第 1 章　绪论

本章学习目标

本章主要介绍数据结构中的一些常用术语以及集合、线性结构、树形结构和图形结构等常用数据结构的表示，用 C 语言实现算法描述的一般规则，算法的时间复杂度和空间复杂度分析与评价。通过本章的学习，读者应掌握如下内容:

- 数据结构中的常用基本术语
- 集合、线性结构、树形结构和图形结构等每一种常用数据结构的逻辑特点
- 抽象数据类型的定义、使用，算法的定义、特性及用 C 语言描述算法的规则
- 评价算法优劣的规则，算法的时间复杂度、空间复杂度的定义及数量级的表示
- 复习 C 语言中的有关语法规则，以便满足“数据结构”课程中进行算法描述的需要

1.1　什么是数据结构

1.1.1　数据结构示例

为了使大家对数据结构有感性认识，先举出几个例子来说明数据结构。

【例 1-1】给出一张学生数据表，如表 1-1 所示。

在学生数据表中，一行为一个学生信息，代表一个学生数据，一列为一个属性，整个二维表格形成学生数据的一个线性序列，每个学生排列的位置有先后次序，它们之间形成一种线性关系，这就是一种典型的数据结构（线性结构），我们将它称为线性表。

表 1-1　学生数据表

学号	姓名	性别	籍贯	电话	通讯地址
01	张三	男	长沙	88639000	麓山路 327 号
02	李四	男	北京	23456789	学院路 435 号
03	王五	女	广州	30472589	天河路 478 号
04	赵六	男	上海	41237568	南京路 1563 号
05	钱七	女	南京	50134712	南京大学
06	刘八	女	武汉	61543726	武汉大学
07	朱九	男	昆明	4089651	云南大学
08	孙十	女	杭州	6154372	西湖路 635 号

