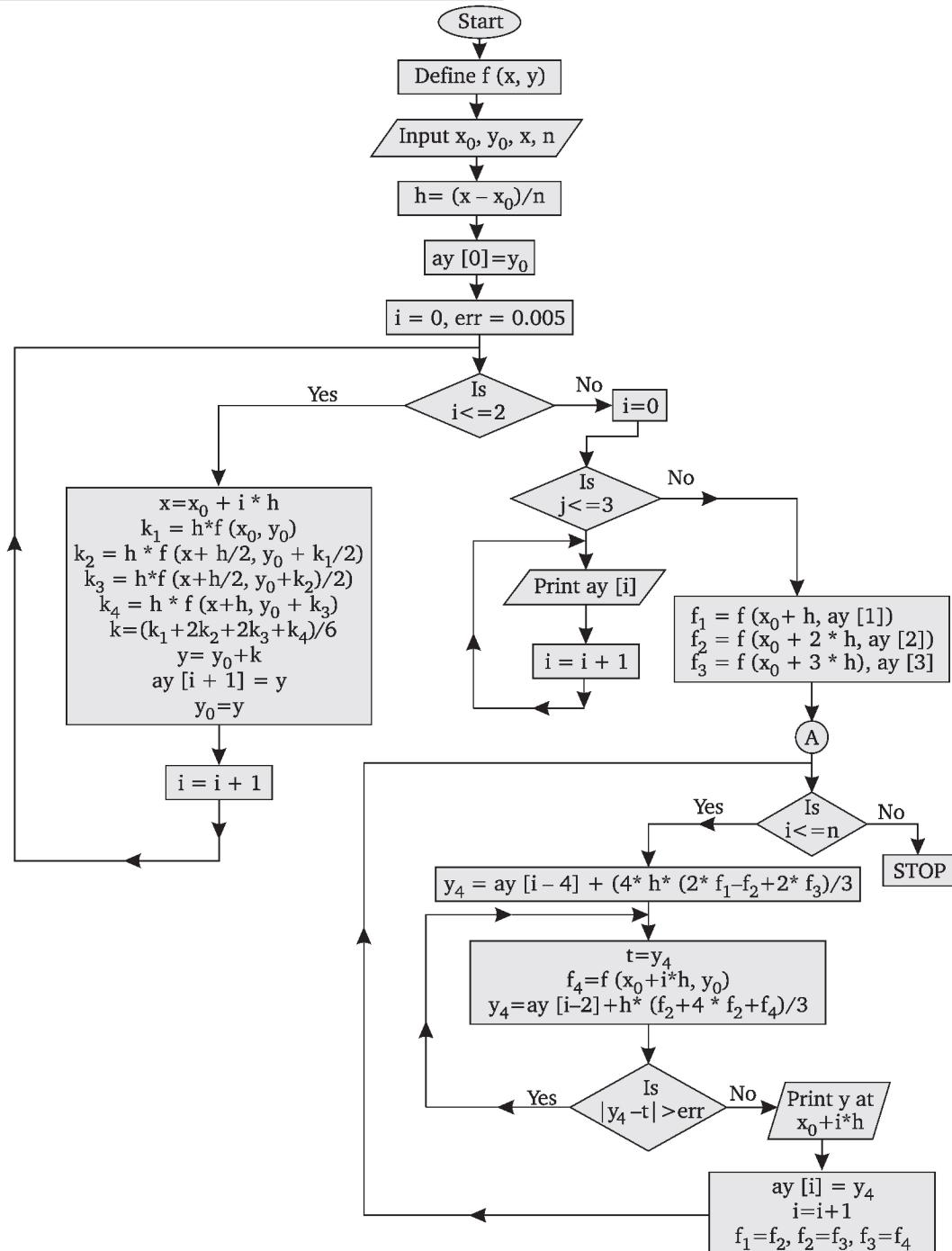
SYMBOLS USED

x_0, y_0 = initial values of x and y
 h = length of subinterval
 x = the value of x at which we have to find y
 n = number of subdivision
 err = allowed error.

