第 1 章　课程概论

当今人类社会已进入了一个以信息化为标志的飞速发展时代。数据是信息的载体，随着信息化社会的发展，由计算机处理的数据数量随之增大，数据类型随之增多，数据结构随之更加复杂。由于数据的组织方式直接关系到程序结构的优劣、程序处理的效率，这就给程序设计带来了一些新的问题。为了编出一个结构好、效率高的处理程序就必须分析待处理的对象特性以及各处理对象之间存在的关系。这就是“数据结构”这门学科形成和发展的背景。

1.1　课程的初步认识

在计算机的使用初期，它的主要应用领域是科学计算。当使用计算机解决一个具体问题时，一般需要经过下列几个步骤：首先要从该具体问题抽象出一个适当的数学模型，然后设计或选择一个解此数学模型的算法，最后编出程序进行测试、调试，直至得到最终的解答。例如求解梁架结构中应力的数学模型为线性方程组，可以使用迭代算法来求解线性方程组。

然而，随着计算机应用领域的不断扩大，出现了许多无法用数值及数学方程加以描述的具体问题。这是一类非数值计算问题，下面所列举的就是属于这一类的具体问题。

例 1-1　人事信息检索问题。当要查找某公司员工的信息时，一般是给出该员工的编号或姓名，通过信息目录卡片和信息卡片来查得该员工的有关信息。但也有可能要查找某一种类别的员工信息，例如要查找该公司的员工中具有“高级程序员”技术职称的人员信息，或者是属于某一行政分组的人员信息等。若利用计算机来检索人事信息，则计算机所处理的对象便是这些信息目录卡片及员工信息卡片中的信息。为此，在计算机中必须建立和存储与此相关的 3 张表，一张是按员工编号排列的员工信息表，其余两张分别是按技术职称与按行政分组顺序排列的索引表。在员工信息表中存放编号、姓名、职称、职务及爱好等信息，在职称索引表中存放职称与员工编号的对应信息，在组别索引表中存放行政分组与员工编号的对应信息，如图 1.1 所示。

在人事信息检索问题中，上述这几张表便是数学模型，计算机的主要操作便是按指定的要求对这些表进行查找。在这一类属于文档管理的数学模型中，计算机的处理对象之间通常存在着一种最简单的线性关系，相应的数据结构可称为线性数据结构。

例 1-2　八皇后问题。在八皇后问题中，处理过程不是根据某种确定的计算法则，而是利用试探和回溯的探索技术求解。为了求得合法布局，在计算机中要存储布局的当前状态。从最初的布局状态开始，一步步地进行试探，每试探一步形成一个新的状态，整个试探过程形成了一棵隐含的状态树，如图 1.2 所示（为了描述方便，将八皇后问题简化为四皇后问题）。回溯法求解过程实质上就是一个遍历这棵状态树的过程。在这个问题中所出

现的‘树’也是一种数据结构,它可以应用在许多非数值计算的问题中。

编　号	姓　名	职　称	职　务	爱　好
990001	丁一	系统分析员	总工	羽毛球、乒乓球
990002	丁二	高级程序员	一组组员	
990003	马一	程序员	一组组员	
990004	马二	程序员	一组组员	
990005	张一	高级程序员	一组组长	网球、乒乓球
990006	张二	高级程序员	二组组员	网球
990007	李一	程序员	二组组员	
990008	李二	程序员	二组组员	
990009	王一	程序员	二组组员	
990010	王二	程序员	二组组员	

（a）员工信息表

系统分析员	1
高级程序员	2,5,6
程序员	3,4,7,8,9,10

（b）职称索引表

第一组	5,2,3,4
第二组	6,7,8,9,10

（c）组别索引表

图 1.1　人事信息检索系统中的数据结构

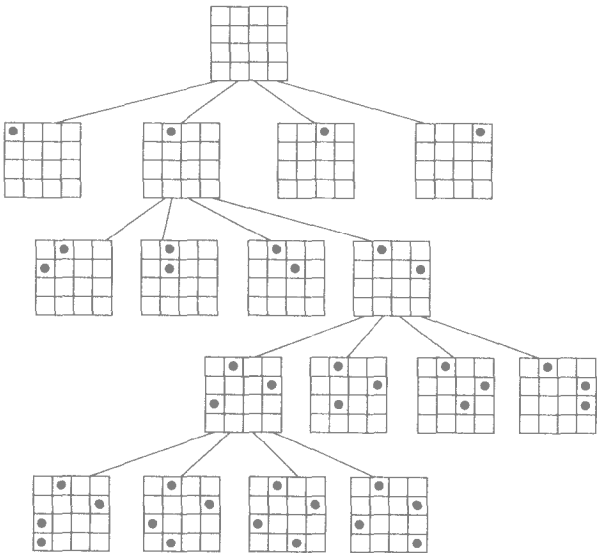

图 1.2　四皇后问题中隐含的状态树

在交通咨询系统中，也可以采用一种图的结构来表示实际的交通网络。图中的顶点表示城市，边表示城市间的交通联系，对边所赋予的权值可以表示两城市间的距离，或途中所需的时间或交通费用等。考虑到交通图的有向性（如航运、逆水和顺水时的船速就不一

样），图中的边可以用弧来表示，如图 1.3 所示。这个咨询系统可以回答旅客提出的各种问题，例如从某地到某地应如何走才最节省费用，也就是要求从某地到某地的最短路径。在这个问题中所使用的图也是一种数据结构，它被用于解决这一类实际问题。

