INTRODUCTION TO PROGRAMING

Chapter 3

Control Structures

GV: Thái Hùng Văn

Khoa Công Nghệ Thông Tin
Trường Đại Học Khoa Học Tự Nhiên
ĐHQG-HCM

Objectives

In this chapter, you will:

- Learn about **statements** and **control structures**
- Examine relational & logical operators, explore how to form and evaluate logical (Boolean) expressions
- Discover how to use the selection control structures **`if`**, **`if...else`**, and **`switch`** in a program
- Learn about repetition (looping) control structures **`while`**, **`do...while`**, and **`for`**
- Explore how to construct and use count-controlled, sentinel-controlled, flag-controlled repetition structures
- Examine `break` and `continue` statements
- Discover how to form and use nested control structures

Statements - Namespace

What are statements?

- Statements are fragments of the C++ program that are executed in sequence
- C++ includes the following types of statements:
  - declaration statements; *// example: int a;*
  - expression statements; *// ex: a += 2019;*
  - compound statements; *// ex: { int a; a=2; cout << "a =" << a; }*
  - selection statements; *// ex: if (a < 9) a++;*
  - repetition statements; *// ex: while (n > 1) cout << n-- <<'\n';*
  - jump statements; *// ex: break; // continue; // return; // go to X;*
  - try blocks ; *// ex:* `try {/* */} catch (...) {/* */}`
  - atomic and synchronized blocks *// atomic_noexcept { return –n; }*

[https://en.cppreference.com/w/cpp/language/statements]

namespace

- A namespace is a declarative region that provides a scope to the identifiers *(names of variables, functions,..)* inside it.
- Using namespaces, we can create two variables or member functions having the same name.
- Ex:

```
include <iostream>
namespace MyN {
   int val = 500; // Variable created inside namespace
}
double val = 100; // Global variable

int main() {
   int val = 200; // Local variable
   std::cout << MyN::val << '\n';
   return 0;
}
```

Global and Local Variable

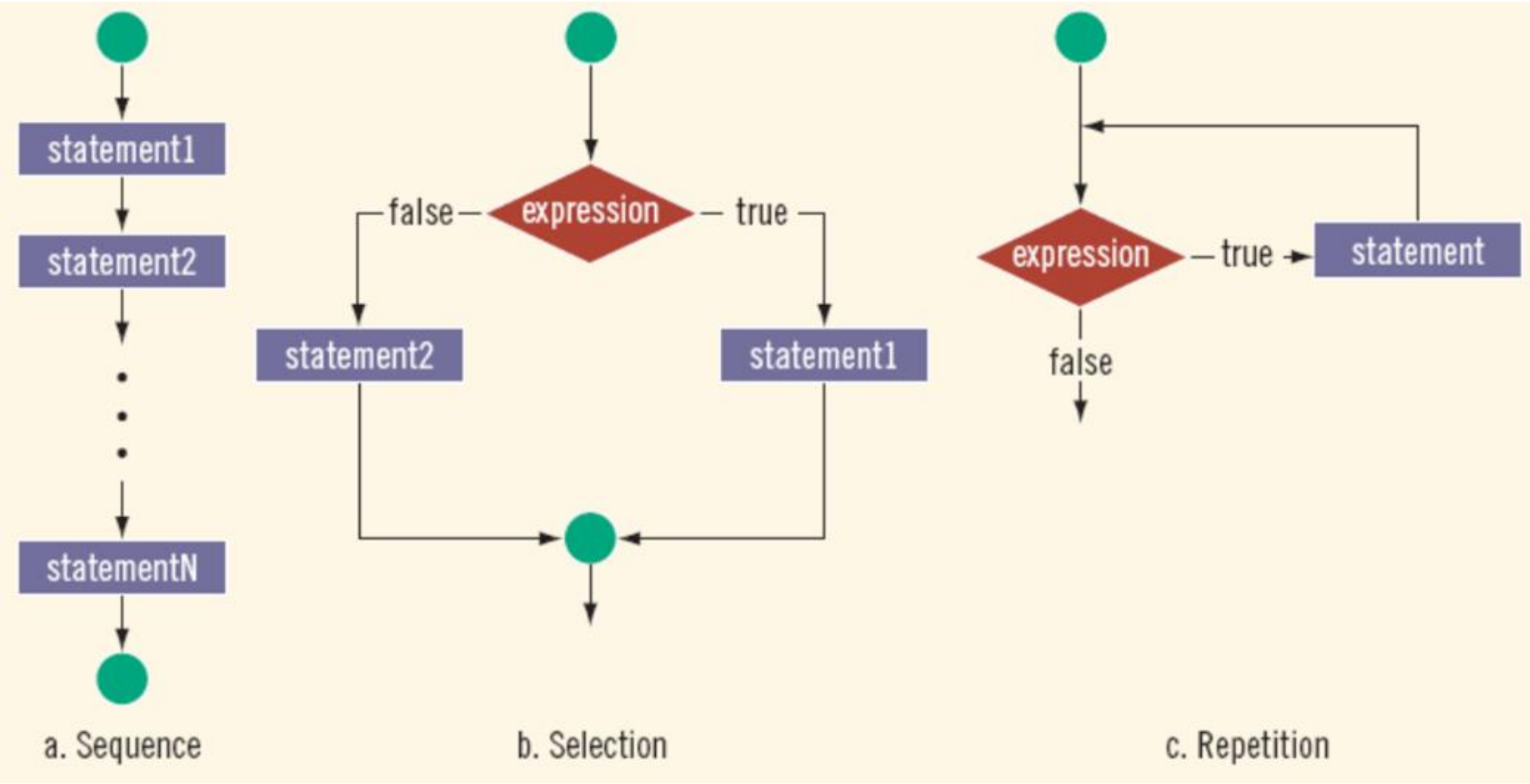
- **Global variables** are declared outside any function, and they can be accessed (used) on any function in the program.
- **Local variables** are declared inside a function, and can be used only inside that function.
- It is possible to have **local variables** with the same name in different functions.

