Chapter
1

项目一

数据结构基本知识

项目要点

- 数据结构
- 数据存储结构
- 算法及算法分析
- C语言基础

引 言

在本项目中，通过1个工作任务，向读者展示什么是数据结构，从而展开对数据结构、数据存储、算法等知识的讲解，为后续项目的完成打下坚实基础。

任务：建立数学模型

任务描述

使用数据结构知识，为企业员工信息管理、组织机构管理、学校排课等问题建立数学模型，从而实现使用计算机解决这些问题。

任务分析

建立数学模型的实质是对现实事物和问题的分析，从中提取出对象以及对象之间特有的关系。计算机能处理的数据也不再是简单的数值，而是字符串、图形、图像和语音等复杂数据。这些复杂数据不仅量大，而且具有一定的结构。数据结构所研究的就是这些有结构的数据。

准备知识

“数据结构”作为一门独立的课程在国外是从 1968 年才开始设立的。美国唐·欧·克努特教授在 1968 年开创了数据结构的最初体系。“数据结构”在计算机科学中是一门综合性的专业基础课，是介于数学、计算机硬件和计算机软件三者之间的一门核心课程。数据结构是计算机各专业必修的核心课程，是研究计算机程序设计的重要理论和技术的专业基础课程。

1. 数据结构概述

随着电子技术的发展，计算机逐渐深入到商业、制造业等人类社会各个领域，从而广泛地应用于数据处理和过程控制。与此相应，计算机能处理的数据也不再是简单的数值，而是字符串、图形、图像和语音等复杂数据。这些复杂数据不仅量大，而且具有一定的结构。数据结构所研究的就是这些有结构的数据，是研究数据对象的特征以及数据的组织和处理方式。在计算机科学中，数据结构（Data Structure）是计算机中存储、组织数据的方式，是指相互之间存在一种或多种特定关系的数据元素的集合。通常情况下，精心选择的数据结构可以带来最优效率（Algorithmic Efficiency）的算法，数据结构往往同高效的检索算法和索引技术有关。

知识链接

一般而言，数据结构的选择首先会从抽象数据类型的选择开始。一个设计良好的数据结构，应该在尽可能使用较少的时间与空间资源的前提下，为各种临界状态下的运行提供支持。数据结构可通过编程语言所提供的数据类型、引用（Reference）及其他操作加以实现。不同种类的数据结构适合于不同种类的应用，而部分甚至专门用于特定的作业任务。例如，当计算机网络依赖于路由表运作时，B 树高度适用于数据库的封装。

在许多类型的程序设计中，选择适当的数据结构是一个主要的考虑因素。许多大型系统的构造经验表明，封装的困难程度与最终成果的质量与表现，都取决于是否选择了最优的数据结构。在许多时候，确定了数据结构后便能很容易地得到算法。而有些时候，方向则会颠倒过来：例如当某个关键作业需要特定数据结构下的算法时，会反过来确定其所使用的数据结构。然而，不管是哪种情况，数据结构的选择都是至关重要的。

2. 基本术语

数据 (Data)：计算机中的数据是广义的，包括了数、字符、字符串、表、文件等，声音、图形、图像都属于数据的范畴。数据指计算机加工的“原料”，是对客观事物采用计算机能够识别、存储和处理的形式所进行的描述。数据就是计算机化的信息，是计算机中符号化的特定表示形式。

数据元素 (Data Element)：是数据的基本单位，是数据集合的个体，在计算机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。此时的数据元素通常称为记录 (Record)。比如例 1–1 中描述的某一条员工信息。

数据项 (Data Item)：是不可分割的、含有独立意义的最小数据单位，数据项有时也称为字段 (Field) 或域。如员工记录中的姓名、性别等。

数据对象 (Data Object)：是性质相同的数据元素的集合。它是数据的一个子集。例如，整型数据对象是集合 {0, ± 1, ± 2,…}，字符数据对象是 {a,b,c,…} 等，数据对象可以是有限的，也可以是无限的。

数据结构 (Data Structure)：是相互之间存在一种或多种关系的数据元素的集合。例如表 1–1 员工基本信息中数据元素集合是员工记录集，而数据元素集合上的关系就是它们在员工基本信息表中的前驱和后继的关系，这种数据之间的关系是一对一的，是线性的。数据元素相互之间的关系称为结构 (Structure)。

数据类型 (Data Type)：是高级程序设计语言中的概念，是数据的取值范围和对数据进行操作的总和。

例如 C 语言中的整型及定义在其上的一组操作 (加、减、乘、除等)。数据类型是一组性质相同的值集合以及定义在这个值集合上的一组操作的总称。数据类型中定义了两个集合，即类型的取值范围和该类型可允许使用的一组运算。例如高级语言中的整型、实型、字符型就是已经实现的数据结构的实例。从这个意义上讲，数据类型是高级语言中允许的变量种类，是程序语言中已经实现的数据结构 (即程序中允许出现的数据形式)。

拓展提高

在 C 语言中，整型类型 (int) 可能的取值范围是 -32768 ~ +32767，可用的运算符集合为加、减、乘、除、取模 (+、−、*、/、%)。从硬件的角度来看，它们的实现涉及“字”、“字节”、“位”、“位运算”等；从用户的观点来看，并不需要了解整数在计算机内是如何表示、运算细节是如何实现的，用户只需要了解整数运算的外部运算特性，而不必了解整数在计算机在内部运算的细节，就可运用高级语言进行程序设计。

抽象数据类型 (Abstract Data Type，ADT)：是指一些数据以及对这些数据所进行的操作的集合。数据之间有一定的关系，不同的抽象数据类型的数据之间的关系不同。对数据进行的操作由数据间的关系决定。数据间的关系不同，对数据的操作也就不同。这些操作既向程序的其余部分描述了这些数据是怎么样的，也允许程序的其余部分改变这些数据。一个 ADT 可能是一个图形窗体以及所有能影响到该窗体的操作，也可以是一个文件以及对这个文件进行的操作，或者是一张保险费率表及相关操作等。

抽象数据类型是基于一类逻辑关系的数据类型以及定义在这个类型之上的一组操作。抽象数据类型的定义取决于客观存在的一组逻辑特性，而与其在计算机内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，都不影响其外部使用。从某种意义上讲，抽象数据类型和数据类型实质上是一个概念。整数类型就是一个简单的抽象数据类型实例。“抽象”的意义在于数学特性的抽象。一个抽象数据类型定义了一个数据对象，数据对象中各元素间的结构关系，以及一组处理数据的操作。

