**Chapter 5 Loops and Relational Expressions**

C++ defines the value of an assignment expression to be the value of the member on the left.

maids = (cooks = 4) + 3;

The expression cooks = 4 has the value 4, so maids is assigned the value 7.

Finally, as mentioned previously, relational expressions such as x < y evaluate to the bool values true or false.

X = 100;

cout << "Now x = " << x << endl;

cout << "The expression x < 3 has the value ";

cout << (x < 3) << endl;

cout << "The expression x > 3 has the value ";

cout << (x > 3) << endl;

cout.setf(ios\_base::boolalpha); //a newer C++ feature

cout << "The expression x < 3 has the value ";

cout << (x < 3) << endl;

cout << "The expression x > 3 has the value ";

cout << (x > 3) << endl;

书上的输出结果为：

Now x = 100

The expression x < 3 has the value 0

The expression x > 3 has the value 1

The expression x < 3 has the value false

The expression x > 3 has the value true

下面的例子是计算阶乘的：

#include <iostream>

const int ArSize = 16; // example of external declaration

int main()

{

long long factorials[ArSize];

factorials[1] = factorials[0] = **1LL**; // 1 of long long type

for (int i = 2; i < ArSize; i++)

factorials[i] = i \* factorials[i-1];

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

std::cout << i << "! = " << factorials[i] << std::endl;

return 0;

}

It’s usually a good idea to **define a const value to represent the number of elements in an array**. You can use the const value in the array declaration and in all other references to the array size, such as in a for loop.

下面的例子是键盘输入string然后倒着输出的：

#include <iostream>

#include <string>

int main()

{

using namespace std;

cout << "Enter a word: ";

**string word;**

**cin >> word;**

// display letters in reverse order

for (int i = word.size() - 1; i >= 0; i--)

cout << word[i];

cout << "\nBye.\n";

return 0;

}

* **The Increment (++) and Decrement (--) Operators**

The prefix version comes before the operand, as in ++x. The postfix version comes after the operand, as in x++.The two versions have the same effect on the operand, but they differ in terms of when they take place.

Roughly speaking, the notation **a++ means “use the current value of a in evaluating an expression, and then increment the value of a**”. Similarly, the notation **++b means “first increment the value of b and then use the new value in evaluating the expression**”.

int x = 5;

int y = ++x; // change x, then assign to y; y is 6, x is 6

int z = 5;

int y = z++; // assign to y, then change z; **y is 5, z is 6**

++这种操作在C++里叫做side effect; 与之对应的是sequence point, which is a point in program execution at which all side effects are guaranteed to be evaluated before going on to the next step.

In C++ the **semicolon (;) in a statement marks a sequence point**. That means all changes made by assignment operators, increment operators, and decrement operators in a statement must take place before a program proceeds to the next statement. Also **the end of any full expression is a sequence point**.

**Test condition for a while loop is a full expression.**

#include <iostream>

using namespace std;

int main()

{

int i = 0;

while (**i++ < 5**)

cout << i << endl;

return 0;

}

自己电脑上运行结果：

1

2

3

4

5

**如果改成while (++i < 5), 输出是 1 2 3 4**

#include <iostream>

using namespace std;

int main()

{

int i = 0;

**for (i = 0; i < 5; i++)**

cout << i << endl;

return 0;

}

自己电脑上运行结果：

0

1

2

3

4

**如果改成for (i = 0; i < 5; ++i) 输出也是 0 1 2 3 4**

所以在判断条件里用++对于while是有区别的, 对for没有；

* **The Increment/Decrement Operators and Pointers**

double arr[5] = {21.1, 32.8, 23.4, 45.2, 37.4};

double \*pt = arr; // pt points to arr[0], i.e. to 21.1, **array name is address**

++pt; // pt points to arr[1], i.e. to 32.8

Applying both \* and ++ to a pointer raises the questions of what gets dereferenced and what gets incremented.Those actions are determined by the placement and precedence of the operators.

