**江苏科技大学经济管理学院《数据结构》实验报告**

**班级： 1722104092 学号：172210409226 姓名**： **林永健**

**实验一：顺序表和单链表**

1. **实验目的：熟练掌握顺序表和单链表的使用**

**二、实验任务及内容：**

**三、实验步骤：**

**1.定义顺序栈的模板类，并在主函数中定义对象，调用成员函数进行相应的测试。**

**【数据结构】**

**1)类的数据成员:需要定义一个模板类型的一维数组T dada[Max]，用来存放表中的元素(要事先定义Max为标识符常量)；还需要一个整型变量len，用来表示当前表中的实际元素个数(即表长)。**

**2)类的成员函数：**

**(1)无参构造函数：**

**功能：创建一个空的顺序表。**

**算法：将表长的变量值设置为0**

(2)有参构造函数：

**功能：建一个顺序表，将形参数组a中的n个元素依次存放在顺序表中，表长设置为形参n的值。**

**算法：**

**如果n>Max，则抛出”参数不当”异常；**

**将n的值赋给len；**

**通过循环，依次将形参数组a的每个元素值依次赋值给data数组的每个元素**

(3)求表长函数

**功能：计算表中实际元素的个数，并返回该值。**

**算法：通过return语句，返回数据成员len的值。**

**(4)判断表空函数**

**功能：判断表是否为空。**

**算法：如果表长len值等于0，则返回true值,否则返回false值。**

**(5)取值函数**

**功能：取表中第i个位置上的元素值。**

**算法：如果表空，抛出“表空异常”；**

**如果位置不当，则抛出“位置不当”异常**

**返回第i个位置上(下标为i-1)的元素值。**

**(6)插入函数**

**功能：在表的第i个位置上插入一个新的元素x。**

**算法：如果表满，抛出“表满异常”；**

**如果位置不当，抛出“位置不当”异常**

**通过循环，从最后一个元素开始，直到第i个元素，依次将每个元素向后移一位**

**将元素x插入到第i个位置上**

**将表长变量len的值增1**

**(7)删除函数**

**功能：将第i个元素删除，并返回被删除的元素值**

**算法：如果表空，则抛出“表空”异常**

**如果位置不当，则抛出“位置不当异常”**

**定义模板T类型的变量x,保存第i个位置上的元素值**

**通过循环，从第i+1位置开始，直到表中的最后一个元素的位置，将每个元素值向前移动一位**

**将表长变量len的值减1**

**返回x的值**

1. **定义主函数，创建对象，分别测试不同的函数调用，验证算法和程序代码的正确性。如果运行结果与分析的结果不一致，需要分析错误的原因，并改正，直到正确为止。**

