Conditionals and Looping

Pallab Dasgupta
Professor,
Dept. of Computer Sc & Engg

Statements and Blocks

An expression followed by a semicolon becomes a statement.

```
x = 5;
i++;
printf (“The sum is %d\n”, sum”) ;
```

Braces { and } are used to group declarations and statements together into a compound statement, or block.

```
{
    sum = sum + count;
    count++;
    printf (“sum = %d\n”, sum) ;
}
```

Control Statements: What do they do?

Branching:

- Allow different sets of instructions to be executed depending on the outcome of a logical test.
    - Whether TRUE (non-zero) or FALSE (zero).

Looping:

- Some applications may also require that a set of instructions be executed repeatedly, possibly again based on some condition.

Conditional Constructs

How do we specify the conditions?

Using relational operators.

- Four relation operators: <, <=, >, >=
- Two equality operations: ==, !=

Using logical operators / connectives.

- Two logical connectives: &&, | |
- Unary negation operator: !

Expressions

```
( count <= 100 )

( (math+phys+chem ) / 3 >= 60 )

( (sex == ’M’) && (age >= 21) )

( (marks >= 80) && (marks < 90) )

( (balance > 5000) | | (no_of_trans > 25) )

( !(grade == ’A’) )
```

The conditions evaluate to …

Zero

- Indicates FALSE.

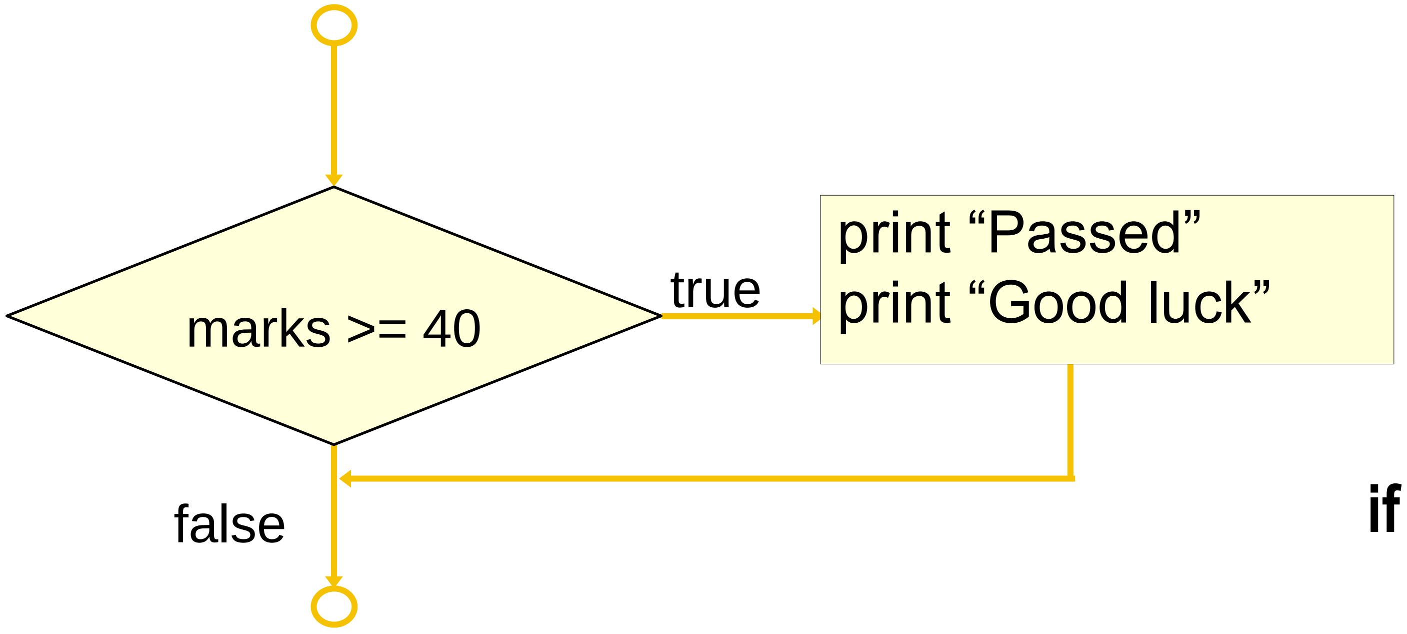
Non-zero

- Indicates TRUE.
- Typically the condition TRUE is represented by the value ‘1’.

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.



```
if (marks>=40)  {
   printf(“Passed \n”);
   printf(“Good luck\n”);
}
printf (“End\n”) ;
```

