

第9章 定义新类型

刘 卉

huiliu@fudan.edu.cn



前言

C++有两种类型

- 内置类型: `int`, `char`, `float`, `double`, `bool`,
- 类: `string`, `vector`, `list`,

创建与内置类型同样易用的类型

- 从计算成绩问题出发, 介绍基本的类定义功能.



9.1 回顾Student_info

□ 存在的问题

1) 使用结构和函数的程序员，必须遵循某些约定.

e.g. 使用新创建的Student_info对象时，首先要把数据读取到这个对象里；否则会产生不可预测的结果.

e.g. 检测一个Student_info对象是否包含有效数据，唯一方式是查看数据成员→必须了解Student_info结构的实现细节.

存在的问题(续)

- 对于Student_info结构而言：
 - 用户(程序员)可直接控制其数据成员, 因为没有任何访问限制.
 - 用户(程序员)必须直接控制其数据成员, 因为没有其它可用操作.

2) 代码无法保证学生记录不会改变.

e.g. `s.name = ""`; //可随意修改学生的名字

3) Student_info结构的接口是分散的.

e.g. 为学生读入数据/计算成绩的函数`read(...)`/`grade(...)`分别
在不同的头文件/源文件中声明和定义.



9.2 类

□ 我们希望：

- 1) 隐藏Student_info的实现细节，不允许用户直接访问其数据成员，只能通过函数访问。
- 2) 为用户提供控制Student_info对象的操作→类的接口。
 - 为Student_info对象读入数据；
 - 求Student_info对象的成绩；
 - 按名字的字典顺序比较2个Student_info对象。



类

□ 类定义放在头文件中

- 在头文件中使用完整限定的名字：提供给别的用户使用，不要(尽可能少地)使用using声明。
- 在源程序文件中则没有这种限制。



9.2.1 成员函数

□ 添加对Student_info对象的访问接口

Student_info.h

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


main.cpp

```
Student_info s;  
s.read(cin);  
cout << s.grade();
```

student_info.cpp

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

函数名



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

直接访问
数据成员

□ 新旧read函数的三处不同：

1) 函数名变为Student_info::read;

□ 限定read函数是Student_info的成员.

2) 不需要Student_info对象作为函数参数(而是调用主体);

s.read(cin); //比较之前的用法: read(cin, s)

□ 调用read函数的对象→隐含的Student_info&参数.

3) 可直接访问对象的数据元素.

□ 无需限定, 表示正在操作对象的成员.

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

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

在类当中调用全局函数grade

- grade函数是一个const成员函数
 - 确保该函数不会改变对象的数据成员.
 - 确保const对象也可以调用grade成员函数.

- ❑ const对象可调用const成员函数,但不能调用非const成员函数.

```
const Student_info s1(s);
```

```
s1.read(cin); //read()不是const成员函数, 编译出错!
```

```
cout << s1.grade() << endl; //grade()是const成员函数, 编译通过
```

- ❑ 非const对象也能调用const成员函数→函数会把它当作const对象来处理.

```
Student_info s;
```

```
s.read(cin); //OK!
```

```
cout << s.grade() << endl; //OK!
```

- ⚠ 在函数声明和定义处都要使用const限定词.

Student_info.h

Student_info.cpp



9.2.2 非成员函数

□ compare函数应定义为类的成员吗？

- 规则一：如果函数会改变对象的状态→成员
- 规则二：用户会以何种方式调用它？

比较两个Student_info对象x, y

compare (x, y) ✓ x和y是平等的，对称的

x.compare(y) 不自然的调用方式



9.3 保护

- 隐藏Student_info对象的数据成员
 - 只允许用户通过成员函数访问。
- C++支持数据隐藏——保护标签
 - public: 可被所有用户访问。
 - private: 只有类本身可以访问, 类用户不能访问。

```
class Student_info {
```

```
public: 保护标签定义了其后成员的可访问性
```

类的外部
也可访问

```
// interface goes here
```

```
double grade() const;
```

```
std::istream& read(std::istream&);
```

```
private: 可按任意顺序出现, 亦可多次出现.
```

仅类本身
可以访问

```
// implementation goes here
```

```
std::string name;
```

```
double midterm, final;
```

```
std::vector<double> homework;
```

```
};
```



class vs struct

默认保护方式不同

- 默认保护方式应用于第一个保护标签之前的成员.
class: 默认保护方式为private.
struct: 默认保护方式为public.
- 使用struct表示简单类型 → 数据结构可公开.

```
struct Student_info {  
    // public by default  
    double grade() const;  
    // etc.  
};
```

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

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

```
class Student_info {  
    //private by default  
    std::string name;  
    // other private members  
public:  
    double grade() const;  
    // other public members  
};
```




9.3.1 访问器函数 (accessor function)

□ 如何在类定义的外部取得学生的名字?

