1. Write a C program without using if-else construct that does the following.
It accepts a sequence of positive integers between 0 and 9 inclusive from the terminal.
The program will stop accepting input once an integer outside the range is entered. The program will finish by printing the number of 0, 1, 2,· · ·9 entered.

CODE:

```
#include <stdio.h>

/*-------------------------------------------------------------------------------------------------------------
 This program accepts sequence of positive integers between 0 and 9 and
 stops accepting input once an integer outside that range is entered.
 The program finishes by printing numbers entered (counts for each digit).

 Name: Abhishek Sonkar
 Reg No: 2025CA005
-------------------------------------------------------------------------------------------------------------*/

int main(void) {
   int counts[10] = {0};
   int num;

   printf("Enter digits between 0 and 9 (any other number to stop):\n");

   while (scanf("%d", &num) == 1)
   {
      if (num >= 0 && num <= 9)
         counts[num]++;

       else
         break;
   }

   printf("\n Digit counts: \n");
   for (int i = 0; i < 10; ++i)

      printf("%d: %d\n", i, counts[i]);


   return 0;
}
```

```
abhishek@laptop:~/Desktop/assignment 5 pps$ gcc question1.c
abhishek@laptop:~/Desktop/assignment 5 pps$ ./a.out
Enter digits between 0 and 9 (any other number to stop):
1
2
3
4
5
6
7
3
4
2
1
33

 Digit counts:
0: 0
1: 2
2: 2
3: 2
4: 2
5: 1
6: 1
7: 1
8: 0
9: 0
```

2. The equation (1 − x) cos x − sin x = 0 has a root between a = 0 and b = 1 since f (a)f (b) < 0. The bisection method of finding the root proceeds as follows:

a. It finds the midpoint r = (a + b)/2.

b. If f (r) = 0, then r is the root. If |b − a| is very small, then also we can take r as the root. In either of the cases, our job is done.

c. If f (r) 6= 0 and f (a)f (r) < 0, then the root lies between a and r. We assign r to b and go to step a.

d. If f (r) 6= 0 and f (b)f (r) < 0, then the root lies between r and b. We assign r to a and go to step a.

e. If the number of iterations is high, we may stop the process with appropriate message

CODE:

```
/*-------------------------------------------------------------------------------------------------------------
 This program finds a root of the nonlinear equation
     (1 - x) * cos(x) - sin(x) = 0
 using the bisection method. The bisection method requires an interval [a,b]
 where the function changes sign (f(a)*f(b) < 0). The algorithm repeatedly
 bisects the interval and selects the subinterval that contains the root until
 the interval width or function value is within a specified tolerance.

 Name : Abhishek Sonkar
 Reg No: 2025CA005
 ------------------------------------------------------------------------------------------------------------- */

#include <stdio.h>
#include <math.h>

#define TOLERANCE 1e-6
#define MAX_ITERATIONS 1000

double f(double x)
{

    /*
       Define the function whose root we want to find:
      f(x) = (1 - x)*cos(x) - sin(x)
      This is a small helper so the main code stays readable.
    */

    return (1 - x) * cos(x) - sin(x);
}

int main(void)
{
```

```
/*
    Initial search interval [a, b]. Change these if you want to search
   a different interval (but ensure f(a)*f(b) < 0).
*/

double a = 0.0, b = 1.0, r = 0.0;
int iteration = 0;

/*
    Sanity check: ensure the function has opposite signs at the endpoints.
   If not, the bisection method cannot be applied directly.
*/

if (f(a) * f(b) >= 0)
{
    printf("No root found in the interval [%.2f, %.2f]\n", a, b);
    return 1;
}

//Print a header for iteration output. Columns: iteration, a, b, r, f(r).
printf("Iter\ta\t\tb\t\tr\t\t f(r)\n");

/* Main bisection loop:
   - Stop when the interval width (b - a) is less than TOLERANCE, or
     when the absolute function value at the midpoint is small enough.
   - Also guard with MAX_ITERATIONS to avoid infinite loops.
*/

while ((b - a) >= TOLERANCE && iteration < MAX_ITERATIONS)
{
    /* Midpoint */
    r = (a + b) / 2.0;
    double fr = f(r);

    /* Print the current iteration's values for inspection */
    printf("%3d\t%.6f\t%.6f\t%.6f\t%.6f\n", iteration + 1, a, b, r, fr);

    /* If the function value at the midpoint is close enough to zero,
       we can stop early. */
    if (fabs(fr) < TOLERANCE)
    {
        break;
    }

    /* Decide which subinterval contains the root by checking the sign
       of f(a)*f(r). If it's negative, the root lies in [a, r], so set
       b = r. Otherwise it lies in [r, b], so set a = r. */
    if (f(a) * fr < 0)
    {
        b = r; /* root in left subinterval */
    } else
        {
```

```
        a = r; /* root in right subinterval */
        }

    iteration++;
  }

  /*
  Report result: either the root (within tolerance) or that the maximum
    number of iterations was reached.
  */

  if (iteration == MAX_ITERATIONS) {
    printf("Maximum iterations reached. Approximate root: %.6f\n", r);
  } else {
    printf("\nRoot found at x = %.6f after %d iterations\n", r, iteration);
  }
  return 0;
}
```

```
collect2: error: ld returned 1 exit status
abhishek@laptop:~/Desktop/assignment 5 pps$ gcc question2.c -lm
abhishek@laptop:~/Desktop/assignment 5 pps$ ./a.out
 Iter    a               b               r               f(r)
    1    0.000000        1.000000        0.500000        -0.040634
    2    0.000000        0.500000        0.250000        0.479280
    3    0.250000        0.500000        0.375000        0.215295
    4    0.375000        0.500000        0.437500        0.085844
    5    0.437500        0.500000        0.468750        0.022175
    6    0.468750        0.500000        0.484375        -0.009345
    7    0.468750        0.484375        0.476562        0.006387
    8    0.476562        0.484375        0.480469        -0.001486
    9    0.476562        0.480469        0.478516        0.002449
   10    0.478516        0.480469        0.479492        0.000481
   11    0.479492        0.480469        0.479980        -0.000502
   12    0.479492        0.479980        0.479736        -0.000011
   13    0.479492        0.479736        0.479614        0.000235
   14    0.479614        0.479736        0.479675        0.000112
   15    0.479675        0.479736        0.479706        0.000051
   16    0.479706        0.479736        0.479721        0.000020
   17    0.479721        0.479736        0.479729        0.000005
   18    0.479729        0.479736        0.479733        -0.000003
   19    0.479729        0.479733        0.479731        0.000001

 Root found at x = 0.479731 after 18 iterations
```