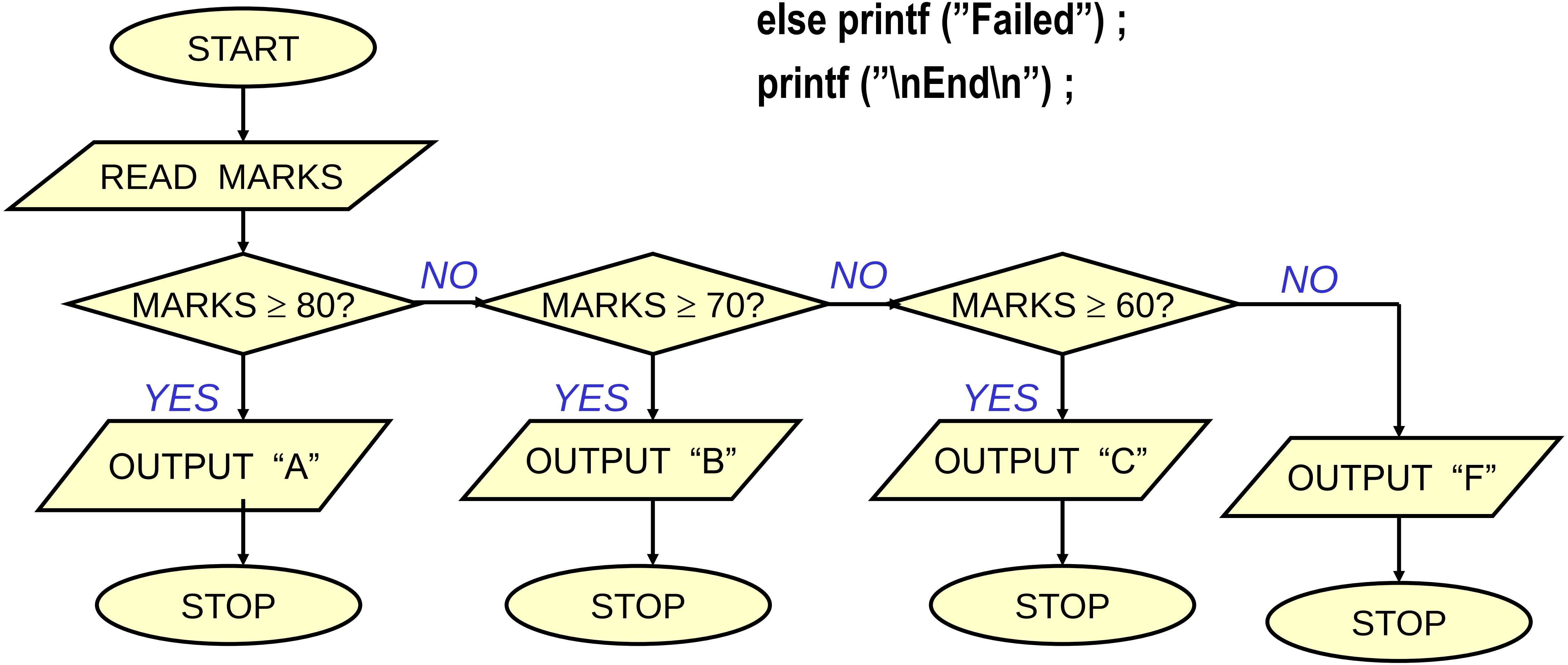
Branching: *if-else Statement*

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

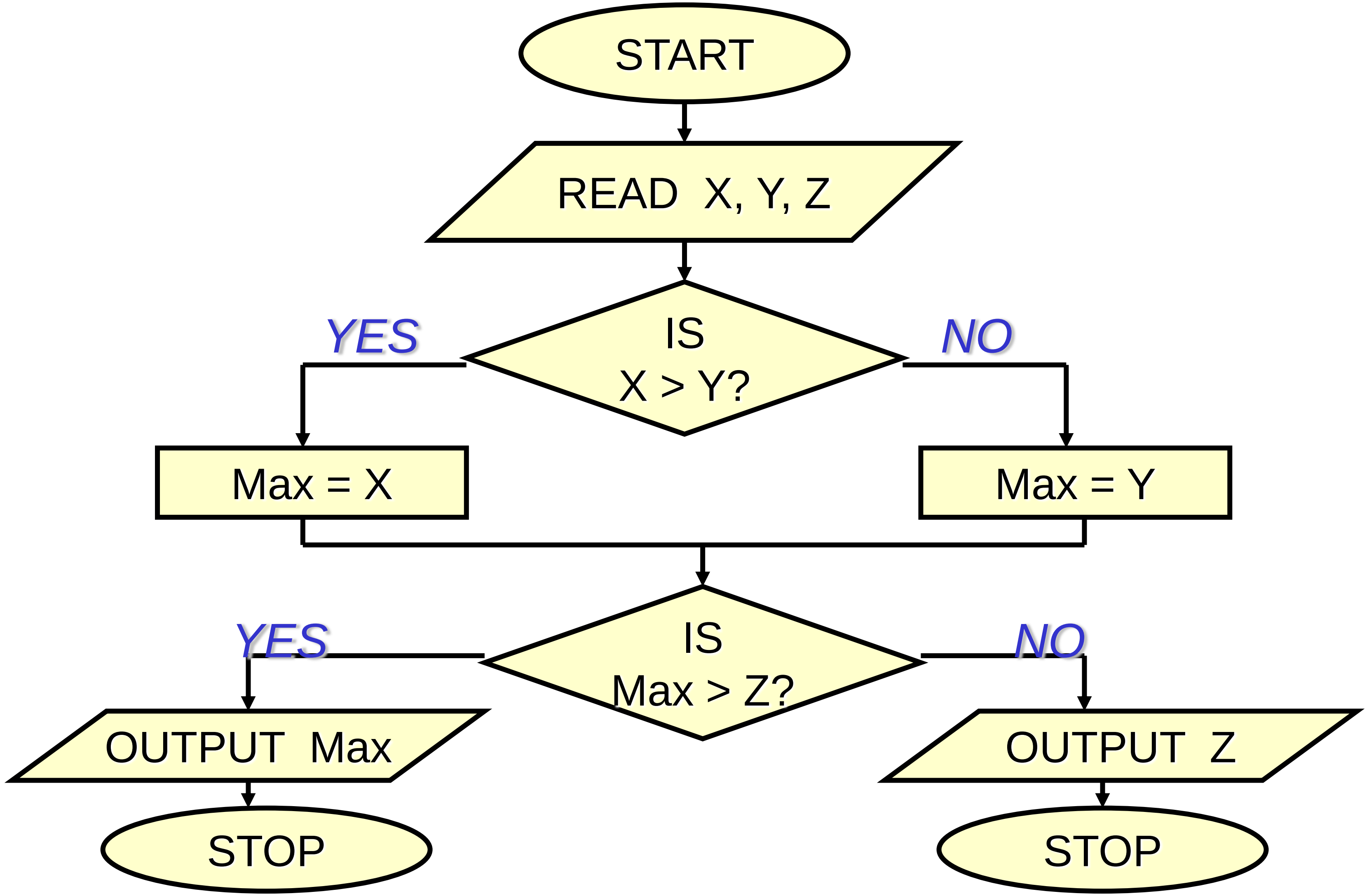
```
if (expression) {
    Block of statements;
}
else if  (expression) {
    Block of statements;
}
else {
    Block of statements;
}
```

