数 据 结 构

实 验 指 导 书

计算机与人工智能学院

实验一 线性表及其应用

一、实验目的

1.掌握线性表的顺序存储结构——顺序表的定义及其基本运算的C++语言实现。

2.掌握线性表的链式存储结构——单链表的定义及其基本运算的C++语言实现。

二、实验内容

项目名称：学生信息管理系统

项目内容：设计一个学生信息管理系统，实现对学生基本信息的添加、删除、修改和查询等操作，其中每个学生信息包含学号，姓名和绩点。要求系统完成以下主要功能：

（1）显示：显示当前所有学生信息记录；

（2）录入：从键盘输入一条学生信息记录，插入到表中指定的位置；

（3）查找：根据学号或者记录的位置查找学生的各项信息；

（4）删除：删除指定位置的学生信息记录；

（5）更新：更新指定位置的学生信息记录；

（6）统计：统计表中学生人数。

（7）排序：按照学号或者绩点进行排序

（8）清空：清空表中所有记录

三、实验实现

1.顺序表及其基本运算的实现

参考以下类的定义,实现顺序表类,完成基于顺序表的学生信息管理系统。

```
// 顺序表模板类的申明
template <typename T>
class SqList
{
// 顺序表的数据成员
    int length;                  // 顺序表的当前长度
    T *data;             // 元素存储空间的首地址
public:
// 顺序表的函数成员
    SqList();   // 构造一个空表
    ~SqList();           // 析构函数
    void CreateList(T v[], int n);  // 根据数组v的内容构造顺序表
    int ListLength() ;       // 取顺序表长度
    int LocateElem(T e) ;   // 元素定位，求指定元素在顺序表中的位置
    bool GetElem(int i, T &e) ; // 取顺序表中第i个元素的值
```

```
    bool SetElem(int i, const T &e);    // 修改顺序表中第 i 个元素的值
    bool ListDelete(int i, T &e);       // 删除顺序表中第 i 个元素
    bool ListInsert(int i, T e); // 在顺序表第 i 个位置插入元素
    void DispList();
};
```

2.单链表及其基本运算的实现

参考以下类的定义,实现单链表类,完成基于单链表的学生信息管理系统。

```
// 结点类
template <typename T>
class Node
{// 数据成员:
 T data;              // 数据域
 Node<T> *next;       // 指针域
// 构造函数:
 Node();                        // 无参数的构造函数
 Node(T e, Node<T> *next = NULL);    // 已知数数据元素值和指针建立结点
};
// 单链表类
template <typename T>
class LinkList
{
//  单链表的数据成员
    Node<T> *head;              // 头结点指针
    int length;                 // 单链表长度
public:
//  单链表的函数成员
    LinkList();                         // 无参数的构造函数
    ~LinkList();                // 析构函数
    void CreateList(T v[], int n);  // 根据数组 v 的内容构造顺序表
    int ListLength() ;      // 取顺序表长度
    int LocateElem(T e) ;   // 元素定位, 求指定元素在顺序表中的位置
    bool GetElem(int i, T &e) ; // 取顺序表中第 i 个元素的值
    bool SetElem(int i, const T &e);    // 修改顺序表中第 i 个元素的值
    bool ListDelete(int i, T &e);       // 删除顺序表中第 i 个元素
```

```
    bool ListInsert(int i, T e); // 在顺序表第 i 个位置插入元素
    void DispList();
};
```

四、实验注意事项

1. 建议使用 c++语言的模板类实现;
2. 设计具有通用性、可复用性,代码可读性强;
3. 实验分析和设计要有详尽的描述;
4. 在完成基本功能的基本上可以自己扩展其他功能。
5. 所给代码仅供参考，书写形式、习惯以依自己而定。

实验二 栈及其应用（1）

一、实验目的

1.掌握栈的顺序存储结构——顺序栈的定义及其基本运算的C++语言实现。

2.掌握栈的链式存储结构——链式栈的定义及其基本运算的C++语言实现。

二、实验内容

项目名称：简易计算器程序

项目内容：编写程序，模拟简单运算器的工作：输入一个算式（没有空格），遇等号“=”说明输入结束，输出结果。假设计算器只能计算加减乘除运算，运算数和结果都是整数。要求完成以下功能：

