Programming and Data Structure

Sudeshna Sarkar

Dept. of Computer Science & Engineering.

Indian Institute of Technology

Kharagpur

16 Jan 2012

More about scanf and printf

Entering input data :: scanf function

- General syntax:

  scanf (control string, arg1, arg2, …, argn);

  – “control string refers to a string typically containing data types of the arguments to be read in;
  – the arguments arg1, arg2, … represent pointers to data items in memory.

  Example:

  scanf (%d %f %c”, &a, &average, &type);

- The control string consists of individual groups of characters, with one character group for each input data item.
  – ‘%’ sign, followed by a conversion character.

- Commonly used conversion characters:

  c single character

  d decimal integer

  f floating-point number

  s string terminated by null character

  X hexadecimal integer

- We can also specify the maximum field-width of a data item, by specifying a number indicating the field width before the conversion character.

  Example: **scanf (“%3d %5d”, &a, &b);**

scanf: return value

- On success, the function returns the number of items successfully read. This count can match the expected number of readings or fewer, even zero, if a matching failure happens.
  In the case of an input failure before any data could be successfully read, EOF is returned.

Writing output data :: printf function

- General syntax:

  printf (control string, arg1, arg2, …, argn);

  – “control string refers to a string containing formatting information and data types of the arguments to be output;

  – the arguments arg1, arg2, … represent the individual output data items.

- The conversion characters are the same as in scanf.

- Examples:

```
printf  ("The average of %d and %d is %f", a, b, avg);
printf  ("Hello \nGood \nMorning \n");
printf ("%3d %3d %5d", a, b, a*b+2);
printf ("%7.2f  %5.1f", x, y);
```

- Many more options are available:
  - Read from the book.
  - Practice them in the lab.
- String I/O:
  - Will be covered later in the class.

printf: return value

- On success, the total number of characters written is returned.
  On failure, a negative number is returned.

Shortcuts in Assignments

- Additional assignment operators:

  +=, −=, *=, /=, % =

  a += b is equivalent to a = a + b

  a *= (b+10) is equivalent to a = a * (b + 10)

  and so on.

Branching: The if Statement

```
if (expression)
        statement;

if (expression) {
        Block of statements;
}
```

The condition to be tested is any expression enclosed in parentheses. The expression is evaluated, and if its value is non-zero, the statement is executed.

Branching: if-else Statement

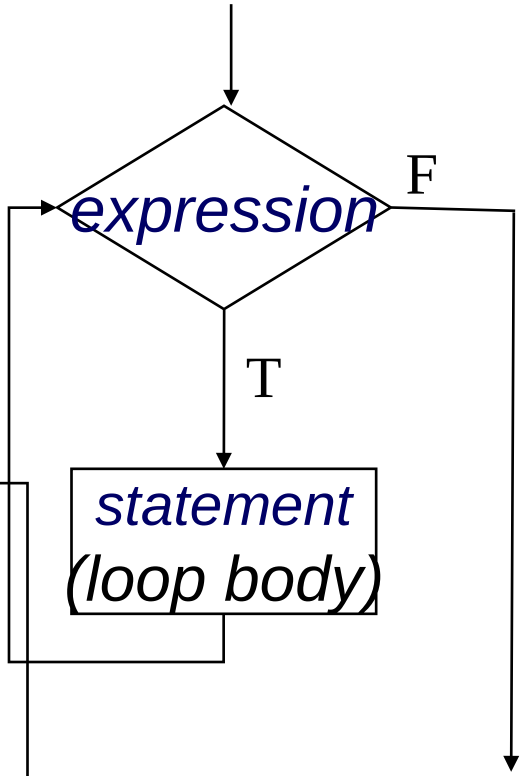
```
if (expression) {
    Block of statements;
}
else {
    Block of statements;
}
```

