中南大学

数据结构试验报告

题 目 实验七 内排序算法比较

学生姓名 张子洋

学 号 8208221223

指导老师 郑瑾

学 院 计算机学院

专业班级 计算机与通信类2212班

完成时间 2023.6

指导老师评定 签名

1. 排序算法的实现（设计性实验）
2. 需求分析

需求：

排序是计算机领域的一项重要技术，是程序设计中的一种重要运算。它的功能是将一个数据元素的任意序列重新排列成一个按键有序的序列。学习和研究各种排序方法是计算机工作者的一项重要工作课题。

基本要求

随机产生n个整数（依次为n赋值100、200、300、1 000和2 000），将其存于数组A[1..n]中。对主要算法（顺序查找、插入排序、归并排序、堆排序和快速排序）进行实验比较，计算出平均比较次数 cn、平均移动次数mn及执行时间tn。cn和mn由程序自动计算，tn由系统时间计算。

对实验结果数据进行对比分析。

分析：

1. 输入的形式和输入值的范围：无
2. 输出的形式：输出int型数字
3. 程序所能达到的功能：排序
4. 测试数据：无
5. 概要设计

本程序包含2个函数：

* 主函数main（）；
* 排序foution（）；
* 冒泡法()；
* 插排()；
* 归并排序()；
* 归并排序辅助函数()；
* 快速排序()；
* 快速排序辅助函数()；
* 堆排序()；
* 堆排序辅助函数()；

1. 详细设计

排序算法的实现.c：

1. 调试分析
2. 采用IDE中自带的调试功能进行调试，手动添加断点和查看程序。
3. 对设计和编码的讨论和分析。该程序实现了顺序栈的操作。分析程序代码的质量，主要从以下几个方面考虑。 

* 正确性。在一定的数据范围内，该程序能实现所需功能，所以正确性是没有问题的。
* 健壮性。在一定的数据输入范围内，该程序能较好的实现链表的操作。但是如果输入数 据非法，该程序还是可能会产生一些预想不到的输出结构，或是不做任何处理。所以， 该程序的健壮性有待进一步的提高。要综合考虑一些情况，当输入有误时，应返回一个 表示错误的值，并中止程序的执行，以便在更高的抽象层次上进行处理。

1. 使用说明

无需输入，运行即可得到结果。

#include<stdlib.h>

#include<stdio.h>

#include<time.h>

#include<sys/timeb.h>

//#define MAX 25000

#define MAX 25000

/\*

n种排序方法，都是从大到小排列

没有太参考别人的程序，大部分是根据定义琢磨的

如有疏漏，恳请批评指教

\*/

//辅助函数：交换两个变量

void swap(int\*a,int\*p)

{

int temp = \*a;

\*a = \*p;

\*p = temp;

}

//冒泡排序

void bubbleSort(int\* arr,int len)

{

int i,j;

//这个是上界

for(i=0;i<len;i++)

{

//一般导数第i+1个以后都已经排好了

for(j=1;j<len-i;j++)

{

//大的一直冒泡到最后面去

if(arr[j-1]>arr[j])

{

swap(&arr[j-1],&arr[j]);

}

}

}

}

//插入排序

//从第1个数开始，往后开始遍历，第i个数一定要放到使得前i个数都变成有序的。

void insertSort(int\* arr,int len)

{

int i,j;

for(i=0;i<len;i++)

{

for(j=i;j>0;j--)

{

if(arr[j]<arr[j-1])

{

swap(&arr[j],&arr[j-1]);

}else

{

//已经不比那个元素小了就提前退出

break;

}

}

}

}

//快速排序

void quickSort(int\* arr,int start,int end)

{

//递归最重要的就是设置退出条件，如下

if(start>=end)

{

return;

}

int i = start;

int j = end;

int temp = arr[i];

//如果右指针j一直没小于左指针i，一直跑

while(i<j)

{

//先从右边找比基准小的，找到和基准交换，但要保留j值

while(i<j)

{

if(arr[j]<temp)

{

swap(&arr[j],&arr[i]);

break;

}

j--;

}

//右边找到一个比基准小的之后，轮到左边找比基准大的，然后和上面空出的j位置交换

while(i<j)

{

if(arr[i]>temp)

{

swap(&arr[j],&arr[i]);

break;

}

i++;

}

}

//排左半区

quickSort(arr,start,i-1);

//排右半区

quickSort(arr,i+1,end);

}

void mergeSort(int\* arr,int start,int end)

{

if(start>=end)

{

return;

