Chapter 05
Iteration Structure

Dr. Le Thanh Sach

Nội dung

- Iteration structure
- `for` statement
- `while` statement
- `do…while` statement
- The role of algorithms in problem-solving
- Some common mistakes when using loops
- Exercise

The use of iteration structure

- Types of controls
  - Sequence:
    - The nature of the program is sequential. The program will execute statements one by one.
  - Branching:
    - Is used to choose to execute some statements
    - We learned this in the previous chapter.
  - Iteration (loop):
    - Execute a task (with parameters) many times.

The use of iteration structure

- Why is iteration structure used a lot?
  - Data processing
    - In reality, there is too much data
    - The program that handles the data must have access to all of the data
      - Access all or a group of student
        - In student management
      - Access all or a group of product
        - In product management
      - Access all or a group of pixels
        - In pixel processing
      - Access all or a group of friend on the social network.
        - In facebook

The use of iteration structure

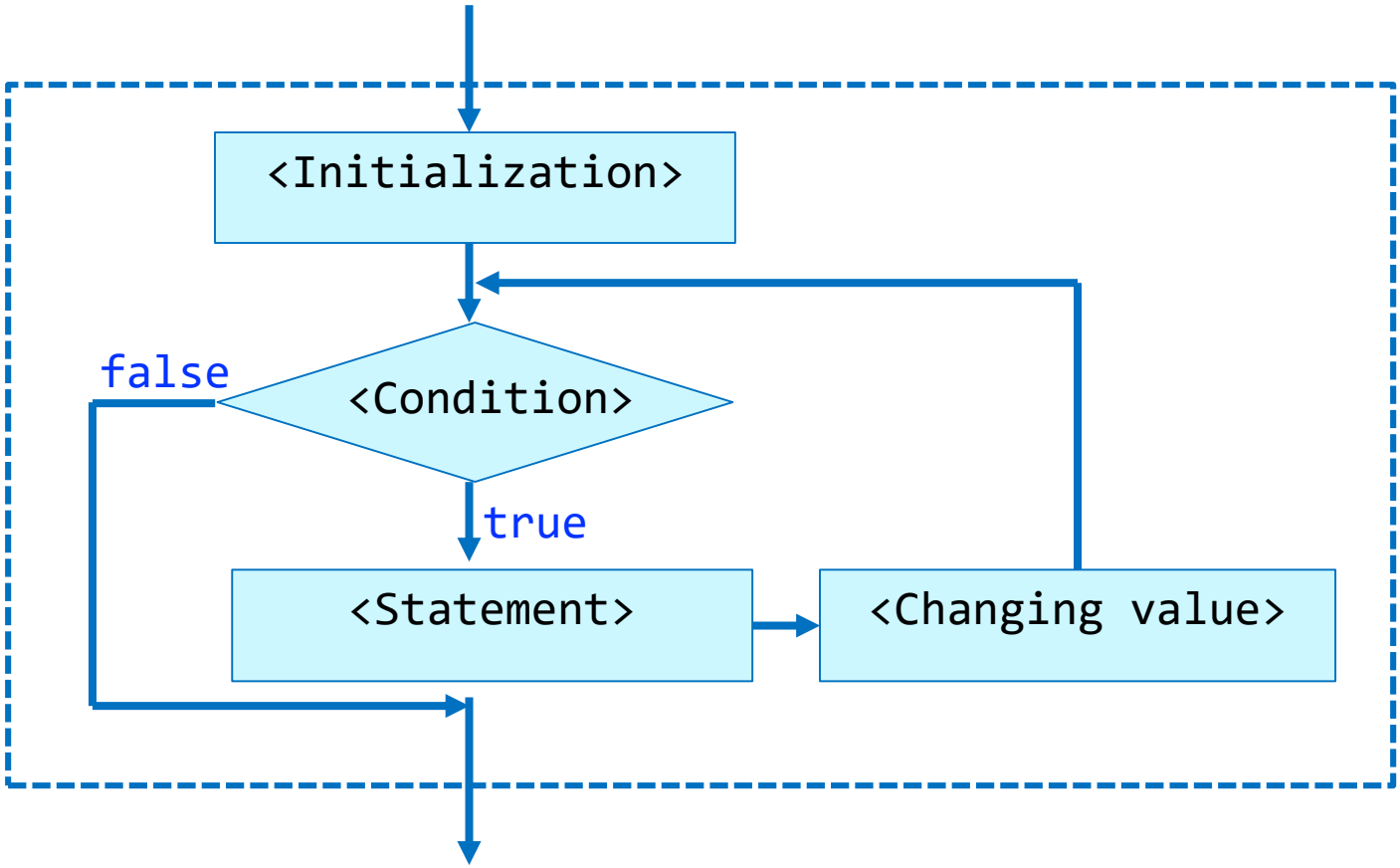
- Why is iteration structure used a lot?
  - Data processing
  - Algorithm
    - Many algorithms, in reality, need iteration structure
      - The problem of approximating nonlinear functions such as: sin(x), cos(x), etc.
      - Finding the solutions of non-analytic equations
      - Etc.

