1. **要求**

定义学生（student）类，其中至少包括姓名、性别、学号、学院、系、班级、宿舍、籍贯和联系电话。

功能要求：

1、设计菜单实现功能选择；

2、输入功能：输入学生信息，并保存到文件中；

3、能按照学生的不同信息排序整理，如按学号、学院、宿舍等；

4、能根据学号修改学生信息；

5、能根据学号删除学生信息；

6、查询功能：

1）能够根据学号查询学生信息；

2）能够根据姓名、学号、学院、系、班级、宿舍等分项信息查询学生记录；

3）按照学院、系或者班级统计学生人数、统计各籍贯地学生的情况。

7、所有的增加、修改、删除能同步到文件；也从文件读取数据到程序。

1. **运行环境**

Linux + GCC

1. **分析**

**3.1 学生信息管理系统的主要功能**

**学生信息管理系统主要功能为：**对学生的姓名、性别、学号、学院、系、班级、宿舍、籍贯和联系电话等信息进行读取，存储，增加，删除，修改，查询，排序，统计。详细的系统功能结构为图1所示。

学生信息管理系统（LMS）

新建模块

统计模块

检索与修改模块

显示模块

排序模块

图1 系统结构图

**系统各模块的功能具体描述为：**

**1、显示模块**

依次打印所有学生的信息。

完整显示所有学生的信息，适合用于检查最近的修改情况。

**2、新建模块**

新増一名学生到系统。

让用户交互式输入学生的各种信息，输入完成之后可选择继续输入或退出该模块。

**3、统计模块**

可以针对某一属性进行学生人数统计。

用户选择一个类别，比如学院、系或者班级，系统按此给出学生人数的分类。

**4、检索与修改模块**

可以查找学生，了解特定学生信息并作出修改或删除。

用户输入字符串，将匹配到任何相关的信息并呈现出来，用户再从中选择一个，然后对其进行查询更改等操作。

1. **排序模块**

可以根据学号、学院、宿舍等进行排序整理。

输入排序的依据，即可完成排序。

**3.2 系统分析及设计**

**系统涉及对象主要有两个类：**学生类和学生链表类。其中学生类作为节点，负责存储每个学生的信息，学生链表类组成完整链表，并提供函数以实现各种功能。

另外还有Unit类，Chart类作为链表，用于暂时存储数据，可为StudentList的

统计模块提供暂存服务。

系统的主要的类结构如图2所示。

StudentList

Student

Chart

Unit

图2 系统主要类结构图

用文本文件进行数据的保存，需要保存的数据包括学生类的所有信息，这些将在学生链表类的构造与析构时完成。

**3.3系统的实现**

**（1）类的编写**

系统工程名为：SIMS。包含了Student类，StudentList类，Chart类，Unit类。

具体类结构声明如下：

* **Student类：**

|  |
| --- |
| #include <iostream>  #include <cstring>  using namespace std;  class Student  {  public:  Char name[20]="000";  char num[15]="000";  char sex[5]="000";  char college[20]="000";  char faculty[20]="000";  char classes[10]="000";  char dorm[10]="000";  char origin[10]="000";  char tele[15]="000";  Student \*next=NULL;  Student(){} //default constructor  Student(char na[20],char nu[15],char se[5],char co[20],char fa[20],char cl[10],char dor[10],char ori[10],char te[15],Student\* p):next(p){  strcpy(name,na);  strcpy(num,nu);  strcpy(sex,se);  strcpy(college,co);  strcpy(faculty,fa);  strcpy(classes,cl);  strcpy(dorm,dor);  strcpy(origin,ori);  strcpy(tele,te);  }  void Disp(); //show its information  void Del\_next(); //delete next node  void Edit(); //edit interactively  }; |

* **StudentList类**

|  |
| --- |
| class StudentList  {  public:  StudentList();  ~StudentList();  /\*Functions\*/  void Show\_Menu();  void Add\_Node();  void Display();  void Search(); //Searching and edit,display or delete  void Save(); //automatically called by destructor  void Sort(); //interactively sort by any variable  void Refresh(); //check `length' and `end'  void Statistic(); //show the number of each class,college,dorm,etc.  private:  /\*StudentList is a linked list made up of Students \*/  Student \*head; //head of the linked list  Student \*end; //tail of the linked list  int length=0; //used for sorting  }; |

