书中，对数有两个典型的用途：

（1）用二进制位来存储编码

许多问题都需要对一些对象进行编码，那么表示 n 个不同的对象至少需要多少二进制位呢？答案为$\lceil \log_2 n \rceil$位。例如，要存储$\{A, B, C, D\}$4 个对象，编码至少需要 2 位，分别是$\{00, 01, 10, 11\}$，要存储 1000 个不同的对象，编码至少需要 10 位。

（2）把一个问题分解为更小的子问题，并且分解总是一分为二的。

考虑在一个有序的序列中采用折半查找，每次将待查值 k 与查找序列的中间元素进行比较，以确定下一步是在序列的前半部分还是后半部分进行查找，每比较一次都将查找序列分半，直到找到给定值，见图 0-1。一个长度为 n 的序列被逐次分半，直到序列中只有一个元素，一共需要分多少次呢？答案为$\lfloor \log_2 n \rfloor + 1$次。

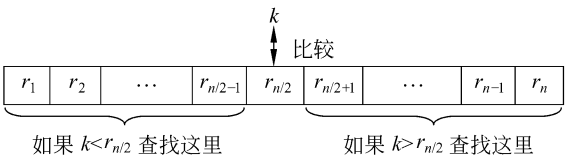

图 0-1　折半查找示意图

0.1.3　级数求和

简单地讲，级数求和就是把函数在一定范围内的值累加起来，通常表示为：

$$\sum_{i=1}^{n} f(i)$$

它表示对作用于某个（整数）范围内的参数 i 的函数 $f(i)$之值求和，函数 $f(i)$ 的参数和它的初值写在符号 $\sum$ 的下面，符号 $\sum$ 的上面表示参数的最大值。因此，这种表示法表示当 i 从 1 到 n 时对函数 $f(i)$的值求和，也可以表示为：

$$\sum_{i=1}^{n} f(i) = f(1) + f(2) + \cdots + f(n)$$

能直接计算级数和的等式称为**闭合形式解**。下面是本书中常用的闭合形式解：

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$\sum_{i=1}^{n} (n-i) = \frac{n(n-1)}{2}$$

$$\sum_{i=1}^{n} i^2 = \frac{2n^3 + 3n^2 + n}{6}$$

$$\sum_{i=1}^{\log n} n = n\log n$$

$$\sum_{i=1}^{n} 2^i = 2(2^n - 1)$$

$$\sum_{i=1}^{n}\frac{1}{2^i}=1-\frac{1}{2^n}$$

$$\sum_{i=1}^{\log n}2^i=2^{\log n+1}-2=2(n-1)$$

大多数程序都带有循环结构，在分析带有循环结构的程序段的时间性能时，需要把循环体的执行次数累加起来，这就是级数求和的应用。

例 0-1　分析下面程序段的执行次数。

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

解：这个程序段的执行次数 $=\sum_{i=1}^{n}\sum_{j=1}^{i}1=\sum_{i=1}^{n}i=\frac{n(n+1)}{2}$。

0.2　常用数学证明方法

下面介绍本书常用的两种证明方法：反证法和数学归纳法。

0.2.1　反证法

支持一个结论(或命题)的实例再多也不足以证明它的正确性，然而，推翻一个结论的最简单的方法就是找一个反例。反证法是一种类似于使用反例进行证明的方法。为了使用反证法证明一个结论，首先假设这个结论是错误的，然后找出由这个假设导致的逻辑上的矛盾。如果寻找矛盾的逻辑是正确的，则惟一解决矛盾的方法就是纠正我们所做结论是错误的假设，即结论是正确的。

例 0-2　证明没有最大的整数。

证明：采用反证法。

第一步，反面假设：假设存在一个最大的整数，记为 A。

第二步，由假设推出矛盾：考虑 $B=A+1$，因为 B 是两个整数的和，所以 B 也是整数，而 $B>A$，因此导出矛盾。在推理过程中，惟一的漏洞就是开始时的假设是错误的，从而结论得到证明。

