Programming and Data Structure

Sudeshna Sarkar

Dept. of Computer Science & Engineering.

Indian Institute of Technology

Kharagpur

19 Jan 2012

The break Statement

- Break out of the loop { }
  - can use with
    - while
    - do while
    - for
    - switch
  - does not work with
    - if
    - if else
- Causes immediate exit from a *while*, *do/while*, *for* or *switch* structure.
- Program execution continues with the first statement after the structure.

Common uses of the break statement

Escape early from a loop

Skip the remainder of a switch structure

An Example

```
#include  <stdio.h>
int main( ) {
    int  fact, i;
    fact = 1;  i = 1;
    while  ( i<10 )    {            /* run loop –break when fact >100*/
        fact = fact * i;
        if ( fact > 100 )  {
            printf ("Factorial of %d  above 100", i);
            break;              /* break out of the while loop */
        }
        i ++ ;
    }
    return 0;
}
```

The continue Statement

- Skips the remaining statements in the body of a *while*, *for* or *do/while* structure.
  - Proceeds with the next iteration of the loop.
- while and do/while
  - Loop-continuation test is evaluated immediately after the continue statement is executed.
- for structure
  - *update* is evaluated, then *expression2(condition)* is evaluated.

An Example with “break” & “continue”

```
fact = 1; i = 1;          /* a program segment  to calculate 10 !
while  (1)  {
    fact = fact * i;
    i ++ ;
    if  ( i<10 )
            continue;     /* not done yet ! Go to loop and
                          perform next iteration*/
    break;
}
```

Avoid using break or continue

- Some people consider the use of break or continue is poor program design
- Try to avoid using them.

Avoid 'break' in loops

```
// A bad loop style
for ( ; ; )
    {
     …
     if (condition)
         break;
    } // for
```

```
// A better loop style
for ( ; !condition ; )
    {
     …
    } // for
```

```
while (x)
    {
     …
     if (condition)
         break;
     else
         …
    } // while
```

```
while  (x && !condition)
    {
     …
     if (!condition)
         …;
    } // while
```

Avoid 'continue' in loops

```c
float readAverage (void)
{
// Local Declarations
int    count = 0;

int    n;
float sum = 0;

// Statements
while(scanf("%d",&n)
        != EOF)
    {
     if (n == 0)
         continue;
     sum += n;
     count++;
    }  // while

return (sum / count);
} // readAverage
```

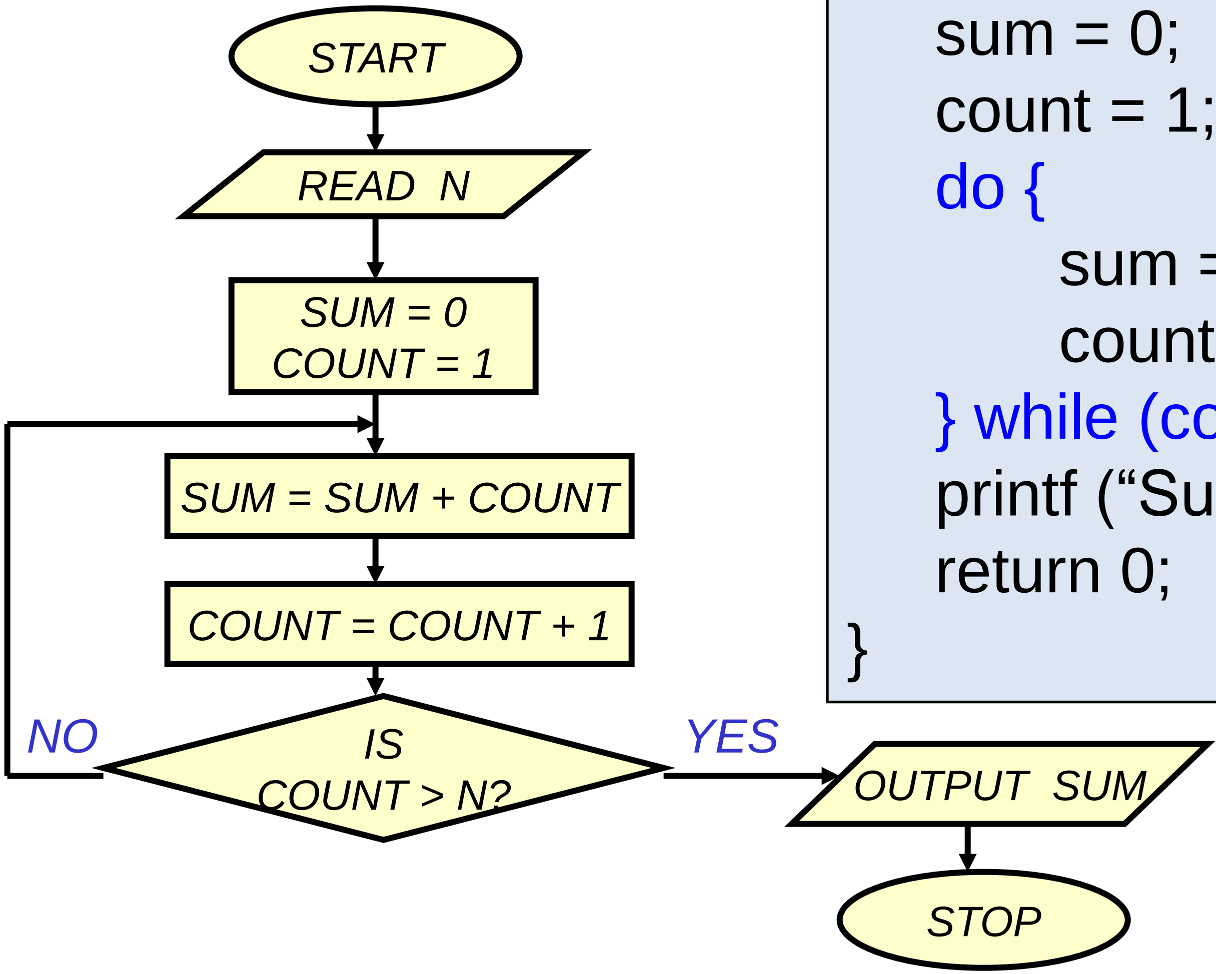
```c
float readAverage (void)
{
// Local Declarations
int    count = 0;

int    n;
float sum = 0;

// Statements
while(scanf("%d",&n)
        != EOF)
    {
     if (n != 0)
         {
          sum += n;
          count++;
         } // if
    } // while
return (sum / count);
} // readAverage
```

