Title: Exploring Functional Values through Newton's Forward and Backward Interpolation Formula.

Theoretical Background:

Newton's Forward Interpolation Formula:

The forward interpolation method estimates the value of a function at a specified point within the range of given values. It involves constructing a forward difference table based on known data points and their respective function values. Using these divided difference values, the formula predicts the value of the function at a specified point.

Given n+1 equally spaced data points (x_0, y_0), (x_1, y_1), ..., (x_n, y_n) with equal intervals h, setting $x=x_0+ph$ we get the formula for forward interpolation is:

$$y_n(x) = y_0 + p\Delta y_0 + \frac{p(p-1)}{2!}\Delta^2 y_0 + \frac{p(p-1)(p-2)}{3!}\Delta^3 y_0 + \ldots. + \frac{p(p-1)(p-2)\ldots\ldots(p-n+1)}{n!}\Delta^n y_0$$

Newton's Backward Interpolation Formula:

Like forward interpolation, backward interpolation estimates the value of a function within the range of given values. However, it works backward from the data points to compute the divided differences and subsequently predicts the value at the specified point.

Given n+1 equally spaced data points (x_0, y_0), (x_1, y_1), ..., (x_n, y_n) with equal intervals h, setting $x=x_n+ph$ we get the formula for forward interpolation is:

$$y_n(x) = y_n + p\nabla y_n + \frac{p(p+1)}{2!}\nabla^2 y_n + \frac{p(p+1)(p+2)}{3!}\nabla^3 y_n + \ldots + \frac{p(p+1)(p+2)\ldots\ldots(p+n-1)}{n!}\nabla^n y_n$$

These methods offer a means to estimate the value of a function at points where the actual function values are not explicitly given but can be interpolated from the available data. They find applications in numerical analysis, engineering, and scientific computations for approximating values within a set of given data points.

Program:

```
#include <iostream>
#include <windows.h>
using namespace std;
double calculateCombination(double n, double r, bool h)
{
    double result = 1;
    for (int i = 1; i <= r; ++i)
    {
        if (h == true)
            result *= (n - i + 1) / i;
        else if (h == false)
            result *= (n + i - 1) / i;
    }
    return result;
}

void NewtonForward(double x, int n, double ys[], double xs[], double h)
{
    double sum = ys[0];
    cout << "x   ";
    for (int i = 0; i < n; i++)
        cout << xs[i] << " ";
    cout << endl;
    cout << "y   ";
    double ys1[n];
    for (int i = 0; i < n; i++)
    {
        ys1[i] = ys[i];
        cout << ys1[i] << " ";
    }

    cout << endl;
    for (int j = 0; j < n - 1; j++)
    {
        cout << "\u0394" << j + 1 << "y  ";
        for (int i = 0; i < n - (j + 1); i++)
        {
            cout << ys1[i + 1] - ys1[i] << " ";
            ys1[i] = ys1[i + 1] - ys1[i];
        }
        cout << endl;
        double p = (x - xs[0]) / h;
        sum += (calculateCombination(p, j + 1, true) * ys1[0]);
```

```
    }
    cout << "The Answer is: " << sum << endl;
}
void NewtonBackward(double x, int n, double ys[], double xs[], double h)
{
    double sum = ys[n - 1];
    cout << "x    ";
    for (int i = n - 1; i >= 0; i--)
        cout << xs[i] << " ";
    cout << endl;
    cout << "y    ";
    for (int i = n - 1; i >= 0; i--)
        cout << ys[i] << " ";
    cout << endl;
    for (int j = 0; j < n - 1; j++)
    {
        cout << "\u0394" << j + 1 << "y  ";
        for (int i = n - 2; i >= j; i--)
        {
            cout << ys[i + 1] - ys[i] << " ";
            ys[i + 1] = ys[i + 1] - ys[i];
        }
        cout << endl;
        double p = (x - xs[n - 1]) / h;
        // cout << p << endl;
        // cout << calculateCombination(p, j + 1, false) << " " << y[n - 1] <<
endl;
        sum += (calculateCombination(p, j + 1, false) * ys[n - 1]);
        // cout << sum << endl;
    }
    cout << "The Answer is: " << sum << endl;
}
int main()
{
    SetConsoleOutputCP(CP_UTF8);
    int n;
    cout << "How many points: ";
    cin >> n;
    double x[n], y[n];
    cout << "Enter x0 Values: ";
    for (int i = 0; i < n; i++)
        cin >> x[i];
    cout << "Enter y0 Values: ";
    for (int i = 0; i < n; i++)
        cin >> y[i];
```

```
    cout << "Enter x value: ";
    double x1;
    cin >> x1;
    double h = x[1] - x[0];
    while (1)
    {
        cout << "InterPolation\n1. Newton's Forward\n2. Newton's Backward\nplease
choose:" << endl;
        int choice;
        cin >> choice;
        switch (choice)
        {
        case 1:
            NewtonForward(x1, n, y, x, h);
            break;
        case 2:
            NewtonBackward(x1, n, y, x, h);
            break;
        }
    }
}
```

Input and Output:

```
PS C:\Users\Tonmo\OneDrive\Documents\RUE
How many points: 4
Enter x0 Values: 1 3 5 7
Enter y0 Values: 24 120 336 720
Enter x value: 8
InterPolation
1. Newton's Forward
2. Newton's Backward
please choose:
1
x   1 3 5 7
y   24 120 336 720
Δ1y  96 216 384
Δ2y  120 168
Δ3y  48
The Answer is: 990
InterPolation
1. Newton's Forward
2. Newton's Backward
please choose:
2
x    7 5 3 1
y    720 336 120 24
Δ1y  384 216 96
Δ2y  168 120
Δ3y  48
The Answer is: 990
InterPolation
1. Newton's Forward
2. Newton's Backward
please choose:
```

Discussion:

Newton's Forward and Backward Interpolation Formulas are distinct methods used to estimate unknown function values between known data points. Newton's Forward Interpolation moves in a forward direction from known data towards the desired point, employing forward differences to calculate values. It is suitable for predicting future values within the given range of data points. Conversely, Newton's Backward Interpolation moves in a reverse direction from known data towards the desired point, using backward differences to estimate values. This method is effective for estimating past values or values beyond the provided data range. Both techniques involve constructing difference tables but operate in opposite directions, catering to different scenarios based on the directionality of required estimations in relation to the available data points. The choice between these methods depends on whether future or past values need to be predicted and their relation to the existing data range.