0.2.2　数学归纳法

设 $T(n)$是一个要证明的结论，其定义域为正整数。数学归纳法通过下列两个步骤来证明对于所有的 $n\geqslant c$(c 是一个较小的常量)，结论 $T(n)$成立：

1. 初始情况：证明 $n=c$ 时 T 成立；

2. 归纳情况：如果在 $n-1$ 时 T 成立，则在 n 时，T 也成立。

证明初始情况通常很容易，只需要用一些较小的值代替结论中的 n，然后应用一些简单的代数学或简单的逻辑来证明即可；证明归纳情况有时会很难，有时会使用如下的强归纳情况。

2′. 强归纳情况：如果对于所有的 $k(k<n)$，T 都成立，则在 n 时，T 也成立。

例 0-3 证明 $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$。

证明：采用数学归纳法。

第一步，初始情况：当 $n=1$ 时，$\sum_{i=1}^{n} i = \frac{1\times(1+1)}{2} = 1$，结论成立。

第二步，归纳情况：假设结论在 $n-1$ 时成立，即 $\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$，而对于 n，$\sum_{i=1}^{n} i = n + \sum_{i=1}^{n-1} i = n + \frac{n(n-1)}{2} = \frac{n(n+1)}{2}$，由此，结论成立。

0.3 离散数学预备知识

0.3.1 集合

1. 集合的概念

定义 0-1 集合是由互不相同的元素构成的一个整体[①]。如果 x 是集合 S 的一个元素，记作 $x\in S$，读作 x 属于 S；如果 y 不是集合 S 的一个元素，记作 $y\notin S$，读作 y 不属于 S。

严格地说，这不能算是集合的定义，这只是一种描述。正如几何学中的点、线等概念一样，集合、元素、属于是集合论中未加定义而直接引入的最基本最原始的概念[②]。

2. 集合的表示方法

集合有多种表示方法，常用的表示方法有以下三种：

(1) 列举法

将组成集合的所有元素全部列举出来并置于花括弧内，元素之间用逗号分隔，空集记为{}或∅。在上下文意义明确的情况下，可以用省略号表示多个元素。例如：

$A=\{1, 2, 3, 4, \cdots, 99\}$

$B=\{a, e, i, o, u\}$

(2) 描述法

通过刻画元素的性质来界定集合的成员。通常，由满足性质 $P(x)$ 的所有元素组成的集合可记为$\{x \mid P(x)\}$。例如：

$C=\{x \mid x$ 是实数$\}$

$D=\{x \mid x$ 是素数并且 $x<20\}$

(3) 图示法

---

① 数据结构中研究的集合，其元素通常具有相同特性，即集合中元素的类型相同。

② 在数学理论的研究中，一般把概念分为原始概念和派生概念两种类型。派生概念是指可以由其他概念给出定义的概念，如平面几何中的正方形可以由邻边相等的矩形来定义；而原始概念是指无法由其他概念给出定义的概念，如平面几何中的点、线。

集合还可以用文氏图来表示，文氏图用圆表示一个集合，用结点表示集合的元素，如图 0-2 所示。

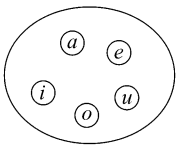

图 0-2　集合 B 的文氏图表示

3. 集合的三大公理

集合论依赖于三大基本公理，它们从根本上规定了集合概念的意义。

(1) 外延公理　两个集合 A 和 B 相等的充分必要条件是它们具有相同的元素。

外延公理刻画了集合的下列特性：

- 互异性：集合中没有重复的元素，即每个元素只出现一次。
- 无序性：集合中的元素可以没有固定顺序，即集合的表示形式不惟一。
- 确定性[①]：任一元素要么属于某集合，要么不属于某集合。

(2) 概括公理　构成一个集合应符合两个条件：

- 纯粹性：凡该集合中的元素都具有某种性质。
- 完备性：凡具有某种性质的元素都在该集合中。

概括公理规定了集合描述法的理论依据和集合元素的确定性。

(3) 正则公理　不存在集合 $A_1, A_2, A_3, \cdots$，使得$\cdots A_3 \in A_2 \in A_1$。

正则公理表明集合和它的元素之间具有层次关系。对任何集合 S，$\{S\} \neq S$，从而规定了集合$\{S\}$与 S 的不同层次性。因为一个集合是由它的成员构成的，是先有成员后才形成集合，所以一个正在形成的集合便不能作为一个实体充当本集合的成员，否则将在概念上产生循环，导致悖论。

