**《空间数据结构基础》**

**上机实验报告（2019级）**

**姓名 武成龙**

**班级 地信19-1班**

**学号 07192329**

**环境与测绘学院**

## 顺序表

**【实验简介】**

顺序表是把线性表中的所有表项按照逻辑顺序依次存储到从计算机存储中指定的存储位置开始的一块连续的存储空间中。本次实验对顺序表的C++类声明和部分操作进行实现**。**

**【实验内容】**

练习顺序表的定义和实现

**【程序代码】**

头文件“seq\_list.h”：

#pragma once

#ifndef seq\_list\_h

#define seq\_list\_h

#include<iostream>

#include<stdlib.h>

using namespace std;

const int default\_size = 100;

//与#define default\_size 100 作用相同，不过前者用只读变量，后者用宏变量

template<class t>

class seq\_list {

protected:

t\* data;//存放数组

int max\_size;//最大可容纳项数

int last;//当前数组最后位置，即实际最大项数-1

void resize(int new\_size);//改变数组空间大小

public:

void RESIZE(int n) { resize(n); }//调用改变数组空间大小私有函数

seq\_list(int sz = default\_size);//构造函数

seq\_list(seq\_list<t>& ls);//复制构造函数

~seq\_list() { delete[]data; }//析构函数

int size()const { return max\_size; };//返回最大元素个数

int length()const { return last+1; };//返回表实际长度

int search(t& x)const;//搜索变量x在表中的位置，返回下标

int locate(int i)const;//定位第i个，返回下标

void input();//输入

void output();//输出

bool get\_data(int i, t& x)const;//取第i个表项的值，不改变原值

void set\_data(int i, t& x);//用x修改第i个表项的值，因此没有const

bool insert(int i, t& x);//插入x，在第i个表项后

bool remove(int i, t& x);//删除第i个表项，返回表项的值

bool is\_empty() { return(last == -1) ? true: false;}//判断表空

bool is\_full() { return(last == max\_size - 1) ? true : false; }//判断表满

seq\_list<t>operator=(seq\_list<t>& l);

//用运算符重载实现整体赋值，和复制构造函数作用相同，不过更方便

};

//扩充顺序表

template<class t>

void seq\_list<t>::resize(int new\_size)

{

if (new\_size <= 0)

{

cout << "大小不合理！";

return;

}

if (new\_size != max\_size)

{

t\* newarray = new t[new\_size];

if (newarray == NULL)

{

cout << "内存分配错误！";

return;

}

int n = last + 1;

t\* srcarray = data;

t\* destarray = newarray;

while (n--) \*destarray++ = \*srcarray++;

delete[]data;

data = newarray;

max\_size = new\_size;

}

}

//构造函数定义

template<class t>

seq\_list<t>::seq\_list(int sz)

{

if (sz > 0)

{

max\_size = sz;

last = -1;

data = new t[max\_size];

if(data==NULL)

{

cout<< "存储分配错误！" << endl; exit(1);

}

cout << "创建成功！" << endl;

}

}

//复制构造函数定义

template<class t>

seq\_list<t>::seq\_list(seq\_list<t>& ls)

{

max\_size = ls.size();

last = ls.length()-1;

t value;//存放原数组的元素

data = new t[max\_size];

if (data == NULL)

{

cout << "存储分配错误！" << endl;

}

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

{

ls.get\_data(i, value);//获得

data[i - 1] = value;//转移

}

cout << "创建成功！" << endl;

}

//搜索

template<class t>

int seq\_list<t>::search(t& x)const

{

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

{

if (data[i] == x)

{

cout << "搜索成功！" ;

return i + 1;

}

}

cout << "搜索失败！" ;

return 0;

}

//定位

template<class t>

int seq\_list<t>::locate(int i)const {

if (i >= 1 && i <= last + 1)

{

cout << "定位成功！" ;

return i;

}

else

{

cout << "定位失败！" ;

return 0;

}

}

//依次输入

template<class t>

void seq\_list<t>::input()

{

cout << "请输入最后一个元素的下标：";

while (true)

