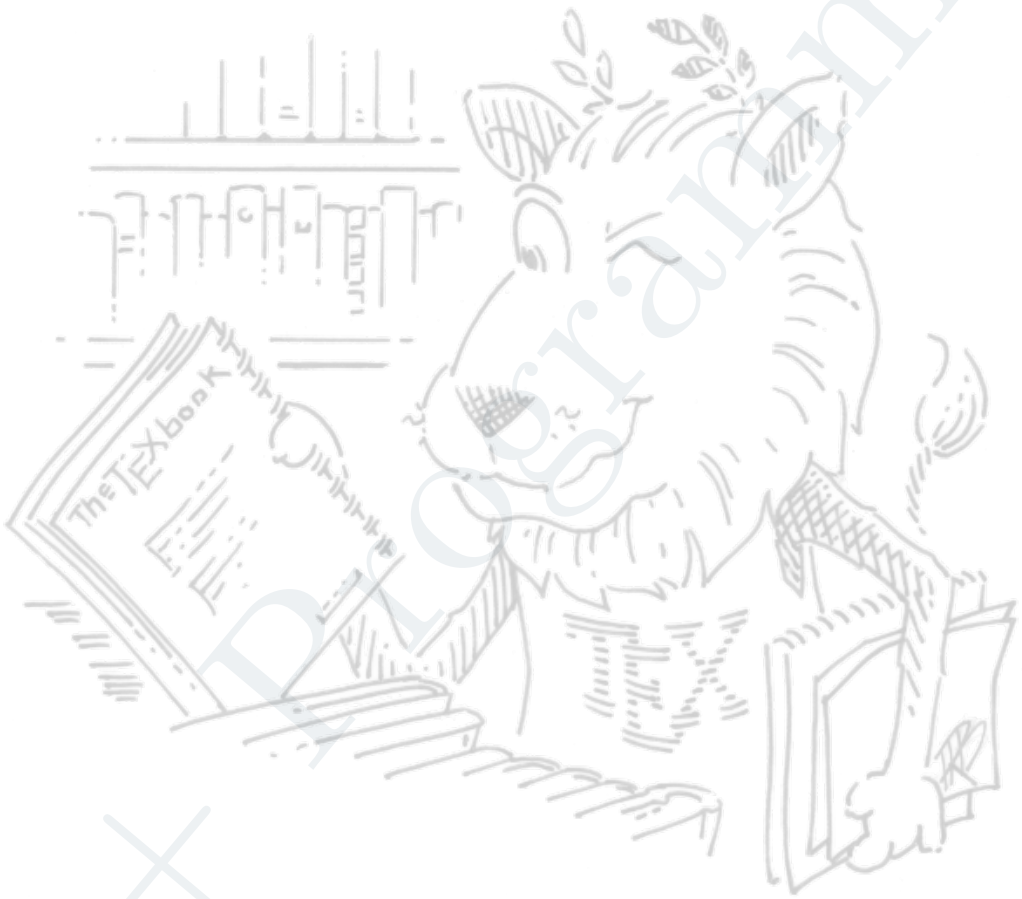
《C++ 程序设计》OJ4（参考答案）

VCG

2023 年 5 月 27 日



1 TableTennisPlayer 类

【描述】

编写 TableTennisPlayer 类和 RatedPlayer 类（RatedPlayer 类继承 TableTennisPlayer 类），其中 TableTennisPlayer 类的定义如下所示：

```
\begin{lstlisting}[language=C++]
{
private:
    string firstname;
    string lastname;
    bool hasTable;
public:
    TableTennisPlayer(const string &, const string &, bool);
    string FirstName() const;
    string LastName() const;
    bool HasTable() const;
};
\end{lstlisting}
```

