**C++学习笔记**

**【第2章 从一个最简短的c++程序讲起】**

1. (3)

* std::是什么？

std::是名字空间标示符，c++标准库中的函数或者对象都是在名字命名空间std中定义的，所以我们要使用标准库中的函数或者对象都要用std来限定。

* std在什么时候使用？

一般来说，std都是要调用c++标准库时使用。

如：使用标准库文件iostream时，要写上std；使用非标准文件iostream.h，不用写。

1. (4)

* iostream与iostream.h的区别

iostream.h为非标准的输入输出流，这个.h的头文件是c语言格式的，由于当时还没有名字空间这个说法，所以也就不存在std这个名字空间标示符。自然用iostream.h也就用不着std或者using namespace std了。

iostream为标准输入输出流，它是c++规范的带有名称空间的头文件，它包含在std名字空间内，在使用iostream标准库中的cout对象时要加上名字空间标示符std，而c格式的非标准的输入输出流iostream.h则不用。

* c++开始引入的名字空间有什么作用？

c++里引入名字空间(namespace)的目的是为了避免和减少命名冲突。一旦程序写大了，就很难避免重名，特别是多人合作的情况下。过去c中的解决办法只有靠人为取长名字避免重名。这样做会使得一些名字看上去没有意义或者难以理解，而且程序员在写程序的时候，也受这个问题的限制，不能自由地命名自己使用的变量或者函数。而有名字空间(namespace)就不存在这些问题了，这就是c++引入namespace这个概念的好处。

**【第6章 面向对象程序设计】**

1. (6.2)

* 面向对象程序语言的主要特征：抽象、封装、继承、多态。

1. (6.12)

* 如果不想让某个成员函数修改成员变量的值，那么不妨将这个成员函数声明为const，即在函数头和函数体中间加const。其它函数亦是如此。
* 当函数的返回值不需要改变时，在返回类型前加const。
* 如果输入参数以值传递的方式传递对象，则宜改用“const &”方式来传递，这样可以省去临时对象的构造和析构过程，从而提高效率。
* 在编程时一定尽量多用const，对于不应当改变对象的成员函数都应该声明为const，这样假如该成员函数试图去修改该对象的成员变量，编译器就会提示错误，从而达到帮助你查找错误的目的。

**【第8章 指针】**

1. (8.17)

* 指针有一些“似是而非”的特征：

1. 指针消亡了，并不表示它所指的内存会被自动释放。
2. 内存被释放了，并不表示指针会消亡或者成了null指针。

* 注意：

1. new/delete必须配对使用，尤其是在函数体中。
2. 指针变量在创建的同时应当被初始化，要么将指针设置为null，要么让它指向合法的内存。
3. 释放后的指针应该被赋为null。
4. 不要让指针操作超越了变量的作用范围。
5. (8.19.1)

* 常量指针：int \* const p 该指针自身的值是不可改变的，但是它指向的目标却是可以改变的。

1. (8.19.2)

* 指向常量的指针：const int \* p 该指针指向的目标不可修改，但是该指针可以被修改。

1. (8.19.3)

* 指向常量的常指针：const int \* const p 该指针指向的目标是不可修改的，并且该指针也不可修改。

**【第9章 引用】**

1. (9.14)

* 指针与引用的区别：

1. 指针可以为空，引用不能为空；
2. 指针可以被赋值，引用不能被赋值；
3. 指针可以指向堆中空间，引用不可以指向堆中空间。

* 引用的主要功能是传递函数的参数和返回值。
* “引用传递”的性质像“指针传递”，而书写方式像“值传递”。
* “引用”可以做的任何事情“指针”也都能够做，为什么还要“引用”这东西？

答案是“用适当的工具做恰如其分的工作”。

1. (9.17)

* 无意中使用空引用会令程序出错。

1. (9.19)

* 对象在哪里创建，就在哪里释放。

**【第10章 深入函数】**

1. (10.5.2)

