

Loops and Iteration

CS10003 PROGRAMMING AND DATA STRUCTURES

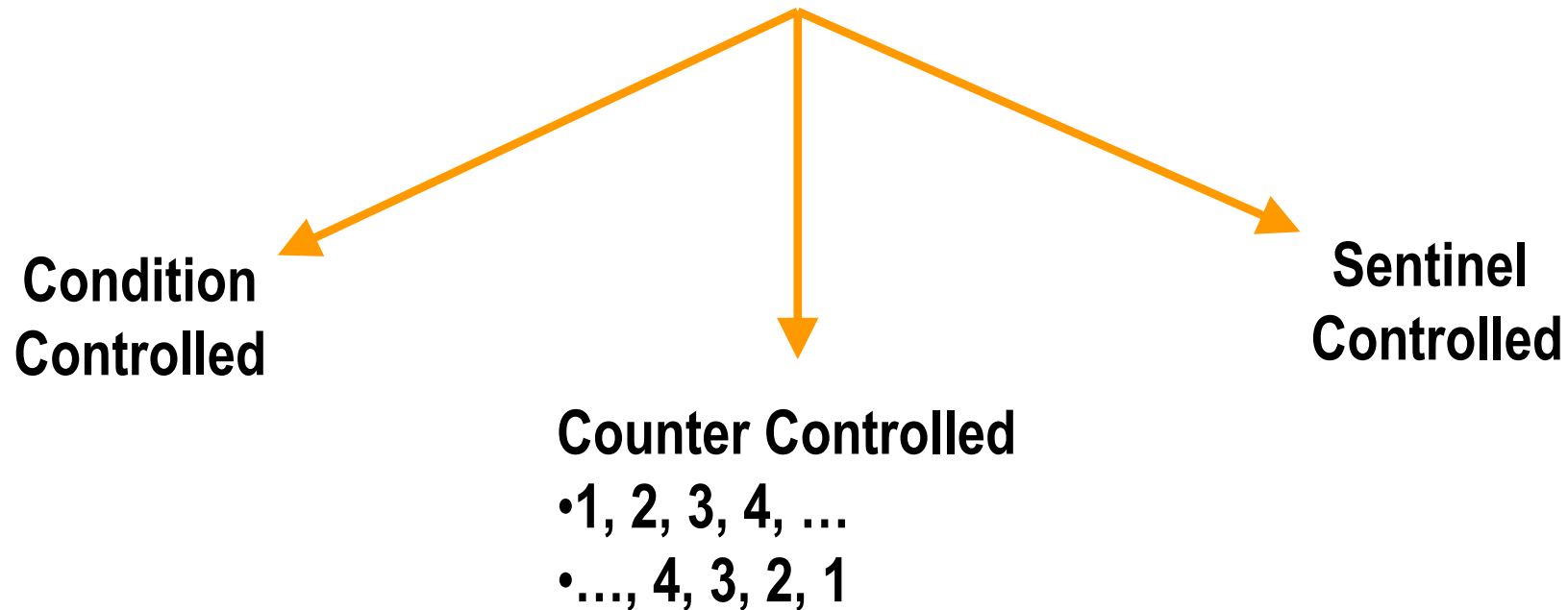


Looping Constructs

Types of Repeated Execution

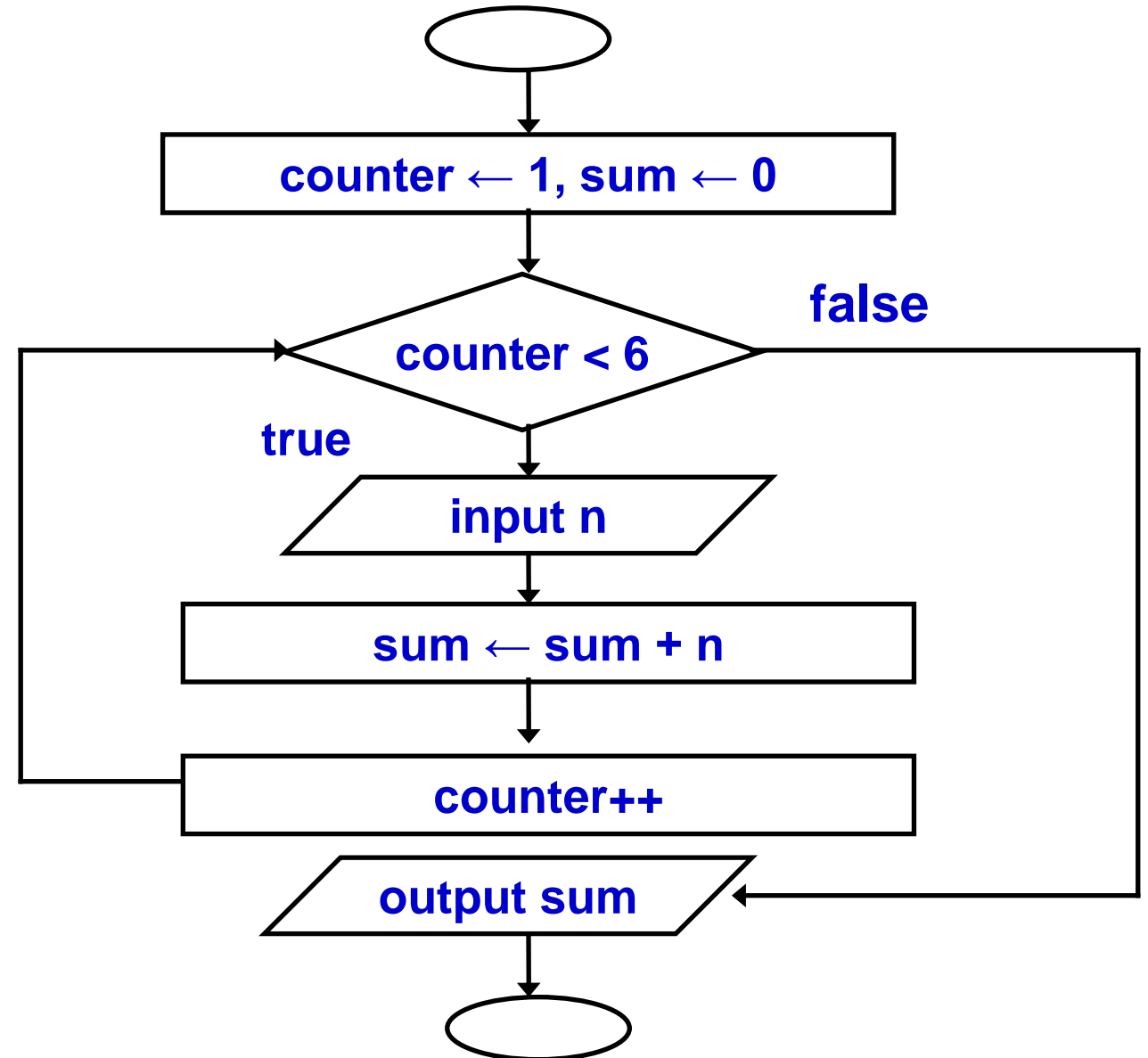
Loop: Group of instructions that are executed repeatedly while some condition remains true.

How loops are controlled



Counter Controlled Loop

Read 5 integers and display the value of their sum.

