**《空间数据结构基础》**

**上机实验报告（2021级）**

**姓名 马骁**

**班级 21级地信1班**

**学号 07212393**

**环境与测绘学院**

上机实验报告要求：

1. 几个实验放到一个word文档中，word文档的命名规则为“学号 姓名”；
2. 每个实验提交的源代码放到一个目录中，只保留.h和.cpp文件，把其它文件删除；
3. C++中的类与对象

**【实验简介】**【示例，根据实际实验改写】

学会用算法语言C++描述抽象数据类型，使用模板建立数据结构。理解数据结构的组成分为两部分，第一部分是数据集（数据元素），第二部分是在此数据集上的操作。从面向对象的观点看，这两部分代表了对象的属性和方法。掌握用C++描述数据结构的基本方法，即通过建立类来描述抽象数据类型。类的数据成员提供对象属性，成员函数提供操作方法，方法是公共接口，用户通过调用方法实现对属性的访问。

**【实验内容】**【示例，根据实际实验改写】

1. 定义三维空间的坐标点TPoint
2. 描述三维空间的球TBall，实现其主要操作（如计算体积和表面积，输出空间坐标等）。

**【程序代码】**

1、定义三位空间的坐标点Tpoint

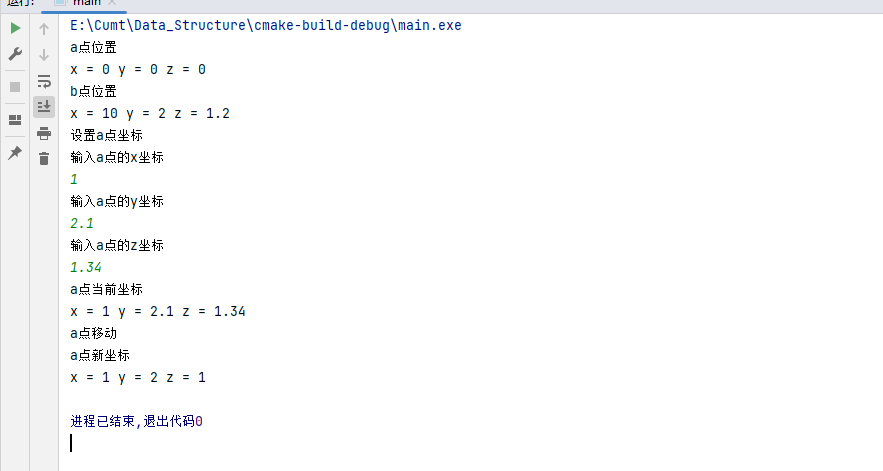
头文件 TPoint.h

#ifndef **DATA\_STRUCTURE\_TPOINT\_H**#define **DATA\_STRUCTURE\_TPOINT\_H**#include<iostream>  
using namespace std;  
class TPoint{  
private:  
 double x;  
 double y;  
 double z;  
public:  
 TPoint(){  
 x = 0;  
 y = 0;  
 z = 0;  
 }  
  
 TPoint(double x1,double y1,double z1):x(x1),y(y1),z(z1){} *//带参构造函数* ~TPoint(){}  
 void move(double x1,double y1,double z1) *//移动位置* {  
 x = x1;  
 y = y1;  
 z = z1;  
 }  
 void show() *//输出坐标* {  
 cout<<"x = "<<x<<" "<<"y = "<<y<<" z = "<<z<<endl;  
 }  
 void setx(double x1)  
 {  
 x = x1;  
  
 }  
 void sety(double y1)  
 {  
 y = y1;  
 }  
 void setz(double z1)  
 {  
 z = z1;  
 }  
};  
  
#endif *//DATA\_STRUCTURE\_TPOINT\_H*

源文件main.cpp

#include"TPoint.h"  
  
int main()  
{  
 TPoint a;  
 TPoint b(10,2.0,1.2);  
 cout<<"a点位置"<<endl;  
 a.show();  
 cout<<"b点位置"<<endl;  
 b.show();  
  
  
 double x1=0,y1=0,z1=0;  
 cout<<"设置a点坐标"<<endl;  
 cout<<"输入a点的x坐标"<<endl;  
 cin>>x1;  
 a.setx(x1);  
 cout<<"输入a点的y坐标"<<endl;  
 cin>>y1;  
 a.sety(y1);  
 cout<<"输入a点的z坐标"<<endl;  
 cin>>z1;  
 a.setz(z1);  
 cout<<"a点当前坐标"<<endl;  
 a.show();  
 cout<<"a点移动"<<endl;  
 a.move(1,2,1);  
 cout<<"a点新坐标"<<endl;  
 a.show();  
  
}

**【实验过程】**主要包括界面截图

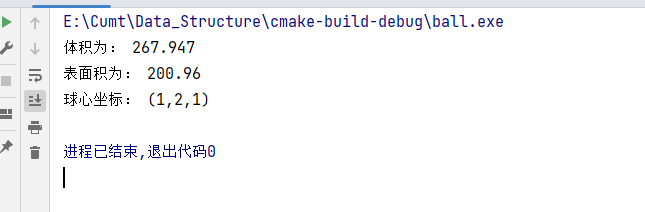


2、描述三维空间的球TBall，实现其主要操作（如计算体积和表面积，输出空间坐标等）。

头文件TBall.h

#ifndef **DATA\_STRUCTURE\_TBALL\_H**#define **DATA\_STRUCTURE\_TBALL\_H**#define **PI** 3.14  
#include<iostream>  
#include<math.h>  
using namespace std;  
  
class TBall{  
private:  
 double x,y,z,r; *//球心坐标和半径*public:  
 TBall();  
 TBall(double x1,double y1,double z1,double r1):x(x1),y(y1),z(z1),r(r1){}  
 double volume()  
 {  
 return 4.0/3.0\***PI**\*pow(r,3);  
 }  
 double SurfaceArea()  
 {  
 return 4.0\***PI**\*pow(r,2);  
 }  
 void printPoint()  
 {  
 cout<<"球心坐标： ("<<x<<","<<y<<","<<z<<")"<<endl;  
 }  
};  
#endif *//DATA\_STRUCTURE\_TBALL\_H*

源文件ball.cpp#include "TBall.h"  
  
int main()  
{  
 TBall ballA(1,2,1,4);  
 cout<<"体积为： "<<ballA.volume()<<endl;  
 cout<<"表面积为： "<<ballA.SurfaceArea()<<endl;  
 ballA.printPoint();  
 return 0;  
}



**【实验体会】**简要总结

本次实验相对来说难度较低，但还是很考验我们的基础观察能力和对C++语言的理解，

可以巩固基础知识，熟悉操作，为接下来的实验打好基础。

2. 顺序表

**【实验简介】**

学会用算法语言C++描述抽象数据类型，使用模板建立顺序表数据结构。深入理解数据结构的组成分为两部分，第一部分是数据集（数据元素），第二部分是在此数据集上的操作。从面向对象的观点看，这两部分代表了对象的属性和方法。掌握用C++描述数据结构的基本方法，即通过建立类来描述顺序表数据类型。类的数据成员提供如maxSize等对象属性，成员函数提供插入等操作方法，方法是公共接口，用户通过调用方法实现对属性或保护函数的访问。

