说明：该文档是对C++程序设计语言的读书笔记，以及摘录的比较有用的代码片段。

C++基本概念

<1> 声明：在C++程序里，每个名字都有一个与之相关联的类型，这个类型决定了可以对这个名字应用什么操作，并决定这些操作将如何做出解释。

一个声明由四个部分组成：一个可选的“描述符”，一个基础类型，一个声明符，还有一个可选的初始式。除了函数和名字空间外，其他声明都应该由分号结束。

描述符是一个开始关键字，例如virtual,extern，说明了被声明事物的某些非类型的属性。

声明符由一个名字和可选的若干声明运算符组成。最常用的声明运算符是：

\* 指针　　　　前缀

\*const 常量指针　　前缀

&　　　　引用　　　　前缀

[] 数组　　　　后缀

()　　　　函数　　　　后缀

注意：在一个声明中不能没有类型。

一个声明将一个名字引进一个作用域；也就是说，这个名字只能在程序正文的一个特定部分内使用。

指针：对类型T，Ｔ\*是“到Ｔ的指针”类型，也就是说，一个类型为T\*的变量能保存一个类型Ｔ的对象的地址。

名字空间是一种描述逻辑分组的机制。也就是说，如果有一些声明按照某种准则在逻辑上属于同一个集团，就可以将它们放入同一个名字空间，以表明这个事实。

迭代器是指向序列元素的指针概念的一种抽象，其最关键的属性是：

１　当前被指向的元素（间接，用运算符\*和->表示）

２　指向下一个元素（增量，用运算符++表示）

３　相等（用运算符==表示）

序列也是一个抽象概念，是”我们可以从其头部开始，通过使用下一个元素操作，最终到达结束处的某种东西”。

运算符函数的名字是由关键字operator后跟对应的运算符构成的；例如operator<<。运算符函数的定义和使用都可以像其他函数一样。使用运算符不过是显式调用运算符函数的一种简写形式。

[二元运算符可以定义为取一个参数的非静态成员函数，也可以定义为取两个参数的非成员函数。对于任何二元运算符@，aa@bb可以解释为aa.operator@(bb)，或者解释为operator@(aa,](mailto:二元运算符可以定义为取一个参数的非静态成员函数，也可以定义为取两个参数的非成员函数。对于任何二元运算符@，aa@bb可以解释为aa.operator@(bb)，或者解释为operator@(aa,) bb)。

[对于一元运算符，无论它是前缀的还是后缀的，都可以定义为无参数的非静态成员函数，或者定义为取一个参数的非成员函数。对任何前缀一元运算符@，@aa可以解释为aa.operator@()或者operator@(aa)。对任何后缀一元运算符@，aa@可以解释为aa.operator@(int)或者operator@(aa,](mailto:对于一元运算符，无论它是前缀的还是后缀的，都可以定义为无参数的非静态成员函数，或者定义为取一个参数的非成员函数。对任何前缀一元运算符@，@aa可以解释为aa.operator@()或者operator@(aa)。对任何后缀一元运算符@，aa@可以解释为aa.operator@(int)或者operator@(aa,) int)。运算符只能按照语法定义的形式进行声明。

operator=，operator[]，operator()和operator->只能作为非静态的成员函数，这就保证了它们的第一个运算对象一定是一个左值。运算符=(赋值)，&(取地址)和,(序列)在应用于类对象时已经有了预先定义的意义。通过将它们定义为私用，就可以使普通用户无法访问这些预定义的意义。

class X{

private:

void operator=(const X&);

void operator&();

void operator,(const X&);

};

void f(X a, X b)

{

a = b; //错误，operator=为私用

&a; //错误，operator&为私用

a,b; //错误，operator,为私用

}

### 代码片段：

<1> 统计元素的出现次数

template<class C, class T> int count(const C& v, T val)

{

typename C::const\_iterator iter = find(v.begin(), v.end(), val);

Int n = 0;

while(iter != v.end())

{

++n;

++iter;

iter = find(i, v.end(), val);

}

return n;

}

<2> 读入一个文件，对所读的东西排序，去掉重复，最后将结果写入一个文件。

#include <string>

#include <fstream>

#include <iterator>

#include <sstream>

#include <algorithm>

#include <iostream>

int main(int argc, char\*\* argv)

{

string from("read.txt");

string to("write.txt");

//cin>>from>>to;

ifstream is(from.c\_str()); //输入流

istream\_iterator<string> ii(is); //流的输入迭代器

istream\_iterator<string> eos; //输入的哨兵

//读入文件内容

vector<string> b(ii, eos);

cout<<"file ["<<from<<"] has lines : "<<b.size()<<endl;

copy(b.begin(), b.end(), ostream\_iterator<string>(cout, "\n"));

//排序

sort(b.begin(), b.end());

cout<<"after sort :"<<endl;

copy(b.begin(), b.end(), ostream\_iterator<string>(cout, "\n"));

//写入文件

ofstream os(to.c\_str());

ostream\_iterator<string> oo(os, "\n");

unique\_copy(b.begin(), b.end(), oo);

return !is.eof() || !os;

}

<3>