1. 循环和关系表达式

1.for循环

（1）语法

for(i=0 ; i<5 ; i++)//循环在i的值为1~4时执行

或for(int i=5 ; i ; i--)//循环在i的值为5~1时执行，执行完后i值为0

或for(i=0 ; i<100 ; i=i+by)//按照用户选择的步长值by将循环计数递增，

\且表达式可以是任何有效的表达式，如i=i\*i+10

1. 不能把for循环赋给变量。
2. 使用for循环访问字符串

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

注意整个word字符串（自定义的字符串）是从0到word.size()-1。

1. ++i与i++的区分

y=++i是先把i的值加一，再把i的值赋给y；

y=i++是先把i的值赋给有y，再把i的值加一。

注意不要在同一条语句对同一个值递增或递减多次。

对于内置类型，采用哪种格式不会有差别；但对于用户定义的类型，如果有用户定义的递增和递减运算符，则前缀格式的效率更高。

1. 递增/递减运算符和指针

\*++pt;//将++应用于pt（因为++位于\*的右边），然后将\*应用于被递增后的pt。

++\*pt;//先取得pt指向的值，然后将这个值加1。

(\*pt)++;//圆括号指出，首先对指针解除引用，得到指针所指向的值24.4，然后运算符++将这个值递增到25.4。

x=\*pt++;//后缀运算符++的优先级更高，所以将运算符用于pt，而不是\*pt，因此是对指针递增。然而后缀运算符意味着将对原来的地址而不是递增后的新地址解除引用，因此\*pt++的值为arr[2]，但该语句执行完毕后，pt的值将为arr[3]的地址。

1. 组合赋值运算符



（7）注意for语句后要用花括号{语句块}来包含需执行循环的语句。

在语句块内定义的变量在语句块外是不可用的，并且在若一个变量在语句块内改变了其值，那么在语句块外它仍为初始值。

（8）逗号运算符：对表达式完成同样的任务，允许将两个表达式放到C++句法只允许放一个表达式的地方。逗号表达式的值是第二部分的值，并且优先级是最低的。

逗号并不总是逗号运算符。例如，int i,x中的逗号将变量列表中相邻的名称分开。

这使得for循环表达式也可以是：

for( j=0 , i = word.size() - 1 ; j<i ; --i ,++j ){}

（9）数组名是数组的地址。同样，用引号括起的字符串常量也是其地址。

（10）strcmp（word，"mate"）//如果字符串不相等，则它的值为非零（true）；如果字符串相等，则它的值为零（false）。如果str1在str2的前面，则strcmp(str1,str2)<0。

2.while循环

（1）语法： while(测试条件){循环体}

（2）设计循环的指导原则：

指定循环终止的条件；

在首次测试之前初始化条件；

在条件被再次测试之前更新条件。

1. 延时循环：可以让程序等待一段时间。若读者担心自己错过了重要的、无法恢复的消息，可以用while循环让程序在显示其他内容之前等待5秒钟。

如：long wait=0;

while(wait<10000)

Wait++;

1. clock( )//返回程序开始执行后所用的系统时间。

用clock( )和头文件ctime来创建延迟循环：

#include<iostream>

#include<ctime>

int main()

{

using namespace std;

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

float secs;

cin>>secs;

clock\_t delay=secs\*CLOCKS\_PER\_SEC;

cout<<"starting\a\n";

clock\_t start=clock();

while(clock()-start<delay)

;

cout<<"done\a\n";

return 0;

}

1. 类型别名

C++为类型建立别名的方式有两种。一种是使用预处理器：

#define BYTE char;

或者typedef char byte;(通用格式为typedef typename aliasName;)

1. do while循环

不同于另外两种循环，因为它是出口条件（exit condition）循环。这意味着这种循环将首先执行循环体，然后再判定测试表达式，决定是否应继续执行循环。所以循环体至少会执行一次。

1. 句法

do

{Body}

while(测试条件);

