# 1

/\*

1.使用数组保存每一天的桃子数，递推求解

2.使用链表保存每一天的桃子数，递推求解

3.天数n递减，桃子数s递增，当n<2时，输出s

4.s储存桃子数，递推增加s

\*/

#include <iostream>

#include <cstdlib>

using namespace std;

void q1(int n);

void q2(int n);

void q3(int n,int s);

void q4(int n);

int main()

{

int n,r=1;;

cin >> n;

q1(n); q2(n); q3(n,r); q4(n);

return 0;

}

void q1(int n) //数组结构

{

int s[100];

s[1]=1;

for(int i=2;i<=n;++i)

s[i]=(s[i-1]+1)\*2;

cout << s[n] <<endl;

}

typedef struct S

{

int m;

S\* next;

}S;

void q2(int n) //链表结构

{

S\* L=(S\*)malloc(sizeof(S));

S\* p=L;

p->m=1; p->next=NULL;

for(int i=1;i<n;++i)

{

p->next=(S\*)malloc(sizeof(S));

p->next->m=(p->m+1)\*2;

p=p->next;

p->next=NULL;

}

cout << p->m <<endl;

p=L;

while(p->next)

{

L=p->next;

free(p);

p=L;

}

free(p);

}

void q3(int n,int s) //递归方法

{

if(n<2) cout << s <<endl;

else q3(--n,(s+1)\*2);

}

void q4(int n) //递推

{

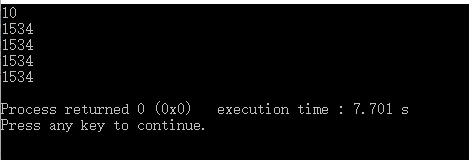
int a=1;

for(int i=1;i<10;++i)

a=(a+1)\*2;

cout << a <<endl;

}



# 3

/\*

使用双向链表保存长整数。

相加时使用尾指针向前移动相加。递归相加，b表示是否进位，边运行边输出，第一位最先输出。

最后释放链表内存。

\*/

#include <iostream>

#include <cstdio>

#include <cstdlib>

using namespace std;

typedef struct longint{ //双向链表

int n; //一个结点储存一位数

longint\* prior;

longint\* next;

}longint;

void get();

void sum(longint\* l1,longint\* l2,bool b);

void d\_free(longint\* l);

void get() //读取长整数并构建链表

{

char c;

longint\* p=NULL;

while((c=getchar())!='\n') //回车表示输入结束

{

longint\* t=(longint\*)malloc(sizeof(longint));

t->n=c-48; //字符-48为对应数字

t->prior=p; t->next=NULL;

if(p)

p->next=t;

p=t;

}

longint\* l1=p; //第一个长整数

p=NULL;

while((c=getchar())!='\n')

{

longint\* t=(longint\*)malloc(sizeof(longint));

t->n=c-48;

t->prior=p; t->next=NULL;

if(p)

p->next=t;

p=t;

}

longint\* l2=p; //第二个长整数

sum(l1,l2,false);

cout << endl;

d\_free(l1);

d\_free(l2);

}

//相加函数，递归方式，b表示是否进位，边加边输出

void sum(longint\* l1,longint\* l2,bool b)

{

int s=b?1:0; //s为当前位相加的结果

b=false;

if(!l1&&!l2) //第一位

{

if(s) cout << s;

return;

}

else if(l1&&!l2) //第二个长整数相加完毕

{

s+=l1->n;

if(s>=10)

{

s-=10; b=true;

}

sum(l1->prior,NULL,b);

}

else if(!l1&&l2) //第一个长整数相加完毕

{

s+=l2->n;

if(s>=10)

{

s-=10; b=true;

}

sum(NULL,l2->prior,b);

}

else

{

s+= l1->n + l2->n;

if(s>=10)

{

s-=10; b=true;

}

sum(l1->prior,l2->prior,b);

}

cout << s; //输出当前位结果

}

void d\_free(longint\* l) //释放内存

{

longint\* p=l;

while(p->prior)

{

l=p->prior;

free(p);

p=l;

}

free(p); p=NULL;

l=NULL;

}

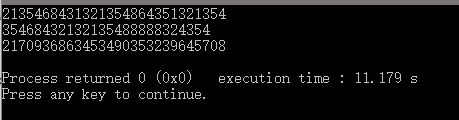
int main()

{

get();

return 0;

}



# 4

/\*

1.先读入文件1的表头并写入文件3，再用getline函数逐行读取数据并写入文件3。读完文件1后读文件2，读取文件2表头但不写入文件3，之后操作同前面。

2.读取文件3，如果出现分数低于60的学生，就写入文件4。

3.先将数据读入程序并储存在stu数组中，对stu进行排序（选择排序，冒泡排序），排序完后写入文件。

4.从头遍历stu数组进行顺序查找。另一种方法先对stu进行姓名排序，再进行折半查找。

\*/

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

typedef struct student{

string\* sname;

string\* sno;

int grade[3];

}student;

student stu[100];

int length=0;

void display() //显示函数

{

for(int i=0;i<length;++i)

{

cout << \*(stu[i].sname) << "\t" << \*(stu[i].sno);

for(int j=0;j<3;++j)

cout << "\t" << stu[i].grade[j];

cout << endl;

}

}

string read() //读入文件

{

ifstream f1("3.txt");

if(!f1)

{

cout <<"文件不存在！" << endl;

return "";

}

string title;

getline(f1,title);

int num=0;

string name; //储存姓名

string no; //储存学号

while(f1 >> name)

{

stu[num].sname=new string(name);

f1 >> no;

stu[num].sno=new string(no);

for(int i=0;i<3;++i)

f1 >> stu[num].grade[i];

++num;

}

length=num;

return title;

}

void sort1() //选择排序法

{

int sum[length]={0}; //存储总分

for(int i=0;i<length;++i)

sum[i]+=stu[i].grade[0]+stu[i].grade[1]+stu[i].grade[2];

for(int i=0;i<length-1;++i)

{

int maxindex=i; //j记录最大值下标

for(int j=maxindex;j<length;++j)

if(sum[maxindex]<sum[j])

maxindex=j;

if(i!=maxindex) //进行交换

{

int temp=sum[i];

sum[i]=sum[maxindex];

sum[maxindex]=temp;

student st=stu[i];

stu[i]=stu[maxindex];

stu[maxindex]=st;

}

}

}

void sort2() //冒泡排序法

{

int sum[length]={0}; //存储总分

for(int i=0;i<length;++i)

sum[i]+=stu[i].grade[0]+stu[i].grade[1]+stu[i].grade[2];

int i=length-1;

int index; //每次循环最后一次交换的下标

while(i>0)

{

index=0;

for(int j=0;j<i;++j)

if(sum[j]<sum[j+1])

{

index=j;

int temp=sum[j];

sum[j]=sum[j+1];

sum[j+1]=temp;

student st=stu[j];

stu[j]=stu[j+1];

stu[j+1]=st;

}

i=index;

