**《数据结构》实验报告**

**姓名： 杨孟衡 学号： 8002118240**

**班级： 软件工程1809班 专业： 软件工程**

**报告日期： 2019 年 9 月 13 日**

**实验一 线性表的就地逆置**

**一、问题描述与分析**

1. 设某线性数据元素类型为浮点型，以线性顺序表为存储结构。试编程实现：

1. 编写程序，对由键盘输入的有n个元素的线性表，输出其逆置前和逆置后的所有元素
2. 无论输入还是输出，要给出适当的提示信息
3. 用静态顺序结构实现

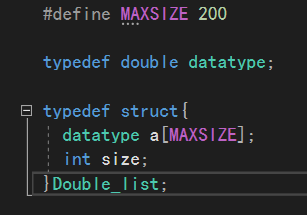
2.分析

先创建一个线性空表，提示输入数据元素，然后对实验数据进行逆置处理，最后提示输出数据元素。

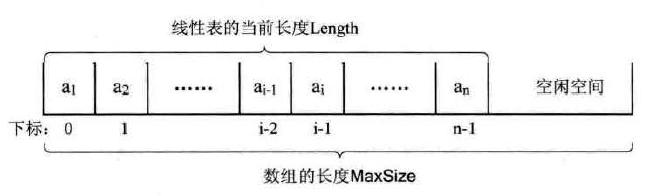
**二、数据结构与算法设计**

1.数据结构：

静态线性表采用基本数据类型数组作为基本元素构成，搭配一个整型变量int作为数据长度（size）。代码如下：



线性表图示如下：



2.算法

⑴ 生成一个线性空表

初始化线性表，初始化size为0

⑵ 提示输入数据信息，以0号结束。

设置一个循环，直到用户输入0结束输入操作。

⑶ 数据逆置交换

声明一个中间变量特temp，用于存放临时变量数值。

⑸ 提示输出信息，输出逆置后的线性表