* **Chart类：**

|  |
| --- |
| class Chart{  public:  Chart(){head=new Unit;end=head;}  ~Chart(){  Unit \*pos = head;  Unit \*temp;  while(pos!=NULL)  {  temp = pos;  pos = pos->next;  //cout<<"deleted!"<<endl;  delete temp;  }  }  void App(char k[20]){end->next=new Unit(k);end=end->next;}  void Add(char k[20]){  int f=0;  for(Unit \*p= head->next;p!=NULL;p=p->next){  if(strcmp(p->key,k)==0){  p->Add();  f=1;  }  }  if(f==0)  App(k);  }  void Disp(){  for(Unit \*p= head->next;p!=NULL;p=p->next)  p->Disp();  }  private:  Unit \*head;  Unit \*end;  }; |

* **Unit类：**

|  |  |
| --- | --- |
| class Unit{  public:  Unit(){};  Unit(char k[20]){strcpy(key,k);}  void Add(){value++;}  void Disp(){cout<<key<<'\t'<<value<<endl;}  Unit \*next=NULL;  char key[20];  private:  int value=1;  }; |  |

**（2）链表的使用**

系统实现采用文件的输入输出流对文本数据进行读取与写入，但是由于学生信息、是一个数据的集合，于是对数据的存储组织使用了单向链表。

因为学生信息管理系统在登录、查找、修改、添加的时候都需要处理大量的数据，所以使用链表十分必要。在Student类的基础上定义一个对应的StudentList类来管理学生数据，具体的结构声明如下：

|  |
| --- |
| class StudentList{  public:  StudentList();  ~StudentList();  void Show\_Menu();  void Add\_Node();  void Display();  void Search(); //Searching and edit,display or delete  void Save(); //automatically called by destructor  void Sort(); //interactively sort by any variable  void Refresh(); //check `length' and `end'  void Statistic(); //show the number of each class,college,dorm,etc.  private:  /\*StudentList is a linked list made up of Students \*/  Student \*head; //head of the linked list  Student \*end; //tail of the linked list  int length=0; //used for sorting  }; |

在运用时，令当前学生的next结点指向新的学生结点，即结点的指针next保存新的学生结点的地址（如下图3所示），以此类推，所有学生信息就通过链表的形式串联起来了。

Student1 Student2 Studentn

data

data

data

......

\*next

\*next

\*next

图3 学生链表的建立

学生信息管理系统的信息的管理就具体表现为链表的操作。拿学生信息来说，学生信息的显示、统计、排序、查找、修改、添加和删除与链表的此类功能一一对应。

* **学生信息的显示：**

**StudentList类包含一个Display()完成显示功能。**

**Student包含Disp()函数，可以按格式输出单个学生的信息。**

|  |
| --- |
| **void Student::Disp(){**  **cout<<"name:"<<name<<'\t';**  **cout<<"num:"<<num<<'\t';**  **cout<<"sex:"<<sex<<'\t';**  **cout<<"college:"<<college<<'\t';**  **cout<<"faculty:"<<faculty<<'\t';**  **cout<<"class:"<<classes<<'\t';**  **cout<<"dorm:"<<dorm<<'\t';**  **cout<<"origin:"<<origin<<'\t';**  **cout<<"Tel:"<<tele<<'\t';**  **cout<<endl;**  **}** |

**如果判断出没有学生，则会提示**

|  |
| --- |
| if(head==end){  cout<<"Empty..."<<endl;  } |

**StudentList的Display()负责依次调用Disp()，既可完成输出全部信息。**

|  |
| --- |
| for(Student \*pos= head->next;pos!=NULL;pos=pos->next){  pos->Disp(); *//display each node*  } |

* **学生信息的增加:**

提供用户交互式输入信息并创建学生类的功能。

由StudentList提供Add\_Node()方法

执行类似下面的主要代码来完成信息的输入

|  |
| --- |
| char na[20];  cout<<"please enter student's name(<20 words):";  cin.getline(na,20); |

然后增加到尾部

|  |
| --- |
| end->next=new Student(na,nu,se,co,fa,cl,dor,ori,te,NULL);  end = end->next; |

