**Chapter 6 Branch Statements and Logic Operations**

C++ provides that the || operator is a **sequence point**. That is, any value changes indicated on the left side take place before the right side is evaluated. (Or in the newer parlance of C++11, the sub-expression to the left of the operator is sequenced before the sub-expression to the right.) For example, consider the following expression:

i++ < 6 || i == j

Suppose i originally has the value 10. **By the time the comparison with j takes place, i has the value 11**.

#include <cctype> // prototypes for character functions

cin.get(ch); // get first character

while (ch != '@') // test for sentinel

{

if(isalpha(ch)) // is it an alphabetic character?

chars++;

else if(isspace(ch)) // is it a whitespace character?

whitespace++;

else if(isdigit(ch)) // is it a digit?

digits++;

else if(ispunct(ch)) // is it punctuation?

punct++;

else

others++;

cin.get(ch); // get next character

}

switch(choice)

{

case 'a': **\\执行到这一分支时，会接着执行A分支，然后在那里回到switch判断处；**

case 'A': cout << "\a\n";

break;

case 'r':

case 'R': report();

break;

case 'l':

case 'L': cout << "The boss was in all day.\n";

break;

case 'c':

case 'C': comfort();

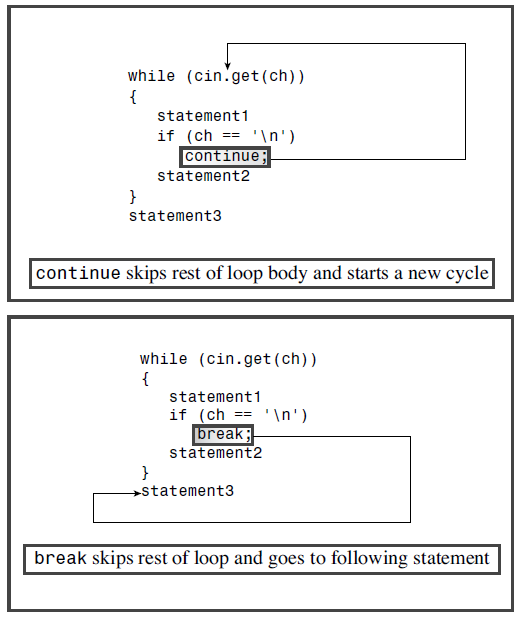
break;

default : cout << "That's not a choice.\n";

}

* The **break** and **continue** Statements

***Break*** causes program execution to pass to the next statement following the switch or the loop. The ***continue*** statement is used in loops and causes a program to skip the rest of the body of the loop and then start a new loop cycle.



In a ***for*** loop, the continue statement makes the program skip directly to the update expression and then to the test expression. For a while loop, however, continue makes the program go directly to the test expression. So any update expression in a while loop body following the ***continue*** would be skipped. In some cases, that could be a problem.

* Number-Reading Loops

Say you’re preparing a program to read a series of numbers into an array. You want to give the user the option to terminate input before filling the array. One way to do this is utilize how cin behaves. Consider the following code:

int n;

cin >> n;

What happens if the user responds by entering a word instead of a number? **Four things occur** in such a mismatch:

1. The value of n is left unchanged;
2. The mismatched input is left in the input queue;
3. An error flag is set in the cin object;
4. The call to the cin method, if converted to type bool, returns false;

The fact that the method returns false means that you can use non-numeric input to terminate a number-reading loop. The fact that non-numeric input sets an error flag means that you have to reset the flag before the program can read more input. The clear() method, which also resets the end-of-file (EOF) condition, resets the bad input flag.

Listing 6.13 uses a loop that terminates if the array is full or if you enter non-numeric input.

**< Listing6-13.cpp >**

#include <iostream>

const int Max = 5;

int main()

{

using namespace std;

double fish[Max];

cout << "Please enter the weights of your fish.\n";

cout << "You may enter up to " << Max

<< " fish <q to terminate>.\n";

cout << "fish #1: ";

int i = 0;

**while (i < Max && cin >> fish[i])** {

if (++i < Max)

cout << "fish #" << i+1 << ": ";

}

// calculate average

double total = 0.0;

for (int j = 0; j < i; j++)

total += fish[j];

// report results

if (i == 0)

cout << "No fish\n";

else

cout << total / i << " = average weight of "

<< i << " fish\n";

cout << "Done.\n";

return 0;

}

自己电脑测试结果：

momo@HMI:~/C++PrimerPlus/Chapter6$ ./List6-13

Please enter the weights of your fish.

