第九章　程序设计实例

程序设计的关键是算法，算法是实际问题求解步骤的描述。第三章中已经介绍了如何用自然语言、流程图或 N-S 图来描述算法，但没有介绍算法设计方法。在实际编程中，肯定是先要设计算法，然后描述算法并实现。因此，本章首先通过实例介绍常用的算法设计方法，如迭代法、穷举法、递推法、递归法、回溯法、贪婪法、查找算法、排序算法等，然后介绍一个综合的模块化程序设计实例。

第一节　常用算法实例

一、迭代法

迭代法也称辗转法，是一种不断用变量的旧值递推新值的过程。迭代算法是用计算机解决问题的一种基本方法。它利用计算机运算速度快、适合做重复性操作的特点，让计算机对一组指令重复执行，在每次执行这组指令时，都从变量的原值推出它的一个新值。

迭代法的特点是：把一个复杂问题的求解过程转化为相对简单的迭代算式，然后重复执行这个简单的算式，直到得到最终解。迭代法分为精确迭代和近似迭代。精确迭代是指迭代算法本身提供了问题的精确解。如 N 个数求和、求均值、求方差等问题都适合使用精确迭代法解决。而近似迭代是指迭代算法不能得到精确解，而只能通过控制迭代次数或精度无限接近精确解。如求解方程根常用的“二分法”和“牛顿迭代法”就属于近似迭代法。

例 9-1　计算 s＝1＋2＋3＋4＋…＋100。

用精确迭代法求解，迭代方法如下：

① 首先确定迭代变量 s 的初始值为 0；

② 其次确定迭代公式 s＋i→s；

③ 当 i 分别取值 1，2，3，4，…，100 时，重复计算迭代公式 s＋i→s，迭代 100 次后，即可求出 s 的精确值。i 的取值是一个有序数列，所以可以由计数器产生，即使 i 的初始值为 1，然后每迭代一次就对 i 加 1。

完整的 C 语言程序及运行结果如下：

```
#include  <stdio.h>
int main()
{
    int i=1,s=0;
    while(i<=100)
    {
        s=s+i;                                          /* 迭代计算 */
        i++;
```

```
    }
    printf("和为:%d, 迭代次数为:%d\n",s,i-1);
    return 0;
}
```

运行结果为：

```
和为:5050,迭代次数为:100
```

迭代法的应用更主要的是数值的近似求解，它既可以用来求解代数方程，又可以用来求解微分方程。在科学计算领域，人们常会遇到求微分方程的数值解或解方程 f(x)＝0 等计算问题。这些问题无法用像求和或求均值那样的精确求解方法。例如，一般的一元五次或更高次方程、几乎所有的超越方程以及描述电磁波运动规律的麦克斯韦方程等，它们的解都无法用解析方法表达出来。为此，人们只能用数值计算的方法求出问题的近似解，而解的误差是人们可以估计和控制的。

我们以求解方程 f(x)＝0 为例说明近似迭代法的基本方法。

首先把求解方程变换成为迭代算式 x＝g(x)，然后由估计的一个根的初始近似值 x0 出发，应用迭代计算公式 xk＝g(xk)求出另一个近似值 x1，再由 x1 确定 x2…最终构造出一个序列 x0，x1，x2，…，xn，…就可逐次逼近方程的根。

例 9-2　用迭代法求方程 $x^3-x-1=0$ 在 x＝1.5 附近的一个根。

迭代方法如下：

① 首先将方程改写成迭代算式：

$$x=\sqrt[3]{x+1}$$

② 用给定的初始近似值 x0＝1.5 代入迭代算式的右端，得到：

$$x1=\sqrt[3]{1.5+1}=1.357\ 209$$

③ 再用 x1 作为近似值代入迭代算式的右端，又得到：

$$x2=\sqrt[3]{1.357\ 209+1}=1.330\ 861$$

按这种方法重复以上步骤，可以逐次求得更精确的值，这一过程即为迭代过程。显然，迭代过程就是通过重复执行一系列计算来获得问题近似答案，且每一次重复计算将产生一个更精确的答案。

