Assignment 3

By CO24BTECH11023

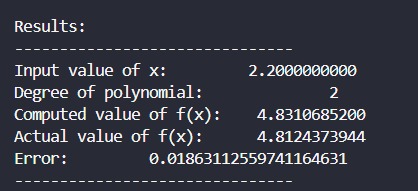
# Question

The following table gives Equi spaced data of f(x)=. Compare error (PK – f(x)) where K=2,3,4. At x=2.2.

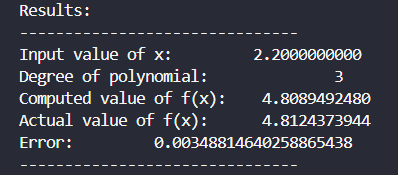
|  |  |
| --- | --- |
| X | f(x) |
| 1 | 1.225541 |
| 1.5 | 2.320117 |
| 2 | 3.953032 |
| 2.5 | 6.389056 |
| 3 | 10.023176 |

# Answers

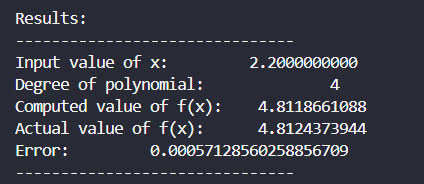
### For Degree 2:



### For Degree 3:

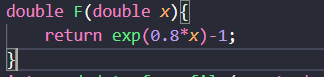


### For Degree 4:



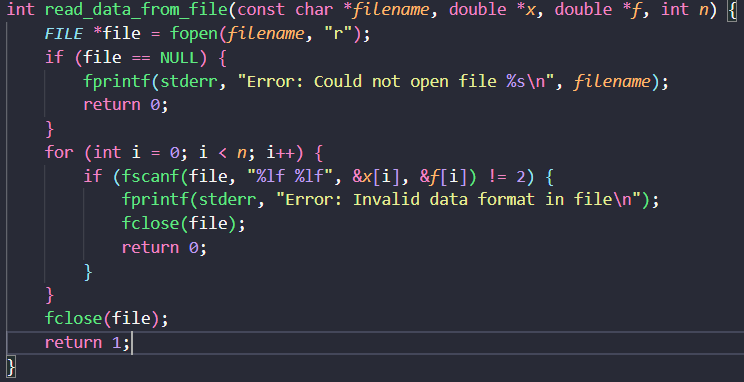
# Code Explanation

### Function F:



* Inputs : x ,the input for f(x).
* Returns
* Uses math.h library.

### Function read\_data\_from\_file:



This function reads data from a file and stores it in two arrays, x and f. It is designed to read pairs of floating-point numbers from the file, which can represent x-values and corresponding function values (f(x)).

**Inputs**

1. **const char \*filename**:
   * The name of the file to be read. This file should contain data in a specific format (pairs of floating-point numbers separated by spaces or newlines).
2. **double \*x**:
   * A pointer to an array where the first column of data (x-values) will be stored.
3. **double \*f**:
   * A pointer to an array where the second column of data (f-values) will be stored.
4. **int n**:
   * The number of data points to read from the file. The function assumes that both arrays (x and f) are pre-allocated with at least n elements.

**Outputs**

1. **Return Value (int)**:
   * Returns 1 if the function successfully reads all n data points.
   * Returns 0 if there is an error (e.g., file not found, invalid format, or insufficient data).
2. **Side Effects**:
   * Populates the arrays x and f with the data read from the file.

**Algorithm**

* The read\_data\_from\_file function reads n pairs of floating-point numbers from a file and store them in two arrays, x and f.
* It begins by opening the specified file in read mode and checks for errors in opening. If successful, it iterates through the file, using fscanf to read each pair of numbers into the arrays.
* During each iteration, it validates the input format by ensuring that exactly two numbers are read; if this condition is not met, the function prints an error message, closes the file, and exits with a failure status.
* Once all data points are successfully read, the file is closed, and the function returns a success status.

**Reason for Creation:**  
The code was designed with scalability in mind, as it is intended to handle large datasets efficiently. Manually inputting such extensive data would be time-consuming and error-prone, so reading the data directly from a file simplifies the process, ensures accuracy, and enhances usability for general-purpose applications.

### Function compute\_forward\_differences:

A computer screen shot of a code

AI-generated content may be incorrect.

This function computes the forward difference table for a given set of function values f.

**Inputs**

1. **double \*f**:
   * A 1D array containing the function values f[i] corresponding to the given data points.
   * Size: n.
2. **double \*\*forward\_diff**:
   * A 2D array to store the computed forward difference table.
   * The first column is initialized with the function values, and subsequent columns contain higher-order differences.
3. **int n**:
   * The number of data points.