（1） 从键盘录入中缀表达式，将中缀表达式转换为后缀表达式输出；

（2） 输入后缀表达式，计算后缀表达式的值。

三、实验实现

1.顺序栈及其基本运算的实现

参考以下类的定义,实现顺序栈类,完成基于顺序栈的简易计算器程序。

```
template<typename T>// 顺序栈模板类
class SqStack
{// 顺序栈的数据成员：
    int top;                                    // 栈顶指针
    T *data;                                  // 元素存储空间
public:
//  顺序栈的方法声明及重载编译系统默认方法声明：
    SqStack();          // 构造函数
    ~SqStack();         // 析构函数
    bool StackEmpty();                          // 判断栈是否为空
    bool Push(T e);                  // 入栈
    bool Pop(T &e);                      // 出栈
    bool GetTop(T &e);               // 取栈顶元素
};
```

2.链式栈及其基本运算的实现

参考以下类的定义,实现链式栈类,完成基于链式栈的简易计算器程序。

```
// 结点类
template <typename T>
class Node
```

```
{// 数据成员:
    T data;          // 数据域
    Node<T> *next;       // 指针域
// 构造函数:
    Node();                      // 无参数的构造函数
    Node(T item, Node<T> *next= NULL);  // 已知数数据元素值和指针建立结构
};
//链栈类
template <typename T>
class LinkStack
{
//  单链表的数据成员
    Node<T> *head;               // 链栈头结点指针
public:
//  链栈类的函数成员
    LinkStack();             // 构造函数
    ~LinkStack();            // 析构函数
    bool StackEmpty();                       // 判断栈是否为空
    bool Push(T e);                  // 入栈
    bool Pop(T &e);                      // 出栈
    bool GetTop(T &e);               // 取栈顶元素
};
```

四、实验注意事项

1.建议使用c++语言的模板类实现;

2.设计具有通用性、可复用性,代码可读性强;

3.实验分析和设计要有详尽的描述;

4.在完成基本功能的基本上可以自己扩展其他功能。

5.所给代码仅供参考，书写形式、习惯以依自己而定。

实验二 队列及其应用（2）

一、实验目的

1.掌握队列的顺序存储结构——循环队列的定义及其基本运算的C++语言实现。

2.掌握队列的链式存储结构——链式队列的定义及其基本运算的C++语言实现。

二、实验内容

项目名称：医院排队系统

项目内容：编写一个程序模拟患者到医院看病排队挂号就诊的情况。在患者排队过程中：

（1）患者到达诊室，将病历本交给护士，排到等待队列中候诊。

（2）护士从等待队列中取出下一位患者的病历，该患者进入诊室就诊。

（3）医生下班，提醒余下患者明日就诊。

三、实验实现

1.循环队列及其基本运算的实现

参考以下类的定义,实现循环队列类,完成基于循环队列的医院排队系统。

```
#define DEFAULT_SIZE 100
// 循环队列类
template<typename T>
class SqQueue
{
    int front, rear;                                    // 队头队尾指针
    T *data;                                        // 元素存储空间
public:
    SqQueue();                      // 构造函数
    ~SqQueue();                                 // 析构函数
    bool QueueEmpty();                                  // 判断队列是否为空
    bool deQueue(T &e);                         // 出队操作
    bool enQueue(T e);                      // 入队操作
    bool GetHead(T &e);                     // 取队头操作
};
```

2.链式队列及其基本运算的实现

参考以下类的定义,实现链式队列类,完成基于链式队列的医院排队系统。

```
// 结点类
template <typename T>
```

```
class Node
{
// 数据成员:
    T data;             // 数据域
    Node<T> *next;      // 指针域
// 构造函数:
    Node();                         // 无参数的构造函数
    Node(T item, Node<T> *next = NULL); // 已知数数据元素值和指针建立结构
};
// 链式队列结点类
template <typename T>
class QueueNode
{
// 数据成员:
    Node<T> *front;         // 指向队首结点
    Node<T> *rear;      // 指向队尾结点
};
// 链式队列类
template<typename T>
class LinkQueue
{
    QueueNode<T> *Q;                                // 链队结点指针 Q
public:
    LinkQueue();                        // 构造函数
    ~ LinkQueue();                      // 析构函数
    bool QueueEmpty();                  // 判断队列是否为空
    bool deQueue(T &e);                         // 出队操作
    bool enQueue(T e);                      // 入队操作
    bool GetHead(T &e);                     // 取队头操作
};
```