Control Structures

Control Structures

- A computer program can proceed:
  - In sequence
  - Selectively (branch) - making a choice
  - Repetitively (iteratively) - looping
- So there are **three types** of control structures:
  a. **Sequence structure** (straight line paths)
  b. **Selection structure** (one or many branches)
  c. **Repetition /Looping structure** (repetition of a set of activities)
- **Selection structures** in C/C++ are implemented using **If**, **If_Else** and **Switch** statements.
- **Looping structures** are implemented using **While**, **For** and **Do_While** statements.

Control Structures *(cont.)*



Control Structures *(cont.)*

- The Expression in Selection and Repetition Structures is a Logical (Boolean) expression that can be **`true`** or **`false`**
- We can use the data type **`bool`** for logical variables
- The data type **`bool`** is stored by integer type, and the identifier **`true`** has the value 1, **`false`** has 0
- The relational operators: **`<`**, **`<=`**, **`==`**, **`>=`**, **`>`**, **`!=`**
- The boolean operators: **`&&`** (and), **`||`** (or), **`!`** (not)
- Examples:
  - (1 < Num) && (Num < 10) // ==True if 1 < Num < 10
  - (1 < Num < 10) // ==True with any Num>1 ➔ be careful
  - (! found || (x != y))

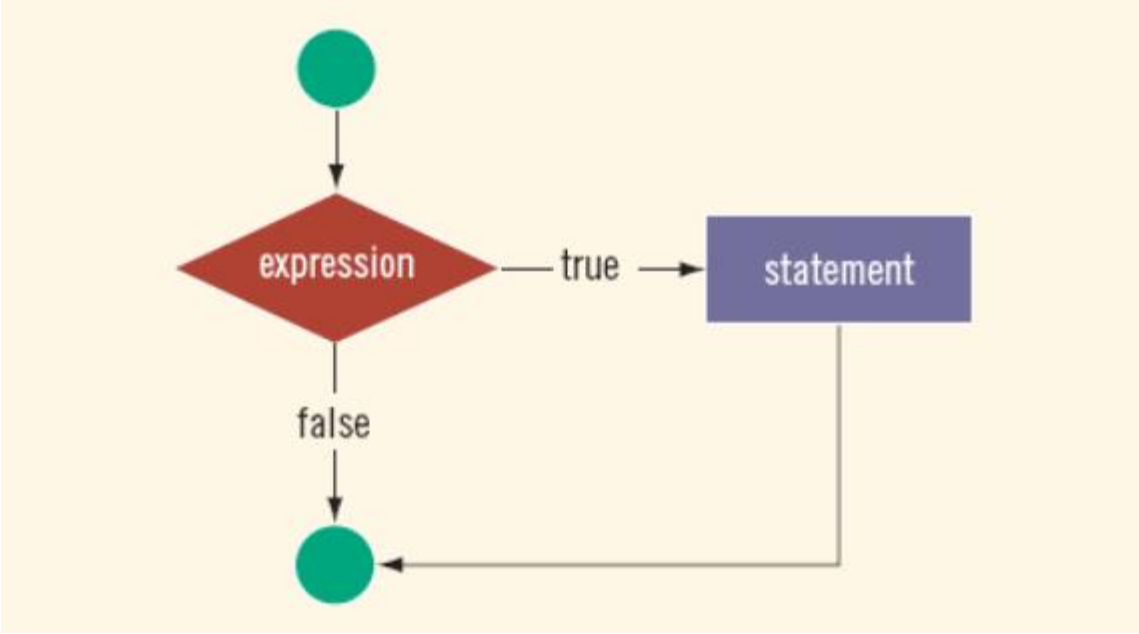
Selection control structures

IF structure

- **IF** structure is a one-way selection
- Syntax:

```
if (expression)
    statement
```

