

Service Management of Campus Card

Group Five

July 7, 2019

Abstract

We finish a service management system of campus card having two login models. It can store students' and shops' information, record the interactions in shops, library and clinic. In addition to the basic functions of campus cards, the students can make the reservation in the library and search the medicine of the clinic in advance. And the administrator is able to modify and print the bills.

Contents

1	Introduction	3
1.1	Problem Statement	3
1.2	Advantages	3
2	Group Division	4
2.1	Member1:Wang Jianing	4
2.2	Member2:Yu Huacheng	4
2.3	Member3:Sun Haokun	4
3	Analysis	4
4	Design	6
4.1	Design of C codes	6
4.1.1	Structures	6
4.1.2	Basic functions	6
4.1.3	Library function	7
4.1.4	Medicine function	8
4.2	Design of Qt	9
5	Implementation	10
5.1	Overview	10
5.2	main.c	11
5.3	functions.h	11
5.4	medicine.h	13
5.5	queue.h	14
6	Testing and Debugging	15
6.1	Test of our C codes	15
6.2	Test of Qt codes	16
7	Results and Conclusion	19

1 Introduction

1.1 Problem Statement

1. Which functions are necessary in our system?
A: Two types of logging in, information of students and shops, record of the interaction, revise of the information, library reservation, and medicine sorting.
2. How to design a user-friendly interface different from today's common system?
A: To find a different way with other groups, we use QT to design our interface, which is a stronger way than common C in interface design.
3. How can we save data?
A: To get more complex data, we use database instead of struct so that we can quickly use database in QT.
4. How to make our cooperation efficient?
A: First, we made the design and created the ports in advance to avoid the conflicts between the codes written by different teammates. Then after the debug and tests, we ensured that the codes and functions are correct. Finally, according to the C codes, we completed the interface.

1.2 Advantages

1. We designed more functions.
2. We used the algorithm of quicksort and the knowledge of data structure.
3. Our interface is very interactive.
4. When the user inputs incorrectly, we are able to give proper prompt.

2 Group Division

Table 1: Personal information

Name	student ID	College
Wang Jianing	11170203	Physics
Yu Huacheng	27180223	Chemistry
Sun Haokun	13180423	Biology

2.1 Member1:Wang Jianing

I'm in charge of three aspects including the design of functions and program structures, implementation of basic statements in about 1200 lines codes, and the application of quicksort in the function of medicine. Also, I created the ports to make our cooperation go smoothly.

2.2 Member2:Yu Huacheng

I am in charge of the whole QT interface design, do over 80 percent of QT work(900 lines of codes). Meanwhile, I write a database in our interface to save our data.

2.3 Memeber3:Sun Haokun

First,I helped Yu Huacheng to do part of the QT work. Second,There were several warnings at first, and I used debug to complete code maintenance. Finally, I used queue to complete the function of reservation in the library in C codes.

3 Analysis

To solve the problem, we need to create two branches: users and administrators.

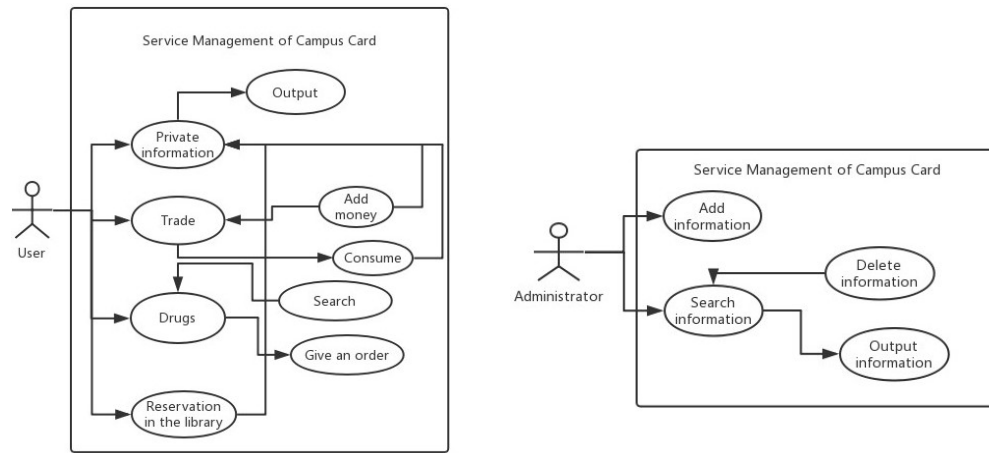


Figure 1: Use case diagram of the project

- For users:

First, it is able to store the private information of users including card numbers, left money, tickets, time and so on. We must record the information in time to output a file. Second, when users trade with supermarkets, canteens, and the bathhouse, there will be two probable choices including the consumption and recharge. So it is necessary to change the value in the structure. Third, to search for the medicine in the clinic, users must search a classification of all the medicine, and then the information of chosen medicine will be showed in a specific order. Finally, students can reserve the room in the library and their information need to be updated. Also, when the maximum number of reservations is achieved, the function should be terminated.

