第 1 章　数据结构的基本概念

本章学习目标

通过对本章内容的学习，学生应该能够做到：

1）了解：数据结构在计算机数据处理中的作用；基于面向对象描述数据结构算法的优势，以及“对象=数据结构+算法”的概念。

2）理解：数据结构研究的数据之间的逻辑关系、数据在计算机内部的存储结构，以及在数据的各种结构上实施有效的操作或处理（算法）等概念和相互关系。

3）掌握：数据结构的相关基本概念与术语；面向对象的基本概念和基本思想。

计算机已经深入到人类社会的各个领域，计算机的应用已不再局限于科学计算，而是更多地用于控制、管理及数据处理等非数值计算的处理工作。与此相应，计算机加工处理的对象由纯粹的数值发展到字符、表格和图像等各种具有一定结构的数据，这就给程序设计带来了一些新的问题。为了编写出一个好的程序，必须分析待处理对象的特性以及它们之间存在的关系，这就是“数据结构”这门学科形成和发展的背景。分析数据对象之间的逻辑关系，并用计算机存储结构体现出这些逻辑结构并操作这些数据，就是数据结构这门课程要解决的问题。

1.1　数据结构的概念和术语

在讨论数据结构之前，让我们先来介绍几个与数据结构密切相关的概念和术语。

数据（Data）：数据是对客观事物的符号表示，在计算机科学中是指所有能输入到计算机中并被计算机处理的符号的总称。它是信息的载体，是计算机程序加工的原料。对计算机科学而言，数据的含义极为广泛。一般来说，数据主要有两大类，一类是数值数据，包括整数、实数、复数等，主要用于工程、科学计算和商业事务处理；另一类是非数值数据，主要包括字符、字符串、图像、声音等，它们可以通过编码而转变为可被计算机处理的数据。

数据元素（Data Element）：数据元素是数据的基本单位，在计算机程序中通常作为一个整体进行考虑和处理。一个**数据元素**可由若干个**数据项**组成，例如，一个学生的基本信息为一个数据元素，可以包括学号、姓名、性别、年龄、成绩、家庭地址等多个数据项。数据项是数据处理中不可分割的最小单位。

数据对象（Data Object）：数据对象是性质相同的数据元素的集合，是数据的一个子集。例如，英文字母数据对象可以是集合 L={‘A’、‘B’、‘C’……‘Z’}，整数数据对象可以是集合 N={–32767，–32766，…，–1，0，1，2，…，32768}。

数据结构（Data Structure）：数据结构是相互之间存在一种或多种特定关系的数据元素的集合。任何问题中，数据元素都不是孤立存在的，它们之间存在某种关系，这种数据元素相互之间的关系称为**结构**。

根据数据元素之间关系的不同特性，通常有如图 1.1.1 所示的四类基本结构。

（1）集合结构

这种结构中的数据元素是无序且没有重复的元素，它们之间除了“同属于一个集合”的

关系外，无其他关系。

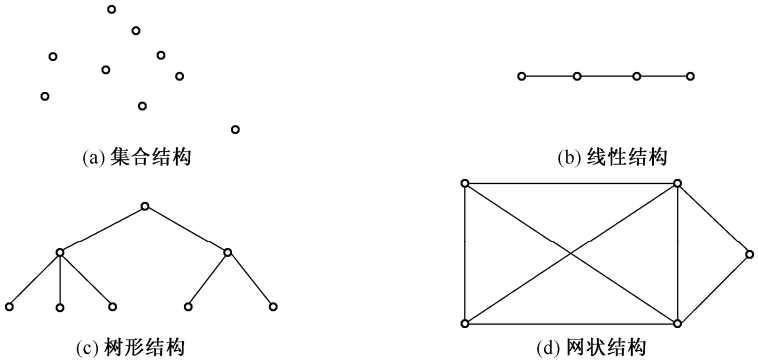


图 1.1.1　四类基本结构

（2）线性结构

这种结构中的数据元素之间存在一个对一个的关系，所有的数据成员按某种次序排列在一个序列中，除第一个元素外，每个元素都有一个且仅有一个直接前驱，第一个数据元素没有直接前驱；除最后一个元素外，每个元素都有一个且仅有一个直接后继，最后一个数据元素没有直接后继。

（3）树形结构

这种结构中的数据元素之间存在一个对多个的关系。

（4）图状结构或网状结构

这种结构中的数据元素之间存在多个对多个的关系。

数据结构是一个二元组：

$$\text{Data\_Structure} = (D, S)$$

其中，D 是数据元素的有限集，S 是 D 上关系的有限集。

上述结构定义中关系描述的是数据元素之间的逻辑关系，因此，又称为数据的**逻辑结构**。然而，讨论数据结构的目的是在计算机中实现对它的操作，因此，我们还需研究如何在计算机中表示它。