知识链接

抽象数据类型通常是指由用户定义用以表示应用问题的数据模型，通常由基本的数据类型组成，并包括一组相关的操作。

抽象数据类型包括定义和实现两方面，其中定义是独立于实现的。定义仅给出一个抽象数据类型的逻辑特性，不必考虑如何在计算机中实现。抽象数据类型的特征是使用与实现分离，实现封装和信息隐蔽，也就是说，在抽象数据类型设计时，类型的定义与其实现分离。另外，抽象数据类型的含义更广，不仅限于各种不同的计算机处理器中已定义并实现的数据类型，还包括设计软件系统时用户自己定义的复杂数据类型。所定义的数据类型的抽象层次越高，含有该抽象数据类型的软件复用程度就越高。抽象数据类型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，用户可以使用公共界面中的服务对该抽象数据类型进行操作。从用户的角度来看，只要了解该抽象数据类型的规格说明，就可以利用其公用界面中的服务来使用这个类型，不必关心其物理实现，从而集中考虑如何解决实际问题。

抽象数据类型是近年来计算机科学中提出的最重要的概念之一，它集中体现了程序设计中一些最基本的原则：分解、抽象和信息隐藏。严格地可以用代数系统形式定义一个抽象数据类型，可以把抽象数据类型看成是定义了一组运算的数学模型。

ADT 的定义采用下述格式：

```
ADT<ADT 名 >
{
   数据对象 : < 数据对象的定义 >
   数据关系 : < 结构关系的定义 >
   基本操作 : < 基本操作的定义 >
} ADT<ADT 名 >
```

其中数据对象和结构关系的定义采用数学符号和自然语言描述，而基本操作的定义格式为 C 语言基本函数的定义形式：

```
函数类型　函数名（[ 参数列表 ]）
[ 类型定义列表 ]
{
    函数体
}
```

参数表中的参数有两种：第一种参数只为操作提供待处理数据，又称值参；第二种参数既能为操作提供待处理数据，又能返回操作结果，也称变量参数。操作前提描述了操作执行之前数据结构和参数应满足的条件，操作结果描述操作执行之后，数据结构的变化状况和应返回的结果。抽象数据类型可用现有计算机语言中已有的数据类型，即固有数据类型来表示和实现。

在“组织机构管理问题”中描述的数据元素集合是组织机构的集合，而组织机构之间的关系就是数据元素之间的关系。这种数据元素相互之间的关系也是一种结构，是层次结构或称树状结构。

数据结构的形式定义为：

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。

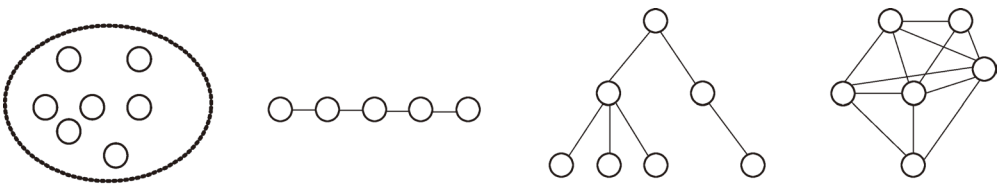
根据数据元素之间关系的不同特性，通常有下列 4 类基本的数据结构，如图 1-1 所示。

①集合 (Set)：该结构中的数据元素除了存在“同属于一个集合”的关系外，不存在任何其他关系，如图 1-1a 所示。

②线性结构 (Linear Structure)：该结构中的数据元素存在着一对一的关系，如图 1-1b 所示。

③树形结构 (Tree Structure)：该结构中的数据元素存在着一对多的关系，如图 1-1c 所示。

④图状结构 (Graphic Structure)：该结构中的数据元素存在着多对多的关系，如图 1-1d 所示。



(a) 集合　(b) 线性结构　(c) 树形结构　(d) 图状结构

图 1-1　基本的数据结构

有时也称树状结构和图状结构为非线性结构。

某课题小组中有一位教授，教授可以带 1 ～ 3 名研究生，每名研究生可以带 1 ～ 2 名本科生。设计课题小组的数据结构。

设计课题小组的数据结构为：Group = (P, R)

其中，P 表示数据对象；R 表示数据对象中数据元素的关系。

P、R 的描述如下：

P = {T, G1, …, Gn, S11, …, Snm} (1 ≤ n ≤ 3, 1 ≤ m ≤ 2)

R = {R1, R2}

R1 = {<T, Gi> | 1 ≤ i ≤ n, 1 ≤ n ≤ 3}

R2 = {<Gi, Sij> | 1 ≤ i ≤ n, 1 ≤ j ≤ m, 1 ≤ n ≤ 3, 1 ≤ m ≤ 2}

其中，T 表示教授；G 表示研究生；S 表示大学生。

严格地说，数据结构包括两方面的内容：数据的逻辑结构 (Logical Structure) 和数据的物理结构 (Physical Structure)。Group 定义的数据结构是对操作对象的一种数学描述，其中“关系”是描述数据元素之间的逻辑关系，因此称为数据的逻辑结构。与逻辑结构相对应的是数据的物理结构。数据的物理结构又称为数据的存储结构，是指数据的逻辑结构在计算机中的映像 (又称表示)，即数据结构在计算机中的存储方法。由于数据结构包括数据元素集合及数据元素之间的关系，所以数据的存储结构也应该包含这两部分内容，即包括数据元素的映像和元素间关系的映像两部分。数据元素之间的关系在计算机中有两种不同的表示方法，即顺序映像和非顺序映像，并由此得到两种不同的存储结构，即顺序存储结构和链式存储结构。顺序存储结构的特点是借助于元素在连续空间的存储器中的相对位置来表示数据元素之间的逻辑关系。非顺序映像的特点是借助于指示元素存储地址的指针 (Pointer) 来表示数据元素之间的逻辑关系。

知识链接

任何一个程序都涉及数据的存储结构。因为对于逻辑结构上的同一种运算，其具体的实现方法依存储结构的不同而变化。所以，当描述问题的模型确定之后，首要问题就是确定其存储结构。

3. 数据的存储结构

数据的存储结构是建立一种由逻辑结构到存储空间的映射：对于逻辑结构 (K, r)，其中 r ∈ R，对它的结点集合 K 建立一个从 K 到存储器 M 的单元的映射：K → M，其中每个结点 j ∈ K 都对应一个唯一的连续存储区域 c ∈ M。