- The (compound) statement is executed if the value of the expression is `true` (bypassed if the value is `false`; program goes to the next statement)



IF structure - example

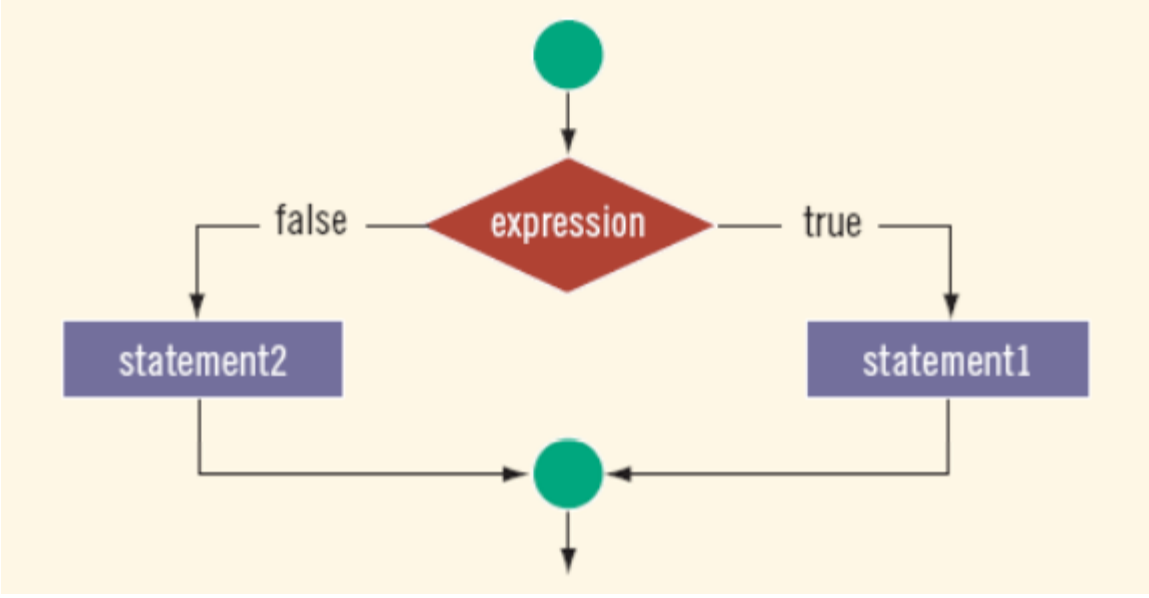
```
#include <iostream>

int main() {
    float a, b, c;
    std::cout << “Enter 3 numbers: ”;
    std::cin >> a >> b >> c;
    float max = a;
    if (b > max)
            max = b;
    if (c > max)
            max = c;
    std::cout << “The largest number is: ” << max;
}
```

IF..ELSE structure

- **IF..ELSE** statement is a two-way selection
- Syntax:

```
if (expression)
     statement1
else
     statement2
```

- If expression is `true`, `statement1` is executed; otherwise, `statement2` is executed



IF..ELSE structure - example

```
#include <iostream>
using namespace std;
int main() { // Solving the equation Ax+B=0
    float a, b, x;
    cout << "Enter A & B: ";  cin >> a >> b;
    if (a == 0)
        if (b == 0)
            cout << "Infinite solution";
        else
            cout << "No solution";
    else
        cout << "The root is: " << -b/a;
}
```

