程序设计基础

课程设计报告

题　　目：学生管理系统

设 计 者：赵宜珺、王金鹏、陈思

专业班级：信 17（8）班

学　　号：17263829、17263824、17263815

指导教师：蔡秋枫

2018 年 05 月 18 日

盐城师范学院 信息工程学院

目录

1 题目

学生管理系统的分析与设计

学生管理系统主要是对学生信息进行管理，并根据学生信息可以快速的进行学生信息查询和修改。本系统以 C 语言为基础，提供简单、易操作的用户操作界面，实现

对学生信息的管理。系统以结构体为基础，涉及结构体链表、函数、全局变量与局部变量、文件操作和多文件编写等知识，旨在通过程序的学习，更好地掌握系统结构化设计方法和编程技巧，为将来开发出高质量的应用系统打下坚实的基础。

2 系统描述

2.1 基本功能描述

学生管理系统采用 Visual Studio Code 与 Dev C++ 开发工具，主要实现对学生信息进行查询、删除、显示、保存、排序、学分计算。学生信息最终保存在外部文件中。同时系统提供简单的操作界面用于用户与系统之间的交互。系统的主要功能如下：

(1)系统用户界面：允许用户选择想要的操作。包括学生信息的录入并显示、查询、删除、排序。其中删除操作,查询操作，排序操作都包含子界面。

(2)学生信息成绩录入：用户根据系统提示输入需要录入文件的地址。用户输入后系统把学生的学号，班级，课程名称，学分，平时分，实验分，卷面分等信息依次导入内存并显示，之后返回系统用户界面。其中，学生的综合成绩和实得学分系统会根据用户所提供学生成绩信息根据一定的规则自动计算。

(3)学生基本信息查询：允许用户根据系统提示，进行对学生基本信息的查询：

1.用户可输入一个学号或姓名，查出此生的基本信息并显示。

2.用户可输入一个宿舍号码，可查询出本宿舍的所有的基本信息并显示。

或学生成绩信息的查询：

输入一个学号，可查出此生所有的课程情况。

(4)学生信息的删除：用户可根据学号实现对某个学生所有信息的删除。删除后的信息直接保存。

(5)学生信息的排序：用户能根据提示完成按综合成绩或实得学分升序或降序排序并显示数据。

(6)显示学生信息：系统能显示学生所有的信息。

(7)保存：在用户完成删除操作后自动保存。

(8)退出：完成所有操作后，用户可以推出学生管理系统。

2.2 运行环境

WINDOWS10 系统

Turbo C2.0/Visual C++6.0 编译环境

2.3 开发语言及工具

C 语言、Dev C++集成开发工具

3 总体与函数设计

3.1 功能模块设计

学生管理系统实现对学生实现对学生信息的管理，包括数据录入，学生信息查询，信息删除，学生成绩的排序等功能，系统功能模块如图 3.1 所示：

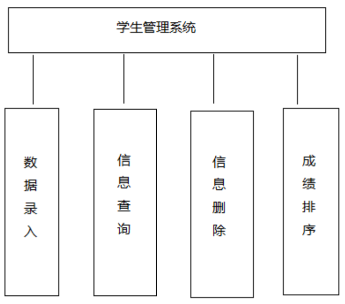

图 3.1 学生管理系统

系统的执行从用户界面的菜单选择开始，允许用户在 0 和 4 之间选择要进行的操作，输入其他字符都是无效的，系统会给出出错的提示信息。若输入 1，则调用“数据录入”模块，录入数据并计算综合成绩。若输入 2，则调用“信息查询”模块，查询学生信息。若输入 3，则调用“信息删除”模块，输入学号即可删除学生信息。若输入 4，则调用“成绩排序”模块，用户可实现选择按综合成绩或实得学分升序或降序排序并显示数据。若输入 0，则调用“退出”模块，退出系统。在录入，查询，删除的模块调用中需要调用“保存”模块，保存所有的学生信息在外部文件中。

3.2 数据结构设计

定义 2 个结构体，分别包括学生基本信息中学号、姓名、宿舍、电话等和学生成绩信息中的学号、科目、成绩等信息，并根据学生信息的多少定义结构体链表，如下所示：

```
   typedef struct studentInformation{      //Student's Basic Information
    int num;
    char name[30];
    char sex[4];
    int room;
    long int phone;
    struct studentInformation *next;
} Basic;
typedef struct gardeInformation{        //Student's Grade Information
    int num;
    char classnum[5];
    char classname[30];
    int credit;
    float daily;
    float experiment;
    float test;
    float Comprehension;
    float truecredit;
    struct gardeInformation *next;
} Grade;
```

3.3 函数和文件的设计

3.3.1 文件及函数组成

该系统采用多文件编写，各功能模块存在主文件中，并由主函数调用。该系统的文件与函数的构成如表 3.1 所示。

表 3.1 文件与函数的构成

文件	函数名或其他成分	功能
	main	实现系统处理的主流程,对

学生管理系统.c		其他函数进行调用
	printmenu	显示主界面
	Editmenu	显示信息修改子界面
	Querymenu	显示信息查询子界面
	Sortmenu	显示信息排序子界面
	judge	确保输入的准确性
	caculate	用于计算学生综合分数与实得学分
	Dataloding	用于数据录入
	QueryGrade	把文件中的信息加载到内存中
	QueryBasic	把文件中的信息加载到内存中
	FindGrade	通过学号查找信息
	FindBasic	通过学号查找信息
	FindBasicRoom	通过宿舍号查找信息
	FindBasicName	通过姓名查找信息
	copy	内容拷贝
	AscendbyComprehension	按综合成绩升序
	DescendbyComprehension	按综合程序降序
	AscendbyTrueCredits	按实得学分升序
	DescendbyTrueCredits	按实得学分升序
	delete	删除学生信息
	del_student	删除学生信息
grade.txt	无	存放学生的成绩选课信息
basic.txt	无	存放学生的基本信息