1. The **prefix increment, prefix decrement, and dereferencing** operators all have the same precedence and associate **from right to left**;

double x = \*++pt; // increment pointer, take the value; i.e., arr[2], or 23.4

++\*pt; // increment the pointed to value; i.e., change 23.4 to 24.4

1. The **postfix increment and decrement** operators both have the same precedence, which is **higher than the prefix precedence**. These two operators associate from **left to right**;

(\*pt)++; // increment pointed-to value

x = \*pt++; // **dereference original location, then increment pointer**

If ***pt*** points to the first member of an array, ***++pt*** changes pt so that it points to the second member.

int i, j;

for (j = 0, i = word.size() - 1; j < i; --i, ++j) **// comma, 逗号, 可以这么用**

{

temp = word[i];

word[i] = word[j];

word[j] = temp;

}

**C++ does provide the operator comma (,) with two additional properties**. First, it guarantees that the first expression is evaluated before the second expression.

i = 20, j = 2 \* i; // i set to 20, then j set to 40

Second, C++ states that the value of a comma expression is the value of the second part of the expression. The value of the preceding expression, for example, is 40 because that is the value of j = 2 \* i.

* **Comparing C-Style Strings**

Suppose you want to see if a string in a character array is the word *mate*. If **word is the array name**, the following test might not do what you think it should do:

word == "mate"

Remember that the **name of an array is a synonym for its address**. Similarly, **a quote quoted string constant is a synonym for its address**. **Thus, the preceding relational expression doesn’t test whether the strings are the same; it checks whether they are stored at the same address**. The answer to that is no, even if the two strings have the same characters.

Because C++ handles C-style strings as addresses, you can go to the C-style string library and use the strcmp() function to compare strings. This function takes two string addresses as arguments. That means the arguments can be **pointers**, **string constants**, or **character array names**.

C-style strings are defined by the terminating null character, not by the size of the containing array. This means that two strings below are identical even if they are contained in differently sized arrays:

char big[80] = "Daffy"; // 5 letters plus \0

char little[6] = "Daffy"; // 5 letters plus \0

// 下面的code也是可以的

for (ch = ‘a’; ch <= ’z’; ch++)

cout << ch;

* **Comparing string Class Strings**

#include <iostream>

#include <string> // string class

int main()

{

using namespace std;

string word = "?ate";

for (char ch = ‘a’; **word != "mate"**; ch++) **// C++里可以这么用**

{ **// 如果是C，要用strcmp()**

cout << word << endl;

word[0] = ch;

}

cout << "After loop ends, word is " << word << endl;

return 0;

}

The way the string class **overloads the != operator** allows you to use it as long as **at least one of the operands is a string object**; the remaining operand can be either a string object or a C-style string. 这里的关键是 != 操作符的重载,而不是string本身.

* The ***while*** Loop

int i = 0; // start at beginning of string

while (**name[i] != ‘\0’**) // process to end of string

{

cout << name[i] << ": " << **int(name[i])** << endl;

i++;

}

for (init-expression; test-expression; update-expression)

{

statement(s)

}

could be rewritten this way:

init-expression;

while (test-expression)

{

statement(s)

update-expression;

}

C++ library defines ***CLOCKS\_PER\_SEC***, which equals the number of system time units per second. And function ***clock (),*** which returns the system time elapsed since a program started execution.

#include <iostream>

#include <ctime> // describes clock() function, clock\_t type

int main()

{

using namespace std;

cout << "Enter the delay time, in seconds: ";

float secs;

cin >> secs;

**clock\_t delay = secs \* CLOCKS\_PER\_SEC**; // convert to clock ticks

cout << "starting\a\n";

**clock\_t start = clock();**

while (**clock() - start < delay** ) // wait until time elapses

; // note the semicolon

cout << "done \a\n";

return 0;

}

* Do-while Loop

do

body

while (test-expression);

do

{

cin >> n; // execute body

} while (n != 7); // then test

cout << "Yes, 7 is my favorite.\n" ;

* The Range-Based for Loop (C++11)

The C++11 adds a new form of loop called the **range-based for loop**. It simplifies one common loop task—that of doing something with each element of an array, or, more generally, of **one of the container classes**, such as **vector or array**. Here is an example:

#include <iostream>

using namespace std;

int main()

{

double prices[5] = {1.11,2.22,3.33,4.44,5.55};

for (**double index : prices**)

cout << index << endl;

return 0;

}

编译：

momo@HMI:~/C++PrimerPlus/Chapter5$**g++ TestRangeBased.cpp -o TestRangeBased**

TestRangeBased.cpp: In function ‘int main()’:

TestRangeBased.cpp:8:22: error: range-based ‘for’ loops are not allowed in C++98 mode

for (double index : prices)

^

momo@HMI:~/C++PrimerPlus/Chapter5$ g++ **-std=c++11** TestRangeBased.cpp -o TestRangeBased

momo@HMI:~/C++PrimerPlus/Chapter5$ ./TestRangeBased

**1.11**

**2.22**

**3.33**

**4.44**

**5.55**

To modify array values, you need a different syntax for the loop variable:

for (double &x : prices)

x = x \* 0.80; //20% off sale

The & symbol identifies x as a reference variable, a topic we’ll discuss in Chapter 8.