用while括起来，添加完之后将询问是否继续添加。默认(直接回车)为不继续。若需要继续，输入y或Y

|  |
| --- |
| char quit[100];  cout<<"continue or not?,y/N"<<endl;  cin.getline(quit,100);  if(strlen(quit)==1){  switch(quit[0]){ *//enter to stop,or y/Y to continue*  case'Y':  case'y':break;  case'n':  case'N':  default:return;  }  }  else  return; |

用char数组代替char避免用户输入太多入缓冲区的问题

增加学生的流程图如下:

是

否

开始

用户依次输入信息

用户选择继续？

继续添加

结束

* **学生信息的统计:**

为暂时存储信息，另构建两个类:Unit，Chart

Unit作为存储信息的单元，而Chart是以Unit为节点组成的链表，并提供增加节点，改变节点的方法。当需要某个对Unit的某值+1时，Chart先根据类别名查找对应Unit，若没有找到则在尾部新建一个Unit对应于此类，并初始化其值为1。以后的统计只需使其值+1。

|  |
| --- |
| *//定义于Chart内*  void App(char *k*[20]){end->next=new Unit(k);end=end->next;}  void Add(char *k*[20]){  int f=0;  for(Unit \*p= head->next;p!=NULL;p=p->next){  if(strcmp(p->key,k)==0){  p->Add();  f=1;  }  }  if(f==0)  App(k);  } |

Chart也提供Display函数。将用此函数输出统计结果

|  |
| --- |
| void Disp(){  for(Unit \*p= head->next;p!=NULL;p=p->next)  p->Disp();  } |

Unit，Chart的成员函数构造与Student，StudentList类似

在StudentList的Statistic()中，首先由用户输入需要根据什么进行分类统计

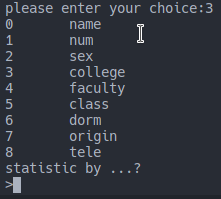


图3 学生信息统计界面

然后根据输入情况调用Chart的Add输出统计结果

|  |
| --- |
| Chart c; *//create a new linked list to store data*  cout<<"0\tname\n1\tnum\n2\tsex\n3\tcollege\n4\tfaculty\n5\tclass\n6\tdorm\n7\torigin\n8\ttele"<<endl;  cout<<"statistic by ...?\n>";  /\*此处读取输入\*/  char staby[100];  cin.getline(staby,100);  if(strlen(staby)==1 && staby[0]>='0' && staby[0]<='8'){//确保输入合法  cout<<"-\*-\*-\*-\*-\*-\*-\*-\*-"<<endl;  */\*此段见下面\*/*  cout<<"-\*-\*-\*-\*-\*-\*-\*-\*-"<<endl;  }  else  cout<<"invalid input..."<<endl; |

|  |
| --- |
| switch(staby[0])  {  case '0':  for(Student \*p = head->next;p!=NULL;p=p->next)  c.Add(p->name);  break;  /\* ... \*/  }  c.Disp(); |

* **学生信息的查找(含删除，修改)：**

考虑到对用户的友好度，参考多数用户的习惯，本程序集成所有搜索于同一输入位置，然后列出搜索结果，由用户选择出其中一个学生来选择是删除还是编辑还是什么。

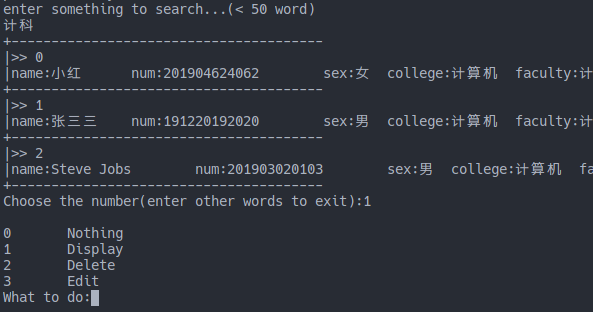


图4 学生信息查找界面

先创建一个动态数组用来存储筛选出的节点。从StudentList类的第一个节点开始，会 将节点的所有成员变量和用户输入比较，如果一致，扩容动态数组，把节点加入动态数 组，然后检查下一个节点。