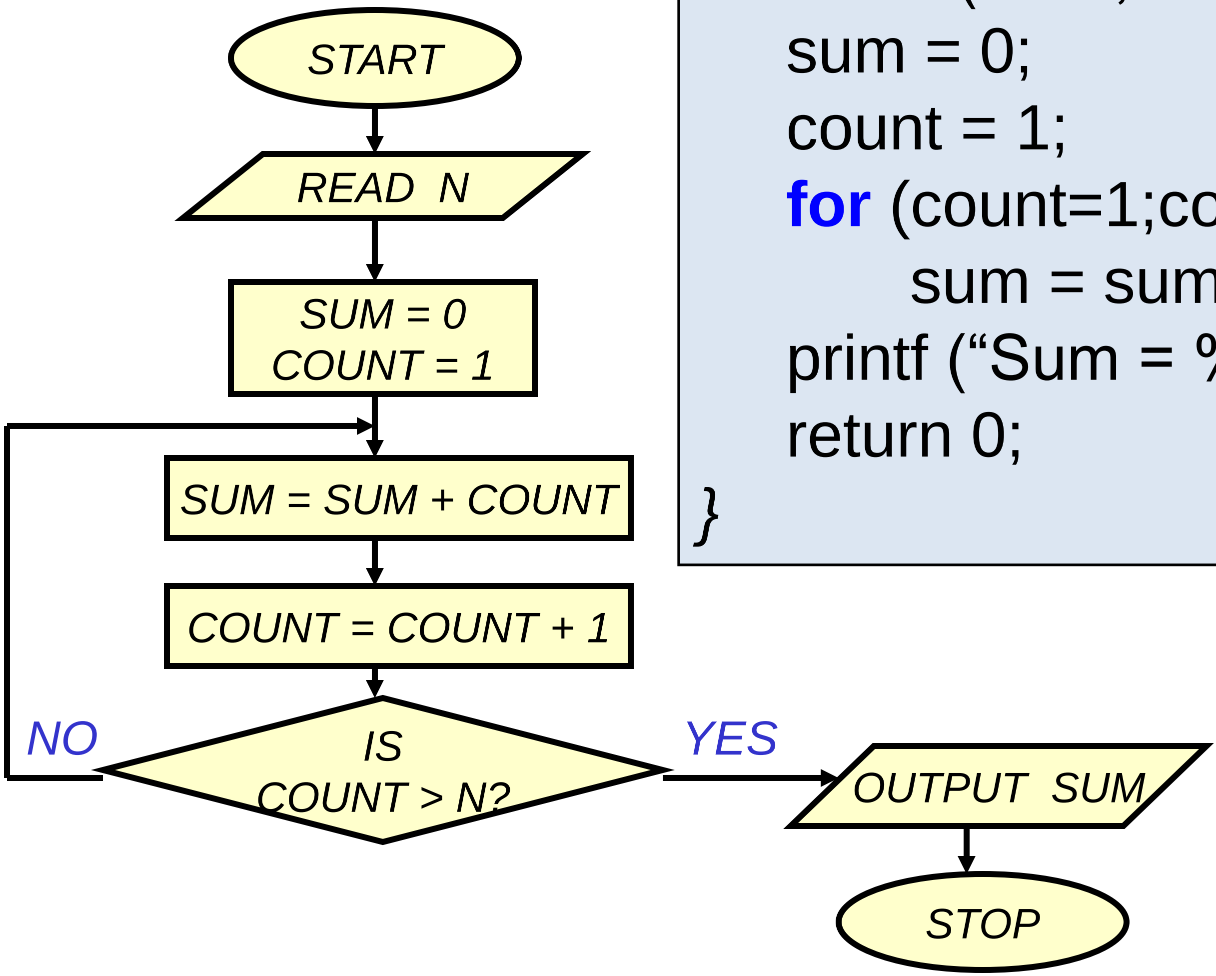
Programming Examples

1. Sum of first N natural numbers




```
int main () {
    int N, count, sum;
    scanf (“%d”, &N) ;
    sum = 0;
    count = 1;
    do {
        sum = sum + count;
        count = count + 1;
    } while (count<=N) ;
    printf (“Sum = %d\n”, sum) ;
    return 0;
}
```

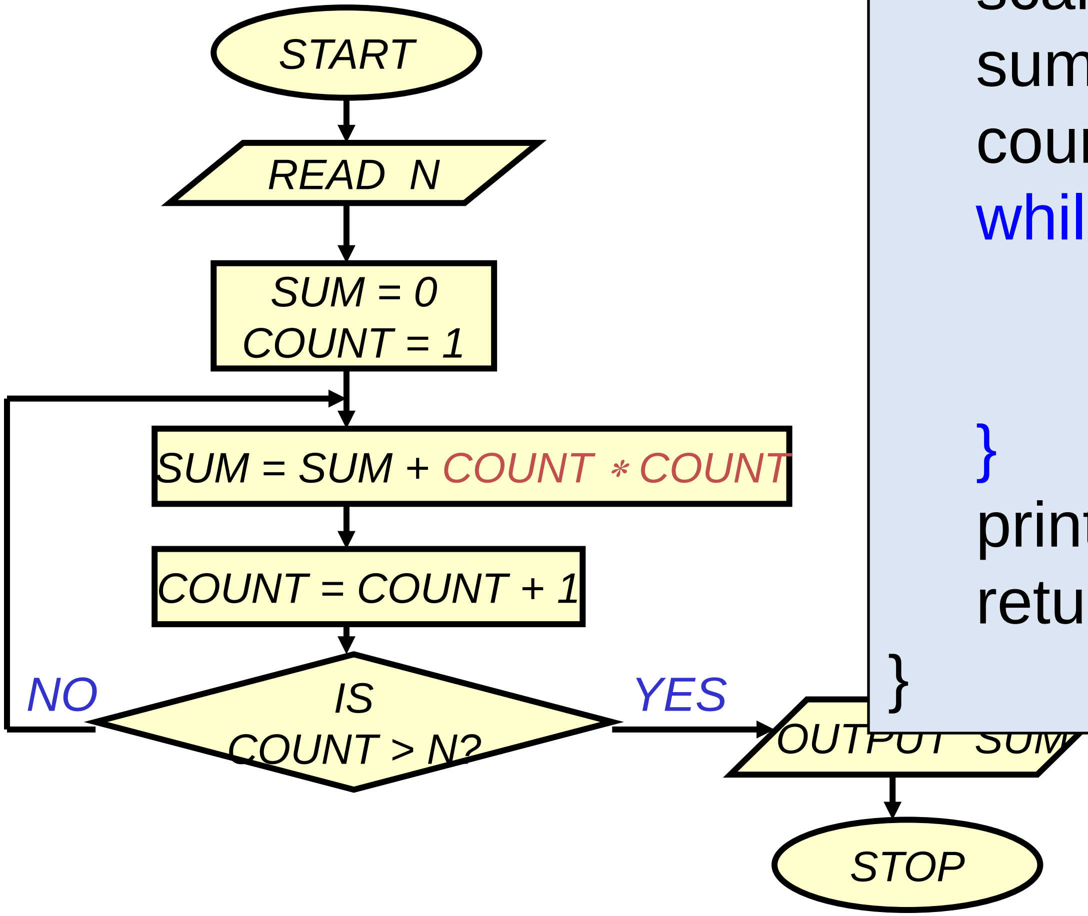
Sum of first N natural numbers




```
int main () {
    int N, count, sum;
    scanf (“%d”, &N) ;
    sum = 0;
    count = 1;
    for (count=1;count <= N;count++)
        sum = sum + count;
    printf (“Sum = %d\n”, sum) ;
    return 0;
}
```