}

}

void search1(string name) //顺序查找

{

bool b=false; //判断是否查找成功

for(int i=0;i<length;++i)

if(name==\*(stu[i].sname))

{

cout << \*(stu[i].sname) << "\t" << \*(stu[i].sno);

for(int j=0;j<3;++j)

cout << "\t" << stu[i].grade[j];

cout << endl;

b=true;

break;

}

if(!b)

cout << "查找失败！" << endl;

}

void search2(string name) //折半查找法

{

//先对姓名进行排序

int i=length-1;

int index; //每次循环最后一次交换的下标

while(i>0)

{

index=0;

for(int j=0;j<i;++j)

if(\*(stu[j].sname)>\*(stu[j+1].sname))

{

index=j;

student st=stu[j];

stu[j]=stu[j+1];

stu[j+1]=st;

}

i=index;

}

//查找

int low=0,high=length;

bool b=false;

while(true)

{

int t=(low+high)/2;

if(name==\*(stu[t].sname))

{

cout << \*(stu[t].sname) << "\t" << \*(stu[t].sno);

for(int j=0;j<3;++j)

cout << "\t" << stu[t].grade[j];

cout << endl;

b=true;

break;

}

else if(low==high)

break;

else if(name<\*(stu[t].sname)) //在左边

high=t-1;

else //在右边

low=t+1;

}

if(!b)

cout << "查找失败！" << endl;

}

void q1()

{

ifstream f1("1.txt");

ifstream f2("2.txt");

ofstream f3("3.txt");

if(!f1||!f2)

{

cout <<"文件不存在！" << endl;

return;

}

if(!f3)

{

cout<<"创建文件失败！"<<endl;

return;

}

string title;

getline(f1,title);

getline(f2,title);

f3 << title << endl; //写入表头

string line;

while (getline (f1, line)) //写入文件1

f3 << line << endl;

while (getline(f2,line)) //写入文件2

f3 << line << endl;

cout << "文件合并成功！" << endl;

f1.close();

f2.close();

f3.close();

}

void q2()

{

ifstream f1("3.txt");

ofstream f2("4.txt");

if(!f1)

{

cout <<"文件读取错误！" << endl;

return;

}

if(!f2)

{

cout<<"创建文件失败！"<<endl;

return;

}

string title;

getline(f1,title);

f2 << title << endl;

string name; //储存姓名

while(f1 >> name)

{

string sno; //储存学号

int grade[3]; //储存成绩

bool b=false;

f1 >> sno;

for(int i=0;i<3;++i)

{

f1 >> grade[i];

if(grade[i]<60) b=true; //判断是否需要补考

}

if(b) //如果有补考，则写入文件

{

f2 << name << "\t" << sno;

for(int i=0;i<3;++i)

f2 << "\t" << grade[i];

f2 << endl;

}

}

f1.close();

f2.close();

cout << "生成文件“4.txt”成功！" << endl;

}

void q3()

{

string title=read(); //读入数据，返回表头

cout << "排序前：" << endl << title << endl;

display();

sort1(); //选择排序

//sort2(); //冒泡排序

cout << "排序后：" << endl << title << endl;

display();

ofstream f("3.txt");

if(!f)

{

cout << "写入失败！" << endl;

return;

}

//写入文件

f << title << endl;

for(int i=0;i<length;++i)

{

f << \*(stu[i].sname) << "\t" << \*(stu[i].sno);

for(int j=0;j<3;++j)

f << "\t" << stu[i].grade[j];

f << endl;

}

f.close();

cout << "排序成功！" << endl;

}

void q4()

{

string title=read(); //读入数据

string name;

cin >> name;

//search1(name); //顺序查找

//search2(name); //折半查找

}

int main()

{

//q1();

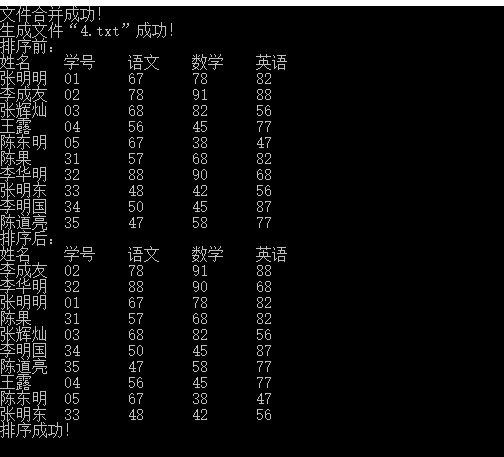
//q2();

//q3();

//q4();

return 0;

}



# 5

/\*

获取要读取的文件名称，然后读入内容存在data中。

统计每个字符出现的次数储存在c\_count中，c\_count定义为256个字符的数组，字符对应ASCII值作为下标，出现就++数组对应下标。

对出现的字符构建结点，出现次数作为权值，构建哈夫曼树。

遍历哈夫曼树，将每个字符对应的哈夫曼编码存入pcode结构体中。

对data进行遍历，用哈夫曼编码代替字符。

编译后的数据存在t\_code中，最后用二进制方式写入dat文件中。

\*/

#include <fstream>

#include <string>

#include <iostream>

#include <cstdlib>

#include <iomanip>

#include <cstring>

using namespace std;

//结点

typedef struct point{

char c; //数据

int weight; //权

int parent,lc,rc; //左右孩子

}point;

//哈夫曼编码

typedef struct pcode{

char c;

string\* code; //数据对应的哈夫曼编码

}pcode;

//读入英文文章

bool fread(string & data,char\* filename)

{

char s[20];

//获取文件名

strcpy(s,filename);

strcat(s,".txt");

ifstream in(s);

string line;

if(!in)

{

cout <<"文件不存在！" << endl;

return false;

}

data=""; //清空data

while (getline (in, line)) // line中不包括每行的换行符

data+=line+'\n';

return true; //表示读入成功

}

//统计字符出现个数

int\* c\_count(const string & data)

{

int\* c\_n=(int\*)malloc(256\*sizeof(int)); //c\_n用于储存每个字符出现的次数

for(int i=0;i<256;++i) //初始化c\_n

c\_n[i]=0;

for(int i=0;i<data.length();++i)

++c\_n[(int)data.at(i)];

return c\_n;

}

//构建哈夫曼树

bool select(point\* & root,int n,int & s1,int & s2)

{

bool first=true; //判断是否为第一次达到条件

for(int i=1;i<n;++i)

if(root[i].parent==0) //判断是否已加入哈夫曼树

{

if(!first)

{

if(s1==s2) //第二次达到条件

{

if(root[i].weight<root[s1].weight)

{

s2=s1; s1=i;

}

else

s2=i;

continue;

}

else if(root[i].weight<=root[s1].weight)

{

s2=s1; s1=i;

}

else if(root[i].weight<=root[s2].weight)

s2=i;

}

else

{

s1=s2=i; first=false;

}

}

if(s1==s2) return false;

if(root[s1].c==0 && root[s2].c!=0) // 新建结点在右边

{

int t=s1; s1=s2; s2=t;

}

return true;

}

point\* setTree(int\* c\_n,int & len)