* 构造函数的初始化顺序：按照数据成员列表的顺序来进行构造的。
* 析构函数的顺序与数据成员列表的顺序恰好相反！
* 两者都和赋值时的顺序无关！

1. (10.5.2)

* 对成员列表初始化最好在构造函数函数头和函数体中间加冒号初始化，这样可以避免常量和引用不能初始化！

格式：函数名(变量) : 私有成员(变量) { }；

1. (10.8)

* 如果创建了构造函数，不管带不带参数，编译器都不会再提供默认构造函数。假如还想要不带参数的构造函数，就必须自己再创建一个。

1. (10.11)

* 可以把数字当做一个对象赋给一个人对象，在进行赋值运算时，先对数字进行类转换，同时判断该数字是否与构造函数的参数类型是否匹配，假如匹配就用构造函数创造一个对象并赋值。在析构函数前使用explicit关闭类的强制转换。避免这种转换出现意外。

1. (10.13)

* 拷贝构造函数的调用时机

1. 对象以值传递的方式传入函数参数；
2. 对象以值传递的方式从函数返回；
3. 对象需要通过另外一个对象进行初始化（在定义时）。
4. (10.13)

* 创建自己的拷贝构造函数解决浅层拷贝导致的迷途指针问题。深拷贝和浅拷贝的定义可以简单理解成：如果一个类拥有资源(堆，或者是其它系统资源)，当这个类的对象发生复制过程的时候，这个过程就可以叫做深拷贝，反之对象存在资源，但复制过程并未复制资源的情况视为浅拷贝。浅拷贝资源后在释放资源的时候会产生资源归属不清的情况导致程序运行出错。浅拷贝很容易出现问题，例如两次析构会找不到释放的内存而出错。
* 在C++中，析构函数是用来释放资源的，如果你的类中指针类型 ，并且你是在堆上动态分配的空间（无论你是用的new 或者是malloc等分配资源的函数 ），那么你需要对他们进行释放。析构函数在你的对象被释放的时候调用，用来释放对象占用的动态资源，如果你的对象中的成员没有在堆上分配资源 ，那么析构函数中可以什么都不写，如果有，那么需要在析构函数中调用delete释放占用的资源（当然你也可以手动的每次申请了，在不需要的时候释放掉，这样不免有点辛苦），如果不释放堆上分配的资源 ，会造成内存泄露，危害很大。

【例】

#include <iostream>

class A

{

public:

int \*t;

A() {t = NULL;}

~A()

{

if(t != NULL)

{

delete t;

t = NULL;

std::cout << "t is released !" << std::endl;

}

}

};

int main()

{

A a;

a.t = new int; // 在堆上给t分配空间

\*a.t = 5;

std::cout << a.t << " : " << \*(a.t) << std::endl;

return 0;

}

在析构函数处设置一个断点，或者F11单步调试一下 ，当对象a给释放释放时，会调用a的析构函数释放a占用的资源，其中a中的成员t占用了堆上的资源，需要被释放。

你至少要保证，堆上申请的一定要释放，如果你的指针t的空间不是堆上的空间 ，那么就不需要delete ，delete和new是配对的 。

如果你的程序是这样子的 ：

#include <iostream>

class A

{

public:

int \* t;

A() {t = NULL;}

~A() {std::cout << "invoked! but nothing to be free!" << std::endl;}

};

int main ()

{

A a;

int num = 8;

a.t = &num; // a.t的地址是栈上的 ，那么不需要释放资源，程序会自动的释放。

std::cout << a.t << " : " << \*(a.t) << std::endl;

return 0;

}

**【第11章 运算符重载】**

1. (11.3)

* 重载前置运算符与前置与后置自加运算符的区别是是否含有参数。参数无意义，仅仅是作为区分。

1. (11.9.5)

* c++允许级联，即a=b=c;

1. (11.11)

* 不能重载的运算符：“.” “::” “\*” “: ?” “#”。

**【第12章 继承】**

1. (12.3)