3. The error function, $\mathrm{erf}(x)$, is defined using the following series:

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \sum_{n=0}^{\infty} \frac{(-1)^n x^{2n+1}}{n!(2n+1)}.$$

Write a C program that accepts a value of x and return the value of $\mathrm{erf}(x)$. The program takes the sum of the terms in the series as long as their *magnitude* are greater then a predefined tolerance level `eps`.

CODE:

```
/*-------------------------------------------------------------------------------------------------------
This program that accepts a value of x and return the value of erf(x). The
program takes the sum of the terms in the series as long as their magnitude
are greater then a predefined tolerance level eps.
Name: Abhishek Sonkar
Reg No: 2025CA005
-------------------------------------------------------------------------------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <float.h>

#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif

// Usage: question3 [x [tol]]
// If x is not provided on the command line, the program reads a single number from stdin.
int main(int argc, char *argv[]) {
    double x_d = 0.0;
    double tol_d = 1e-14; // default tolerance (absolute)

    if (argc >= 2) {
        char *end;
        x_d = strtod(argv[1], &end);
        if (end == argv[1] || *end != '\0') {
            fprintf(stderr, "Invalid numeric argument for x: '%s'\n", argv[1]);
            return 1;
        }
    } else {
        // read from stdin (preserves original behavior)
        if (scanf("%lf", &x_d) != 1) {
            fprintf(stderr, "Please provide a numeric input (e.g. 0.5)\n");
            return 1;
        }
    }

    if (argc >= 3) {
        char *end;
        tol_d = strtod(argv[2], &end);
```

```
        if (end == argv[2] || *end != '\0' || tol_d <= 0.0) {
            fprintf(stderr, "Invalid tolerance argument: '%s' (must be positive)\n", argv[2]);
            return 1;
        }
    }

    if (!isfinite(x_d)) {
        fprintf(stderr, "Input is not finite\n");
        return 1;
    }

    // Work in long double for better accumulation of the series
    long double x = (long double)x_d;
    long double tol = (long double)tol_d;
    const long double two_over_sqrt_pi = 2.0L / sqrt((long double)M_PI);

    long double term = x;      // a_0 = x
    long double sum = term;

    const int max_iter = 1000000; // safety guard
    int n = 0;
    int iterations_used = 1;

    while (n < max_iter) {
        long double mult = - (x * x) * (2.0L * n + 1.0L) / ((n + 1.0L) * (2.0L * n + 3.0L));
        term *= mult;
        sum += term;
        n++;
        iterations_used = n + 1; // number of terms summed
        if (fabsl(term) < tol) break;
    }

    long double erf_series = two_over_sqrt_pi * sum;

    double erf_series_d = (double)erf_series; // for comparison with library double erf()
    double erf_lib = erf((double)x);

    printf("erf_series(%g) = %.15g\n", (double)x, erf_series_d);
    printf("erf(lib)      = %.15g\n", erf_lib);
    printf("abs error     = %.15g\n", fabs(erf_lib - erf_series_d));
    printf("iterations used = %d\n", iterations_used);
    if (n >= max_iter) {
        printf("warning: reached max_iter=%d before meeting tolerance\n", max_iter);
    }

    return 0;
}
```

```
abhishek@laptop:~/Desktop/assignment 5 pps$ gcc question3.c -lm
abhishek@laptop:~/Desktop/assignment 5 pps$ ./a.out
5
erf_series(5) = 1.00000000001823
erf(lib)      = 0.999999999998463
abs error     = 1.97670768642411e-11
iterations used = 92
```