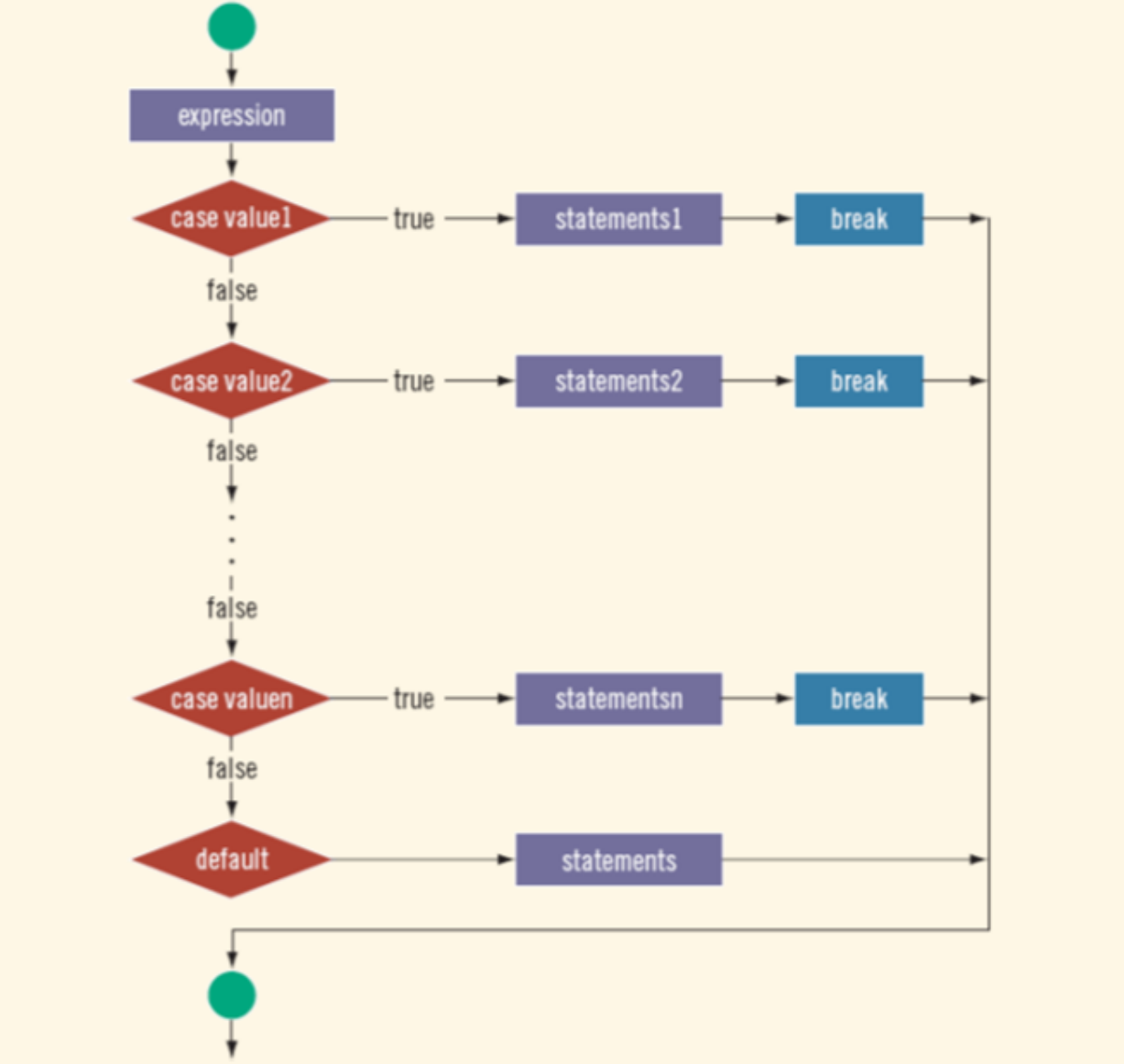
SWITCH structure

- **SWITCH** statement is a multi-way selection
- `switch` expression is evaluated first
- Value of the expression determines which corresponding action is taken
- One or more statements may follow a case label
- Braces are not needed to turn multiple statements into a single compound statement
- The `break` statement may or may not appear after each statement

- Syntax:

```
switch (expression)
{
case value1:
     statements1
     break;
case value2:
     statements2
     break;
     .
     .
     .
case valuen:
     statementsn
     break;
default:
     statements
}
```

SWITCH structure – flow chart



SWITCH structure – example

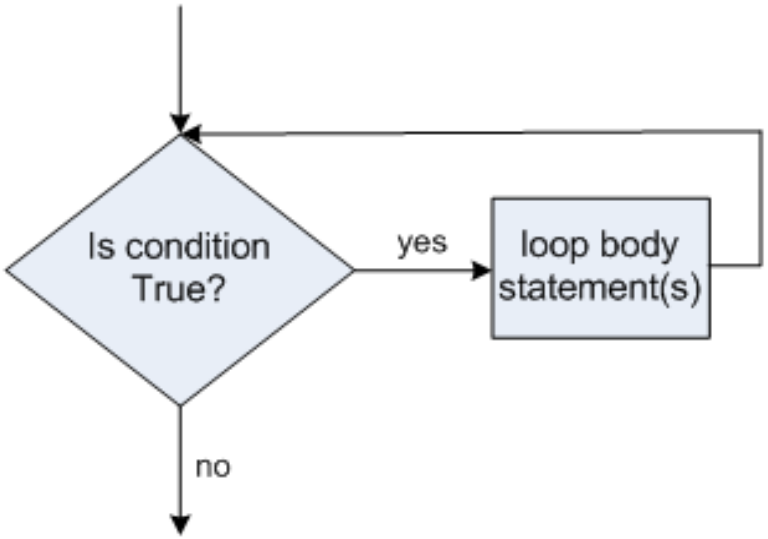
```
#include <iostream>
void main() {
    int month;
    cout << “Enter month (1-12): ”;  cin >> month;
    cout << “This month has “;
    switch (month)  {
        case 2:
            cout << “28 or 29 days”;
            break;
        case 4: case 6: case 9: case 11:
            cout << “30 days”;
            break;
        default :
            cout << “31 days”;
    }
}
```

