#### 继承

继承：继承是软件复用的一种方式，通过继承，可以吸收现有数据和行为来创建新类，并增贴新的性能增强此类。

public：基类的public成员可以在它自己体内访问的，而且程序中凡是出现基类或其他派生类的对象句柄处也是可以访问的。

private：只能在基类体内即友元函数中访问。

protected：提供了一种介于public和private访问之间的中间保护层次。基类的protected成员可以在基类的体内被基类的成员和友元访问，也可以由基所派生出的任何类的成员和友元访问。

派生类的成员函数引用基类的public和protected成员时，只需直接使用成员名。当派生类的成员函数重新定义基类的成员函数是，可以通过在基类成员名前加上基类名和二元作用域分辨运算符而在派生类中访问相应的基类成员。

包含头文件的原因：编译器需要使用类定义来决定类对象的大小，以及编译器可以据此判断派生类是否正确地使用了有基类继承而来的成员。

protected注意事项：

1. 派生类对象不必使用成员函数设置基类的protected数据成员的值，因此，派生类可以很容易地将无效的值赋给基类的protected数据，导致对象处于不可靠的状态。
2. 派生类成员函数实现很可能太依赖基类的实现。实际上，派生类应该只依赖基类提供的服务，而不应该基类的实现。当基类中有protected数据时，如果修改了基类的实现，那么同时还需要修改基类所有的派生类。

派生类中的构造函数和析构函数：派生类不会继承基类的构造函数、析构函数和重载的赋值运算符。但是，派生类的构造函数、析构函数和重载的赋值运算符可以调用基类的构造函数、析构函数和重载的赋值运算符函数。

##### 多态性

类继承层次中对象之间的关系：派生类的对象可以当成它的基类对象进行处理，因此可以创建乐意基类指针的数组，数组元素指向许多派生类对象。尽管事实上派生类对象和基类对象是不同的类型，但编译器允许这种操作，因为每个派生类对象都是一个基类对象。然而，不能把基类对象当成任何一个派生类对象处理。

从派生类对象调用基类函数：

1. 将基类指针指向基类对象
2. 将派生类指针指向派生类对象，并调用派生类的功能。
3. 通过基类对象指向派生类对象。

virtual函数：不用虚函数的话，句柄类型决定了哪个类的函数被调用的。如果利用virtual函数，调用哪个版本的virtual函数有句柄所指向的对象类型决定，而非句柄类型。

当virtual函数通过按名引用特定对象和使用圆点成员选择运算符的方式别调用是，调用哪个函数在编译时已经决定了（称为静态绑定），所调用的virtual函数正是为该特定对象所属的类定义的函数，并不是多态性行为。因此，使用virtual函数进行动态绑定只能通过指针句柄来完成。

抽象类和纯virtual函数：有些情况下，定义程序员永远不断算实例化任何对象的类是有用的，这样的类称为抽象类。因为通常抽象类在类的继承层次结构中作为基类，所以我们称它们为抽象基类。

构造抽象类的目的是为其他类提供适合的基类。可以用来实例化对象的类称为具体类。具体类提供其定义每个成员函数的实现。

继承层次不需要包含任何抽象类，很多优秀的面向对象系统都有抽象基类打头的类继承层次。

通过声明类的一个或多个virtual函数为纯virtual函数，可以是一个类成为抽象类。一个纯virtual函数是在声明时初始化值为0的函数，如下所示：

virtual void draw() const = 0; //pure virtual function

“=0”称为纯指示符。纯virtual函数不提供函数具体实现，每个派生的就具体类必须 重写所有基类的纯virtual函数定义，提供这些函数具体实现。

###### 模板

模板：函数模板和类模板可以使程序员只需指定一个单独的代码段，就可以表示一整套称为函数模板特化的相关函数或是表示一整套称为类模板特化的重载函数或是表示一整套称为类模板特化的相关的类。

函数模板：重载的函数通常对不同类型的数据进行相同的或者相似的操作，程序员只需要写一个单函数模板的定义，它通过显式或隐式提供的实参类型进行函数调用，而编译器产生不同的目标代码函数来适当地处理每个函数调用。

