Chapter 01

1. “<<”和“>>”运算符返回其左侧对象，因此可以连接使用。

2.std::endl有两个作用：换行；刷新缓冲。

Chapter 02

1.可以为字面值指定类型，如：42ULL(unsigned long long)

2.区分好初始化和赋值。初始化表示对象创建时予以的值，赋值代表擦除当前值并给予一个新值。

3.extern 关键字用在变量名前表示声明这个变量而不是定义。

4.引用是一个别名不占用空间，指针是一种变量需要空间。

5.局部的const限定，只能在编译时检查出来；还有一种更高层次的即是放在只读数据段中。

6.头文件不应包含using声明。

7.范围for循环其实是for循环的一种简写形式，然而这里需要注意的就是，范围for循环中往往用到的都是一些容器的迭代器，因此在循环过程中如果向容器添加了元素，这时可能导致迭代器失效。

Chapter 04

1.两个指针相减得到的是一个整形，代表着相差元素个数。（注意：得到的并不是指针）

2.逻辑与运算符：先求左侧运算对象的值，因此可以优化。

3.习惯于这种书写方式：

cout << \*p++ << endl;这里\*p++ = \*(p++)；即将p的值+1，然后将其初值取值返回。

4.p++有两层含义：p加一；返回p的原始值。

Chapter 05

1.函数声明的时候可以不带形参名。

2.返回数组指针：

（1）typedef int arrT[10];

arrT\* func(int);

等价于：

int (\*func(int i))[10];

解释：

* func(int i)表示调用func函数时需要一个int类型的参数。
* (\*func(int i))表示对函数执行后的结果进行解引用。
* (\*func(int i))[10]表示解引用后得到的是一个大小为10的数组。
* 前面加int表示这个数组的类型是int。

（2）void (\*signal(int signo, void (\*func)(int)))(int)

等价于：

typedef void Sigfunc(int);

Sigfunc \* signal(int signo, Sigfunc \*func);

3.默认实参：

（1）一旦某个形参被赋予了默认值，那它之后的所有形参都必须有默认值。

（2）函数解析填补的时候是从左往右填补的。

（3）尽量让不怎么使用默认值得形参出现在前面，让那些经常使用的在后面。

4.assert的使用：若要将其关闭，一定是放在main函数所在的文件的最开始部分（包含头文件之前）加上#define NDEBUG.

Chapter 07

1.默认构造函数：

（1）只有当类没有声明任何构造函数的时候，编译器才会自动生成默认构造函数。

（2）如果类包含有内置类型或者复合类型的成员，则只有当这些成员全部被赋予了类内的初始值时，这个类才适合于使用合成的默认构造函数。

2.拷贝和赋值函数的区别：赋值是=；拷贝则可能是初始化变量或以值的方式传递或返回一个对象。

3.当定义inline函数时，最好在类外定义的时候加在函数的前面，这样让类显得更容易理解，因为使用时只会关心接口。（在类内定义的函数自动变成内联函数）

4.常量对象只能调用常量版本的函数。通过区分成员函数是否为常量的，则可以对其进行重载。

5.类的定义分两步：编译成员的声明；直到类全部可见后才编译函数体。这样的好处：成员可以使用类里面声明了的所有成员，不用管声明的先后顺序。

6.成员的初始化顺序与它们在类定义中的出现顺序一致。

7.静态成员需要在类外定义，即它应该像成员函数一样定义。

Chapter 08

1.iostream相当于基类，其他则继承于它。因此设计接口时应使用它。

2.IO对象无拷贝或赋值，因此进行IO操作的函数通常以引用方式传递和返回流。

3.确定一个流对象的状态的最简单的方法是将它当作一个条件来使用：

While(cin >> word)或者if(cin >> word)。

4.

for(auto p = argv + 1; p != avgv + argc; ++p)

{

ifstream input(\*p);

if(input)

{

process(input);

}

else

{

cerr << "couldn't open: " + string(\*p);

}

}

（1）如何处理main的输入

（2）如何判断流是否有效

（3）注意应该尽量用不等于来进行条件判断

5.在gcc编译器下，对临时变量的引用需要是常量引用。但是对临时变量进行引用并不是一个好的习惯，因为不知道这个临时变量何时会无效。

Chapter 09

1.程序设计时，如果还不确定该用哪种容器或者以后会有变化，那么功能实现时尽量用容器的公共操作，这样可以减少代码的修改量。

2.删除deque中除首尾位置之外的任何元素都会使所有迭代器、引用和指针失效。指向vector或string中删除点之后位置的迭代器、引用和指针都会失效。

3.当程序需要随机的插入和删除时，最好先用链表，等待处理妥当后，再将其放入vector中，方便后续处理。

4.使用迭代器时，应该最小化要求迭代器必须保持有效的程序片段。

Chapter 10

1.在使用算法时，需要明白一点：那些只接受一个单一迭代器来表示第二个序列的算法，都假定第二个序列至少与第一个序列一样长。（原因：第一个序列给定了明确的范围，而第二个序列只给出了开始位置，会有越界访问的危险）

2.算法不会执行容器操作，因此它们自身是不可能改变容器的大小的。（保证算法的正确使用是程序员的事）

3.用插入迭代器（back\_inserter）解决了2中提到的问题，此时算法就可以改变容器的大小。

4.普通迭代器和插入迭代器：普通迭代器就如指针，它是指向一个已经有意义的地址，因此对于容器来说，它只能对容器中已经存在的元素进行操作；而插入迭代器却是另一种情况，由于对它的一些赋值操作进行了重载，当对一个插入迭代器赋值时，其实是调用了相关容器的插入操作，因此它可以改变容器的大小。

5.当使用一种算法，其中需要将元素从源拷贝到目的地时，目的处的迭代器最好用插入迭代器，这样就可以避免空间不够发生错误的情况。但有一点要注意这里执行的是插入操作，因此以前有的元素还是会保留下来。

6.链表有一些特定的算法，在使用时，应该使用这些版本，因为链表交换元素只需要改变链接就可以了。