**C++ Primer Plus**

**目录**

[1. 预备知识 2](#_Toc105408129)

[2. 开始学习C++ 2](#_Toc105408130)

[3. 处理数据 2](#_Toc105408131)

[4. 复合类型 3](#_Toc105408132)

[5. 循环和关系表达式 4](#_Toc105408133)

[6. 分支语句和逻辑运算符 4](#_Toc105408134)

[7. 函数——C++的编程模块 5](#_Toc105408135)

[8. 函数探幽 5](#_Toc105408136)

[9. 内存模型和名称空间 6](#_Toc105408137)

[10. 对象和类 7](#_Toc105408138)

[11. 使用类 8](#_Toc105408139)

[12. 类和动态内存分配 9](#_Toc105408140)

[13. 类继承 9](#_Toc105408141)

[14. C++中的代码重用 10](#_Toc105408142)

[15. 友元、异常和其他 10](#_Toc105408143)

[16. string类和标准模板库 11](#_Toc105408144)

[17. 输入、输出和文件 11](#_Toc105408145)

[18. 探讨C++新标准 12](#_Toc105408146)

# 预备知识

发展历史，面向过程vs面向对象，编译过程

1. **C语言历史：**汇编直接操作硬件，但不可移植；希望能有有一种语言兼具低级语言的效率和硬件访问能力，同时还可移植，有通用性
2. C语言面向过程，自上而下；C++在C基础上添加面向对象以及支持泛型编程，自下而上，不仅能够提供低级硬件访问，还提供了高级抽象
3. **编程步骤：**源代码->编译器->目标代码->链接程序（启动代码+库代码）->可执行代码
4. Linux中最常用的编译器是g++
5. Build和make是递增过程，可以只重新编译修改的某个文件（不重新编译未修改的源代码文件）

# 开始学习C++

一般格式，输入输出注释，声明使用变量

1. 所有语句后面都要带分号
2. main函数被启动代码调用，启动代码由编译器添加到程序，连接程序和操作系统
3. 注释双斜杠开头//，或者/\* text \*/
4. 预处理器处理#开头的编译指令，将iostream文件内容取代 #include <iostream>
5. #define x 50 也是一个#开头的预处理器编译宏指令
6. C++文件名没有拓展名（math），c旧式风格（math.h），转化后的C（cmath）
7. 名称空间（namespace），有助于编译器选择同名函数是出自哪个版本（空间）。比如cout其实是std这个名称空间的对象，可以写成std::cout
8. 名称空间std包含的常用符号：endl，cin, cout等等
9. cin是istream类的一个实例，cout是osteam类的一个实例
10. cout<<”abc”将字符串插入到输出流中，cin>>var将输入流插入到变量中
11. 定义变量需要声明类型和名称，初始化可选
12. 如果还是函数定义在main之后，需要在开头声明函数原型，比如double sqrt(double)

# 处理数据

变量：整型、浮点型；常量，优先级、算术运算符，类型转换

1. 变量
2. 变量名只能用字母数字和下划线，不能以数字开头，区分大小写。命名时可用前缀来表示变量类型，比如nMyWright，n(整数)，b(bool)，p(指针)，c(字符)
3. C++基本整型：char, short, int, long, long long，每个都有有符号和无符号两种
4. 16位无符号表示范围0~65536，有符号表示范围-32768~32767
5. C++中整型如果超出了范围就会发生上溢或下溢，值变成另一端的取值
6. int表示范围在不同计算机上也不同(16,24,32位)，因此有时需要指定short, long
7. \用来转义，在字符串中输出特定字符
8. Bool类型，非零值为1，只有零值为0
9. 常量
10. 使用预处理器# define X 10或者const int X = 10; 后者更C++，同时能明确类型
11. 浮点数
12. 浮点数能够表示带小数的数字，12.34或2.52e+8
13. 浮点类型：float, double, long double
14. 浮点数默认为double类型，如果希望常量储存为特定类型，可以加后缀，如1.234f(float)，2.2L(long double)
15. Float只能表示数字中前6位或7位，会影响精度
16. C++运算符：加 减 乘 除 取模
17. 除法如果有一个是浮点数那么结果就为浮点数，否则执行整数除法，省略小数点
18. 类型转化：自动或强制，auto variable还可以让编译器推断变量类型