**四、程序代码**

**#include<iostream.h>**

**const int Max=10;**

**template<class T>**

**class SeqList**

**{**

**private:**

**T date[Max];**

**int lenght;**

**public:**

**SeqList(){lenght=0;}**

**SeqList(T a[],int n);**

**int Len();**

**bool empty();**

**void Get(int i);**

**void Insert(T x,int i);**

**T Delete(int i);**

**void print();**

**};**

**template<class T>**

**SeqList<T>::SeqList(T a[],int n)**

**{**

**if(n>Max) throw"参数不正当";**

**cout<<endl;**

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

**date[i]=a[i];**

**lenght=n;**

**}**

**template<class T>**

**int SeqList<T>::Len()**

**{**

**int count=0;**

**for(int i=0;i<lenght;i++)**

**count++;**

**return count;**

**}**

**template<class T>**

**bool SeqList<T>::empty(){**

**if(lenght==0)**

**return 0;**

**return 1;**

**}**

**template<class T>**

**//取值函数，i代表寻找的位置**

**void SeqList<T>::Get(int i)**

**{**

**if(empty()==0) cout<<"表空";**

**if(i<1||i>lenght) cout<< "位置不正当";**

**else cout<<date[i-1];**

**}**

**template<class T> //插入函数,i代表插入的位置**

**void SeqList<T>::Insert(T x,int i)**

**{**

**if(lenght==Max) cout<<"表满";**

**if(i<1||i>lenght+1) cout<< "插入位置不正确";**

**else**

**{**

**for(int j=lenght-1;j>=i-1;j--)**

**date[j+1]=date[j];**

**date[i-1]=x;**

**lenght++;**

**}**

**}**

**template<class T> //删除元素，将第i个元素删除，并返回删除的函数**

**T SeqList<T>::Delete(int i)**

**{**

**T x;**

**if(empty()==0) throw"表空";**

**cout<<endl;**

**if(i<1||i>=lenght+1) throw"位置不正常";**

**cout<<endl;**

**x=date[i-1];**

**for(int j=i-1;j<=lenght-2;j++)**

**date[j]=date[j+1];**

**lenght--;**

**return x;**

**}**

**template<class T>**

**void SeqList<T>::print()**

**{**

**for(int i=0;i<lenght;i++)**

**cout<<date[i]<<" ";**

**}**

**void main()**

**{**

**int a[]={1,2,3,4,5,6,7};**

**int n=7;**

**SeqList<int>seq(a,n);**

**cout<<"打印出来该顺序表:"<<endl;**

**seq.print();**

**cout<<endl;**

**cout<<"表判空函数"<<seq.empty()<<endl;**

**cout<<"表长为："<<seq.Len()<<endl;**

**cout<<"寻找第三个位置的元素："<<endl;**

**seq.Get(3);**

**cout<<endl;**

**cout<<"寻找第10 个位置的元素"<<endl;**

**seq.Get(10);**

**cout<<endl;**

**cout<<"寻找第0个位置的元素"<<endl;**

**seq.Get(0);**

**cout<<endl;**

**cout<<"在第0个位置插入一个T类型的值为20的元素"<<endl;**

**seq.Insert(20,0);**

**cout<<endl;**

**cout<<"在第10个位置插入一个T类型的值为20的元素"<<endl;**

**seq.Insert(20,0);**

**cout<<endl;**

**cout<<"在第四个位置插入一个T类型的值为10的元素,插入后打印出该表"<<endl;**

**seq.Insert(10,4);**

**seq.print();**

**cout<<endl;**

**cout<<"删除第四个位置的元素,并返回删除元素的值:"<<seq.Delete(4)<<endl;**

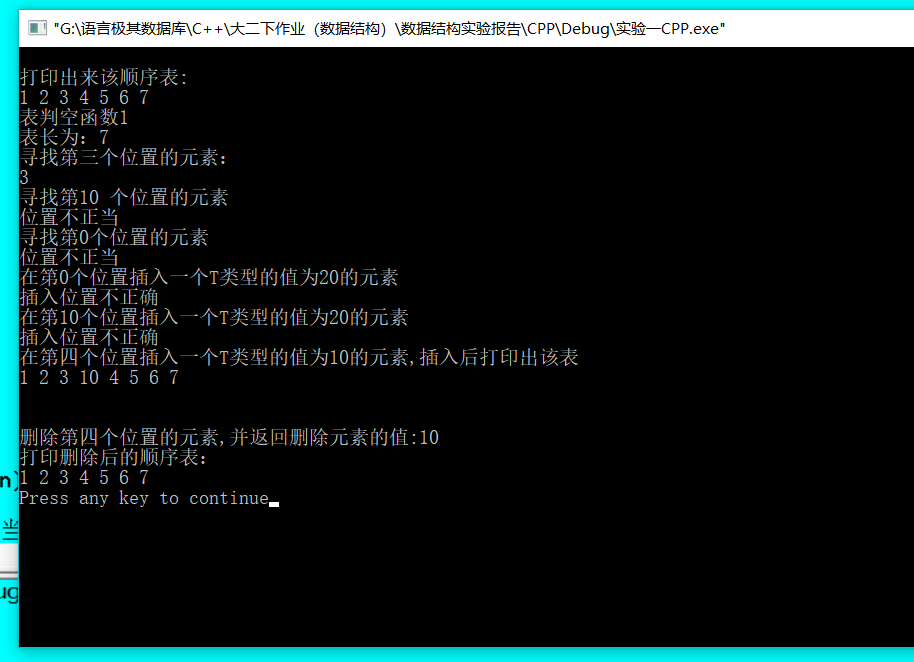
**cout<<"打印删除后的顺序表："<<endl;**

**seq.print();**

**cout<<endl;**

**}**

**五、运行结果**



**六、存在问题及解决方法**

**存在犯低级错误，少打和漏打变量的类型，代码存在就是少量的低级错误**

**七、实验体会**

**通过本次实验，让我更加的体会到了数据结构中链表的强大以及模板类的使用方法，让我对数据结构中的顺序表的相关操作以及语法有了进一步的认识。**

**//模板类单链表的使用**

**一、实验目的：熟练掌握数据结构中单链表的使用**

**二、实验任务及内容：单链表的创建以及单链表中一些功能的实现**

**三、实验步骤：**

**定义一个结构体，数据域用于存放节点的数据。**

**定义一个模板类型的类的LinkList**

1. **类的数据成员：**

**结构体类型的head，以及一些备用的模板类的变量p，s，t，u；**

**表的长度n**

1. **类的成员函数：**
2. **构造函数**

**功能：构造一条带头结点的单链表**

**算法：**

**初始化链表的长度；**

**利用循环以及头插法构建单链表。**

1. **析构函数：**

**功能：逐步释放各个节点**

**算法：**

**将头节点所在的地址赋值给p**

**利用循环，当p不为空的时候，利用一个u值将p的地址赋值给u**

**p=p->next;**

**释放u，上诉，直至p为空。**

1. **求表长度的函数：**

**功能：求链表的长度**

**将head->next的地址赋值给p**

**设置一个计数器初始值为1**

**利用循环逐步遍历链表的各个结点，直至p为空，每次遍历的是啥计数器自增。**

1. **查找节点值的函数**

**功能：查找第i个位置的元素**

**算法：**

**判断查找的位置是否正确**

**将head->next赋值给p**

**定义一个j=1**

**定义一个T类型的num；**

**利用循环，num=p->data;**

**p=p->next;**

**j++;**

**直到j=i；**

**返回num的值**

1. **插入函数**

**功能：插入一个模板类型的节点在链表中**

**算法：**

**将head->next赋值给p**

**设置一个计数器count初始化为1**

**利用循环条件为p不为并且count<i-1的时候**

**逐步遍历节点，直至p为空或者找到插入位置的前一个节点的位置。**

**P的下一个节点指向新的节点s**

**将p下一个节点的地址赋值给s下一个节点的地址**

**完成插入**

**表长加1**

1. **删除函数**

**功能：删除链表上的一个节点**

**算法：**

**将head->next的地址赋给p 定义一个整形的j初始化为1**

**判断删除节点的位置是否正确**

**利用循环找到删除节点的前驱节点**

**将p->next的地址赋值给t**

**将t->next的地址赋值p->next的**

**释放t**

**表长减一**

1. **查找位置的函数**

**功能：查找某个值的的节点的位置**

**算法：**

**将head->next赋给p**

**定义一个int类型的初始化为1**

**利用while循环去遍历每次i++**

**当p为空或者p->data==x的时候跳出循环**

**p=p‑>next;**

**p为空的时候 找不到元素**

**p 不为空的时候返回i**

**（8）打印函数**

**功能：打印每部操作后的链表**

**算法：利用循环去打印。**

**四、程序代码**

**//单链表的的链表模式**

**#include<iostream>**

**using namespace std;**

**//结构体**

**template<class T>**

**struct STU**

**{**

**T data;**

**STU<T>\*next;**

**};**

**template<class T>**

**class LinkList**

**{**

**private:**

**STU<T>\*head, \*p, \*s,\*t,\*u;**

**int n;**

**public:**

**LinkList(T a[], int n); //将数组里面的元素初始化data**

**~LinkList(); //析构函数。**

**int Length(); //求该链表的长度**

**T Get(int i); //查找第 i个节点的位置的元素的值**

**int Locate(T x); //查找元素x的所在的位置**

**void Insert(int i, T y); //在i的位置插入某个元素**

**void Delete(int i); //删除元素，删除第几个位置的元素**

**void PrintList(); //打印该链表；**

**};**

**//头插法建立链表**

**template<class T>**

**LinkList<T>::LinkList(T a[], int n)**

**{**

**this->n = n;**

**head = new STU<T>;**

**head->next=NULL;**

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

**{**

**s = new STU<T>;**

**s->data = a[i];**

**s->next=head->next;**

**head->next=s;**

**}**

**}**

**//求链表的长度（不包括头结点的求长度）**

**template <class T>**

**int LinkList<T>::Length()**

**{**

**p = head->next;**

**int count = 1;**

**while (p->next!=NULL)**

**{**

**p = p->next;**

**count++;**

**}**

**return count;**

**}**

**//查找第i个位置的元素的值**

**template<class T>**

**T LinkList<T>::Get(int i)**

**{**

**if (i<1 || i>n) throw " 位置异常";**

**int j = 1;**

**p = head->next;**

**T num;**

**while (j!= i)**

**{**

**num = p->data;**

**p = p->next;**

**j++;**

**}**

**return num;**

**}**

**//查找元素X的位置**

**template<class T>**

**int LinkList<T>::Locate(T x)**

**{**

**p = head->next;**

**int i = 1;**

**while(p!=NULL&&p->data!=x)**

**{**

**p=p->next;**

**i++;**

**}**

**if(!p) throw"找不到该元素";**

**return i;**

**}**

**//在i的位置插入一个值y**

**template<class T>**

**void LinkList<T>::Insert(int i, T y)**

**{**

**p = head->next;**

**int count = 1;**

**while (p!=NULL&&count<i-1) //找到i这个节点**

**{**

**p = p->next;**

**count++;**

**}**

**if (p == NULL) throw "找不到插入的位置"; //找不到i这个点**

**else**

**{**

**s = new STU<T>;**

**s->data = y;**

**s->next = p->next;**

**p->next = s;**

**n++;**

**}**

**}**

**//删除某个位置的元素**

**template<class T>**

**void LinkList<T>::Delete(int i)**

**{**

**p = head->next;**

**int j = 1;**

**if(i<1||i>n) throw"删除异常";**

**while (j<i-1) //找到前一个节点**

**{**

**p = p->next;**

**j++;**

**}**

**t=p->next;**

**p->next=t->next;**

**delete t;**

**n--;**

**}**

**//打印这张表**

**template<class T>**

**void LinkList<T>::PrintList()**

**{**

**p = head->next;**

**cout << "打印的表如下" << endl;**

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