1. 基于范围的for循环

例如double prices[5]={4,10,6,7,8};

for(double x :prices)

cou<< x<<std::endl;//该循环显示数组中的每个值。

1. 循环和文本输入
2. cin.get(char)//读取输入中的下一个字符（即使它是空格)，并将其赋给变量ch。
3. 使用cin.eof()

检测到EOF（文件尾）后，cin将两位（eofbit和failbit）都设置为1。可以通过成员函数eof( )来查看eofbit是否被设置；如果检测到EOF，则cin.eof( )将返回bool值true，否则返回false。

例如，每次读取一个字符，直到遇到EOF的输入循环的基本设计如下：

cin.get(ch);

While(cin.fail()==false)

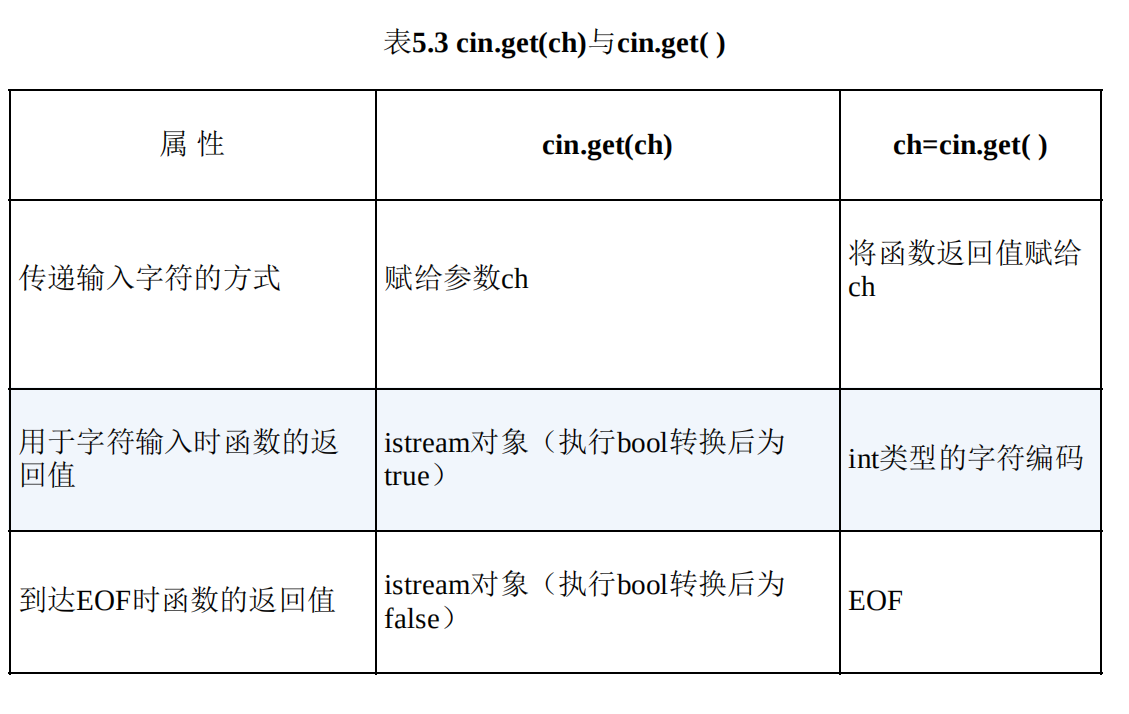
{

...

