作业:软件测试

选择题

1. （ **B** ）在软件开发的过程中，若能推迟暴露其中的错误，则为修复和改成错误所花费的代价就会降低。

   A. 真

   B. 假

2. （ **A** ）好的测试是用少量的测试用例运行程序，发现被测程序尽可能多的错误。

   A. 真

   B. 假

3. （ **B** ）好的测试用例应能证明软件是正确的。

   A. 真

   B. 假

4. （ **A** ）白盒测试仅仅与程序的内部结构有关，完全可以不考虑程序的功能要求。

   A. 真

   B. 假

5. （ **A** ）等价类划分方法将所有可能的输入数据分成若干部分，然后从每一部分中选取少数具有代表性的数据作为测试用例。

   A. 真

   B. 假

6. 使用独立测试团队的最好理由是（ **C** ）。

   A. 软件开发人员不需要做任何测试

   B. 测试人员在测试开始之前不参与项目

   C. 测试团队将更彻底地测试软件

   D. 开发人员与测试人员之间的争论会减少

7. 类的行为应该基于（ **D** ）进行测试。

   A. 数据流图

   B. 用例图

   C. 对象图

   D. 状态图

8. 下面的（ **CDE** ）说法是正确的。

   A. 恢复测试是以各种方式迫使软件失效从而检测软件是否能够继续执行的一种系统

   B. 安全测试是检测系统中的保护机制是否可以保护系统免受非正常的攻击

C. 压力测试是在极限环境中使用系统时施加在用户上的压力

D. 功能测试是根据软件需求规格说明和测试需求列表，验证产品的功能实现是否符合需求规格

E. 安装测试是保证应用程序能够被成功地安装

作业

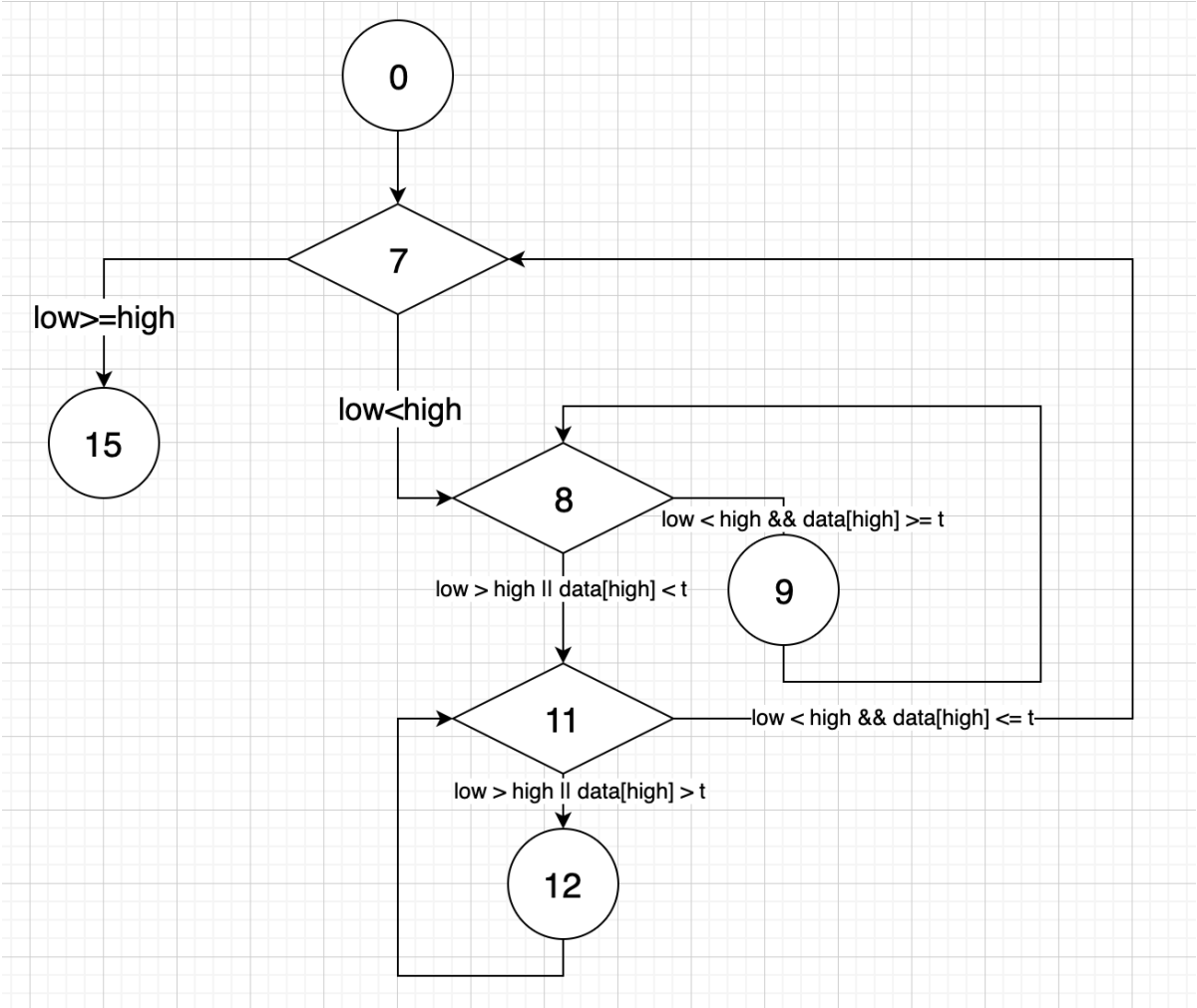
1. 理解下面的程序结构，请使用基本路径方法设计该程序测试用例

```c
#include <stdio.h>
int partition(int *data, int low, int high)
{
  int t = 0;

  t = data[low];
  while (low < high) {
    while (low < high && data[high] >= t)
      high--;
    data[low] = data[high];
    while(low < high && data[low] <= t)
      low++;
    data[high] = data[low];
  }
  data[low] = t;

  return low;
}
```

该程序结构的主要功能是快速排序的一遍。输入一个数组data和要排序的范围的上界和下界，返回的是排序前第一个元素在排序后的数组中的序号。

在具体的实践中，一个不太复杂的程序的路径也都会是一个庞大的数字，要在测试中完全覆盖到所有的路径是不现实的。为了解决这一问题，一般会把覆盖的路径压缩到一定限度内，例如循环体只执行一次。

上述代码的控制流图如下：



由控制流图可知，该流图中共有9（边）-7（点）+2=4个区域，因此确定4个路径如下：

路径1:0-7-15

路径2:0-7-8-9-8-...