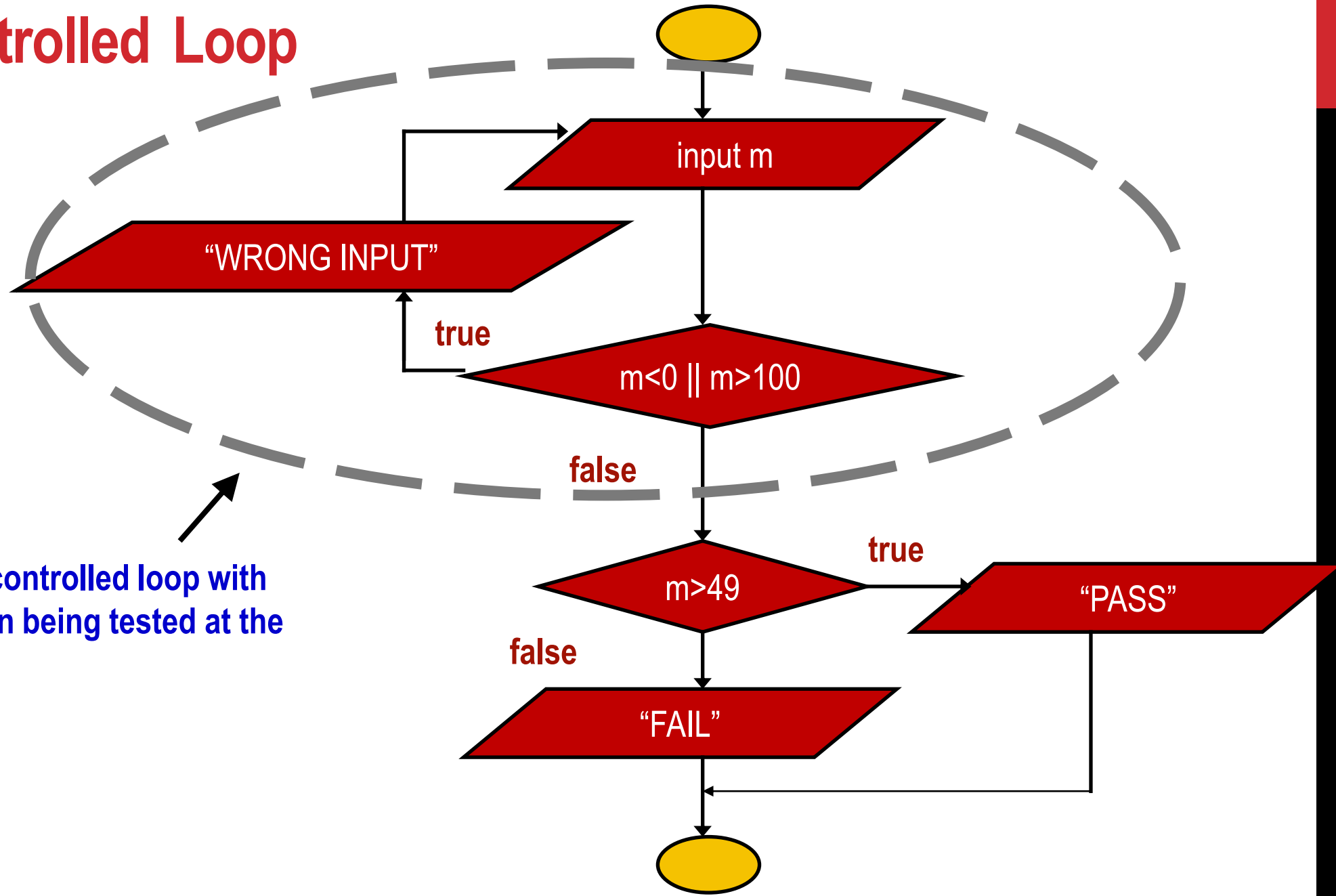


Condition-controlled Loop

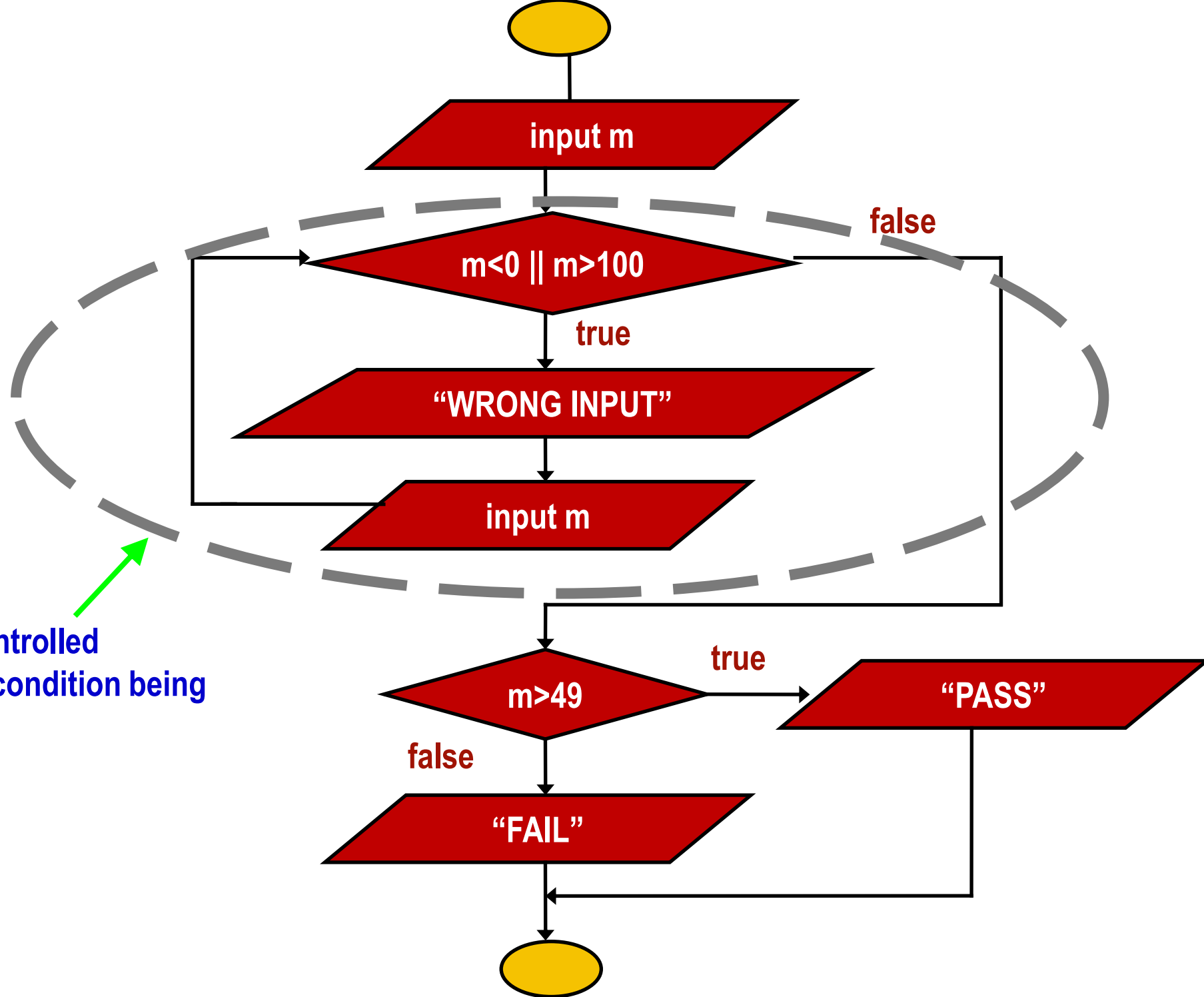
Given an exam marks as input, display the appropriate message based on the rules below:

- If marks is greater than 49, display “PASS”, otherwise display “FAIL”
- However, for input outside the 0-100 range, display “WRONG INPUT” and prompt the user to input again until a valid input is entered

Condition-Controlled Loop



Condition-controlled loop with its condition being tested at the end



Condition-controlled loop with its condition being tested first

Sentinel-Controlled Loop

Receive a number of positive integers and display the summation and average of these integers.

