# 实验三 单链表的应用

开发环境：CLion 2023.2.2

语言标准：C++11

1. **算法设计与实现**

实验要求使用数据结构：单链表来实现一个学生成绩管理系统。

1.学生基本信息的存储

每个学生的基本信息包括字符数组变量ID、字符数组变量name和双精度浮点型变量grade(此项目保存单科成绩)，运用结构体变量struct Student储存学生的基本信息；同时再用另一个结构体struct StudentNode储存结点，每个结点中包括学生的基本信息：struct Student、结点的标号：int length（在头结点中定义该变量存储学生信息条数；在其他结点中定义该变量存储信息的下标，第一个下标从1开始计数）和指向下一个结点的指针struct StudentNode\* next:

typedef struct Student {  
 char name[24]{};  
 char ID[24]{};  
 double grade{};  
} Student;

typedef struct StudentNode {  
 Student info;  
 int length{};  
 *//头结点的length表示信息条数，学生结点的length表示下标（从1开始）* struct StudentNode\* next{};  
} StudentNode, \*StudentList;

2.基本操作的定义

为实现学生管理系统，定义以下基本操作，相关操作的功能在注释中标出：

void updateIndex (StudentList &L);*//更新结点index的信息*void inputStudent (StudentList &L);*//输入学生信息*char\* locateID (StudentList L, int index);*//查找表内给定序号的学生ID，返回ID*bool addStudent (StudentList &L, const Student &information);*//添加学生成员信息*void showStudent (StudentList L);*//展示系统内信息*void initializeList (StudentList &L);*//初始化学生链表*void removeStudent (StudentList &L);*//删除学生成员信息*Student\* findStudent (StudentList &L);*//查找给定ID的学生信息，返回学生信息*void sortStudentsAscending (StudentList &L, int length);*//将学生信息按成绩升序排序*

3.基本操作的实现

(1)updateIndex

更新结点index的信息，可以通过工作指针从第一个元素开始先遍历一遍链表，同时用i来对当前元素的length进行赋值，通过循环和i的自增实现对每一个元素下标的赋值；在遍历完链表后，对头结点的length进行赋值，此时i的值和元素个数的值相同。

实现细节如下：

void updateIndex (StudentList &L) {  
 StudentNode\* p = L; int i = 0;  
 while (p->next) {  
 p = p->next;  
 i++;  
 p->length = i;  
 }  
 L->length = i;  
}

(2)inputStudent

输入学生基本信息的函数使用一个do-while循环实现，当输入信息的ID与已有信息的ID重复时会被要求重新输入信息，判断ID是否重复用到另外一个定义的函数addStudent。

循环内层有两个do-while循环，分别用来判断输入的ID和grade是否合法：

ID合法条件：ID全为数且长度等于10（不包括’\0’）；

Grade合法条件：grade范围在0~100之间；

I．

判断ID是否合法，思路是动态分配一个字符指针指向一个存储ID的空间并将输入的字符串存入这个空间，并判断以下两个命题：

通过循环遍历和指针自增来分别判断①每个字符是数字字符；

用cstdlib库中的strlen函数②判断长度合法；

当①②同时为真时ID的输入合法。

II．

判断grade是否合法，只需一个if语句即可判断。

实现细节如下：

void inputStudent (StudentList &L) {  
 cout << "\n<addStudent>\n";  
 Student newStudent; static bool flag1;  
 do {  
 cout << "Name: ";  
 fflush(**stdout**);  
 scanf("%s", newStudent.name);  
 *//输入的ID:全数字、长度为10* static bool isLengthValid, isInputValid;  
 do {  
 isLengthValid = false;  
 isInputValid = true;  
 printf("ID: ");  
 fflush(**stdout**);  
 *// 限制输入ID的格式为全数字，且ID长度固定为10* char \*inputID = new char[24];  
 scanf("%s", inputID);  
 *// 输入长度超过11则停止输入(包括字符'/0')* char \*id = inputID;*// 检查指针。* if (strlen(inputID) == 10) {  
 isLengthValid = true;  
 } *// 输入ID的长度小于10则不满足输出，应重新输出* while (id && \*id!='\0') {  
 *// p指针每次后移检查字符是否为数字* if (\*id < '0' || \*id > '9') {  
 isInputValid = false;  
 break;  
 }  
 id++;  
 }  
 if (isInputValid && isLengthValid) {  
 strcpy(newStudent.ID, inputID);  
 } else {  
 if (!isInputValid) {  
 cout << "ID Input Invalid\n"<<"Please type in again!\n";  
 }  
 if (!isLengthValid) {  
 cout << "ID Length Invalid\n"<<"Please type in again!\n";  
 }  
 }  
 delete[] inputID;  
 } while (!isLengthValid || !isInputValid);  
 *//输入的成绩：大于等于0且小于等于100* static bool isGradeValid = true;  
 do {  
 cout << "grade: ";  
 cin >> newStudent.grade;  
 if (newStudent.grade < 0 || newStudent.grade > 100) {  
 isGradeValid = false;  
 cout << "Grade Invalid\n" << "Please type in again!\n";  
 }  
 } while (!isGradeValid);  
  