**{**

**cout << p->data << '\t';**

**p = p->next;**

**}**

**}**

**template <class T>**

**LinkList<T>::~LinkList()**

**{**

**p=head->next;**

**while(p!=NULL)**

**{**

**u=p;**

**p=p->next;**

**delete u;**

**}**

**cout<<"释放完毕"<<endl;**

**}**

**//主函数**

**void main()**

**{**

**int n = 5;**

**int test[5] = { 1, 2, 3, 4, 5 };**

**LinkList<int>List(test, n);**

**List.PrintList();**

**cout << endl;**

**cout << "表的长度" << endl;**

**cout << List.Length();**

**cout << endl;**

**cout << "找到第三个位置的元素的值" << endl;**

**cout << List.Get(3);**

**cout << endl;**

**cout << "查找元素值为4的位置" << endl;**

**cout << List.Locate(4);**

**cout << endl;**

**cout << "插入一个元素 插入2到第二个位置";**

**List.Insert(2, 2);**

**cout << endl;**

**cout << "再次打印这List" << endl;**

**List.PrintList();**

**cout << endl;**

**cout << "删除第二个位置的元素" << endl;**

**List.Delete(2);**

**cout << endl;**

**cout << "再次打印这张List" << endl;**

**List.PrintList();**

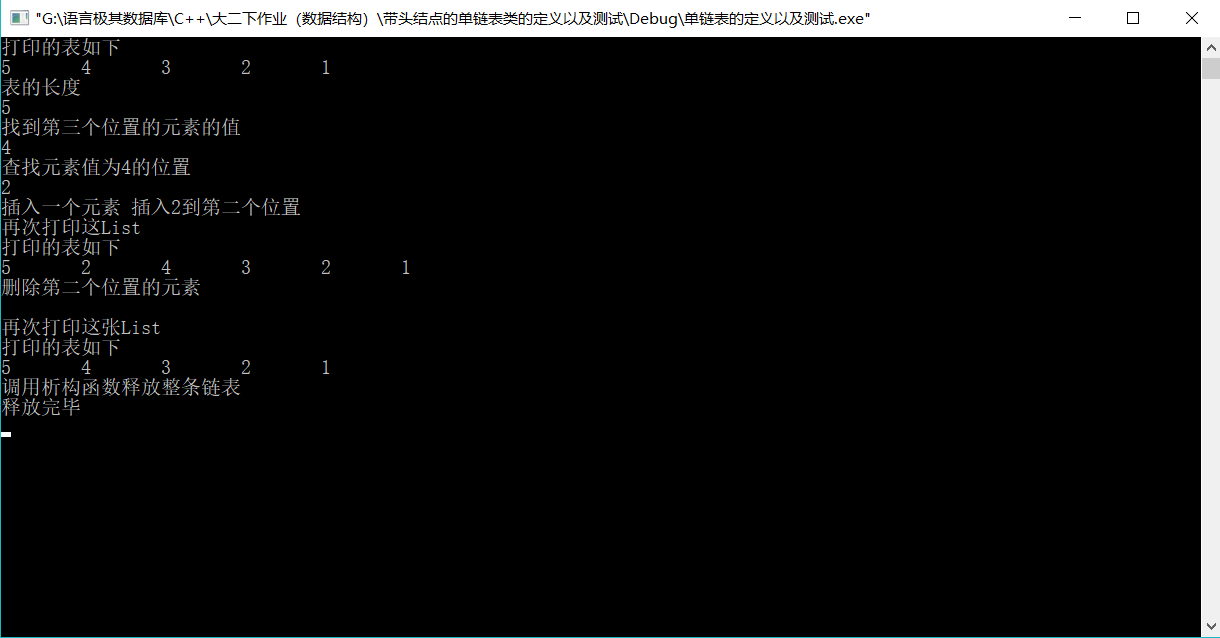
**cout<<endl;**

**cout<<"调用析构函数释放整条链表"<<endl;**

**List.~LinkList();**

**}**

**五、运行结果**



**六、存在问题及解决方法**

**不存在啥问题。**

**七、实验体会**

**可以说，我就我个人来看，本次实验对于所有的编程者都是非常的有用的，本次实验基本就是一个常用的链接存储的存储结构，对于以后的学习也是非常有用的。**

**实验二：栈和队列**

**一、实验目的：熟练掌握数据结构中链栈和链队列的使用**

**二、实验任务及内容：定义链栈和链队的模板类，并在主函数中定义对象，调用成员函数进行相应的测试。**

**三、实验步骤：**

**链栈：**

**1)类的数据成员:需要先定义一个结构体，有一个指针域用来记录头指针的位置，以及数据域，记录栈元素。**

**2)类的成员函数：**

**结构体类型的top指针，始终指向栈顶；**

(1)有参构造函数：

**功能：建立一个空链栈，将头节点的指针域赋值为空**

**算法：**

**Top=NuLL**

(2)入栈操作

**功能：将元素X入栈**

**算法：开辟一个新的节点S.**

**并将X赋值给S—>data;**

**将S的地址赋值给top**

**(3)出栈操作函数**

**功能：返回栈顶元素的值，并删除栈顶所在的节点**

**算法：判断链栈是否为空**

**定义一个模板类型的X，以及模板类型的指针p；**

**将top->date赋值给该变量 ；**

**将top赋值的地址赋值给P**

**将下一个节点的地址赋值给top**

**释放p，并返回X的值**

**(4)取栈顶元素函数**

**功能：取出链栈中，栈顶的元素的值**

**算法：如果栈空，抛出“下溢”；**

**返回top指向的元素的值；**

**(5)判空函数**

**功能：判断该链栈是否为空栈**

**算法：如果top=NULL，**

**返回0；**

**否则返回1；**

**（6）析构函数：**

**功能：将所有的栈元素出栈**

**算法：定义模板类的结构体类型的指针P**

**将栈顶top的地址赋值给P**

**释放top=top—>next，并释放P**

**循环操作上面两部，直至top==NULL**

**（7）定义主函数，创建对象，分别测试不同的函数调用，验证算法和程序代码的正确性。如果运行结果与分析的结果不一致，需要分析错误的原因，并改正，直到正确为止。**

**四、程序代码**

**#include<iostream.h>**

**template<class T>**

**struct Node**

**{**

**T date;**

**Node<T>\*next;**

**};**

**template<class T>**

**class LinkStack**

**{**

**private:**

**Node<T>\*top,\*s,\*t,\*h,\*p;**

**public:**

**LinkStack(){top=NULL;}**

**void Push(T x); //入栈元素**

**T Pop();**

**T GetTop();**

**int Empty();**

**~LinkStack();**

**void print();**

**};**

**template<class T>**

**void LinkStack<T>::Push(T x)**

**{**

**s=new Node<T>;**

**s->date=x;**

**s->next=top;**

**top=s;**

**}**

**template<class T>**

**T LinkStack<T>::Pop()**

**{**

**if(top==NULL) return 0;**

**T x;**

**p=new Node<T>;**

**p=top;**

**x=p->date;**

**top=top->next;**

**delete p;**

**return x;**

**}**

**template<class T>**

**T LinkStack<T>::GetTop(){**

**return top->date;**

**}**

**template<class T>**

**int LinkStack<T>::Empty()**

**{**

**if(top==NULL) return 0;**

**return 1;**

**}**

**template<class T>**

**LinkStack<T>::~LinkStack()**

**{**

**t=new Node<T>;**

**while(top!=NULL)**

**{**

**t=top;**

**top=top->next;**

**delete t;**

**}**

**}**

**template<class T>**

**void LinkStack<T>::print()**

**{**

**h=new Node<T>;**

**h=top;**

**while(h!=NULL)**

**{**

**cout<<h->date<<" ";**

**h=h->next;**

**}**

**}**

**void main()**

**{**

**LinkStack<int>stack;**

**stack.Push(10);**

**stack.Push(20);**

**stack.Push(30);**

**stack.Push(40);**

**cout<<"判断是否栈空"<<endl;**

**cout<<stack.Empty()<<endl;**

**cout<<"入栈后的元素如下："<<endl;**

**stack.print();**

**cout<<endl;**

**cout<<"得到栈顶元素"<<endl;**

**cout<<stack.GetTop()<<endl;**

**cout<<"出栈操作,并打印出栈后各节点元素的值"<<endl;**

**stack.Pop();**

**stack.print();**

**cout<<endl;**

**cout<<"调用析构函数释放改链栈,并判空"<<endl;**

**stack.~LinkStack();**

**cout<<stack.Empty()<<endl;}**

**五、运行结果**



**六、存在问题及解决方法**

**少些和漏写代码，以至于出现低级错误。**

**七、实验体会**

**通过本次实验，让我更接触到了，之前一直好奇的栈队以及他们使用的方法，通过本次实验，我对栈和队有了清晰的认知。而且能在之后的的编程路上希望能用得上栈队的编程思想。**

**链队列：**

**需要先定义一个结构体，并且里面有next指针，以及数据域，记录队元素。**

1. **类的数据成员:**

**结构体类型的队头指针front以及队尾指针rear，以及一个哨所指针s**

**2)类的成员函数：**

(1)有参构造函数：

**功能：建立一个空链队，将队头指针和队尾指针域赋值相等并且将头结点的地址赋值给队尾和对头指针**

**算法：**

**开辟一个新的结构体类型的指针，s->next=NULL，并将s的地址赋值给队头指针和队尾指针。**

(2)入队操作

**功能：将元素X入队**

**算法：开辟一个新的节点S.**

**并将X赋值给S—>data;**

**将S的地址赋值给rear**

**(3)出队操作函数**

**功能：将入队后的元素出队**

**算法：暂存队头元素**

**将队头元素所在的结点摘链**

**(4)取队头元素元素函数**

**功能：取出队列中，队头元素的值**

**算法：如果队空，抛出“下溢”；**

**返回front->next,所在的节点的数据；**

**释放节点front->next;**

**(5)判空函数**

**功能：判断该链队是否为空队列**

**算法：如果rear=front，**

**返回0；**

**否则返回1；**

**（6）定义主函数，创建对象，分别测试不同的函数调用，验证算法和程序代码的正确性。如果运行结果与分析的结果不一致，需要分析错误的原因，并改正，直到正确为止。**

**四、程序代码**

**/\*#include<iostream.h>**

**template<class T>**

**struct Node**

**{**

**T date;**

**Node<T>\*next;**

**};**

**template<class T>**

**class LinkStack**

**{**

**private:**

**Node<T>\*top,\*s,\*t,\*h,\*p;**

**public:**

**LinkStack(){top=NULL;}**

**void Push(T x); //入栈元素**

**T Pop();**

**T GetTop();**

**int Empty();**

**~LinkStack();**

**void print();**

**};**

**template<class T>**

**void LinkStack<T>::Push(T x)**

**{**

**s=new Node<T>;**

**s->date=x;**

**s->next=top;**

**top=s;**

**}**

**template<class T>**

**T LinkStack<T>::Pop()**

**{**

**if(top==NULL) return 0;**

**T x;**

**p=new Node<T>;**

**p=top;**

**x=p->date;**

**top=top->next;**

**delete p;**

**return x;**

**}**

**template<class T>**

**T LinkStack<T>::GetTop(){**

**return top->date;**

**}**

**template<class T>**

**int LinkStack<T>::Empty()**

**{**

**if(top==NULL) return 0;**

**return 1;**

**}**

**template<class T>**

**LinkStack<T>::~LinkStack()**

**{**

**t=new Node<T>;**

**while(top!=NULL)**

**{**

**t=top;**

**top=top->next;**

**delete t;**

**}**

**}**

**template<class T>**

**void LinkStack<T>::print()**

**{**

**h=new Node<T>;**

**h=top;**

**while(h!