}

int i = start;

int mid = (start+end)/2;

int j = mid + 1;

mergeSort(arr,i,mid);

mergeSort(arr,j,end);

//合并

//其实我觉得不用这个额外的空间也行，两个子序列再排一次能减少空间，不过速度肯定会有影响

int\* temp = (int\*)malloc((end-start+1)\*sizeof(int));

int index = 0;

//开始对比两个子序列，头部最小的那个数放到新空间

while(i<=mid&&j<=end)

{

if(arr[i]<=arr[j])

{

temp[index++] = arr[i++];

}else

{

temp[index++] = arr[j++];

}

}

//总有一个序列是还没有放完的，这里再遍历一下没放完的

while(i<=mid)

{

temp[index++] = arr[i++];

}

while(j<=end)

{

temp[index++] = arr[j++];

}

//排完再把新空间的元素放回旧空间

int k = start;

for(k;k<=end;k++)

{

arr[k] = temp[k-start];

}

free(temp);

}

//堆排序

void adjust(int\* arr,int len,int index)

{

//调整函数，把孩子、父亲中的最大值放到父亲节点

//index为待调整节点下标,一开始设它最大

int max = index;

int left = 2\*index+1;//左孩子

int right = 2\*index+2;//右孩子

if(left<len && arr[left] > arr[max])

{

max = left;

}

if(right<len && arr[right] > arr[max])

{

max = right;

}

//如果父亲节点不是最大

if(max!=index)

{

//一旦上层节点影响了某个孩子节点，还要观察以这个孩子节点为父节点的子树是不是也不是大顶堆了

swap(&arr[index],&arr[max]);

//因为发生了交换，还要继续调整受到影响的孩子节点

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

adjust(arr,len,max);//这句话非常非常关键

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

/\*

只有父亲和孩子节点发生了交换，才有继续调整孩子的必要，如果无脑在不是这里面递归，堆排序的效果不会比冒泡好到哪去

而如果写在了这里面，虽然还是pk不过快排，但好歹和快排的差距只缩小到个位数倍数的量级（小数据量的时候）

堆排序一个优点是空间复杂度也不高

\*/

}

}

//主要排序部分

void heapSort(int\* arr,int len)

{

int i = len/2-1;

for(i;i>=0;i--)

{

adjust(arr,len,i);

}

swap(&arr[0],&arr[len-1]);

//第二次之后，只需要从根节点从上到下调整，遇到没发生交换的直接可以退出循环了

//微调得到大顶堆（因为只有堆顶不满足而已）

int j = len -1; //去掉尾节点后的数组长度

//把最大值交换到最后

for(j;j>0;j--)

{

adjust(arr,j,0);

swap(&arr[0],&arr[j-1]);

}

}

//计算时间，精确到毫秒

long getTime()

{

struct timeb tb;

ftime(&tb);//取得当前的时间

//前面是毫秒，后面是微秒

return tb.time\*1000+tb.millitm;

}

//小型主函数

void test()

{

int\* a1 = (int\*)malloc(MAX\*sizeof(int));

int\* a2 = (int\*)malloc(MAX\*sizeof(int));

int\* a3 = (int\*)malloc(MAX\*sizeof(int));

int\* a4 = (int\*)malloc(MAX\*sizeof(int));

int\* a5 = (int\*)malloc(MAX\*sizeof(int));

int i = 0;

srand(time(NULL));

for(i;i<MAX;i++)

{

int temp = rand()%(MAX+1);

// int temp = i;//升序测试

// int temp = MAX-i-1;//降序测试

a1[i] = temp;

a2[i] = temp;

a3[i] = temp;

a4[i] = temp;

a5[i] = temp;

}

long t1 = getTime();

//冒泡排序

bubbleSort(a1,MAX);

long t2 = getTime();

printf("冒泡排序排%d个随机数据耗时%ld毫秒\n",MAX,t2-t1);

//插入排序

insertSort(a2,MAX);

long t3 = getTime();

printf("插入排序排%d个随机数据耗时%ld毫秒\n",MAX,t3-t2);

//快速排序

quickSort(a3,0,MAX-1);

long t4 = getTime();

printf("快速排序排%d个随机数据耗时%ld毫秒\n",MAX,t4-t3);

//归并排序

mergeSort(a4,0,MAX-1);

long t5 = getTime();

printf("归并排序排%d个随机数据耗时%ld毫秒\n",MAX,t5-t4);

//堆排序

heapSort(a5,MAX);

long t6 = getTime();

printf("堆排序排%d个随机数据耗时%ld毫秒\n",MAX,t6-t5);

