实验八　　多态：虚函数

1　实验目的

（1）学习为什么要使用虚函数；

（2）学习如何声明函数为虚函数；

（3）学习如何声明异类数组（基类指针分别指向不同的派生类对象）；

（4）学习如何使用虚函数和异类数组实现多态调用。

2　实验内容

2.1 模拟银行帐户管理程序

（1）问题描述

创建一个银行账户的继承层次，表示银行的所有客户账户。所有的客户都能在他们的银行账户存钱，取钱，并且账户还可以分成更具体的类型。例如，存款账户 SavingsAccount 依靠存款生利，支票账户 CheckingAccount 会对每笔交易（即存款或取款）收取费用。

创建一个类层次，以 Account 作为基类，SavingsAccount 和 CheckingAccount 作为派生类。

基类 Account 应该包括一个 double 类型的数据成员 balance，表示账户的余额。该类应当提供三个成员函数。成员函数 credit 可以向当前余额加钱；成员函数 debit 负责从账户中取钱，并且保证账户不会被透支。如果提取金额大于账户金额，函数将保持 balance 不变，并打印信息“Debit amount exceeded account balance”；成员函数 getBalance 则返回当前 balance 的值。

派生类 SavingsAccount 不仅继承了基类 Account 的功能，而且应提供一个附加的 double 类型数据成员 interestRate 表示这个账户的比率（百分比）。SavingsAccount 的构造函数应接受初始余额值和初始利率值，还应提供一个 public

成员函数calculateInterest，返回代表账户的利息的一个double值，这个值是balance和 interestRate 的乘积。类 SavingsAccount 应继承成员函数 credit 和 debit，不需要重新定义。

派生类 CheckingAccount 不仅继承基类 Account 的功能，还提供一个附加的 double 类型数据成员表示每笔交易的费用。CheckingAccount 的构造函数应接受初始余额值和交易费用值。类 CheckingAccount 需要重新定义成员函数 credit 和 debit，当每笔交易完成时，从 balance 中减去每笔交易的费用。重新定义这些函数时应分别使用基类 Account 的相关函数来执行账户余额的更新。CheckingAccount 的 debit 函数只有当钱被成功提取时（即提取金额不超过账户余额时）才应收取交易费。

提示：定义 Account 的 debit 函数使它返回一个 bool 类型值，表示钱是否被成功提取。然后利用该值决定是否需要扣除交易费。如果取款或存款后，账户的余额小于每笔交易的费用，则废弃这次交易，使账户余额恢复到取款或存款之前的值，并打印“Transaction fee exceeded account balance while debiting”或“Transaction fee exceeded account balance while crediting”。

（2）问题要求

要求将每个类的定义和实现分开在不同的文件里，并严格按照上述名称定义成员变量和成员函数，所有类的成员变量均定义为 private 的。当这个继承层次中的类定义完毕后，编写一个主程序，能够多态地调用不同账户的成员函数。

```
int main() {
   Account *accounts[3];
   accounts[0] = new SavingsAccount(100, 3);   //余额100元，利息3%
   accounts[1] = new CheckingAccount(100, 5);  //余额100元，交易费5元
   accounts[2] = new CheckingAccount(50, 5);   //余额50元，交易费5元

   for (int i = 0; i < 3; i++) {
      cout << "第" << i + 1 << "次循环的结果：" << endl;
      accounts[i]->debit(200);   //借款200元
      accounts[i]->debit(40);
      accounts[i]->credit(50);   //存款50元
      accounts[i]->debit(49);
      accounts[i]->debit(43);
      accounts[i]->credit(1);
      //将Account指针强制转换为SavingAccount指针
      SavingsAccount *derivedPtr =
            dynamic_cast<SavingsAccount *>(accounts[i]);
      if(derivedPtr != NULL)   //如果类型兼容，转换成功
         derivedPtr->credit(derivedPtr->calculateInterest());
      cout << fixed << setprecision(2);   //使用定点数格式，2位小数部分
      cout << "账户的余额为：" << accounts[i]->getBalance() << endl;
   }
}
```