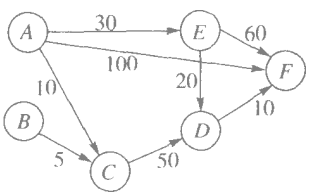

图 1.3 一个表示交通网的例图

综上 3 个例子可见，描述这类非数值计算问题的数学模型不再是数学方程，而是诸如表、树、图之类的数据结构。因此，可以说数据结构课程主要是研究非数值计算的程序设计问题中所出现的计算机操作对象以及它们之间的关系和操作的学科。

在计算机科学中，数据结构不仅是一般程序设计（特别是非数值计算的程序设计）的基础，而且是设计和实现编译程序、操作系统、数据库系统以及其他系统程序的大型应用程序的重要基础

在计算机专业中，数据结构是一门综合性的专业基础课程，它不仅是计算机专业教学计划中的核心课程之一，而且是非计算机专业的主要选修课程。

学习数据结构课程的目的是为了了解计算机处理对象的特性，将现实世界中实际问题中所涉及的处理对象在计算机中表示出来并对它们进行处理。与此同时，通过算法训练提高计算机思维的能力，通过程序设计的技能训练来促进综合应用能力和专业素质的提高。

本书按面向对象程序设计的基本方法，以数据类型及其相应的实现为主线索，以实例引入、数据存储、基本操作、程序实现、归纳概括为基本环节，由浅入深地展开讲解。目的就是使读者掌握课程中最基本最重要的内容——掌握面向对象的实现方法，培养运用数据结构的知识解决实际问题的能力

1.2 数据结构的基本概念

在系统学习数据结构知识之前，我们先了解一些概念和术语的确切含义。

1.2.1 基本术语

数据（Data）是信息的载体，它能够被计算机识别、存储和加工处理。它是计算机程序加工的原料。例如，一个利用数值分析方法解代数方程的程序所用的数据是整数和实数，而一个编译程序或文本编辑程序所使用的数据是字符串。随着计算机软、硬件技术的发展，应用领域的扩大，数据的含义也随之拓宽。像多媒体技术中所涉及的视频和音频信号，经采集转换后都能形成被计算机接受的数据。

数据元素（Data Element）是数据的基本单位。在不同条件下，数据元素又可称为元素、结点、顶点、记录。例如，在人事信息检索问题中员工信息表的一个记录，在八皇后问题中状态树的一个状态，在交通咨询系统中交通网的一个顶点等。在数据元素是记录的情形下，一个数据元素又可由若干数据项（也称为字段、域）组成，数据项是具有独立含义的最小单位

数据元素类(Data Element Class)是具有相同性质的数据元素的集合。在某个具体问题中，数据元素都具有相同的性质（元素值不一定相等），属于同一数据元素类，数据元素是数据元素类的一个实例。例如在交通咨询系统的交通网中，所有的顶点是一个数据元素类，顶点 A 和顶点 B 各自代表一个城市，是该数据元素类中的两个实例，其数据元素的值分别为 A 和 B。

1.2.2 数据结构的概念

数据结构(Data Structure)是指互相之间存在着一种或多种关系的数据元素的集合。在任何问题中，数据元素之间都不会是孤立的，在它们之间都存在着这样或那样的关系，这种数据元素之间的关系称之为结构。根据数据元素间关系的不同特性，通常有下列 4 类基本结构：

集合　在集合结构中，数据元素间的关系是“属于同一个集合”，集合是元素关系极为松散的一种结构。

线性结构　结构中的数据元素之间存在一个对一个的关系。

树形结构　结构中的数据元素之间存在一个对多个的关系。

图状结构　结构中的数据元素之间存在多个对多个的关系，图状结构也称网状结构。图 1.4 表示上述 4 类基本结构的关系图。

图 1.4　4 类基本结构的关系图

1.2.3 逻辑结构和物理结构

数据结构包括数据的逻辑结构和数据的物理结构。数据的逻辑结构可以看作是从具体问题中抽象出来的数学模型，它与数据的存储无关，而我们研究数据结构的目的是为了在计算机中实现对它的操作，为此还需要研究如何在计算机中表示一个数据结构。数据结构在计算机中的表示（又称映像）称为数据的物理结构，或称存储结构。它所研究的是数据结构在计算机中的实现方法，包括数据结构中元素的表示及元素间关系的表示。

数据的存储结构可采用顺序存储或链式存储两种。

顺序存储方法是把逻辑上相邻的元素存储在物理位置相邻的存储单元中，元素间的逻辑关系由存储单元的相邻关系来体现。由此得到的存储表示称为顺序存储结构。顺序存储结构通常是借助于程序语言中的数组来实现的。

链式存储方法对逻辑上相邻的元素不要求其物理位置相邻，元素间的逻辑关系通过附设的指针字段来表示。由此得到的存储表示称为链式存储结构。链式存储结构通常是借助于程序语言中的指针类型来实现的。

除了通常采用的顺序存储结构和链式存储结构外，有时为了查找方便还采用索引存储方法和散列存储方法。

1.2.4 数据结构形式定义

从 1.2.2 节所介绍的数据结构的概念中我们可以知道，一个数据结构有两个元素，一个是元素的集合，另一个是关系的集合。因此在形式上，数据结构通常可以采用一个二元组来表示，即：

```
Data Structure = (D,R)
```