Example 2:

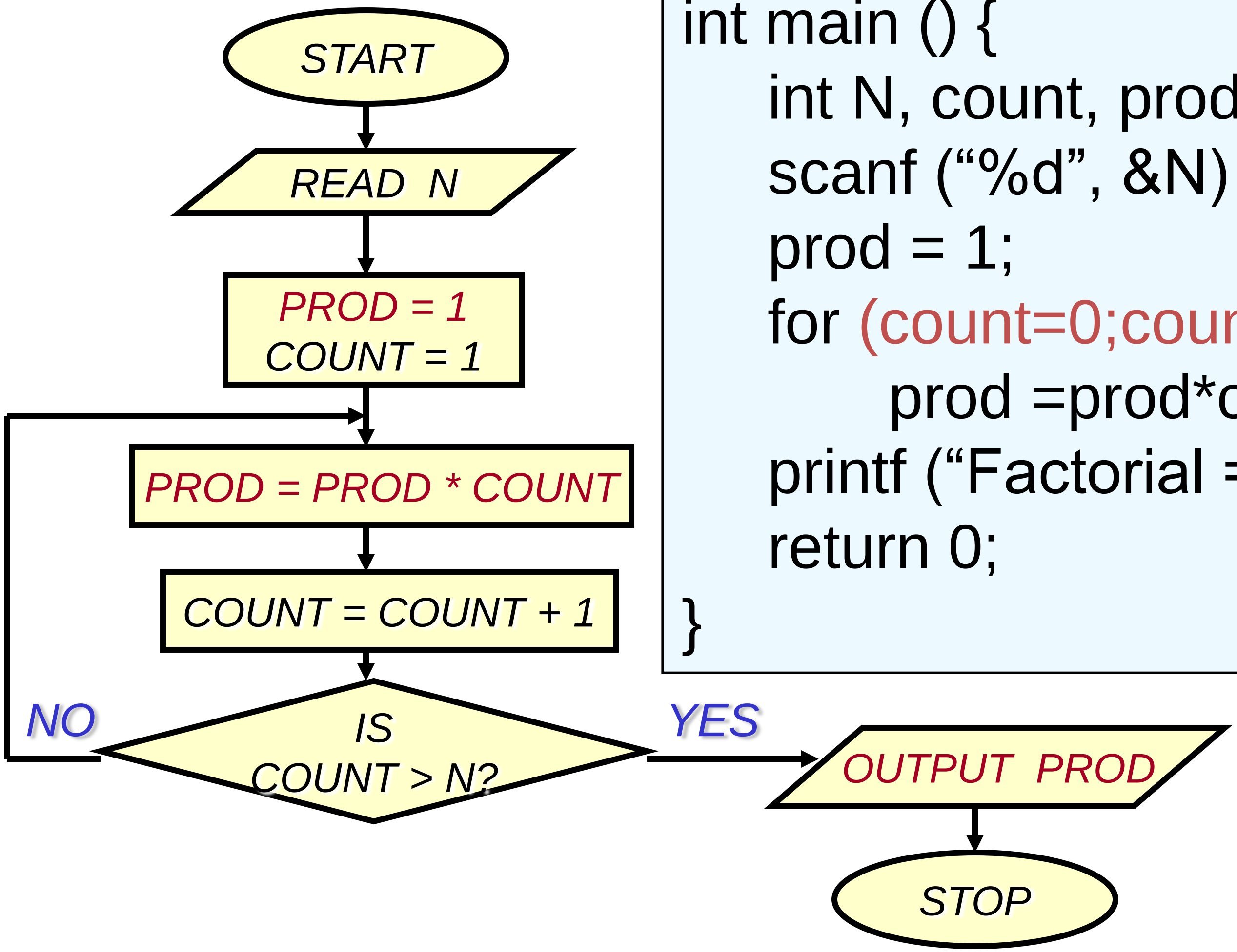
$SUM = 1^2 + 2^2 + 3^2 + N^2$




```
int main () {
    int N, count, sum;
    scanf ("%d", &N) ;
    sum = 0;
    count = 1;
    while (count <= N)  {
        sum = sum + count∗count;
        count = count + 1;
    }
    printf ("Sum = %d\n", sum) ;
    return 0;
}
```