Control Flow: Looping

while statement

```
while (expression)
      statement
```

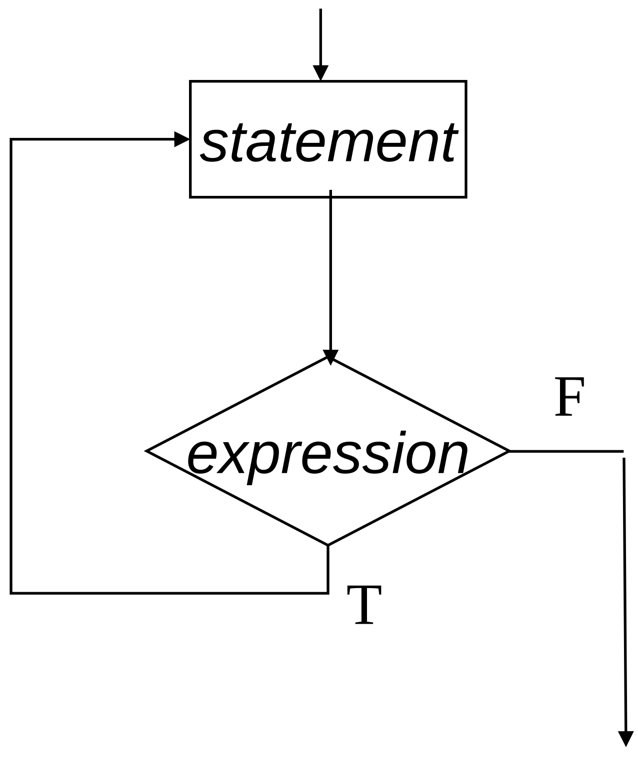
```
while (i < n)  {
      printf (“Line no : %d.\n”,i);
      i++;
}
```



do-while statement

do *statement* while (*expression*)

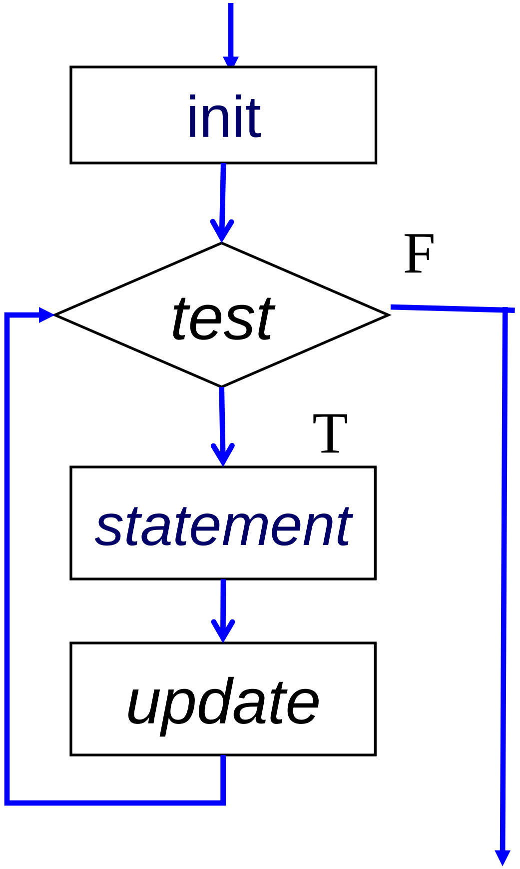
```
int digit=0;
do
    printf("%d\n",digit++);
while (digit <= 9) ;
```



for Statement

For (init; test; update) *statement*

```
sum=0 ;
term = 1 ;
for (i=1; i<n; i++) {
    term = term*i ;
    sum = sum + term ;
}
```



Another 2-D Figure

Print the following pattern:

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

Another 2-D Figure

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

```
#define ROWS 5
....
int row, col;
for (row=1; row<=ROWS; row++) {
    for (col=1; col<=row; col++) {
        printf("* ");
    }
    printf("\n");
}
```

For - Examples

- **Problem 1**: Write a for statement that computes the sum of all odd numbers between 1000 and 2000.
- **Problem 2**: Write a for statement that computes the sum of all numbers between 1000 and 10000 that are divisible by 17.
- **Problem 3**: Print a hollow square of size n.
- **Problem 4**: Print

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

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

The comma operator

- We can give several statements separated by commas in place of “expression1”, “expression2”, and “expression3”.

```
for  (fact=1, i=1; i<=10; i++)
   fact = fact * i;

for (sum=0, i=1; i<=N, i++)
   sum = sum + i * i;
```

for :: Some Observations

- Arithmetic expressions
  - Initialization, loop-continuation, and increment can contain arithmetic expressions.

    for (k = x;   k <= 4 * x * y;   k += y / x)
- "Increment" may be negative (decrement)

    for (digit=9; digit>=0; digit--)
- If loop continuation condition is initially *false:*
  - Body of *for* structure not performed.
  - Control proceeds with statement after *for* structure.