**【实验内容】**

1.将所学的顺序表进行编写

2.将顺序表用于简单实现

3.使用顺序表各函数的功能，加深理解

4.调试教材中顺序表的模板类定义和操作

**【程序代码】**

头文件 seqList.h#ifndef **DATA\_STRUCTURE\_SEQLIST\_H**#define **DATA\_STRUCTURE\_SEQLIST\_H**#include <iostream>  
#include <stdlib.h>  
using namespace std;  
const int defaultSize = 100;  
  
template <class T>  
class SeqList{  
protected:  
 T \*data;  
 int maxSize;  
 int last;  
 void reSize(int newSize);  
public:  
 SeqList(int sz = defaultSize);  
 SeqList(SeqList<T>& L);  
 ~SeqList(){delete[] data;}  
 int Size()const{return maxSize;}  
 int Length()const{return last+1;}  
 int Search(T& x)const;  
 int Locate(int i)const;  
 bool getData(int i, T& x)const;  
 void setData(int i, T& x); *//int& x == int &x* bool Insert(int i, T& x);  
 bool Remove(int i, T& x);  
 bool IsEmpty(){return (last == -1)?true:false;}  
 bool IsFull(){return (last == maxSize)?true:false;}  
 void Expand();  
 int getBlank(){return maxSize-last;}  
 void input();  
 void output();  
 SeqList<T> operator=(SeqList<T>&L);  
};  
  
  
template<class T>  
SeqList<T>::SeqList(int sz)  
{  
 if(sz>0)  
 {  
 maxSize = sz;  
 last = -1;  
 data = new T[maxSize];  
 cout<<"顺序表初始化完成，空间大小为："<<sz<<endl;  
 if(data == **NULL**)  
 {  
 cerr<<"存储分配错误"<<endl;  
 exit(1);  
 }  
 }  
}  
  
template<class T>  
SeqList<T>::SeqList(SeqList<T> &L) { *//复制构造函数* maxSize = L.Size();  
 last = L.Length()-1;  
 T value;  
 data = new T[maxSize];  
 if(data == **NULL**)  
 {  
 cerr<<"存储分配错误！"<<endl;  
 exit(1);  
 }  
 for(int i=1;i<=last+1;i++)  
 {  
 L.getData(i,value);  
 data[i-1] = value;  
 }  
}  
  
template <class T>  
void SeqList<T>::reSize(int newSize){ *//私有函数：扩充顺序表的存储数组空间为newSize个* if(newSize<=0)  
 {  
 cerr << "无效的数组大小！" << endl;return;  
 }  
 if(newSize != maxSize){  
 T \* newarray = new T[newSize];  
 if(newarray == **NULL**)  
 {  
 cerr << "存储分配错误！" << endl;exit(1);  
 }  
 int n = last+1;  
 T \* srcptr = data;  
 T \* desptr = newarray;  
 while(n--) \*desptr++=\*srcptr++;  
 delete []data;  
 data = newarray;maxSize = newSize;  
 }  
}  
  
template <class T>  
int SeqList<T>::Search(T& x)const{ *//搜索函数返回顺序表中x的序号* for (int i = 0; i<=last; i++)  
 if(data[i] == x)  
 return i+1;  
 return 0;  
}  
  
  
*//返回序号为i的值*template <class T>  
int SeqList<T>::Locate(int i)const{  
 if(i >= 1 && i <= last+1)  
 return data[i-1];  
 else  
 return 0;  
}  
template<class T>  
bool SeqList<T>::getData(int i, T &x) const {  
 if(i>0 && i<= last+1)  
 {  
 x = data[i-1];  
 return true;  
 }  
}

template <class T>  
void SeqList<T>::setData(int i, T &x) {  
 if(i >0 &&i<= last+1)  
 {  
 data[i-1] = x;  
 }  
}  
  
  
*//插入至序号为2的地方，而不是数组的下标为2*template<class T>  
bool SeqList<T>::Insert(int i, T &x) {  
 if(last == maxSize)  
 {  
 cout<<"插入失败！顺序表已满"<<endl;  
 }  
 if(i<0 || i>last+1)  
 {  
 cout<<"插入失败！指定位置不存在！"<<endl;  
 }  
 for(int j = last;j>=i-1;j--)  
 {  
 data[j+1] = data[j];  
 }  
 data[i-1] = x;  
 last++;  
 return true;  
}  
  
template <class T>  
bool SeqList<T>::Remove(int i, T& x){  
 if(last == -1)  
 {  
 cout << "删除失败！顺序表为空！" << endl;  
 return false;  
 }  
 if(i < 1||i > last+1)  
 {  
 cout << "删除失败！指定位置不存在！" << endl;  
 return false;  
 }  
 x = data[i-1];  
 for(int j = i;j <= last;j++)  
 data[j-1] = data[j];  
 last--;  
 return true;  
}  
  
template <class T>  
void SeqList<T>::Expand(){  
 if ((double)maxSize/(double)last >= 0.5)  
 {  
 reSize(2 \* maxSize);  
 cout << "已自动扩展顺序表空间至原来的两倍" << endl;  
 }  
}  
  
template<class T>  
void SeqList<T>::input() {  
 cout<<"开始进行输入操作,请输入表中的元素个数： ";  
 while(1)  
 {  
 cin>>last;  
 if(last <= maxSize) break;  
 cout<<"元素个数输入有误，范围不能超过"<<maxSize<<".请重新输入:";  
 }  
 for(int i =0;i<last;i++)  
 {  
 cin>>data[i];  
 cout<<"//已输入顺序为"<<i+1<<"的元素"<<endl;  
 }  
}  
  
template<class T>  
void SeqList<T>::output(){  
 cout<<"顺序表当前元素的最后位置为: "<<last<<endl;  
 cout<<"顺序表输出"<<endl;  
 for(int i =0;i<last;i++)  
 {  
 cout<<"["<<i+1<<"]: "<<data[i]<<endl;  
 }  
}  
  
  
  
template <class T>  
SeqList<T> SeqList<T>::operator=(SeqList<T> &L){  
 maxSize = L.Size();  
 last = L.Length() - 1;  
 T value;  
 data = new T[maxSize];  
 if(data == **NULL**)  
 {  
 cerr << "存储分配错误！" << endl;  
 exit(1);  
 }  
 for(int i = 1;i<=last+1;i++)  
 {  
 L.getData(i, value);  
 data[i-1]=value;  
 }  
 return \*this;  
}  
#endif *//DATA\_STRUCTURE\_SEQLIST\_H*

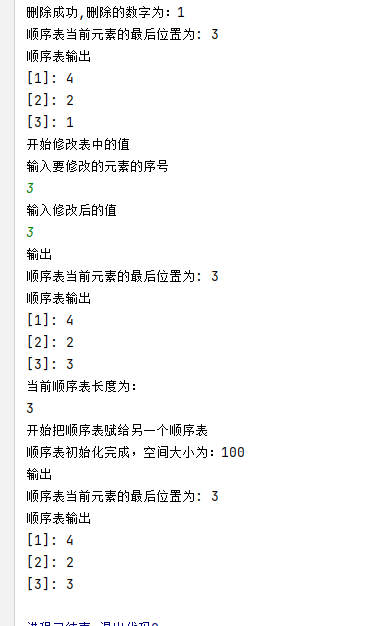
源文件 seqList1.cpp

#include "seqlist.h"  
int main()  
{  
 int size;  
 cout<<"准备建立顺序表，请输入顺序表最大容量：";  
 cin>>size;  
 SeqList<int> L(size);  
 if(L.IsEmpty())  
 {  
 cout<<"该顺序表为空"<<endl;  
 }  
 L.input();  
 cout<<"顺序表输入完成"<<endl;  
  