|  |
| --- |
| char q[50];  int s=0;  Student\*\* qresult=new Student\*[0]; *//store query results*  cout<<"enter something to search...(< 50 word)"<<endl;  cin.getline(q,50);  for(Student \*p = head;p!=end;p=p->next)  {  */\*match all possible attributes;aiming for the previous node to*  *make deleting convenient\*/*  if(strcmp(p->next->name,q)==0 || strcmp(p->next->num,q)==0  || strcmp(p->next->sex,q)==0 || strcmp(p->next->college,q)==0  || strcmp(p->next->faculty,q)==0  || strcmp(p->next->classes,q)==0  || strcmp(p->next->dorm,q)==0 || strcmp(p->next->origin,q)==0  || strcmp(p->next->tele,q)==0 )  {  Student\*\* presult=new Student\*[++s];  for(int i=0;i<s-1;i++){  presult[i]=qresult[i];  }  presult[s-1]=p;  delete[] qresult;  qresult=presult;  cout<<"+---------------------------------------\n|>> "<<s-1<<"\n|";  p->next->Disp();  }  } |

如果有搜索结果，根据是否为1判断是否需要用户指定学生

|  |
| --- |
| if(s==0){  cout<<"Not Found"<<endl;  }  else{  cout<<"+---------------------------------------"<<endl;  Student\* pp;  if(s==1){  pp=qresult[0]; *//needn't choose manually*  }  else{  cout<<"Choose the number(enter other words to exit):";  char chosen[100]; *//manage long wrong input*  cin.getline(chosen,100);  if(chosen[1]!='\0' || chosen[0]<'0' || chosen[0]>=s+'0'){ */\*avoid*  *infinite error\*/*  cout<<"Value not listed.exiting..."<<endl;  delete[] qresult;  return;  }  pp=qresult[chosen[0]-'0'];  } |

然后由用户指定操作

|  |
| --- |
| cout<<"\n0\tNothing\n1\tDisplay\n2\tDelete\n3\tEdit\n";  cout<<"What to do:";  char method[100];  cin.getline(method,100);  if(method[1]!='\0'|| method[0]<'0' || method[0]>'3'){  cout<<"invalid input..."<<endl;  }  else{  switch(method[0]-'0')  {  case 0:break;  case 1:pp->next->Disp();break;  case 2:pp->Del\_next();Refresh();break;  case 3:pp->next->Edit();break;  default:cout<<"invalid input..."<<endl;  }  }  }  delete[] qresult; |

需要注意的是为了能较简单地实现删除指定节点的功能，选择的节点应该是真正匹配节点的前一个。如果不是删除，就使用next指向的节点。

**学生信息查找的流程图如下：**

>1

是

开始

用户输入要查找的内容

显示查找菜单

显示搜索结果列表

有多少匹配结果?

0

提示无搜索结果

用户从中选择目标

执行对应操作

结束

1

提示用户选择操作:查看，删除，修改，或什么也不做

用户选择了删除/编辑/显示?

否

搜索完成后，用户将给出操作。编辑、删除、显示的函数写在Student类中。如果选择编辑，就对下一个节点调用调用Edit()，列出所有可编辑的属性，用户选择后再提示用户输入替换后的值。然后用输入值替换原先值。如果选择删除，调用Del\_next()则删除下一节点，指针指为下下节点。如果选择显示，就调用Disp()。

|  |
| --- |
| void Student::Edit(){  cout<<"\n0\t\t1\t\t\t2\t3\t\t4\t\t5\t\t6\t\t7\t\t8"<<endl; *//trim in most conditions*  Disp();  cout<<"Which To Change:";  char attr[100];  cin.getline(attr,100);  if(attr[1]!='\0'|| attr[0]<'0' || attr[0]>'8'){  cout<<"invalid input..."<<endl;  }  else{  cout<<"enter the new value:";  switch(attr[0]-'0')  {  case 0:cin.getline(name,20);;break;  case 1:cin.getline(num,15);break;  case 2:cin.getline(sex,5);break;  case 3:cin.getline(college,20);break;  case 4:cin.getline(faculty,20);break;  case 5:cin.getline(classes,10);break;  case 6:cin.getline(dorm,10);break;  case 7:cin.getline(origin,10);break;  case 8:cin.getline(tele,15);break;  default:cout<<"invalid input..."<<endl;  }  }  } |

