**PROGRAM: 9**

**Name: Kiran Kanyal**

**Roll No: 29**

**Section: A1**

**Aim: Write a program in C language to implement Newton’s Forward Interpolation method.**

**ALGORITHM:**

**START**

1. Prompt the user to input the number of observations n.
2. Prompt the user to input the value of x[i] and f(x[i]) for n observations.
3. Construct a forward difference table where each entry is the difference between consecutive values of f(x) from the previous level.
4. Prompt the user to input the value of x for which f(x) needs to be interpolated.
5. Find the value of h and u.
   1. h = x[1] – x[0]
   2. u = (x – x[0]) / h
6. Use Newton’s Forward Interpolation formula to compute the interpolated value of f(x).
7. Print the forward difference table & the interpolated value of f(x).

**STOP**

**PROGRAM:**

#include <stdio.h>

#include <math.h>

int main()

{

int n;

printf("Enter the number of observations: ");

scanf("%d", &n);

float x[n];

float y[n][n];

printf("Enter the value of X and Y (=f(X)) for %d terms :\n", n);

printf("X Y\n");

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

{

scanf("%f", &x[i]);

scanf("%f", &y[i][0]);

}

float xValue;

printf("Enter the point at which you want to find the value: ");

scanf("%f", &xValue);

float a = x[0];

float h = x[1] - x[0];

float u = (xValue - a) / h;

for (int j = 1; j < n; j++)

{

for (int i = 0; i < n - j; i++)

{

y[i][j] = y[i + 1][j - 1] - y[i][j - 1];

}

}

printf("Forward Difference Table: \n ");

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

{

for (int j = 0; j < n - i; j++)

{

printf("%.2f\t", y[i][j]);

}

printf("\n");

}

float res = y[0][0];

float uTerm = 1;

float fact = 1;

for (int i = 1; i < n; i++)

{

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

fact \*= i;

res += (uTerm \* y[0][i]) / fact;

}

printf("The interpolated value at x = %.4f is %.4f", xValue, res);

return 0;

}

**OUTPUT:**

For the given observation: f (1.7) =?

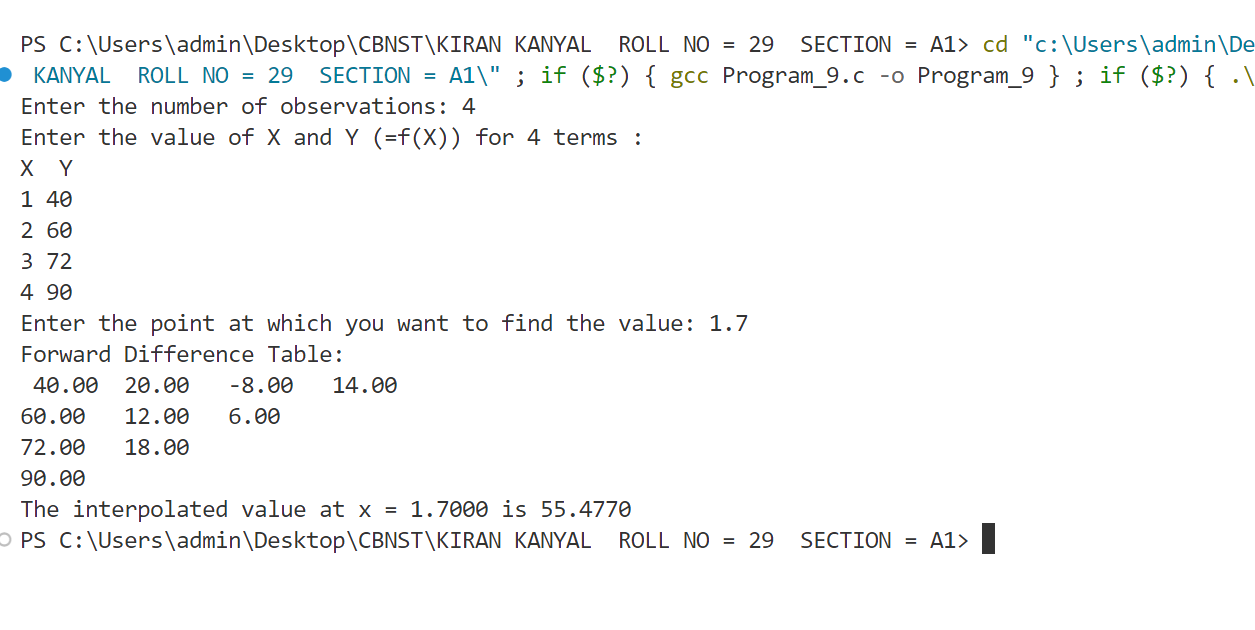
X Y

1 40

2 60

3 72

4 90



**PROGRAM: 8**

**Name: Kiran Kanyal**

**Roll No: 29**

**Section: A1**

**Aim: Write a program in C language to find the solution for the given system of equations using the Gauss-Seidel method.**

**ALGORITHM:**

**START**

1. Prompt the user to input the order of the matrix n.
2. Input the coefficients of the system in matrix A[n][n+1] (including constants).
3. Check for diagonal dominance:
   1. For each row, if the diagonal element is less than the sum of the absolute values of the non-diagonal elements, terminate the program.
4. Prompt the user to input the allowed error allowedErr.
5. Initialize initial guess values for unknown as zero.
6. Repeat until the solution converges (i.e. until the maximum error for variables is greater than allowedErr):
   1. For each unknown i: Calculate the new value of x[i] by isolating it from the equation.
   2. Calculate the error as the absolute difference between the old and new values of x[i].
   3. Update the maximum error.
   4. Print the iteration count and the current values of x.
7. Print the final values of x when the maximum error is within the tolerance.

**STOP**

**PROGRAM:**

#include <stdio.h>

#include <math.h>

int main() {

int n;

printf("Enter the number of unknowns (order of the matrix): ");

scanf("%d", &n);

float A[n][n + 1];

float x[n], x\_old[n];

float allowedErr;

int i, j, count = 1;

printf("Enter coefficients of the system and constants for x1, x2, ..., x%d:\n", n);

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

float diagonal = 0;

float sum = 0;

for (j = 0; j <= n; j++){

scanf("%f", &A[i][j]);

if (i == j)

diagonal = fabs(A[i][j]);

else if (j != n)

sum += fabs(A[i][j]);

}

if (diagonal < sum){

printf("The system does not satisfy the diagonal dominance condition.\n");

return -1;

}

}

printf("Enter allowed error: ");

scanf("%f", &allowedErr);

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

x[i] = 0;

x\_old[i] = 0;

}

float max\_error;

do {

max\_error = 0;

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

float sum = 0;

for (j = 0; j < n; j++) {

if (i != j) {

sum += A[i][j] \* x[j];

}

}

x\_old[i] = x[i];

x[i] = (A[i][n] - sum) / A[i][i];

float error = fabs(x[i] - x\_old[i]);

if (error > max\_error)

max\_error = error;

}

printf("Iteration %d:\n", count);

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

printf("x%d = %0.4f\t", i + 1, x[i]);

}

printf("\n");

count++;

} while (max\_error > allowedErr);

printf("Solution converged:\n");

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

printf("x%d = %0.4f\n", i + 1, x[i]);

}

return 0;

}

**OUTPUT:**

For the given system of equations:

10x1 + x2 + 2x3 = 44

2x1 + 10x2 + x3 = 51

x1 + 2x2 + 10x3 = 61

