附 录

实 验 指 导

人类用认识的活动去了解事物，用实践的活动去改变事物；用前者去掌握宇宙，用后者去创造宇宙。

——克罗齐（意）

1．实验目的及要求

数据结构是一门实践性很强的课程，只靠看书和做习题是不能提高实践能力的。数据结构的实验是一种自主性很强的学习过程，是对读者的一种全面的综合训练，是与课堂听讲、自学和练习相辅相成的、必不可少的教学环节。与程序设计语言课程中的实验不同，数据结构课程中的实验多属创造性的活动，需要读者自己分析问题，设计数据结构和算法，再上机调试和测试程序。数据结构的实验主要有两个教学目的：第一是深化理解和掌握书本上的理论知识，将书本上的知识变“活”，达到深化理解和灵活掌握教学内容的目的；第二是理论和实践相结合，使读者学会如何把书本中有关数据结构和算法的知识用于解决实际问题，培养数据结构的应用能力和软件工程所需要的实践能力，同时综合性实验鼓励同学们多人合作，培养他们的团队合作精神。通过实验，要求在数据结构的选择和应用、算法的设计及实现等方面加深对课程基础内容的理解，同时，在程序设计方法以及上机操作等基本技能方面受到比较系统和严格的训练。

为达到上述目的，本书安排了如下3类实验。

- 验证性实验：其主要内容是将书上的重要数据结构进行上机实现，深化理解和掌握理论知识，这部分实验不需要读者自己设计，只需将给定的方案实现即可。
- 设计性实验：其主要内容是针对具体问题，应用某一个知识点，自己设计方案，并上机实现，目的是培养读者对数据结构的简单应用能力。
- 综合性实验：其主要内容是针对具体问题，应用某几个知识点，自己设计方案，并上机实现，目的是培养读者对数据结构的综合应用能力。

这3类实验都是由问题描述、基本要求、测试数据、实现提示、思考题等几部分组成。问题描述为读者建立问题的背景环境，指明“问题是什么”；基本要求是对问题的实现进行基本规范，保证预定的训练意图，使某些难点和重点不会被绕过去，而且也便于教学检查；测试数据给出了实验中的基本测试用例；实现提示给出了设计数据结构和算法的主要思路；思考题引导读者在做完实验后进行总结和扩充。

虽然在设计性实验和综合性实验中都给出了一定的设计方案，但是，读者不应拘泥于这些分析和设计，要尽量发挥想象力和创造力。对于一个实际问题，每个人可能会给出不同的解决办法，本书给出的设计方案只是给读者提供解题思路引人正轨，提出思考问题的方法，但不会限制读者的思维，我们鼓励读者自己设计解决方案。

实验有以下几个基本要求。

(1) 由指导老师讲授实验的基本原理、基本操作方法。

(2) 及时记录并处理数据。

(3) 每次实验前要作好充分准备，书写预习报告。每次实验从阅读实验指导书开始。对于每次实验的实验目的、实验题目、实现提示以及思考题目、选做题目等应有详细的了解，理解实验要求，认真完成各算法并充分估计可能出现的错误。

(4) 实验时要求严肃认真，注意观察并记录各种错误现象，纠正错误，使程序满足预定的要求，实验记录应作为实验报告的一部分。

(5) 实验后要及时总结，写出实验报告。报告应分析实验结果，讨论实验中的问题，并附上所打印的问题解答、程序清单、所输入的数据及相应的运行结果。

2. 实验步骤

2.1 验证性实验的一般步骤

验证性实验安排的内容在书上都能找到具体的实现方法。这些验证性实验是读者在学习完一种数据结构后进行的，对于深化理解和掌握相应数据结构具有很重要的意义。

1. 预备知识的学习

由于篇幅所限，本书没有整理验证性实验所用到的预备知识，但书中对此已做了清晰陈述，需要读者在实验前复习实验所用的预备知识。这要求读者有自主的学习意识和整理知识的能力。

2. 上机前的准备

写出实验预习报告，将实现提示中给出的数据类型和算法转换为对应的程序，并进行静态检查，尽量减少语法错误和逻辑错误。

很多读者在上机时只带一本数据结构书或实验指导书，而书上只有算法设计的代码而没有实验程序，于是就直接在键盘上输入程序，结果不仅程序的输入速度慢，而且编译后出现很多错误。上机前的充分准备能高效利用上机时间，在有限的时间内完成更多的实验内容。

3. 上机调试和测试程序

调试程序是一个辛苦但充满乐趣的过程，也是培养程序员素质的重要环节。很多读者都有这样的经历：花了好长时间去调试程序，错误却越改越多。究其原因，一方面是对调试工具不熟悉，出现了错误提示却不知道这种错误是如何产生的；另一方面是没有认识到努力预先避免错误的重要性，也就是对程序进行静态检查不够。

对程序进行测试，首先需设计测试数据。在数据结构中测试数据需考虑以下两种情况：第一是一般情况，例如循环的中间数据、随机产生的数据等；第二是特殊情况，例如循环的边界条件、数据结构的边界条件等。

4. 实验报告

在实验后要总结和整理实验报告。书写实验报告的基本要求见后面的“实验报告”部分。

2.2 设计性实验和综合性实验的一般步骤

随着计算机性能的提高，软件开发的复杂度也日趋增加，因此软件开发需要系统的方法。虽然数据结构课程中实验题的复杂度远不及实际中的软件系统，但为了培养一个软件工作者所应具备的科学的工作方法和作风，我们制订了完成实验的5个步骤。

1. 问题分析

实验题目通常叙述比较简洁，因此，在进行设计之前，首先应该充分地分析和理解问题，明确问题要求做什么，限制条件是什么。注意，这里强调的是做什么，而不是怎么做。对问题的描述应避开算法和所涉及的数据类型，要对所需完成的任务作出明确的回答。例如，输入数据的类型、值的范围

以及输入的形式，输出数据的类型、值的范围及输出的形式，哪些属于非法输入等。在问题分析时还应该准备好充分的测试数据。

2．系统设计

此步骤分概要设计和详细设计两步实现。

概要设计指的是对问题描述中涉及的操作对象定义相应的数据类型，并按照以数据结构为中心的原则划分模块，定义主程序模块和各抽象数据类型。在这个过程中，要综合考虑系统的功能，使系统结构清晰、合理、简单。抽象数据类型尽可能做到数据封闭，基本操作的说明尽可能明确，而不必过早地考虑存储结构以及语言的实现细节，不必过早地表述辅助数据结构和局部变量。作为概要设计的结果，要写出每个抽象数据类型的定义（包括数据结构的描述和每个基本操作的规格说明），各个主要模块的算法，并画出模块之间的调用关系图。

详细设计是对数据结构和基本操作的规格说明进一步求精，定义相应的存储结构（即用C++描述抽象数据类型对应的类）以及算法所需的辅助数据结构，并按照算法书写规范写出各过程和函数的伪码算法或用C++语言写出过程或函数形式的算法框架。

此外，还要设计一定的用户界面。数据结构课程实验的主要目的是为了培养读者对数据结构的应用能力，因此在实验中不要求图形界面，只要在屏幕上提示用户每一步操作的输入，并将结果输出即可。但界面的设计和信息的交互是软件工程项目的一个十分重要的环节。

3．编码实现和静态检查

编码是把详细设计的结果进一步求精为C++语言程序。如何编写程序才能较快地完成调试是需要特别注意的问题。程序的每行不要超过60个字符。每个过程（函数）体一般不要超过40行，最长不得超过60行，否则应该分割成较小的过程（函数）。要控制`if`语句连续嵌套的深度，分支过多时应考虑使用`switch`语句，甚至可以考虑利用多形性。对函数功能和重要变量进行注释。一定要按格式书写程序，分清每条语句的层次，对齐括号，这样便于发现语法错误。