* **学生信息的排序:**

排序前后会分别调用一次类内成员函数Refresh()，用于确保end节点以及length值正确。

首先列出菜单，由用户指定排序的依据:

|  |
| --- |
| Refresh();  cout<<"sort by ...?"<<endl;  cout<<"0\tname\n1\tnum\n2\tsex\n3\tcollege\n4\tfaculty\n5\tclass\n6\tdorm\n7\torigin\n8\ttele\n"<<endl;  char sortby[100];  cin.getline(sortby,100);  if(strlen(sortby)==1 && sortby[0]>='0' && sortby[0]<='8'){  switch(sortby[0]){  */\*略，此处代码参见下文\*/*  }  Refresh();  cout<<"sort success!"<<endl;  }  else  cout<<"invalid input..."<<endl; |

使用冒泡排序，此处需要length来控制循环次数。优点是只需考虑前后节点的交换，不用考虑跨节点的交换。

|  |
| --- |
| case '0':  */\*Bubble Sorting\*/*  for(int i=0;i<length-1;i++)  {  for(Student \*q=head;q->next->next!=NULL;q=q->next)  {  if(strcmp(q->next->name,q->next->next->name)>0)  {  Student \*temp;  temp=q->next->next->next;  q->next->next->next=q->next;  q->next=q->next->next;  q->next->next->next=temp;  }  }  }  break;  */\*后略\*/* |

**（3）交互界面以及登录菜单的实现**

系统运行开始的界面如图5所示：

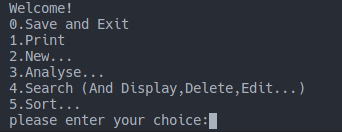


图5 开始菜单界面

主要通过选择结构和循环结构实现界面的前进和后退。例如，菜单界面出6个选择：1.保存并退出，2.新建学生，3.按类别统计分析人数，4.搜索并执行操作(显示信息，删除，编辑)5.按条件排序。选择之后转到下一个界面，执行结束或者在某些地方输入不合法数据后又会回到主菜单。

1. **存储数据的实现**

系统在初始化StudentList类时将自动调用读取文件的函数。在创建完毕head，end节点后，依次从二进制文件data.dat中读取Student信息至下一节点。如果文件没有信息或找不到文件会提示没有找到数据。

|  |
| --- |
| StudentList::StudentList()  {  cout<<"Welcome!"<<endl;  head = new Student;  end = head;  */\*read from data file\*/*  ifstream infile("data.dat", ios::binary);  if (infile){  while(infile.peek()!=EOF){  end->next=new Student;  end=end->next;  infile.read((char\*)&(\*end), sizeof(Student));  }  infile.close();  }  if(end==head){ *//no file or empty file*  cout<<"Data not Found..."<<endl;  }  } |

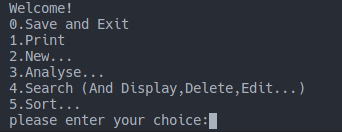
系统以选项「0」退出时会写入链表数据到文件。对应的Save函数在StudentList对象析构时被调用。

1. **实验调试、测试、运行记录及分析**

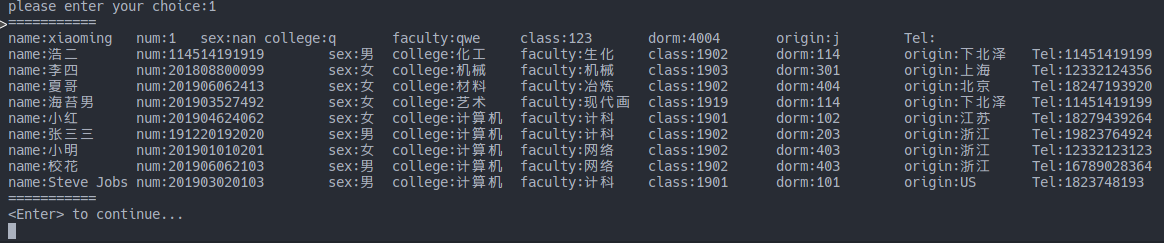
系统在调试测试过程中遇到若干问题，不过经过仔细反复的检查已经消除各种bug，基本上无论用户怎么输入都会给出友好提示，不会崩溃。