}

int main()

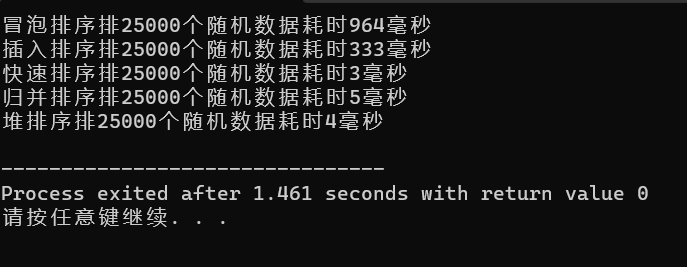
{

test();

return 0;

}

1. 测试程序的运行结果



1. 心得体会

我认为在学习排序算法的过程中，我们需要深入理解算法的原理和流程，并且进行不断的练习和实践。只有在实践中不断尝试和优化，才能够掌握排序算法的核心思想和优化技巧，从而开发出高效、稳定的程序。总的来说，排序算法是程序开发中不可或缺的一部分，需要我们在学习中注重理解和实践，并且结合实际场景进行选择和优化，从而开发出高效、稳定的程序。

附录：源程序文件清单

各程序源代码文件随本实验报告电子版一起打包，存放在文件夹二分查找算法中。

文件清单如下：

排序算法的实现.c…………………………………………排序算法的实现

1. 统计成绩（综合性实验）
2. 需求分析

需求：

给出n个学生的m门考试的成绩表，每个学生的信息由学号、姓名及各科成绩组成。对学生的考试成绩进行有关统计，并打印统计表。

基本要求

(1) 按总数高低次序，打印出名次表，分数相同的为同一名次。

(2) 按名次打印出每个学生的学号、姓名、总分以及各科成绩。

分析：

1. 输入的形式和输入值的范围：按屏幕提示将成绩数据放入文件
2. 输出的形式：输出统计结果。
3. 程序所能达到的功能：统计，排序学生成绩
4. 测试数据：
5. 概要设计
6. 本程序包含2个函数：

* 主函数main()；
* 比较函数compare();