A negative or zero input indicates the end of input process

Draw the flow chart.

while Statement

The “while” statement is used to carry out looping operations, in which a group of statements is executed repeatedly, as long as some condition remains satisfied.

```
while (condition)
    statement_to_repeat;
```

```
while (condition) {
    statement_1;
    ...
    statement_N;
}
```

Note:

The while-loop will not be entered if the loop-control expression evaluates to false (zero) even before the first iteration.

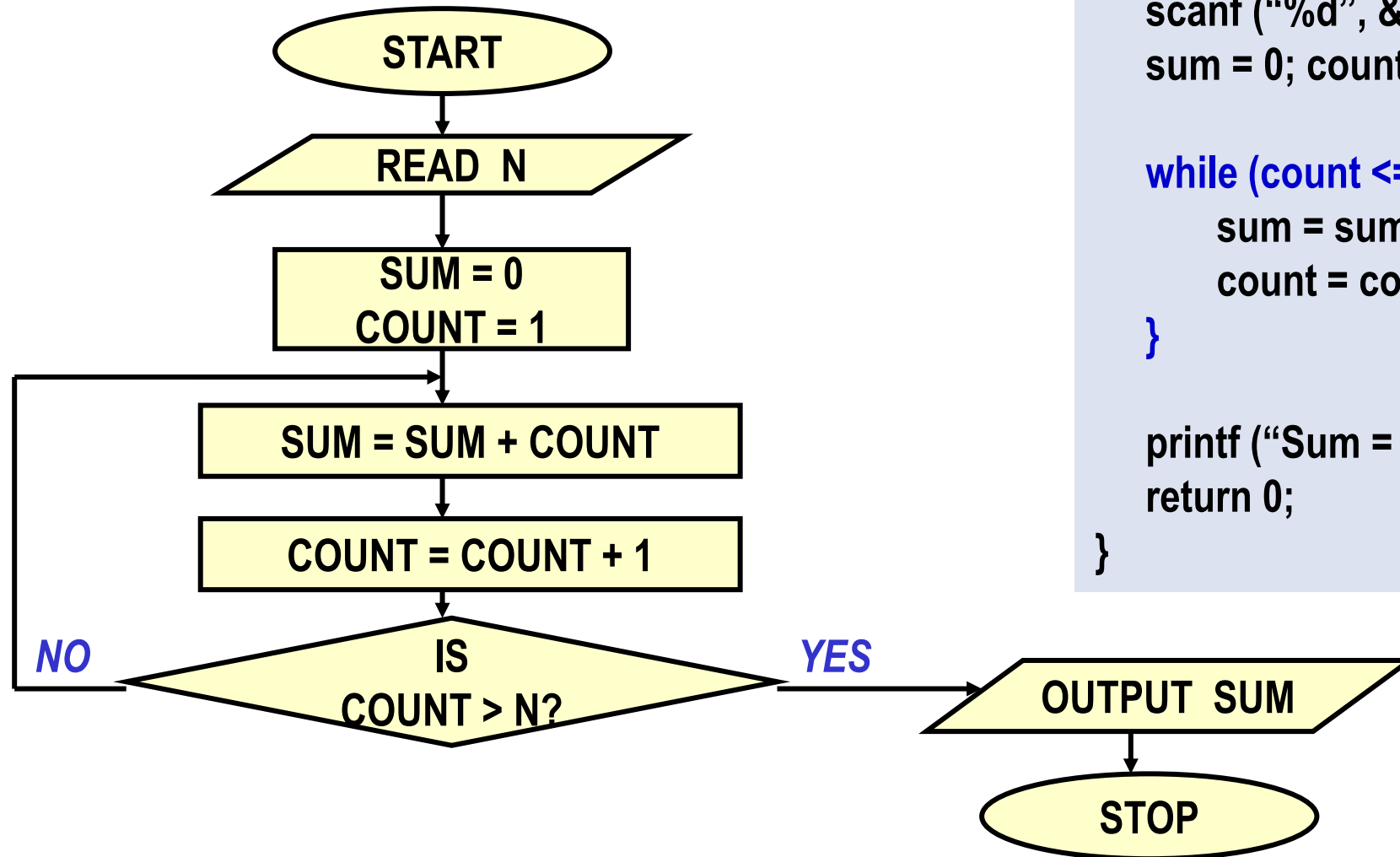
The *break* statement can be used to come out of the while loop.