其中，D 是数据元素的有限集，R 是 D 关系的有限集。

例如，在 1.1 节中所介绍的人事信息检索问题中，数据元素集合主要是员工信息表中的所有记录，而元素间关系的集合则可以从不同的视点去建立：可以按员工的编号来建立元素间的线性关系从而形成线性的数据结构；可以按员工的行政分组来建立元素间的层次关系从而形成树形的数据结构；可以按员工的爱好建立元素间的网状关系从而形成网状的数据结构，如图 1.5 所示。

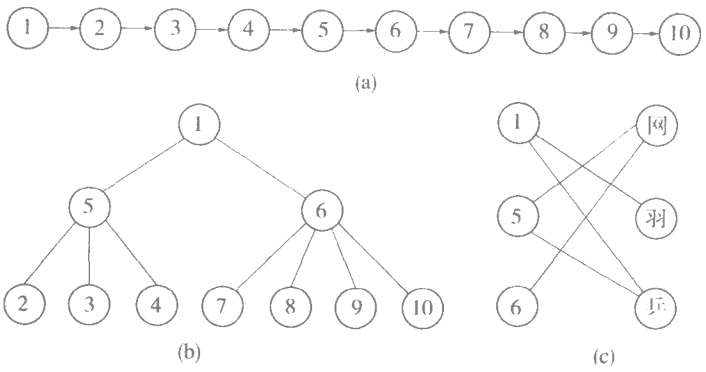

图 1.5 员工信息表中的数据结构

如果我们使用二元组来表示上述的线性结构，则可表示为：

```
Linear-List = (D,R)
```

其中：

```
D = {01,02,03,04,05,06,07,08,09,10}   (用编号的后两位表示)
R = { <01,02>, <02,03>  ,… <09,10> }
```

树形结构则可表示为：

```
Tree = (D,R)
```

其中：

```
D = {01,02,03,04,05,06,07,08,09,10}
R = { <01,05>, <01,06>, <05,02>,… <06,10> }
```

图状结构则可表示为：

Graph =(D,R)

其中：

D = {01,02,03,04,05,06,07,08,09,10, 网,羽,乒}
R = { <01,羽>, <01,乒>, <05,网>, <05,乒>, <06,网> }

1.3 数据类型及面向对象概念

1.3.1 数据类型概述

数据类型(Data Type)是和数据结构密切相关的一个概念，它最早出现在高级程序语言中，用以刻画程序中操作对象的特性。在用高级语言编写的程序中，每个变量、常量或表达式都有一个它所属的确定的数据类型。类型明显或隐含地规定了在程序执行期间变量或表达式所有可能的取值范围，以及在这些值上允许进行的操作。因此数据类型是一个值的集合和定义在这个值集上的一组操作的总称。例如 PASCAL 语言中的整数类型，其取值的范围为[-maxint, maxint]上的整数(maxint 是依赖特定计算机的最大整数)，定义在其上的一组操作为加、减、乘、除及取模等。

在高级程序语言中，数据类型可分为两类：一类是原子类型，另一类则是结构类型。原子类型的值不可分解，如 PASCAL 语言中的整型、实型等标准类型，而结构类型的值由若干成分按某种结构组成，因此是可分解的，并且它的成分可以是非结构的也可是结构的。例如数组的值由若干分量组成，每个分量可以是整数，也可以是数组等。在某种意义上，数据结构可以看成是“一组具有相同结构的值”，而数据类型则可看成是由一种数据结构和定义在其上的一组操作组成。

1.3.2 抽象数据类型

抽象数据类型(Abstract Data Type，简称 ADT)是指一个数学模型以及定义在该模型上的一组操作。抽象数据类型的定义仅取决于它的一组逻辑特性，而与其在计算机内部如何表示和实现无关，即不论其内部结构如何变化，只要它的数学特性不变，都不影响其外部的使用。

抽象数据类型和数据类型实质上是一个概念。例如，各个计算机都拥有的整数类型就是一个抽象数据类型，尽管它们在不同处理器上的实现方法可以不同，但由于其定义的数学特性相同，在用户看来都是相同的。因此“抽象”的意义在于数据类型的数学抽象特性。

但在另一方面，抽象数据类型的范畴更广，它不再局限于前述各处理器中已定义并实现的数据类型，还包括用户在设计软件系统时自己定义的数据类型。为了提高软件的重用率，在近代程序设计方法学中要求在构成软件系统的每个相对独立的模块上，定义一组数据和施于这些数据上的一组操作，并在模块的内部给出这些数据的表示及其操作的细节，

而在模块的外部使用的只是抽象的数据及抽象的操作。这也就是面向对象的程序设计方法。

抽象数据类型的定义可以由一种数据结构和定义在其上的一组操作组成，而数据结构又包括数据元素及元素间的关系，因此抽象数据类型一般可以由元素、关系及操作 3 种要素来进行定义。在本书中，对每一种数据类型都给出它的 ADT 定义。例如栈的 ADT 定义：

元素　属于同一个数据元素类。

关系　数据元素间呈线性关系。

操作　PUSH(S,X）进栈操作、POP(S)出栈操作等。

对于一个数据成员完全相同的数据类型，如果使它赋予不同的操作，则可形成不同的抽象数据类型。例如分别将在第 3、第 4 章中介绍的栈和队列，它们可能都是相同的顺序表结构，但由于其操作不同，栈是先进后出，队列是先进先出，因而是属于两种不同的抽象数据类型。