  
 L.output();  
  
 cout<<"返回序号为3的值"<<endl;  
 cout<<L.Locate(3)<<endl;  
  
 cout<<"开始插入元素："<<endl;  
 int a,b,c,d;  
 cout<<"输入插入的序号："<<endl;  
 cin>>a;  
 cout<<"输入插入的值："<<endl;  
 cin>>b;  
 if(L.Insert(a,b))  
 cout<<"插入成功";  
 L.output();  
  
 cout<<"开始搜索： "<<endl;  
 int x;  
 cin>>x;  
 if(L.Search(x))  
 {  
 cout<<"搜索成功"<<" "<<x<<"位于表中第"<<L.Search(x)<<"个元素"<<endl;  
 }  
  
 cout<<"开始删除元素："<<endl;  
 cout<<"输入要删除的序号："<<endl;  
 cin>>c;  
 if(L.Remove(c,d))  
 cout<<"删除成功,删除的数字为："<<d<<endl;  
  
 L.output();  
  
 cout<<"开始修改表中的值"<<endl;  
 int e,f;  
 cout<<"输入要修改的元素的序号"<<endl;  
  
 cin>>e;  
 cout<<"输入修改后的值"<<endl;  
 cin>>f;  
 L.setData(e,f);  
 cout<<"输出"<<endl;  
 L.output();  
 cout<<"当前顺序表长度为："<<endl;  
 cout<<L.Length()-1<<endl;  
  
 cout<<"开始把顺序表赋给另一个顺序表"<<endl;  
 SeqList<int> L1;  
 L1 = L;  
 cout<<"输出"<<endl;  
 L1.output();

return 0;  
}

**【实验过程】**主要包括界面截图





**【实验体会】**

本次实验是顺序表的编写，难度适中，但是处理和分析下标造成了一些困扰，最后成功解决，其中模板类更能体现出C++程序的特色。

3.

3. 用模板和非模板定义的顺序表排序类

**【实验简介】**

学会用算法语言C++描述抽象数据类型，理解使用模板建立数据结构。理解数据结构的组成分为两部分，第一部分是数据集（数据元素），第二部分是在此数据集上的操作。从面向对象的观点看，这两部分代表了对象的属性和方法。掌握用C++描述数据结构的基本方法，即通过建立类来描述抽象数据类型。类的数据成员提供对象属性，成员函数提供操作方法，方法是公共接口，用户通过调用方法实现对属性的访问。

**【实验内容】**

1. 调试PPT中的相关程序
2. 理解两种方式的不同与相同之处并进行比较

**【程序代码】**

**模板定义的顺序表排序类**

头文件seqlist.h

#ifndef DATA\_STRUCTURE\_SEQLIST\_H  
#define DATA\_STRUCTURE\_SEQLIST\_H  
  
  
#include <iostream>  
#include <stdlib.h>  
using namespace std;  
const int defaultSize = 100;  
  
template <class T>  
class SeqList{  
protected:  
 T \*data;  
 int maxSize;  
 int last;  
 void reSize(int newSize);  
 void Swap(int m1,int m2);  
 int maxKey(int low,int high);  
public:  
 void sort();  
 SeqList(int sz = defaultSize);  
 SeqList(SeqList<T>& L);  
 ~SeqList(){delete[] data;}  
 int Size()const{return maxSize;}  
 int Length()const{return last+1;}  
 int Search(T& x)const;  
 int Locate(int i)const;  
 bool getData(int i, T& x)const;  
 void setData(int i, T& x); //int& x == int &x  
 bool Insert(int i, T& x);  
 bool Remove(int i, T& x);  
 bool IsEmpty(){return (last == -1)?true:false;}  
 bool IsFull(){return (last == maxSize)?true:false;}  
 void Expand();  
 int getBlank(){return maxSize-last;}  
 void input();  
 void output();  
 SeqList<T> operator=(SeqList<T>&L);  
};  
  
  
template<class T>  
SeqList<T>::SeqList(int sz)  
{  
 if(sz>0)  
 {  
 maxSize = sz;  
 last = -1;  
 data = new T[maxSize];  
 cout<<"顺序表初始化完成，空间大小为："<<sz<<endl;  
 if(data == NULL)  
 {  
 cerr<<"存储分配错误"<<endl;  
 exit(1);  
 }  
 }  
}  
  
template<class T>  
SeqList<T>::SeqList(SeqList<T> &L) { //复制构造函数  
 maxSize = L.Size();  
 last = L.Length()-1;  
 T value;  
 data = new T[maxSize];  
 if(data == NULL)  
 {  
 cerr<<"存储分配错误！"<<endl;  
 exit(1);  
 }  
 for(int i=1;i<=last+1;i++)  
 {  
 L.getData(i,value);  
 data[i-1] = value;  
 }  
}  
  
template <class T>  
void SeqList<T>::reSize(int newSize){ //私有函数：扩充顺序表的存储数组空间为newSize个  
 if(newSize<=0)  
 {  
 cerr << "无效的数组大小！" << endl;return;  
 }  
 if(newSize != maxSize){  
 T \* newarray = new T[newSize];  
 if(newarray == NULL)  
 {  
 cerr << "存储分配错误！" << endl;exit(1);  
 }  
 int n = last+1;  
 T \* srcptr = data;  
 T \* desptr = newarray;  
 while(n--) \*desptr++=\*srcptr++;  
 delete []data;  
 data = newarray;maxSize = newSize;  
 }  
}  
  
template <class T>  
int SeqList<T>::Search(T& x)const{ //搜索函数返回顺序表中x的序号  
 for (int i = 0; i<=last; i++)  
 if(data[i] == x)  
 return i+1;  
 return 0;  
}  
  
  
//返回序号为i的值  
template <class T>  
int SeqList<T>::Locate(int i)const{  
 if(i >= 1 && i <= last+1)  
 return data[i-1];  
 else  
 return 0;  
}  
template<class T>  
bool SeqList<T>::getData(int i, T &x) const {  
 if(i>0 && i<= last+1)  
 {  
 x = data[i-1];  
 return true;  
 }  
}  
  
  
//不知到编书的人怎么想的，处理下标问题处理这么麻烦，给自己增加工作量  
template <class T>  
void SeqList<T>::setData(int i, T &x) {  
 if(i >0 &&i<= last+1)  
 {  
 data[i-1] = x;  
 }  
}  
  
