注：以下内容中，*typename* 表示某一个类。

面向对象程序设计初步

1. 面向对象程序设计的重要特点是代码重用和实现隐藏。
2. 直接定义在类中的成员函数都是比较简单的函数，默认为内联函数。

构造函数

3. 构造函数由系统在定义变量的时候自己调用，
4. 构造函数没有返回类型，且构造函数不需要返回值。
5. 不带参数的构造函数称为默认构造函数，一个类通常需要有一个默认构造函数。
6. 必须使用构造函数的初始化列表的情况：
   a) 数据成员是某一个类的对象（如继承时调用父亲构造函数）；
   b) 需要初始化 const 类型，或一个对象引用的时候；
7. 初始化的顺序是按照定义中数据成员的定义次序，与初始化列表写的顺序无关！也就是说，先调用父类的构造函数对父类初始化，再初始化成员。
8. 需要特别注意的是，如果没有显示调用构造函数，编译器仍会自动调用默认构造函数！

【例】

```
class test {
private:
  int a, b;
public:
  test() {a = 0, b = 0; cout << "test generate!\n";}
};

class testing : public test {
private:
  test a, b; int c;
public:
  testing() {cout << "testing generate!\n";}
};
```

在这个代码片段中，如果我们定义了一个 testing 类型对象 a，那么 a 会调用 testing 类型默认构造函数，在 testing 类型默认构造函数中隐式调用了 test 的基类构造函数和两个 test 成员的构造函数。所以这份程序应该输出 3 个 test generate 和 1 个 testing generate。

9. 用同类型对象初始化的叫做复制构造函数，原型为 *typename*(const *typename*&);
10. 如果类的设计者没有定义复制构造函数，编译器会自动生成一个复制构造函数，该函数仅仅完全复制了所有内容，称为默认复制函数。
11. 系统自动调用复制构造函数的场合：
    a) 对象定义时：*typename* a = b, c(d); 其中 a 和 c 均需要使用复制构造函数；
    b) 函数调用时传参；
    c) 函数返回的时候会创建一个新的临时对象，调用复制构造函数，用 a 初始化这个临时对象，然后返回这个临时对象。

this 指针

12. 成员函数中对对象的访问是通过 this 指针实现的。
13. 通常写成员函数的时候可以省略 this，编译时会自动补全。但是如果在成员函数中要把对象当作整体访问时，必须显式使用 this 指针。

析构函数

14. 析构函数没有返回类型、返回值和参数。
15. 在一个对象中，析构函数析构顺序与构造函数构造顺序完全相反。

常量与静态

16. 常量对象只能调用常量成员函数，不能调用其他成员函数，即使函数没有改变变量的值。
17. 类的静态数据成员拥有一块单独的存储区，不管创建多少对象，所有对象的静态成员共享同一块空间。
18. 为静态数据成员分配空间称为静态数据成员的定义，应该在类的实现文件中对其定义。
19. 静态成员函数用于操作静态数据成员。
20. 静态常量成员时静态成员的常量形式，必须在类的定义时初始化。可使用枚举类型代替。

友元

21. 友元关系是单向的，不能反向，也不能继承。
22. 重载 istream 运算符>>和 ostream 运算符<<的时候，必须使用友元重载（由于成员函数会自动添加 this）。

运算符重载

23. 运算符重载不改变操作符的优先级、结合性和操作数个数。
24. 赋值运算符（=）、下标运算符（[]）、函数调用运算符（()）和成员访问运算符（->）必须重载为成员函数，由于该运算符的第一个运算对象必须为相应类的对象。
25. 一般来说，需要定义复制构造函数的类也需要自定义赋值运算符重载函数。
26. 需要注意的是，当定义的时候使用“=”的时候并非调用赋值运算符重载函数!

【例】Rational（有理数）类存在一个构造函数 Rational(double x)将 x 变为分数形式的有理数，那么定义 Rational a=2; 时默认调用的是构造函数！若 Rational a=b;，其中 b 也是 Rational 的对象，则调用的是复制构造函数!

27. 后置++或--有一个多余的 int 类型参数。
28. 只要类有一个单个参数的构造函数，就会执行参数类型到该类型对象的隐式类型转换!（如存在一个只有 double 类型的构造函数，那么执行 a=2 的时候实际上调用的是这个构造函数!）在构造函数前加 explicit 可以阻止这种隐式转换。
29. 类型转换函数必须定义为类的成员函数，格式为 operator *typename*() {}。

【例】

```
class test {
private:
  int a, b;
public:
  test() {a=0, b=0;}
  test(int c) {a=c;}
  test operator+(test d) {return d;}
  operator int() const{return a;}
};
```

这是一个没有实际意义的类的定义，但是在执行语句 b+3 的时候（b 为 test 类型），会报错。原因：二义性（既可以将 b 转换为 int 执行 int 的加法，又可以利用构造函数将 3 转化为 test 类型执行 test 类型的加法!）

组合继承

30. 最大特点：代码重用。

31. 重定义基类的成员函数后，派生类对象只调用派生类的成员函数。
32. 可以通过重载运算符=实现基类对派生类的赋值。
33. 将派生类对象赋值给基类对象时，只保留基类的全部内容，舍弃派生类增加的所有内容。
34. 从基类的指针出发，只能访问派生类的基类部分，不能访问派生类的新增加部分。
35. 基类对象引用派生类的对象，只能通过基类对象引用访问派生类中基类的部分（理解为基类部分的别名）。

虚函数

36. 派生类中重新定义虚函数的时候，原型必须和基类虚函数完全相同。
37. 构造函数不能是虚函数，但是析构函数可以是虚函数且最好是虚函数。
38. 纯虚函数是一个在基类中声明的虚函数，他在基类中没有定义，但是要求在派生类中定义自己的版本。
39. 如果一个类含有至少一个纯虚函数，则称为抽象类。
40. 无法定义抽象类的对象！

模板

41. 类模板 vs 模板类 vs 类模板的实例 vs 模板类的实例：

【例】

```
template <class T>
class array { … }
```

array 是类模板，array<int>是类模板的实例，模板类是类模板实例化的产物，所以array<int>也是模板类，array<int> b;，其中 b 就是一个模板类的实例。