（3）程序执行结果

程序执行结果如下：

```
第1次循环的结果：
Debit amount exceeded account balance
账户的余额为：19.57
第2次循环的结果：
Debit amount exceeded account balance
Transaction fee exceeded account balance while debiting
账户的余额为：42.00
第3次循环的结果：
Debit amount exceeded account balance
Transaction fee exceeded account balance while debiting
Transaction fee exceeded account balance while crediting
账户的余额为：2.00
```

2.2 继续完善停车场程序

（1）问题描述

请根据题目要求完成简单的停车场管理程序。

1．停车场（Park）有 N 个停车位（Space），每个停车位可以停放不同类型的汽车（Automobile），包括卡车（Truck）、轿车（Car）、公交车（Bus），但同一时刻一个停车位只能停放 0 或 1 辆汽车。如果没有空余停车位，显示提示信息，但不会为车辆安排停车位。

2．程序模拟车辆停车的情况：新来车辆时如果有空位，按顺序为该车分配停车位；车辆开走时，应交纳停车费。

3．停车场可以显示当前停放的车辆的车牌号码，以及当前的全部停车费收入（income）。

4．定义汽车基类 Automobile，包括车牌号码（字符串）成员数据。

5．定义派生类 Truck、Car、Bus。这些车辆除了拥有车牌号码之外，还各自拥有不同的属性。Truck 还包括载重量属性（浮点数，单位吨）；Car 还拥有品牌属性（字符串），Bus 还包括核定载员数量（整型）。

此外，每个派生类中要实现 pay()函数，用于显示车辆信息并交纳停车费。其中，Truck 收费 3 元/次，Car 收费 1 元/次，Bus 收费 2 元/次。

（2）问题要求

编写程序，测试上述所要求的各种功能。要求创建新的工程项目 ParkManager，添加必要的源文件和头文件，并在程序适当的位置中编写注释。

```
class Automobile {   // 汽车类
public:
   void enter(Park *park);
   void leave(Park *park);
protected:
   virtual void pay(Park &park) = 0;  // 向停车场支付停车费，由派生类实现
};
```

```
class Park {};          // 停车场类

class Car: public Automobile {
protected:
   void pay(Park *park);
}

void Automobile::leave(park *park) {
   park.reclaimSpace(this);    // 让停车场收回停车位
   pay(park);                  // 向支付支付停车费，由派生类实现本方法
}

void Car::pay(Park *park) {
   cout << "车牌号离开停车场，缴纳停车费1元" << endl;
   park->getPaid(1);
}

int main() {
   cout << "请输入停车位数量：";
   cin >> N;// 输入停车位数量，此处输入2

   Park park(N);// 创建一个停车场对象

   Automobile *auto1 = new Car("鲁B-12345", "奥迪A6");  // 创建轿车对象
   Automobile *auto2 = new Truck("鲁B-23456", 15);      // 创建卡车对象
   Automobile *auto3 = new Bus("鲁B-34567", 50);        // 公交车对象
   Automobile *auto4 = new Car("鲁B-45678", "宝马320");// 创建轿车对象

   auto1.enter(&park);   // car进入停车场，分配停车位
   auto2.enter(&park);   // truck进入停车场，分配车位
   auto1.leave(&park);   // car离开停车场，缴纳停车费
   auto3.enter(&park);   // bus进入停车场，分配车位

   /* 显示当前停放的车辆的车牌号码，以及当前的全部停车费收入*/
   park.showInfo();

   auto4.enter(&park);      // car进入停车场，分配停车位
   // car进入停车场，分配停车位。因为没有空余停车位，所以无法分配
```

```
auto3.leave(&park);  // bus离开停车场，缴纳停车费
auto2.leave(&park);  // truck离开停车场，缴纳停车费

/* 显示当前停放的车辆的车牌号码，以及当前的全部停车费收入*/
park.showInfo();

return 0;
}
```

（2）程序执行结果

程序执行结果如下：

```
请输入停车位数量：2
鲁B-12345进入停车场，分配停车位
鲁B-23456进入停车场，分配停车位
鲁B-12345离开停车场，缴纳停车费1元
鲁B-34567进入停车场，分配停车位
停车场目前停放了2辆汽车：鲁B-23456，鲁B-34567，共收入1元停车费
无法为鲁B-45678分配停车位
鲁B-34567离开停车场，缴纳停车费2元
鲁B-23456离开停车场，缴纳停车费3元
停车场目前停放了0辆汽车，共收入6元停车费
```