The Basics of C++

闫宏飞

Institute of Network
Peking University

Oct 16, 2007
导航

1. 打印一行文本程序
2. 修改前面的程序
3. 整数相加程序
4. 算数和逻辑运算符
5. 实例学习: ATM
OBJECTIVES

- To write simple computer programs in C++.
- To use fundamental types.
- To use arithmetic operators.
- The precedence of arithmetic operators.
- To write simple decision-making statements.
Comments

- Explain programs to other programmers
  - Improve program readability

- Ignored by compiler

- Single-line comment
  - Begin with //

- Multi-line comment
  - Start with /*
  - End with */
// Fig. 2.1: fig02_01.cpp
// Text--printing program.

#include <iostream> // allows program to output data to the screen

// function main begins program execution
int main() {
    std::cout << "Welcome to C++!\n"; // display message

    return 0; // indicate that program ended successfully
}

// end function main
Preprocessor directives & White space

- Processed by preprocessor before compiling
- Begin with `#`
- Example
  - `#include <iostream>`
    - Tells preprocessor to include the input/output stream header file `<iostream>`
- White space
  - Blank lines, space characters and tabs
  - Used to make programs easier to read
  - Ignored by the compiler
Function main & Statements

Function main

- A part of every C++ program
  - Exactly one function in a program must be `main`

- Can “return” a value

Example

- `int main()`
  - This main function returns an integer (whole number)

- Body is delimited by braces (`{}`).

Statements

- Instruct the program to perform an action

- All statements end with a semicolon (`;`).
Namespace & Standard output stream object

Namespace

- std::
  - Specifies using a name that belongs to "namespace" std
  - Can be removed through use of using statements

Standard output stream object

- std::cout
  - “Connected” to screen
  - Defined in input/output stream header file <iostream>
Stream insertion operator & Escape characters

**Stream insertion operator `<<`**

- Value to right (right operand) inserted into left operand
- Example, `std::cout << "Hello";`
  - Inserts the string "Hello" into the standard output
  - Displays to the screen

**Escape characters**

- A character preceded by `"\"`
- Example, `"\n"`
  - Cursor moves to beginning of next line on the screen
**return statement**

- One of several means to exit a function
- When used at the end of `main`
  - The value 0 indicates the program terminated successfully
  - Example: `return 0;`
**Escape sequence**

<table>
<thead>
<tr>
<th>Escape</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\n</td>
<td>Newline. Position the screen cursor to the beginning of the next line.</td>
</tr>
<tr>
<td>\t</td>
<td>Horizontal tab. Move the screen cursor to the next tab stop.</td>
</tr>
<tr>
<td>\r</td>
<td>Carriage return. Position the screen to the beginning of the current line; do not advance to the next line.</td>
</tr>
<tr>
<td>\a</td>
<td>Alert. Sound the system bell.</td>
</tr>
<tr>
<td>\</td>
<td>Backslash. Used to print a backslash character.</td>
</tr>
<tr>
<td>'</td>
<td>Single quote. Used to print a single quote character.</td>
</tr>
<tr>
<td>&quot;</td>
<td>Double quote. Used to print a double quote characters.</td>
</tr>
</tbody>
</table>
// Fig. 2.3: fig02_03.cpp
// Printing a line of text with multiple statements.

#include <iostream> // allows program to output data to the screen

// function main begins program execution
int main()
{
    std::cout << "Welcome to C++!\n";
    std::cout << "to\nC++!\n";

    return 0; // indicate that program ended successfully

} // end function main
// Fig. 2.4: fig02_04.cpp
// Printing multiple lines of text with a single statement.
#include <iostream>  // allows program to output data to the screen

// function main begins program execution
int main()
{
    std::cout << "Welcome\nto\n\nC++!\n";

    return 0;  // indicate that program ended successfully

}  // end function main
Variables

Location in memory where value can be stored

**Common data types (fundamental, primitive or built-in)**

- **int** - integer number
- **char** - characters
- **double** - floating point numbers

**Declare variables with name and data type before use**

- int integer1;
- int integer2;
- int sum;
Variables (Cont.)

Can declare several variables of same type in one declaration

- Comma-separated list

  `int integer1, integer2, sum;`

Variable names

- Valid identifier
  - Series of characters (letters, digits, underscores)
  - Cannot begin with digit
  - Case sensitive
// Addition program that displays the sum of two numbers.
#include <iostream> // allows program to perform input and output

int main()
{
    // variable declarations
    int number1; // first integer to add
    int number2; // second integer to add
    int sum; // sum of number1 and number2

    std::cout << "Enter first integer: "; // prompt user for data
    std::cin >> number1; // read first integer from user into number1

    std::cout << "Enter second integer: "; // prompt user for data
    std::cin >> number2; // read second integer from user into number2

    sum = number1 + number2; // add the numbers; store result in sum

    std::cout << "Sum is " << sum << std::endl; // display sum; end line
} // end function main
Input stream object

std::cin from <iostream>

- Usually connected to keyboard
- Stream extraction operator `>>`
  - Waits for user to input value, press Enter (Return) key
  - Stores value in variable to right of operator
  - Converts value to variable data type
- Example, `std::cin >> number1`
  - Reads an integer typed at the keyboard
  - Stores the integer in variable `number1`
Assignment operator = & Stream manipulator std::endl