用计算机算法实现这一计算过程，不可能让迭代无限制循环进行，因此只能通过控制迭代次数或精度无限接近精确解。下面介绍计算机算法的设计。

设方程为 f(x)＝0，用某种数学方法导出等价的形式 x＝g(x)，然后按以下步骤执行。

① 选一个方程的近似根，赋给变量 x0；

② 将 x0 的值保存于变量 x1，然后计算 g(x1)，并将结果存于变量 x0；

③ 当 x0 与 x1 的差的绝对值还不满足指定的精度要求时，重复步骤②的计算。

若方程有根，并且用上述方法计算出来的近似根序列收敛，则按上述方法求得的 x0 就认为是方程的根。上述算法用 C 程序的形式表示为：

```
{   x0=初始近似根;
    do {
        x1=x0;
        x0=g(x1);                                    /* 迭代公式 */
```

```
        } while(fabs(x0-x1)>Epsilon);                    /* Epsilon 是精度 */
        printf("方程的近似根是%f\n",x0);
}
```

例 9-3 编写求解例 9-2 问题的 C 语言程序。

```
#include  <stdio.h>
#include <math.h>
int main()
{
    float x0=1.5,x1;
    int n=0;
    do
    {  n++;
       x1=x0;
       x0=pow(x1+1,1.0/3);                               /* 迭代计算 */
       printf("第%d 次迭代后,近似根 x0=%f\n",n,x0);
    }while(fabs(x1-x0)>=1e-5);
    printf("满足精度要求的根为:%f, 迭代次数为:%d\n",x0,n);
    return 0;
}
```

程序运行结果为：

```
第 1 次迭代后,近似根 x0=1.357209
第 2 次迭代后,近似根 x0=1.330861
第 3 次迭代后,近似根 x0=1.325884
第 4 次迭代后,近似根 x0=1.324939
第 5 次迭代后,近似根 x0=1.324760
第 6 次迭代后,近似根 x0=1.324726
第 7 次迭代后,近似根 x0=1.324719
满足精度要求的根为:1.324719,迭代次数为:7
```

本书第三章的例 3-25 也是用迭代法求方程根的一个典型例子。

由以上介绍可知，使用近似迭代法构造算法的基本方法是：首先确定一个合适的迭代公式，选取一个初始近似值以及解的误差，然后用循环处理实现迭代过程。终止循环过程的条件是前后两次得到的近似值之差的绝对值小于或等于预先给定的误差，并认为最后一次迭代得到的近似值为问题的解。

具体使用迭代法求根时应注意以下两种可能发生的情况。

① 如果方程无解，算法求出的近似根序列就不会收敛，迭代过程会变成死循环，因此在使用迭代算法前应先考察方程是否有解，并在程序中对迭代的次数给予限制。

② 方程虽然有解，但迭代公式选择不当，或迭代的初始近似根选择不合理，也会导致迭代失败。

二、穷举法

穷举法是对可能是解的众多候选解按某种顺序进行逐一枚举和检验，并从中找出那些符合要求的候选解作为问题的解。穷举法的基本思想是，首先根据问题的部分条件预估答案的范围，然后在此范围内对所有可能的情况进行逐一验证，直到全部情况均通过了验证为止。若某个情况使验证符合题目的全部条件，则该情况为本题的一个答案；若全部情况验证结果均不符合题目的全部条件，则说明该题无答案。