路径3:0-7-8-11-12-11-...

路径4:0-7-8-11-12-11-7-...

因此设计测试用例如下：

序号	输入参数	期望输出
1	[3,9,6], 2, 0	2([3,9,6])
2	[3,9,6], 0, 1	0([3,9,6])
3	[9,6,3], 0, 2	2([3,6,9])
4	[6,9,3],0,2	1([3,6,9])

2. 请使用等价类划分和边界值分析的方法，给出getNumDaysInMonth(int month, int year)方法的测试用例，其中getNumDaysInMonth方法根据给定的月份和年份返回该月份的总天数

   （1）请给出划分的等价类

月的等价类：31天的月份、30天的月份、2月

年的等价类：闰年、非闰年

（2）测试用例设计

序号	输入参数	期望输出
1	month : 7 ,year : 2019	31
2	month : 7 ,year : 2020	31
3	month : 6 ,year : 2019	30
4	month : 6 ,year : 2020	30
5	month : 2 ,year : 2019	28
6	month : 2 ,year : 2020	29
7	month : 6 ,year : 0	无效
8	month : 2 ,year : -1	无效
9	month : 0 ,year : 2020	无效
10	month : 13 ,year : 2020	无效

3. 现在要对一个咨询公司的薪酬支付软件进行功能测试，该软件的规格说明如下：

   对于每周工作超过40小时的咨询人员，前40小时按照他们的小时薪酬计算，多余的小时数按照双倍小时薪酬计算；对于每周工作不足40小时的咨询人员，按照特们的小时薪酬计算，并生成一份缺勤报告；对于每周工作超过40小时的长期工作人员，直接按照他们的固定工资计算。

   （1）使用决策表描述上述的薪酬支付规则

规则	1	2	3	4
工作不足40小时	T	F	F	T
是长期工作人员	F	T	F	T
动作				
前40小时按照小时薪酬计算	T		T	
多余小时按照双倍小时薪酬计算			T	
按照固定工资计算		T		
生成缺勤报告	T			
题干未给出计算方式				T

   （2）根据上述决策表，使用下表列出自己设计的测试用例

序号	输入参数	期望输出
1	时薪：40，固定工资：2000，工作时间：30，长期工作人员：F	薪酬：1200，缺勤报告
2	时薪：40，固定工资：2000，工作时间：60，长期工作人员：F	薪酬：3200
3	时薪：40，固定工资：2000，工作时间：60，长期工作人员：T	薪酬：2000
4	时薪：40，固定工资：2000，工作时间：-1，长期工作人员：F	无效
5	时薪：-1，固定工资：2000，工作时间：100，长期工作人员：F	无效
6	时薪：20，固定工资：-1，工作时间：40，长期工作人员：T	薪酬：2000

4. 下面给出的是购买车票（Purchase Ticket）用例的正常流，除此之外还应包括无零钱找（NoChange）、缺票（OutofOrder）、超时（TimeOut）和取消（Cancel）等四个备选流，请结合场景法和其他方法设计测试用例

用例名称	购买车票
前置条件	乘客站在售票机前，有足够的钱买车票。
正常流	1. 乘客选择需要达到的地点，如果按下了多个地点按钮，售票机只考虑最后一次按下的地点。 2. 售票机显示出应付款数。 3. 乘客投入钱。 4. 如果乘客在投入足够的钱之前选择了新的地点，售票机应该把所有的钱退还乘客。 5. 如果乘客投入的钱比应付款多，售票机应该退出多余的零钱。 6. 售票机给出车票。 7. 乘客拿走找零和车票。
后置条件	乘客买到了他选择的车票。

场景设计：

场景名称	流程
PurchaseTicket	正常流
NoChange	正常流、备选流1
OutofOrder	正常流、备选流2
TimeOut	正常流、备选流3
Cancel	正常流、备选流4

要求：使用下表列出自己设计的测试用例

测试用例编号	1
测试标题	成功购买车票
预置条件	乘客有足够的钱购买车票、售票机有足够的零钱找零、售票机不缺票
操作步骤	乘客选择需要达到的地点，如果按下了多个地点按钮，售票机只考虑最后一次按下的地点=>乘客按照售票机显示投入足够的钱=>乘客取走车票和找零
预期输出	乘客成功购买车票

测试用例编号	2
测试标题	成功购买车票
预置条件	乘客有足够的钱购买车票、售票机没有足够的零钱找零
操作步骤	乘客选择到达地点=>乘客按照售票机显示投入足够的钱=>乘客看到售票机显示无零钱后取走所有欠款
预期输出	因售票机没有足够零钱找零，乘客购票失败

