封装与接口

1. 函数重载

“名一样，意不同”。靠**参数类型**来区分。返回值、参数名称不能作为区分的标识。

编译器将根据函数调用语句的实际参数来决定哪一个函数被调用。

```
sum(int a, int b);
sum(double a, double b);
```

【常识】c++中，NULL被定义为**0**. c++11中引入nullptr，是真正意义上的空**指针**。

2. 类

类的成员（数据、函数）访问权限有public, private, protected.

- 被public修饰的成员可以在类外用“.”操作符访问。
- 被private修饰的成员不允许在类外用“.”操作符访问

```
class Matrix {
public:
    void fill(char dir){
        ...
        this->data[0][0] = 1;//等价于 data[0][0] = 1;
    }
private:
    int data[6][6];
};
```

【this指针】所有成员函数的参数中，隐含着一个指向**当前对象**的指针变量，其名称为this。这也是成员函数与普通函数的重要区别。

3. 运算符重载

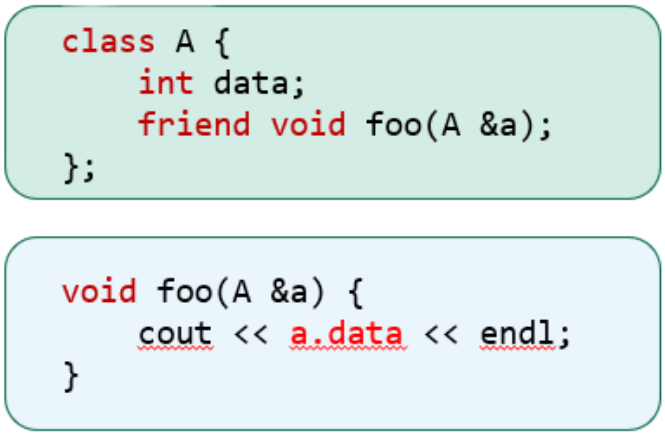
```
class Test {
public:
  int operator() (int a, int b) { //()运算符重载
    cout << "operator() called. " << a << ' ' << b << endl;
    return a + b;
  }
  int& operator[] (const char* name){ // []运算符重载
        for (int i = 0; i < 7; i++) {
            if (strcmp(week_name[i], name) == 0)
```

```
                return temp[i];
            }
        }
        Test operator++ () { //++运算符重载
            ++data;
            return Test(data);
        }
    };
```

4.友元

•在类内进行友元的声明。

•被声明为友元的函数或类，具有对出现友元声明的类的**private及protected**成员的访问权限。即可以访问该类的一切成员。



44

•友元不传递：朋友的朋友不是你的朋友

•友元不继承：朋友的孩子不是你的朋友

可以

•声明别的类的成员函数，为当前类的友元。

•其中，构造函数、析构函数（后续内容）也可以是友元。

例如：

```
class Y {
    int data;
    friend char* X::foo(int); //类X的函数foo为友元
    friend X::X(char), X::~X(); //类X的构造函数、析构函数为友元
};
```

5.内联函数

使用内联函数，编译器自动产生等价的表达式。

内联函数的注意事项:

•避免对大段代码使用内联修饰符。这是因为内联修饰符相当于把该函数在所有被调用的地方拷贝了一份，所以大段代码的内联修饰会增加负担。（代码膨胀过大）

•避免对包含循环或者复杂控制结构的函数使用内联定义。因为内联函数优化的，只是在函数调用的时候，会产生的压栈、跳转、退栈和返回等操作。所以如果函数内部执行代码的时间比函数调用的时间长得多，优化几乎可以忽略。

•内联修饰用在函数定义的时候，而不是函数声明的时候。

•定义在类声明中的函数默认为内联函数。

•一般构造函数、析构函数都被定义为内联函数。

•在头文件中加入或修改 inline 函数时，使用了该头文件的所有源文件都必须重新编译。

•内联修饰符更像是**建议**而不是命令。编译器"有权"拒绝不合理的请求，例如编译器认为某个函数不值得内联，就会忽略内联修饰符。

•编译器会对一些没有内联修饰符的函数，自行判断可否转化为内联函数，一般会选择短小的函数。