Specifying “Infinite Loop”

```
while  (1)  {
   statements
}
```

```
for  (; ;)
{
     statements
}
```

```
do  {
   statements
}  while (1);
```

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

- 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
```

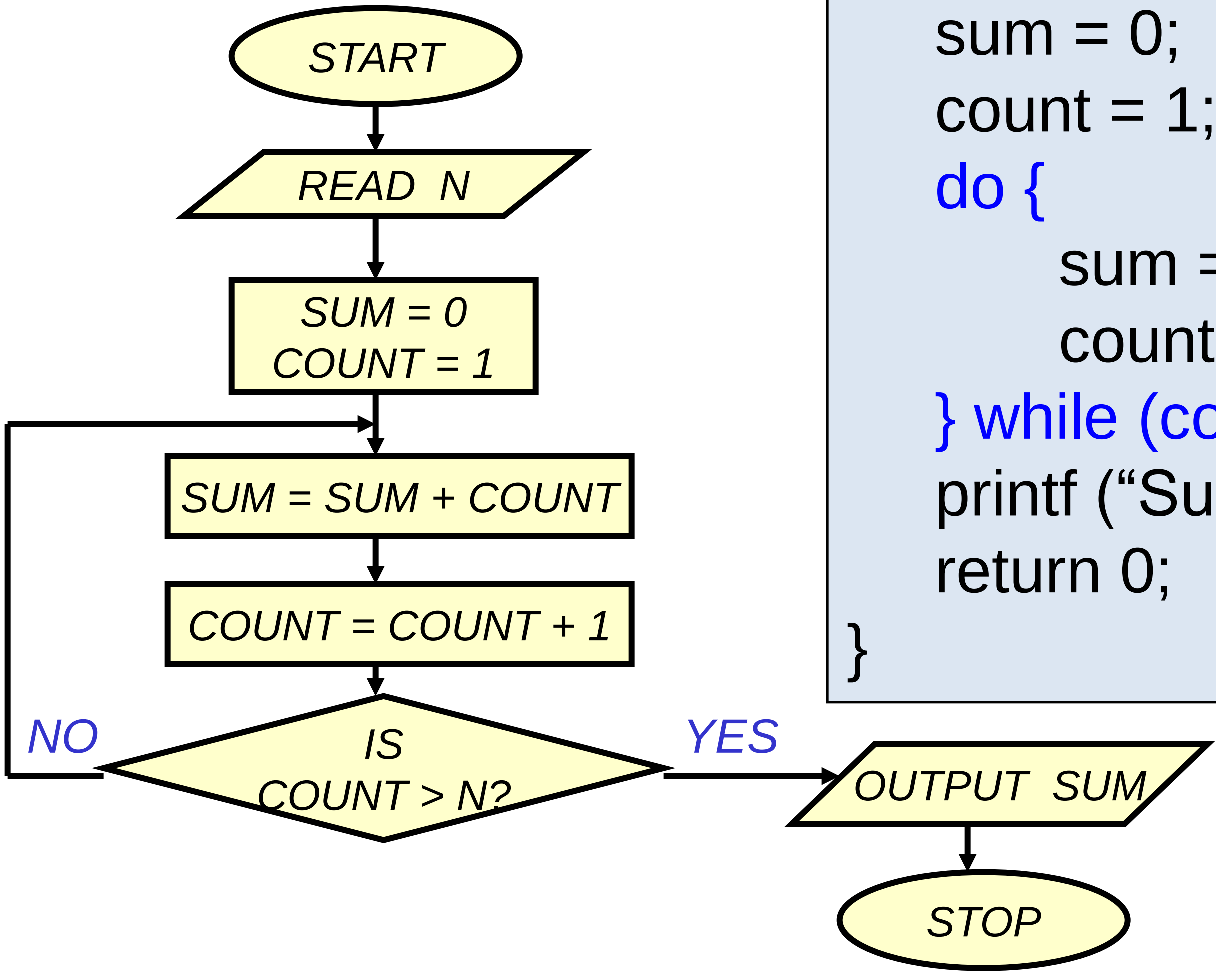
```c
float readAverage (void)
{
// Local Declarations
int    count = 0;

int    n;
float sum = 0;