模板形参名只能在模板头部的参数列表中声明一次，但是可以在函数头和函数体内重复使用。函数模板形参名不必是唯一的。

重载函数模板：编译器使用重载的方式来调用适当的函数。一个函数模板可以被一个具有相同函数名但不同函数参数的非模板函数所重载。编译器在函数调用是进行匹配处理来决定调用哪个函数。首先，编译器会找出所有和调用函数名匹配的函数模板，然后根据调用函数的参数类型对函数模板进行特化，接着编译器找出所有与调用函数里的普通函数想匹配的函数。

类模板：

类模板的非类型参数：非类型参数可以作为默认的参数并作常量处理。

默认类型参数：作为模板的默认类型，必须在模板参数列表的最右边。当用两个或两个以上的默认类型初始化一个类时，如果其中一个默认参数不是在模板参数列表的右边，那么该参数右边的所有参数都将被忽。

模板和静态成员的注意事项：一个静态数据成员的副本将被所有类的对象共享，并且静态数据成员都必须在文件范围内初始化。

每一个有相同类模板实例化产生的类模板特化都有它自己的类模板静态成员数据成员的副本。一个模板类特化产生的所有对象都共享一个静态数据成员。另外，和非模板类的静态成员一样，类模板特化的静态成员数据必须被定义，并且尽可能在文件范围内初始化。每一个类模板特化都将得到一份属于它自己的类模板静态成员函数的副本。

#### 异常处理

异常处理是程序员能够将错误处理代码从程序执行的主流程中分离处理啊，这样能提高程序的清晰度以及是程序更易于修改。程序员能够决定处理它们选择的任何以尝：所有异常、同一类型的所有异常或者相关的一组异常。

异常处理过程：

1. 定义一个异常类，描述可能发生的问题类型
2. try语句快封装代码，是指能够进行异常处理
3. 定义一个catch处理器处理异常
4. 异常处理的终止模式：如果由于try语句块中的一条语句发生了异常，那么这个try语句块就会终止。然后，程序搜索能够处理已经发生异常的第一个catch处理器。程序通过将抛出的异常类型和每一个catch的异常参数进行比较来找到匹配的catch，并定位在那里。如果抛出的类型和异常的参数类型相同，或者抛出的类型是异常参数类型的继承类，那么就能匹配。当匹配成功是，该catch处理器体中的代码就会被执行。执行结束是，异常被认为已经处理并且在catch处理器体内定义的局部变量在域外消失。程序控制并不会回到异常发生点，因为try语句块已经终止了。控制将从try后面的最后一个catch处理器之后的第一条语句开始。

什么时候使用异常处理：异常处理是设计用来处理同步错误的，这些错误发生在一个语句正在执行的时候。异常处理并不处理异步事件，例如键盘点击等，这些与程序的控制流并行并互相独立的。

重新抛出异常：异常处理器在接收到异常时，有可能判断该异常是它不能处理的或是只能部分处理的。无论哪种情况，处理器都可以通过throw语句重新抛出异常。无论处理是否额能够处理异常，处理器都可以为了处理器外更进一步地处理而重新抛出异常。下一个封装的try语句将检测这个重新抛出的异常，而列在该封装try语句块后面的一个catch处理器试图处理该异常。

异常规格：一个可选的异常规格列举了一系列函数可以抛出的异常。如果函数抛出了非异常规格中的异常，则异常处理机制将调用函数unexpected来结束程序。没有提供异常规格的函数可以抛出任何异常。在函数参数表后防止throw()，代表一个空的异常规格，表明函数不抛出异常。

处理意料之外的异常：

1. 对于抛出的异常，异常机制找不到匹配的catch块
2. 析构函数试图在堆栈展开时抛出异常
3. 在没有异常处理试图重新抛出异常
4. 调用函数unexpected将默认调用函数terminate

堆栈展开：当异常被抛出但没有在一个特定的域内被捕获时，该函数调用堆栈就会展开，并试图在下一个外部捕获异常。展开函数调用堆栈意味着在其中没有捕获异常的函数将会结束，并且控制将返回到最初调用该函数的语句，而此函数中所有的局部变量将被销毁。如果该语句被一个try语句快封装，那么就会试图捕获该异常。如果该语句没有被一个try语句块封装，堆栈展开将再次发生。