 flag1 = addStudent(L, newStudent);  
 if (flag1) {  
 cout << "\n<addStudent> SUCCESS!\n";  
 } else {  
 cout << "\n<addStudent> FAILED! ID REPETITION!\n"  
 << "Please type in new data.\n";  
 }  
 } while (!flag1);  
}

(3)locateID

该函数输入两个参数StudentList L和int index，找到链表L中下标为index的结点并返回ID信息，使用一个循环遍历链表并每次进行ID比对，返回字符指针（找不到时返回空指针）。

实现细节如下：

char\* locateID (StudentList L, int index) {  
 *//第一个元素的下标记作1。* StudentNode\* p = L; char \*targetID = new char[16];  
 if (index > L->length) {  
 return targetID;  
 }  
 for (int i = 0; i < index; i++) {  
 p = p->next;  
 }  
 strcpy(targetID, (p->info).ID);  
 return targetID;  
}

(4)addStudent

在(2)中已说明该函数判断ID是否重复，通过一个循环遍历链表并每次使用strcmp函数将输入的ID和已有ID进行对比，若重复则返回false，不重复则在原有链表上以头插法插入新信息并更新下标，并返回true。

实现细节如下：

bool addStudent (StudentList &L, const Student &information) {  
 *//采用头插法建立链表。* for (int i = 1; i <= L->length; i++) {  
 if (strcmp(information.ID, locateID(L, i)) == 0) {  
 return false;  
 }  
 }  
 auto \*NewStudent = new StudentNode;  
 NewStudent->next = L->next;  
 L->next = NewStudent;  
 strcpy((NewStudent->info).name, information.name);  
 strcpy((NewStudent->info).ID, information.ID);  
 (NewStudent->info).grade = information.grade;  
 updateIndex(L);  
 return true;  
}

(5)showStudent

该函数通过循环遍历链表并每次打印每条信息，并打印出学生信息的条数。

实现细节如下：

void showStudent (StudentList L) {  
 StudentNode \*p = L->next;  
 int i = 1;  
 cout << "<showStudent>\n";  
 cout << "—————————————————————————————————————————————\n";  
 cout << "| INDEX | ID | NAME | GRADE |\n";  
 cout << "—————————————————————————————————————————————\n";  
 while (p) {  
 if (i < 10) {  
 cout << "| <00"<<i<<"> |";  
 } else if (i < 100) {  
 cout << "| <0"<<i<<"> |";  
 } else {  
 cout << "| <"<<i<<"> |";  
 }  
 printf("%-10s|%-16s|%-7.2f|\n", (p->info).ID, (p->info).name, (p->info).grade);  
 p = p->next;  
 i++;  
 }  
 cout << "—————————————————————————————————————————————\n";  
 printf("Numbers of student: %d\n\n", L->length);  
}

(6)initializeList

初始化链表，通过new关键字动态分配头结点，并初始化值。

实现细节如下：

void initializeList (StudentList &L) {  
 L = new StudentNode;  
 *//头结点的创建和命名。* L->next = nullptr;  
 L->length = 0; *//头结点成员length存储学生信息数目，学生结点length存储下标（从1开始）* (L->info).grade = 0;  
}

(7)removeStudent

该函数使用do-while循环判断每次输入的ID是否在原有链表中存在，其中还添加了退出操作的功能。若不存在则重新输入；若存在则删除该信息并更新下标。判断ID是否存在则使用一个循环和strcmp函数实现。