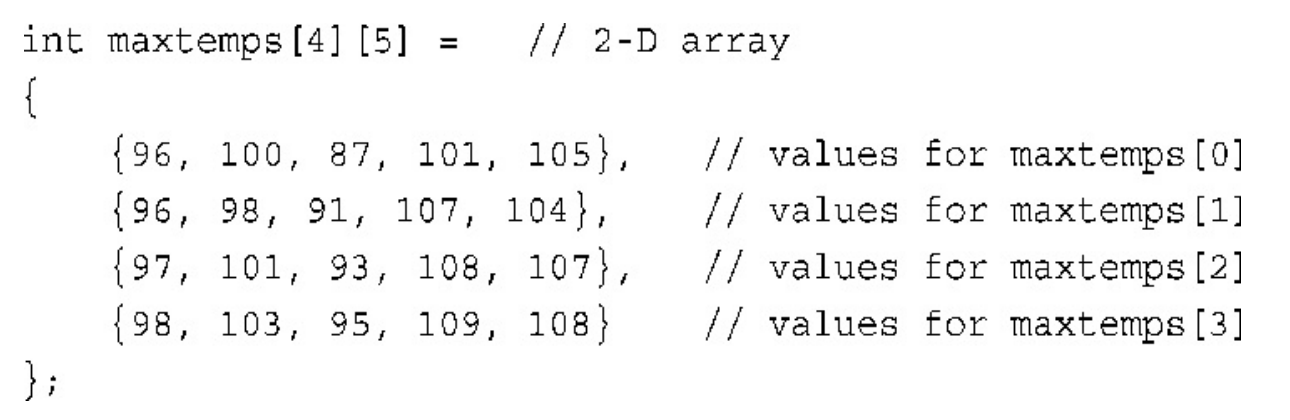
cin.get(ch);

}

（3）



1. 嵌套循环和二维数组
2. 定义二维数组



1. 分支语句和逻辑运算符
2. if语句：使程序有条件地执行语句或语句块。
3. 语法

if（测试条件）{...}

1. if else语句

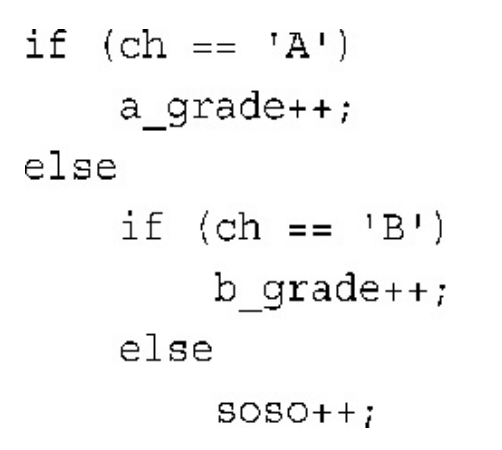
选择执行两个语句或语句块之一。

语法：if（测试条件）{}

else {}

1. if else if else结构

语法示例：



1. 逻辑表达式

C++逻辑OR和逻辑AND运算符的优先级都低于关系运算符(<、>)。!运算符的优先级高于所有的关系运算符和算术运算符。

1. 逻辑OR运算符：||

注意||运算符是个顺序点（sequence point）。即运算符左边的子表达式先于右边的子表达式）

1. 逻辑AND运算符：&&

仅当运算符左右两边的表达式都为true时，得到的表达式的值才为true。

注意以下表达式是错误的：

if（17<age<35）

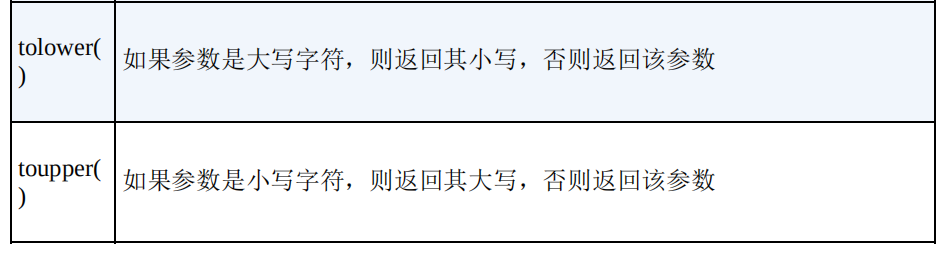
因为它相当于if(( 17 <age ) < 35)，而17 < age的值要么为true（1），要么为false（0）。不管是哪种情况，表达式17 < age的值都小于35，因此整个测试的结果总是true。

1. 逻辑NOT运算符：!