{

len=0;

point\* root=(point\*)malloc(256\*sizeof(point));

for(int i=0;i<256;++i) //创建结点

if(c\_n[i]) //统计字符的种类并初始化结点

{

++len;

root[len].c=(char)i;

root[len].lc=0;

root[len].rc=0;

root[len].parent=0;

root[len].weight=c\_n[i];

}

root=(point\*)realloc(root,2\*len\*sizeof(point)); //重新分配内存

for(int i=len+1;i<2\*len;++i)

{

root[i].c=0;

root[i].lc=0;

root[i].rc=0;

root[i].parent=0;

root[i].weight=0;

}

int s1=1,s2=1;

for(int i=len+1;i<2\*len;++i)

{

if(select(root,i,s1,s2))

{

root[s1].parent=i;

root[s2].parent=i;

root[i].lc=s1;

root[i].rc=s2;

root[i].weight=root[s1].weight+root[s2].weight;

}

else break;

}

return root;

}

//生成哈夫曼编码

pcode\* Huffmancode(point\* root,int len)

{

pcode\* HC=(pcode\*)malloc((len+1)\*sizeof(pcode));

string stemp="";

int p=2\*len-1;

int l=0;

for(int i=0;i<=p;++i) root[i].weight=0;

while(p)

{

if(root[p].weight==0) //第一次访问结点

{

root[p].weight=1;

if(root[p].lc!=0)

{

p=root[p].lc; stemp+='0';

}

else if(root[p].rc==0)

{

++l;

HC[l].c=root[p].c;

HC[l].code=new string(stemp);

}

}

else if(root[p].weight==1) //第二次访问结点

{

root[p].weight=2;

if(root[p].rc!=0)

{

p=root[p].rc;

stemp+='1';

}

}

else{ //第三次访问结点

root[p].weight=0;

p=root[p].parent;

stemp = stemp.substr(0, stemp.length()-1);

}

}

return HC;

}

//编码

string coding(pcode\* HC,const string & data,int len)

{

string s=""; //储存文章的哈夫曼编码结果

for(int i=0;i<data.length();++i)

for(int j=1;j<=len;++j)

if(HC[j].c==data.at(i))

{

s+=\*HC[j].code;

break;

}

return s;

}

//译码

bool encoding(string & code,string & t\_code,pcode\* HC,int len) //code编码后的文章，t\_code储存译码后的文章

{

t\_code="";

int index=0;

while(index<code.length())

{

int clength=0; //记录匹配的字符的哈夫曼编码长度（优先与最长的哈夫曼编码匹配）

int falg=0; //记录最终匹配的字符下标

for(int i=1;i<=len;++i)

{

string hc=\*HC[i].code;

if(hc.length()<=clength || hc.at(0)!=code.at(index))

continue;

string temp=code.substr(index,hc.length());

if(temp==hc)

{

clength=hc.length();

falg=i;

}

}

if(falg) //匹配成功

{

t\_code+=HC[falg].c;

index+=(\*HC[falg].code).length();

}

else

return false;

}

return true;

}

//压缩储存文件

bool write(string & code,const char\* filename)

{

char s[20];

strcpy(s,filename);

strcat(s,"\_code.dat");

ofstream out(s,ios::binary);

if(!out)

return false;

out.write((char\*)&code,sizeof(code));

out.close();

return true;

}

//输出哈夫曼编码

void display(pcode\* HC,int len)

{

cout << "各字符的哈夫曼编码：" << endl;

for(int i=1;i<=len;++i)

{

char c=HC[i].c;

if(c==' ')

cout << "space";

else if(c=='\n')

cout << "\\n";

else

cout << HC[i].c;

cout << ":" << \*HC[i].code << endl;

}

cout << endl;

}

//压缩率

void compressibility(const char\* filename)

{

char s1[20];

char s2[20];

strcpy(s1,filename);

strcat(s1,".txt");

strcpy(s2,filename);

strcat(s2,"\_code.dat");

ifstream f1(s1);

ifstream f2(s2,ios::binary);

f1.seekg(0,ios::end);

long l1=f1.tellg();

f1.close();

f2.seekg(0,ios::end);

long l2=f2.tellg();

f2.close();

cout << "压缩前：" << l1 << "字节" << endl;

cout << "压缩后：" << l2 << "字节" << endl;

cout << "压缩率:" << setprecision(2) << (double)l2/l1\*100 << "%" << endl;

}

//清除HC

void delHC(pcode\* HC,int len)

{

for(int i=1;i<=len;++i)

delete HC[i].code;

free(HC);

HC=NULL;

}

int main()

{

string data;

char filename[20];

cout << "输入要读取的文件名（不含后缀名）:";

gets(filename);

if(fread(data,filename)) //读取文件

cout << "原文：" << endl << data << endl;

else

exit(1);

int len=0;

int\* c\_n=c\_count(data); //字符个数

point\* root=setTree(c\_n,len); //哈夫曼树

free(c\_n);

pcode\* HC=Huffmancode(root,len); //哈夫曼编码

display(HC,len);

free(root);

string code=coding(HC,data,len); //文章编码

cout << "编码后：" << endl << code << endl << endl;

string t\_code;

if(encoding(code,t\_code,HC,len)) //译码

cout << "译码后：" << endl << t\_code << endl;

if(write(code,filename)) //写入文件

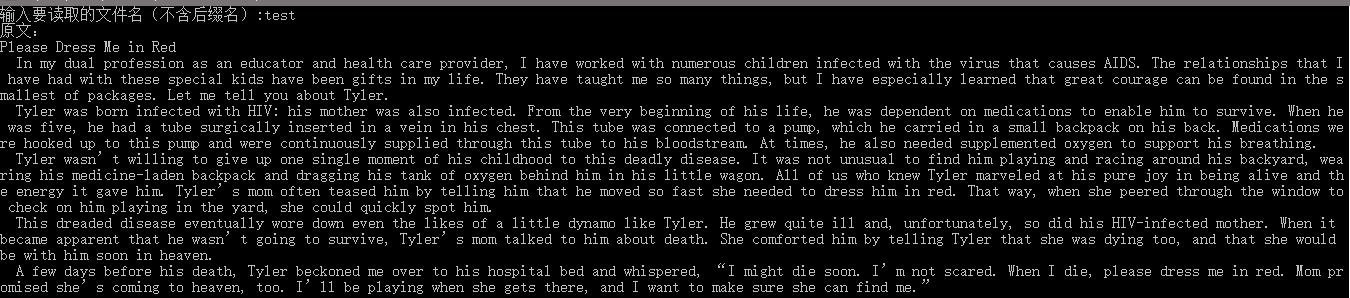
cout << "成功储存！" << endl << endl;