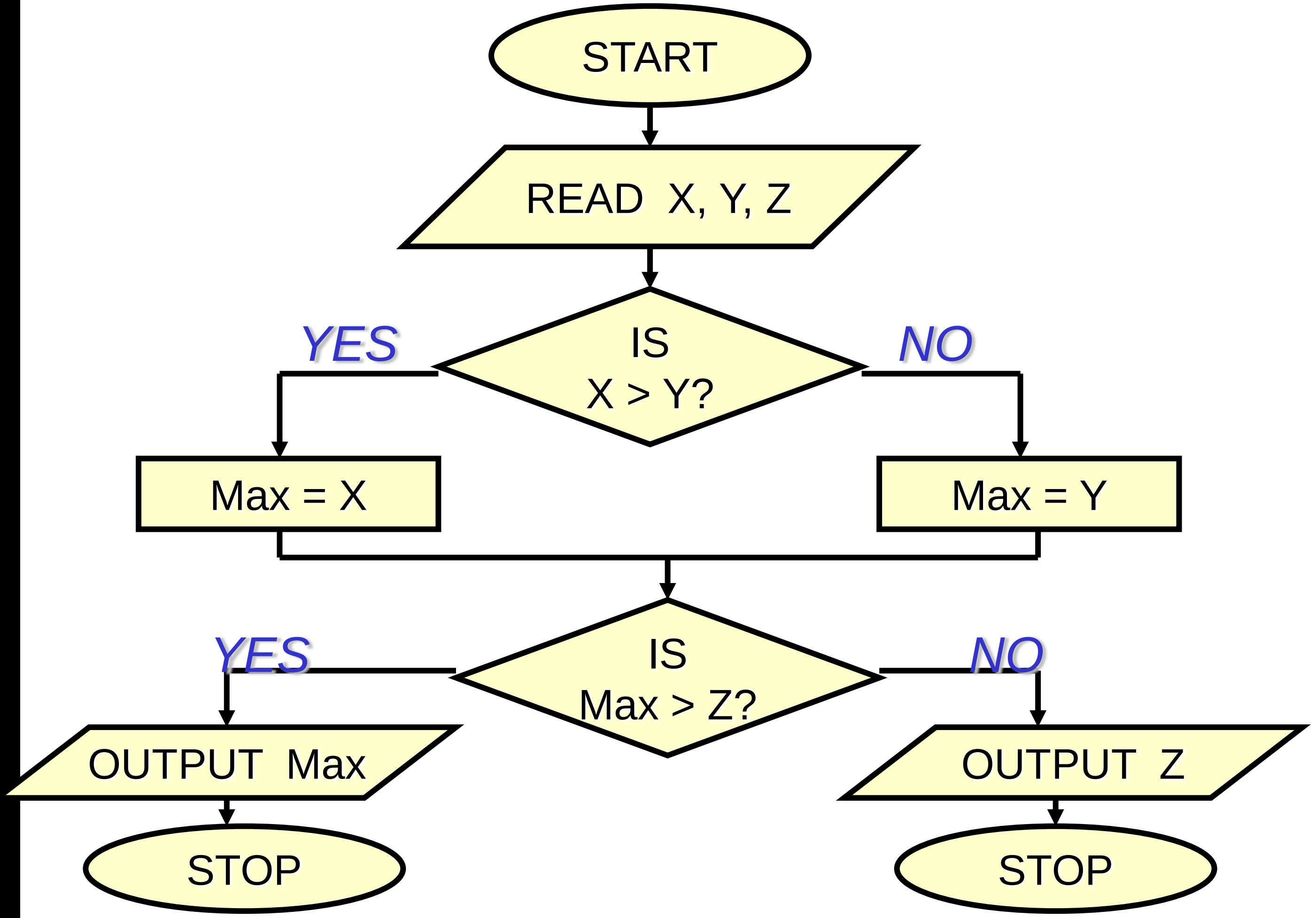
Grade Computation

```
if (marks >= 80) printf (”A”) ;
else if (marks >= 70) printf (”B”) ;
else if (marks >= 60) printf (”C”) ;
else printf (”Failed”) ;
printf (”\nEnd\n”) ;
```



Largest of three numbers





```
int main () {
      int x, y, z, max;
      scanf ("%d%d%d",&x,&y,&z);
      if (x>y)
            max = x;
      else max = y;
      if (max > z)
             printf ("%d", max) ;
      else printf ("%d",z);
}
```

Confusing Equality (==) and Assignment (=) Operators

Dangerous error

- Does not ordinarily cause syntax errors.
- Any expression that produces a value can be used in control structures.
- Nonzero values are true, zero values are false.

Example:

```
if ( payCode == 4 )
   printf( "You get a bonus!\n" );
```

```
if ( payCode = 4 )
   printf( "You get a bonus!\n" );        X
```

Dangling else problem

if (exp1) if (exp2) stmta else stmtb

```
if (exp1) {

    if (exp2)
           stmta
    else
           stmtb

}
```

OR

```
if (exp1)  {

    if (exp2)
           stmta
}
else
           stmtb
```

?

x

An “else” clause is associated with the closest preceding unmatched “if”.

Which one is the correct interpretation?

More examples

```
if e1 s1
else if e2 s2

if e1 s1
else if e2 s2
else s3

if e1 if e2 s1
else s2
else s3

if e1 if e2 s1
else s2
```

?

```
if e1 s1
else { if e2 s2 }

if e1 s1
else { if e2 s2
         else s3 }

if e1 { if e2 s1
         else s2 }
else s3

if e1 { if e2 s1
         else s2 }
```

Common Errors

```
c = getchar( );
if ((c == ’y’) && (c == ’Y’)) printf(“Yes\n”);
else printf(“No\n”);
```

```
c = getchar( );
if ((c != ’n’) || (c != ’N’)) printf(“Yes\n”);
else printf(“No\n”);
```

The Conditional Operator ?:

This makes use of an expression that is either true or false. An appropriate value is selected, depending on the outcome of the logical expression.

Example:

$$\text{interest =} \quad \underbrace{\text{(balance>5000) ? balance*0.2 : balance*0.1;}}_{\textit{Returns a value}}$$

Equivalent to:

```
if (balance > 5000)
          interest = balance * 0.2;
else      interest = balance * 0.1;
```

The *switch* statement

This causes a particular group of statements to be chosen from several available groups.

- **Uses “switch” statement and “case” labels.**
- **Syntax of the “switch” statement:**

```
switch (expression)  {
   case expression-1: { …….. }
   case expression-2: { …….. }

   case expression-m: { …….. }
   default: { ……… }
}
```

Examples

```
switch ( letter ) {
        case 'A':
                printf ("First letter \n");
                break;
        case 'Z':
                printf ("Last letter \n");
                break;
        default :
                printf ("Middle letter \n");
                break;
}
```

Will print this statement for all letters other than A or Z

Examples

```
switch ( choice = getchar( ) ) {
    case ‘r’ :
    case ‘R’: printf(“Red”);
                break;
    case ‘b’ :
    case ‘B’ : printf(“Blue”);
                 break;
    case ‘g’ :
    case ‘G’: printf(“Green”);
                 break;
    default: printf(“Black”);
}
```

Since there isnt a* break *statement here, the control passes to the next statement* (printf) *without checking the next condition.

Another way

```
switch  ( choice = toupper( getchar( ) ) )  {
        case ‘R’:           printf (“RED \n”);
                            break;
        case ‘G’:           printf (“GREEN \n”);
                            break;
        case ‘B’:           printf (“BLUE \n”);
                            break;
        default:  printf (“Invalid choice \n”);
}
```

Rounding a Digit

```
switch (digit)   {
                case 0:
                case 1:
                case 2:
                case 3:
                case 4: result = 0; printf (“Round down\n”); break;
                case 5:
                case 6:
                case 7:
                case 8:
                case 9: result = 10; printf(“Round up\n”); break;
}
```

A Look Back at Arithmetic Operators:

The Increment and Decrement

Increment (++) and Decrement (--)

Both of these are unary operators; they operate on a single operand.