调用Display函数输出整个线性表，Display函数采用for循坏的方式遍历整个线性表。

(6)测试插入，删除函数。

**三、算法复杂度分析**

⑴ 生成一个线性表

生成一个含200浮点型海事局的线性表配存储空间，算法复杂度为O(1)。

⑵ 循环实现逆转置

采用一个过渡变量+for循环的方式实现逆转置，算法复杂度为O(n)。

⑶ 循环输出线性表

此函数要遍历所有线性表元素，采用for循环遍历，算法复杂度为O(n)。

(4)在random.cpp测试一组数据用于测试插入删除函数是否实现。

**四、测试计划**

1.编写目的

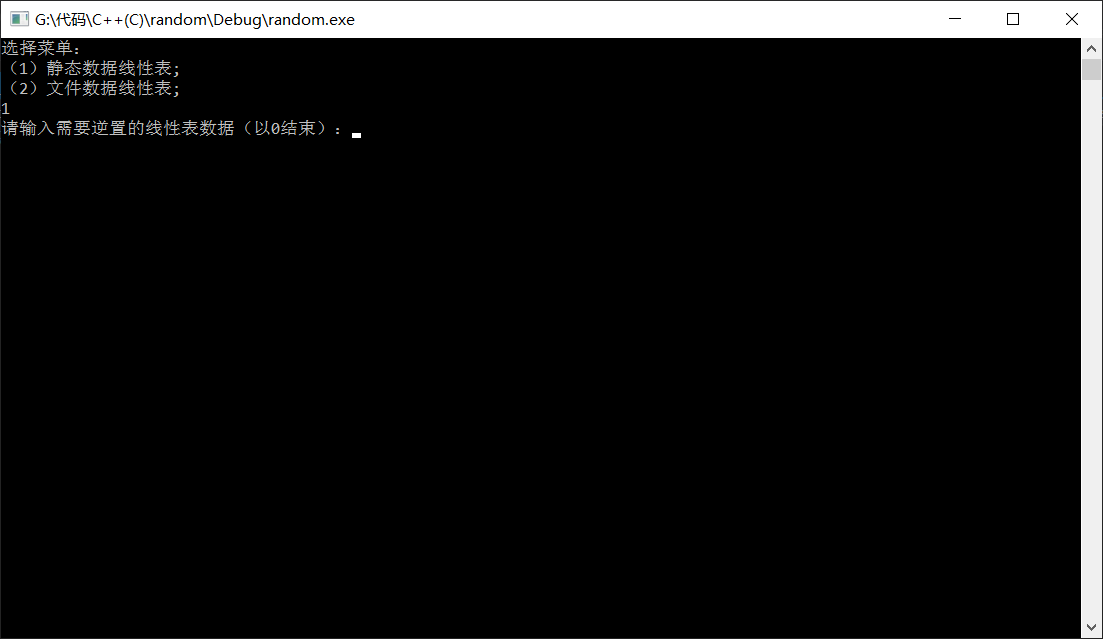
加深对数据结构线性表的顺序表存储结构和其算法思想的理解。

2.开发及运行环境

Visual Studio 2017

3.测试截图

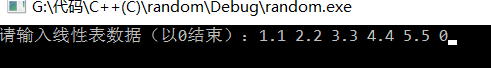
（1）提示菜单，在这里演示第一种静态顺序表，第二种采用文件输入数据的方式创建线性表，经过测试无任何问题，不再演示。



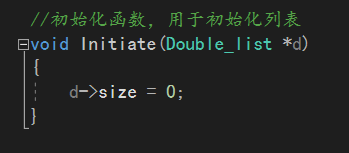
（2）提示输入数据



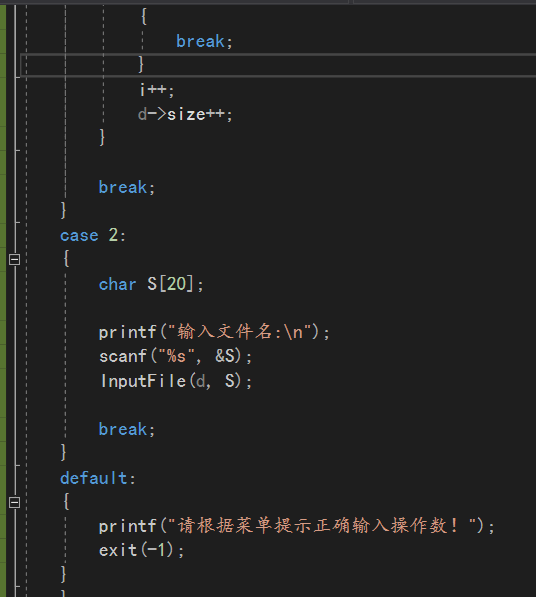
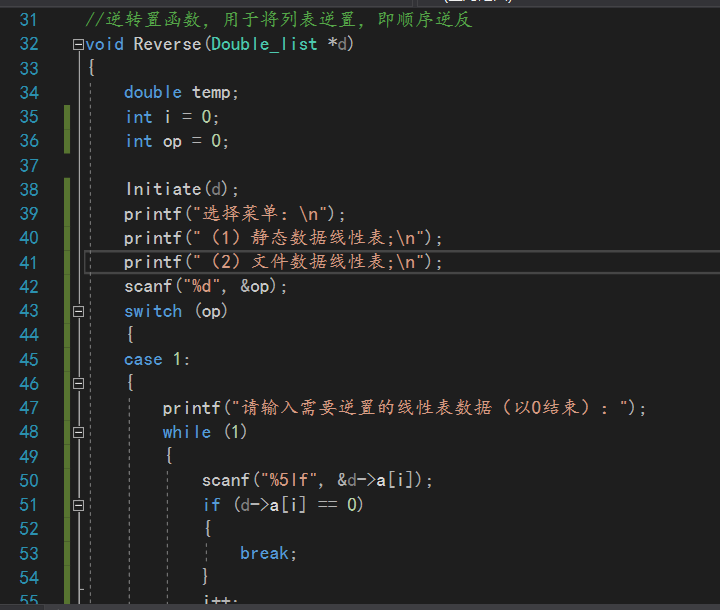
（3）输入数据，创建线性表