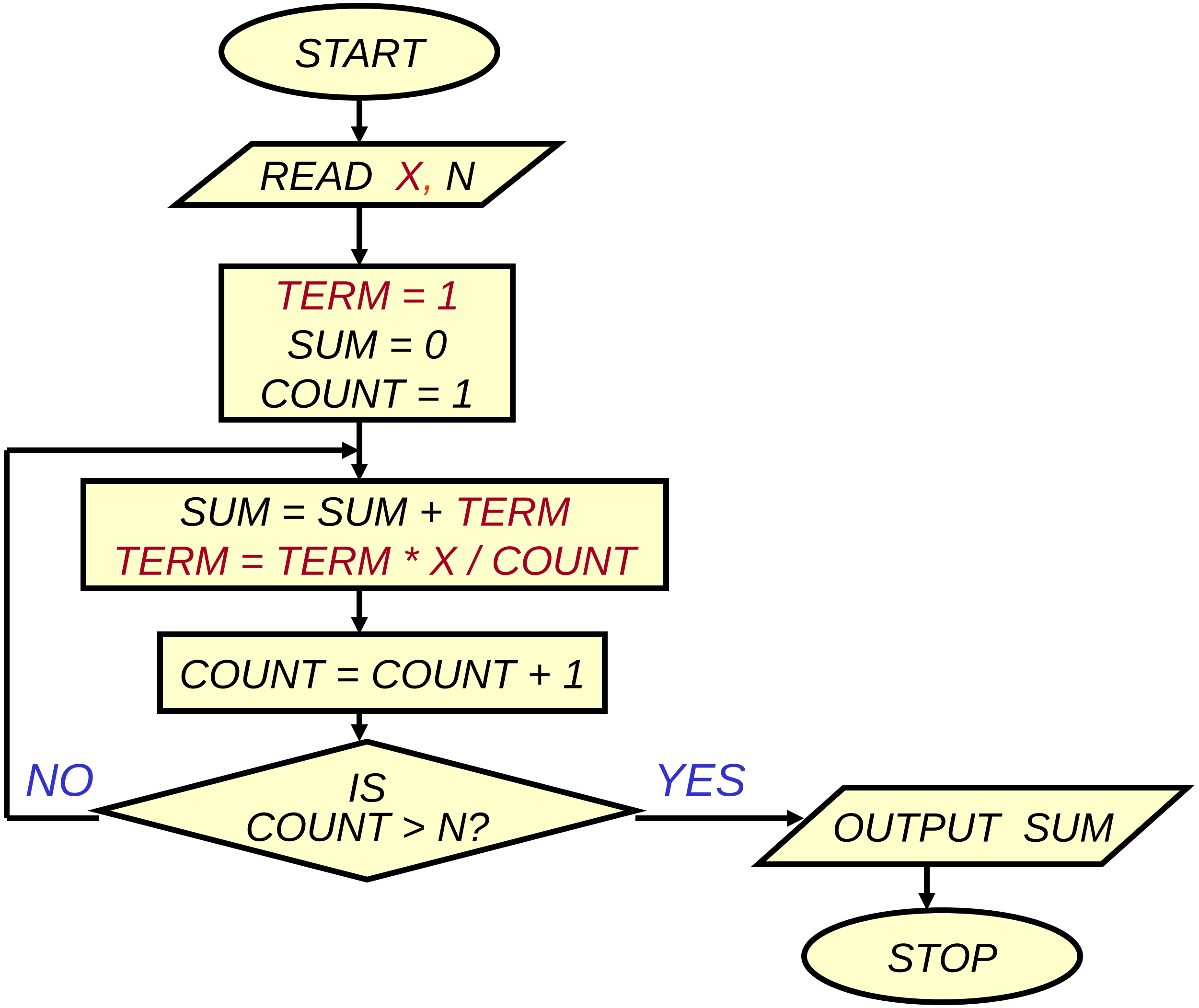
Example 3:

Computing Factorial




```
int main () {
    int N, count, prod;
    scanf (“%d”, &N) ;
    prod = 1;
    for (count=0;count < N; count++)  {
        prod =prod*count;
    printf (“Factorial = %d\n”, prod) ;
    return 0;
}
```

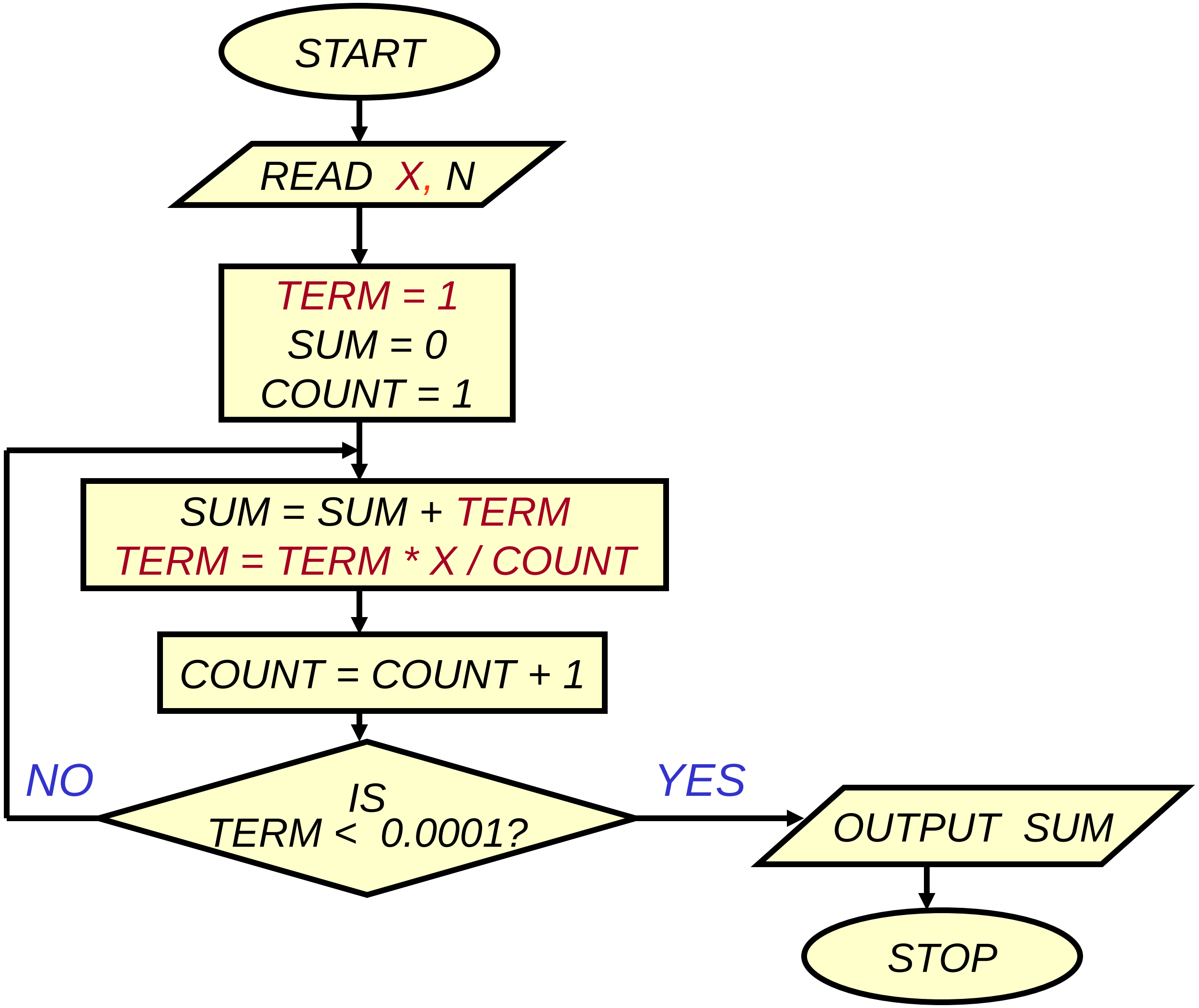
Example 4: Computing e^x series up to N terms



```
int main ( ) {
    float x, term, sum;
    int n, count;

    scanf ("%d", &x) ;
    scanf ("%d", &n) ;
    term = 1.0; sum = 0;
    for (count = 0; count < n; count++) {
        sum += term;
        term = term * x/count;
    }
    printf ("%f\n", sum) ;
    return 0;
}
```

Example 5: Computing e^x series up to 4 decimal places



```
int main () {
    float x, term, sum;
    int n, count;

    scanf ("%d", &x) ;
    scanf ("%d", &n) ;
    term = 1.0; sum = 0;
    for (count = 0; term<0.0001; count++)  {
        sum += term;
        term *= x/count;
    }
    printf ("%f\n", sum) ;
    return 0;
}
```