穷举法由于会进行大量的重复计算，自然会用到程序的循环结构，因此灵活运用循环结构进行程序设计是穷举法程序设计的关键。

例 9-4　将一张面值为 100 元的人民币换成面值为 20 元、10 元和 5 元的人民币，且各种面值人民币至少要有一张，请列出所有可能的换法。

解题思路　这是一个典型的穷举问题。如果用 x、y、z 分别代表 20 元、10 元、5 元的数量，根据题意可列出方程：20x＋10y＋5z＝100。最简单的解题方法是：假设一组 x、y、z 的值，直接代入方程求解，若满足方程则是一组解。那么，通过循环在变量的取值范围内不断变化 x、y、z 的值，穷举 x、y、z 全部可能的组合，即可得到问题的全部解。程序如下：

```
#include  <stdio.h>
int main()
{
    int x,y,z,n=0;
    for(x=1;x<=5;x++)                                   /* 变量 x 为 20 元面值人民币的张数 */
        for(y=1;y<=10;y++)                              /* 变量 y 为 10 元面值人民币的张数 */
            for(z=1;z<=20;z++)                          /* 变量 z 为 5 元面值人民币的张数 */
            {
                if(20 * x+10 * y+5 * z==100)
                {
                    n++;
                    printf("第%d 种换法:x=%d,y=%d,z=%d\n",n,x,y,z);
                }
            }
    printf("穷举所有可能,共有%d 种换法。\n",n);
    return 0;
}
```

程序运行结果为：

```
第 1 种换法:x=1,y=1,z=14
第 2 种换法:x=1,y=2,z=12
第 3 种换法:x=1,y=3,z=10
第 4 种换法:x=1,y=4,z=8
第 5 种换法:x=1,y=5,z=6
第 6 种换法:x=1,y=6,z=4
第 7 种换法:x=1,y=7,z=2
第 8 种换法:x=2,y=1,z=10
```

第 9 种换法:x=2,y=2,z=8
第 10 种换法:x=2,y=3,z=6
第 11 种换法:x=2,y=4,z=4
第 12 种换法:x=2,y=5,z=2
第 13 种换法:x=3,y=1,z=6
第 14 种换法:x=3,y=2,z=4
第 15 种换法:x=3,y=3,z=2
第 16 种换法:x=4,y=1,z=2
穷举所有可能,共有 16 种换法。

例 9-5 判断真假问题。张三说李四说谎,李四说王五说谎,王五说张三和李四都在说谎。这三个人中谁说的是真话,谁说的是假话?

解题思路 本题用穷举法求解。假设三个人所说话的真假用变量 zhang、li、wang 表示,相应变量的值等于“1”表示该人说的是真话;等于“0”表示该人说的是假话。则可根据题意列出求解的条件。

① 题意 1:张三说“李四说谎”

a. 若张三说“真话”,则李四说“假话”(zhang==1&&li==0)

b. 若张三说“假话”,则李四说“真话”(zhang==0&&li==1)

所以题意 1 可以表示为条件 1:(zhang==1&&li==0)||(zhang==0&&li==1)

② 题意 2:李四说“王五说谎”

a. 若李四说“真话”,则王五说“假话”(li==1&&wang==0)

b. 若李四说“假话”,则王五说“真话”(li==0&&wang==1)

所以题意 2 可以表示为条件 2:(li==1&&wang==0)||(li==0&&wang==1)

③ 题意 3:王五说“张三和李四都在说谎”

a. 若王五说“真话”,则张三和李四都在说“假话”((wang==1&&(zhang==0&&li==0))

b. 若王五说“假话”,则张三和李四不都在说“假话”((wang==0&&(zhang==1||li==1))

所以题意 3 可以表示为条件 3:((wang==1&&(zhang==0&&li==0))||((wang==0&&(zhang==1||li==1))

上述 3 个条件之间是“与”的关系。将表达式进行整理就可得到问题求解的完整的条件:

```
((zhang&&!li)||(!zhang&&li)) && ((li&&!wang)||(!li&&wang)) && ((wang&&!zhang&&!li)||(!wang&&(zhang||li)))
```

用一个三重循环穷举 zhang、li、wang 所有可能的取值组合,代入上述条件进行判断,使上述条件为“真”的情况即为所求。程序如下:

```
#include  <stdio.h>
int main()
{
    int zhang,li,wang;
    for(zhang=0; zhang<2; ++zhang)
      for(li=0; li<2; ++li)
        for(wang=0; wang<2; ++wang)
```

```
        {
            if (((zhang&&! li)||(! zhang&&li)) && ((li&&! wang)||(! li&&wang))&&
               ((wang&&! zhang&&! li)||(! wang&&(zhang||li))))
            {
                printf("张三说的是%s\n",zhang?"真话":"假话");
                printf("李四说的是%s\n",li?"真话":"假话");
                printf("王五说的是%s\n",wang?"真话":"假话");
            }
        }
}
```