四、实验注意事项

1. 建议使用 c++语言的模板类实现；

2. 设计具有通用性、可复用性，代码可读性强；

3. 实验分析和设计要有详尽的描述；

4. 在完成基本功能的基本上可以自己扩展其他功能。

5. 所给代码仅供参考，书写形式、习惯以依自己而定。

实验三 树及其应用（1）

一、实验目的

1. 通过二叉树的构造掌握二叉树的链式存储结构

2. 掌握二叉树的递归遍历算法和非递归遍历算法以及查找等基本树的运算方法

二、实验内容

项目名称：基于二叉树的家谱系统

项目内容：采用一棵二叉树来表示一个家谱关系，一个家谱可表示为一棵树，首先将其转换成一棵二叉树表示，如下图为红楼梦家谱的一部分：

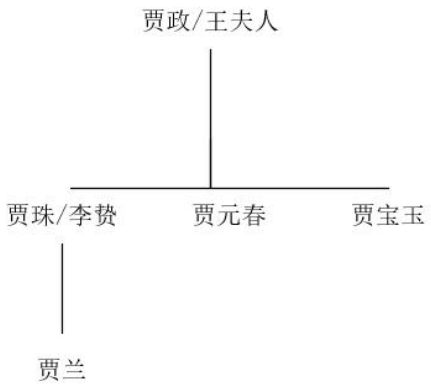


图 1.1 家谱的树形表示

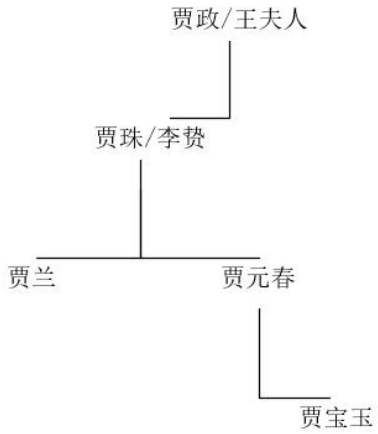


图 1.2 家谱的二叉树表示

要求完成的功能如下：

（1） 输入一棵二叉树的括号表示法，完成树的构建

（2） 使用后序遍历递归算法遍历二叉树并输出

（3） 使用先序遍历非递归算法遍历二叉树并输出

（4） 指定家谱中的某一成员，输出其所有长辈

三、实验实现

参考以下类的定义完成基于二叉树的家谱系统

其中树的实现类定义如图 2。在其具体实现过程中可以调用前面学习过的栈结构，栈的实现类定义如图 3。其中找出某一成员所有长辈需要与找二叉树中某一节点的所有祖先元素区分，家谱的二叉树表示中左子树代表孩子，右子树代表兄弟，可以基于后序非递归算法实现，其具体实现可以参考图 4。示例输出如图 5。

测试例:

输入：A(B(C(E,F),D(G(M,N),H)),)

输出：

M 的长辈为：FCNGHDBA

N 的长辈为：HDBA

```
#pragma once
#ifndef __BTREE_H__
#define __BTREE_H__
#include<stdio.h>
#include<malloc.h>
#include<iostream>
#define MaxSize 100
using namespace std;
template<class ElemType>  <T>
class BTree
{
private:
    //树的结构体定义
    typedef struct node
    {
        ElemType data;//树节点值
        struct node* lchild;//树节点左孩子
        struct node* rchild;//树节点右孩子
    } BTNode;
     BTNode* b;//根节点

public:
    BTree(std::string str);//根据括号表示法初始化树
    ~BTree();
    void PostOrder();
    void PostOrderRe(BTNode* b=b);//后序遍历非递归算法
    void PreOrderNotRe();//先序遍历非递归算法
    void FindAllAncestor(char object);//输出节点object的所有长辈
    void ShowAncestor(BTNode* ancestor);//输出ancestor的值及其同辈的值
                                        //孩子兄弟表示法中右子树代表兄弟
    void DestroyBTree(BTNode*& b);//销毁树
};
#endif __BTREE_H__ // !__BTREE_H__
```

图 2 树的实现类

```
#pragma once
#ifndef __STACK_H__
#define __STACK_H__
#include<stdio.h>
#include<malloc.h>
#include<iostream>
#define MaxSize 100

template<class ElemType>  <T>
class Stack
{
private:
    typedef struct
    {
        ElemType* data[MaxSize];
        int Top;//栈顶指针
    }SqStack;
    SqStack* s;
public:
    Stack();
    ~Stack();
    bool Push(ElemType* e);//将元素e压入栈中
    bool Pop(ElemType*& e);//栈顶元素出栈赋值给e
    bool GetTop(ElemType*& e);//将栈顶元素赋值给e，并不进行出栈操作
    bool StackEmpty();//判断栈是否为空
    int GetNum();//得到栈顶指针
    ElemType* GetElement(int suf);//得到栈中指针为suf的元素
};
#endif // !__STACK_H__