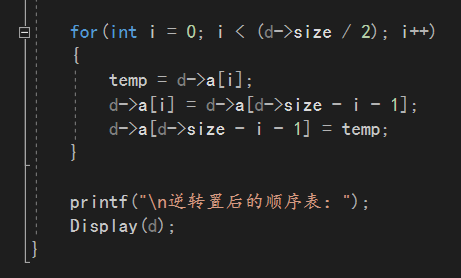
（4）线性表初始化



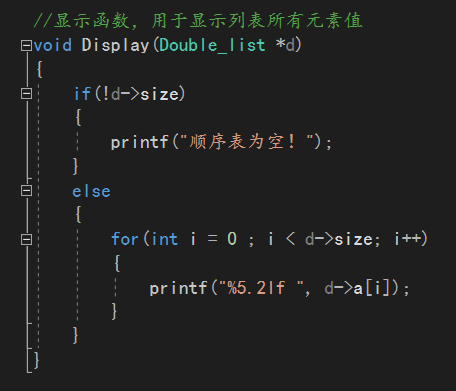
（5）读入数据



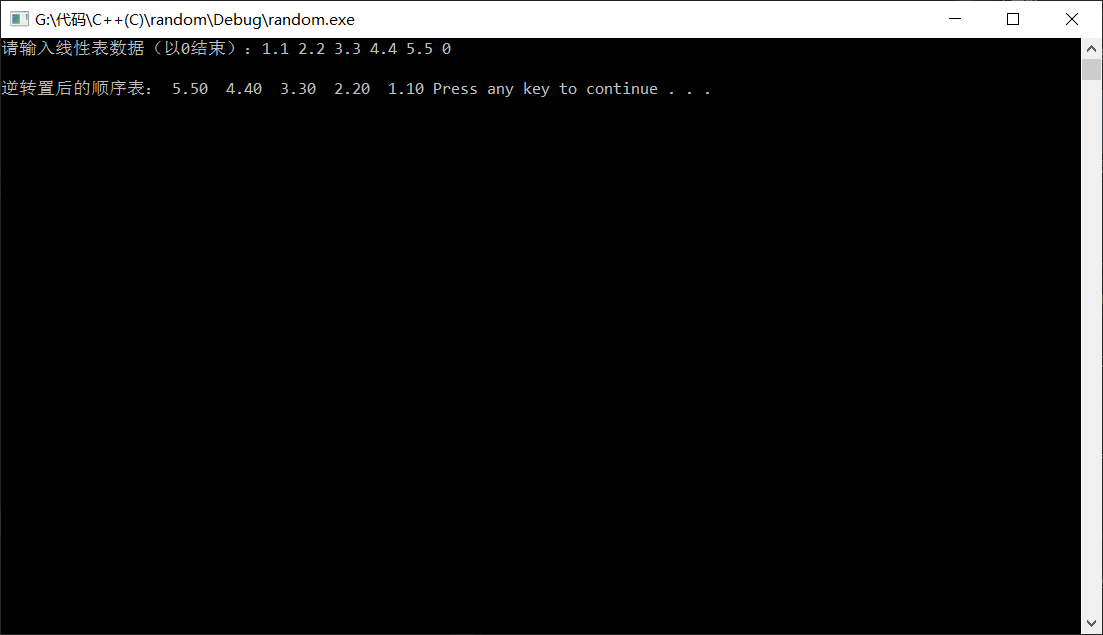
（6）逆转置数据



（6）输出数据

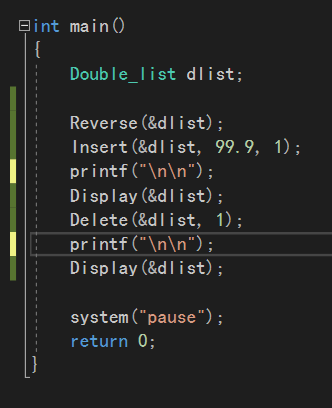


（7）最终程序输出

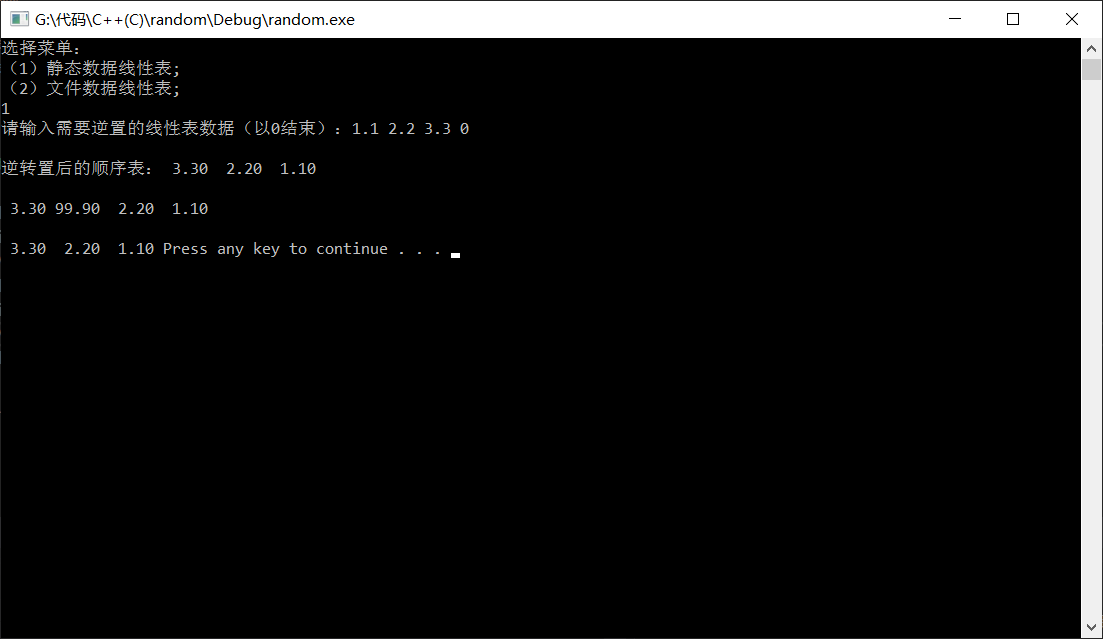


（8）测试插入，删除函数

（8）1. 修改random.cpp，准备测试数据



（8）2.数据说明：在顺序表第二的位置插入一个元素99.9，然后打印顺序表，急着删除顺序表第二的位置的元素，然后再打印一次顺序表，观察结果。



**五、源程序**

**<Double\_list.h>:**

#include <stdio.h>

#include <stdlib.h>

#define MAXSIZE 200

typedef double datatype;

typedef struct{

datatype a[MAXSIZE];

int size;

}Double\_list;

void Initiate(Double\_list \*d);

void Reverse(Double\_list \*d);

void Input(Double\_list \*d, double D);

int Empty(Double\_list \*d);

void Display(Double\_list \*d);

int Find(Double\_list d, datatype D);

void InputFile(Double\_list \*d, char \* f);

void Delete(Double\_list \*d, int position);

void Insert(Double\_list \*d, datatype x, int position);

void OutputFile(Double\_list \*d, char \* f);

//初始化函数，用于初始化列表

void Initiate(Double\_list \*d)

{

d->size = 0;

}

//逆转置函数，用于将列表逆置，即顺序逆反

void Reverse(Double\_list \*d)

{

double temp;

int i = 0;

int op = 0;

Initiate(d);

printf("选择菜单：\n");

printf("（1）静态数据线性表;\n");

printf("（2）文件数据线性表;\n");

scanf("%d", &op);

switch (op)

{

case 1:

{

printf("请输入需要逆置的线性表数据（以0结束）：");

while (1)

{

scanf("%5lf", &d->a[i]);

if (d->a[i] == 0)

{

break;

}

i++;

d->size++;

}

break;