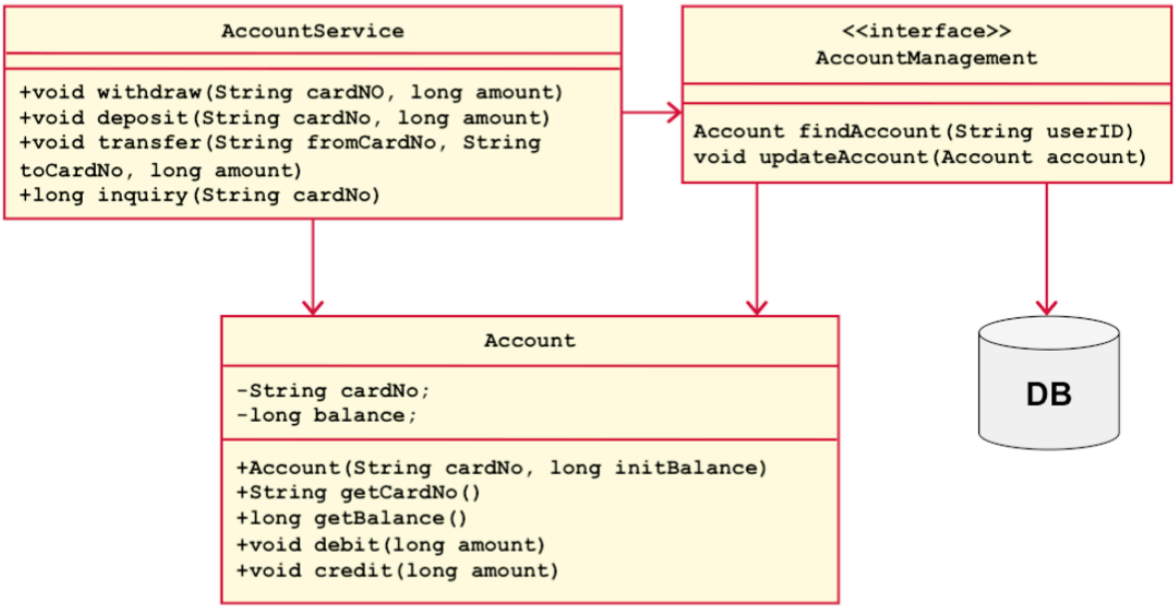
测试用例编号	3
测试标题	售票机缺票
预置条件	乘客有足够的钱购买车票、售票机没有足够的车票
操作步骤	乘客选择到达地点=>乘客按照售票机显示投入足够的钱=>乘客看到售票机显示缺票后取走所有欠款
预期输出	因售票机缺票，乘客购票失败

测试用例编号	4
测试标题	售票超时
预置条件	乘客有足够的钱购买车票、售票机有足够的零钱找零、售票机不缺票
操作步骤	乘客选择到达地点=>乘客未按照售票机显示投入足够的钱=>乘客看到售票超时后取回已付款
预期输出	因售票超时，乘客购票失败

测试用例编号	5
测试标题	售票超时
预置条件	乘客有足够的钱购买车票、售票机有足够的零钱找零、售票机不缺票
操作步骤	乘客选择到达地点=>乘客未按照售票机显示一定的钱=>乘客在售票成功之前取消=>乘客看到售票机显示取消交易后取回已付款
预期输出	因取消交易，乘客购票失败

作业

下面的UML图显示了银行卡账户的服务功能：



要求：

（1）请使用Java或C++语言和测试驱动开发方法编写上述程序，并进行单元测试和代码覆盖分析；

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

#define ASIZE 10

using namespace std;

class Account
{
private:
    string cardNo;
    long balance;
public:
    Account()
    {
        this->cardNo = "";
        this->balance = 0;
    };
    Account(string cardNo, long initBalance)
    {
        this->cardNo = cardNo;
        this->balance = initBalance;
    };
    ~Account(){};

    string getCardNo()
    {
        return this->cardNo;
```

```
    };

    long getBalance()
    {
        return this->balance;
    };

    void setCardNo(string cardNo)
    {
        this->cardNo = cardNo;
    };

    void setBalance(long balance)
    {
        this->balance = balance;
    };
};

class AccountManagement
{
private:
    Account db[ASIZE];//假定数据库大小为10
    long getBalance(string userID)
    {
        for (int i = 0; i < ASIZE; i++)
        {
            if (userID == this->db[i].getCardNo())
            {
                return this->db[i].getBalance();
            }
        }
        return -1;
    }

public:
    AccountManagement()
    {
        this->db[0].setCardNo("1");
        this->db[0].setBalance(1000);
        this->db[1].setCardNo("2");
        this->db[1].setBalance(2000);
        this->db[2].setCardNo("3");
        this->db[2].setBalance(3000);
    };
    ~AccountManagement(){};

    Account findAccount(string userID)
    {
        Account tmp(userID, this->getBalance(userID));
```

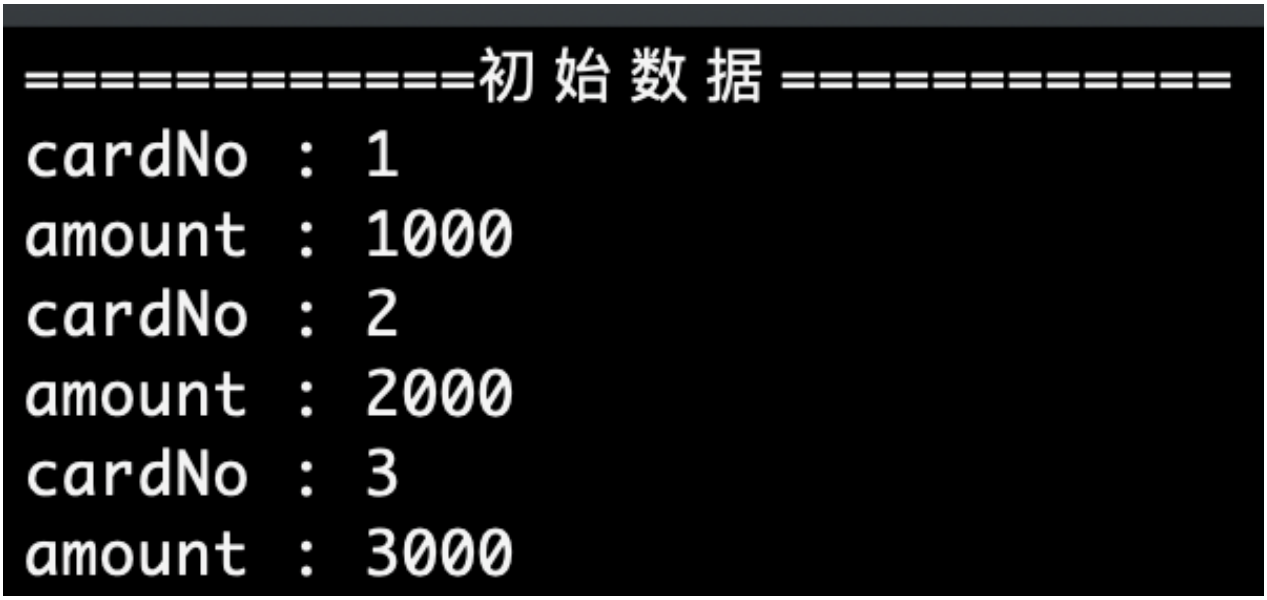
```
        return tmp;
    };