4. 集合的基

定义 0-2　集合 A 中含有元素的个数称为集合 A 的基，记为$|A|$。若集合 A 的基是有限数，则称 A 为有限集[②]，否则称 A 为无限集。

例如：集合 $A=\{3, 4, 5\}$，则$|A|=3$。

5. 子集

定义 0-3　对集合 A 和 B，如果 A 中的每个元素都属于 B，则称 A 是 B 的子集，记作 $A \subseteq B$；如果 A 中至少有一个元素不属于 B，则称 A 不是 B 的子集，记作 $A \not\subset B$；如果 A 是 B 的子集，但 A 和 B 不相等，在 B 中有一些元素不属于 A，则称 A 是 B 的真子集，记作$A \subset B$。

例如：集合 $P=\{1, 2, 3, 4, 5\}$，$Q=\{1, 3, 5\}$，$R=\{2, 4\}$，则集合 Q 和 R 都是 P 的子集且都是真子集，但集合 Q 不是 R 的子集，集合 R 也不是 Q 的子集。

6. 集合的幂集

定义 0-4　一个集合 A 的幂集是指由 A 所有可能的子集组成的集合。

例如：若集合 $A=\{a, b, c\}$，则 A 的幂集为：

$\{\varnothing, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}, \{a, b, c\}\}$

---

① 经典集合论中不存在界限不明的情况，不确定的元素构成的集合属于模糊集合的研究范畴。

② 数据结构中讨论的集合通常都是有限集。

0.3.2 谓词

一个简单命题是一个能判断真假的陈述句，在谓词演算中，进一步将简单命题分解为个体与谓词两部分。

定义 0-5 可以独立存在的事物称为**个体**。表示具体的、特指的个体，称为**个体常元**，表示抽象的、泛指的或在一定范围内变化的个体称为**个体变元**。用来刻画一个个体的性质或多个个体之间关系的词称为**谓词**，谓词中包含个体的数目称为谓词的**元数**。

例 0-4 将下列命题符号化：(1)熊猫是动物；(2)上海位于南京和杭州之间。

解：(1) $A(x)$表示 x 是动物。x 是个体变元，它可以在动物范围内任意取值。熊猫是个体常元，则命题可符号化为 A(熊猫)。

$A(x)$是一元谓词，一元谓词通常用来刻画个体的性质。

(2) $B(x,y,z)$表示 x 位于 y 与 z 之间。x、y、z 是个体变元，上海、南京、杭州是个体常元，则命题可符号化为 B(上海，南京，杭州)。

$B(x,y,z)$是多元谓词(二元及二元以上的谓词)，多元谓词通常用来刻画个体之间的关系。

0.3.3 关系

定义 0-6 由两个具有固定次序的元素组成的序列，称为**序偶**，记作(a,b)[①]。

例如：在笛卡儿坐标系中，二维平面上一个点的坐标(x,y)就是一个序偶。

定义 0-7 集合 $A_1,A_2,\cdots,A_n$ 的笛卡儿积 $A_1\times A_2\times\cdots\times A_n$ 的任意一个子集称为 $A_1,A_2,\cdots,A_n$ 上的一个 ***n* 元关系**。特别地，$A_1\times A_2$ 的任意一个子集称为 A_1 到 A_2 的一个**二元关系**。

设 R 是集合 A 到 B 的一个二元关系，对于 $a\in A,b\in B$，若$(a,b)\in R$，则称 a 与 b 有关系 R，记作 aRb；若$(a,b)\notin R$，则称 a 与 b 没有关系 R，记作 $a\not R b$。

定义 0-8 设 R 是集合 A 上的一个二元关系，对于 $a\in A,b\in A$，若$(a,b)\in R$，则称 a 是 b 的前驱，b 是 a 的后继；若不存在 $a\in A$，使$(a,b)\in R$，则 b 无前驱，称为始端；若不存在 $b\in A$，使$(a,b)\in R$，则 a 无后继，称为终端。