while:: Examples

```
int weight;
```

```
while ( weight > 65 ) {  
    printf ("Go, exercise, ");  
    printf (" ... then come back. \n");  
    printf ("Enter your weight: ");  
    scanf ("%d", &weight);  
}
```

Sum of first N natural numbers



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

Double your money

Suppose your Rs 10000 is earning interest at 1% per month. How many months until you double your money ?

```
my_money=10000.0;
n=0;

while (my_money < 20000.0) {
    my_money = my_money * 1.01;
    n++;
}

printf ("My money will double in %d months.\n",n);
```

Maximum of inputs

```
printf ("Enter positive numbers to max, end with -1.0\n");  
max = 0.0; count = 0;  
scanf ("%f", &next);  
while (next != 1.0) {  
    if (next > max) max = next;  
    count++;  
    scanf ("%f", &next);  
}  
printf ("The maximum number is %f\n", max);
```

Printing a 2-D Figure

How would you print the following diagram?

* * * * *

* * * * *

* * * * *


repeat 3 times

print a row of 5 stars

repeat 5 times
print *

Nested Loops

```
#define ROWS 3
#define COLS 5
...
row=1;
while (row <= ROWS) {
    /* print a row of 5 '*'s */
    ...
    printf("\n");
    row++;
}
```

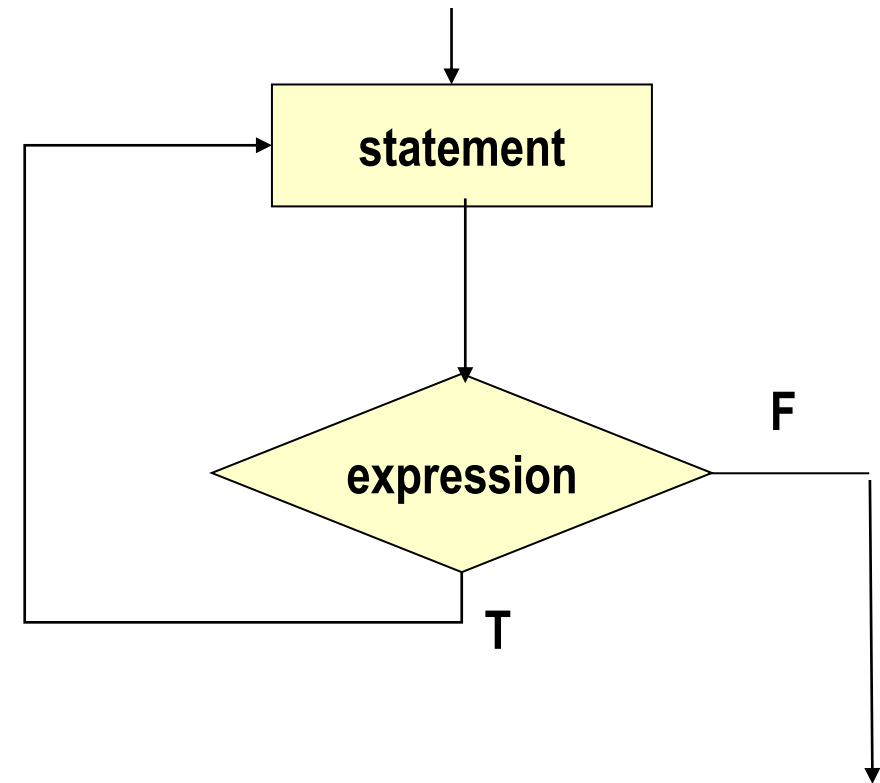


```
while (col <= COLS) {
    printf ("* ");
    col++;
}
```

do-while statement

`do statement while (expression)`

```
main () {  
    int digit=0;  
    do  
        printf(“%d\n”,digit++);  
    while (digit <= 9) ;  
}
```



for Statement

The “for” statement is the most commonly used looping structure in C.

General syntax:

```
for ( expr1; expr2; expr3) statement
```

expr1: initializes loop parameters

expr2: test condition, loop continues if this is satisfied

expr3: used to alter the value of the parameters after each iteration

statement: body of the loop

Sum of first N natural numbers

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

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

2-D Figure

Print

* * * * *

* * * * *

* * * * *

```
#define ROWS 3
#define COLS 5
....
for (row=1; row<=ROWS; row++) {
    for (col=1; col<=COLS; col++) {
        printf("*");
    }
    printf("\n");
}
```

Another 2-D Figure

Print

*

* *

* * *

* * * *

* * * * *

```
#define ROWS 5
```

```
....
```

```
int row, col;
```

```
for (row=1; row<=ROWS; row++) {
```

```
    for (col=1; col<=row; col++) {
```

```
        printf("* ");
```

```
    }
```

```
    printf("\n");
```

```
}
```

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**
 - **else**

Causes immediate exit from a ***while***, ***do/while***, ***for*** or ***switch*** structure.

Program execution continues with the first statement after the structure.