实现后，通过以下 main 函数的测试：

```
\begin{lstlisting}[language=C++]
int main()
{
    string firstname, lastname;
    bool hasTable;
    int rating;
    char flag;
    while (cin >> flag)
    {
        if (flag == 'T')
        {
            cin >> firstname >> lastname >> hasTable;
            TableTennisPlayer tp(firstname, lastname, hasTable);
            if (tp.HasTable())
                cout << tp.FirstName() << " " << tp.LastName()
                    << " has a table.\n";
            else
                cout << tp.FirstName() << " " << tp.LastName()
                    << " hasn't a table.\n";
        }
        else if (flag == 'R')
        {
            cin >> firstname >> lastname >> hasTable >> rating;
```

```
            RatedPlayer rp(rating, firstname, lastname, hasTable);
            if (rp.HasTable())
                cout << rp.FirstName() << " " << rp.LastName()
                << " has a table. The rating is " << rp.Rating() << ".\n";
            else
                cout << rp.FirstName() << " " << rp.LastName()
                << " hasn't a table. The rating is " << rp.Rating() << ".\n";
        }
    }
    return 0;
}
\end{lstlisting}
```

\subsection*{【输入描述】}
输入多行，每一行以'T' 或'R' 开头，'T' 表示本行接下来输入一个 TableTennisPlayer 对象的信息，包括 fi

\subsection*{【输出描述】}
一行输入对应一行输出，输出详见 main 函数

\subsection*{【输入样例 1】}
\begin{Verbatim}
T Bill Gates 1

【输出样例 1】

```
Bill Gates has a table.
```

【输入样例 2】

```
R Jike Zhang 0 19000
```

【输出样例 2】

```
Jike Zhang hasn't a table. The rating is 19000.
```

【提示】

main.cpp

```cpp
#include<iostream>
#include<string>

class TableTennisPlayer
{
private:
    std::string firstname;
    std::string lastname;
    bool hasTable;
public:
    TableTennisPlayer(const std::string&, const std::string&, bool);
    std::string FirstName() const;
    std::string LastName() const;
    bool HasTable() const;
};

TableTennisPlayer::TableTennisPlayer(const std::string& fn, const std::string& ln,
    bool ht)
{
    firstname = fn;
    lastname = ln;
    hasTable = ht;
}

std::string TableTennisPlayer::FirstName() const
{
    return firstname;
}

std::string TableTennisPlayer::LastName() const
{
    return lastname;
}

bool TableTennisPlayer::HasTable() const
{
    return hasTable;
}

class RatedPlayer : public TableTennisPlayer
{
private:
    int rating;
public:
    RatedPlayer(int, const std::string&, const std::string&, bool);
    int Rating() const;
};

```

```cpp
RatedPlayer::RatedPlayer(int r, const std::string& fn, const std::string& ln, bool ht)
    : TableTennisPlayer(fn, ln, ht)
{
    rating = r;
}

int RatedPlayer::Rating() const
{
    return rating;
}

int main()
{
    std::string firstname, lastname;
    bool hasTable;
    int rating;
    char flag;
    while(std::cin >> flag)
    {
        if(flag == 'T')
        {
            std::cin >> firstname >> lastname >> hasTable;
            TableTennisPlayer tp(firstname, lastname, hasTable);
            if(tp.HasTable())
                std::cout << tp.FirstName() << " " << tp.LastName() << " has a table.\n";
            else
                std::cout << tp.FirstName() << " " << tp.LastName() << " hasn't a table.\n";
        }
        else if(flag == 'R')
        {
            std::cin >> firstname >> lastname >> hasTable >> rating;
            RatedPlayer rp(rating, firstname, lastname, hasTable);
            if (rp.HasTable())
                std::cout << rp.FirstName() << " " << rp.LastName()
                    << " has a table. The rating is " << rp.Rating() << ".\n";
            else
                std::cout << rp.FirstName() << " " << rp.LastName()
                    << " hasn't a table. The rating is " << rp.Rating() << ".\n";
        }
    }
    return 0;
}
```

2 Person 和 Student

【描述】

实现一个 Person 类，再实现一个 Student 类，要求 Student 类继承 Person 类，通过以下测试：

```
int main()
{
    Person * p;

    p = new Person;
    p->input();
    p->display();
    delete p;

    p = new Student;
    p->input();
    p->display();
    delete p;

    return 0;
}
```