抽象数据类型的特征是使用与实现相分离，实行封装和信息隐蔽。就是说，在抽象数据类型设计时，把类型的定义与其实现分离开来。

1.3.3　实现方法

ADT 定义为数据类型建立了一个数学模型，其中包括数据结构及其相应的一组操作，而计算机上的具体实现则需借助于高级程序语言。在具体实现时大致可选择面向过程与面向对象两种不同的方法。

面向过程的方法着眼于系统要实现的功能。从系统的输入和输出出发，分析系统要做那些事情，进而考虑如何做这些事情，自顶向下地对系统的功能进行分解，建立系统的功能结构和相应的程序模块结构。但是，当程序因某种原因需要修改时，常常要涉及到许多模块，有时因功能的改变要导致全部模块都要变更，这样的修改工作量极大并容易产生新的错误

面向对象的方法着眼于应用问题中所涉及的对象，识别为解决问题所需的各种对象、对象的属性及相应的操作，从而建立起对象的类结构。通过对类的实体施行相应的操作以及各实体间的消息传递来实现系统的功能。类的定义充分体现了抽象数据类型的思想，基于类的体系结构可以局部修改程序。当修改类中数据的存储方式及操作实现过程时，不会影响外界对该类实体的操作，从而使整个系统保持稳定。因此，用面向对象开发方法建立起来的软件易于修改，与传统的方法相比，程序具有更好的可靠性、适用性、可修改性、可维护性、可复用性和可理解性。

接下来结合栈的演示程序实现过程来比较这两种方法的不同特点。虽然有关栈的内容我们还没有作过介绍，但从下面的比较中，读者就可以大致领略到这两种程序设计方法的不同风格及其特点。

栈的特点是先进后出。这就像我们在生活中洗盘子时，总是把洗好的盘子逐个放在其他已洗好的盘子上面，而在使用盘子时，总是从上面逐个取出。一迭盘子相当于一个栈，放盘子的动作相当于进栈，取盘子的动作相当于出栈。如果要编制一个教学演示程序，模

拟对栈进行入栈出栈等操作的执行过程，则可以采用以下两种方法。

. 面向过程的方法

这是比较传统的方法，在这种方法中，可使用类型定义来描述其存储结构，并可借助于函数与过程描述其相应的操作。但在这二者之间并没有建立必然的联系。在程序中用一个字符表示栈中的一个元素，输入一个字符，若为“ P ”则表示执行出栈操作，若为“ E ”则表示退出执行，否则将该字符推入栈中。程序从开始起顺序地执行直至输入字符“E”。

```
link top; char ch;
void push(char ch);
…
char pop( );
  …
  top = NULL; ch = 'a';
  while (ch! = 'E' )
  {
    输入一个字符存入 ch;
    对 ch 进行判别并进行相应的处理
    输出栈中的当前元素
  };
```

2. 面向对象的方法

在面向对象的程序设计语言中，相关的数据及操作被统一在一个整体——对象之中。我们可以先将栈定义成类：

```
struct node
  { char data;
   struct node *next;
  };
class Tlz
  {
    private:
    struct node *top;
  public:
    void init();
    void prnt();
    char pop();
    void push (char el);
};
```

并建立一个相应的实体 Tlz *lz1 ; lz1 = new Tlz;

然后通过对该实体执行相应的操作来实现演示程序的功能。例如，先将栈清空，然后将一个元素推入栈中，再显示这个栈，可执行下述代码：

```
lz1 ->init( );
lz1 ->push(el);
lz1 ->prnt( );
```

上述代码段不仅比较简洁，容易理解，而且也不会随类中实现细节的改变而改变。

由于面向对象实现方法中的类定义充分体现了抽象数据类型的思想，因此可以将数据类型的 ADT 定义用一个类定义来表示。这种面向对象的实现方法与数据类型的 ADT 定义比较贴近，因而也就比较自然。在本书中我们采用了面向对象的实现方法。

1.3.4 面向对象的概念

下面介绍面向对象实现方法中的有关概念。

- 对象是指应用问题中所出现的各种实体，它由一组属性值和在这组值上的一组操作（方法）构成。其中，属性值确定了对象的状态。例如，在程序中常用的字符串、线性表、栈、队列等或在 Windows 应用程序中常见的窗体、组合框、编辑框、无线按钮等。对于一个编辑框对象，由它的 TOP、LEFT 属性确定了它在窗体中的位置，由它的 TEXT 属性确定了显示在该编辑框中的字符串，可以使用 GetTextLen 方法来求得该字符串的长度，也可用 Clear 方法来清除这个字符串。
- 类和实例对象在语言中是用类来定义的，类中定义了与某一种对象相关联的一组数据以及施与该数据中的一组基本操作，对象是数据与方法（操作）的统一体。在面向对象的程序设计中，如果某一个变量被定义成属于某一个类，那么该变量即可成为这种对象中的一个实例。因此，对象与实例这两个概念在程序设计中可对应于类与变量。对象、类是抽象的概念，而实例、变量代表具体事物。例如，在一个窗体中可以设置多个编辑框，尽管其位置及外观都不相同，但它们都属于编辑框组件（类）所派生的对象实体。
- 数据封装（信息隐蔽）是指在面向对象的程序设计中，对象的实现过程（包括数据的存储方式、操作的执行过程）作为私有部分被封装在类结构中，使用者不能看到，也不能直接操作该类型所存储的数据，而只能根据对象提供的外部接口来访问或操作这些数据。数据封装无论是对于使用者还是实现者都相当有利。从使用者的角度来看，只要了解类定义的接口部分，即可操作对象实例，而不必关心其实现细节。这就好比我们使用手表，只要按操作说明使用它，而不必了解手表的内部结构。这样使用者在开发过程中就可以集中精力去解决应用中所出现的问题，使问题得到简化，而且程序设计的表达方式也更加简练、直观。从实现者的角度来看，数据封装有利于编码、测试及修改。因为只有类中的成员函数才能访问它的私有成员，这样做可以使错误局部化，一旦出现错误或者有必要改变数据的存储方式或改变内部的处理过程，也不至于影响其他模块。只要向外界提供的接口方式不变，其他所有该对象的程序都可以不变，从而大大提高了程序的可靠性和稳定性。