```

图 3 栈的实现类

```
95  void BTree<ElemType>::FindAllAncestor(char object)
96  {
97      BTNode* r, * p;//r为空或指定上一步访问的节点元素，p指向当前元素
98      bool flag;//标记是否将右子树进栈
99      Stack<BTNode> stack;//初始化栈
100     p = b;
101     do
102     {
103         //将p节点的所有左子树入栈
104         while (p != NULL) { ... }
109         r = NULL; flag = true;
110         while (!stack.StackEmpty() && flag)
111         {
112             stack.GetTop(p);
113             if (p->rchild == r)//右子树为空或已经访问过
114             {
115                 if (p->data == object)//如果当前元素为要查找的成员
116                 {
117                     bool flag_ancestor = false;//用来标记下层的右子树，因为右子树代表兄弟
118                     int i = stack.GetNum();
119                     while (!flag_ancestor && i > 0)//用于去掉下层以右子树连接的元素
120                     {
121                         if (stack.GetElement(i-1)->lchild != NULL)
122                         {
123                             if (stack.GetElement(i - 1)->lchild->data == stack.GetElement(i)->data)
124                                 flag_ancestor = true;
125                         }
126                         i--;
127                     }//调用ShowAncestor(BTNode*)输出栈中剩余元素及其同辈
128                     while (i > -1 && flag_ancestor) { ... }
133                 }
134                 stack.Pop(p);
135                 r = p;
136             }//将右子树入栈
137             else { ... }
142         }
143     } while (!stack.StackEmpty());
144     printf("\n");
145 }
```

图 4 查找某一成员长辈的具体实现

```
请以括号表示法输入二叉树结构:A(B(C,D(,E)),)
后序遍历递归算法输出：CEDBA
前序遍历非递归算法输出：ABCDE
请输入指定人物代号，以查询其所有长辈：C
C的所有长辈为：BDEA
```

图 5 示例输出

四、实验注意事项

1. 建议使用 c++语言的模板类实现；

2. 设计具有通用性、可复用性，代码可读性强；

3. 实验分析和设计要有详尽的描述；

4. 在完成基本功能的基本上可以自己扩展其他功能。

5. 所给代码仅供参考，书写形式、习惯以依自己而定。

实验三 无向图及其应用（2）

一、实验目的

1. 掌握基于邻接矩阵存储结构的图的定义与实现。

2. 掌握图的 Prim 算法和 Kruskal 算法的实现以及应用

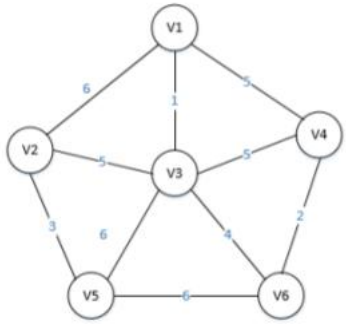
二、实验内容

项目名称：通信网构建

项目内容：在 n 个城市之间建立通信联络网，则连通 n 个城市只需要 n-1 条线路。要求在 最节省经费的前提下建立这个通信网。

（1） 完成城市信息的输入。

表 1 城市信息

城市间关系图	城市编号	城市名称
	V1	北京
	V2	成都
	V3	武汉
	V4	上海
	V5	广州
	V6	深圳

（2） 完成城市信息的编辑，包括城市以及城市间距离的增加，删除，信息修改等。

（3） 允许用户指定下列两种策略进行通信网的构建

1）采用 Prim 算法进行通信网的构建；

2）采用 Kruskal 算法进行通信网的构建；

测试数据不限于此。

三、实验要求

1、 线性表的应用

可以直接调用在前面课程中实现的线性表的类，该类中包含线性表的初始化操作，线性表中元素的增、删、改、查等功能。该线性表主要用来存放城市以及城市间距离信息，完成城市以及城市间距离的编辑功能，包括城市以及城市间距离的增加，删除，信 息修改。 线性表的定义参考如图 1 所示。

```
template <class T>  <T>
class SqList
{
private:
    T  *elem;      //保持不变，NULL 不存在
    int length;    //实际存放元素的个数
    int listsize;  //可以容纳的最大元素的个数

public:
    SqList();
    ~SqList();
    void InputList();
    void OutputList();
    Status Insert(int i, T e);   //在i位置插入一个元素
    Status Delete(int i, T &e);  //删除i位置的元素
    Status Update(int i, T e);   //在i位置更新一个元素
    int Locate(T e);             //根据元素查找在线性表中的位置
};
```

