

MH1402 Algorithms & Computing II

Lecture 3 Control Statements

Wu Hongjun

Overview

- **Control Statements**

- **Selection statements: if (if-else, else if), conditional operator, switch**
- **Loop statements: while, do-while, for**
- **break; continue;**

Control Statements

- **Control statements are important in programming**
 - Change the execution of program based on conditions
 - Allow the program to perform complicated task with relatively small code
- **Control statements in C++ are very similar to those in MATLAB**
 - But the syntax are **different**
- **Control statements are critical when you are learning programming**
 - The major obstacle in learning programming is how to use control statements to solve problem
 - Once you get familiar with the use of control statements, programming becomes easy to you

Control Statements

- **Control statements**
 - **Selection statements**
 - **if, if-else (else if), conditional operator, switch**
 - **Loop statements**
 - **for, while, do ... while**
- **Syntax quite different from MATLAB**
 - **In MATLAB, the word 'end' is used to indicate the end of a control statement (if, for, while, switch);**
 - **In C++, we use curly braces to indicate a statement block; and only one statement block (or one statement) following a control statement corresponds to that control statement**

Selection Statements: if

- **'if'** selection statement has the form (syntax quite different from MATLAB)

```
if (condition)
{
    statement1;
    statement2;
    .....
}
```

- The condition is (or returns) a bool type value. If the condition is true, then the statement block following **'if'** gets executed. Otherwise, the statement block corresponding to **'if'** is ignored.
- **If there is only one statement in the statement block following 'if', then the curly braces may be omitted:**

```
if (condition)
    statement;
```

- Example of 'if' statement:

```
int main()
{
    int x = 4, y = 4, z = 4;

    if ( x > 5 )
    {
        y = 3;
        z = 1;
    }
    cout << "y is " << y << endl;
    cout << "z is " << z << endl;
    return 0;
}
```

```
y is 4
z is 4
```

- The second example:

```
int main()
{
    int x = 4, y = 4, z = 4;

    if ( x > 5 )
        y = 3;
        z = 1;

    cout << "y is " << y << endl;
    cout << "z is " << z << endl;
    return 0;
}
```

```
y is 4
z is 1
```

Selection Statement: if-else

- The 'if-else' statement is used to execute one of two statement blocks:
if (condition)

```
{  
    statementA1;  
    statementA2;  
    .....  
}  
else  
{  
    statementB1;  
    statementB2;  
    .....  
}
```

- If the condition is true, the statement block corresponding to 'if' is executed. Otherwise, the statement block corresponding to the 'else' is executed.
- Only one of the two statement blocks gets executed.

```
int main()
{
    int x = 4, y, z;

    if ( x < 5)
    {
        y = 3;
        z = 1;
    }
    else
    {
        y = 2;
        z = 5;
    }
    cout << y << z;
    return 0;
}
```

```
int main()
{
    int x = 4, y, z;

    if ( x < 5)
        y = 3;
        z = 1;
    else
        y = 2;
        z = 5;

    cout << y << z;
    return 0;
}
```

**error: 'else' without
a previous 'if'**

Why?

The previous if
statement
ends at **y = 3;**
If **z = 1;** is removed,
then there is no
compiling error.
What are the
outputs?

- The third example of 'if-else' :

```
int main()
{
    int x = 4, y, z;

    if ( x < 5) ;
        y = 3;
    else
        y = 2;
        z = 5;

    cout << y << endl;
    cout << z << endl;
    return 0;
}
```

**error: 'else' without
a previous 'if'**

Why?

The previous if statement
ends at ';' before 'y = 3';
If the semicolon before y = 3
is removed, then
there is no compiling error.

Selection Statement: else if

- In C++, it must be **'else if'** (in MATLAB, it is elseif)

```
if (condition1)
{
    statementA1;
    statementA2;
    ....
}
else if (condition2)
{
    statementB1;
    statementB2;
    .....
}
```

- There may be more than one 'else if'
- An else may be added to the end of an 'if-else-if'.