【例 1-2】描述一个磁盘的目录及文件结构，包含一个根目录、若干个一级子目录（文件

夹），每个一级子目录中又包含若干个二级子目录（子文件夹），如图 1-1 所示。

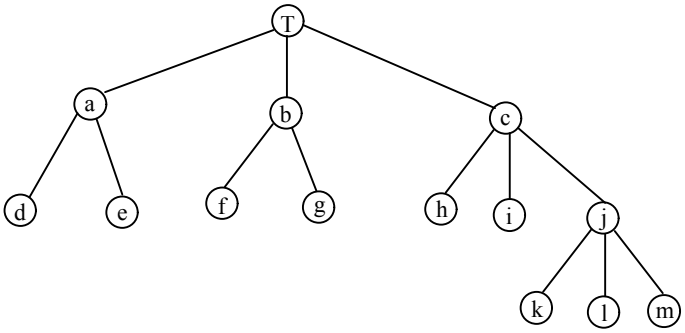

图 1-1　树形结构示意图

在此种结构中，数据之间呈现一对多的非线性关系，这也是我们常用的一种数据结构（非线性结构），我们将它称为树形结构。

【例 1-3】描述一个大学的校园网（圆圈代表站点，边表示网线），如图 1-2 所示。

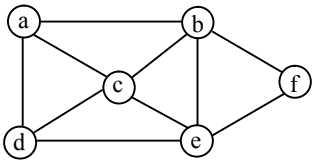

图 1-2　图形结构示意图

在此种结构中，数据之间呈现多对多的非线性关系，这也是我们常用的一种数据结构（非线性结构），我们将它称为图形结构。

综合上述三个例题，我们将对数据结构及一些基本术语作进一步说明。

1.1.2　基本术语

1．*数据*（data）

数据是指能够输入到计算机中，并被计算机识别和处理的符号的集合。

例如：数字、字母、汉字、图形、图像、声音都称为数据。

2．*数据元素*（data element）

数据元素是组成数据的基本单位。

数据元素是一个数据整体中相对独立的单位，但它还可以分割成若干个具有不同属性的项（字段），故其不是组成数据的最小单位。

3．*数据对象*（data object）

数据对象是性质相同的数据元素组成的集合，是数据的一个子集。

例如，整数数据对象的集合可表示为 N={0,±1,±2,…}，大写字母字符数据对象的集合可表示为 C={'A', 'B',…,'Z'}。

4．*数据类型*（data type）

数据类型是一组性质相同的值的集合以及定义于这个集合上的一组操作的总称。

例如，高级语言中用到的整数数据类型，是指由-32768～32767 中的整数数值构成的集合及一组操作（加、减、乘、除、乘方等）的总称。

5. 抽象数据类型（abstract data types）

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，抽象数据类型由基本数据类型组成，并包括一组相关的操作。抽象数据类型有些类似于 C 语言中的 struct 类型和 Pascal 语言中的 record 类型，但它增加了相关的操作。

【例 1-4】给出自然数（natural number）的抽象数据类型定义。

```
ADT  natural  number  is
Data:
一个整数的有序子集合，它开始于 0，终止于机器能表示的最大整数（maxint）。
Operation:
对于所有 x,y∈natural number，定义如下操作：
add(x,y)                求 X＋Y
sub(x,y)                求 X－Y
mul(x,y)                求 X×Y
div(x,y)                求 X÷Y
equal(x,y)              判断 X、Y 是否相等
End
```

在本书中，描述一种抽象数据类型将采用如下书写格式：

```
ADT <抽象数据类型名> is
Data: <数据描述>
Operation: <操作声明>
End
```

1.1.3　数据结构

1. 数据结构（data structure）

数据结构是指相互之间存在一种或多种特定关系的数据元素所组成的集合。具体来说，数据结构包含三个方面的内容，即数据的逻辑结构、数据的存储结构和对数据所施加的运算。这三个方面的关系为：

（1）数据的逻辑结构独立于计算机，是数据本身所固有的。

（2）存储结构是逻辑结构在计算机存储器中的映像，必须依赖于计算机。

（3）运算是指所施加的一组操作总称。运算的定义直接依赖于逻辑结构，但运算的实现必依赖于存储结构。

比如，例 1-1 提到的学生数据表，除了有 8 个学生的数据外，还存在着一对一的线性关系，例 1-2 提到的磁盘目录及文件结构，除包含文件数据外，还存在着目录之间一对多的非线性关系，例 1-3 提到的大学校园网，除包含站点数据外，还存在着站点间的多对多的非线性关系。

2. 从逻辑结构划分数据结构

数据结构从逻辑结构划分为：

（1）线性结构

元素之间为一对一的线性关系，第一个元素无直接前驱，最后一个元素无直接后继，其余元素都有一个直接前驱和直接后继。

（2）非线性结构

元素之间为一对多或多对多的非线性关系，每个元素都有多个直接前驱或多个直接后继。

（3）集合结构

元素之间无任何关系，元素的排列无任何顺序。

3. 从存储结构划分数据结构

数据结构从存储结构划分为：

（1）顺序存储（向量存储）

所有元素都存放在一片连续的存储单元中，逻辑上相邻的元素存放到计算机内存中仍然相邻。

（2）链式存储

所有元素存放在可以不连续的存储单元中，但元素之间的关系可以通过地址确定，逻辑上相邻的元素存放到计算机内存后不一定是相邻的。

（3）索引存储

使用该方法存放元素的同时，还要建立附加的索引表，索引表中的每一项称为索引项，索引项的一般形式是：（关键字，地址），其中的关键字能唯一标识一个结点的数据项。

（4）散列存储

通过构造散列函数，用函数的值来确定元素存放的地址。

4. 数据结构的抽象描述

数据结构可用二元组 D=(K,R)的形式来描述。其中，$K=\{a_1,a_2,\cdots,a_n\}$为元素集合，$R=\{r_1,r_2,\cdots,r_m\}$为关系的集合。

