第4章（二）

类的组合

组合的概念

- 类中的成员是另一个类的对象。
- 可以在已有抽象的基础上实现更复杂的抽象。

类组合的构造函数设计

- 原则：不仅要负责对本类中的基本类型成员数据初始化，也要对对象成员初始化。
- 声明形式：

```
类名::类名(对象成员所需的形参，本类成员形参)
        :对象1(参数)，对象2(参数)，.....
{
//函数体其他语句
}
```

构造组合类对象时的初始化次序

- 首先对构造函数初始化列表中列出的成员（包括基本类型成员和对象成员）进行初始化，初始化次序是成员在类体中定义的次序。
  - 成员对象构造函数调用顺序：按对象成员的声明顺序，先声明者先构造。
  - 初始化列表中未出现的成员对象，调用用默认构造函数（即无形参的）初始化
- 处理完初始化列表之后，再执行构造函数的函数体。

类组合程序举例

- 例4-4 类的组合，线段（Line）类

```
//4_4.cpp
#include <iostream>
#include <cmath>
using namespace std;
```

```
class Point {        //Point类定义
public:
    Point(int xx = 0, int yy = 0) {
      x = xx;
      y = yy;
    }
    Point(Point &p);
    int getX() { return x; }
    int getY() { return y; }
private:
    int x, y;
};

Point::Point(Point &p) { //复制构造函数的实现
    x = p.x;
    y = p.y;
    cout << "Calling the copy constructor of Point" << endl;
}

//类的组合
class Line {//Line类的定义
public:      //外部接口
    Line(Point xp1, Point xp2);
    Line(Line &l);
    double getLen() { return len; }
private:     //私有数据成员
    Point p1, p2;  //Point类的对象p1,p2
    double len;
};

//组合类的构造函数
Line::Line(Point xp1, Point xp2) : p1(xp1), p2(xp2) {
    cout << "Calling constructor of Line" << endl;
```

```
    double x = static_cast<double>(p1.getX() - p2.getX());
    double y = static_cast<double>(p1.getY() - p2.getY());
    len = sqrt(x * x + y * y);
}
Line::Line (Line &l): p1(l.p1), p2(l.p2) {//组合类的复制构造函数
    cout << "Calling the copy constructor of Line" << endl;
    len = l.len;
}

//主函数
int main() {
    Point myp1(1, 1), myp2(4, 5);      //建立Point类的对象
    Line line(myp1, myp2);      //建立Line类的对象
    Line line2(line);       //利用复制构造函数建立一个新对象
    cout << "The length of the line is: ";
    cout << line.getLen() << endl;
    cout << "The length of the line2 is: ";
    cout << line2.getLen() << endl;
    return 0;
}
```

前向引用声明

- 类应该先声明，后使用
- 如果需要在某个类的声明之前，引用该类，则应进行前向引用声明。
- 前向引用声明只为程序引入一个标识符，但具体声明在其他地方。
- 例：

```
class B;  //前向引用声明
class A {
public:
  void f(B b);
};
class B {
```

```
public:
  void g(A a);
};
```

前向引用声明注意事项

- 使用前向引用声明虽然可以解决一些问题，但它并不是万能的。
- 在提供一个完整的类声明之前，不能声明该类的对象，也不能在内联成员函数中使用该类的对象。
- 当使用前向引用声明时，只能使用被声明的符号，而不能涉及类的任何细节。
- 例

```
class Fred; //前向引用声明
class Barney {
   Fred x; //错误：类Fred的声明尚不完善
};
class Fred {
   Barney y;
};
```

结构体

- 结构体是一种特殊形态的类
  - 与类的唯一区别：类的缺省访问权限是private，结构体的缺省访问权限是public
  - 结构体存在的主要原因：与C语言保持兼容
- 什么时候用结构体而不用类
  - 定义主要用来保存数据、而没有什么操作的类型
  - 人们习惯将结构体的数据成员设为公有，因此这时用结构体更方便

结构体的定义

```
struct 结构体名称 {
      公有成员
protected:
    保护型成员
private:
     私有成员
```

};

结构体的初始化

- 如果一个结构体的全部数据成员都是公共成员，并且没有用户定义的构造函数，没有基类和虚函数（基类和虚函数将在后面的章节中介绍），这个结构体的变量可以用下面的语法形式赋初值

类型名 变量名 = { 成员数据1初值, 成员数据2初值, ... };