程序运行结果为：

```
张三说的是假话
李四说的是真话
王五说的是假话
```

穷举法的特点是算法简单，容易理解，但运算量较大。对于可确定取值范围但又找不到其他更好的算法时，就可以采用穷举法。通常穷举法用来解决“有几种组合”、“是否存在”、“求解不定方程”等类型的问题。利用穷举法设计算法大多以循环控制结构实现。

三、递推法

递推是通过数学推导，将复杂的运算化解为若干重复的简单运算，而每一次简单运算的结果将作为下一次简单运算的输入，这样便能逐级计算出最终结果。

能采用递推法构造算法的问题有重要的递推性质，即当得到问题规模为 n－1 的解后，由问题的递推性质，能从已求得的规模为 1，2，…，n－1 的一系列解，构造出问题规模为 n 的解。这样，程序可从 n＝0 或 n＝1 出发，通过递推，获得规模为 n 的解。

例 9-6　采用递推法求 5 的阶乘值。

递推过程：初始条件为 fact(1)＝1，然后可逐次推得。

```
fact(2)=fact(1) * 2=1 * 2=2
fact(3)=fact(2) * 3=2 * 3=6
fact(4)=fact(3) * 4=6 * 4=24
fact(5)=fact(4) * 5=24 * 5=120
```

递推求阶乘的参考程序如下：

```
#include  <stdio.h>
int main()
{
    int fact=1,i=0;
    for(i=2;i<=5;i++)
      fact =fact  * i;
    printf(“5!=%d\n”,fact);
    return 0;
}
```

例 9-7　斐波那契数列为：1，1，2，3，5，8，13，21，34，55，…即满足

$$fib(n)=\begin{cases}1 & (n=1\text{ 或 }2\text{ 时})\\ fib(n-1)+fib(n-2) & (n>2\text{ 时})\end{cases}$$

请编写求斐波那契(Fibonacci)数列的第 n 项的函数 fib(n)。

用递推法编写 fib(n)函数如下：

```
int fib(int n)
{
    int i,f1=1,f2=1,f3;
    if(n==1||n==2)
        return 1;
    if(n>2)
    {
        for(i=3;i<=n;i++)
        {
          f3=f1+f2;
          f1=f2;
          f2=f3;
        }
        return f3;
    }
}
```

四、递归法

递归是设计和描述算法的一种有力的工具，由于它在复杂算法的描述中被经常采用，为此在进一步介绍其他算法设计方法之前先讨论它。

能采用递归描述的算法通常有这样的特征：为求解规模为 N 的问题，设法将它分解成规模较小的问题，然后从这些小问题的解构造出大问题的解，并且这些规模较小的问题也能采用同样的分解和综合方法，分解成规模更小的问题，并从这些更小问题的解构造出规模较大问题的解。特别地，当规模 N＝1 时，能直接得解。

例 9-8　用递归法编写例 9-7 所需函数 fib(n)。

用递归法编写 fib(n)函数如下：

```
int fib(int n)
{
    if(n==1||n==2) return 1;
    if(n>2) return fib(n-1)+fib(n-2);
}
```

递归算法的执行过程分递推和回归两个阶段。在递推阶段，把较复杂的问题(规模为 n)的求解推到比原问题简单一些的问题(规模小于 n)的求解。例如上例中，求解 fib(n)，把它推到求解 fib(n－1)和 fib(n－2)。也就是说，为计算 fib(n)，必须先计算 fib(n－1)和 fib(n－2)，而计算 fib(n－1)和 fib(n－2)，又必须先计算 fib(n－3)和 fib(n－4)。依次类推，直至计算 fib(2)和 fib(1)，分别能立即得到结果 1。在递推阶段，必须要有终止递推的情况，例如在 fib 函数中，当 n 为 2 或 1 时终止递推。

在回归阶段，当获得最简单情况的解后，逐级返回，依次得到稍复杂问题的解，例如得到fib(2) 和 fib(1)后，返回得到 fib(3)的结果……在得到了 fib(n－1)和 fib(n－2)的结果后，返回得到 fib(n)的结果。

