1. 类定义后面带分号（改错会考）
2. 类：成员变量，成员函数，公有成员public，私有成员private（默认权限），保护成员protected
3. 成员变量是数据成员，可以是普通变量，数组，指针，结构，类对象（对象数组，对象指针）
4. 成员函数可以在类定义中，也可以在类定义外（类内有原型+<类名>：：限定）
5. 凡是在类体中定义的函数成员均隐含为内联函数
6. 私有成员不能从外部访问（类的定义外，类的成员函数定义外）
7. 公有成员可以从外部访问
8. 保护成员涉及派生类对基类成员的访问权限
9. 友元friend不是本类的成员，但有本类成员函数的权利，可以访问私有成员
10. 访问限定符Public，private，protected，friend顺序可以任意/多次
11. 紧跟左花括号的第一个说明段为private可省略，默认private/系统默认private
12. Private/protected体现了类的封装性
13. 类成员变量说明时可以给出默认值，若未初始化，可以使用默认值
14. 类的成员函数定义：直接在类的定义内（}后分号可加可不加）或类外定义（<类名>::）
15. 类是一种数据类型，定义类时系统不为类分配存储空间，类中任何数据成员也不能用extern限定其存储类型
16. 类的封装性：类把数据/函数封装为一个整体，成员函数可以直接使用类定义中任何成员，可以处理数据成员，可以调用函数成员
17. 类可以看做自定义的数据类型（int），类的对象就是类的变量，对象是类的实例，类只是模版
18. 类对象的说明：普通对象/对象数组【】/指针\*/引用&
19. 同类型的类可以相互赋值
20. 类的对象可以做函数形参，对象指针做函数参数，函数返回值可以是对象/对象指针
21. 在一个类中成员可以是该类的对象，也可是其他类的对象
22. 在类说明中定义函数，系统为每一个对象分配了全套的内存。数据区安放成员数据，代码区安放成员函数。
23. 在类说明外部定义函数,为每个对象分配一个数据区，代码区（放成员函数的区域）为各对象类共用。
24. 普通类对象用“.”，指针对象用“->”
25. This指针
26. 指向成员的指针<类型名><类名>::\*<指针变量名>
27. 构造函数：默认构造函数，自定义构造函数，重载构造函数
28. 如果类的数据成员是公有的，对象的初始化与一般变量没有区别，类似结构
29. 直接初始化：初始化列表的方法
30. 使用构造函数：默认构造函数，自定义构造函数，重载构造函数
31. 自定义构造函数：类内说明，类定义内或外定义，函数名和返回值类型与默认构造函数相同，可以有参数，也可以无参数，赋初值/参数列表化
32. 默认构造函数：<类名>(){}不在类定义内说明，也不定义，类定义中部包含显式构造函数时，由系统自动给出，并在说明类对象时自动调用
33. 自定义默认构造函数：类中包含自定义构造函数时，此时默认构造函数就失效，若仍想让对象被默认构造，采用（eg address类），address(){}/address()=default
34. 构造函数在初始化类对象时：有自定义构造函数就自定义，无则默认构造函数
35. 构造函数在初始化类对象时：1.<类名><对象名>，无构造函数->默认构造函数，有无参构造函数->默认构造函数失效，无参构造函数。有含参构造函数（参数值已给出）->该函数，不能同时重现默认含参构造函数，无参自定义构造函数
36. 对象数组的初始化：Poker poker[2]={Poker(11,a),Poker(10,b)}; //Poker \*poker=new Poker(11,a)
37. C++标准支持带有一个参数的构造函数，将参数类型隐式转换为相应的类类型，若需要去除隐式转换机制，在构造函数说明时使用explicit关键字即可
38. 委托构造函数：类内包含多个函数，一个构造函数在初始化列表中调用另一个构造函数，将构造工作委托给另一个构造函数。Box(int a,int b,int c):......;Box(int size):Box(size,size,size){}
39. 析构函数：在撤销对象占用的内存前完成的一些清理工作，~<类名>，一个类只有一个析构函数，可以缺省，但没有重载的析构函数，系统默认的构造函数~<类名>(){}
40. 构造函数与析构函数的执行顺序：先构造的后析构，后构造的先析构
41. 拷贝构造函数：特殊的构造函数，只有一个形参<类名>(<类名>&)，作用是用一个已存在的对象对初始化另一个正在创建的对象，类对象间的一般赋值，有拷贝构造函数实现（通过深拷贝或浅拷贝）