`for` statement

- Typical problems to use for statement.
  - Very well suited to the problem that needs to repeat with a determined number of iterations (This number is an integer)
    - Many technical problems use arrays to store data
      - We will learn about array in the next chapter
    - To process array data (browse through the elements), the for structure is the best fit
  - for statement, when combined with `break` statement, is also possible to stop the loop statement
    - for is also used with other iteration types.

for statement

Concept



for statement

Concept

- <Initialization>
  - Uses:
    - Variable declaration: used only in the iteration statements
    - Initialize variables that controls the loop
  - Quantity:
    - None, one or many variables (of the same type) is declared and intialized
    - The initialization is separated by commas: ,
- <Condition>
  - Uses:
    - To check the stop condition of the loop
  - Quantity
    - None, one or many boolean expressions or expressions that can be transformed to boolean
    - Expressions are separated by commas
    - If there is no expression, the condition is assumed to be true. At that point, the stop condition should be put inside the loop

for statement

Concept

- <Changing value>
  - Uses
    - To change the value of control variables
      - The loop only stops when the conditional expression, evaluated based on these control variables, is `true`
  - Quantity
    - None, one or many statements to change the value of control the variables
    - Separated by commas
- <Statement>
  - Any single or composite statement

`for` statement

Concept

- Principles of execution
  - (1) The program will declare and initialize variables in the <`Initialization`> scope and check the conditional expression
  - (2) If <`Condition`> is `true`
    - Execute statements in <`Statement`>
    - Make changes in <`Changing value`>
    - Re-check the condition in Step (2)
  - (3) Else
    - Go to the statement after this loop

for statement

Syntax

```
for (<Initialization>; <Condition>; <Changing value>)
    <Statement>
```

Implement case for complex statement

```
for (<Initialization>; <Condition>; <Changing value>){
    <statement 1>
    <statement 2>
    <statement N>
}
```

```
for (<Initialization>; <Condition>; <Changing value>)
{
    <statement 1>
    <statement 2>
    <statement N>
}
```

for statement

Syntax

- Note about the syntax
  - Between the pair of brackets ( ) of for.
    - There are always 2 semiconlons (;) that divide into 3 scopes
      - Initialization
      - Conditional expression
      - Change value
    - All three scopes may be empty, in this case the loop will go on forever unless there is a break statement inside it:

```
for(;;){
        //statement
}
```

`for` statement

Syntax

- Notes about the syntax
  - Variables that ís initialize in **`for`** loop.
    - Can be used only in **`for`** loop
    - Not visible and unusable in the statements after **`for`** loop
  - **`break;`** statement
    - When the program see a **`break`**; statement in the **`for`** loop, the program will exit the loop immediately. It means that the program jumps to the statement after the **`for`** statement
  - **`continue;`** statement
    - When the program see a **`continue`**; statement in the **`for`** loop, the program does not execute the remaining statement (after **`continue`**) of current loop. The program will go to the condition checking step to check if it should execute the next loop.

for statement

Example

- Print out squares of even integers 0,2, .., 8

```
int i;
for(i=0; i < 10;  i+= 2){
        cout << i*i << "\t";
}
cout << "\n";


i=0;
for(;;){
        cout << i*i << "\t";
        i += 2;
        if(i >= 10) break;
}
cout << "\n";
```

Option 1: brief

Option 2: `for(;;)`
→ Must used `break`; statement

for statement

Example

- Print out squares of even integers 0,2, .., 8, in reverse order

```
for(int k=8; k >=0; k-= 2){
        cout << k*k << "\t";
}
cout << "\n";

i = 8;
for(;;){
        cout << k*k << "\t";
        i -= 2;
        if(i < 0) break;
}
cout << "\n";
```

