



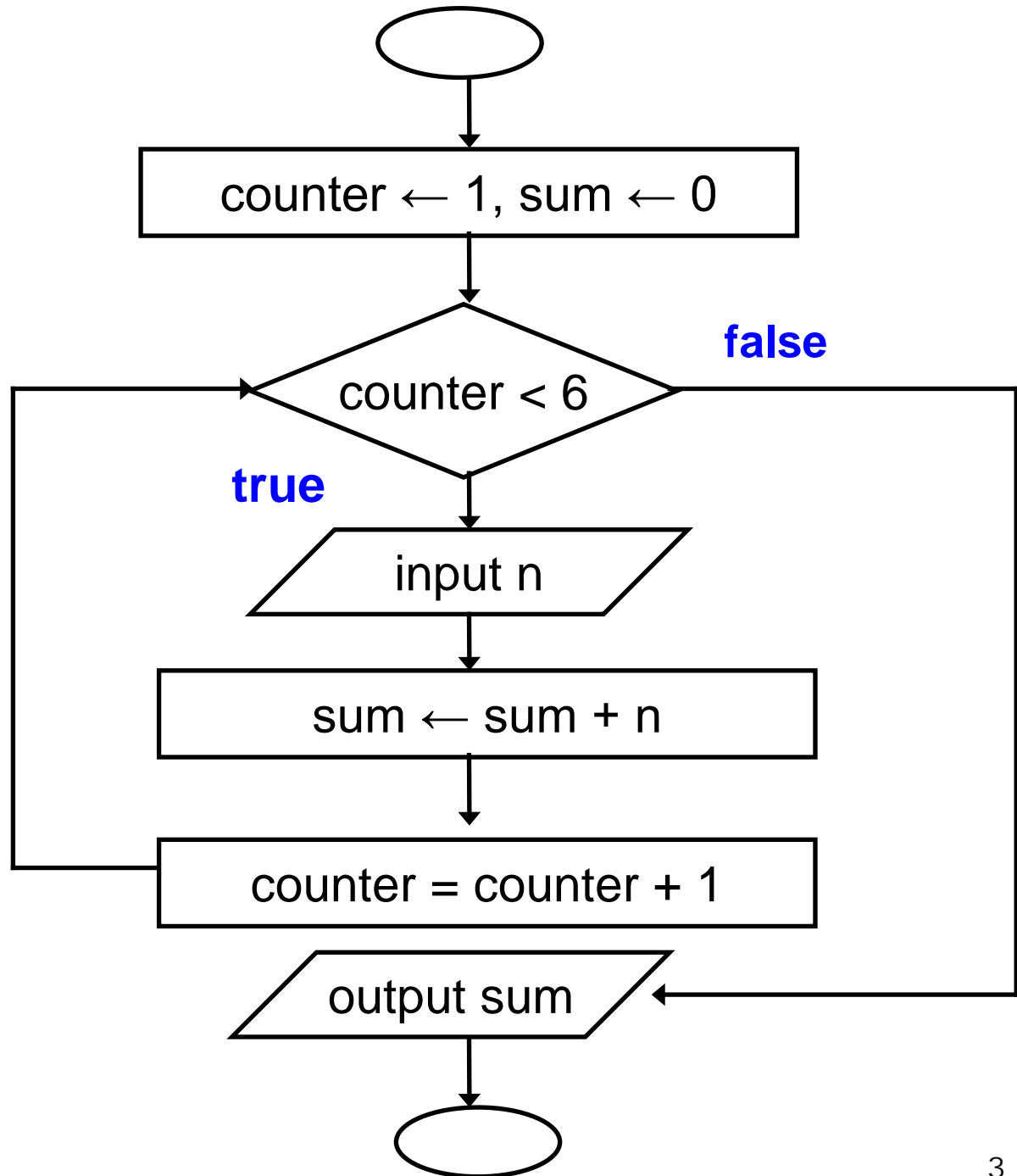
Looping

Loops

- Group of statements that are executed repeatedly while some condition remains true
- Each execution of the group of statements is called an **iteration** of the loop