* 继承格式： class Son : public Father

public是指以公有的方式访问继承的成员。

* 继承机制：在构造子类对象时，会首先调用父类的构造函数，再调用子类的。析构时相反。

1. (12.5)

* 一个类的私有成员只有这个类的公有成员可以访问（封装的特性）。
* 公有成员可以被所有的类所访问。
* 保护成员是受保护的，其他类不能访问。能被派生的子类（间接）访问。

1. (12.6)

* 公有派生的公有权限：父类的公有成员在子类中也是公用成员，所以子类可以直接调用继承父类的公用成员。

1. (12.7)

* 公有派生的保护权限：父类的保护成员在子类中也是保护成员，所以必须用父类的公有成员进行访问，具体就是子类用自身公有成员的函数来调用父类公有成员的函数来访问父类的保护成员。不能直接用子类的公有成员直接访问继承父类的保护成员，这是由于被保护了。

1. (12.8)

* 公有派生的私有权限：子类的公共成员不能访问父类的私有成员。但子类可以用继承的公有成员来访问继承的私有成员。

1. (12.9)

* 子类可以赋值给父类，反之不可。
* 父类的指针可以指向子类，对指针的操作仍是对父类，而不是子类。
* 父类可以作为子类的别名。

1. (12.10)

* 私有派生：父类的公有和保护成员在子类中都是私有的，所以子类可以调用自身的公有成员来访问继承的成员。
* 总结：无论是那种派生方式，派生类无法直接使用不可访问变量的，换句话说，派生类不能访问基类的私有成员。

具体如图：

|  |  |  |
| --- | --- | --- |
| **基类成员访问控制** | **继承访问控制** | **在派生类中的访问控制** |
| public | public | public |
| protected | protected |
| private | 不可访问 |
| public | protected | protected |
| protected | protected |
| private | 不可访问 |
| public | private | private |
| protected | private |
| private | 不可访问 |

* + 继承后的法则：

1. 派生类的成员函数访问基类成员时：

无论是公有、私有、受保护继承，基类的私有变量均不可访问，但继承后属性为公有、受保护成员均可以直接访问。

1. 派生类的对象类外访问基类成员时：

若受保护、私有继承时，派生类对象均不可访问基类的成员（无论是哪种访问属性）。

若公有继承，派生类的对象可以访问基类的公有成员。

一句话：就派生类而言，基类私有变量不可访问，其他成员的访问情况，按具体继承后的属性而定。

1. (12.12.2)
   * 多重继承的构造顺序与继承时给定的顺序相同，析构则相反。
2. (12.13.3)
   * 派生类构造函数向基类构造函数传递参数：在派生类的构造函数处用基类的构造函数初始化，基类成员在函数头初始化，派生类成员在函数体赋值，这样可以分清基类成员和子类成员，增强代码的可读性，减小出错的可能性，效率更高。
3. (12.14)
   * 子类的成员函数没有两义性，父类的成员函数会产生两义性，解决办法是在调用函数名加成员名的限定。
   * 作用域分辨：用于区分各个类的函数或者变量的过程。使用的符号为类名和类标示符 :: 组成的作用域操作符。
4. (12.16.1)

* 一个派生类从多个基类派生，而这些基类又有一个共同的基类，那么在派生类中访问共同的基类中的成员时会产生两义性。

1. (12.16.2)

* 子类不能直接访问相隔的父类。调用时要搞清子类的父类，不能越级调用。

1. (12.18)

* 虚基类可以避免两义性。

1. 虚基类的作用

当一个类的部分或者全部基类来自另一个共同的基类时，这些直接从一级共同基类继承来的成员就拥有相同的名称，这样就可能出现派生的二义性；且在派生类对象中，这些同名成员在内存中同时拥有多个拷贝，可以使用作用域分辨符来唯一标识并分别访问他们，也可以将共同基类设置成为虚基类，这时从不同路径继承过来的该类成员在内存中就只有一个拷贝，这样就解决了唯一标识问题。