You may enter up to 5 fish <q to terminate>.

fish #1: **10** (回车)

fish #2: **20**(回车)

fish #3: **30**(回车)

fish #4: **40**(回车)

fish #5: **w**(回车)

25 = average weight of 4 fish

Done.

momo@HMI:~/C++PrimerPlus/Chapter6$ ./List6-13

Please enter the weights of your fish.

You may enter up to 5 fish <q to terminate>.

fish #1: **10**(回车)

fish #2: **20**(回车)

fish #3: **30**(回车)

fish #4: **w**(回车)

20 = average weight of 3 fish

Done.

momo@HMI:~/C++PrimerPlus/Chapter6$ ./List6-13

Please enter the weights of your fish.

You may enter up to 5 fish <q to terminate>.

fish #1: **10**

fish #2: **10**

fish #3: **20**

fish #4: **(回车)**

^Z**(这里按了Ctrl+C)**

[6]+ Stopped ./List6-13

momo@HMI:~/C++PrimerPlus/Chapter6$

The expression **cin >> fish[i] in Listing 6.13 is really a cin method function call**, and **the function returns cin**. **If cin is part of a test condition, it’s converted to type bool**. The conversion value is true if input succeeds and false otherwise. A false value for the expression terminates the loop.

**Listing6-13 example *doesn’t attempt to read any input after non-numeric input***. Let’s look at a case that does.

You are required to submit exactly five golf scores to a C++ program to establish your average. If a user enters non-numeric input, the program should object, insisting on numeric input. As you’ve seen, you can use the value of a cin input expression to test for non-numeric input. **Suppose the program finds that the user enters the wrong stuff. It needs to take three steps:**

1. Reset cin to accept new input;
2. Get rid of the bad input;
3. Prompt the user to try again;
4. **< Listing6-14.cpp >**

#include <iostream>

const int Max = 5;

int main()

{

using namespace std;

int golf[Max];

cout << "Please enter your golf scores.\n";

cout << "You must enter " << Max << " rounds.\n";

int i;

for (i = 0; i < Max; i++)

{

cout << "round #" << i+1 << ": ";

while (!(cin >> golf[i]))

{

cin.clear(); // reset input

**while (cin.get() != '\n')**

**continue;**  // get rid of bad input

cout << "Please enter a number: ";

}

}

// calculate average

double total = 0.0;

for (i = 0; i < Max; i++)

total += golf[i];

// report results

cout << total / Max << " = average score " << Max << " rounds\n";

return 0;

}

当输入非整数时，the expression !(cin >> golf[i]) is true, and the program enters the inner while loop. The first statement in the loop uses the clear() method to reset input. If you omit this statement, the program refuses to read any more input. **Next, the program uses cin.get() in a while loop to read the remaining input through the end of the line**. This gets rid of the bad input, along with anything else on the line.

* Simple File Input/Output

1. **Writing to a Text File**

In short, these are the main steps for using file output:

1. Include the fstream header file.
2. Create an ofstream object.
3. Associate the ofstream object with a file.
4. Use the ofstream object in the same manner you would use cout.

**<List6-15.cpp>**

#include <iostream>

**#include <fstream>** **// for file I/O**

int main()

{

using namespace std;

char automobile[50];

int year;

double a\_price;

double d\_price;

**ofstream** outFile; // create object for output

**outFile.open("carinfo.txt"); // associate with a file**

cout << "Enter the make and model of automobile: ";

**cin.getline**(automobile, 50);

cout << "Enter the model year: ";

cin >> year;

cout << "Enter the original asking price: ";

cin >> a\_price;

d\_price = 0.913 \* a\_price;

// display information on screen with cout

**cout << fixed;**

**cout.precision(2);**

**cout.setf(ios\_base::showpoint);**

cout << "Make and model: " << automobile << endl;

cout << "Year: " << year << endl;

cout << "Was asking $" << a\_price << endl;

cout << "Now asking $" << d\_price << endl;

// now do exact same things using outFile instead of cout

**outFile << fixed;**

outFile.precision(2); // 小数点后两位

outFile.setf(ios\_base::showpoint); // 用带小数点的自然计数法

**outFile << "Make and model: " << automobile << endl;**

outFile << "Year: " << year << endl;

outFile << "Was asking $" << a\_price << endl;

outFile << "Now asking $" << d\_price << endl;

**outFile.close(); // done with file**

return 0;

}

书上的测试结果：

[在显示器上]

Enter the make and model of automobile: **Flitz Perky**

Enter the model year: **2009**

Enter the original asking price: **13500**

Make and model: Flitz Perky

Year: 2009

Was asking $13500.00

Now asking $12325.50

[在文件里]

Make and model: Flitz Perky

Year: 2009

Was asking $13500.00

Now asking $12325.50

As you can see, outFile sends precisely the same sequence of characters to the carinfo.txt file that cout sends to the display.