- Example: this program converts a test score into its equivalent letter grade

```
int  score;
//.....
if (score >= 90)
    cout << "Grade is A." << endl;
else if (score >= 80)
    cout << "Grade is B." << endl;
else if (score >= 70)
    cout << "Grade is C." << endl;
else if (score >= 60)
    cout << "Grade is D." << endl;
else
    cout << "Grede is F." << endl;
```

Selection Statement: Conditional Operator

- **Conditional operator:** it uses the '**?**' and '**:**' symbols in the form:

condition ? Expression1 : expression 2

- It combines three operands to produce a value
- The resulting value is either expression1 (if condition is true) or the value of expression 2 (if condition is false)
- Example:

$z = (x < y ? x : y) ;$

z is assigned the minimum of x and y.

Why? If x is less than y, return x (the minimum of x and y)
 if x is not less than y, return y (the minimum of x and y)

- **Conditional operator is closely related to 'if-else' statement**

Selection Statement: switch – parallel selection

```
switch (expression)
{
    case constant1: statementA1;
                    statementA2;
                    .....
                    break;
    case constant2: statementB1;
                    statementB2;
                    .....
                    break;

    .....
    default:
        statementZ1;
        statementZ2;
        .....
}
```

'switch', 'case', 'default', 'break' are C++ keywords;

The switch evaluates *expression*,
If expression is equal to constant1, then the statements beneath '*case constant 1:*' are executed **until a 'break' is encountered**.
If *expression* is not equal to *constant1*, then *expression* is compared to *constant2*, if they are equal, then the statements beneath '*case constant 2:*' are executed until a break is encountered. If not equal, then the same process repeats for each of the constants... If none of the constants match, then the statements beneath '*default:*' are executed.

```
int score;
// .....
switch (score/10)
{
    case 10:
        cout << "Grade is A.";
        break;
    case 9:
        cout << "Grade is A.";
        break;
    case 8:
        cout << "Grade is B.";
        break;
    case 7:
        cout << "Grade is C.";
        break;
    case 6:
        cout << "Grade is D.";
        break;
    default:
        cout << "Grade is F.";
}
```

Example of switch

What would happen if all the break statements are removed?

Loop: while

- The 'while' loop has the form (syntax different from MATLAB)

```
while (condition)
{
    statement1;
    statement2;
    .....
}
```

- If the condition is true, the statement block of the 'while' loop will be executed; otherwise, the statement block will not be executed and this 'while' loop terminates.
- After the statement block of the 'while' loop gets executed, the condition is checked again. As long as the condition is true, the statement block will be repeatedly executed.

Loop: do-while

- The 'do-while' loop is a variant of the '*while*' loop
- The 'do-while' loop ensures that the statement block will get executed at least once.
 - After the first execution of the statement block, if the condition is true, the program goes to the top of the statement block. As long as the condition is true, the statement block would get executed repeatedly.

do

{

statement1;

statement2;

.....

}

while (condition);

The semicolon is required after
the while condition!

- ‘while’ loop example :
print 1 to 10

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

int main()
{
    int x = 1;
    while (x < 11)
    {
        cout << x << endl;
        x++;
    }
    return 0;
}
```

- ‘do-while’ loop example :
print 1 to 10

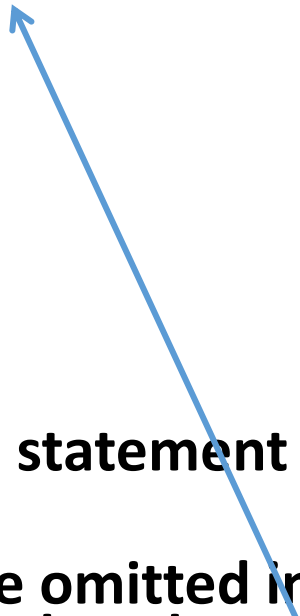
```
#include <iostream>
using namespace std;

int main()
{
    int x = 1;
    do
    {
        cout << x << endl;
        x++;
    }
    while (x < 11);
    return 0;
}
```

Loop: for

- The '*for*' loop works like the while loop, but with change in syntax
- And the syntax of *for* loop is quite different from that in MATLAB

```
for (initialization; condition; update )  
{  
    statement1;  
    statement2;  
    .....  
}
```



- If there is only one statement in the statement block, the curly braces may be omitted.
- Initialization can be omitted in the for loop if the initialization has been done before the for loop, but the semicolon must be there.