数据结构在计算机中的表示称为数据的**物理结构**，又称为**存储结构**，它包括数据元素的表示和关系的表示。在计算机中表示信息的最小单位是二进制的一位，叫做位。在计算机中，我们可以用一个由若干位组合起来形成的一个位串表示一个数据元素，通常称这个位串为**元素或结点**。当数据元素由若干数据项组成时，位串中对应于各个数据项的子位串称为**数据域**。因此，元素或结点可看做数据元素在计算机中的映像。

数据元素之间的关系在计算机中有两种不同的表示方法：**顺序映像和非顺序映像**，对应两种不同的存储结构，即**顺序存储结构和链式存储结构**。顺序映像的特点如下：借助元素在存储器中的相对位置来表示数据元素之间的逻辑关系。非顺序映像的特点如下：借助指示元素存储地址的指针表示数据元素之间的逻辑关系。数据的逻辑结构和物理结构是密切相关的两个方面。一个算法的设计取决于问题的逻辑结构，而算法的实现依赖于采用的存储结构。

通常我们讨论数据结构时，不但要讨论各种在解决问题时可能遇到的典型的逻辑结构，还要讨论这些逻辑结构的存储映像（存储实现），此外，还要讨论这种数据结构的相关操作及其实现。因此，数据结构要研究的主要内容可以简要地归纳为以下三个方面。

1）研究数据之间固有的客观联系（**逻辑结构**）。

2）研究数据在计算机内部的存储方法（**存储结构**）。

3）研究如何在数据的各种结构上实施有效的操作或处理（**算法**）。

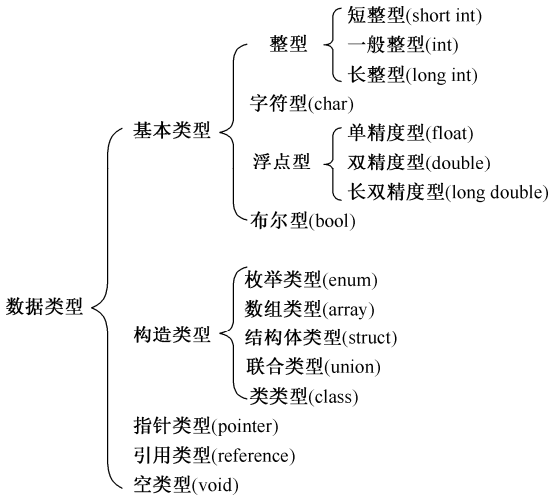
1.2 抽象数据类型

数据结构要研究逻辑结构和存储结构，那么，如何描述存储结构呢？存储结构涉及数据元素及其关系在存储器中的物理表示，由于本书是在高级语言的层次上讨论数据结构的，因此不能直接用内存地址来描述存储结构，我们可以借用高级程序语言中提供的数据类型来描述它。例如，可用一维数组类型来描述顺序存储结构，用指针来描述链式存储结构。下面来回顾一下什么是数据类型。

1.2.1 数据类型

数据类型是一组性质相同的值的集合以及定义于这个值集合上的一组操作的总称。

计算机处理的数据是以某种特定的形式存在的，如整数、浮点数、字符等形式。C++可以使用的数据类型有以下几种：



C++没有统一规定各类数据的精度、数值范围和在内存中所占的字节数，计算机所能表示的实际数据范围根据编译器和计算机系统结构不同而不同。表 1.2.1 是 Visual C++数值型和字符型数据在内存中所占的字节数和数值范围。

表 1.2.1 数值型和字符型数据的字节数和取值范围

类型	类型标识符	字节	数 值 范 围
整型	[signed] int	4	−2147483648～+2147483647
无符号整型	unsigned [int]	4	0～4294967295
短整型	short [int]	2	−32768～+32767
无符号短整型	unsigned short [int]	2	0～65535
长整型	long [int]	4	−2147483648～+2147483647
无符号长整型	unsigned long [int]	4	0～4294967295
字符型	[signed] char	1	−128～+127
无符号字符型	unsigned char	1	0～255
单精度型	float	4	3.4×10^{-38}～$3.4\times10^{+38}$
双精度型	double	8	1.7×10^{-308}～$1.7\times10^{+308}$
长双精度型	long double	8	1.7×10^{-308}～$1.7\times10^{+308}$

1）整型数据有长整型（long int）、一般整型（int）和短整型（short int）之分。C++没有规定每一种数据所占的字节数，只规定 int 型数据所占的字节数不大于 long 型数据，不小于 short 型数据。一般而言，在 16 位机的 C++系统中，short 型数据和 int 型数据占 2 字节，long 型数据占 4 字节；在 32 位机的 C++系统中，short 型数据占 2 字节，int 型数据和 long 型数据占 4 字节。