3.3.2 主函数

函数功能：主函数仅调用用户操作界面（主菜单）函数，由操作界面函数供用户选择相应的功能模块。

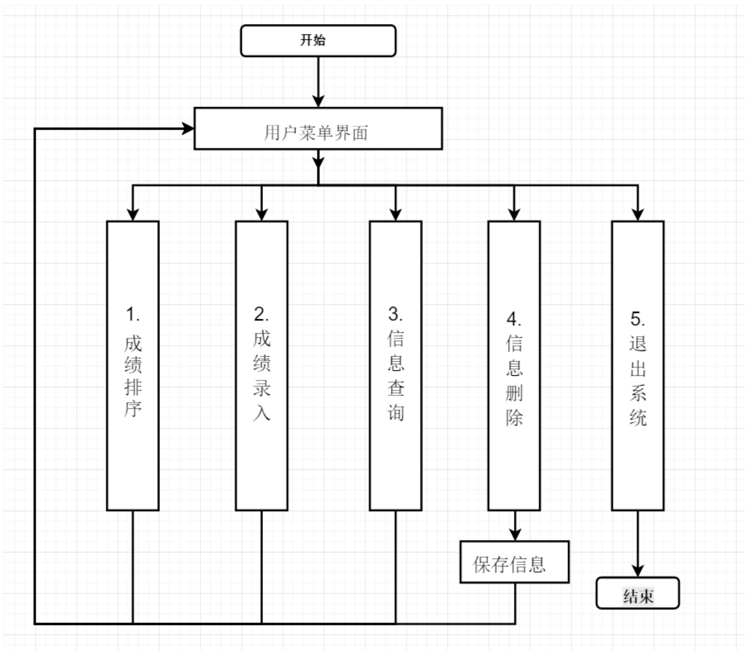

图 3.1 主函数功能图

1.成绩排序对应的模块

函数原型：Grade* AscendbyTrueCredits(Grade* SortHead);

Grade*AscendbyComprehension(Grade* SortHead)

Grade*DescendbyComprehension(Grade* SortHead)

Grade*DescendbyTrueCredits(Grade* SortHead)

功　　能：用户在选择 1 后，根据显示的界面选择相应的排序。

参　　数：Grade 结构体指针

返 回 值：Grade 结构体指针

工作方式：选择功能并从键盘输入文件路径，在界面中显示文件中的信息排序后的结果。

要　　求：根据显示的界面进行选择，若选择错误，则回到选择界面。

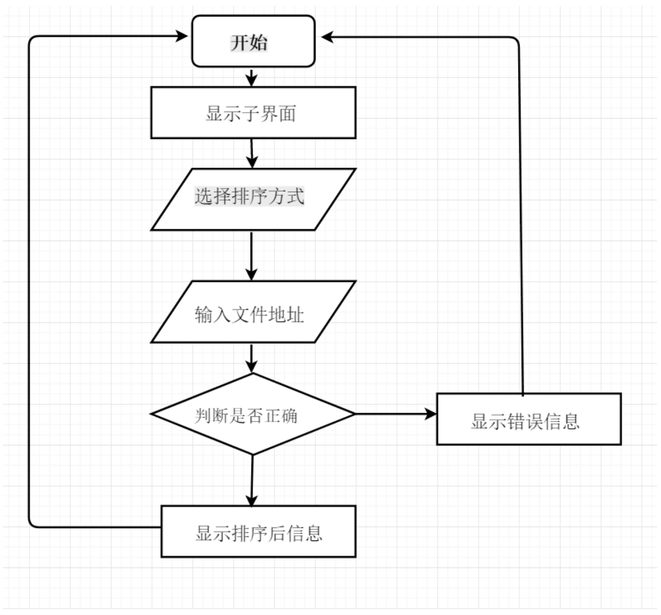

图 3.2 排序功能图

2.成绩查询对应的模块

函数原型：Basic* FindBasic(Basic*head,int num)

Basic* FindBasicName(Basic*head,char name[]);

Basic* FindBasicRoom(Basic*head,int room);

Grade* FindGrade(Grade*head,int num);

功　　能：用户在选择 3 后，根据显示的界面选择查询的路径，选择后输入相应的信息，在界面中显示查询结果。

参　　数：Basic 结构体指针，整型变量

Basic 结构体指针，字符数组

Basic 结构体指针，整型变量

Grade 结构体指针，整型变量

返 回 值：Grade 结构体指针

工作方式：用户选择查询的路径并从键盘输入，选择后输入相应的信息，在界面中显示查询结果。

要　　求：根据显示的界面进行选择，若选择错误，则回到选择界面；若输入的文件不存在，则显示错误，并回到输入环节

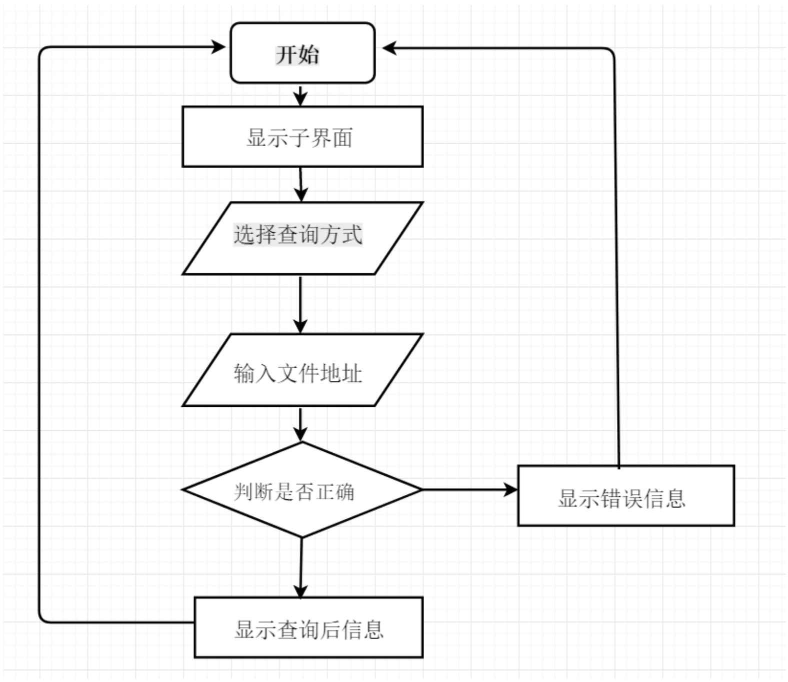

图 3.3 查询功能图

3.成绩录入对应的模块

函数原型：int DataLoding(char *file,Grade* head)