存在条件：循环结束时工作指针非空。

不存在条件：循环结束时工作指针为空。

实现细节如下：

void removeStudent (StudentList &L) {  
 cout << "\n<removeStudent>\n";  
 char \*rTargetID = new char[24];  
 static bool flag2;  
 do {  
 flag2 = **FAIL**;  
 cout << "removeID: "; cin>>rTargetID;  
 StudentNode \*p = L->next;  
 StudentNode \*pre = L; *//pre指向p指向结点的前驱结点* while (p) {  
 if (strcmp(rTargetID, (p->info).ID) == 0) {  
 flag2 = **SUCCESS**;  
 cout << "\n<removeStudent> SUCCESS!\n";  
 pre->next = p->next;  
 updateIndex(L);  
 }  
 p = p->next;  
 pre = pre->next;  
 }  
 if (!flag2) {  
 int commandR;  
 cout << "\n<removeStudent> FAILED!\n"  
 << "Do you want to quit?";  
 do {  
 cout << "\nType: <0> to QUIT/ <1> to CONTINUE.\n";  
 cin >> commandR;  
 if (commandR != 0 && commandR != 1) {  
 cout << "Command Error!\n";  
 } else if (commandR == 1){  
 cout << "CONTINUE!\n";  
 } else {  
 cout << "QUIT!\n";  
 flag2 = true;  
 }  
 } while (commandR != 0 && commandR != 1);  
 }  
 delete p;  
 } while (!flag2);  
 delete[] rTargetID;  
}

(8)findStudent

该函数使用do-while循环判断每次输入的ID是否在原有链表中存在，若不存在则重新输入；若存在则返回该结点的信息。判断ID是否存在则使用一个循环和strcmp函数实现。

存在条件：循环结束时工作指针非空

不存在条件：循环结束时工作指针为空。

实现细节如下：

Student\* findStudent (StudentList &L) {  
 static bool flag3;  
 static Student \*key;  
 do {  
 flag3 = **FAIL**;  
 char\* fTargetID = new char[24];  
 cout << "Find ID: ";  
 cin >> fTargetID;  
 StudentNode \*p = L->next;  
 while (p) {  
 if(strcmp((p->info).ID, fTargetID) == 0) {  
 flag3 = **SUCCESS**;  
 cout << "<Instruction> FOUND!\n";  
 cout << "\nID: " << (p->info).ID  
 << "\nName: " << (p->info).name  
 << "\nGrade: " << (p->info).grade << endl << endl;  
 break;  
 }  
 p = p->next;  
 }  
 if (!flag3) {  
 int commandF;  
 cout << "<Instruction> NOT FOUND!\n"  
 << "Do you want to quit?";  
 do {  
 cout << "\nType: <0> to QUIT/ <1> to CONTINUE.\n";  
 cin >> commandF;  
 if (commandF != 0 && commandF != 1) {  
 cout << "Command Error!\n";  
 } else if (commandF == 1){  
 cout << "CONTINUE!\n";  
 } else {  
 cout << "QUIT!\n";  
 flag3 = true;  
 }  
 } while (commandF != 0 && commandF != 1);  
 }  
 delete[] fTargetID;  
 } while (!flag3);  
 return key;  
}

(9)sortStudentAscending

该函数通过递归实现链表的冒泡排序。交换值由于不能直接将结构体交换，故使用指针进行操作。排序完一趟后返回表长减去1的表再次进行排序，递归出口为表长等于0。

实现细节如下：

void sortStudentsAscending (StudentList &L, int length) {  
 *//使用冒泡排序的递归算法。  
 //递归出口：待排序的表长为0* if (length == 0) {  
 updateIndex(L);  
 cout << "\n<sortStudentAscending>\n";  
 cout << "Data has been sorted.\n\n";  
 return;  
 }  
 StudentList p = L->next; int i = 1; bool isSorted = false;  
 while (p->next) {  
 if ((p->info).grade > (p->next->info).grade) {  
 auto \*temp = new Student;  
 Student \*a = &p->info;  
 Student \*b = &(p->next->info);  
 \*temp = \*a;  
 \*a = \*b;  
 \*b = \*temp;  
 delete temp;  
 isSorted = true;  
 }  
 p = p->next;  
 i++;  
 if (i == length) {  
 break;  
 }  
 }  
 if (!isSorted) {  
 return;  
 }  
 *//排序好表长为length的一趟后，从头排序表长为length-1一趟，以此递归* sortStudentsAscending(L, length-1);  
}