Example 6: Test if a number is prime or not

```
#include <stdio.h>
int main( ) {
    int  n;
    scanf ("%d", &n);



}
```

Example 6: Test if a number is prime or not

```
int main( ) {
    int  n;
    scanf (“%d”, &n);
    i = 2;
    while (i < n)  {
          if (n % i == 0)  {
                printf (“%d is not a prime \n”, n);
          }
          i++;
    }
    printf (“%d is a prime \n”, n);
    return 1;
}
```

Example 6: Test if a number is prime or not

```
int main( ) {
    int  n, prime = 1;
    scanf ("%d", &n);
    i = 2;
    while (i < n)  {
         if (n % i == 0)  {
                  printf ("%d is not a prime \n", n);
                  prime = 0;
         }
         i++;
    }
     if (prime == 1)
         printf ("%d is a prime \n", n);
    return 0;
}
```

```
int main( ) {
    int  n, prime = 1;
    scanf (“%d”, &n);
    i = 2;
    while (i < n)  {
          if (n % i == 0)  {
                    prime = 0;
                    break;
          }
          i++;
    }
     if (prime == 1)
          printf (“%d is a prime \n”, n);
     else printf (“%d is not a prime \n”, n);
    return 0;
}
```

```
int main( ) {
    int  n, prime = 1;
    scanf (“%d”, &n);
    i = 2;
    while (i < n)  {
            if (n % i == 0)  {
                        printf (“%d is not a prime \n”, n);
                        return 0;
            }
            i++;
    }
     if (prime == 1)
            printf (“%d is a prime \n”, n);
    return 1;
}
```

```
int main( ) {
    int  n, prime = 1;
    scanf (“%d”, &n);
    i = 2;
    while ((i < n) && (prime ==1))  {
            if (n % i == 0)  {
                        prime = 0;
            }
            i++;
    }
     if (prime == 1)
            printf (“%d is a prime \n”, n);
     else printf (“%d is not a prime \n”, n);
    return 0;
}
```

More efficient – less number of iterations

```
int main( ) {
    int  n, i=2;
    scanf (“%d”, &n);
    while (i < sqrt(n))  {
          if (n % i == 0)  {
                printf (“%d is not a prime \n”, n);
                exit;
          }
          i = i + 1;
    }
    printf (“%d is a prime \n”, n);
    return 0;
}
```

Example 7: Find the sum of digits of a number

Example 7: Find the sum of digits of a number

```
#include  <stdio.h>
int main( )  {
    int   n, sum=0;
    scanf (“%d”, &n);
    while (n != 0)  {
          sum = sum + (n % 10);
          n = n / 10;
    }
    printf (“The sum of digits of the number is %d \n”, sum);
    return 0;
}
```

Example 8: Approximating the logarithm

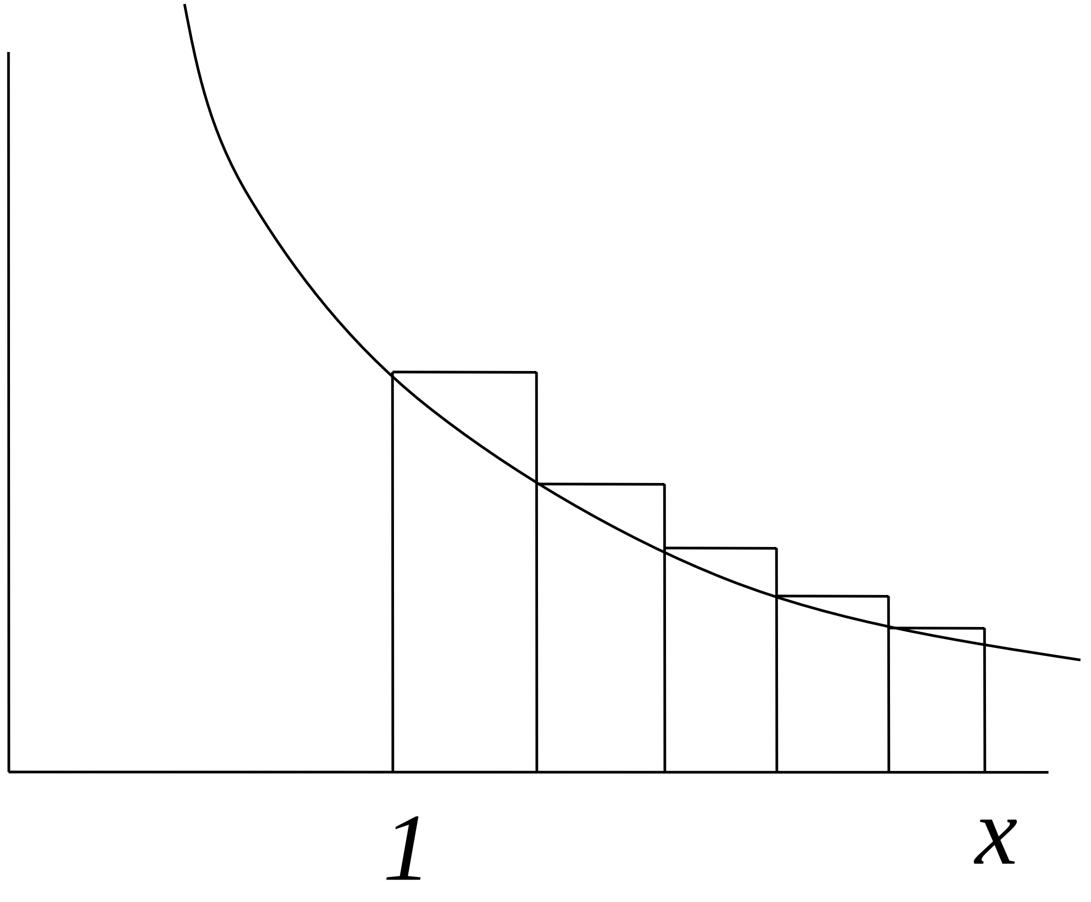
- How many times must we divide a number x by 10 until the result goes below 1?

```
float x;
scanf (“%f”, &x);
int numDivs = 0;
while (x > 1) {
  x = x / 10;
  numDivs = numDivs + 1;
}
printf(“%d\n”,numDivs);
```

Example 9: Computing ln x

- Must use arithmetic operations.
- Estimate the area under $f(x) = 1/x$ from 1 to x.
- Area approximated by small rectangles.

Riemann Integral



How many rectangles?