图 1 线性表的定义参考图

2、 基于邻接矩阵存储结构的图结构的定义与实现

设计并实现基于邻接矩阵的图存储结构，需要包括顶点增删改，边增删改等方法。图的定义参考如图 2，图 3 所示。

```
#pragma once
#define numMAX 20
#define StrMAX 100
#define MAX 10000
#include "SqList.h"

//顶点信息
struct Vex
{
    char Code[StrMAX];
    char Name[StrMAX];
};

//边的信息
struct Edge
{
    Vex vex1;
    Vex vex2;
    int weight;
};
```

图 2 结构体定义

```
class Graph
{
private:
    int AdjMatrix[numMAX][numMAX];   //邻接矩阵
    SqList<Vex> Vexs;                //点的集合
    SqList<Vex> Edges;               //边的集合
    int VexNum;                      //点的个数
public:
    Graph();
    ~Graph();

    bool InsertVex(Vex svex);
    bool DeleteVex(Vex svex);
    bool UpdateVex(Vex svex);
    bool InsertEdge(Edge sedge);
    bool DeleteEdge(Edge sedge);
    bool UpdateEdge(Edge sedge);

    Edge GetEdge(char *vex1Code, char *vex2Code);
    Vex GetVex(char *vex1Code);
    void SetVexNum(int);

    int PrimMinTree(Edge aPath[]);
    int KruskalMinTree(Edge aPath[]);

};
```

图 3 类定义

3、 Prim 算法的实现与应用

假设 G=(V, E)是一个具有 n 个顶点的带权无向连通图，T(U, TE)是 G 的最小生成树，其中 U 是 T 的顶点集，TE 是 T 的边集，则构造 G 的最小生成树 T 的步骤如下：

1）初始状态，TE 为空，U={v0}，v0∈V；

2）在所有 u∈U，v∈V-U 的边(u，v) ∈E 中找一条代价最小的边(u′，v′)并入 TE，同 时将 v′并入 U；

重复执行步骤（2）n-1 次，直到 U=V 为止。

4、 Kruskal 算法的实现与应用

假设 G=(V, E)是一个具有 n 个顶点的带权无向连通图，T(U, TE)是 G 的最小生成树，其中 U 是 T 的顶点集，TE 是 T 的边集，则构造 G 的最小生成树 T 的步骤如下：

1）置 U 的初值等于 V，TE 的初值为空

2）将图 G 中的边按权值从小到大的顺序依次选取：若选取的边未使生成树 T 形成回路， 则加入 TE，否则舍弃，直到 TE 中包含（n-1）条边为止。

5、 主函数的实现

主函数要求控制良好的界面操作，提示用户进行各种不同功能操作的选择。

四、实验注意事项

1. 建议使用 c++语言的模板类实现；

2. 设计具有通用性、可复用性,代码可读性强；

3. 实验分析和设计要有详尽的描述；

4. 在完成基本功能的基本上可以自己扩展其他功能。

5. 所给代码仅供参考，书写形式、习惯以依自己而定。

实验三 有向图及其应用（3）

一、 实验目的

1. 掌握基于邻接表存储结构的图的定义与实现。

2. 掌握图的拓扑排序算法的实现以及应用

二、 实验内容

项目名称：教学计划编制系统

项目内容：大学的每个专业都要制定教学计划。假设任何专业都有固定的学习年限，每学年包含两个学期，每个学期的时间长度和学分上限均相等。每个专业开设的课程都是确定的，而且课程在开设时间的安排上必须满足先修关系。每门课程有哪些先修课程是确定的，可以有任意多门，也可以没有。每门课恰好占一个学期。试在这样的前提下设计一个教学计划编制系统，该系统需要满足以下功能。