Example: Find smallest n such that $n!$ exceeds 100

```
#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 ++ ;
    }
}
```

Test if a number is prime or not

```
int main() {  
    int n, i=2;  
    double limit;  
    scanf ("%d", &n);  
    limit = sqrt(n);  
    for (i = 2, i <= limit; i++) {  
        if (n % i == 0) {  
            printf ("%d is not a prime \n", n);  
            break;  
        }  
    }  
    if (i > limit) printf ("%d is a prime \n", n);  
    return 0;  
}
```


Another Way

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

- *expression3* is evaluated, then *expression2* 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;  
}
```

Example with **break** and **continue**:

Add positive numbers until a 0 is typed, but ignore any negative numbers typed

```
int main() {  
    int sum = 0, next;  
    while (1) {  
        scanf("%d", &next);  
        if (next < 0) continue;  
        else if (next == 0) break;  
        sum = sum + next;  
    }  
    printf ("Sum = %d\n", sum) ;  
    return 0;  
}
```

Output

```
10  
-20  
30  
40  
-5  
10  
0  
Sum = 90
```

Some Loop Pitfalls

```
while (sum <= NUM) ;  
    sum = sum+2;
```

```
for (i=0; i<=NUM; ++i);  
    sum = sum+i;
```

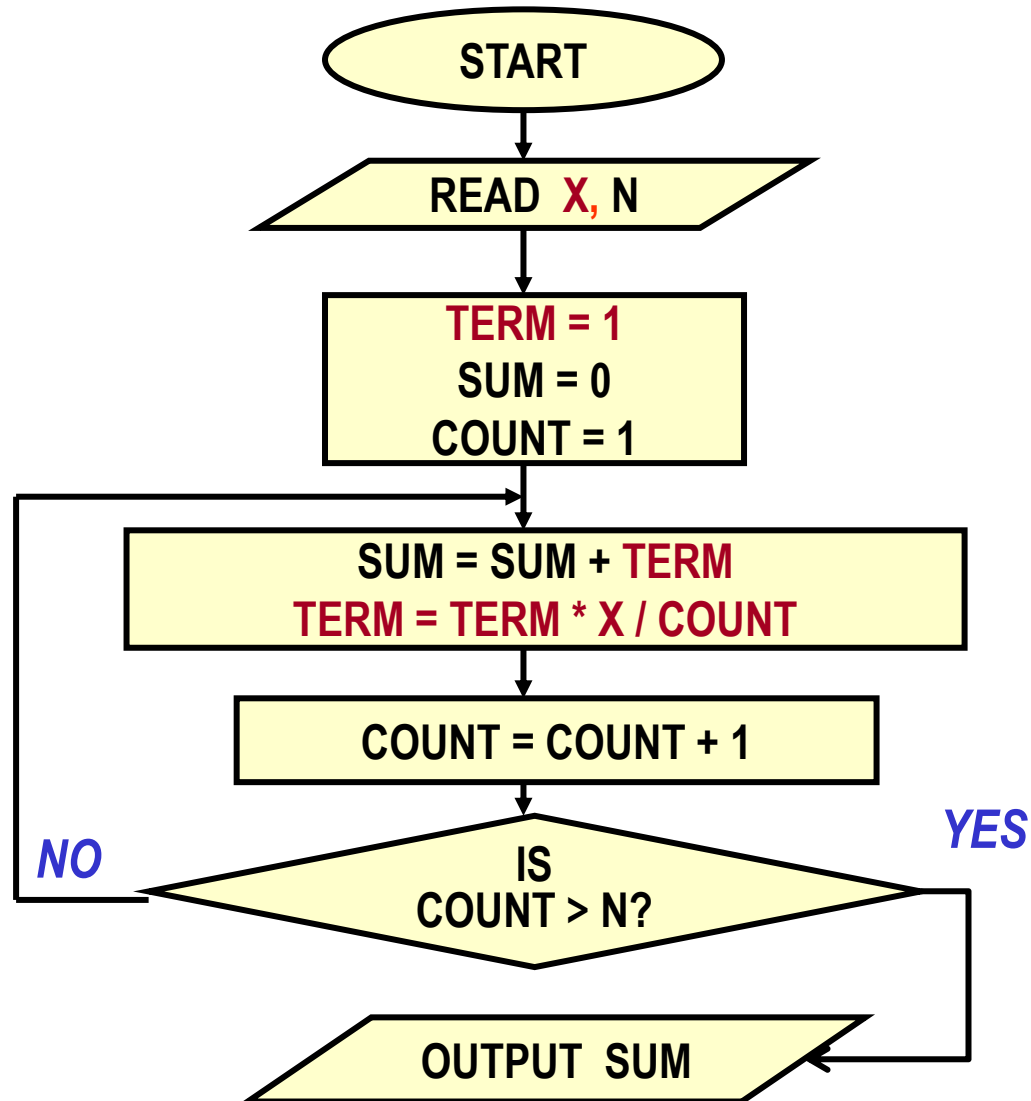
```
for (i=1; i!=10; i=i+2)  
    sum = sum+i;
```