compressibility(filename); //压缩率

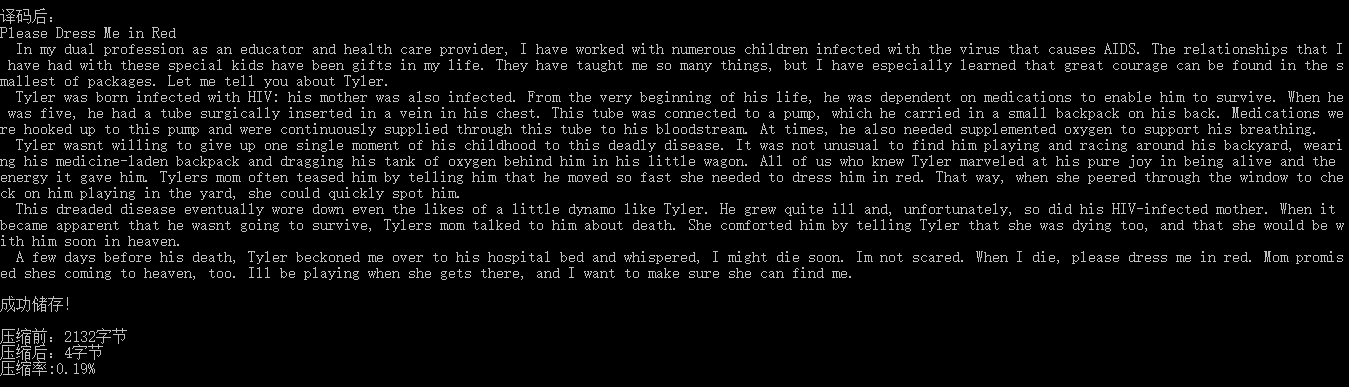
delHC(HC,len); //清除HC

return 0;

}

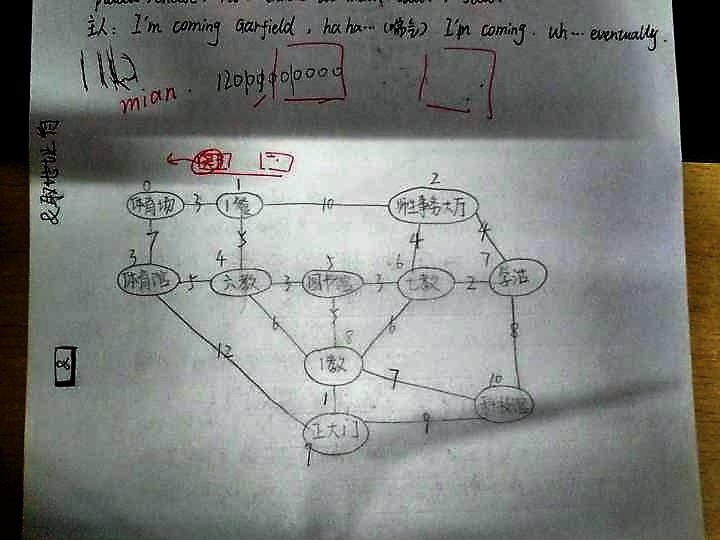






# 6

选用如下图：



/\*

根据图构建相应的结构体数组Map和其邻接矩阵Link。

使用顺序查找查找并输出景点简介。

最短路径采用弗洛伊德算法。

输出路径时从起点出发，寻找与起点直接相连且处于最短路径的顶点i，移动至顶点i并输出相关信息，重复上述操作，直到i移动到终点结束循环。

\*/

#include <stdio.h>

#include <string.h>

#define COUNT 11 //景点个数

#define MAX\_LENGTH 100 //两点不直接连通时的路径长度

typedef struct Unit{

char name[20];

char profile[100]; //景点简介

}Unit;

Unit Map[COUNT]; //存放景点

int Link[COUNT][COUNT]; //邻接矩阵

void display()

{

printf("100表示无穷大\n\n");

for(int i=0;i<COUNT;++i)

{

printf("%-14s:",Map[i].name);

for(int j=0;j<COUNT;++j)

printf("%d\t",Link[i][j]);

printf("\n");

}

}

void linkmap(int i,int j,int weight) //建立两点的联系

{

Link[i][j]=weight;

Link[j][i]=weight;

}

bool setMap()

{

//导入景点数据

strcpy(Map[0].name,"体育场");

strcpy(Map[0].profile,"西北田径场，校运会举办场所。");

strcpy(Map[1].name,"一餐");

strcpy(Map[1].profile,"第一餐厅，饭菜价格实惠。");

strcpy(Map[2].name,"师生事务大厅");

strcpy(Map[2].profile,"师生事务处理场所。");

strcpy(Map[3].name,"体育馆");

strcpy(Map[3].profile,"室内运动场所。");

strcpy(Map[4].name,"六教");

strcpy(Map[4].profile,"第六教研楼。");

strcpy(Map[5].name,"图书馆");

strcpy(Map[5].profile,"自习首选场所。");

strcpy(Map[6].name,"七教");

strcpy(Map[6].profile,"第七教研楼。");

strcpy(Map[7].name,"学生活动中心");

strcpy(Map[7].profile,"学生活动都在这里举办。");

strcpy(Map[8].name,"一教");

strcpy(Map[8].profile,"第一教研楼。");

strcpy(Map[9].name,"正大门");

strcpy(Map[9].profile,"500W的大门。");

strcpy(Map[10].name,"科技馆");

strcpy(Map[10].profile,"科技展示场所。");

//建立连接关系

for(int i=0;i<COUNT;++i) //初始化

for(int j=0;j<COUNT;++j)

Link[i][j]=MAX\_LENGTH;

linkmap(0,1,3);

linkmap(0,3,7);

linkmap(1,2,10);

linkmap(1,4,5);

linkmap(2,6,4);

linkmap(2,7,4);

linkmap(3,4,5);

linkmap(3,9,12);

linkmap(4,5,3);

linkmap(4,8,6);

linkmap(5,8,5);

linkmap(5,6,3);

linkmap(6,7,2);

linkmap(6,8,6);

linkmap(7,10,8);

linkmap(8,9,1);

linkmap(8,10,7);

linkmap(9,10,9);

return true;

}

void searchUnit() //查询景点信息

{

printf("查询景点的名称：");

char sname[20];

gets(sname);

for(int i=0;i<COUNT;++i)

if(strcmp(sname,Map[i].name)==0)

{

puts(Map[i].profile); //输出景点简介

break;

}

}

int getindex(char\* s) //获取景点所在下标

{

for(int i=0;i<COUNT;++i)

if(strcmp(s,Map[i].name)==0)

return i;

return -1;

}

void minlength() //最短路径，弗洛伊德算法

{

char start[20];

char end[20];

printf("起点：");

gets(start);

printf("终点：");

gets(end);

if(strcmp(start,end)==0)

{

printf("输入无效！\n");

return;

}

int vs=getindex(start);

int ve=getindex(end);

int D[COUNT][COUNT]; //两点间的长度

bool P[COUNT][COUNT][COUNT]={false}; //最短路径中包含的顶点

int v,w,u;

for(v=0;v<COUNT;++v) //初始化

for(w=0;w<COUNT;++w)

{

D[v][w]=Link[v][w];

if(D[v][w]<MAX\_LENGTH)

{

P[v][w][v]=true;

P[v][w][w]=true;