Example

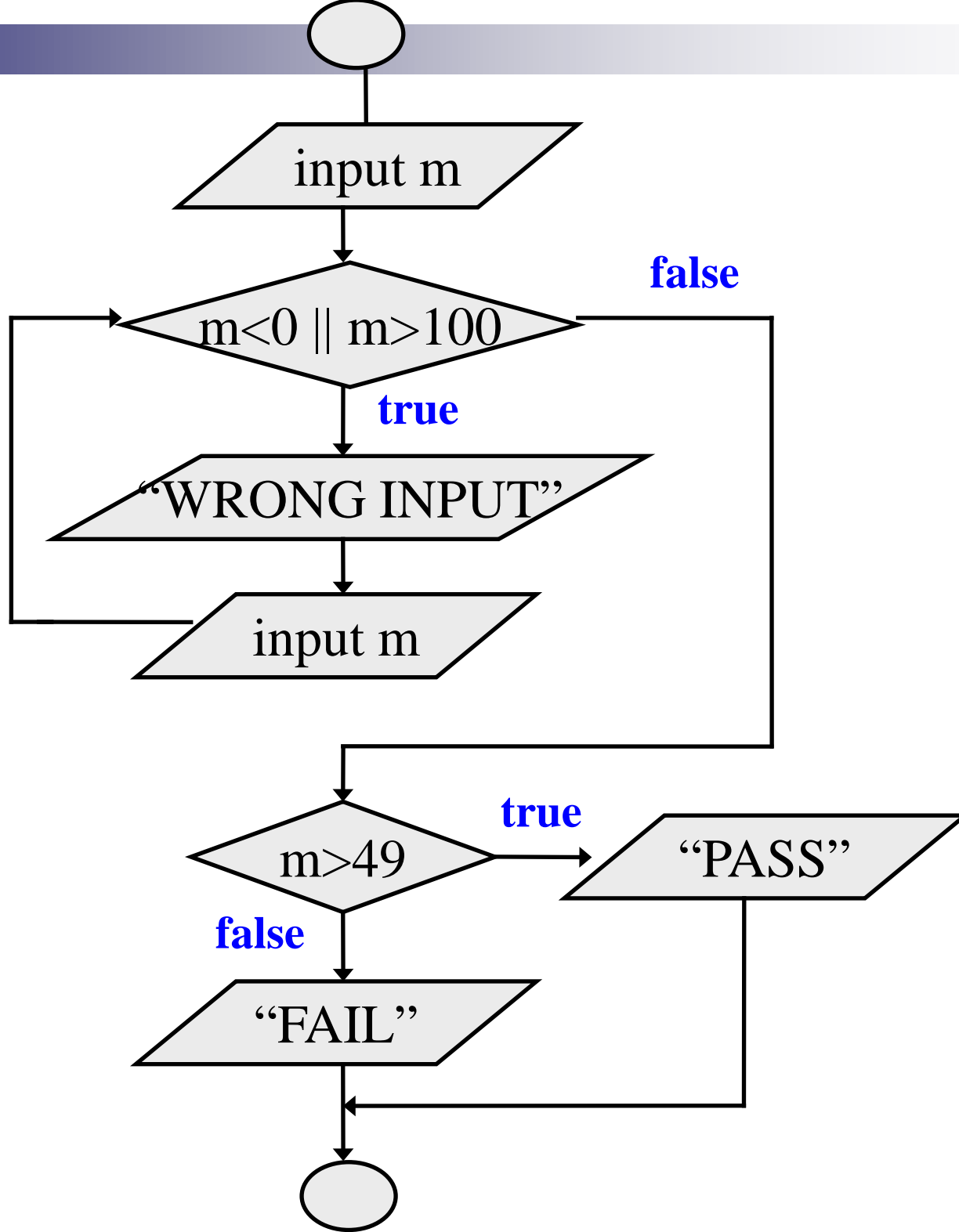
Read 5 integers and display the their sum