**Outputs**

1. **Side Effects**:
   * Populates the 2D array forward\_diff with the forward difference table:
     + The first column contains the original function values (f).
     + Each subsequent column contains higher-order forward differences.
2. **No Return Value**:
   * The function modifies the forward\_diff array in-place.

**Algorithm**

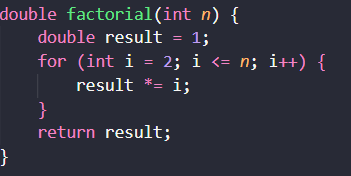
**Initialize the First Column**:

The first column of the forward\_diff table is initialized with the given function values f[i].

**Compute Higher-Order Differences**:

1. Outer Loop (j): Iterates over the columns of the table, starting from the second column (first-order differences) to the last column (n-th order differences).
   * Each column corresponds to a higher-order difference.
2. Inner Loop (i): Iterates over the rows of each column.
   * Computes each forward difference as difference between consecutive entries in the previous column.
3. **Storage in Triangular Form**:
   * The resulting forward\_diff table is triangular, where:
     + The first column contains f[i].
     + The second column contains first-order differences.
     + The third column contains second-order differences, and so on.
   * For example, for n = 4, the table will look like this:
     + f[0] Δf[0] Δ²f[0] Δ³f[0]
     + f[1] Δf[1] Δ²f[1]
     + f[2] Δf[2]
     + f[3]

### Function Factorial:



This function computes the factorial of a given non-negative integer n.

**Inputs**

1. **int n**:
   * A non-negative integer for which the factorial is to be computed.

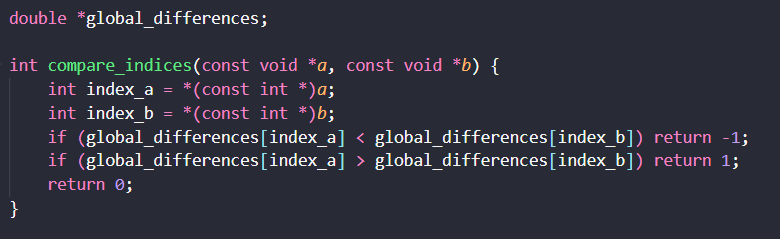
**Outputs**

1. **Return Value (double)**:
   * Returns the factorial of n as a double-precision floating-point number (result).
   * If n = 0, the function returns 1 (by definition, 0! = 1).

**Algorithm**

* A simple factorial function made by multiplying numbers using loops.

### Function compare\_indices and variable global\_differences:



This comparator function is designed to sort indices based on the values stored in the global array global\_differences. It enables sorting an array of indices (e.g., [0, 1, 2, ..., n-1]) such that when accessed via these indices, the global\_differences array appears in ascending order.

**Inputs**

1. **const void \*a** and **const void \*b**:
   * Pointers to two integers representing indices in the global\_differences array.
   * Example: If a points to 3 and b points to 5, the function compares global\_differences and global\_differences.

**Outputs**

1. **Return Value (int)**:
   * **-1**: If the value at index a is **less than** the value at index b.
   * **1**: If the value at index a is **greater than** the value at index b.
   * **0**: If the values at indices a and b are equal.

**Algorithm**

**Step-by-Step Explanation:**

1. **Extract Indices**:
   * Convert the void pointers to integer indices.
2. **Compare Values**:
   * Compare the values in global\_differences at the extracted indices.
   * Return a sorting order compatible with qsort.

**Reason for Creation:**

* The given function is going to be used in qsort which is sorting function in stdlib.h which allows the user to sort arrays/unions/structures etc using a user defined sorting algorithm. Its time complexity is nlogn making it very fast.
* We are using it to indirectly sort the array of indices (e.g., [0, 1, 2, ...]) based on the values in global\_differences.
* Since qsort does not allow passing additional parameters to the comparator, global\_differences is declared globally to provide access to the data being compared.

### Function find\_start\_index:

A computer screen shot of a program code

AI-generated content may be incorrect.

This function identifies the optimal starting index in a dataset x for constructing an interpolating polynomial of a specified degree around a target point x\_i. It ensures the selected points are the closest to x\_i and contiguous in the original dataset.

**Inputs**

1. **double \*x**:
   * Array of data points (e.g., x-values from a dataset).
2. **int n**:
   * Number of data points in x.
3. **double x\_i**:
   * Target point for interpolation.
4. **int degree**:
   * Degree of the interpolating polynomial.

**Outputs**

* **Return Value (int)**:  
  The starting index in x for the closest contiguous points around x\_i to construct the polynomial.

**Algorithm**

**Compute Absolute Differences**:

**double** \*differences = malloc(n \* **sizeof**(**double**));

**for** (**int** i = 0; i < n; i++) {

differences[i] = fabs(x\_i - x[i]); *// Distance from x\_i to each x[j]*

}

* + Calculate the absolute difference between x\_i and every point in x.

