# STL 应用：调度算法（60 分）

## 一、题目描述

调度算法广泛存在于计算机的世界里，当多个用户或者进程对同一资源进行竞争时，就需要调度算法来进行合理的资源分配。
经典的调度算法有这三种:
1),FCFS 算法: 对先到的 Job 先处理，后到的 Job 后处理。
2),RoundRobin 算法: 系统设置一定的时间片(time silce)，每次给队首 Job 分配时间片。如果此 Job 运行结束，即使时间片没用完，立刻从队列中去除此 Job，并给下一个 Job 分配新的时间片；如果时间片用完没有运行结束，则将此 Job 重新加入就绪队列尾部等待调度.
3),PrioritySchedual: 只要系统中出现一个新的就绪 Job，就进行优先权比较 。若出现优先权更高的 Job，则立即停止当前执行，并处理优先权最高的 Job.

现在，要求实现一个 JobSchedual 类，实现上述三个经典的调度算法

```
struct Job{
    int id; //Job 的标识符，全局唯一
    int enter_time;//Job 进入系统的时间
    int duration;//Job 的持续时间
    int priority;//Job 的优先级
    //可以实现其它辅助方法
};

//可以实现其它辅助结构或方法

class JobSchedual{
    public:
        JobSchedual(Job js[],int length);//js 表示 job 列表，length 表示 job 个数
        ~JobSchedual();

        void FCFS()//先来先服务算法
        void RoundRobin(int time_slice)//轮询算法, time_slice 表示时间片长度
        void PrioritySchedual()//优先级调度算法
    private:
        //自行定义适当的变量，可以使用 STL。
};
```

## 二、注意事项（请认真阅读）

1,对于 FCFS 和 RoundRobin，任意两个 job 的 enter_time 不相同。
2,对于 PrioritySchedual,任意两个 job 的 priority 不相同,并且 priority 的值越大，优

先级越高。
3,可以使用 STL
4,对于 RoundRobin,若一个 job 在一个 time_slice 执行完成的同时,有若干加入系统的 job,则新加入系统的 job 先于刚用完 time_slice 的 job.
例如,job1 = {1,1,6,1} job2 = {1,4,1,1},time_slice = 3,
则在时间点 4 的时候，job1 用完其 time_slice,并且此时 job2 加入系统，则下一个 time-_slice 分配给 job2.
5，对于上面的每一个算法，输出格式为(详细见测试用例)
id job 完成时间点

# 三、提交要求

1 提交一个源码文件:JobSchedual.h，直接打包成 zip 压缩格式的压缩包。不要添加任何其 他目录。
2 实现和声明都写在 h 头文件中(**包括 struct Job 以及自行实现的辅助结构和方法**)，文件编码格式为 utf-8。
3 请严格按照给定的接口进行编码，否则无法调用测试用例。
4 提交的源码文件中，不能包含 main 函数，否则无法编译通过

# 四、具体实例(由于编码问题，最好不要直接复制下面的代码)

**用例一:**

```
int n = 2;
Job myjobs[2];
myjobs[0] = {1,1,4,1};
myjobs[1] = {2,6,2,2};
JobSchedual *jobs = new JobSchedual(myjobs,n);
jobs->FCFS();
delete jobs;
```

**结果:**

```
1 5
2 8
```

**用例二:**

```
int n = 2;
Job myjobs[2];
myjobs[0] = {1,1,4,1};
myjobs[1] = {2,3,2,2};
JobSchedual *jobs = new JobSchedual(myjobs,n);
jobs->FCFS();
```

```
delete jobs;
```
**结果:**
```
1 5
2 7
```

**用例三:**
```
int n = 4;
Job myjobs[4];
myjobs[0] = {1,1,9,1};
myjobs[1] = {2,4,3,2};
myjobs[2] = {3,5,1,3};
myjobs[3] = {4,17,5,4};
JobSchedual *jobs = new JobSchedual(myjobs,n);
jobs->RoundRobin(3);
delete jobs;
```
**结果**
```
2 7
3 11
1 14
4 22
```

**用例四:**
```
int n = 4;
Job myjobs[4];
myjobs[0] = {1,1,9,1};
myjobs[1] = {2,2,8,2};
myjobs[2] = {3,3,7,3};
myjobs[3] = {4,4,6,4};
JobSchedual *jobs = new JobSchedual(myjobs,n);
jobs->PrioritySchedual();
delete jobs;
```
**结果:**
```
4 10
3 16
2 23
1 31
```