  
//插入至序号为2的地方，而不是数组的下标为2  
template<class T>  
bool SeqList<T>::Insert(int i, T &x) {  
 if(last == maxSize)  
 {  
 cout<<"插入失败！顺序表已满"<<endl;  
 }  
 if(i<0 || i>last+1)  
 {  
 cout<<"插入失败！指定位置不存在！"<<endl;  
 }  
 for(int j = last;j>=i-1;j--)  
 {  
 data[j+1] = data[j];  
 }  
 data[i-1] = x;  
 last++;  
 return true;  
}  
  
template <class T>  
bool SeqList<T>::Remove(int i, T& x){  
 if(last == -1)  
 {  
 cout << "删除失败！顺序表为空！" << endl;  
 return false;  
 }  
 if(i < 1||i > last+1)  
 {  
 cout << "删除失败！指定位置不存在！" << endl;  
 return false;  
 }  
 x = data[i-1];  
 for(int j = i;j <= last;j++)  
 data[j-1] = data[j];  
 last--;  
 return true;  
}  
  
template <class T>  
void SeqList<T>::Expand(){  
 if ((double)maxSize/(double)last >= 0.5)  
 {  
 reSize(2 \* maxSize);  
 cout << "已自动扩展顺序表空间至原来的两倍" << endl;  
 }  
}  
  
template<class T>  
void SeqList<T>::input() {  
 cout<<"开始进行输入操作,请输入表中的元素个数： ";  
 while(1)  
 {  
 cin>>last;  
 if(last <= maxSize) break;  
 cout<<"元素个数输入有误，范围不能超过"<<maxSize<<".请重新输入:";  
 }  
 for(int i =0;i<last;i++)  
 {  
 cin>>data[i];  
 cout<<"//已输入顺序为"<<i+1<<"的元素"<<endl;  
 }  
}  
  
template<class T>  
void SeqList<T>::output(){  
 cout<<"顺序表当前元素的最后位置为: "<<last<<endl;  
 cout<<"顺序表输出"<<endl;  
 for(int i =0;i<last;i++)  
 {  
 cout<<"["<<i+1<<"]: "<<data[i]<<endl;  
 }  
}  
  
  
  
template <class T>  
SeqList<T> SeqList<T>::operator=(SeqList<T> &L){  
 maxSize = L.Size();  
 last = L.Length() - 1;  
 T value;  
 data = new T[maxSize];  
 if(data == NULL)  
 {  
 cerr << "存储分配错误！" << endl;  
 exit(1);  
 }  
 for(int i = 1;i<=last+1;i++)  
 {  
 L.getData(i, value);  
 data[i-1]=value;  
 }  
 return \*this;  
}  
  
  
#endif //DATA\_STRUCTURE\_SEQLIST\_H

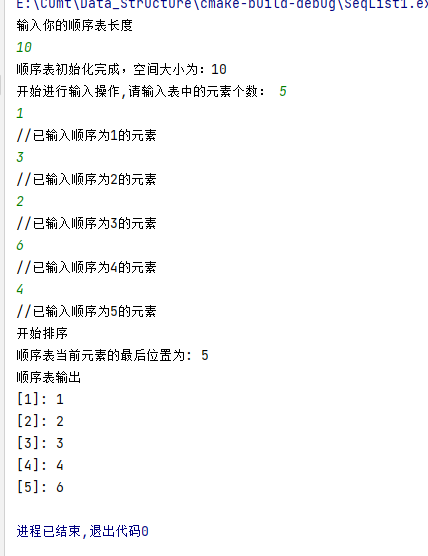
头文件selectT.h#ifndef **DATA\_STRUCTURE\_SELECTT\_H**#define **DATA\_STRUCTURE\_SELECTT\_H**#include"seqlist.h"  
template<class T>  
void SeqList<T>::Swap(int m1,int m2)  
{  
 T temp = data[m1];  
 data[m1] = data[m2];  
 data[m2] = temp;  
}  
  
template<class T>  
int SeqList<T>::maxKey(int low, int high) {  
 int max = low;  
 for(int k = low+1;k<=high;k++)  
 {  
 if(data[max]<data[k])  
 {  
 max = k;  
 }  
 }  
 return max;  
}  
  
  
template<class T>  
void SeqList<T>::sort() {  
 for(int i = last-1;i>0;i--)  
 {  
 int j = maxKey(0,i);  
 if(j!=i) Swap(j,i);  
 }  
}  
#endif *//DATA\_STRUCTURE\_SELECTT\_H*

源文件

Seqlist1.cpp

#include"selectT.h"  
#include "seqlist.h"  
  
int main()  
{  
 int size;  
 cout<<"输入你的顺序表长度"<<endl;  
 cin>>size;  
 SeqList<int>List(size);  
 List.input();  
 cout<<"开始排序"<<endl;  
 List.sort();  
 List.output();  
 return 0;  
  
}

**【实验过程】**主要包括界面截图



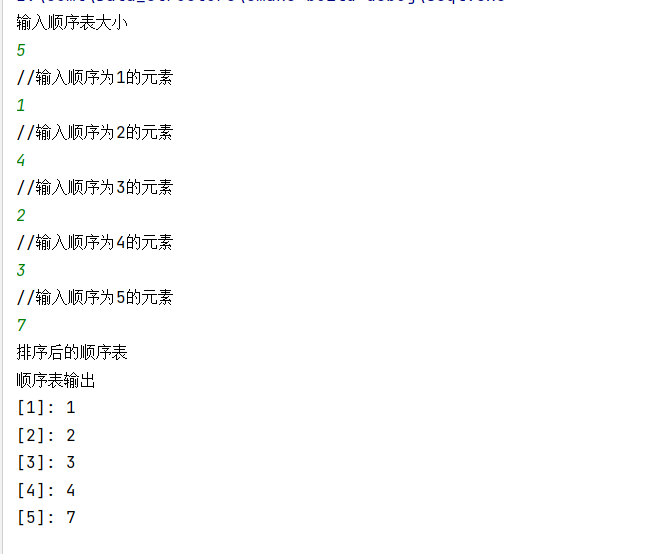
**非模板定义的顺序表排序类**

头文件seqList.h#ifndef DATA\_STRUCTURE\_SEQLIST\_H  
#define DATA\_STRUCTURE\_SEQLIST\_H  
  
#include<iostream>  
using namespace std;  
class seq{  
private:  
 int \*data;  
 int Size;  
public:  
 void Swap(int m1,int m2);  
 int maxKey(int low,int high);  
 seq(int size = 10)  
 {  
 Size = size;  
 data = new int[size];  
 }  
 ~seq()  
 {  
 delete[]data;  
 }  
 void Sort();  
 void input();  
 void output();  
};  
void seq::input() {  
 for(int i =0;i<Size;i++)  
 {  
 cout<<"//输入顺序为"<<i+1<<"的元素"<<endl;  
 cin>>data[i];  
 }  
}  
  
void seq::output(){  
  
 cout<<"顺序表输出"<<endl;  
 for(int i =0;i<Size;i++)  
 {  
 cout<<"["<<i+1<<"]: "<<data[i]<<endl;  
 }  
}  
  
  
void seq::Sort()  
{  
 for(int i = Size-1;i>0;i--)  
 {  
 int j = maxKey(0,i);  
 if(j!=i) Swap(j,i);  
 }  
}  
  
void seq::Swap(int m1,int m2)  
{  
 int temp = data[m1];  
 data[m1] = data[m2];  
 data[m2] = temp;  
}  
  
int seq::maxKey(int low, int high) {  
 int max = low;  
 for(int k = low+1;k<=high;k++)  
 {  
 if(data[max]<data[k])  
 {  
 max = k;  
 }  
 }  
 return max;  
}  
  