Option 1: brief

Option 2: `for(;;)`
→ Must used `break`; statement

for statement

Example

- Using multiple control variables

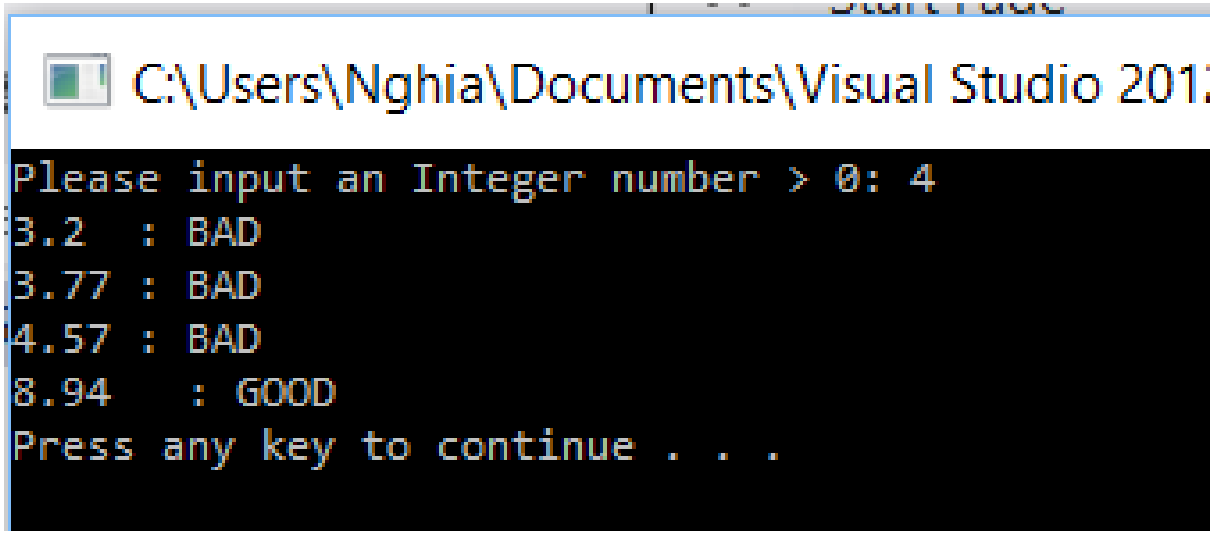
```
for(int i=0, k= 10; i < k; i++, k--){
        cout << i*k << "\t";
}
cout << "\n";

for(int i=0, k= 10, n=0; n < 10; i++, k--, n++){
        cout << i*k << "\t";
}
cout << "\n";
```

What is the output of two programs above, Why?

for statement

Example

- Write a program that
  - Allow user to input an Integer number N > 0
  - The program generates randomly N scores (from 0 to 10, real-valued)
  - The program prints out the scores and the corresponding rating as shown



for statement

Example

```
#include <iostream>
#include <iomanip>
using namespace std;

int main(){
        int N;
        cout << ”Please input an Interger Number >0: ”;
        cin >> N;

        if(N <= 0)
                cout << ”The program is not working with negative
number\n”;
        else{
                //TODO: The code on the next slide is place here
        }//end if
        cout << "\n\n";
        system("pause");
        return 0;
}
```

for statement

Example

Generate randomly N points

for loop statement

Rate & print

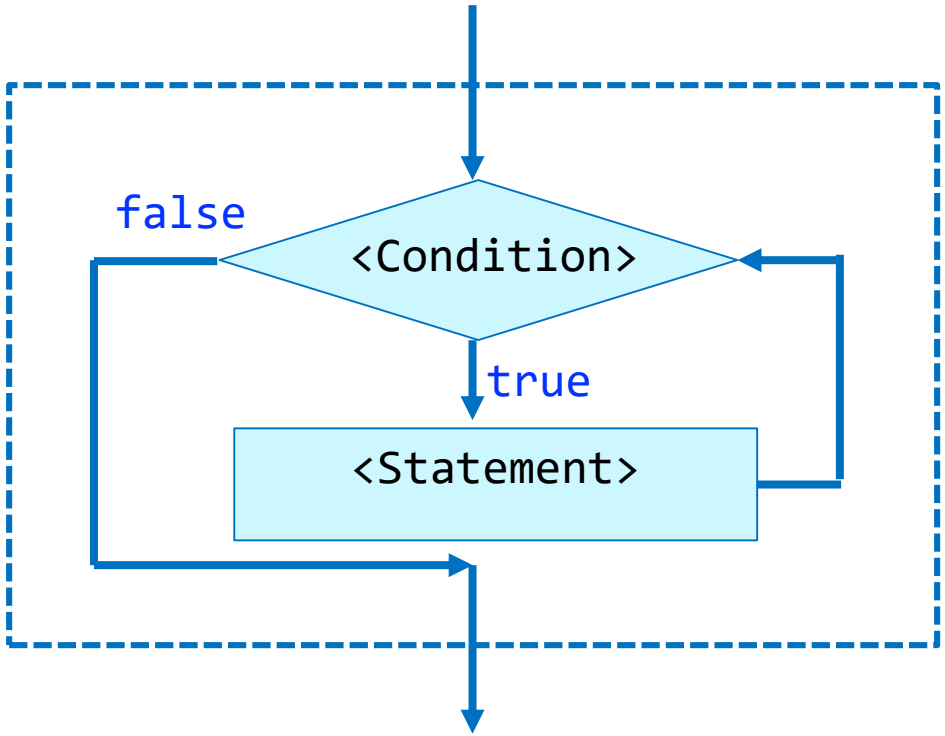
```
time_t t;
srand((unsigned) time(&t));
for(int i=0; i<N; i++){
        float diem = ((float) rand() / RAND_MAX)*10;

        if(diem < 5.0f){
                cout << left << setprecision(3) << diem <<
        đ: BADđ << endl;
        }
        else if(diem < 6.5f){
                cout << left << setprecision(3) << diem <<
        đ: AVERAGEđ << endl;
        }
        else if(diem < 8.5f){
                cout << left << setprecision(3) << diem <<
        đ: FAIRLY GOODđ << endl;
        }
        else if(diem < 9.5f){
                cout << left << setprecision(3) << diem <<
        đ: GOODđ << endl;
        }
        else{
                cout << left << setprecision(3) << diem <<
        đ: EXCELLENTđ << endl;
        }
}//end for
```