1. 详细设计
2. #include <stdio.h>
3. #include <string.h>
4. #include <stdlib.h>
5. #include <conio.h> //conio是Console Input/Output(控制台输入输出)的简写，其中定义了通过控制台进行数据输入和数据输出的函数，
6. //主要是一些用户通过按键盘产生的对应操作，比如getch()函数等等。
7. /\*
8. 5
9. 12345 小李 91 92 93
10. 12346 小王 98 80 98
11. 12348 小张 97 99 100
12. 12349 小明 95 94 93
13. 12290 小林 89 97 96
14. \*/
15. //成绩结构体
16. struct score
17. {
18. char snum[9];
19. char sname[11];
20. int chinese;
21. int math;
22. int english;
23. int sumcj;//成绩总数
24. int rank;//排名
25. };
26. //函数声明
27. void showmenu(void);//菜单
28. int xg(int);//修改成绩
29. void lr(int);//录入成绩
30. void cx(int);//查询成绩
31. void xhcx(int);//学号查询
32. void xmcx(int n);//姓名查询
33. void allrankcx(int);//排名查询
34. void showcxmenu(void);//二级菜单，查询菜单
35. void xs(int);//成绩显示
36. void jspm(int);//计算排名
37. int sc(int);//删除成绩
38. void bccj(void);//保存成绩并退出
39. //main函数
40. int main()
41. {
42. int flag,n=0,count=0;//flag标志位,count计数,当前总人数
43. //-------------判断文件里面有多少条记录--------------------//
44. FILE \*fp;//文件指针
45. //表示fp是指向FILE结构的指针变量，通过fp即可找存放某个文件信息的结构变量，
46. //然后按结构变量提供的信息找到该文件，实施对文件的操作。习惯上也笼统地把fp称为指向一个文件的指针。
47. fp=fopen("cj.txt","a+");//打开这个文件，并将文件放入缓冲区
48. //"a+"：可读/写数据，原来的文件不被删去，位置指针移到文件末尾
49. if(fp==NULL) //文件指针指向空
50. {
51. printf("打开文件失败,请重试！\n");
52. getch();
53. fclose(fp); //fclose是一个函数名，功能是关闭一个流。
54. exit(0);
55. }
56. int c = 0;
57. // feof()是检测流上的文件结束符的函数，如果文件结束，则返回非0值，否则返回0
58. //
59. //一般在文件操作，中经常使用feof()判断文件是否结束。
60. while(!feof(fp))//当条件为不是文件尾时均执行后续代码
61. {
62. char cc = fgetc(fp);//读取下一个字符
63. if(cc==EOF)
64. {
65. break;
66. }
67. //输出字符
68. // putchar(cc);
69. c++;
70. }
71. count = c/14;//IO流的读取是按字符读的，有意义文字7+空格6+换行符1=14
72. // printf("%d", count);
73. // getch();
74. //fclose是一个函数名，功能是关闭一个流。
75. //注意:使用fclose()函数就可以把缓冲区内最后剩余的数据输出到内核缓冲区，
76. //并释放文件指针和有关的缓冲区。
77. fclose(fp);//返回一个流，并将文件储存进入磁盘
78. //---------------------------------//
79. do
80. {
81. showmenu();
82. scanf("%d",&flag);//选择操作
83. switch(flag)
84. {
85. case 1: printf("请输入您要录入的人数：");
86. scanf("%d",&n);
87. lr(n);
88. count += n;break;
89. case 2: count=sc(count);break;
90. case 3: xg(count);break;
91. case 4: cx(count);break;
92. case 5: jspm(count);break;
93. case 6: xs(count);break;
94. case 0: bccj();break;
95. default:printf("输入错误，请重试！\n");getch();//getch()用户输入一个字符函数便自动读取，不需要回车
96. }
97. }while(flag!=0);
98. getch();
99. return 0;
100. }
101. //0显示菜单
102. void showmenu()
103. {
104. system("cls");
105. printf("\*----------------------------------------------------\*\n");
106. printf("\* 欢迎使用\*\*成绩管理系统 \*\n");
107. printf("\*----------------------------------------------------\*\n");
108. printf("\* 1： 录入成绩 \*\n");
109. printf("\* 2： 删除成绩 \*\n");
110. printf("\* 3： 修改成绩 \*\n");
111. printf("\* 4： 成绩查询 \*\n");
112. printf("\* 5： 成绩计算并排名 \*\n");
113. printf("\* 6： 显示成绩信息 \*\n");
114. printf("\* 0： 保存信息并退出系统 \*\n");
115. printf("\*----------------------------------------------------\*\n");
116. printf("选择操作<0-6> \n");
117. return;
118. }//展示界面
120. //1录入成绩
121. void lr(int n)
122. {
123. struct score stu[100];
124. FILE \*fp;
125. fp=fopen("cj.txt","a+");
126. if(fp==NULL)
127. {
128. printf("打开文件失败,请重试！\n");
129. getch();
130. fclose(fp);
131. exit(0);
132. }
133. printf("请输入全部学生的学号，姓名，语文成绩，数学成绩，英语成绩；\n");
134. int i;
135. for(i=0;i<n;i++)
136. {