【输入描述】

输入两行。
第一行为一个姓名（不包含空格）；
第二行为一个学号和一个姓名（学号、姓名都不包含空格），学号和姓名之间用空格间隔

【输出描述】

输出为两行。
第一行为一个姓名
第二行为学号和姓名，学号和姓名之间用空格间隔

【输入样例】

```
Mary
001 Mary
```

【输出样例】

```
Mary
001 Mary
```

【提示】

不能修改 main 函数，否则不得分

main.cpp

```cpp
#include<iostream>
#include<string>

class Person
{
protected:
    std::string Name;
public:
    virtual void input()
    {
        std::cin >> Name;
    }
    virtual void display()
    {
        std::cout << Name << std::endl;
    }

};

class Student : public Person
{
private:
    std::string No;
public:
    void input()
    {
        std::cin >> No >> Name;
    }
    void display()
    {
        std::cout << No << " " << Name << std::endl;
    }

};

int main()
{
    Person* p;

    p = new Person;
    p->input();
    p->display();
    delete p;

    p = new Student;
    p->input();
    p->display();
    delete p;
```

```
49
50      return 0;
51  }
```

3 图书商品

【描述】

编写以下两个类，请实现并使得以下的 main 函数成立

```
class Item_base
{  //图书商品
public:
    Item_base(const string & book_ISBN = "", double sales_price = 0.0);
    string Get_ISBN() const;
    virtual double Net_price(int) const;  //返回购买指定数量的图书的总价
    virtual ~Item_base();
protected:
    string ISBN;    //图书序列号
    double price;    //单价
};

class Bulk_Item : public Item_base
{  //根据购买数量打折
public:
    Bulk_Item(const string & book_ISBN = "", double sales_price = 0.0,
        int min_qty = 0, double discount = 0.0);
    double Net_price(int) const;  //返回根据购买数量打折后的总价
private:
    int min_qty;   // 买够这个数量可以打相应的折扣
    double discount;  //折扣
};

int main()
{
    Item_base book("0-001-0001-1", 10.0);
    Bulk_Item bulk1("0-001-0001-1", 10.0, 5, 0.1);
    Bulk_Item bulk2("0-001-0001-1", 10.0, 10, 0.2);

    int num;
    while (cin >> num) {

        cout << bulk1.Get_ISBN() << "\t" << num << "\t";

        Item_base * p;
        if (num >= 10) p = &bulk2;
        else if (num >= 5) p = &bulk1;
        else p = &book;

        cout << p->Net_price(num) << "\n";
    }
    return 0;
}
```

【输入描述】

需要购买的图书的数量

【输出描述】

输出购买的图书的 ISBN，它的数量以及总的价格。(具体见 main 中输出)

【输入样例】

```
2
6
11
```

【输出样例】

```
0-001-0001-1    2    20
0-001-0001-1    6    54
0-001-0001-1    11   88
```

【提示】

main.cpp

```cpp
#include<iostream>
#include<string>

class Item_base
{  //图书商品
public:
    Item_base(const std::string& book_ISBN = "", double sales_price = 0.0);
    std::string Get_ISBN() const;
    virtual double Net_price(int) const;  //返回购买指定数量的图书的总价
    virtual ~Item_base();
protected:
    std::string ISBN;    //图书序列号
    double price;    //单价
};

Item_base::Item_base(const std::string& book_ISBN, double sales_price) {
    ISBN = book_ISBN;
    price = sales_price;
}

std::string Item_base::Get_ISBN() const {
    std::string IS = ISBN;
    return IS;
}

Item_base::~Item_base() {}

double Item_base::Net_price(int n) const
{
    double sum = n * price;
    return sum;
}

class Bulk_Item : public Item_base
{  //根据购买数量打折
public:
    Bulk_Item(const std::string& book_ISBN = "", double sales_price = 0.0, int
        min_qty = 0, double discount = 0.0);
    double Net_price(int) const;  //返回根据购买数量打折后的总价
private:
    int min_qty;    // 买够这个数量可以打相应的折扣
    double discount;  //折扣
};

Bulk_Item::Bulk_Item(const std::string& book_ISBN, double sales_price, int min_qty,
    double discount)
    : Item_base(book_ISBN, sales_price)
{
```

```
47      this->min_qty = min_qty;
48      this->discount = discount;
49  }
50
51  double Bulk_Item::Net_price(int n) const
52  {
53      double sum = n * price * (1 - discount);
54      return sum;
55  }
56
57  int main()
58  {
59      Item_base book("0-001-0001-1", 10.0);
60      Bulk_Item bulk1("0-001-0001-1", 10.0, 5, 0.1);
61      Bulk_Item bulk2("0-001-0001-1", 10.0, 10, 0.2);
62
63      int num;
64      while(std::cin >> num) {
65          std::cout << bulk1.Get_ISBN() << "\t" << num << "\t";
66          Item_base * p;
67          if(num >= 10) p = &bulk2;
68          else if(num >= 5) p = &bulk1;
69          else p = &book;
70          std::cout << p->Net_price(num) << "\n";
71      }
72      return 0;
73  }
```