（1） 完成课程进修目录信息的读取。

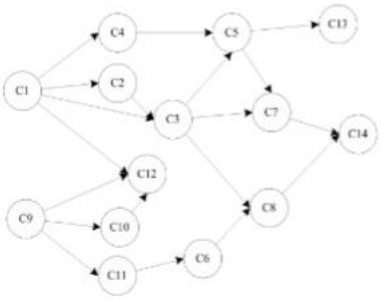
（2） 完成课程进修目录信息的编辑，包括课程的增加，删除，信息修改等。

（3） 学生的教学计划学期为 6，每个学期的学分上限为 10 分，允许用户指定下列编排策略进行教学计划的输出

1）使学生在各个学期中的负担尽量均匀；

2）使课程尽可能地集中在前几个学期中 若根据给定的条件问题无解，则报告适当信息；否则将教学计划输出到用户指定的文件中。计划的表格格式自行设计。

表 1 计算机专业进修课程

课程进修关系图	课程编号	课程名称	学分
	C1	程序设计基础	2
	C2	离散数学	3
	C3	数据结构	4
	C4	汇编语言	3
	C5	程序设计与分析	2
	C6	计算机原理	3
	C7	编译原理	4
	C8	操作系统	4
	C9	高等数学	7
	C10	线性代数	5
	C11	普通物理	2
	C12	数值分析	3
	C13	软件工程	3
	C14	数据库原理	3

测试数据不限于此。

三、 实验要求

1、 线性表的应用

可以直接调用在前面课程中实现的线性表的类，该类中包含线性表的初始化操作，线性表中元素的增、删、改、查等功能。该线性表主要用来存放课程进修目录信息，完成课程进修目录信息的编辑功能，

包括课程的增加，删除，信息修改。 线性表的定义参考如图 1 所示。

图 1 线性表的定义参考图

2、 堆栈的应用

可以直接调用在前面课程中实现的栈的类，该类中包含栈的初始化操作，判断栈是否为空操作，入栈和出栈操作。该栈主要用来存放入度为 0 的顶点，即当前没有先修关系，可以编排的课程。栈的定义参考如图 2 所示。

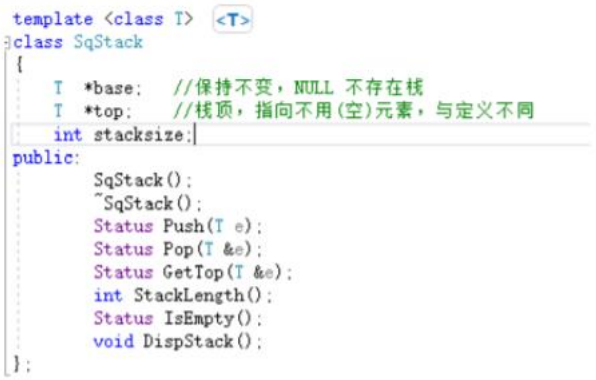


图 2 栈的定义参考图

3、 基于邻接表存储结构的图结构的定义与实现

设计并实现基于邻接表的图存储结构，需要包括创建图、展现图、统计图中各个顶点的入度等方法。图的定义参考如图 3，图 4 所示。

```
//弧信息
template <class T>
struct ArcInfo
{
    T From;   //起点
    T To;     //终点
    int weight;
};

//弧结点
template <class T> <T>
struct ArcNode
{
    int adjvex;   //临接点位置
    int weight;   //权值
    struct ArcNode *nextarc;
};

//顶点节点
template <class T>
struct VNode
{
    T data;
    int in;
    ArcNode *firstarc;
};
```

图 3 结构体定义

```
#pragma once
#include "GraphInfo.h"
#include "SqList.h"
#include "SqStack.h"

template<class T> <T>
class ALGraph
{
public :
    int vexnum;                 //顶点数目
    int arcnum;                 //弧数目
    VNode<T> vertices[Max];     //邻接表
public:
    ALGraph();
    ~ALGraph();
    void CreateGraph(int vnum, int anum,
        VNode<T> data[], ArcInfo arcList[]); //创建图
    void DispGraph();               //展示图
    int TopOrder();
    void IndegreeCal();             //统计每个顶点的入度
private:
    int LocateVex(VNode<T> v);      //根据顶点信息，返回顶点的坐标

};
```