2）整型数据以二进制形式存储，如十进制数 65 的二进制为 01000001，在内存中的存储形式如图 1.2.1 所示。

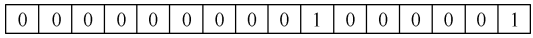


图 1.2.1　整型数据的存储形式

3）整型数据和字符型数据都有带符号和无符号两种形式，分别由修饰符 signed 和 unsigned 表示。如果指定为 signed，则数值以补码形式存放，存储单元最高位表示数值的符号。如果指定为 unsigned，则数值没有符号，全部二进制都用来表示数值本身。占 2 个字节带符号短整型和无符号短整型的存储情况如图 1.2.2 所示。

图 1.2.2　有无符号短整型数据的存储情况

4）浮点数有单精度（float）、双精度（double）和长双精度（long double）之分。在 Visual C++ 6.0 中，float 有 6 位有效数字，double 有 15 位有效数字；float 占 4 字节，double 和 long double 占 8 字节。

数据类型不但规定了使用该类型时的取值范围，还规定了该类型可以使用的一组操作。例如，与整型有关的操作有+、–、*、\、%等。C++不但定义了一些基本的数据类型，还提供了复合的数据类型，如数组、结构体、共用体、类等，程序员可以利用这些复合的数据类型，自行定义一些实际所需要的数据类型，例如，程序员可定义自己的线性表类、栈类等数据类型。

1.2.2　数据抽象与抽象数据类型

在面向对象的程序设计中，常常提到“抽象”一词，那么，什么是抽象呢？抽象的本质就是抽取反映问题本质的东西，忽略非本质的细节。

抽象数据类型通常是指由用户定义，用来表示应用问题的**数据模型**。抽象数据类型由基本的数据类型组成，并包括一组相关的操作，抽象数据类型类似于 C++中的类。对于一个数据成员完全相同的数据类型，如果给它定义不同的功能，则可形成不同的抽象数据类型。

抽象数据类型的特点是使用与实现分离，实行封装和信息隐蔽。在抽象数据类型设计时，把类型的声明与其实现分离开来。首先根据问题的要求，定义该抽象数据类型需要包含哪些信息，并根据功能确定公共界面的服务，使用者可以使用公共界面的服务对该抽象数据类型进行操作。此外，抽象数据类型的具体实现作为私有部分封装在其实现模块内，使用者不能

看到，也不能直接操作该类型所存储的数据，只能通过界面中的服务来访问这些数据。

从实现者的角度来看，把抽象数据类型的具体实现封装起来，有利于编码、测试，也有利于将来的修改。因为这样做可以使得错误局部化，一旦出现错误，其传播范围不至于影响其他模块，如果为了提高效率希望改进数据结构，可能需要改变抽象数据类型的具体实现，但只要界面中的服务的使用方式不变，其他所有使用该抽象数据类型的程序都可以不变，从而大大提高了系统的稳定性。

从使用者的角度来看，只要了解该抽象数据类型的规格说明，就可以利用其公共界面中的服务来使用这个类型，而不必关心其物理实现，这样使用者可以在开发过程中抓住重点，集中精力考虑如何解决应用问题，使问题得到简化。例如，我们在求解一个最优化问题时常常要使用一个栈，那么，我们应当首先考虑此栈应存放什么信息、应如何组织，至于栈怎样实现、可能会出现哪些例外情况、这些例外情况如何处理等，可以忽略，直接调用堆栈类提供的相关服务即可。

1.3 算法和算法分析

1.3.1 算法

数据结构除了要研究数据的逻辑结构和存储结构外，还要研究如何在数据的各种结构上实施有效的操作或处理，这就涉及算法。算法是对特定问题求解步骤的一种描述，它是指令的有限序列，其中每一条指令表示一个或多个操作。一个算法应具有下列五个重要特性。

1）**有穷性**：对任何合法的输入值，一个算法必须在执行有穷步之后结束，且每一步都应该在有穷时间内完成。

2）**确定性**：算法中每一条指令必须有确切的含义，读者理解时不会产生二义性，在任何条件下，算法只有唯一的一条执行路径，对于相同的输入只能得出相同的输出。

3）**可行性**：一个算法是可行的，即算法中描述的操作都可以通过已经实现的基本运算执行有限次来实现。

4）**输入**：一个算法有 0 个或多个输入，这些输入取自于某个特定的数据对象的集合，它可以使用输入语句从外部提供，也可以在算法内通过赋初值给定。

5）**输出**：一个算法有一个或多个输出，这些输出是同输入有着某些特定关系的量。

1.3.2 算法设计的要求

要高质量、高效率地完成好的算法或好的代码，需要编程人员养成好的编程习惯、编程风格。通常一个好的算法应考虑达到以下目标。

1. 正确性

算法应当满足具体问题的需求。通常一个大型问题的需求，要以特定的规格说明方式给出，而一个实习问题或练习题，往往就不那么严格。目前大多采用自然语言描述需求，它至少应当包括对于输入、输出和加工处理等明确的无歧义性的描述。设计或选择的算法应当能满足这种需求。