The increment operator causes its operand to be increased by 1.

- Example: a++, ++count

The decrement operator causes its operand to be decreased by 1.

- Example: i--, --distance

Pre-increment versus post-increment

Operator written before the operand (++i, --i))

- Called pre-increment operator.
- Operator will be altered in value ***before*** it is utilized for its intended purpose in the program.

Operator written after the operand (i++, i--)

- Called post-increment operator.
- Operator will be altered in value ***after*** it is utilized for its intended purpose in the program.

Examples

Initial values :: a = 10; b = 20;

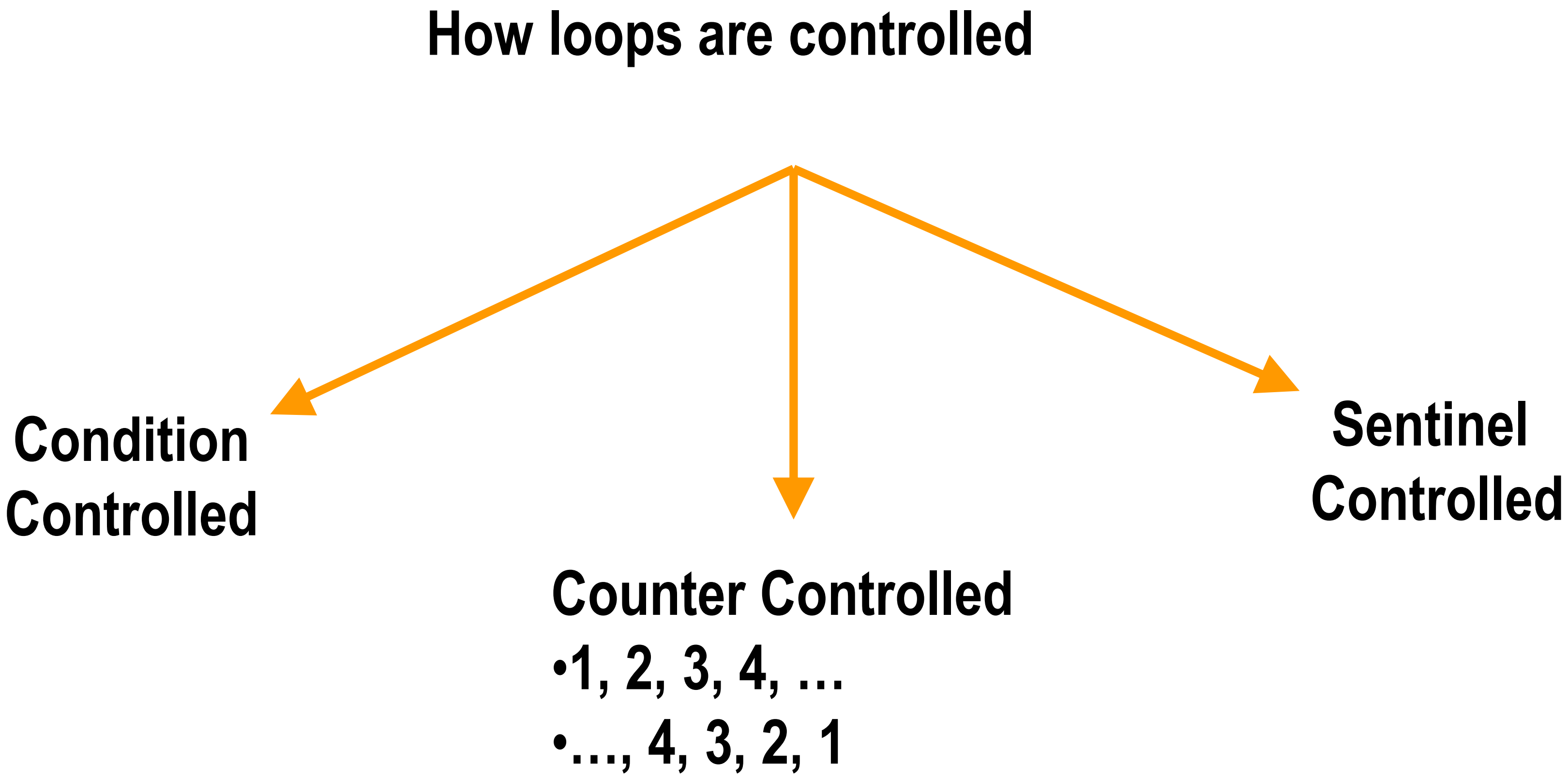
x = 50 + ++a;	a = 11, x = 61
x = 50 + a++;	x = 60, a = 11
x = a++ + --b;	b = 19, x = 29, a = 11
x = a++ – ++a;	??

Called side effects:: while calculating some values, something else get changed.