}

case 2:

{

char S[20];

printf("输入文件名:\n");

scanf("%s", &S);

InputFile(d, S);

break;

}

default:

{

printf("请根据菜单提示正确输入操作数！");

exit(-1);

}

}

for (int i = 0; i < (d->size / 2); i++)

{

temp = d->a[i];

d->a[i] = d->a[d->size - i - 1];

d->a[d->size - i - 1] = temp;

}

printf("\n逆转置后的顺序表：");

Display(d);

}

//顺序表尾部插入元素，比如数组增加size，尾部插入新的元素

void Input(Double\_list \*d, double D)

{

if(d->size == MAXSIZE)

{

printf("顺序表已满！");

exit(-2);

}

d->size++;

d->a[d->size - 1] = D;

}

//判断顺序表是否为空，若为空返回-1，否则返回当前顺序表的size值

int Empty(Double\_list \*d)

{

return (d->size == 0?-1:d->size);

}

//显示函数，用于显示列表所有元素值

void Display(Double\_list \*d)

{

if(!d->size)

{

printf("顺序表为空！");

}

else

{

for(int i = 0 ; i < d->size; i++)

{

printf("%5.2lf ", d->a[i]);

}

}

}

//查找函数，查找顺序表中的某个元素,若没有找到返回-3，若找到返回下标值

int Find(Double\_list d, datatype D)

{

int i = 0;

while(i < d.size && d.a[i] != D)i++;

return (i < d.size?i:-1);

}

//随机访问顺序表中的某个元素，如数组可以随机访问任意一个元素，返回-1表示没找到，程序异常

datatype Get(Double\_list d, int i)

{

if(i < 0 || i >= d.size)

{

printf("该位置不存在元素！非法访问其他内存！");

}

else

{

return d.a[i];

}

return -1;

}

//从文件读取浮点型数据

void InputFile(Double\_list \*d, char \* f)

{

FILE \* fp=fopen(f, "r");

Initiate(d);

if (fp)

{

while (!feof(fp))

{

fscanf(fp, "%lf", &d->a[d->size++]);

}

fclose(fp);

}

}

//数据写入文件，将线性表数据保存

void OutputFile(Double\_list \*d, char \* f)

{

FILE \* fp = fopen(f, "w");

if (fp)

{

for (int i = 0; i < d->size; i++)

{

fprintf(fp, "%lf", &d->a[i]);

}

fclose(fp);

}

}

//线性表的插入操作，在顺序表的某个position插入某个元素

void Insert(Double\_list \*d, datatype x, int position)

{

if (d->size == MAXSIZE)

{

printf("\n顺序表已满！无法插入！");

exit(-3);

}

if (position < 0 || position > d->size)

{

printf("\n指定的插入位置不存在！");

exit(-3);

}

for (int i = d->size; i > position; i--)

{

d->a[i] = d->a[i - 1];

}

d->a[position] = x;

d->size++;

}

//线性表的删除操作,在顺序表的某个position删除某个元素

void Delete(Double\_list \*d, int position)

{

if (d->size == 0)

{

printf("\n顺序表为空！无法删除！");

exit(-4);

}

if (position < 0 || position > d->size)

{

printf("\n指定的删除位置不存在！");

exit(-4);

}

for (int i = position; i < d->size - 1; i++)

{

d->a[i] = d->a[i + 1];

}

d->size--;

}

**<random.cpp>:**

#include <iostream>

#include<string>

#include<fstream>

#include<map>

#include"Double\_list.h"

using namespace std;

int main()

{

Double\_list dlist;

Reverse(&dlist);

system("pause");

return 0;

}

测试插入删除函数时的random.cpp：

#include <iostream>

#include<string>

#include<fstream>

#include<map>

#include"Double\_list.h"

using namespace std;

int main()

{

Double\_list dlist;

Reverse(&dlist);

Insert(&dlist, 99.9, 1);

printf("\n\n");

Display(&dlist);

Delete(&dlist, 1);

printf("\n\n");

Display(&dlist);

system("pause");

return 0;

}