正确性大体可分为以下 4 个层次。

1）程序不含语法错误。

2）程序对于几组输入的数据能够得出满足规格说明要求的结果。

3）程序对于精心选择的、典型的、苛刻的几组输入数据能够得出满足规格说明要求的结果。

4）程序对于一切合法的输入数据都能产生满足规格说明要求的结果。

要达到第四层含义下的正确是极为困难的，所有不同输入数据的数据量大得惊人，逐一验证的方法是不现实的。对于大型软件需要进行专业测试，而在一般情况下，通常以第 3 层意义的正确性作为衡量一个程序是否合格的标准。

为保证程序的正确性，需要掌握一定的测试、调试技巧。利用这些技巧，可以事半功倍地发现、排除编码错误，因而，当使用一个编程工具或开发环境时，要注重对其提供的调试功能的了解和使用。一个虽然简单但是非常有用的基本调试方法是对代码设置断点，使得代码运行到断点处停止，这样，程序的整个运行过程将不再是一个不可控的黑匣子。当程序暂停于断点处时，编程者可以查看各种变量或内存状况从而帮助自己分析代码问题，保证程序的正确性或其他质量。

2. 可读性

算法主要是为了人的阅读与交流，其次才是机器执行。可读性好有助于人对算法的理解，晦涩难懂的程序易于隐藏较多错误，往往难以调试和修改。现代软件注重代码的复用，以期更加高效地对软件进行扩展。良好的可读性，不但是代码团队协作开发的需要，也是代码复用的需要。可读性涉及很多方面，如算法本身的逻辑是否清晰易懂，程序结构是否明晰合理，等等。但是作为初学者，可以从一些基本的细节开始，注重培养保持代码可读性的习惯。例如，对函数、重要变量的命名，尽量含义清晰，遵守变量命名的一些习惯（例如，指针前面加“p”），合理地加上代码注释，等等。一些语法规则的运用，既有利于代码的安全性，也能提高代码的可读性。例如，在函数传递的参数前，如果加上了 const 限定，既可以防止函数内部错误地修改该变量，又可告诉代码阅读者此变量在函数体内不需要改变，因而提高了代码的可读性。后面将结合代码示例，适当介绍提高可读性的相关知识。

3. 健壮性

当输入数据非法时，算法也能适当地做出反应或进行处理，而不会产生莫名其妙的结果，处理错误的方法可以是返回一个表示错误或错误性质的值。除了对非法输入的容错之外，健壮性对其他异常情况，如内存异常、磁盘文件异常、网络异常等也要有容错性。另外，健壮性还包括对程序长期稳定运行的要求。有些程序启动后需要长时间运行，因此如果没有很好地管理内存、资源，将会导致程序消耗的内存或资源越来越多,从而引起程序运行迟缓、系统崩溃等问题。还有些程序没有很好地管理产生的临时文件，也会导致时间长了以后运行迟缓、占用存储空间过多等问题。所以，良好的编程的风格和习惯，对提高代码的健壮性很有帮助。

4. 效率

效率指的是算法执行时计算机资源的消耗，它包括运行时间代价和存储空间代价。对于同一个问题，如果有多个算法可以解决，则执行时间短、存储量需求小的算法效率高。效率与问题的规模和性质有关，求 10 个人的平均工资与求 1000 个人的平均工资所花的执行时间或运行空间显然有一定的差别。有时候，程序的效率与可读性、健壮性等是有竞争的，这时需要结合使用背景、程序生命周期等因素进行综合考虑，取得平衡。接下来，我们将重点讨论算法的效率。

1.3.3 算法效率的度量

算法的效率包括算法运行时间代价和存储空间代价，它们分别由**时间复杂度和空间复杂度**来度量。

1. 算法的时间复杂度

算法执行时间需依据该算法编制的程序在计算机上运行时所消耗的时间来度量。而度量

一个程序的执行时间通常有两种方法，即事后统计的方法和事前分析估算的方法。

（1）事后统计的方法

很多计算机内部都有计时功能，有的甚至可精确到毫秒级，不同算法的程序可通过一组或若干组统计数据来分辨优劣。但这种方法有两个缺陷，一是必须先运行依据算法编制的程序；二是统计数据依赖于计算机的硬件、软件等环境因素，有时容易掩盖算法本身的优劣。因此，人们常常采用另一种事前分析估算的方法。

（2）事前分析估算的方法

一个用高级程序语言编写的程序在计算机上运行时所消耗的时间取决于下列因素。

① 问题的规模。

② 算法选用的策略。

③ 书写程序的语言，对于同一个算法，实现语言的级别越高，执行效率就越低。

④ 编译程序所产生的机器代码的质量。

⑤ 机器执行指令的速度。

显然，同一个算法用不同的语言实现，或者用不同的编译程序进行编译，或者在不同的计算机上运行时，效率均不相同。这表明使用绝对的时间单位衡量算法的效率是不合适的。撇开这些与计算机硬件、软件有关的因素，可以认为，一个特定算法的运行时间，只依赖于问题的规模，或者说，它是问题规模的函数。