基本操作的定义和实现被分别封装在头文件Manage.h，源文件Manage.cpp中，在源文件main.cpp中可以进行调用：

*Manage.h*

*//  
// Created by YangYujie on 2023/11/1.  
//*#ifndef **MANAGE\_H**#define **MANAGE\_H**typedef struct Student {  
 char name[24]{};  
 char ID[24]{};  
 double grade{};  
} Student;  
  
typedef struct StudentNode {  
 Student info;  
 int length{};  
 *//头结点的length表示信息条数，学生结点的length表示下标（从1开始）* struct StudentNode\* next{};  
} StudentNode, \*StudentList;  
  
void updateIndex (StudentList &L);*//更新结点index的信息*void inputStudent (StudentList &L);*//输入学生信息*char\* locateID (StudentList L, int index);*//查找表内给定序号的学生ID，返回ID*bool addStudent (StudentList &L, const Student &information);*//添加学生成员信息*void showStudent (StudentList L);*//展示系统内信息*void initializeList (StudentList &L);*//初始化学生链表*void removeStudent (StudentList &L);*//删除学生成员信息*Student\* findStudent (StudentList &L);*//查找给定ID的学生信息，返回学生信息*void sortStudentsAscending (StudentList &L, int length);*//将学生信息按成绩升序排序*#endif *//MANAGE\_H*

*Manage.cpp*

*//  
// Created by YangYujie on 2023/11/1.  
//*#include <iostream>  
#include <string>  
#include <cstdio>  
#include <cstring>  
#include "Manage.h"  
  
#define **SUCCESS** true  
#define **FAIL** false  
  
using namespace std;

*/\**

*…*

*(implementations)*

*…*

*\*/*

4.源文件调用

源文件调用头文件Manage.h等实现学生成绩管理系统，main函数主要实现用户与系统的交互功能，使用do-while循环实现用户指令输入进行不同的操作。

具体实现如下:

*main.cpp*

*//  
// Created by YangYujie on 2023/10/31.  
//*#include "Manage.h"  
using namespace std;  
int main() {  
 StudentList test;  
 cout << "< Student Grades Management System >\n\n";  
 initializeList(test);  
 cout << "The system is successfully initialized!\n\n";  
 int command;  
 do {  
 cout << "Command:\n"  
 << "<0>——————showStudent\n"  
 << "<1>——————addStudent\n"  
 << "<2>——————removeStudent\n"  
 << "<3>——————findStudent\n"  
 << "<4>——————sortStudentsAscending\n"  
 << "<5>——————programTerminate\n\n"  
 << "Please input the command: ";  
 cin >> command;  
 switch (command) {  
 default: {  
 cout << "Command Invalid!\n"  
 << "Please input the valid command!\n";  
 }  
 break;  
  
 case 0: {  
 showStudent(test);  
 }  
 break;  
  
 case 1: {  
 inputStudent(test);  
 showStudent(test);  
 }  
 break;  
  
 case 2: {  
 removeStudent(test);  
 showStudent(test);  
 }  
 break;  
  
 case 3: {  
 findStudent(test);  
 }  
 break;  
  
 case 4: {  
 sortStudentsAscending(test, test->length);  
 showStudent(test);  
 }  
 break;  
  
 case 5: {  
 cout << "The program terminated!\n";  
 } break;  
 }  
 } while (command != 5);  
 return 0;  
}

1. **程序运行结果**

点击“调试”后，控制台界面显示“学生管理系统”的用户操作界面，即指令输入界面：



此时在控制台输入数字“1”即可进行操作：增加学生信息。依次输入学生的姓名、学号和成绩，若输入的数据合法，则该信息被添加到系统中，并以表格形式展示当前系统中含有的信息。