# 复合类型

数组、字符串、结构、共用体·、枚举、指针、动态内存、动态数组、vector和array。符合类型的意思是基于基本类型（整型、浮点型等）而来的类型。

1. 数组：在内存中依次储存多个同类型的值，int months[12]; 如果12是变量的话需要new来分配内存，从0开始编号，可以用花括号来初始化数组，初始化一部分的话其他元素设置为0。
2. C-style字符串：以空字符\0结尾，数组应该足够大能够存储包括空字符在内的所有字符，用cin.getline()不保存换行符和cin.get()来输入，**confused here** P81
3. cin只能读取一个单词，输入New York的话在空格处就会停止，这个时候应该换成cin.getline(object, 20)来读取整行，或者cin.get(object, 20)，区别就是cin.getline()会丢弃换行符，而cin.get()保留换行符。cin的话也是会保留换行符的，所以cin完了想再输入的话需要再次cin.get()来丢弃这个换行符。
4. C++字符串：用string类，需要#include<string>，将string类对象看成变量，自动处理大小，不需要\0结尾，可以直接用+运算，str.size()可以知道大小
5. 用string类的话想读取一行带空格的单词，比如New York，可以用getline(cin, object);
6. 结构：可以储存多种类型的数据，**与类有区别**，struct name {}; 定义一种新类型，用.访问成员，初始化也是花括号并用逗号分隔
7. 共用体：只能同时储存不同给定数据类型的其中一种，union name {}; 长度为最大成员的长度，常用于操作系统或硬件数据结构
8. 枚举：enum colors {red, orange, yellow, blue}; 定义新类型colors，colors类型的变量值只能从枚举定义中选取，比如colors band = blue; 常用来定义符号**常量**
9. 指针：是一个储存值的地址的变量，updates与\*p\_updates，&updates与p\_updates完全等价，声明int\* ptr指向int类型数据的指针。**指针指向地址起始位置，步长是由类型决定的**。指针必须先指向一个内存才能解除引用\*，不然会读取一个随机地址。指针可以在运行阶段分配未命名内存，int\* p = new int; 用delete释放指针指向的内存，而且只能释放new分配的内存，所以**new和delete要成对使用**。在运行阶段创建的数组称为动态数组，int \*p = new int [10];与delete [] p; 动态创建和释放数组。C++将数组名解释为地址，因此指针和数组基本等价，可以直接p[0]来访问元素。指针+1具体移动多少取决于类型大小。C++中相同字符串字面值可能储存在多个地址。用指针来访问结构成员用->而不是点。
10. 如果指针指向的数据对象是简单变量，那么用\*来访问；如果对象是数组，那么像数组名一样访问；如果对象是结构，那么用->来访问成员。
11. 如果name是数组名，那么name[i]就等价于\* (name+i)，其中数组名为第一个元素地址
12. Vector：是动态数组，自动用new和delete来管理内存，vector<int> name(10); 就可以创建一个动态数组。需要#include<vector>
13. Array：长度固定的数组，array<int, 10> name; 需要#include<array>

# 循环和关系表达式

For循环，表达式和语句，递增递减运算符，逗号运算符，关系运算符，while循环，typedef，do while循环，二维数组。

1. For循环：设置初始值、执行测试、执行循环、更新测试值，for (int i=0; i < 10; i++) {}，循环前先检查条件是否满足。循环体如果不加花括号的话就只有第一条语句会被循环
2. 语句int age; 表达式age = 100;
3. a++表示先用a的值运算再递增1；++a表示先递增1再用a的值运算
4. 逗号运算符：将两个表达式放在一个地方，比如++j, --i。比如cats=17,240;实际cats值为17。比如cats=(17,240); 实际cats赋值为240，因为括号优先级最高。
5. 关系表达式：< <= > >= == != 算术运算符优先级比关系运算符高。C-style字符串比较要用strcmp()，不能用==，字符串相等返回0.
6. While循环：也是入口条件循环。
7. Do while循环：出口条件循环，首先执行循环体。
8. 基于范围的for循环，对数组每个元素执行相同的操作，类似python。 for (auto x: prices) cout<<x<<endl;
9. 循环和文本输入，cin忽略空格和换行符，按下回车之后换行符还在输入缓存中。cin.get()读取输入的下一个字符，包括空格。cin可以作为bool判断条件，如果成功输入则返回True。cin,eof()可以判断是否到达文件尾。
10. 嵌套循环和二维数组：int temp[4][5]