ALGORITHM : MILNE'S PREDICTOR-CORRECTOR METHOD

Step 1 : Start
Step 2 : define $f(x, y)$
Step 3 : Input x_0, y_0, x, n
Step 4 : $h = (x - x_0)/n$
Step 5 : $ay[0] = y_0, err = 0.0005$
Step 6 : For i to 2
Step 7 : $x = x_0 + i * h$
Step 8 : $k_1 = h * f(x, y_0)$
Step 9 : $k_2 = h * f(x + h/2, y_0 + k_1/2)$
Step 10 : $k_3 = h * f(x + h/2, y_0 + k_2/2)$
Step 11 : $k_4 = h * f(x + h, y_0 + k_3)$
Step 12 : $k = (k_1 + 2 * k_2 + 2 * k_3 + k_4)/6$
Step 13 : $y = y_0 + k$
Step 14 : $ay[i + 1] = y$
Step 15 : $y_0 = y$
Step 16 : End of For loop
Step 17 : Print "Starting fair values for the Milne's method by Runga-Kutta method of order 4 are"
Step 18 : For i = 0 to 3
Step 19 : Print $ay[i]$
Step 20 : $f_1 = f(x_0 + h, ay[i])$
Step 21 : $f_2 = f(x_0 + 2 * h, ay[2])$
Step 22 : $f_3 = f(x_0 + 3 * h, ay[3])$
Step 23 : Repeat steps 24 to 31 until ($i > n$)
Step 24 : $y_4 = ay[i - 4] + (4 * h * (2 * f_1 - f_2 + 2 * f_3))/3$
Step 25 : Repeat steps 26 to 28 until ($|y_4 - t| < err$)
Step 26 : $t = y_4$
Step 27 : $f_4 = f(x_0 + i * h, y_4)$
Step 28 : $y_4 = ay[i - 2] + h * (f_2 + 4 * f_3 + f_4)/3$
Step 29 : print "Value of y at x =", $x_0 + i * h, ay[i]$
Step 30 : $i = i + 1$
Step 31 : $f_1 = f_2, f_2 = f_3, f_3 = f_4$
Step 32 : Stop.

FLOW CHART : MILNE'S PREDICTOR CORRECTOR METHOD



PROGRAM .

Following program shows the Milne's Predictor-Corrector Method to find the approximate value of y for $x = 0.4$ for the equation $dy/dx = xy + y^2$ with initial condition $y = 1$ at $x = 0$ dividing the range into four equal parts.

```
/* MILNE'S PREDICTOR CORRECTOR METHOD FOR dy/dx = X*Y+Y*Y */

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

float f(float x, float y)
{
    return x*y+y*y;
}

void main()
{
    clrscr();
    float ay[5], x0, y0, x, y, h, t, k1, k2, k3, k4, k, err=.0005;
    float f1, f2, f3, f4, y2, y4;
    int i, n;

    puts("Enter the value of x0, y0, x, n\n");
    scanf(" %f %f %f %d", &x0, &y0, &x, &n);
    h=(x-x0)/n;
    ay[0]=y0;

    for(i=0;i<n;i++)
    {
        x=x0+i*h;
        k1=h*f(x, y0);
        k2=h*f(x+h/2, y0+k1/2);
        k3=h*f(x+h/2, y0+k2/2);
        k4=h*f(x+h, y0+k3);
        k=(k1+2*(k2+k3)+k4)/6;
        y=y0+k;
        ay[i+1]=y;
        y0=y;
    }

    printf("\n Starting 4 values by runge -kutta mrthod are\n");
}
```

```
for(i=0;i<=3;i++)
printf("\n y= %d= %.5f",i,ay[i]);

f1=f(x0+h,ay[1]);
f2=f(x0+2*h,ay[2]);
f3=f(x0+3*h,ay[3]);


while(i<=1)
{
    y4=ay[i-4]+(4*h*(2*f1-f2+2*f3))/3;
    do
    {
        t=y4;
        f4=f(x0+i*h,y4);
        y4=ay[i-2]+h*(f2+4*f3+f4)/3;
        } while (fabs(y4-t)>err);
    printf("\n\n value of y at x=% .2f= %.5f",x0+i*h,y4);
    ay[i]=y4;
    i++;
    f1=f2; f2=f3, f3=f4;
}
getch();
}
```

Output: MILNE'S PREDICTOR CORRECTOR METHOD FOR X*Y+Y*X

Enter the value of x0,y0,x,n

0 1 .4 4

Starting 4 values by runge -kutta mrthod are

y= 0= 1.00000
y= 1= 1.11689
y= 2= 1.27739
y= 3= 1.50412

Value of y at x=0.40 is 1.83941

LAB ASSIGNMENT : MILNE'S PREDICTOR CORRECTOR METHOD

1. Write a C program for Milne's Predictor Corrector method to find value of y for $x = 0.5$ for $dy/dx = 2e^x - y$ with initial condition $x = 0, y = 2$ by dividing range into 5 equal parts.

Hint : Define $f(x, y) = 2 * \exp(x) - y$

Input : $x_0 = 0, y_0 = 2, x = 0.5, n = 5$

Output : $y(0.50) = 2.25525$

2. Write a C program to find value of y at $x = 0.5$ of the differential equation $dy/dx = x + y$ with initial condition $y(0) = 1$ by predictor-corrector method.

Hint : Define $f(x, y) = x + y$

Input : $x_0 = 0, y_0 = 1, x = 0.5, n = 5$

Output : $y(0.5) = 1.7968$

3. Write a C program for Milne's method to find $y(1)$ for the equation $dy/dx = x - y^2$ with initial condition $y(0) = 0$.

Hint : Define $f(x, y) = x - y * y$

Input : $x_0 = 0, y_0 = 0, x = 1, n = 5$

Output : $y(1) = 0.45552$
