**DATE:-**

**ASSIGNMENT NUMBER:-**

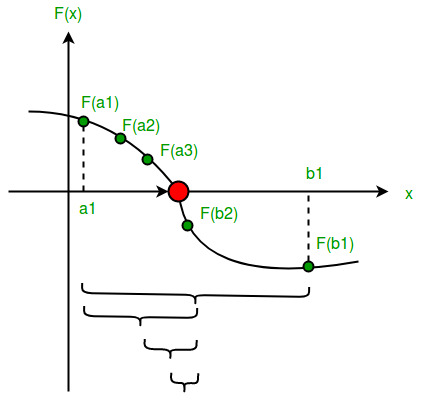
**PROBLEM STATEMENT:-**

**THEORY:-**

The bisection method in mathematics is a root-finding method that repeatedly [bisects](https://en.wikipedia.org/wiki/Bisection) an [interval](https://en.wikipedia.org/wiki/Interval_(mathematics)) and then selects a subinterval in which a [root](https://en.wikipedia.org/wiki/Root_of_a_function) must lie for further processing. It is a very simple and robust method, but it is also relatively slow. Because of this, it is often used to obtain a rough approximation to a solution which is then used as a starting point for more rapidly converging methods.

The method is applicable for numerically solving the equation *f*(*x*) = 0 for the [real](https://en.wikipedia.org/wiki/Real_number) variable *x*, where *f* is a continuous function defined on an interval [*a*, *b*] and where *f*(*a*) and *f*(*b*) have opposite signs. In this case *a* and *b* are said to bracket a root since, by the intermediate value theorem, the continuous function *f* must have at least one root in the interval (*a*, *b*).

Since root may be a floating point number, we repeat above steps while difference between a and b is less than a value ? (A very small value).

[](https://cdncontribute.geeksforgeeks.org/wp-content/uploads/bisection.jpg)

At each step the method divides the interval in two by computing the midpoint *c* = (*a*+*b*) / 2 of the interval and the value of the function *f*(*c*) at that point. Unless *c* is itself a root (which is very unlikely, but possible) there are now only two possibilities: either *f*(*a*) and *f*(*c*) have opposite signs and bracket a root, or *f*(*c*) and *f*(*b*) have opposite signs and bracket a root. The method selects the subinterval that is guaranteed to be a bracket as the new interval to be used in the next step. In this way an interval that contains a zero of *f* is reduced in width by 50% at each step. The process is continued until the interval is sufficiently small.

Explicitly, if *f*(*a*) and *f*(*c*) have opposite signs, then the method sets *c* as the new value for *b*, and if *f*(*b*) and *f*(*c*) have opposite signs then the method sets *c* as the new *a*. (If *f*(*c*)=0 then *c* may be taken as the solution and the process stops.) In both cases, the new *f*(*a*) and *f*(*b*) have opposite signs, so the method is applicable to this smaller interval.

Iteration tasks

The input for the method is a continuous function *f*, an interval [*a*, *b*], and the function values *f*(*a*) and *f*(*b*). The function values are of opposite sign (there is at least one zero crossing within the interval). Each iteration performs these steps:

1. Calculate *c*, the midpoint of the interval, *c* = *a* + *b*/2.
2. Calculate the function value at the midpoint, *f*(*c*).
3. If convergence is satisfactory (that is, *c* - *a* is sufficiently small, or |*f*(*c*)| is sufficiently small), return *c* and stop iterating.
4. Examine the sign of *f*(*c*) and replace either (*a*, *f*(*a*)) or (*b*, *f*(*b*)) with (*c*, *f*(*c*)) so that there is a zero crossing within the new interval.

**ALGORITHM:-**

**STEPS:-**

1. Decide initial values for x1 and x2 and stopping criterion, E.
2. Compute **f1 = f(x1)** and **f2 = f(x2)**.
3. If **f1 \* f2>0**, x1 and x2 do not bracket any root and go to step 7;  
   Otherwise continue.
4. Compute **x0 = (x1+x2)/2** and compute **f0 = f(x0)**
5. If **f1\*f0 < 0** then  
   **set x2 = x0**  
   else  
   **set x1 = x0**  
   **set f1 = f0**
6. If absolute value of **(x2 – x1)/x2** is less than error E, then  
   **root = (x1 + x2)/2**  
   write the value of root  
   go to step 7  
   else  
   go to step 4
7. Stop.

