

复旦大学

面向对象程序语言C++

类

周雅倩
zhouyaqian@fudan.edu.cn
2016/4/13

复旦大学媒体计算研究所

定义新类型

- 内置类型：核心语言的一部分
 - char, int, double
- 类
 - 库中的类：string, vector, istream
 - 程序员设计的类
- C++程序设计思想：允许程序员创建与内置类型一样易用的类型。

2016/4/13

复旦大学媒体计算研究所

回顾Student info

```
struct Student_info{
    std::string name;
    double midterm, final;
    std::vector<double> homework;
};
```

- 缺点：不好用
 - 若数据读入失败，则成员值无效。导致产生不正确的结果或程序崩溃。
 - 若要检测对象是否包含有效数据，需要用户了解实现细节。
- 无法支持一些约定
 - 如：一个学生信息一旦从文件读入，则不会改变
- “接口”分散
 - 可以改进，但费事：把改变对象的状态的函数（如read）放在一个单独的头文件中

2016/4/13

复旦大学媒体计算研究所

类

- 使用student_info结构类型的程序员都可以（也必须）直接控制这些数据元素。
- 但是，我们希望隐藏其存储方式的实现细节，而不是允许用户直接访问这些数据。
- 特别地，希望用户只能通过函数来访问这些对象。这些函数就是类的接口。

2016/4/13

复旦大学媒体计算研究所

第一个成员函数

```
istream & Student_info::read(istream & in)
{
    in >> name >> midterm >> final;
    read_hw(in, homework);
    return in;
}

istream & read(istream & is, Student_info & s)
{
    is >> s.name >> s.midterm >> s.final;
    read_hw(is, s.homework);
    return is;
}
```

- 与原先代码的不同之处
 - 函数的名字是 Student_info::read 而不是简单的 read
 - 由于它是成员函数，所以不需要把一个 Student_info 对象作为参数传递，而且在函数体中不需要定义一个 Student_info 对象。
 - 可以直接访问对象的数据元素。

2016/4/13

复旦大学媒体计算研究所

const 函数

```
double Student_info::grade() const
{
    return ::grade(midterm, final, homework);
}

double grade(const Student_info & s)
{
    return grade(s.midterm, s.final, s.homework);
}
```

- 把 :: 放在一个名字前表示我们要使用这个名字的一个版本，但是这个名字决不是任何对象的成员
- 紧跟在 grade 函数的参数列表后的 const 关键字，表示该函数不能改变他们所操作的对象的内部数据。

2016/4/13

const 函数

- 通过在参数列表后插入**const**关键字，可以把成员函数定义为**const**函数
 - `double Student_info::grade() const`
- 这样的成员函数不能改变调用它们的对象的状态
- **const**对象只能调用**const**成员函数

2016/4/13

成员函数

- 成员函数是通过特定的对象来隐式地调用。
- 在一个成员函数中，引用到的成员数据和函数会隐式的绑定到调用这个函数的对象上。
- 在类外定义成员函数：
 - `class-name::member-name`

2016/4/13

非成员函数

- 通用规则：
 - 如果这个函数会改变对象的状态，那么这个函数就应该成为这个对象的成员。
- 我们在新的设计中把**read**和**grade**函数都变成了成员函数。那么**compare**函数呢？
 - 考虑这个函数是做什么的？比较大小。

2016/4/13

保护

| | |
|---|--|
| <pre>class Student_info{ public: //接口 double grade() const; std::istream & read(std::istream &); private: std::string name; double midterm, final; std::vector<double> homework; };</pre> | <pre>struct Student_info{ double grade() const; std::istream & read(std::istream &); std::string name; double midterm, final; std::vector<double> homework; };</pre> |
|---|--|

2016/4/13

保护标签

- 保护标签控制对类的成员的访问
 - **public**: 公有成员可以被所有用户访问
 - **private**: 私有成员只有这个类的成员可以访问，用户不能访问。
 - **protected**
- 在一个类中，保护标签可以按任意顺序出现，并且可以多次出现。

2016/4/13

struct Vs. class

- 用户定义的类型可以是**struct**，也可以是**class**。
- 它们之间唯一的区别在于默认的方式不同
 - **struct** : 公有
 - **class** : 私有
- 一般**struct**用来表示简单的类型

2016/4/13

例子

```
class Student_info{
public:
    double grade() const;
    //etc
};

class Student_info{
    std::string name;
    //etc
public:
    double grade() const;
    //etc
};
```

```
struct Student_info{
    double grade() const;
    //etc
};

struct Student_info{
private:
    std::string name;
    //etc
public:
    double grade() const;
    //etc
};
```

2016/4/13

访问器函数

```
class Student_info{
public:
    //接口
    double grade() const;
    std::istream & read(std::istream
    &);
    std::string name() const{return
    n;}
```

```
class Student_info{
public:
    //接口
    double grade() const;
    std::istream & read(std::istream
    &);
private:
    std::string name;
    double midterm, final;
    std::vector<double> homework;
};
```