- For administrators:

The administrator can manage the information easily. Therefore, we should implement the functions of add, change, delete, and output into every stored records, which are mainly similar to the functions in the users' branch.

4 Design

4.1 Design of C codes

4.1.1 Structures

Totally, we have four structures to complete the definition of some elements in the functions. In this part, we will explain three of them. Because we design a linked list having lots of nodes, in which every user stores their private card information. So the card structure is necessary, and it has to contain another two structures used to record the trade time and trade money. There is also a pointer in the card structure, which is used to find the next user.

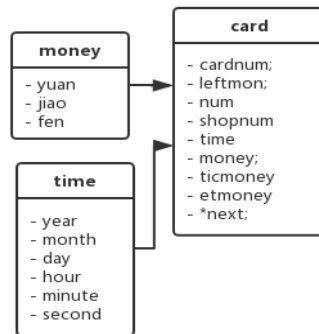


Figure 2: Class diagram of the structures

4.1.2 Basic functions

There are eleven different functions used to implement the basic functions of our project- add, store, read, seek, recharge, del, change, print and so on. The next flowchart is an overview of these similar functions. After choosing the login model, there are more cases, and one specific function is under one of these cases. Due to the similarity of the codes, we decide to put all these functions in one header file.

Now, we take the second branch in users as an example. After the user choose a case, he need to input a signal(card number in this part)

to find the location of his own node. At this time, the working pointer goes through all the nodes and compare the elements in every node. If the working pointer finds the correct node, we can add, change or delete the information in the structure. On the contrary, there is a proper prompt when the working pointer don't find the right node.

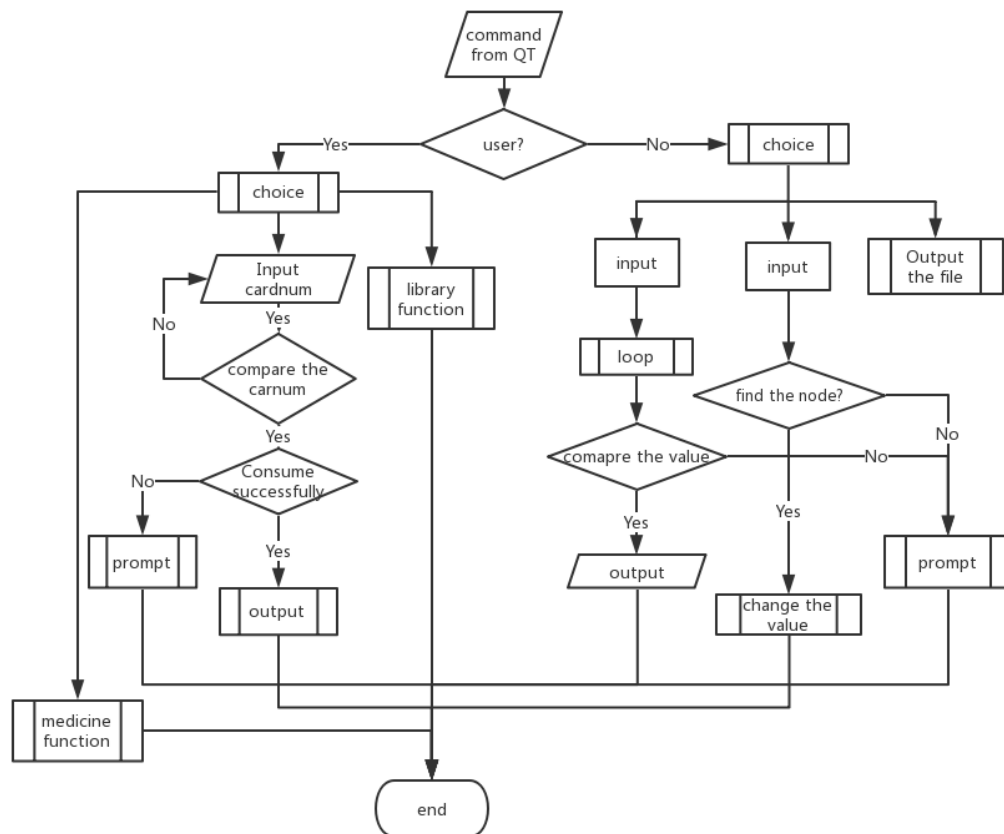


Figure 3: Flowchart for major functions

4.1.3 Library function

When the users want to make a reservation, we should make a space used to restore the users having reserved the room. And when the first us-

er finishes the reservation, other users can move their location in the space and there is another one chance left. Therefore, we decide to use queue to store all the information. The picture below is a simple explanation of queue, every stack represents one user and by the judgement of rear and front, we are able to update the information and print the prompt.