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

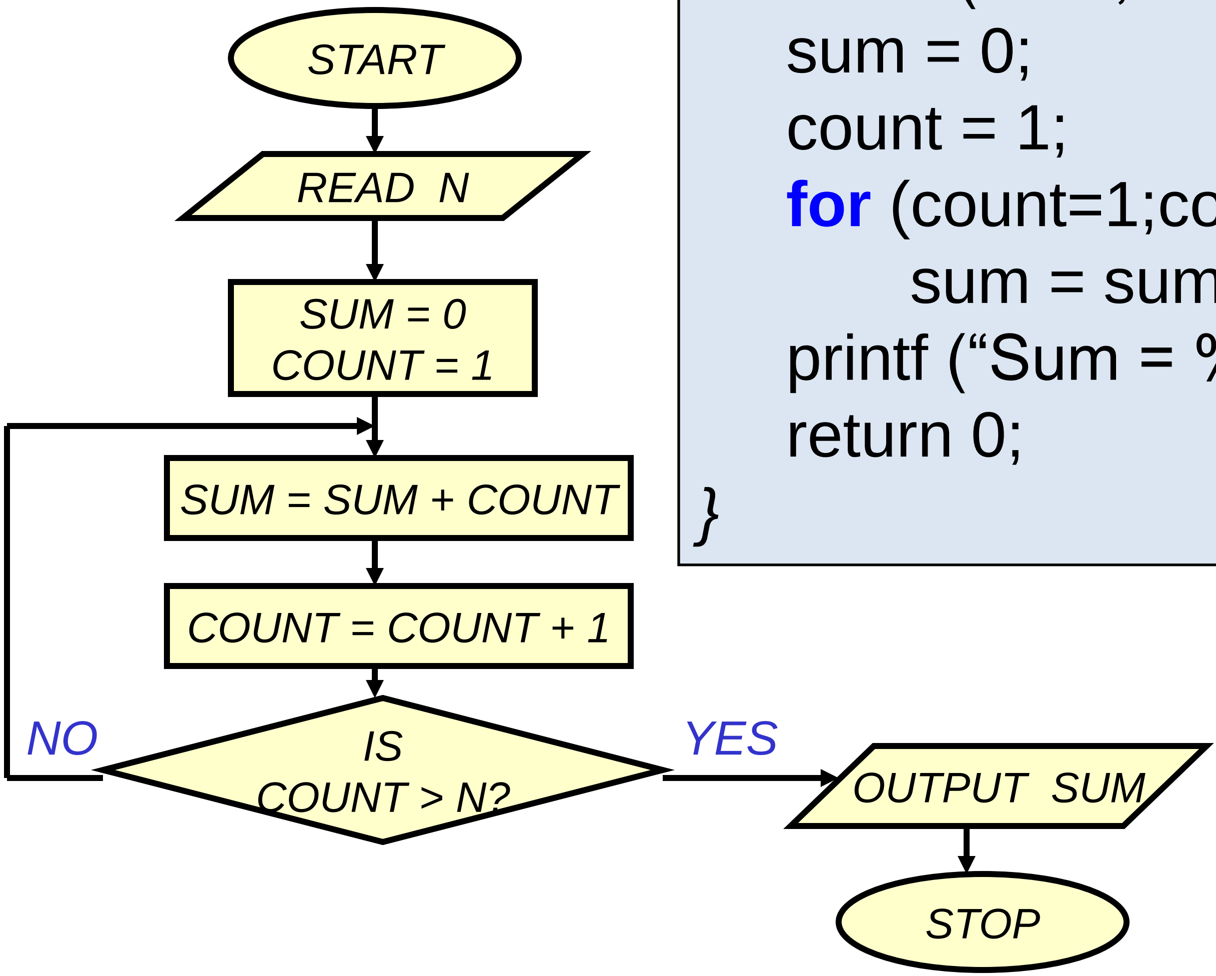
Programming Examples

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;
}
```

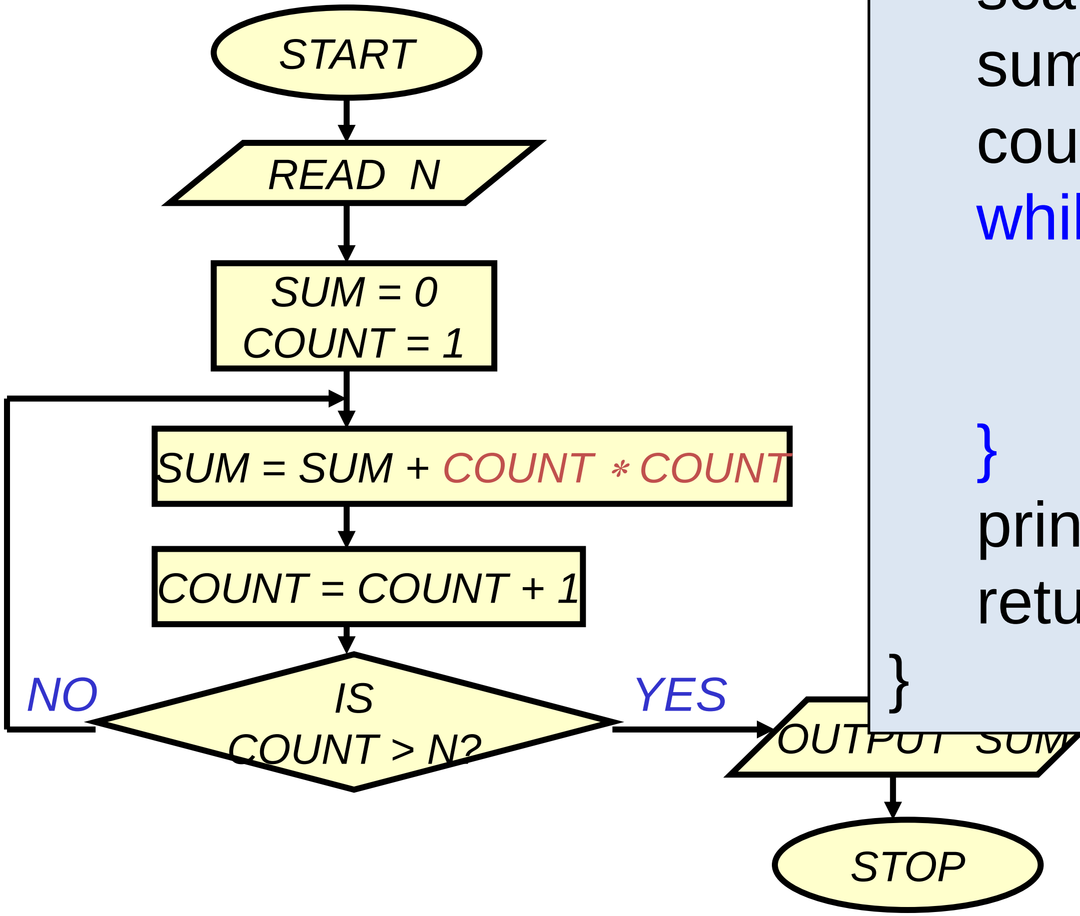