Nested `for` statement

- Application
  - Used when dealing with multidimensional arrays
  - When we need to access the data in nested arrays
    Example: When we need to access student information, we can access by two step:
    - Browse all classes in the school (an array)
    - For each class, access the information of each student (another array inside the class element)
  - etc.

while statement

Concept



<Condition>:

Can be a boolean expression or an expression that can be transformed to boolean type

<Statement>:

Can be single or composite statement

`while` statement

`Concept`

- Principles of execution
  - The program that checks the conditional expression
  - If condition is `true`
    - Execute statement
    - Jump to step checking stop condition
      - => The loop must have an operation that changes the conditional expression so that the program does not loop infinitely
  - In contrast, (`false`) the program jump to the statement after the loop.

while statement

Syntax

```
while(<Condition>)
    <Statement>
```

The case implemented for complex statement

```
while(<Condition>){
    <statement 1>
    <statement 2>
    <statement N>
}
```

`while` statement
Syntax

- Note with the `while` statement
  - The preceding statements (before while) often made the assignment so that the execution condition is satisfied
    - It is possible to assign control variables
    - It is possible to assign counter variables that count the index of the iterations
    - Etc.
  - There may be cases
    - `while(true){ …}`
    - `while(1){…}`
      - For these form, we need use `break;` statement
  - The meaning of `break` and `continue` statement is shown in section "for statement"

while statement
Example

- The program that calculates and prints the sum of squares of ten number from 1 to 10

```
#include <iostream>
using namespace std;

int main(){
        int i =0;
        int sum = 0;
        while(++i <= 10){
                sum += i*i;
        }
        cout << "Sum = " << sum <<endl;

        system("pause");
        return 0;
}
```

Pre-loop initialization is essential and important

Must change the condition expression inside the loop

do while statement

Concept

- Principles of execution
  - (1) The program immediately executes <Statement>
    - Therefore, <Statement> is executed at least **01** times
    - Once executed, the program evaluates the conditional expression and checks its value.
  - (2) If conditional expression is true
    - Jump to step (1) to execute <Statement>
  - (3) Otherwise, (false), the program jumps to the statement after the loop.

while statement

Example

- The program approximates the value of nonlinear functions
  - Dùng Taylor Maclaurin

$$\frac{1}{1-x} = \sum_{n=0}^{\infty} x^n = 1 + x + x^2 + x^3 + \ldots \qquad R = 1$$

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots \qquad R = \infty$$

$$\sin x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots \qquad R = \infty$$

$$\cos x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots \qquad R = \infty$$