常用的数据存储结构有两类：一类是顺序存储结构，一类是链式存储结构。

（1）顺序存储结构

顺序存储结构是指逻辑上相邻的数据元素，其结点的物理位置也相邻，数据元素之间的关系由结点的邻接关系体现。顺序存储把一组结点存放在按地址相邻的存储单元里，结点间的逻辑关系用存储单元的自然顺序关系来表达的，即用一块存储区域存储线性数据结构。顺序存储法为使用整数编码访问数据结点提供了便利。因为，顺序

存储方法的存储空间除了存储有用数据外，没有用于存储其他附加的信息，所以顺序存储结构一般也被称为紧凑存储结构。计算机的内存单元是一维结构，这种存储结构很方便实现。

比如在本项目“企业员工信息管理问题”的题目中给出的员工信息逻辑结构，这个结构是线性的，各个员工记录前后是物理相邻的。

顺序存储结构有下列特点：

①结点中只存放数据元素本身的信息，无附加内容。

②由于每个结点所占存储空间大小一致，并且按顺序存储，所以只要知道第 1 个结点的地址，就可以通过计算直接确定结构中第 i 个结点的地址，从而直接存取第 i 个数据元素。

③由于可根据公式直接确定第 i 个结点的地址而直接存取第 i 个元素，所以数据元素的存取操作速度较快。

④插入、删除数据元素时，由于需要保持数据元素之间的逻辑关系，必须移动大量元素，因此实现起来较慢。关于这一点，将在第 2 章介绍线性表时详细说明。

⑤顺序存储结构有两种方式分配空间，一种是静态结构，另一种动态结构。当用静态结构时，存储空间一旦分配完毕，其大小就难以改变。因此，当表中元素个数难以估计时，分配的空间大小也就难以确定。预分配的空间太大会造成浪费，空间太小，又可能发生存放不下的溢出现象。

（2）链式存储结构

链式存储是在结点的存储结构中附加指针字段来存储结点间的逻辑关系。链式存储中数据结点包括两部分：数据字段存放结点本身的数据，指针字段存放指向其后继结点的指针。链式存储方法适用于那些需要经常进行增删结点的复杂数据结构。

在链式存储结构中，逻辑上相邻的数据元素，其结点的物理位置不一定相邻，因此结点之间是否邻接并不能反映数据元素的逻辑顺序。在这种情况下，存储结构要反映数据元素在逻辑结构中的关系，只有在结点中增加信息来指明与其他结点之间的这种关系，这个增加的信息就是指针。前一个结点的指针指向后一个结点，多个结点的指针一起形成一个链，因此称这种存储结构为链式存储结构。链式存储结构既可用于实现线性数据结构，也可用于实现非线性数据结构。对于非线性数据结构来说，每个数据元素在逻辑上可能与多个数据元素相邻，而计算机内存的一维地址结构限制了每一个结点只能与前、后各一个结点相邻，结点的相邻反映不出一个元素与多个元素的相邻关系，所以非线性逻辑结构只能用链式存储结构来实现。树、图等就是这种非线性数据结构。在链式存储结构的每个结点中，数据域可分为两类：一类用于存放数据元素本身的信息，称作信息域；另一类用于存放指针，称作指针域。

链式存储结构的特点主要有以下几方面。

①结点中除存放数据元素本身的信息外，还需存放附加的指针。

②不能直接确定第 i 个结点的存储位置。要存取第 i 个结点的信息，必须从第 1 个结点开始查找，沿指针顺序取出第 i–1 个结点的指针域，再取出第 i 个结点的信息，

存取速度较慢。

③链式存储结构的一个主要优点是插入、删除元素时不必移动其他元素，速度较快。因此当数据元素个数变动较大、插入删除操作频繁时可用链式存储结构来实现。

④链式存储是一种动态存储结构，当元素数量增加时可随时申请所需的空间，删除时无用的空间可归还给系统，故空间利用率较高，也不存在预分配空间的问题。

一般来说，一种数据结构既可用顺序存储结构实现，也可用链式存储结构实现。究竟用哪种存储结构，应根据具体情况来选择。选择时的主要依据有两个：一个是考虑数据结构上要执行的主要操作速度，另一个是对数据元素数目的估计。若执行的主要操作是插入或删除，则最好用链式存储结构，否则应该用顺序存储结构；若预先可以估计出元素的个数，则可以采用顺序存储结构，否则宜用链式存储结构。

拓展提高

数据结构的存储结构可采用顺序存储结构或链式存储结构，也可采用两者相结合的方式。除此之外，索引存储是顺序存储的一种推广，用于大小不等的数据结点的顺序存储。通过建造一个由整数域 Z 映射到存储地址域的函数，把整数索引值映射到结点的存储地址，从而形成一个存储一串指针的索引表，每个指针指向存储区域的一个数据结点。而散列法作为索引存储的一种延伸和扩展，利用散列函数进行索引值的计算，然后通过索引表求出结点的指针地址。

4. 算法及算法分析

（1）算法

算法 (Algorithm) 是规则的有限集合，是为解决特定问题而规定的一系列操作。算法是为完成某一特定任务的有限命令的集合。一个算法应该具备以下 5 个特性。

①有穷性 (Finity)。一个算法总是在执行有穷步之后结束，即算法的执行时间是有限的。

②确定性 (Unambiguousness)。算法的每一个步骤都必须有确切的含义，即无二义性，并且对于相同的输入只能有相同的输出。

③输入 (Input)。一个算法具有零个或多个输入，即它是在算法开始之前给出的量。这些输入是某数据结构中的数据对象。

④输出 (Output)。一个算法具有一个或多个输出，并且这些输出与输入之间存在着某种特定的关系。

⑤能行性 (Realizability)。算法中的每一步都可以通过已经实现的基本运算的有限次运行来实现。

知识链接

算法的含义与程序非常相似，但两者有区别。一个程序不一定满足有穷性。例如，操作系统，只要整个系统不遭破坏，它将永远不会停止。还有，一个程序只能用计算机语言来描述，也就是说，程序中的指令必须是机器可执行的，而算法不一定用计算机语言来描述，自然语言、框图、伪代码都可以描述算法。

（2）算法的评价

算法和数据结构密切相关。对于一个特定的问题，采用的数据结构不同，其设计的算法一般也不同，即使在同一种数据结构下，也可以采用不同的算法。数据结构的选择直接影响着算法的效率。那么，对于解决同一问题的不同算法，选择哪一种算法比较合适，以及如何对现有的算法进行改进，从而设计出更适合于数据结构的算法，这就是算法评价的问题。