图 1 指令输入界面

数据合法条件：

1. ID：

使用字符数组进行存储，每个字符必须为数字字符，且字符串的长度应等于10（不包括字符’\0’）；

输入的新ID与已有的ID不能重复。

1. grade：使用float变量进行存储，应大于等于0或者小于等于100。

以ID为5601121157，grade为95.40为例，依次输入后发现信息被成功添加进系统中：

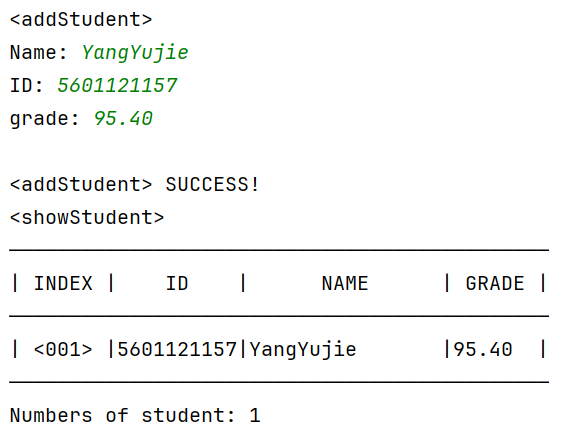


图 2 成功增加学生信息

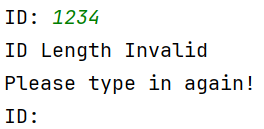
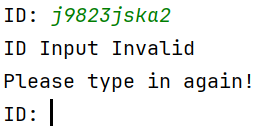
若输入的ID长度不为10，或输入的字符中含有非数字字符，则系统会提示输入长度非法或输入格式非法，需要重新输入：

图 3 学号长度非法

图 4 学号输入格式非法

同样的，如果输入的ID与已有信息中的ID重复，系统会提示ID输入重复，需要重新输入ID：

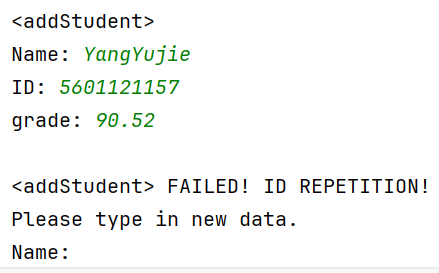


图 5 ID输入重复提示

输入的grade若超出范围会提示重新输入：

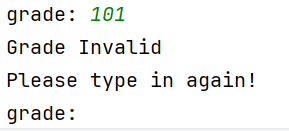


图 6 成绩输入非法

操作成功后系统会自动返回指令输入界面，输入指令“2”即可进行删除学生信息的操作。输入想要删去的学生学号，系统会匹配相同的ID信息并将其从系统中删除，如果系统内没有该学号学生的信息，系统会提示重新输入或选择退出该操作。

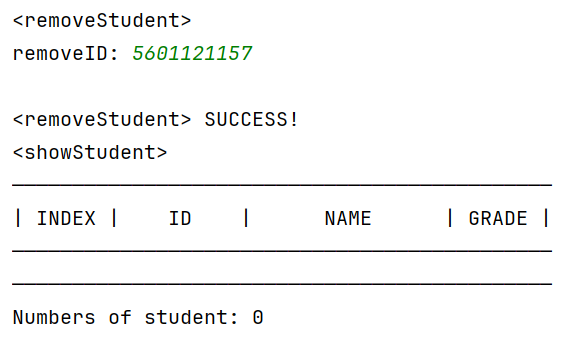
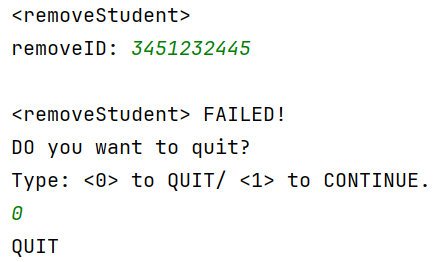