```
double x;  
for (x=0.0; x != 2.0; x=x+0.2)  
    printf("%.18f\n", x);
```

Some Examples



Example: Computing e^x series up to N terms ($1 + x + (x^2 / 2!) + (x^3 / 3!) + \dots$)



```
int main () {  
    float x, term, sum;  
    int n, count;  
    scanf ("%d", &x) ;  
    scanf ("%d", &n) ;  
    term = 1.0; sum = 0;  
    for (count = 1; count <= n; count++) {  
        sum += term;  
        term *= x/count;  
    }  
    printf ("%f\n", sum) ;  
}
```

Computing e^x 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 = 1; term>=0.0001; count++) {  
        sum += term;  
        term *= x/count;  
    }  
    printf ("%f\n", sum) ;  
}
```


Example: Decimal to binary conversion

```
#include <stdio.h>
main()
{
    int dec;
    scanf ("%d", &dec);
    do
    {
        printf ("%2d", (dec % 2));
        dec = dec / 2;
    } while (dec != 0);
    printf ("\n");
}
```

In which order are the bits printed?

break and continue with nested loops

For nested loops, break and continue are matched with the nearest loops (for, while, do-while)

Example:

```
while (i < n) {  
    for (k=1; k < m; ++k) {  
        if (k % i == 0) break;  
    }  
    i = i + 1;  
}
```

Breaks here

Example

```
int main()
{
    int low, high, desired, i, flag = 0;
    scanf("%d %d %d", &low, &high, &desired);
    i = low;
    while (i < high) {
        for (j = i+1; j <= high; ++j) {
            if (j % i == desired) {
                flag = 1;
                break;
            }
        }
        if (flag == 1) break;
        i = i + 1;
    }
    return 0;
}
```

Breaks here

Breaks here

The comma operator

- Separates expressions
- Syntax
 - `expr-1, expr-2, ...,expr-n`
 - `expr-1, expr-2,...` are all expressions
- Is itself an expression, which evaluates to the value of the last expression in the sequence
- Since all but last expression values are discarded, not of much general use
- But useful in for loops, by using side effects of the expressions

Example

- We can give several expressions separated by commas in place of `expr1` and `expr3` in a for loop to do multiple assignments for example

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

```
    fact = fact * i;
```

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

```
    sum = sum + i * i;
```

Practice Problems

Practice Problems (do each with both for and while loops separately)

1. Read in an integer N. Then print the sum of the squares of the first N natural numbers
2. Read in an integer N. Then read in N numbers and print their maximum and second maximum (do not use arrays even if you know it)
3. Read in an integer N. Then read in N numbers and print the number of integers between 0 and 10 (including both), between 11 and 20, and > 20 . (do not use arrays even if you know it)
4. Repeat 3, but this time print the average of the numbers in each range.
5. Read in a positive integer N. If the user enters a negative integer or 0, print a message asking the user to enter the integer again. When the user enters a positive integer N finally, find the sum of the logarithmic series $(\log_e(1+x))$ upto the first N terms
6. Read in an integer N. Then read in integers, and find the sum of the first N positive integers read. Ignore any negative integers or 0 read (so you may actually read in more than N integers, just find the sum with only the positive integers and stop when N such positive integers are read)
7. Read in characters until the '\n' character is typed. Count and print the number of lowercase letters, the number of uppercase letters, and the number of digits entered.

Additional Examples



ISBN Numbers



Checking for Legal ISBN Numbers

10 th	9 th	8 th	7 th	6 th	5 th	4 th	3 rd	2 nd	1 st
D ₁₀	D ₉	D ₈	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁

An ISBN number must:

- Contain 10 symbols , D₁ ,..., D₁₀ where D₁ is a checksum between 1 and 10.
 - If D₁ is 10, then it is represented as X.

- The sum:

$$10 * D_{10} + 9 * D_9 + 8 * D_8 + 7 * D_7 + 6 * D_6 + 5 * D_5 + 4 * D_4 + 3 * D_3 + 2 * D_2 + 1 * D_1$$

should be divisible by 11

- Given digits 2 to 10, the correct 1st digit has to be computed such that the remainder of dividing the sum by 11 (unless the remainder is already 0)

Read the 9 digit integer and compute the weighted sum

```
#include <stdio.h>
int main(void) {
    int isbn, i, digit, sum=0;
    printf("Enter the first 9 digits of the ISBN Number:");
    scanf("%d",&isbn);

    // Compute the sum:  $10 * D_{10} + 9 * D_9 + \dots + 3 * D_3 + 2 * D_2$ 
    for ( i=2; i<=10; i++ ) {
        digit = isbn % 10 ;
        isbn = isbn / 10 ; // Note the use of integer division
        sum = i * digit ;
    }
}
```