#endif //DATA\_STRUCTURE\_SEQLIST\_H

源文件

#include"seqList.h"  
  
int main()  
{  
 int size;  
 cout<<"输入顺序表大小"<<endl;  
 cin>>size;  
 seq a(size);  
 a.input();  
 a.Sort();  
 cout<<"排序后的顺序表"<<endl;  
 a.output();  
 return 0;  
}

**【实验过程】**主要包括界面截图



【实验体会】

利用已经写过的顺序表头文件，避免重复书写代码。在排序功能实现时定义了一个maxKey函数，以获取顺序表元素中最大数的下标，利用选择排序对顺序表进行排序操作。加深了数据结构的基础知识，提高编写代码的能力。

4. 带附加表头结点的单链表的类

**【实验简介】**

学会用算法语言C++描述抽象数据类型，使用模板建立单链表数据结构。掌握用C++描述带附加表头的单链表数据结构的基本方法，即通过建立类来描述抽象数据类型。类的数据成员提供对象属性，成员函数提供操作方法，方法是公共接口，用户通过调用方法实现对属性的访问。

**【实验内容】**

调试教材中带附加表头节点的单链表的模板类定义和操作

**【程序代码】**

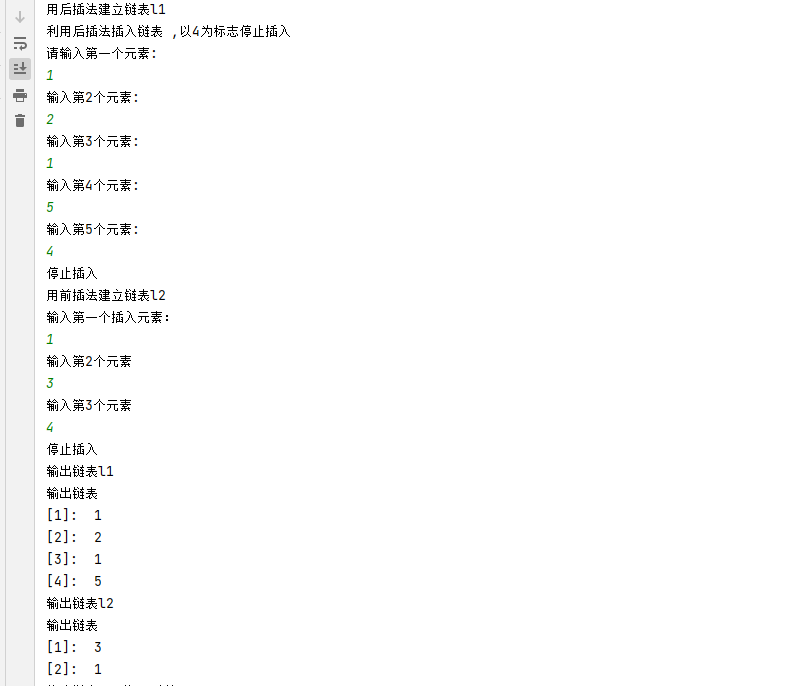
头文件 LinkList.h

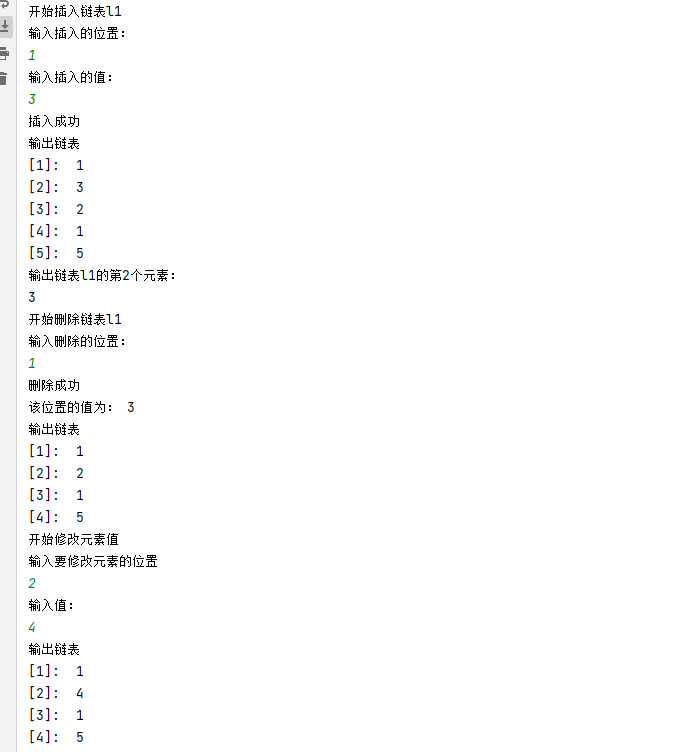
#ifndef **DATA\_STRUCTURE\_LINKLIST\_H**#define **DATA\_STRUCTURE\_LINKLIST\_H**#include<iostream>  
using namespace std;  
  
template<class T>  
struct LinkNode{  
  
 T data;  
 LinkNode<T> \*next;  
 LinkNode(LinkNode<T>\*ptr = **NULL**) {  
 next = ptr;  
 }  
 LinkNode(T x,LinkNode \*next1 = nullptr)  
 {  
 data = x;  
 next = next1;  
  
 }  
};  
  
  
template<class T>  
class List{  
private:  
 LinkNode<T> \*first;  
public:  
 List()  
 {  
 first = nullptr;  
 };  
 List(const T& x){  
 first = new LinkNode<T>(x);  
 }  
  
 ~List()  
 {  
 makeEmpty();  
 };  
  
  
  
 LinkNode<T>\* getHead()const{return first;}  
 void output();  
 void makeEmpty();  
 bool Insert(int i,T& x);  
 bool Remove(int i,T& x);  
 LinkNode <T> \*Locate(int i); *//取出第i个元素的地址* void inputRear(T end);*//后插法建立单链表* void inputFront(T end);*//前插法建立单链表* void setData(int i,T&x);  
 bool getData(int i,T&x);  
 bool IsEmpty()const{  
 return first->next == **NULL**? true:false;  
 }  
 void sort();  
 List<T>& operator=(const List<T>& L);  
  
};  
  
  
template<class T>  
void List<T>::makeEmpty() {  
 LinkNode<T> \*cur = first;  
 while(cur != nullptr)  
 {  
 LinkNode<T> \*tmp = cur;  
 cur = cur->next;  
 delete tmp;  
 }  
 first = nullptr;  
}  
  
template<class T>  
bool List<T>::Insert(int i, T &x) {  
 if(i==0)  
 {  
 LinkNode<T> \*node = new LinkNode<T>(x);  
 node->next = first;  
 first = node;  
 }  
 else{  
 LinkNode<T> \* cur = Locate(i);  
 LinkNode<T> \*node = new LinkNode<T>(x);  
 node->next = cur->next;  
 cur->next = node;  
 return true;  
 }  
}  
  
  
template<class T>  
bool List<T>::Remove(int i, T &x) {  
 LinkNode<T> \*cur = Locate(i-1);  
 if(cur == **NULL** ||cur->next == **NULL**) return false;  
  