异常和继承：各种异常可以有公共基类派生出来。如果catch处理器捕获到基类的异常对象的指针或引用，它同样能够捕获该基类的public派生类的所有对象的指针或引用，这样可以利用多态来处理相关错误。

处理new失败：

1. 抛出bad\_alloc
2. 使用函数set\_new\_handler处理new失败

new处理器要完成一下任务的一个：

1. 通过释放其他动态分配的内存来增加更多的可用内存，并且返回运算符new来尝试再次分配内存
2. 抛出bad\_alloc型异常
3. 调用函数abort或exit来结束程序

类auto\_ptr和动态内存分配：一个普遍的编程习惯是进行动态内存分配，分配指针地址，利用指针操作内存，以及在不需要内存时回收内存空间。如果异常发生在成功分配内存后、delete语句执行之前，那么就会发生内存泄漏。auto\_ptr类对象维护指向动态分配内存的指针。当一个auto\_ptr对象的析构函数别调用时，它将对其指针的数据成员执行delete操作。由于auto\_ptr类模板提供了\*和->，所以auto\_ptr对象可以作为一般指针变量使用。

###### 文件处理

**数据层次**：位，字符，字符集：所有在特定的计算机上用来编写程序以及表示数据项的字符的几何称为计算机的字符集、字节、字段：由字符组成

**记录**：一组相关的字段

**关键字**：为方便的获得一个文件的某些特定记录，每个记录中至少有一个字段被选为记录关键字。

文件和流：C++将每一个文件看成是字节序列。每个文件都以一个文件结束符或是以存储在操作系统维护、管理的数据结构中的一个特定字节数作为结尾。当打开一个文件或是创建一个对象时，将一个流关联到这个对象上。cerr和clog对象允许程序将错误信息输出到屏幕和其他设备。

创建顺序文件：创建一个ofstream对象后尝试打开它是，程序会测试打开操作是否成功。使用重载的ios运算符成员函数operator！来判定打开操作是否成功。如果在打开操作中，failbit或badbit位中的任何一个被设置了，则该条件返回true。这些错误可能是尝试打开并读取一个不存在觉得文件，在没有权限的情况下对文件进行读写操作，或是在打开文件并写入是没有磁盘空间。另一个重载的ios运算符成员函数operator void\*将流转换为指针，因此可以对流进行：0（为空指针），非0（其他指针的值）。当把一个指针值用做判定条件时，C++将一个空指针转换为bool值false，将一个非空指针转换为true。如果failbit或者badbit位被设置，则返回0.运算符void\*函数可以用来测试一个输入对象的文件结束，而不必在输入对象上显示地调用eof成员函数。

从顺序文件读取数据：ifstream的构造函数将接收到的文件名和文件打开模式作为参数。为了顺序地从文件中取得数据，程序一般从文件起始位置开始连续地读取所有数据，知道找到所需要的数据为止。在程序的执行的过程中，有可能需要顺序地读取文件好几遍。istream和ostream都提供了成员函数来重定位指针。在istream中，这个成员函数为seekg；在ostream中，这个成员函数为seekp。在istream对象都有一个get指针来之处文件中下一个输出的字节号，每一个ostream对象都有一个put指针来之处文件中下一个输出的字节号。

更新顺序文件：格式化的输入\输出模式通常不会用来更新记录。

随机存取文件：随机存取文件的一条记录可以在不查找其他记录的情况下直接对其进行访问，应用程序如果想要使用随机存取文件，则必须自己创建。可以在不破坏文件中其他数据的情况下将数据插入到一个随机存取文件中。之前存入的数据页可以在不重写整个文件的情况下被更行或删除。

创建随机存取文件：ostream的成员函数write从内存中一个指定位置输出固定数目的字节到指定的流。当流被关联到文件时，函数write在文件中从”put“文件定位指针指定的位置开始写入数据。istream成员函数read则将固定数目的字节从一个指定的流输入到内存中指定地址开始的一部分空间。如果流被关联到一个文件，那么函数read在文件中从由get文件定位指针所指定的位置读取字节数据。