评价一个算法一般从正确性、可读性、健壮性和时空效率几方面来考虑。评价一个算法优劣的主要标准如下。

①正确性 (Correctness)。算法的执行结果应当满足预先规定的功能和性能的要求，这是评价一个算法的最重要也是最基本的标准。算法的正确性还包括对于输入、输出处理的明确而无歧义的描述。

②可读性 (Readability)。算法主要是为了人阅读和交流，其次才是机器的执行。所以，一个算法应当思路清晰、层次分明、简单明了、易读易懂。即使算法已转变成机器可执行的程序，也需要考虑人能否较好地阅读理解。同时，一个可读性强的算法也有助于对算法中隐藏错误的排除和算法的移植。

③健壮性 (Robustness)。一个算法应该具有很强的容错能力，当输入不合法的数据时，算法应当能做适当的处理，使得不至于引起严重的后果。健壮性要求表明算法要全面细致地考虑所有可能出现的边界情况，并对这些边界情况做出完备的处理，尽可能使算法没有意外的情况。

④时间效率 (Running Time)。时间效率是指算法在计算机上运行所花费的时间，它等于算法中每条语句执行时间的总和。对于同一个问题，如果有多个算法可供选择，应尽可能选择执行时间短的算法。一般来说，执行时间越短，性能越好。

⑤空间存储量需求 (Storage Space)。空间存储量需求是指算法在计算机存储上所占用的存储空间，包括存储算法本身所占用的存储空间、算法的输入及输出数据所占用的存储空间和算法运行过程中临时占用的存储空间。算法占用的存储空间是指算法执行过程中所需要的最大存储空间，对于一个问题如果有多个算法可供选择，应尽可能选择存储量需求低的算法。实际上，算法的时间效率和空间效率经常是一对矛盾，相互抵触。在时间运用中要根据问题的需要进行灵活处理，有时需要牺牲空间来换取时间，有时需要牺牲时间来换取空间。

拓展提高

通常把算法在运行过程中临时占用的存储空间的大小称为算法的空间复杂度 (Space Complexity)。算法的空间复杂度比较容易计算，它主要包括局部变量所占用的存储空间和系统为实现递归所使用的堆栈占用的存储空间。

（3）常用的数学术语

①计量单位 (Unit)：按照 IEEE 规定的表示法标准，字节缩写为“B”，位缩写为“b”，兆字节 (2^{20}B) 缩写为“MB”，千字节 (2^{10}B) 缩写为“KB”。

②阶乘函数 (Factorial Function)：阶乘函数 n! 是指从 1 ～ n 之间所有整数的连乘，其中 n 为大于 0 的整数。因此，5!=1・2・3・4・5=120。特别地，0!=1。

③对数：一般地，如果 a(a ＞ 0，a ≠ 1) 的 b 次幂等于 N，就是 $a^b=N$，那么数 b 叫作以 a 为底 N 的对数，记作 $\log_a N=b$，其中 a 叫作对数的底数，N 叫作真数。从定义可知，负数和零没有对数。事实上，因为 a ＞ 0，所以不论 b 是什么实数，都有 a^b ＞ 0，这就是说不论 b 是什么数，N 永远是正数，因此负数和零没有对数。

编程人员经常使用对数，它有两个用途。第一，许多程序需要对一些对象进行编码，那么表示 n 个编码至少需要多少位呢？答案是 $\lceil \log_2 n \rceil$。例如，如果要存储 1000 个不同的编码，至少需要 $\log_2 1024=10$ 位 (10 位可以产生 1024 个不同的可用编码)。第二，对数普遍用于分析把问题分解为更小子问题的算法。在一个线性表中查找指定值所使用的折半查找算法就是这样一种算法。折半查找算法首先与中间元素进行比较，以确定下一步是在上半部分进行查找还是在下半部分进行查找。然后继续将适当的子表分半，继续进行查找（折半查找算法，若查找元素不存在集合中，则会退出折半查找）。一个长度为 n 的线性表被多次分半，直到最后的子表中只有一个元素，一共需要进行多少次呢？答案是 $\log_2 n$ 次。

拓展提高

注意：本书中用到的对数几乎都是以 2 为底，所以通常缩写为 log n。

（4）算法分析

计算复杂性理论所研究的资源中最常见的是时间（要通过多少步才能解决问题）和空间（在解决问题时需要多少内存）。其他资源亦可考虑，例如在并行计算中，需要多少并行处理器才能解决问题。

一种数据结构的优劣由实现其各种运算的算法具体体现，对数据结构的分析实质上就是对实现运算算法的分析，除了要验证算法是否能正确解决该问题之外，还需要对算法的效率做性能评价。

在计算机程序设计中，算法分析拥有十分重要的地位。对于一个实际问题的解决，可以提出若干个算法，那么如何从这些可行的算法中找出最有效的算法呢？如果有了一个解决实际问题的算法，如何来评价它的好坏？通过算法分析可以对这些问题给出衡量的标准。因此算法分析是每个程序设计人员应该掌握的技术。

①数量级的概念。

一个算法的执行时间等于其所有语句执行时间的总和。而任一语句的执行时间为该语句执行一次所需时间与执行次数的乘积。要精确地计算各语句执行一次所需要的时间是十分困难的，因为它取决于执行算法的具体的计算机及其所配备的编译系统的质量以及运行的环境等众多因素。因此，在我们的分析工作中只是粗略地将算法中语句执行的最大次数（称语句的执行频度）作为算法时间的量度，而不需要计算出它的具体执行时间。

设有以下三个程序段：

A:　{x ++; s+=x ; }　　（最大执行次数为 1 ）

B: for (i =1; i<=n; i++) （最大执行次数为 n ）
{x ++; s+=x ; }

C: （1） for （i = 1 ; i<=n ; i++ ） （最大执行次数为 n^2 ）
（2） for （j =1 ; j<=n ; j++ ）
（3） {x ++; s+=x ; }

对于情况 C ，虽然只是一个语句，但是我们可以把它分为若干个子句。这样，既可使问题得到简化，又不影响分析的结论。其各步执行的次数如下：

行号	次数
（1）	n+1
（2）	n*(n+1)
（3）	n^2

该语句总执行次数 f(n) 为：

$f(n)=2n^2+2n+1$

显然，执行次数 f(n) 为 n 的函数，其中，n 为问题的规模，如线性表的长度、多项式的项数、矩阵的阶、图中的顶点数等。算法分析就是要确定 f(n) 是 n 的什么函数，进而分析 f(n) 随 n 变化的情况和确定 f(n) 的数量级 (order of magnitude)。