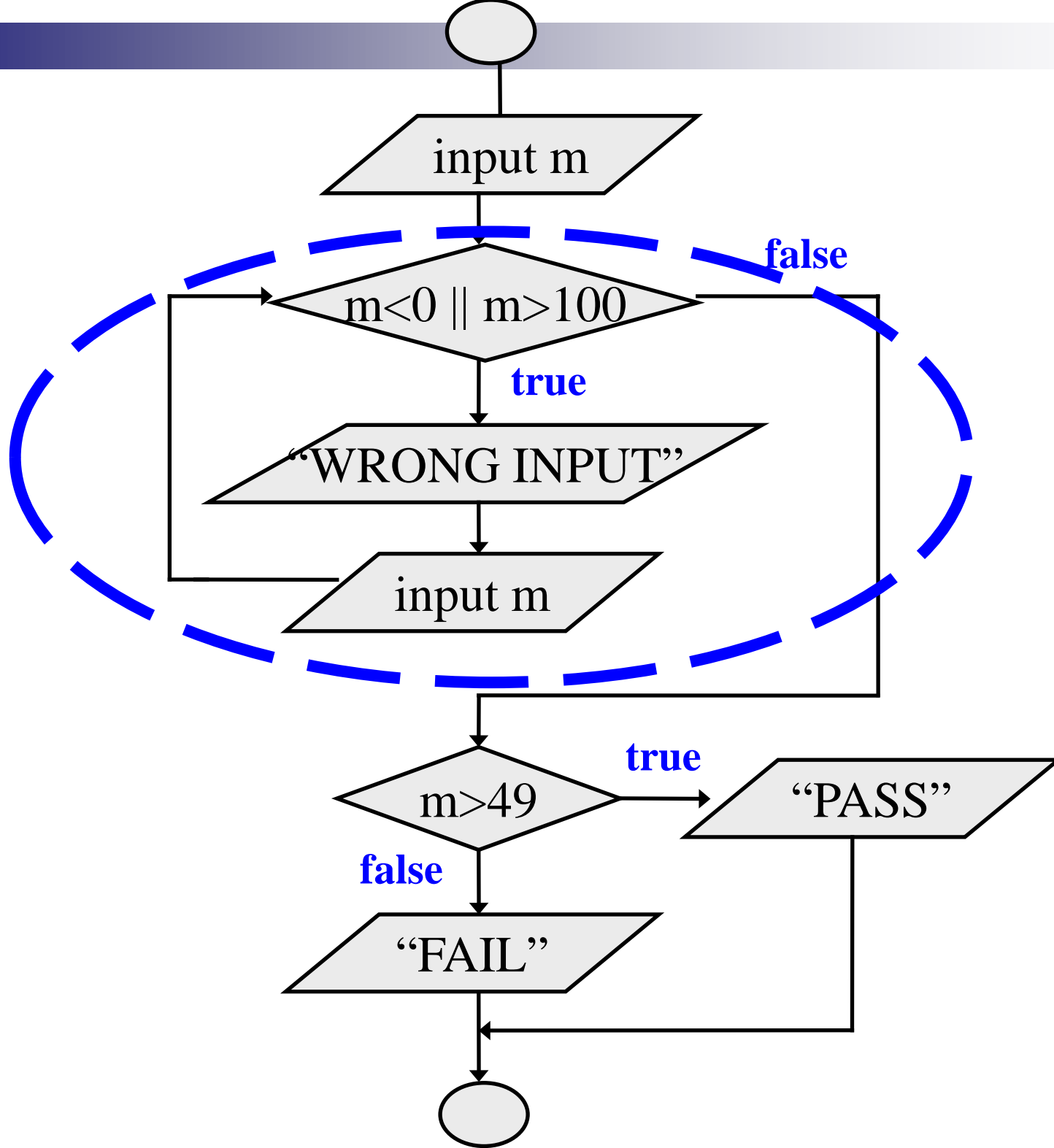


Example

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





Looping: **while** statement

```
while (expression)  
    statement;
```

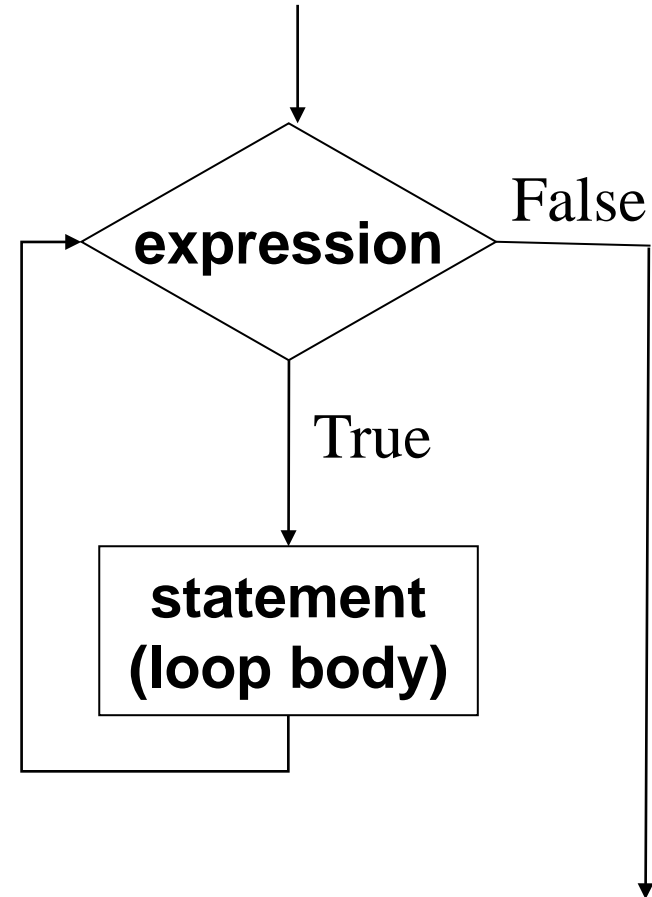
```
while (expression) {  
    Block of statements;  
}
```

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. Then the expression is evaluated again and the same thing repeats. The loop **terminates** when the expression evaluates to 0.

Looping: **while** statement

```
while (expression)  