42. 系统自动生成的拷贝函数：浅拷贝，使用下列形式的说明语句<类名> <对象名2>(<对象名1>);<类名> <对象名2> = <对象名1>;对象作为函数的赋值参数，函数的返回值为类的对象

当程序出现上述三类情况时，自动调用拷贝构造函数

1. 显式拷贝构造函数：深拷贝，但自定义拷贝构造函数无法解决42中三种情况之外的度下个赋值，比如=赋值语句
2. 常对象：const <类名><常对象名>(<实参表>)任何成员变量都不能被修改，可以读取 成员变量的值，不能调用任何成员函数（常量函数成员除外）
3. 类的常量成员：const 修饰的类成员说明，常量数据成员，常量函数成员
4. 类的常量数据成员必须进行初始化，而且只能通过构造函数的成员初始化列表的方式来进行，在对象被创建以后，常量数据成员的值就不允许被修改
5. 常量函数成员：<类型说明符><函数名>(<参数表>)const,常函数可以调用常量函数成员
6. 类的静态成员：static修饰的类成员说明，静态数据成员，静态函数成员，类的静态成员为所有对象共享，静态成员只有一份存于公用内存中。
7. 必须在类外文件作用域的某个静态数据成员赋初值：<类型><类名>::<静态数据成员>=<初值>，访问静态数据成员，<类名>::<静态数据成员名>，<对象名>.<静态数据成员名>，<对象指针>-><静态数据成员名>
8. 类的静态函数成员没有this指针，从而无法处理不同调用者对象各自数据成员值，通常情况下，类的静态函数只处理静态数据成员<类名>::<静态函数成员调用>，<对象名>.<静态函数成员调用>，<对象指针>-><静态函数成员调用>
9. 友元：能够访问类的任何数据成员，友元函数，友元类
10. 友元函数：普通函数，定义在类外，friend void fun()，其他类的成员函数，friend void B::Fun()
11. 友元函数不是类的成员函数，友元函数可以访问所有成员
12. 某类的友元函数的作用域并非该类域。如果该友元函数是另一类的成员函数，则其作用域为另一类的类域，否则与一般函数相同。
13. 友元类：将一个类B说明为另一个类A的友元类，类B中的所有函数都是类A的友元函数，可以访问类A中的所有成员
14. 友元类的关系是单向的，如果说明类B是类A的友元类，不等于类A也是类B的友元类
15. 友元类的关系不能传递，如果类B是类A的友元类，而类C是类B的友元类，不等于类C是类A的友元类
16. 除非确有必要，一般不把整个类说明为友元类，而把成员函数说明为友元函数
17. 友元的概念破坏了类的封装性，但有助于数据共享，能够提高程序的效率
18. 类与类之间的关系：一个类的对象是另一个类的成员，一个类的成员函数是另一个类的友元，一个类定义在另一个类的说明中，一个类作为另一个类的派生类
19. 在定义（生成）一个含有对象成员的类对象时，它的构造函数被系统调用，这时将首先按照初始化符表来依次执行各对象成员的构造函数，完成各对象成员的初始化工作，而后执行本类的构造函数体。析构函数的调用顺序恰好与之相反。
20. 如果初始化符表中没有对象成员的显式初始化，则调用无参构造函数初始化类对象成员
21. 类的嵌套：一个类的说明包含在另一个类说明中，私有嵌套类（private）类外无法使用，公有嵌套（public）可以，CC::C2
22. 运算符重载：函数重载的补充，可以给运算符扩充添加新用法，可以类的友元函数定义或类的公有成员函数定义
23. 如下5个运算符不可重载： . :: ?: .\* sizeof
24. 只能以类成员而不能以友元身份重载的运算符:= ( ) [ ] -> typedef
25. 不可自创新的运算符。
26. 赋值运算符重载：默认赋值赋值运算符（浅拷贝），自定义赋值运算符重载函数，只能以成员方式重载<类名>&operator=(const <类名>&<形参对象名>)使用const保证形参不被修改
27. 使用引用保证形参、this对象和返回值具有一致性，形参相当于赋值运算的右值，返回值相当于赋值运算的左值
28. 复合赋值运算符重载定义+=、-=、\*=等复合赋值运算符的重载函数，定义+、-、\*等运算符重载函数，定义赋值运算符=的重载函数，利用上面的函数，定义复合赋值运算符重载函数
29. 插入和提取运算符：ostream& operato<<(ostream &out,const <><>),istream &operator>>(istream &in,const <><>)