功　　能:用户选择 2 后，根据提示输入文件路径后，系统显示出文件中的信息。

参　　数：字符指针，Grade 机构体指针

返 回 值：整型变量

工作方式：用户从键盘输入文件路径，在界面中显示出来。

要　　求：若输入的文件不存在，则显示错误，并回到输入环节

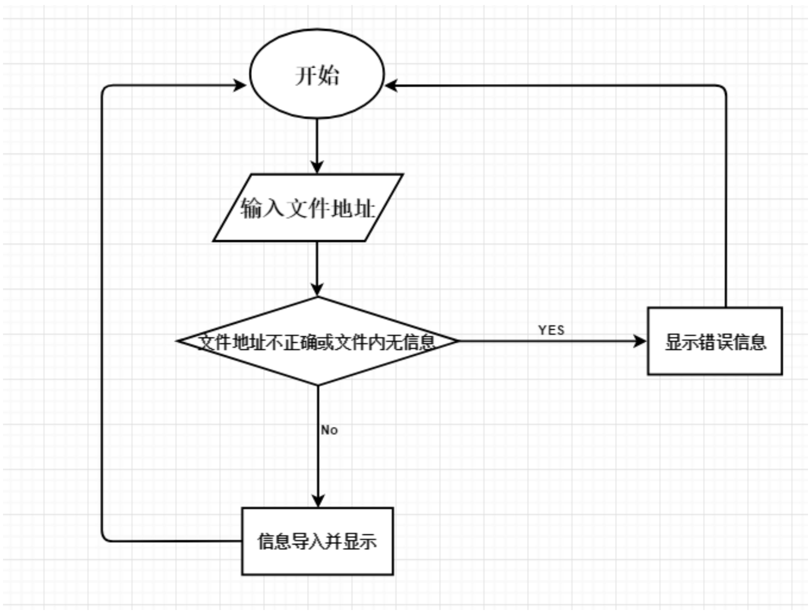

图 3.4 录入功能图

4.信息删除对应的模块

函数原型：int delete（）

功　　能：找到指定学号对应的信息并删除。

参　　数：无

返 回 值：整型变量

工作方式：用户根据输入的学号，删除此学号对应学生的信息。

要　　求：根据显示的界面进行选择，若选择错误，则回到选择界面：若文件中没有数据，则会做出提示。

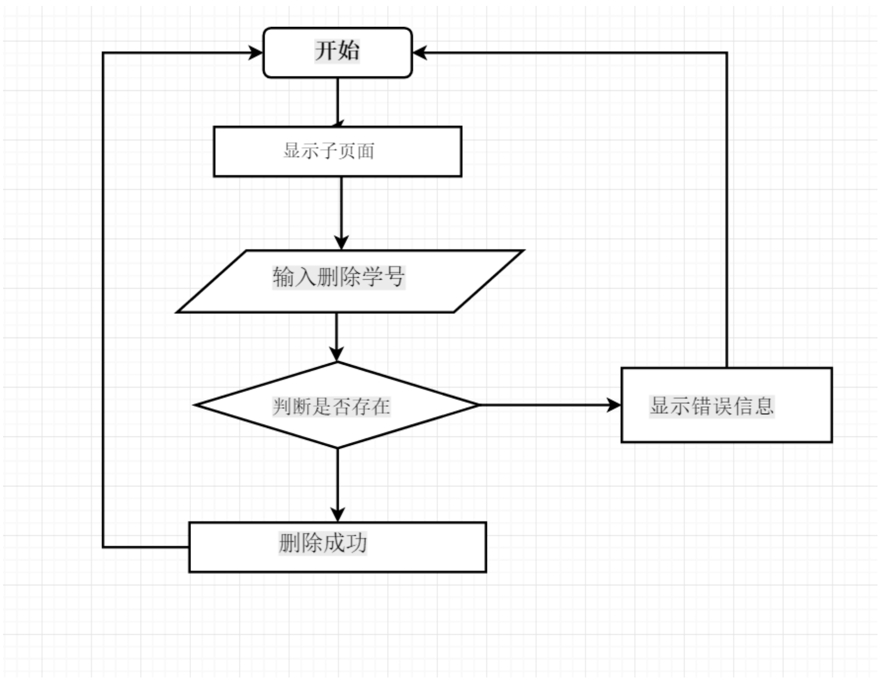

图 3.5 删除功能图

4 具体代码实现

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct studentInformation{          //Student's Basic Information
        int num;
        char name[30];
        char sex[4];
        int room;
        long int phone;
```

```
    struct studentInformation *next;
} Basic;

typedef struct gardeInformation{              //Student's Grade Information
    int num;
    char classnum[5];
    char classname[30];
    int credit;
    float daily;
    float experiment;
    float test;
    float Comprehension;
    float truecredit;
    struct gardeInformation *next;
} Grade;

void printmenu()
{
    printf("**********Student Manage System****************");
    printf("\n**********************************************\n");
    printf("                        Menu Items \n");
    printf("**********************************************\n");
    printf("*       1__Data Sort                             *\n");
    printf("*       2__Data Loading                          *\n");
    printf("*       3__Query Students Information            *\n");
    printf("*       4__Edit Students Information             *\n");
    printf("*       0__Exit System                           *\n");
    printf("**********************************************\n");
}
void Editmenu()
{
    printf("-----------------------------------------------\n");
    printf("                        Edit   Sub_Meun\n");
    printf("-----------------------------------------------\n");
    printf("                    1__Delect Record\n");
    printf("                    2__Return\n");
    printf("-----------------------------------------------\n");
}

void Querymenu()
{
    printf("-----------------------------------------------\n");
    printf("                        Query Sub_Meun\n");
    printf("                    1__Query Grades by No\n");
    printf("                    2__Query Basic Information by No\n");
    printf("                    3__Query Basic Information by Name\n");
    printf("                    4__Query Basic Information by Room\n");
    printf("                    5__Return\n");
    printf("----------------------------------------------\n");
}

void Sortmenu()
{
    printf("----------------------------------------------\n");
    printf("                        Sort Sub_Meun\n");
    printf("                    1__Ascending     by Comprehension\n");
    printf("                    2__Descending    by Comprehension\n");
    printf("                    3__Ascending     by TrueCredits\n");
    printf("                    4__Descending    by TrueCredits\n");
    printf("                    5__Return\n");
    printf("----------------------------------------------\n");
}

int judge(char *x,char a[],char b[]){                                    //guarantee input
right
    return (strlen(x) != 1 || (strcmp(x, a) > 0) || (strcmp(x, b) < 0));
}

int caculate(Grade*p){
    if(!p)
        return 0;
    if (p->experiment == -1)
    {
        p->Comprehension = (p->daily) * 0.30 + (p->test) * 0.70;
        if (p->Comprehension <= 100 || p->Comprehension >= 90)
            p->truecredit = p->credit * 1.00;
        else if (p->Comprehension < 90 || p->Comprehension >= 80)
            p->truecredit = p->credit * 0.80;
        else if (p->Comprehension < 80 || p->Comprehension >= 70)
            p->truecredit = p->credit * 0.75;
        else if (p->Comprehension < 70 || p->Comprehension >= 60)
            p->truecredit = p->credit * 0.60;
        else if (p->Comprehension < 60)
            p->truecredit = p->credit * 0.00;
    }
    else
    {
        p->Comprehension = (p->daily) * 0.15 + (p->experiment) * 0.15 + (p->test) * 0.70;
```