②算法的时间复杂度。

一个算法的时间复杂度 (Time Complexity) 是指该算法的运行时间与问题规模的对应关系。一个算法是由控制结构和原操作构成的，其执行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，通常把算法中基本操作重复执行的次数（频度）作为算法的时间复杂度。算法中的基本操作一般是指算法中最深层循环内的语句，因此，算法中基本操作重复执行的频度 T(n) 是问题规模 n 的某个函数 f(n)，随 n 的变化情况并确定 T(n) 的数量级 (Order of Magnitude)。在这里用“O”表示数量级。所谓算法的时间复杂度，就是算法的时间量度，记作：

$T(n)=O(f(n))$。

其中“O”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，或者说，用“O”符号表示数量级的概念。例如，$T(n)=\frac{1}{2}n(n-1)$，则$\frac{1}{2}n(n-1)$的数量级与 n^2 相同，所以 $T(n)=O(n^2)$。

如果一个算法没有循环语句，则算法中基本操作的执行频度与问题规模 n 无关，记作 O(1)，也称为常数阶。如果算法只有一个一重循环，则算法的基本操作的执行频度与问题规模 n 呈线性增大关系，记作 O(n)，也叫线性阶。常用的还有平方阶 $O(n^2)$、立方阶 $O(n^3)$、对数阶 $O(\log_2 n)$ 等。

知识链接

算法在机器上执行的时间，通常的做法是从算法中选取一种对于研究的问题来说是基本运算的原操作，以该基本操作要执行的次数（或称语句的频度）作为算法的时间量度。

下面举例来说明计算算法时间复杂度的方法。

两个 N × N 矩阵的算法中，乘法运算是“矩阵相乘问题”的基本操作。整个算法的执行时间与基本操作（乘法）重复执行的次数与 n^3 成正比，记作：

$T(n) = O(n^3)$

具体的程序代码如下：

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

数量级 O(1)、O(n)、$O(n^2)$ 和 $O(n^3)$ 分别称为常量阶、线性阶、平方阶和立方阶，算法出现的数量级还可能有对数阶 $O(\log_2 n)$ 和指数阶 $O(2^n)$ 等。若一个算法执行的时间为 $O(\log_2 n)$，则当 n 充分大时，他比执行时间为 O(n) 的算法速度要快的多。同样 $O(n*\log_2 n)$ 比 $O(n^2)$ 要佳，但不如 O(n)。这 7 种类型的增长率由小到大的排列顺序为：

$O(1)< O(\log_2 n)<O(n)< O(n*\log_2 n)< O(n^2)< O(n^3)< O(2^n)$

图 1-2 展示了时间复杂度不同的 4 个程序在同样的机器与编译系统上的运行时间。时间单位为 s。假设需要解决一个问题，但只有 1000s 的机时，那么能解决多大规模的问题呢？从表 1-1 中第二列的数据可以看到，4 个程序在 1000s 内所能解决问题的规模相差不大。

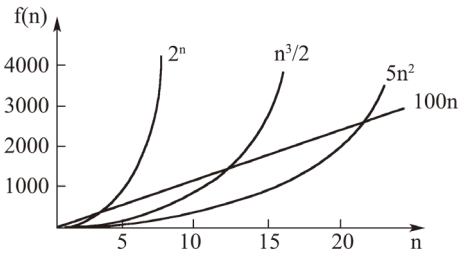


图 1-2 4 个算法的运行时间

假设在另一台比上述机器快 10 倍的机器上运行上述 4 个程序，这相当于在原来的机器上进行 10^4s 运算。这时各程序所能求解的最大规模如表 1-1 中的第三列所示。该表第四列中的数据是前两列中问题规模的增长比率。可以看到，当计算机的速度提高 1000% 时，4 个程序所能处理的问题规模分别提高了 10.0、3.2、2.3 和 1 ～ 3 倍。如果计算机的速度继续提高，或者在原来的机器上继续增加运行时间，则这些程序将保持这样的效率运行。

表 1-1 计算时间增加 10 倍的效果

运行时间函数 t(n)	10^3s 可解决的问题规模	10^4s 可解决的问题规模	最大输入量的问题规模的增长比率（倍）
100n	10	100	10.0
$5n^2$	14	45	3.2
$n^3/2$	12	27	2.3
2^n	10	13	1～3

实际中，有些算法的运行时间不仅与问题的规模 n 有关，而且还与处理数据集的初始状态有关。在这种情况下，我们还可以定义最坏情况下的运行时间，也就是说，对于长度为 n 的所有输入，取运行时间的最大值。有时，还考虑平均的运行时间，即对于所有长度为 n 的输入，取它们的运行时间的平均值。

③算法的空间复杂度。

评价算法的标准很多，评价一个算法主要看这个算法所占用机器资源的多少，而这些资源中时间代价与空间代价是两个主要的方面，通常是以算法执行所需的机器时间和所占用的存储空间来判断一个算法的优劣。

知识链接

时间复杂度是衡量一个算法优劣的重要参数。时间复杂度越小，说明该算法效率越高，则该算法越有价值。

而空间复杂度是指计算机科学领域完成一个算法所需要占用的存储空间，一般是输入参数的函数。它是算法优劣的重要度量指标，一般来说，空间复杂度越小，算法越好。我们假设有一个图灵机来解决某一类语言的某一问题，设有 X 个字（Word）属于这个问题，把 X 放入这个图灵机的输入端，这个图灵机为解决此问题所需要的工作带格子数总和称为空间。

关于算法的存储空间需求，类似于算法的时间复杂度。用空间复杂度作为算法所需存储空间的量度，记作：

$S(n) = O(f(n))$

其中，n 为问题的规模。一般情况下，一个程序在机器上执行时，除了需要寄存本身所用的指令、常数、变量和输入数据以外，还需要一些对数据进行操作的辅助存储空间。其中对于输入数据所占的具体存储量只取决于问题本身，与算法无关，这样只需要分析该算法在实现时所需要的辅助空间单元个数就可以了。若算法执行时所需要的辅助空间相对于输入数据量而言是个常数，则称这个算法为原地工作，辅助空间为 O(1)。