Repetition control structures

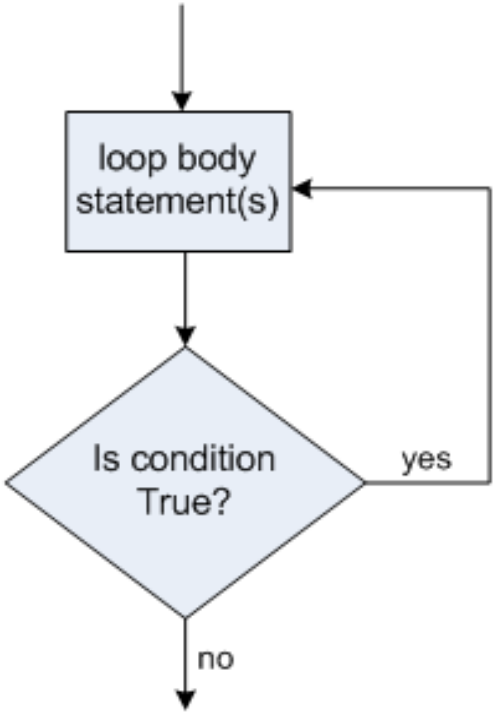
What is Repetition control structure?

- Repetition structure, or Repetition /Iteration statement, or Loop, is used to repeat the same code multiple times.
- The number of repetitions is based on criteria defined in the Loop structure, usually a logical expression

Pre-test Loop



Post-test Loop

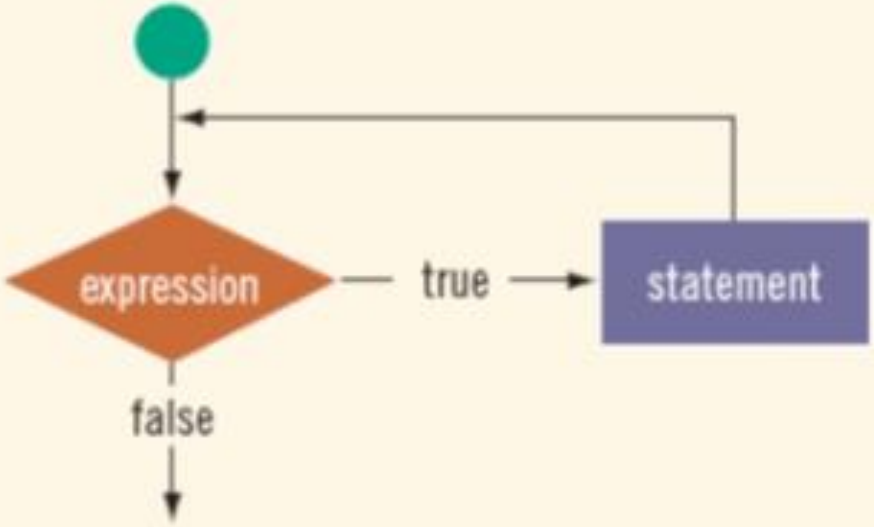


WHILE structure

- Syntax:

```
while (expression)
    statement
```

- The *expression* is handled the same as in the **IF** statement
- Statement is called the body of the Loop



WHILE structure - example

Example1:

```
int k=0;
while (k<20) {
        k+=5;
        cout << k << " - ";
}       // => Sample Run:       5 – 10 – 15 – 20 -
```

Example2:

```
int k=0;
while (k<20) {
        cout << k << " - ";
        k+=5;
}       // => Sample Run:       0 - 5 – 10 – 15 -
```

Example3:

```
int k=0;
while (0<k && k<20) {
        k+=5;
        cout << k << " - ";
}       // => The body of while loop never executes
```

WHILE structure # Counter-controlled Loops

- If you know exactly how many times need to be repeated, the while loop becomes a **counter-controlled loop**
- The **counter-controlled while** loop takes the form:

```
int  counter=0; // Initialize the loop control variable
while (counter < N) // Test the loop control variable
{
        … … …;
        counter++;      // Update the loop control variable
        … … …;
}
```