**outFile.open("carinfo.txt"); // associate with a file**

这一句把一个文件关联到了ofstream outFile上；

**outFile.open("carinfo.txt");**

In this case, the **file carinfo.txt does not exist before the program runs**. In this circumstance, the open() method creates a brand new file by that name. **When the file carinfo.txt exists**, what happens if you run the program again? **By default, open() first truncates the file; that is, it trims carinfo.txt to zero length, discarding the current contents**.

1. **Reading from a Text File**

#include <iostream>

#include <fstream> // file I/O support

#include <cstdlib> // support for exit()

const int SIZE = 60;

int main()

{

using namespace std;

char filename[SIZE];

**ifstream inFile;** // object for handling file input

cout << "Enter name of data file: ";

**cin.getline(filename, SIZE);**

**inFile.open(filename);** // associate inFile with a file

if (!**inFile.is\_open()**) // failed to open file

{

cout << "Could not open the file " << filename << endl;

cout << "Program terminating.\n";

exit(EXIT\_FAILURE);

}

double value;

double sum = 0.0;

int count = 0; // number of items read

**inFile >> value;** **// get first value**

while (**inFile.good()**) **// while input good, no mismatch, and not at EOF**

{

++count; // one more item read

sum += value; // calculate running total

**inFile >> value;** **// get next value**

}

**// Below checks why while loop ends**

if (**inFile.eof()**)

cout << "End of file reached.\n";

else if (**inFile.fail()**)

cout << "Input terminated by data mismatch.\n";

else

cout << "Input terminated for unknown reason.\n";

if (count == 0)

cout << "No data processed.\n";

else

{

cout << "Items read: " << count << endl;

cout << "Sum: " << sum << endl;

cout << "Average: " << sum / count << endl;

}

inFile.close(); // finished with the file

return 0;

}

这里是从文件输入, 所以定义的是ifstream, 前面的例子中因为是输出, 所以定义的是ofstream;

if (!**inFile.is\_open()**) // failed to open file

What happens if you attempt to open a **non-existent file for input**? This error causes subsequent attempts to use the ifstream object for input to fail. The preferred way to check whether a file was opened successfully is to use the ***is\_open()*** method.

The ***is\_open()*** method **returns true if the file was opened successfully**, so the expression !inFile.is\_open() is true if the attempt fails. The exit() function is prototyped in the ***cstdlib header file***, which also defines **EXIT\_FAILURE as an argument value used to communicate with the operating system**. The exit() function terminates the program. The is\_open() method is relatively new to C++. If your compiler doesn’t support it, you can use the older good() method instead.

***eof()* tests just for the EOF** and **fail() tests for both the EOF and type mismatch**, this code tests for the EOF first. That way**, if execution reaches the *else if* test, the EOF has already been excluded**, so a true value for fail() unambiguously identifies type mismatch as the cause of loop termination.

It’s particularly important that you understand that *good()* reports on the most recent attempt to read input. That means there should be an attempt to read input immediately before applying the test. A standard way of doing that is to have one input statement immediately before the loop, just before the first execution of the loop test, and a second input statement at the end of the loop, just before subsequent executions of the loop test:

// standard file-reading loop design

inFile >> value; // get first value

while (inFile.good()) // while input good and not at EOF

{

// loop body goes here

inFile >> value; // get next value

}

**You can condense this somewhat by using the fact** that the following expression evaluates to inFile and that ***inFile*, when placed in a context in which a bool value is expected, evaluates to inFile.good()**—that is, to true or false:

inFile >> value

**Thus, you can replace the two input statements with a single input statement used as a loop test**. That is, you can replace the preceding loop structure with this:

// abbreviated file-reading loop design

// omit pre-loop input

**while (inFile >> value)** // read and test for success

{

// loop body goes here

// omit end-of-loop input

}