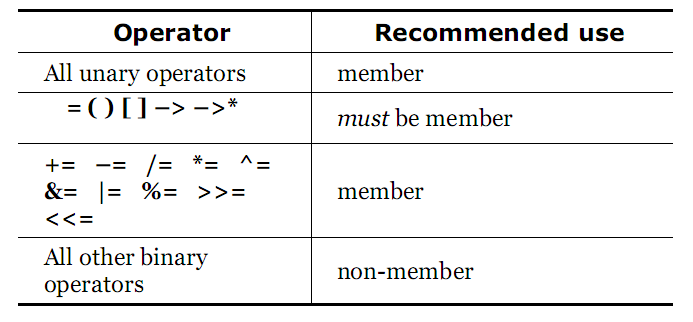
*判断对错题，简答题 林晨 17302010021*

***1章介绍对象*①面向对象：基于类型的分解如果一个系统在演化过程中需要修改时，基本上可以通过增加或者修改类来实现，那么这样的系统就适合于采用面向对象的方式来实现。面向过程：基于功能的分解  
如果大部分的演化可以通过增加或者修改功能来实现，那么这样的系统就适合于采用面向过程的方法来实现。②抽象：Abstraction is the process or result of generalization by reducing the information content of a concept or an observable phenomenon, typically to retain only information which is relevant for a particular purpose.程序设计语言的演化过程是抽象层次不断提高的过程。纯的面向对象的**程序设计语言不能够保证写出面向对象风格的程序，需要配合相应的各种方法**。p12.③绑定：**函数调用通常被编译成指针跳转到函数的开始位置，这是在编译和连接是就确定下来的。这被称为early binding。然而OOP的编译器多态采用了late binding的技术，函数调用是在运行时才确定的。**④C++的优点**：一个较好的C和较快的Java；延续式的学习过程；效率；系统更容易表达和理解；有成熟的类库；可以利用模板实现源代码的重用；支持异常处理

***2章介绍C++*①java与C++运行代码过程：java source-javaC-class file-jvm。而C+：C++ source-prossor-c++ source-complier-object file-Linker-excutable- OS Exe loader.②编译指令：Java：Javac。C++：预处理：g++ -E main.cpp –stdlib=libstdc++ 汇编：g++ -S main.cpp –stdlib=libstdc++编译：g++ -c main.cpp –stdlib=libstdc++预处理,编译,连接g++ main.cpp –stdlib=libstdc++。目标文件与模板：连接器把由编译产生的目标模块（一般是带”.o”或”.obj”扩展名的文件）连接成为操作系统可以加载和执行的程序。③示例：C文件复制：FILE \*in\_file, \*out\_file;char data[BUF\_SIZE];size\_t bytes\_in, bytes\_out;in\_file = fopen(argv[1], "rb"); out\_file = fopen(argv[2], "wb");   while ( (bytes\_in = fread(data, 1, BUF\_SIZE, in\_file)) > 0 )    {        bytes\_out = fwrite(data, 1, bytes\_in, out\_file);        if ( bytes\_in != bytes\_out )    { perror("Fatal write error.\n");  
            return 1;        }    }fclose(in\_file);fclose(out\_file);④三种包含头文件方式的区别。⑤#error命令是C/C++语言的预处理命令之一，当预处理器预处理到#error命令时将停止编译并输出用户自定义的错误消息。**

***3章C*①基本数据类型修饰符(Specifiers)Long／Short修饰int 和double Sighed/Unsigned②int dog, cat, bird, fish; void f(int pet) { cout << "pet id number: " << pet << endl;} int main() { int i, j, k; cout << "f(): " << (long)&f << endl; cout << "dog: " << (long)&dog << endl; cout << "cat: " << (long)&cat << endl; cout << "bird: " << (long)&bird << endl; cout << "fish: " << (long)&fish << endl; cout << "i: " << (long)&i << endl; cout << "j: " << (long)&j << endl cout << "k: " << (long)&k << endl; } ///:输出~f(): 4199374 dog: 4485128 cat: 4485132bird: 4485136fish: 4485140i: 2293540j: 2293536 k: 2293532。③指针与引用（结合java）：指针和引用的最大区别是，引用不能够像指针一样运算，也就是说他不能够随意指向任意的内存地址。④下面代码定义了一个指向无参无返回值的函数：函数指针 void (\*funcPtr)();⑤指向函数的指针数组：一个更为有趣的结构是指向函数的指针数组(FunctionTable.c)：// A macro to define dummy functions: #define DF(N) void N() { \ cout << "function " #N " called..." << endl; } DF(a); DF(b); DF(c); DF(d); DF(e); DF(f); DF(g); void (\*func\_table[])() = { a, b, c, d, e, f, g }; int main() { while(1) {**

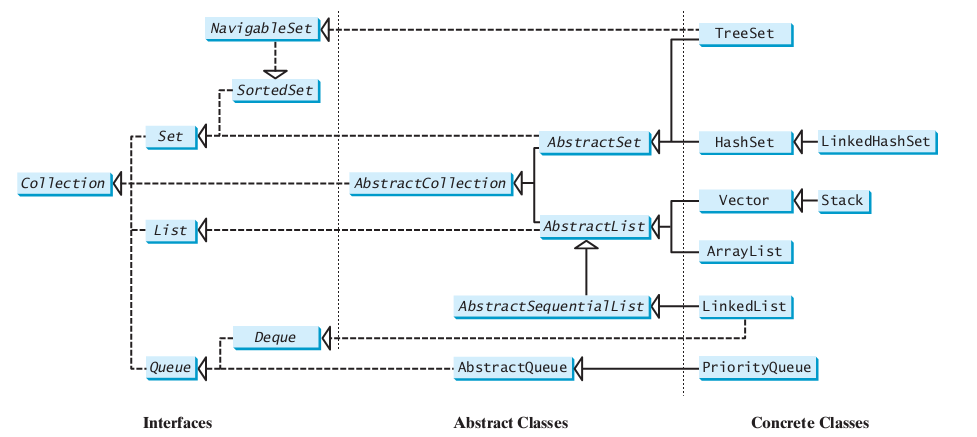
**cout << "press a key from 'a' to 'g' " "or q to quit" << endl; char c, cr; cin.get(c); cin.get(cr); // second one for CR if ( c == 'q' ) break; // ... out of while(1) if ( c < 'a' || c > 'g' ) continue; (\*func\_table[c - 'a'])(); }} ///:~⑥typedef：用于给类型定义别名（而不是定义新的类型） typedef existing-type-description alias-name 例如： typedef int\* IntPtr; IntPtr x,y; typedef典型的用途是用在结构(struct)的定义中⑦可见性。java和c++见下面表。**



***12章①***C++中，可以为类定义运算符，称为**运算符重载**。恰当地使用运算符，可以简化代码的书写，改善代码的可读性。**滥用**：运算符重载确实是一个有用的工具，但他仅仅只是一种语法上的方便，是函数调用的另一种方式。只有在能够使得代码更容易书写和阅读时，才考虑重载运算符。一个好的重载方案能够促进代码的书写和理解。**语法**：定义重载的运算符实际上是定义函数，只是该函数的名字是operator@，其中， @代表了被重载的运算符。**函数参数表中的参数的个数取决于两个因素**：①运算符是一元的（一个参数）还是二元的（两个参数）。②运算符被定义为全局函数（参数同上）还是成员函数（一元没有参数，二元一个参数）**C++中的大多数运算符都可以重载，但重载时有以下的限制：**不能使用C++中没有定义的运算符，比如试图将” \*\*”定义为求幂。不能改变运算符的优先级。不能改变运算符参数的个数。**这些约束的目的是**，尽可能保证运算符的语义，否则运算符将令人感到迷惑。**可重载的一元运算符包括**：**没有副作用的运算符**：+ , - , & , ! 。**有副作用的运算符**：++(前缀和后缀), --(前缀和后缀)**参数和返回的形式：**传(常数)引用；返回(常数)值；返回(常数)引用。**示例中需要注意的内容**：①有副作用的运算符与无副作用的运算符的区别。②返回原值和返回新值的区别。③成员运算符和非成员运算符的区别。④前缀运算符和后缀运算符的区别。**可重载的二元运算符包括：**创建了经修改的新值+, -, \*, /, % ,^, &, |, <<, >>修改左值的值+=, -=, \*=, /=, %=, ^=, &=, |=, >>=, <<=返回真/假的条件表达式==, !=, <, >, <=, >=, &&, ||。**Java中的对象相等和C++中的对象相等：**java写函数，c++重载==操作符。对于任何函数参数，如果仅需要从参数中读而不需要改变，默认地应当作为**const引用来传递**。如果需要改变，则应该作为引用来传递。**返回值的类型取决于运算符的具体含义**。如果使用该运算符的结果是产生一个新值，就需要产生一个作为返回值的新对象，这个对象通过传值的方式返回。否则返回引用。**赋值运算**总是会改变左值。为了使得赋值结果能够用于链式表达式，如(a=b=c),应该返回一个刚刚改变了的左值的引用。对于**逻辑运算符**，返回一个int或bool类型的值。**建议以常量方式返回，**例如在二元操作符+的定义中，返回值被声明为常量，通过这种方式可以限制如下形式的没有意义的调用：(a+b).g()返回值优化。**对于return Integer(left.i + right.i)，编译器将进行优化处理**，直接将值创建在外部返回值的内存单元。**特殊操作符**：**operator ,**（逗号操作符）当逗号出现在一个对象的左右，而该对象的类型是逗号操作符所支持的类型时，将调用逗号操作符。**Operator ->**当希望一个对象表现得像一个指针时，就需要用到该操作符，有时又称为smart pointer。该操作符有一个限制：它必须返回一个指针或一个定义了 ->运算符的对象。**下标运算符[]**必须是成员函数并且只能接受一个参数，通常将返回值定义为引用类型这样就能写如下形式的代码：Course OOP; courses[‘oop’] = OOP;**类型转换：**在任何时候，构造一个对象的同时，如果使用一个”=”来初始化一个对象，无论等号右侧是什么，编译器都会寻找一个接受右边参数类型的构造函数对左边的值进行构造。**自动创建operator=**。如果用户没有定义operator=，编译器将自动创建一个。这个操作符的行为与自动创建的拷贝构造函数的行为类似。**自动类型转换：**如果编译器看到一个表达式或者函数使用了一个不合适的类型，在可能的情况下（不丢失精度）将会执行一个自动类型转换。如果希望重新定义类型转换的方式，可以定义特定的构造函数，或者重载运算符。**由于有多种类型转换的方式，可能存在冲突**C12:TypeConversionAmbiguity.cpp存在多个类到同一种类型的转换时，发生冲突C12:TypeConversionFanout.cpp 。在前面的例子中，有些是成员运算符，有些是非成员运算符。应该如何选择？**原则：如果没有什么差异，尽可能采用成员运算符。**如果运算符的两侧可能是不同类型的对象，考虑到扩充的可能性，该运算符应该定义为非成员运算符。在可用的运算符集合里存在一些不能重载的运算符，这样的限制通常是**出于安全的考虑：如果这些运算符被重载，将会造成危害或破坏机制，**使得事情变得困难或混淆现有的习惯。 成员选择操作符：operator . 成员指针间接引用：operator .\*不能自定义运算符。不能改变运算符的优先级规则。**小结：**C++中，可以为类定义一个运算符，称为**运算符重载**。恰当地使用运算符，可以简化代码的书写，改善代码的可读性。**运算符重载语**法：①参数个数、类型，返回值类型，成员运算符和非成员运算符；②一些需特别注意的操作符重载③有无副作用④前后缀表达式⑤赋值运算⑥逻辑等于⑦自动类型转换。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| c++ Modifier | Class | Package | Subclass | World |
| public | Y | N/A | Y | Y |
| protected | Y | N/A | Y | N |
| *no modifier* | Y | N/A | Y | Y |
| private | Y | N/A | N | N |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Modifier | Class | Package | Subclass | World |
| public | Y | Y | Y | Y |
| protected | Y | Y | Y | N |
| *no modifier* | Y | Y | N | N |
| private | Y | N | N | N |



|  |  |  |
| --- | --- | --- |
| 02406acc916e4ab53796c572d1fd0ce | 6d75164a3e6833ccf2ddd53af53b1b6 | 6dfe8146b39967a9ccb353706c82120 |
| 38542eed8e6c2084ed22107e5d65f70 | bb59ad903ae90508aaa84daa810b18c | bc36efd6d838ac975238a2da9a8eeca |
| f8738b793c2e34d745e14cc162c8d6d | 9e155d775490a083278ed5e03f4aa70 | 75b0489c2a3ade0ed6ceea64c80508c |
| bbee4abb59117979cfe47023d807ab6 | d88447092315bcfce5266d31b8a338c | 281f195f57b471b62b562354731a340 |
| 0192af1e3ec7bf5d2fff7239169c0de | 02952db60174f83cffd112e90de2b04 | 1194551c949c1e24154414f605e1319 |
| 9bd5fbd9d91de688cb731f7b61b8f72 | 85e4eb934cafd7d21d9b8c4aa963b3c | d173bf542f743235114eeaae5fcba85 |
| ac680877ad797de56adba4a736af2c1 | a1b4d0301ebf78d2532d07ea5d7134c | be8d9fcf6f3d4c82ef5d42a7a40fc4e |
| 032b627d7d66040a1c2fc5167f7753c | 5665a36e9070bcbffdbaba1c97d8f69 |  |