在传统的面向过程的方式中，虽然也提供了过程与函数的调用接口，但并没有将对象作为程序的基本构件，将一组相关的数据及操作集中在一起，在数据与操作之间缺乏明确的关系，这就不能达到数据封装的目的。

- 继承与派生继承机制是面向对象方法中最有特色的方面。在面向对象的方法中，类与类之间存在着继承关系，这种继承关系与客观世界中存在的一般与特殊的关系类似。例如，在 Windows 的界面设计中，可将窗体分为一般窗体和对话框，而

对话框又可分为打开对话框、确认对话框等，这些窗体都有窗体的共同特征，但不同的窗体又有自身的不同特征。我们可以将窗体定义成基类，建立它的派生类，如对话框、打开对话框等。将各派生类中的共同部分集中到基类中去，派生类中只保留自己特有的属性和方法，派生类的各对象独享该派生类的属性和方法，同时还能共享基类中共有的属性和方法。这样做的好处是可以将各个对象的属性和方法合理分配到所有的类中，减少数据存储的程序代码的重复。

如上所述，所谓继承是指从已定义的类中导出新类时，新类将自动包括原有类的全部数据和方法，这种导出新类的过程称为派生。

继承是一个传递的过程，从而构成了类的层次体系。除最上层的类以外每个类都有一个父类（基类），除最下层的类以外每个类都有一些子类（派生类），派生类既具有从它的基类那里继承来的数据和操作，也可以扩充自己特有的数据和操作。这样，越是后面的类，包括的数据及方法就越丰富。

利用类的层次结构和继承性，不同对象的共同性质只需定义一次，这符合软件重用的目标。它为我们带来的最大好处是能够充分利用前人的成果。大量经过验证的类以层次的结构存放在类库中，用户可以根据需要加以继承，从而改变了程序设计中一切从零开始的弊端。

算法及算法分析

算法与数据结构关系紧密，在算法设计时先要确定相应的数据结构，而在讨论某一种数据结构时也必然会涉[illegible]

[illegible]

处于动态等待中，因此操作系统不是一个算法。另外，程序中的指令必须是机器可执行的，而算法中的指令则无此限制算法代表了对问题的解，而程序则是算法在计算机上的特定实现。一个算法若用程序设计语言来描述，它就是一个程序。

在计算机科学的研究中，算法与数据结构相辅相成。解决某一特定类型问题的算法可以选定不同的数据结构，而且选择恰当与否直接影响算法的效率。反之，一种数据结构的优劣要由各种算法的执行来体现。因此有人称“算法 + 数据结构 = 程序”。

1.4.2 算法描述

算法的描述可以使用各种不同的方法。

最简单的方法是使用人们通常使用的自然语言。用自然语言来描述算法的优点是简单且便于人们对算法的阅读与理解，也不需要使用注释。

其次是使用流程图、N-S 图等用于算法描述的工具，其特定是描述过程简洁、明了。用以上两种方法描述的算法不可直接在计算机中执行，若要将它转换成可执行的程序还有一个编程的问题。

也可以直接用某种高级程序设计语言来描述算法。不过直接使用高级语言来描述并不容易，而且不太直观，常常需要借助于注释才能看明白。

为了解决理解与执行这两者之间的矛盾，人们常常使用一种称为伪码语言的描述方法来进行算法描述。伪码语言介于这两者之间：它忽略高级语言中一些严格的语法规则与描述细节，因此它比程序语言更容易描述与被人理解，而比自然语言更接近程序语言。它虽然不能直接执行，但很容易转换成高级语言。例如在许多教材中所使用的类 PASCAL 语言或类 C 语言等都属于这一种伪码语言。

在本教材中，为了体现高职教育的特点，侧重编程能力与综合应用能力的培养，我们选择了可直接执行的高级语言来描述算法。同时，为了便于学生对算法的理解与掌握，通常分以下几个步骤进行说明：

(1) 算法的含义或问题的说明。
(2) 参数与功能。
(3) 工作变量说明。
(4) 处理过程。
(5) 程序。
(6) 执行实例。

1.4.3 算法设计的要求

要设计一个好的算法通常要考虑以下几点：

正确 算法应当满足具体问题的需求。正确性是设计和评价一个算法的首要条件。如果一个算法不正确，其他方面就无从谈起。一个正确的算法是指在输入合法的数据后，能在有限的运行时间内得出正确的结果。