一个算法是由控制结构（顺序结构、分支结构和循环结构）和原操作（指固有数据类型的操作）构成的。算法的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，以该基本操作的重复执行的次数作为算法的时间度量。

一般情况下，算法中基本操作重复执行的次数是问题规模 n 的某个函数 $f(n)$，算法的时间量度记为

$$T(n) = O(f(n)) \tag{1.3.1}$$

式（1.3.1）表示随问题规模 n 的增大，算法执行时间的增长率和 $f(n)$ 的增长率相同，$T(n)$ 称为算法的渐近时间复杂度，简称**时间复杂度**。

在大多数情况下原操作是最深层循环内的语句中的原操作，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：

1）
```
++ x;
```

2）
```
for （ i = 0； i < n； i ++） x ++;
```

3）
```
for（ i = 0; i < n； i ++）
        for（ j=0； j < n； j ++）
            x ++;
```

以上三例中，含基本操作“x ++”的语句的频度分别为 1、n 和 n^2，这三个程序段的时间复杂度相应为 $O(1)$，$O(n)$ 和 $O(n^2)$，分别称为常量阶、线性阶和平方阶。算法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。不同数量级时间复杂度的性状不同，我们应该尽可能选用多项式阶 $O(n^k)$ 的算法，而不希望用指数阶的算法。算法复杂度示意图如图 1.3.1 所示。

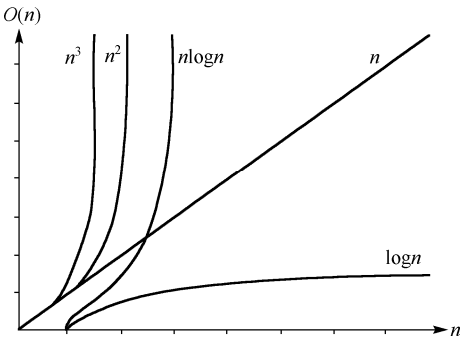


图 1.3.1　算法复杂度示意图

一般情况下，对一个问题（或一类算法）只需选择一种基本操作来讨论算法的时间复杂度即可，有时也需要同时考虑几种基本操作，甚至可以对不同的操作赋以不同的权值，以反映执行不同的操作所需要的相对时间，这种做法便于综合比较解决同一问题的两种完全不同的算法。

有的情况下，算法中基本操作重复执行的次数还随问题的输入数据集不同而不同，如下面起泡排序法的算法：

```
void bubblesort(int *a, int n)
{   for( i = n-1, flag =1; i >=1&& flag; --i )
    {   flag = 0;
        for( j = 0; j < i; ++ j )
            if( a[ j ] > a[ j+1] )
            {   t = a[ j ], a[ j ] = a [ j+1], a[ j+1] = t;
                flag =1;
            }
    }
 }
```

此例中，“交换序列中相邻的两个整数”是基本操作。当数组 a 中初始系列为自小到大有序时，基本操作的执行次数为 0。当初始系列为自大到小有序时，基本操作的执行次数为 n（n–1）/2。对这类算法的分析，一种解决方法是计算它的平均值，即考虑它对所有可能的输入数据集的期望值，此时相应的时间复杂度为算法的平均时间复杂度。然而，在很多情况下，各种输入数据集出现的概率难以确定，算法的平均时间复杂度也难以确定。因此，另一种更可行也更常用的办法是讨论算法在最坏情况下的时间复杂度，例如，上述起泡排序的最坏情况为数组 a 中初始序列为自大到小有序，则起泡排序算法在最坏情况下的时间复杂度为 $T(n) = O(n^2)$。在本书以后各章中讨论的时间复杂度，除特别指明外，均指**最坏情况**下的时间复杂度。

实际中我们可以把事前估算和事后统计两种办法结合起来使用。以两个矩阵相乘为例，若上机运行两个 10×10 的矩阵相乘，执行时间为 12ms，则由算法的时间复杂度 $T(n) = O(n^3)$ 可估算两个 20×20 的矩阵相乘所需时间大致为

$$(20/10)^3 \times 12 = 96\text{ms}$$

2. 算法的空间复杂度

类似于算法的时间复杂度，以**空间复杂度**作为算法所需存储空间的量度，记为

$$S(n) = O(f(n)) \qquad (1.3.2)$$

式（1.3.1）中，n 为问题的规模（或大小）。

一个上机执行的程序除了需要存储空间来存储本身所用指令、常数、变量和输入数据外，也需要一些辅助空间存储一些所需的中间信息。若输入数据所占空间只取决于问题本身，和算法无关，则只需要分析除输入和程序之外的额外空间，否则应同时考虑输入本身需要的空间。若额外空间是常数，则称此算法为原地工作。如果所占空间量依赖于特定的输入，则除特别指明外，均按**最坏情况**来分析。