将它后面的表达式的真值取反。

1. cctype字符函数库
2. 如果ch是一个字母，则isalpha（ch）函数返回一个非零值，否则返回0。同样，如果ch是标点符号（如逗号或句号），函数 ispunct（ch）将返回true。



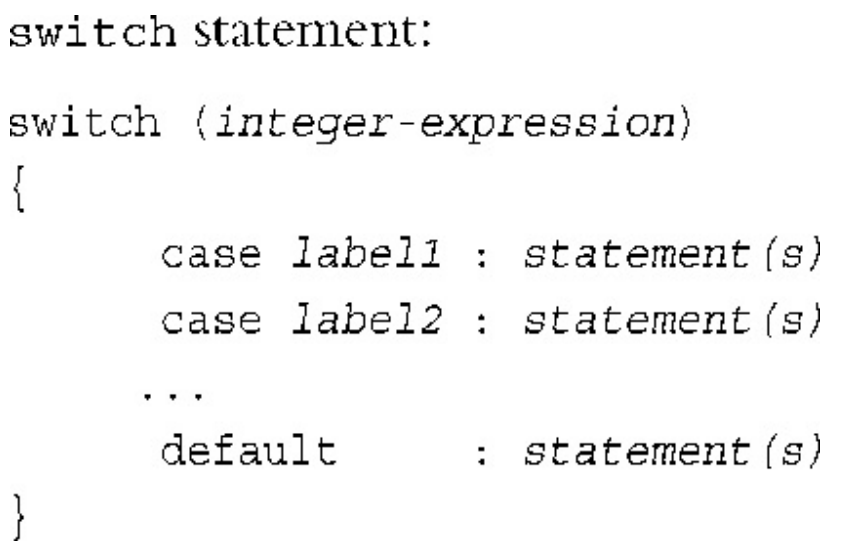


1. 条件运算符（?:）

通用格式:

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。

1. switch语句：执行一系列选项之一。
2. 通用格式



执行到switch语句时，程序将跳到使用integer-expression的值标记的那一行。例如，如果integer-expression的值为4，则程序将执行标签为 case 4：那一行。所以integer-expression必须是一个结果为整数值的表达式。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。每个标签都必须是整数常量表达式。

程序跳到switch中特定代码行后，将依次执行之后的所有语句，而不会在执行到下一个case处自动停止，要使其执行完一组特定语句后停止，必须使用break语句（加在每个case的后面）。

1. continue和break语句

break：跳出循环，执行下一条语句；

continue：跳过循环体的剩余部分，开始新一轮循环。

1. 基本文件输入/输出
2. 下面演示了如何声明对象并将这种对象与特定的文件关联起来：

ofstream outFile;

    ofstream fout;

    outFile.open("fish.txt");

    char filename[50];

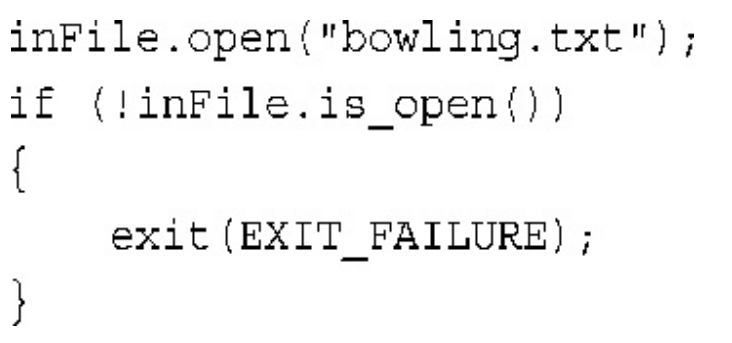
    cin>>filename;

fout.open(filename);

使用文件输出的步骤： 1．包含头文件fstream； 2．创建一个ofstream对象； 3．将该ofstream对象同一个文件关联起来； 4．就像使用cout那样使用该ofstream对象。

1. 读取文本文件

检查文件是否被成功打开的首先方法是使用方法is\_open()，例如：



1. 函数——C++的编程模块
2. 函数的基本知识

创建自己的函数时，必须定义、提供原型和调用。前四章总结过，在此不赘述。

1. 按值传递函数参数

注意：函数可以有多个参数。在调用函数时，只需使用逗号将这些参数分开：

N\_chars(‘R’,25)