 LinkNode<T> \*del = cur->next;  
 cur->next = del->next;  
 x = del->data;  
 delete del;  
 return true;  
}  
  
template<class T>  
LinkNode<T>\* List<T>::Locate(int i) {  
 if(i == 0) return first;  
 if(i < 0) return nullptr;  
 LinkNode<T> \*current = first;  
 for(int j = 1; j < i && current != nullptr; j++) {  
 current = current->next;  
 }  
 return current;  
}  
  
template<class T>  
void List<T>::inputFront(T end) {  
  
 LinkNode<T> \*node;  
 T val;  
 makeEmpty();  
 cout<<"输入第一个插入元素："<<endl;  
 cin>>val;  
 int i=1;  
 if(val == end)  
 {  
 cout<<"停止插入"<<endl;  
 }  
 while(val != end)  
 {  
 LinkNode<T> \*node = new LinkNode<T>(val);  
 if(node == nullptr)  
 {  
 cerr<<"存储分配错误！"<<endl;  
 exit(1);  
 }  
 node->next = first;  
 first = node;  
 i++;  
 cout<<"输入第"<<i<<"个元素"<<endl;  
 cin>>val;  
 if(val == end)  
 {  
 cout<<"停止插入"<<endl;  
 }  
  
 }  
}  
  
  
  
template<class T>  
bool List<T>::getData(int i, T &x) {  
 if(i<=0) return **NULL**;  
 LinkNode<T> \*cur = Locate(i);  
 if(cur == **NULL**) return false;  
 else{  
 x = cur->data;  
 return true;  
 }  
}  
  
template<class T>  
void List<T>::setData(int i, T &x) {  
 if(i<=0) return;  
 LinkNode<T> \*cur = Locate(i);  
 if(cur == **NULL**) return;  
 else cur->data = x;  
}  
  
template<class T>  
void List<T>::inputRear(T end) {  
 LinkNode<T> \*tail = nullptr;  
 makeEmpty();  
 T val;  
 cout<<"利用后插法插入链表 ,以"<<end<<"为标志停止插入"<<endl;  
 cout<<"请输入第一个元素: "<<endl;  
 cin>>val;  
 int i =1;  
 while(val != end)  
 {  
 LinkNode<T> \*node = new LinkNode<T>(val);  
 if(first == nullptr) *//如果链表为空，将新节点设置为头结点和尾节点* {  
 first = node;  
 tail = node;  
 }  
 else{  
 tail->next = node;  
 tail = node;  
 }  
 i++;  
 cout<<"输入第"<<i<<"个元素: "<<endl;  
 cin>>val;  
 if(val == end)  
 {  
 cout<<"停止插入"<<endl;  
 }  
 }  
}  
  
template<class T>  
void List<T>::output()  
{  
 cout<<"输出链表"<<endl;  
 LinkNode<T> \*cur = first; *//从头结点开始遍历* int i=1;  
 while(cur)  
 {  
 cout<<"["<<i<<"]: "<<cur->data<<endl;  
 cur = cur->next;  
 i++;  
 }  
  
}  
  
  
template<class T>  
List<T> &List<T>::operator=(const List<T> &L) {  
 if (this == &L) return \*this; *// 处理自我赋值的情况* makeEmpty(); *// 清空目标链表* T value;  
 LinkNode<T> \*srcptr = L.getHead(); *// 指向源链表的第一个数据节点* LinkNode<T> \*destptr = first = new LinkNode<T>; *//新链表的头指针* while(srcptr != **NULL**){  
 value = srcptr->data; *// 复制源链表节点的值到目标链表节点* destptr->next = new LinkNode<T>(value); *//新链表初始化节点，用L的每个节点的value* destptr = destptr->next;  
 srcptr = srcptr->next;  
 }  
 destptr->next = **NULL**; *//新链表末尾置空* return \*this;  
}  
  
template<class T>  
void List<T>::sort()  
{  
 LinkNode<T> \*p, \*q, \*end = **NULL**;  
 for (p = first; p->next != end; end = p, p = first) {  
 for (q = p->next; q != end; p = q, q = q->next) {  
 if (p->data > q->data) {  
 swap(p->data, q->data);  
 }  
 }  
 }  
 cout << "排序成功！" << endl;  
}  
  
  
#endif *//DATA\_STRUCTURE\_LINKLIST\_H*

源文件 list.cpp

#include<iostream>  
#include"LinkList.h"  
using namespace std;  
int main()  
{  
  
 List<int> l1,l2;  
 cout<<"用后插法建立链表l1"<<endl;  
 l1.inputRear(4);  
 cout<<"用前插法建立链表l2"<<endl;  
 l2.inputFront(4);  
 cout<<"输出链表l1"<<endl;  
 l1.output();  
 cout<<"输出链表l2"<<endl;  
 l2.output();  
  
 cout<<"构建链表l3,将l1赋给l3"<<endl;  
 List<int> l3;  
 l3 = l1;  
 l3.output();  
  
 cout<<"开始插入链表l1"<<endl;  
 int a1,b1;  
 cout<<"输入插入的位置： "<<endl; *//插入在该位置后一个结点* cin>>a1;  
 cout<<"输入插入的值： "<<endl;  
 cin>>b1;  
 if(l1.Insert(a1,b1))  
 {  
 cout<<"插入成功"<<endl;  
 }  
 else cout<<"输入插入失败"<<endl;  
 l1.output();  
 cout<<"输出链表l1的第2个元素： "<<endl;  
 cout<<l1.Locate(2)->data<<endl;  
  
 cout<<"开始删除链表l1"<<endl;  
 int a2,b2;  
 cout<<"输入删除的位置： "<<endl;  
 cin>>a2;  
 if(l1.Remove(a2,b2))  
 {  
 cout<<"删除成功"<<endl;  
 cout<<"该位置的值为： "<<b2<<endl;  
 }  
 l1.output();  
  
 cout<<"开始修改元素值"<<endl;  
 int a3,b3;  
 cout<<"输入要修改元素的位置"<<endl;  
 cin>>a3;  
 cout<<"输入值： "<<endl;  
 cin>>b3;  
 l1.setData(a3,b3);  
 l1.output();  
  
 cout<<"开始获取元素值："<<endl;  
 int a4,b4;  
 cout<<"输入位置"<<endl;  
 cin>>a4;  
 if(l1.getData(a4,b4))  
 {  
 cout<<"获取成功"<<endl;  
 cout<<"获取的值是： "<<b4<<endl;  
 }  
  
 cout<<"开始排序"<<endl;  
 l1.sort();  
 l1.output();  
  