【例 1-5】设有一个线性表（a_1,a_2,a_3,a_4,a_5），它的抽象描述可表示为 D=(K,R)，其中 $K=\{a_1,a_2,a_3,a_4,a_5\}$，$R=\{<a_1,a_2>,<a_2,a_3>,<a_3,a_4>,<a_4,a_5>\}$，则它的逻辑结构的图形描述见图 1-3。

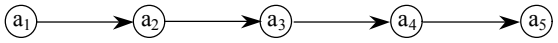

图 1-3　线性表结构抽象描述示意图

【例 1-6】设一个数据结构的抽象描述为 D=(K,R)，其中 K={a,b,c,d,e,f,g,h}，R={<a,b>, <a,c>, <a,d>,<b,e>,<c,f>,<c,g>,<d,h>}，则它的逻辑结构的图形描述见图 1-4。

【例 1-7】设一个数据结构的抽象描述为 D=(K,R)，其中 K={1,2,3,4}，而 R={(1,2),(1,3),(1,4),(2,3),(2,4),(3,4)}，则它的逻辑结构的图形描述见图 1-5。

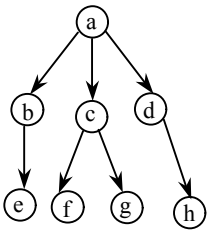

图 1-4　树形结构抽象描述示意图

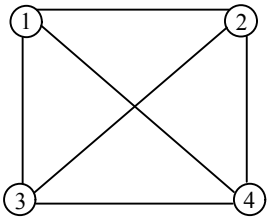

图 1-5　图形结构抽象描述示意图

1.2　算法描述

1.2.1　基本概念

1. 算法（algorithm）

通俗地讲，算法就是一种解题的方法。更严格地说，算法是由若干条指令组成的有穷序列，它必须满足下述条件（也称为算法的五大特性）：

（1）输入：具有 0 个或多个输入的外界量（算法开始前的初始量）。

（2）输出：至少产生一个输出，它们是算法执行完后的结果。

（3）有穷性：每条指令的执行次数必须是有限的。

（4）确定性：每条指令的含义都必须明确，无二义性。

（5）可行性：每条指令的执行时间都是有限的。

2. 算法和程序的关系

算法的含义与程序十分相似，但二者是有区别的。一个程序不一定满足有穷性（死循环），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。一个算法若用计算机语言来书写，则它就可以是一个程序。

1.2.2　算法描述

1. 用流程图描述算法

一个算法可以用流程图的方式来描述，输入、输出、判断、处理分别用不同的框图表示，用箭头表示流程的流向。这是一种描述算法的较好方法，目前在一些高级语言程序设计中仍然采用。

2. 用自然语言描述算法

用我们日常生活中的自然语言（可以是中文形式，也可以是英文形式）也可以描述算法。

例如，某同志某天做的工作可以描述为一个算法形式：若今天我有空并且天不下雨，则我上街购物，否则待在家里看书。

3. 用其他方式描述算法

我们还可以用数学语言或约定的符号语言来描述算法。

4. 用 C 语言描述算法

在本书中，我们将采用 C 语言或类 C 语言来描述算法。用 C 语言描述算法遵循如下规则：

（1）所有算法的描述都用 C 语言中的函数形式。

```
函数类型 函数名(形参及类型说明)
{函数语句部分
  return(表达式值);
}
```

（2）函数中的形式参数有两种传值方式。

若形式参数为一般变量名，则为单向传值参数，若在变量前面增加“&”符号，则为双向传地址参数。

例如有一个函数为 void swap(&i, &j,k)，则 i、j 为双向传地址参数，k 为单向传值参数。

（3）输入函数。

C 语言中的输入函数调用为：scanf("格式控制",地址列表)。

（4）输出函数

C 语言中的输出函数调用为：printf("格式控制",变量列表)。

（5）C 语言的作用域。

在 C 语言中，每个变量都有一个作用域。在函数内声明的变量，仅能在该函数内部有效，在类中声明的变量，可以在该类内部有效。在整个程序中都能使用的变量为全局变量，否则称为局部变量。若一个全局变量与一个局部变量同名，则该范围内全局变量不起作用。

1.3 算法分析

求解同一个问题，可以有许多不同的算法，那么怎样来衡量这些算法的优劣呢？首要的条件是选用的算法必须是正确的，其次，考虑如下三点：

（1）执行算法所耗费的时间。

（2）执行算法所占用的内存开销（主要考虑占用的辅助存储空间）。

（3）算法应易于理解，易于编码，易于调试等。

1.3.1 时间复杂度

1. 时间频度

一个算法执行时所耗费的时间，从理论上说是不能算出来的，必须上机运行测试才能知道。但我们不可能也没有必要对每个算法都上机测试（因为，计算机的运行速度与 CPU 等因素有关。同一算法在不同的计算机上运行的时间是不同的），只需知道在相同的条件下，哪个算法花费的时间多，哪个算法花费的时间少就可以了。并且一个算法花费的时间与算法中语句的执行次数成正比，哪个算法中语句执行次数多，它花费的时间就多。一个算法中的语句执行次数称为语句频度或时间频度，记为 T(n)。

【例 1-8】求下列算法段的语句频度。

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

分析：该算法为一个二重循环，执行次数为内、外循环次数相乘，但内循环次数不固定，与外循环有关，因此，时间频度 $T(n)=1+2+3+\cdots+n=\frac{n(n+1)}{2}$。

2. 时间复杂度

在刚才提到的时间频度中，n 称为问题的规模，当 n 不断变化时，时间频度 T(n)也会不断变化。但有时我们想知道它变化时呈现什么规律，为此，引入时间复杂度概念。

设 T(n)的一个辅助函数为 g(n)，定义当 n 大于或等于某一足够大的正整数 n_0 时，存在两个正的常数 A 和 B（其中 A≤B），使得 $A\leqslant\frac{T(n)}{g(n)}\leqslant B$ 均成立，或有 $\lim\limits_{n\to+\infty}\frac{T(n)}{g(n)}=A$（其中 A 为

常数），则称 g(n)是 T(n)的同数量级函数。把 T(n)表示成数量级的形式为：T(n)=O(g(n))，其中大写字母 O 为英文 Order（即数量级）一词的第一个字母。

例如，若 $T(n)=\frac{n(n+1)}{2}$，则有 $\lim\limits_{n\to+\infty}\frac{T(n)}{n^2}=\frac{1}{2}$，故它的时间复杂度为 $O(n^2)$，即 T(n)与 n^2 数量级相同。

【例 1-9】分析下列算法段的时间频度及时间复杂度。

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

分析算法规律可知时间频度 T(n)=1+(1+2)+(1+2+3)+…+(1+2+3+…+n)

$$
\begin{aligned}
&=\sum_{k=1}^{n}(1+2+3+\cdots+k)\\
&=\sum_{k=1}^{n}\frac{k(k+1)}{2}\\
&=\sum_{k=1}^{n}\frac{k^2}{2}+\sum_{k=1}^{n}\frac{k}{2}\\
&=\frac{1}{2}\left[\frac{n(n+1)(2n+1)}{6}+\frac{n(n+1)}{2}\right]
\end{aligned}
$$

由于有 $\lim\limits_{n\to+\infty}\frac{T(n)}{n^3}=\frac{1}{6}$，故时间复杂度为 $O(n^3)$。

在各种不同算法中，若算法中语句执行次数为一个常数，则时间复杂度为 O(1)，另外，在时间频度不相同时，时间复杂度有可能相同，如 $T(n)=n^2+3n+4$ 与 $T(n)=4n^2+2n+1$，它们的时间频度不同，但时间复杂度相同，都为 $O(n^2)$。

按数量级递增排列，常见的时间复杂度有：常数阶 O(1)，对数阶 $O(\log_2 n)$，线性阶 O(n)，线性对数阶 $O(n\log_2 n)$，平方阶 $O(n^2)$，立方阶 $O(n^3)$，k 次方阶 $O(n^k)$，指数阶 $O(2^n)$，$O(3^n)$，…，$O(k^n)$，阶乘阶 O(n!)等。

随着问题规模 n 的不断增大，上述时间复杂度不断增大，算法的执行效率降低。

1.3.2　空间复杂度

1. 空间频度

一个算法在执行时所占有的内存开销，称为空间频度，但我们一般是讨论算法占用的辅助存储空间。讨论方法与时间频度类似，此处不再赘述。

2. 空间复杂度

与时间复杂度类似，空间复杂度是指算法在计算机内执行时所占用的内存开销规模。我们一般所讨论的是除正常占用内存开销外的辅助存储单元规模。讨论方法与时间复杂度类似，此处不再赘述。

本章小结

1．数据结构研究的是数据的表示形式及数据之间的相互关系，从逻辑结构来看，有线性结构、树形结构、图形结构和集合四种。从存储结构看，有顺序存储、链式存储、索引存储和散列存储四种。

2．线性结构的数据之间存在一对一的线性关系，树形结构的数据之间存在一对多的非线性关系，图形结构的数据之间存在多对多的非线性关系，集合中不存在数据之间的关系。

3．顺序存储相当于C语言中一维数组存储；链式存储相当于C语言中的链表存储；索引存储也称为索引查找，是关键字、记录的一个有序对，包含主表和索引表；散列存储也称为散列查找，是用构造函数的方法得到关键字的地址。

4．算法的评价指标主要为正确性、健壮性、可读性和有效性四个方面，有效性又包括时间复杂度和空间复杂度两方面。

5．算法的时间复杂度和空间复杂度，通常采用数量级的形式来表示，而数量级的形式有常量阶、对数阶、线性阶、线性对数阶、平方阶、立方阶、指数阶、阶乘阶等。若一个算法的时间复杂度和空间复杂度越好，则算法的执行效率越高。

习题一

1-1 对下列用二元组表示的数据结构，画出它们的逻辑结构图，并指出它们属于何种结构。

（1）A=(K,R)，其中

K={a_1,a_2,a_3,…,a_n}

R={<a_i,a_{i+1}>,i=1,2,…,n-1}

（2）B=(K,R)，其中

K={a,b,c,d,e,f}

R={<a,b>,<b,c>,<c,a>,<a,d>,<d,e>,<e,f>,<c,f>}

（3）C=(K,R)，其中

K={1,2,3,4,5,6}

R={(1,2),(2,3),(2,4),(3,4),(3,5),(3,6),(4,5),(4,6)}

（4）D=(K,R)，其中

K={48,25,64,57,82,36,75,43}

R={r1,r2,r3}

r1={<48,25>,<25,64>,<64,57>,<57,82>,<82,36>,<36,75>,<75,43>}

r2={<48,25>,<48,64>,<64,57>,<64,82>,<25,36>,<82,75>,<36,43>}

r3={<25,36>,<36,43>,<43,48>,<48,57>,<57,64>,<64,75>,<75,82>}

1-2 简述概念：数据、数据元素、数据结构、逻辑结构、存储结构、线性结构、非线性结构。

1-3 试举日常生活中的一个例子来说明数据结构三个方面的内容。

1-4 什么是算法？它的五个特性是什么？

1-5 设n为整数，分析下列程序中用#标明的语句的语句频度及时间复杂度。

（1）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];
          }
```

（2）
```
int i=1, j=1;
while (i<=n&&j<=n)   {# i=i+1; j=j+i; };
```

（3）
```
int i=1;
do {for (j=1; j<=n; j++)   # i=i+j
} while (i<100+n);
```

（4）
```
for (i=1; i<=n; i++)
    for (j=i; j>=1; j--)
        for (k=1; k<=j; k++)   # x=x+1;
```

1-6　设计出一个算法，将 n 个元素按升序排列，并分析出它的时间频度及时间复杂度。

1-7　计算 $\sum_{i=0}^{n}\frac{x^i}{i+1}$ 之值，用 C 函数写出算法，并求出它的时间复杂度。

1-8　用 C 函数描述如何将一维数组中的元素逆置，并分析出时间频度及时间复杂度。

1-9　将三个元素 X、Y、Z 按从小到大排列，用 C 函数描述算法，要求所用的比较和移动元素次数最少。

1-10　用 C 语言中冒泡排序和选择排序两种方法，分别描述出 n 个元素 $a_1,a_2,a_3,\cdots,a_n$ 的升序排列算法，并分析两种方法的平均比较和移动元素次数。

1-11　设计一个二次多项式 ax^2+bx+c 的抽象数据类型，假定起名为 AX2BXC，该类型的数据部分分为三个系数 a、b 和 c，操作部分为：

（1）初始化成员 a、b 和 c（假定用结构体 D 来定义 a、b、c）。

```
AX2BXC initAX2BXC(x,y,z);
```

（2）做两个多项式的加法运算，即使它们的系数相加，并返回相加的结果。

```
AX2BXC add(AX2BXC f1, AX2BXC f2);
```

（3）根据给定的 x 值，求多项式的值。

```
float value(AX2BXC f,float x);
```

（4）计算方程 $ax^2+bx+c=0$ 的两个实根，要求分有实根、无实根和不是二次方程这三种情况讨论，并返回不同的值，以便调用时做不同的处理。

```
int root(AX2BXC f, float &r1,float &r2);
```

（5）按照 ax**2+bx+c 的格式（x^2 用 x**2 表示）输出二次多项式，在输出时必须注意去掉系数为 0 的项，并且当 b 和 c 的值为负时，其前面不能出现加号。

```
void print(AX2BXC f);
```

请写出上面每一个操作的具体实现。

1-12　指出下列各算法的功能并求出其时间复杂度。

（1）
```
int prime(int n)
{
  int i=2;
  int x=(int)sqrt(n);
  while(i<=x)
```

```
    { if(n%i==0) break;
      i++;
    }
    if(i>x) return 1;
    else return 0;
  }
（2）int sum1(int n)
  { int p=1,s=0;
    for(int i=1;i<=n;i++)
    { p*=i;
      s+=p;}
    return s;
  }
（3）int sum2(int n)
  { int s=0;
    for(int i=1;i<=n;i++)
    { int p=1;
      for(int j=1;j<=i;j++)
      p*=j;
      s+=p;}
    return s;
  }
（4）void print(int n)
  { int i;
     for(i=1;i<=n;i++)
     {
       if(i%2) printf("*");
       else continue;
       printf("#");
     }
  printf("$\n");}
（5）void print1(int n)
  { int i,k;
    long j;
    for(i=1;i<=n;i++)
    { j=i*i;
      if(j<10)
      {if(j==i)printf("%d %d\n",i,j);}
      else if(j<100)
      {if(j%10==i) printf("%d %d\n",i,j);}
      else if(j<1000)
      {if(j%100==i) printf("%d %d\n",i,j);}
      else if(j<10000)
      {if(j%1000==i) printf("%d %d\n",i,j);}
      else if(j<100000)
      {if(j%10000==i) printf("%d %d\n",i,j);}
```

```
        else
        {if(j%100000==i) printf("%d %d\n",i,j);}
      }
    }
（6）void matrix(int a[m][n],int b[n][l],int c[m][l])
    { int i,j,k;
      for(i=0;i<m;i++)
      for(j=0;j<l;j++)
      { c[i][j]=0;
        for(k=0;k<n;k++)
        c[i][j]+=a[i][k]*b[k][j];
      }
    }
（7）void xyx(int n)          //n<1000
    { int i,j,k,l;
      for (l=100;l<=n;l++)
      {
          i=l/100;
          j=l/10%10;
          k=l%10;
          if(k*100+j*10+i==l)
          printf("%d\n",l);
      }
    }
（8）int sum3(int n)
    { int i,s=0;
      for(i=1;i<=n;i++)
      if(i%2==1)
        s+=i;
      else s+=i*i;
      return s;
    }
```