(c) Program No. 3 LAGRANGE'S FORMULA

```
# include<iostream.h>
# include<iostream.h>

void main (void)
  {
   float table[10][2], xp,temp,ans=0.0;
   int no, y=0,a=7,i,j;

   cout<<"How Many Values Of X   :   ";
   cin>>no;
   cout<<"\nEnter The Values Of X and f(x)\n";
   cout<<"\n\t      x              |              f(x)";
   cout<<"\n\t------------------------------------";

   for(i=0;i<no;i++)                                    // Input of X & Fx
     {
       gotoxy(11,a);
       cin>>table[i][y];
       gotoxy(21,a);
       cin>>table[i][y+1];
       a++;
     }

  cout<<"\nEnter The Value Of X   :   ";
  cin>>xp;

  for(j=0;j<no;j++)                                     // calculation of formula
    {
      temp=1;
      for(i=0;i<no;i++)
        if(i!=j)
```



```
           temp*=((xp-table[i][0] / (table[j][0]-[i][0]));

      ans+=temp*table[j][1];
    }
  cout<<"\nA N S W E R =          : "<<ans;        //output
}
```

Computer Output

How Many Values of X : 4

Enter the Values of x and f(x)

x	f(x)
1	4
3	7
4	8
6	11

Enter The Value of X : 5

A N S W E R : 9.2

Computer Program No 7: Trapezoidal Rule

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

float returnval;

float f(float x);
  {
    returnval = 0;
    returnval = sqrt (x);
    cout<<"\n\tX: "<<x<<\t\tf(x) : "<< returnval";
    return returnval;
  }

void main ( )
  {
    float low, up, interval, sum=0, steplen;

    clrscr ( );
    cout<<"\n\tENTER THE LOWER LIMIT : "; cin>>low;
    cout<<"\n\tENTER THE UPPER LIMIT : "; cin>>up;
    cout<<"\n\tENTER THE INTERVAL : "; cin>>interval;
    steplen = (up – low / interval;
    sum = f(low) + f(up);
    cout<<"\n\n\tTHE STEPLENGTH IS : "; >>steplen;
    cout<<"\n\tTHE SUM IS : "; "<<sum<<"\n";
    for(int i=1; i< interval; i++)
```



```
      {
        sum += 2 * f(low + i*steplen);
        cout<<"\tSUM : "<<sum;
      }
    sum =(sum*steplen) / 2.0;
    cout<<"\n\n\tFINAL RESULT BY TRAPEZOIDAL RULE IS : "sum;
  }
```

Computer Output

```
ENTER THE LOWER LIMIT  : 1

ENTER THE UPPER LIMIT  : 2

ENTER THE INTERVAL :  4

    X: 1            f(x) : 1
    X: 2            f(x) : 1.414214

THE STEPLENGTH IS :  0.25
THE SUM IS :  2.414214

    X: 1.25         f(x) : 1.118034         SUM : 4.650282
    X: 1.5          f(x) : 1.224745         SUM : 7.099771
    X: 1.75         f(x) : 1.322876         SUM : 9.745522

FINAL RESULT BY TRAPEZOIDAL RULE IS :  1.21819
```

Computer Program No 8: Trapezoidal Rule

Program No. 9: **Simpson's $\frac{1}{3}$rd Rule**

Note: The input functional values are generated using the given function.

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

float returnval;

float f(float x);
  {
    returnval = 0;
    returnval = sqrt (x);
    cout<<"\n\tX: "<<x<<\t\tf(x) : "<< returnval";
    return returnval;
  }

void main ( );
  {
    float low, up, interval, sum=0, steplen, multi=4;

    clrscr ( );
    cout<<"\n\tENTER THE LOWER LIMIT :  "; cin>>low;
    cout<<"\n\tENTER THE UPPER LIMIT :  "; cin>>up;
    cout<<"\n\tENTER THE INTERVAL :  "; cin>>interval;
    steplen = (up – low / interval;
    sum = f(low) + f(up);
    cout<<"\n\n\tTHE STEPLENGTH IS :  "; >>steplen;
    cout<<"\n\tTHE SUM IS :  "; "<<sum<<"\n";
    for(int i=1; i<interval; i++)
      {
        sum += multy * f(low + i*steplen);
        multy =6 – multi;
        cout<<"\tsum : "<<sum;
      }
    sum =(sum*steplen) / 3.0;
```

```
cout<<"\n\n\tENTER VALUE OF X          : " ; cin >>x ;
cout<<"\n\tENTER VALUE OF Y  :  "; cin >> y ;
cout<<"\n\tENTER UPPER LIMIT OF X : "; cin >> xup;
cout<<"\n\tENTER THE INTERVAL        : "; cin >> h ;
n = (xup-x) / h;
cout<<"\n\tX\tYn\t\tY(n+1)";
cout<<"\n\t-------------------------------\n";
for(int i=0;i<=n;i++)
{
    ynew = y + h * f(x,y);
    cout<<"\n\t"<<x<<"\t"<<y<<"\t\t"<<ynew;
    y = ynew;
    x = x+h;
  }
}
```