 return 0;  
}

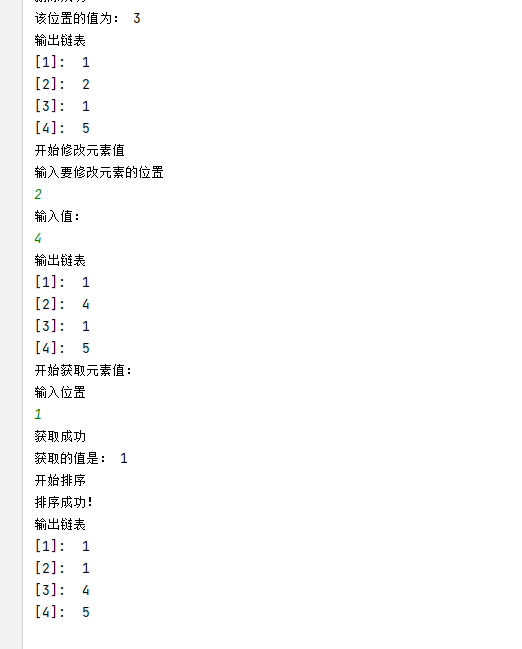
**【实验过程】**主要包括界面截图





【实验体会】

编写附加表头节点的单链表类，需要实现分功能较多，任务量较大，花费时间最长，但也加深了对链表的基础知识。



5. 字符串

【实验简介】字符串是由零个或多个字符的顺序排列所组成的数据结构，字符串在计算机处理中使用非常广泛。通过本次实验理解字符串运算的原理，掌握主要算法的实现。

【实验内容】

建立字符串类，并实现求子串、字符串赋值、字符串连接等运算符重载函数，实现字符串的模式匹配功能。编写一个能够统计字符串中各个字符出现频度的函数。

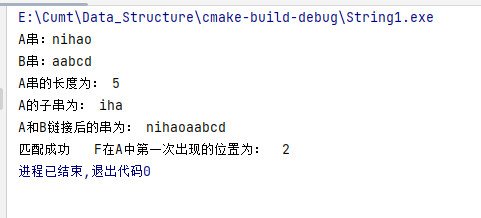
【主要代码】

String.h#include<iostream>  
#include<string.h>  
using namespace std;  
  
#ifndef **DATA\_STRUCTURE\_STRING\_H**#define **DATA\_STRUCTURE\_STRING\_H**#define **maxSize** 128  
  
class String{  
public:  
 String(int sz = **maxSize**); *//创建一个最大长度为maxSize，实际长度为0的串* String(const char \*init);  
 String(const String& ob); *//复制构造函数，用已有的ob创建一个新串* ~String()  
 {  
 delete []ch;  
 }  
 int Length()const  
 {  
 return curLength;  
 }  
 char& operator[](int i);  
 String& operator +=(String& ob); *//串链接* friend ostream &operator << (ostream &out,String &P); *//字符串输出* String& operator=(const String&ob);  
 String operator() (int pos,int len);  
 int Find(String& pat,int k);  
  
 char \*ch;  
 int curLength; *//字符串实际长度* int maxLength; *//存放数组的最大长度*};  
  
String::String(int sz) {  
 maxLength = sz;  
 ch = new char[maxLength +1];  
 if(ch == **NULL**){  
 cerr<<"分配错误"<<endl;  
 exit(1);  
 }  
 curLength = 0;  
 ch[0] = '\0';  
}  
  
String::String(const char \*init) {  
 int len = strlen(init);  
 maxLength = (len>**maxSize**)?len:**maxSize**;  
 ch = new char[maxLength+1];  
 if(ch == **NULL**){  
 cerr<<"分配错误"<<endl;  
 exit(1);  
 }  
 curLength = len;  
 strcpy(ch,init);  
}  
  
String::String(const String &ob) {  
 maxLength = ob.maxLength;  
 ch = new char[maxLength + 1];  
 if(ch == **NULL**)  
 {  
 cerr<<"存储分配失败"<<endl;  
 exit(1);  
 }  
 curLength = ob.curLength;  
 strcpy(ch,ob.ch);  
}  
String& String::operator=(const String &ob) {  
 if(&ob != this)  
 {  
 delete []ch;  
 ch = new char[ob.maxLength];  
 if(ch==**NULL**){  
 cerr<<"存储分配失败"<<endl;  
 exit(1);  
 }  
 curLength = ob.curLength;  
 strcpy(ch,ob.ch);  
 }  
 else cout<<"字符串自身赋值错误"<<endl;  
 return \*this;  
}  
  
  
char& String::operator[] (int i) {  
  
 if(i<0||i>=curLength)  
 {  
 cout<<"字符串下标超界限"<<endl;  
 exit(1);  
 }  
 return ch[i];  
}  
  
String& String::operator+=(String &ob) {  
 char \*tmp = ch;  
 int n = curLength+ob.curLength;  
 int m = (maxLength>=n)?maxLength:n;  
 ch = new char[m];  
 if(ch == **NULL**){  
 cerr<<"分配错误"<<endl;  
 exit(1);  
 }  
 maxLength = m; *//新字符串的最大长度 为m* curLength = n; *//新字符串的当前长度 n* strcpy(ch,tmp);  
 strcat(ch,ob.ch);  
 delete []tmp;  
 return \*this;  
}  
  
  
String String::operator()(int pos, int len) {  
 String tmp;  
 if(pos<0||pos+len>=maxLength||len<0)  
 {  
 tmp.curLength =0;  
 tmp.ch[0] = '\0';  
 }  
 else{  
 if(pos+len>=curLength) len = curLength-pos;  
 tmp.curLength = len;  
 for(int i =0,j = pos;i<len;i++,j++) tmp.ch[i] = ch[j];  
 tmp.ch[len] = '\0';  
 }  
 return tmp;  
}  
int String::Find(String &pat, int k) {  
 int i,j;  
 for(i = k;i<=curLength -pat.curLength;i++)  
 {  
 for(j = 0;j<pat.curLength;j++)  
 {  
 if(ch[i+j]!=pat.ch[j]) break;  
  
 }  
 if(j == pat.curLength) return i;  
  
 }  
 return -1;  
}  
  
ostream &operator << (ostream &out,String &P){ *//打印字符串* for(int i=0;i<P.curLength;i++)  
 cout<<P.ch[i];  
 return out;  
}  
  
  
#endif *//DATA\_STRUCTURE\_STRING\_H*

源文件String1.cpp

#include "String.h"  
int main()  
{  
 char a[30] = "nihao"; *//数组存储的字符串* char b[6] = "aabcd";  
 char f[10] = "hao";  
 char c[10] = "abcLetabc";  
 String A(a); *//定义为类对象的字符串* String B(b);  
 String C(c);  
 String F(f);  
 cout << "A串：" << A << endl;  
 cout << "B串：" << B << endl;  
 cout<<"A串的长度为： "<<A.Length()<<endl;  
 String D = A(1,3);  
 String E = A += B;  
 cout<<"A的子串为： "<<D<<endl;  
 cout<<"A和B链接后的串为： "<<E<<endl;  
 if(A.Find(F,0) != -1){  
 cout<<"匹配成功 "<<"F在A中第一次出现的位置为： "<<A.Find(F,0); *//k表示从当前字符串第k个位置开始进行匹配* }  
  
 return 0;  
}

【实验过程】



【实验体会】

本次试验要求实现模式匹配，需要提供的主要操作有求字符串的长度，提取子串，判断两个串是否相等、是否不相等，字符串是否为空串，字符串复制，字符串连接，求串的失效函数值，模式匹配等。