复杂度理论和可计算性理论不同，可计算性理论的重心在于问题能否解决，不管需要多少资源。而复杂性理论作为计算理论的分支，某种程度上被认为和算法理论是一种“矛”与“盾”的关系。算法的执行时间的耗费和所占存储空间的耗费两者是矛盾的，难以兼得，即算法执行时间上的节省一定是以增加存储空间为代价的，反之亦然。即算法理论专注于设计有效的算法，而复杂性理论专注于理解为什么对于某类问题，不存在有效的算法。

就一般情况而言，常常以算法执行时间作为算法优劣的最主要衡量指标。

（5）算法的描述

著名的计算机科学家沃思给出了一个著名的公式：算法 + 数据结构 = 程序。沃思的公式表明，数据结构和算法是程序的两大要素，二者相辅相成，缺一不可。算法可用自然语言、框图或高级程序设计语言进行描述。自然语言简单但易于产生歧义，框

图直观但不擅长表达数据的组织结构，而高级程序语言则较为准确，又比较严谨。

如果将算法变成在计算机上运行的程序，它必须是严格按照语法规定，用相应的语言来编写，例如 C 语言。本书采用 C 语言来描述和实现算法，使读者能够阅读或上机查阅，以便更好地理解算法。

尽管我们主张安排 C 语言课程为本课程的先修课，但并不是所有读者都已学习了 C 语言课程。为了使读者能够理解本书用 C 语言描述的算法，下节将介绍一些 C 语言的基本知识，基本能满足本书的算法描述要求。如果读者需要进一步深入了解 C 语言，请参阅相关书籍。

5. C 语言预备知识

（1）预定义常量

本书中用到了一些常量符号，如 TRUE、FALSE、MAXSIZE 等，约定用宏定义预先定义如下：

```
#define MAXSIZE      1000
#define TRUE         1
#define FALSE        0
#define ERROR        0
#define OK           1
#define SUCCESS      1
#define FAILURE      0
```

（2）结构体类型

结构体 (Struct) 是由一系列具有相同类型或不同类型的数据构成的数据集合，也叫结构。

在 C 语言中，可以定义结构体类型，将多个相关的变量包装成为一个整体使用。在结构体中的变量，可以是相同、部分相同，或完全不同的数据类型。在 C 语言中，结构体不能包含函数。在面向对象的程序设计中，对象具有状态 (属性) 和行为，状态保存在成员变量中，行为通过成员方法 (函数) 来实现。C 语言中的结构体只能描述一个对象的状态，不能描述一个对象的行为。

知识链接

C 语言中引入结构体的主要目的是为了将具有多个属性的事物作为一个逻辑整体来描述，从而允许扩展 C 语言数据类型。作为一种自定义的数据类型，在使用结构体之前，必须完成其定义。

结构体定义的语法形式如下：

```
struct 结构体标识符 {
   成员变量列表 ;
   …
};
```

其中 struct 为关键字 (Keyword)，说明当前定义一个新的结构体类型。结构体标识符遵循 C 语言标识符的命名规则。在 {} 之间通过分号分隔的变量列表称为成员变量 (Member Variables)，用于描述此类事物的某一方面特性。成员变量可以是基本数据类型 (如 float)、数组和指针类型，也可以为结构体。由于不同的成员变量分别描述事物某一方面的特性，因此成员变量不能重名。

为了描述三维世界中的坐标点，可以定义结构体 struct Point。

```
struct Point{
   double x;                    /*x 坐标 */
   double y;                    /*y 坐标 */
   double z;                    /*z 坐标 */
};
```

（3）算法描述

本书中所有的算法都以 C 语言中函数的形式表示，其中的结构类型使用前面已有的定义：

```
/* 函数说明 */
数据类型 函数名 ( [ 形式参数定义及说明 ] )
{
   内部数据说明 ;
   执行语句组 ;
}
```

函数的定义主要由函数名和函数体组成，函数体用大括号 {} 括起来。这里函数中用方括号括起来的部分如“[形式参数定义及说明]”为可选项，函数名之后的圆括号不可省略。函数的操作结果可由形参或函数类型传递，形式参数是函数的参数，分为值参数和变参数。值参数只实现单向传递给函数；变参数是将值传递给函数，函数执行完后再将结果传给主调函数。

知识链接

执行语句可由 C 语言各种类型的语句组成，两个语句之间用分号“;”分隔。可将函数执行的结果通过 return 语句返回给调用它的函数。“/ * 函数头 * /”为注释部分，这是一种习惯写法，可按实际情况取舍。

（4）赋值语句

C 语言中赋值用“=”号。

```
变量名 1 = 变量名 2 或 表达式 ;
```

（5）分支语句

①分支形式 1。

```
if ( 条件表达式 ) 语句 ;
```

②分支形式 2。

```
if（条件表达式） 语句 1;
else 语句 2;
```

③分支形式 3。

```
swith（表达式）
{
   case 值 1: 语句 1; break;
   case 值 2: 语句 2; break;
   ...
   case 值 n: 语句 n; break;

   [default: 语句 n+1;]
}
```

④ switch 语句。

switch 语句是先计算表达式的值，然后用其值与判断值相比较，若它们相一致时，就执行相应 case 下的语句组；若不一致，则执行 default 下的语句组；执行 break 语句则跳出 switch 语句体。

知识链接

使用这种语句时，重要的是要善于使用 switch 来简化多重条件和嵌套条件，使多分支结构清晰。

switch 条件选择的另一种形式如下：

```
swith
{
   case 条件 1: 语句 1; break;
   case 条件 2: 语句 2; break;
   ...
   case 条件 n: 语句 n; break;
   [default: 语句 n+1;]
}
```

方括号括起来的部分如“[default: 语句 n+1;]”为可选项。

（6）循环语句

①循环形式 1。

```
for（赋值表达式； 条件表达式； 修改表达式） 循环体语句；
```

首先计算赋值表达式的值，然后求条件表达式的值，若结果非零（即为真）则执行循环体语句，最后对修改表达式做运算（主要是修改循环变量），如此循环，直到条

件表达式的值为零 (即不成立，为假) 时为止。

②循环形式 2。

```
while ( 条件表达式 ) 循环体语句 ;
```

while 循环首先计算条件表达式的值，若条件表达式的值非零 (即条件成立)，则执行循环体语句，然后再次计算条件表达式的值，重复执行，直到条件表达式的值为零 (即为假) 时退出循环，执行该循环之后的语句。