**(1)程序测试**

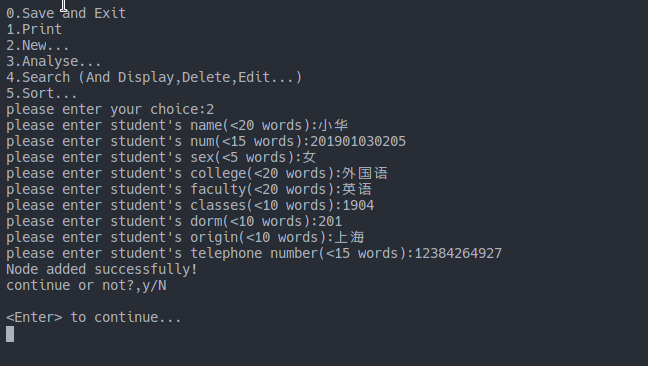
开始界面



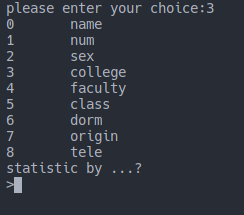
输入“1”，屏幕上打印所有学生信息。



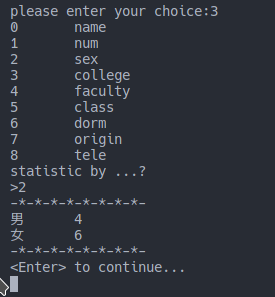
输入“2”，根据提示输入学生数据，输完后可选择按y/Y继续输入。如果直接回车或者输入其他内容则返回至主菜单。