# 分支语句和逻辑运算符

If、if else、switch、逻辑运算符、条件运算符、continue和break、文件的输入输出

1. If语句：If (condition1) {} else if (condition2) {} else {}
2. 逻辑表达式：或||，与&&，非!。与或优先级比关系运算符低，所以判定前或先处理左侧的运算，比如i=10，i++<6 || i==j 比较的是**11==j**。从左到右依次判定。不能用10<age<35，应该用10<age && age<35，因为编译器看到的是（10<age）<35，也就是将35与1或者0比较。非！优先级高于关系运算符和算术运算符，记得加括号。
3. 字符函数库cctype：常用函数isalnum(), isalpha(), isdigit(), islower(), isupper(), toupper(), tolower(), isspace()
4. 条件运算符：expression1 ? res1 : res2，判断条件，为真的会返回res1，为false的话返回res2。相当于简便版的if else
5. Switch语句：switch (expression) {case 1: actions1; case2: actions2; default: actions3; }。标签最好是整数值或者枚举量，跳**到某个case后会依次向下执行，所以要加上break才会只处理一个case**。
6. Break和continue：break是跳出循环，continue是跳过循环体剩下的开启新的循环。有goto语句但最好别用。
7. 读取数字：int n; 那么cin>>n如果不是数字的话会返回false
8. 文件输入输出：①包含头文件fstream ②用ofstream/ifstream类声明一个输出/输入对象 ③将对象与文件关联起来，用open，比如ofstream out; out.open(“fish.txt”) ④接下来就把cout/cin替换成对象就可以对文件进行操作了 ⑤用完了记得对象.close();

# 函数——C++的编程模块

函数原型，按值传递，处理数组/字符串/结构/string的函数，const指针参数，调用自身的函数（递归），指向函数的指针

1. 函数基础：定义函数（有无返回值），提供函数原型（将函数的返回值类型，参数的类型数量告诉编译器），调用函数
2. 按值传递参数：函数在被调用时创建一个新的变量来**复制**值。
3. 按地址传递参数：以地址作为参数可以节省复制数组所需的时间和内存。
4. 处理数组的函数：int sum\_arr (int arr[], int size); 将数组名作为参数传递，并没有传递数组内容，而是传递**数组的地址，元素类型和元素数目**给函数，函数修改会造成原数组的修改，因为是改的地址里的值。需要提供元素数目是因为传递过去的指针sizeof大小不同，函数里的指针只指向第一个元素，而原本的函数名指向整个数组。声明形参时使用const可以避免函数修改数组内容，int sum\_arr (**const** int arr[], int size); 函数将数组视为只读数据。
5. 处理二维数组的函数：int sum\_arr (int arr[][4], int size); 表示写死列数，接受任意行数的二维数组。
6. 处理字符串的函数：C-style字符串（char数组、引号字符串常量、char指针）。C-style字符串与char数组的区别是字符串有内置的结束字符/0，因此不需要将字符串长度作为参数。
7. 处理结构的函数：也可通过**值传递、地址传递、引用传递**来修改。
8. 处理string的函数：与结构类似
9. 处理array的函数：按值传递效率低，但按地址传递会更复杂
10. 递归：函数调用自己，分治，调用次数呈几何次数增长。
11. 函数指针：函数的地址就是函数名，不加参数即可。假如有函数double a (int); 那么可以声明指针double (\*p) (int)，只需要将原型中的函数名替换为(\*p)，也可以用**auto**让编译器自动推断。

# 函数探幽

内联函数、引用变量、按引用传递参数、默认参数、函数重载、函数模板

1. 内联函数：将编译代码与程序代码内联，程序无需调到另外的内存位置去执行代码，运行速度比常规函数稍快，但需要占用更多内存（10个不同地方调用将包含该函数的10个副本）。如果代码执行时间很短那么就可以用内联调用，省去跳到函数地址的开销。通常做法是省略原型，将整个定义放到最前面，**inline** int add(int a) {return a \*\* 2; }
2. C语言中用预处理器语句来提供宏，#define SQUARE(X) X\*X，但要注意没有括号的运算顺序问题，并且不能按值传递（只能传递地址，这样如果有++的话可能会出错）
3. 引用变量：主要用途是做函数的形参，让函数使用原始数据而不是副本。创建引用变量int &rodents = rats; int &表示指向int的引用，这样两个变量名指向相同的值和内存单元。**&不是地址运算符了而是将类型声明为int &**。声明引用变量时必须初始化。
4. 按引用传递：就是在函数形参里加上&，比如void swap(int &a, int &b); 与按指针传递效果类似，只是按指针传递声明的时候使用int \*a，并且要使用解引用运算符\* 。将引用用于结构/类，比起按值传递赋值原始结构可以节省时间和内存。
5. 继承：派生类继承基类的方法，基类引用可以指向派生类对象而无需进行类型转换。