可读 算法最主要的目的是为了阅读与交流，并将它转换成可实现的程序在计算

机中执行。可读性好不仅有助于人们对算法的理解，而且也有利于程序的调试与修改。在保证算法正确的前提下，应十分强调算法的可读性。

健壮　健壮性是指算法对各种可能的情形都考虑得非常完善。当输入数据非法时，算法也能适当地作出反应或进行处理，而不会发生异常或输出莫名其妙的结果。例如，输入三角形的三条边长时，若两边之和小于另一条边，则算法执行中应输出相应的通告信息而不是发生异常。

高效　是指算法的执行效率要高。算法的效率包括时间效率与空间效率两个方面。时间效率是指执行算法所需要的时间，而空间效率是指执行该算法所需的存储量。对应同一个问题和同一种问题规模，若采用不同的算法进行处理，其执行的效率就可能不相同。对于一种算法来说，如果执行所需的时间越短并且所需的存储量越小，其效率就越高。

当然，我们希望采用一个占用存储空间小、执行时间短，其他性能也好的算法，然而实际上却往往很难做到，因为在许多情况下各种因素是互相制约的。例如要求算法的可读性好，则其执行效率就不一定理想；如果要求执行时间短，则所需的存储量就可能比较大；如果要求使用的存储量较小，则执行时间就可能较大。在这些情形下，如何选择应根据具体的情形有所侧重，若程序使用的次数较少，则应侧重于算法的可读性，力求算法简明易懂。对于反复多次使用程序的情况，则应侧重于算法的执行效率，力求算法快速执行。若程序的数据量很大，而所提供的存储空间较小时，则相应的算法应主要考虑如何节省存储空间。但在一般情况下，在可读性与效率之间应侧重于可读性，在时间与空间效率之间应侧重于时间效率。

1.4.4　算法分析

我们可以从一个算法的时间复杂度与空间复杂度来评价该算法的优劣。

当将一个算法转换成程序并在计算机上执行时，其运行所需的时间取决于下列因素：

- 硬件的速度，例如使用 486 机还是 586 机。
- 书写程序的语言，实现语言的级别越高，执行效率就越低。

  编译程序所生成的程序质量，代码优化较好的编译程序所生成的程序质量较高。

  问题的规模，例如求 100 以内的素数与求 1000 以内的素数其执行时间必然是不同的。

显然，在各种因素都不确定的情况下，很难比较算法的执行时间，也就是说使用执行算法的绝对时间来衡量算法的效率是不合适的。为此，我们可以将上述各种与计算机有关的软、硬件因素都确定下来，这样一个特定算法的运行工作量的大小就只依赖于问题的规模(通常用正整数 n 表示），或者说它是问题规模的函数。

一个算法由控制结构和原操作构成，其执行时间取决于两者的综合效果。为了便于比较同一问题的不同算法。通常的做法是：从算法中选取一种对于所研究的问题来说是基本运算的原操作，以该原操作重复执行的次数作为算法的时间量度。一般情况下，算法中原操作重复执行的次数是问题规模 n 的某个函数，算法的时间量度记作：

```
T(n) = o( f (n) )
```

该式表示算法中原操作的执行次数与问题规模 n 的某个函数同阶。显然，算法中原操作应该是其重复执行的次数与算法的执行时间成正比的原操作，在多数情况下它是最内层循环中语句中的原操作，它的执行次数和包含它的语句的频度相同。语句的频度指的是该语句重复执行的次数，例如，在下面 3 个程序段中，

```
x = x +1;
for (i =1;i <=n;i ++) x = x +1;
for (i =1;i <=n;i ++)
    for (j =1;j <=n;j ++) x = x +1;
```

含基本操作 x 增 1 的语句 $x = x+1$ 的频度分别为 1，n，n^2，则这 3 个程序段的时间复杂度分别为 $O(1)$、$O(n)$、$O(n^2)$，分别称为常数阶、线性阶、平方阶。

又如：在如下所示的两个 $N \times N$ 矩阵相乘的算法中，

```
for(i =1;i <=n;i ++)
  for (j =1;j <=n;j ++)
      {  s =0;
         for (k =1;k <=n;k++)  s =s +a[i][k] *b[k][j];
         c[i][j] =s;
      };
```

其基本操作为乘法运算，所在的语句的执行频度为 n^3，该程序段的时间复杂度为 $O(n^3)$，称之为立方阶。算法中还可能呈现的时间复杂度有：对数阶 $O(\log n)$、指数阶 $O(2^n)$ 等。

有些情况下，算法的复杂度不仅仅是问题规模 n 的函数，还与它所处理问题的数据集状态有关。在这种情况下，可对数据集的分布作某种假定（如等概率），并讨论在此情况下算法的平均时间复杂度。

例如，要在数组 $a[n]$ 中查找值为 K 的元素，若找到则输出其位置 $i(1 \leqslant i \leqslant n)$，否则输出零。算法可用程序表示如下：

```
i =n;
While ((i >0) && (a[i]! =k)) i =i -1;
Write(i)
```

在此程序中 While 循环的执行次数不仅与问题规模 n 有关，还与 K 和数组 $a[n]$ 中各分量的取值有关。在最坏的情况下， While 循环要执行 n 次，在最顺利的情况下，While 循环只要执行 1 次，平均执行次数为 $(n+1)/2$，时间复杂度为线性阶。

与分析时间效率类似，在存储空间的使用量方面，处理同一个问题的不同算法对存储空间的要求也可能会有较大的差异。例如对于以下问题：

将存放在一维数组 a 中的 n 个整数反向存放，即将 $a[1]$ 存放在原 $a[n]$存放的位置中，将 $a[2]$存放在原 $a[n-1]$ 存放的位置中，以此类推，直至将 $a[n]$存放在原 $a[1]$存放的位置中。