1.4 面向对象概述

本节介绍面向对象的思想、面向对象的基本概念以及面向对象程序设计的基本特征。

1.4.1 面向对象的思想

面向对象的方法是为计算机软件的创建提出的一种模型化世界的抽象方法，其基本思想如下：尽可能地运用人类的自然思维方式来建立问题空间的模型，构造尽可能直观、自然地表达问题求解方法的软件系统。现实世界中的问题是由客观实体和实体之间的联系构成的，对象就是客观实体的抽象。面向对象方法将数据和操作放在一起，作为一个相互依存、不可分割的整体来处理。

为了理解面向对象的含义，请看一个现实世界对象的例子——学生。“本科生”是更大的“学生”类的一个成员（称为“子类”）。所有的学生都有一些共同的属性，如姓名、性别、年龄、学习成绩、学分等，这些属性对于“学生”类的成员，如研究生、本科生、专科生等总是可以使用的。因为本科生是“学生”类的成员，“本科生”继承了“学生”类所定义的一切属性和操作。

一旦某个类被定义，当该类的一个新子类被创建时，属性可以被复用。例如，定义一个新的称为“研究生”的对象，它也是“学生”类的成员，“研究生”也继承了“学生”类的所有属性，如姓名、性别、年龄、学习成绩等。

在“学生”类的每个对象上可以定义一系列操作，如入学注册、选课等。这些操作将修改对象的一个或多个属性，如选课操作将修改学习成绩和学分这两个属性的值。软件工程专家用如下等式简明地描述“面向对象”，面向对象＝对象＋分类＋继承＋消息通信。也就是说，面向对象就是既使用对象又使用类和继承等机制，而且对象之间仅能通过消息的传递实现通信。如果一个软件系统使用了这四个概念来设计和实现，则这个软件系统就是面向对象的。

1.4.2 面向对象程序设计

高级语言程序设计包括面向过程和面向对象两种方法。面向过程的程序设计中数据是公用的或共享的，一个数据可以被多个函数调用，这些数据是缺乏保护的，同时数据的交叉使用很容易导致程序的错误。实际上，程序中的每一组数据都是为某一种操作准备的，即一组数据总是与一种操作相对应的。因此，人们设想将一组数据与一种操作放在一起，形成一个整体，与外界相对分隔，此即面向对象程序设计中的对象。与面向过程不同的是，基于对象的程序设计以类和对象为基础，程序的操作是围绕对象进行的。在此基础上利用继承机制和多态性，就称为面向对象的程序设计。基于对象的程序设计所面对的是一个个对象，所有的数据分别属于不同的对象。

在面向过程的程序设计中，常用公式“程序=数据结构+算法”来表示程序，同时将数据结构和算法分开来独立设计。但是，人们在实践中发现数据结构和算法是不可分割的，应当以一个算法对应一组数据结构或者一组数据结构对应多个算法。基于对象和面向对象的程序设计就是将一个算法和一组数据结构封装在一个对象实体中，形成了“对象=数据结构+算法”的新概念，因此，面向对象的程序设计的关键就是对象的设计和使用。

1.4.3 面向对象的语言

面向对象的程序设计语言必须支持抽象数据类型和继承性。面向对象的程序设计语言经历了一个漫长的发展过程。例如，LISP 语言、Simula 67 语言和 SmallTalk 语言，它们都或多或少地引入了面向对象的一些概念，如数据抽象、类结构与继承机制。目前，应用最广泛的面向对象程序设计语言是在 C 语言基础上扩充出来的 C++语言。C++对 C 的向后兼容，使得很多基于 C 的程序稍加修改就可以重用，许多有效的算法也可以重用。C++是一种混合型的面向对象程序设计语言，它的出现使得面向对象的各种语言越来越多地得到重视和广泛应用。

1.4.4 面向对象的基本概念

对象是面向对象程序设计的核心，正确地认识和定义对象，对进一步掌握面向对象的思想大有帮助。

1. 对象

对象是现实世界中存在的一个事物，对象可以是具体的有形物体，如学生、房屋、汽车等，也可以是无形的事物或概念，如国家、生产计划等。对象是构成世界的一个独立单位，它具有自己的静态特征（也称属性，描述内部状态的数据）和动态特征（具有的功能或行为）。从系统建模和实现的角度，对象描述了客观事物的一个实体，是构成系统的一个独立的基本单元，它由一组属性（数据）和一组服务（操作）组成。系统的服务是通过新对象的建立和对象之间的消息通信来完成的。对象中的服务（操作）是对象动态特征的体现，它常是一个可执行的语句或过程，对属性进行操作，实现某种服务。从系统实现的角度，可以把对象看做由一组数据以及有权对这些数据施加操作的一组服务封装在一起构成的统一体，即对象＝数据＋动作（方法、操作）。

在面向对象程序设计中，对象包含两方面的含义：对象的属性和它的行为。属性是指对象的自然属性或自身状态，如人的年龄、身高、肤色和体重等。行为是指对象的功能，如人的特长或技能等。对象是其自身所具有的特征以及可以对这些状态施加的操作结合在一起构成的一个独立实体，它具有以下特征。