在上机之前，应该事先在纸上写出详细的程序编码，并认真做静态检查。多数初学者在编好程序后处于以下两种状态之一：一种是对自己精心设计的程序的正确性毫不怀疑；另一种是认为上机前的任务已经完成，纠查错误是上机的工作。这两种态度都是极为有害的。事实上，即使有几十年经验的高级软件工程师，也经常利用静态检查来查找程序中的错误。对一般的程序设计者而言，当编写的程序长度超过50行时，通常会含有语法错误或逻辑错误。上机动态调试决不能代替静态检查，否则调试效率将是极低的。静态检查主要有两种方法：一是用一组测试数据手工执行程序（通常应先检查单个模块）；二是通过阅读或给别人讲解自己的程序而深入全面地理解程序逻辑，在这个过程中再加入一些注释和断言。如果程序中逻辑概念清楚，后者将比前者有效。

4．上机准备和上机调试

上机准备包括以下几个方面。

(1) 熟悉C++语言用户手册或程序设计指导书。

(2) 注意语言集成开发环境与标准C++语言之间的差别。

(3) 熟悉机器的操作系统和语言集成开发环境的用户手册，尤其是最常用的命令操作，以便顺利进行上机的基本活动。

(4) 掌握调试工具，考虑调试方案，设计测试数据并手工得出正确结果，读者应该熟练运用高级语言的程序调试器调试程序。

上机调试程序时要带一本高级编程语言教材或手册。调试最好分模块进行，自底向上，即先调试低层过程或函数。必要时可以另写一个调用驱动程序。这种看似麻烦的工作实际上可以大大降低调试的难度，提高调试工作效率。

调试程序应该分步骤、分层次进行。程序由简到繁，规模由小到大，数据量由少到多，逐步完成。比如，一个类可能有许多函数，建议首先仅仅调试类的构造函数和输入/输出函数，这一步比较简单。

即便如此，实验数据的规模也要从少量几个开始（1～3个）。程序调通之后，再用相对多的测试数据进行实验。此时，还可以排除一些错误。通过这一阶段，可以排除数据结构设计、构造函数和输入/输出函数设计的错误。

然后，再把体现重要算法的函数加入到源程序之中，这包括类代码中的函数原型声明、函数实现的程序代码以及函数调用语句等。此时，实验数据规模也从少量几个开始，以便检查算法设计是否正确。程序基本调通之后，再用大量数据进行实验。本阶段还可进一步排除一些错误。但是，出现错误往往位于新加入的代码段之中。

调试过程中要借助程序调试器的各种功能，提高调试效率。调试中遇到的各种异常现象往往是预料不到的，此时不应“苦思冥想”，而应借助系统提供的调试工具确定错误。调试正确后，认真整理源程序及其注释，打印出带有完整注释且格式良好的源程序清单和结果。

5．总结和整理实验报告

在上机开始之前要充分准备实验数据，在上机实践过程中要及时记录实验数据，在上机实践完成之后必须及时总结分析，写出实验报告。

2.3 数据结构程序设计风格和注释要求

1．文档级的代码要求

每个源文件和头文件都必须在文件首部的注释中注明设计者姓名、项目名（即上机题目名）、创建日期和最近一次修改日期。包含main()函数的源文件必须在首部注释后另加一段注释，简要描述程序的目的和用到的主要数据结构。文件注释格式如下。

文件名称:
项目名称:
创建者:
创建时间:
最后修改时间:
功能:
文件中的函数名称和简单功能描述:
文件中定义的全局变量和简单功能描述:
文件中用到的他处定义的全局变量及其出处:
与其他文件的依赖关系:

每个类必须包含首部注释块，适度地描述这个类的目的。类的声明（一般在头文件里）应该紧随类的首部注释。类的首部注释格式如下。

类名称:
定义该类的目的:
类属性:
类中函数及功能:
与其他类的关系（调用/被调用哪类对象中的什么函数）:

每个函数必须有首部注释块，描述该函数的简要功能，每个参数的逻辑含义（包括它是输入还是输出或者输入/输出），函数调用之前的预备条件，返回后的处理，返回值（如果存在），该函数要调用到的函数列表（如果存在）。这些函数首部注释可能和函数原型或函数实现放在一起。应该注意，这项要求不仅适用于单独的函数，同样也适用于类的成员函数。函数的首部注释格式如下。

函数名称:
函数功能描述:
函数调用之前的预备条件:

返回后的处理:

返回值（如果存在）:

函数的输入参数:

函数的输出参数:

函数的抽象算法（伪码）:

函数与其他对象中函数的调用和被调用关系:

所有局部变量或常量的声明后应该简要说明它们的含义和用途。

主要的控制结构，如循环或分支结构，应该在前面注明其后代码将要完成什么功能。

采用清晰一致的缩进格式和其他格式化风格（如括号的定位）来提高代码可读性。

2. 算法级的代码要求

在编写算法或者程序时，要注意下列几个要求。

(1) 标识符名称（常量、变量、函数、类等）应该具有描述性，便于理解。

(2) 要用到某个常数时，最好设置一个常量来代替这个数字。

(3) 采用枚举类型来表示内部标签和状态的分类。

(4) 任何情况下都不要用全局或文件范围变量，但是允许采用全局范围内的类型定义（包括类定义）。

(5) 采用适当的方式传递函数参数。当被调用函数需要修改实参的值时，一般只采用引用传参或指针传参。当传递结构较大且不能被修改的参数时，采用常量引用传参或者常量指针传参。当被调用函数只需改变形参（调用内部）而保持实参不变的时候，采用传值传参。

(6) 采用String对象来存储字符串数据（除了单个字符），而不用字符数组存储。

(7) 任何时候都采用新式的C++。例如，采用<iostream>代替<iostream.h>。不要混合使用旧式C++和新式C++。

(8) 采用I/O流代替C风格的I/O。

在编写输入和输出程序时，对于输入和输出应考虑以下原则。

(1) 输入操作步骤和输入格式尽量简单。

(2) 应检查输入数据的合法性、有效性，报告必要的输入状态信息及错误信息。

(3) 输入一批数据时，使用数据或文件结束标志，而不要用计数来控制。

(4) 交互式输入时，提供可用的选择和边界值。

(5) 当程序设计语言有严格的格式要求时，应保持输入格式的一致性。

(6) 输出数据表格化、图形化。

(7) 输入和输出风格还受其他因素（如输入设备、输出设备，用户经验及通信环境等）的影响。

3. 面向对象级的代码要求

(1) 尽量采用类，不要用成员函数来实现结构类型。

(2) 建议采用类模板来表示容器型结构，如列表、树等，以提高可复用性。

(3) 设计类时，让每个类都拥有连贯的响应度。

(4) 类的所有数据成员都应该是私有的。

(5) 很多情况下，类的某些成员函数应该也是私有的，但要视情况而定。

(6) 如果一个类数据成员是一个指向动态分配内存的指针，要求写出析构函数来释放内存。

(7) 仅当有必要时才采用继承机制。

(8) 尽量少使用库中的类（如STL的类）。如果要编图形界面，请尽量把与编译环境（如Visual C++、C++ Builder）有关的类限制在少数几个文件中。也就是说，尽量把算法部分和界面部分的源程序分割开来。当然，String类例外，大多数情况下可以用它来替代char *类型。

3. 实验考核方式

实验的成绩评定建议以上机情况、程序质量、实验报告相结合的形式综合评分。满分为100分，建议实验成绩占本课程成绩的10%~15%。

(1) 上机情况（占15%）。包括出勤、实验过程的表现情况。

(2) 预习报告（占15%）。要求写明实验目的、实验内容、问题分析和主要算法的设计。

(3) 程序质量（占50%）。总体来说，所设计的程序应该全部符合要求，问题模型、求解算法以及存储结构清晰；具有友好、清晰的界面，有足够的信息交互；设计要包括所需要的辅助程序，如必要的数据输入、输出、显示和错误检测功能；程序的整体结构及局部结构要合理；操作使用要简便。