}

}

for(u=0;u<COUNT;++u) //弗洛伊德算法

for(v=0;v<COUNT;++v)

for(w=0;w<COUNT;++w)

if(D[v][u]+D[u][w]<D[v][w])

{

D[v][w]=D[v][u]+D[u][w];

for(int i=0;i<COUNT;++i)

P[v][w][i]=P[v][u][i] || P[u][w][i];

}

//输出路径

bool visited[COUNT]={false};

int t=vs; //记录当前走到的位置

printf("%s",Map[vs].name);

visited[t]=true;

while(t!=ve)

{

for(int i=0;i<COUNT;++i)

{

if(!P[vs][ve][i] || visited[i]) //判断i是否在最短路径上

continue;

int b=false; //记录结点i与结点j的路径是否还存在其他结点

for(int j=0;j<COUNT;++j)

if(j!=t && j!=i && P[t][i][j])

{

b=true;

break;

}

if(!b) //t->i处于最短路径

{

t=i;

printf(" -> %s",Map[t].name); //t移动到i

visited[t]=true; //将i置为已访问

break;

}

}

}

printf("\n");

}

int main()

{

if(setMap())

{

display();

printf("\n");

searchUnit();

printf("\n");

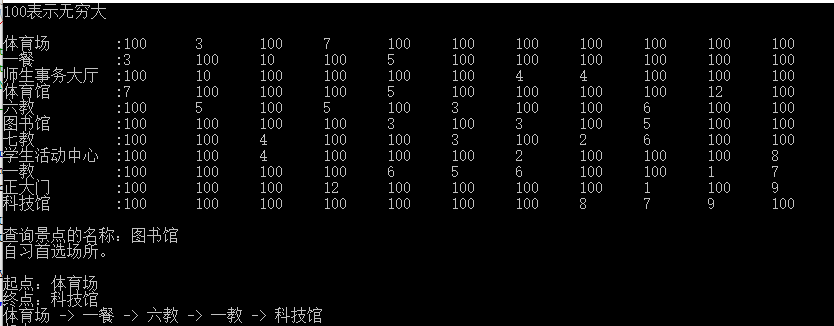
while(1)

minlength();

}

return 0;

}



# 8

/\*

使用结构体储存学生数据，sub储存表头数据。

因为子函数数量众多，所以通过采用结构体指针数组pstu作为参数进行传递来提高效率。

排序采用冒泡排序，查找采用顺序查找。

插入时找到插入位置后，原数据后移一位。

拷贝边读取边写入新文件。

\*/

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

#include<ctype.h>

#include<conio.h>

#define MAX\_STUDENT 100

struct student{

long number;

char name[20];

int sex;

double c1;

double c2;

double c3;

double c4;

double sum;

double average;

};

typedef struct student student;

void main\_menu(); //主菜单

void search\_menu(); //数据查询菜单

void sat\_menu(); //数据统计菜单

int Key(); //密码函数

void xian(); //输出分割线

void sort(int b,int m); //排序函数，b：0为总分排序，1为学号排序，2为姓名排序 。m：0不输出排序 ，1输出排序

void sub\_class(int s); //课程等级百分比函数，s表示输出第s+1门课程的情况

void sub\_sat(); //课程平均分

void save(); //数据储存

void load(); //数据导入

void change(int i); //数据修改函数 ,i为修改数据的数组下标

void print(); //控制台输出

void add\_menu(); //添加数据菜单

void insert(); //插入

void add(); //追加

void fcopy(); //拷贝

char sub[4][10]={'\0'}; //储存课程名称

char\* psub=sub[0];

student\* pstu[MAX\_STUDENT]; //结构体指针数组，用于对结构体进行操作

int n=0,f=0; //n为学生人数，f判断是否存在数据

void load()

{

FILE\* fp=NULL;

int count=0,i;

if((fp=fopen("student.dat","rb"))==NULL) return; //如果没有数据文件，就返回

//读取课程名

for(i=0;i<4;++i)

if(fread(psub+10\*i,10\*sizeof(char),1,fp)!=1)

{

printf("读入文件错误\n"); xian(); return;

}

//读取学生数据

while(1)

{

pstu[count]=(student\*)malloc(sizeof(student));

if(fread(pstu[count],sizeof(student),1,fp)!=1)

{

if(feof(fp)) {fclose(fp); break;}

printf("读入文件错误\n"); xian(); return;

}

++count; //计算读入数据组数 ，即学生个数

}

n=count;

if(n) f=1;

}

int Key()

{

char M[20]="student";

char mima[20];

int t=0;

while(t<3)

{

int count=0;

printf("密码：");

while(mima[count]=getch())

{

if(mima[count]=='\r') break;

if(mima[count]=='\b')

{

if(count>0)

{printf("\b \b"); count--;}

continue;

}

printf("\*"); count++;

}

mima[count]='\0';

printf("\n");

if(strcmp(mima,M)==0)

{printf("密码正确！\n"); xian(); return 1;}

else {printf("密码错误！\n"); ++t;}

}

printf("\n由于你多次输入密码错误，程序自动退出。\n"); getchar();

return 0;

}

void xian()

{

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\n\n");

}

void print()

{

if(f==0)

{

printf("不存在数据！\n"); return;

}

printf("name\tnum\tsex\t%s\t%s\t%s\t%s\tsum\taverage\n",sub[0],sub[1],sub[2],sub[3]);

for(int i=0;i<n;++i)

printf("%-s\t%-ld\t%-d\t%-.1lf\t%-.1lf\t%-.1lf\t%-.1lf\t%-.1lf\t%-.1lf\n"

,pstu[i]->name,pstu[i]->number,pstu[i]->sex,pstu[i]->c1,pstu[i]->c2,pstu[i]->c3,pstu[i]->c4,pstu[i]->sum,pstu[i]->average);

}

void input()

{

if(f) //判断是否存在数据

{

printf("已拥有数据，是否重新输入？（Y为是，其他键为否）\n");

char t=getchar(); getchar();

if(toupper(t)!='Y') {xian(); return;}

}

r\_input:

int i;

printf("班级人数：");

scanf("%d",&n);

getchar();

if(n<1||n>MAX\_STUDENT) {printf("输入人数异常,请重新输入!\n"); goto r\_input;}

//输入课程名称

for(i=0;i<4;++i)

{

printf("课程%d名称：",i+1);

gets(psub+10\*i);

}

//输入学生数据

printf("name\tnum\tsex\t%s\t%s\t%s\t%s\n",sub[0],sub[1],sub[2],sub[3]);

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

{

pstu[i]=(student\*)malloc(sizeof(student));

scanf("%s %ld %d %lf %lf %lf %lf",pstu[i]->name,&pstu[i]->number,&pstu[i]->sex,&pstu[i]->c1,&pstu[i]->c2,&pstu[i]->c3,&pstu[i]->c4);

pstu[i]->sum=pstu[i]->c1+pstu[i]->c2+pstu[i]->c3+pstu[i]->c4;

pstu[i]->average=pstu[i]->sum/4.0;