```
for (initialization; condition; update )  
{  
    statement1;  
    statement2;  
    .....  
}
```

- At the beginning of the '*for*' loop, initialization of some variables (the initialization is done only once)
- Test the condition, if true, execute the statement block; after the execution of statement block, update the counter variables. Then test the condition, if true, execute the statement block, update ... (repeat until the condition is false)

Loop: for

- Example 1:

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

```
int main()
{
    for ( int x = 3; x < 10; x = x+1)
        cout << x << endl;
    return 0;
}
```

In this example, x is declared within this for loop, so the scope of this x is only within this for loop

3
4
5
6
7
8
9

- Example 2:

```
int main( )
{
    int counter;
    for ( int x = 1, counter = 0; x < 10; x = x+1)
        counter += x;

    cout << counter;    // print a random number.
    // Reason: counter in the for loop is declared again.
    // the scope of this counter is within the for loop.
    // the counter outside the for loop is uninitialized
    return 0;
}
```

- Example 3:

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

```
int main( )
{
```

```
    int counter;
```

```
    for (counter = 0, int x = 1; x < 10; x = x+1)
        counter += x;
```

```
    cout << counter;
    return 0;
```

```
}
```

Syntax Error:
Cannot compile
(should be **no comma before int**)



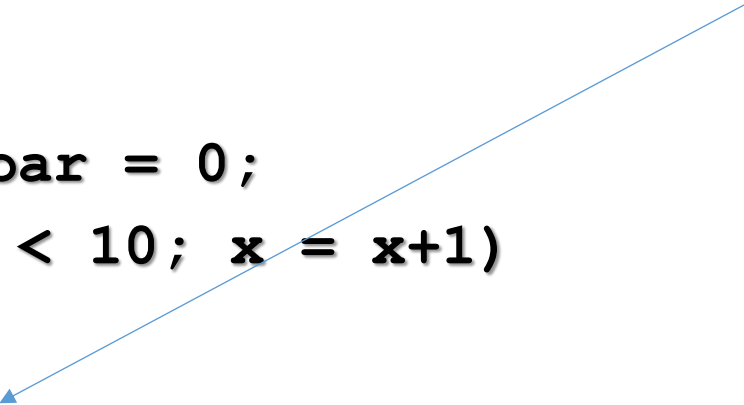
- Example 4:

Compilation error:

x is accessed outside of its scope (for loop)

```
int main( )
{
    int counter = 0, bar = 0;
    for (int x = 1; x < 10; x = x+1)
        bar += x;
        counter += x;

    cout << counter << endl;
    cout << bar << endl;
    return 0;
}
```



- Example 5:

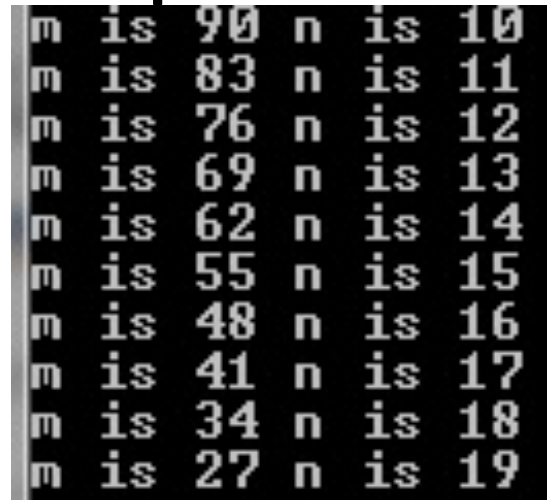
```
int main( )
{
    int counter = 0, bar = 0, x = 1;
    for ( ;x < 4; x = x+1)
        bar += x;
        counter += x;           // not in the for loop

    cout << counter << endl;    // what is the output?
    cout << bar << endl;       // what is the output?
    return 0;
}
```


Loop: for

- Multiple initializations and multiple updates must be separated with comma **,** (not semicolon)
- Example:

```
int main ()
{
    for (int m = 90, n = 10; m > n; m -= 7, n++)
        cout << "m is " << m << " n is " << n << endl;
}
```



m	is	90	n	is	10
m	is	83	n	is	11
m	is	76	n	is	12
m	is	69	n	is	13
m	is	62	n	is	14
m	is	55	n	is	15
m	is	48	n	is	16
m	is	41	n	is	17
m	is	34	n	is	18
m	is	27	n	is	19

Loop: for

- But we **should not use multiple conditions separated with comma ,**
 - Reason: if there are several conditions separated by comma, the only the right most condition is use.
- Example:
 for (i = 0; j >= 0, i <= 5; i++)
is equivalent to:
 for (i = 0; i <= 5; i++)

- A for loop can be changed to while loop and vice-versa.