(4) 实验报告（占20%）。使用说明（书）要清晰，设计报告要符合规范。

4. 实验报告

4.1 实验报告规范

实验报告一般包括封面、正文两部分组成。实验报告的封面应给出题目、班级、姓名、学号和完成日期，正文包括以下7个内容。

(1) 需求分析。以无歧义的陈述说明程序设计的任务，强调程序要做什么，并明确规定：

- 输入的形式和输入值的范围；
- 输出的形式；
- 程序所能达到的功能；
- 测试数据，包括正确的输入及其输出结果，以及错误的输入及其输出结果。

(2) 概要设计。说明本程序中用到的所有抽象数据类型的定义、主程序的流程以及各程序模块之间的层次（调用）关系。

(3) 详细设计。实现概要设计中定义的所有数据类型，对每个操作只需要写出伪码算法，对主程序和其他模块也都需要写出伪码算法（伪码算法达到的详细程度建议为：伪码算法可以在计算机键盘直接输入高级编程语言程序），画出函数和过程的调用关系图。

(4) 调试分析。调试分析是评价实验的主要依据。应该有下列几部分内容。

- 记录所输入的原始测试数据（有些原始数据还需要画出图示），记录运行输出结果，还可以含其他测试数据及其运行输出（有时需要多组数据）。这里的记录要遵循实况，且要求完整。
- 调试过程中遇到的主要问题是如何解决的，以及对设计和编码的讨论和分析。
- 时间和空间分析。
- 改进设想。
- 经验、心得和体会等。这里的归纳要简明扼要，避免烦琐。

实验者必须重视这几个环节，否则等同于没有完成实验任务。这里可以体现个人特色或创造性思维。

(5) 使用说明。说明如何使用你编写的程序，详细列出每一步的操作步骤。

(6) 测试结果。列出你的测试结果，包括输入和输出。这里的测试数据应该完整、严格，基于并且多于需求分析中所列的测试数据。

(7) 附录。提交带注释的源程序。如果提交源程序软盘或源文件，可以只列出程序文件名的清单。

值得注意的是，实验报告的各种文档资料，如上述的前3部分要在程序开发过程中逐渐充实形成，而不是最后补写。

4.2　实验参考题目

实验一　线性表

本实验的主要目的在于使读者熟悉线性表的基本操作在两种存储结构上的实现，其中以熟悉各种链表的操作为重点。本实验还可帮助读者复习高级编程语言的使用方法。

1．单向链表操作的实现（验证性实验）

问题描述

(1) 用头插法（或尾插法）建立带头结点的单向链表。

(2) 对已建立的单向链表实现插入、删除、查找等基本操作。

2．城市链表（设计性实验）

问题描述

将若干城市的信息存入一个带头结点的单向链表。结点中的城市信息包括城市名、城市的位置坐标。要求能够利用城市名和位置坐标进行有关查找、插入、删除、更新等操作。

基本要求

(1) 给定一个城市名，返回其位置坐标。

(2) 给定一个位置坐标P和一个距离D，返回所有与P的距离小于等于D的城市。

测试数据

由读者依据软件工程的测试技术自己确定。注意测试边界数据。

3．约瑟夫环（综合性实验）

问题描述

约瑟夫问题的一种描述是，编号为1, 2, …, n的n个人按顺时针方向围坐一圈，每人持有一个密码（正整数）。一开始任选一个正整数作为报数上限值m，从第一个人开始按顺时针方向自1开始顺序报数，报到m时停止报数。报m的人出列，将他的密码作为新的m值，位于他顺时针方向上的下一个人开始重新从1报数，如此下去，直至所有人全部出列。试设计一个程序求出出列顺序。

基本要求

利用单向循环链表存储结构模拟此过程，按照出列的顺序打印出每个人的编号。

测试数据

m的初值为20，密码分别为3、1、7、2、4、8和4（正确的结果应为6、1、4、7、2、3和5）。

实现提示

程序运行后首先要求用户指定初始报数上限值，然后读取每个人的密码（设n≤30）。

思考题

若m的值依据每个出列人手中所持的数而改变呢？

4．长整数运算（综合性实验）

问题描述

设计一个程序实现两个任意长的整数求和运算。

基本要求

利用双向循环链表实现长整数的存储，每个结点含一个整型变量。任何整型变量的范围是$-(2^{15}-1)$～$(2^{15}-1)$。输入和输出形式：每4位一组，组间用逗号隔开。

测试数据

(1) 0和0，应输出“0”。

(2) –2345,6789，–7654,3211，应输出“–1,0000,0000”。

(3) –9999,9999，1,0000,0000,0000，应输出“9999,0000,0001”。

(4) 1,0001,000，–1,0001,0001，应输出“0”。

(5) 1,0001,0001，–1,0001,0000，应输出“1”。

实现提示

(1) 每个结点中可以存放的最大整数为2^{15}–1＝32 767，这样才能保证两数相加不会溢出。但若按32 768进制数存放，在十进制数与32768进制数之间的转换十分不方便，故可以在每个结点中仅存放十进制数的4位，即不超过9 999的非负整数，整个链表视为万进制数。

(2) 可以利用头结点数据域的符号代表长整数的符号。用其绝对值表示元素结点数目。相加过程中不要破坏两个操作数链表。两操作数的头指针存于指针数组中是简化程序结构的一种方法。不能给长整数位数规定上限。

实验二 栈、队列与递归算法设计

本实验的目的在于使读者深入了解栈和队列的特征，以便在实际问题中灵活运用它们。同时还将巩固这两种结构的构造方法，接触较复杂问题的递归算法设计。

1．栈和队列操作的实现（验证性实验）

问题描述

(1) 建立一个顺序栈。

(2) 建立一个循环顺序队列。

(3) 分别实现栈和队列的基本操作。

2．数制转换问题（设计性实验）

问题描述

十进制数*n*和其他*d*进制数的转换是计算机实现计算的基本问题，其解决方案很多，其中最简单的方法基于下列原理，即除*d*取余法。例如，$(1348)_{10}=(2504)_8$。

N	*n* div 8	*n* mod 8
1348	168	4
168	21	0
21	2	5
2	0	2

从中可以看出，最先产生的余数4是转换结果的最低位，这正好符合栈先进后出的特性，所以可以用顺序栈来模拟这个过程。

基本要求

从键盘输入任意一个非负的十进制整数，打印输出与其等值的八进制数。由于上述的计算过程是从低位到高位顺序产生的八进制数的各个数位，而打印输出一般来说应从高位到低位进行，恰好与计算过程相反。因此，可以先将计算过程中得到的八进制数的各位进栈，待相对应的八进制数的各位均产生以后，再使其按顺序出栈，并打印输出。这样就得到了与输入的十进制数相对应的八进制数。

测试数据

由读者依据软件工程的测试技术自己确定。注意测试边界数据。

3．商品货架管理（设计性实验）

问题描述

商品货架可以看成一个栈，栈顶商品的生产日期最早，栈底商品的生产日期最晚。上货时，需要倒货架，以保证生产日期较晚的商品在货架较下的位置。

基本要求

针对一种特定商品，实现上述管理过程。

实现提示

用栈模拟货架和周转空间。

测试数据

由读者依据软件工程的测试技术自己确定。注意测试边界数据，如空栈。

思考题

怎样将转换后的序列形成相应进制的数据？

4. 括号匹配的检验（设计性实验）

问题描述

假设表达式中允许有两种括号，即圆括号和方括号，其嵌套的顺序随意，即(()[ ])或[([ ] [ ])]等均为正确格式，[(])或(((]均为错误格式。检验括号是否匹配的方法可用“期待的紧迫程度”这个概念来描述。例如，考虑下列的括号序列：

```
[  (  [  ]  [  ]  )  ]
1  2  3  4  5  6  7  8
```

当计算机接受了第1个括号“[”以后，它期待着与其匹配的第8个括号“]”的出现，然而等来的却是第2个括号，此时第1个括号“[”只能暂时靠边，而迫切等待与第2个括号相匹配的第7个括号“]”的出现。类似地，因只等来了第3个括号“[”，此时，其期待的紧迫程度比第2个括号更紧迫，则第2个括号只能靠边，让位于第3个括号，显然第3个括号的期待紧迫程度高于第2个括号，而第2个括号的期待紧迫程度高于第1个括号。在接受了第4个括号“]”之后，第3个括号的期待得到了满足，消解之后，第2个括号的期待匹配就成了最急迫的任务了，依次类推。可见这个处理过程正好和栈的特点相吻合。