30. 增量运算符和减量运算符：<类名>& operator++();//前缀增量，<类名>& operator--();//前缀减量 <类名>& operator++(int);//后缀增量 <类名>& operator--(int);//后缀减量
31. 下标运算符重载，[ ]访问链表等线性结构的元素
32. 链表：重要的数据结构，功能与数组类似，数组存放数据的地址连续，链表的不连续；链表的主要操作：建立，遍历，显式，插入，删除，查找
33. 栈：只有一个对数据进行存入和取出的端口，后进者后出
34. 队列：线性数据结构，只允许在表的一端插入元素，而在另一端删除元素，队列具有先进先出的特性，顺序队列/链式队列
35. C++通过类派生->基类/超类，派生类/子类，类继承层次结构
36. 如果基类和派生类共享相同的公有接口，则派生类被称作基类的子类型（subtype）
37. 允许派生类中的成员函数与基类的成员函数重名，按如下规定进行处理：对子类而言，不加类名限定时默认为是处理子类成员，而要访问父类重名成员时，则要通过类名限定，man2.employee::print(); //调用基类的print
38. 在类定义时，使用关键字final限定，则该类不允许任何类继承，class Box final
39. 派生类只有一个直接基类的情况称为单继承
40. 多级继承：派生出来的新类同样可以作为基类再派生出更新的类，直接参与派生某类称为直接基类，基类的基类，更深层的基类称为间接基类
41. 多重继承：一个派生类可以同时有多个基类，派生类得到多个已有类的特征
42. 派生继承的步骤：吸收基类成员（数据成员+函数成员（非私有成员），除构造函数析构函数），改造基类成员（同名新成员，同名覆盖override），发展新成员（新成员必须与基类成员不重名），重写构造函数与析构函数
43. 派生类的访问权限（public，protected，private）重要！
44. 派生类的成员：不可访问成员（基类的private），私有成员（派生类的private+基类private继承来的），保护成员（protected+基类protected继承），公有成员（public）
45. 派生类构造函数的执行：调用各基类的构造函数，调用顺序为派生继承时的基类声明顺序。若派生类含有对象成员的话，调用各对象成员的构造函数，调用顺序按照派生类中对象成员的声明顺序。执行派生类构造函数的函数体。（构造函数严格按照继承顺序，其他按照声明顺序，且类对象优先）
46. 在派生类构造函数中，只要基类不是使用无参的构造函数或默认构造函数都要显式给出基类名和参数表。如果基类没有定义构造函数，则派生类也可以不定义，全部采用系统给定的默认构造函数。如果基类定义了带有形参表的构造函数时，派生类就应当定义构造函数。
47. 可以为派生类编写拷贝构造函数，其格式与普通类的拷贝构造函数相同，适用场景也相同。派生类的拷贝构造函数，可以在成员初始化符表位置调用基类的拷贝构造函数，“拷贝”派生类中的基类部分；如果不显式地调用基类的拷贝构造函数，将自动调用基类的无参构造函数（如果有定义）或默认构造函数（如果有效）为派生类创建基类部分
48. 派生类中，可以使用using关键字，显式地“继承”基类的构造函数（无参构造函数除外），实质上是将基类构造函数当做派生类的构造函数使用，初始化派生类对象的基类部分。
49. 析构函数各部分执行次序与构造函数相反。首先对派生类新增一般成员析构，然后对新增对象成员析构，最后对基类成员析构。
50. 基类的友元不继承。如果基类有友元类或友元函数，则其派生类不因继承关系也有此友元类或友元函数。
51. 如果基类是某类的友元类，则这种友元关系是被继承的。即，被派生类继承过来的成员，如果原来是某类的友元，那么它作为派生类的成员仍然是某类的友元。
52. 基类的友元不一定是派生类的友元。基类的成员是某类的友元，则其作为派生类继承的成员仍是某类的友元。
53. 如果基类中被派生类继承的成员是静态成员，则其静态属性也随静态成员被继承。如果基类的静态成员是公有的或是保护的，则它们被其派生类继承为派生类的静态成员。<类名>::<成员名>。这些成员无论有多少个对象被创建，都只有一个拷贝。它为基类和派生类的所有对象所共享。