4 Vehicle

【描述】

设计一个抽象类 Vehicle，由它派生出类 Car 和类 Truck，类 Car 包含名称、颜色和载客数三个数据成员，类 Truck 包含名称、颜色和载重量三个数据成员。
使用如下函数测试你的程序:

```
int main()
{
    char type;
    while (cin >> type)
    {
        Vehicle *p;
        string name, color;
        cin >> name >> color;
        if (type == 'C')
        {
            int pas;
            cin >> pas;
            Car car(name, color, pas);
            p = &car;
            p->display();
        }
        else if (type == 'T')
        {
            double cap;
            cin >> cap;
            Truck truck(name, color, cap);
            p = &truck;
            p->display();
        }
    }
    return 0;
}
```

【输入描述】

多组输入。
每组输入的开头是'C' 或者'T'，代表此时我们要输入的车的信息分别为 Car 和 Truck.
当输入的是 Car 的时候，我们要输入它的名称，颜色和载客数;
当输入的是 Truck 的时候，我们要输入它的名称，颜色和载重量。
然后用基类 Vehicle 指针来指向派生类对象, 从而实现不同类中 display() 函数的多态性。

【输出描述】

根据不同车种类，输出不同信息，具体见样例输出。

【输入样例】

```
C Benz black 3
T Dongfeng white 8.5
```

【输出样例】

```
Benz car, black, passenger volume: 3
Dongfeng truck, white, cargo capacity: 8.5(t)
```

【提示】

Vehicle 中可包含名称和颜色数据成员，并且有纯虚函数以提供接口完成信息的显示；
在派生类 Car 和 Truck 中根据需要实现纯虚函数以及添加成员。

main.cpp

```cpp
#include<iostream>
#include<string>

class Vehicle
{
protected:
    std::string name;
    std::string color;
public:
    Vehicle(const std::string& nn, const std::string& cc) : name(nn), color(cc){}
    virtual void display() = 0;
};

class Car: public Vehicle
{
protected:
    int passenger;
public:
    Car(const std::string& nn, const std::string& cc, int n) : Vehicle(nn, cc)
    {
        passenger = n;
    }
    void display()
    {
        std::cout << name << " car, " << color << ", passenger volume: " <<
            passenger << std::endl;
    }
};

class Truck: public Vehicle
{
protected:
    double weight;
public:
    Truck(const std::string& nn, const std::string& cc, double n) : Vehicle(nn, cc)
    {
        weight = n;
    }
    void display()
    {
        std::cout << name << " truck, " << color << ", cargo capacity: " << weight
            << "(t)" << std::endl;
    }
};

int main()
{
    char type;
```

```
47      while(std::cin >> type)
48      {
49          Vehicle *p;
50          std::string name, color;
51          std::cin >> name >> color;
52          if(type == 'C')
53          {
54              int pas;
55              std::cin >> pas;
56              Car car(name, color, pas);
57              p = &car;
58              p->display();
59          }
60          else if(type == 'T')
61          {
62              double cap;
63              std::cin >> cap;
64              Truck truck(name, color, cap);
65              p = &truck;
66              p->display();
67          }
68      }
69      return 0;
70  }
```

5 面积与体积

【描述】

编写程序计算长方体、圆柱体和球的表面积和体积。要求先定义一个抽象类 Shape 如下：

```
class Shape
{
public:
    Shape() {}
    virtual ~Shape() {}
    virtual double Area() = 0;
    virtual void Input() = 0;
    virtual double Volume() = 0;
};
```

使用 Shape 类派生出长方体类、圆柱体类、球类，在这些类里分别实现继承的纯虚函数。使用如下代码测试运行。

```
void work(Shape *s)
{
    s->Input();
    cout << s->Area() << " " << s->Volume() << endl;
}

int main()
{
    char c;
    while (cin >> c)
    {
        switch (c)
        {
        case 'y':
        {
            Shape *s = new Cylinder();
            work(s);
            delete s;
            break;
        }
        case 'c':
        {
            Shape *s = new Cuboid();
            work(s);
            delete s;
            break;
        }
        case 'q':
        {
            Shape *s = new Ball();
            work(s);
            delete s;
            break;
```

```
34          }
35          default:
36              break;
37          }
38      }
39      return 0;
40  }
```