    statement;
```

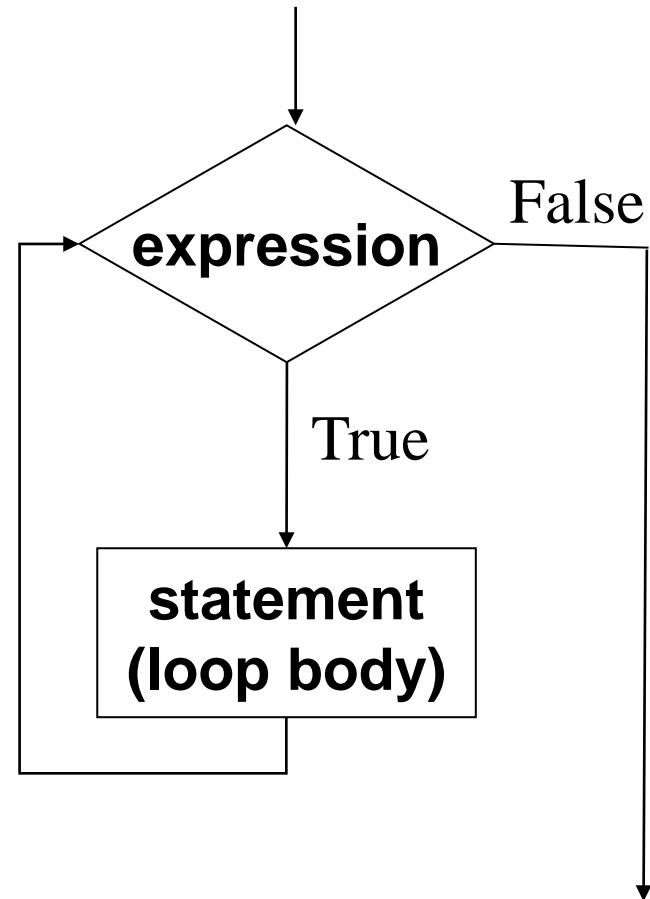
```
while (expression) {  
    Block of statements;  
}
```



Looping: **while** statement

```
while (expression)  
    statement;
```

```
while (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. Then the expression is evaluated again and the same thing repeats. The loop **terminates** when the expression evaluates to 0.

Example


```
int i = 1, n;  
scanf("%d", &n);  
while (i <= n) {  
    printf ("Line no : %d\n",i);  
    i = i + 1;  
}
```

Example