}

getchar();

printf("你已成功输入成绩！\n"); xian();

save();

f=1;

}

void sort(int t,int m)

{

int i,j;

student\* pt=NULL;

for(i=0;i<n-1;++i)

for(j=0;j<n-i-1;++j)

{

//总分排序

if((pstu[j]->sum<pstu[j+1]->sum)&&(t==0))

{

pt=pstu[j];

pstu[j]=pstu[j+1];

pstu[j+1]=pt;

}

//学号排序

if((pstu[j]->number<pstu[j+1]->number)&&(t==1))

{

pt=pstu[j];

pstu[j]=pstu[j+1];

pstu[j+1]=pt;

}

//姓名排序

if((strcmp(pstu[j+1]->name,pstu[j]->name)>0)&&(t==2))

{

pt=pstu[j];

pstu[j]=pstu[j+1];

pstu[j+1]=pt;

}

}

//排序输出部分

if(m!=0)

{

char ch;

sort:

printf("1.从小到大\t2.从大到小\t3.返回上一菜单\n");

scanf("%c",&ch); getchar();

if(ch=='1')

for(i=0;i<n/2;++i)

{pt=pstu[i]; pstu[i]=pstu[n-1-i]; pstu[n-1-i]=pt;}

if(ch=='3') return;

if((ch<'1')||(ch>'3')) {printf("请输入正确的选项！\n\n"); goto sort;}

print(); //排序输出

xian();

}

}

void change(int i)

{

r\_change:

char ch;

student\* pt=NULL;

printf("是否对此人的数据进行操作？\n1.修改数据\t2.删除数据\t3.返回上一菜单\n");

ch=getchar(); getchar();

if(ch=='1')

{

char key1;

printf("请重新输入此人数据\n");

printf("name\tnum\tsex\t%s\t%s\t%s\t%s\n",sub[0],sub[1],sub[2],sub[3]);

printf("%-s\t%-ld\t%-d\t%-.1lf\t%-.1lf\t%-.1lf\t%-.1lf\n",pstu[i]->name,pstu[i]->number,pstu[i]->sex,pstu[i]->c1,pstu[i]->c2,pstu[i]->c3,pstu[i]->c4);

pt=(student\*)malloc(sizeof(student));

scanf("%s %ld %d %lf %lf %lf %lf",pt->name,&pt->number,&pt->sex,&pt->c1,&pt->c2,&pt->c3,&pt->c4);

getchar();

printf("是否确认修改？(Y为是，其他按键为否)\n");

key1=getchar(); getchar();

if(toupper(key1)=='Y')

{

pt->sum=pt->c1+pt->c2+pt->c3+pt->c4;

pt->average=pt->sum/4.0;

free(pstu[i]);

pstu[i]=pt; pt=NULL;

printf("数据修改成功！\n");

xian();

}

else {xian(); return;}

}

else if(ch=='2')

{

char key2;

printf("是否确认删除？(Y为是，其他按键为否)\n");

key2=getchar(); getchar();

if(toupper(key2)=='Y')

{

free(pstu[i]);

printf("数据删除成功！\n"); xian();

--n; //减少学生个数

if(n<1) {f=0; return;} //保证存在学生数据

for(int j=i;j<n;++j)

pstu[j]=pstu[j+1];

}

else {xian(); return;}

}

else if(ch=='3') {xian(); return;}

else {printf("请输入正确的选项！\n\n"); goto r\_change;}

}

void search1() //按姓名查询

{

int i,f\_n=-1; //f\_n判断是否有此人

char s[20];

printf("姓名：");

gets(s);

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

if(strcmp(s,pstu[i]->name)==0)

{

printf("%-s\t%-ld\t%-d\t%-.1lf\t%-.1lf\t%-.1lf\t%-.1lf\t%-.1lf\t%-.1lf\n\n"

,pstu[i]->name,pstu[i]->number,pstu[i]->sex,pstu[i]->c1,pstu[i]->c2,pstu[i]->c3,pstu[i]->c4,pstu[i]->sum,pstu[i]->average);

f\_n=i;

}

if(f\_n==-1) {printf("查找不到此人，自动为你返回上一菜单。\n"); xian(); return;}

else change(f\_n);

}

void search2() //按学号查询

{

int i,d,f\_n=-1;

printf("学号：");

scanf("%d",&d); getchar();

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

if(d==pstu[i]->number)

{

printf("%-s\t%-ld\t%-d\t%-.1lf\t%-.1lf\t%-.1lf\t%-.1lf\t%-.1lf\t%-.1lf\n\n"

,pstu[i]->name,pstu[i]->number,pstu[i]->sex,pstu[i]->c1,pstu[i]->c2,pstu[i]->c3,pstu[i]->c4,pstu[i]->sum,pstu[i]->average);

f\_n=i;

}

if(f\_n==-1) {printf("查找不到此人，自动为你返回上一菜单。\n"); xian(); return;}

else change(f\_n);

}

void search\_menu()

{

if(!f) {printf("不存在成绩数据，请先进行成绩输入！\n"); xian(); input();}

r\_search:

printf("1.按姓名查询\t2.按学号查询\t3.返回主菜单\n");

char ch = getchar(); getchar();

switch (ch)

{

case '1':search1(); break;

case '2':search2(); break;

case '3':return;

default:printf("请输入正确的选项！\n\n"); goto r\_search;

}

}

void sat\_menu()

{

if(!f) {printf("不存在成绩数据，请先进行成绩输入！\n"); xian(); input();}

r\_sat:

printf("1.排序\t2.课程等级数据\t3.课程平均分\t4.返回主菜单\n");

char ch = getchar();

getchar();

if (ch == '1')

{

r\_sat1:

printf("1.按总分排序\t2.按学号排序\t3.按姓名排序\t4.返回上一菜单\n");

char c = getchar(); getchar();

switch (c)

{

case '1':sort(0,1); break;

case '2':sort(1,1); break;

case '3':sort(2,1); break;

case '4':goto r\_sat; break;

default:printf("请输入正确的选项！\n\n"); goto r\_sat1;

}

}

else if(ch=='2')

{

r\_sat2:

printf("请从以下课程中选择：\n");

printf("1.%s\t2.%s\t3.%s\t4.%s\t5.返回上一菜单\n",sub[0],sub[1],sub[2],sub[3]);

char c=getchar(); getchar();

switch(c)

{

case '1': sub\_class(1); break;

case '2': sub\_class(2); break;

case '3': sub\_class(3); break;

case '4': sub\_class(4); break;

case '5': goto r\_sat; break;

default: {printf("请输入正确的选项！\n"); goto r\_sat2;}

}

}

else if(ch=='3') sub\_sat();

else if(ch=='4') {xian(); return;}

else {printf("请输入正确的选项！\n\n"); goto r\_sat;}

}

void sub\_class(int s)

{

int A=0,B=0,C=0,D=0,E=0,i;

double grade[n]; //建立数组储存成绩

if(s==1) for(i=0;i<n;++i) grade[i]=pstu[i]->c1;

else if(s==2) for(i=0;i<n;++i) grade[i]=pstu[i]->c2;

else if(s==3) for(i=0;i<n;++i) grade[i]=pstu[i]->c3;

else if(s==4) for(i=0;i<n;++i) grade[i]=pstu[i]->c4;