③循环形式 3。

```
do 循环体语句 ;
while ( 条件表达式 );
```

该循环语句首先执行循环体语句，然后计算条件表达式的值，若条件表达式成立，则再次执行循环体，再计算条件表达式的值，直到条件表达式的值为零，即条件不成立时结束循环。

拓展提高

两种 while 语句的区别在于第一种先判断再执行，第二种先执行一次，然后再判断条件表达式。

(7) 逻辑运算符

与运算符为“&&”，或运算符为“||”，取反运算符为“!”。运算规则如图 1-3 所示。

A && B

A \ B	0	1
0	0	0
1	0	1

A || B

A \ B	0	1
0	0	1
1	1	1

!A

A	!A
0	1
1	0

图 1-3 逻辑运算规则

(8) 结束语句

常用的结束语句包括如下：

① return < 表达式 > 或 return：用于函数结束。

② break 语句：可用于循环语句中循环过程结束或跳出的情况；例如 switch 语句中结束或跳出的情况。

③ continue 语句：可用在循环语句中结束本次循环过程，进入下一次循环过程。

④ exit 语句：表示出现异常情况时，退出程序。

(9) 注释形式

注释形式如下：

```
/* 说明性文字 */
```

注释句的作用是增强算法的可阅读性，在算法描述中要求在函数首部加上对算法功能的必要注释和描述。加注释说明时，对涉及的参量可做必要的说明，对变量的类型要有相应的定义或解释。

（10）数组

数组是经常在数据结构的实现中使用。在程序设计中，为了处理方便，把具有相同类型的若干变量按有序的形式组织起来。这些按序排列的同类数据元素的集合称为数组。在C语言中，数组属于构造数据类型。一个数组可以分解为多个数组元素，这些数组元素可以是基本数据类型或是构造类型。因此按数组元素的类型不同，数组又可分为数值数组、字符数组、指针数组、结构数组等各种类别。

在C语言中使用数组必须先进行定义。

一维数组的定义方式为：

类型说明符 数组名 [常量表达式];

其中：

- 类型说明符是任一种基本数据类型或构造数据类型。
- 数组名是用户定义的数组标识符。
- 方括号中的常量表达式表示数据元素的个数，也称为数组的长度。

例如：

int a[10]; 说明整型数组 a，有 10 个元素。

float b[10],c[20]; 说明实型数组 b 有 10 个元素，实型数组 c 有 20 个元素。

char ch[20]; 说明字符数组 ch，有 20 个元素。

对于数组类型说明应注意以下几点：

①数组的类型实际上是指数组元素的取值类型。对于同一个数组，其所有元素的数据类型都是相同的。

②数组名的书写规则应符合标识符的书写规定。

③数组名不能与其他变量名相同。

例如：

```
main()
       {
         int a;
         float a[10];
         ……
}
```

是错误的。

④方括号中常量表达式表示数组元素的个数，如 a[5] 表示数组 a 有 5 个元素。但是其下标从 0 开始计算。因此 5 个元素分别为 a[0],a[1],a[2],a[3],a[4]。

⑤不能在方括号中用变量来表示元素的个数，但是可以是符号常数或常量表达式。

例如：

```
#define FD 5
    main()
   {
    int a[3+2],b[7+FD];
    ……
}
```

是合法的。

但是下述说明方式是错误的。

```
main()
    {
     int n=5;
     int a[n];
     ……
}
```

⑥允许在同一个类型说明中，说明多个数组和多个变量。

例如：

```
int a,b,c,d,k1[10],k2[20];
```

知识链接

数组元素是组成数组的基本单元。数组元素也是一种变量， 其标识方法为数组名后跟一个下标。下标表示了元素在数组中的顺序号。

数组元素的一般形式为：

数组名 [下标]

其中下标只能为整型常量或整型表达式。如为小数时，C 编译将自动取整。

例如：

a[5];

a[i+j];

a[i++];

都是合法的数组元素。

数组元素通常也称为下标变量。必须先定义数组， 才能使用下标变量。在 C 语言中只能逐个地使用下标变量，而不能一次引用整个数组。

例如，输出有 10 个元素的整型数组必须使用循环语句逐个输出各下标变量：

```
for(i=0; i<10; i++)
    printf("%d",a[i]);
```

C 语言中不能用一个语句输出整个数组，必须用循环实现。

（11）指针

指针是 C 语言中广泛使用的一种数据类型。运用指针编程是 C 语言最主要的风格

之一。利用指针变量可以表示各种数据结构；能很方便地使用数组和字符串；并能像汇编语言一样处理内存地址，从而编出精练而高效的程序。指针极大地丰富了 C 语言的功能。

指针 (Pointer) 与整型一样，是一种数据类型，只不过是一种特殊的数据类型。指针实际上就是地址。我们知道，变量（实际上对常量也一样，只不过谈论更多的是变量）在计算机内占有一块存储区域，变量的值就存放在这块区域之中。在计算机内部，通过访问或修改这块区域的内容来访问或修改相应的变量。对于变量的访问形式之一，就是先求出变量的地址，然后再通过地址对它进行访问，这就是这里所要论述的指针及其指针变量。

所谓变量的指针，实际上是指变量的地址。变量的地址虽然在形式上类似于整数，但在概念上不同于整数，它属于一种新的数据类型，即指针类型。

准备存放指针的变量必须首先声明。指针的声明由一个基类型 (Base Type)、一个星号 (*) 和变量名字组成。声明指针变量的一般形式为：

```
type  *name;
```

其中，type 是指针的基类型，是任何有效的类型，name 是指针变量的名字。

例如：

```
int *p;
```

其中，p 是指针变量，它保存一个整型变量的地址。

指针的基类型定义指针可以指向变量的类型。技术上，任何类型的指针都可以指向任何内存的任何位置。然而，指针的操作是基于基类型的。例如，当声明指针为类型 int * 时，编译程序假定它所保存的任何地址都指向一个整数，无论实际上是否如此（也就是说，int * 指针总是“认为”它指向 int 对象，而无论内存片是否真正包含它）。

拓展提高

声明指针时，必须确保它的类型与要指向的对象类型兼容。

& 运算符也称为地址运算符，在一个变量前加 & 运算符，表示该变量的指针；* 运算符称为指针运算符，在一个指针变量前加 *，表示该指针所指向的内存单元的值。也就是说，指针变量保存的是一个内存的起始地址值，指针变量加 * 后表示该地址对应的存储单元的值。

C 语言规定，指针变量也可以定义为 void 型，比如：

```
void  *p;
```

这里 p 仍然是一个指针变量，有自己的内存空间，占用 2 个字节 (Turbo C 环境)。但是不指定 p 指向哪种类型的变量。下面的例子是利用指针变量存取数组的一个元素。

```
void main(void)
```

```
  Void *p;
{
   int a[3] = {1, 2, 3}, *p;
   p = &a[2];
   printf("*p=%d",  *p);
}
```