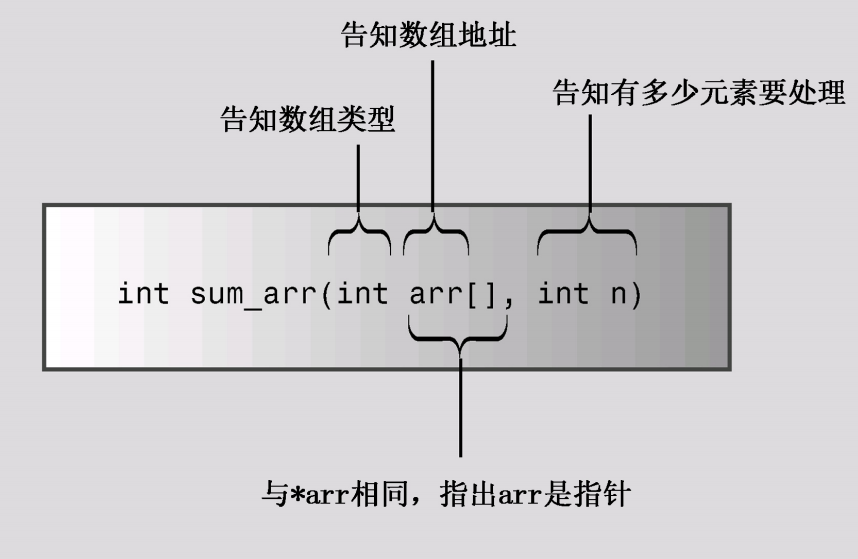
1. 设计处理数组的函数

（1）需要将一个形参声明为数组名。函数头如下：

int sum\_arr(int arr[], int n)//注意arr其实是指针，但在编写函数的其余部分时，可以将arr看作是数组。

当且仅当用于函数头或函数原型中，int \*arr和int arr [ ]的含义才是相同的（都是

指针）。



注意：为将数组类型和元素数量告诉数组处理函数，应通过两个不同的参数来传递它们，而不能用方括号表示法来传递数组长度（如 void fillArray(int arr[size];)

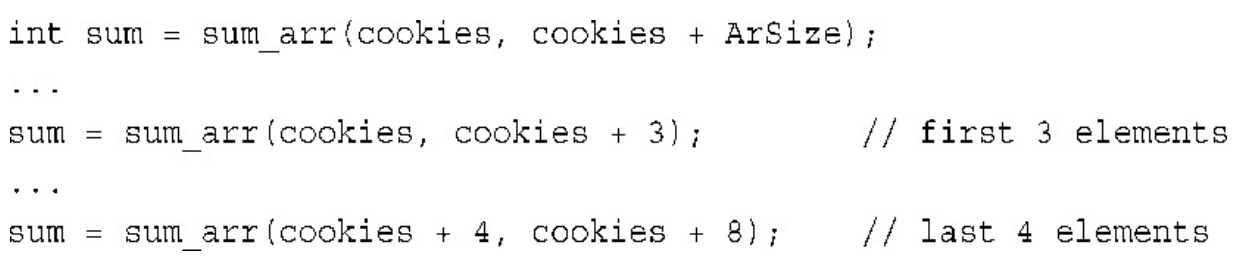
（2）显示数组及用const保护数组

为防止函数无意中修改数组的内容，可在声明形参时使用关键字const：

void show\_array(const double ar[],int n);

1. 使用数组区间的函数

函数不能使用 sizeof来获悉原始数组的长度，而必须依赖于程序员传入正确的元素数。几种指定数组不同区间的方法：



1. 使用const指针参数

可以用两种方式将const关键字用于指针。第一种：让指针指向一个常量对象，这样可以防止使用该指针来修改所指向的值；第二种：将指针本身声明为常量，这样可以防止改变指针指向的位置。

例如： int age=39;

const int \*pt= &age;//\*pt的值为const，不能被修改。可以直接通过age变量来修改age的值，但不能用指针pt来修改它。

1. 设计处理文本字符串的函数
2. 可以说是将字符串作为参数来传递，但实际传递的是字符串第一个 字符的地址。C-风格字符串与常规char数组之间的一个重要区别：字符串有内置的结束字符。
3. 编写一个返回字符串的函数