```
int weight;  
scanf("%d", &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

```
void 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) ;  
}
```


$$\text{SUM} = 1^2 + 2^2 + 3^2 + \dots + N^2$$

```
void 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;  
}
```

Compute GCD of two numbers

```
void main() {
    int A, B, temp;
    scanf ("%d %d", &A, &B);
    if (A > B) {
        temp = A; A = B; B = temp;
    }
    while ((B % A) != 0) {
        temp = B % A;
        B = A;
        A = temp;
    }
    printf ("The GCD is %d", A);
}
```

$$\begin{array}{r} 12 \) \ 45 \ (\ 3 \\ \underline{36} \\ 9 \) \ 12 \ (\ 1 \\ \underline{9} \\ 3 \) \ 9 \ (\ 3 \\ \underline{9} \\ 0 \end{array}$$

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

Double your money

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

```
void main() {  
    double my_money = 10000.0;  
    int 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 positive Numbers

```
void main() {
    double max = 0.0, next;
    printf ("Enter positive numbers, end with 0 or a
negative number\n");
    scanf("%lf", &next);
    while (next > 0) {
        if (next > max) max = next;
        scanf("%lf", &next);
    }
    printf ("The maximum number is %lf\n", max) ;
}
```


Find the sum of digits of a number

```
void 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);
}
```

Looping: **for** Statement

- Most commonly used looping structure in C

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

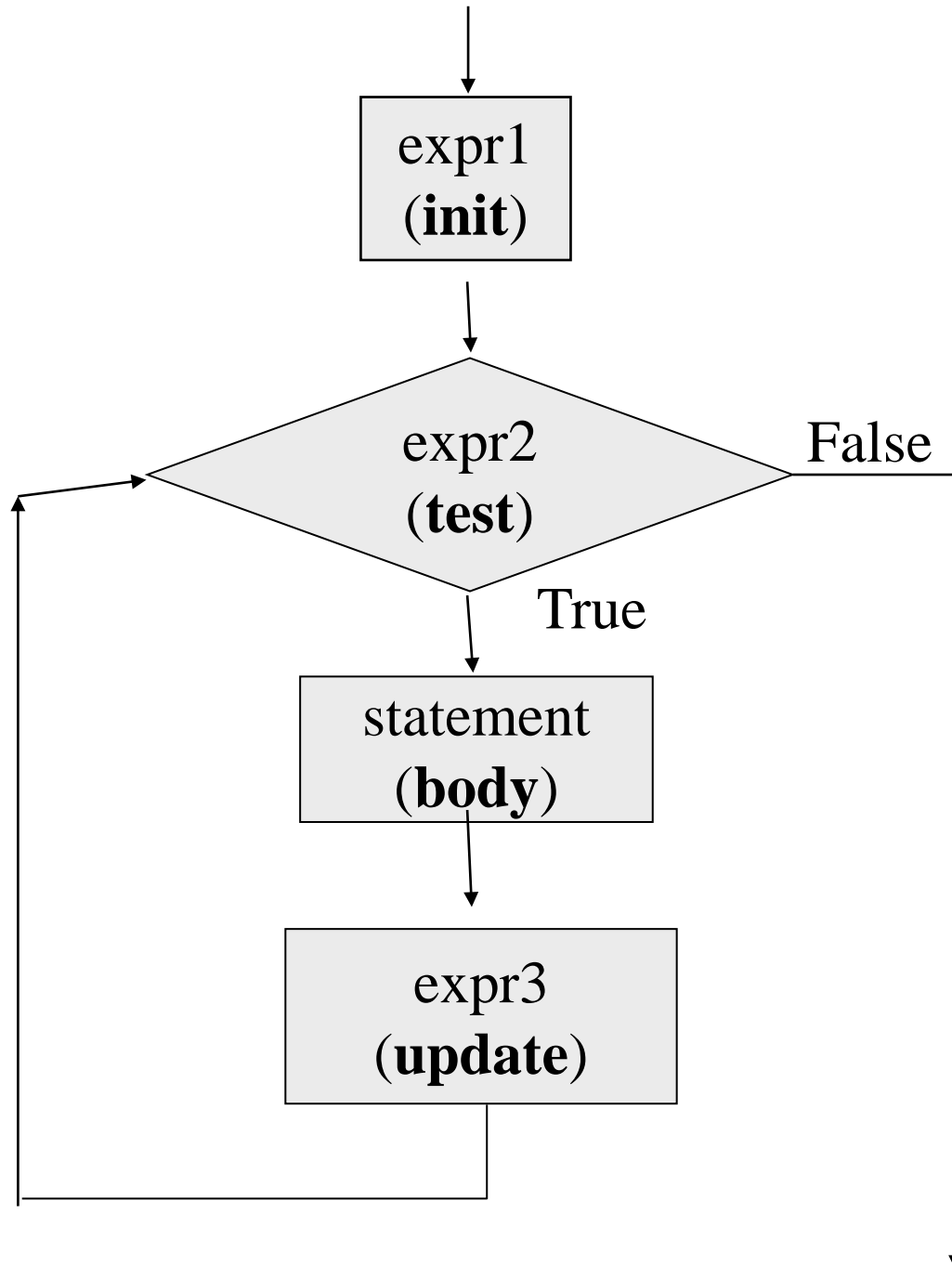
```
for ( expr1; expr2; expr3)  
{  
    Block of statements;  
}
```

expr1 (init) : initialize parameters

expr2 (test): test condition, loop continues if expression is non-0

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

statement (body): body of loop



Example: Computing Factorial

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

Computing e^x series up to N terms