{

cin >> last;

if (last <= max\_size - 1)

break;

cout << "输入有误！范围不能超过" << max\_size-1;

}

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

{

cout << "第" << i + 1 << "个元素为：";

cin >> data[i];

}

cout << "输入成功！" << endl;

}

//依次输出

template<class t>

void seq\_list<t>::output()

{

cout << "最大可容纳项数：" << max\_size << endl;

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

{

cout << "第" << i + 1<<"个元素为："<< data[i]<<endl;

}

cout << "输出成功！" << endl;

}

//取值

template<class t>

bool seq\_list<t>::get\_data(int i, t& x) const

{

if (i >= 1 && i <= last + 1)//大于1说明存在，小于last+1说明在范围内

{

x = data[i - 1];

return true;

}

else

return false;

}

//修改

template<class t>

void seq\_list<t>::set\_data(int i, t& x)

{

if (i >= 1 && i <= last + 1)

data[i - 1] = x;

cout << "修改成功！" << endl;

}

//插入

template<class t>

bool seq\_list<t>::insert(int i, t& x)

{//i=0时插入到第一个元素位置。i=last+1时插入到最后一个元素位置。

if (last == max\_size - 1)

return false;

if (i<0 || i>last + 1)

return false;

for (int j = last; j >= i; j--)//从最后一个开始依次往后移位

data[j + 1] = data[j];

data[i] = x;

last++;

cout << "插入成功！" << endl;

return true;

}

//删除

template<class t>

bool seq\_list<t>::remove(int i, t& x) {

if (last == -1)

return false;

if (i<1 || i>last+1)

return false;

x = data[i - 1];

for (int j = i; j <= last; j++)//从第i个开始依次往前移位

data[j - 1] = data[j];

last--;

cout << "删除成功！" << endl;

return true;

}

//整体赋值

template<class t>

seq\_list<t> seq\_list<t>::operator=(seq\_list<t>& nwl)

{

max\_size = nwl.size();

last = nwl.length() - 1;

t\* new\_array = new t[max\_size];//分配新数组

t\* src = new\_array;

for (int i = 0; i < last + 1; i++)

{

nwl.get\_data(i + 1, src[i]);

}

delete[] data;

data = src;

return \*this;

}

#endif seq\_list\_h

源文件“顺序表1.cpp”：

#include <iostream>

#include "seq\_list.h"

using namespace std;

int main()

{

int a = 3, b = 0, c = 5;

seq\_list<int> go(a);//调用构造函数

go.input();//输入元素

cout << "最大元素个数为：" << go.size() << endl;

cout << "实际长度为：" <<go.length() << endl;