例如：$A=\{1,2,3,4,5\}$，$R\subseteq A\times A$，$R=\{(1,2),(2,3),(3,4),(4,5)\}$，则元素 1 为始端，元素 5 为终端，元素 2 的前驱是 1，元素 2 的后继是 3。

定义 0-9 设 $R\subseteq A\times A$，

(1) 对于任意的 $a\in A$，都有 aRa，则称 R 是自反的。

(2) 对于任意的 $a\in A$，都有 $a\not R a$ 则称 R 是反自反的。

(3) 对于任意的 $a\in A,b\in A$，若 aRb,bRa，则称 R 是**对称**的。

(4) 对于任意的 $a\in A,b\in A$，若 aRb,bRa，必有 $a=b$，则称 R 是反对称的。

(5) 对于任意的 $a\in A,b\in A,c\in A$，若 aRb,bRc，必有 aRc，则称 R 是可传递的。

例如：对于自然数，“≤”是自反的、反对称的和可传递的，“<”是反自反的和可传递

① 序偶也称为有序对，有些参考书使用符号$<a,b>$表示由 a 和 b 组成的序偶。

的，“=”是自反的、反对称的和可传递的。

定义 0-10　设 R⊆$A\times A$，如果它是自反、对称且可传递的，则称 R 是 A 上的等价关系，此时，若 aRb，则称 a 与 b 等价。

定义 0-11　设 R 是集合 A 上的等价关系，对任意 $a\in A$，则 A 中与 a 等价的全体元素所组成的集合，称为由 a 生成的**等价类**，记作$[a]_R$。

例如：A={1,2,3,4,5,6,7}，R 是 A 上的模 3 同余关系。显然 R 是 A 上的等价关系，A 中各元素关于 R 的等价类分别是：

$[1]_R$={1,4,7}；$[2]_R$={2,5}；$[3]_R$={3,6}。

定义 0-12　设 R⊆$A\times A$，如果 R 是自反、反对称且可传递的，则称 R 是 A 上的偏序关系，通常记作“≤”。

例如：设 A 为正整数集合，整除关系是 A 上的一个偏序关系。

定义 0-13　设在集合 A 中存在偏序关系 R，对任意 $a\in A$，$b\in A$，都有 aRb 或 bRa，则称 R 是 A 上的全序关系或线性序关系。

全序关系中的任意两个元素均存在某种比较关系。例如，实数集合上的“＜”、“≤”、“＞”、“≥”等关系是全序，而正整数集合上的整除关系不是全序关系。

0.4　C++ 程序设计语言预备知识

下面介绍在本书中用到的 C++ 语言的基本知识。

0.4.1　程序结构

一个 C++ 程序可由若干个文件组成。C++ 的文件分为头文件和源文件两种。头文件以.h 为后缀，用于存放函数声明（也称函数原型）。有些头文件是系统定义的，如＜iostream.h＞，有些头文件是用户定义的。可通过预处理指令 #include 将头文件包含在适当的源文件中。源文件是以.cpp 为后缀，用于存放 C++ 的源代码。

下面是一个典型的 C++ 程序，源文件名为 MyFirstPro.cpp。

```
#include <iostream.h>          //含有 cout 函数的原型
/* 比较两个数的大小并返回较大者 */
int GetMax(int x, int y)
{
  if (x>=y) return x;
else return y;
}
void main( )
{
  cout<<GetMax(5, 8);     //调用函数 GetMax
}
```