### string类和字符串处理

类的复制：就类对象而言，相同类型的类对象是通过拷贝构造函数来完成整个复制过程的。

字符串的容量：是值在不获取更多内存的情况下字符串所能存储的最大字符数。

迭代器：string类提供迭代其来正向和反向遍历字符串。迭代器提供类似指针运算的语法来访问单个字符。其中，const\_iterator是一个不能修改字符串内容的迭代器。begin函数有两个版本：一个返回用于迭代一个非常量字符串的迭代器，一个返回用于迭代一个常量字符串的const\_iterator

字符串流处理：除了标准流I/O和文件流I/O之外，C++流I/O还包括了在内存中从字符串输入以及输出到字符串的功能。

输出到一个字符串是利用C++流的，强大的格式化输出功能的方法。在一个字符串中，数据预先变成已编辑屏幕格式。可以将这个字符串写入磁盘以保存屏幕图像。ostringstream对象使用一个string对象来存储输出数据。类ostringstream的成员函数str返回字符串的副本。

istringstream对象从内存的字符串将数据输入到程序变量中。数据以字符的形式存储在istringstream对象中。从istringstream对象输出与文件中输出是相同的。在istringstream对象中字符串的结尾与文件尾结束符一样的。

指针

指针变量的声明和初始化：指针变量把内存地址作为它们的值。通常，变量直接包含特定的值。但是，**指针包含的是包含特定值的变量的内存地址，从这个意义上来将，变量名直接引用数组，而指针间接引用数值，因此，通过指针引用数值称为间接引用。**

**使用指针的按引用传递方式：**

1. 按值传递
2. 用引用参数按引用传递
3. 用指针参数按引用传递：**可以用指针和间接运算符（\*）来完成按引用传递**。当调用一个函数，需要修改实参时，就传递参数的地址。通常这是通过在值需要修改的变量的名字前应用地址运算符来实现。数组名就是数组在内存中的起始地址，所以不用运算符&传递数组。当一个变量的地址传递给函数时，**可以在函数中用间接运算符（\*）来构成该变量名的一个别名---它可以在调用者的内存位置修改该变量的值。**

使用const修饰指

1. 指向非常量数据的非常量指针
2. 指向常量数据的非常量指针
3. 指向非常量数据的常量指针
4. 指向常量数据的常量指针

**sizeof运算符**：sizeof运算符可应用于任何变量名、类型名或常量值。**当sizeof用于一个变量名或一个常量值时，返回存储该特定类型变量或常量所需要的字节数。**

**指针表达式和指针算术运算**：指针自增或自减，整数可以加到指针上，可以从指针减去整数，或者两个指针可以相减。**指向同一个数组的指针变量可以相减**。

如果两个指针是同一类型的，那么可以把一个指针赋给另一个指针。否则，要用**强制类型转换运算符**。**void指针是一般性的指针**，可以表示任何指针类型，所有的指针类型都可以赋值给void指针，而不需要强制类型转换。但是，v**oid指针不能直接赋值给其他类型的指针，需要强制类型转换**。

**void指针不能间接引用**，因为只包含一个未知数据类型的内存地址，编译器不知道指针所指向的确切字节数和该数据的类型。编译器必须知道特定指针的数据类型，才能确定该指针间接引用的字节数。

**指针和数组之间的联系**：一个数组名就是一个常量指针。

**指针数组**：数组可以包含指针。这种数据结构的一常见用法是构成基于指针的字符串的数组，简称为字符串数组。字符串数组中的每一项都是一个字符串。但是，在C++中，字符串本质上就是一个指向其第一个字符的指针，所以字符串数组中的每一项就是指向相应字符串第一个字符的指针。

**函数指针**：指向函数的指针包含内存中该函数的地址。函数名实际上是执行这个函数任务的代码在内存中的开始地址。

###### 类的深入剖析

类的作用域和类成员的访问：public类成员通过对象的句柄之一而引用。句柄可以是对象名称、对象的引用或者对象的指针。对象、引用或指针的类型指定了客户可访问的接口。