【输入描述】

输入包含多行。
首先是一个字符’c’,’y’,’q’，分别表示输入长方体、圆柱体或球的信息。
接下来是对应的输入。

【输出描述】

每行输入对应一行输出，表示该形状的表面积和体积，以空格分隔

【输入样例】

```
c 3 4 5
y 3 5
q 5
```

【输出样例】

```
94 60
150.796 141.372
314.159 523.599
```

【提示】

main.cpp

```cpp
#include<iostream>
#include<cmath>

const double PI = acos(-1.0);

class Shape
{
public:
    Shape() {}
    virtual ~Shape() {}
    virtual double Area() = 0;
    virtual void Input() = 0;
    virtual double Volume() = 0;
};

class Cylinder: public Shape
{
private:
    double r, h;
public:
    Cylinder() : Shape()
    {
        r = 0;
    }
    double Area()
    {
        double ba = PI * r * r;
        double bc = 2 * PI * r;
        return 2 * ba + bc * h;
    }
    void Input()
    {
        std::cin >> r >> h;
    }
    double Volume()
    {
        double ba = PI * r * r;
        return ba * h;
    }
    ~Cylinder() {}
};

class Cuboid: public Shape
{
private:
    double a, b, c;
public:
    Cuboid() : Shape()
```

```
49      {
50          a = b = c = 0;
51      }
52      double Area()
53      {
54          return 2 * a * b + 2 * a * c + 2 * b * c;
55      }
56      double Volume()
57      {
58          return a * b * c;
59      }
60      void Input()
61      {
62          std::cin >> a >> b >> c;
63      }
64      ~Cuboid() {}
65  };
66
67  class Ball: public Shape
68  {
69  private:
70      double r;
71  public:
72      Ball() : Shape()
73      {
74          r = 0;
75      }
76      double Area()
77      {
78          return 4 * PI * r * r;
79      }
80      double Volume()
81      {
82          return 4.0 / 3 * PI * r * r * r;
83      }
84      void Input()
85      {
86          std::cin >> r;
87      }
88      ~Ball() {}
89  };
90
91  void work(Shape *s)
92  {
93      s->Input();
94      std::cout << s->Area() << " " << s->Volume() << std::endl;
95  }
96
97  int main()
98  {
```

```
99      char c;
100     while (std::cin >> c)
101     {
102         switch (c)
103         {
104         case 'y':
105         {
106             Shape *s = new Cylinder();
107             work(s);
108             delete s;
109             break;
110         }
111         case 'c':
112         {
113             Shape *s = new Cuboid();
114             work(s);
115             delete s;
116             break;
117         }
118         case 'q':
119         {
120             Shape *s = new Ball();
121             work(s);
122             delete s;
123             break;
124         }
125         default:
126             break;
127         }
128     }
129     return 0;
130 }
```

6 Animal

【描述】

不同的动物既有共性也有个性。鸟类会飞，鱼会游泳。请设计一个类层次结构表示。并通过以下测试

```
int main()
{
    Animal *animal;

    string  type, color;
    bool Osteichthyes, daytime;

    cin >> type >> color >> Osteichthyes;
    Fish fish(type, color, Osteichthyes);
    fish.Print();
    animal = &fish;
    animal->Print();

    cin >> type >> color >> daytime;
    Bird bird(type, color, daytime);
    bird.Print();
    animal = &bird;
    animal->Print();

    return 0;
}
```

【输入描述】

鱼类型鱼的颜色是否硬骨
鸟类型鸟的颜色是否白天活动

【输出描述】

见测试样例

【输入样例】

```
chub white 1
swallow black 1
```

【输出样例】

```
type: chub, color: white, Osteichthyes: 1
type: chub, color: white, Osteichthyes: 1
type: swallow, color: black, daytime: 1
type: swallow, color: black, daytime: 1
```