基本要求

读入圆括号和方括号的任意序列，输出“匹配”或“此串括号匹配不合法”。

测试数据

输入([ ]())，结果“匹配”。

输入[[( )]，结果“此串括号匹配不合法”。

实现提示

设置一个栈，每读入一个括号，若是左括号，则作为一个新的更急迫的期待压入栈中；若是右括号，并且与当前栈顶的左括号相匹配，则将当前栈顶的左括号退出，继续读取下一个括号，如果读入的右括号与当前栈顶的左括号不匹配，则属于不合法的情况。在初始和结束时，栈应该是空的。

思考题

如何考虑括号的优先级呢？

5. 停车场管理（综合性实验）

问题描述

设停车场内是一个可停放n辆汽车的狭长区域，且只有一个大门可供汽车进出。在停车场内，汽车按到达时间的先后顺序，依次由北向南排列（假设大门在最南端，最先到达的那辆车停放在车场的最北端）。若车场内已停满n辆汽车，则后来的汽车只能在门外的便道上等候。一旦有车开走，则排在便道上的第一辆车即可开入。当停车场内某辆车要离开时，在它之后开入的车辆必须先退出车场为它让路，待该车开出大门外，其他车再按原次序返回停车场。每辆停放在车场的车离开停车场时，必须按其停留时间的长短交费。试为停车场编制按上述要求进行管理的模拟程序。

测试数据

设$n=2$，输入数据为(A, 1, 5)、(A, 2, 10)、(D, 1, 15)、(A, 3, 20)、(A, 4, 25)、(A, 5, 30)、(D, 2, 35)、(D, 4, 40)和(E, 0, 0)。每一组输入数据包括3个数据项：汽车“到达”或“离去”信息、汽车牌照号码及到达或离去的时刻，其中，“A”表示到达，“D”表示离去，“E”表示输入结束。

基本要求

以顺序栈模拟停车场，以链队列模拟便道，按照从终端读入的输入数据序列进行模拟管理。每一组输入数据包括3项：是“到达”还是“离去”、汽车牌照号码及“到达”或“离去”的时刻。对每一组输入数据进行操作后的输出数据为：如果是到达的车辆，则输出其在停车场内或便道上的停车位置；如果是离去的车辆，则输出其在停车场内的停留时间和应交的费用（在便道上停留的时间不收费）。栈以顺序结构实现，队列以链表实现。

实现提示

需另设一个栈，临时停放为给要离去的汽车让路而从停车场退出来的汽车，也用顺序存储结构实现。输入数据按到达或离去的时刻有序。栈中每个元素表示一辆汽车，包含两个数据项：汽车的牌照号码和进入停车场的时刻。

选作内容

(1) 两个栈共享空间，思考应开辟数组的空间是多少？

(2) 汽车可有不同种类，则它们的占地面积不同，收费标准也不同，如1辆客车和1.5辆小汽车的占地面积相同，1辆十轮卡车占地面积相当于3辆小汽车的占地面积。

(3) 汽车可以直接从便道上开走，此时排在它前面的汽车要先开走让路，然后再依次排到队尾。

(4) 停放在便道上的汽车也收费，收费标准比停放在停车场的车低，请思考如何修改结构以满足这种要求。

实验三 串及其应用

本实验的目的是使读者熟悉串类型的实现方法和文本模式匹配方法，熟悉一般文字处理软件的设计方法，较复杂问题的分解求精方法，在实验二的基础上，进一步强化这样一个观念：程序是数据结构结合定义在其上的操作。此外还希望起到训练合作能力和熟悉文件操作的目的。本实验的难度较大。

1．文学研究助手（设计性实验）

问题描述

文学研究人员需要统计某篇英文小说中某些形容词的出现次数和位置。试写一个实现这一目标的文字统计系统，称为“文学研究助手”。

基本要求

英文小说存于一个文本文件中。待统计的词汇集合要一次输入完毕，即统计工作必须在程序的一次运行之后就全部完成。程序的输出结果是每个词的出现次数和出现位置的行号，格式自行设计。

测试数据

以源程序模拟英文小说，编程语言保留字集作为待统计的词汇集。

实现提示

设小说非空且以文件形式存放，其中的词汇一律不跨行。这样，每读入一行，就统计每个词在这行中的出现次数和出现位置的行号，后者可以用链表存储。若某行中出现了不止一次，不必存多个相同的行号。数据结构采用二维链表，单词结点链接成一个链表，每个单词的行号组成一个链表，单词结点作为行号链表的头结点。

选作内容

(1) 模式匹配要基于KMP算法。

(2) 整个统计过程中只对小说文字扫描一遍以提高效率。

(3) 假设小说中的每个单词或者从行首开始，或者前置以一个空格符。利用单词匹配特点另写一个高效的统计程序，与KMP算法统计程序进行效率比较。

(4) 推广到更一般的模式匹配问题，并设待查模式串可以跨行。

思考题

怎样考虑分词问题？怎样考虑多模式匹配问题？

2．简单行编辑程序（综合性实验）

问题描述

文本编辑程序是利用计算机进行文字加工的基本软件工具，实现对文本文件的插入、删除等修改操作。限制这些操作以行为单位进行的编辑程序称为行编辑程序。

被编辑的文本文件可能很大，全部读入编辑程序的数据空间（内存）的做法既不经济也不总能实现。一种解决方法是逐段地编辑，任何时刻只把待编辑文件的一段放在内存，称为活区。试按照这种方法实现一个简单的行编辑程序。设文件每行不超过320个字符，并且很少超过80字符。

基本要求

实现以下4条基本编辑命令。

(1) 行插入。格式：i<行号><回车><文本><回车>。将<文本>插入活区中第<行号>行之后。

(2) 行删除。格式：d<行号1>[□<行号2>]<回车>。删除活区中第<行号1>行（到第<行号2>行）。两种格式的例子如“d10↙”和“d10□14↙”。

(3) 活区切换。格式：n<回车>。将活区写入输出文件，并从输入文件中读入下一段，作为新的活区。

(4) 活区显示。格式：p<回车>。逐页地（每页20行）显示活区内容，每显示一页之后请用户决定是否继续显示以后各页（如果存在）。印出的每一行要包含该行的行号和一个空格符，行号固定占4位，增量为1。

各条命令中的行号均须在活区中各行行号范围之内，只有插入命令的行号可以等于活区第一行行号减1，表示插入当前屏幕中第一行之前，否则命令参数非法。

测试数据

由读者依据软件工程的测试技术自己确定。注意测试边界数据，如首行、尾行。

实现提示

(1) 设活区的大小用行数activemaxlen（可设为100）来描述。考虑到文本文件行长通常为正态分布，且峰值为60～70，用320×activemaxlen大小的字符数组实现存储将造成大量浪费。可以以标准行块为单位为各行分配存储，每个标准行块含81个字符。这些行块可以组成一个数组，也可以利用动态链表连接起来。一行文字可能占多个行块。行尾可用一个特殊的ASCII字符（如012）标识。此外，还应记住活区起始行号，因为行插入将引起随后各行行号的顺序下推。

(2) 初始化过程包括请用户提供输入文件名（空串表示无输入文件）和输出文件名，两者不能相同。然后尽可能多地从输入文件中读入各行，但不超过activemaxlen-x。x的值可以自定，例如20。

(3) 在执行行插入命令的过程中，每接收到一行时要检查活区大小是否已达到activemaxlen。如果是，则为了在插入这一行之后仍保持活区大小不超过activemaxlen，应将插入点之前活区部分的第一行输出到输出文件中。若插入点为第一行之前，则只得将新插入的这一行输出。