    void updateAccount(Account account)
    {
        for (int i = 0; i < ASIZE; i++)
        {
            if (account.getCardNo() == this->db[i].getCardNo())
            {
                this->db[i].setBalance(account.getBalance());
            }
        }
    };
};

class AccountService
{
public:
    AccountManagement management;

    AccountService(){};
    ~AccountService(){};

    void withdraw(string cardNo, long amount)
    {
        Account account = this->management.findAccount(cardNo);
        if (account.getBalance() != -1 && amount >= 0 &&
account.getBalance() - amount >= 0)
        {
            account.setBalance(account.getBalance() - amount);
            this->management.updateAccount(account);
        }
    };

    void deposit(string cardNo, long amount)
    {
        Account account = this->management.findAccount(cardNo);
        if (account.getBalance() == -1)
        {
            if (amount >= 0)
            {
                account.setBalance(amount);
                this->management.updateAccount(account);
            }
        }
        else
        {
            if (amount >= 0)
            {
```

```
                account.setBalance(account.getBalance() + amount);
                this->management.updateAccount(account);
            }
        }
    };

    void transfer(string fromCardNo, string toCardNo, long amount)
    {
        Account fa = this->management.findAccount(fromCardNo);
        Account ta = this->management.findAccount(toCardNo);
        if (fa.getBalance() != -1 && ta.getBalance() != -1 && amount >= 0
&& fa.getBalance() - amount >= 0)
        {
            fa.setBalance(fa.getBalance() - amount);
            ta.setBalance(ta.getBalance() + amount);
            this->management.updateAccount(fa);
            this->management.updateAccount(ta);
        }
    };

    long inquiry(string cardNo)
    {
        Account account = this->management.findAccount(cardNo);
        return account.getBalance();
    };
};

void testWithdraw(AccountService *as, int no, string cardNo, long amount,
long forward, string tips)
{
    cout << "测试用例" << no << endl;
    cout << "cardNo : " << cardNo << endl;
    cout << "amount : " << amount << endl;
    cout << "期望结果 : " << tips << endl;
    as->withdraw(cardNo, amount);
    if (as->management.findAccount(cardNo).getBalance() == forward)
    {
        cout << "ACCEPT" << endl;
    }
    else
    {
        cout << "ERROR" << endl;
    }
    cout << endl;
};

void testDeposit(AccountService *as, int no, string cardNo, long amount,
long forward, string tips)
{
```

```
172     cout << "测试用例" << no << endl;
173     cout << "cardNo : " << cardNo << endl;
174     cout << "amount : " << amount << endl;
175     cout << "期望结果 : " << tips << endl;
176     as->deposit(cardNo, amount);
177     if (as->management.findAccount(cardNo).getBalance() == forward)
178     {
179         cout << "ACCEPT" << endl;
180     }
181     else
182     {
183         cout << "ERROR" << endl;
184     }
185     cout << endl;
186 };
187
188 void testTransfer(AccountService *as, int no, string fromCardNo, string
    toCardNo, long amount, long forwardFrom, long forwardTo, string tips)
189 {
190     cout << "测试用例" << no << endl;
191     cout << "fromCardNo : " << fromCardNo << endl;
192     cout << "toCardNo : " << toCardNo << endl;
193     cout << "amount : " << amount << endl;
194     cout << "期望结果 : " << tips << endl;
195     as->transfer(fromCardNo, toCardNo, amount);
196     if (as->management.findAccount(fromCardNo).getBalance() ==
    forwardFrom && as->management.findAccount(toCardNo).getBalance() ==
    forwardTo)
197     {
198         cout << "ACCEPT" << endl;
199     }
200     else
201     {
202         cout << "ERROR" << endl;
203     }
204     cout << endl;
205 };
206
207 void testInquiry(AccountService *as, int no, string cardNo,string tips)
208 {
209     cout << "测试用例" << no << endl;
210     cout << "cardNo : " << cardNo << endl;
211     cout << "期望结果 : " << tips << endl;
212     as->inquiry(cardNo);
213     if (as->management.findAccount(cardNo).getBalance() == as-
    >inquiry(cardNo))
214     {
215         cout << "ACCEPT" << endl;
216     }
```

```
    else
    {
        cout << "ERROR" << endl;
    }
    cout << endl;
};

int main()
{
    AccountService as;

    cout << "============初始数据============" << endl;
    cout << "cardNo : " << "1" << endl;
    cout << "amount : " << 1000 << endl;
    cout << "cardNo : " << "2" << endl;
    cout << "amount : " << 2000 << endl;
    cout << "cardNo : " << "3" << endl;
    cout << "amount : " << 3000 << endl;
    cout << endl;

    cout << "========withdraw单元测试========" << endl;
    testWithdraw(&as, 1, "4", 4000, -1, "账户不存在，取款失败");
    testWithdraw(&as, 2, "3", 4000, 3000, "卡内余额不足，取款失败");
    testWithdraw(&as, 2, "3", -1, 3000, "数据非法，取款失败");
    testWithdraw(&as, 3, "3", 500, 2500, "取款成功");
    cout << endl;

    cout << "========deposit单元测试========" << endl;
    testDeposit(&as, 1, "4", 4000, -1, "账户不存在，存款失败");
    testDeposit(&as, 2, "3", -1, 2500, "数据非法，存款失败");
    testDeposit(&as, 2, "3", 500, 3000, "存款成功");
    cout << endl;