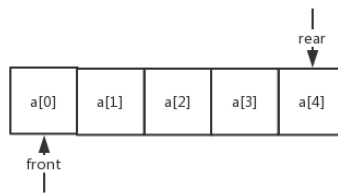


Figure 4: Explanation of queue

4.1.4 Medicine function

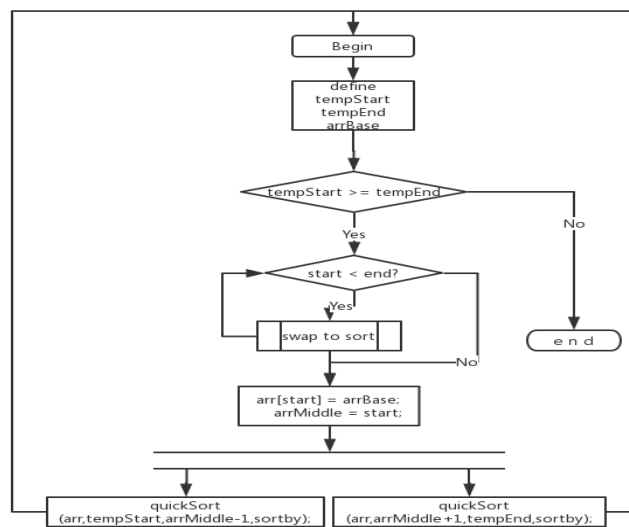


Figure 5: Flowchart of qsort

In this function, two goals are direct: match the string and sort the information. So when scanning the content in the file, we need to compare the type of the medicine to give a list waiting to be ordered. Then, in order to sort the medicine by price or remaining quickly, we think the quick sort algorithm is the best choice.

4.2 Design of Qt

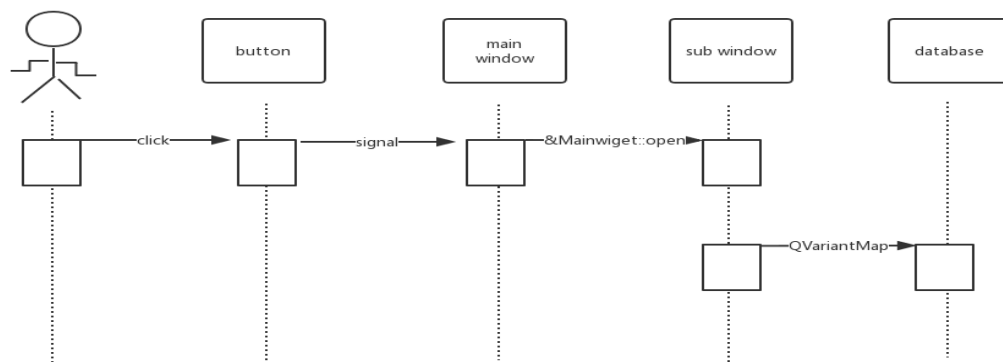


Figure 6: Interaction diagram of Qt design

Signal and slot is a best partner in QT design. So I will describe our interface by describing a button named `addstudent`. It is a button on the main window and its function is to open a sub window used to record students' information. First of all, we need to design a button on the main window. Then we give it a signal called 'press', which means the signal will be cast to the whole main window when we press it. Then we can connect the slot to main window by a slot, which can bring out a sub window.

Obviously, we don't want to waste time on repeat signal-slot part, so we can open the UI document and then use QT's plug-in unit to design the sub window and connect it to our database. Then we need a function called `detector` to detect if we have a legal input. If our input is wrong style, it will warn us of it then close the window. Otherwise, our database will show you it succeeds. Then close the window.