- More the better! Say 1000.
- Total width of rectangles = x - 1.
- Width w of each = (x - 1)/1000
- x coordinate of left side of ith rectangle

  1 + (i-1)w.
- Height of ith rectangle = 1/(1+(i-1)w)

Program to compute ln

```
#define INTERVALS 1000
int main(  ){
    float x, area=0, w;
    int i;
    scanf (“%f”, &x) ;
    w = (x-1)/INTERVAL;
    for(i=1 ; i <= INTERVAL ; i=i+1){
        area = area + w*(1/(1+(i-1)*w);
    }
    printf (“ln %f = %f\n”, x,  area) ;
    return 0;
}
```

Program to compute ln

```
#define INTERVALS 1000
int main(  ){
    float x, area=0, w;
    int i;
    scanf (“%f”, &x) ;
    w = (x-1)/INTERVAL;
    for(i=1 ; i <= INTERVAL ; i=i++){
        area   *= w/(1 + i*w) ;
    }
    printf (“ln %f = %f\n”, x,  area) ;
    return 0;
}
```

Example 10: Decimal to binary conversion

```
int  dec;
scanf (“%d”, &dec);
do
{
      printf (“%2d”,  (dec % 2));
      dec = dec / 2;
}  while (dec != 0);
printf (“\n”);
```

Example 11:
Compute greatest common divisor (GCD) of two numbers

The standard gcd algorithm is based on successive Euclidean division.

Let us try to render it as a sequence of repetitive computations.

For the sake of simplicity, we assume that whenever we write gcd(a,b) we mean a>=b.

[*Euclidean gcd theorem*]

- Let a, b be positive integers and r = a % b. Then gcd(a,b) = gcd(b,r).
- If a is an integral multiple of b, we have r=0, and so by the theorem gcd(a,b)=gcd(b,0)=b.

12) 45 (3

36

9) 12 (1

9

3) 9 (3

9

0

GCD algorithm

As long as b is not equal to 0 do the following:

Compute the remainder r = a rem b.

Replace a by b and b by r.

Report a as the desired gcd.

GCD algorithm

As long as b is not equal to 0 do the following:
    Compute the remainder r = a rem b.
    Replace a by b and b by r.
Report a as the desired gcd.

```
if  (a > b)  {
        temp = a;  a = b;  b = temp;
}
while (b != 0)  {
        rem = a % b;
        a = b;
        b = rem;
}
```

Example 11: Compute GCD of two numbers

```
int main( ) {
    int  a, b, rem, temp;
    scanf (%d %d”, &a, &b);

    if  (a > b)  {
          temp = a;  a = b;  b = temp;
    }
    while (b != 0)  {
          rem = a % b;
          a = b;
          b = rem;
    }
    printf (“The GCD is %d”, a);
     return 0;
}
```

12) 45 (3

36

9) 12 (1

9

3) 9 (3

9

0

Initial: A=12, B=45

Iteration 1: temp=9, B=12,A=9

Iteration 2: temp=3, B=9, A=3

B % A = 0 → GCD is 3

Exercise 1

sin() takes a value in radians and returns the sin of it. Use the sin function to plot a sin wave vertically using stars (it should look something like this):

```
        *
            *
               *
                *
               *
            *
        *
     *
 *
*
 *
    *
        *
```

Hint: Obviously, sin returns a number between -1 and 1. Convert this to a number between 0 and 60 and print that many spaces before printing the * then print a '\n'

```
[sudeshna@facweb temp]$ ./a.out
                              *
                                    *
                                          *
                                               *
                                                    *
                                                        *
                                                          *
                                                            *
                                                            *
                                                            *
                                                          *
                                                       *
                                                   *
                                              *
                                         *
                                   *
                            *
                      *
                *
           *
       *
   *
 *
*
*
 *
   *
      *
           *
                *
                     *
                           *
[sudeshna@facweb temp]$
```

Exercise 2

Write a C program to compute the following series:

$x - x^2/(2*1) + 2*x^3/(3*2*1) - 3*x^4/(4*3*2*1) + ...$

The value of x will be read from the user. The sum is to be computed over 10 terms. Print the partial sums as well as the final sum.

Exercise 3

It is known that the harmonic number H_n converges to **k + ln n** as n tends to infinity.

Here ln is the natural logarithm and k is a constant known as *Euler's constant*. In this exercise you are asked to compute an approximate value for Euler's constant.

Generate the values of H_n and *ln n* successively for n=1,2,3,..., and compute the difference $k_n = H_n - \ln n$. Stop when $k_n - k_{n-1}$ is less than a specific error bound (say 10^{-8}).

Exercise 4

Write a C program that takes as input a number and computes and prints the following:

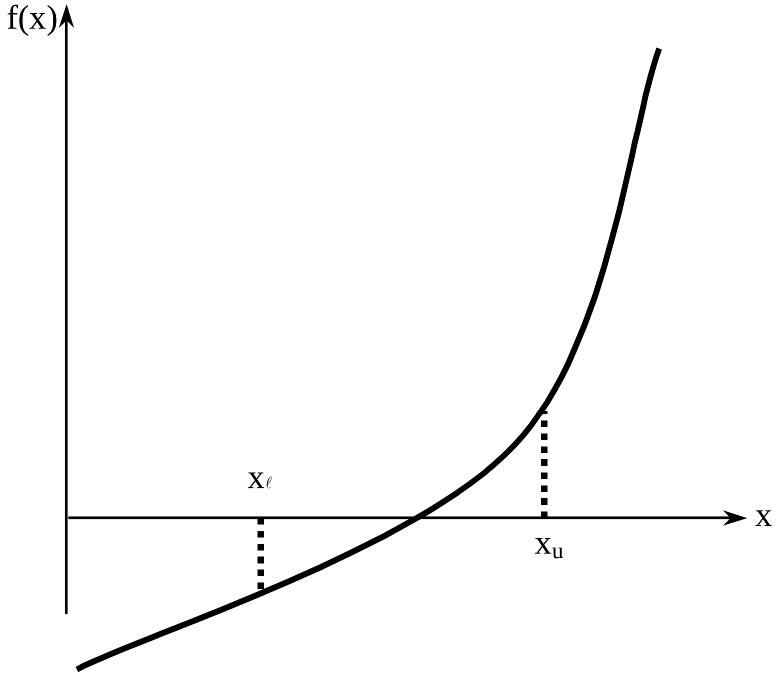
1. the sum of the digits of the number
2. the number reversed
3. the sum of the original number and the reversed number

Exercise 5

Write a program that find can find the roots of a mathematical function using the bisection method. Assume that the function has exactly one root in that interval.