对于上述问题，我们可以使用一组工作单元，即设置一个数组 $b[n]$，然后使用以下的算法实现：

```
for (i =1; i <=n; i ++) b[n - i +1] = a[i];
```

```
for (i=1; i<=n; i++) a[i] = b[i];
```

但也可以只使用工作单元 i 与 w，然后使用以下的算法实现：

```
for (i=1; i<=n/2; i++)
  {w = a[i];
   a[i] = b[n-i+1];
   b[n-i+1] = w;
  };
```

显然，采用后一种算法比采用前一种算法所需的存储空间要少很多。

类似于算法的时间复杂度，空间复杂度可作为算法所需的存储空间的量度，记作：

```
S(n) = o(f(n))
```

其中 n 为问题的规模。在上述问题中，前一算法的空间复杂度为线性阶，而后一算法的空间复杂度为常数阶。

一个算法在计算机存储器中所占用的存储空间，包括算法本身所占用的存储空间、输入数据所占用的存储空间、以及算法在运行过程中所需的工作单元即实现算法的辅助空间。如果输入数据的表示形式确定的话，输入数据所占的存储空间只取决于问题的本身，与算法无关，所以一般只需分析除输入数据和算法之外的辅助空间即可。

1.5 实习一：常用算法

目的

- 加强算法训练，提高相应的思维能力。
- 了解一些常用算法的基本设计思想及处理过程。
- 熟悉实现及开发的环境，为以后实现综合性演示程序作准备。

内容

- 设有一个整数，除 2 余 1，除 3 余 2，除 5 余 4，除 6 余 5，除 7 余 0，求该数。提示：该问题可用穷举法求解。由除 7 余 0 可知，该数必为 7 的倍数，于是从 7 开始，逐个判别即可求得该数。
- f 数列可定义如下：

```
f(1) = f(2) = 1
f(n) = f(n-1) + f(n-2)  (n>2)
```

数列中的每个 $f(n)$ 称为 f 数，求 4000 之内最大的 f 数。提示：该问题可用递推与递归两种算法求解。

- 试写一个算法，实现将数组 $a[n]$ 中的 n 个整数循环后移一个位置，并给出它的时间复杂度和辅助空间的大小。

第2章　线　性　表

线性表是一种最基本的，也是最简单的数据结构。在线性表中数据元素之间的关系是线性的，数据元素可看成是排列在一条线上或一个环上的。线性表的主要操作有插入、删除和查找等。

2.1　线性表实例及概念

在计算机应用中，线性表是一种常见的数据类型，诸如在文件、内存、数据库等管理系统中经常需要对属于线性表的数据类型进行处理。

例如，图 2.1 所表示的是一个有关学生情况的信息文件，文件中每个记录对应于一个学生的情况，它由学号、姓名、性别、年龄及籍贯等信息组成。

学　号	姓　名	性　别	年　龄	籍　贯	…
970001	普琪坤	男	21	长沙	…
970002	刘然	男	22	岳阳	…
970003	冯静	女	21	长沙	…
…	…	…	…	…	…

图 2.1　学生情况信息文件

在这个实例中可以看到，文件中的记录一个接着一个，它们之间存在着一种前后关系。这好比日常生活中所看到的大街上驶过的车队，后面的车紧接着前面的车，一辆接着一辆。为了研究这种数据结构中元素间的关系，可以忽略记录中的具体内容，而只将它看作结构中的一个元素。从数据结构的观点来看，可以将这个实例中的整个信息文件看作为一个线性表，而文件中的每一个记录看作为线性表中的一个数据元素。

一般，一个线性表是由 n 个元素组成的有限序列，可记作：

$$L = (a_1, a_2, \cdots, a_{i-1}, a_i, a_{i+1}, \cdots, a_n)$$

其中每个 a_i 都是线性表 L 的数据元素，数据元素可以是各种各样的，例如它可以是一个数、一个符号或一个记录等，但同一线性表中的元素必须属于同一种数据元素类。

线性表的结构通过数据元素之间的相邻关系来体现。在线性表 L 中，元素 a_{i-1} 与 a_i 相邻并位于 a_i 之前，称为 a_i 的直接前驱，而 a_{i+1} 与 a_i 相邻并位于 a_i 之后，称为 a_i 的直接后继。元素 a_1 称为 L 的最先元素，除最先元素外，L 中的其他元素都有且仅有一个直接前驱；元素 a_n 称为 L 的最后元素，除最后元素外，L 中的其他元素都有且仅有一个直接后继。元素 a_i 是第 i 个数据元素，称 i 为数据元素 a_i 在线性表中的位序。线性表中的元素个数 n 称为线性表的长度，长度为 0 的线性表称为空表。

另一方面，通过上述实例也可以看到，这种结构有可能要进行改变。在车队的例子中，我们可能要对车队进行改编，或者要插入一些车辆，或者要退出一些车辆。在文件的例子中，通常需要对文件进行维护操作，例如要在文件的某个位置插入一个记录，或者删除某个记录等。同样，在线性表中也有相应的操作。通过对线性表中的相应操作可以实现方法的研究，来实现具体问题中的有关操作。

综上所述，线性表是由具有相同特性的 n 个元素所组成的有限序列，相邻元素之间存在着序偶关系。线性表的数据结构相当灵活，它的长度可根据需要增长或缩短，即对线性表的数据元素不仅可以访问，还可以进行插入和删除操作。