在写 C++ 程序时，要养成给程序加注释的习惯。C++ 中有两种注释方法：

(1) 多行注释：包含在定界符“/*”和“*/”之间的所有文本内容均为注释；

（2）单行注释：在符号“//”之后至本行末的所有文本内容均为注释。

0.4.2 变量、常量与数据类型

1. 标识符

标识符是用来标识变量、函数、类型等程序要素的有效字符序列。简单地说，标识符就是一个名字，如变量名、函数名等。C++ 语言中标识符的命名规则[①]如下：

（1）由字母、数字或下划线组成；

（2）字母或下划线作为第一个字符，其后跟零个或多个字母、数字、下划线；

（3）区分大小写；

（4）不能与关键字相同。

2. 基本数据类型

基本数据类型是 C++ 语言预定义的数据类型，共有 4 种：布尔型（bool）、字符型（char）、整型（int）和浮点型（float、double）。另外，C++ 语言还提供了 4 个修饰符：signed（有符号）、unsigned（无符号）、short（短型）和 long（长型），用来作为前缀修饰字符型、整型和浮点型。

3. 变量

在程序运行过程中可以改变的数据称为变量。在 C++ 语言中，使用变量前必须对变量进行声明，变量声明的格式如下：

```
数据类型 变量名 1,变量名 2,…,变量名 n;
```

例如：int length, data[n];

变量在声明后，编译器就会根据所属的数据类型，为变量分配一定数量的连续内存单元，并用变量名来标识。

变量具有以下属性：

（1）变量名：用于标识变量的标识符；

（2）地址：变量所占据的存储单元的首地址，变量的地址属性也称为左值；

（3）大小：变量所占据的存储单元的数量（以字节数来度量）；

（4）类型：变量所取的值域以及对变量所能执行的运算集；

（5）值：变量所占据的存储单元的内容，变量的值属性也称为右值；

（6）作用域：变量被引用的语句范围。

4. 常量

在程序运行过程中不允许改变的数据称为常量。与变量相同，常量也对应着一组内存单元，也有自己所属的数据类型。C++ 语言提供了两种类型的常量：文字常量和符号

---

① 对程序中的类型、函数、变量等程序要素的命名，采用一套好的命名规范可以提高程序的可读性。本书中采用如下命名规范：(1)类名和类型名的每个单词的首字母大写，如 LinkList；(2)函数名的每个单词的首字母大写，如 PrintList；(3)变量名尽量用一个单词且全部小写字母，如 data；如果用多个单词，则第一个单词全部小写，其他单词的首字母大写；(4)符号常量名的每个单词的首字母大写，如 MaxSize。

常量。

(1) 文字常量

文字常量就是以文字形式出现的常量，其数据类型是由它的表示方法决定的，例如，8 代表一个整型常量，－5.6 代表一个实型常量，"teacher"代表一个字符串常量。

(2) 符号常量

用标识符表示的常量称为符号常量。符号常量用常量名来访问，使用前需要声明，在声明时一定要赋初值，而且在初始化后不能再改变。符号常量的声明格式如下：

```
const  数据类型  常量名 = 常量值;
```

例如：const int MaxSize＝100；

5. 自定义数据类型

(1) 指针类型

指针就是地址，指针变量就是保存内存地址的变量，其声明形式如下：

```
T *指针变量名;
```

其中，T 是任意一个合法的数据类型，是该指针变量所指向变量的数据类型[①]。例如：

```
int *p;        //指针变量 p 指向 int 型变量
double *q;     //指针变量 q 指向 double 型变量
```

在 C++ 中，声明一个变量后，变量就在内存中占据一组内存单元，并且在其生存期内具有固定的存储地址，可以使用取地址运算符“&”来获取变量的地址；如果利用指针来访问变量，需要使用间接访问运算符“*”。例如：

```
int i=1, *p;    //声明一个整型变量 i 和一个指向整型变量的指针变量 p
p=&i;           //将变量 i 的存储地址赋给变量 p
cout<<*p;       //输出指针 p 所指变量值，即 1
```

(2) 枚举类型

在声明枚举类型时，需要把变量的可能取值一一枚举出来，其声明格式如下：

```
enum  枚举类型名 {变量值 1,变量值 2,…,变量值 n};
```

枚举出的可能取值又称为枚举常量。没有初始化时，枚举常量取默认值，即第一个常量的值为 0，其他常量的值为前一个常量的值加 1。例如：

```
enum Color {red, blue, yellow, green, white};
```