* Loops and Text Input

Let’s look at one of the most common and important tasks assigned to loops: reading text character-by-character from a file or from the keyboard. The **cin object** supports **three distinct modes** of single-character input, each with a different user interface.

1. **Using Unadorned cin for Input**

If a program is going to use a loop to read text input from the keyboard, it has to have some way of knowing when to stop. How can it know when to stop? One way is to choose some special character, sometimes called a sentinel character, to act as a stop sign.

**<List5-16.cpp>**

#include <iostream>

int main()

{

using namespace std;

char ch;

int count = 0; // use basic input

cout << "Enter characters; enter # to quit:\n";

cin >> ch; // get a character

while **(ch != '#')** // test the character

{

cout << ch; // echo the character

++count; // count the character

cin >> ch; // get the next character

}

cout << endl << count << " characters read\n";

return 0;

}

在自己电脑上测试结果：

momo@HMI:~/C++PrimerPlus/Chapter5$ ./List5-16

Enter characters; enter # to quit:

abcdef # (回车)// 中间是空格

abcdef

6 characters read

momo@HMI:~/C++PrimerPlus/Chapter5$ ./List5-16

Enter characters; enter # to quit:

abcdef(回车)

abcdef(回车)

#(回车)

6 characters read

momo@HMI:~/C++PrimerPlus/Chapter5$ ./List5-16

Enter characters; enter # to quit:

abc def(回车)

abcdef#

(回车)

6 characters read

momo@HMI:~/C++PrimerPlus/Chapter5$ ./List5-16

Enter characters; enter # to quit:

abc def**#**def

abcdef

6 characters read

书上的结果：

Enter characters; enter # to quit:

see ken run#really fast

seekenrun

9 characters read

**So why does the program omit the spaces on output?** Blame cin. **When reading type char values**, just as when reading other basic types, **cin skips over spaces and newline characters**. The spaces in the input are not echoed, so they are not counted.

To further complicate things, **the input to cin is buffered**. **That means the characters you type don’t get sent to the program until you press Enter**. **This is why you are able to type characters after the # when running the program in Listing 5.16**. **After you press Enter, the whole sequence of characters is sent to the program, but the program quits processing the input after it reaches the # character.**

**在上一章里讲过用cin往字符串里输入一行时, 一遇到white space就停了, 注意那里说的是往字符串里输，这里说的是When reading type char values.**

1. **cin.get(char) to the Rescue**

Usually, programs that read input character-by-character need to examine every character, including spaces, tabs, and newlines. Function **cin.get(ch) reads the next character, even if it is a space, from the input and assigns it to the variable ch**.

**<List5-17.cpp>**

momo@HMI:~/C++PrimerPlus/Chapter5$ cat List5-17.cpp

#include <iostream>

int main()

{

using namespace std;

char ch;

int count = 0;

cout << "Enter characters; enter # to quit:\n";

**cin.get(ch)**; // use the cin.get(ch) function

**while (ch != '#')**

{

cout << ch;

++count;

cin.get(ch); // use it again

}

cout << endl << count << " characters read\n";

return 0;

}

在自己电脑上测试结果：

momo@HMI:~/C++PrimerPlus/Chapter5$ ./List5-17

Enter characters; enter # to quit:

(回车)

abc def# dfg

abc def

**8** characters read **// 8个字符是包含一开始的回车键的**

momo@HMI:~/C++PrimerPlus/Chapter5$ ./List5-17

Enter characters; enter # to quit:

(回车)

abc def #(回车)

abc def

**9** characters read **// 9个字符是包含一开始的回车键的**

momo@HMI:~/C++PrimerPlus/Chapter5$ ./List5-17

Enter characters; enter # to quit:

abc(Tab键)def(回车)

abc(Tab键)def

#

8 characters read **//Tab键和回车键也被计入了**

momo@HMI:~/C++PrimerPlus/Chapter5$

这里，cin.get()在一开始遇到回车键并没有立即退出，这和第三章里输入字符串时不太一样；

**If you are familiar with C, this program may strike you as terribly wrong**. The cin.get(ch) call places a value in the ch variable, which means it alters the value of the variable.