【提示】

标点符号是半角字符，:, 后面有一个空格

main.cpp

```cpp
#include<iostream>
#include<string>

class Animal
{
protected:
    std::string type;
    std::string color;
public:
    Animal(const std::string& type ="", const std::string& color = "")
    {
        this->type = type;
        this->color = color;
    }
    virtual void Print()const
    {
        std::cout << "type: " << type << ", color: " << color;
    }
};

class Fish : public Animal
{
private:
    bool Osteichthyes;
public:
    Fish(const std::string& type = "", const std::string& color = "", bool
        Osteichthyes = true) :
        Animal(type, color)
    {
        this->Osteichthyes = Osteichthyes;
    }
    void Print()const
    {
        Animal::Print();
        std::cout << ", Osteichthyes: " << Osteichthyes << std::endl;
    }
};

class Bird : public Animal
{
private:
    bool daytime;
public:
    Bird(const std::string& type = "", const std::string& color = "", bool daytime
       = true) :
        Animal(type, color)
    {
        this->daytime = daytime;
```

```
47      }
48      void Print()const
49      {
50          Animal::Print();
51          std::cout << ", daytime: " << daytime << std::endl;
52      }
53  };
54
55  int main()
56  {
57      Animal *animal;
58      std::string type, color;
59      bool Osteichthyes, daytime;
60
61      std::cin >> type >> color >> Osteichthyes;
62      Fish fish(type, color, Osteichthyes);
63      fish.Print();
64      animal = &fish;
65      animal->Print();
66
67      std::cin >> type >> color >> daytime;
68      Bird bird(type, color, daytime);
69      bird.Print();
70      animal = &bird;
71      animal->Print();
72      return 0;
73  }
```

7 Person student teacher

【描述】

编写程序，定义一个基类 Person，包含 name 和 age 两个数据成员；再由它派生出学生类 Student 和教师类 Teacher，其中学生类添加学号 no 数据，教师类添加职称 title 数据；要求每个类均有构造函数、析构函数和显示数据的函数。
使用以下函数运行你的程序:

```
int main()
{
    Person p("Mary", 18);
    p.show();
    Student s("Tom", 20, 10001);
    s.show();
    Teacher t("John", 30, "Associate Professor");
    t.show();
    return 0;
}
```

【输入描述】

无

【输出描述】

见测试样例

【输入样例】

无

【输出样例】

```
Constructor of Person
Name: Mary, Age: 18
Constructor of Person
Constructor of Student
Name: Tom, Age: 20, No: 10001
Constructor of Person
Constructor of Teacher
Name: John, Age: 30, Title: Associate Professor
Destructor of Teacher
Destructor of Person
Destructor of Student
Destructor of Person
Destructor of Person
```

【提示】

标点符号是半角字符，:, 后面有一个空格

main.cpp

```cpp
#include<iostream>
#include<string>

class Person
{
protected:
    std::string name;
    int age;
public:
    Person(const std::string& name, int age)
    {
        this->name = name;
        this->age = age;
        std::cout << "Constructor of Person\n";
    }
    ~Person()
    {
        std::cout << "Destructor of Person\n";
    }
    void show()
    {
        std::cout << "Name: " << name << ", " << "Age: " << age << std::endl;
    }
};

class Student : public Person
{
private:
    int no;
public:
    Student(const std::string& nname, int aage, int number) : Person(nname, aage),
        no(number)
    {
        std::cout << "Constructor of Student\n";
    }
    ~Student()
    {
        std::cout << "Destructor of Student\n";
    }
    void show()
    {
        std::cout << "Name: " << name << ", " << "Age: " << age << ", " << "No: "
            << no << std::endl;
    }
};

class Teacher : public Person
{
```

```
47  private:
48      std::string title;
49  public:
50      Teacher(const std::string& n, int a, const std::string& ti) :Person(n, a),
            title(ti)
51      {
52          std::cout << "Constructor of Teacher\n";
53      }
54      ~Teacher()
55      {
56          std::cout << "Destructor of Teacher\n";
57      }
58      void show()
59      {
60          std::cout << "Name: " << name << ", " << "Age: " << age << ", " << "Title:
                " << title << std::endl;
61      }
62  };
63
64  int main()
65  {
66      Person p("Mary", 18);
67      p.show();
68      Student s("Tom", 20, 10001);
69      s.show();
70      Teacher t("John", 30, "Associate Professor");
71      t.show();
72      return 0;
73  }
```