54. 赋值兼容性问题：如果派生类有自己的赋值运算符的重载定义，即按重载后的运算符含义处理。派生类未定义自己的赋值操作，而基类定义了赋值操作，则系统自动定义派生类赋值操作（按位拷贝），其中基类成员的赋值按基类的赋值操作进行。二者都未定义专门的赋值操作，系统自动定义缺省赋值操作（按位进行拷贝）。
55. 允许将派生类对象作为基类对象来使用，但是反过来不允许，可以基类成员=派生类成员，指向基类的指针=派生类对象的地址，基类的引用=派生类对象
56. 二基类间重名成员的处理：对于子类而言，不加类名限定时默认处理子类成员，访问父类重名成员时，要通过类名限定
57. 多级混合继承(非虚拟继承)包含两个基类实例情况的处理多级混合继承情况下，若类D从两条不同“路径”同时对类A进行了一般性继承（非虚拟继承）的话，则类D的对象中会同时包含着两个类A的实例。此时，对类D而言，要通过类名限定来指定访问两个类A实例中的哪一个。
58. 派生类中可能包含了多个基类的实例。多级混合继承情况下，若类D从两条不同“路径”同时对类A进行了虚拟继承的话，则类D的对象中只包含着类A的一个实例，这种继承也称为共享继承。被虚拟继承的基类A被称为虚基类（注意，虚基类的说明是在定义派生类时靠增加关键字virtual来指出的）。
59. 多态性和虚函数：函数重载/静态联编/函数超载/动态联编/纯虚函数/抽象基类
60. 函数超载：尽在基类和派生类之间，允许多个不同函数使用完全相同的函数名，函数参数，函数返回类型
61. 通过虚函数，指针实现动态联编，virtual关键字
62. 纯虚函数virtual <>=0，纯虚函数不能直接被调用
63. 含有纯虚函数的基类称为抽象基类，不可使用抽象基类来说明并创建它自己的对象，如果一个抽象基类的派生类没有定义基类中那一个纯虚函数，而只是继承了基类纯虚函数的话，那这个派生类还是一个抽象基类
64. 函数模版的特例，函数模版的实例化（显式指定类型参数）
65. 带有多个参数的模版<typename typename typename>->显式<int double,int>(1.5,2),或者使用auto也可以
66. 函数模版参数的默认值<typename TReturn=double>
67. 函数模版的非类型参数(typename T,int N)
68. 类模版的实例化：类模版名<具体实参表>对象名称 void stack<complex>::show\_top()
69. 类模版的静态成员：类模板也允许有静态成员。实际上，它们是类模板之实例化类的静态成员。也就是说，对于一个类模板的每一个实例化类，其所有的对象共享其静态成员，static T t,类模版的静态成员在模版定义后是不会被创建的，创建是在类的实例化之后
70. 类模版的友元：该友元函数为一般函数，则它将是该类模板的所有实例化类的友元函数。该友元函数为一函数模板，但其类型参数与类模板的类型参数无关。则该函数模板的所有实例化（函数）都是类模板的所有实例化类的友元。更复杂的情形是，该友元函数为一函数模板，且它与类模板的类型参数有关。例如，函数模板可以用该类模板作为其函数参数的类型。在友元函数模板定义与相应类模板的类型参数有关时，该友元函数模板的实例有可能只是该类模板的某些特定实例化（而不是所有实例化）类的友元
71. 类型参数检测与特例版本
72. 一般类（其中不使用类型参数的类）作基类，派生出类模板（其中要使用类型参数）template <typename T> class CA:public CB
73. 类模板作基类，派生出新的类模板。但仅基类中用到类型参数T，而派生的类模板中不使用T template <typename T> class CA : public CB<T>
74. 类模板作基类，派生出新的类模板，且基类与派生类中均使用同一个类型参数T template <typename T> class CA : public CB<T>
75. 类模板作基类，派生出新的类模板，但基类中使用类型参数T2，而派生类中使用另一个类型参数T1(而不使用T2) template <typename T1,typename T2>class CA : public CB<T2>
76. C++标准库：容器，迭代器，算法，适配器，分配器，仿函数
77. 容器：STL的所有容器都是类模板，向量（vector）相当于一个动态数组，其可以动态存储元素，并提供对容器元素的随机访问。为了提高效率，vector并不是随着每一个元素的插入而增加长度，而是当vector要增加长度的时候，他分配的空间比当前所需的空间要多一些。这多一些的内存空间使需要添加新元素的时候不必再重新分配内存。