5 Implementation

5.1 Overview

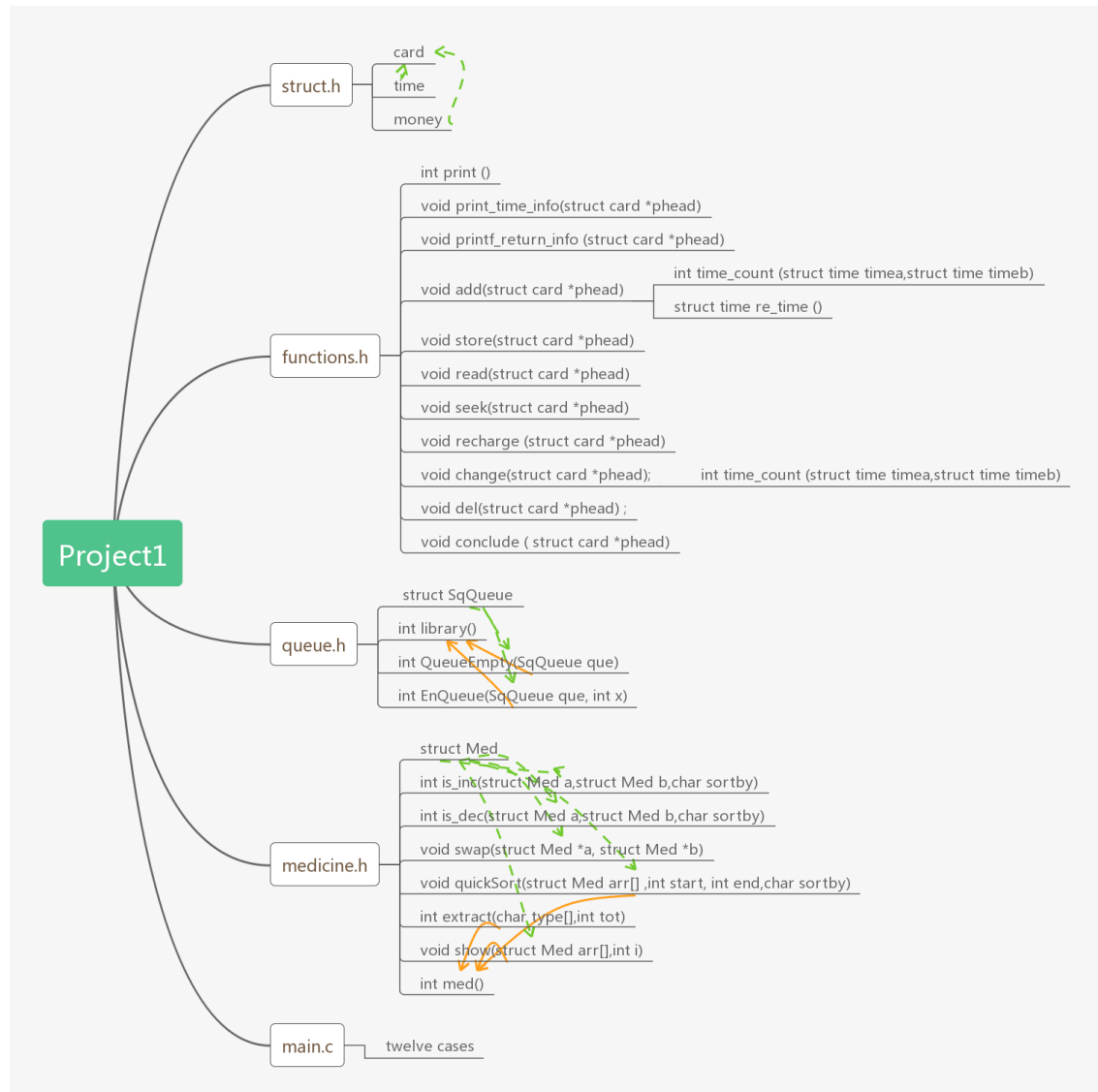


Figure 7: Mindmap for the project

To make the project clear, our group divided the codes into five parts including four header files and a main function. These four header files contain three major functions which will be explained as follows.

5.2 main.c

```
{
    pnew=(struct card *)malloc(sizeof (struct card));
    ptail->next=pnew;
    pnew->next=NULL;
    ptail=pnew;
    pnew->cardnum=9999;
    pnew->leftmon=100;
    pnew->num=0;
    pnew->money=0;
    pnew->retmoney=0;
    pnew->shopnum=00000;
    pnew->sum=0;
    pnew->ticmoney=0;
    pnew->strtime=temp;
}
```

In the main function, we just give the initial value of the elements in the structure and use the switch to go into every different functions.

5.3 functions.h

There are too many functions in this part, so we take some classic functions as an example.

- If the user want to add some information, no matter the private information or left money changed by consumption, the procedure is similar as mentioned above. First, it is a must to find the location of the user's struct in the linked list, and flag is a judgement used to give a proper prompt. During the movement of p, we will find the target. Second, we want to change the elements in the structure. However, we don't want to make use of the working pointer, so we create pnew, and change the information through the pnew. Meanwhile, other elements such as time will be updated, too.

```

48 void recharge (struct card *phead) {
49     struct time re_time ();
50     printf("请输入充值卡号 \n");
51     int cardnum=0, flag=0;
52     scanf("%d", &cardnum);
53     struct card *p=phead, *next=phead->next, *pnew;
54     while (p->next!=NULL) {
55         if (p->cardnum==cardnum)
56             if (next->cardnum!=cardnum || next->next==NULL)
57             {
58                 flag=1;
59                 break;
60             }
61         p=p->next;
62         next=next->next;
63     }
64     if (flag) {
65         printf("请输入充值金额\n");
66         int plus=0;
67         scanf("%d", &plus);
68         if (p->leftmon+plus<=1000)
69         {
70             pnew=(struct card *)malloc(sizeof (struct card));
71             pnew->leftmon=p->leftmon+plus;
72             pnew->strtime=re_time();
73             pnew->shopnum=999999;
74             pnew->num=p->num+1;
75             pnew->cardnum=cardnum;
76             pnew->money=plus;
77             printf("充值成功 当前余额为");
78             printf_money(pnew->leftmon);
79             printf("\n");
80         }
81         else
82         {
83             printf("温馨提示 校园卡余额不能超过1000\n");
84         }
85         return ;
86     }
87     printf("此卡不存在 请重新输入\n");
88 }

```

- The codes used to delete information are similar to the codes mentioned above. But we need to free the t in order to delete the node in the lists.

```

847 printf("请输入你要删除的信息的卡号\n");
848 scanf("%d", &cnum);
849 t=phead->next;
850 next=t->next;
851 front=phead;
852 while((t->cardnum!=cnum || t->next->cardnum==cnum)&&next!=NULL)//若找到要删除的节点或者到达链表末端结束
853 {
854     ...
855 }
856 front->next=t->next;
857 free(t);
858 for(i=1; i<=120; i++)
859 {
860     printf("-");
861 }
862 printf("\n");
863 printf("删除成功\n");
864 }