函数无法返回一 个字符串，但可以返回字符串的地址，这样做的效率更高。

可定义了一个函数，该函数返回一个指针，接受两个参数：一个字符和一个数字。并使用new创建一个长度与数字参数相等的字符串，然后将每个元素都初始化为该字符。然后，返回指向新字符串的指针。

1. 设计处理结构的函数
2. 使用结构编程时，最直接的方式是将结构作为参数传递，并在需要时将结构用作返回值使用。但如果结构非常大，则复制结构将增加内存要求，降低系统运行的速度。
3. 当结构比较小时，按值传递结构最合理。

定义结构：

struct travel\_time

{

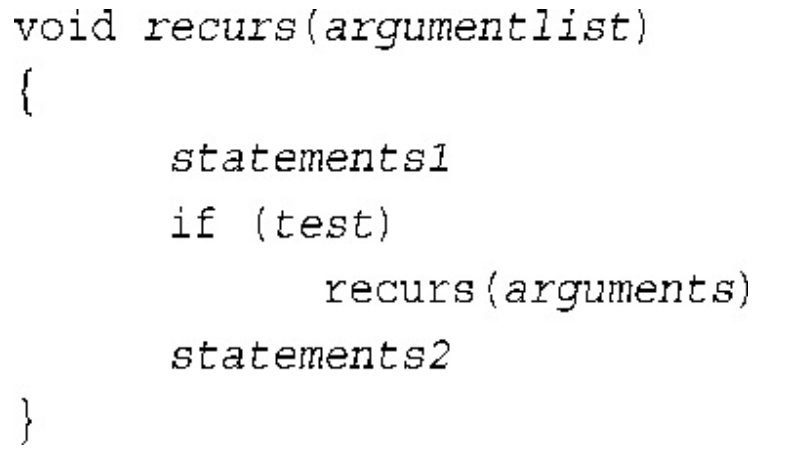
int hous;

int mins;

}

对应的函数原型：travel\_time sum(travel\_time t1,travel\_time t2);

1. 调用自身的函数（递归）
2. 注意：C++不允许main( )调用自己。
3. 句法示例：



1. 指向函数的指针

与数据项相似，函数也有地址。函数的地址是存储其机器语言代码的内存的开始地址。

1. 获取函数的地址：只要使用函数名（后面不跟参数）即可。也就是说，如果think( )是一个函数，则think就是该函数的地址。
2. 声明函数指针

若函数原型：double pam(int );

则指针类型声明为：double (\*pf)(int );

1. 使用指针来调用函数

前面讲过，（\*pf）扮演的角色与函数名相同，因此使用（\*pf）时，只需将它看作函数名即可。

1. 函数探幽

1.内联函数

常规函数和内联函数之间的主要区别不在于编写方式，而在于C++编译器如何将它们组合到程序中。如果执行函数代码的时间比处理函数调用机制的时间长，则节省的时间将只占整个过程的很小一部分。反之，内联调用可以节省非内联调用使用的大部分时间。