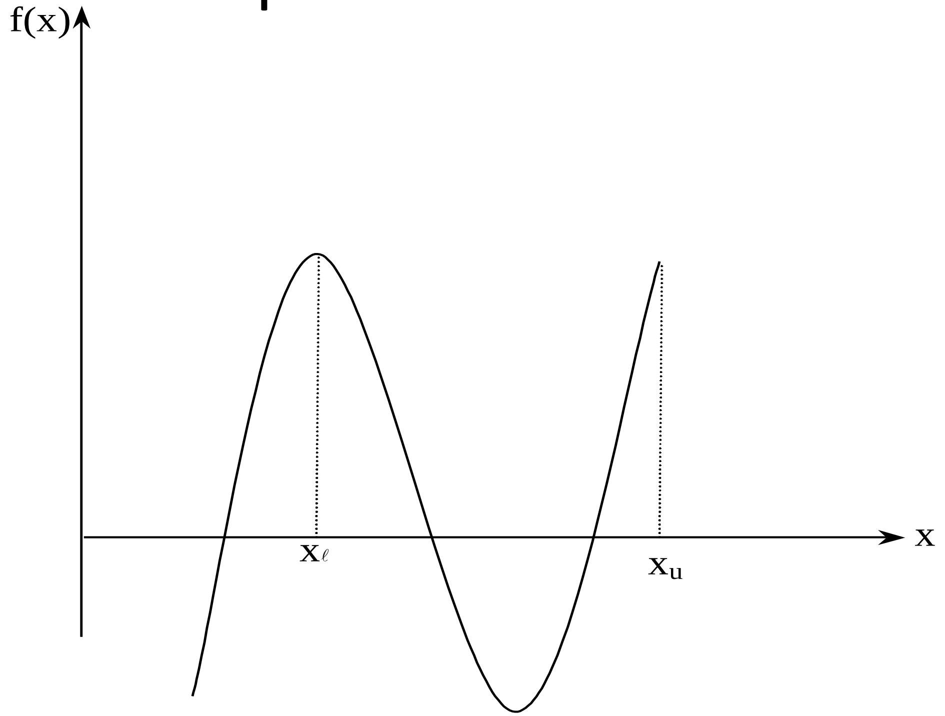
Basis of Bisection Method - 1

Theorem An equation f(x)=0, where f(x) is a real, continuous function, has at least one root between x_l and x_u if $f(x_l)\, f(x_u) < 0$.



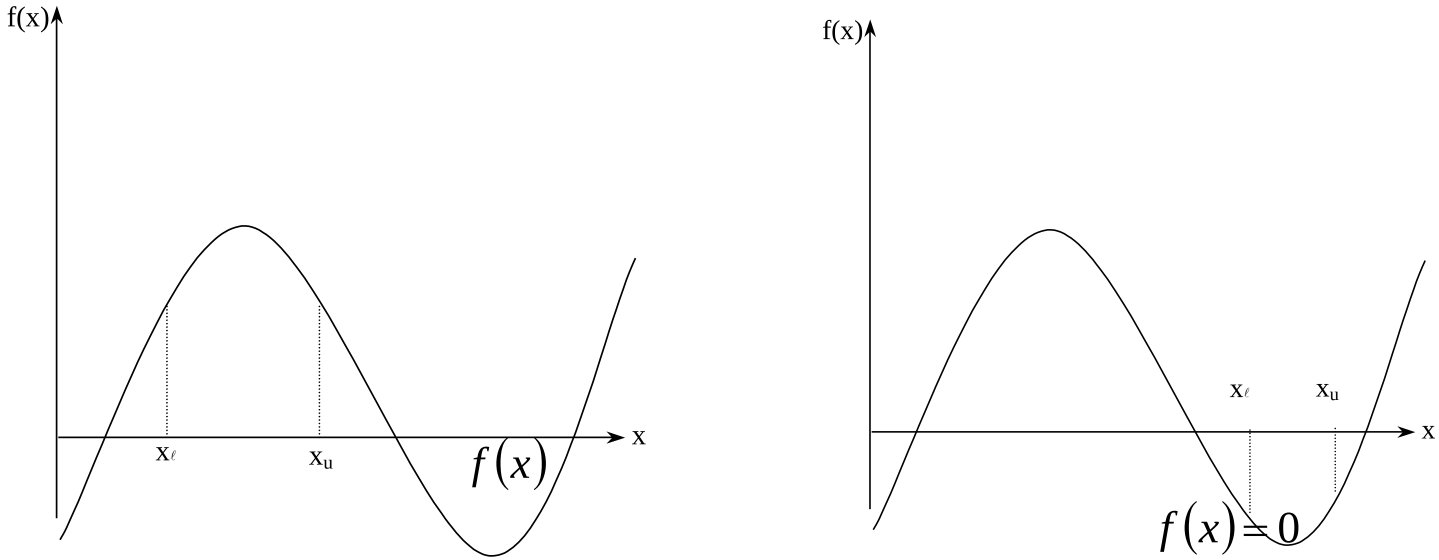
Basis of Bisection Method - 2

If function f(x) does not change sign between two points, roots of the equation f(x)=0 may still exist between the two points.



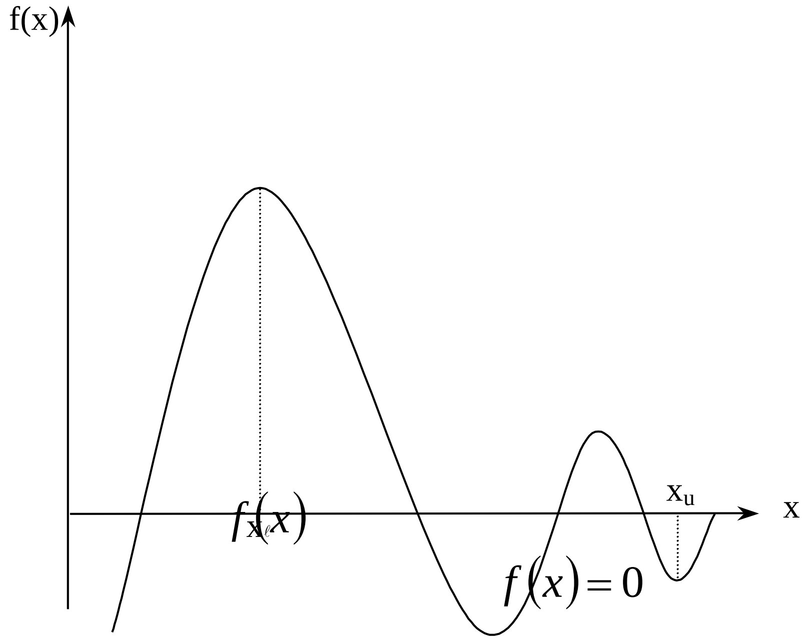
Basis of Bisection Method - 3

If the function f(x) does not change sign between two points, there may not be any roots for the equation f(x) =0 between the two points.



Basis of Bisection Method - 4

If the function f(x) changes sign between two points, more than one root for the equation f(x)=0 may exist between the two points.