1. 虚基类的申明方式

virtual <继承方式> <基类名>

其中关键字 virtual 可以放在继承方式的后面。

**【第13章 虚函数】**

1. (13.1)

* Father \*p = new Son 好处：Son对象可以直接访问father的数据和函数。

c++允许将子类的对象赋给指向基类的指针，并且使用该指针可以访问基类的数据和函数。这是c++多态性的一个特征，会根据指针指向哪个类调用哪个类的成员函数或者数据成员。

1. (13.2)

* 虚函数：在成员函数前加virtual，表示该函数是有多种形态的，即该函数可能被多个对象所拥有，而且功能不一，换句话说多个对象在调用同一名字的函数时产生的效果也不一样。那么系统在执行到有关键字virtual的函数时会自动判断是哪个对象调用了它，然后调用该对象的同名函数。
* 派生类中与基类的虚函数同名也为虚函数，为了增强程序的可读性，也应该在函数前加virtual。

1. (13.4.2)

* c++对重载函数使用静态联编，使用virtual后，c++对函数进行动态联编。
* 联编：将一个调用函数者联结上正确的被调用函数，这一过程叫做函数联编，一般简称为联编。 联编分为静态联编和动态联编。

因此在未加virtual说明时，该函数是静态联编，即被调和调用函数的关系以及它们的内存地址在编译时都已经确立好，运行时不再发生变化。这样的好处是速度快，因为运行的时候不用对各个对象的参数进行追踪，只需要传递参数、执行确定好的函数并并在函数调用完毕后清理内存即可。

与此相反，动态联编就要牺牲掉一些速度，因为每个函数调用在运行前是不可确立的，要随着用户的操作来执行相应的函数。

1. (13.5)

* 在编译时的静态联编：静态联编是指联编工作出现在编译阶段，这种联编又称早期联编，它解决了程序中的操作调用与执行该操作代码间的关系。在编译时所进行的这种联编又称静态约束，在编译时就解决了程序中的操作调用与执行该操作代码间的关系。对于静态联编来说，由于它已经确定了调用与被调用的关系，因此代码在编译时与执行时的操作效果都是一样的。

1. (13.6)

* 在运行时的静态联编：假设用指针在运行时再动态地指向某个对象，然后再用该对象调用它的成员函数，由于静态联编的对象与指针关系在编译时就已确定，所以运行时在想对它改变也是无效的。

1. (13.7)

* 在运行时的动态联编：编译程序在编译阶段并不能确切知道将要调用的函数，只有在程序执行时才能确切将要调用的函数，为此要确切知道该调用函数，要求联编工作要在程序运行时进行，这种在程序运行时进行联编工作称为动态联编，或称动态约定，又称晚期联编。

1. (13.8)

* 在使用虚函数中没有采用指针或引用，那么就无法实现动态联编。

1. (13.10)

* 系统是如何调用虚函数的？

每个对象创建虚函数时，对象都得记录这个虚函数，因此编译器建立了一个叫做t表的虚函数表。每个对象都有一个指向该表的指针，叫做虚表指针。该指针用来指向虚函数表。相反虚函数表也有一个指针指向该对象，当创建派生类对象的基类部分时，该对象的指针就自动初始化为指向虚函数表的正确部分。当调用派生类对象的构造函数时，这个对象就会添加到虚函数表中去，并将指针指向该对象的重载函数。

当使用指向基类的指针时，将根据对象的实际类型，将该对象的指针继续指向正确的函数。

1. (13.11)

* 在虚函数中使用成员名限定可以强行解除动态联编。会强制使用静态联编来调用指定成员名中的函数。

1. (13.12.1)