[例] `Student_info s;`

`s.read(cin);`

`s.name = "Jack";` ✗

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

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

`{`

`return x.name < y.name;` ✗

`}`

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

通过复制n而非返回其引用⇒确保用户可读取n, 但无法修改它.



内联函数

- name函数的定义在类定义的内部
 - 暗示编译器，把该函数的调用作内联(inline)展开，避免函数调用的开销。



还需要其它访问器函数吗?

- 获取学生的期中/期末/作业成绩的访问器函数?
 - 不能随意访问隐藏的数据→破坏封装性.
 - 仅提供学生名字的访问器函数, 而不提供期中/期末/作业成绩等数据的访问器函数→只与实现相关, 不是接口的组成部分.



非成员函数compare

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

- compare函数是接口的一部分，应在Student_info.h中包含该函数的声明。
- 在定义成员函数的源文件Student_info.cpp中，包含该全局函数的定义。



9.3.2 检测对象是否为空

□ 帮助用户确定是否可以计算成绩

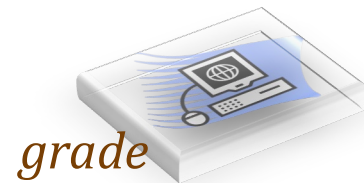
```
class Student_info {  
public:  
    bool valid() const { return !homework.empty(); }  
    // as before  
};
```

- 在调用grade函数之前，先检测当前对象是否有效，从而避免异常。



9.4 Student_info类

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





克服了 struct Student_info 的问题

- 1) 使用结构和函数的程序员，必须遵循某些约定。
- 2) 代码无法保证学生记录不会改变. `struct Student_info`
- 3) Student_info结构的接口是分散的。



- 1) 隐藏了实现细节，用户不能直接访问. `class Student_info`
- 2) 用户只能通过read成员函数改变Student_info对象的状态。
- 3) 对Student_info对象的所有操作从逻辑上集合在一起。



9.5 构造函数 (constructor)

- 创建一个对象时会发生什么? `Student_info s;`
 - 创建标准库中某个类的对象时，会有一个适当的初值。
e.g. 创建一个string或vector对象时，如果提供了初值，将按初值初始化对象；如果没有，就会得到一个空对象。
- 构造函数是特殊的成员函数
 - 定义了对象如何初始化。
 - ⚠ 不能显式调用构造函数，创建类的对象时会自动调用。

合成构造函数

- 如果没有定义构造函数，编译器会合成一个。 `Student_info s;`
- 合成构造函数会初始化对象的数据成员，成员的初值取决于对象的创建方式。
 - 1) 对象是局部变量→数据成员被默认初始化。

内置类型成员的默认初始化：不确定值。
 - 2) 对象作为全局变量/容器的元素→成员被值初始化。

内置类型成员的值初始化：设置为0；
- 数据成员是标准库类型：按其默认构造函数初始化。
- 数据成员是自定义类型：初始化过程递归进行。

```
class Student_info {  
public:  
    ...  
private:  
    std::string n;  
    double midterm, final;  
    std::vector<double> homework;  
};  
int main()  
{  
    Student_info record;  
    ...  
    return 0;  
}
```

□ 必须初始化

- 应确保每个数据成员任何时候都有一个有意义的值.

- `record.n`, `record.homework`: 自动初始化为空`string`和空`vector`, 因为这两个类都有自己的构造函数.
- `record.midterm`, `record.final`: 默认初始化, 即, 值不确定⇒潜在错误风险!



自定义构造函数

□ 构造函数 vs 其它成员函数

- 构造函数名与类名相同，且没有返回值类型。

□ 可定义多个版本的构造函数

e.g. `string s`, `string t(s)`, `string z(n, c)`

e.g. `container<T> c`, `container<T> c(c2)`,
`container<T> c(n)`, `container<T> c(n, t)`,
`container<T> c(b, e)`



Student_info类需定义哪些构造函数?