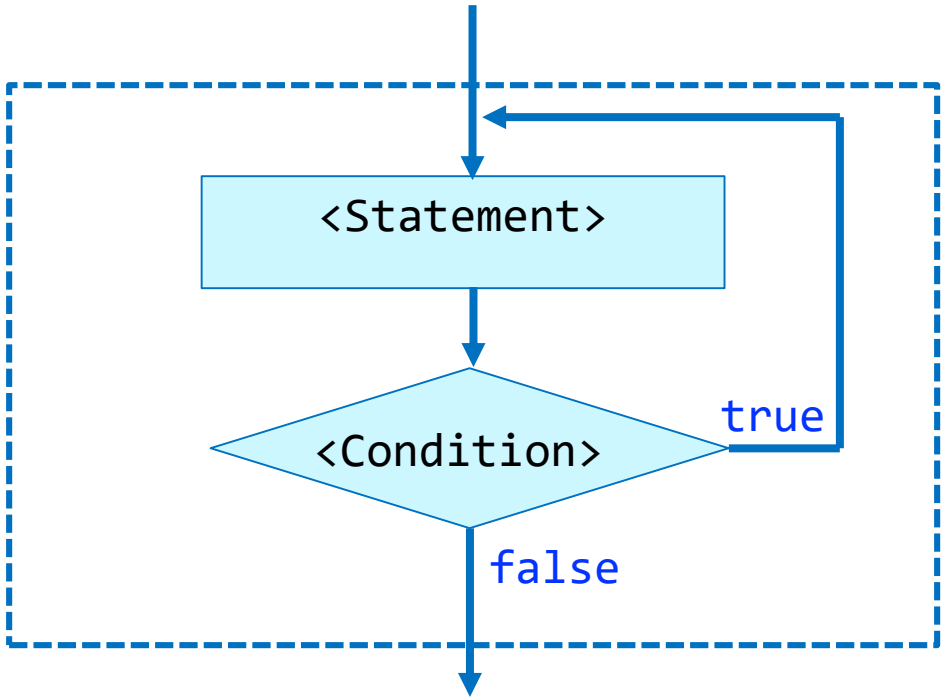
$$\tan^{-1} x = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{2n+1} = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \ldots \qquad R = 1$$

$$\ln(1+x) = \sum_{n=1}^{\infty} (-1)^{n-1} \frac{x^n}{n} = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots \qquad R = 1$$

$$(1+x)^k = \sum_{n=0}^{\infty} \binom{k}{n} x^n = 1 + kx + \frac{k(k-1)}{2!}x^2 + \frac{k(k-1)(k-2)}{3!}x^3 + \ldots \qquad R = 1$$

do while statement

Concept



<Condition>:

Can be a logical expression or an expression that can be transformed to logical

<Statement>:

Can be single or complex statement

do while statement

Syntax

```
do{
    <Statement>
} while <Condition>)
```

Implementation for composite statement:

```
do{
    <statement 1>
    <statement 2>
    <statement N>

} while <Condition>)
```

do while statement

Syntax

- Note with the `while` statement
  - The preceding statements (before `do`) often made the assignments to determine the stop condition of the problem
    - The initial sum is zero
    - You can assign values to control variables
    - You can assign counter variable to count the number of loops
    - Etc.
  - The following cases can be used
    - `do { …} while(true)`
    - `do { …} while(1)`
      - For these forms, we need to use `break;` statement
  - The meanings of `break` and `continue` statement are the same as when “for statement” is used.

do while statement

Syntax

- Notes with the `while` statement
  - `while` and `do-while` are fairly similar except for one point:
    - `while`:
      - statement may not be executed
    - `do While`:
      - statement is executed at least 01 times
  - Assignment statements before (above) `while` and `do While` are very important to determine the stop condition of the loop

Some common mistakes when using loops

- Infinite loop
  - Program failed to see or find the terminating condition
  - Program is able to find the termination condition, but the condition statement is not satisfied

Some common mistakes when using loops

- Infinite loop

```
for (int i = 0; i < N; i++)
{
    int j = 2;
    cout << i << ": ";
    while (j < i) {
        if (i % j == 0)
        {
            cout << j << " ";
            j++;
        }
    }
    cout <<endl;
}
```

Exercise 1

- Write a program to output the following string
  - Input: N (Number of line)
  - Output: String is shown below

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

Exercise 2

- Write a program that output the following string
  - Input: N (Number of line)
  - Output: String is as shown below

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

Exercise 3

- Write the program that output to console likes bellow
  - Input: N (Number of lines of the square matrix)
  - Output: The matrix contains numbers in spiral form

    With N = 4, we have a matrix

```
 1   2   3   4
12  13  14   5
11  16  15   6
10   9   8   7
```

Exercise 4

- Write a program that
  - Enter the number N and output the sequence of square number not greater than N (that is, $\leq N$)
  - Enter the number N and output the Fibonacci sequence that not greater than N
    Example: With N = 20
    The sequence of square numbers which is not greater than 20:
    1, 4, 9, 16
    The sequence of Fibonacci numbers which is not greater than 20:
    1, 1, 2, 3, 5, 8, 13

Summary

- You should understand the execution principles of loop statements
- Know how to use the loop statements to solve the practical problem