* 虚析构函数：一个派生类对象在创建时会首先调用基类的构造函数，然后调用该类的构造函数。一般情况下，在使用虚函数是时候，我们都会将派生类对象传递给指向基类的指针，那么如果指向派生类对象的指针被删除时会发生什么情况呢？如果析构函数是虚函数，那么就会进行正确操作，它会先调用派生类的虚析构函数，由于派生类的析构函数会自动调用基类的析构函数，因此构造整个对象都会被销毁。
* 一般情况下任何类的构造函数都可声明为虚析构函数，当指针被删除时，系统会获得对象运行时的类型并调用正确的构造函数。但要注意三点：

1. 由于析构函数不允许有参数，因此它不可能实现重载，那么一个类就只有一个虚析构函数。
2. 只要基类的析构函数被说明为虚函数，那么派生类的析构函数无论说明与否，都自然称为虚函数。
3. 在c++中虚构造函数是不存在的，因此也无法声明。
4. (13.12.2)

* 如果基类中定义虚函数，析构函数也应说明为虚函数。这样对内存的回收会更准确些。

**【第15章 链表】**

1. (15.2)

* 结构体的成员默认为公有成员，类的成员默认为私有成员。

1. (15.6.2)

* 字符串转换成整形

string str;

int a = atoi(str.c\_str());

注：str.c\_str()将C++的字符串转换成c语言格式的字符串

1. (15.2)

* 使用静态库

编译后的程序直接包含了调用MFC的部分库，文件可能会大一点，但是可以直接移到其他机器上运行

* 使用共享DLL

编译后的程序直不包含MFC库，文所以件会比较小，但是可如果将程序直接移动到一个没有安装MFC的机器上时，可能会导致找不到MFC的DLL。

* MFC：微软基础类(Microsoft Foundation Class)，是微软提供的一套基础类库
* WinMain会调用InitInstance函数对应用程序对象进行初始化，而WinMain类似于控制台的main函数，它不用程序员来编写，由类库提供，并在应用程序启动时调用。

1. (15.9)

* GDI类：GDI是Graphic Device Interface的缩写，意思是“图形设备接口”。由于程序员无法直接操作显示卡，所以MFC提供GDI类，用来做程序员与显卡沟通的桥梁。GDI含有许多绘图类，比如处理位图的CBitmap类，处理画刷的CBrush类，处理字体的CFont类，处理调色板的CPalette类，处理画笔的CP恩类以及图形设备接口区域的CRgn类。

1. (15.11)

* LoadImage函数可以加载位图

1. (15.12)

* 句柄是一个指向指针的指针

1. (15.13)

* 将图片在窗口中显示分为四个步骤

1. 将位图复制到内存中
2. 获得作画区域，即窗口
3. 将内存与窗口相关联
4. 将内存中图片粘贴到窗口中，完成绘画

**【第16章 多态性】**

1. (16.2)

* dynamic\_cast的作用是针对不同类之间的数据转换，它可以将一个基类的指针转化成一个派生类的指针。

格式：dynamic\_cast<转换类型\*>(要转换的指针)

* 注意：
  1. dynamic\_cast是RTTI中的一个方法，RTTI是"runtime type information"的缩写，意思是：运行时类型信息。它提供了运行时确定对象类型的方法。它有两个方法，分别是typeid（运算子）和danamic\_cast。
  2. 在使用dynamic\_cast时要在编译器启动RTTI。
  3. 在程序中应当尽量少使用RTTI，因为RTTI是硬性的通过将基类指针转换为派生类指针达到访问派生类函数的目的，所以它实现的不是多态性。

1. (16.4)

* 可以定义一个抽象类，多个父类采用虚基类的方式继承抽象类，从而避免子类多重继承时产生的两义性问题。抽象类没有什么实际作用，只是一个抽象的类。就像人类，它不是某个具体的人，没有什么实际功用，只是在人的大脑中抽象出来用来概括人类的名词。不过它可以保证所有进入人类这个级别的动物都具有人类的行为特征，不存在狮子或者恐龙也成为人类的这种可能。由于抽象类只是一个接口类，因此我们又称为adt。需要注意的是这个抽象类并不是真正的抽象类，它只是一个虚拟的抽象类，因为它的虚函数仍然具有功能，那就是什么事也不做，真正的的抽象类必须具备一个或一个以上的真正没有任何功能的虚函数，这个虚函数的存在仅仅是为了让它的子类来继承并具体化功能。