Computer Output

SIMPLE EULER'S METHOD

ENTER THE VALUE OF X : 0.0

ENTER THE VALUE OF Y : 1.0

ENTER UPPER LIMIT OF X : 0.5

ENTER THE INTERVAL : 0.1

X	Yn	Y(n+1)
0.0	1.0	1.1
0.1	1.1	1.22
0.2	1.22	1.362
0.3	1.362	1.5282
0.4	1.5282	1.72102
0.5	1.72102	1.943122

RUNGE-KUTTA METHODS

The Runge-Kutta methods are a family of methods derived from the Taylor series

Computer Program No 13: Runge-Kutta Method

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

float function(float x0, float y0)
  {
    float result;
    result=(y0-x0)/(y0+x0);
    return results;
  }

void main(void)
  {
    float k1,k2,k3,k4,k,h,x0,y0,yn;
    int n, i, row, col;
    clrscr( );

    cout<<"\n\tCLASSIC RUNGE-KUTTA METHOD";
    cout<<"\n\tENTER THE VALUE OF X0:  ";
    cin>>x0;
    cout<<"\n\tENTER THE VALUE OF Y0:  ";
    cin>>y0;
    cout<<"\n\tENTER THE VALUE OF h:  ";
    cin>>h;
    cout<<"\n\tENTER THE VALUE OF n:  ";
    cin>>n;
    cout<<"\nn  xn  yn   k1   k2   k3    k4   y(n+1)=y(n)+k";
    cout<<"\n-----------------------------------------------------------";
```



```
row=12;
col=0;
for(i=0;i<n+1; i++)
  {
    k1=h*function(x0,y0);
    k2=h*function(x0+h/2,y0+k1/2);
    k3=h*function(x0+h/2,y0+k2/2);
    k4=h*function(x0+h,y0+k3);
    k=(k1+2*k2+2*k3+k4)/6;
    yn=y0+k;
    gotoxy(col, row);
    cout<<i;
    gotoxy(col+4, row);
    cout<<x0;
    gotoxy(col+8, row);
    cout<<y0;
    gotoxy(col+17, row);
    cout<<k1;
    gotoxy(col+26,row);
    cout<<k2;
    gotoxy(col+35,row);
    cout<<k3;
    gotoxy(col+44, row);
    cout<<k4;
    gotoxy(col+56, row);
    cout<<yn;

    y0+=k;
    x0+=h;
    row+=2;
  }
}
```

```
        k1=h*function(x0,y0);
        k2=h*function(x0+h/2,y0+k1/2);
        k3=h*function(x0+h/2,y0+k2/2);
        k4=h*function(x0+h,y0+k3);
        k=(k1+2*k2+2*k3+k4)/6;
        yn=y0+k;
        gotoxy(col, row);
        cout<<i;
        gotoxy(col+4, row);
        cout<<x0;
        gotoxy(col+8, row);
        cout<<y0;
        gotoxy(col+17, row);
        cout<<k1;
        gotoxy(col+26,row);
        cout<<k2;
        gotoxy(col+35,row);
        cout<<k3;
        gotoxy(col+44, row);
        cout<<k4;
        gotoxy(col+56, row);
        cout<<yn;

        y0+=k;
        x0+=h;
        row+=2;
      }
  }
```

Computer Output

CLASSIC RUNGE-KUTTA METHOD

ENTER THE VALUE OF X0 : 0

ENTER THE VALUE OF Y0 : 1

ENTER THE VALUE OF h : 0.1

ENTER THE VALUE OF n : 5

Computer Program:

```
# include<iostream.h>
# include<conio.h>
# include<process.h>

float interval, x0, p, array [20][20] = {0.0};
int no, col, x,y;
void difftable( )
  {
    cout<<"\tDIFFERENETTABLE";
    cout<<"\n\n\tENTER THE FIRST VALUE :  "; cin>>array[0][0];
    cout<<"\n\tENTER THE INTERVAL :  "; cin>>interval;
    cout<<"\n\tENTER TOTAL NO. OF X :  "; cin>>no;

    for(int i=1; i<no; i++)
        {
          array[i][0]=array[i-1][0]+interval;
        }

    cout<<"\n\tENTER FUNCTIONAL VALUES :  \n";
    for(i=0;i<no;i++)
        {
          cout<<"\tX("<<i<<") =  ";cin>>array[i][1];
        }
```



```
    cout<<"\n\tHOW MANY COLUMNS ARE REQUIRED :  "; cin>>col;
    for(i=2; i<=(col+2); i++)
        {
          for(int j=0; j<=(no-i); j++)
            {
              array[j][i]=array[j+1][i-1]-array[j][i-1];
            }
        }

    clrscr( );
    cout<<"\t\tDIFFERENCE TABLE\n";
    cout<<" X          F(X)  ";
    for(j=1;i<=col;i++)
        {
          cout<<"     col     "<<i;
        }

    cout<<"\n";

    for(i=0;i<no;i++)
        {
          cout<<"      "<<array[i][0]<<"\n\n";
        }

    x=8; y=3;
```

```
    while(((xp-array[i][0])/interval>1)&&(I<no))
      {
        i++;
      }
    x0=i;
    p=(xp-array[x0][0])/interval;
  }

void nford( )
  {
    findx( );

    cout<<"\n\ntanswer =  ";
    cout<<(array[x0][1]+(p*array[x0][2]+(p*(p-1) /2 * array[x0][3])
    +p*(p-1)*(p-2) /6 * array[x0][4])+(p*(p-1)*(p-2)*(p-3) /24 * array[x0][5]);
  }

void nback( )
  {
    findx( );
    cout<<"\n\n\tanswer =  ";
    cout<<(array[x0][1]+(p*array[x0-1][2]+(p*(p+1) /2 * array[x0-2][3])
    +p*(p+1)*(p+2) /6 * array[x0-3][4])+(p*(p+1)*(p+2)*(p+3) /24 * array[x0-4][5]);
  }

void main (void)
  {
    clrscr ( ); difftable ( ); getch ( );

    int choice;
    while (1)
      {
        clrscr ( );
        cout<<"\n\n\t\tMAIN MENU";
        cout<<"\n\n\tFORWARD DIFFERENCE INTERPOLATION FORMULA --- 1";
        cout<<"\n\n\tBACKWARD DIFFERENCE INTERPOLATION FORMULA --- 2";
        cout<<"\n\n\tTO EXIT ------------------------------------------------";
        cout<<"\n\n\n\tENTER YOUR CHOICE  :  ";
        cin>>choice;
        switch(choice)
          {
            case 1:clrscr ( );nford( );getch( );break;
            case 2:clrscr ( );nback( );getch( );break;
            case 3:exit(0)
          }
```

Global error = $|Y(0.5) - y(0.5)| \le |1.79744 - 1.72102| = 0.0764$

Program No. 12: **Euler's Method**

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

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

void main ( )
  {
    float x, y, xup, h, n, ynew;

    cout<<"\n\tSIMPLE EULER'S METHOD";
```



```
    cout<<"\n\n\tENTER VALUE OF X       : "; cin >>x ;
    cout<<"\n\tENTER VALUE OF Y  : "; cin >> y ;
    cout<<"\n\tENTER UPPER LIMIT OF X : "; cin >> xup;
    cout<<"\n\tENTER THE INTERVAL     : "; cin >> h ;
    n = (xup-x) / h;
    cout<<"\n\tX\tYn\t\tY(n+1)";
    cout<<"\n\t-------------------------------\n";
    for(int i=0;i<=n;i++)
    {
      ynew = y + h * f(x,y);
      cout<<"\n\t"<<x<<"\t"<<y<<"\t\t"<<ynew;
      y = ynew;
      x = x+h;
    }
  }
```

Computer Output

SIMPLE EULER'S METHOD

ENTER THE VALUE OF X : 0.0

ENTER THE VALUE OF Y : 1.0

ENTER UPPER LIMIT OF X : 0.5

ENTER THE INTERVAL : 0.1

X	Yn	Y(n+1)
0.0	1.0	1.1
0.1	1.1	1.22
0.2	1.22	1.362
0.3	1.362	1.5282
0.4	1.5282	1.72102
0.5	1.72102	1.943122

7 RUNGE-KUTTA METHODS

The Runge-Kutta methods are a family of methods derived from the Taylor series