出题人:何泽昊

# 多对象交互：模拟师生校园卡系统（40 分）

## 一、题目

类主要分为表示各种卡的 Card 类和表示人的 Campus_Person 类。

卡有充值和消费的接口，银行卡的充值消费可以指定数值，在向校园卡充值时需要绑定的银行卡才能进行。并且需要对外提供 get 类的接口，满足外部访问需求。学生的校园卡只能绑定一张银行卡，但是教师的校园卡能够有多个银行卡向其充值，单也限于自己的学生的银行卡和自己的银行卡。并且教师的校园卡享受消费折扣，0 到 1 之间的小数，保留一位小数，比如 0.9。则当教师卡调用 consume(10)时，教师卡上减少 9 元

Campus_Person 类维持对卡进行绑定的接口。同时教师需要管理他的学生。

具体如下:

卡类:

```
class Card {
    string get_card_id();      //得到卡id
    int get_accunt_balance(); //得到余额
    Card(string card_id, int account_balance);
    ~Card();
    string card_id;               //卡id
    int account_balance;       //卡余额
};

class Bank_Card : public Card {
    Bank_Card(string card_id, int account_balance);
    ~Bank_Card();
    int recharge(int num);     //充值函数
    int consume(int num);      //消费函数
};

class Student_Card : public Card {
    Bank_Card* get_b_card();  //得到学生卡绑定的银行卡
    Student_Card(string card_id, int account_balance, Bank_Card* b_card);
    ~Student_Card();
    int consume(int num);      //消费函数
    int recharge(Bank_Card* b_card, int num);//充值函数，需要提供银行卡，并且只能是该学生卡绑定的银行卡，如果不是则返回-1
    Bank_Card* b_card;             //学生卡绑定的银行卡
};

class Teacher_Card : public Card {
    double get_discount();             //得到折扣
```

```
    vector<Bank_Card*>* get_b_cards(); //得到绑定的卡，教师卡可以绑定多个银行卡
    Teacher_Card(string card_id, int account_balance, double discount, Bank_Card*
b_card);
    ~Teacher_Card();
    int consume(int num);      //消费函数
    int recharge(Bank_Card* b_card, int num);//充值函数，银行卡可以使b_cards中的任何一
张，失败则返回-1
    double discount; //折扣，0-1之间的小数，保留一位小数，比如0.9。则当教师卡调用
consume(10)时，教师卡上减少9元
    vector<Bank_Card*> b_cards; //与该教师卡绑定的银行卡
};
```

师生类:

```
class Campus_Person {
    string get_campus_id();
    Bank_Card* get_b_card();
    Campus_Person(string campus_id, Bank_Card* b_card);
    ~Campus_Person();
    string campus_id;     //校园ID
    Bank_Card* b_card;         //每一个人都至少有一张银行卡
};

class Student : public Campus_Person {
    Student_Card* get_s_card();
    Student(string campus_id, Bank_Card* b_card, Student_Card* s_card);
    ~Student();
    int bind_card(Bank_Card* b_card);  //将该学生持有的校园卡设置为新的银行卡
    Student_Card* s_card; //该学生持有的校园卡
};

class Teacher : public Campus_Person {
    Teacher_Card* get_t_card();
    vector<Student*>* get_students();
    Teacher(string campus_id, Bank_Card* b_card, Teacher_Card* t_card);
    ~Teacher();
    int add_bind_card(Bank_Card* b_card);   //向老师的校园卡中增加绑定银行卡
    int delete_bind_card(Bank_Card* b_card);//向老师的校园卡中增加删除银行卡
    int add_student(Student* stu);          //增加老师的学生
    int delete_student(Student* stu);       //删除老师的学生
    Teacher_Card* t_card; //该老师持有的校园卡
    vector<Student*> stus;  //老师的学生
};
```

## 二、注意事项

- discount 取 0.1、0.2…0.9 等 ， 注意 account_balance 为 int 值，测试数据保证每次消费数据为 10 的倍数，所以直接相乘后转化为 int 值即可
- 各 ID 需要手动保证不同
- 返回值为 int 的函数，如果操作成功，则返回 0，失败则返回 1
- 保证 get 类、消费、充值、绑定等方法的外部访问，其他成员访问属性可以自定义