**In C you must pass the address of a variable to a function if you want to change the value of that variable**. **But the call to cin.get() in Listing 5.17 passes ch, not &ch**. In C, code like this won’t work. **In C++ it can work, provided that the function declares the argument as a reference**. The reference type is something that C++ added to C. The iostream header file declares the argument to cin.get(ch) as a reference type, so this function can alter the value of its argument. You’ll learn the details in Chapter 8. Meanwhile, the C mavens among you can relax; **ordinarily, argument passing in C++ works just as it does in C. For cin.get(ch), however, it doesn’t**.

**Which cin.get() Should You Use?**

**Chapter 4 uses this code**:

char name[ArSize];

...

cout << "Enter your name:\n";

cin.get(name, ArSize).get();

**This version of cin.get()** takes two arguments: *the array name*, which is the *address of the string* (technically, type char\*), and *ArSize*, which is an integer of type int. (Recall that the name of an array is the address of its first element, so the name of a character array is type char \*.) Then the program uses cin.get() with no arguments.

**And most recently, we’ve used cin.get() this way**:

char ch;

cin.get(ch);

**This time cin.get() has one argument, and it is type char.**

**It is function overloading**, which allows you to create different functions that have the same name, provided that they have different argument lists. If, for example, you use cin.get(name, ArSize) in C++, the compiler finds the version of cin.get() that uses a char\* and an int as arguments. But if you use cin.get(ch), the compiler fetches the version that uses a single type char argument. And if the code provides no arguments, the compiler uses the version of cin.get() that takes no arguments.

* **The End-of-File Condition**

If the input comes from a file, you can employ a much more powerful technique—detecting the end-of-file (EOF) to terminate input.

但是, many operating systems allow you to simulate the EOF condition from the keyboard. In Unix you do so by pressing Ctrl+D at the beginning of a line. Microsoft Visual C++, Borland C++ 5.5, and GNU C++ for the PC **recognize *Ctrl+Z* when it’s the first character on a line, but they require a subsequent *Enter***. In short, many PC programming environment recognize Ctrl+Z as a simulated EOF, but the exact details (anywhere on a line versus first character on a line, Enter key required or not required) vary.

**Unix and Linux users would press Ctrl+D at the first character of a new line instead.** In Unix and Unix-like systems, including Linux and Cygwin, Ctrl+Z suspends execution of the program; the fg command lets execution resume.

If your programming environment can test for the EOF, you can use a program similar to **Listing 5.17 with redirected files (<)** and you can use it for keyboard input in which you simulate the EOF.

**When cin detects the EOF, it sets two bits** (the ***eofbit*** and the ***failbit***) to 1.You can use a member function named eof() to see whether the eofbit has been set; the call ***cin.eof()*** returns the bool value **true if the EOF has been detected and false otherwise**. Similarly, the ***fail()*** member function **returns true if *either the eofbit or the failbit* has been set to 1 and false otherwise.** Note that the eof() and fail() methods report the result of the most recent attempt to read; that is, they report on the past rather than look ahead. So a cin.eof() or cin.fail() test should always follow an attempt to read. The design of Listing 5.18 reflects this fact. It uses **fail() instead of eof() because the former method appears to work with a broader range of implementations**.

momo@HMI:~/C++PrimerPlus/Chapter5$ cat **List5-18.cpp**

#include <iostream>

int main()

{

using namespace std;

char ch;

int count = 0;

**cin.get(ch);** // attempt to read a char

while (**cin.fail() == false**) // test for EOF

{

cout << ch; // echo character

++count;

cin.get(ch); // attempt to read another char

}

cout << endl << count << " characters read\n";

return 0;

}

在自己电脑上测试结果：

momo@HMI:~/C++PrimerPlus/Chapter5$ ./List5-18

i am momo(回车)

i am momo

i am momo kukumalu(回车)

i am momo kukumalu

**(Ctrl+D)**

29 characters read

momo@HMI:~/C++PrimerPlus/Chapter5$ **./List5-18 < Testfile**

Iam a Unix file. I am proud

to be a Unix file.