□ 定义两个构造函数

- 1) 不带任何参数, 创建一个空的Student_info对象

```
Student_info s; //an empty Student_info
```

- 2) 带一个istream&参数, 从中读取学生记录来初始化该对象.

```
Student_info s2(cin); //initialize s2 by reading from cin
```



更新类定义

□ 添加两个构造函数

```
class Student_info {  
public:  
    Student_info(); //construct an empty Student_info object  
    Student_info(std::istream&); //construct one by reading a stream  
    //as before  
};
```

- 构造函数名与类名相同，且没有返回值类型。



9.5.1 默认构造函数

□ 不带任何参数的构造函数

- 确保对象的数据成员被适当的初始化.

构造函数初始化列表

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

- 编译器将使用()中的值初始化给定成员.
- n和homework被隐式初始化.
 - n被string类的默认构造函数初始化;
 - homework被vector类的默认构造函数初始化.

```
class Student_info {  
public:  
    Student_info(): midterm(0), final(0) { }  
    //as before  
private:  
    std::string n;  
    double midterm, final;  
    std::vector<double> homework;  
};  
int main()  
{  
    Student_info record;  
    ...  
    return 0;  
}
```

- record.n, record.homework: 自动初始化为空string和空vector.
- record.midterm, record.final: 初始化为0⇒消除潜在错误风险!

□ 创建一个新的类对象时，

step1. 系统分配内存空间，以保存该对象；

step2. 根据构造函数的初始化列表，初始化该对象；
包括成员的隐式初始化(n和homework)

step3. 执行构造函数的函数体。

■ 问题：n和homework在midterm, final之前/之后初始化？

答案：成员初始化的顺序取决于它们被声明的顺序。

- 在初始化列表中为数据成员提供初值，优于在函数体中对该成员赋值。

```
Student_info::Student_info()
```

```
{ midterm = 0; final = 0; } //不建议这样做!
```



9.5.2 带参数的构造函数

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

- 把实际工作交给read函数.

□ 带一个参数的构造函数→隐式类型转换

```
string s;
```

```
s = "test"; //将字符串"test"隐式转换为匿名string对象,赋给s
```

- string t(s):定义t并用s初始化t, s是string对象或字符串常量.

```
Student_info s;
```

```
s = cin; // s=Student_info(cin), 等待从标准输入流输入数据
```

- 系统自动调用构造函数Student_info(istream&), 将cin隐式转换为一个匿名Student_info对象, 赋值给s.
- 这种隐式类型转换不自然, 如何避免?

□ explicit

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

- 仅在用户明确调用该构造函数的地方, 编译器才使用它.

```
Student_info s(cin); // 自动调用Student_info(cin)
```

```
Student_info s; s = cin; // 不会调用Student_info(cin), 编译错误
```

- 只能用在带一个参数的构造函数定义中.

9.6 使用Student_info类

- 修改Student_info结构的目的
 - 阻止用户修改数据成员的值;
 - 用户使用类的接口编写程序.
- 格式化输出学生的最终成绩
 - 使用Student_info的新结构重新解决该问题.





小 结

- 用户自定义类型: struct/class
- 保护标签——控制对类成员的访问
- 成员函数
 - 通过类的对象来调用
 - 成员函数的声明必须放在类定义中，成员函数的定义可放在类定义中/类定义外。
 - const成员函数

小结

□ 构造函数

- 优先使用构造函数初始化列表.
- 成员初始化的顺序取决于它们被声明的顺序.
- 用类的一个数据成员初始化另一个数据成员时, 放在构造函数的函数体中进行.



如何抽象出类定义

- 类的数据属性全部定义为private成员
 - 外部需要访问的数据，提供public访问器函数
 - 类的操作全部定义为成员函数
 - 仅涉及类的实现→private成员
 - 提供给类用户的接口→public成员
- e.g. 类的用户要创建类的对象→构造函数
- 类的用户要为类的对象读入/输出数据→read()/output()

- 与类紧密相关的全局函数声明也放在类定义的头文件中
 - 对应的全局函数定义则放在类实现的源文件中。
- 重要原则： 只要不修改类数据属性的成员函数均定义为 `const`
 - 以便类的 `const` 对象可以调用。