    cout << "========transfer单元测试=======" << endl;
    testTransfer(&as, 1, "4", "3", 4000, -1, 3000, "源账户不存在，转账失败");
    testTransfer(&as, 2, "1", "4", 1000, 1000, -1, "目的账户不存在，转账失
败");
    testTransfer(&as, 3, "1", "2", 2000, 1000, 2000, "转账金额过大，转账失
败");
    testTransfer(&as, 2, "3", "1", -1, 3000, 1000, "数据非法，转账失败");
    testTransfer(&as, 4, "3", "1", 500, 2500, 1500, "转账成功");
    cout << endl;

    cout << "========inquiry单元测试========" << endl;
    testInquiry(&as, 1, "4", "账户不存在，查询失败");
    testInquiry(&as, 1, "1", "查询成功");

    return 0;
}
```

单元测试结果如下：

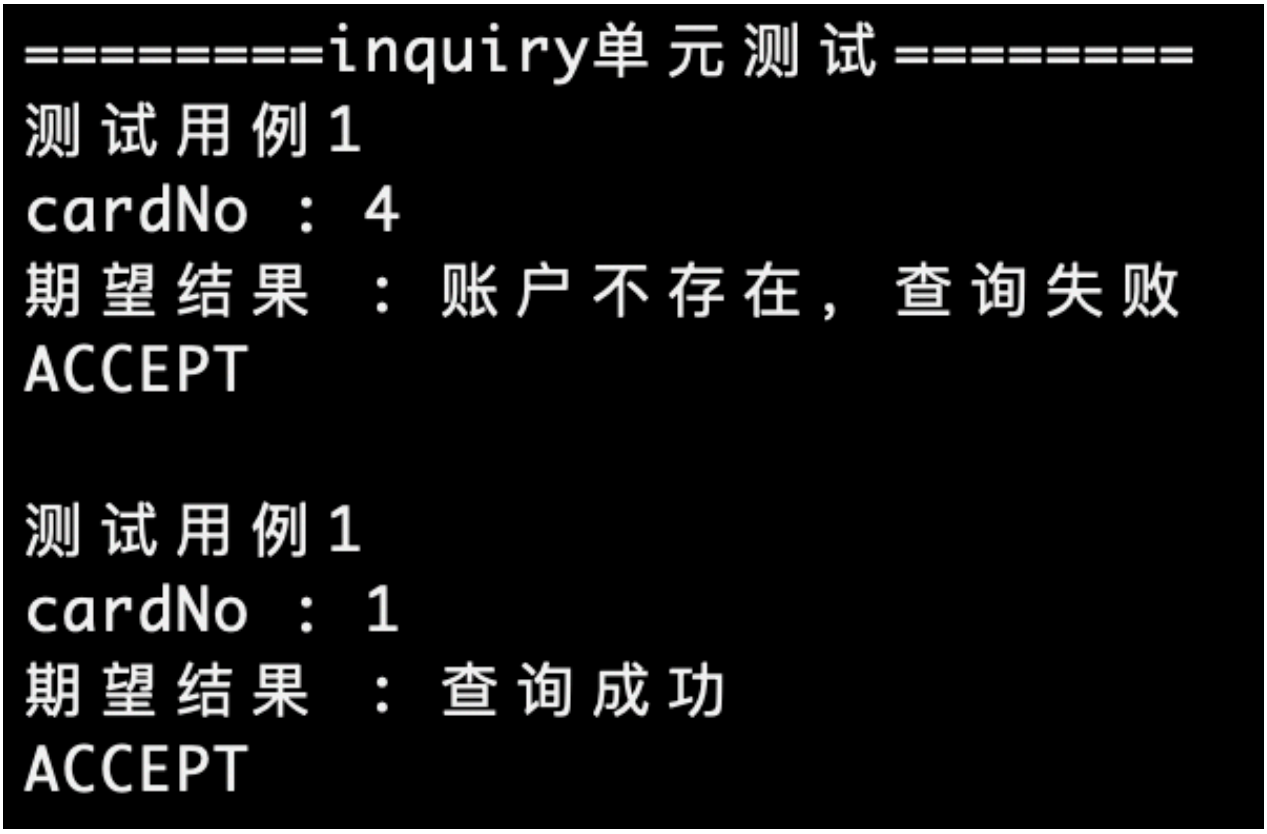

==========初始数据==========
cardNo : 1
amount : 1000
cardNo : 2
amount : 2000
cardNo : 3
amount : 3000

```
=========withdraw单元测试=========
测试用例1
cardNo : 4
amount : 4000
期望结果 : 账户不存在, 取款失败
ACCEPT

测试用例2
cardNo : 3
amount : 4000
期望结果 : 卡内余额不足, 取款失败
ACCEPT

测试用例2
cardNo : 3
amount : -1
期望结果 : 数据非法, 取款失败
ACCEPT

测试用例3
cardNo : 3
amount : 500
期望结果 : 取款成功
ACCEPT
```

```
=========deposit单元测试=========
测试用例1
cardNo : 4
amount : 4000
期望结果 : 账户不存在, 存款失败
ACCEPT

测试用例2
cardNo : 3
amount : -1
期望结果 : 数据非法, 存款失败
ACCEPT

测试用例2
cardNo : 3
amount : 500
期望结果 : 存款成功
ACCEPT
```

```
=========transfer单元测试=======
测试用例1
fromCardNo : 4
toCardNo : 3
amount : 4000
期望结果 : 源账户不存在, 转账失败
ACCEPT
```