```

- If someone wants to print the file of the trades, we need to read the information in linked lists and write them into a file. First, we have to ensure the storage of a file exists, and create a file pointer. Second, according to the program syntax, we create another pointer to

go through all information and write them into the opened file. To test the codes, we usually make the project print the content in the console. Finally, don't forget to close the completed file. So we make a judgement through the value of a.

```

465 void store(struct card *phead)
466 {
467     FILE *fp=fopen("E:\\source.txt","w");
468
469     if(fp==NULL)
470     {
471         printf("文件打开失败!\n\n");
472         exit(0);
473     }
474     printf("文件打开成功!\n\n");
475
476     struct card *pu;
477     pu=(struct card*)malloc(sizeof(struct card));
478     pu=phead;
479     pu=pu->next;
480     fwrite (pu,10*sizeof(float),1,fp);
481     fprintf(fp,"卡号\t余额\t编号\t商家号\t\t时间\t\t消费金额\t券金额\t\t补贴金额\t\n");
482     while(pu)
483     {
484         fprintf(fp,"%d\t%.2f\t%d\t%d\t%d年%d月%d日%d时%d分%d秒\t\t\t\t\t",
485             pu->cardnum,pu->leftmon,pu->num,pu->shopnum,pu->strtime.year,pu->strtime.month,pu->strtime.day,pu->strtime.hour,pu->strtime.minute,pu->strtime.second,
486             pu->pu->next);
487     }
488     int a=fclose(fp);
489     if(a!=0)
490     {
491         printf("文件关闭失败!\n\n");
492         exit(0);
493     }
494     printf("文件关闭成功!\n\n");
495
496 }
497

```

5.4 medicine.h

```

71 void quickSort(struct Med arr[],int start, int end,char sortby){
72     struct Med arrBase;
73     int arrMiddle;
74     int tempStart = start,
75         tempEnd = end;
76     //a necessary return condition
77     if(tempStart >= tempEnd)
78         return;
79
80     arrBase = arr[start];
81     while(start < end){
82         while(start < end && (is_dec(arr[end],arrBase,sortby)==1) )
83             end--;
84         if(start < end)
85         {
86             swap(&arr[start], &arr[end]);
87             start++;
88         }
89
90         while(start < end && (is_inc(arr[start],arrBase,sortby)==1))
91             start++;
92         if(start < end)
93         {
94             swap(&arr[start], &arr[end]);
95             end--;
96         }
97     }
98     arr[start] = arrBase;
99     arrMiddle = start;
100     //recursion
101     quickSort(arr,tempStart,arrMiddle-1,sortby);
102     quickSort(arr,arrMiddle+1,tempEnd,sortby);

```

It is the most important function in medicine.h. At the beginning, we have to give a Med, choose the start and end(the first and the last value, and give the types(by the price or the remaining number of medicine) of sorting. For the recursion function like this, we must create a return function in case the loop keeps going on. Then arrBase is the benchmark value, and we compare the value and swap the value that breaks the regulation of sorting. So we divide the data into two parts, and each of these parts will do this function again.

```
19 int is_inc(struct Med a,struct Med b,char sortby)
20 {
21     if(sortby=='p') // price
22     {
23         if(a.price<b.price)
24             return 1;
25         if(a.price>b.price)
26             return -1;
27         return 0;
28     }
29
30     if(sortby=='l') // Left
31     {
32         if(a.left<b.left)
33             return 1;
34         if(a.left>b.left)
35             return -1;
36         return 0;
37     }
38
39     return 0;
40 }
41
42 int is_dec(struct Med a,struct Med b,char sortby)
```

These two functions are used to give the order of a and b, but we make a integer return value to make the judgement of the swap easier.

5.5 queue.h

First of all, we need to define a queue.

```
3 typedef struct {
4     int data[5];
5     int front,rear;
6 }SqQueue;
7
```

```

12 int QueueEmpty(SqQueue que){
13     return (que.front == que.rear);
14 }
15 int EnQueue(SqQueue que, int x){
16     if((que.rear + 1)%5 == que.front)
17         return 0;
18     que.data[que.rear] = x;
19     que.rear = (que.rear + 1) % 5;
20     return 1;
21 }

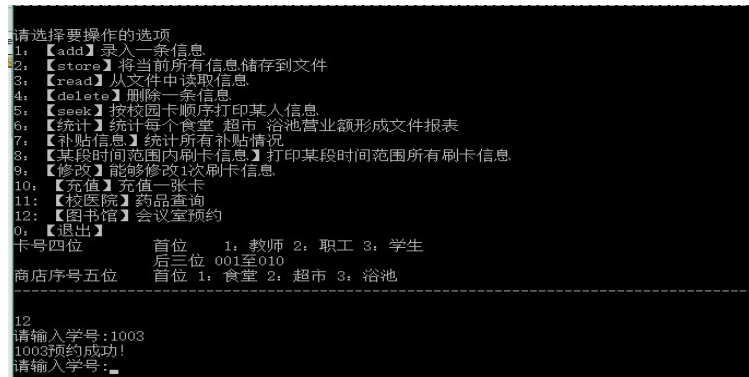
```

If the user wants to make a reservation, he need to find a location of the stack and enter the queue. The integer rear and front are both zero at first, if the queue is empty (rear, front = 0), the reservation is successful, the data will be the card number of the user, and the value of rear will be changed. More users will repeat the same procedure. But when the rear meets the front, the queue has no more stacks, and the reservation fails.