=NULL)**

**{**

**cout<<h->date<<" ";**

**h=h->next;**

**}**

**}**

**void main()**

**{**

**LinkStack<int>stack;**

**stack.Push(10);**

**stack.Push(20);**

**stack.Push(30);**

**stack.Push(40);**

**cout<<"判断是否栈空"<<endl;**

**cout<<stack.Empty()<<endl;**

**cout<<"入栈后的元素如下："<<endl;**

**stack.print();**

**cout<<endl;**

**cout<<"得到栈顶元素"<<endl;**

**cout<<stack.GetTop()<<endl;**

**cout<<"出栈操作,并打印出栈后各节点元素的值"<<endl;**

**stack.Pop();**

**stack.print();**

**cout<<endl;**

**cout<<"调用析构函数释放改链栈,并判空"<<endl;**

**stack.~LinkStack();**

**cout<<stack.Empty()<<endl;**

**}\*/**

**#include<iostream.h>**

**template<class T>**

**struct Node**

**{**

**T date;**

**Node<T> \*next;**

**};**

**template<class T>**

**class LinkQueue**

**{**

**private:**

**Node<T>\*front,\*rear,\*s;**

**public:**

**LinkQueue();**

**void EnQueue(T x);**

**T DeQueue();**

**T GetQueue();**

**int Empty();**

**void print();**

**};**

**template<class T>**

**LinkQueue<T>::LinkQueue()**

**{**

**s=new Node<T>;**

**s->next=NULL;**

**front=rear=s;**

**}**

**template<class T>**

**void LinkQueue<T>::EnQueue(T x)**

**{**

**s=new Node<T>;**

**s->date=x;**

**s->next=NULL;**

**rear->next=s;**

**rear=s;**

**}**

**template<class T>**

**T LinkQueue<T>::DeQueue()**

**{**

**T x;**

**s=new Node<T>;**

**if(front==rear) return 0;**

**s=front->next;**

**x=s->date;**

**front->next=s->next;**

**delete s;**

**return x;**

**}**

**template<class T>**

**T LinkQueue<T>::GetQueue()**

**{**

**s=new Node<T>;**

**T x;**

**s=front->next;**

**x=s->date;**

**return x;**

**}**

**template<class T>**

**int LinkQueue<T>::Empty()**

**{**

**if(front==rear)**

**return 0;**

**return 1;**

**}**

**template<class T>**

**void LinkQueue<T>::print()**

**{**

**if(front==rear) throw" 链队空";**

**s=new Node<T>;**

**s=front->next;**

**while(s!=NULL)**

**{**

**cout<<s->date<<" ";**

**s=s->next;**

**}**

**}**

**void main()**

**{**

**LinkQueue<int>queue;**

**queue.EnQueue(10);**

**queue.EnQueue(20);**

**queue.EnQueue(30);**

**queue.EnQueue(40);**

**cout<<"将10,20,30,40入队，并打印入队后的元素如下："<<endl;**

**queue.print();**

**cout<<endl;**

**cout<<"取队头元素："<<queue.GetQueue()<<endl;**

**cout<<"出队操作，出了最先入队的元素并返回该元素"<<queue.DeQueue()<<endl;**

**cout<<"打印出队后的队列的元素的值:"<<endl;**

**queue.print();**

**cout<<endl;**

**cout<<"判断队列是否为空:"<<queue.Empty()<<endl;**

**}**

**五、运行结果截图：**



**六、存在问题及解决方法**

**少些代码，以至于出现遗漏性的错误**

**七、实验体会**

**通过本次实验，让我更接触到了，之前一直好奇的栈队以及他们使用的方法，通过本次实验，我对栈和队有了清晰的认知。而且能在之后的的编程路上希望能用得上栈队的编程思想。**

**实验三：稀疏矩阵的压缩存储**

**一、实验目的：熟练掌握稀疏矩阵的压缩存储**

**二、实验任务及内容：稀疏矩阵的压缩存储以及转置矩阵的应用**

**三、实验步骤：**

**需要先定义一个结构体，并且存放数据域矩阵元素的所在的行数和列数，以及模板类型的矩阵的item。以及定义一个const int Max=10；**

1. **类的数据成员:**

**模板类结构体的数组date[Max];**

**稀疏矩阵的行数 mt 以及列数nt 以及非零元素的个数rt**

**2)类的成员函数：**

(1)有参构造函数：

**功能：初始化mt,nt,rt,并输入非零元素的值进入压缩存储的矩阵**

**算法：**

**初始化 mt，nt，rt；**

**利用for循环逐步的输入非零元素所在的行数,。**

**非零元素所在的列数**

**以及非零元素的值**

(2)无参构造函数操作

**功能：用作转置矩阵的备用**

**算法：将类的数据成员mt，nt，rt全部初始化为0**

**(3)打印输出函数函数**

**功能：将压缩存储后的矩阵所在的行数，列数，以及值以三元组的形式打印输出**

**算法：利用for循环逐步的输出该压缩后的矩阵的行数以及列数，以及非零元素的值**

**(4)转置矩阵函数**

**功能：将矩阵进行转置处理**

**算法：引用已经压缩存入好的矩阵所形成的一个模板类型的对象SpareMatrix<T> &jz**

**将引用的矩阵的行数无参构造函数里面的列数nt=jz.mt**

**将引用的矩阵的列数数无参构造函数里面的行数 mt=jz.nt**

**定义一个变量K**

**利用第一个for循环遍历转置后的矩阵的列数，只要列数小于nt**

**再利用嵌套for循环遍历转置之前的非零元素**

**如果转置之后的矩阵的列数等于转置之前矩阵的元素所在的列数则进行如下交换：**

**date[k].row=jz.date[i].col;**

**date[k].col=jz.date[i].row;**

**date[k].item=date[i].item;**

**k++ ;**

**（6）定义主函数，创建对象，分别测试不同的函数调用，验证算法和程序代码的正确性。如果运行结果与分析的结果不一致，需要分析错误的原因，并改正，直到正确为止。**

**四、程序代码**

**#include<iostream.h>**

**const int Max=10;**

**template<typename T>**

**struct element**

**{**

**T item;**

**int row,col;**

**};**

**template<class T>**

**class SpareMatrix**

**{**

**private:**

**element<T> data[Max];**

**int mt,nt,rt; //稀疏矩阵的行数，列数，以及非0元素的个数**

**public:**

**SpareMatrix(int mt,int nt,int rt)**

**{**

**this->mt=mt;**

**this->nt=nt;**

**this->rt=rt;**

**cout<<"以行为主，请依次输入非0元素的行以及列，以及非零元素的值："<<endl;**

**for(int i=0;i<rt;i++)**

**cin>>data[i].row>>data[i].col>>data[i].item;**

**}**

**SpareMatrix(){mt=nt=rt=0;} //定义一个参数值为空的构造函数用于转置矩阵的用途**

**void print()**

**{**

**cout<<"依次输出的如下："<<endl;**

**cout<<"row"<<"\t"<<"col"<<"\t"<<"item"<<endl;**

**for(int i=0;i<rt;i++)**

**cout<<data[i].row<<"\t"<<data[i].col<<"\t"<<data[i].item<<endl;**

**}**

**//转置矩阵的定义**

**void jzzz(SpareMatrix<T> &jz)**

**{**

**nt=jz.mt;**

**mt=jz.nt;**

**rt=jz.rt;**

**int k=0;**

**for(int Col=0;Col<nt;Col++)**

**{**

**for(int i=0;i<rt;i++)**

**if(jz.data[i].col==Col)**

**{**

**data[k].row=jz.data[i].col;**

**data[k].col=jz.data[i].row;**

**data[k].item=jz.data[i].item;**

**k++;**

**}**

**}**

**}**

**};**

**void main()**

**{**

**int mt=4;**

**int nt=4;**

**int rt=4;**

**SpareMatrix<int>SM(mt,nt,rt);**

**SM.print();**

**SpareMatrix<int>SA;**

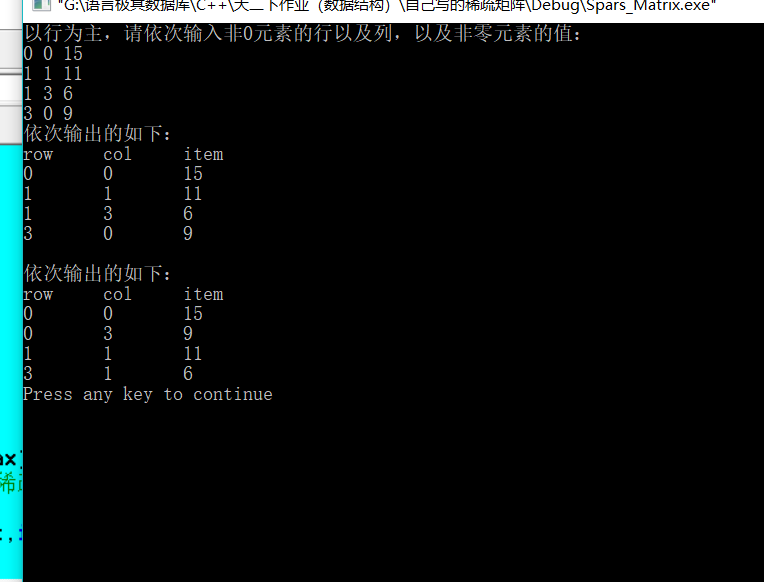
**SA.jzzz(SM);**

**cout<<endl;**

**SA.print();**

**}**

**五、运行结果**



**六、存在问题及解决方法**

**对转置矩阵方面不够熟悉，通过画图列出来 最终解决。**

**七、实验体会**

**通过过本次实验，我更加的深刻的意识到了，数据结构中稀疏矩阵的压缩存储方式，矩阵的用法，虽然在线代中也有提现，本次的数据结构是将工程线性代数的完美使用。**

**实验四：二叉树、二叉树的链表存储**

**一、实验目的：熟练掌握数据结构中二叉树，以及二叉树的链表存储使用**

**二、实验任务及内容：**

**初始化一颗二叉树，完成对应的相关操作。**

1. **实验步骤：**

**先定义模板类型的结构体，结构体中有存放该模板类型的数据域date**

**，以及存放左右孩子的地址得 指针域。**

**1）类的数据成员：**

**定义一个模板类型结构体的指针root，指向树的根节点。**

1. **类的成员函数：**
2. **模板类型结构体的函数**

**功能：用来给主函数中引用跟节点所在的地址**

**算法：定义一个模板类型的的结构体的函数，返回root所在的地址**

（2）有参构造函数：

**功能：初始化构建一颗完全二叉树**

**算法：通过增加虚结点变成完全二叉树**

**按完全二叉树层序遍历的序列**

**依次输入每个结点名称，虚结点用'@'表示**

**结束用'#'表示,建立二叉链表\n**

**首先定义一个模板类型结构的的 指针数组Q[Max]，用于存放各个入队顶点的地址**

**定义两个整形变量 front=1，rear=0；**

**输入，如果一开始输入#，则root=NULL，并输出树为空树。**

**先定义一个模板类型结构体的指针变量s并初始化**

**并在s开辟一个新的结构体将该输入的值赋值给s->date**

**并将左右s指向的左右孩子的地址都置空为NULL**

**逐步将s入队，指针rear++**

**判断rear是否等于1，等于就root=s**

**否则判断rear%2==0，等于零就Q[front]->lchild=s**

**否则Q[front]->rchild=s;**

**判断rear%2是否等于1，等于1，front++地址往后移**

**继续输入顶点的值**

**重复上述操作**

（3）释放函数：

**功能：释放二叉链表中的节点**

**算法：判断跟节点是否为空**

**利用递归函数先释放左子树**

**利用递归函数释放右子树**

**释放根节点**

（4）析构函数：

**功能：释放二叉链表中的节点**

**算法：调用释放函数，并输出释放完毕。**

（5）前序遍历函数：

**功能：前序遍历二叉链表中的节点的元素的值**

**算法：判断根节点是否为空。**

**先输出根节点的元素的值**

**递归遍历左子树。**

**递归遍历右子树**

（6）中序遍历函数：

