```c
/**
 * @author  Aviruk Basak, CSE214047, Sem 3, Year 2
 * @topic   Using Newton's Forward and Backward Interpolation to find f(a) for a given data set and a given value of 'a'
 * @date    27-7-2022
 */

# include <stdio.h>
# include <stdlib.h>
# include <stdbool.h>

typedef const enum {
    FORWARD = 0,
    BACKWARD = 1,
    RESET = 2
} Mode;

size_t factorial(size_t n);
bool validateDataSetX(double *arr_x, size_t total_points);
double getNextDiff(Mode mode, double *arr_y, size_t total_points);
double newtonsInterp(Mode mode, double *arr_x, double *arr_y, size_t total_points, double a);

int main()
{
    size_t total_points, i;
    double *arr_x,         // X coordinates array
           *arr_y,         // Y coordinates array
           a,              // value whose f(a) is to be found
           rslt_fw,        // result from forward interpolation
           rslt_bw,        // result from backward interpolation
           err_a,          // exact error b/w rslt_fw and rslt_bw
           err_r,          // relative error b/w rslt_fw and rslt_bw
           err_p;          // percentage error b/w rslt_fw and rslt_bw

    printf("Enter total points = ");
    scanf("%zu", &total_points);

    arr_x = malloc(sizeof(double) * total_points);
    arr_y = malloc(sizeof(double) * total_points);

    printf("Enter items of array X = ");
    for (i = 0; i < total_points; i++)
        scanf("%lf", &arr_x[i]);

    if (!validateDataSetX(arr_x, total_points)) {
        printf("error: X data set doesn't have a common difference\n");
        abort();
    }

    printf("Enter items of array Y = ");
    for (i = 0; i < total_points; i++)
        scanf("%lf", &arr_y[i]);

    while (1) {
        printf("\nEnter value of a = ");
        scanf("%lf", &a);
        rslt_fw = newtonsInterp(FORWARD, arr_x, arr_y, total_points, a);
        rslt_bw = newtonsInterp(BACKWARD, arr_x, arr_y, total_points, a);
        if (a - (long long int) a == 0) {
            printf("fwd: f(%lld) = %0.5lf\n", (long long int) a, rslt_fw);
            printf("bkw: f(%lld) = %0.5lf\n", (long long int) a, rslt_bw);
        } else {
            printf("fwd: f(%0.5lf) = %0.5lf\n", a, rslt_fw);
            printf("bkw: f(%0.5lf) = %0.5lf\n", a, rslt_bw);
        }

        printf("\nErrors:\n");
        err_a = rslt_fw - rslt_bw;
        err_r = err_a / rslt_fw;
        err_p = err_r * 100;
        printf("exact error      = %lf\n", err_a);
        printf("relative error   = %lf\n", err_r);
        printf("percentage error = %lf\n", err_p);
    }

    getNextDiff(RESET, arr_y, total_points);
    free(arr_x);
    free(arr_y);

    return 0;
}

size_t factorial(size_t n)
{
    size_t f = 1;
    for (; n > 1; n--) {
        f *= n;
    }
    return f;
}
```

```c

bool validateDataSetX(double *arr_x, size_t total_points)
{
    size_t i;
    for (i = 1; i < total_points -1; i++) {
        if (arr_x[i +1] - arr_x[i] != arr_x[i] - arr_x[i -1]) {
            return false;
        }
    }
    return true;
}

/**
 * use mode = FORWARD, BACKWARD or RESET
 */
double getNextDiff(Mode mode, double *arr_y, size_t total_points)
{
    static double *arr_term = NULL;     // array of difference term
    static size_t diff_index = 0;       // index of the difference term, 0 value indicates y array
    size_t i;                           // index of the difference term array
    // resetting static variables
    if (mode == RESET) {
        if (arr_term) {
            free(arr_term);
            arr_term = NULL;
        }
        diff_index = 0;
        return 0;
    }
    // 1st time this fn runs
    if (diff_index == 0) {
        arr_term = malloc(sizeof(double) * total_points);
        // copy y array to terms array
        for (i = 0; i < total_points; i++) {
            arr_term[i] = arr_y[i];
        }
    }
    // calculation loop
    if (diff_index) {
        for (i = 0; i < total_points - diff_index +1; i++) {
            arr_term[i] = arr_term[i +1] - arr_term[i];
        }
    }
    diff_index++;
    return mode == BACKWARD ? arr_term[total_points - diff_index] : arr_term[0];
}

double newtonsInterp(Mode mode, double *arr_x, double *arr_y, size_t total_points, double a)
{
    getNextDiff(RESET, arr_y, total_points);
    double p = (a - arr_x[0]) / (arr_x[1] - arr_x[0]);
    size_t i, j, result = 0;
    for (i = 0; i < total_points; i++) {
        double product = 1;
        for (j = 0; j < i; j++) {
            product *= (mode == FORWARD ? p - j : p + j);
        }
        product /= factorial(i);
        double dt = getNextDiff(mode, arr_y, total_points);
        product *= dt;
        result += product;
    }
    return result;
}
```