go.output();//输出元素

cout << go.search(a) << endl;//在顺序表中搜索值为a的元素，若找到返回是第几个元素，没找到返回0

cout<<go.locate(1) << endl;//在顺序表中搜索第i个元素，若找到返回第i个元素位置，没找到返回0

go.get\_data(2, b);//取第2个表项的值

go.set\_data(1, c);//用c修改第1个表项的值

go.insert(0, b);//在第0个表项之后插入b

go.remove(1, b);//删除第1个表项，通过b返回表项的值

seq\_list<int> to(c);//复制构造函数

to = go;//重载=

cout<<go.is\_empty()<<endl;;//判断go是否空，空返回true，否则返回false

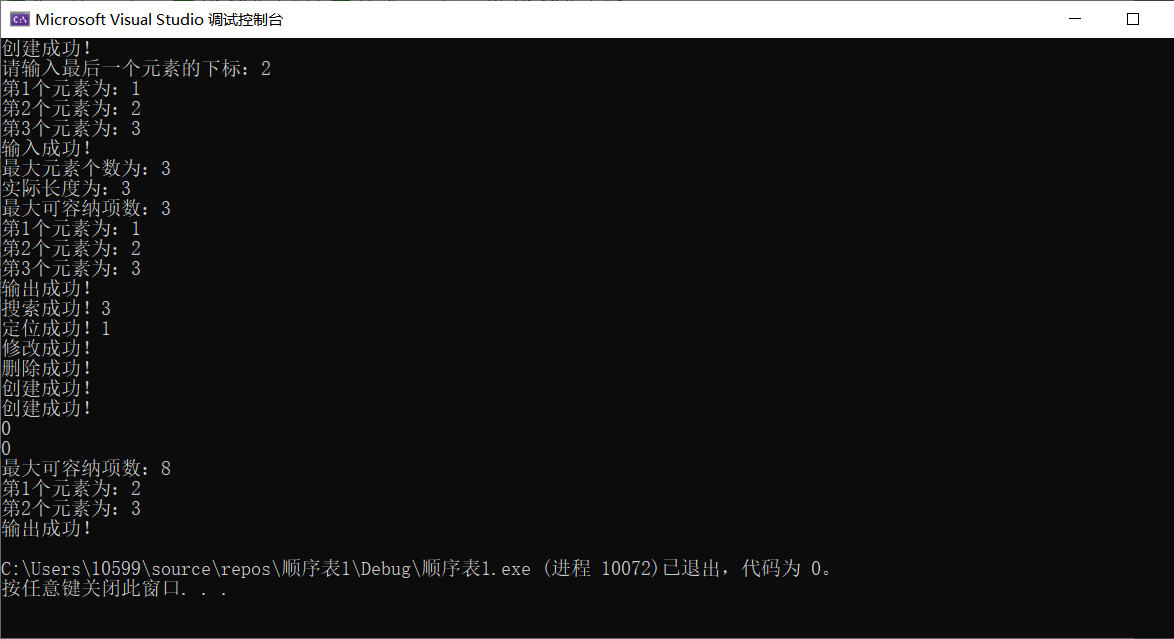
cout<<go.is\_full()<<endl;;//判断go是否满，满返回true，否则返回false

go.RESIZE(8);//改变data数组空间大小

go.output();//输出元素

}

**【实验过程】**



**【实验体会】**

先定义模板类，然后定义构造函数和析构函数，写好之后立刻调试构造函数和析构函数。通过之后再一个个添加成员函数，实现各个功能，每一个函数体的第一行加上断点并调试各种情况。千万注意maxsize和last两个关键的变量。最后再调试复制构造函数和运算符重载函数。比较困难的是插入，删除，定位和搜索函数。

## 单链表

**【实验简介】**

顺序表是把线性表中的所有表项按照逻辑顺序依次存储到从计算机存储中指定的存储位置开始的一块连续的存储空间中。本次实验对顺序表的C++类声明和部分操作进行实现**。**

**【实验内容】**

练习顺序表的定义和实现

**【程序代码】**

头文件：

#pragma once

#ifndef head\_01\_h

#define head\_01\_h

#include<stdio.h>

#include <iostream>

using namespace std;

template<class t>

struct link\_node

{

t data;//数值域

link\_node<t>\* link;//指针域

link\_node(link\_node<t>\* ptr = NULL) { link = ptr; }//只初始化指针成员的构造函数

link\_node(const t& item, link\_node<t>\* ptr = NULL) { data = item; link = ptr; }//初始化数据和指针成员的构造函数

};

template<class t>

class list

{

public:

list() { first = new link\_node<t>; cout << "创建成功！\n"; }//没有传递参数时给头指针请求内存，即让头指针指向一个新的结点。调用构造函数1

list(const t& x) { first = new link\_node<t>(x); cout << "创建成功！\n"; }//传递一个变量，请求内存，调用构造函数2

~list() {make\_empty();}

void make\_empty();//删除

int length()const;//返回长度

link\_node<t>\* search(t x);//搜索含有数据x的结点，返回元素的地址

link\_node<t>\* locate(int i);//定位，返回第i个元素的地址

bool get\_data(int i, t& x);//先定位，再取出第i个元素值

void set\_data(int i, t& x);//用x修改第i个元素值

bool insert(int i, t& x);//在第i个元素后插入

bool remove(int i, t& x);//删除第i个元素，x返回该元素的值

bool is\_empty()const { return first->link == NULL; }//first->link==null?true:false

void input\_f(t end);//前插法

void input\_r(t end);//前插法

void output();//输出

protected:

link\_node<t>\* first;//头指针

};

template<class t>

void list<t>::make\_empty() {

link\_node<t>\* q;

while (first->link != NULL)