```
            if (p->Comprehension <= 100 || p->Comprehension >= 90)
                p->truecredit = p->credit * 1.00;
            else if (p->Comprehension < 90 || p->Comprehension >= 80)
                p->truecredit = p->credit * 0.80;
            else if (p->Comprehension < 80 || p->Comprehension >= 70)
                p->truecredit = p->credit * 0.75;
            else if (p->Comprehension < 70 || p->Comprehension >= 60)
                p->truecredit = p->credit * 0.60;
            else if (p->Comprehension < 60)
                p->truecredit = p->credit * 0.00;
        }
        return 1;
}

int DataLoding(char *file,Grade* head){                    //DataLoading
        int ret;
        if(!head)       return 0;
        FILE *FP = NULL;
        Grade *p = NULL;
        Grade *point = (Grade *)malloc(sizeof(Grade));
        point = head;
        FP = fopen(file, "r+");
        if(!FP){
            printf("Open File:'%s' fail!\n", file);
            return 0;
        }
        while(!(feof(FP))){
            Grade *Node = (Grade *)malloc(sizeof(Grade));
            ret=fscanf(FP,  "%d%s%s%d%f%f%f",  &Node->num,  Node->classnum,  Node->classname,
&Node->credit, &Node->daily, &Node->experiment, &Node->test);
            if(ret!=EOF&&ret!=0){
            point->next = Node;
            Node->next = NULL;
            point = Node;
            }else{
                return 0;
            }
        }
        p = head->next;
        if(point==head){
            printf("There is no data in file:'%s'\n", file);
        }else{
            printf("Below is the data in the file'%s':\n", file);
            while (p != NULL)
```

```
			{
				caculate(p);
				printf("%d    %-4s %4s %-4d %-.2f %-.2f %-.2f %-.2f %-.2f\n", p->num, p->classnum,
p->classname, p->credit, p->daily, p->experiment, p->test,p->Comprehension,p->truecredit);
				p = p->next;
			}
		}
		fclose(FP);
		return 1;
}

Grade* QueryGrade(char* file,Grade* head){				//Take Grade Information to memory
		int ret;
		if(!head)	return NULL;
		FILE *FP = NULL;
		Grade *point = (Grade *)malloc(sizeof(Grade));
		point = head;
		FP = fopen(file, "r+");
		if(!FP){
			printf("Open File:'%s' fail!\n", file);
			return head;
		}
		while (!(feof(FP)))
		{
			Grade *Node = (Grade *)malloc(sizeof(Grade));
			ret=fscanf(FP, "%d%s%s%d%f%f%f", &Node->num, Node->classnum, Node->classname,
&Node->credit, &Node->daily, &Node->experiment, &Node->test);
			if(ret!=EOF&&ret!=0){
			point->next = Node;
			Node->next = NULL;
			point = Node;
			}else{
				return NULL;
			}
		}
		if (point == head)
		{
			printf("There is no data in file:'%s'\n", file);
			fclose(FP);
			return NULL;
		}else{
			fclose(FP);
			return head;
		}
```

```
}

Basic* QueryBasic(char* file,Basic* head){                    //Take Basic Information to memory
        int ret;
        if(!head)      return NULL;
        FILE *FP = NULL;
        Basic *point = (Basic *)malloc(sizeof(Basic));
        point = head;
        FP = fopen(file, "r+");
        if(!FP){
                printf("Open File:'%s' fail!\n", file);
                return head;
        }
        while (!(feof(FP)))
        {
                Basic *Node = (Basic *)malloc(sizeof(Basic));
                ret=fscanf(FP, "%d%s%s%d%ld", &Node->num, Node->name, Node->sex, &Node->room,
&Node->phone);
                if(ret!=EOF&&ret!=0){
                point->next = Node;
                Node->next = NULL;
                point = Node;
                }else{
                        return NULL;
                }
        }
        if (point == head)
        {
                printf("There is no data in file:'%s'\n", file);
                fclose(FP);
                return NULL;
        }
        else
        {
                fclose(FP);
                return head;
        }
}

Grade* FindGrade(Grade*head,int num){                              //Find the Grade Information by
num
        if(!head)      return NULL;
        Grade *pointer = head->next;
        if(!pointer)      return NULL;
```

```
    Grade *NewHead = (Grade *)malloc(sizeof(Grade));
    NewHead->next = NULL;
    Grade *Gpointer = NewHead;
    do
    {
        if (pointer->num != num)
        {
            pointer = pointer->next;
            continue;
        }
        else
        {
            Grade *Node = (Grade *)malloc(sizeof(Grade));
            Node->num = pointer->num;
            strcpy(Node->classnum, pointer->classnum);
            strcpy(Node->classname, pointer->classname);
            Node->credit = pointer->credit;
            Node->daily = pointer->daily;
            Node->experiment = pointer->experiment;
            Node->test = pointer->test;
            caculate(Node);
            Gpointer->next = Node;
            Node->next = NULL;
            Gpointer = Node;
            pointer = pointer->next;
        }
    } while (pointer);
    return NewHead;
}

Basic* FindBasicRoom(Basic*head,int room){                    //Find the Basic Information by Room
    if(!head)     return NULL;
    Basic *pointer = head->next;
    if(!pointer)      return NULL;
    Basic *NewHead = (Basic *)malloc(sizeof(Basic));
    NewHead->next = NULL;
    Basic *Bpointer = NewHead;
    do
    {
    if (pointer->room!= room)
    {
        pointer = pointer->next;
        continue;
    }
```

```
        else
        {
                Basic *Node = (Basic *)malloc(sizeof(Basic));
                strcpy(Node->name, pointer->name);
                strcpy(Node->sex, pointer->sex);
                Node->num = pointer->num;
                Node->phone = pointer->phone;
                Node->room = pointer->room;
                Bpointer->next = Node;
                Node->next = NULL;
                Bpointer = Node;
                pointer = pointer->next;
        }
}while (pointer);

return NewHead;
}

Basic* FindBasicName(Basic*head,char name[]){                    //Find the Basic Information by name
        if(!head)      return NULL;
        Basic *pointer = head->next;
        if(!pointer)        return NULL;
        Basic *NewHead = (Basic *)malloc(sizeof(Basic));
        NewHead->next = NULL;
        Basic *Bpointer = NewHead;
        do{
                if(strcmp(pointer->name,name)){
                        pointer=pointer->next;
                        continue;
                }else{
                        Basic *Node = (Basic *)malloc(sizeof(Basic));
                        strcpy(Node->name, pointer->name);
                        strcpy(Node->sex, pointer->sex);
                        Node->num = pointer->num;
                        Node->phone = pointer->phone;
                        Node->room = pointer->room;
                        Bpointer->next = Node;
                        Node->next = NULL;
                        Bpointer = Node;
                        pointer = pointer->next;
                }
        } while (pointer);
        return NewHead;
}
```

