Newton Forward Interpolation:

Code:

```
#include<stdio.h>
#include<math.h>
int main()
{
 float x,u1,u,y;
 int i,j,n,fact;
 printf("Enter no. of terms\n");
 scanf("%d",&n);
 float a[n][n+1];
 printf("Enter Values of X\n");
 for(i=0;i<n;i++)
 scanf("%f",&a[i][0]);
 printf("Enter Values of Y\n");
 for(i=0;i<n;i++)
 scanf("%f",&a[i][1]);
 printf("Enter value of x for which you want y\n");
scanf("%f",&x);
 for(j=2;j<n+1;j++){
 for(i=0;i<n-j+1;i++)
 a[i][j] = a[i+1][j-1]-a[i][j-1];
 }
 printf("The Difference Table is as follows:\n");
for(i=0;i<n;i++){
 for(j=0;j<=n-i;j++)
 printf("%f ",a[i][j]);
 printf("\n");
 }
```

```
u= (x - a[0][0])/(a[1][0]-a[0][0]);
y=a[0][1];

u1=u;

fact=1;

for(i=2;i<=n;i++){

y=y+(u1*a[0][i])/fact;

fact=fact*i;

u1=u1*(u-(i-1));

}

printf("\n\nValue at X=%g is = %f", x,y);

return 0;


}
```

Output:

```
Enter no. of terms
5
Enter Values of X
1891 1901 1911 1921 1931
Enter Values of Y
46 66 81 93 101
Enter value of x for which you want y
1895
The Difference Table is as follows:
1891.000000 46.000000 20.000000 -5.000000 2.000000 -3.000000
1901.000000 66.000000 15.000000 -3.000000 -1.000000
1911.000000 81.000000 12.000000 -4.000000
1921.000000 93.000000 8.000000
1931.000000 101.000000
```

Newton Backward Interpolation:

Code:

```
#include<stdio.h>
#include<math.h>
int main(){
 float x,u1,u,y;
 int i,j,n,fact;
 printf("Enter no. of terms\n");
 scanf("%d",&n);
 float a[n][n+1];
 printf("Enter Values of X \n");
 for(i=0;i<n;i++)
 scanf("%f",&a[i][0]);
 printf("Enter Values of Y\n");
 for(i=0;i<n;i++)
 scanf("%f",&a[i][1]);

 printf("Enter value of x for which you want y\n");
scanf("%f",&x);

 for(j=2;j<n+1;j++){
 for(i=0;i<n-j+1;i++)
 a[i][j] = a[i+1][j-1]-a[i][j-1];
 }
 printf("The Difference Table is as follows:\n");
for(i=0;i<n;i++){
 for(j=0;j<=n-i;j++)
 printf("%f ",a[i][j]);
 printf("\n");
```

```
}
u= (x - a[n-1][0])/(a[1][0]-a[0][0]);
y=a[n-1][1];
u1=u;
fact=1;
j=2;
for(i=n-2;i>=0;i--){
y=y+(u1*a[i][j])/fact;
fact=fact*j;
u1=u1*(u+(j-1));
j++;
}
printf("\n\nValue at X=%g is = %f", x,y); }
```

Output:

```
Enter no. of terms
5
Enter Values of X
1891 1901 1911 1921 1931
Enter Values of Y
46 66 81 93 101
Enter value of x for which you want y
1927
The Difference Table is as follows:
1891.000000 46.000000 20.000000 -5.000000 2.000000 -3.000000
1901.000000 66.000000 15.000000 -3.000000 -1.000000
1911.000000 81.000000 12.000000 -4.000000
1921.000000 93.000000 8.000000
1931.000000 101.000000


Value at X=1927 is = 98.468811
```

Lagrange Interpolation:

Code:

```
#include<stdio.h>
int main()
{
 int n;
 printf("Enter no. of terms\n");
 scanf("%d",&n);
 float X[n],Y[n],x,sum=0,term;
 int i,j;

 printf("Enter Values of X \n");
 for(i=0;i<n;i++)
 scanf("%f",&X[i]);
 printf("Enter Values of Y\n");
 for(i=0;i<n;i++)
 scanf("%f",&Y[i]);

 printf("Enter value of x for which you want y\n");
scanf("%f",&x);
 for(i=0;i<n;i++){
 term=1;
 for(j=0;j<n;j++){
 if(i!=j)
 term = term * ((x - X[j])/(X[i]-X[j]));  }
 sum=sum + term * Y[i];
 }
 printf("\nValue at X=%g is = %f", x,sum); }
```

Output:

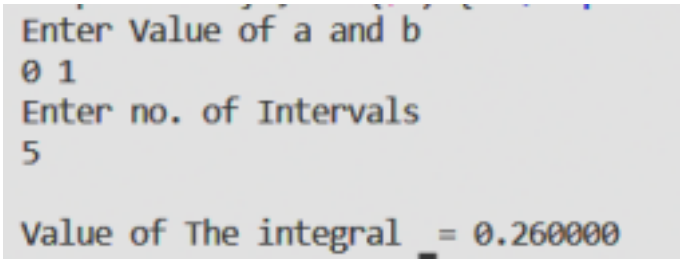
```
Enter no. of terms
4
Enter Values of X
5 6 9 11
Enter Values of Y
12 13 14 16
Enter value of x for which you want y
10

Value at X=10 is = 14.666668
```