78. 双端队列是一种增加了访问权限的队列。在队列中，我们只允许从队列的一端添加元素，在队列的另一端提取元素；在双端队列中，其支持两端的出队和入队，这个我们可以通过前面所述的顺序容器接口看出。vector与deque同属于随机访问容器，vector拥有的成员函数deque也都含有。这个我们在顺序容器一般接口表中可以看出
79. 链表（list）是由节点组成的双向链表，每一个节点都包括一个元素（即实际存储的数据）、一个前驱指针和一个后继指针，可提供两个方向的遍历功能。list无需分配指定的内存大小且可以任意伸缩，这是因为它存储在非连续的内存空间中，并且由指针将各元素链接起来
80. 关联容器也是一组特定类型对象的集合，它通过关键字（key）高效地查找和读取元素，而顺序容器通过位置查找元素。集合（set,multiset,unordered\_set），映射（map,multimap,unordered\_map）
81. 映射以键/值对（key-value）的方式组织数据，键即关键字，起到索引的作用，值即为关键字所对应的数据值。
82. C++流类库：简明与可读性/类型安全/易于扩充
83. 文件：文本文件/二进制文件
84. 所有流的行为都是相同的，但是不同的文件可能具有不同的行为，磁盘文件可读可写，显示器文件只可进行写操作，键盘文件只可进行读操作
85. 程序与一个文件交互信息->通过打开文件的操作将一个文件与另一个流联系起来，通过关闭文件的操作将一个文件与流的联系断开
86. 基本流类：ios基本流类的基类，istream由ios派生，支持输入>>提取操作，ostream由ios派生，支持输出/插入<<操作，iostream 由istream和ostream共同派生，支持输入输出双向操作
87. 预定义的流类对象：extern istream cin,extern ostream cout,extren ostream cerr,extern ostream clog,流类对象与文件之间的联系已经预定义，可认为系统已经为每一个程序都隐含了对他们打开和关闭的操作，程序中可以直接对这四个预定义流类对象，而不必先进行打开文件的操作，使用后也不需要进行关闭文件的操作
88. 用于磁盘操作的文件流类：ifstream 输入/读数据，ofstream 输出/写数据，fstream 对磁盘文件进行输入和输出数据的双向操作
89. 自定义文件流类对象：C++中用到的所有文件流类对象都要自定义 ifstream infile(“myfile.txt”),infile>>x--------ofstream outfile,outfile.open(“myfile.txt”),outfile<<
90. 插入提取运算符重载 friend istream &operator>>(istream &in,complex&com) friend ostream&operator<<(ostream &out,complex &com)
91. ios类中还设置了一个long型的数据成员用来记录当前 被设置的格式状态，该数据成员被称为格式控制标志 字(或标志状态字)。标志字是由格式控制标志位来“合 成 ” 的 。
92. 格式控制标志字：每一枚举常量值都代表着格式控制标志字中的某 一个二进制位(bit),当设置了某个标志位属性时 ,该位将取值“1”,否则该位取值“0”
93. 格式控制标志字设置函数：setf(),格式控制标志字重载函数 unsetf()
94. 通过使用位运算符“I”将多个格式控制标志位属 性进行“合成”。但从使用角度看，所设置的标 志位属性不能产生互斥。格式控制标志字中设立了三组平行的标志位，任何时刻只设置其中的某一个生态位
95. 数制标志位：ios::dec,ios::oct,ios::hex
96. 对齐标志位：ios::left,ios::right,ios::internal
97. 实数格式标志位：ios::scientific,ios::fixed
98. Ios::flags ：long flags(long IFlags)通过参数IFlags来重新设置标志字并返回,long flags()返回当前的标志字
99. long setf(long IFlags);通 过 参 数IFlags 来设置指定的格式控制标志位。注意，与flags函数的“替换”方式不同，此处为“添加”方式，即是说，它并不更改其它IFlags不涉及到的那些标志位的当前值。