1. (16.5)

* 一个虚函数通过初始化为0就变成了纯虚函数。

格式：virtual 函数头 = 0。

注意：纯虚函数是彻底无任何功能的，你不能直接来调用它，因为它是抽象的，它只有被子类继承并赋予新功能后才能被使用。

* 一个类中可以有多个纯虚函数，包含有纯虚函数的类叫做抽象类。它不是上一节的那个模拟的抽象类，而是真正的抽象类。由于任何一个从抽象类派生的新类都会继承纯虚函数的特征——无任何功能，因此要创建这个新类的对象必须为每一个纯虚函数赋予功能。抽象类只是为派生类提供一个接口的作用。
* 注意抽象类不是实际存在的类，所以你不能定义一个抽象的对象。但可以指向抽象类的指针，这个指针既是一个指向抽象类的指针，同时又是一个指向基类的指针，因此在程序编译或运行时，这个指针可以动态地指向子类的对象，从而实现程序的多态性。

1. (16.7)

* 一个抽象类仍然可以派生出抽象类，只要该类没有把纯虚函数全部覆盖掉，仍然可以派生出来的抽象类。假如其子类没有将它的纯虚函数全部覆盖掉，那么该子类也是抽象类。

1. (16.8)

* 慎用多重继承：虽然多重继承比单一继承有很多优点，但是由于虚基类的定义复杂，所以很多程序员不愿意使用它。至于什么时候使用单一继承，什么时候使用多重继承，只需要记住一点，那就是：在用单一继承就可以实现的情况下，不要使用多重继承。

**【第17章 特殊类成员】**

1. (17.1)

* 公有的静态成员变量：能够被所有对象共享使用的变量。

格式：在公有成员中使用关键字static声明。 例：static int a;

* 静态成员变量不专属于某个对象，它是属于整个类中所有对象的成员变量，因此要实例化该类的对象的时候无法分配内存，这样就必须在全局为它开辟内存空间。注意一定不要忽略定义静态变量这个步骤，否则编译器会报错。 例：int Oneclass::a = 0;
* 静态成员变量和普通成员变量的区别：

1. 静态成员变量时属于整个类的全局变量，它不属于某个对象，所有对象都可以调用。普通成员变量是属于类中对象的变量，它不能被所有对象共享。
2. 静态成员变量必须在全局进行定义，二成员变量由于是属于某个对象的，而不是属于整个类的，因此不用在全局进行定义。
3. 在使用静态成员变量时，可以不限定为对某个具体的对象，至于类名连用即可。

如：Oneclass::a

而不用：Oneclass a;

a.total;

1. 静态成员在没有对象之前就存在。
2. (17.2)

* 私有的静态成员变量：在私有成员中声明。也是在全局部分定义。只有创建一个对象，用该对象的成员函数才能访问。
* 如果要在一个类中所有对象间共享某种数据，不如将该数据声明为静态成员变量。如果不想让该类之外的所有函数都可以访问该数据，可以将其定义为私有成员，那么只有该类的公有成员才可以访问
* 它。当然访问的前提是必须创建一个属于该类的对象。

1. (17.3)

* 静态成员函数与静态成员变量一样，它不单属于一个对象，而是属于整个类。

格式：void static show() { }

* 静态成员函数在未创建对象时也可以使用。
* 注意：静态成员函数是由于属于整个类的，所以它不能访问某个对象的成员变量，因为它没有指向该对象的this指针。不过它可以访问该类的静态成员变量。可以通过对象来访问静态成员函数。
* 关于静态成员：