//等级统计

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

{

if(grade[i]>=90.0) ++A;

else if(grade[i]>=80.0) ++B;

else if(grade[i]>=70.0) ++C;

else if(grade[i]>=60.0) ++D;

else ++E;

}

printf("等级\t人数\t百分比\n");

printf("A\t%d\t%.2lf%%\n",A,A\*100.0/n);

printf("B\t%d\t%.2lf%%\n",B,B\*100.0/n);

printf("C\t%d\t%.2lf%%\n",C,C\*100.0/n);

printf("D\t%d\t%.2lf%%\n",D,D\*100.0/n);

printf("E\t%d\t%.2lf%%\n",E,E\*100.0/n);

xian();

}

void sub\_sat()

{

double sub\_sum[4]={0.0}; //建立数组储存课程成绩总和

int i;

for(i=0;i<n;++i) sub\_sum[0]+=pstu[i]->c1;

for(i=0;i<n;++i) sub\_sum[1]+=pstu[i]->c2;

for(i=0;i<n;++i) sub\_sum[2]+=pstu[i]->c3;

for(i=0;i<n;++i) sub\_sum[3]+=pstu[i]->c4;

for(i=0;i<4;++i)

{

if(i) printf("\n");

printf("%s:%.2lf",sub[i],sub\_sum[i]/n);

}

printf("\n"); xian();

}

void save()

{

int i;

FILE\* fp=NULL;

if((fp=fopen("student.dat","wb"))==NULL)

{printf("数据导出错误\n"); return;}

//储存课程名称

for(i=0;i<4;++i)

if(fwrite(psub+10\*i,10\*sizeof(char),1,fp)!=1)

{printf("数据导出错误\n"); return;}

//储存学生数据

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

if(fwrite(pstu[i],sizeof(student),1,fp)!=1)

{printf("数据导出错误\n"); return;}

fclose(fp);

}

void add\_menu()

{

if(!f) {printf("不存在成绩数据，请先进行成绩输入！\n"); xian(); input();}

r\_add:

printf("1.插入成绩\t2.追加成绩\t3.返回主菜单\n");

char ch = getchar(); getchar();

switch (ch)

{

case '1':insert(); break;

case '2':add(); break;

case '3':return;

default:printf("请输入正确的选项！\n\n"); goto r\_add;

}

}

void insert()

{

print();

printf("输入插入位置:");

int index;

scanf("%d",&index);

char key;

student\* pt=(student\*)malloc(sizeof(student));

printf("输入要插入的数据\n");

printf("name\tnum\tsex\t%s\t%s\t%s\t%s\n",sub[0],sub[1],sub[2],sub[3]);

scanf("%s %ld %d %lf %lf %lf %lf",pt->name,&pt->number,&pt->sex,&pt->c1,&pt->c2,&pt->c3,&pt->c4);

getchar();

printf("是否确认插入？(Y为是，其他按键为否)\n");

key=getchar(); getchar();

if(toupper(key)=='Y')

{

pt->sum=pt->c1+pt->c2+pt->c3+pt->c4;

pt->average=pt->sum/4.0;

for(int i=n;i>index-1;--i)

pstu[i]=pstu[i-1];

pstu[index-1]=pt;

++n;

printf("数据插入成功！\n");

print();

xian();

}

else {xian(); return;}

}

void add()

{

char key;

student\* pt=(student\*)malloc(sizeof(student));

printf("请输入要追加的数据\n");

printf("name\tnum\tsex\t%s\t%s\t%s\t%s\n",sub[0],sub[1],sub[2],sub[3]);

scanf("%s %ld %d %lf %lf %lf %lf",pt->name,&pt->number,&pt->sex,&pt->c1,&pt->c2,&pt->c3,&pt->c4);

getchar();

printf("是否确认追加？(Y为是，其他按键为否)\n");

key=getchar(); getchar();

if(toupper(key)=='Y')

{

pt->sum=pt->c1+pt->c2+pt->c3+pt->c4;

pt->average=pt->sum/4.0;

pstu[n]=pt;

++n;

printf("数据追加成功！\n");

print();

xian();

}

else {xian(); return;}

}

void fcopy()

{

FILE\* fp1=NULL;

if((fp1=fopen("student.dat","rb"))==NULL)

{

printf("文件不存在！\n"); return; //如果没有数据文件，就返回

}

FILE\* fp2=NULL;

if((fp2=fopen("student\_copy.dat","wb"))==NULL)

{printf("数据导出错误\n"); return;}

//储存课程名称

for(int i=0;i<4;++i)

if(fwrite(psub+10\*i,10\*sizeof(char),1,fp2)!=1)

{printf("数据导出错误\n"); return;}

//读取学生数据

while(1)

{

student\* pt=(student\*)malloc(sizeof(student));

if(fread(pt,sizeof(student),1,fp1)!=1)

{

if(feof(fp1)) {fclose(fp1); break;}

printf("读入文件错误\n"); xian(); return;

}

//储存学生数据

if(fwrite(pt,sizeof(student),1,fp2)!=1)

{printf("数据导出错误\n"); return;}

free(pt);

}

printf("拷贝成功！\n");

fclose(fp2);

}

void main\_menu()

{

while(1)

{

printf("1.成绩输入\t2.成绩输出\t3.成绩查询\t4.成绩统计\t5.添加成绩\t6.文件拷贝\t7.退出系统\n");

char ch=getchar();

getchar();

if(ch=='7') break;

switch(ch)

{

case '1':input(); break;

case '2':print(); xian(); break;

case '3':search\_menu(); break;

case '4':sat\_menu(); break;

case '5':add\_menu(); break;

case '6':fcopy(); break;

default:printf("请输入正确的选项！\n"); xian(); break;

}

}

}

int main()

{

printf("成绩管理系统\n");

if(Key())

{

load();

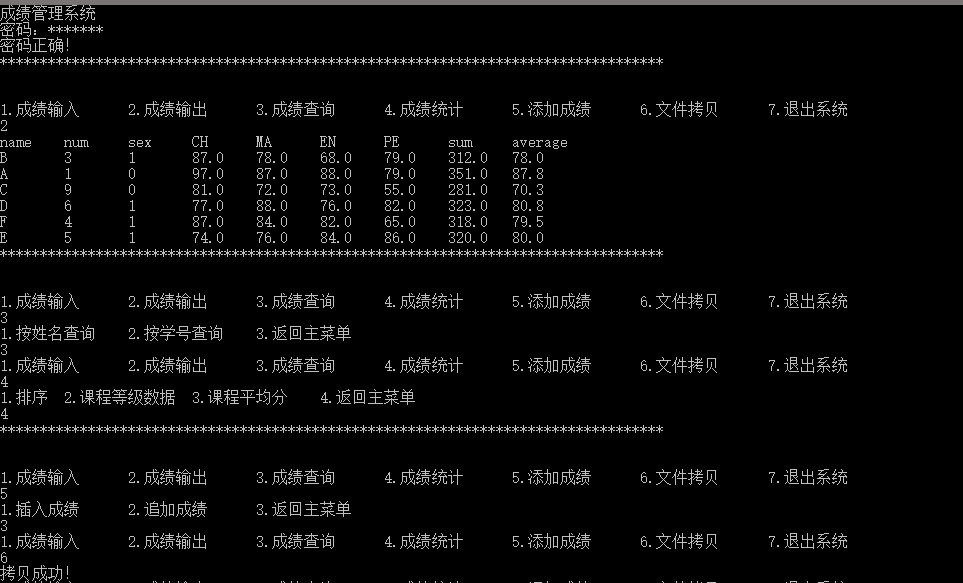
main\_menu();

save();