(4) 若输入文件尚未读完，活区切换命令可将原活区中最后几行留在活区顶部，以保持阅读连续性；否则，它意味着结束编辑或开始编辑另一个文件。

(5) 可令前3条命令执行后自动调用活区显示。

选作内容

(1) 对于命令格式非法等一切错误做严格检查和适当处理。

(2) 加入更复杂的编辑操作，如对某行进行串替换、在活区内进行模式匹配等，格式可以分别为S<行号>@<串1>@<串2><回车>和m<串><回车>。

实验四 树结构及其应用

本实验继续突出了数据结构加操作的程序设计观点，但根据树结构的非线性特点，将操作进一步集中在遍历操作上，因为遍历操作是其他众多操作的基础。本实验还希望达到使读者熟悉各种存储结构的特征，以及掌握如何应用树解决具体问题（即原理与应用的结合）等目的。

1．二叉树的建立与遍历（验证性实验）

问题描述

建立一棵二叉树，并对其进行遍历（先序、中序和后序），打印输出遍历结果。

基本要求

从键盘接受输入（先序），以二叉链表作为存储结构，建立二叉树（以先序来建立），并采用递归算法对其进行遍历（先序、中序和后序），将遍历结果打印输出。

测试数据

ABC□□*DE*□*G*□□*F*□□□（其中□表示空格字符），则输出结果为：先序为*ABCDEGF*，中序为*CBEGDFA*，

- 后序为*CGBFDBA*。

2. 计算机目录树的基本操作（设计性实验）

问题描述

对计算机中的目录树实现建立目录、修改目录结构、查询和删除等操作。

基本要求

(1) 按二叉链表的存储方式存储计算机中的目录树。

(2) 实现目录树的建立、遍历及插入结点和删除结点操作。

测试数据

由读者依据软件工程的测试技术自己确定。注意测试边界数据。

实现提示

(1) 根据树与二叉树的对应关系，对树的操作最终要借助对二叉树进行操作来实现。

(2) 查询操作可以实现是否存在某目录或查询指定目录的父目录或子目录。

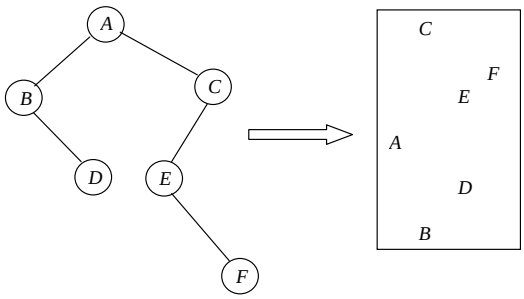
思考题

如果引入Windows系统中的快捷方式，该如何处理？

3．打印二叉树结构（设计性实验）

问题描述

按凹入表形式横向打印二叉树结构，即二叉树的根在屏幕的最左边，二叉树的左子树在屏幕的下边，二叉树的右子树在屏幕的上边。例如：



测试数据

由读者依据软件工程的测试技术自己确定。注意测试边界数据，如空二叉树。

实现提示

(1) 利用中序遍历方法。

(2) 利用结点的深度控制横向位置。

4．赫夫曼编码（综合性实验）

问题描述

设某编码系统共有n个字符，使用频率分别为$w_1, w_2, \cdots, w_n$，设计一个不等长的编码方案，使得该编码系统的空间效率最好。

基本要求

一个完整的系统应具有以下功能。

(1) I：初始化（Initialization）。从终端读入字符集大小n，以及n个字符和n个权值，建立赫夫曼树，并将它存于文件hfmTree中。

(2) E：编码（Encoding）。利用已建好的赫夫曼树对文件ToBeTran中的正文进行编码，然后将结果存入文件CodeFile中。

(3) D：译码（Decoding）。利用已建好的赫夫曼树将文件CodeFile中的代码进行译码，结果存入文件TextFile中。

(4) P：打印代码文件（Print）。将文件CodeFile以紧凑格式显示在终端上，每行50个字符。同时将此字符形式的编码文件写入文件CodePrint中。

(5) T：打印赫夫曼树（Tree printing）。将已在内存中的赫夫曼树以直观的方式显示在终端上，同时将此字符形式的赫夫曼树写入文件TreePrint中。

测试数据

由读者依据软件工程的测试技术自己确定。注意测试边界数据。

实现提示

利用赫夫曼编码树求得最佳的编码方案。

(1) 文件CodeFile的基类型可以设为字节型。

(2) 用户界面可以设计为“菜单”方式，除显示上述功能符号外，还应显示“Q”（Quit），表示退出运行。请用户键入一个选择功能符。此功能执行完毕后再显示此菜单，直至某次用户选择了“E”为止。

(3) 在程序的一次执行过程中，第一次执行I、D或C命令之后，赫夫曼树已经在内存了，不必再读入。每次执行时不一定执行I命令，因为文件hfmTree可能早已建好。

实验五　图结构及其应用

图是一种比树和表要复杂得多的数据结构。在图形结构中，结点之间的关系可以是任意的。图中的任意结点之间的两个数据元素都可以相关。本实验希望读者理解图的数据结构，掌握图的邻接表存储结构，建立邻接表的算法，以及如何应用图解决具体问题（即原理与应用的结合）等。

1．从键盘输入的数据建立图，并进行深度优先搜索和广度优先搜索（验证性实验）

问题描述

很多涉及图上操作的算法都是以图的遍历操作为基础的。试编写一个程序，演示无向图的遍历操作。

在主程序中提供下列菜单：

(1)“1”代表图的建立；

(2)“2”代表深度优先遍历图；

(3)“3”代表广度优先遍历图；

(4)“0”代表结束。

基本要求

以邻接表为存储结构，实现连通无向图的深度优先和广度优先遍历。以用户指定的结点为起点，分别输出每种遍历下的结点访问序列和相应生成树的边集。

测试数据

由读者依据软件工程的测试技术自己确定。注意测试边界数据，如单个结点。

实现提示

设图的结点不超过30个，每个结点用一个编号表示（如果一个图有n个结点，则它们的编号分别为1, 2, …, n）。通过输入图的所有边输入一个图，每个边为一个数对，可以对边的输入顺序作出某种限制。注意，生成树的边是有向边，端点顺序不能颠倒。