{

q = first->link;

first->link = q->link;

delete q;

}

}

template<class t>

int list<t>::length()const {

int length = 0;

link\_node<t>\* p = first;

while (p->link != NULL) {

++length;

p = p->link;

}

/\*课本写法

link\_node<t>\* p = first->link;

while (p!= NULL) {

p = p->link;

length++;

}

\*/

return length;

}

template<class t>

link\_node<t>\* list<t>::search(t x) {

link\_node<t>\* current = first->link;

while (current!=NULL)

{

if (current->data == x) break;

else

current = current->link;

}

return current;

}

template<class t>

link\_node<t>\* list<t>::locate(int i) {

if (i < 0)return NULL;//i太小

link\_node<t>\* current = first;

int k = 0;

while (current!= NULL && k < i) {

current = current->link;

k++;

}

return current;//若返回假，i太大

}

template<class t>

bool list<t>::get\_data(int i, t& x){

if (i <= 0)return false;//i太小

link\_node<t>\* current = locate(i);

if (current == NULL)return false;//i太大

else{

x = current->data;

return true;

}

}

template<class t>

void list<t>::set\_data(int i, t& x){

if (i <= 0)return;//i太小

link\_node<t>\* current = locate(i);

if (current == NULL)return;//i太大

else {

current->data = x;

cout << "修改成功！" << endl;

}

}

template<class t>

bool list<t>::insert(int i, t& x){

link\_node<t>\* current = locate(i);

if (current == NULL) return false;

link\_node<t>\* new\_node = new link\_node<t>(x);

if (new\_node == NULL) {

cout << "插入失败！" << endl;

}

new\_node->link = current->link;

current->link = new\_node;

cout << "插入成功！" << endl;

return true;

}

template<class t>

bool list<t>::remove(int i, t& x) {

link\_node<t>\* current = locate(i-1);

if (current == NULL||current->link==NULL) return false;

link\_node<t>\* del = current->link;

x = del->data;

current->link = del->link;

delete del;

return true;

}

template<class t>

void list<t>::input\_f(t end) {

link\_node<t>\* new\_node=NULL;

t val;

make\_empty();

cout << "请输入数据（输入-1结束）" << endl;

cin >> val;

while(val != end){

new\_node = new link\_node<t>(val);

if (new\_node == NULL)

{

cout << "内存分配错误！" << endl;

exit(1);

}

new\_node->link = first->link;

first->link = new\_node;

cout << "请输入数据（输入-1结束）" << endl;

cin >> val;

}

}

template<class t>

void list<t>::input\_r(t end) {

link\_node<t>\* new\_node = NULL, \* last;

t val;

make\_empty();

cout << "请输入数据（输入-1结束）" << endl;

cin >> val;

last = first;

while(val != end){

new\_node = new link\_node<t>(val);

if (new\_node == NULL)

{

cout << "内存分配错误！" << endl;

exit(1);

last->link = new\_node;

last = new\_node;

cout << "请输入数据（输入-1结束）" << endl;

cin >> val;

}

last->link = NULL;

}

}

template<class t>

void list<t>::output() {

link\_node<t>\* current = first->link;

while (current!=NULL)//如果是current=first，然后条件是current->link呢？

{

cout << current->data << " ";

current = current->link;

}

}

#endif head\_01\_h

源文件：

#include"head\_01.h"

#include <iostream>

using namespace std;

int main()

{

int a(3),b(5),c(1);

list<int> A;//初始化A

list<int> B(a);//初始化B

B.insert(0, b);//第1个元素设为5

cout << "前插法：" << endl;

B.input\_f(-1);//前插法

cout << "后插法：" << endl;

B.input\_f(-1);//后插法

cout << "B的长度:"<<B.length() << endl;//B的长度

cout << "第1个元素的地址：" << B.locate(c) << endl;//定位

cout << "元素3的地址：" << B.search(a) << endl;//搜索

cout << "空为0，不空为1：" << B.is\_empty() << endl;//判断是否空

cout << "链表元素：";