1）有一个名称以区别于其他对象。

2）有一个状态用来描述它的一些属性。

3）有一组操作，每一个操作决定对象的一种功能或行为。

对象的操作可以分为两类：一类是自身所承受的操作，另一类是施加于其他对象的操作。

例如，有一个人叫李玉，身高 1.65m，体重 60kg，可以教高等数学，会程序设计。下面是这个对象的特性描述。

对象名：李玉

对象的状态：

身高 1.65m；

体重 60kg。

对象的功能：

回答身高
回答体重　} 自身所承受的操作

教高等数学课
会程序设计　} 施加于其他对象的操作

2. 对象的特性

对象具有以下 3 个特性。

1）模块的独立性。利用封装技术将对象的特性隐藏在模块内部，使用者只需了解它的功能，不必知道功能实现的细节。因此，外界的变化不会影响模块内部的状态，各模块可独立为系统所组合选用，也可被程序员重用。

2）动态的连接性。可以通过消息激活机制，将对象之间动态联系连接在一起，使整个机体运转起来。

3）易维护性：由于对象的数据和操作都被封装在模块内部，所以对它们的修改及完善都限制在模块内部，不会涉及对象外部，从而使整个对象和整个系统变得更易于维护。

3. 消息

消息（Message）是对象之间在交互中所传送的通信信息。在 C++中，消息的具体表现为函数，它是对象之间相互请求和协同工作的手段，是请求某个对象执行其中某个功能操作的规格说明，对象之间的联系只能通过传递消息来进行。只有当对象接收到消息时，才会激活有关的对象代码，“知道”如何去操作它的私有数据以完成所要求的服务操作。

消息具有 3 个性质。

1）同一对象可接收不同形式的多个消息，并产生不同的响应。

2）相同形式的消息可以传递给不同类型的对象，所产生的响应可能截然不同。

3）消息的发送可以不考虑具体的接收者，对象可以响应消息，也可以不响应消息。

在面向对象系统中，为了完成某个事件，有时需要向同一个对象发送多个消息或者向不同的对象发送多个消息，我们把多个消息称为消息序列。对一个对象而言，由外界对象直接向它发送的消息称为公有消息；由它自己向本身发送的消息称为私有消息。

某个特定对象的消息，根据功能的不同可分为以下 3 种类型。

1）可以返回对象内部状态的消息。

2）可以改变对象内部状态的消息。

3）可以完成一些特定操作，并改变系统状态的消息。

4. 类

类（Class）是一种具有相同属性和相同操作的对象的集合。一个类就是对一组相似对象的共同描述，它整体地代表一组对象。类封装了对描述某些现实世界对象的内容和行为所需要的数据和服务（操作）的抽象，它给出了属于该类的全部对象的抽象定义，包括类的属性、服务（操作）。对象只是符合某个类定义的一个实体，属于某个类定义的一个具体对象称为该类的一个实例（Instance）。可以把类看做某些对象的模板（Template），它抽象地描述了属于该类的全部对象的共有属性和操作。类与对象的关系是抽象与具体的关系，类是多个对象（实例）的抽象，对象（实例）是类的个体实物。例如，王洋是一位教师。教师是一个类，属于抽象的概念；而王洋是教师类的一个具体对象，即教师类的实例。

面向对象的系统设计主要归结为类的建立和使用，类的确定采用归纳法来完成。在系统设计中，通过对相同性质的物质对象进行类比分析，归纳出它们的共性，包括数据特性和行为特性，构建一个类。C++的类是对所研究问题的若干对象的抽象描述，它对逻辑上相关的数据和函数进行封装。

1.4.5 面向对象的基本特性

在面向对象的程序设计中，几个核心的概念是封装性、继承性和多态性。

1. 封装性

封装（Encapsulation）是将一段程序代码“包装”起来，应用时只需要知道这段代码所完成的功能，而不必知道该功能的实现细节。类和对象是实现封装的重要机制。类是对具有相同属性的客观对象的抽象描述，并将抽象出来的数据和操作行为封装在类中，构成一个不可分割的、独立的整体。在类中将一部分代码作为对外部的接口，而将数据和其他行为尽可能隐蔽。外界只能通过外部接口才能访问封装在类中的数据，从而实现了对数据访问权限的有效控制。

封装的目的是将对象的设计者和对象的使用者分开。使用者只需要知道对象所表现的外部行为，利用设计者提供的消息来访问该对象，而不必了解对象行为的内部实现细节。封装作为面向对象方法的一种信息隐蔽技术，应具有以下几个条件。