Sum of first N natural numbers




```
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:

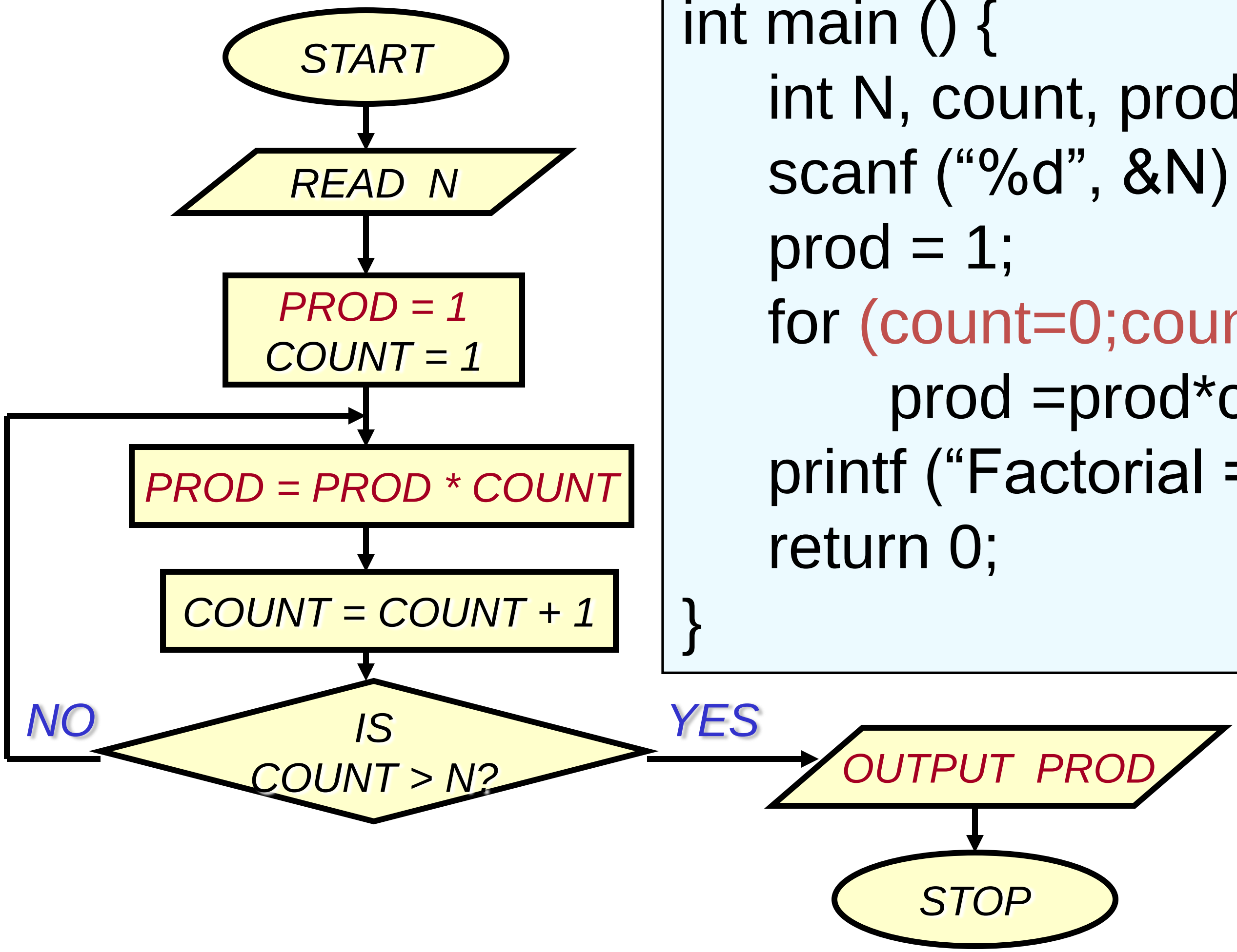
$SUM = 1^2 + 2^2 + 3^2 + N^2$




```
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;
}
```