WHILE structure # Counter-controlled Loops

- **Example: Calculate S = 1/1 + 1/2 + 1/3 + … + 1/N**

```
float S =0; int  i=1;
while (i <= N) {
        S += 1.0 / i ;
         i++;
}
```

- **Example: Calculate S = 1! + 2! + … + N!**

```
 int  i=1;
double S =0, factorial_i = 1;
while (i <= N) {
        factorial_i *=  i ;
        S += factorial_i ;
        i++;
}
```

WHILE structure # Sentinel-controlled Loops

- Sentinel variable is tested in the condition and loop ends when sentinel is encountered
- The **sentinel -controlled `while`** loop takes the form:

```
cin >> variable;            // Inittalize the loop control variable
while (variable != sentinel) // Test the loop control variable
{
        … … …;
        cin >> variable;        // Update the loop control variable
        … … …;
}
```

WHILE structure # Sentinel-controlled Loops

- Example:

```
float f, max;
cin >> f;
max = f;
while (f != 0)
{
        cout << "Enter a number: ";
        cin >> f;
         if (max < f)
                max = f;
}
cout << "Maximum number is : " << max;
```

WHILE structure # Flag-controlled Loops

- A flag-controlled `while` loop uses a `bool` variable to control the loop
- The **flag-controlled `while`** loop takes the form:

```
found = false;   // Inittalize the loop control variable
while ( ! found) // Test the loop control variable
{
        … … …;
        if (expression)
                found = true; // Update the loop control variable
         … … …;
}
```

WHILE structure # Flag-controlled Loops

- Example:

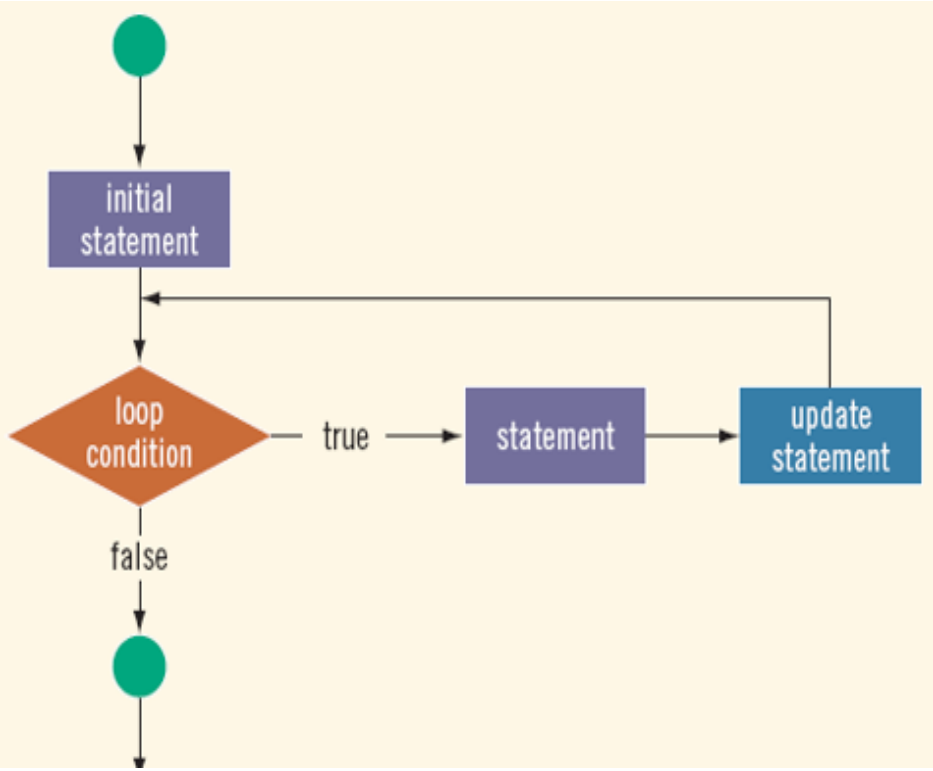
```
cin >> N ;
bool flag = true;
int digit1, digit2=N%10;
while ( flag && N>10) { // Test the loop control variable
        N /= 10 ;
        digit1 = N % 10;
        if (digit1 > digit2)
                flag = false; // Update the loop control variable
         digit2 = digit1;
}
if ( flag ) cout << " The digits of N are in ascending order " ;
else      cout << " The digits of N are not in ascending order " ;
```

FOR structure

- Syntax:

```
for (initial statement; loop condition; update statement)
    statement
```