137. printf("----第%d个学生----\n", i+1);
138. printf("学号:");
139. scanf("%s",&stu[i].snum);
140. for(int k=0;k<i;k++)
141. {
142. if(strcmp(stu[i].snum,stu[k].snum)==0)
143. {
144. printf("学号重复，请重新输入！");
145. scanf("%s",&stu[i].snum);
146. }
147. }
149. printf("姓名：");
150. scanf("%s,",&stu[i].sname);
152. printf("语文成绩：");
153. scanf("%d",&stu[i].chinese);
155. printf("数学成绩：");
156. scanf("%d",&stu[i].math);
158. printf("英语成绩：");
159. scanf("%d",&stu[i].english);
161. stu[i].sumcj=0;
162. stu[i].rank=0; //暂时先让排行和总成绩为零
163. //写入数据
164. fprintf(fp,"%s %s %d %d %d %d %d\n",stu[i].snum,stu[i].sname,stu[i].chinese,stu[i].math,stu[i].english,stu[i].sumcj,stu[i].rank);}
165. //写入文件
166. fclose(fp);
167. printf("添加学生成功,请按任意键返回\n");
168. getch();
169. }
171. //2删除成绩
172. int sc(int n)
173. {
174. struct score stu[100];
175. struct score sub[99];//删除后的结构体数组
176. char stnum[9];//学号
177. int find=0;
178. printf("请输入你要删除的学生学号:");
179. scanf("%s", &stnum);
180. FILE \*fp;
181. fp=fopen("cj.txt","r");//"r"：只能从文件中读数据，该文件必须先存在，否则打开失败
182. if(fp==NULL)
183. {
184. printf("打开文件失败,请联系管理员\n");
185. getch();
186. return -1;
187. }
188. int c=0;
189. int index=0;//用于记录目标序号
190. while(!feof(fp))//不为空 当条件为不是文件尾时均执行后续代码
191. {
192. fscanf(fp,"%s %s %d %d %d %d %d\n", &stu[c].snum,&stu[c].sname,&stu[c].chinese,&stu[c].math,&stu[c].english,&stu[c].sumcj,&stu[c].rank);
193. if(strcmp(stnum,stu[c].snum)==0)//字符串比较（查找学号一样的）
194. {
195. find = 1;//根据学号查找，找到赋值为1
196. index = c;//记录编号
197. }
198. c++;//运用一个简单的循环，依次向下找
199. }
200. fclose(fp);
201. if(n==0)
202. {
203. printf("已经没有学生成绩信息了,请按任意键返回\n");
204. getch();
205. showmenu();
206. return n;
207. }
208. if(find==0)
209. {
210. printf("没有该学生成绩信息,请按任意键返回\n");
211. getch();
212. showmenu();
213. return n;
214. }
215. else
216. {
217. int i;
218. //删除
219. for(i=0;i<c;i++) //
220. {
221. if(i < index)
222. {
223. sub[i] = stu[i];//在需删除的学号前，一一对应
224. }
225. else
226. {
227. sub[i] = stu[i+1];//sub是删除后的
228. }//需删除的学号之后，移一个位置
229. }
231. fp=fopen("cj.txt","w");//"w"：只能向文件写数据，若指定的文件不存在则创建它，如果存在则先删除它再重建一个新文件
232. //写入数据
233. for(i=0;i<c-1;i++)
234. {
235. fprintf(fp,"%s %s %d %d %d %d %d\n",sub[i].snum,sub[i].sname,sub[i].chinese,sub[i].math,sub[i].english,sub[i].sumcj,sub[i].rank);
236. }
237. fclose(fp);//写入删除后的成绩，并储存进入磁盘
238. printf("学号为%s的学生成绩删除成功,请按任意键返回\n",stnum);
239. getch();
240. showmenu();
241. return n-1;//学生人数-1
242. }
243. }
245. ///3修改成绩
246. int xg(int n)
247. {
248. struct score stu[100];
249. FILE \*fp;
250. char stnum[9];
251. int find=0;
252. printf("请输入要修改的学生学号:");
253. scanf("%s",stnum);
254. fp=fopen("cj.txt","r");
255. if(fp==NULL)
256. {
257. printf("文件打开失败,请联系管理员\n");
258. getch();
259. return -1;
260. }
261. int c=0;
262. while(!feof(fp))
263. {
264. // fscanf 位于头文件<stdio.h>中，函数原型为 int fscanf(FILE \* stream, const char \* format, [argument...]);
265. //其功能为根据数据格式(format)从输入流(stream)中读入数据(存储到argument);与fgets的差别在于:fscanf遇到空格和换行时结束，注意空格时也结束，fgets遇到空格不结束。
266. //
267. fscanf(fp,"%s %s %d %d %d %d %d\n",&stu[c].snum,&stu[c].sname,&stu[c].chinese,&stu[c].math,&stu[c].english,&stu[c].sumcj,&stu[c].rank);
268. if(strcmp(stnum,stu[c].snum)==0)
269. {
270. find=1;
271. //修改操作
272. printf("找到你要修改成绩的同学,请输入他的成绩:\n");
273. printf("语文成绩: %d,新成绩为:",stu[c].chinese);
274. scanf("%d",&stu[c].chinese);