图 4 类定义

4、 拓扑结构算法的实现与应用

拓扑排序的算法

（1） 在 AOV 网中选一个入度为 0 的顶点（没有前驱）且输出之；

（2） 从 AOV 网中删除此顶点及该顶点发出来的所有有向边；

（3） 重复（1）、（2）两步，直到 AOV 网中所有顶点都被输出或网中不存在入度为 0 的顶点。

5、 主函数的实现

主函数要求控制良好的界面操作，提示用户进行各种不同功能操作的选择。

四、实验注意事项

1. 建议使用 c++语言的模板类实现；

2. 设计具有通用性、可复用性,代码可读性强；

3. 实验分析和设计要有详尽的描述；

4. 在完成基本功能的基本上可以自己扩展其他功能。

5. 所给代码仅供参考，书写形式、习惯以依自己而定。

实验四 基于词频的文件相似度比对（1）

一、实验目的

（1）掌握 hash 查找方法；

（2）倒排索引表的应用。

二、实验内容

实现一种简单原始的文件相似度计算，即以两文件的公共词汇占总词汇的比例来定义相似度。为简化问题，这里不考虑中文，只考虑长度不小于 3、且不超过 10 的英文单词，长度超过 10 的只考虑前 10 个字母。

三、实验要求

1.输入说明：输入首先给出正整数 N（N <= 100），为文件总数。随后按照以下格式给出每个文件的内容：首先给出文件正文，最后在一行中只给出一个字符“#”，表示文件结束。在 N 个文件内容结束之后，给出查询总数 m（M <= 10^4），随后 M 行，每行给出一对文件编号，其间以空格分隔。这里假设文件按给出得到顺序从 1 到 N 编号。

2.输出说明：针对每一条查询，在一行输出两文件的相似度，即两文件的公共词汇占总词汇的百分比，精确到小数点后 1 位。

3.测试用例：

输入	输出	说明
3 Aaa Bbb Ccc # Bbb Ccc Ddd # Aaa2 ccc Eee is at Ddd@Fff # 2 1 2 1 3	 50.0% 33.3%	 注意 Aaa2 被解析为 Aaa； Ccc 与 ccc 是相同的； is 和 at 被忽略； Ddd@Fff 被分解为 Ddd 和 Fff 两个词；

4.实验分析：

本题的关键难点在于快速找出两文件的公共词汇。一方面，在读文件内容时需要将单词解析出来，统计出有多少个不同的单词，保存为该文件对应的词汇表；另一方面，在求两文件的公共词汇表时，需要能快速的判断文件 A 中的某单词是否在文件 B 的词汇表中，也就是必须建立从单词找到文件的“倒序索引”。（由于本体不涉及删除，且建立充分大的散列表保证插入不会失败，所以采用了简化版散列表定义，省略了各种异常操作）。

参考代码：

```h
#pragma once
#ifndef __HASHTABLE_H
#define __HASHTABLE_H
#define MAXS 10   //最大字符串长度
#define MINS 3    //最小字符串长度
#define MAXB 5    //每个字符占得位数
#define MAXTable 500009   //散列表规模
typedef struct FileEntry {              //文件的词汇索引表
    int Words;
    struct FileEntry* Next;
}WList;
/*------------------简化版散列表定义及初始化------------------*/
typedef struct WordEntry {
    short FileNo;
    struct WordEntry* Next;
}FList;
typedef char ElementType[MAXS + 1];
typedef struct HashEntry {
    short FileNo;           //为0是结点为空
    FList* InvIndex;        //倒排索引
    ElementType Element;
}HashEntry;

class HashTable {
private:
    int TableSize;
    HashEntry* TheCells;
public:
    HashTable InitializeTable(int TableSize);             //创建散列表
    WList* InitializeFileIndex(int Size);                 //初始化文件的词汇索引表
    int GetAWord(ElementType Word);                       //从当前字符开始读到单词尾的第一个非字母符号为止
    int Hash(char* Key, int P);                           //字符串key移位法散列函数
    int Find(ElementType Key, HashTable* H);              //返回key的位置，或是适合插入key的位置
    int InsertAndIndex(int FileNo, ElementType Key, HashTable* H);        //将key插入散列表，同时插入倒排索引表
    void FileIndex(WList* File, int FileNo, int pos);      //将单词再散列中的位置pos存入文件FileNo对应的索引表
    double ComputeSim(WList* File, int F1, int F2, HashTable* H);         //计算文件F1和F2相似度
};
#endif //__HASHTABLE_H
```