}

return 0;

}



# 9

/\*

用rand函数生成随机数，存储在3个长度相同的数组中，方便进行3次排序。

选择排序每次记录最小值的下标，与无序堆的第一个位置交换。

冒泡排序每次记录最后一次交换的下标，提高效率。

插入排序将数一个个加入temp数组，插入时进行比较使temp数组保持有序。

\*/

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <conio.h>

#define MAX\_COUNT 21000

int number1[MAX\_COUNT];

int number2[MAX\_COUNT];

int number3[MAX\_COUNT];

bool fwrite(int number[],char\* name) //写入文件

{

FILE \*fp;

if((fp=fopen(name,"w"))==NULL)

return false;

for(int i=0;i<MAX\_COUNT;++i)

fprintf(fp,"%d\n",number[i]);

fclose(fp);

return true;

}

void sort1() //选择排序法

{

for(int i=0;i<MAX\_COUNT-1;i++)

{

int minindex=i; //最小值下标

for(int j=i;j<MAX\_COUNT;++j)

if(number1[minindex]>number1[j])

minindex=j;

int temp=number1[i];

number1[i]=number1[minindex];

number1[minindex]=temp;

}

if(fwrite(number1,"sort\_1.txt"))

printf("选择排序：写入成功！\n");

else

printf("选择排序：写入失败！\n");

}

void sort2() //冒泡排序

{

int i=MAX\_COUNT-1;

while(i>0)

{

int index=0; //记录最后一次交换的下标

for(int j=0;j<i;++j)

if(number2[j]>number2[j+1])

{

int temp=number2[j];

number2[j]=number2[j+1];

number2[j+1]=temp;

index=j;

}

i=index;

}

if(fwrite(number2,"sort\_2.txt"))

printf("冒泡排序：写入成功！\n");

else

printf("冒泡排序：写入失败！\n");

}

void sort3() //插入排序

{

int temp[MAX\_COUNT];

for(int i=0;i<MAX\_COUNT;++i)

{

bool last=false; //是否放在末尾

for(int j=0;j<i;++j)

if(number3[i]<temp[j])

{

for(int k=i;k>j;--k)

temp[k]=temp[k-1];

temp[j]=number3[i];

last=true;

break;

}

if(!last)

temp[i]=number3[i];

}

if(fwrite(temp,"sort\_3.txt"))

printf("插入排序：写入成功！\n");

else

printf("插入排序：写入失败！\n");

}

int main()

{

srand((unsigned) time(NULL)); //用时间做种，每次产生随机数不一样

for (int i=0; i<MAX\_COUNT; i++) //生成21000个数

{

number1[i] = number2[i] = number3[i] = rand(); //产生随机数

printf("%d %d\n",i,number1[i]);

}

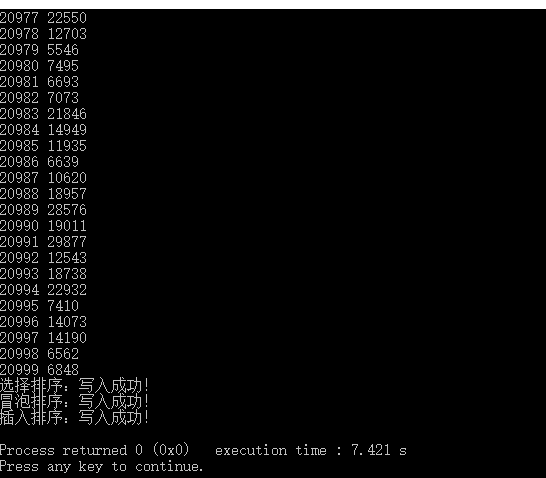
sort1(); //选择排序

sort2(); //冒泡排序

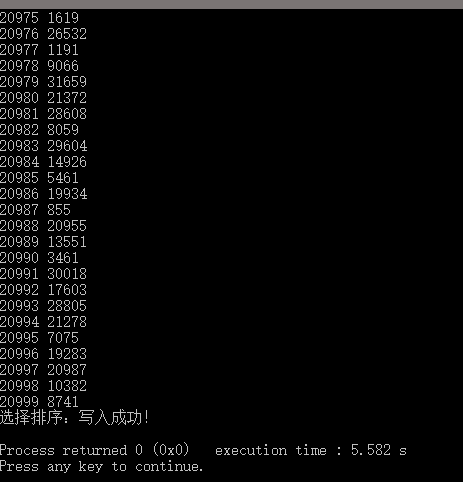
sort3(); //插入排序

return 0;

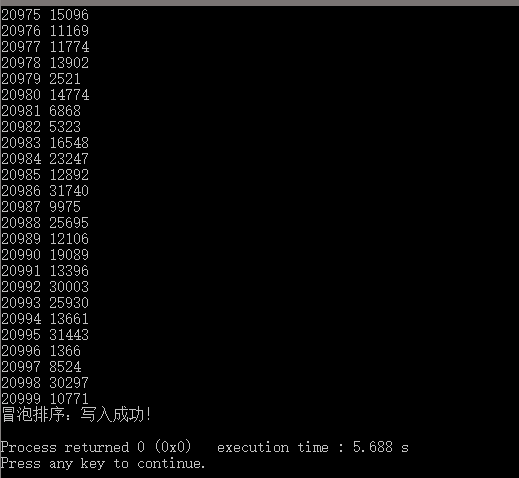
}



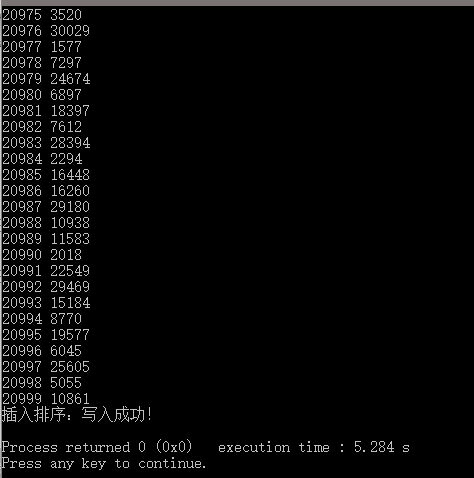
选择排序：



冒泡排序：



插入排序：



较快的排序方法：插入排序、选择排序