1）有一个清楚的边界，对象的所有私有数据和服务（操作）都被限定在该边界内，外界是不可直接访问的。

2）至少有一个接口，这个接口描述了该对象与其他对象之间的相互请求和响应的消息格式和功能。

3）对象行为的内部实现细节是隐蔽的，即其他对象不能直接修改该对象所拥有的相关数据和程序代码。

2. 继承性

继承（Inheritance）是面向对象系统的另一个重要特性。类和类之间有时相互独立，有时会出现一些相似的特征。继承所表达的就是对象之间的一种相交关系，它使得子类可以自动拥有、共享父类的全部属性与服务（操作），即某类对象可以继承另一个类的特征和能力。也就是说，原本为父类设计的数据结构和算法，可无条件地被子类使用。

C++为用户提供了类的继承机制，允许在保持原有类（基类）特性的基础上，为其继承类（子类）进行更具体、更详细的类的定义。也就是说，可以定义一个包含公共成员的基类，通过继承从基类中派生出其他类，为新类增添新的属性和操作，还可以改写基类的部分内容。

继承分为单继承和多继承，从一个基类派生出新类，称为单继承；从多个基类派生出新类，称为多继承。由某个类派生所需要的任意多个类，或者类之间通过单继承和多继承派生出多个类，这样就形成了类的层次关系。在面向对象系统中，引入继承机制后，主要有以下优点。

1）能清晰地体现相关类之间的层次关系。

2）通过继承关系，使子类可以自动共享父类的全部属性与服务特性，提高了程序的复用性。

3）通过增强一致性来减少模块之间的接口和界面，提高了程序的可维护性。

4）继承可以实现对类的扩充，为程序的扩展和可用性提供了有效方法，有利于软件系统的逐步细化。

3. 多态性

所谓多态性，是指同一名称（函数、运算符）在不同的场合具有不同的意义；或者同一个界面，有多种实现。前者称为重载，后者称为虚函数。C++中的多态性，是指同一消息形式，可以根据发送消息对象的不同，采用不同的行为方式。C++语言支持以下两种多态性。

1）编译时的多态性：通过重载来实现。C++允许为已有的函数和运算符重新赋予新的含义，就是说具有相同名称的函数和运算符在不同的场合可以表现出不同的行为，即函数和运算符重载。定义函数重载时，要求函数名相同，但是函数所带的参数类型或个数必须有所区别，否则会出现二义性。

2）运行时的多态性：通过虚函数来实现。C++的虚函数使得用户可以在同一个类族（基类及其各级派生子类）中使用同名函数的不同版本，每个函数均属于同一个类族中不同的类。究竟使用哪个版本，需要在运行中决定。虚函数的各个版本中，要求函数返回值、函数参数的类型和个数必须一致。

本书以面向对象的观点来介绍数据结构，因此，我们对每一种数据结构都采用类的方式来描述。由于在软件开发和维护过程中需要花费大量的人力和时间，如果能够重复利用以前开发完成的数据结构或算法模块，把它们有选择地组装在新的软件中，将可以大大减少需花费的人力和时间，这就是软件复用问题。为了实现软件的复用，一个重要的方法就是使用 C++模板，写出适合多种数据类型的类定义或算法，然后在特定环境中通过简单的实例化，把它们变成针对具体某种数据类型的类定义或算法。在以后的讨论中，大部分采用模板机制来实

现各种数据结构。

1.5 本 章 小 结

本章介绍的基本内容包括数据结构的基本概念、抽象数据类型、算法和算法分析以及面向对象概念等。这些内容是读者进一步学习后继内容的基础。

分析待处理对象的特性以及它们之间存在的关系是编写好的程序的基础，因此，分析数据对象之间的逻辑关系，并用计算机存储结构体现出这些逻辑结构并操作这些数据，是数据结构这门课程要解决的问题。

抽象数据类型将使用与实现分离，实行封装和信息隐蔽，有利于编码、测试和修改。

数据结构在研究数据的逻辑结构和存储结构基础上，还要研究如何在数据的各种结构上实施有效的操作或处理，因此，算法设计及实现十分重要。算法的设计主要与数据的逻辑结构密切相关，而算法的实现则与数据的存储结构密切相关。算法的效率由算法的时间复杂度和空间复杂度衡量。

面向对象的方法能很好地描述数据结构，面向对象程序设计的基本特性将一个算法和一组数据结构封装在一个对象实体中，形成了“对象=数据结构+算法”的新概念，可方便地实现各种数据结构。

习 题 1

1.1 什么是数据结构？数据结构主要讨论哪几方面的问题？

1.2 什么是数据结构的物理结构和逻辑结构？它们的主要区别是什么？

1.3 数据之间的逻辑关系分为哪几类？它们各自的特点是什么？

1.4 什么是抽象数据类型？

1.5 什么是算法？算法的 5 个特性是什么？试比较算法与程序的区别。

1.6 什么是面向对象的程序设计？它与结构化程序设计有什么不同？

1.7 解释以下概念：

（1）对象 （2）消息 （3）类 （4）实例 （5）公有消息 （6）私有消息

1.8 对象有哪些特性？对象是如何确定和划分的？

1.9 类与实例的关系如何？

1.10 叙述面向对象系统的特性。这些特性在面向对象系统中有什么作用？