6 Testing and Debugging

6.1 Test of our C codes

1. We choose the case 12 and print the card number, then the console will give the prompt of the reservation status.



```

请选择要操作的选项
1: 【add】录入一条信息
2: 【store】将当前所有信息储存在文件
3: 【read】从文件中读取信息
4: 【delete】删除一条信息
5: 【seek】按校园卡顺序打印某人信息
6: 【统计】统计每个食堂 超市 浴池营业额形成文件报表
7: 【补贴信息】统计所有补贴情况
8: 【某段时间范围内刷卡信息】打印某段时间范围内所有刷卡信息
9: 【修改】能够修改1次刷卡信息
10: 【充值】充值一张卡
11: 【校医院】药品查询
12: 【图书馆】会议室预约
0: 【退出】
卡号四位      首位 1: 教师 2: 职工 3: 学生
               后三位 001至010
商店序号五位  首位 1: 食堂 2: 超市 3: 浴池
-----
12
请输入学号:1003
1003预约成功!
请输入学号:

```

Figure 8: Test of the reservation

2. If we choose the type zhusheji or pianji, we can sort the medicine by the price or the many it remains. In the Figure8, we can see the comparison between the different two types. And by the comparison

between Figure8 and Figure9, it's obvious to see the different ways to sort medicine.

53	地塞米松磷酸酯注射液	注射液	1.100000	1364
21	安乃近注射液	注射液	2.500000	600
2	0.9%氯化钠注射液	注射液	4.970000	2
9	5%葡萄糖注射液	注射液	5.580000	27
6	1%葡萄糖注射液	注射液	5.630000	66
1	0.9%氯化钠注射液	注射液	5.380000	26
8	5%葡萄糖注射液	注射液	6.290000	23
4	1%葡萄糖注射液	注射液	6.360000	36
7	0.9%氯化钠注射液	注射液	6.490000	8
10	5%葡萄糖注射液	注射液	7.620000	19
5	1%葡萄糖注射液	注射液	7.710000	19
20	艾迪注射液	注射液	21.520000	69
56	丹红注射液	注射液	28.720000	39
54	地特胰岛素注射液	注射液	191.400000	8
18	阿替卡因肾上腺素注射液	注射液	350.000000	4
51	曲美替尼己微特尼节苷脂注射液	注射液	407.480000	4
45	醋酸曲安注射液	注射液	464.600000	3

47	醋酸泼尼松片	片剂	7.500000	200
43	苯磺唑酮片	片剂	14.400000	104
30	苯磺唑酮氯地平片	片剂	17.500000	86
46	醋酸地塞米松片	片剂	26.100000	57
55	地榆升白片	片剂	26.500000	57
21	苯磺唑酮氯地平片	片剂	27.240000	55
29	苯磺唑酮氯地平片	片剂	28.520000	51
19	阿托伐他汀钙片	片剂	43.400000	35
13	阿托伐他汀片	片剂	61.320000	4
25	奥卡西干片(通顺衣)	片剂	96.650000	16
48	醋酸泼尼松片	片剂	161.420000	9
24	奥卡西干片(通顺衣)	片剂	164.390000	9

Figure 9: Qsort by the price

2	0.9%氯化钠注射液	注射液	4.970000	2
45	醋酸曲安注射液	注射液	464.600000	3
51	曲美替尼己微特尼节苷脂注射液	注射液	407.480000	4
18	阿替卡因肾上腺素注射液	注射液	350.000000	4
54	地特胰岛素注射液	注射液	191.400000	8
2	0.9%氯化钠注射液	注射液	6.390000	8
10	5%葡萄糖注射液	注射液	7.620000	19
5	1%葡萄糖注射液	注射液	7.710000	19
8	5%葡萄糖注射液	注射液	6.290000	23
1	0.9%氯化钠注射液	注射液	5.650000	26
9	5%葡萄糖注射液	注射液	5.580000	27
4	1%葡萄糖注射液	注射液	6.360000	36
50	丹红注射液	注射液	28.720000	39
6	1%葡萄糖注射液	注射液	5.630000	66
20	艾迪注射液	注射液	21.520000	69
21	安乃近注射液	注射液	2.500000	600
53	地塞米松磷酸酯注射液	注射液	1.100000	1364

13	阿托伐他汀片	片剂	61.320000	4
24	奥卡西干片(通顺衣)	片剂	164.390000	9
48	醋酸泼尼松片	片剂	161.420000	9
25	奥卡西干片(通顺衣)	片剂	96.650000	16
19	阿托伐他汀钙片	片剂	43.400000	35
29	苯磺唑酮氯地平片	片剂	28.520000	51
31	苯磺唑酮氯地平片	片剂	27.240000	55
46	醋酸地塞米松片	片剂	26.100000	57
55	地榆升白片	片剂	26.500000	57
30	苯磺唑酮氯地平片	片剂	17.500000	86
43	苯磺唑酮片	片剂	14.400000	104
47	醋酸泼尼松片	片剂	7.500000	200

Figure 10: Qsort by the remaining number of medicine

Because these two functions haven't been added in the interface, we have to show our results in the console.