图 1 头文件.h

```cpp
int HashTable::InsertAndIndex(int FileNo, ElementType Key, HashTable* H) {
    FList *F;
    int pos = Find(Key, H);    //找位置

    if (H->TheCells[pos].FileNo != FileNo) {    //插入散列表
        if (!H->TheCells[pos].FileNo)    //新单词
            strcpy_s(H->TheCells[pos].Element, Key);
        H->TheCells[pos].FileNo = FileNo;    //最近更新
        F = (WordEntry*)malloc(sizeof(struct WordEntry));  //插入倒排索引
        F->FileNo = FileNo;
        F->Next = H->TheCells[pos].InvIndex;
        H->TheCells[pos].InvIndex = F;
        return pos;
    }
    else
        return -1;   //同一文件的重复单词，不插入
}
```

图 2 InsertAndIndex 函数

```cpp
int HashTable::Hash(char* Key, int P) {   //字符串key移位法散列函数
    unsigned int h = 0;
    while (*Key != '\0')
        h = (h << MAXB) + (*Key++ - 'a');
    return h % P;
}
```

图 3 Hash 函数

```
void HashTable::FileIndex(WList* File, int FileNo, int pos) {
    WList* W;
    if (pos < 0)   //重复的单词不处理
        return;
    W = (FileEntry*)malloc(sizeof(struct FileEntry));
    W->Words = pos;
    W->Next = File[FileNo - 1].Next;
    File[FileNo - 1].Next = W;
    File[FileNo - 1].Words++;          //头结点累计词汇量
}
```

图 4 FileIndex 函数

```
输入文件总数：2
aaA,bbb,Ddd,
#
Aaa,aaa,vVv,
#
输入查询总数：1
输入查询的两文件：1 2
25.0%
```

图 5 示例输出

四、实验注意事项

1. 建议使用 c++语言的模板类实现；
2. 设计具有通用性、可复用性，代码可读性强；
3. 实验分析和设计要有详尽的描述；
4. 在完成基本功能的基本上可以自己扩展其他功能；
5. 同一文件内重复出现的单词不重复计算；
6. 这里的“单词”只包括由英文字母组成的、长度不小于 3、且不超过 10 的英文单词，太长的单词只考虑前 10 个字母；
7. 单词之间以任何非英文字母隔开，另外大小写不同的同一单词被认为是相同的单词。
8. 所给代码仅供参考，书写形式、习惯以依自己而定。

实验四 排序（2）

一、实验目的

1.掌握内部排序的思想和算法

2.能够对对象数组实现任意内部排序的算法

二、实验内容

【项目名称】

奖学金公示

【项目内容】

某班级要公示期末成绩前5名学生发奖学金。每个同学都有三门课成绩（数学、语文、英语）。首先按照总分排序，如果两个同学总分相同，再按数学成绩从高到低排序，如果两个同学总分和数学成绩都相同，那么规定学号小的同学排在前面。

任务：

先根据输入的3门课的成绩计算总分，然后按上述规则排序，最后按排名顺序输出前五名名学生的姓名、学号和总分。使用内部排序的任意一种方式，尽可能地缩短排序时间。

输入：

包含n+1行：（6<=n<=300）

第1行为一个正整数n，表示该校参加评选的学生人数。

第2到n+1行，每行有五个内容 分别为学生的姓名、学号以及语文、数学、英语的成绩。

输出：

共有5行，每行有三个内容，依次表示前5名学生的姓名、学号和总分。

样例输入：

6

Taylor 1 90 67 80

Beth 2 87 66 91

Simone 3 78 89 91

Jade 4 88 99 77

Rob 5 67 89 64

Tom 6 78 89 98

样例输出：

Tom 6 265

Jade 4 264

Simone 3 258

Beth 2 244

Taylor 1 237

三、实验实现

基于以下 Student 类的定义补全类函数，实现本实验：

```
class Student
{
private:
    string _name;
    int _number;
    int _chinese;
    int _math;
    int _english;
    int _sum;
public:
    Student() {}
    void putMessage(string name,int number,int chinese,int math,int english);
    ~Student() {}
    void printStudent();
    int stu_Chinese();
    string stu_name();
    int stu_english();
    int stu_number();
    int stu_math();
    int stu_sum();
    void operator=(Student & s);
};
```

四、实验注意事项

1. 建议使用 c++语言的模板类实现；

2. 设计具有通用性、可复用性，代码可读性强；

3. 实验分析和设计要有详尽的描述；

4. 在完成基本功能的基本上可以自己扩展其他功能。