2．利用最小生成树算法解决通信网的总造价最低问题（设计性实验）

问题描述

若在n个城市之间建通信网络，只需架设n–1条线路即可。如何以最低的经济代价建设这个通信网是一个网的最小生成树问题。

基本要求

以邻接表为存储结构，利用Prim算法或Kruskal算法求网的最小生成树。

测试数据

由读者依据软件工程的测试技术自己确定。注意测试边界数据，如单个结点。

实现提示

设通信线路一旦建立，必然是双向的。因此，构造最小生成树的网一定是无向网。为简单起见，图的顶点数不超过10，网中边的权值设置成小于100。

3. 教学计划编制问题（设计性实验）

问题描述

软件专业的读者要学习一系列课程，其中有些课程必须在其先修课完成后才能学习。

基本要求

假设每门课程的学习时间为一个学期，试为该专业的学生设计教学计划，使他们能在最短时间内修完这些课程。

测试数据

由读者依据软件工程的测试技术自己确定。注意测试边界数据，如单个结点。

实现提示

利用拓扑排序实现。

4．导游问题（综合性实验）

问题描述

给出一张某公园的导游图，游客通过终端询问可知：

(1) 从某一景点到另一个景点的最短路径；

(2) 游客从公园大门进入，选一条最佳路线，使游客可以不重复的游览各景点，最后回到出口。

基本要求

(1) 将导游图看作一张带权无向图，顶点表示公园的各个景点，边表示各景点之间的道路，边上的权值表示距离，选择适当的数据结构。

(2) 为游客提供图中任意景点相关信息的查询。

(3) 为游客提供任意两个景点之间最短的简单路径。

(4) 为游客选择最佳游览路径。

测试数据

由读者依据软件工程的测试技术自己确定。注意测试边界数据，如单个结点。

实现提示

以邻接表为存储结构，利用Dijkstra算法或Floyd算法求最短路径，利用搜索求最佳路径。

实验六　查找和排序

本实验旨在集中对几个专门的问题做较为深入的探讨和理解，不强调对某些特定编程技术的训练。

1．二叉排序树（设计性实验）

问题描述

从键盘读入一组数据，建立二叉排序树并对其进行查找、遍历、格式化打印等有关操作。

基本要求

建立二叉排序树并对其进行查找，包括成功和不成功两种情况，并给出查找长度。

测试数据

由读者依据软件工程的测试技术自己确定。注意测试边界数据。

选作内容

实现二叉排序树的插入和删除操作。

2．二分查找算法（设计性实验）

问题描述

从键盘读入一串整数和一个待查键，查找在该整数串中是否有这个待查键。如果有，就输出它在整数串中的位置；如果没有，输出–1。

基本要求

掌握二分查找算法。

测试数据

由读者依据软件工程的测试技术自己确定。注意测试边界数据，如单个结点。

实现提示

利用二分查找算法查找实现。

3．散列表设计（设计性实验）

问题描述

针对某个集体中的人名设计一个散列表，使得平均查找长度不超过R，并完成相应的建表和查表程序。

基本要求

假设人名为汉语拼音的形式。待填入散列表的人名共有30个，取平均查找长度的上限为2。散列函数用除留余数法构造，用线性探测法或链地址法处理冲突。

测试数据

取读者周围较熟悉的30个人名。

选作内容

(1) 利用本书介绍的几种散列函数构造方法中的某种，设计几个不同的散列函数，比较其地址冲突率（可以用更大的名字集合做实验）。

(2) 研究这30个人名的特点，努力找出一个散列函数，使得对于不同的拼音名一定不发生地址冲突。

(3) 在散列函数确定的前提下，尝试各种不同的处理冲突的方法，考察平均查找长度的变化和建好的散列表中键的聚集性。

思考题

考虑散列函数的设计，提高查找的性能。

4．简单个人电话号码查询系统（综合性实验）

问题描述

人们在日常生活中经常要查找某个人或某个单位的电话号码，本实验将实现一个简单的个人电话号码查询系统，根据用户输入的信息（如姓名等）进行快速查询。

基本要求

(1) 在外存上，用文件保存电话号码信息。

(2) 在内存中，设计数据结构存储电话号码信息。

(3) 提供查询功能：根据姓名实现快速查询。

(4) 提供其他维护功能，如插入、删除、修改等。

测试数据

由读者依据软件工程的测试技术自己确定。注意测试边界数据，如单个结点。

实现提示

由于要管理的电话号码信息较多，而且要在程序运行结束后仍然保存电话号码信息，所以电话号码信息采用文件的形式存放到外存中。在系统运行时，要将电话号码信息从文件调入内存来进行查找等操作。为了接收文件中的内容，要有一个数据结构与之对应，可以设计如下结构类型的数组来接收数据。

```
const int max=10;
struct TeleNumber {
    String name;// 姓名
    String phoneNumber,// 固定电话号码
    String mobileNumber,// 移动电话号码
    String email;// 电子邮箱
} Tele[max];
```

为了实现对电话号码的快速查询，可以将上述结构数组排序，以便应用二分查找，但是，在数组中实现插入和删除操作的代价较高。如果记录需频繁进行插入或删除操作，可以考虑采用二叉排序树组织电话号码信息，这样查找和维护都能获得较高的时间性能。更复杂地，需考虑该二叉排序树是否平衡，如何使之达到平衡。

实验七　内排序算法比较

本实验目的是使读者，掌握常用排序方法的基本思想，通过实验加深理解各种排序算法，通过实验掌握各种排序方法的时间复杂度分析，了解各种排序方法的优缺点及适用范围。

1．排序算法的实现（设计性实验）

问题描述

排序是计算机领域的一项重要技术，是程序设计中的一种重要运算。它的功能是将一个数据元素的任意序列重新排列成一个按键有序的序列。学习和研究各种排序方法是计算机工作者的一项重要工作课题。

基本要求

随机产生n个整数（依次为n赋值100、200、300、1 000和2 000），将其存于数组`A[1..n]`中。对主要算法（顺序查找、插入排序、归并排序、堆排序和快速排序）进行实验比较，计算出平均比较次数c_n、平均移动次数m_n及执行时间t_n。c_n和m_n由程序自动计算，t_n由系统时间计算。

对实验结果数据进行对比分析。

测试数据

由随机数生成器决定。

实现提示

(1) 设计一个驱动程序，其任务是，随机输入一组原始数据$A[1..n]$，对于查找还需准备一个键码值（可以随机产生，也可在$A[1..n]$中按索引号挑选若干有代表性的数据）。对每组原始数据，分别调用查找或排序算法过程。

(2) 为了自动计算c_n和m_n需要在每个算法过程中的适当位置插入计数操作。

手工计算每个算法的执行时间t_n时，为了减少计时误差，删除所插入的计数操作，并按n的大小确定一个K，对每个查找或排序算法过程反复调用K次，在调用开始前设置一个计时起点t_0，在K次调用执行后可打印信息。设计时的终点为t_1，则$(t_1-t_0)/K$便是算法的大致执行时间。

选作内容

对不同的输入表长做试验，观察含义相同的变量相对于表长的变化关系，还可以对稳定性做验证。

2．统计成绩（综合性实验）

问题描述

给出n个学生的m门考试的成绩表，每个学生的信息由学号、姓名及各科成绩组成。对学生的考试成绩进行有关统计，并打印统计表。

基本要求

(1) 按总数高低次序，打印出名次表，分数相同的为同一名次。

(2) 按名次打印出每个学生的学号、姓名、总分以及各科成绩。

测试数据

由读者依据软件工程的测试技术自己确定。注意测试边界数据。

选作内容

对各科成绩设置不同的权值。

4.3　实验报告样例

最后附上实验二的实验报告样例，其中实验报告的正文按实验题目分节描述。

★★★★大学

数据结构实验报告

题　　目 栈 与 队 列

学生姓名 _____

学　　号 _____

指导教师 _____

学　　院 _____

专业班级 _____

完成时间 _____

指导教师评定：_____　　签　　名：_____

顺序栈的实现和基本操作

1. 需求分析

栈（stack）的典型操作是入栈（push）和出栈（pop），前者将新元素压入栈中，后者弹出栈顶元素。栈只提供对栈顶元素的访问操作，由top完成。push、pop和top三者构成栈的最小功能接口。为方便使用，栈还应提供若干基本操作。这些操作可按使用型（或称只读访问型）操作和加工型操作进行分类。这些基本操作同时也包括了与C++语言实现相关（可能与栈的高级抽象无关）的基本操作。引用型包括判空（empty）、求栈的大小（size）和栈的容量（capacity），加工型包括清空栈（clear）等。

删除的内容: 使用

(1) 输入的形式和输入值的范围：程序是按数字键选择对应功能的，在各项功能中，只有入栈需要输入数据，输入的数据应为整数。

(2) 输出的形式：在顺序栈的初始化后显示初始化成功，在检查顺序栈是否为空时显示顺序栈是否为空的结果，在将顺序栈置空后显示顺序栈已置空，在入栈后显示输入的数据或栈满，在出栈后显示栈顶元素已出栈或栈为空，在取栈顶元素后显示栈顶元素或栈为空，在顺序输出栈中元素后依次显示栈中元素的值或栈为空。

(3) 程序所能达到的功能：完成顺序栈的初始化、检查栈是否为空、置空栈、入栈、出栈、取栈顶元素和输出栈中元素这些操作。

(4) 测试数据：

- 初始化后执行5次入栈操作，依次输入数据1、2、3、4和5；
- 选择检查栈是否为空，选择取栈顶元素，选择输出栈中元素；
- 执行3次出栈操作；
- 执行2次入栈操作，依次输入数据6、7；
- 选择取栈顶元素，选择输出栈中元素。