Looping Constructs

Types of Repeated Execution

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



How loops are controlled

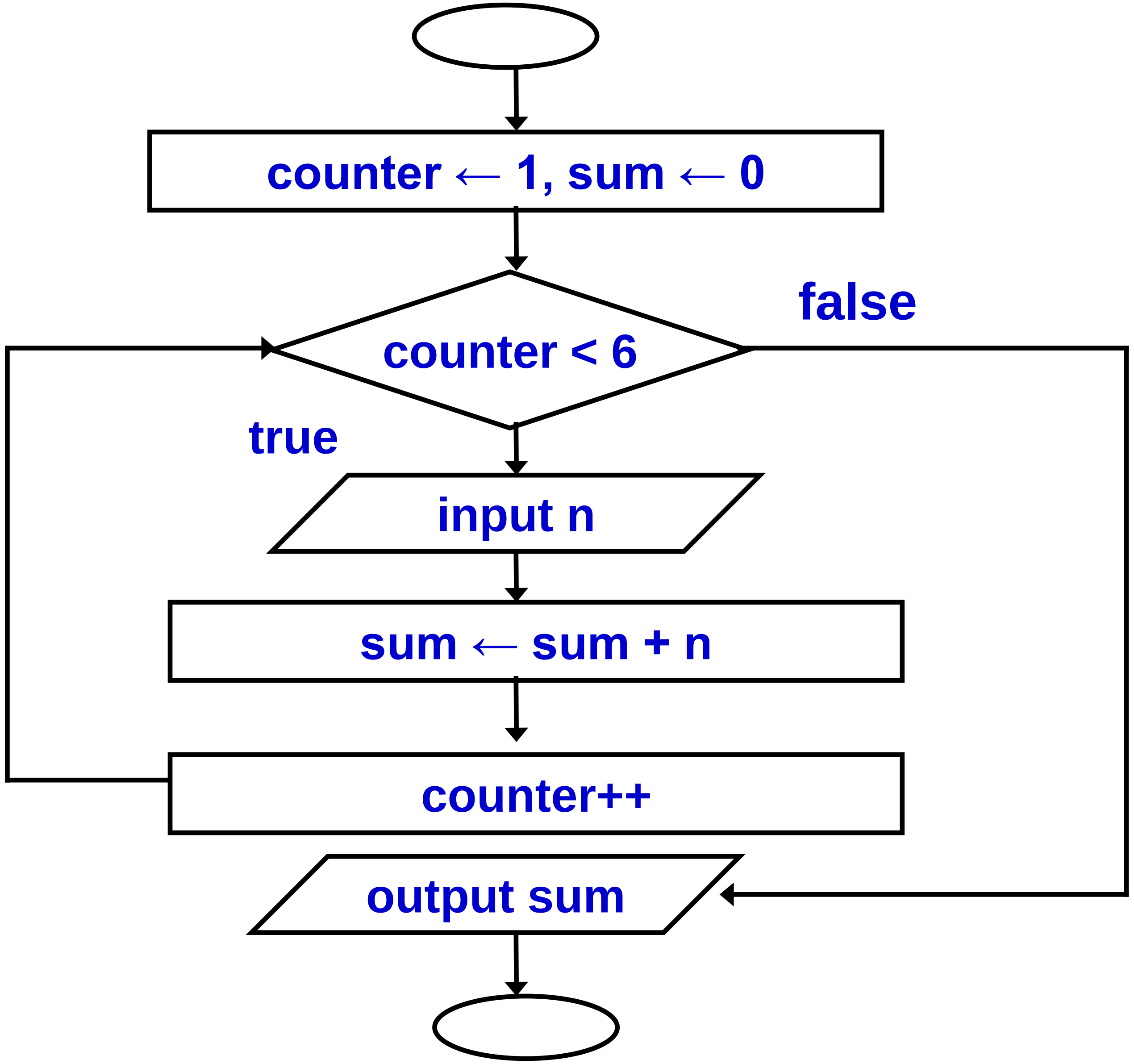
Condition Controlled

Counter Controlled
- 1, 2, 3, 4, …
- …, 4, 3, 2, 1

Sentinel Controlled

Counter Controlled Loop

Read 5 integers and display the value of their sum.