**功能： 遍历二叉链表的节点的元素**

**算法： 递归遍历左子树**

**访问根节点**

**递归遍历右子树**

（6）后序遍历函数：

**功能： 后序遍历二叉链表的节点的元素**

**算法： 递归遍历左子树**

**递归遍历右子树**

**访问根节点的元素**

（7）层序遍历函数：

**功能： 层序遍历二叉链表的节点的元素**

**算法： 先定义一个模板类型结构体的指针的数组**

**将-1赋值给头指针和尾指针**

**判断根节点是否为空，不为空则讲跟节点入队**

**当front！=rear的时候循环如下操作**

**定义一个结构体类型的指针变量s并将出队的Q[++front]的地址赋值给s，输出s->date;**

**判断s->lchild是否为空，不为空则将s->lchild入队**

**判断s->rchild是否为空，不为空则将s->rchild入队**

（8）求树的高度的函数：

**功能： 求该二叉树的高度**

**算法： 先判断该二叉树是否为空，为空返回0**

**再判断该二叉树的左右孩子是否为空，为空则返回1**

**否则定义一个变量h1**

**将递归该函数并且参数为左孩子地址的结果赋值给h1**

**定义一个变量h2**

**将队规函数并且参数为右孩子地址的结果赋值给h2**

**判断h1>=h2?h1+1:h2+1**

（9）求树的叶子节点的个数：

**功能： 求二叉树叶子节点的个数**

**算法： 先判断该二叉树是否为空，为空返回0**

**再判断该二叉树的左右孩子是否为空，为空则返回1**

**否则返回两个递归函数相加的结果参数分别为左孩子的地址，**

**右孩子的地址。**

（10）求树的所有叶子节点的个数：

**功能： 求二叉树叶子节点的个数**

**算法： 先判断该二叉树是否为空，为空返回0**

**用递归函数求根节点左边节点的个数**

**用递归函数再求右边节点的个数**

**返回左右节点相加并加1**

**（11）定义主函数，创建对象，分别测试不同的函数调用，验证算法和程序代码的正确性。如果运行结果与分析的结果不一致，需要分析错误的原因，并改正，直到正确为止。**

**四、程序代码**

**#include<iostream.h>**

**const int Max=32;**

**template<typename T>**

**struct biNode{**

**T data;**

**biNode<T> \*lChild,\*rChild;**

**};**

**template<typename T>**

**class biTree{**

**biNode<T> \*root;**

**public:**

**biNode<T> \*getRoot(){ return root;}**

**biTree();//构造函数，初始化二叉链表**

**~biTree();//析构函数，释放二叉链表中的结点**

**void release(biNode<T> \*rt);**

**//释放以rt所指向的结点为根结点的二叉树**

**void preOrder(biNode<T> \*rt);**

**//对以rt所指向的结点为根结点的二叉树进行前序遍历**

**void inOrder(biNode<T> \*rt);**

**//对以rt所指向的结点为根结点的二叉树进行中序遍历**

**void postOrder(biNode<T> \*rt);**

**//对以rt所指向的结点为根结点的二叉树进行后序遍历**

**void leverOrder(); //层序遍历**

**int BiHight(biNode<T> \*rt); //求树的高度的函数**

**int LeafNumber(biNode<T> \*rt); //求叶节点的个数的函数**

**int AllNode(biNode<T> \*rt); //求所有节点个数的函数**

**};**

**//前序遍历**

**template<typename T>**

**void biTree<T>::preOrder(biNode<T> \*rt){**

**if(rt){**

**cout<<rt->data;//访问根结点**

**preOrder(rt->lChild);//先序遍历左子树**

**preOrder(rt->rChild);//先序遍历右子树**

**}**

**//else return;**

**}**

**//中序遍历**

**template<typename T>**

**void biTree<T>::inOrder(biNode<T> \*rt){**

**if(rt){**

**inOrder(rt->lChild);//中序遍历左子树**

**cout<<rt->data;//访问根结点**

**inOrder(rt->rChild);//中序遍历右子树**

**}**

**//else return;**

**}**

**//后序遍历**

**template<typename T>**

**void biTree<T>::postOrder(biNode<T> \*rt){**

**if(rt){**

**postOrder(rt->lChild);//后序遍历左子树**

**postOrder(rt->rChild);//后序遍历右子树**

**cout<<rt->data;//访问根结点**

**}**

**//else return;**

**}**

**template<typename T>**

**void biTree<T>::release(biNode<T> \*rt){**

**if(rt){**

**release(rt->lChild);//释放左子树**

**release(rt->rChild);//释放右子树**

**delete rt;//释放根结点**

**}**

**}**

**template<typename T>**

**biTree<T>::~biTree(){**

**release(root);**

**cout<<"释放完毕"<<endl;**

**}**

**//构造数组**

**template<typename T>**

**biTree<T>::biTree(){**

**cout<<"通过增加虚结点变成完全二叉树，";**

**cout<<"按完全二叉树层序遍历的序列，";**

**cout<<"依次输入每个结点名称，虚结点用'@'表示，";**

**cout<<"结束用'#'表示,建立二叉链表\n";**

**biNode<T> \*Q[Max];**

**int front=1,rear=0;**

**//front表示当前结点的双亲结点在队列中的下标**

**//rear表示当前产生结点的层序编号**

**char ch;//存放从键盘输入的字符值**

**cin>>ch;**

**if(ch=='#'){ biNode<T> \*root=NULL;cout<<"该树为空树，无法进行以下操作"<<endl; return;}**

**while(ch!='#'){**

**biNode<T> \*s=NULL;**

**if(ch!='@'){**

**s=new biNode<T>;**

**s->data=ch;**

**s->rChild=s->lChild=NULL;**

**}**

**//将s入队**

**Q[++rear]=s;**

**if(rear==1) root=s;**

**else{**

**if(s)**

**if(rear%2==0)**

**Q[front]->lChild=s;**

**else**

**Q[front]->rChild=s;**

**if(rear%2==1) front++;**

**}**

**cin>>ch;**

**}**

**}**

**//层序遍历 //画图理解**

**template<typename T>**

**void biTree<T>::leverOrder(){**

**biNode<T> \*Q[Max];**

**int front=-1,rear=-1;**

**if(root) Q[++rear]=root;**

**while(front!=rear){**

**biNode<T> \*s=Q[++front];**

**cout<<s->data;**

**if(s->lChild) Q[++rear]=s->lChild;**

**if(s->rChild) Q[++rear]=s->rChild;**

**}**

**cout<<endl;**

**}**

**//求树的高度**

**template<typename T>**

**int biTree<T>::BiHight(biNode<T> \*rt)**

**{**

**if(!rt) return 0;**

**if(!(rt->lChild)&&!(rt->rChild)) return 1;**

**else**

**{**

**int h1=BiHight(rt->lChild);**

**int h2=BiHight(rt->rChild);**

**return h1>=h2?h1+1:h2+1;**

**}**

**}**

**//求树的叶子节点**

**template<typename T>**

**int biTree<T>::LeafNumber(biNode<T> \*rt)**

**{**

**if(!rt) return 0;**

**if(!(rt->lChild)&&!(rt->rChild)) return 1;**

**else**

**{**

**return LeafNumber(rt->lChild)+LeafNumber(rt->rChild);**

**}**

**}**

**//求树的所有节点的个数, 记录返回0的个数**

**template<typename T>**

**int biTree<T>::AllNode(biNode<T> \*rt)**

**{**

**if(!rt) return 0;**

**int leaf=AllNode(rt->lChild);**

**int right=AllNode(rt->rChild);**

**return leaf+right+1;**

**}**

**void main(){**

**cout<<"请输入一颗树，按照层序输入从左到右的输入方法"<<endl;**

**biTree<char> bt;**

**cout<<"前序遍历序列：";**

**bt.preOrder(bt.getRoot());**

**cout<<endl;**

**cout<<"中序遍历序列：";**

**bt.inOrder(bt.getRoot());**

**cout<<endl;**

**cout<<"后序遍历序列：";**

**bt.postOrder(bt.getRoot());**

**cout<<endl;**

**cout<<"层序遍历序列：";**

**bt.leverOrder();**

**cout<<endl;**

**cout<<"树的高度：";**

**cout<<bt.BiHight(bt.getRoot())<<endl;**

**cout<<"树的叶子节点数：";**

**cout<<bt.LeafNumber(bt.getRoot())<<endl;**

**cout<<"所有节点的个数：";**

**cout<<bt.AllNode(bt.getRoot())<<endl;**

**cout<<"释放该二叉树：";**

**bt.~biTree();**

**}**

**五、运行结果**



**六、存在问题及解决方法**

**对二叉树的构造函数不够熟悉**

**七、实验体会**

**通过本次二叉树、以及二叉树的链表的存储结构的使用，让我对树有了进一步的认识，之前对树的认识一直停留在学习过的一些Javascript的算法思想中，学完了，c++版的数据结构让我有了更明确的认识。**

**实验五：图、（无向图，领接矩阵和邻接表的两种不同结构）**

**//邻接矩阵方面**

**一、实验目的：熟练掌握数据结构中图，无向图以及相关的知识的使用**

**二、实验任务及内容：**

**三、实验步骤：**

**定义一个const int Max=20的整形最大变量，以及模板类T。**

1. **类的数据成员：**

**T类型的数组**

**Int 类型的存放边关系的arc[Max][Max]的数组**

**Int类型的vertexnum，arcnum两个变量分别表示顶点的个数以及边的个数**

**Int 类型的visited[Max]的数组用于记载访问的标志**

1. **类的成员函数**
2. **构造函数**

**功能：用于初始化构造无向图**

**算法：**

**1 确定图的顶点的个数和边的个数**

**2初始化领接矩阵**

**3 一次输入每条边存储在领结矩阵的arc中**

**3.1输入边衣服的两个顶点的编号k，m**

**3.2将邻接矩阵的k行m列的元素值设置为1；**

**3.3将领接矩阵的m行k列的元素值设置为1；**

**（2）深度优先遍历**

**功能：用于对无向图的邻接矩阵存储的遍历**

**算法：**

**1 输出开始顶点的下标对应的元素的值**

**2 将输出的顶点的下标的访问标志设置为1**

**3 利用循环进行遍历寻找与该顶点具有边的并且没有被访问过的下标。**

**3.1 利用for循环遍历所有的顶点**

**3.2 如果找到具有边关系且没被访问过的顶点**

**3.2 递归调用该函数，参数为未被访问过的顶点的下标**

**（3）广度优先遍历**

**功能：用于对无向图的邻接矩阵存储的遍历**

**算法：**

**1 定义一个int 类型的Q[Max]数组用于作为队列**

**2 定义int类型的 front，rear 以及初始化为-1；**

**3 输出要开始访问的顶点的元素的值，并将此顶点的访问标志该为1**

**4 将访问过的顶点元素的下标进行入队**

**5 利用循环当队头下标不等于队尾的下标的时候**

**5.1 将front队头元素出队。**

**5.2 循环判断出队元素的对应的顶点是否与其他顶点具有边，并且**

**并且没有被访问过。**

**5.3 如有找到的话就输出该下标的顶点的元素，并将该顶点的访问标志改为1，并将此顶点元素的下标进行入队。**

**（3）该变函数：**

**功能：改变顶点元素的访问标志**

**算法；**

**利用for循环对visited[Max]中的所有元素的值改变为0**

**（4）析构函数：**

**功能：释放该无向图顶点元素所在的数组，以及边关系的数组。**

**四、程序代码**

**//无向图的领接矩阵存储结构**

**#include<iostream.h>**

**const int Max=20;**

**template<class T>**

**class MGraph**

**{**

**private:**

**T vertex[Max]; //存放顶点元素**

**int arc[Max][Max]; //存放边的关系**

**int vertexnum,arcnum; //顶点的个数以及边的个数**

**int visited[Max]; //访问标志**

**public:**

**MGraph(T a[],int vertexnum,int arcnum); //构造函数**

**~MGraph(){}; //析构函数**

**void DFS(int v); //深度优先遍历**

**void BFS(int v); //广度优先遍历**

**void Change();**

**};**

**template<class T>**

**MGraph<T>::MGraph(T a[],int vertexnum,int arcnum)**

**{**

**this->vertexnum=vertexnum;**

**this->arcnum=arcnum;**

**//初始化顶点元素**

**for(int i=0;i<vertexnum;i++)**

**vertex[i]=a[i];**

**//输入具有边关系的两个顶点，先对二维数组进行初始化**

**for(i=0;i<vertexnum;i++)**

**for(int j=0;j<vertexnum;j++)**

**arc[i][j]=0;**

**//输入具有联系的两条边**

**cout<<"请输入具有联系的边数"<<endl;**

**for(i=0;i<arcnum;i++)**

**{**

**int k,m;**

**cin>>k>>m;**

**arc[k][m]=1;**

**arc[m][k]=1;**

**}**

**//初始化访问确定的visited**

**for(i=0;i<vertexnum;i++)**

**visited[i]=0;**

**}**

**//V表示顶点的下标**

**template<class T>**

**void MGraph<T>::DFS(int v)**

**{**

**cout<<vertex[v];**

**visited[v]=1;**

**for(int j=0;j<vertexnum;j++)**

**if(arc[v][j]==1&&visited[j]==0)**

**DFS(j);**

**}**

**//v表示顶点的下标**

**template<class T>**

**void MGraph<T>::BFS(int v)**

**{**

**int Q[Max];**

**int front,rear;**

**front=rear=-1;**

**cout<<vertex[v];**

**visited[v]=1;**

**Q[++rear]=v; //将访问过的元素入队**

**while(front!=rear)**

**{**

**int k=Q[++front];**

**for(int i=0;i<vertexnum;i++)**

**{**

**if(arc[k][i]==1&&visited[i]==0)**

**{**

**cout<<vertex[i];**

**visited[i]=1;**

**Q[++rear]=i;**

**}**

**}**

**}**

**}**

**//对visited进行重置的函数**

**template<class T>**

**void MGraph<T>::Change()**

**{**

**for(int i=0;i<vertexnum;i++)**

**visited[i]=0;**

**}**

**//析构函数**

**template<class T>**

**MGraph<T>::~MGraph()**

**{**

**cout<<"直接释放顶点的数组以及边关系的数组"<<endl;**

**delete vertex;**

**delete arc;**

**cout<<endl;**

**}**

**//主函数**

**void main()**

**{**

**char a[]={'A','B','C','D','E','F','G','H'};**

**int vertexnum=8;**

**int arcnum=9;**

**MGraph<char>MG(a,vertexnum,arcnum);**

**//先进行深度优先遍历**

**cout<<"请输入你要从那个顶点的下标进行深度优先遍历:"<<endl;**

**int v;**

**cin>>v;**

**MG.DFS(v);**

**cout<<endl;**

**MG.Change();**

**cout<<"请输入你要从那个点进行广度优先遍历:"<<endl;**

**int j;**

**cin>>j;**

**MG.BFS(j);**

**cout<<endl;**

**}**

**五、运行结果**



**六、存在问题及解决方法**

**代码方面问题还不够熟悉，算法方面有些遗漏的想法，得去翻翻书。**

**七、实验体会**

**通过本次的实验，我对无向图有了更深刻的理解以及认知，让我对此有了更进一步的认识，无向图的强大之处也有所领悟。**

**//邻接表方面**

**一、实验目的：熟练掌握数据结构中图，无向图的邻接表存储的知识的使用**

**二、实验任务及内容：**

**三、实验步骤：**

**定义一个const int Max=20的整形最大变量，以及模板类T。**

**定义一个边表结构体Arcnode**

**定义一个顶点表的结构体Vertexnode**

**1）类的数据成员：**

**存放顶点表的数组Vertexnode类型的数组**

**图的顶点数和边的个数**

1. **类的成员函数**

**（1）构造函数**

**功能：用于初始化构造无向图**

**算法：**

**1 确定图的顶点的个数和边的个数。**

**2 输入顶点信息存储在顶点的表中，并初始化该顶点的边表**

**3 一次输入边的信息，并将边所对应的领接点信息存储在边表中**

**3.1输入边所依附的两个顶点的编号k，m**

**3.2生成表节点s，其领接点的编号为m；**

**3.3将节点s插入到第k个边表的表头；**

**（2）深度优先遍历**

**功能：用于对无向图的邻接表存储的遍历**

**算法：**

**1 输出开始顶点的下标对应的元素的值**

**2 将输出的顶点的下标的访问标志设置为1**

**3 定义一个Arcnode\*p=new Arcnode，并将该输出顶点的参数adjvex[].firstedge赋值给p**

**4 当p不为空的时候**

**4.1将p->adjvex赋值给新定义的一个变量**

**4.2 判断p->adjvex是否被访问过**

**4.3 没被访问过的时候递归调用该函数。**

**4.4 p=p->next;**

**（3）广度优先遍历**

**功能：用于对无向图的邻接表存储的遍历**

**算法：**

**1 定义一个int 类型的Q[Max]数组用于作为队列**

**2 定义int类型的 front，rear 以及初始化为-1；**

**3 输出要开始访问的顶点的元素的值，并将此顶点的访问标志该为1**

**4 将访问过的顶点元素的下标进行入队**

**5 利用循环当队头下标不等于队尾的下标的时候**

**5.1 将front队头元素出队。**

**5.2将出队元素所在的位置adjlist[].firstedge赋值给p**

**5.3判断p是否为空，不为空的时候。**

**5.4 将p->adjvex赋值给新的变量**

**5.5判断一该变量为下标的vistied访问标志是否被访问过**

**5.6没访问过就输入并改变访问标志，并将该元素入队**

**5.7 p=p->next**

**（3）该变函数：**

**功能：改变顶点元素的访问标志**

**算法；**

**利用for循环对visited[Max]中的所有元素的值改变为0**

**（4）析构函数：**

**功能：释放该无向图顶点元素所在的数组，以及边关系的数组。**

**四、程序代码**

**//无向图的领接矩阵存储结构**

**/\*#include<iostream.h>**

**const int Max=20;**

**template<class T>**

**class MGraph**

**{**

**private:**

**T vertex[Max]; //存放顶点元素**

**int arc[Max][Max]; //存放边的关系**

**int vertexnum,arcnum; //顶点的个数以及边的个数**

**int visited[Max]; //访问标志**

**public:**

**MGraph(T a[],int vertexnum,int arcnum); //构造函数**

**~MGraph(); //析构函数**

**void DFS(int v); //深度优先遍历**

**void BFS(int v); //广度优先遍历**

**void Change();**

**};**

**template<class T>**

**MGraph<T>::MGraph(T a[],int vertexnum,int arcnum)**

**{**

**this->vertexnum=vertexnum;**

**this->arcnum=arcnum;**

**//初始化顶点元素**

**for(int i=0;i<vertexnum;i++)**

**vertex[i]=a[i];**

**//输入具有边关系的两个顶点，先对二维数组进行初始化**

**for(i=0;i<vertexnum;i++)**

**for(int j=0;j<vertexnum;j++)**

**arc[i][j]=0;**

**//输入具有联系的两条边**

**cout<<"请输入具有联系的边数"<<endl;**

**for(i=0;i<arcnum;i++)**

**{**

**int k,m;**

**cin>>k>>m;**

**arc[k][m]=1;**

**arc[m][k]=1;**

**}**

**//初始化访问确定的visited**

**for(i=0;i<vertexnum;i++)**

**visited[i]=0;**

**}**

**//V表示顶点的下标**

**template<class T>**

**void MGraph<T>::DFS(int v)**

**{**

**cout<<vertex[v];**

**visited[v]=1;**

**for(int j=0;j<vertexnum;j++)**

**if(arc[v][j]==1&&visited[j]==0)**

**DFS(j);**

**}**

**//v表示顶点的下标**

**template<class T>**

**void MGraph<T>::BFS(int v)**

**{**

**int Q[Max];**

**int front,rear;**

**front=rear=-1;**

**cout<<vertex[v];**

**visited[v]=1;**

**Q[++rear]=v; //将访问过的元素入队**

**while(front!