1. **Initialize Indices**:

**int** \*indices = malloc(n \* **sizeof**(**int**));

**for** (**int** i = 0; i < n; i++) {

indices[i] = i; *// [0, 1, 2, ..., n-1]*

}

* + Create an array of indices (0 to n-1) to track original positions.

1. **Sort Indices by Proximity**:

global\_differences = differences;

qsort(indices, n, **sizeof**(**int**), compare\_indices);

* + Use qsort to sort indices based on differences, so indices is the index of the closest point to x\_i.

1. **Select Closest Points**:

**int** \*selected\_points = malloc((degree + 1) \* **sizeof**(**int**));

**for** (**int** i = 0; i <= degree; i++) {

selected\_points[i] = indices[i]; *// Top (degree+1) closest points*

}

* + Extract the first degree + 1 indices (closest points to x\_i).

1. **Find Minimum Index**:

**int** start\_index = selected\_points[0];

**for** (**int** i = 1; i <= degree; i++) {

**if** (selected\_points[i] < start\_index) {

start\_index = selected\_points[i]; *// Find leftmost index*

}

}

* + Determine the smallest index among the selected points to ensure continuity in the original dataset.

1. **Cleanup**:
   * Free dynamically allocated memory.

### Function newtonian\_forward\_polynomial:

A screen shot of a computer code

AI-generated content may be incorrect.

This function evaluates the Newton Forward Difference Interpolation Polynomial at a given point x\_i using precomputed forward differences. It is designed for evenly spaced data .(Here instead of ‘r’ we have taken ‘u’.)

Now Newtonian polynomial is given by the formula:

Where k is the degree of the polynomial.

**Inputs**

1. **double \*x**:
   * Array of x-values from the dataset (assumed to be **evenly spaced**).
2. **double \*\*forward\_diff**:
   * Precomputed forward difference table (2D array).
3. **int degree**:
   * Degree of the interpolating polynomial.
4. **double x\_i**:
   * The point at which to evaluate the interpolated value.
5. **int start\_index**:
   * Starting index in x for the closest contiguous points around x\_i.

**Outputs**

* **Return Value (double)**:  
  The interpolated value at x\_i computed using the Newton Forward Difference formula.

**Algorithm**

**Initialize Result**:

**double** result = forward\_diff[start\_index][0];

* + The first term of the polynomial is the function value at start\_index (from the forward difference table).

1. **Compute (u)**:

**double** u = (x\_i - x[start\_index]) / (x[start\_index + 1] - x[start\_index]);

* + u represents the distance from x[start\_index] to x\_i, scaled by the step size h = x[start\_index + 1] - x[start\_index].

1. **Iterative Polynomial Construction**:

**double** term = 1.0;

**for** (**int** i = 1; i <= degree; i++) {

term \*= (u - (i - 1));

result += (term \* forward\_diff[start\_index][i]) / factorial(i);