```
}

Basic* FindBasic(Basic*head,int num){                    //Find the Basic Information by num
        if(!head)  return NULL;
        Basic *pointer = head->next;
        if(!pointer)        return NULL;
        Basic *NewHead = (Basic *)malloc(sizeof(Basic));
        NewHead->next = NULL;
        Basic *Bpointer=NewHead;
        do
        {
                if(pointer->num != num){
                        pointer = pointer->next;
                        continue;
                }else{
                        Basic *Node = (Basic *)malloc(sizeof(Basic));
                        strcpy(Node->name, pointer->name);
                        strcpy(Node->sex, pointer->sex);
                        Node->num = pointer->num;
                        Node->phone = pointer->phone;
                        Node->room = pointer->room;
                        Bpointer->next = Node;
                        Node->next = NULL;
                        Bpointer = Node;
                        pointer = pointer->next;
                }
        } while (pointer);
        return NewHead;
}

void copy(Grade *Temp,Grade* Pointer){                              //Copy Pointer's Information to Temp's
Information
        Temp->num = Pointer->num;
        strcpy(Temp->classnum, Pointer->classnum);
        strcpy(Temp->classname, Pointer->classname);
        Temp->credit = Pointer->credit;
        Temp->daily = Pointer->daily;
        Temp->experiment = Pointer->experiment;
        Temp->test = Pointer->test;
        Temp->Comprehension = Pointer->Comprehension;
        Temp->truecredit = Pointer->truecredit;
}

Grade*DescendbyTrueCredits(Grade* SortHead){                     //Descending by True Credits
        if(!SortHead)      return NULL;
```

```
    if(!SortHead->next)         return NULL;
    Grade *CalPointer = SortHead->next;
    Grade *SortPointer = SortHead->next;
    Grade *SortPointer2 = NULL;
    Grade *TempNode = (Grade *)malloc(sizeof(Grade));
    int size = 0, i = 0, j = 0, k = 0;
    if (!CalPointer)
    { //No data
        return NULL;
    }
    while (CalPointer)
    {
        caculate(CalPointer);
        CalPointer = CalPointer->next;
        size++;
    }
    Grade Temp[size];
    for (i = 0; i < size; i++)
    {
        copy(&Temp[i], SortPointer);
        SortPointer = SortPointer->next;
    }
    for (j = 0; j < size - 1; j++)
    {
        for (k = 0; k < size - 1 - j; k++)
        {
            if ((Temp[k].truecredit) < (Temp[k + 1].truecredit))
            {
                copy(TempNode, &Temp[k + 1]);
                copy(&Temp[k + 1], &Temp[k]);
                copy(&Temp[k], TempNode);
            }
        }
    }
    SortPointer2 = SortHead->next;
    for (i = 0; i < size; i++)
    {
        copy(SortPointer2, &Temp[i]);
        SortPointer2 = SortPointer2->next;
    }

    return SortHead;
}
```

```
Grade*DescendbyComprehension(Grade*   SortHead){                          //Descending   by
Comprehension
    if(!SortHead)     return NULL;
    if(!SortHead->next)         return NULL;
    Grade *CalPointer = SortHead->next;
    Grade *SortPointer = SortHead->next;
    Grade *SortPointer2 = NULL;
    Grade *TempNode = (Grade *)malloc(sizeof(Grade));
    int size = 0, i = 0, j = 0, k = 0;
    if (!CalPointer)
    { //No data
        return NULL;
    }
    while (CalPointer)
    {
        caculate(CalPointer);
        CalPointer = CalPointer->next;
        size++;
    }
    Grade Temp[size];
    for (i = 0; i < size; i++)
    {
        copy(&Temp[i], SortPointer);
        SortPointer = SortPointer->next;
    }
    for (j = 0; j < size - 1; j++)
    {
        for (k = 0; k < size - 1 - j; k++)
        {
            if ((Temp[k].Comprehension) < (Temp[k + 1].Comprehension))
            {
                copy(TempNode, &Temp[k + 1]);
                copy(&Temp[k + 1], &Temp[k]);
                copy(&Temp[k], TempNode);
            }
        }
    }
    SortPointer2 = SortHead->next;
    for (i = 0; i < size; i++)
    {
        copy(SortPointer2, &Temp[i]);
        SortPointer2 = SortPointer2->next;
    }
```

```
        return SortHead;
}

Grade*AscendbyComprehension(Grade* SortHead){                    //Ascendint by Comprehension
        if(!SortHead)      return NULL;
        if(!SortHead->next)          return NULL;
        Grade *CalPointer = SortHead->next;
        Grade *SortPointer = SortHead->next;
        Grade *SortPointer2=NULL;
        Grade *TempNode = (Grade*)malloc(sizeof(Grade));
        int size = 0,i=0,j=0,k=0;
        if(!CalPointer){                //No data
                return NULL;
        }
        while (CalPointer)
        {
                caculate(CalPointer);
                CalPointer = CalPointer->next;
                size++;
        }
        Grade Temp[size];
        for (i = 0; i < size;i++){
                copy(&Temp[i], SortPointer);
                SortPointer = SortPointer->next;
        }
        for (j = 0; j < size -1;j++){
                for (k = 0; k < size -1 - j;k++){
                        if((Temp[k].Comprehension)>(Temp[k+1].Comprehension)){
                                copy(TempNode, &Temp[k+1]);
                                copy(&Temp[k + 1], &Temp[k]);
                                copy(&Temp[k], TempNode);
                        }
                }
        }
        SortPointer2 = SortHead->next;
        for (i = 0; i < size;i++){
                copy(SortPointer2, &Temp[i]);
                SortPointer2 = SortPointer2->next;
        }
                return SortHead;
}