Compute and print the checksum digit

```
#include <stdio.h>
int main(void) {
    int isbn, i, digit, sum=0;
    char checksum;
    printf("Enter the first 9 digits of the ISBN Number:");
    scanf("%d",&isbn);
    for ( i=2; i<=10; i++ ) {
        digit = isbn % 10; isbn = isbn / 10; sum = i * digit;
    }
    if (sum % 11 == 1) checksum = 'X';
    else if (sum % 11 == 0) checksum = '0';
    else if checksum = '0' + 11 - (sum%11) ;
    printf("Checksum digit = %c\n", checksum);
}
```

BISECTION METHOD FOR ROOT FINDING

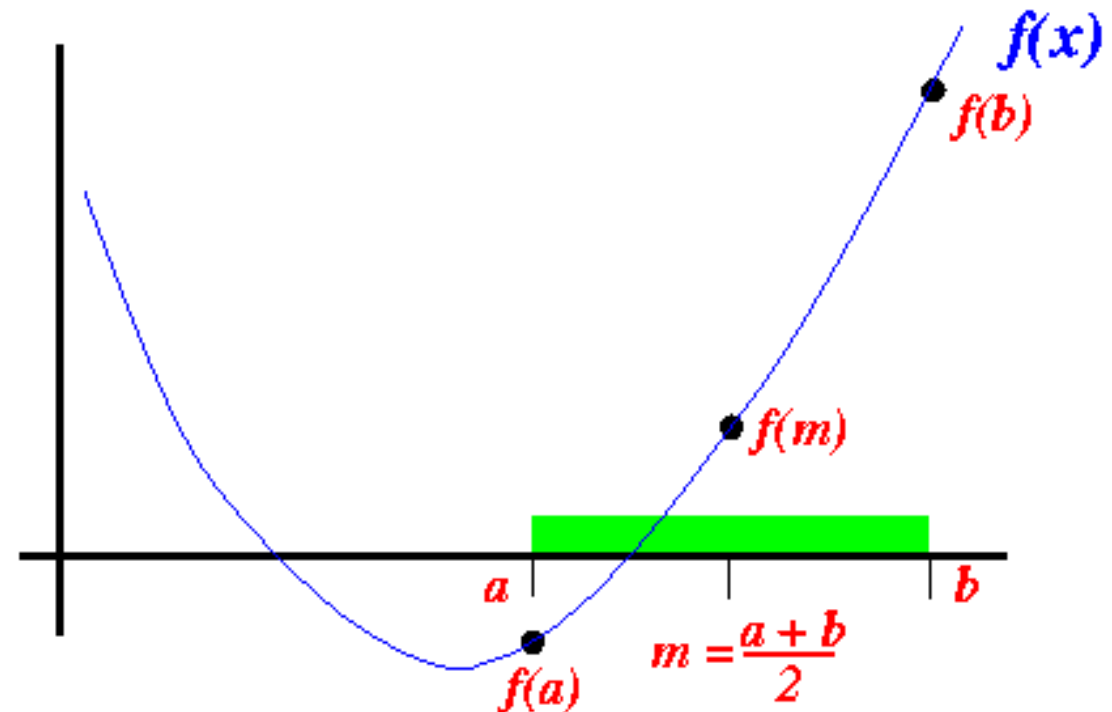
A method for finding the root of a function

Observation: *If the sign of $f(a)$ and $f(b)$ are different, then there is a root between a and b*

In each iteration:

- Find the mid point, m , between a and b
- If $f(a)$ and $f(m)$ have opposite signs then revise b to m
- If $f(b)$ and $f(m)$ have opposite signs then revise a to m

Continue until desired accuracy is reached



Bisection Method for $4x^3 - 3x^2 + 2x - 5$

```
int main(void)
{
    double a, b, m;
    printf("Enter initial left and right bounds:");
    scanf("%lf %lf", &a, &b); // For simplicity, we will assume that the bounds are valid

    while ( to be explained )
    {
        m = (a + b) / 2;
        if ((4*b*b*b - 3*b*b + 2*b - 5) * (4*m*m*m - 3*m*m + 2*m - 5) >= 0) b = m;
        else a = m;
    }
}
```

When to terminate?

```
int main(void)
{
    double a, b, m, margin;
    printf("Enter initial left and right bounds and the margin:");
    scanf("%lf %lf%lf", &a, &b, &margin);

    while ( (b - a) > margin )
    {
        m = (a + b) / 2;
        if ((4*b*b*b - 3*b*b + 2*b - 5) * (4*m*m*m - 3*m*m + 2*m - 5) >= 0) b = m;
        else a = m;
    }
}
```


Terminate after some iterations if it does not reach margin

```
int main(void)
{
    double a, b, m, margin;
    int bound;
    printf("Enter initial left and right bounds , the margin, and iteration bound:");
    scanf("%lf%lf %lf%d", &a, &b, &margin, &bound);

    while ( ((b - a) > margin) && (bound > 0) )
    {
        bound -- ;
        m = (a + b) / 2;
        if ((4*b*b*b - 3*b*b + 2*b - 5) * (4*m*m*m - 3*m*m + 2*m - 5) >= 0) b = m;
        else a = m;
    }
    printf ("Root = %lf\n", (a+b)/2 );
}
```