- The **`statement`** is called the **for** body (statement); the **`initial statement`**, **`loop condition`**, and **`update statement`** are called `for` loop control statements



FOR structure - example

Ex1:

```
for (int k=0; k<20;  k+=5)
    cout << k << " - ";        // => Sample Run:      0 - 5 - 10 - 15 -
```

Ex2:

```
for (int k=0; 0<k && k<20;  k+=5) // => The FOR body never executes
    cout << k << " - ";
```

Ex3:

```
for (int k=0; ;  k+=5)
    cout << k << " - ";
// => Sample Run:  0 - 5 - 10 - 15 – 20 - 25 - 30 - 35 - 40 - 45 - 50 -…
```

Ex4:

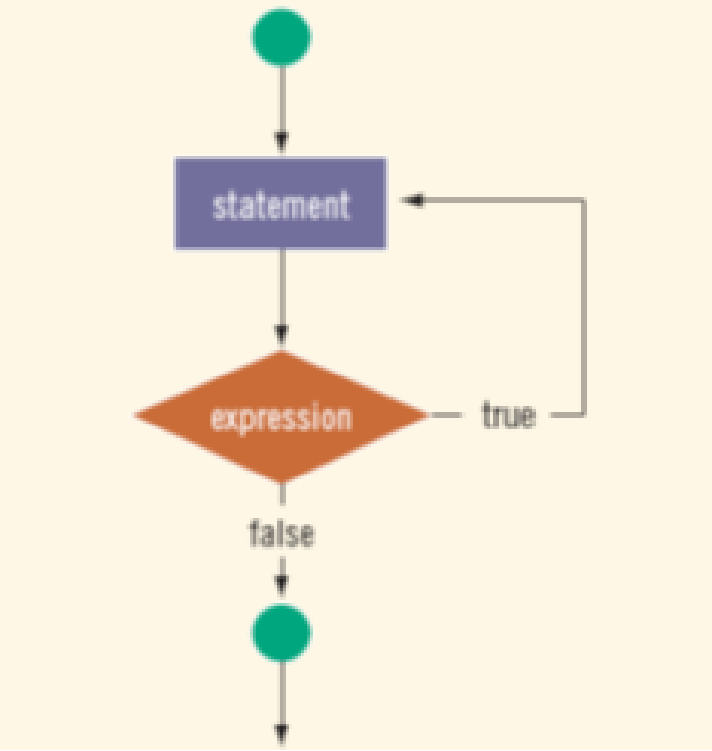
```
for ( ; ; )     cout << k << " - ";   // => Sample Run:???
```

DO WHILE structure

- Syntax:

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

- The **`statement`** executes first, and then **`expression`** is evaluated
- To avoid an infinite loop, body must contain a statement that makes the expression `false`
- DO WHILE loop always iterates at least once



DO WHILE structure - example

Ex1:

```
int k=0;
do {
        k+=5;
        cout << k << " - ";
} while (k<20);    // => Sample Run:       5 – 10 – 15 - 20 -
```

Ex2:

```
int k=0;
do {
        cout << k << " - ";
        k+=5;
} while (k<20);    // => Sample Run:       0 - 5 - 10 - 15 -
```

Ex3:

```
int k=0;
do {
        k+=5;
        cout << k << " - ";
} while (0<k && k<20);        // Sample Run:          5 – 10 – 15 – 20 -
```

break and continue statements

- **`break`** and **`continue`** statement alter the flow of control
- They can be used in the loop structures to skip the loop body remainder
- Example:

```
while (true) {
        cout <<“Enter a positive integer (or 0 to quit): ”;
        cin >> N;
        if (N==0)  break;
        if (N<0)  continue;
        if (N%2==0) cout <<“This is an even number ”;
        else      cout <<“This is an odd number ”;
}
```

break statement

- `break` statement is used for two purposes:
  - To exit early from a loop
    Can eliminate the use of certain (flag) variables
  - To skip the remainder of the `switch` structure
- After the `break` statement executes, the program continues with the first statement after the structure
- Example:

```
do {
        score = ResultOfAction();
        if (score == 0) total_score -= 10;
        else        total_score += score;
         if (total_score < 0)        // game over
              break;
        cout << total_score;
} while (total_score < limit);
```

continue statement

- `continue` is used in the Loop structures, not use in the `switch` structure
- `continue` skips remaining statements and proceeds with the next iteration of the loop
- Example:

```
for (int k = 10; k < 100; k += 2)
{
        if (k % 10 == 0)
            continue;
        cout <<  k << " ";
}
```

Nested Control Structures

- To create the following pattern:

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

- We can use the following code:

```
for (i = 1; i <= 5 ; i++)
{
      for (j = 1; j <= i; j++)
                  cout << "*";
      cout << endl;
}
```

Nested Control Structures

- What is the result if we replace the first `for` statement with the following?

```
for (i = 5; i >= 1; i--)
```

- Answer:

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

The Loops Sumary

- C/C++ has 3 looping structures: `while`, `for`, & `do while`
- `while` and `for` loops are pretest loops, `do`...`while` loop is a posttest loop
- `while` and `for` may not execute at all, but `do`...`while` always executes at least once
- Executing a `break` statement in the body of a loop immediately terminates the loop
- Executing a `continue` statement in the body of a loop skips to the next iteration

Choosing the Right Looping Structure

- If you know or can determine in advance the number of repetitions needed, the `for` loop is the correct choice
- If you do not know and cannot determine in advance the number of repetitions needed, and it could be zero, use a `while` loop
- If you do not know and cannot determine in advance the number of repetitions needed, and it is at least one, use a `do...while` loop

Reference

- **Thinking in C**, Bruce Eckel, E-book, 2006.
- **Theory and Problems of Fundamentals of Computing with C++**, John R.Hubbard, Schaum's Outlines Series, McGraw-Hill, 1998.

Bài tập

1. VCT nhập một số nguyên dương n. Tính:
   - S = 1 + 1/2 + … + 1/n
   - S = 1! + 2! + … + n!
   - $S = 1^1 + 2^2 + 3^3 + \ldots + n^n$
2. VCT nhập các số thực cho đến khi gặp số 0.
   - Xác định số lớn nhất.
   - Xác định số thực dương nhỏ nhất.
   - Xác định số thực âm gần với -123.45 nhất.
3. Một máy ATM đang có 4 loại tiền là 500K, 200K, 100K, 50K. Nhập số tiền muốn rút và liệt kê tất cả các phương án mà máy có thể chi trả. *(Ví dụ, nếu rút 200K thì các phương án có thể là: #1: 1 tờ 200K; #2: 2 tờ 100K; #3: 1 tờ 100K + 2 tờ 50K; #4: 4 tờ 50K)*
4. VCT nhập vào 1 số nguyên dương, kiểm tra xem tổng các chữ số có bằng tích và xuất ra thông báo tương ứng, lặp lại việc trên cho đến khi nhập số 0.
5. VCT tính căn bậc ba của một số thực chính xác đến 0.01 – không dùng đến các hàm tính toán
6. VCT hiển thị đồng hồ theo đúng dạng **hh:mm:ss AM/PM** (giờ của đồng hồ được lấy theo giờ của máy), khi chỉ số giây ss=0 thì phát ra 1 tiếng bíp. Chương trình kết thúc khi người dùng nhấn phím Esc hoặc khi đã chạy đủ số giây được nhập vào trước đó.

Bài tập

7. VCT nhập một số nguyên dương n. Xác định:
   - Tổng các chữ số của số vừa nhập.
   - Các chữ số này có tăng dần hay giảm dần không?
   - Số này có phải là số đối xứng? *(các chữ số đối xứng, vd: 77, 101, 2002,..)*
   - Số này có phải là số nguyên tố?
   - Số này có phải là số hoàn chỉnh *(bằng tổng các ước nhỏ hơn nó, vd 6 = 1+2+3)*

8. Viết chương trình (VCT) nhập ba số nguyên a,b,c

   a/ Kiểm tra chúng có thể là ba cạnh của tam giác hay không.

   b/ Nếu chúng là ba cạnh của tam giác thì đó là tam giác gì.

   c/ Kiểm tra xem chúng có thứ tự như thế nào (tăng , giảm, hay không tăng cũng không giảm). Sau đó thực hiện các việc:

   - Nếu tăng: Xác định xem chúng có thể là một bộ <tháng, ngày, năm> hợp lệ hay không.

   - Nếu giảm: Xác định xem chúng có thể là ba cạnh của tam giác hay không – nếu phải thì đó là tam giác gì.

   - Không tăng không giảm: Giải phương trình bậc hai ***aX^2+bX+c=0***

In-class exercises

1. Trình bày thuật toán dạng mã giả và lưu đồ tính căn bậc ba của một số thực chính xác đến 0.001 (không dùng đến các hàm tính toán có sẵn trong thư viện)
2. VCT nhập một số nguyên dương N, xuất ra tất cả các số nguyên "9 nút" nhỏ hơn N.

Ghi chú: Số "9 nút" là số có <tổng các chữ số> Mod 10 = 9

Ví dụ: Nhập N=200 => Xuất 9, 18,.. , 90, ~~99~~, 108, 117,.. , 171, 180, ~~189~~, 199