275. printf("数学成绩: %d,新成绩为:",stu[c].math);
276. scanf("%d",&stu[c].math);
277. printf("英语成绩: %d,新成绩为:",stu[c].english);
278. scanf("%d",&stu[c].english);
279. }
280. c++;
281. }
282. fclose(fp);
283. if(find==0)
284. {
285. printf("没有找到要修改的学生记录,请按任意键退出\n");
286. getch();
287. }
288. else
289. {
290. fp=fopen("cj.txt","w");
291. int i;
292. for(i=0;i<n;i++)
293. {
294. //更新数据
295. fprintf(fp,"%s %s %d %d %d %d %d\n",stu[i].snum,stu[i].sname,stu[i].chinese,stu[i].math,stu[i].english,stu[i].sumcj,stu[i].rank);
296. }
297. fclose(fp);
298. printf("学号为%s同学的成绩已经修改成功,请按任意键返回",stnum);
299. getch();
300. }
301. showmenu();
302. return 1;
303. }
305. //4成绩查询
306. void cx(int n) //n全班人数
307. {
308. int k;
309. do
310. {
311. showcxmenu();
312. scanf("%d",&k);
313. switch(k)
314. {
315. case 1:xhcx(n);break;//学号查询
316. case 2:xmcx(n);break;//姓名查询
317. case 3:allrankcx(n);break;//全班排名查询
318. case 0:break;
319. default:printf("输入错误，请重新选择\n");getch();
320. }
321. }while(k!=0);
322. return;
323. }
324. //4.0查询菜单
325. void showcxmenu()
326. {
327. system("cls");
328. printf("\*----------------------------------------------------\*\n");
329. printf("\* 成绩查询菜单 \*\n");
330. printf("\*----------------------------------------------------\*\n");
331. printf("\* 1： 按学号查询 \*\n");
332. printf("\* 2： 按姓名查询 \*\n");
333. printf("\* 3： 全班排名查询 \*\n");
334. printf("\* 0： 退出查询系统，返回主系统 \*\n");
335. printf("\*----------------------------------------------------\*\n");
336. printf("选择操作<0-3> \n");
337. }
338. //4.1学号查询
339. void xhcx(int n)
340. {
341. struct score stu[100];
342. char snum[9];
343. printf("请输入你要查询学生的学号：");
344. scanf("%s",&snum);
345. FILE \*fp;
346. fp=fopen("cj.txt ","r");//"r"：只能从文件中读数据，该文件必须先存在，否则打开失败
347. if(fp==NULL)
348. {
349. printf("打开文件失败！\n");
350. getch();
351. exit(0);
352. }
353. int c = 0;
354. while(!feof(fp)){
355. fscanf(fp,"%s %s %d %d %d %d %d\n",&stu[c].snum,&stu[c].sname,&stu[c].chinese,&stu[c].math,&stu[c].english,&stu[c].sumcj,&stu[c].rank);
356. c++;
357. }
358. int i;
359. for(i=0;i<n;i++)
360. {
361. if(strcmp(snum,stu[i].snum)==0)//二者相等返回0
362. {
363. printf("学号\t姓名\t语文\t数学\t英语\t总分\t名次\n");
364. printf("%s\t%s\t%d\t%d\t%d\t%d\t%d\n",stu[i].snum,stu[i].sname,stu[i].chinese,stu[i].math,stu[i].english,stu[i].sumcj,stu[i].rank);
365. printf("按任意键返回\n");
366. getch();
367. fclose(fp);
368. return;
369. }
370. }
371. printf("未找到要查询学生的成绩，请重试！\n");
372. getch();
373. fclose(fp);
374. return;
375. }
376. //4.2姓名查询
377. void xmcx(int n)
378. {
379. struct score stu[100];
380. char sname[11];
381. printf("请输入你要查询学生的姓名：");
382. scanf("%s",&sname);
383. FILE \*fp;
384. fp=fopen("cj.txt ","r");
385. if(fp==NULL)
386. {
387. printf("打开文件失败！\n");
388. getch();
389. exit(0);
390. }
391. int c = 0;
392. while(!feof(fp)){
393. fscanf(fp,"%s %s %d %d %d %d %d\n\n",&stu[c].snum,&stu[c].sname,&stu[c].chinese,&stu[c].math,&stu[c].english,&stu[c].sumcj,&stu[c].rank);
394. c++;
395. }
396. int i,p=0;
397. for(i=0;i<n;i++)
398. {
399. if(strcmp(sname,stu[i].sname)==0)
400. {
401. p++;
402. printf("学号\t姓名\t语文\t数学\t英语\t总分\t名次\n");
403. printf("%s\t%s\t%d\t%d\t%d\t%d\t%d\n",stu[i].snum,stu[i].sname,stu[i].chinese,stu[i].math,stu[i].english,stu[i].sumcj,stu[i].rank);
404. printf("按任意键返回\n");
405. getch();
406. fclose(fp);
407. }
408. }