Trapezoidal Rule:

Code:

```
#include<stdio.h>
float findValueAt(float x){
 return x*x*x;
}
int main(){
 int n;
 float i,a,b,sum=0,h;
 printf("Enter Value of a and b\n");
scanf("%f%f",&a,&b);
 printf("Enter no. of Intervals\n");
scanf("%d",&n);
 h=(b-a)/n;
 sum = findValueAt(a) +findValueAt(b);
for(i=a+h;i<b;i=i+h)
 sum = sum + 2*findValueAt(i);
 sum = (h * sum)/2;
 printf("\nValue of The integral = %f",sum);   }
```

Output:



Simpson’s 1/3 Rule:

Code:

```
#include<stdio.h>

float findValueAt(float x){
 return 1/(1+x*x);
}
int main(){
 int n;
 float i,a,b,sum=0,h;
 int position_of_term=1;
 printf("Enter Value of a and b\n");
scanf("%f%f",&a,&b);
 printf("Enter no. of Intervals\n");
scanf("%d",&n);
 h=(b-a)/n;
 sum = findValueAt(a) +findValueAt(b);
 for(i=a+h;i<b;i=i+h){
 if(position_of_term %2 ==0)  sum
= sum + 2*findValueAt(i);  else
 sum = sum + 4*findValueAt(i);
position_of_term++;
 }
 sum = (h * sum)/3;

 printf("\nValue of The integral =
%f",sum);
 return 0;
}
```

Output:

```
Enter Value of a and b
0 6
Enter no. of Intervals
6

Value of The integral  = 1.366174
```

Simpson’s 3/8 Rule:

Code:

```
#include<stdio.h>
float findValueAt(float x){
 return 1/(1+x*x);
}
int main(){
 int n;
 float i,a,b,sum=0,h;
 int position_of_term=1;
 printf("Enter Value of a and b\n");
scanf("%f%f",&a,&b);
 printf("Enter no. of Intervals\n");
scanf("%d",&n);

 h=(b-a)/n;
 sum = findValueAt(a) +findValueAt(b);

 for(i=a+h;i<b;i=i+h){
 if(position_of_term %3 ==0)
 sum = sum + 2*findValueAt(i);
else
 sum = sum + 3*findValueAt(i);
position_of_term++;
 }
 sum = (3*h)/8 * sum;
 printf("\nValue of The integral = %f",sum);
}
```

Output:

```
Enter Value of a and b
0 6
Enter no. of Intervals
6

Value of The integral  = 1.357081
```

// CPP Program to find approximation of a ordinary differential equation using Euler method.

```
#include<iostream>
 using namespace std;
 // Consider a differential equation
 // dy/dx=(x + y + xy)
float func(float x, float y) {
return (x + y + x * y);
 }
 // Function for Euler formula
 void euler(float x0, float y, float h, float x) {
 float temp = -0;
// Iterating till the point at which we need approximation
 while (x0 < x) {
 temp = y;
y = y + h * func(x0, y);
 x0 = x0 + h;
 }
// Printing approximation
 cout << "Approximate solution at x = "<< x << " is " << y << endl;
 }
 // Driver program
int main() {
 // Initial Values
float x0 = 0;
float y0 = 1;
 float h = 0.025;
 // Value of x at which we need approximation
```

```
 float x = 0.1;
euler(x0, y0, h, x);
 return 0;
 }
```

Output:

// C++ program of Runge-Kutta Method to Solve Differential Equation

```
#include <bits/stdc++.h>
using namespace std;
// A sample differential equation "dy/dx = (x - y)/2"
float dydx(float x, float y)
{
return((x - y)/2);
}
// Finds value of y for a given x using step size h and initial value y0 at x0.
float rungeKutta(float x0, float y0, float x, float h)
{
// Count number of iterations using step size or step height h
int n = (int)((x - x0) / h);
float k1, k2, k3, k4, k5;
// Iterate for number of iterations
float y = y0;
for (int i=1; i<=n; i++)
{
// Apply Runge Kutta Formulas to find
// next value of y
k1 = h*dydx(x0, y);
k2 = h*dydx(x0 + 0.5*h, y + 0.5*k1);
k3 = h*dydx(x0 + 0.5*h, y + 0.5*k2);
k4 = h*dydx(x0 + h, y + k3);
// Update next value of y
y = y + (1.0/6.0)*(k1 + 2*k2 + 2*k3 + k4);;
// Update next value of x
x0 = x0 + h;
```

```
}

return y;

}

// Driver Code

int main()

{

float x0 = 0, y = 1, x = 2, h = 0.2;

cout << "The value of y at x is : " <<rungeKutta(x0, y, x, h);

return 0;

}
```

Output: