21世纪高等学校规划教材

数据结构

曲朝阳　郭晓利　王晓惠　孙鸿飞　编著

中国电力出版社
www.infopower.com.cn

21世纪高等学校规

数据结构

曲朝阳　郭晓利　王晓惠　孙鸿飞　编著

中国电力出版社
www.infopower.com.cn

内容提要

本书主要包括数据结构的基本概念、基本的数据结构（线性表、栈和队列、串、数组与广义表、树、图）以及基本技术（查找方法与排序方法）等三个部分。全书共8章，第1章绪论中引入了数据结构与算法的一些基本概念，是全书的综述；第2～6章分别介绍了线性表、栈、队列、串、数组、广义表、树和图等几种基本的数据结构；第7章和第8章分别介绍了查找和排序的方法，它们都是数据处理时需要广泛使用的技术。

本书是在作者多年教学实践的基础上编写而成的，内容丰富，概念清晰，技术实用，同时还配有大量的例题、习题和上机习题。本书可作为高等院校计算机及相关专业本科生的教材，也可作为专科和成人教育的教材，还可供从事计算机应用的科技人员参考之用。

图书在版编目（CIP）数据

数据结构 / 曲朝阳等编著. —北京：中国电力出版社，2007

21世纪高等学校规划教材

ISBN 978-7-5083-6112-3

I. 数… II. 曲… III. 数据结构－高等学校－教材 IV. TP311.12

中国版本图书馆CIP数据核字（2007）第118348号

丛 书 名：21世纪高等学校规划教材

书　　名：数据结构

出版发行：中国电力出版社

地　　址：北京市三里河路6号　　邮政编码：100044

电　　话：（010）68362602　　传　　真：（010）68316497，88383619

服务电话：（010）58383411　　传　　真：（010）58383267

E-mail：infopower@cepp.com.cn

印　　刷：北京丰源印刷厂

开本尺寸：185mm×260mm　　印　　张：13.25　　字　　数：330千字

书　　号：978-7-5083-6112-3

版　　次：2007年8月北京第1版

印　　次：2007年8月第1次印刷

印　　数：0001—4000册

定　　价：21.00元

前　言

数据结构是计算机科学与技术专业教学计划中的一门核心课程，同时也是信息计算、电子信息技术等非计算机专业的一门重要专业基础课程。计算机科学与技术及其相关学科都会用到各种数据结构，数据结构已经成为计算机科学与技术工作者，尤其是计算机应用领域开发人员必备的知识。

数据结构的任务是根据从各种实际问题中归纳、抽象出来的对象的数据特征和对象之间的关系，选择合适的数据组织方法、存储方法和相应的算法。这些方法有助于设计出周密、有效和风格良好的程序。

数据结构课程的教学要求是让学生学会分析和研究计算机加工的数据对象的特征，以便在实际应用中选择适当的数据逻辑结构、存储结构和相应算法，掌握算法的时间和空间性能分析技巧，学习应对复杂程序设计方面的技巧。

本书循序渐进地介绍了线性表、栈和队列、串、数组和广义表、树、图、查找、排序等知识。考虑到本书是面向计算机专业和非计算机专业学生的教科书，因而遵循了简明、实用、通俗易懂的原则，尽量避免繁琐、深奥的理论推导和说明。读者只要具备一定的 C 语言知识就能轻松地学习。

本书深入浅出地讲解了数据结构方面的理论知识，同时又重视实践。每一章的开头都列出了本章的学习目标；每章最后配有大量不同类型的习题和上机实验题目，以方便读者在计算机上进行实践，有助于理解算法的实质和基本思想。

限于编者水平，疏漏之处在所难免，希望广大读者批评指正。

编　者

2007 年 7 月

目　　录

前　言
第 1 章　绪论 1
1.1　数据结构的基本概念 1
1.2　算法和算法分析 3
1.3　算法描述语言与 C 语言数据类型 6
本章小结 14
习题 15
第 2 章　线性表 17
2.1　线性表的基本概念 17
2.2　线性表的顺序存储结构及其运算 18
2.3　线性表的链式存储结构及其运算 22
2.4　顺序表和链表的比较 29
2.5　线性表的应用 30
本章小结 32
习题 32
本章实验 34
第 3 章　栈和队列 41
3.1　栈 41
3.2　队列 46
3.3　应用 51
本章小结 58
习题 59
本章实验 60
第 4 章　串、数组和广义表 63
4.1　串 63
4.2　数组 68
4.3　广义表 79
本章小结 81
习题 81
本章实验 82
第 5 章　树和二叉树 87
5.1　树 87
5.2　二叉树 90

5.3 二叉树的遍历 98
5.4 线索二叉树 103
5.5 树、森林与二叉树的转换 108
5.6 哈夫曼树 110
本章小结 112
习题 112
本章实验 114
第 6 章 图 124
6.1 图的基本概念 124
6.2 图的存储结构 127
6.3 图的遍历 131
6.4 图的连通性 134
6.5 最短路径 136
6.6 AOV 网与拓扑排序 141
6.7 AOE 网与关键路径 142
本章小结 144
习题 144
本章实验 147
第 7 章 查找 156
7.1 基本概念 156
7.2 静态查找 157
7.3 动态查找表 161
7.4 哈希法查找 167
本章小结 170
习题 171
本章实验 172
第 8 章 排序 179
8.1 排序基本概念 179
8.2 插入类排序 180
8.3 交换类排序 184
8.4 选择类排序 189
8.5 归并排序 193
8.6 基数排序 194
8.7 各类排序方法的比较 196
8.8 外部排序 196
本章小结 197
习题 197
本章实验 198

第1章 绪 论

本章学习目标

- 了解数据结构的基本概念，理解常用术语
- 掌握数据元素间的3种结构关系
- 掌握算法的定义及特性，掌握算法设计的要求
- 初步掌握分析算法的时间复杂度和空间复杂度的方法
- 掌握C语言中的数据类型

信息技术的迅速发展，为计算机的发展提供了更为广阔的应用空间。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。与此相应地，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，数据结构就是研究数据组织、存储和运算的一般方法的学科。本章主要介绍数据结构的基本概念、算法及C语言的数据类型。

1.1 数据结构的基本概念

1.1.1 数据结构实例

【例1.1】最短路径问题。如图1.1所示，假设有A、B、C、D、E、F共六座城市，图中的带箭头的连线表示城市间有开通的单向航班，连线上的数值表示该航班飞行所需要的时间，请问，如果要从城市A出发去城市F（中间可以在其他城市换机，并忽略换机时间），耗费时间最少的路径是什么？

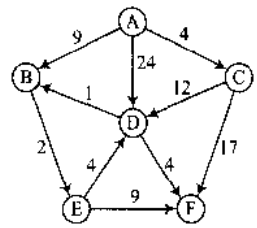


图1.1 最短路径问题示意图

这是一个非常经典的关于称为“图”的数据结构的应用问题，它显然不是数值计算问题，本书在以后将对这类问题给出解决方案。

可见，数据结构是一门研究非数值计算的程序设计问题中计算机的操作对象以及它们之间的关系和操作等的学科。学习数据结构的目的是了解计算机处理对象的特性，以便将实际问题中所涉及的处理对象及这些对象之间的关系进行抽象并在计算机中表示出来，最后再对它们进行处理。

数据结构是计算机学科非常重要的一门专业基础理论课程，要想编写针对非数值计算问题的高质量的程序，就必须熟练掌握这门课程涉及的知识。另外，它与计算机其他课程都有密切联系，具有独特的承上启下的重要作用，对于学习计算机专业的其他课程，如操作系统、数据库管理系统、软件工程等都是有益的。

1.1.2 数据结构概念

1. 数据

数据是信息的载体，是对客观事物的符号表示。通俗地说，凡是能被计算机识别、存取和加工处理的符号、字符、图形、图像、声音、视频信号、程序等一切信息都可以称为数据。数据可以是数值数据，也可以是非数值数据。数值数据包括整数、实数、浮点数、复数等，主要用于科学计算和商务处理等；非数值数据包括文字、符号、图形、图像、动画、语音、视频信号等。随着多媒体技术的飞速发展，计算机中处理的非数值数据越来越多。

2. 数据元素

数据元素是对现实世界中某独立个体的数据描述，是数据的基本单位。在计算机中，数据元素通常作为一个整体来处理。一个数据元素可以由若干个数据项组成，在 C 程序设计中一个数据元素可以由一个 struct 表示。数据项是具有独立意义的最小数据单位，是对数据元素属性的描述。在例 1.1 中对每个城市的描述可用一个数据元素来描述；而关于每个城市的基本信息，如城市的名称、机场的位置、航线的多少等，可用数据项来描述。

3. 数据对象

数据对象是具有相同性质的数据元素的集合，是数据的一个子集。例如，字母字符数据对象是集合 C={'A','B','C',...,'Z'}。

4. 数据结构

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。根据数据元素之间的关系特性，数据结构可由一个二元组（D，S）定义，其中 D 是数据元素的有限集，S 是 D 中元素的关系的有限集。通常，基本的数据结构有如下 3 类：

（1）线性结构。结构中的数据元素之间存在一对一的关系。

（2）树形结构。结构中数据元素之间存在一对多的关系。

（3）图结构或网结构。结构中的数据元素之间存在多对多的关系。

图 1.2 描述了这 3 种数据结构。3 种数据结构可以划分为两大类：线性结构和非线性结构。线性结构的逻辑特征是，有且仅有一个开始结点和一个终端结点，并且所有的结点都最多只有一个直接前驱和一个直接后继。非线性结构的逻辑特征是一个结点可能有多个直接前驱和直接后继。非线性结构包括树型结构和图结构。

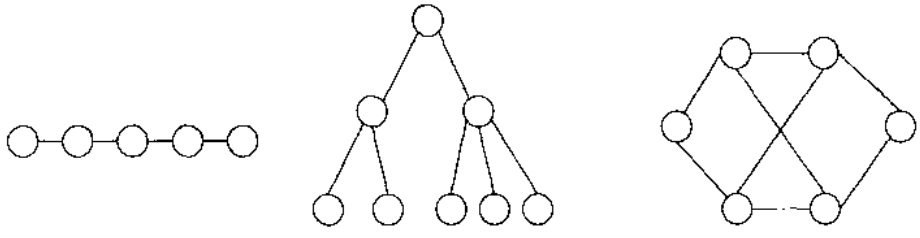


图 1.2 3 类基本数据结构示意图

5. 逻辑结构

逻辑结构描述的是数据元素之间的逻辑关系。通常所说的线性结构、树形结构和图结构就是指数据元素的逻辑结构。

6. 存储结构

存储结构（又称物理结构）是数据结构在计算机中的表示。具有某种逻辑结构的数据元素在计算机中有顺序映像和非顺序映像两种不同的表示方法，由此得到两种不同的存储结构——顺序存储结构和链式存储结构。

顺序存储结构（sequential storge structure）是把必要的数据元素存储在存储单元中，通过对存储单元的地址进行一个固定的计算就能直接表达数据元素之间逻辑上的关系的物理结构。在顺序存储结构中，逻辑上相邻的数据元素在存储器中物理位置也相邻，即数据元素之间的逻辑关系可通过数据元素在存储器中的相对位置来体现。顺序存储结构是一种最基本的存储表示方法，本书主要借助C语言中的一维数组来实现。

链式存储结构（linked storge structure）是把数据元素存储在附设了指针域的存储单元中，通过指针域的值来表达数据元素之间逻辑上的关系的物理结构。本书对链式存储结构主要借助于C语言中的指针类型和结构体类型来实现。

除了通常采用的顺序存储结构和链式存储结构外，有时为了特定操作的方便还可以采用索引存储和散列存储等，这些都是顺序存储结构和链式存储结构的变形或综合。

1.2 算法和算法分析

1.2.1 算法

算法（Algorithm）是解决特定问题的方法，是对数据施加的一系列操作。严格地说，算法是由若干指令组成的有限序列。程序是算法的实现。根据实际需要，一个实际问题的解决可以有多种算法。算法包括数值算法和非数值算法两种。解决数值问题的算法称为数值算法，例如求解线性方程组、求解代数方程、求解微分方程等。解决非数值问题的算法称为非数值算法，数据处理方面的算法都属于非数值算法，例如各种排序算法、查找算法、插入算法、删除算法、遍历算法等。

一种算法必须具有以下5个特性：

（1）有穷性。一个算法包括的指令数必须有限，每一条指令的执行次数也必须有限。

（2）确定性。算法的每一条指令必须有确切的定义，无二义性。

（3）可行性。算法中的每一条指令都可以通过有限次、可实现的基本运算且在有限的时间内实现。

（4）输入。一个算法具有零个或多个输入。

（5）输出。一个算法具有一个或多个输出。

算法设计的要求如下：

（1）正确性。算法的执行结果应当满足预先设定的功能和要求。在实际应用中，算法的“正确性”有多个层次的含义。

（2）可读性。一个算法应当思路清晰、层次分明、易读易懂。

（3）健壮性。当输入非法数据时，应能作适当反应和处理，不至于引起莫名其妙的后果。

（4）高效性和低存储量。对同一个问题，执行时间越短，算法的效率就越高；完成相同的功能，执行算法时所占用的存储空间应尽可能少。

实际上，一种算法很难做到十全十美，原因是上述要求有时是相互抵触的。例如，时间和空间就是一对矛盾，如果要节约算法的执行时间，往往以牺牲一定存储空间为代价；而为了节省存储空间，就可能耗费更多的计算时间。所以实际操作中应以算法的正确性为前提，根据具体情况而有所侧重。若一个程序使用的次数较少，一般要求简明易懂即可；对于需要反复多次使用的程序，应尽可能选用快速的算法；若待解决的问题数据量极大，而计算机的存储空间又相对较小，则应主要考虑如何节约存储空间。

1.2.2 算法分析

算法执行时间需要根据该算法编制的程序在计算机上的执行时间来确定。一个算法所耗费的时间等于算法中每条语句的执行时间之和。每一条语句的执行时间是该语句的执行次数（frequency count，频度）与该语句执行一次所需时间的乘积，而每条语句的执行时间取决于其对应机器指令的执行时间。一个使用高级程序语言编写的程序在计算机上运行所需时间取决于如下因素。

（1）使用何种程序设计语言。实现编程语言的级别越高，其执行效率就越低。

（2）选用何种策略的算法。

（3）算法涉及问题的规模（求解问题的输入量，通常用 n 表示）。例如，求 50 以内的素数与求 1000 以内的素数的执行时间必然是不同的。

（4）编译程序所生成目标代码的质量。对于代码优化较好的编译程序，所生成的程序质量较高。

（5）机器执行指令的速度。

（6）计算机的体系结构。并行计算通常能缩短算法的计算时间。

显然，在各种因素不确定的情况下，使用执行算法的绝对时间来衡量算法的效率是不合适的。在上述各种与计算机相关的软、硬件因素确定以后，一个特定算法的运行时间就只依赖于问题的规模（通常用正整数 n 表示）。

算法的效率通常用时间复杂度和空间复杂度来评价，它们反映了算法的执行所需要的时间和空间与问题规模之间的一种数量上的依赖关系。

1. 算法的时间复杂度

通常，使用算法中所有语句的频度之和 $f(n)$ 表示该算法所需的时间。在假定执行一条语句需要的时间固定的情况下，语句的数量能大致表示算法运行所需的时间，因此 $f(n)$ 粗粒度地描述了一个算法所需时间。假设当问题规模 n 趋向于无穷大时，有下式成立：

$$T(n)= O(f(n)) \tag{1-1}$$

它表示随着问题规模的扩大，$T(n)$ 的增长率和 $f(n)$ 的增长率相同。称 $T(n)$ 为算法的渐近时间复杂度（asymptotictime complexity），简称为时间复杂度（time complexity）。

【例 1.2】 交换 A 和 B 的内容。

（1）temp=A；

（2）A=B；

（3）B=temp；

这 3 条语句的频度都是 1，$f(n)$=3。所以，该程序的执行时间与问题规模无关，$f(n)$=3。算法的时间复杂度为常数阶，记为 $T(n)= O(1)$。

【例 1.3】累加。

```
(1) x=0;          /*执行 1 次*/
(2) y=0;          /*执行 1 次*/
    for(k=1; k<=n;  k++)
(3) x++;          /*执行 n 次*/
    for(i=1; i<=n; i++)
      for(j=1; j<=n; j++)
(4)   y++;        /*执行 n² 次*/
```

所有语句的频度之和 $f(n)$为 n^2+n+2。

当 $n\to\infty$时，显然有：

$$\lim f(n)/n^2=\lim(n^2+n+2)/n^2=1$$

所以，　$T(n)=O(n^2)$。

【例 1.4】已知某算法的核心代码段如下，计算该算法的时间复杂度。

```
for(i=1; i<=n; i*=2)
    x++;
```

该算法的时间复杂度是 $O(\log_2 n)$。

【例 1.5】已知某算法的核心代码段如下，计算该算法的时间复杂度。

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

显然，算法基本操作 x++重复执行次数随着问题规模 n 增长的函数是 n^2，所以该算法的时间的复杂度是 $O(n^2)$。

有时，如果难以精确计算基本操作的执行次数，算法的时间复杂度只需用基本操作关于 n 的增长的阶来表示即可。

分析类似于这种形式的算法的时间复杂度时，一种方案是计算该算法基本操作的平均值，也就是要计算出“条件”成立的概率，这样计算得到的时间复杂度称为平均时间复杂度。但是，算法的基本操作的平均值通常很难计算，这时，经常按输入数据集的内容最不理想的情形来计算，这样得到的时间复杂度称为最坏时间复杂度。本书中如未加指明，均指最坏时间复杂度。

2. 算法的空间复杂度

一个算法在机器上运行时需要占用存储空间，这些存储空间主要包括两部分：第一部分用于存放算法本身的指令、常数、变量和输入数据；第二部分是对数据进行操作时所需的辅助空间。在通常情况下，对同一类问题，输入数据的表示形式是相同的，这时可以近似认定第一部分的存储空间是固有的，而第二部分存储空间是额外的，分析算法的空间占有情况重点是分析其额外存储空间的占有情况。

通常，一个算法的空间复杂度（space complexity）反映了程序运行从开始到结束所需的额外存储量。常使用“大 O 表示法”表示，$S(n)=O(f(n))$。

其中，$f(n)$是算法的额外存储空间随问题规模 n 增长的函数，$S(n)$是空间复杂度。如果 $S(n)=O(1)$，表示算法为本地工作，即随问题规模 n 的增长，额外存储空间不变。

1.3 算法描述语言与C语言数据类型

1.3.1 算法描述语言

算法总是施加在特定类型的数据结构之上的。在算法的设计过程当中，对数据的各种运算都是定义在数据的逻辑关系之上的，是处于比较抽象层次上的操作；也就是说，主要考虑的是“做什么”的问题。而究竟这些操作如何实现则依赖于数据所采用的存储结构。当然，在算法讨论中，我们对于存储结构的描述是在高级程序设计语言的基础上进行的，而不是像在机器语言当中用内存地址来直接描述数据的存储结构。由于在每种高级程序设计语言中所定义的数据类型都对应于一定的物理结构，实际上就是对数据的存储结构的一种抽象，因此，本书所讨论的存储结构指的是在C语言的数据类型基础上的一种较为抽象的虚拟存储结构。

本书对算法的描述采用的是结合自然语言的一种类 C 语言的描述方法。用自然语言描述便于人们相互交流和理解，也可以更加突出重点，而不拘泥于局部的细节实现，且条理性更好，也更便于表达。同时，结合 C 程序设计语言的描述使算法的表达更加准确，也更便于最终的程序实现和对算法性能的分析。

下面是对本书中所使用的算法描述语言的简介。

（1）数据类型。

本书中所使用的算法描述语言的数据类型包括 C 语言中的所有数据类型（整型、实型、字符型、数组、指针、结构、共用体等）。

（2）变量和符号常量。

变量的定义形式为：

```
数据类型  变量名序列;
```

符号常量的定义形式为：

```
#define  符号常量名  常量值;
```

其中，符号常量名常采用具有一定含义的标识符来命名，例如：

```
#define  TRUE  1
#define  FALSE  0
#define  OK  1
#define  ERROR  0
#define  OVERFLOW  -2
```

（3）数据运算。

数据运算主要包括算术运算、关系运算和逻辑运算三种。

1）算术运算符有+、-、*、/、%、++、-- 等。

2）关系运算符有>、<、==、>=、<=、!=。

3）逻辑运算符有!、&&、||。

另外还有指针运算符*、&，分量运算符（.）、->和下标运算符（[]）。

（4）赋值语句。

赋值语句的形式为：

变量名=表达式;

在表达式中，除了可以出现上述数据运算之外，还可以出现常用的数学符号，如Σ、∪、∩、∈等。

(5) 控制语句。

控制语句包括选择语句和循环语句。

1) 选择语句，包括条件语句、多项选择语句。

条件语句的形式为：

```
if(条件表达式)
语句块;
```

或

```
if(条件表达式)
语句块 1;
else
语句块 2;
```

多项选择语句的形式为：

```
switch(表达式)
{
case 常量表达式 1: 语句块 1;break;
case 常量表达式 2: 语句块 2;break;
⋮
default: 语句块 n;
}
```

2) 循环语句，包括 for 语句、while 语句和 do-while 语句。

for 语句的形式为：

```
for(循环变量初始表达式; 终止条件表达式; 循环变量修改表达式)
语句块;
```

while 语句的形式为：

```
while(条件表达式)
语句块;
```

do-while 语句的形式为：

```
do{
语句块;
}while(条件表达式);
```

(6) 函数的定义、声明与调用。

函数的定义形式为：

```
返回类型  函数名(形式参数列表)
{
  函数定义语句块;
};
```

函数的声明形式为：

```
返回类型  函数名(形式参数列表);
```

函数的调用形式为：

```
函数名(实际参数列表);
```

（7）输入、输出。

输入语句为：

```
scanf(格式字符串，输入变量序列);
```

输出语句为：

```
printf(格式字符串，输出表达式序列);
```

（8）结束语句。

异常结束语句 exit，其形式为：

```
exit(异常代码);
```

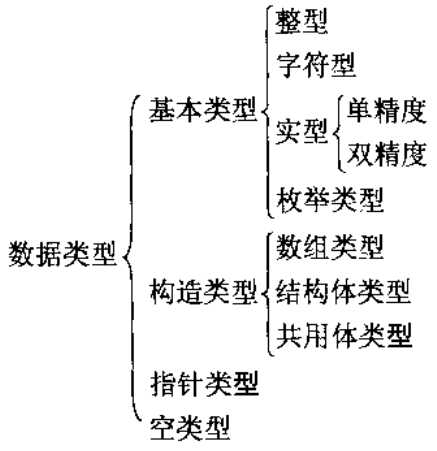
（9）注释。

注释语句的形式为：

```
/*注释内容*/
```

1.3.2 C 语言的数据类型

C 语言的数据结构是以数据类型形式出现的。C 语言的数据类型如下：



C 语言中的数据有常量与变量之分，它们分别属于以上这些类型。由以上这些数据类型还可以构成更复杂的数据类型。例如，利用指针和结构体类型可以构成表、树、栈等复杂的数据结构。下面分别对常用的数据类型作简单介绍。

1.3.2.1 基本数据类型

1. 整型数据

整型常量即整型常数，如 123、-456 等。

整型变量的基本类型符为 int。可以根据数值的范围将变量定义为基本整型、短整型或长整型。在 int 之前可以根据需要分别加上修饰符 short 或 long。因此，有以下三类整型变量：

（1）基本整型，以 int 表示。

（2）短整型，以 short int 表示。

（3）长整型，以 long int 表示。

一个 int 型变量的值的范围为-2^{15}～（$2^{15}-1$），即-32768～32767。在实际应用中，变量的值通常是正的（如学号、库存量、年龄、存款额等）。为了充分利用变量的表示数的范围，此时可以将变量定义为“无符号”类型。对以上三类都可以加上修饰符 unsigned，以指定是“无符号数”；如果加上修饰符 signed，则指定是“有符号数”；如果既不指定为 signed，也不指定为 unsigned，则隐含为有符号（signed）。实际上，signed 是完全可以不写的。归纳起来，可以用以下 6 种基本整型，即：

（1）有符号基本整型，以[signed] int 表示。

（2）无符号基本整型，以 unsigned int 表示。

（3）有符号短整型，以[signed]short [int]表示。

（4）无符号短整型，以 unsigned short [int]表示。

（5）有符号长整型，以[signed]long [int]表示。

（6）无符号长整型，以 unsigned long[int]表示。

2. 实型数据

实型常量即实数，如 123.345、6.45e6 等。

实型变量分为单精度（float 型）、双精度（double 型）和长双精度（long double）三类。如：

```
float x,y;          （指定 x、y 为单精度实数）
double z;           （指定 z 为双精度实数）
long double t;      （指定 t 为长双精度实数）
```

3. 字符数据

C 语言中的字符常量是用单引号括起来的一个字符，如 'a'、'A' 等都是字符常量。注意，'a' 和 'A' 是不同的字符常量。除了以上形式的字符常量外，C 语言还允许用一种特殊形式的字符常量，就是以一个“\”开头的字符序列。例如，在 printf 函数中的 '\n' 它代表一个“换行”符。

字符变量用来存放字符常量，用 char 来定义。

1.3.2.2 构造类型

1. 数组类型

数组是相同类型的数据元素的有序集合。数组中的每一个元素都属于同一个数据类型，具有相同的名字——数组名。数组的下标指示了数据元素在数组中的位置，因此数组名和下标唯一地确定了一个数组元素。

数组的定义方式如下所示：

```
数据类型  数组名[常量表达式];
```

其中，数据类型指的是数组元素的类型；数组名由一个标识符所指定；常量表达式表示数组元素的个数，即数组长度，它必须是一个不小于 0 的整型常量或是一个可以在编译时确定大小的表达式（表达式的值必须是一个不小于 0 的整数）。也就是说，常量表达式中可以包括常量和符号常量，不能包含变量，即不能定义一个长度可变的数组。数组的下标从 0 开始，如：

```
int a[5];
```

它表示数组名为 a，该数组有 5 个数组元素，每个数组元素的类型为 int，它们分别为 a[0]、a[1]、a[2]、a[3]和 a[4]。

在内存中，一个数组占据了一片连续的存储空间，每个数组元素占据了这个连续的存储空间中的一个单元，而数组名代表了这片空间的起始地址，即该数组的第一个元素在内存中的地址。图 1.3 所示是 a[5]在内存中的存储示意图。在 C 语言中，只能通过下标运算符[ ]逐个地引用数组元素，而不能一次引用整个数组。下标可以是整型常量或整型表达式，如 a[2]或 a[1+1]。

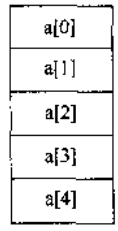


图 1.3 数组的存储示意

C 语言使用特定的语法来进行数组的初始化，如：

```
int a[5]={1, 2, 3, 4, 5};
```

在初始化时，也可以只对部分元素进行初始化，如：

```
int  a[5]={1, 2};
```

这就初始化为 a[0]=1、a[1]=2。

如果初始化时对数组的全部元素都指定初值，就可以省略对数组长度的定义，系统会自动地根据初值的个数来判断数组的长度，例如：

```
int a[ ]={1, 2, 3, 4, 5};
```

系统根据 5 个初值就判断数组 a 的长度为 5。

如果初始化时只对部分数据元素进行初始化，则不能省略对数组长度的定义。

数组的类型可以是 C 语言中允许的任何数据类型（基本类型、构造类型和指针类型等），可以使用定义在该数据类型上的所有合法操作来操作数组元素。数组的基本操作是对数据元素的存和取，例如：

```
int i,a[5]={1, 2, 3, 4, 5};
a[0]=6;
i=8*a[1];
```

当数组作为函数参数传递时，采用了地址传递的方法，即将数组第一个元素的地址值传递给函数的形参。

2. 结构体类型

在 C 语言中，数组是用来存储多个相同数据类型的元素的有序集合，但是，有时数据类型的数据组合成一个有机的整体，以便于引用。在这种情况下，程序虽然用多个变量分别存储这些数据，但是由于这些数据之间具有相互联系，它们的值的组合构成了对一个事物的完整描述。如果能用一个数据类型的变量来存放多个不同类型的信息，那就能使程序更好地表示这些由多个部分组成的数据。C 语言中的结构（struct）就是这样的一种数据类型，它由多个成员组成，每个成员可以分属不同的数据类型。结构的使用减少了程序所管理的变量的数目和传递给函数的参数的数目。

定义结构变量时首先定义所需的结构类型：

```
struct 结构名称
{
   成员定义列表
};
```

在说明了结构数据类型之后，编译系统了解了如何为该结构的变量分配存储空间，但并不为该结构类型分配存储空间。

然后，定义具有该结构类型的结构变量：

```
struct  结构名称  变量名列表;
```

在定义了该结构类型的变量后，编译系统才为该变量分配存储空间。在存储器中，结构变量的各个成员按它们在结构类型定义时的说明次序连续顺序存放。但为了提高结构变量的存取效率，在结构的存储中一般将每个成员都从机器字的边界处开始分配，这样，结构类型所占用的存储空间大小可能会大于各个成员所占空间之和。

也可以将变量的定义直接跟在结构类型定义之后：

```
struct 结构名称
{
   成员定义列表;
}变量名列表;
```

例如：

```
struct person
{
  char name[20];
  int  age;
  char  sex;
  int height;
  float weight;
}person1,*person2;
```

结构变量的初始化形式与数组变量的初始化形式相似，都采用了初始化列表的方法，将各个成员的初始化数据用花括号“{}”括在一起，各个数据值之间用逗号“,”分开。例如：

```
struct person person1={"Zhang san",20,'M',175,68.5};
```

结构成员的类型可以为C语言所允许的任何类型。对结构中成员的访问，可以通过“.”运算符来进行。它的使用方法与同类型的普通变量相同，例如：

```
strcpy(person1.name, "Zhang san");
person1.age=20;
person1.sex='M';
person1.height=175;
person1.weight=68.5;
```

如果是通过指向结构变量的指针来访问结构变量的成员，就要使用“->”运算符来访问，例如：

```
strcpy(person2->name, "Xiao wang");
person2->age=18;
person2->sex='F';
person2->height=165;
person2->weight=60.2;
```