**SOURCE CODE:-**

#include<stdio.h>

#include<math.h>

float fun (float x)

{

    return (x\*x\*x - 4\*x - 9);

}

void bisection (float \*x, float a, float b, int \*itr)

/\* this function performs and prints the result of one iteration \*/

{

    \*x=(a+b)/2;

    ++(\*itr);

    printf("Iteration no. %3d X = %7.5f\n", \*itr, \*x);

}

int main ()

{

    int itr = 0, maxmitr;

    float x, a, b, allerr, x1;

    printf("\nEnter the values of a, b, allowed error and maximum iterations:\n");

    scanf("%f %f %f %d", &a, &b, &allerr, &maxmitr);

    bisection (&x, a, b, &itr);

    do

    {

if (fun(a)\*fun(x) < 0)

 b=x;

else

 a=x;

bisection (&x1, a, b, &itr);

if (fabs(x1-x) < allerr)

{

 printf("After %d iterations, root = %6.4f\n", itr, x1);

 return 0;

}

x=x1;

    }while (itr < maxmitr);

    printf("The solution does not converge or iterations are not sufficient");

    return 0;

**INPUT & OUTPUT:-**

Equation : x^3 - 3\*x^2 + 3\*x - 1

Enter initial approximation of the root : 1 0

Iteration a f(a) b f(b) c f(c)

========= ========= ========= ========= ========= ========= =========

1 1.000000 0.000000 0.000000 -1.000000 0.500000 -0.125000

2 1.000000 0.000000 \*0.500000 -0.125000 0.750000 -0.015625

3 1.000000 0.000000 \*0.750000 -0.015625 0.875000 -0.001953

4 1.000000 0.000000 \*0.875000 -0.001953 0.937500 -0.000244

5 1.000000 0.000000 \*0.937500 -0.000244 0.968750 -0.000031

6 1.000000 0.000000 \*0.968750 -0.000031 0.984375 -0.000004

7 1.000000 0.000000 \*0.984375 -0.000004 0.992188 -0.000000

8 1.000000 0.000000 \*0.992188 -0.000000 0.996094 -0.000000

9 1.000000 0.000000 \*0.996094 -0.000000 0.998047 -0.000000

10 1.000000 0.000000 \*0.998047 -0.000000 0.999023 -0.000000

11 1.000000 0.000000 \*0.999023 -0.000000 0.999512 -0.000000

12 1.000000 0.000000 \*0.999512 -0.000000 0.999756 -0.000000

13 1.000000 0.000000 \*0.999756 -0.000000 0.999878 -0.000000

14 1.000000 0.000000 \*0.999878 -0.000000 0.999939 -0.000000

15 1.000000 0.000000 \*0.999939 -0.000000 0.999969 -0.000000

16 1.000000 0.000000 \*0.999969 -0.000000 0.999985 -0.000000

17 1.000000 0.000000 \*0.999985 -0.000000 0.999992 -0.000000

18 1.000000 0.000000 \*0.999992 -0.000000 0.999996 0.000000

19 1.000000 0.000000 \*0.999996 0.000000 0.999998 0.000000

20 1.000000 0.000000 \*0.999998 0.000000 0.999999 0.000000

21 1.000000 0.000000 \*0.999999 0.000000 1.000000 0.000000

22 1.000000 0.000000 \*1.000000 0.000000 1.000000 0.000000

23 1.000000 0.000000 \*1.000000 0.000000 1.000000 0.000000

24 1.000000 0.000000 \*1.000000 0.000000 1.000000 0.000000

Root found : 1.000000

**DISCUSSION:-**

1. Bisection method is the safest and it always converges. The bisection method is the simplest of all other methods and is guaranteed to converge for a continuous function.
2. It is always possible to find the number of steps required for a given accuracy and the new methods can also be developed from bisection method and bisection method plays a very crucial role in computer science research.

-------------------------