Grade* AscendbyTrueCredits(Grade* SortHead){                    //Ascending by True Credits
        if(!SortHead)      return NULL;
        if(!SortHead->next)          return NULL;
        Grade *CalPointer = SortHead->next;
        Grade *SortPointer = SortHead->next;
        Grade *SortPointer2 = NULL;
        Grade *TempNode = (Grade *)malloc(sizeof(Grade));
        int size = 0, i = 0, j = 0, k = 0;
        if (!CalPointer)
        { //No data
                return NULL;
        }
        while (CalPointer)
        {
                caculate(CalPointer);
                CalPointer = CalPointer->next;
                size++;
        }
        Grade Temp[size];
        for (i = 0; i < size; i++)
        {
                copy(&Temp[i], SortPointer);
                SortPointer = SortPointer->next;
        }
        for (j = 0; j < size - 1; j++)
        {
                for (k = 0; k < size - 1 - j; k++)
                {
                        if ((Temp[k].truecredit) > (Temp[k + 1].truecredit))
                        {
                                copy(TempNode, &Temp[k + 1]);
                                copy(&Temp[k + 1], &Temp[k]);
                                copy(&Temp[k], TempNode);
                        }
                }
        }
        SortPointer2 = SortHead->next;
        for (i = 0; i < size; i++)
        {
                copy(SortPointer2, &Temp[i]);
                SortPointer2 = SortPointer2->next;
        }
        return SortHead;
}

int delete ()                                        //Delete one Information
```

```
{
        int del_student(FILE * f, FILE * fp);
        FILE *f, *fp;
        int ret=0;
        f = fopen("grade.txt", "r");
        fp = fopen("basic.txt", "r");
        if (fp == NULL || f == NULL)
        {
                return 0;
        }
        ret=del_student(f, fp);
        if(ret==-1){
                return -1;
        }else if(ret==0)        return 0;
        return 1;
}

int del_student(FILE *f, FILE *fp)             //Delete one Information
{       int ret=0;
        int Num=0;
        FILE *p, *q;
        Basic *head_1, *point_1;
        Grade *head_2, *point_2;
        head_1 = (Basic *)malloc(sizeof(Basic));
        head_2 = (Grade *)malloc(sizeof(Grade));
        head_1->next = NULL;
        head_2->next = NULL;
        head_1 = QueryBasic("basic.txt", head_1);
        head_2 = QueryGrade("grade.txt", head_2);
        if(head_1&&head_2){
        point_1 = head_1->next;
        point_2 = head_2->next;
flag5:          printf("\nPlease input the student's number you want to Delete:\n");
        ret=scanf("%d",&Num);
        if(!ret){
                printf("Please check your input!\n");
                fflush(stdin);
                goto flag5;
        }
        if(!point_1&&!point_2){
                printf("Open file error! Please check your path!\n");
                return 0;
        }else{
        if(point_1){
```

```
        p = fopen("c.txt", "a+");
        while (point_1)
        {
                if (point_1->num != Num)
                {
                        if (point_1->next)
                                fprintf(p, "%d %s %s %d %ld\n", point_1->num, point_1->name, point_1->sex,
point_1->room, point_1->phone);
                        else
                                fprintf(p, "%d %s %s %d %ld", point_1->num, point_1->name, point_1->sex,
point_1->room, point_1->phone);
                }
                point_1 = point_1->next;
        }
        fclose(fp);
        fclose(p);
        remove("basic.txt");
        rename("c.txt", "basic.txt");
        }
        if(point_2){
        q = fopen("d.txt", "a+");
        while (point_2)
        {
                if (point_2->num != Num)
                {
                        if (point_2->next)
                                fprintf(q, "%d %s %s %d %f %f %f\n", point_2->num, point_2->classnum,
point_2->classname, point_2->credit, point_2->daily, point_2->experiment, point_2->test);
                        else
                                fprintf(q, "%d %s %s %d %f %f %f", point_2->num, point_2->classnum,
point_2->classname, point_2->credit, point_2->daily, point_2->experiment, point_2->test);
                }
                point_2 = point_2->next;
        }
        fclose(f);
        fclose(q);
        remove("grade.txt");
        rename("d.txt", "grade.txt");
        }
      }
}else{
        if(!head_1&&!head_2){
                printf("There is no data in the file!\n");
                return -1;
```

```
    }else{
    if(head_1){
        point_1 = head_1->next;
        if(!point_1){
            printf("Open file error!\n");
            return 0;
        }
        p = fopen("c.txt", "a+");
    while (point_1)
    {
        if (point_1->num != Num)
        {
            if (point_1->next)
                fprintf(p, "%d %s %s %d %ld\n", point_1->num, point_1->name, point_1->sex, 
point_1->room, point_1->phone);
            else
                fprintf(p, "%d %s %s %d %ld", point_1->num, point_1->name, point_1->sex, 
point_1->room, point_1->phone);
        }
        point_1 = point_1->next;
    }
    fclose(fp);
    fclose(p);
    remove("basic.txt");
    rename("c.txt", "basic.txt");
    }
    if(head_2){
        point_2 = head_2->next;
        if(!point_2){
            printf("Open file error!\n");
            return 0;
        }
        q = fopen("d.txt", "a+");
    while (point_2)
    {
        if (point_2->num != Num)
        {
            if (point_2->next)
                fprintf(q, "%d %s %s %d %f %f %f\n", point_2->num, point_2->classnum, 
point_2->classname, point_2->credit, point_2->daily, point_2->experiment, point_2->test);
            else
                fprintf(q, "%d %s %s %d %f %f %f", point_2->num, point_2->classnum, 
point_2->classname, point_2->credit, point_2->daily, point_2->experiment, point_2->test);
        }
```

```
		point_2 = point_2->next;
	}
	fclose(f);
	fclose(q);
	remove("grade.txt");
	rename("d.txt", "grade.txt");
	}
		}
			}
  return 1;
}

int main(){
	char X[3],X2[3],X3[3],x,x2,x3;
	char file[30],file2[30],Basicfile[30],SearchName[30],Sortfile[30];
	int Querynum,cnt=0,SearchRoom,ret,ret1=0,ret2=0,ret3=0;
	float truecredit=0;
	Grade *head = (Grade *)malloc(sizeof(Grade));
	head->next = NULL;
	Grade *head2 = (Grade *)malloc(sizeof(Grade));
	head2->next = NULL;
	Grade *SortHead = (Grade *)malloc(sizeof(Grade));
	SortHead->next = NULL;
	Basic *BasicHead = (Basic *)malloc(sizeof(Basic));
	BasicHead->next = NULL;
	Basic *BasicHead2 = (Basic *)malloc(sizeof(Basic));
	BasicHead2->next = NULL;
	Basic *BasicHead3 = (Basic *)malloc(sizeof(Basic));
	BasicHead3->next = NULL;
	Basic *BasicHead4 = (Basic *)malloc(sizeof(Basic));
	BasicHead4->next = NULL;
	Grade *pointer = NULL;
	Grade *pointer2 = NULL;
	Grade *AscendSort = NULL;
	Grade *DescendSort = NULL;
	Basic *Bpointer = NULL;
	Basic *BasicPointer = NULL;
	Basic *BasicPointer2 = NULL;
	Basic *BasicPointer3 = NULL;
	Basic *BasicPointer4 = NULL;
flag1:printmenu();
	printf("Please Select an Option(0-4):");
	fflush(stdin);
	scanf("%s", X);
```