在成员函数定义的变量具有函数作用域，只有该函数知道它们。如果成员函数定义了与类作用域内变量同名的另一个变量，那么在函数作用域内，函数作用域中的变量将隐藏类作用域中的变量。这样所隐藏的变量可以通过在其名前加类名和二元作用域分辨运算符（：：）的方法而访问。同样，隐藏的全局变量可以用一元作用域分辨运算符来访问。

接口与实现的分离：程序易于修改，对类的客户来说，只要提供给客户原始接口保持不变，那么类实现的方法的改变就不会对客户造成影响。

访问函数：访问函数可以读取或者显示数据。访问函数另一个常用用法是测试条件是真还是加，常常称这样的函数称为判定函数。

工具函数：工具函数不属于类的public接口部分。更确切地说，它是支持类的public成员函数操作的private函数。工具函数也不供类的客户使用的。

默认实参的构造函数：一个默认所有实参的构造函数也是一个默认的构造函数，即一个调用时可以不带任何实参的构造函数。每个类最多只有一个默认构造函数。

析构函数：当对象撤销时，类的析构函数会隐式调用。析构函数并不释放对象占用的内存空间，它只是在系统回收对象的内存空间之前进行扫尾工作，这样内存可以重新用于保存新的对象。

析构函数不接受任何参数，也不返回任何值。析构函数不可以指定返回类型，甚至void也不行。一个类只能有一个析构函数，而且析构函数不允许重载。并且析构函数必须是public的。

何时调用构造函数和析构函数：全局作用域内定义的对象的构造函数，在文件内任何其他函数开始执行之前调用。当main函数执行结束时，相应的析构函数被调用，exit函数迫使程序立即结束，不执行自动对象的析构函数。

默认的逐个成员赋值：赋值运算符（=）可以将一个对象赋给另一个类型相同的对象。默认情况下，这样的赋值通过逐个成员赋值的方式进行，即赋值运算符右边的对象的每个数据成员逐一赋值给运算符左边对象中的同一数据成员。警告:当所用类的数据成员包含指向动态分配内存的指针时，逐个成员赋值可能会引发严重的问题。

对象可以作为函数的实参进行传递，也可以由函数返回。这种传递和返回默认情况下是以安置传递的方式执行。

const对象和const成员函数：对于const对象，C++编译器不允许进行成员函数的调用，除非成员函数本身也声明为const。

要将函数指定为const，则既要在原型中指定，又要在其定义中指定。试图将一构造函数和析构函数声明为const会变异错误。

使用成员初始化器来初始化const数据成员。所有的数据成员均可以用成员初始化器形式进行初始化，但是const数据成员和引用的数据成员以及成员对象必须使用成员初始化器进行初始化。

成员初始化器出现在构造函数的参数列表和作为构造函数体开始的左花括号之间。

组成：对象作为类的成员。

Friend函数和friend类：类的friend函数在类的作用域之外定义，却具有访问类的非public以及public成员的权限。单独的函数或整个类都可以被声明为另一个类的友元。在类定义中函数原型前加上保留字friend，就将函数声明为该类的友元。

友元关系是授予的而不是索取的。也就是说，对于类B成为类A的友元，类A必须显式地声明类B是它的友元。另外，友元关系既不是对称的也不是传递的。

使用this指针：每个对象都可以使用一个称为this的指针来访问自己的地址。对象this指针不是对象本身的一部分，也就是this指针占用的内存大小不会反映在对对象进行sizeof运算得到的结果中。相反，this指针作为一个隐式的参数传递给对象的每一个非static成员函数。

对象隐式地使用this或者显式地使用this指针来引用它们的数据成员和成员函数。this指针的类型取决于对象的类型以及使用this成员函数是否被声明为const。

使用this指针使用串联函数调用成为可能：每个成员函数都在其函数体最后一句返回\*this，返回类型要类名 &。

static类成员：static数据成员表示了整个类范围上的信息，也就是类的所有实例所共享的一个性质，而不是类的某个特定对象一个属性。

静态数据成员的作用域和初始化：static成员可以被声明为public、private或protected。基本类型的static数据成员默认情况下将初始化为0.如果希望初始化一个不同的值，static数据成员只可以被初始化一次。int或者枚举类型的congst的static数据成员在类定义中的声明处初始化。但是，所有其他static数据成员必须在文件作用域进行定义，并只可能呢写定义中初始化。