输入“3”，列出下图所示菜单



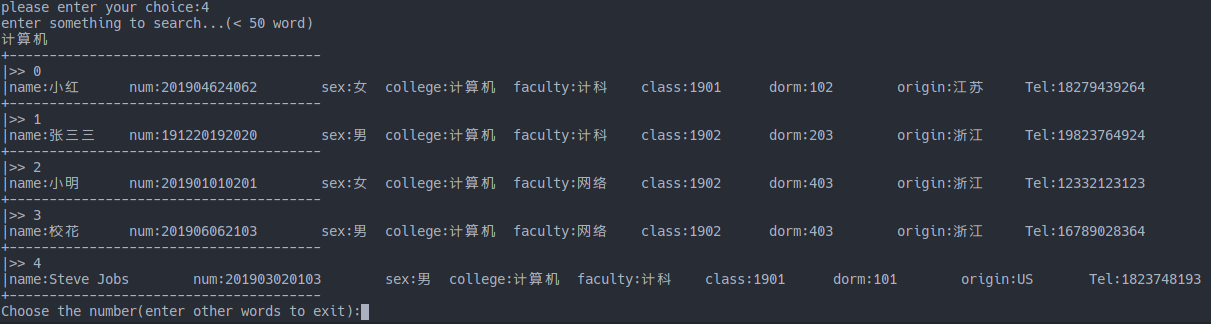
接着输入0~8中的一个



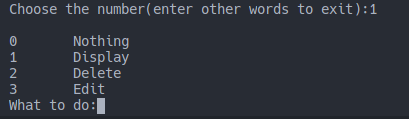
输入“4”，输入需要搜索的内容，可以是任意类别

Screenshot_20200614_220655

比如输入计算机，将匹配到计算机院系的学生



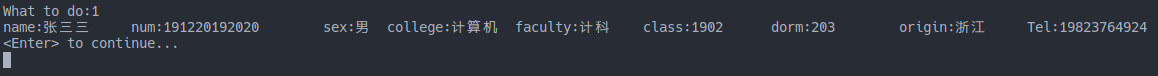
从给出的列表中选择一个学生



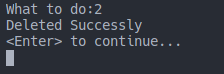
可以什么都不做

Screenshot_20200614_220934

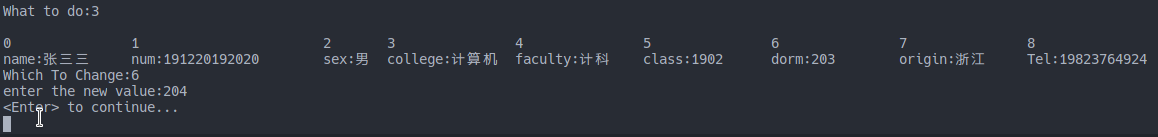
可以打印ta的信息



可以删除这个学生

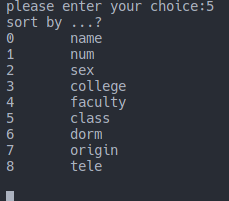


也可以编辑ta的信息

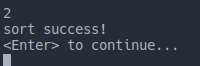


输入“5”按指定类别给学生排序

首先给出菜单，选择类别



然后会给链表排序。可以再按1查看排序结果



**(2)遇到的问题及解决方法如下：**

* + **问题1：**

**问题描述：**读取二进制文件时，如果用!infile.eof()判断结尾，会发生已经读完但仍返回0的情况。

**解决方法：**这是因为eof()函数本质上就是返回eofbit的状态，当文件read掉最后的数据时，eofbit仍然为false，直到我们已经到文件结尾却尝试读取数据时，eofbit才会被设置为true。同理，当刚创建fstream file时,就判断file.eof()的话,一定为false

可以在file.eof()前加file.peek(); 这里peek函数用于读取并返回下一个字符，但并不提取该字符到输入流中;此时file.eof()就能正确的返回了

这里我选择用while(infile.peek()!=EOF)，可以使文件为空时不出错，也能正确判断结尾，不至于多读取一次。

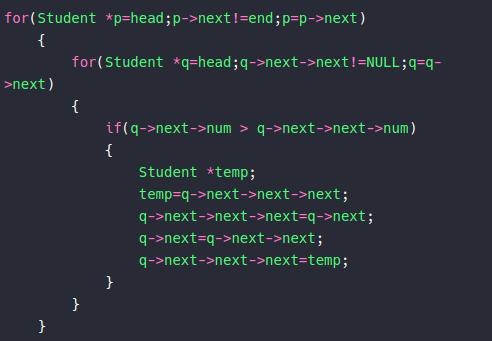
* + **问题2：**

**问题描述：**试图在student类以string定义name等变量时，会出现读取后出现段错误的问题。

**解决方法：**怀疑是2string的特性造成，改为char[]这样的数组后可以正常读取存储数据，需要用strcpy来给类初始化而参数列表给字符串赋值会各种出错。

* + **问题3：**

**问题描述：**排序链表时 试图不用头节点实现冒泡，像这样。。



发现排序不完整

**解决方法：**p会随着交换而飘到后面去。所以只能事先得到链表长度

* + **问题4：**

**问题描述：**排序出现段错误

**解决方法：**一些修改操作后length和end会变得不正确，需要刷新一下。写了Refresh()用来重置length和end

* + **问题5：**

**问题描述：**搜索结果的存储需要动态扩容，没有学过的现成的方法能在已生成的动态数组上添加元素

**解决方法：**每次重新生成一个动态数组，备份数据后删除原来的。或者另写一个链表存储。

* + **问题6：**

**问题描述：**排序等函数时因为要对多种变量可排序，而我的课设要求类别又特别多，希望可以只写一次函数来搞定，但c++不允许以值来对应变量名

**解决方法：**搜索后发现可以map解决这个问题，然而我用的是链表。只能老实switch下一项项写。

1. **其他测试：**

**处理非法输入** 对于用户输入的非法数据，比如在需要数字的地方输入字母，字符串甚至超出长度的字符串，大部分采用getline读取，对于超出长度的错误，在回到主菜单前使用cin.clear()避免之后读取不了输入。

**兼容性** 为兼容windows，Linux，mac OS系统作出了一些细节处理

由于清屏的命令在dos下是cls，在Unix下是clear，解决方法是定义一个CLEAR来替换，在不同系统下命令会不同。

|  |
| --- |
| #ifdef \_WIN32 //In Windows  #define CLEAR "cls"  #else //In any other OS  #define CLEAR "clear"  #endif |

另外由于unix不能使用system(“pause”)，改为使用while (( c = getchar()) != EOF && c != '\n');来实现。这个命令还有的好处是可以保证输入缓冲区为空。

经测试，在windows环境下也可以正确编译运行。(编译器MinGW 8.2.0。理论上至少兼容Dev-C++)