1. 尽量使用类成员名限定来访问静态成员，不要使用对象名，因为静态成员是类的成员而不是对象的成员。
2. 静态成员（成员和函数）可以被继承，基类和派生类对象都可以共享该静态成员。
3. 类中任何成员函数都可以访问静态成员，但是静态成员函数不能直接访问非静态成员，只能通过对象名访问该对象的非静态成员函数。这是因为静态成员函数是属于整个类的，没有特指向某个对象的this指针。
4. 静态成员函数不能被说明为虚函数。
5. (17.5)

* 数组名是指向数组第一个元素的常量指针，函数名也是指向函数第一条指令的常量指针。一般来说，在不出现异常的情况下，程序在编译后，每个函数都有一个首地址，也就是函数的第一条指令的地址，我们用一个指针来保存这个地址，那么这个指针就是函数指针，该指针可看做是函数名，因此可以通过该指针调用函数。
* 函数指针的声明方法：int(\*func)(int);

该语句声明了一个int型指针func，它指向一个函数，这个函数带有一个int型参数并返回int值。

注意：一个指向函数的指针必须确保该函数被定义且分配了内存，否则它将指向一个空地址，这是指针的大忌。

* 使用函数指针可以减少一些重复的代码，简化代码，因为函数指针名可以看做函数名的代号，我们可以通过它直接调用函数，所以函数指针经常会在条件或者判断语句里出现，以便于用户选择调用不同名字但又类型和参数相同的函数。另外要注意的是：函数指针可以指向某个函数，但是前提是被指向的函数的参数和返回值都与该函数指针被声明时的返回值和参数相吻合。

1. (17.6)

* 函数指针数组的声明方法：int(\*p[5])(int &, int &);

该语句声明了一个有5个元素的数组指针。它与函数指针的区别仅仅是指针数量的不同。

1. (17.7)

* 函数指针也可以作为函数的参数。

1. (17.8)

* 使用typedef简化函数指针的声明

例：typedef int(\*p[5])(int &, int &);

将该指针简化为p

1. (17.9)

* 类的成员函数指针声明方法：void (a::\*p)(int a, int b);

1. (17.10)

* 假如将许多函数指针放在一个数组中，那么这个数组就叫成员函数指针数组，并且可以用成员函数的内存地址来对数组的各个成员函数指针进行初始化。
* 注意：

1. 要在类的特定的对象中调用成员函数指针。
2. 能不用成员函数指针就不用。

**【第18章 字符串】**

1. (18.1-18.5)

* cin遇到空字符就停止输入。使用cin.get(数组名,可保存字符的个数)可以避免这个问题。

cin.get的结束标志是‘\n’，也就是换行。因此遇到空格不会结束，而是把空格也看做一个字符。同时在输入结束后，也就是用户输入一个回车后，cin.get自动为当前接受输入的数组添加字符串结束标志‘\0’。实际保存（第二个参数-1）个参数，如果超出则舍去，最后必有字符串结束标志‘\0’。

* cout和cin一样遇到空字符会停止输出，但不同的是cout可以输出空格、制表符等空字符，或者说不可见字符。
* 关于cin和cout

cin遇到不可见字符（指制表符和空格）和空字符就会停止输入。

cout只遇到空字符才会停止停止输出。

空格是ASCII码为32，而空字符的ASCII码为0。

* strlen()返回的是字符串结束标志‘\0’之前的字符串长度。

sizeof()返回的是数组的长度。

1. (18.6-18.10)

* char型字符串是c语言风格的字符串，它是用来保存字符串的，在c++时代，由于诞生了类，出现了string型字符串。

种风格的字符串是用string类来定义字符串的，因此要使用这个string类，必须要在程序开头添加头文件string，string类存在于名字空间std中，这样就必须用std::string指令，或者直接用std::string来访问它。

* string类在定义时隐藏了字符串的数组性质，因此在使用string类定义字符串时不用考虑如何将字符串放在数组中，string类已经做好了这一部，只需像定义一个变量那样使用它。
* 由于string是个类，那么string定义的字符串就可以看作是string类的一个对象，或者可以这样说：C++风格的字符串就是用对象来保存字符串的。
* 由于string类的成员函数重载了比较运算符(==)之类的关系运算符，因此string类允许使用运算符对它的字符串对象进行比较。这是string型字符串与char型字符数组的第一个区别。