访问静态数据成员：类的private和protrected的static成员通常通过类的public成员函数或者类的友元访问。即使在没有任何类的对象存在时，类的static成员仍然存在。当没有类的对象存在是，要访问类的public static成员，只需简单的在此数据成员名前加类名和二元作用域分辨运算符（：：）即可。

静态数据成员的说明：如果成员函数不访问类的非static数据成员和非static成员函数，那么它应该声明为static。与非static成员函数不同的是static态成员函数不具有this指针，因为static数据成员和static成员函数独立于类的任何对象而存在的。

#### 运算符重载

定义：C++中运算符与类对象结合在一起使用，这个过程称为运算符重载。

运算符重载的基础知识：如果以成员函数方式重载运算符，那么这样的成员函数必须是非static的，因为它必须由该类的对象调用，并作用在这个对象。

所有的类对象可以用赋值运算符（=）、取值（&）以及逗号（，）。取值：返回对象在内存中地址。逗号运算符从其左侧的表达式开始求值至右侧表达式。

运算符重载限制：

不能重载的运算符：.、.\* 、::、 ?:

优先级、结合律和操作数个数：重载不能改变运算符的优先级；重载也不可能改变运算符的元数；不能创建新的运算符。

作为类成员函数和全局函数的运算符函数之比较

必须作为成员函数进行重载的运算符：（）、[]、 ->

作为成员函数和全局函数的运算符：当运算符函数作为成员函数实现时，最左边的操作数必须是运算符的一个类对象（或者对该类对象的一个引用）。如果左操作数必须是一个不同类的对象或者是一个基本类型对象，那么该运算符函数必须作为全局函数来实现。如果全局函数必须直接访问类的private或protected成员时，那么该函数可以指定为友元函数。

可交换的运算符：选择全局函数来重载运算符的另一个原因是使运算符具有可交换性。

动态内存管理：C++允许程序员在程序中对任何内置的或用户自定义的类型控制的内存进行分配和释放。这称为动态内存管理，由运算符new和delete完成。

1. 使用new来动态获取内存
2. 使用delete来动态获取内存
3. 使用delete动态释放内存
4. 动态内存初始化 new 类型名()
5. 使用new[]动态分配数组
6. 使用delete[]动态释放内存

重载一元运算符

1. 类的一元运算符可以重载成不带参数的非static成员函数，或者可以重载成带一个参数的全局函数。参数必须为类的对象或者该类对象的引用。
2. 实现重载的运算符成员函数是非static的，这样它们才能访问该类每个对象的非static数据

重载二元运算符：二元运算符可以重载成带一个参数的非static成员函数，或者可以重载成带两个参数的全局函数。

类型转换：

1. 编译器无法预先知道如何在用户自定义类型之间、用户自定义类型和基本类型之间进行转换，因此程序员必须指定如何进行这样的转换。上述转换可以类型构造函数：能把其他类型的对象转换成特定类对象单参数构造函数，来完成
2. 任何单参数构造函数都可以看成是转换构造函数。
3. 转换运算符可以将某个类对象转换成另一个类对象或者基本类型的对象。这种转换运算符必须是非static成员函数。通过定义重载的强制类型转换运算符函数，用户自定义类型的对象可以转换成基本类型，或者基本类型转换为用户自定义类型。
4. 重载强制类型转换运算符不指定任何返回类型，因为返回类型即是对象正要转换成的目标类型。
5. 重载类型转换运算符和转换构造函数的一个极佳特征是：在必要时，编译器可以隐式调用这些函数创建临时对象。

explicit构造函数：任何单参数构造函数都可以被编译器用来执行隐式转换，即构造函数接收的类型会转换成定义了该构造函数的类的定义。程序员不必使用强制类型转换运算符进行这种转换，它是自动进行的。

C++提供了关键子explicit，用于禁止不允许的由转换构造函数完成的隐式转换。

重载前置的自增运算符：前置的自增运算按引用返回类对象，这是因为在进行自增前，后置的自增运算符通常返回的一个包含对象原始值的临时对象。C++将这样的对象作为右值处理，使其不能用在赋值运算符的左侧。前置的自增运算符返回实际自增后的具有新值的对象。