```
for (initialization; condition; update )  
{  
    statement1;  
    statement2;  
    .....  
}
```

```
initialization;  
while (condition)  
{  
    statement1;  
    statement2;  
    .....  
    update;  
}
```

- A for loop can be changed to while loop and vice-versa.
 - Example: note the curly braces in the while loop

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

int main()
{
    for (int x = 3; x < 10; x++)
        cout << x << endl;
    return 0;
}
```

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

int main()
{
    int x = 3;
    while (x < 10)
    {
        cout << x << endl;
        x++;
    }
    return 0;
}
```

Interrupting loops with break and continue

- **Why do we need to interrupt loops?**
 - Improve efficiency by skipping part of the computation of the loop
- **Two statements**
 - **break**
Terminate a loop completely
 - **continue**
Skip part of an iteration of a loop
(Each repetition of the code is called an iteration.)

Interrupting loops with **break**

- **break can be used to terminate a for/while loop immediately**
 - **When do we need break in a loop?**
 - => When a loop has fulfilled its purposes, but the loop is still running.**

Interrupting loops with break

- **Example:** To compute the sum of integers (starting from 1, 2, 3 ...). Once the sum is larger than 500, print out the result and stop the loop immediately

```
sum = 0;
for (int t = 1; t < 100; t++)
{
    sum += t;
    if (sum > 500)
    {
        cout << "The sum of the first " << t
            << " positive integers is larger than 500" << endl;
        break;
    }
}
```

Interrupting loops with break

- Example (nested loops):

```
while (condition1) // or you have the for loop here.
{
    while (condition2) // or you have the for loop here.
    {
        // some codes .....
        break;
        // some codes .....
    }
    // some codes ....
}
```

In the above code, the break in the inner loop does NOT affect the outer loop.

Interrupting loops with `continue`

- A loop may consist of a number of iterations
- In the middle of an iteration, if the purpose of that iteration has been fulfilled, `continue` can be used to skip the rest of that iteration

Interrupting loops with continue

- **Example: print the numbers in the range 1, 2, 3, ... 100 to the screen which are not divisible by 13.**

```
for (int i = 1; i <= 100; i++)  
{  
    if ( (i % 13) == 0)  
        continue;  
    cout << i << endl;  
}
```

A common mistake when using control statements

- A common mistake when using control statements in C++ program is the improper use of braces
- To reduce the errors, you may always use braces when there is control statement.

```
if (i > 3)
{
    a = 6;
}
for (i = 0; i < 5; i++)
{
    t = t+2;
}
```

- To reduce the errors, you should follow the format given in the following slides so that you can easily identify the statement block following control statements

Format the code properly!!!

Format your code properly!!!

To make your code readable and to reduce the errors, **proper indent is necessary.**

There are several ways to indent the C++ code.

We can follow the following Allman style when we write C++ code.

- 0) For each statement block, two corresponding braces are on the same vertical line.
- 1) In each statement block, all the statements are indented to the right by one tab position (relative to the braces).

```
{
    statement1;
    statement2;
    {
        statementa;
        statementb;
    }
}
```

Format your code properly!!!

2) When we have control statements, the first brace is on the next line, and it is indented to the same level as the control statement, Example:

```
while (condition)
{
    statement1;
    statement2;
    statement3;
}
```

Format your code properly!!!

3) Here is an example of a main function:

```
int main( )
{
    statement1;
    statement2;

    while (condition)
    {
        statement21;
        statement22;

        for (initialization; condition; update)
        {
            statement51;
            statement52;
            while (condition)
            {
                statement61;
                statement62;
            }
        }
    }
}
```