```
测试用例2
fromCardNo : 1
toCardNo : 4
amount : 1000
期望结果 : 目的账户不存在，转账失败
ACCEPT

测试用例3
fromCardNo : 1
toCardNo : 2
amount : 2000
期望结果 : 转账金额过大，转账失败
ACCEPT

测试用例2
fromCardNo : 3
toCardNo : 1
amount : -1
期望结果 : 数据非法，转账失败
ACCEPT

测试用例4
fromCardNo : 3
toCardNo : 1
amount : 500
期望结果 : 转账成功
ACCEPT
```



代码覆盖分析：

经过单元测试的测试样例的统计可以发现，所用的测试用例覆盖了所有的判定结果、条件判断结果，且所设计用例可以遍历代码中的所有语句，故 语句覆盖率、判定覆盖率、条件覆盖率均为100%。

（2）如果银行卡的转账服务增加一个新的需求“一次转账金额限定为3000元”，请修改代码并重新进行单元测试和覆盖分析

修改后代码：

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

#define ASIZE 10

using namespace std;

class Account
{
private:
    string cardNo;
    long balance;
public:
    Account()
    {
        this->cardNo = "";
        this->balance = 0;
    };
```

```
    Account(string cardNo, long initBalance)
    {
        this->cardNo = cardNo;
        this->balance = initBalance;
    };
    ~Account(){};

    string getCardNo()
    {
        return this->cardNo;
    };

    long getBalance()
    {
        return this->balance;
    };

    void setCardNo(string cardNo)
    {
        this->cardNo = cardNo;
    };

    void setBalance(long balance)
    {
        this->balance = balance;
    };
};

class AccountManagement
{
private:
    Account db[ASIZE];//假定数据库大小为10
    long getBalance(string userID)
    {
        for (int i = 0; i < ASIZE; i++)
        {
            if (userID == this->db[i].getCardNo())
            {
                return this->db[i].getBalance();
            }
        }
        return -1;
    }

public:
    AccountManagement()
    {
        this->db[0].setCardNo("1");
        this->db[0].setBalance(1000);
```

```
        this->db[1].setCardNo("2");
        this->db[1].setBalance(2000);
        this->db[2].setCardNo("3");
        this->db[2].setBalance(3000);
    };
    ~AccountManagement(){};

    Account findAccount(string userID)
    {
        Account tmp(userID, this->getBalance(userID));
        return tmp;
    };

    void updateAccount(Account account)
    {
        for (int i = 0; i < ASIZE; i++)
        {
            if (account.getCardNo() == this->db[i].getCardNo())
            {
                this->db[i].setBalance(account.getBalance());
            }
        }
    };
};

class AccountService
{
public:
    AccountManagement management;

    AccountService(){};
    ~AccountService(){};

    void withdraw(string cardNo, long amount)
    {
        Account account = this->management.findAccount(cardNo);
        if (account.getBalance() != -1 && amount >= 0 &&
account.getBalance() - amount >= 0)
        {
            account.setBalance(account.getBalance() - amount);
            this->management.updateAccount(account);
        }
    };

    void deposit(string cardNo, long amount)
    {
        Account account = this->management.findAccount(cardNo);
        if (account.getBalance() == -1)
        {
```