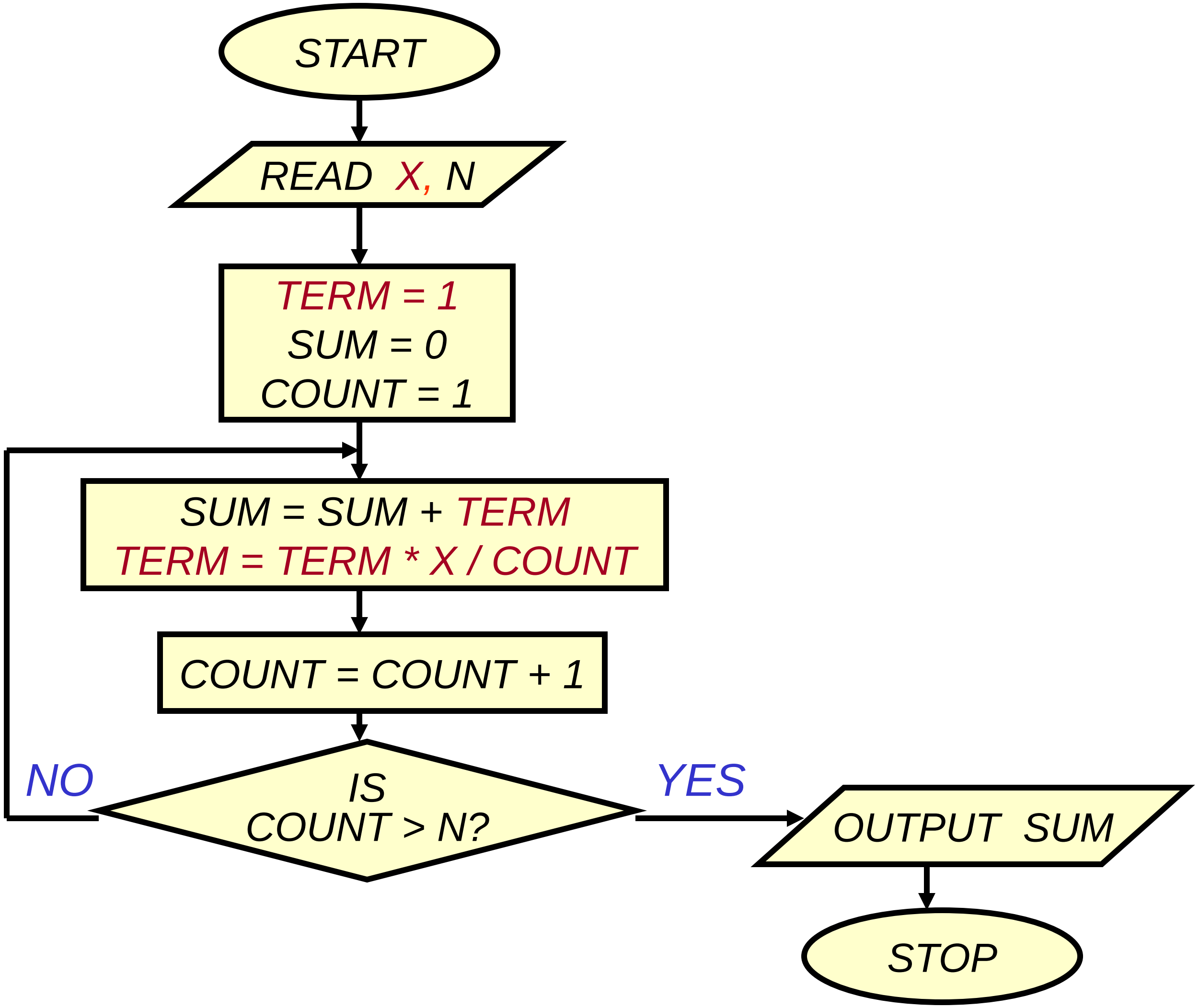
Example 3:

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;
}
```

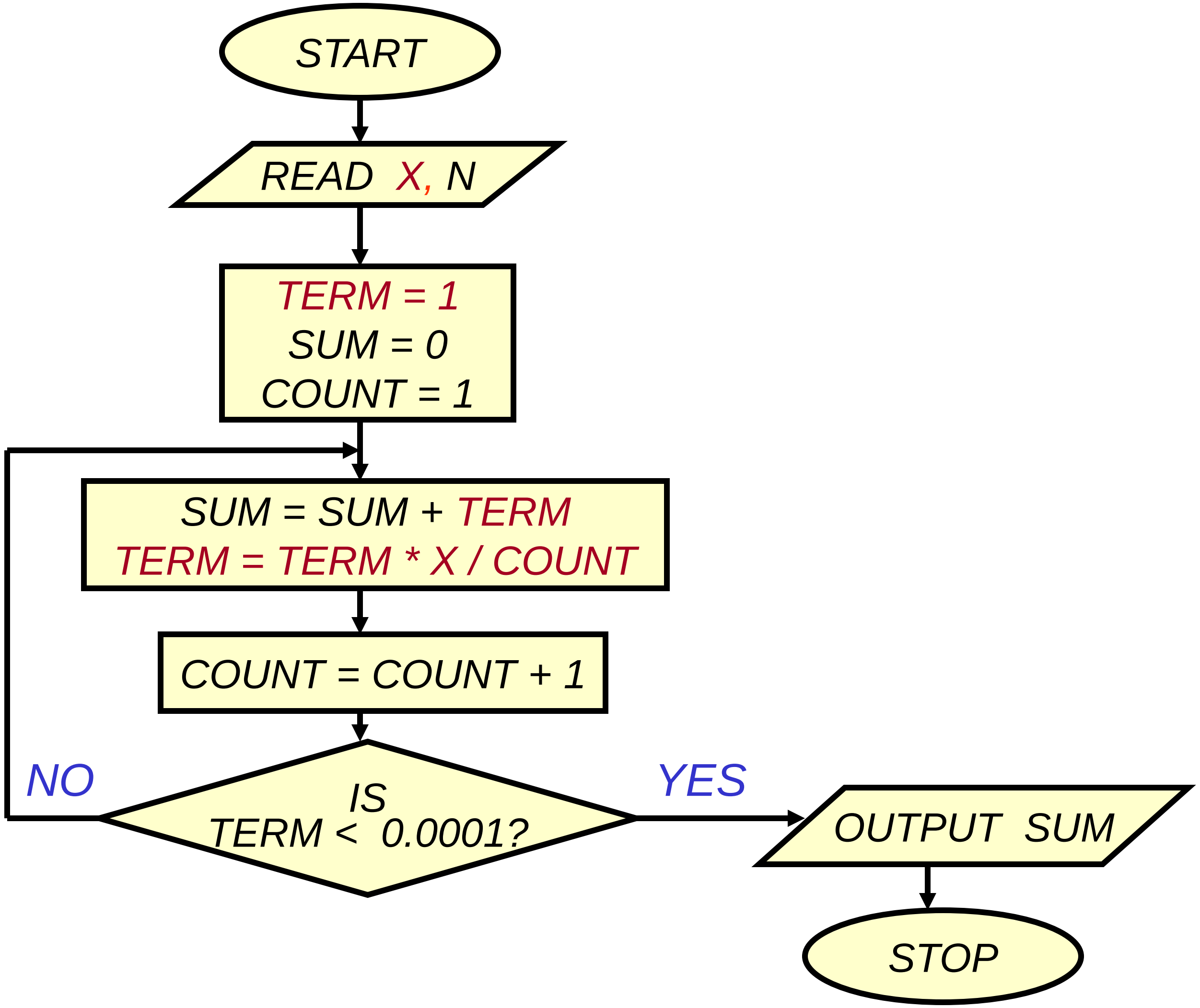
Example 4: Computing e^x series up to N terms