1. (18.12)

* string类中使用operater函数重载了运算符，因此string对象之间可以直接赋值。
* string类中还有一个专门的赋值函数assign，该函数可以取第二个字符串的任意字符复制给第一个字符串，string类具有自动调节字符串大小的功能，会将多余字符删除。

1. (18.14)

* string类的对象可以直接相加，而且string类有一个特殊的功能，它会随着字符的增多或者减少自动调节数组的大小。
* str1.c\_str() 将string字符串转换成char型字符串，也是讲对象转换为数组，转换后数组带有一个字符串结束标志，可以用strlen(str1.c\_str())函数求其有效长度。
* 也可以用str1.size()求出有效长度
* 也可以用str1.length()求出有效长度，length()是早期版本string类中的成员，size()函数是后添加进去的，为的是兼容STL。

1. (18.15)

* string型字符串部分合并

1. string &operator+=(const string &s); //把字符串s连接到当前字符串的结尾
2. string &append(const char \*s); //把c类型字符串s连接到当前字符串结尾
3. string &append(const char \*s,int n);//把c类型字符串s的前n个字符连接到当前字符串结尾
4. string &append(const string &s); //同operator+=()
5. string &append(const string &s,int pos,int n);

//把字符串s中从pos开始的n个字符连接到当前字符串的结尾

1. string &append(int n,char c); //在当前字符串结尾添加n个字符c
2. string &append(const\_iterator first,const\_iterator last);

//把迭代器first和last之间的部分连接到当前字符串的结尾

1. (18.16)

* string型字符串的替换

1. string &replace(int p0, int n0,const char \*s);

//删除从p0开始的n0个字符，然后在p0处插入串s

1. string &replace(int p0, int n0,const char \*s, int n);

//删除p0开始的n0个字符，然后在p0处插入字符串s的前n个字符

1. string &replace(int p0, int n0,const string &s);

//删除从p0开始的n0个字符，然后在p0处插入串s

1. string &replace(int p0, int n0,const string &s, int pos, int n);

//删除p0开始的n0个字符，然后在p0处插入串s中从pos开始的n个字符

1. string &replace(int p0, int n0,int n, char c);

//删除p0开始的n0个字符，然后在p0处插入n个字符c

1. string &replace(iterator first0, iterator last0,const char \*s);

//把[first0，last0）之间的部分替换为字符串s

1. string &replace(iterator first0, iterator last0,const char \*s, int n);

//把[first0，last0）之间的部分替换为s的前n个字符

1. string &replace(iterator first0, iterator last0,const string &s);

//把[first0，last0）之间的部分替换为串s

1. string &replace(iterator first0, iterator last0,int n, char c);

//把[first0，last0）之间的部分替换为n个字符c

1. string &replace(iterator first0, iterator last0,const\_iterator first, const\_iterator last);

//把[first0，last0）之间的部分替换成[first，last）之间的字符串

【第19章 代码重用】

1. (19.1)

在用静态static与外部extern定义变量时，系统立刻给其分配内存，而定义自动类auto与强制放入寄存器register类型变量时，只有当使用时才分配内存。

常用的线性结构有：线性表，栈，队列，双队列，数组，串。

串是零个或多个字符组成的有限序列。一般记S=‘a1a2....an ’其中，S是串名，单引号括起的字符序列是串值；ai（1〈=i〈=n）可以是字母，数字或其它字符；串中所包含的字符个数为该串的长度。长度为零的串称为空串，它不包含任何字符。

在程序运行中局部变量所在的函数被调用时，编译系统根据需要临时分配内存，调用结束空间解放；全局变量一经定义，编译系统为其分配固定的内存单元，在程序运行的自始至终都占用固定的单元。在考虑内存未知的情况下，最好不要使用全局变量。