Algorithm for Bisection Method

Step 1

Choose x_ℓ and x_u as two guesses for the root such that $f(x_\ell)\, f(x_u) < 0$, or in other words, f(x) changes sign between x_ℓ and x_u. This was demonstrated in Figure 1.

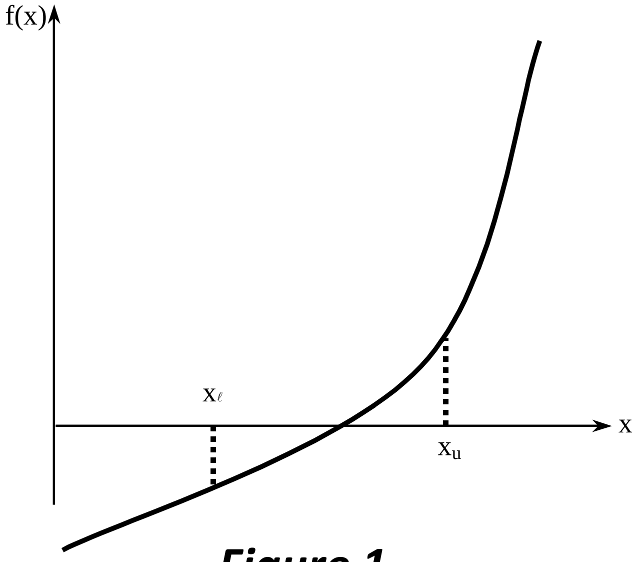


Figure 1

Step 2

Estimate the root, x_m of the equation f (x) = 0 as the mid point between x_ℓ and x_u as

$$x_m = \frac{x_\ell + x_u}{2}$$

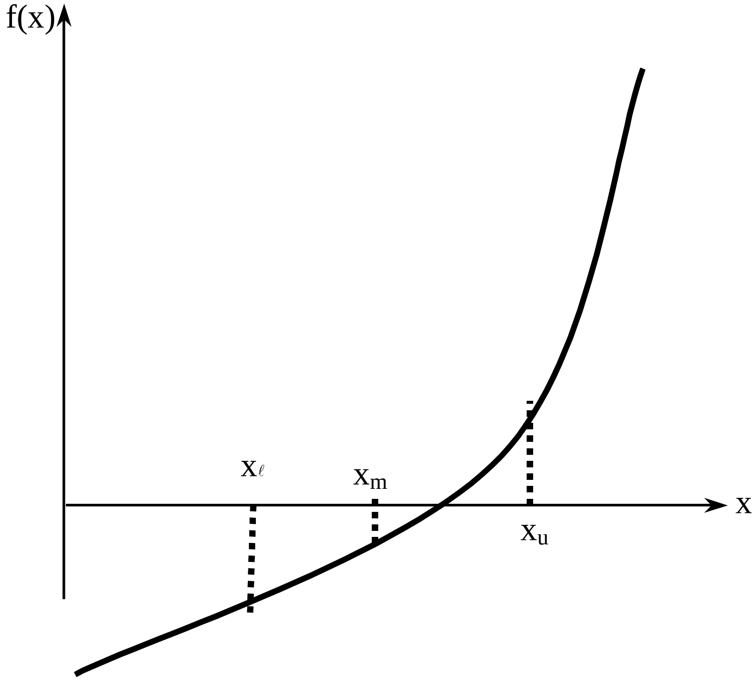


Figure 5 *Estimate of x_m*

Step 3

Now check the following

a) If $f(x_\ell)f(x_m)<0$, then the root lies between x_ℓ and x_m; then $x_\ell = x_\ell$; $x_u = x_m$.

b) If If $f(x_\ell)f(x_m)>0$, then the root lies between x_m and x_u; then $x_\ell = x_m$; $x_u = x_u$.

c) If If $f(x_\ell)f(x_m)==0$, then the root is x_m. Stop the algorithm if this is true.

Step 4

Find the new estimate of the root

$$x_m = \frac{x_\ell + x_u}{2}$$

Find the absolute relative approximate error

$$\left|\in_a\right| = \left|\frac{x_m^{new} - x_m^{old}}{x_m^{new}}\right| \times 100$$

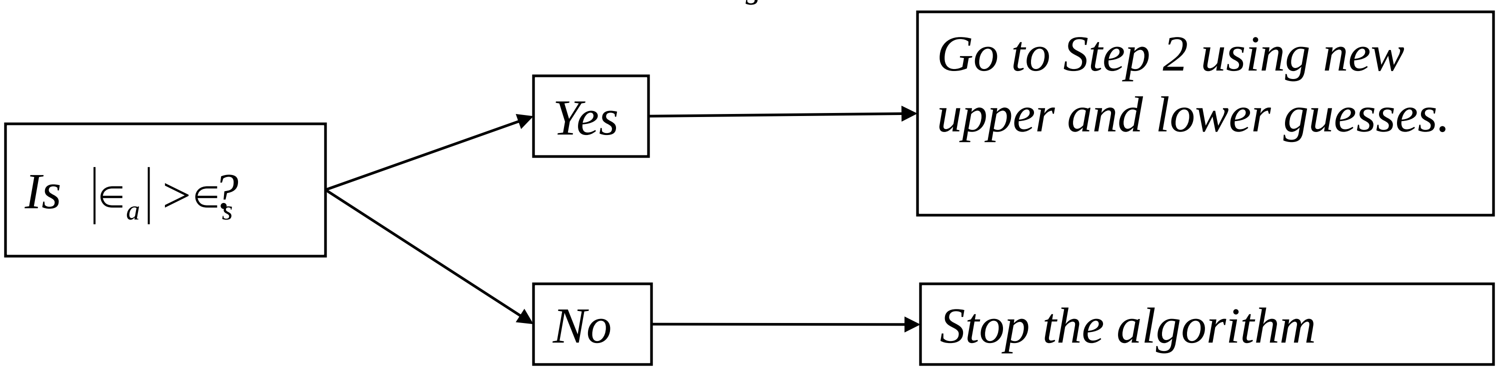
where

$$x_m^{old} = \text{previous estimate of root}$$

$$x_m^{new} = \text{current estimate of root}$$

Step 5

Compare the absolute relative approximate error $\left|\in_a\right|$ with the pre-specified error tolerance $\in_s$.



Note one should also check whether the number of iterations is more than the maximum number of iterations allowed. If so, one needs to terminate the algorithm and notify the user about it.

Bisection Method

Check the value of the function at the middle of the interval.

if it is positive,

replace the left endpoint with the middle point;

if it is negative, replace the right endpoint with the middle point.

Stay in a loop doing this until the interval size is less than epsilon. The interval end points (xleft and xright) and the tolerance for the approximation (epsilon) are entered by the user.