=rear)**

**{**

**int k=Q[++front];**

**for(int i=0;i<vertexnum;i++)**

**{**

**if(arc[k][i]==1&&visited[i]==0)**

**{**

**cout<<vertex[i];**

**visited[i]=1;**

**Q[++rear]=i;**

**}**

**}**

**}**

**}**

**//对visited进行重置的函数**

**template<class T>**

**void MGraph<T>::Change()**

**{**

**for(int i=0;i<vertexnum;i++)**

**visited[i]=0;**

**}**

**//析构函数**

**template<class T>**

**MGraph<T>::~MGraph()**

**{**

**cout<<"直接释放顶点的数组以及边关系的数组"<<endl;**

**delete vertex;**

**delete arc;**

**cout<<endl;**

**}**

**//主函数**

**void main()**

**{**

**char a[]={'A','B','C','D','E','F','G','H'};**

**int vertexnum=8;**

**int arcnum=9;**

**MGraph<char>MG(a,vertexnum,arcnum);**

**//先进行深度优先遍历**

**cout<<"请输入你要从那个顶点的下标进行深度优先遍历:"<<endl;**

**int v;**

**cin>>v;**

**MG.DFS(v);**

**cout<<endl;**

**MG.Change();**

**cout<<"请输入你要从那个点进行广度优先遍历:"<<endl;**

**int j;**

**cin>>j;**

**MG.BFS(j);**

**cout<<endl;**

**}\*/**

**//无向图的邻接表的存储**

**#include<iostream.h>**

**//定义边表节点**

**const int Max=20;**

**struct Arcnode**

**{**

**int adjvex;**

**Arcnode \*next;**

**};**

**template<class T>**

**struct Vertexnode**

**{**

**T vertex;**

**Arcnode \*firstedge;**

**};**

**template<class T>**

**class ALGraph**

**{**

**private:**

**Vertexnode<T>adjlist[Max]; //定义一个存放顶点元素的结构体类型的数组**

**int vertexnum,arcnum;//定义顶点的个数以及边的个数**

**int visited[Max]; //定义一个访问标志**

**public:**

**ALGraph(T a[],int vertexnum,int arcnum);**

**~ALGraph(){};**

**void DFS(int v);**

**void BFS(int v);**

**void Change();**

**};**

**template<class T>**

**ALGraph<T>::ALGraph(T a[],int vertexnum,int arcnum)**

**{**

**this->vertexnum=vertexnum;**

**this->arcnum=arcnum;**

**//先对访问标志进行该表为0；**

**for(int i=0;i<vertexnum;i++)**

**visited[i]=0;**

**//对顶点元素进行初始化**

**for(i=0;i<vertexnum;i++)**

**{**

**adjlist[i].vertex=a[i];**

**adjlist[i].firstedge=NULL;**

**}**

**cout<<"请输入具有边的两个顶点之间的下标"<<endl;**

**//输入具有边关系的两个顶点**

**for(int k=0;k<arcnum;k++)**

**{**

**int m;**

**int n;**

**cin>>m>>n;**

**Arcnode \*s=new Arcnode;**

**s->adjvex=n;**

**s->next=adjlist[m].firstedge;**

**adjlist[m].firstedge=s;**

**//还要进行再次的处理,就是边表**

**s=new Arcnode;**

**s->adjvex=m;**

**s->next=adjlist[n].firstedge;**

**adjlist[n].firstedge=s;**

**}**

**}**

**//深度优先遍历**

**template<class T>**

**void ALGraph<T>::DFS(int v)**

**{**

**cout<<adjlist[v].vertex;**

**visited[v]=1;**

**Arcnode \*p=new Arcnode;**

**p=adjlist[v].firstedge;**

**while(p!=NULL)**

**{**

**int t=p->adjvex;**

**if(visited[t]==0)**

**DFS(t);**

**p=p->next;**

**}**

**}**

**//广度优先遍历**

**template<class T>**

**void ALGraph<T>::BFS(int v)**

**{**

**int Q[Max];**

**int front,rear;**

**front=rear=-1;**

**cout<<adjlist[v].vertex;**

**visited[v]=1;**

**Q[++rear]=v;**

**while(front!=rear)**

**{**

**int t=Q[++front];**

**Arcnode \*T=new Arcnode;**

**T=adjlist[t].firstedge;**

**while(T!=NULL)**

**{**

**int k=T->adjvex;**

**if(visited[k]==0)**

**{**

**cout<<adjlist[k].vertex;**

**visited[k]=1;**

**Q[++rear]=k;**

**}**

**T=T->next;**

**}**

**}**

**}**

**template<class T>**

**void ALGraph<T>::Change()**

**{**

**for(int i=0;i<vertexnum;i++)**

**visited[i]=0;**

**}**

**void main()**

**{**

**char a[]={'A','B','C','D','E','F','G','H'};**

**int vertexnum=8;**

**int arcnum=9;**

**ALGraph<char>AL(a,vertexnum,arcnum);**

**//先进行深度优先遍历**

**cout<<"请输入你要从那个顶点的下标进行深度优先遍历:"<<endl;**

**int v;**

**cin>>v;**

**AL.DFS(v);**

**cout<<endl;**

**AL.Change();**

**cout<<"请输入你要从那个点进行广度优先遍历:"<<endl;**

**int j;**

**cin>>j;**

**AL.BFS(j);**

**cout<<endl;**

**}**

**五、运行结果**



**六、存在问题及解决方法**

**代码方面问题还不够熟悉，算法方面有些遗漏的想法，得去翻翻书。**

**七、实验体会**

**通过本次的实验，我对无向图有了更深刻的理解以及认知，让我对此有了更进一步的认识，无向图的邻接表的存储方式强大之处也有所领悟。**

**实验六：排序和查找（直接插入排序、希尔排序、冒泡排序、快速排序、简单选择排序、堆排序、顺序表的顺序查找和折半查找）**

**一、实验目的：熟练掌握数据结构中直接插入排序**

**二、实验任务及内容：完成直接插入排序的运用。**

**三、实验步骤：**

**函数：创建直接插入排序的函数。**

**功能：对一组无序的下标从1开始的数组进行数组进行直接从插入排序。**

**算法：**

1. **先假定无序区数组的下标为i为2到n，有序区为j=i-1**
2. **把r[i]的值赋给哨所r[0]**
3. **在有序区中如果r[0]<r[j]**
4. **交换r[j]<->r[j+1]的位置,j--**
5. **再把有序区的元素的值赋值给无序区交被换元素的位置**
6. **重复上诉操作**
7. **程序代码**

**void StInsert\_sort(int r[],int n)**

**{**

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

**{**

**r[0]=r[i];**

**for(int j=i-1;r[0]<r[j];j--)**

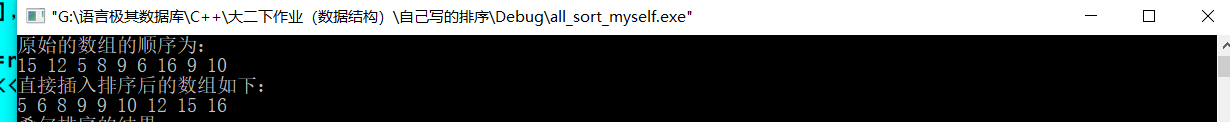
**r[j+1]=r[j];**

**r[j+1]=r[0];**

**}**

**}**

**五、运行结果**



**六、存在问题及解决方法**

**不存在问题。**

**七、实验体会**

**可以说，我就我个人来看，本次实验对于所有的编程者都是非常的有用的，本次实验基本就是一些流程控制中的算法，用到的更是所有的编程语言都有的流程控制语句，通过本次实验让我对各种查询和排序的方法有了更深刻的理解。**

**一、实验目的：熟练掌握数据结构中希尔排序的算法思想。**

**二、实验任务及内容：定义希尔排序的函数，并返回每趟希尔排序的结果。**

**三、实验步骤：**

**函数：定义一个希尔排序的函数**

**功能：对一组无序的数组进行排序**

**算法：**

1. **设置一个计数器count**
2. **对数组进行划分分组每次分组为原来的一半记为d**
3. **将分组后的将无序区的第一个元素赋给r[0]**
4. **将有序区的初始元素的下标设置为j=i-d**
5. **逐步比较r[0]跟有序区中元素的大小，如果r[0]小则将有序区的元素赋值给无序区**
6. **结束后将r[0]的元素赋值给有序区的交换元素的位置**
7. **打印本次希尔排序的结果**
8. **循环上述过程**

**四、程序代码**

**void Shell\_sort(int r[],int n)**

**{**

**int count=1;**

**cout<<"希尔排序的结果："<<endl;**

**for(int d=n/2;d>=1;d/=2)**

**{**

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

**{**

**r[0]=r[i];**

**for(int j=i-d;j>0&&r[0]<r[j];j=j-d)**

**r[j+d]=r[j];**

**r[j+d]=r[0];**

**}**

**cout<<"第"<<count<<"趟希尔排序的结果："<<endl;**

**Print2(r,n);**

**cout<<endl;**

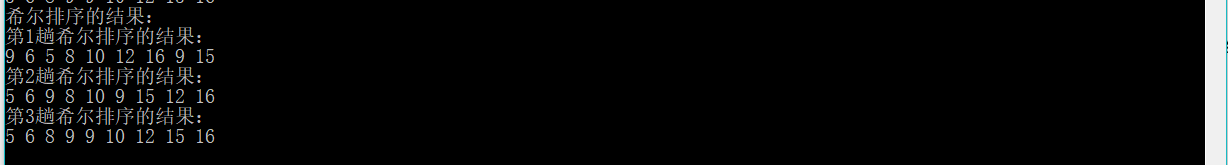
**count++;**

**}**

**cout<<endl;**

**}**

1. **运行结果**



**六、存在问题及解决方法**

**不存在问题。**

**七、实验体会**

**可以说，我就我个人来看，本次实验对于所有的编程者都是非常的有用的，本次实验基本就是一些流程控制中的算法，用到的更是所有的编程语言都有的流程控制语句，通过本次实验让我对各种查询和排序的方法有了更深刻的理解。**

**一、实验目的：熟练掌握数据结构中冒泡排序的算法思想。**

**二、实验任务及内容：定义冒泡排序的函数，并对一组无序数组进行排序。**

**三、实验步骤：**

**函数：冒泡排序的函数。**

**功能：对一组无序的数组进行冒泡排序**

**算法：**

1. **先设置一个记录交换位置的变量exchange，将变量初始化为n**
2. **利用循环逐步比较两两对应元素的大小，前一个元素大于后一个元素的时候进行交换，交换后将交换的后面的一个元素的地址赋值给exchange**
3. **如果没有交换则将0赋值给exchange**
4. **判断exchange是否为零，不为0则进行上诉循环，遍历以及两两比较的数组位置为1到exchange。**

**四、程序代码**

**void Bubble\_sort(int r[],int n)**

**{**

**cout<<"冒泡排序的结果："<<endl;**

**int exchange=n;**

**while(exchange!=0)**

**{**

**int bound=exchange;**

**exchange=0;**

**for(int j=1;j<bound;j++) //一趟排序**

**{**

**if(r[j]>r[j+1]){**

**r[0]=r[j];**

**r[j]=r[j+1];**

**r[j+1]=r[0];**

**exchange=j; //记录交换的位置**

**}**

**}**

**}**

**}**

**五、运行结果**



**六、存在问题及解决方法**

**不存在问题。**

**七、实验体会**

**可以说，我就我个人来看，本次实验对于所有的编程者都是非常的有用的，本次实验基本就是一些流程控制中的算法，用到的更是所有的编程语言都有的流程控制语句，通过本次实验让我对各种查询和排序的方法有了更深刻的理解。**

**一、实验目的：熟练掌握数据结构中快速排序的算法**

**二、实验任务及内容：定义快速排序的函数，对一组无序数组进行快速排序。**

**三、实验步骤：**

**1）函数：一次快速排序的函数**

**功能：对数组进行一次快速排序**

**算法：**

1. **将数组的第一个元素的地址赋值给i，将数组的最后一个元素的地址赋值给j**
2. **将第一个元素的元素值赋值给r[0]**
3. **从右则开始扫描，判断r[j]是否小于r[0],小于则将r[j]的元素值赋值给r[i],并且i++，否则j--；**
4. **从左侧开始扫描，判断r[i]是否大于r[0]，如果大于则将r[i]所在元素的地址赋值给r[j],并且j--；**
5. **当i=j的时候 然后i所在的位置**
6. **重复上诉操作。**

**2）函数：快速排序函数**

**功能：对数组进行多趟的快速排序**

**算法：**

1. **定义一个变量t将一趟排序所返回的位置赋值给t**
2. **从左侧进行快速排序范围为1到t-1（递归调用该函数）**
3. **从右则进行快速排序范围为t+1到n（递归调用该函数）**
4. **判断i是否等与j，不相等则重复上诉过程。**