409. if(p!=0) return;
410. printf("未找到要查询学生的成绩，请重试！\n");
411. getch();
412. fclose(fp);
413. return;
414. }
415. //4.3全班排名查询
416. void allrankcx(int n)
417. {
418. struct score stu[100];
419. struct score temp;
420. FILE \*fp;
421. fp=fopen("cj.txt ","r");
422. if(fp==NULL)
423. {
424. printf("打开文件失败！\n");
425. getch();
426. exit(0);
427. }
428. int c = 0;
429. while(!feof(fp)){
430. fscanf(fp,"%s %s %d %d %d %d %d\n",&stu[c].snum,&stu[c].sname,&stu[c].chinese,&stu[c].math,&stu[c].english,&stu[c].sumcj,&stu[c].rank);
431. c++;
432. }
433. int i,j;
434. for(i=0;i<n;i++)
435. {
436. if(stu[i].rank == 0){
437. printf("请先进行成绩计算进行排名！！\n");
438. getch();
439. fclose(fp);
440. return;
441. }
442. stu[i].sumcj = stu[i].chinese + stu[i].math + stu[i].english;
443. //冒泡排序算法：进行 n-1 轮比较
444. for(i=0; i<n-1; i++){
445. //每一轮比较前 n-1-i 个，也就是说，已经排序好的最后 i 个不用比较
446. for(j=0; j<n-1-i; j++){
447. if(stu[j].sumcj > stu[j+1].sumcj){
448. temp = stu[j];
449. stu[j] = stu[j+1];
450. stu[j+1] = temp;
451. }
452. }
453. }
454. printf("学号\t姓名\t语文\t数学\t英语\t总分\t名次\n");
455. for(i=0;i<n;i++)
456. {
457. printf("%s\t%s\t%d\t%d\t%d\t%d\t%d\n",stu[i].snum,stu[i].sname,stu[i].chinese,stu[i].math,stu[i].english,stu[i].sumcj,stu[i].rank);
458. }
459. printf("按任意键返回\n");
460. getch();
461. fclose(fp);
462. return;
463. }
464. printf("查询失败，请重试！\n");
465. getch();
466. fclose(fp);
467. return;
468. }
469. //5计算排名函数
470. void jspm(int n)
471. {
472. struct score stu[100];
473. int i,j,temp;
474. int count;
475. FILE \*fp;
476. fp=fopen("cj.txt","r");
477. if(fp==NULL)
478. {
479. printf("打开文件失败！\n");
480. getch();
481. exit(0);
482. }
483. int c = 0;
484. while(!feof(fp)){
485. fscanf(fp,"%s %s %d %d %d %d %d\n",&stu[c].snum,&stu[c].sname,&stu[c].chinese,&stu[c].math,&stu[c].english,&stu[c].sumcj,&stu[c].rank);
486. stu[c].sumcj = stu[c].chinese + stu[c].math + stu[c].english;
487. c++;
488. }
489. fclose(fp);
490. //计算排名并存储
491. for(i=0;i<n;i++)
492. {
493. temp=1;//第1名
494. for(j=0;j<n;j++){
495. if(stu[i].sumcj<stu[j].sumcj) temp++;//如果较小，排名下降1名
496. }
497. stu[i].rank=temp;
498. }
499. //更新记录
500. fp=fopen("cj.txt","w");
501. for(i=0;i<n;i++)
502. {
503. fprintf(fp,"%s %s %d %d %d %d %d\n",stu[i].snum,stu[i].sname,stu[i].chinese,stu[i].math,stu[i].english,stu[i].sumcj,stu[i].rank);
504. }
505. fclose(fp);
506. printf("数据重新计算和排名成功,请按任意键返回主菜单。\n");
507. getch();
508. return;
509. }
510. //6显示成绩信息
511. void xs(int n)
512. {
513. struct score stu[100];
514. int i;
515. printf("成绩信息如下：\n");
516. FILE \*fp;
517. fp=fopen("cj.txt","r");
518. if(fp==NULL)
519. {
520. printf("打开文件失败,请重试！\n");
521. getch();
522. fclose(fp);
523. exit(0);
524. }
525. printf("学号\t姓名\t语文\t数学\t英语\t总分\t名次\n");
526. int c = 0;
527. while(!feof(fp))
528. {
529. fscanf(fp,"%s %s %d %d %d %d %d\n",&stu[c].snum,&stu[c].sname,&stu[c].chinese,&stu[c].math,&stu[c].english,&stu[c].sumcj,&stu[c].rank);
530. c++;
531. }
532. if(n!=0){//人数不为0
533. for(i=0;i<c;i++)
534. {
535. printf("%s\t%s\t%d\t%d\t%d\t%d\t%d\n",stu[i].snum,stu[i].sname,stu[i].chinese,stu[i].math,stu[i].english,stu[i].sumcj,stu[i].rank);
536. }
537. }
538. fclose(fp);
539. printf("显示完毕 ,请按任意键返回\n");
540. getch();
541. return;
542. }
543. //0保存信息并退出系统
544. void bccj()
545. {
546. printf("感谢您的使用，欢迎下次使用!\n");
547. getch();
548. return;