B.output();//逐个输出

B.get\_data(1, b);///取第1个元素

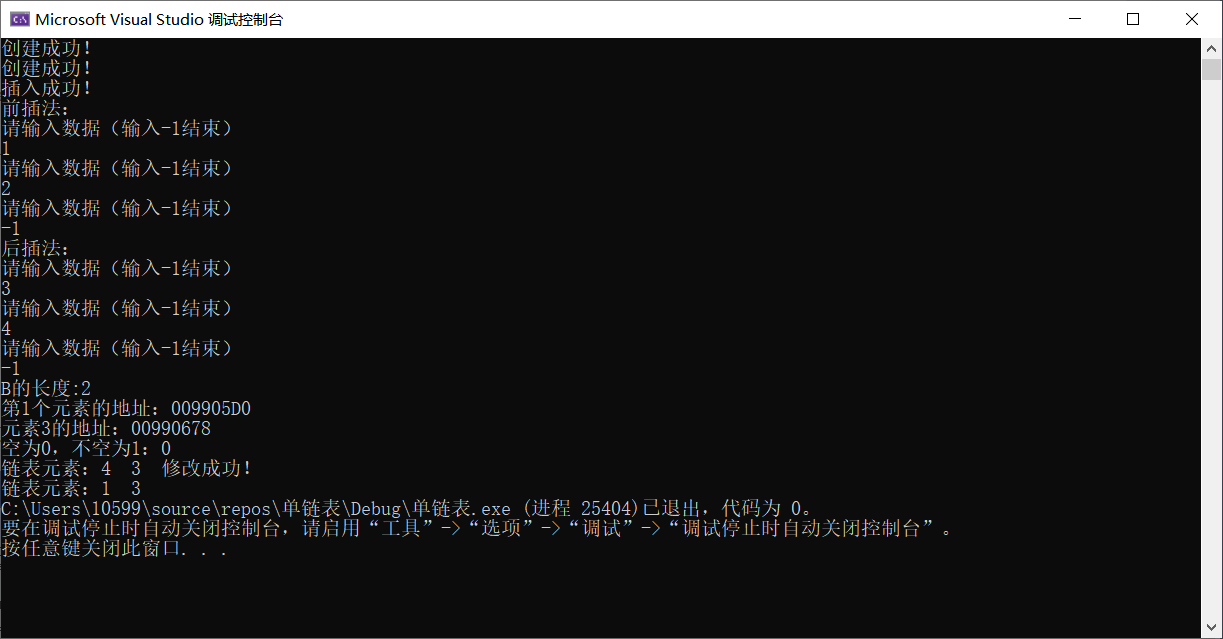
B.set\_data(1, c); //第1个元素设为1

cout << "链表元素：";

B.output();//逐个输出

}

**【实验过程】**



**【实验体会】**

实验过程中，遇到循环结构，要注意循环的结束出条件，是先进行相关操作还是先操作再移位。如果稍有不注意，就会提前结束循环或是发生越界。对链表的结构产生不理想的改变。

## 字符串类

**【实验简介】**

顺序表是把线性表中的所有表项按照逻辑顺序依次存储到从计算机存储中指定的存储位置开始的一块连续的存储空间中。本次实验对顺序表的C++类声明和部分操作进行实现**。**

**【实验内容】**

练习顺序表的定义和实现

**【程序代码】**头文件：

#pragma once

#include <string>

#include <iostream>

using namespace std;