```
void main () {
    float x, term, sum;
    int n, count;
    scanf ("%f", &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 series up to 4 decimal places

```
void main () {  
    float x, term, sum;  
    int cnt;  
    scanf ("%f", &x) ;  
    term = 1.0; sum = 0;  
    for (cnt = 1; term >= 0.0001; ++cnt) {  
        sum += term;  
        term *= x/cnt;  
    }  
    printf ("%f\n", sum) ;  
}
```

Equivalence of **for** and **while**

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

Same as



```
expr1;  
while (expr2) {  
    statement  
    expr3;  
}
```

Sum of first N Natural Numbers

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

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


Some observations on **for**

- Initialization, loop-continuation test, and update can contain arithmetic expressions

```
for ( k = x; k <= 4 * x * y; k += y / x )
```

- Update may be negative (decrement)

```
for (digit = 9; digit >= 0; --digit)
```

- If loop continuation test is initially 0 (**false**)

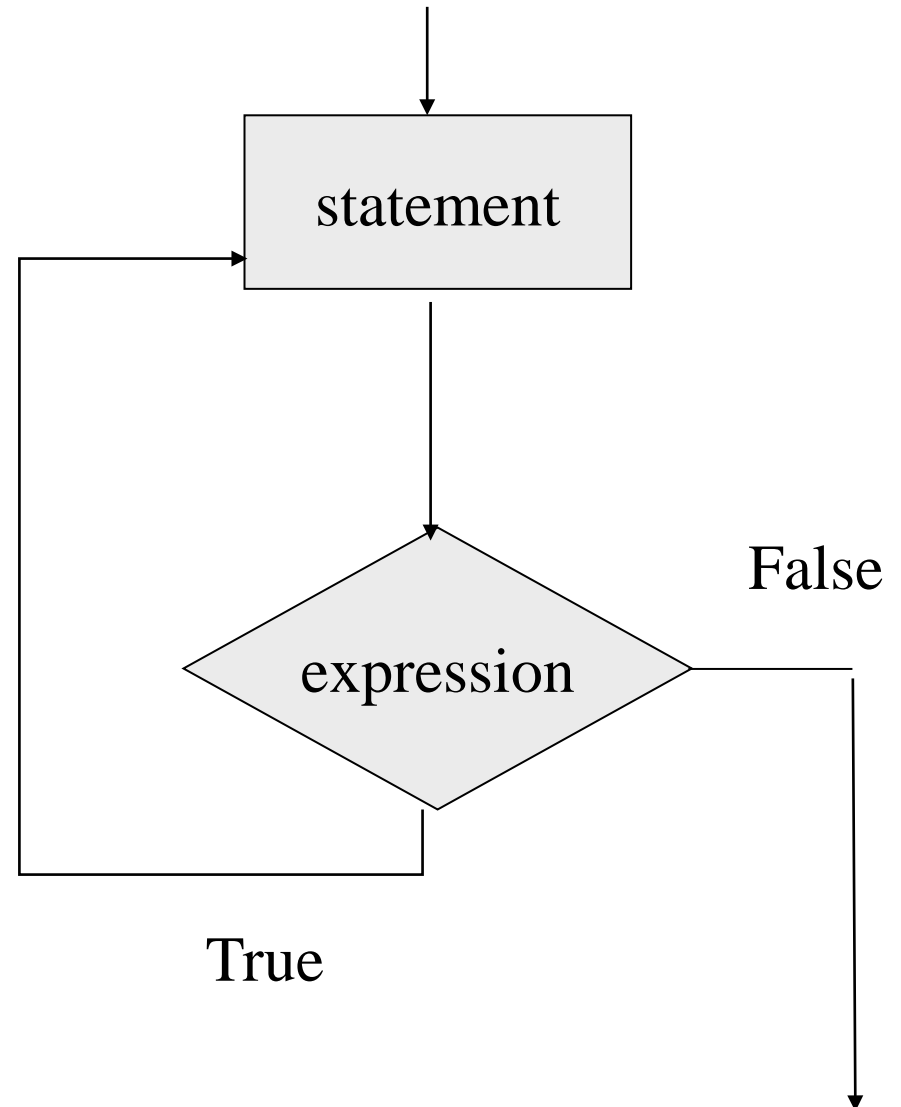
- Body of **for** structure not performed