2. 概要设计

(1) 为了实现程序功能，需要定义顺序栈的抽象数据类型。

```
ADT Stack {
    数据对象 D={a_i|a_i∈Elemset, i=1, 2, …, n, n≥0}
    数据关系 R={<a_i-1, a_i>|a_i-1, a_i∈D, i=2, …, n}
    基本操作
        InitStack(&S)
            操作结果：构造了一个空栈
        DestroyStack(&S)
            初始条件：栈S已存在
            操作结果：栈S被销毁
        ClearStack(&S)
```

```
            初始条件：栈S已存在
            操作结果：将S清为空栈
        StackEmpty(S)
            初始条件：栈S已存在
            操作结果：如果栈S为空则返回1,栈S非空则返回0
        StacktLength(S)
            初始条件：栈S已存在
            操作结果：返回S的元素个数,即栈的长度
        SeqStackGetTop(S&e)
            初始条件：栈S已存在且非空
            操作结果：用e返回S的栈顶元素
        SeqStackPush(&S e)
            初始条件：栈S已存在
            操作结果：插入元素e为新的栈顶元素
        SeqStackPop(&S &e)
            初始条件：栈S已存在且非空
            操作结果：删除S的栈顶元素,并用e返回其值
}ADT Stack
```

(2) 本程序包含8个函数：

- 主函数main()；
- 初始化顺序栈Stack()；
- 检查顺序栈是否为空SeqStackEmpty()；
- 把S置为空栈ClearStack()；
- 把元素x压入栈SeqStackPush()；
- 弹出栈顶元素SeqStackPop()；
- 取栈顶元素SeqStackGetTop()；
- 输出顺序栈中元素SeqStackPrint()。

各函数之间的关系如图1所示。

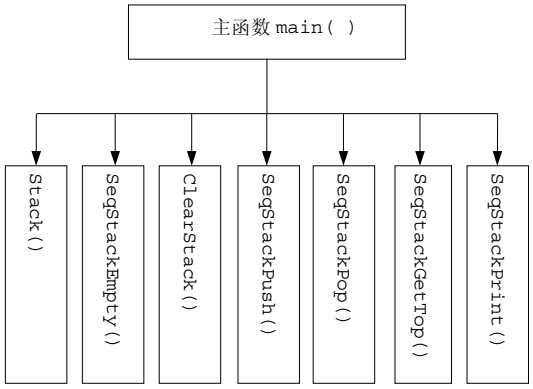


图1 函数模块及关系图

使用C++模板类实现栈。确定所需提供的可供用户及外部程序访问的公共接口的函数原型，以及相关实现细节部分。

- 构造函数、复制控制与析构函数。
- 基本操作的公共接口，包括引用型操作和修改型操作。
- 数据成员。使用3个指针保存栈的存储位置及状态。
- 辅助函数。将某些可能被复用的功能代码设计为辅助函数供其他函数调用，对于仅供内

删除的内容: 访问

部函数访问而不希望外部访问的辅助函数，应声明为私有函数。

3．详细设计

(1) 公共接口。根据需求分析的结果，进一步加入对C++实现细节的考虑。例如，SeqStackGetTop函数返回栈顶元素的值，在C++实现细节中，可进一步细分为const型只读访问和引用型可修改访问，这样该函数在C++类中将有两个实现：一个const型SeqStackGetTop函数和一个非const型SeqStackGetTop函数。另外，加上C++类设计中的正规函数（包括构造函数、析构函数和赋值函数等），得到顺序栈模板类的公共接口如下：

```
#include<assert.h>  // 引入逻辑断言语句
template<class ELEM  class Alloc=stdallocator<ELEM> >
class Stack {
public:
    ELEM*ElmList;  // 存放数据元素的指针变量
    Stack(): base(0), first_free(0), end(0) {}  // 初始化栈（默认构造函数）
    Stack(const Stack &x) { construct(x); }      // 复制构造函数
    Stack &operator=(const Stack &x);            // 赋值操作函数
    ~Stack()  // 析构函数,撤销栈实例
    int size() const  { return first_free-base; }     // 查询栈中元素个数
    int capacity() const  { return end-base; }        // 查询栈容量
    void SeqStackPush(ELEM item);  // 元素item进栈
    ELEM SeqStackPop() { alloc.destroy(--first_free); }       // 弹出栈顶元素
    const ELEM SeqStackGetTop() const;  // 取栈顶元素,但不出栈
    ELEM SeqStackGetTop();      // 取栈顶元素
    void MakeEmpty()            // 清空栈
    Boolean SeqStackEmpty()     // 判断栈是否为空,若为空栈,返回true,否则返回false
    Boolean SeqStackFull()      // 判断栈是否为满,若为满,返回true,否则返回false
private:
    static Alloc alloc;         // 内存分配器
    void reallocate();          // 扩充栈容量
    void construct(const Stack &x);
    void destroy();
    ELEM *base, *first_free, *end;
    ...
};
```

(2) 数据成员。“现代C++程序一般应该使用allocator类来分配内存，它更安全更灵活。”（引自《C++ Primer中文版第4版》）类似STL的容器类型设计，顺序栈包含两个模板参数，一个指定元素类型，一个可选参数（默认为stdallocator）设定内存配置器类型。类的数据成员包括一个内存配置器类型的static成员和3个指针成员base、first_free和end。base指向栈的内存空间的基地址；first_free实际是栈顶指针，指向当前可用的第一处内存空间，即新元素插入的栈顶位置；end指向栈所有可用空间的末端位置。end-base得到栈容量，比较end和first_free可知栈是否已满。以上数据成员属于类的实现细节，不应该被用户直接访问，因此均为私有成员。

(3) 辅助函数（私有函数成员）。在栈的赋值操作中，需先销毁旧栈，再创建新栈，这些操作包括复制构造函数和析构函数可复用的代码，所以栈的构造过程和销毁过程单独实现为辅助函数construct和destroy。向栈中添加元素时，如果栈满，已重新分配更多内存，这个过程通过辅助函数reallocate实现。以上3个函数都是类的内部实现细节，不希望用户访问这些函数，所以这3个函数均声明为私有函数成员。

(4) 主要操作的实现。

```
// 元素item入栈
template<class ELEM >
```

```
void Stack<ELEM >::SeqStackPush(ELEM item);
{
    if  (SeqStackFull());  // 栈满
        reallocate();      // 重新分配更多的内存空间, 扩充栈容量
    alloc.construct(first_free, elem);
        ++first_free;
}

// 复制构造新栈
template<class ELEM, class Alloc >
inline void Stack<ELEM, Alloc >::construct(const Stack &x)
{
    base = alloc.allocate(x.capacity());
    uninitialized_copy(x.base, x.first_free, base);
    first_free = base + x.size();
    end = base + x.capacity();
}
// 取栈顶元素
template<class ELEM >
ELEM Stack<ELEM >::SeqStackGetTop()
{
    assert(!SeqStackEmpty());  // 栈空, 溢出, 退出运行
    return *(first_free-1);    // 取出栈顶元素, 但不弹出
}
// 将栈清空
template<class ELEM >
void Stack<ELEM >::MakeEmpty()
{
    first_free = base;
}
// 判断栈是否已满
template<class ELEM >
Boolean Stack<ELEM >::SeqStackFull()
{
    return (first_free == end);     // 栈满时, 返回true
}
// 判断栈是否为空
template<class ELEM >
Boolean Stack<ELEM >: : SeqStackEmpty()
{
return (first_free == base);   // 栈空时, 返回true
}

// 销毁栈
template<class elem, class Alloc >
inline void Stack<elem, Alloc >::destroy()
{
    for (Tp *p = first_free; p != base; )  // 逆序析构原有元素
         alloc.destroy(--p);
    if (base)  // 不能使用0指针调用deallocate函数
       alloc.deallocate(base, capacity());
}

template<class elem, class Alloc >
inline Stack<elem, Alloc > &Stack<Tp, Alloc >::operator = (const Stack &x)
{
    destroy();         // 销毁旧栈
    construct(x);      // 构造新栈
    return *this;
}
```

```
template<class elem, class Alloc >
inline void Stack<elem, Alloc >::reallocate()
{
    int sz = size();
    int newcapacity = 2 * std :: max(sz, 1);          // 新容量大小
    Tp *newbase = alloc.allocate(newcapacity);        // 分配新的内存空间
    uninitialized_copy(base, first_free, newbase);    // 复制构造原有元素到新内存
    destroy();                              // 销毁旧栈
    base = newbase;                         // 更新指针地址指向新内存
    first_free = base + sz;
    end = base + newcapacity;
}
```