6. 二叉树的操作

**【实验简介】**

建立链表结构的二叉树，由类BinaryTree定义。二叉树结点由类BinTreeNode定义。为了降低模板类的使用难度和缩小程序规模，算法没有使用栈和队列的模板类，只是在二叉树类成员中定义了两个数组作为栈和队列来使用（不保证栈和队列功能上的完整性）。

**【实验内容】**

理解二叉树的结构和性质，掌握建立二叉树和遍历二叉树的算法实现，能够用C++程序实现操作。对于输入的n个元素数据，建立一棵二叉树，并提供遍历和输出等操作。

**【程序代码】**

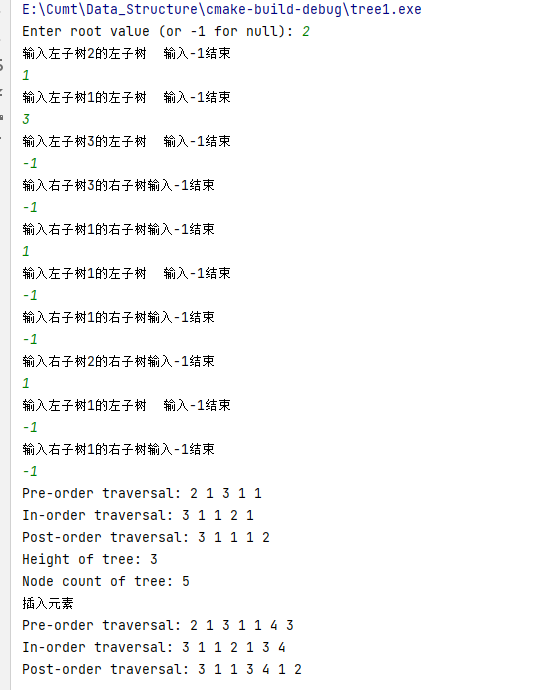
头文件BinaryTree.h

#ifndef **DATA\_STRUCTURE\_BINARYTREE\_H**#define **DATA\_STRUCTURE\_BINARYTREE\_H**#include<iostream>  
#include<iomanip>  
using namespace std;  
  
template<class T>  
struct Node{  
 T data;  
 Node \*left;  
 Node \*right;  
 Node():left(nullptr),right(nullptr){}  
 Node(T val):data(val),left(nullptr),right(nullptr){}  
};  
  
template<class T>  
class BinaryTree{  
private:  
 Node<T> \*root;  
  
  
  
 void destroy(Node<T> \*node)  
 {  
 if(node)  
 {  
 destroy(node->left);  
 destroy(node->right);  
 delete node;  
 }  
 }  
  
 void preOrder(Node<T> \*node) *//前序遍历* {  
 if(node)  
 {  
 cout<<node->data<<" ";  
 preOrder(node->left);  
 preOrder(node->right);  
 }  
 }  
  
 void inOrder(Node<T> \*node) *//中序遍历* {  
 if(node)  
 {  
 inOrder(node->left);  
 cout<<node->data<<" ";  
 inOrder(node->right);  
 }  
 }  
  
 void postOrder(Node<T> \*node) *//后序遍历* {  
 if(node)  
 {  
 postOrder(node->left);  
 postOrder(node->right);  
 cout<<node->data<<" ";  
 }  
 }  
 int getHeight(Node<T> \*node) *//求树的高度* {  
 if(!node)  
 {  
 return 0;  
 }  
 else{  
 int leftHeight = getHeight(node->left);  
 int rightHeight = getHeight(node->right);  
 return max(leftHeight,rightHeight)+1;  
 }  
 }  
  
 int getNodeCount(Node<T> \*node){ *//求节点数量* if(!node){  
 return 0;  
 }  
 else{  
 return getNodeCount(node->left) + getNodeCount(node->right) + 1;  
 }  
 }  
  
 void createBinaryTree(Node<T> \*node) {  
 cout<<"输入左子树"<<node->data<<"的左子树"<<" 输入-1结束"<<endl;  
 T data;  
 cin>>data;  
 if(data != -1)  
 {  
 node->left = new Node<T>(data);  
 createBinaryTree(node->left);  
 }  
  
 cout<<"输入右子树"<<node->data<<"的右子树"<< "输入-1结束"<<endl;  
 cin>>data;  
 if(data != -1)  
 {  
 node->right = new Node<T>(data);  
 createBinaryTree(node->right);  
 }  
 }  
public:  
 BinaryTree():root(nullptr){}  
 ~BinaryTree(){  
 destroy(root);  
 }  
 bool isEmpty() {return (root == **NULL**)? true: false;}  
  
 void Insert(const T& val);  
 void preOrder()  
 {  
 preOrder(root);  
 cout<<endl;  
 }  
 void inOrder()  
 {  
 inOrder(root);  
 cout<<endl;  
 }  
 void postOrder()  
 {  
 postOrder(root);  
 cout<<endl;  
 }  
 int getHeight(){ return getHeight(root);}  
 int getNodeCount() { return getNodeCount(root);}  
 void createBinaryTree() {  
 std::cout << "Enter root value (or -1 for null): ";  
 T data;  
 std::cin >> data;  
 if (data == -1) {  
 root = nullptr;  
 } else {  
 root = new Node<T>(data);  
 createBinaryTree(root);  
 }  
 }  
  
 Node<T> \*getRoot() const{  
 return root;  
 }  
  
  
};  
  
template<class T>  
void BinaryTree<T>::Insert(const T &val) {  
 if(!root)  
 {  
 root = new Node<T>(val);  
 }  
 else{  
 Node<T> \*cur = root;  
 while(1)  
 {  
 if(val<cur->data)  
 {  
 if(!cur->left)  
 {  
 cur->left = new Node<T>(val);  
 break;  
 }  
 cur = cur->left;  
 }  
 else{  
 if(!cur->right)  
 {  
 cur->right = new Node<T>(val);  
 break;  
 }  
 cur = cur->right;  
 }  
 }  
 }  
  
}  
  
  
  
  
#endif *//DATA\_STRUCTURE\_BINARYTREE\_H*

源文件tree.cpp

#include "BinaryTree.h"  
  
  
int main() {  
 BinaryTree<int> tree;  
 tree.createBinaryTree();  
  
 cout << "Pre-order traversal: ";  
 tree.preOrder();  
  
 cout << "In-order traversal: ";  
 tree.inOrder();  
  
 cout << "Post-order traversal: ";  
 tree.postOrder();  
  
 cout << "Height of tree: " << tree.getHeight() << std::endl;  
 cout << "Node count of tree: " << tree.getNodeCount() << std::endl;  
  
 cout<<"插入元素"<<endl;  
 tree.Insert(4);  
 tree.Insert(3);  
  
  
 cout << "Pre-order traversal: ";  
 tree.preOrder();  
  
 cout << "In-order traversal: ";  
 tree.inOrder();  
  
 cout << "Post-order traversal: ";  
 tree.postOrder();  
  
 return 0;  
}

【实验过程】



【实验体会】

本次实验为二叉树的操作，作为上机实验的拓展内容，二叉树本身难度较大，利用递归遍历，需要对数据结构和c++的知识熟练掌握，本次编写过程中遇到的问题也有一些，但最后都顺利解决，上机实验内容全部编写完毕，为下一步的课程设计做准备。