#ifndef string\_head\_H

#define a\_string\_head\_H

#define default\_size 128

class a\_string {

private:

char\* ch;//存放字符串的数组

int length;//实际长度

int max\_size;//最大长度

public:

a\_string(int sz = default\_size);//使用默认参数的构造函数

a\_string(const char\* init);//构造函数

a\_string(const a\_string& ob);//复制构造函数

~a\_string() { delete[]ch; }//析构函数

int get\_length()const { return length; };//返回实际长度

bool operator==(a\_string& ob)const { return strcmp(ch, ob.ch) == 0; }//&和const都可以省略，还可以用友元重载

friend bool operator!=(a\_string& ob1,a\_string& ob2) { return strcmp(ob1.ch, ob2.ch) != 0; }//用友元写

bool operator!()const { return length == 0; }//判断是否为空，空为1，非空为0

a\_string& operator=(a\_string& ob);//串赋值，必须有&，否则和this指针无法比较

a\_string& operator+=(const a\_string& ob);//串连接

char& operator[](int i);//取下标为i的字符

int find(a\_string& pat)const;//匹配,返回第一个匹配到的下标

void output() { cout << ch; }

};

a\_string::a\_string(int sz) {

max\_size = sz;

ch = new char[max\_size + 1];

if (ch == NULL) {

cout << "存储分配错误！";

}

length = 0;

ch[0] = '\0';//初始化

cout << "创建成功！" << endl;

}

a\_string::a\_string(const char \*init) {

int len = strlen(init);

max\_size = (len > default\_size) ? len : default\_size;

ch = new char[max\_size + 1];

if (ch == NULL) {

cout << "存储分配错误！";

}

length = len;

strcpy\_s(ch, len+1,init);//拷贝

cout << "创建成功！" << endl;

}

a\_string::a\_string(const a\_string& ob) {

length = ob.length;

max\_size = ob.max\_size;

ch = new char[max\_size + 1];

if (ch == NULL)

cout << "内存分配错误！";

strcpy\_s(ch, max\_size + 1, ob.ch);

cout << "创建成功！" << endl;

}

a\_string& a\_string::operator=(/\*const \*/a\_string &ob) {

if (&ob != this) {

delete[]ch;

ch = new char[ob.max\_size + 1];

if (ch == NULL)

cout << "内存分配错误！" << endl;

length = ob.length;

strcpy\_s(ch, length+1,ob.ch);//拷贝

}

else

cout << "字符串自身赋值出错！" << endl;

return \*this;

}

a\_string& a\_string::operator+=(const a\_string & ob)

{

char\* temp = ch;

int n= length + ob.length;

int m = (max\_size > n) ? max\_size : n;

ch = new char[m + 1];

if (ch == NULL)

cout << "存储分配错!";

max\_size = m; length = n;

strcpy\_s(ch, max\_size + 1, temp);//拷贝

strcat\_s(ch, max\_size + 1, ob.ch);//连接

delete[]temp;

return \*this;

}

char& a\_string::operator[](int i) {

if (i < 0 || i >= length)

cout << "字符串下标错误！";

return ch[i];

}

int a\_string::find(a\_string& pat)const {//普通的模式匹配

int i, j;

if (length < pat.length) {

cout << "\n长度超限，匹配失败！";

return -1;

}

for (i = 0; i < length - pat.length; i++) {

for (j = 0; j < pat.length; j++) {

if (ch[i+j] != pat.ch[j])//+j

break;

}

if (j == pat.length)

return i;

}

return -1;

}

#endif string\_head\_H

源文件：

#include <iostream>

#include "string\_head.h"

using namespace std;

int main()

{

char a[]= "dream",b[]="ea";

a\_string A(a);

a\_string B(b);

a\_string C(A);

cout<<"\nA的长度为："<<A.get\_length();

cout << "\n两者是否相等："<<(A == B);

//cout << (A.operator==(B));//等效于上一行，用函数实现

cout << "\n两者是否不等：" <<(A != B);

cout << "\nA是否为空：" <<!A;//A!不行

//A.operator!();//等效于上一行

cout << "\n" << A.find(B);//搜索成功返回第一个下标，失败返回-1

cout << "\n" << B.find(A);

A += B;//连接

cout << "\n连接后：";

A.output();

A = B;//赋值

cout << "\n赋值后：";

1. output();

}

**【实验过程】**



**【实验体会】**

字符串主要是重载运算符比较复杂，要搞清楚运算符是单目还是双目，采用不同的定义方式，然后成员函数中注意是使用运算符左侧还是右侧的对象。