程序运行结果为：

```
*p=3
```

在这个例子中，指针变量 p 存放的是数组元素 a[2] 的地址，因此用 * 操作符取其对应的内存内容时，得到整数 3。

拓展提高

用 C 语言表示和实现抽象数据类型描述时，主要包括以下两个方面的内容：

- 通过结构体将 int、float 等固有类型组合到一起，构成一个结构类型，用 typedef 为该类型或该类型指针重新起一个名字。
- 用 C 语言函数实现各操作。

任务实施

（1）企业员工信息管理问题

各企业都有不同数量的员工，少则几十人，多则上万人。在计算机系统中怎样管理好这些员工的信息数据？员工信息中反映了与每个员工的相关数据，如工号、姓名、性别、部门、技术职称、出生年月、家庭住址等信息，如表 1-2 所示。

表 1-2 员工信息表

工 号	姓 名	性 别	部 门	技术职称	出生年月	家庭住址
GH20120001	张三	男	软件部	高级工程师	81/02/02	北京海淀区中关村大街 1 号
GH20120002	李四	女	硬件部	高级工程师	78/08/02	上海徐家汇 12 号
GH20120003	王五	女	行政部	无	82/01/15	北京海淀区西四环北路 14 号楼 101
GH2012004	陈亮	男	销售部	助理工程师	90/08/31	北京西城区三里河三区 8 栋 11 号
GH2012005	赵立国	男	销售部	无	87/08/31	北京市海淀区大钟寺 312 号
…						

员工信息以表的形式存入计算机后，对员工信息的处理工作就转化为对数据表的处理。例如，查询员工信息处理为在表中查询基本信息，对新入职的员工处理为在表中增加一行基本信息，对员工离职处理为删除表中相应的行。因此由计算机实现员工信息管理问题抽象出的描述模型就是：

①建立包含每个员工基本信息的若干个表。

②对该表进行插入、删除、修改、查询等操作。

在这类数学模型中，每一行元素之间的关系是一种线性关系，这类数学模型是一种线性表数据结构。

（2）组织机构管理问题

组织机构包括行政组织机构和企业组织机构。某企业组织机构如图 1-4 所示。

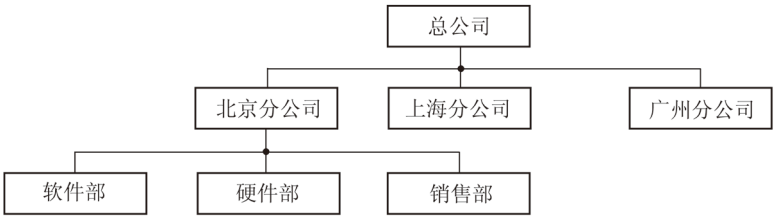


图 1-4　组织机构图

知识链接

从图 1-4 可以看出，组织机构问题中所描述的模型就是建立具有层次结构的组织机构。这类模型是具有层次结构的数据模型，是一种树结构，像每个家庭中的家谱、行政组织机构，高校院系关系等。把它叫作“树”是因为它看起来像一棵倒挂的树，也就是说它是根朝上，而叶朝下的。

（3）学校排课问题

学生在校期间都要按规定学习一系列的课程，有的课程需要另一门课程作为先修课。下面以软件专业为例，列出一个课程表，共 12 门，如表 1-3 所示。排课时怎样排才能合理呢？当然排一门课程必须等到其先修课全部排完后。这是一种特殊的排序问题。

表 1-3　课程关系表

课程编号	课程名称	先 修 课
C_1	程序设计基础	
C_2	C 语言	C_1
C_3	数据结构	C_1, C_2
C_4	汇编语言	C_1
C_5	计算机组成原理	C_3, C_4
C_6	数据库	C_{11}
C_7	编译原理	C_3, C_5
C_8	操作系统	C_3, C_6
C_9	高等数学	
C_{10}	线性代数	C_9
C_{11}	离散数学	C_9
C_{12}	数值分析	C_1, C_9, C_{10}

图 1-5 中所描述的数据模型是表 1-3 所列出的课程依赖关系的图示，是一种被称为“图”的数据结构。一个图看起来是由一些小圆点（称为顶点）和连接这些圆点的直线或曲线（称为边）组成的。

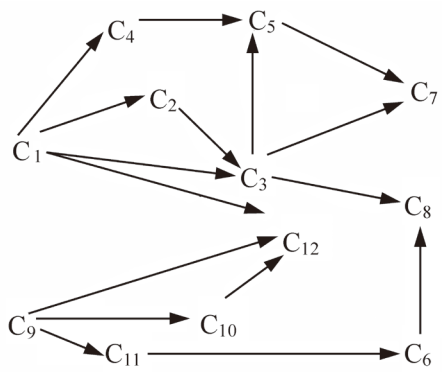


图 1-5 课程关系图

上述三个例子分别描述了线性、树形和图的数据结构，人们将数据归纳为某一种进行相应的运算，完成数据处理过程。

项目小结

本项目主要介绍数据结构的基本概念和术语、数据的存储结构、算法的描述与算法分析、C 语言预备知识和程序结构化。通过对本项目的学习，使读者对数据库（C 语言）有初步了解。

项目考核

填空题

（1）_____是计算机中存储、组织数据的方式，是指相互之间存在一种或多种特定关系的数据元素的集合。

（2）计算机中的数据是广义的，包括了数，字符，_____，表，文件等，_____，_____，图像都属于数据的范畴。

（3）算法具备_____、_____、_____、_____、_____5 个特性。

选择题

（1）执行语句可由 C 语言各种类型的语句组成，两个语句之间用分号（　　）分隔。

A. .　　B. :　　C. 、　　D. ;

（2）函数的定义主要由函数名和函数体组成，函数体用（　）括号括起来。

A. {}　　B. []　　C. ()　　D. （）

问答题

（1）试举一个数据结构的例子、叙述其逻辑结构、存储结构、运算三个方面的内容。

（2）常用的存储表示方法有哪几种形式？

（3）简述算法的定义与特性。

（4）算法的时间复杂度仅与问题的规模相关吗？

（5）按增长率由小至大的顺序排列下列各函数：　2100，(3/2)n，(2/3)n，　nn，n0.5，n!，2n，lgn，nlgn，n(3/2) 。