100. long setf(long IFlags,long IMasks);设置指定的格式控制标志位的值,首先将第二参数IMask 所指定的那些位清零,而后用第一参数IFlags所给定的值来重置这些标志位
101. ios::unsetf long unsetf(long lFlags)通过参数IFlags 来清除指定的格式控制标志位。
102. ios::fill char fill(char cFill)将填充字符设置为cFILL，并返回原填充字符
103. Ios::precision int precision(int np)设置浮点数精度为np并返回原精度，当格式是ios::scientific 或ios::fixed，精度指小数点后的位数，否则指有效数字
104. ios::width int width (int nw)设置当前被显示数据的域宽nw 并返回原域宽。默认值为0,将按实际需要的域宽进行输出。此设置只对随后的一个数据有效，而后系统立刻恢复域宽为系统 默认值0
105. 格式控制符：直接用于提取和插入算符之后，而不像格式控制成员函数那样被单独调用
106. 读写磁盘文件的一般处理过程：打开文件（构造函数/.open），读写操作，关闭文件（.close()）
107. 打开文件：ofstream outfile(“myfile.txt”)//ofstream out file;outfile.open(“myfile.txt”)
108. 文件流类的构造函数：ifstream::ifstream (const char\*SZName,int nMode=iosLLin,int nProt=filebufLLopenprot) nMode:打开文件的方式，ios::in ios::nocreate,ios::binary
109. 文件流类的构造函数：ofstream ::ofstream(const char \*SZName,int nMode=ios::out,int nProt=filebuf::openprot)
110. 文件流类的构造函数：fstream::fstream(const char\* sZName ,int nMode,int nProt=filebuf::openprot)
111. 文件流类的成员函数：get(),put()(读写字符，多用于读写文本文件)read(),write()（读写二进制数据，多用于读写二进制文件），getline()（读字符串，多用于读文本文件）
112. 文本文件：txt以文本形式存储，有较高的兼容性，缺点是存储一批纯数值信息时，要在数据之间人为地添加分隔符，输入输出过程中，系统要对内外存的数据格式进行相应转换，text文件不便于对数据进行随机访问
113. 二进制文件：binary形式存储，便于对数据进行随机访问，每一铜类型数据所占磁盘空间的大小均相同，不必在数据之间人为添加分隔符，输入输出过程中，系统不对数据进行任何转换，缺点是兼容性低
114. 缺省打开方式时，默认为text 文件形式。若欲使用binary文件形式，要将打开方式设为“ios::binary”
115. 通常将纯文本信息(如字符串)以text 文件形式存储，而将数值信息以binary文件形式存储
116. 通常使用read()与write()对二进制文件进行读写，但若非要使用它们对文本文件进行读写时，系统在write时有可能多写出了一些东西(如，回车换行符号等)。这样将导致read时产生错误
117. 对数据文件进行随机访问：使用类成员函数write与read,并配合使用类成员函数seekp 和seekg，就可以对文件进行“随机性”(非顺序性)的读写操作
118. 字符串流类：ostrstream 通过ostrstream 类的使用，可将不同类型的信息转换为字符串，并存放在(输出到)一个用户设定的字符数组中 istrstream 通过istrstream 类的使用，则可将用户字符数组中的 字符串取出(读入),而后反向转换为各种变量的内部形式,sstream 由iostream继承，向string内写函数
119. 输入输出操作状态字，输入输出 (I/O) 操作状态字在类ios中定义 ios::goodbit ,ios::eodbit,ios::failbit,ios::badbit
120. 其它输入输出控制函数：good(),eof(),fail(),bad(),rdstate(),operator(),clear()

--NKU sfy