区分：常规函数将流程转到独立的函数；

内联函数用内联代码替换函数调用。

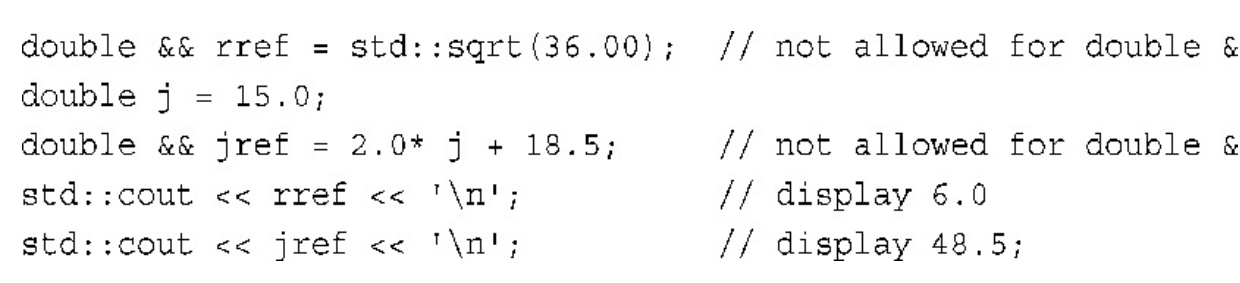
通常省略原型，将整个定义（即函数头和所有函数代码）放在本应提供原型的地方。

2.引用变量

引用更接近const指针，必须在创建时进行初始化，一旦与某个变量关联起来，就将一直效忠于它。

1. 右值引用（rvalue reference）

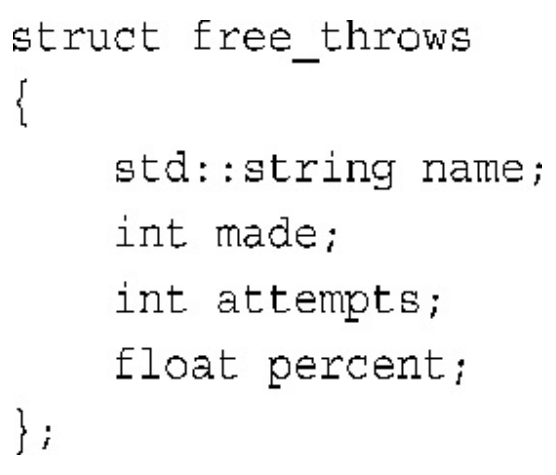
这种引用可指向右值，是使用&&声明的，例如：



1. 将引用用于结构

使用结构引用参数的方式与使用基本变量引用相同，只需在声明结 构参数时使用引用运算符&即可。

如有结构定义如下：



则可以这样编写函数原型，在函数中将指向该结构的引用作为参数：

void set\_pc(free\_throws & ft);

1. 何时使用引用参数

对于使用传递的值而不作修改的函数：

如果数据对象很小，如内置数据类型或小型结构，则按值传递。

如果数据对象是数组，则使用指针，因为这是唯一的选择，并将指针声明为指向const的指针。

如果数据对象是较大的结构，则使用const指针或const引用，以提高程序的效率。这样可以节省复制结构所需的时间和空间。

如果数据对象是类对象，则使用const引用。

对于修改调用函数中数据的函数：

如果数据对象是内置数据类型，则使用指针。

如果数据对象是数组，则只能使用指针。

如果数据对象是结构，则使用引用或指针。

如果数据对象是类对象，则使用引用。

1. 默认参数

默认参数指的是当函数调用中省略了实参时自动使用的一个值，必须通过函数原型设置默认值。通过使用默认参数，可以减少要定义的析构函数、方法以及方法重载的数量。

1. 函数重载

函数多态（函数重载）让用户能够使用多个同名的函数。“函数重载”指的是可以有多个同名的函数，因此对名称进行了重载。可以通过函数重载来设计一系列函数，它们完成相同的工作，但使用不同的参数列表。

何时使用？如果需要使用不同类型的参数，则默认参数便不管用了，在这种情况下，应该使用函数重载。

1. 函数模板

函数模板是通用的函数描述，它们使用泛型来定义函数，其中的泛型可用具体的类型（如int或double）替换。

1. 内存模型和名称空间
2. 单独编译

为减少程序出现的问题，可以将原来的程序分成三部分：头文件（包含结构声明和使用这些结构的函数的原型）、源代码文件（包含与结构有关的函数的代码）、 源代码文件（包含调用与结构相关的函数的代码）。

注意：不要将函数定义或变量声明放到头文件中。在包含头文件时，使用“coordin.h”，而不是<coodin.h>。另外，不要使用#include来包含源代码文件，这样做将导致多重声明。

