C++

Side Effects and Sequence Points

Sequence points: 程序执行过程中的一个点，在C++中，语句中的分号就是一个顺序点，这意味着在执行下一条语句之前，赋值、增减运算符执行的所有修改都要完成。

```
while ( guest++ < 10)
    cout << guest << endl;
```

例如 `while` 循环的测试条件表达式的末尾就是一个顺序点，确保了 **side effect(将 guest 加1)** 完成后进入循环体内部.

Prefixing Versus Postfixing

```
++x;    // Prefixing
x++;    // Postfixing
```

- 对于**类**而言，**前缀版本**比后缀版本**效率高**
  - 前缀函数：将值加1，然后返回结果；
  - 后缀函数：首先复制一个副本，将其加1，然后将复制的副本返回。
- 优先级：**后缀 > 前缀 = 解除引用**
  - 前缀运算符和解除引用以**从右到左**的方式进行结合：
    - `*++pt`：先做 `++` 将指针加一指向下一个元素，然后 `*` 取出其中的值；
    - `++*pt`：先做 `*` 取出 `pt` 地址中的值，然后做 `++` 对其加一；
  - 后缀运算符优先级高以**从左到右**的方式进行结合：
    - `(*pt)++`：先做 `*` 取出 `pt` 地址中的值，然后做 `++` 将这个值加一，`pt` 指向不变；
    - `*pt++`：先做 `++` 将指针递增，但由于是后缀所以 `*` 对原来的地址做解引用，而不是新的递增后的地址 **(遍历的时候可以这么用)**

The Comma Operator

```
char temp;
int i, j;
for ( j = 0, i = word.size() - 1; j < i; j++, i-- ) {
    temp = word[i];
    word[i] = word[j];
    word[j] = temp;
}
```

逗号运算符最常见的用途是将两个或更多的表达式放到一个 `for` 循环表达式中

- 逗号表达式确保**先计算第一个表达式**，然后计算第二个表达式；
- 逗号表达式确保**最终**这条语句的值是**第二部分的值**；
- 在所有的运算符中，**逗号**运算符的优先级是**最低的。**

The Range-Based `for` Loop

```
double prices[5] = {4.9, 5.9, 6.9, 10.9, 11.9};
for ( double x : prices )
    cout << x << std::endl;
```

`x` 表示数组中的某一个元素，随着循环的进行会改变，但是这样不会对数组中的元素做出改变。

```
for ( double &x : prices )
    x = x * 0.8;
```

`&x` 表示 `x` 是一个引用变量，这个声明让接下来的代码可以修改数组的内容。

Loops and Text Input

```
int ch, count = 0;
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;

>>> see ken run#really fast
    seekenrun               // 程序在输出的时候省略了空格
    9 characters read
```

- 程序在输出的时候省略了空格是因为 `cin` 在读取 `char` 值的时候和其他类型一样**将忽略空格和换行符**，所以输入中的空格没有被回显。
- 发送给 `cin` 的输入被首先保存在**缓冲**里，只有在用户按下回车键之后，输入的内容才会被发送给程序，所以这就是运行程序的时候可以在 `#` 后面输入字符的原因。

`cin.get(ch)`：**读取输入中的下一个字符，无论是空格、制表符还是换行符……** 然后将其赋值给 `ch`，使用 `cin.get(ch)` 而不是 `ch = cin.get(ch)`.

The End-of-File Condition

如果输入来自文件的话可以使用**检测文件尾(EOF)**，来检测文件尾表示文件输入结束并将其告知程序。在检测到 `EOF` 之后，`cin` 会将两位 (`eofbit` 和 `failbit`) 都设置为1，这可以通过成员函数 `cin.eof()` 和 `cin.fail()` 来报告最近读取结果(事后报告)

很多系统都允许使用键盘来模拟文件尾条件：

- Unix：可以在行首按下 `Ctrl + D` 来实现；
- Windows 命令行条件：可以在任意位置按 `Ctrl + Z` 和 `Enter`.

`cin` 方法检测到 `EOF` 之后将设置 `cin` 对象中一个**指示 `EOF` 条件的标记**，设置这个标记之后 `cin` 将**不再读取输入**，再次调用 `cin` 也没用；只有 `cin.clear()` 可以**清除**这个 `EOF` 标记，使输入继续进行。

```
while ( cin.get(ch) ) {     // while input is successful
    ...                     // do stuff
}
```

为判断循环测试条件，程序必须首先调用 `cin.get(ch)`：如果成功则将值放入 `ch` 中。**然后**获得函数调用的**返回值** `cin`，接下来程序对 `cin` 进行 `bool` 转换，如果输入成功结果为 `true`，否则为 `false`.

```
int ch;
ch = cin.get();             // Return a **cin object**
while ( ch != EOF ) {
    cout.put(ch);           // Same as putchar() in C
    ++count;
    ch = cin.get();         // Same as getchar() in C
}
```

- `cin.get(ch)` 有一个参数的：**返回值为 `cin` 对象**，可直接使用 `while(cin.get(ch))` 自动进行 `bool` 转换，在到达 `EOF` 时，不会将任何值赋给 `ch`.
- `cin.get()` 没有参数的：**返回值为值或者 `EOF(一般是-1)`**，所以必须将返回值赋值给 `int` 而不是 `char`，但是如果 `ch` 的变量类型为 `int` 在显示 `ch` 时需要将其转换成 `char` 类型——`cout.put(char(ch))`.