```
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);
}
```

Example 8: Approximating the logarithm

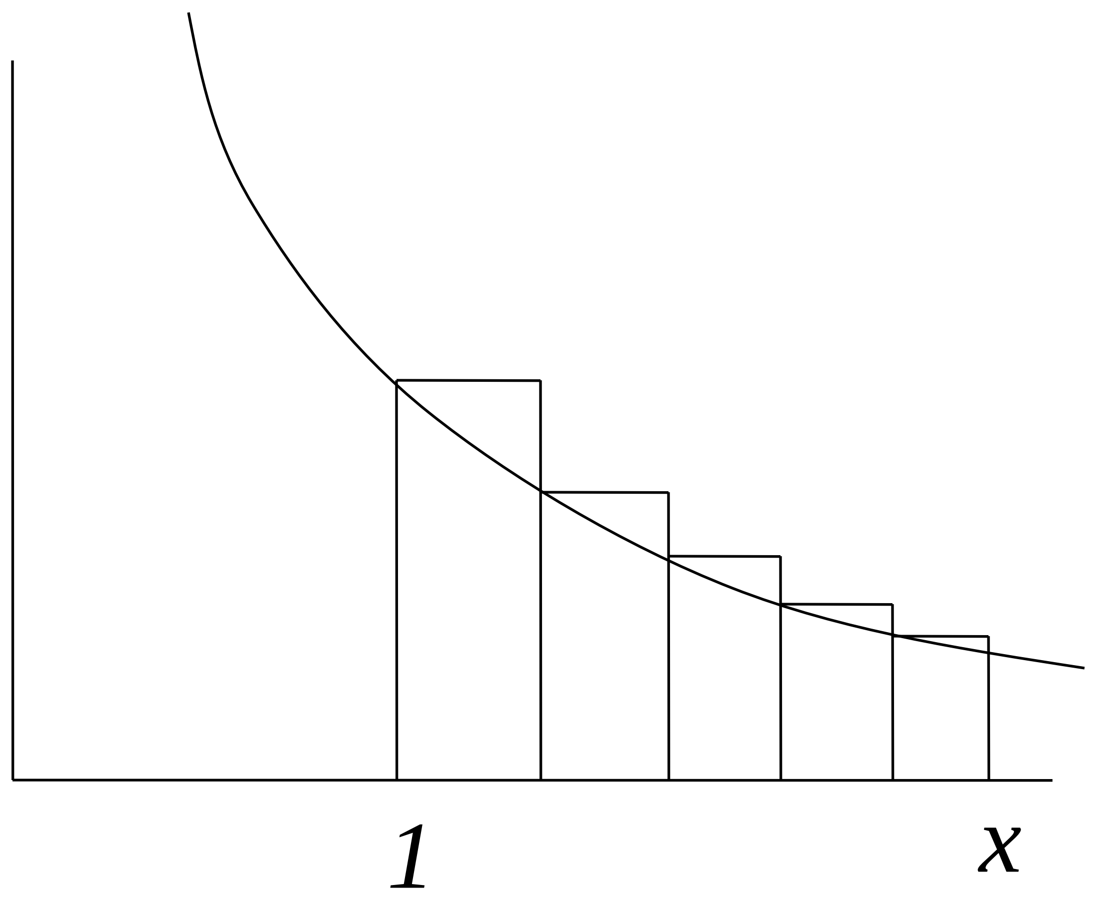
- How many times must we divide a number x by 10 until the result goes below 1?

```
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.

12) 45 (3
36
9) 12 (1
9
3) 9 (3
9
0

[*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.

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

```
#include  <stdio.h>
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);
}
```

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:

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

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.

The *bisection* method works as follows:

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. This halves the size of the interval. 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.

For this lab, consider finding the root of the function
$p(x) = 5x^3 - 2x - 2$
over the interval [0,2] using epsilon = 0.0001.
Also print the number of iterations required for this value of

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.