根据面向对象程序设计的原则，实现部分与接口部分两者应该分离。接口部分可以用 ADT 定义即抽象数据类型定义来进行描述。一种数据类型的 ADT 定义由数据元素、结构及操作 3 部分组成。以下是线性表的 ADT 定义：

元素　a_i 同属于一个数据元素类，$i=1, 2, \cdots, n$　$n \geqslant 0$。

结构　对所有的数据元素 $a_i(i=1, 2, \cdots, n-1)$ 存在次序关系 $< a_i, a_{i+1} >$，a_1 无前驱，a_n 无后继。

操作　对线性表可执行以下的基本操作。

initiate(L）初始化操作。设定一个空的线性表 L。

length(L)　求长度函数。函数值为给定线性表 L 中数据元素的个数。

empty(L)　判空表函数。若 L 为空表，则返回布尔值 True，否则返回布尔值 False。

clear(L）清空表操作。操作的结果使 L 成为空表。

get(L, i)　取元素函数。若 $1 \leqslant i \leqslant$ length(L) ，则函数值为给定线性表 L 中第 i 个数据元素，否则为空元素 NULL。称 i 为该数据元素在线性表中的位序。

locate(L, x)　定位函数。若线性表 L 中存在其值与 x 相等的数据元素，则函数值为该数据元素在线性表中的位序，否则为 0。若线性表中与 x 相等的数据元素不止一个，则函数值为这些元素在线性表中的位序的最小值。

prior(L, elem　求前驱函数。已知 elem 为线性表 L 中的一个数据元素，若它的位序大于 1，则函数值为 elem 的前驱，否则为空元素。

next(L, elem　求后继函数。已知 elem 为线性表 L 中的一个数据元素，若它的位序小于 length(L) ,则函数值为 elem 的后继，否则为空元素。

insert(L, i, b)　插入操作(前插)在线性表 L 的第 i 号元素之前插入一个新元素 b。此操作仅在 $1 \leqslant i \leqslant$ length(L) $+1$ 时才可行。

lete(L, i)　删除操作。若 $1 \leqslant i \leqslant$ length(L)，则删除线性表 L 中的第 i 号元素，否则此操作无意义。

对线性表还可进行一些更复杂的操作。如：将两个或两个以上的线性表合并成一个线性表；把一个线性表拆成两个或两个以上的线性表；复制一个线性表；对线性表中的元素进行排序（由此得到的线性表称为有序表）；对有序表进行插入（仍保存有序性）等。

面向对象实现方法中的类定义充分体现抽象数据类型的思想，因此可以将数据类型的 ADT 定义用一个类定义来表示。按照面向对象程序设计的方法，可以按 ADT 定义进行类定义，并在应用程序中生成相应的实体，然后可通过对实体施行某些操作来实现程序的各

种功能。在进行类定义时必须确定数据的存储方式，因此接下来先讨论线性表的存储方式，然后再介绍线性表的类定义及其各种操作的实现方法。

2.2 线性表的存储方式

在编制任何程序之前，总要考虑该程序中涉及到哪些数据，这些数据应该用何种方式存储到计算机中。如果是使用某种程序设计语言来编程，则要考虑在该程序中应定义哪些变量，这些变量的类型是什么或者应如何定义等。

线性表一般有顺序存储结构与链式存储结构两种存储方式。按顺序存储结构建立起来的线性表称为顺序表，按链式存储结构建立起来的线性表称为线性链表。

2.2.1 线性表的顺序存储结构

在计算机内可以用不同的方式表示线性表，其中最简单和最常用的方式是用一片连续的存储区域，也就是用一组地址连续的存储单元来依次存储线性表的各个元素。线性表$(a_1, a_2, \cdots, a_i, \cdots, a_n)$的顺序存储结构如图 2.2 所示，这种存储方式的特点是用存储单元物理位置的相邻来表示相邻元素间的逻辑关系。

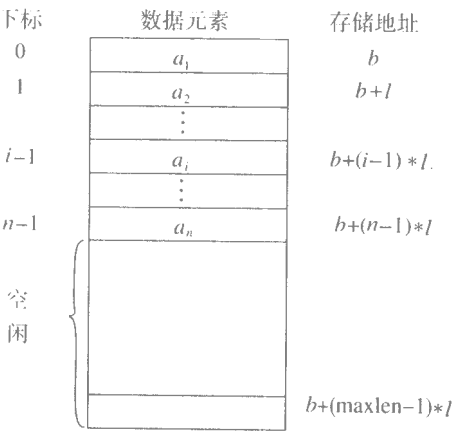

图 2.2 线性表顺序存储结构示意图

假设线性表的每个元素需占用 L 个存储单元，并用第一个存储单元的地址作为数据元素的存储位置。则线性表中第 $i+1$ 个数据元素的存储位置 $\text{LOC}(a_{i+1})$和第 i 个数据元素的存储位置 $\text{LOC}(a_i)$ 之间满足下列关系：

$$\text{LOC}(a_{i+1}) = \text{LOC}(a_i) + L$$

一般来说，线性表中第 i 个元素 a_i 的存储地址为：

$$\text{LOC}(a_i) = \text{LOC}(a_1) + (i-1) * L$$

式中 $\text{LOC}(a_1)$ 为线性表的第一个元素 a_1 的存储地址，通常称为线性表的开始地址或基地址。