6.2 Test of Qt codes

Input	Plan	True outcome
Wrong id	attention! fail!	attention! fail!
Wrong password	warning! Incorrect input!	warning! Incorrect input!
A right student information	Our database shows the information to us in main window.	Our database shows the information to us in main window.
A right student information	Our database changes the information of relevant student in main window.	Our database changes the information of relevant student in main window.
Choose one information and delete it	It disappears.	Only when we refresh the window, our database will give a right answer.

1. Record information: shop have a five number id. If we give a wrong id, what will happen?

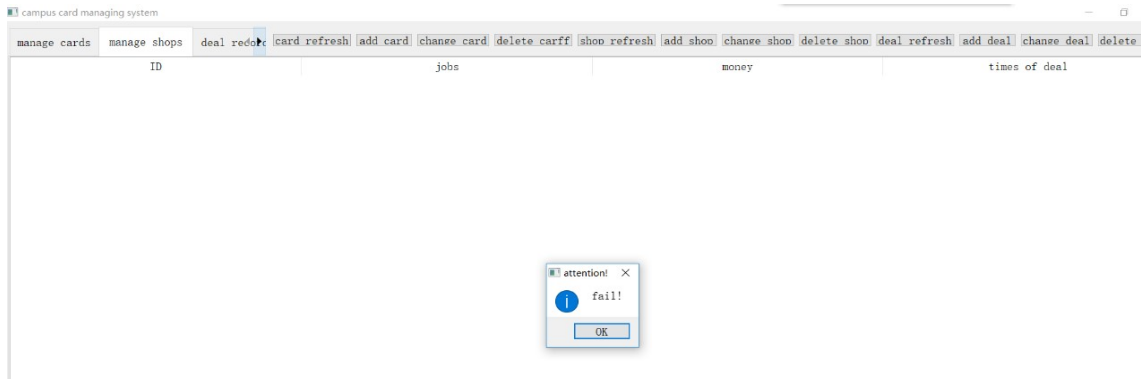


Figure 11: Test one

2. Login unit : we have two models(user model and admin model). If we don't give it a correct password, what will happen?