1. 存储持续性、作用域和链接性
2. 作用域和链接

作用域（scope）描述了名称在文件（翻译单元）的多大范围内可见。链接性（linkage）描述了名称如何在不同单元间共享。

1. 自动存储持续性

在两个不同函数中声明的变量只有在函数自身内部有效。代码块中定义的变量也一样。

1. 关键字register最初是由C语言引入的，它建议编译器使用CPU寄存器来存储自动变量：register int coun\_fast;
2. 静态持续变量

C++也为静态存储持续性变量提供了3种链接性：外部链接性（可在其他文件中访问）、内部链接性（只能在当前文件中访问）和无链接性（只能在当前函数或代码块中访问）。



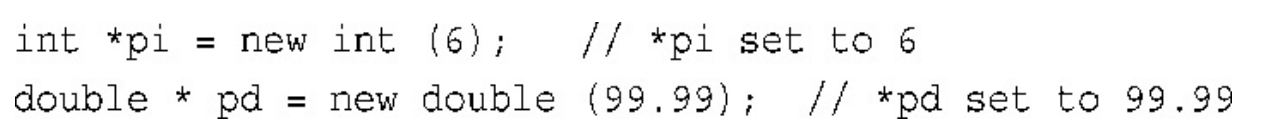
（5）函数和链接性

C++不允许在一个函数中定义另外一个函数，因此所有函数的存储持续性都自动为静态的，即在整个程序执行期间都一直存在。

1. new

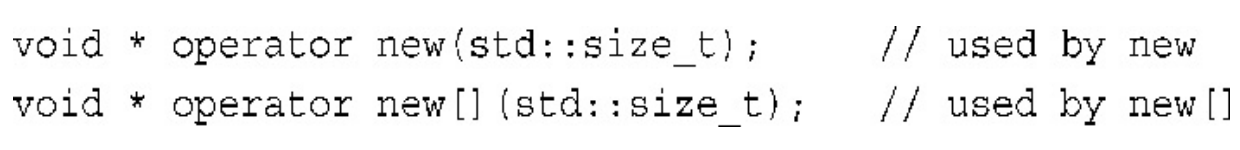
动态内存由运算符new和delete控制，而不是由作用域和链接性规则控制。

1. 使用new运算符初始化



要初始化常规结构或数组，需要使用大括号的列表初始化。

1. 运算符new和new []分别调用如下函数：



1. 定位new运算符

通常，new负责在堆（heap）中找到一个足以能够满足要求的内存块。它还有另一种变体，被称为定位（placement）new运算符，它让您能够指定要使用的位置。要使用定位new特性，首先需要包含头文件new。除需要指定参数外，句法与常规new运算符相同。

1. 名称空间
2. 传统C++名称空间

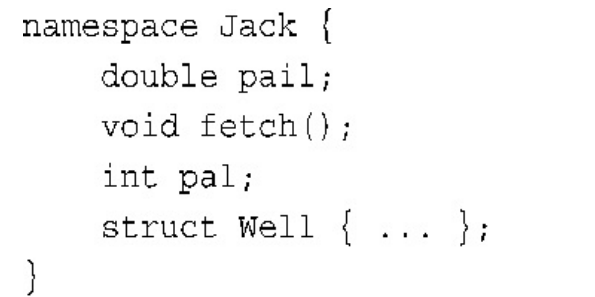
声明区域（declaration region）：是可以在其中进行声明的区域。

潜在作用域（potential scope）：变量的潜 在作用域从声明点开始，到其声明区域的结尾。因此潜在作用域比声明区域小，这是由于变量必须定义后才能使用。

1. 新的名称空间特性

通过定义一种新的声明区域来创建命名的名称空间，以提供一个声明名称的区域。一个名称空间中的名称不会与另外一个名称空间的相同名称发生冲突，同时允 许程序的其他部分使用该名称空间中声明的东西。

例如，使用namespace创建名称空间的做法：



using声明由被限定的名称和它前面的关键字using组成：

如using Jill::fetch;

1. 名称空间的其他特性

可以将名称空间声明进行嵌套，也可以在名称空间中使用using编译指令和using声明。

1. 未命名的名称空间

可以通过省略名称空间的名称来创建未命名的名称空间：