```
        while (judge(X,"4","0"))
        {
                printf("Input Error! Please Input Again!\n");
                fflush(stdin);
                scanf("%s", X);
        }
        x = X[0];
        switch(x){
                case '1'://Sort
flag3:    Sortmenu();
                printf("Please Select an Option(1-5):");
                fflush(stdin);
                scanf("%s", X3);
                while (judge(X3, "5", "1"))
                {
                        printf("Input Error! Please Input Again!\n");
                        fflush(stdin);
                        scanf("%s", X3);
                }
                x3 = X3[0];
                switch(x3){
                case '1':            //Ascending    by Comprehension
                printf("Please input the file path!\n");
                fflush(stdin);
                scanf("%s", Sortfile);
                SortHead = QueryGrade(Sortfile, SortHead);
                if(!SortHead){
                        printf("There is No data in the file\n");
                        goto flag3;
                }
                SortHead=AscendbyComprehension(SortHead);
                if(!SortHead){
                        printf("There is No data in the file\n");
                        goto flag3;
                }
              AscendSort = SortHead->next;
              while (AscendSort)
              {
                      printf("%d    %-4s  %4s  %-4d  %-.2f  %-.2f  %-.2f  %-.2f  %-.2f\n",  AscendSort->num,
AscendSort->classnum,        AscendSort->classname,        AscendSort->credit,        AscendSort->daily,
AscendSort->experiment, AscendSort->test, AscendSort->Comprehension, AscendSort->truecredit);
                      AscendSort = AscendSort->next;
              }
              goto flag3;
              break;

                case '2':
                printf("Please input the file path!\n");
                fflush(stdin);
                scanf("%s", Sortfile);
                SortHead = QueryGrade(Sortfile, SortHead);
                if(!SortHead){
                        printf("There is No data in the file\n");
                        goto flag3;
                }
                if (!SortHead)
                {
                        printf("There is No data in the file\n");
                        goto flag3;
                }
                SortHead = DescendbyComprehension(SortHead);
                if(!SortHead){
                        printf("There is No data in the file\n");
                        goto flag3;
                }
                DescendSort = SortHead->next;
                while (DescendSort)
                {
                        printf("%d   %-4s %4s %-4d %-.2f %-.2f %-.2f %-.2f %-.2f\n", DescendSort->num,
DescendSort->classnum, DescendSort->classname, DescendSort->credit, DescendSort->daily,
DescendSort->experiment, DescendSort->test, DescendSort->Comprehension,
DescendSort->truecredit);
                        DescendSort = DescendSort->next;
                }
                goto flag3;
                break;

                case'3':
                printf("Please input the file path!\n");
                fflush(stdin);
                scanf("%s", Sortfile);
                SortHead = QueryGrade(Sortfile, SortHead);
                if(!SortHead){
                        printf("There is No data in the file\n");
                        goto flag3;
                }
                SortHead = AscendbyTrueCredits(SortHead);
                if(!SortHead){
```

```
			printf("There is No data in the file\n");
			goto flag3;
		}
		AscendSort = SortHead->next;
		while (AscendSort)
		{
			printf("%d   %-4s  %4s  %-4d  %-.2f  %-.2f  %-.2f  %-.2f  %-.2f\n",  AscendSort->num,
AscendSort->classnum,        AscendSort->classname,        AscendSort->credit,        AscendSort->daily,
AscendSort->experiment, AscendSort->test, AscendSort->Comprehension, AscendSort->truecredit);
			AscendSort = AscendSort->next;
		}
		goto flag3;
		break;
		case '4':
		printf("Please input the file path!\n");
		fflush(stdin);
		scanf("%s", Sortfile);
		SortHead = QueryGrade(Sortfile, SortHead);
		if(!SortHead){
			printf("There is No data in the file\n");
			goto flag3;
		}
		SortHead = DescendbyTrueCredits(SortHead);
		if(!SortHead){
			printf("There is No data in the file\n");
			goto flag3;
		}
		DescendSort = SortHead->next;
		while (DescendSort)
		{
			printf("%d   %-4s  %4s  %-4d  %-.2f  %-.2f  %-.2f  %-.2f  %-.2f\n",  DescendSort->num,
DescendSort->classnum,     DescendSort->classname,     DescendSort->credit,     DescendSort->daily,
DescendSort->experiment,                DescendSort->test,                DescendSort->Comprehension,
DescendSort->truecredit);
			DescendSort = DescendSort->next;
		}
		goto flag3;
		break;
		case '5':
		goto flag1;
		break;
		}
		break;
```

```
            case '2'://Data Loading
            printf("Please input the file path!\n");
            fflush(stdin);
            scanf("%s", file);
            if(!DataLoding(file,head)){
                printf("There is no data in the file!\n");
                goto flag1;
            }else{
                goto flag1;
            }
            break;

            case '3'://Query
flag2:     Querymenu();
            printf("Please Select an Option(1-5):");
            fflush(stdin);
            scanf("%s", X2);
            while (judge(X2,"5","1"))
            {
                printf("Input Error! Please Input Again!\n");
                fflush(stdin);
                scanf("%s", X2);
            }
            x2 = X2[0];
            switch(x2){
                case '1':
                //search by num(Grade Information)
flag6:         printf("Please input the Number you want to search:");
                ret1=scanf("%d", &Querynum);
                while(!ret1){
                    printf("Please check you input!\n");
                    fflush(stdin);
                    goto flag6;
                }
                printf("Please input the file path!\n");
                fflush(stdin);
                scanf("%s", file2);
                strcpy(Basicfile, "basic.txt");
                head2=QueryGrade(file2,head2);
                BasicHead = QueryBasic(Basicfile, BasicHead);
                if(!head2||!BasicHead){
                    printf("There is No data in the file\n");
                    goto flag2;
                }
                pointer = FindGrade(head2, Querynum);
                BasicPointer=FindBasic(BasicHead, Querynum);
                if(!pointer||!BasicPointer){
                    printf("Please check the File\n");
                    goto flag2;
                }
                if(!(pointer->next)||!(BasicPointer->next)){
                    printf("There is no such num in file\n");
                    goto flag2;
                }else{
                    printf("Student    Num:%d        Student    Name:%s\n",    BasicPointer->next->num,
BasicPointer->next->name);
                    pointer2 = pointer->next;
                    while(pointer2){
                        printf("ClassNum:%s              ClassName:%s              Comprehension:%.2f
TrueCredits:%.2f\n",     pointer2->classnum,     pointer2->classname,     pointer2->Comprehension,
pointer2->truecredit);
                        cnt++;
                        truecredit += pointer2->truecredit;
                        pointer2 = pointer2->next;
                    }
                    printf("Total Class:%d          Total Credits:%.2f\n", cnt, truecredit);
                }
                  goto flag2;
                  break;