**四、程序代码**

**int Partition(int r[],int first,int end)**

**{**

**int i=first;**

**int j=end;**

**r[0]=r[first];**

**while(i<j)**

**{**

**//右侧扫描**

**while(i<j&&r[0]<=r[j])j--;**

**if(i<j) r[i++]=r[j];**

**//左侧扫描**

**while(i<j&&r[0]>=r[i])i++;**

**if(i<j) r[j--]=r[i];**

**}**

**r[i]=r[0]; //把r[0]的值给赋值回去**

**return i;**

**}**

**void Quick\_sort(int r[],int first,int end)**

**{**

**if(first<end)**

**{**

**int t=Partition(r,first,end);**

**//左边进行快速排序**

**Quick\_sort(r,first,t-1);**

**Quick\_sort(r,t+1,end);**

**}**

**}**

1. **运行结果**



**六、存在问题及解决方法**

**不存在问题。**

**七、实验体会**

**可以说，我就我个人来看，本次实验对于所有的编程者都是非常的有用的，本次实验基本就是一些流程控制中的算法，用到的更是所有的编程语言都有的流程控制语句，通过本次实验让我对各种查询和排序的方法有了更深刻的理解。**

**一、实验目的：熟练掌握数据结构中简单选择排序**

**二、实验任务及内容：定义简单选择排序的函数，对数组进行简单选择排序。**

**三、实验步骤：**

**函数：简单选择排序的函数**

**功能：对数组进行简单的选择排序**

**算法：**

1. **首先定义一个变量index，并将首元素的下标赋值给该变量**
2. **将该元素的下标所对应的元素该数组的所有元素进行比较，如果存在元素比index的下标所对应的元素的值小的，则将该元素的下标赋值给index**
3. **判断index是否等于首元素所对应的下标，如果不等于，则将首元素跟index下标所对应的元素进行交换。**
4. **将下一个元素赋值给index，重复上诉操作，直至除了最后一个数组的元素不去两两比较，其他的都比较后结束。**

**四、程序代码**

**void Select\_sort(int r[],int n)**

**{**

**cout<<"简单选择排序："<<endl;**

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

**{**

**int index=i;**

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

**if(r[index]>r[j]) index=j;**

**if(index!=i){**

**r[0]=r[i];**

**r[i]=r[index];**

**r[index]=r[0];**

**}**

**}**

**}**

**五、运行结果**



**六、存在问题及解决方法**

**不存在问题。**

**七、实验体会**

**可以说，我就我个人来看，本次实验对于所有的编程者都是非常的有用的，本次实验基本就是一些流程控制中的算法，用到的更是所有的编程语言都有的流程控制语句，通过本次实验让我对各种查询和排序的方法有了更深刻的理解。**

**一、实验目的：熟练掌握数据结构中堆排序。**

**二、实验任务及内容：定义堆排序的函数对，无序数组进行堆排序。**

**三、实验步骤：**

**函数：初始化建堆函数。**

**功能：建立大根堆。**

**算法：**

1. **从元素的下标为i所在的元素的地址开始建堆，并将2\*i的值赋值给j**
2. **当j小于或者等于顶点元素的个数时候。**
3. **判断i是否存在右孩子，存在的话就判断左右孩子的大小，将值打的元素所在的下标赋值给j**
4. **判断i下表所在对应的元素跟j下标所对应元素的大小，如果i下标所对应的元素小于j下标所对应的元素的值的是否交换位置**
5. **并将2\*i的值赋给i，j等于2\*i**
6. **判断j是否小于或者等于顶点元素的个数。重复上诉（3），（4）（5）操作**

**函数：堆排序函数**

**功能：对建好的堆进行堆的排序**

**算法：**

1. **利用循环从数组的元素个数除以2的位置开始进行建堆（i）**
2. **调用建堆函数**
3. **将i的上一个下标赋值给i，直至i=0；**
4. **I=0利用循环，将建好的大根堆，数组的第一个元素跟后面的下标为n-i+1的元素进行交换。**
5. **每次交换后调用建堆函数建堆的范围为1到n-i**
6. **判断i是否等于n，不等于的时候i进行自加。**

**四、程序代码**

**void Sift(int r[],int k,int m)**

**{ //m表示最后一个顶点的下标**

**int i=k; //k表示被晒选开始的节点**

**int j=2\*i; //j表示双亲的左孩子节点**

**while(j<=m)**

**{**

**if(j<m&&r[j]<r[j+1]) j++;**

**if(r[i]>=r[j]) break;**

**else {**

**r[0]=r[i];**

**r[i]=r[j];**

**r[j]=r[0];**

**i=j;j=2\*i; //只要发生了交换就进行下一个进行排序**

**}**

**}**

**}**

**//堆排序以及重新建堆**

**//从2份之n开始进行堆的初始化**

**void Heap\_sort(int r[],int n)**

**{**

**cout<<"堆排序的结果："<<endl;**

**for(int i=n/2;i<=1;i--)**

**Sift(r,i,n);**

**//重建堆**

**for(i=1;i<=n;i++){**

**r[0]=r[1];**

**r[1]=r[n-i+1];**

**r[n-i+1]=r[0];**

**Sift(r,1,n-i); //移动之后又进行一个排序，并从移动之后的顶点开始排序**

**}**

**}**

**五、运行结果**



**六、存在问题及解决方法**

**不存在问题。**

**七、实验体会**

**可以说，我就我个人来看，本次实验对于所有的编程者都是非常的有用的，本次实验基本就是一些流程控制中的算法，用到的更是所有的编程语言都有的流程控制语句，通过本次实验让我对各种查询和排序的方法有了更深刻的理解。**

**一、实验目的：熟练掌握数据结构中顺序表中的顺序查找。**

**二、实验任务及内容：**

**找出一组数组中，值为K元素所对应的元素的下标。**

**三、实验步骤：**

**函数：顺序查找函数**

**功能：找出数组中值为K的元素所对应的下标**

**算法：**

1. **将所要查找的元素的值赋值给哨所元素r[0],并将元素的个数n赋值给新定义的一个整形的变量i**
2. **利用循环判断r[i]是否等于r[n]，不等于的时候i—**
3. **返回i**

**四、程序代码**

**int SeqSearch2(int r[],int n,int key)**

**{**

**cout<<"改进后的顺序查询："<<endl;**

**int i=n;**

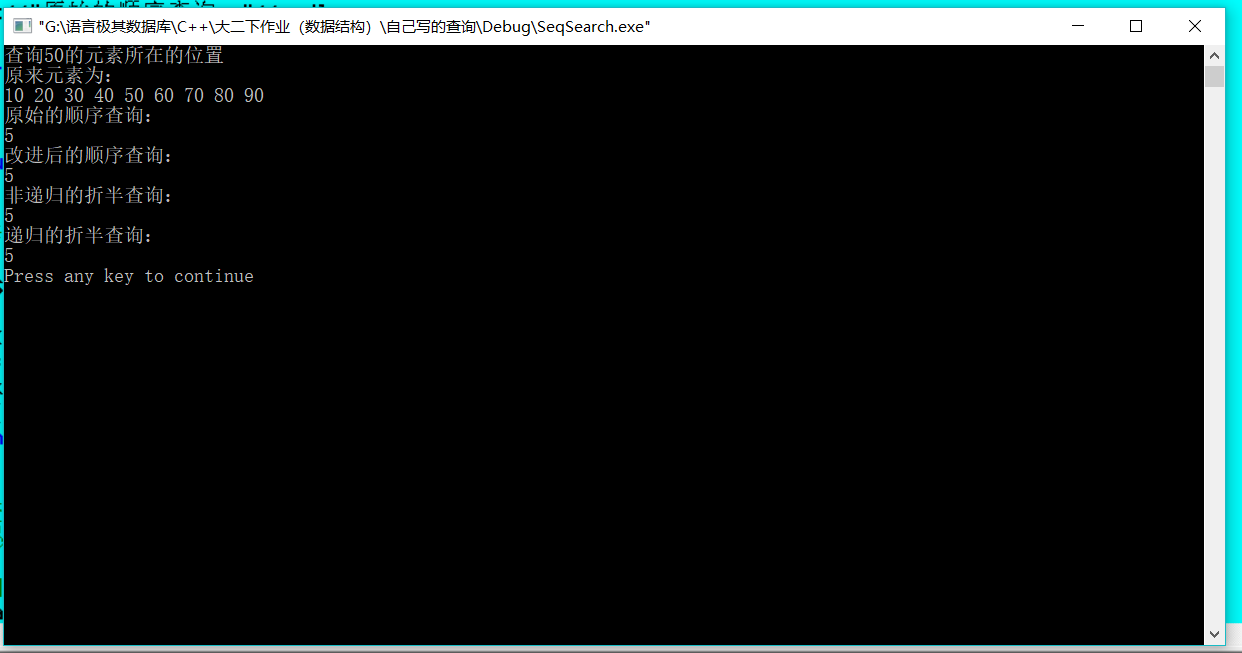
**r[0]=key;**

**while(r[i]!=r[0]) i--;**

**return i;**

**}**

1. **运行结果**



1. **存在问题及解决方法**

**不存在问题。**

**七、实验体会**

**可以说，我就我个人来看，本次实验对于所有的编程者都是非常的有用的，本次实验基本就是一些流程控制中的算法，用到的更是所有的编程语言都有的流程控制语句，通过本次实验让我对各种查询和排序的方法有了更深刻的理解。**

**一、实验目的：熟练掌握数据结构中折半查找**

**二、实验任务及内容：定义一个折半查找的函数，利用折半查找，找出元素K所在的位置。（数组的元素前提定位由小到大的排序）**

**三、实验步骤：**

**函数：折半查找函数。**

**功能：折半查找到元素K所对应的位置。**

**算法：**

1. **将首元素的下标赋值给i，将末尾元素的下标赋值给j**
2. **判断i>j的时候，返回0，错误**
3. **否则定义一个新的变量mid并将（i+j）/2的值赋给mid**
4. **判断mid下标所在的位置的元素的值是否大于K，如果大于K，递归调用该函数，并将mid-1赋值给末尾元素的下标j**
5. **判断mid下标所在的位置的元素的值是否小于K，如果小于K，递归调用该函数，并将mid+1赋值首元素的下标i**
6. **如果mid下标所对应的元素的值等于K则返回 mid**
7. **否则返回0，表示找不到。**

**四、程序代码**

**int BinarySearch2(int r[],int low,int high,int key)**

**{**

**if(low>high) return 0; //递归的边界**

**else {**

**int mid=(low+high)/2;**

**if(r[mid]>key) BinarySearch2(r,low,mid-1,key);**

**else if(r[mid]<key) BinarySearch2(r,mid+1,high,key);**

**else return mid;**

**}**

**return 0;**

**}**

**五、运行结果**



**六、存在问题及解决方法**

**不存在问题**

**七、实验体会**

**可以说，我就我个人来看，本次实验对于所有的编程者都是非常的有用的，本次实验基本就是一些流程控制中的算法，用到的更是所有的编程语言都有的流程控制语句，通过本次实验让我对各种查询和排序的方法有了更深刻的理解。**