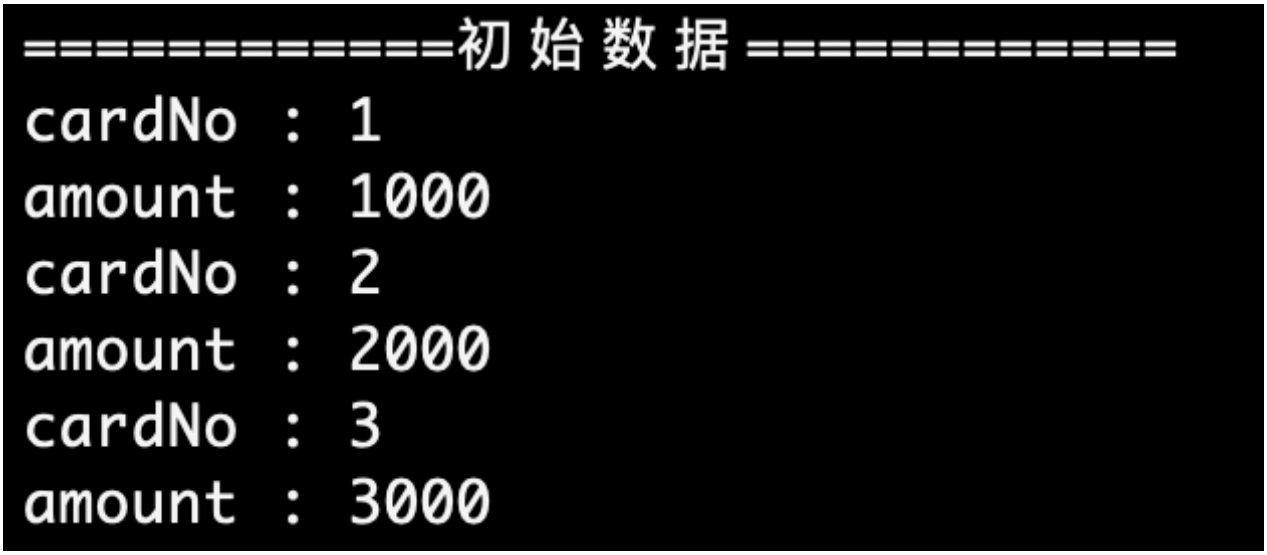
```
            if (amount >= 0)
            {
                account.setBalance(amount);
                this->management.updateAccount(account);
            }
        }
        else
        {
            if (amount >= 0)
            {
                account.setBalance(account.getBalance() + amount);
                this->management.updateAccount(account);
            }
        }
    };

    void transfer(string fromCardNo, string toCardNo, long amount)
    {
        Account fa = this->management.findAccount(fromCardNo);
        Account ta = this->management.findAccount(toCardNo);
        if (fa.getBalance() != -1 && ta.getBalance() != -1 && amount >= 0
&& fa.getBalance() - amount >= 0)
        {
            if (amount <= 3000)
            {
                fa.setBalance(fa.getBalance() - amount);
                ta.setBalance(ta.getBalance() + amount);
                this->management.updateAccount(fa);
                this->management.updateAccount(ta);
            }
        }
    };

    long inquiry(string cardNo)
    {
        Account account = this->management.findAccount(cardNo);
        return account.getBalance();
    };
};

void testWithdraw(AccountService *as, int no, string cardNo, long amount,
long forward, string tips)
{
    cout << "测试用例" << no << endl;
    cout << "cardNo : " << cardNo << endl;
    cout << "amount : " << amount << endl;
    cout << "期望结果 : " << tips << endl;
    as->withdraw(cardNo, amount);
    if (as->management.findAccount(cardNo).getBalance() == forward)
```

```
    {
        cout << "ACCEPT" << endl;
    }
    else
    {
        cout << "ERROR" << endl;
    }
    cout << endl;
};

void testDeposit(AccountService *as, int no, string cardNo, long amount, long forward, string tips)
{
    cout << "测试用例" << no << endl;
    cout << "cardNo : " << cardNo << endl;
    cout << "amount : " << amount << endl;
    cout << "期望结果 : " << tips << endl;
    as->deposit(cardNo, amount);
    if (as->management.findAccount(cardNo).getBalance() == forward)
    {
        cout << "ACCEPT" << endl;
    }
    else
    {
        cout << "ERROR" << endl;
    }
    cout << endl;
};

void testTransfer(AccountService *as, int no, string fromCardNo, string toCardNo, long amount, long forwardFrom, long forwardTo, string tips)
{
    cout << "测试用例" << no << endl;
    cout << "fromCardNo : " << fromCardNo << endl;
    cout << "toCardNo : " << toCardNo << endl;
    cout << "amount : " << amount << endl;
    cout << "期望结果 : " << tips << endl;
    as->transfer(fromCardNo, toCardNo, amount);
    if (as->management.findAccount(fromCardNo).getBalance() == forwardFrom && as->management.findAccount(toCardNo).getBalance() == forwardTo)
    {
        cout << "ACCEPT" << endl;
    }
    else
    {
        cout << "ERROR" << endl;
    }
    cout << endl;
```

```
};

void testInquiry(AccountService *as, int no, string cardNo,string tips)
{
    cout << "测试用例" << no << endl;
    cout << "cardNo : " << cardNo << endl;
    cout << "期望结果 : " << tips << endl;
    as->inquiry(cardNo);
    if (as->management.findAccount(cardNo).getBalance() == as->inquiry(cardNo))
    {
        cout << "ACCEPT" << endl;
    }
    else
    {
        cout << "ERROR" << endl;
    }
    cout << endl;
};

int main()
{
    AccountService as;

    cout << "============初始数据============" << endl;
    cout << "cardNo : " << "1" << endl;
    cout << "amount : " << 1000 << endl;
    cout << "cardNo : " << "2" << endl;
    cout << "amount : " << 2000 << endl;
    cout << "cardNo : " << "3" << endl;
    cout << "amount : " << 3000 << endl;
    cout << endl;

    cout << "========withdraw单元测试========" << endl;
    testWithdraw(&as, 1, "4", 4000, -1, "账户不存在，取款失败");
    testWithdraw(&as, 2, "3", 4000, 3000, "卡内余额不足，取款失败");
    testWithdraw(&as, 2, "3", -1, 3000, "数据非法，取款失败");
    testWithdraw(&as, 3, "3", 500, 2500, "取款成功");
    cout << endl;

    cout << "========deposit单元测试========" << endl;
    testDeposit(&as, 1, "4", 4000, -1, "账户不存在，存款失败");
    testDeposit(&as, 2, "3", -1, 2500, "数据非法，存款失败");
    testDeposit(&as, 2, "3", 2500, 5000, "存款成功");
    cout << endl;

    cout << "========transfer单元测试=======" << endl;
    testTransfer(&as, 1, "4", "3", 4000, -1, 5000, "源账户不存在，转账失败");
```

```
255     testTransfer(&as, 2, "1", "4", 1000, 1000, -1, "目的账户不存在，转账失
    败");
256     testTransfer(&as, 3, "1", "2", 2000, 1000, 2000, "转账金额过大，转账失
    败");
257     testTransfer(&as, 2, "3", "1", -1, 5000, 1000, "数据非法，转账失败");
258     testTransfer(&as, 4, "3", "1", 4000, 5000, 1000, "转账金额超过3000，转账
    失败");
259     testTransfer(&as, 4, "3", "1", 3000, 2000, 4000, "转账成功");
260     cout << endl;
261
262     cout << "========inquiry单元测试========" << endl;
263     testInquiry(&as, 1, "4", "账户不存在，查询失败");
264     testInquiry(&as, 1, "1", "查询成功");
265
266     return 0;
267 }
```