- No statement executed

- Program proceeds with statement after **for** structure

Looping: **do-while** statement

```
do  
    statement;  
while (expression);  
  
do {  
    Block of statements;  
} while (expression);
```



Example

Problem: Prompt user to input “month” value, keep prompting until a correct value of month is given as input

```
do {  
    printf (“Please input month {1-12}”);  
    scanf (“%d”, &month);  
} while ((month < 1) || (month > 12));
```

Decimal to binary conversion (prints binary in reverse order)

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

Echo characters typed on screen until end of line

```
void main () {  
    char echo ;  
    do {  
        scanf ("%c", &echo);  
        printf ("%c",echo);  
    } while (echo != '\n') ;  
}
```

Specifying “Infinite Loop”

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

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

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

The **break** Statement

- Break out of the loop body { }
 - 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

An Example

```
void 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

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

More efficient??

```
void 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);
}
```

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
 - Loop-continuation test is evaluated immediately after the continue statement is executed
- for loop
 - **expr3** is evaluated, then **expr2** is evaluated

An Example with **break** and **continue**

```
void main() {
    int fact = 1, i = 1;
    while (1) {
        fact = fact * i;
        ++i;
        if ( i <=10 )
            continue; /* not done yet ! Go to loop and
                        perform next iteration*/
        break;
    }
}
```

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);
```

Nested Loops: Printing a 2-D Figure

- How would you print the following diagram?

* * * * *

* * * * *

* * * * *

repeat 3 times

print a row of 5 *'s

repeat 5 times
print *

Nested Loops

```
const int ROWS = 3;
const int COLS = 5;
...
row = 1;
while (row <= ROWS) {
    /* print a row of 5 '*'s
    */
    ...
    ++row;
}
```

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

**outer
loop**

**inner
loop**

2-D Figure: with for loop

Print

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

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


Another 2-D Figure

Print

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

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

Yet Another One

Print

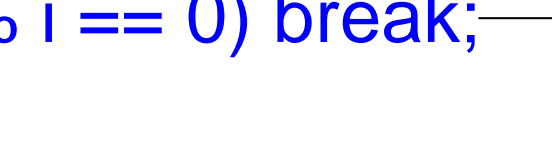
```
* * * * *
 * * * *
  * * *
   * *
    *
```

```
const int ROWS = 5;
....
int row, col;
for (row=0; row<ROWS; ++row) {
    for (col=1; col<=row; ++col)
        printf(" ");
    for (col=1; col<=ROWS-row; ++col)
        printf("* ");
    printf ("\n");
}
```

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

```
void 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;
    }
}
```

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;
```