文本

描述已自动生成

1. 默认参数：只在**函数原型**处进行赋值，不需要修改函数定义！
2. 函数重载：aka函数多态，可以使用多个**同名**的函数，用**特征标**来区分（函数的**参数**列表，主要是参数数目、类型、排列顺序），不区分const变量，返回类型不同也不可以，特征标只看参数。为了解决不同数据类型但相同的任务。
3. 函数模板：用泛型来定义数据类型，template<typename/class T>，就可以用T来代替任意数据类型。比重载更方便一些，但有些操作，比如<不适用于数组和结构。
4. 关键字decltype（declare type）：如果不知道某个变量会是什么类型，就可以int a; decltype(a) b; 将a的类型给b。decltype((a)) c，那么c的类型是int &。

# 内存模型和名称空间

单独编译，存储持续性，作用域和链接性，定位new运算符，名称空间

1. **单独编译**：在大型程序中，应该将各个组件函数放在不同的文件中，通常包含三个部分：①头文件（包含结构声明和函数原型）②源代码文件（包含函数定义）③源代码文件（包含main，调用和使用结构、函数的代码）
2. **头文件常包括内容**：函数原型，#define符号常量，结构声明，类声明，模板声明，内联函数。定义应该放到另一个文件，不然可能会对一个函数有多个定义！源代码中不要include其他源代码文件！
3. 头文件名如果包含在尖括号中，则一般是在标准头文件系统里找#include <iostream>；如果在双引号中，则一般是用户定义头文件，编译器首先查找当前工作目录，#include“test.h”。头文件管理：#ifndef ··· #endif
4. C++的存储方式通过存储持续性、作用域（变量名在文件中可见的范围，作用域解析运算符：：：）和链接性（描述名称如何在不同单元间共享）来描述的。
5. 存储持续性：（自动/静态/线程/动态）存储持续性。**自动存储持续性**作用域为局部，没有链接性，作用域被限制在代码块/函数内，新的定义会隐藏以前的定义，用栈管理。**静态持续变量**：都会被零初始化，分为外部链接性（可在其他文件中访问，代码块外声明，别的文件中要用extern声明），内部链接性（只在当前文件中访问，代码块外声明并用static限定符），无链接性（只在当前函数/代码块中访问，在代码块中声明并用static限定符，只在启动时初始化一次，可用于记录函数调用次数）。**动态存储方案**（用new和delete来分配动态内存，用堆管理，定位new运算符可以在你指定的空间进行动态储存，delete只能用于new分配的堆内存）
6. **存储说明符**：auto, register, static, extern, thread\_local, mutable, const, volatile
7. **名称空间**：为了避免名称冲突，用名称空间来控制名称的作用域，用作用域解析运算符来限定名称，比如myNamespace::cout。using编译指令会导入所有名称，using声明则只导入指定名称，当然也可以不用using，直接用作用域解析运算符std::cout的方式来使用。

# 对象和类

面向过程/对象编程，定义和实现类，公有/私有类访问，数据成员，成员函数，构造函数和析构函数，const成员函数，this指针，对象数组，类作用域，抽象数据类型ADT