48 characters read

momo@HMI:~/C++PrimerPlus/Chapter5$

Remember that **when a cin method detects the EOF**, **it sets a flag in the cin object**, indicating the EOF condition. **When this flag is set, cin does not read any more input**, **and further calls to cin have no effect**. For file input, this makes sense because you shouldn’t read past the end of a file. **For keyboard input, however, you might use a simulated EOF to terminate a loop but then want to read more input later**. **The cin.clear() method clears the EOF flag and lets input proceed again**. Chapter 17,’Input, Output, and Files’, discusses this further. Keep in mind, however, that in some systems, typing Ctrl+Z effectively terminates both input and output beyond the powers of cin.clear() to restore them.

**字符输入的模板式方法**

The following is the essential design of a loop intended to read text a character at a time until EOF:

cin.get(ch); // attempt to read a char

while (**cin.fail() == false**) // test for EOF

{

... // do stuff

cin.get(ch); // attempt to read another char

}

例如上面List5-18.cpp的：

**cin.get(ch);** // attempt to read a char

while (**cin.fail() == false**) // test for EOF

{

cout << ch; // echo character

++count;

cin.get(ch); // attempt to read another char

}

cout << endl << count << " characters read\n";

再在公司的机器上仔细测试一下：

momo@HMI:~/C++PrimerPlus/Chapter5$ ./List5-18

**abc**abc

3 characters read

momo@HMI:~/C++PrimerPlus/Chapter5$

这里我先输入abc，然后输Ctrl+D，结果是在我输入的abc后面紧跟着出现abc，**然后鼠标停在这一行最后面**，再一次输入Ctrl+D，才跳到下一行输出3 characters read.

**[说明] Ctrl+D在输入时只有在一个line的开头才会被认为是EOF，如果在一行输入中插入Ctrl+D，会被当作是一行输入的结束，但不会被记作输入数据，所以第一个Ctrl+D没被算作输入字符，第二个Ctrl+D才被认为是EOF；**

***那为什么输完第一个Ctrl+D后cout才回显呢？***

**因为cin是buffered input. 当我在键盘上敲abc的过程中，敲入的字符被没有给List5-18这个程序，而是给了操作系统的input buffer. 当敲完第一个Ctrl+D(这个Ctrl+D也会进入input buffer)，input buffer判断本行输入结束，然后把这个input buffer里的数据给List5-18这个程序，然后程序才开始执行到while上面的cin.get(ch)这一句；**

[验证]

#include <iostream>

int main()

{

using namespace std;

char ch;

int count = 0;

cin.get(ch);

**cout << "\nAfter first cin.get()" << endl;**

while (cin.fail() == false) // test for EOF

{

cout << ch;

++count;

cin.get(ch);

}

cout << endl << count << " characters read\n";

return 0;

}

再在公司的机器上测试一下：

momo@HMI:~/C++PrimerPlus/Chapter5$ ./List5-18

**abc(Ctrl+D)**

After first cin.get()

abc**(Ctrl+D)**

3 characters read

momo@HMI:~/C++PrimerPlus/Chapter5$

接着测试：

momo@HMI:~/C++PrimerPlus/Chapter5$ ./List5-18

abc**(回车)**

After first cin.get()

abc

**(Ctrl+D)**

4 characters read **// 把回车也算做了一个输入字符**

momo@HMI:~/C++PrimerPlus/Chapter5$ ./List5-18

abc**(回车)**

After first cin.get()

abc

def**(回车)**

def

**(Ctrl+D)**

8 characters read

momo@HMI:~/C++PrimerPlus/Chapter5$

当直接用cin >> ch时，cin会无视输入的space和newline符，一直处于输入状态，除非你指定某个特别的字符作为结束；

**字符输入的模板式方法的改进**

**The return value for the cin.get(char) method is cin, an object**. However, the istream class provides a function that can convert an istream object such as cin to a bool value; this conversion function is called when cin occurs in a location where a bool is expected, such as in the test condition of a while loop. Furthermore, the bool value for the conversion is true if the last attempted read was successful and false otherwise. This means you can rewrite the while test to look like this:

while (cin) // while input is successful

This is a bit more general than using !cin.fail() or !cin.eof() because it detects other possible causes of failure, such as disk failure.