属性	`cin.get(ch)`	`ch=cin.get()`
传递输入字符的办法	赋给参数 `ch`	将函数返回值赋给 `ch`
用于字符输入时函数的返回值	`istream` 对象 `cin` (执行 `bool` 转换后为 `true`)	`int` 类型的字符编码
到达 `EOF` 时函数的返回值	`istream` 对象 `cin` (执行 `bool` 转换后为 `false`)	`EOF` (一般是 -1)

The `cctype` Library of Character Functions

函数名称	返回值
`isalnum(ch)`	如果参数是字母或数字，返回 `true`
`isalpha(ch)`	如果参数是字母，返回 `true`
`iscntrl(ch)`	如果参数是控制字符，返回 `true`
`isdigit(ch)`	如果参数是数字，返回 `true`
`isgraph(ch)`	如果参数是除空格以外的打印字符，返回 `true`
`islower(ch)`	如果参数是小写字母，返回 `true`
`isprint(ch)`	如果参数是打印字符 (包括空格)，返回 `true`
`ispunct(ch)`	如果参数是标点符号，返回 `true`
`isspace(ch)`	如果参数是标准空白字符 (如 `' '`, `'\t'`, `'\n'`...)，返回 `true`
`isupper(ch)`	如果参数是大写字母，返回 `true`
`isxdigit(ch)`	如果参数是十六进制数字 (即 `0~9`, `a~f`, `A~F`)，返回 `true`
`tolower(ch)`	如果参数是大写字母，则返回其小写，否则返回参数本身
`toupper(ch)`	如果参数是小写字母，则返回其大写，否则返回参数本身

The `?:` Operator——条件运算符

```
expression1 ? expression2 : expression3
```

如果 `expression1` 为 `true`，则整个条件表达式的值为 `expression2` 的值；否则整个表达式的值为 `expression3` 的值。

`eg. x = (x > y) ? x : y;` 表示取 `x, y` 中较大的那一个。

The `switch` Statement

```
switch ( integer-expression ) {
    case label1 : statement(s)
    case label2 : statement(s)
    ...
    default     : statement(s)
}
```

- `integer-expression` 必须是一个结果为**整型值**的表达式，每个标签 `label` 也必须是整数常量表达式，最常见的标签是 `int` 或 `char` 常量 (如 `1` 或 `'q'`) 也可以是枚举量；
- 如果 `integer-expression` 不与任何标签匹配，则程序将跳到标签为 `default` 的那一行；

- 这里的 `case` 标签只是行标签，程序不会在执行到下一个 `case` 处自动停止，要让程序执行完一组特定语句后停止，**必须使用 `break` 语句。**
- `switch` 无法处理浮点测试，且 `case` 标签纸必须是常量；
- 如果选项超过两个，就代码长度和执行速度而言，`switch` 语句的效率更高一点。

File I/O preliminary acquaintance

Output

```
#include <ifstream>     // read from file
#include <ofstream>     // write to file
#include <fstream>      // Contains both of them above

/* 声明 ofstream 对象 */
ofstream outFile;       // outFile: an ofstream object
ofstream fout;          // fout: an ofstream object

/* 将 ofstream 对象和特定的文件关联起来 */
outFile.open('fish.txt');   // outFile used to write to the fish.txt file
char filename[50];
cin >> filename;        // user specifies a name
fout.open(filename);    // fout used to read specified file

/* 将上面的两步连在一起做完 */
ofstream outFile("./test.txt");
ifstream inFile("./input.txt");

/* 使用 ofstream 对象 */
double wt = 125.8;
outFile << wt;          // write a number to fish.txt
char line[81] = "Objects are closer than they appear.";
fout << line << endl;   // write a line to text

/* 关闭文件，不需要包含文件名 */
outFile.close();
fout.close();
```

声明一个 `ofstream` 对象并将其同文件关联起来之后，便可以像使用 `cout` 那样使用它，所有的 `cout` 的操作和方法都可以(如 `<<, endl, setf()`) 都可以用于 `ofstream` 对象。

1. 包含头文件 `ofstream`；
2. 创建一个 `ofstream` 对象，并将其同一个文件关联起来 `ofstream_object.open(filename)`
3. 像使用 `cout` 那样使用该 `ofstream` 对象；
4. 使用完文件之后必须用 `ofstream_object.close()` 来将其关闭。

Input

几乎和 output 一摸一样，只不过变成了 `ifstream` 对象，关联文件之后可以像使用 `cin` 一样，但是多了一个要**检查文件是否打开成功的操作**：

```
inFile.open("bowling.txt");
if ( !inFile.is_open() ) {
    exit(EXIT_FILURE);
}
```

有下面的几种情况可能导致文件打开失败：

1. 程序读取文件的时候不应该超过 `EOF`，超过的话就会读取失败，用 `cin.eof()`；
2. 程序可能遇到类型不匹配的情况，用 `cin.fail()` 同时判断 `1, 2`；
3. 意外的问题，文件受损或者硬件故障；
4. 方法 `cin.good()` 在没有任何错误的时候返回 `true`.

```
#include <fstream>
ifstream inFile("./test.txt");
while ( inFile.good() ) {
    ...
}
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.\n1;"
```