1. **面向对象编程**（OOP）特性：抽象、封装、数据隐藏、多态、继承、代码可重用
2. **过程性编程**（按照流程构造步骤函数），**面向对象编程**（考虑数据的表示和使用）。
3. 类包括类声明和类方法定义。通常数据成员放在private中，成员函数放在public中
4. **类与结构的区别**：结构的默认访问类型是public，类是private
5. **抽象：**人们可以通过类方法的共有结构来对类对象执行操作。
6. **封装**：通常将定义放在头文件中，将实现方法放在源代码cpp文件中，使用者不需要知道如何实现，只需调用声明，将公有接口和实现细节分开从而实现**封装**，维护程序只需要改函数不需要改接口，更方便。
7. **数据隐藏**：对象的私有成员只能通过公有成员函数（以及友元函数）来访问，不能用对象.数据来访问，和python不同。默认是private，public需要声明。
8. **类成员函数**：可以用来访问private数据，定义成员函数需要使用作用域解析运算符(::) 来标识函数所属的类。Void Stock::update(double price)。定义位于类声明中的函数都为内联函数。每个对象都有自己的存储空间但共享同一组类方法。
9. **构造函数**：因为数据部分都是私有的，因此需要成员函数来进行初始化。构造函数名字与类名相同，没有返回类型，Stock类构造函数就是Stock(string name, long n = 0); 创建对象时会自动调用构造函数。可以显式调用Stock a = Stock(“XianShi”，20)或者隐式调用Stock b (“YinShi”,30)。已经创建的对象无法调用构造函数。构造函数可以重载来实现默认构造函数，要么都提供默认值，要么都不提供。**默认构造函数**：没有参数或者所有参数都有默认值的构造函数。
10. **析构函数**：对象到期时自动调用，如果是new分配的内存那么用delete来释放内存。析构函数名称就是~Stock()，没有参数没有返回类型和返回值，在类名前加上~。
11. **const成员函数**：如果要确保**对象**不被修改，那么在类中将函数声明为void show() const，在成员函数**后**加const，确保类方法不修改调用对象。
12. **this指针**：如果对象a的成员函数调用了对象b，那么this指针就指向当前对象a（隐式指向调用成员函数的对象，设置为调用它的对象的地址），\*this就是调用对象的别名。
13. **对象数组**：创建同一类的多个对象，与标准类型相同，Stock temp[4]; 初始化必须对每个元素都调用构造函数，如果只初始化一部分，剩下的就是用默认构造函数。
14. **类作用域**：类中定义的名称（类数据成员名和类成员函数名）作用域是类内，都只在类中已知，必须通过对象调用共有成员函数。
15. **抽象数据类型**：以通用的方式描述数据类型，不引入语言或者细节，比如栈、队列等。

# 使用类

运算符重载、友元函数、重载<<运算符、状态成员、类的自动/强制类型转换、rand()、类转换函数

1. **运算符重载**：是多态的一种。比如将时间相加可以重载+：Time operator+(const Time &t) const; 按照引用传递另一个时间t节省时间和内存，返回的是一个新Time对象。重载限制：必须至少有一个操作数是用户定义的类型，不能违反运算符原来的规则，不能创建新的运算符，不能重载某些运算符。
2. **友元**：包括友元函数/类/成员函数，提供除公有类方法外对类私有部成员的访问，访问权限与成员函数相同。如果是成员函数的话（比如运算符重载），两个数相乘只能对象在前，因为要转化成A=B\*2.75=B.operator\*(2.75)的函数调用形式，反过来则不行。因此可以将运算符重载变成非成员函数的形式，但要变成友元函数才能访问类的对象的私有数据。
3. **友元函数创建**：只在原型声明前加上friend，编写定义时不要使用类的限定符，因为不是成员函数，也不需在定义中使用friend，Time operator\*(double m, const Time&t)。对于加法运算符，如果作为成员函数，那么this指代本操作数，只需要将另一个操作数作为函数参数传递；如果作为友元非成员函数版本，就必须将两个操作数都作为参数显式传递。
4. **重载<<运算符**：参数中需要有ostream类对象，并且需要使用友元函数来访问想要输出类的私有数据，friend ostream& operator<<(ostream &os, const Time&t) {return os;}，返回的是ostream的引用，也就是说可以cout<<a<<b连用了。
5. **类的自动类型转换**：只接受一个参数的构造函数可用来做自动类型转化，可用关键字explicit来关闭自动特性，但仍允许显式转换。比如有Stone (double lbs)的构造函数，那么Stone obj; obj = 19.6是自动类型转换，obj=Stone(19.6)是显示类型转换。如果想要将Stone对象转换成double，那么需要定义**转换函数**，operator typeName(); 比如operator double() {return pounds;}; 不能有返回类型和参数，必须是类方法。

# 类和动态内存分配

动态内存分配、显/隐式赋值构造函数、显/隐式重载赋值运算符、构造函数中new、静态类成员、定位new运算符用于对象、指向对象的指针、队列ADT

1. **动态内存**：string类可以处理内存管理细节，但如何在构造函数用new在运行时动态分配内存呢？必须使用析构函数来指向对象成员指针指向的内存。假如在私有数据定义一个char指针但不分配内存，构造函数的字符串并不保存在对象中，对象仅有指向字符串的指针，没有创建副本。
2. **静态类成员**：使用static创建，类的所有对象共享一个静态成员，比如记录类所创建的对象数目。静态类成员不是对象组成部分，不能在声明中初始化，要在类声明外单独初始化。不能通过对象调用，只能用类名和作用域解析运算符来调用。
3. **特殊成员函数**：C++自动给类提供的成员函数：默认构造函数（要么没有参数、要么参数都有默认值）、默认析构函数、复制构造函数（将对象初始化为同类对象、按值传递给函数、按值返回对象、生成临时对象）、赋值运算符（赋值不是初始化）、地址运算符
4. **复制构造函数**：将一个对象复制到新创建的对象中，用于**初始化**（包括按值传递参数），原型Class\_name (const Class\_name &)。每当程序生成了对象副本/临时对象，都将使用复制构造函数。默认复制构造函数**浅复制**成员，只复制成员值，因此如果是字符串的话只复制指针，析构函数将释放指针指向的内存，别的对象就会出错！解决办法：自己定义复制构造函数，进行深复制，创建字符串的副本给新对象。
5. **赋值运算符**：允许类对象赋值是因为C++自动为类重载赋值运算符，函数原型：Class\_name & Class\_name::operator=(const Class\_name &)，返回指向对象的引用。与复制构造函数，也需要提供（深度复制）的定义。重载了之后就可以直接使用常规字符串，不用创建和删除临时对象，比复制构造函数提高效率。
6. **复制构造函数vs赋值运算符**：应定义复制构造函数，通过深度复制来用对象初始化另一个对象；应定义赋值运算符，通过深度复制将一个对象复制给另一个对象。
7. **构造函数new**：构造函数使用new初始化指针，应该在析构函数中用delete。new对应delete，new []对应delete []。如果有多个构造函数，那么必须使用相同的new，要么都带中括号要么都不带。
8. **返回对象**：可以返回指向对象的引用、指向对象的const引用，或const对象。
9. **类初始化**：C++11允许在类定义中进行初始化，也可以通过括号来初始化。

# 类继承

Is-a关系继承、公有派生类、保护访问、构造函数成员初始化列表、强制转换、虚成员函数、静态/动态联编、抽象基类、纯虚函数

1. **继承**：类库提供了声明和实现，从基类派生出新类可以方便地在已有类的基础上添加新功能、添加数据、修改类方法，并且不需要访问源代码。不继承构造函数、析构函数和赋值运算符。
2. **公有派生类**：class child: public parent {}，is-a关系。基类的公有成员被继承，私有部分只能通过基类的公有和保护方法访问。派生类需要自己的构造函数，并且必须使用基类构造函数（需要初始化基类私有数据）。基类指针可以直接指向派生类对象，只能调用基类方法。
3. **派生类构造函数**：创建基类对象->调用基类构造函数->初始化派生类新增数据。成员初始化列表将值传递给基类构造函数Child::Child(int x, int y) : Parent(x, y) {}
4. **多态公有继承**：希望同一个方法在不同类中行为不同，取决于对象。包括重新定义基类方法和使用虚方法。
5. **重新定义继承方法**：并不是重载，而是覆盖基类声明，隐藏同名的基类方法。
6. **虚方法**：使用关键字virtual，程序将根据引用或指针**指向的对象**的类型来选择方法；否则的话，就是根据引用或指针**的类型**来选择方法。关键字virtual只用于类声明的方法原型中。可以定义一个基类类型的指针数组，对于virtual方法，它将根据指向对象的类型来确定调用基类还是子类的方法，必须定义虚析构函数，否则只调用指针类型的析构函数。应该仅将那些预期被重新定义的方法声明为虚的。构造函数和友元函数不能是虚方法，析构函数应该是虚方法。
7. **动态联编**：虚函数使得到底是用哪个函数变得不确定，因此无法在编译时决定。
8. **访问控制protected：**protected中的成员，对于类外相当于private，对于派生类相当于public，派生类成员可以直接访问保护成员。
9. **抽象基类abstract base class（ABC）：**只定义接口，而不涉及实现，如Shape。定义纯虚函数，如virtual double Area() const=0; 结尾处为=0. 类中包含纯虚函数，则不能创建该类的对象，只能用作基类。ABC的派生类应覆盖其纯虚函数。

# C++中的代码重用

has-a关系、包含对象成员的类、模板类valarray、私有和保护继承、多重继承、虚基类、创建使用类模板

1. **包含其他对象成员的类：** has-a关系，**包含**关系，可以直接使用对象成员的方法a.b.func()
2. **私有继承：**private，基类的公有和保护成员都为派生类的私有成员。与包含相比，提供了无名称的子对象成员，使用类名来标识，访问也只能通过作用域解析运算符来访问。不能将指向派生类的引用或指针赋给基类。包含is better.
3. **保护继承：**protected，基类的公有和保护成员都为派生类的保护成员。公有继承则不改变，公有变公有，保护变保护。
4. **多重继承MI：**使用多个基类的继承。主要问题是不同基类的同名方法以及继承同一个类的多个实例。
5. **虚基类：**解决多重继承时出现多个基类对象的问题，使得多个基类相同的类派生出的对象只继承一个基类对象，在类声明中使用virtual，class Child: virtual public Parent.用类限定符来区分同名成员。
6. **类模板：**用template <typename/class Type>，Type为通用类型，尖括号中的内容相当于函数的参数列表。为容器类提供可重用代码，避免因变量类型重写。模板具体化需要带类型，class IC<short> sic;

# 友元、异常和其他

友元类、友元类方法、嵌套类、异常、try-catch块、异常类、运行阶段类型识别RTTI、typeid

1. **友元类/成员函数：**友元类的所有方法都可以访问原始类的私有成员和保护成员。在类中任意位置进行友元声明friend class Parent; / friend void Parent::show(); 即可将Parent类作为友元类。
2. **嵌套类：**在类中声明的类被称为嵌套类，嵌套类仅在该类中有效，相当于定义了一种类型。
3. **异常：**throw引发异常-catch捕获处理异常-使用try块包含可能引起异常的代码。可以用多个catch来处理不同的异常。throw语句类似返回，但将控制权一直返回给包含try的函数，而不仅仅返回给调用它的上一级函数。省略号...可以表示任意异常类似default。
4. **运行阶段类型识别RTTI：**让程序能够检测对象的类型，只适用于包含虚函数的类，三个元素：dynamic\_cast运算符（将派生类转换为基类）、typeid运算符和type\_info类（比较对象是否为特定类型）
5. **类型转换运算符：**dynamic\_cast / const\_cast / static\_cast / reinterpret\_cast

# string类和标准模板库

标准string类、标准模板库STL、智能指针、容器类、迭代器、函数对象、STL算法

1. **string类：**包含方法（9种构造函数、给字符串赋值、合并字符串、比较字符串、访问各个元素、查找字符和子字符串、删除部分内容、替换部分内容、插入、交换）
2. **智能指针模板类：**行为类似指针的类对象，但当智能指针过期时会自动调用析构函数使用delete来释放内存，无需手动释放。auto\_ptr（尽量别用，会有所有权问题）、unique\_ptr（建立所有权，同一个对象只能被一个智能指针指向）、shared\_ptr（允许多个指向同一个对象的指针）
3. **标准模板库STL：**提供了一组容器、迭代器、函数对象和算法的模板。
4. **STL容器：**STL容器是同质的，存储的值类型相同，包括数组队列链表。包括了一些基本方法，比如size() / swap() / begin() / end()等
5. **非成员函数：**适用于所有容器类的非成员函数，比如find()，for\_each()，eandom\_shuffle()，sort()等，for\_each(books.begin(), books.end(), Show) 等价于 for (auto x: books) Show(x)
6. **泛型编程：**关注的是算法。模板使算法独立于数据类型，迭代器使算法独立于容器类型。
7. **迭代器类型：**STL定义五种迭代器：输入/输出/正向/双向/随机访问迭代器
8. **容器种类：**原来的11种（deque、list、queue、priority\_queue、stack、vector、map、multimap、set、multiset、bitset）以及C++11新增的5种（forward\_list、unordered\_map、unordered\_multimap、unordered\_set、unordered\_multiset）。其中map、multimap、set、multiset为关联容器，提供元素的快速访问。
9. **函数对象：**也叫函数符，可以以函数方式与（）结合使用的任意对象。plus<>相当于+
10. **算法：**头文件algorithm（非修改式序列操作、修改式序列操作、排序和相关操作），头文件numeric（通用数字运算）
11. **其他库：**complex、valarray、initializer\_list

# 输入、输出和文件

# 探讨C++新标准

移动语义和右值引用、Lambda表达式、包装器模板、可变参数模板