Finally, because the return value of cin.get(char) is cin, you can condense the loop to this format:

while (cin.get(ch)) // while input is successful

{

... // do stuff

}

**Here, cin.get(char) is called *once* in the test condition instead of twice—once before the loop and once at the end of the loop**. The program first has to execute the call to cin.get(ch), which, if successful, places a value into ch. Then the program obtains the return value from the function call, which is cin.Then it applies the bool conversion to cin, which yields true if input worked and false otherwise.

1. **Yet Another Version of cin.get()**

C users might yearn for C’s character I/O functions, getchar() and putchar().They are available in C++ if you want them.You just use the stdio.h header file as you would in C (or use the more current cstdio).

Or you can use member functions from the istream and ostream classes that work in much the same way. Let’s look at that approach next.

The *cin.get()* member function **with no arguments returns the next character from the input**. That is, you use it in this way:

ch = cin.get();

This function works much the same as C’s getchar(), **returning the character code as a type int value**. Similarly, you can use the cout.put() function (see Chapter 3,“Dealing with Data”) to display the character:

cout.put(ch);

It works much like C’s putchar(), except that its argument should be type char instead of type int.

**Recall that *cin.get(ch)* returns an object, not the character read.**

Typically, EOF is defined as the value -1 because no character has an ASCII code of -1, but you, don’t need to know the actual value. You can just use EOF in a program.

**int ch; // for compatibility with EOF value**

**ch = cin.get()**;

while (ch != **EOF**)

{

**cout.put(ch);** // **cout.put(char(ch)) for some implementations**

++count;

ch = cin.get();

}

You should realize that EOF does not represent a character in the input. Instead, **it’s a signal that there are no more characters.**

Because EOF represents a value outside the valid character codes, it’s possible that it might not be compatible with the char type. For example, on some systems type **char is unsigned**, **so a char variable could never have the usual EOF value of -1**. For this reason, if you use cin.get() (with no argument) and test for EOF, you **must assign the return value to type int instead of to type char**. Also if you make ch type int instead of type char, you might have to do a type cast to char when displaying ch.

**<List5-19.cpp>**

#include <iostream>

int main(void)

{

using namespace std;

**int ch;** **// should be int, not char**

int count = 0;

while **((ch = cin.get()) != EOF)** // test for end-of-file

{

cout.put(**char**(ch));

++count;

}

cout << endl << count << " characters read\n";

return 0;

}

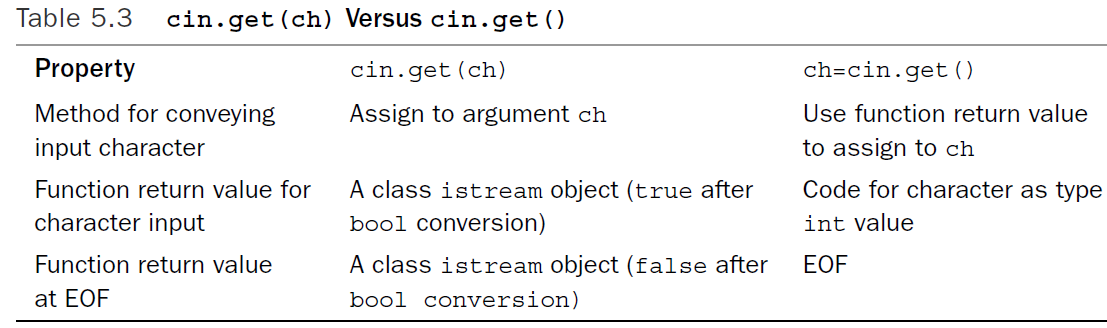
**测试是否为EOF的时候cast到int，但在display时再cast到char.**

Suppose you leave some parentheses out:

while (ch = cin.get() != EOF)

The != operator has higher precedence than =, so first the program compares cin.get()’s return value to EOF. A comparison produces a false or true result; that bool value is converted to 0 or 1, and that’s the value that gets assigned to ch.

Using cin.get(ch) (with an argument) for input, on the other hand, doesn’t create any type problems. Remember that the cin.get(char) function doesn’t assign a special value to ch at the EOF. In fact, it doesn’t assign anything to ch in that case. ch is never called on to hold a non-char value.