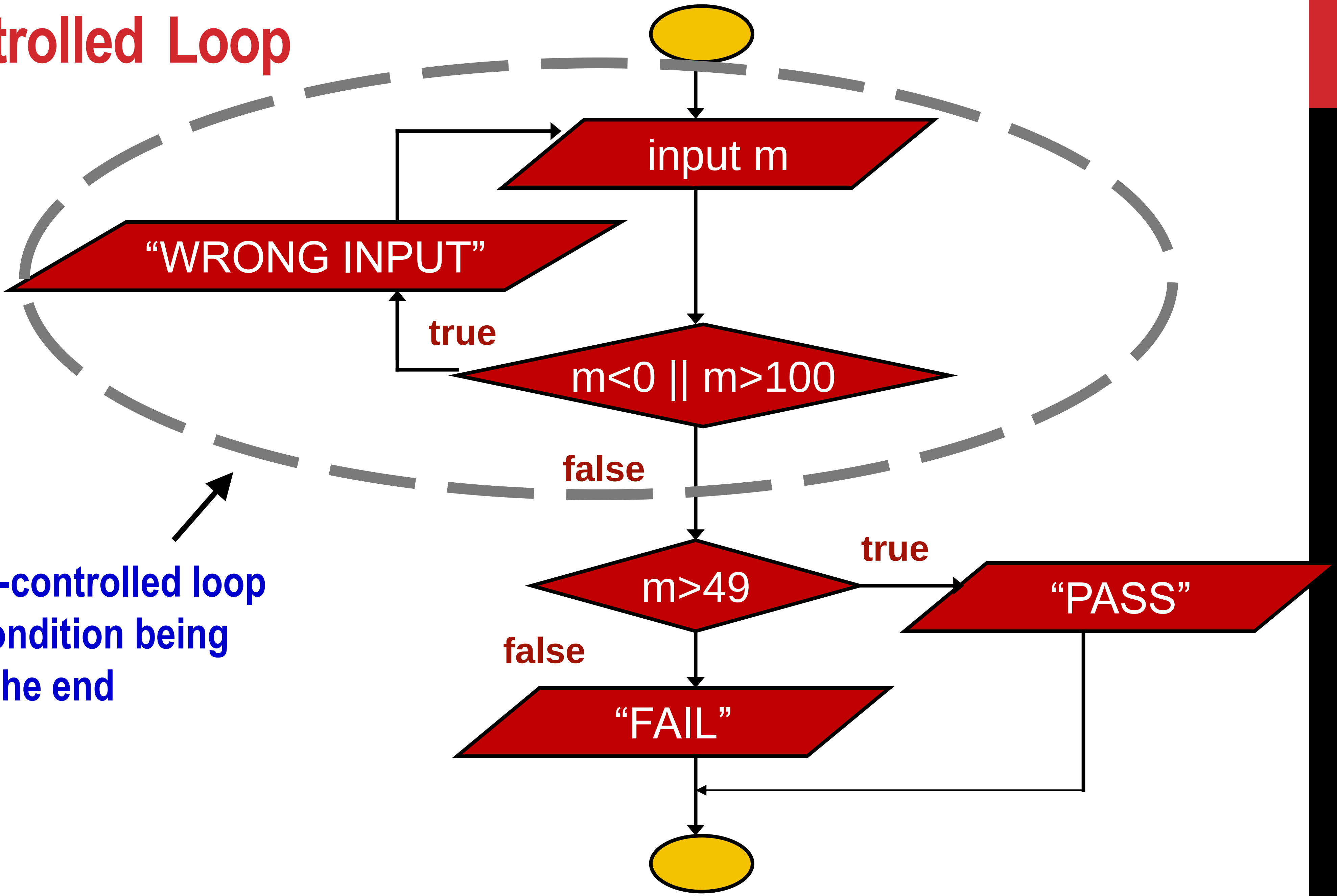


Condition-controlled Loop

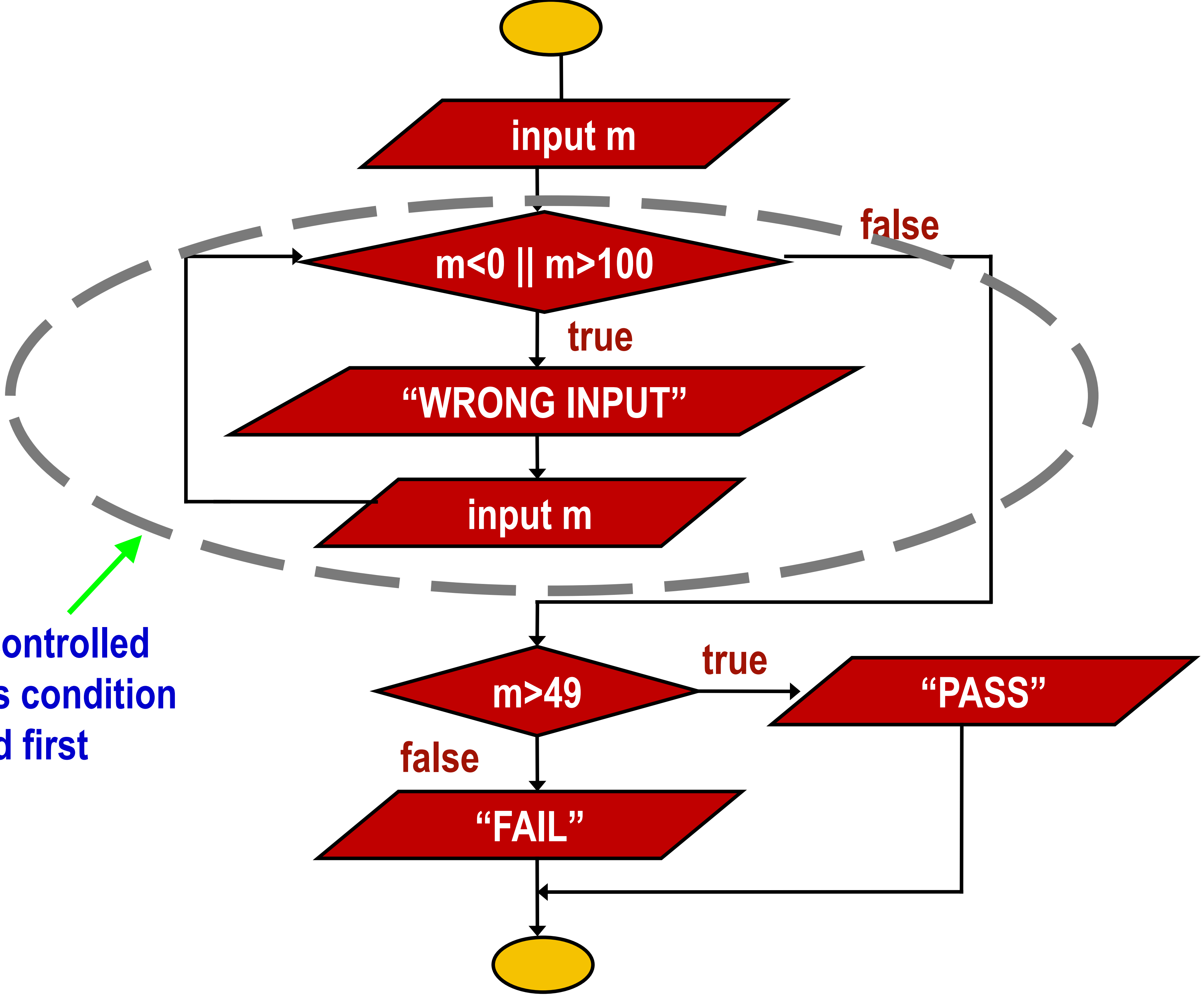
Given an exam marks as input, display the appropriate message based on the rules below:

- ❑ 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

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:

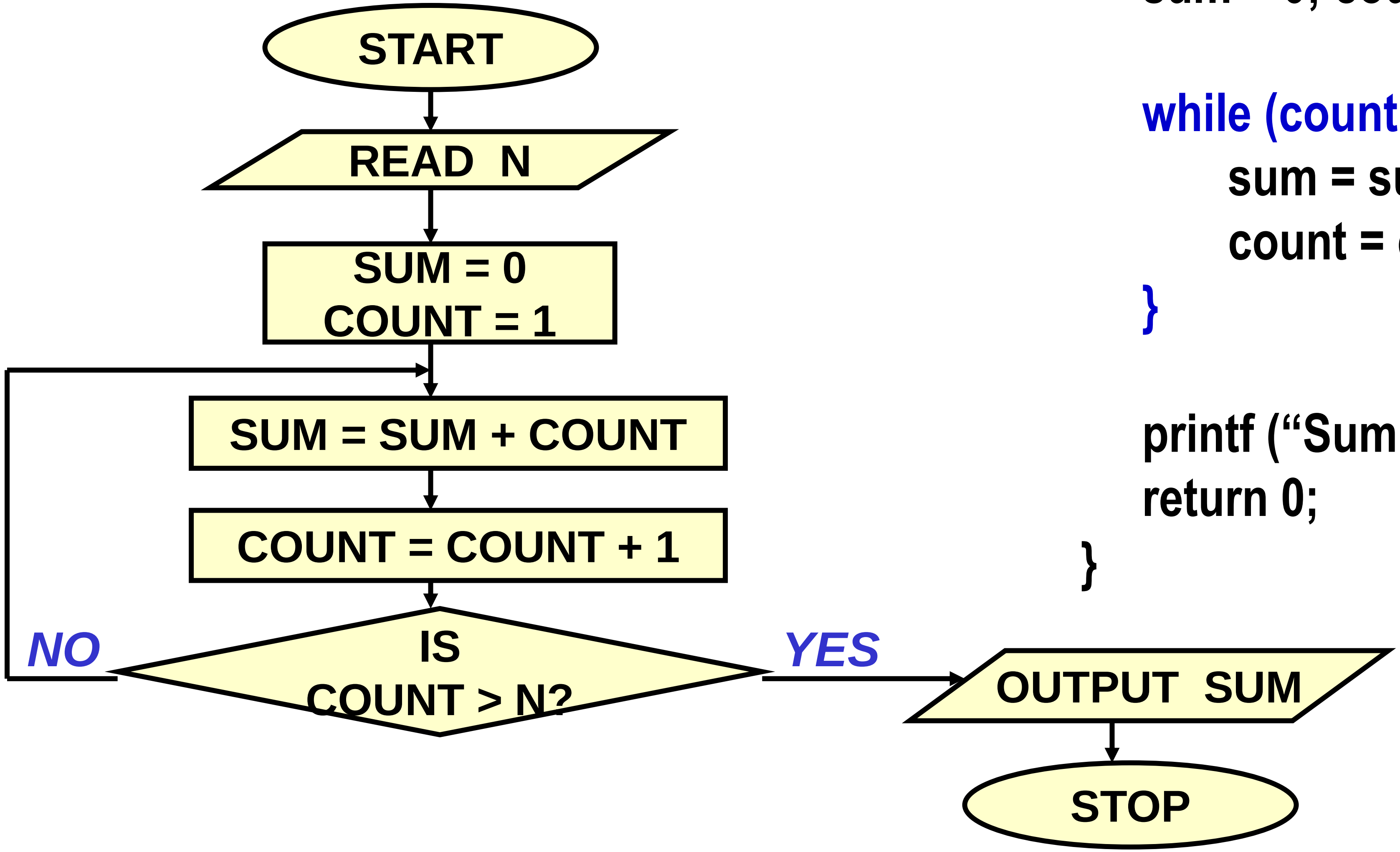
The while-loop will not be entered if the loop-control expression evaluates to false (zero) even before the first iteration.