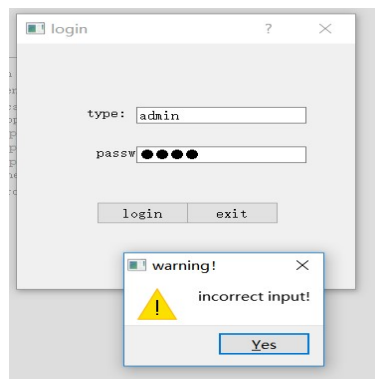


Figure 12: Test two

3. Add students(shop / deal): we can add student in our data base.

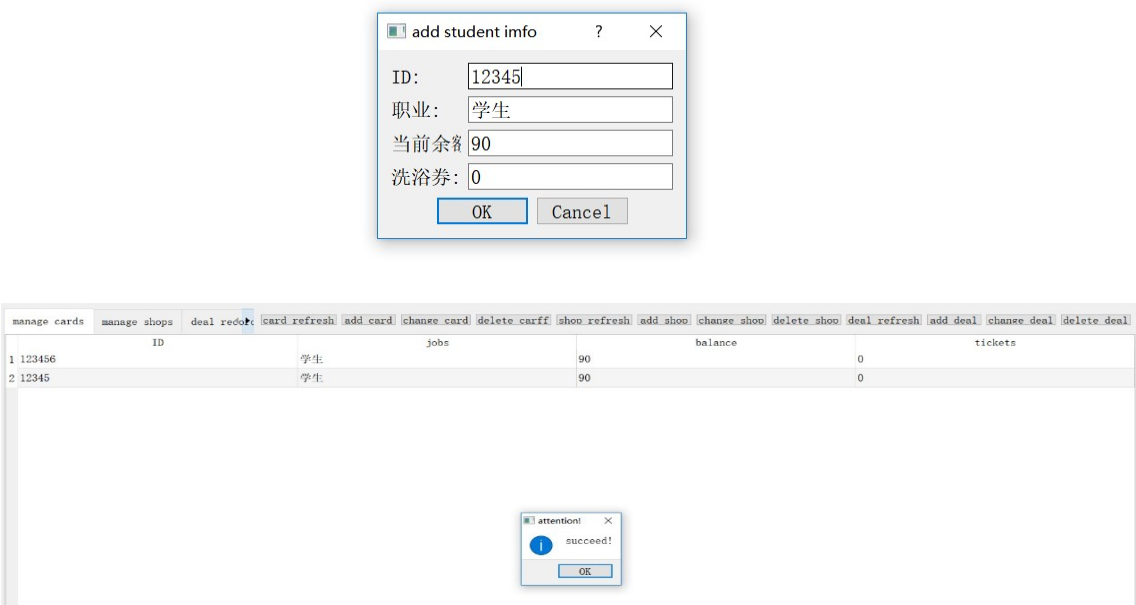


Figure 13: Test Three

4. Change student(shop/deal)information: we can change the information if we have legal input.

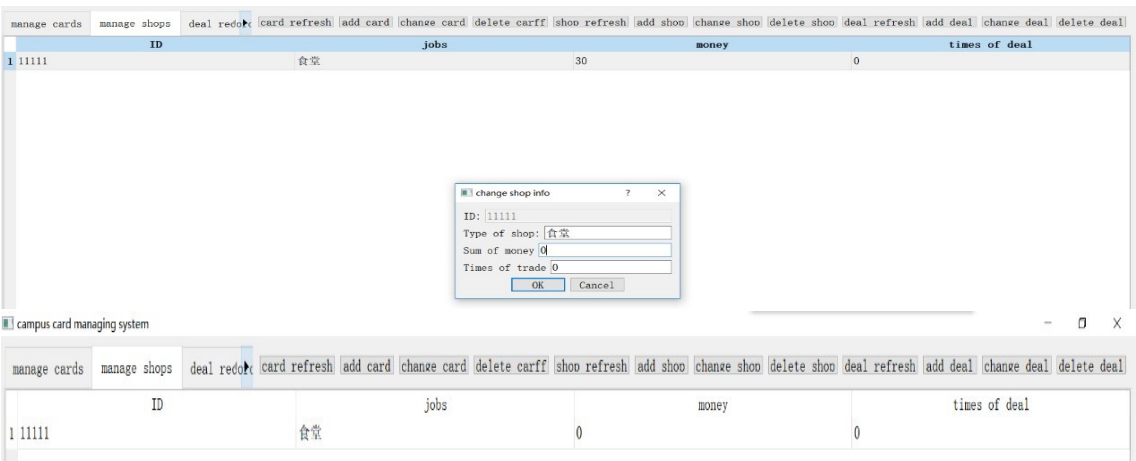


Figure 14: Test Four

5. Delete information: we can delete some information

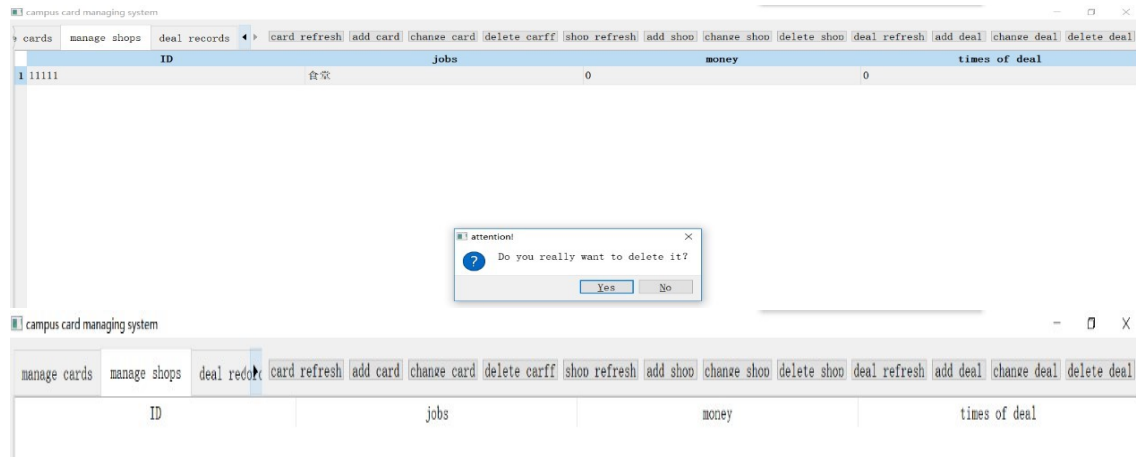


Figure 15: Test Five

7 Results and Conclusion

We feel happy because we realize our aim of writing a user-friendly system and learn from it. We design a good Service Management of Campus Card and now we completely know how the system works and truly learn a lot during our whole design period.

We have finished all the core C codes and QT interface codes(except sorting in QT). We revise the knowledge learned in the class including the structure, linked list, file etc. In addition, we learn something new by ourselves including the use of QT, to write a data base to store our information, to use queue to imitate library booking situation, and complete a new way of sorting.

However, there are also some problems. For example, our database on the main window cannot automatic refresh, and the connect between login window and main window still have some problems. Also, the functions of reservation system are not well-rounded. These problems show us the importance of a careful plan. We can truly come up with some good ideas when we are debugging or testing, but it is hard to add extra code into our carefully designed part. Indeed, we should have held a brainstorm before the design work to have a better system. If we have the chance

to better our work, we can have a larger queue in the library booking part. What's more , we can match our core code better with our QT interface code instead of designing core code first then the QT code. Last but not least, we should use some other algorithms in the system, which can help us learn C better.

We are truly have a nice work, congratulations for my teammates, yeah!