So which should you use, cin.get() or cin.get(char)? **The form with the character argument is integrated more fully into the object approach because its return value is an istream object**. This means, for example, that you can chain uses. For example, the following code means read the next input character into ch1 and the following input character into ch2:

cin.get(ch1).get(ch2);

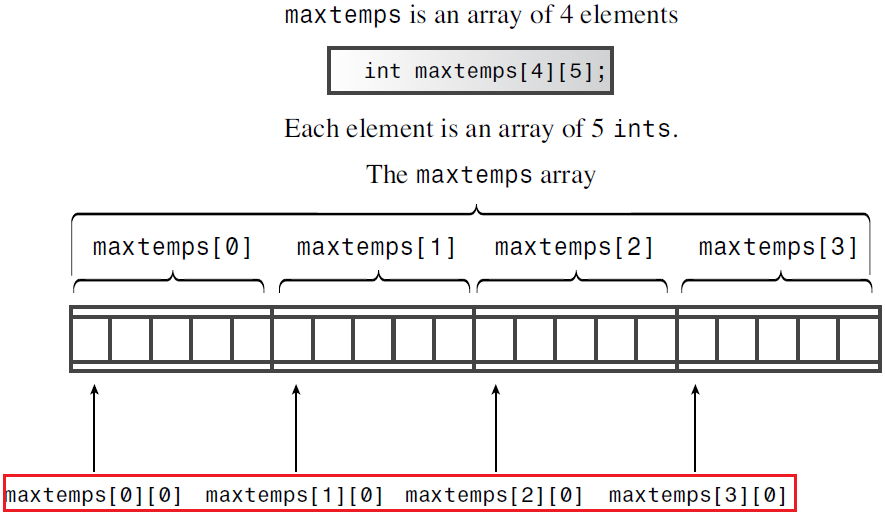
This works because the function call cin.get(ch1) returns the cin object, which then acts as the object to which get(ch2) is attached.

* **Nested Loops and Two-Dimensional Arrays**

For example, suppose you want to store maximum temperature data for five cities over a 4-year period. In that case, you can declare an array as follows:

*int maxtemps[4][5];*

This declaration means that maxtemps is **an array with four elements**. **Each of these elements is an array of five integers**.



* **Initializing a Two-Dimensional Array**

int maxtemps[4][5] = // 2-D array

{

{96, 100, 87, 101, 105}, // values for maxtemps[0]

{96, 98, 91, 107, 104}, // values for maxtemps[1]

{97, 101, 93, 108, 107}, // values for maxtemps[2]

{98, 103, 95, 109, 108} // values for maxtemps[3]

};

#include <iostream>

const int Cities = 5;

const int Years = 4;

int main()

{

using namespace std;

**const char \* cities[Cities]** = // array of pointers

{ // to 5 strings

"Gribble City",

"Gribbletown",

"New Gribble",

"San Gribble",

"Gribble Vista"

};

int maxtemps[Years][Cities] = // 2-D array

{

{96, 100, 87, 101, 105}, // values for maxtemps[0]

{96, 98, 91, 107, 104}, // values for maxtemps[1]

{97, 101, 93, 108, 107}, // values for maxtemps[2]

{98, 103, 95, 109, 108} // values for maxtemps[3]

};

cout << "Maximum temperatures for 2008 - 2011\n\n";

for (int city = 0; city < Cities; ++city)

{

cout << cities[city] << **":\t**"; **// \t是Tab键，让字符对齐**

for (int year = 0; year < Years; ++year)

cout << maxtemps[year][city] << "\t";

cout << endl;

}

// cin.get();

return 0;

}

城市名也可以这么定义：

char cities[Cities][25] = // array of 5 arrays of 25 char

{

"Gribble City",

"Gribbletown",

"New Gribble",

"San Gribble",

"Gribble Vista"

};

The array of pointers stores the addresses of the five string literals, but the array of char arrays copies each of the five string literals to the corresponding five arrays of 25 char.

Also you could use an array of string class objects instead of an array of pointers for the string data.The declaration would look like this:

**const string cities[Cities]** = // array of 5 strings

{

"Gribble City",

"Gribbletown",

"New Gribble",

"San Gribble",

"Gribble Vista"

};