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);
```

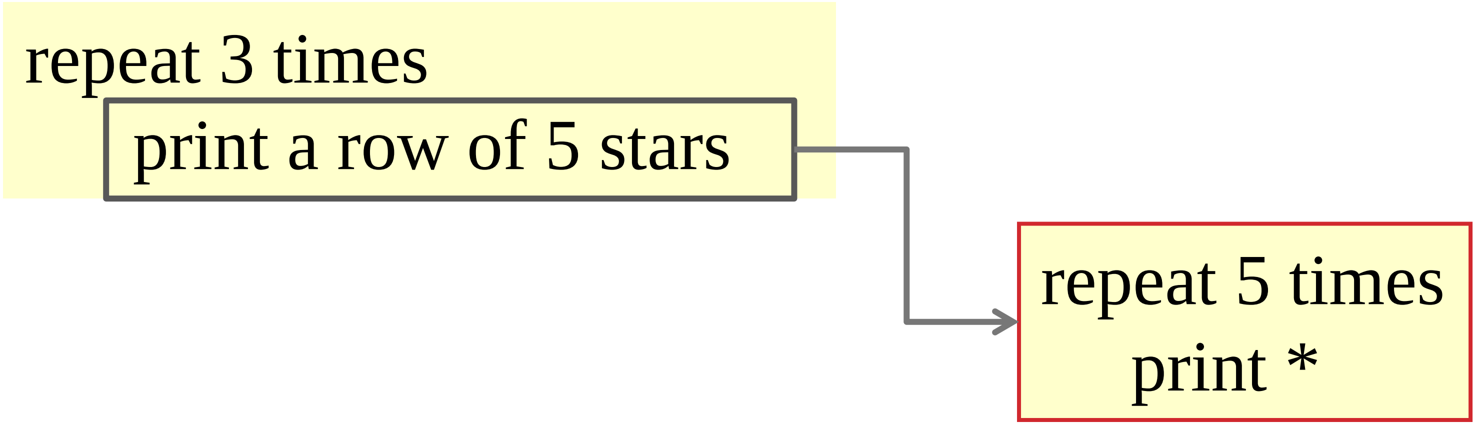
Maximum of inputs

```
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

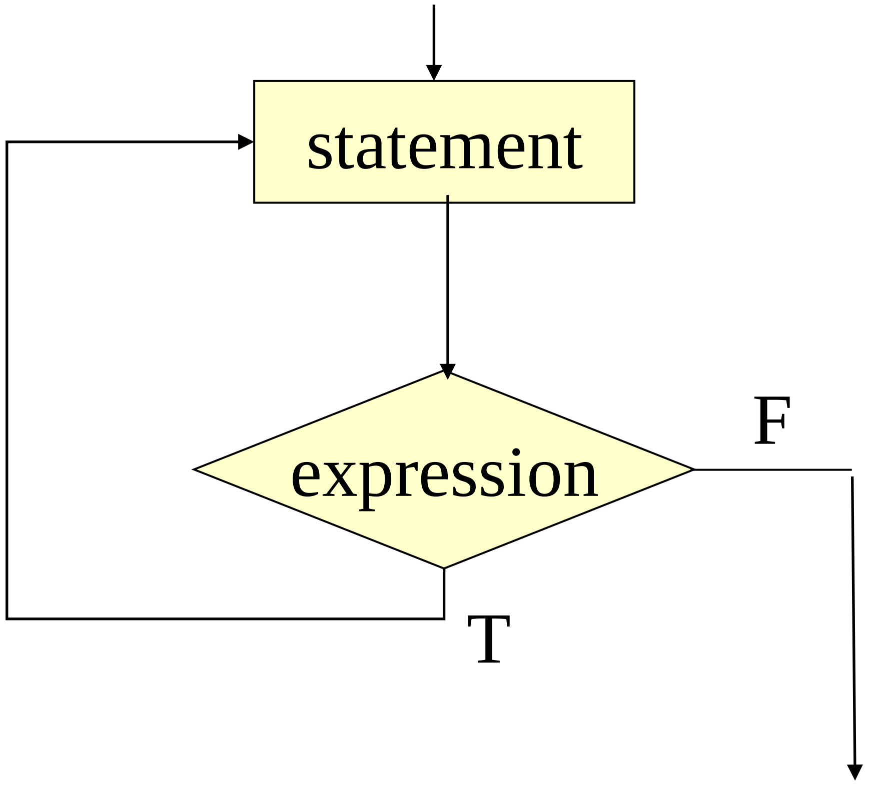
```
#define ROWS 3

#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”);
}
```

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;
```

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 ++ ;
            }
}
```