Assignment operator =
- Assigns value on left to variable on right
- Binary operator (two operands)
- Example: `sum = variable1 + variable2;`
  - Add the values of variable1 and variable2
  - Store result in sum

Stream manipulator std::endl
- Outputs a newline
- Flushes the output buffer
Arithmetic operator

*, Multiplication
Division, Integer division truncates remainder

- $7 / 5$ evaluates to $1$

%, Modulus operator returns remainder

- $7 \% 5$ evaluates to $2$
## Precedence of arithmetic operators

<table>
<thead>
<tr>
<th>Operator(s)</th>
<th>Operation(s)</th>
<th>Order of evaluation(precedence)</th>
</tr>
</thead>
<tbody>
<tr>
<td>()</td>
<td>Parentheses</td>
<td>Evaluated first.</td>
</tr>
<tr>
<td>*</td>
<td>Multiplication</td>
<td>Evaluated second.</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
<td>Evaluated last. If there are several, they are evaluated left to right.</td>
</tr>
<tr>
<td>%</td>
<td>Modulus</td>
<td>Evaluated last. If there are several, they are evaluated left to right.</td>
</tr>
<tr>
<td>+</td>
<td>Addition</td>
<td>Evaluated last. If there are several, they are evaluated left to right.</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction</td>
<td>Evaluated last. If there are several, they are evaluated left to right.</td>
</tr>
</tbody>
</table>
Decision Making: Equality and Relational Operators

**Condition**

- Expression can be either **true** or **false**
- Can be formed using equality or relational operators

**if statement**

- If condition is **true**, body of the if statement executes
- If condition is **false**, body of the if statement does not execute
# Equality and relational operators

<table>
<thead>
<tr>
<th>Standard algebraic equality or relational operator</th>
<th>Meaning of C++ condition</th>
<th>Sample C++ condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>x is greater than y</td>
<td>x &gt; y</td>
</tr>
<tr>
<td>&lt;</td>
<td>x is less than y</td>
<td>x &lt; y</td>
</tr>
<tr>
<td>&gt;=</td>
<td>x is greater than or equal to y</td>
<td>x &gt;= y</td>
</tr>
<tr>
<td>&lt;=</td>
<td>x is less than or equal to y</td>
<td>x &lt;= y</td>
</tr>
<tr>
<td>=</td>
<td>x is equal to y</td>
<td>x == y</td>
</tr>
<tr>
<td>!=</td>
<td>x is not equal to y</td>
<td>x != y</td>
</tr>
</tbody>
</table>

## Relational operators

- \( > \): \( x > y \) \( x \) is greater than \( y \).
- \( < \): \( x < y \) \( x \) is less than \( y \).
- \( \geq \): \( x \geq y \) \( x \) is greater than or equal to \( y \).
- \( \leq \): \( x \leq y \) \( x \) is less than or equal to \( y \).

## Equality operators

- \( = \): \( x = y \) \( x \) is equal to \( y \).
- \( \neq \): \( x \neq y \) \( x \) is not equal to \( y \).
```cpp
#include <iostream> // allows program to perform input and output
using namespace std;

int main()
{
    int number1, number2; // first and second integer to compare
    cout << "Enter two integers to compare: "; // prompt user for data
    cin >> number1 >> number2; // read two integers from user
    if ( number1 == number2 )
        cout << number1 << " == " << number2 << endl;
    if ( number1 != number2 )
        cout << number1 << " != " << number2 << endl;
    if ( number1 < number2 )
        cout << number1 << " < " << number2 << endl;
    if ( number1 > number2 )
        cout << number1 << " > " << number2 << endl;
    if ( number1 <= number2 )
        cout << number1 << " <= " << number2 << endl;
    if ( number1 >= number2 )
        cout << number1 << " >= " << number2 << endl;
} // end function main
```
Precedence and associativity of the operators discussed so far

<table>
<thead>
<tr>
<th>Operators</th>
<th>Associativity</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>()</td>
<td>left to right</td>
<td>parentheses</td>
</tr>
<tr>
<td>*/%</td>
<td>left to right</td>
<td>multiplicative</td>
</tr>
<tr>
<td>++</td>
<td>left to right</td>
<td>additive</td>
</tr>
<tr>
<td>&lt;&lt;&gt;&gt;</td>
<td>left to right</td>
<td>stream insertion/extraction</td>
</tr>
<tr>
<td>&lt;&lt;= &gt; &gt;=</td>
<td>left to right</td>
<td>relational</td>
</tr>
<tr>
<td>== !=</td>
<td>right to left</td>
<td>equality</td>
</tr>
<tr>
<td>=</td>
<td>right to left</td>
<td>assignment</td>
</tr>
</tbody>
</table>
Case Study: ATM

Object-oriented design (OOD) process using UML

- Performed in chapters 3 to 7, 9 and 13
- Requirements document
  - Specifies overall purpose and what the system must do

Object-oriented programming (OOP) implementation

- Complete implementation in appendix G
Case Study: ATM (Cont.)

Requirements document

- New automated teller machine (ATM)
- Allows basic financial transaction
  - View balance, withdraw cash, deposit funds
- User interface
  - Display screen, keypad, cash dispenser, deposit slot
- ATM session
  - Authenticate user, execute financial transaction
PLEASE change to CppHTP5e_02.ppt
and continue on page 75