例 4-7 用结构体表示学生的基本信息

```
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;

struct Student {   //学生信息结构体
    int num;              //学号
    string name;  //姓名，字符串对象，将在第6章详细介绍
    char sex;             //性别
    int age;        //年龄
};

int main() {
    Student stu = { 97001, "Lin Lin", 'F', 19 };
    cout << "Num:  " << stu.num << endl;
    cout << "Name: " << stu.name << endl;
    cout << "Sex:  " << stu.sex << endl;
    cout << "Age:  " << stu.age << endl;
    return 0;
}
```

运行结果：

```
Num:  97001
Name: Lin Lin
Sex:  F
Age:  19
```

联合体

声明形式

```
union 联合体名称 {
   公有成员
protected:
   保护型成员
private:
   私有成员
};
```

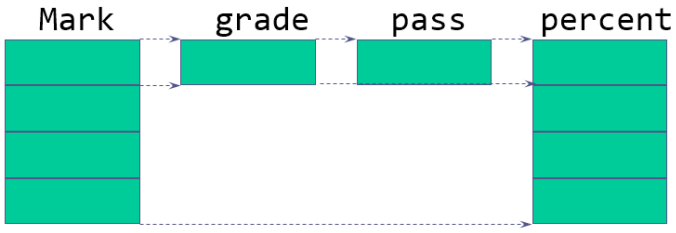
特点：

- 成员共用同一组内存单元
- 任何两个成员不会同时有效

联合体的内存分配

- 举例说明：

```
union Mark {      //表示成绩的联合体
   char grade;    //等级制的成绩
   bool pass;     //只记是否通过课程的成绩
   int percent;   //百分制的成绩
};
```



无名联合

- 例：

```
union {
  int i;
  float f;
}
```

在程序中可以这样使用：

```
    i = 10;
    f = 2.2;
```

下面我们看一个联合体的例题

例 4-8 使用联合体保存成绩信息，并且输出。

```
#include <string>
#include <iostream>
using namespace std;
class ExamInfo {
private:
    string name;  //课程名称
    enum { GRADE, PASS, PERCENTAGE } mode;//计分方式
    union {
      char grade;  //等级制的成绩
      bool pass;    //只记是否通过课程的成绩
      int percent;  //百分制的成绩
    };
public:
    //三种构造函数，分别用等级、是否通过和百分初始化
    ExamInfo(string name, char grade)
      : name(name), mode(GRADE), grade(grade) { }
    ExamInfo(string name, bool pass)
      : name(name), mode(PASS), pass(pass) { }
    ExamInfo(string name, int percent)
      : name(name), mode(PERCENTAGE), percent(percent) { }
    void show();
}

void ExamInfo::show() {
    cout << name << ": ";
    switch (mode) {
      case GRADE: cout << grade;  break;
      case PASS: cout << (pass ? "PASS" : "FAIL"); break;
```

```
        case PERCENTAGE: cout << percent; break;
    }
    cout << endl;
}

int main() {
    ExamInfo course1("English", 'B');
    ExamInfo course2("Calculus", true);
    ExamInfo course3("C++ Programming", 85);
    course1.show();
    course2.show();
    course3.show();
    return 0;
}
```

运行结果：

English: B

Calculus: PASS

C++ Programming: 85

枚举类

枚举类定义

- 语法形式

enum class 枚举类型名: 底层类型 {枚举值列表};

- 例：

```
enum class Type { General, Light, Medium, Heavy};
enum class Type: char { General, Light, Medium, Heavy};
enum class Category { General=1, Pistol, MachineGun, Cannon};
```

枚举类的优势

- 强作用域，其作用域限制在枚举类中。
  - 例：使用Type的枚举值General：

    Type::General
- 转换限制，枚举类对象不可以与整型隐式地互相转换。

- 可以指定底层类型
  - 例：

enum class Type: char { General, Light, Medium, Heavy};

例 4-9 枚举类举例

```
#include<iostream>
using namespace std;
enum class Side{ Right, Left };
enum class Thing{ Wrong, Right };  //不冲突
int main()
{
Side s = Side::Right;
Thing w = Thing::Wrong;
cout << (s == w) << endl;  //编译错误，无法直接比较不同枚举类
return 0;
}
```

小结

- 主要内容
  - 面向对象的基本概念、类和对象的声明、构造函数、析构函数、内联成员函数、复制构造函数、类的组合
- 达到的目标
  - 掌握面向对象的基本概念；
  - 掌握类设计的思想、类和对象声明的语法；
  - 理解构造函数、复制构造函数和析构函数的作用和调用过程，掌握相关的语法；
  - 理解内联成员函数的作用，掌握相关语法；
  - 理解类的组合在面向对象设计中的意义，掌握类组合的语法。
  - 了解枚举类