(3) 数组类型

数组是类型相同、数目一定的变量的有序集合，组成数组的变量称为数组元素。数组的声明形式如下：

```
数据类型  数组名[常量表达式 1][常量表达式 2]…[常量表达式 n];
```

---

① 注意：T 并不是指针变量本身的数据类型，任一指针变量(即地址)的数据类型都是 unsigned long int。

其中，数据类型给出了数组元素的类型；数组名是一个标识符，代表数组在内存中的起始地址；方括号的个数代表数组的维数；方括号中的常量表达式是数组某一维的长度；方括号中常量表达式的值的乘积是数组元素的个数。

对数组元素的访问是通过数组名和下标实现的，其一般形式如下：

```
数组名[下标表达式 1][下标表达式 2]…[下标表达式 n]
```

其中，下标表达式的个数与数组声明时的维数相同，下标表达式的值从 0 开始。例如：

```
int a[5], b[2][4];
```

一维数组 a 共有 5 个元素，分别是 a[0]，a[1]，a[2]，a[3]，a[4]；二维数组 b 共有 8 个元素，分别是 b[0][0]，b[0][1]，b[0][2]，b[0][3]，b[1][0]，b[1][1]，b[1][2]，b[1][3]。

(4) 结构类型[①]

数组是类型相同的变量的集合，组成数组的变量称为元素；而结构是不同类型的变量的集合，组成结构的变量称为成员（或域）。结构类型的声明形式如下：

```
struct  结构类型名
{
  数据类型  成员 1;
  数据类型  成员 2;
  ⋮
  数据类型  成员 n;
};
```

通常，结构声明在所有函数之外，位于 main 函数之前，从而使所声明的结构类型在程序的任何地方都可以使用。

声明一个结构并不分配内存，内存分配发生在声明这个结构类型的变量时。声明结构变量的一般形式如下：

```
结构类型名  结构变量 1,结构变量 2,…,结构变量 n;
```

一旦声明了结构变量，分配了内存空间，就可以使用操作符“.”来访问结构中的成员，左操作元为结构变量，右操作元为结构的成员；还可以使用操作符“->”来访问结构中的成员，左操作元为指针结构变量，右操作元为结构的成员。例如：

```
struct Node
{
  int data;
  Node * next;
};
Node t, * s;
```

① “结构”是根据英文单词 structure 翻译的，有些书将 structure 翻译为“结构体”。

上述语句声明了结构变量 t 和指针结构变量 s，t. data 表示访问结构变量 t 的 data 成员；s—>data 表示访问指针结构变量 s 指向的存储单元的 data 成员，见图 0-3。

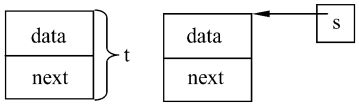

图 0-3　结构变量和指针结构变量

(5) 联合类型[①]

结构类型是每个成员分别占有自己的内存单元，结构变量所占内存长度是各成员所占内存长度之和；而联合类型是所有成员共享相同的内存单元，联合变量所占内存长度等于最长的成员所占的内存长度。联合类型的声明形式如下：

```
union  联合类型名
{
  数据类型   成员 1;
  数据类型   成员 2;
  ⋮
  数据类型   成员 n;
};
```

联合类型提供了在一个存储区域中操作不同类型数据的方法，在任意时刻，联合变量中只有一个成员是活动的，例如：

```
union Uarea
{
  int data;
  int * link;
};
```

见图 0-4。

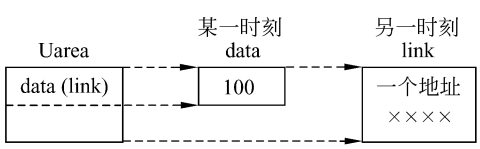

图 0-4　联合类型 Uarea 的存储形式

(6) 类类型

类是 C++ 语言面向对象程序设计的基础。

类是一组变量及其相关函数的封装体。类中的变量称为类的成员变量或数据成员，用于描述类的特征或状态；类中的函数称为类的成员函数或方法，用于处理成员变量，从而描述类的行为。类的声明形式如下：

---

① “联合”是根据英文单词 union 翻译的，有些书将 union 翻译为“共用体”。