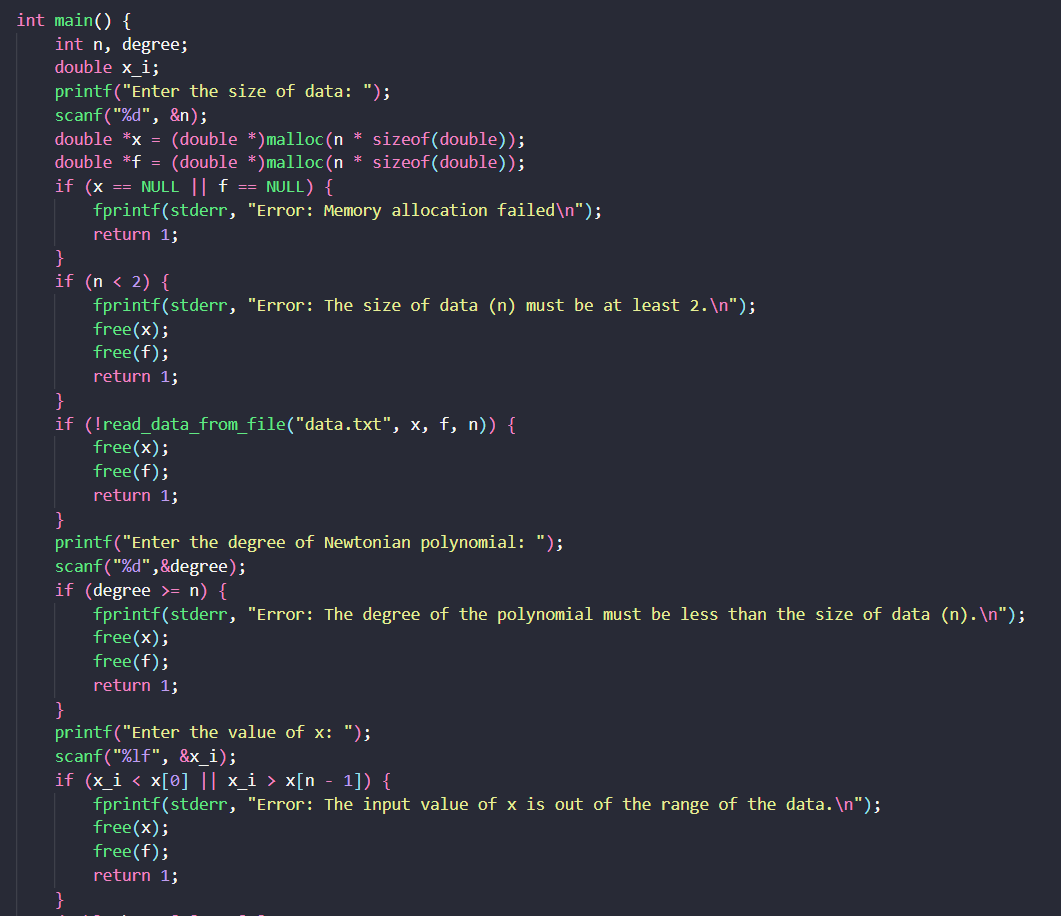
}

* + **Loop**: For each term in the polynomial (up to the specified degree):
    1. **Update term**: Multiply by (u - (i - 1)) to compute the product term u(u-1)(u-2)....
    2. **Add Term**: Incorporate the forward difference and factorial scaling:

result+=

### Function Main:

#### Taking Inputs:



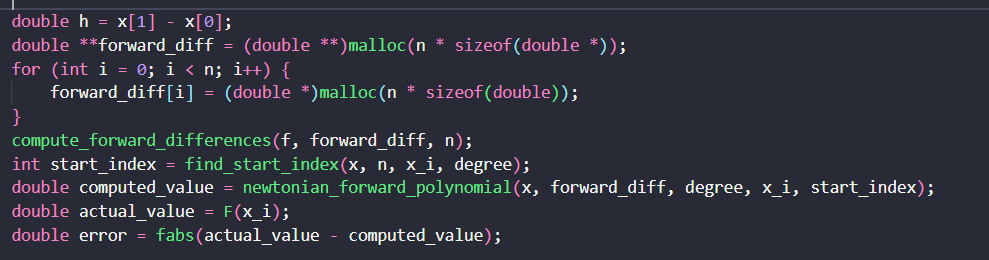
**Inputs:**

1. **n**: Size of the dataset (must be ≥ 2).
2. **x and f**: Arrays to store data points (read from data.txt).
3. **degree**: Degree of the interpolating polynomial (must be < n).
4. **x\_i**: Target x-value for interpolation (must lie within the range [x, x[n-1]]).

**Key Checks:**

1. **Memory Allocation**: Validates if x and f arrays are allocated successfully.
2. **Data Size**: Ensures n ≥ 2 (required for interpolation).
3. **File Input**: Checks if data.txt is read correctly.
4. **Polynomial Degree**: Ensures degree < n (avoids overfitting).
5. **Interpolation Range**: Validates x\_i is within the dataset’s x-range.

#### Computation Part



**Code Explanation:**

1. **Step Size Calculation**:
   * Calculating the step size h between consecutive x-values.
2. **Forward Difference Table Setup**:

**double** \*\*forward\_diff = (**double** \*\*)malloc(n \* **sizeof**(**double** \*));

**for** (**int** i = 0; i < n; i++) {

forward\_diff[i] = (**double** \*)malloc(n \* **sizeof**(**double**));

}

compute\_forward\_differences(f, forward\_diff, n);

* + Allocates memory for an n x n matrix to store forward differences.
  + Populates the matrix using the function compute\_forward\_differences.

1. **Optimal Starting Index**:
   * Finds the starting index in x for the closest contiguous points around x\_i to construct the polynomial using find\_start\_index function.
2. **Interpolation & Error Calculation**:
   * Computes the interpolated value at x\_i using the newtonian\_forward\_polynomial function .
   * Compares against the actual function value F(x\_i) to calculate the absolute error.

#### Printing result and freeing memory:

A computer screen shot of a program code

AI-generated content may be incorrect.

**Output & Cleanup**

1. **Formatted Results Display**:
   * Prints the interpolation results in a structured table:
     + Input x\_i
     + Polynomial degree used
     + Computed value (f(x\_i))
     + Actual value (from F(x\_i))
     + Absolute error
2. **Memory Deallocation**:
   * Frees the 2D forward\_diff table (row-by-row).
   * Releases memory for x and f arrays.

Libraries used –

* + Stdio.h
  + Stdlib.h
  + Math,h

# Thank you