重载后置的自增运算符：后置的自增运算符返回的是类对象

代理类：

1. 通过向类的客户提供只知道类的public接口的代理类，就可以使客户既能够使用类的服务，而又无法访问类的实现细节，即使是类的private数据。
2. 当类定义仅使用指向另一个类的对象的指针或引用时，另一个类的头文件（通常会泄露该类的private数据）不需要用#include包含进来。只需在文件中使用前面所述的另一个类之前，通过前置类的声明将其声明为一种数据类型。
3. 代理类的成员函数实现文件是唯一包含头文件的文件，头文件中包含要隐藏类的私有成员。
4. 代理类的成员函数实现文件以预编译后的目标代码文件的形式和头文件一起提供给客户，其中头文件包含代理类提供的服务的函数原型。

#### 函数和递归入门

函数原型：告诉编译器函数的名称、函数返回数据的类型、函数预期接收的形参个数以及形参的类型和顺序。

函数签名：函数原型的函数名和实参类型部分称为函数签名。函数签名不指定函数的返回类型

实参类型强制转换：函数原型的一个重要特性，把实参类型强制转换成函数原型中声明的形参类型。

存储类别：

1. 局部变量 auto
2. 寄存器变量 register
3. 静态存储类别 extern和static
4. 临界区：mutable

内联函数：减少函数调用的开销——特别是对于小函数。在函数定义中把限定符inline放在函数的返回类型的前面，会建议编译器在适当的地方生成函数代码的副本以避免函数调用。

一元作用域分辨运算符：有可能局部变量和全局变量声明成了相同的名字。C++提供一元作用域分辨运算符（：：），当作用域中的局部变量与全局变量具有相同的名字时返回全局变量。不能使用一元作用域分辨运算符访问外层语句块中具有相同名字的局部变量。

编译器如何区分重载的函数：重载的函数通过它们的签名来区分。签名有函数的名字和它的形参类型组成。编译器对每个函数的标识符利用它的参数数目和类型进行编码，以便呢能够实现类型安全的连接。类型安全的连接保证调用正确的重载函数并且实参类型与形参类型相符。

#### 标准模板库

**容器**：数据存放的地方，分为三类：序列容器，关联容器和容器适配器。

**vector序列容器**：提供了一种占用连续内存地址的数据结构。这使得它可以高效、直接地通过下标运算符[]访问vector中的任一元素。当一个vector的内存空间耗尽时，它会分配一个更大的连续空间，把原先的数据复制到新的空间，并把原空间释放。vector支持就随机访问迭代器。

**list序列容器**：list可以在容器的任一位置有效地进行插入删除操作。如果大部分的插入删除操作发生在容器的两端，那么deque数据结构提供了一个更加有效的实现。list类模板实现了一个双向的链表，这使得list类模板可以支持双向迭代其。

**deque序列容器：**deque类的实现提供了有效的索引访问（使用下标），可以读取或修改它的元素，这与vector类似。deque还提供了在它的首部和尾部进行高效插入和删除的操作的功能，这与list类似。deque类支持随机访问迭代器。deque最常用的功能是实现一个先进先出的队列。

**multiset关联容器**：快速存取关键字，并允许出现重复的关键字。multiset支持双向迭代器。

**set关联容器**：快速存取唯一的关键字。

**multimap关联容器**：用于快速存取关键字和相关值，存储对象为关键字/值对，而不是值本身。

**map关联容器**：用于快速存取唯一地关键字和关联的值。在map中关键字不能重复，所以每个关键字只能和一个值相关。

**stack适配器**：默认stack使用的是deque实现的。

**queue适配器**：默认情况下使用deque实现。

**priority\_queue适配器**：默认使用vector作为底层容器，通常使用堆排序的技术实现的。

**迭代器**：迭代器iterator提供了对顺序或关联容器类型中的每个元素进行连续访问的方法。

迭代器有它们所指的特定容器的状态信息，即迭代器对每种类型的容器都有一个实现。有些迭代器的操作在不同的容器间是统一的。它将容器和算法分开，好让这两者独立设计。