## 三、测试

可使用如下格式进行测试

```
int main()
{
    /*
        创建一个学生
    */
    Bank_Card* s_b_card = new Bank_Card("s_b1",100);
    Student_Card* s_card = new Student_Card("s_s1", 0, s_b_card);
    Student* s = new Student("s_1", s_b_card, s_card);

    /*
        创建一个老师
    */
    Bank_Card* t_b_card = new Bank_Card("t_b1", 100);
    Teacher_Card* t_card = new Teacher_Card("t_s1", 0, 0.9, t_b_card);
    Teacher* t = new Teacher("t_1", t_b_card, t_card);

    /*
        测试银行卡功能
    */
    s_b_card->consume(10);
    if (s_b_card->get_accunt_balance() != 90) cout << "wrong1" << endl;
    s_b_card->recharge(10);
    if (s_b_card->get_accunt_balance() != 100) cout << "wrong2" << endl;

    /*
        测试学生卡功能
    */
    s_card->recharge(s_b_card, 10);
    if(s_b_card->get_accunt_balance() != 90 || s_card->get_accunt_balance()!=10) cout
<< "wrong3" << endl;
    s_card->consume(10);
    if (s_b_card->get_accunt_balance() != 90 || s_card->get_accunt_balance() != 0) cout
```

```
<< "wrong4" << endl;
    //新建一张未绑定的卡，此处应该充值失败
    Bank_Card* temp1 = new Bank_Card("temp1", 100);
    s_card->recharge(temp1, 10);
    if(s_card->get_accunt_balance() != 0 || temp1->get_accunt_balance()!=100) cout <<
"wrong5" << endl;

    /*
        测试老师卡功能
    */
    t_card->recharge(t_b_card, 10);
    if (t_b_card->get_accunt_balance() != 90 || t_card->get_accunt_balance() != 10)
cout << "wrong6" << endl;
    t_card->consume(10);
    if (t_b_card->get_accunt_balance() != 90 || t_card->get_accunt_balance() != 1) cout
<< "wrong7" << endl;
    //使用学生卡充值，此处应该充值失败
    t_card->recharge(s_b_card, 10);
    if (t_b_card->get_accunt_balance() != 90 || t_card->get_accunt_balance() != 1 ||
s_b_card->get_accunt_balance()!=90) cout << "wrong8" << endl;

    /*
        测试学生绑定功能
    */
    s->bind_card(temp1);
    s_card->recharge(s_b_card, 10);
    if (s_b_card->get_accunt_balance() != 90 || s_card->get_accunt_balance() != 0) cout
<< "wrong9" << endl;
    s_card->recharge(temp1, 10);
    if ( temp1->get_accunt_balance() != 90 || s_card->get_accunt_balance() != 10 ) cout
<< "wrong10" << endl;

    /*
        测试老师绑定功能
    */
    t->add_student(s);
    t->add_bind_card(s->get_b_card());
    t_card->recharge(s_b_card, 10);
    if (s_b_card->get_accunt_balance() != 80 || t_card->get_accunt_balance() != 11)
cout << "wrong11" << endl;
    //解除银行卡的绑定之后不能进行充值
    t->delete_bind_card(s->get_b_card());
    t_card->recharge(s_b_card, 10);
    if (s_b_card->get_accunt_balance() != 80 || t_card->get_accunt_balance() != 11)
```

```
cout << "wrong11" << endl;
    //不是老师学生的卡不能进行绑定
    Bank_Card* test1 = new Bank_Card("test1", 100);
    if(t->add_bind_card(test1) != -1) cout << "wrong12" << endl;
    //测试删除学生
    t->delete_student(s);
    if (t->add_bind_card(s->get_b_card()) != -1) cout << "wrong13" << endl;

    return 0;
}
```

# 四、提交

提交一个 ClassInteraction.zip 文件，包含以下文件：

1、 card.h：Card 类、Bank_Card 类、Student_Card 类、Teacher_Card 类的定义

2、 card.cpp：Card 类、Bank_Card 类、Student_Card 类、Teacher_Card 类的实现

3、 person.h：Campus_Person 类、Student 类、Teacher 类的定义

4、 person.cpp：Campus_Person 类、Student 类、Teacher 类的实现

其他要求见之前的实验

出题人：李开鸣