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

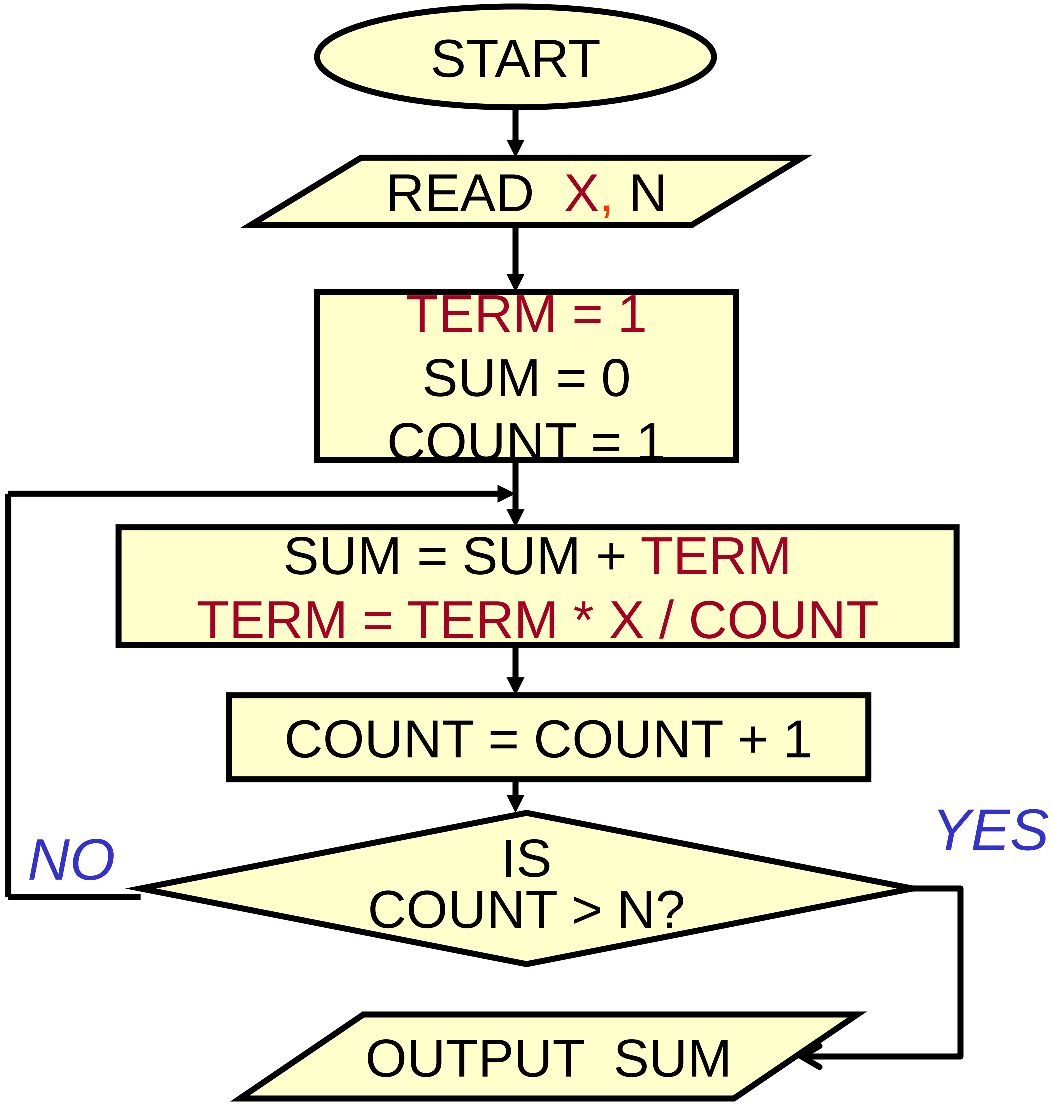
- *expression3* is evaluated, then *expression2* is evaluated.

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;
}
```

Some Examples

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



START

READ X, N

TERM = 1
SUM = 0
COUNT = 1

SUM = SUM + TERM
TERM = TERM * X / COUNT

COUNT = COUNT + 1

IS
COUNT > N?

NO

YES

OUTPUT SUM

```
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 ∗= 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 = 1.0;
        for (count = 1; term<0.0001; count++)  {
                term *= x/count;
                sum += term;
        }
        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?

Practice Problems

ISBN Numbers

Checking for Legal ISBN Numbers

10th	9th	8th	7th	6th	5th	4th	3rd	2nd	1st
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 , D_1 ,..., D_{10} where D_1 is a checksum between 1 and 10.
  - If D_1 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 * $D_9$ + … +  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 = 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  checksum = '0‘ + 11 – (sum%11) ;
            printf(“Checksum digit = %c\n”, checksum);
}
```

BISECTION METHOD FOR ROOT FINDING

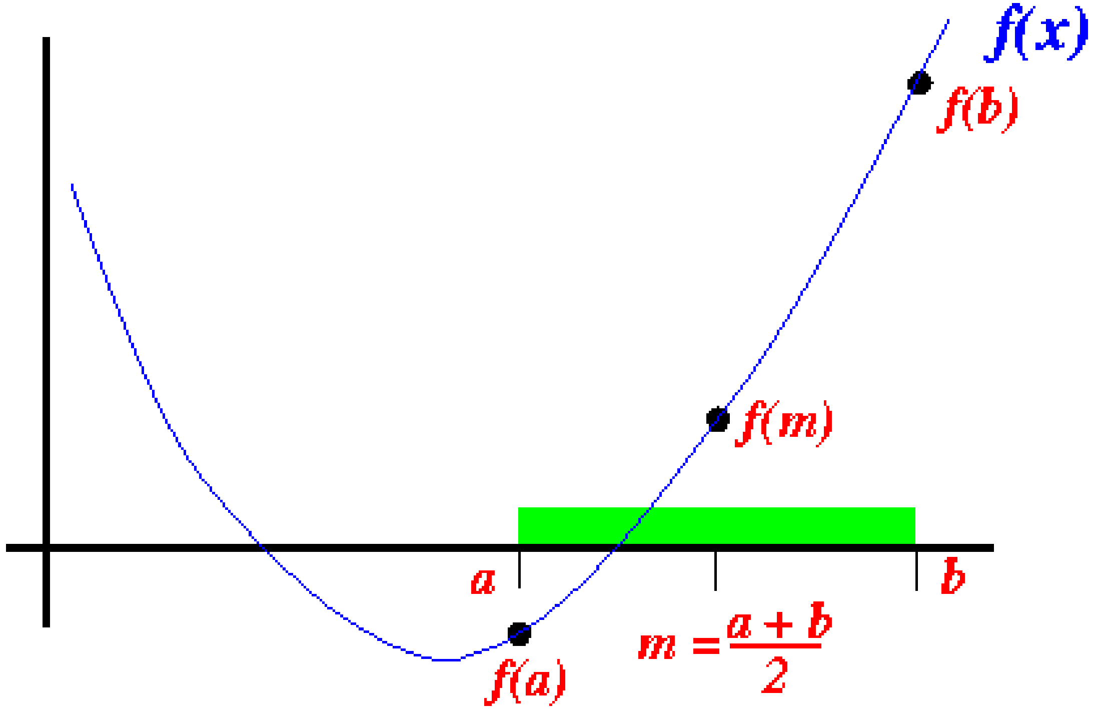
A method for finding the root of a function

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 );
}
```