4. 调试分析

(1) 调试过程中遇到的主要问题及解决过程。使用整数类型作为元素类型实例化一个整数栈，通过菜单选项测试顺序栈的各项操作结果。测试程序略，见电子版压缩包内。

测试过程中曾发现，插入元素后，查询栈的容量返回的是合理的正常值，而查询栈中元素个数size()返回的却是奇怪的值。经跟踪调试发现问题出在reallocate函数中，原先的代码如下，字体加粗部分是代码有问题的地方。

```
template<class Tp class Alloc >
inline void Stack<Tp Alloc > reallocate()
{
    int newcapacity = 2 * stdmax(size() 1);           // 新容量大小
    Tp *newbase = alloc.allocate(newcapacity);        // 分配新的内存空间
    uninitialized_copy(base first_free newbase);      // 复制构造原有元素到新内存
    destroy();                                  // 销毁旧栈
    base = newbase;                             // 更新指针地址指向新内存
    first_free = base + size();
    end = base + newcapacity;
}
```

更新first_free指针时，调用了函数size()，size()函数通过first_free-base计算原栈的长度，但再base = newbase之后，原先base指针值也被改变，从而计算出的是不正确的值，first_free没有正确赋值，因此出现异常。实际上将函数展开，该语句等价于：

```
first_free = base + (first_free - base)
```

可看出，实际上这条语言什么也没做。解决办法是更新指针前使用一个变量记录下原先栈的长度，然后再更新base指针，最后通过保存的原先栈的长度正确定位first_free指针的新位置。更正后的realloc函数如下，字体加粗部分标识出了修改的地方。

```
template<class Tp class Alloc >
inline void Stack<Tp Alloc > reallocate()
{
    int sz = size();
    int newcapacity = 2 * stdmax(sz 1);               // 新容量大小
    Tp *newbase = alloc.allocate(newcapacity);        // 分配新的内存空间
    uninitialized_copy(base first_free newbase);      // 复制构造原有元素到新内存
    destroy();                                  // 销毁旧栈
    base = newbase;                             // 更新指针地址指向新内存
    first_free = base + sz;
    end = base + newcapacity;
}
```

第二处修改stdmax(sz 1)并不是必需的，但既然已经计算出了size()的值，直接使用该值可

减少一次计算。更正后经测试，栈工作正常。

(2) 对设计和编码的讨论和分析。该程序实现了顺序栈的操作。分析程序代码的质量，主要从以下几个方面考虑。

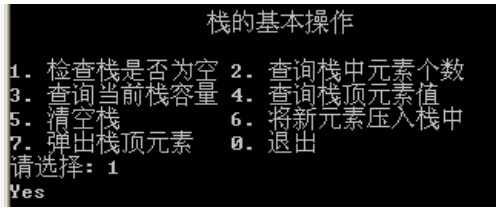
- 正确性。在一定的数据范围内，该程序能实现所需功能，所以正确性是没有问题的。
- 健壮性。在一定的数据输入范围内，该程序能较好的实现链表的操作。但是如果输入数据非法，该程序还是可能会产生一些预想不到的输出结构，或是不做任何处理。所以，该程序的健壮性有待进一步的提高。要综合考虑一些情况，当输入有误时，应返回一个表示错误的值，并中止程序的执行，以便在更高的抽象层次上进行处理。

5．使用说明

顺序栈模板类的全部实现内容均包含在一个独立的C++头文件stack.hpp中，该类放在名字空间han中，避免全局名字空间的污染。要在程序中使用这个类只需包含该头文件即可。这个类实现了一般的复制构造函数和赋值函数，然而这些操作涉及大量的元素复制及内存分配释放，故不推荐经常进行此类操作，如需要使用栈类型作为函数参数，一般应使用const hanStack<T> &或hanStack<T> &作为参数类型。这个类的设计主要考虑了性能的高效，因此没有进行多余的错误检查。可能错误的调用是在栈为空时调用SeqStackPop函数或SeqStackGetTop函数，所以，应该保证在调用这些函数时栈非空，或者调用这两个函数前先调用empty函数检查栈是否为空（参见测试程序里的做法）。

使用方法见使用说明文件。

6．测试程序的运行结果



(1) 初始时栈为空，检查结果为“Yes”。

(2) 建立栈。

- 重复选择6，依次输入数据1、2、3、4和5。

(3) 检查栈是否为空，取栈顶元素，输出栈中元素个数。

- 选择1，得到执行结果：栈非空。
- 选择4，得到执行结果：栈顶元素为5。
- 选择2，得到执行结果：栈中元素个数为5。

(4) 进行出栈操作。

- 选择7，得到执行结果：栈顶元素5已弹出。
- 选择7，得到执行结果：栈顶元素4已弹出。
- 选择7，得到执行结果：栈顶元素3已弹出。

(5) 进行压栈操作。

- 重复选择6，依次输入数据6、7。

(6) 取栈顶元素，输出栈中元素个数。

- 选择4，得到执行结果：栈顶元素为7。
- 选择2，得到执行结果：栈中元素个数为4。

(7) 清空栈，测试栈空。

- 选择5，显示：栈已经被清空。
- 选择1，显示Yes，表示栈已经为空。

7．心得体会

使用C++语言编写程序时，内存管理是很繁琐的事情。简单情况下可以使用new、delete进行内存分配和释放（对象的构造和析构会伴随new和delete进行）。为了使用的高效和灵活，C++没有垃圾回收机制，因此管理内存时要十分小心，不再被使用的对象应注意调用delete析构并回收内存空间，类的构造函数和析构函数要考虑周全，设计妥当，防止发生内存泄漏。更灵活的方式（如分配内存但不构造对象）可使用allocator，此时更要小心，因为还要注意控制对象的构造与析构等。

当程序出现异常时，单步跟踪调试程序，同时检查运行结果。思考程序代码的真实意义和可能的运行情况，寻找其中的关联和影响，一般就能找到问题的所在。某些细微的地方，可使用step into的方式步进跟踪。

在编程过程中，我觉得算法是一个非常重要的方面，上机操作不能仅仅只会键盘操作，重要的还是要动脑，构思出解决问题方法，即找出解决问题的算法。程序设计过程有如解决实际问题：首先要了解这个问题的基本要求和要实现的功能；其次，从问题的要害入手，从前到后解决问题的每个方面，即从输入开始入手，着重思考从输入导出输出的方法。只有算法也是不行的，算法是主要脉络，而上机操作就是丰富脉络的必需枝叶，对于一个算法程序，如果上机实际操作不过关，算法设计得再好也无法正常运行。只有多实践，才能有更优化的程序。这次实验在程序的可读性和可用性方面花费了大量时间，学到更多的是界面优化思想以及如何才可以令用户满意。

总之，通过这次实验我深刻认识到要编好程序，不仅需要掌握理论知识，实践操作也是非常重要的。任何学习过程也都一样，要把理论与实践结合起来。同时我们也要端正态度，编程是一个需要缜密思维和严谨态度的复杂工程，来不得半点马虎，我们要认真地对待。

附录：源程序文件清单

各程序源代码文件随本实验报告电子版一起打包，存放在src子目录。

文件清单如下：

stack.hpp	顺序栈定义及实现，C++头文件
test_stack.cpp	顺序栈测试程序
queue.hpp	循环队列定义及实现，C++头文件
test_queue.cpp	循环队列测试程序
reverse.cpp	利用栈将队列中的元素逆置
job_scheduling.cpp	操作系统作业调度模拟
readme.txt	说明文件