2016/4/13

内联函数

■ 成员函数可以定义在类定义中，也可以定义在类定义外。

inline string Student_info::name() const{ return n; }

■ 定义在类定义中的成员函数称为内联函数，可以避免调用函数的开销。

2016/4/13

compare函数

```
bool compare(const Student_info &x, const Student_info &y)
{
    return x.name() < y.name();
}
```

注意：这里compare函数是非成员函数

2016/4/13

检测对象是否为空

■ 以下代码会产生意外：

```
Student_info s;
cout << s.grade() << endl;
```

— (因为homework为空)：

■ 一种方式：直接检查homework成员

— 缺点：破坏封装性

■ 更好的方式：提供检测函数

```
class Student_info{
public:
    bool valid()const {return homework.empty();}
    //etc
}
```

2016/4/13

Student info类

```
class Student_info{
public:
    double grade() const;
    std::istream & read(std::istream &);
    bool valid()const {return homework.empty();}
    std::string name() const{return n;}
private:
    std::string n;
    double midterm, final;
    std::vector<double> homework;
};

bool compare(const Student_info &x, const Student_info &y);
```

2016/4/13

构造函数

- 是特殊的成员函数，定义了对象如何初始化。
- 名字和类的名字相同，没有返回值。
- 一个类可以定义多个构造函数，只要参数个数或者类型不同。

```
class Student_info{
public:
    Student_info();           定义对象:
    Student_info(std::istream&); Student_info s;
                                Student_info s(cin);
}
```

2016/4/13

构造函数

- 不能显式的调用一个构造函数，但是创建类的一个对象时就会自动的调用适当的构造函数
- 如果没有定义任何构造函数编译器就会为我们合成一个。
- 名字和类的名字相同，没有返回值。
- 一个类可以定义多个构造函数，只要参数个数或者类型不同。
- 编程习惯：
 - 从任何一个构造函数退出前，确保每个数据成员都有一个有意义的值。

2016/4/13

对象初始化规则

- 若一个类定义了一个或多个构造函数，那么适当的构造函数会完全控制这个类的对象的初始化
- 若一个对象的类型是内置类型的话，那么值初始会把它设置为0，而默认初始化会提供一个不明确值。
- 若一个对象的类没有定义任何构造函数，那么对这个对象进行值初始化或默认初始化就会对它的每个数据成员进行，整个过程递归进行

2016/4/13

默认构造函数

- 不带有参数的构造函数被认为是默认构造函数。它的工作是确保对象的数据成员被适当地初始化。
- Student_info: 初始化表示还没有读取一条记录
 - homework为空
 - n为空string
 - midterm和final为0

```
Student_info::Student_info() : midterm(0), final(0){}
```

2016/4/13

对象创建过程

- 当我们创建一个新的类对象时，会发生下面几个连续的步骤
 - 系统会分配内存空间，以保存这个对象
 - 根据构造函数的初始化列表，初始化这个对象
 - 执行构造函数的函数体


2016/4/13

构造函数初始化列表


```
Student_info::Student_info() : midterm(0), final(0){}
```

- 构造函数初始化列表是一个用逗号分隔的列表，这个列表包含的是 member-name(value)对。没有明显的初始化的数据成员会被隐式的初始化。
- 通常来说，显式地为一个成员提供一个初始值，要比在构造函数的函数体中对这个成员进行赋值要好得多，可以避免做两次相同的工作。

2016/4/13




初始化顺序



复旦大学
媒体计算
研究所

- 成员初始化的顺序取决于它们在类中声明的顺序。
- 所以在用类的一个数据成员来初始化另外一个成员时一定要小心。
- 安全的做法：在构造函数的函数体中对这些成员赋值，而不是在构造函数初始化列表中初始化它们。
 - 因为成员初始化的顺序取决于它们在类中声明的顺序

2016/4/13





带有参数的构造函数




复旦大学
媒体计算
研究所


```
Student_info::Student_info(istream & is){
    read(is);
}
```

2016/4/13





使用Student_info类



复旦大学
媒体计算
研究所

```
int main()
{
    vector<Student_info> students;
    Student_info record;
    string::size_type maxlen=0;
    while(record.getline())
        if(record.name().size() > maxlen) maxlen= record.name().size();
    students.push_back(record);
}

sort(students.begin(), students.end(), compare);
for(std::vector<Student_info>::iterator it=students.begin(); it!=students.end(); it++){
    cout << *it << endl;
    cout << string(maxlen+1 - it->name().size(), ' ');
    try{
        double final_grade = it->grade();
        streamsize prec=out.precision();
        cout << "Your final grade is " << setprecision(3)
            << final_grade
            << setprecision(prec) << endl;
    }catch(domain_error e){
        cout << e.what();
    }
    cout << endl;
}
return 0;
}
```

2016/4/13