                case'2':      //search by num(Basic Information)
flag7:                 printf("Please input the Number you want to search:");
                    ret2=scanf("%d", &Querynum);
                    if(!ret2){
                        printf("Please check your input!\n");
                        fflush(stdin);
                        goto flag7;
                    }
                    strcpy(Basicfile, "basic.txt");
                    BasicHead2=QueryBasic(Basicfile, BasicHead2);
                    BasicPointer2 = FindBasic(BasicHead2, Querynum);
                    if(!BasicHead2||!BasicPointer2){
                        printf("There is No data in the file\n");
                        goto flag2;
                    }else{
                    if(!(BasicPointer2->next)){
                        printf("There is no such num in file\n");
                        goto flag2;
                    }else{
                         printf("Student's  Num:%d         Student's  Name:%s      Sex:%s     Room:%d
PhoneNumber:%ld\n",              BasicPointer2->next->num,              BasicPointer2->next->name,
BasicPointer2->next->sex, BasicPointer2->next->room, BasicPointer2->next->phone);
                    }
                         }
                    goto flag2;
                    break;

                    case'3':       //search by name (Basic Information)
                    printf("Please input Student's Name you want to search:");
                    scanf("%s", SearchName);
                    strcpy(Basicfile, "basic.txt");
                    BasicHead3 = QueryBasic(Basicfile, BasicHead3);
                    BasicPointer3 = FindBasicName(BasicHead3, SearchName);
                    if (!(BasicHead3))
                    {
                         printf("There is No data in the file\n");
                         goto flag2;
                    }
                    else
                    {
                         if (!(BasicPointer3->next))
                         {
                              printf("There is no such Name in file\n");
                              goto flag2;
                         }
                         else
                         {
                              printf("Student's   Num:%d             Student's   Name:%s        Sex:%s
Room:%d            PhoneNumber:%ld\n",    BasicPointer3->next->num,    BasicPointer3->next->name,
BasicPointer3->next->sex, BasicPointer3->next->room, BasicPointer3->next->phone);
                         }
                    }
                    goto flag2;
                    break;
                    case '4':            //search by room (Basic Information)
flag8:                  printf("Please input The Room you want to search:");
                         ret3=scanf("%d", &SearchRoom);
                         if(!ret3){
                         printf("Please check your input!\n");
                         fflush(stdin);
                         goto flag8;
                    }
```

```
                        strcpy(Basicfile, "basic.txt");
                        BasicHead4 = QueryBasic(Basicfile, BasicHead4);
                        BasicPointer4 = FindBasicRoom(BasicHead4, SearchRoom);
                        if(!BasicPointer4){
                            printf("There is no data in the file\n");
                            goto flag2;
                        }
                        Bpointer = BasicPointer4->next;
                        if(!Bpointer){
                            printf("There is no such Room in file\n");
                            goto flag2;
                        }else{
                            while (Bpointer)
                            {
                                printf("Student's   Num:%d           Student's   Name:%s       Sex:%s
Room:%d      PhoneNumber:%ld\n", Bpointer->num, Bpointer->name, Bpointer->sex, Bpointer->room,
Bpointer->phone);
                                Bpointer = Bpointer->next;
                            }
                        }
                        goto flag2;
                        break;
                        case '5':
                        goto flag1;
                        break;
            }
          break;

          case '4'://Delete Information
flag4:    Editmenu();
     printf("Please Select an Option(1-2):");
     fflush(stdin);
     scanf("%s", X2);
     while (judge(X2, "2", "1"))
     {
          printf("Input Error! Please Input Again!\n");
          fflush(stdin);
          scanf("%s", X2);
     }
     x2 = X2[0];
     switch(x2){
          case '1':
          ret=delete();
          if(ret==0){
              printf("No data in the file!\n");
              goto flag4;
          }else if(ret==-1){
              printf("There is no data in the file!\n");
              goto flag4;
          }else if(ret==1){
              printf("Delete Done!\n");
              goto flag4;
          }
          break;
          case '2':
          goto flag1;
          break;
     }
          case '0':
          printf("Thanks to use!\n");
          exit(1);
     }
     }
```

5 课程设计总结

通过这次的课程设计，我们对 c 语言得到了更加深入的了解和熟练的运用。巩固了专业知识，虽然在编译程序的过程中遇到了很多问题，但是我们组内成员通过深入讨论，提出解决方案，并不断对程序进行调试最终完成了这个学生管理系统。

在调试过程中我们认识到了数据结构的灵活性与严谨性的重要性。因为在本次的编译过程中在对删除功能进行编译过程中，由于我们的灵活性和严谨性出了问题，从而拖慢了进度。

在本次的课程设计中我们也认识到了自己的对一些知识点掌握的不熟练，如做到有关文本的部分，才发现自己对文本运用的不熟练。所以在以后的学习中，我们要集中精力、端正态度，争取把知识学得更扎实、更全面。但是也有高兴的地方就是写算法时我们对题的要求掌握的清楚明了，知道应该运用哪些知识来解决问题。

在写算法时要注意细节，对每一步要分析到位，否则可能出现错误。通过上机编写算法了解到熟练掌握知识是很重要的，如果不能熟练掌握，就会对题目感到无从下手。有时在程序设计中，不一定非要全部采用自己编写的源代码；如果想要较好地利用某种编程语言自带的库函数，则必须充分的理解该编程语言库函数的功能特点。

总之此次实训，受益良多，让我们意识到只有自己多实践才能获得更高的能力经验，学得更多的知识与技巧。如何让书本上与老师课上讲的知识点运用到实践中是我们以后学习中的重点。

6 参考文献

[1]王景丽，姚景丽. C 语言应用案例教程[M]. 北京：清华大学出版社，2016.1

[2]刘振安，孙忱，刘燕君．C 程序设计课程设计[M]．北京：机械工业出版社，2004.12．

[3]周林．C 语言程序设计实训教程[M]．北京：机械工业出版社，2014.10．