单元测试以及结果：

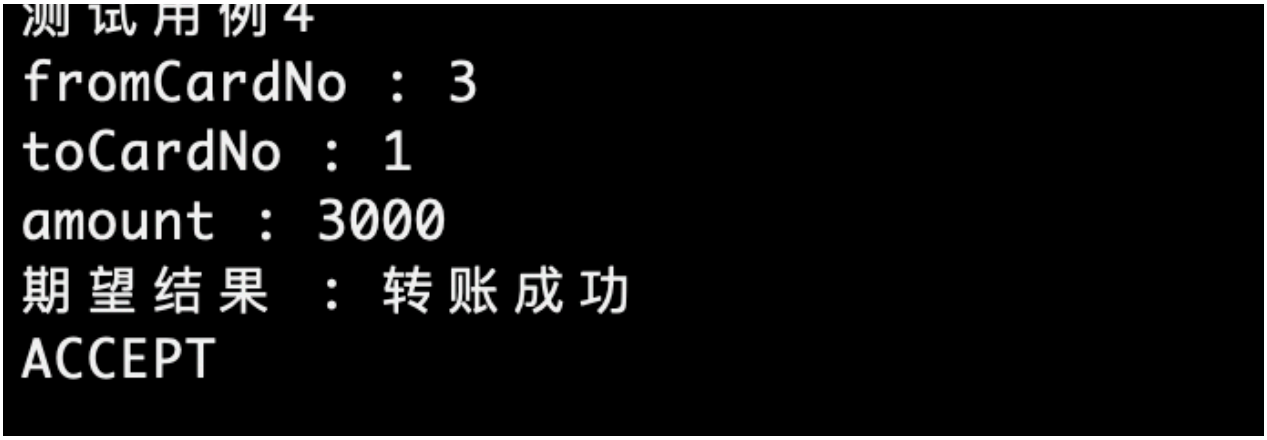


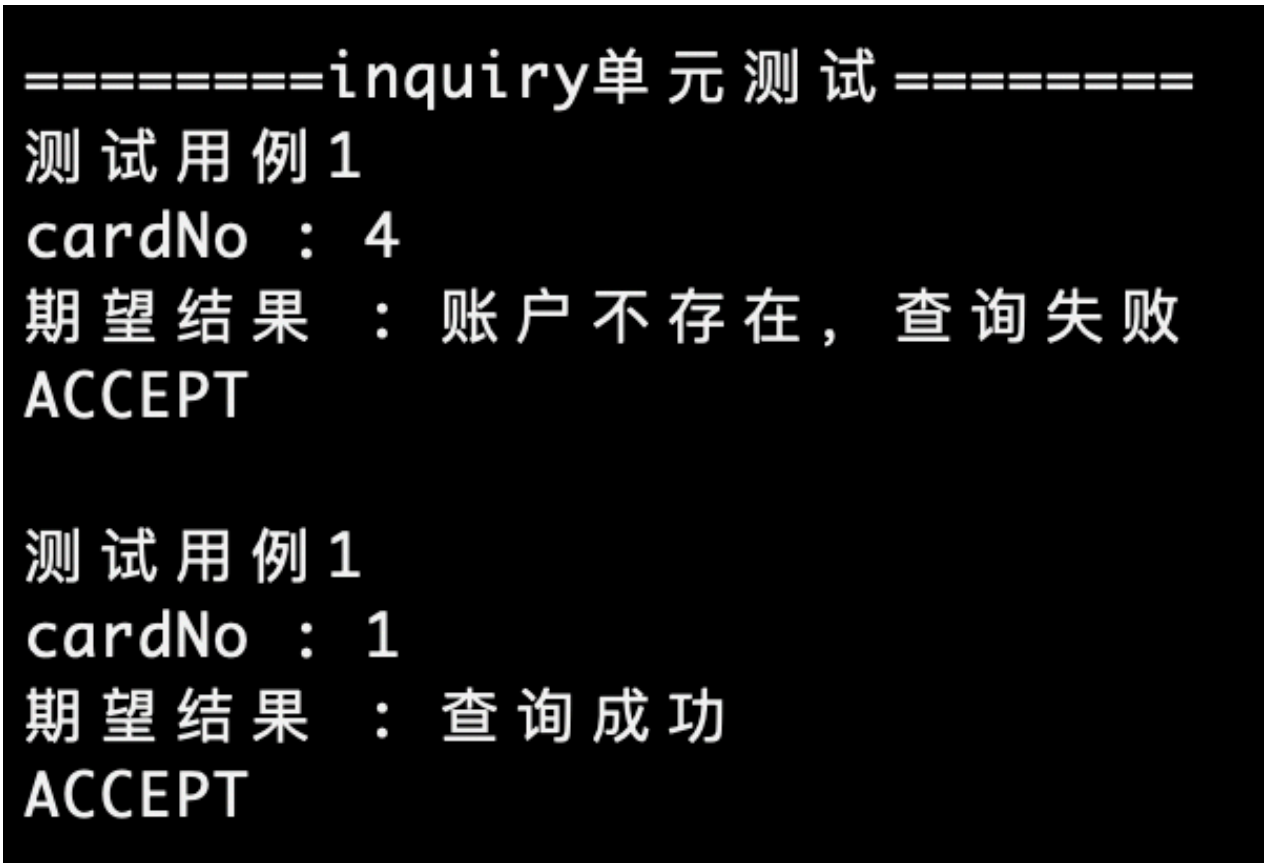
```
=========withdraw单元测试=========
测试用例1
cardNo : 4
amount : 4000
期望结果 : 账户不存在, 取款失败
ACCEPT

测试用例2
cardNo : 3
amount : 4000
期望结果 : 卡内余额不足, 取款失败
ACCEPT

测试用例2
cardNo : 3
amount : -1
期望结果 : 数据非法, 取款失败
ACCEPT

测试用例3
cardNo : 3
amount : 500
期望结果 : 取款成功
ACCEPT
```

```
=========deposit单元测试=========
测试用例1
cardNo : 4
amount : 4000
期望结果 : 账户不存在, 存款失败
ACCEPT

测试用例2
cardNo : 3
amount : -1
期望结果 : 数据非法, 存款失败
ACCEPT

测试用例2
cardNo : 3
amount : 2500
期望结果 : 存款成功
ACCEPT
```

```
========transfer单元测试=======
测试用例1
fromCardNo : 4
toCardNo : 3
amount : 4000
期望结果 : 源账户不存在, 转账失败
ACCEPT
```

```
测试用例2
fromCardNo : 1
toCardNo : 4
amount : 1000
期望结果 : 目的账户不存在，转账失败
ACCEPT

测试用例3
fromCardNo : 1
toCardNo : 2
amount : 2000
期望结果 : 转账金额过大，转账失败
ACCEPT

测试用例2
fromCardNo : 3
toCardNo : 1
amount : -1
期望结果 : 数据非法，转账失败
ACCEPT

测试用例4
fromCardNo : 3
toCardNo : 1
amount : 4000
期望结果 : 转账金额超过3000，转账失败
ACCEPT

测试用例4
```





覆盖分析：

经过单元测试的测试样例的统计可以发现，所用的测试用例覆盖了所有的判定结果、条件判断结果，且所设计用例可以遍历代码中的所有语句，故 语句覆盖率、判定覆盖率、条件覆盖率均为100%。