图 7 成功删除学生信息

图 8 删除操作的退出

输入指令“3”可进行查找学生信息的操作，实现的逻辑与删除信息操作类似，如果要查找的ID在系统中存在，则显示要查找的学生的信息。

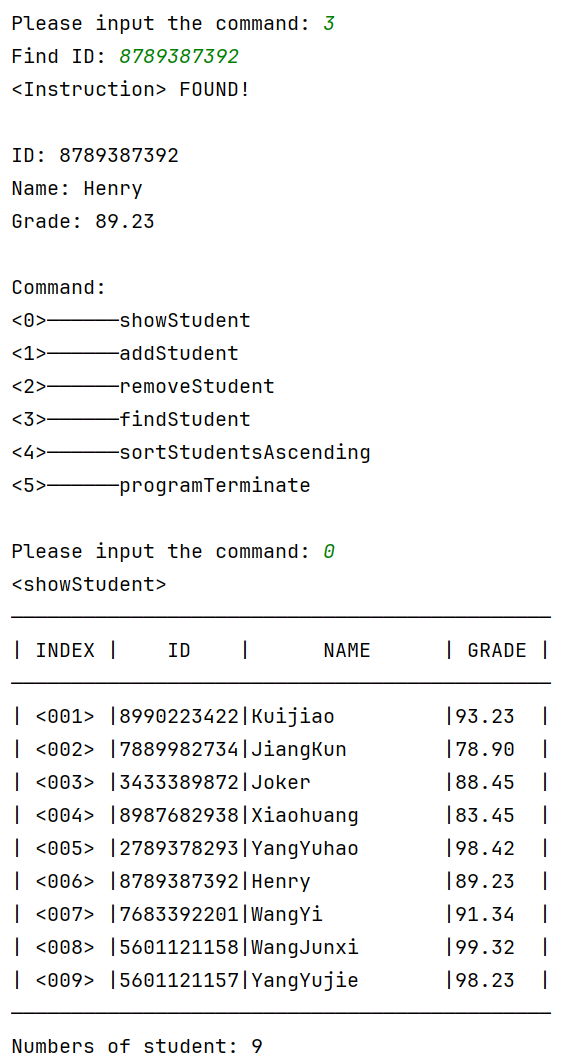


图 9 成功查找学生信息

输入指令“4”即可对现有的学生信息进行升序排序，输入以下学生信息：

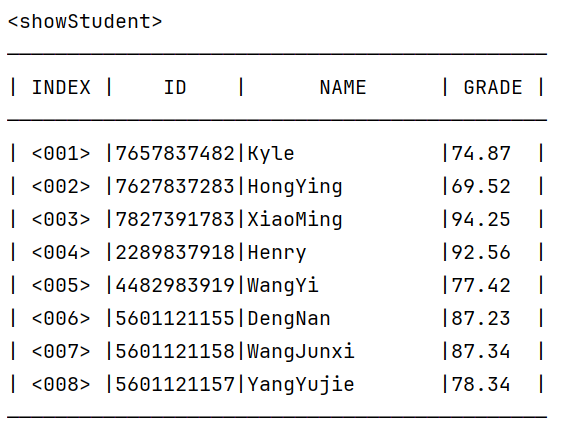


图 10 升序排序前的信息

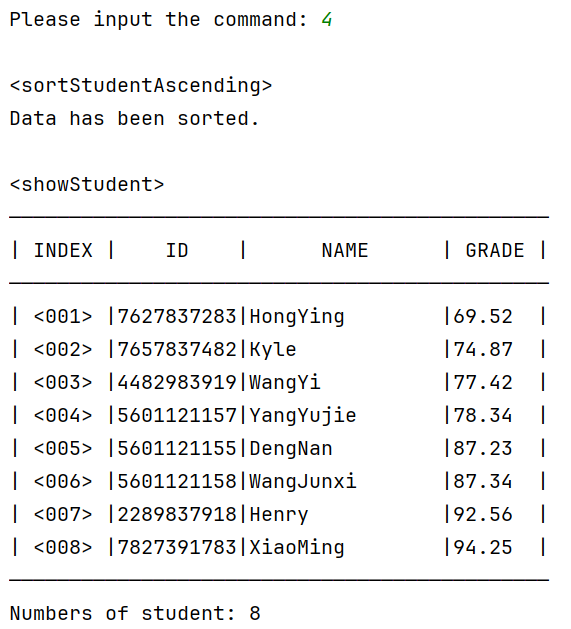


图 11 升序排序信息

1. **实验结果分析与总结**

该程序使用单链表的数据结构实现了一个简单的学生成绩管理系统。基本操作的实现和单链表的运用没有太大问题，在实际调试后总结以下几个可以改进的方面：

1. 该程序实现的数据结构将数据仅仅是存储在内存，关闭程序后数据即丢失；若结合数据库存储数据，可以实现数据的长期存储和管理，还可以进一步实现批量存储等便捷操作，提高程序功能性。
2. 该程序指令输入界面存在漏洞：当输入的指令类型为非整型时，程序会进入死循环，应再开发输入指令的判断函数来限制指令格式的键入；还可以进一步将指令输入进行优化：比如点击相关按钮进行操作。
3. 学生信息的显示格式是通过预先计算字符串长度固定了表格的大小，可以进一步优化。
4. 学生信息的姓名输入格式没有做限制，但是可以进一步优化：比如不能输入连续的空格等。
5. 排序算法采取的是冒泡排序，在实际应用中，随着信息条数的增加，排序算法的开销会增加，可以寻找时间复杂度更低的算法进行排序的设计。
6. 该程序设计理念是面向过程的程序设计，虽然将数据类型、操作实现等封装进了头文件用到了一些面向对象的理念，但是还可以进一步优化：比如使用面向对象的程序设计，提升可维护性和程序运行效率。
7. 在设计程序时，可以遵循软件工程的设计流程来进行设计。