}

4.调试分析

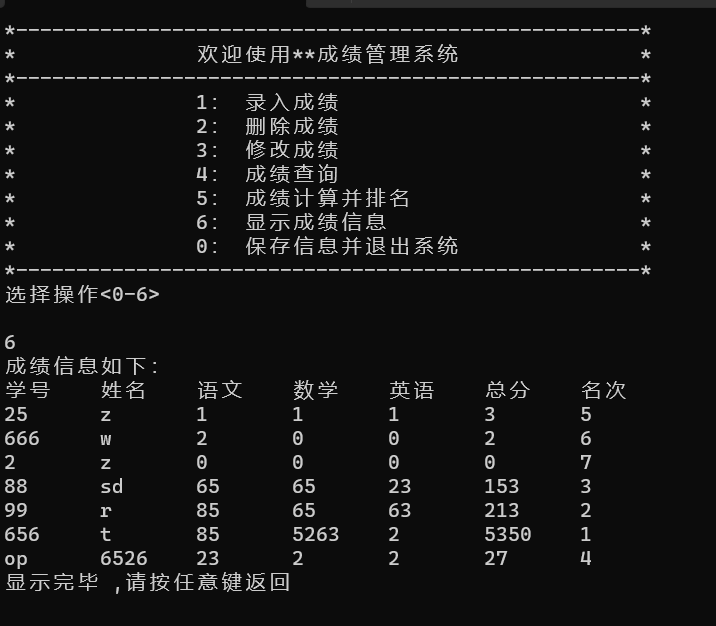
1. 采用IDE中自带的调试功能进行调试，手动添加断点和查看程序。
2. 对设计和编码的讨论和分析。该程序实现了城市链表的操作。分析程序代码的质量，主要从以下几个方面考虑。 

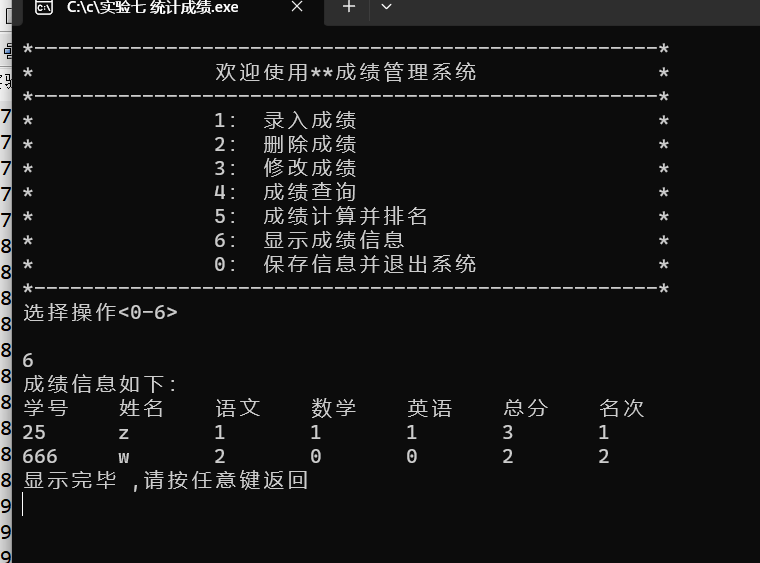
* 正确性。在一定的数据范围内，该程序能实现所需功能，所以正确性是没有问题的。
* 健壮性。在一定的数据输入范围内，该程序能较好的实现链表的操作。但是如果输入数 据非法，该程序还是可能会产生一些预想不到的输出结构，或是不做任何处理。所以， 该程序的健壮性有待进一步的提高。要综合考虑一些情况，当输入有误时，应返回一个 表示错误的值，并中止程序的执行，以便在更高的抽象层次上进行处理。

5.使用说明

按照屏幕提示，将数据放入对应文件中，按ENTER键，即可得到想要的结果。

6.测试程序的运行结果





1. 心得体会

在统计成绩中，排序是常见的一种操作。在使用C语言进行排序时，需要掌握各种排序算法，并且了解它们的时间和空间复杂度，以便选择合适的算法来满足实际需求。 在实现排序算法时，我们还可以使用一些优化技巧，如递归到循环的转化、避免重复比较、随机化快速排序等，从而提高程序的运行效率。在使用C语言进行排序时，需要深入掌握各种排序算法的特点和实现方法，并结合实际场景来选择和实现合适的算法.

附录：源程序文件清单

各程序源代码文件随本实验报告电子版一起打包，存放在文件夹统计成绩中。

文件清单如下：

统计成绩.c…………………………………………………………………………统计成绩的实现