在编写递归函数时要注意，函数中的局部变量和参数只是局限于当前调用层，当递推进入“简单问题”层时，原来层次上的参数和局部变量便被隐蔽起来。在一系列“简单问题”层，它们各自有自己的参数和局部变量。

由于递归引起一系列的函数调用，并且可能会有一系列的重复计算，递归算法的执行效率相对较低。当某个递归算法能较方便地转换为递推算法时，通常按递推算法编写程序。例如上例计算斐波那契数列的第 n 项的函数 fib(n)应采用例 9-7 所示的递推算法，即从斐波那契数列的前两项出发，逐次由前两项计算出下一项，直至计算出第 n 项。

本书第四章第三节详细介绍了递归方法。

五、回溯法

回溯法也称为试探法，该方法首先暂时放弃关于问题规模大小的限制，并将问题的候选解按某种顺序逐一枚举和检验。当发现当前候选解不可能是解时，就选择下一个候选解；若当前候选解除了不满足问题规模要求外，满足所有其他要求时，继续扩大当前候选解的规模，并继续试探。如果当前候选解满足包括问题规模在内的所有要求时，该候选解就是问题的一个解。在回溯法中，扩大当前候选解的规模，以继续试探的过程称为向前试探。放弃当前候选解，寻找下一个候选解的过程称为回溯。

回溯法是一种选优搜索法，按选优条件向前搜索，以达到目标。但当探索到某一步时，发现原先选择并不优或达不到目标，就退回一步重新选择，这种走不通就退回再走的技术称为回溯法。例 9-9 就是一个典型的利用回溯法求解的例子。

例 9-9　迷宫问题。在指定的迷宫中找一条从入口到出口的可通路径。

解题思路　图 9-1 所示的方块图表示迷宫，其中空白方块表示通道，黑色方块表示墙，在迷宫数组中分别用 0 和 1 表示。现要在迷宫中找一条从入口到出口的可通路径，基本的算法思想是：从入口处开始向前探路(探的方向有右、下、左和上)，当沿着某个方向往前探一步时，若可通则继续往前探，若不通则换个方向后再探。若 4 个方向上均不通(四个方向上的相邻方块要么是墙块，要么是已探过的通道块)，则按原路回退一步后换个方向再探。在探路的过程中，用一个二维数组记录迷宫的可通路径。

下面对程序做一些说明。

1. 程序中使用的函数

① 输出迷宫数组的函数 printMaze( )。

② 在迷宫中探寻可通路径的函数 findOnePath( )。

2. 函数 findOnePath( )中使用的数组

① maze[N1][N2] ——迷宫数组。数组元素值为 0 代表通(通道)，为 1 代表不通(墙)。

② stack[N1 * N2][2] ——可通路径数组。在程序中表现为堆栈，记录走迷宫的路径，第 1 个下标表示是当前正在试探的可通路径上的第几步，第 2 个下标表示该步所在位置的

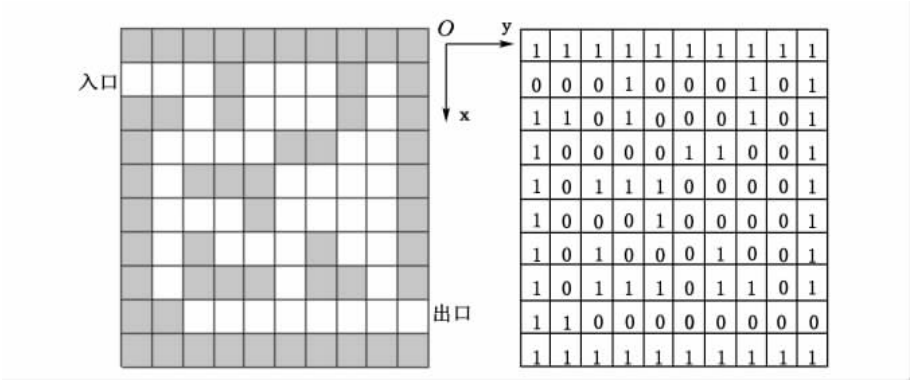

图 9-1　迷宫问题示意图

坐标是行坐标 x 还是列坐标 y。例如,第 0 步(入口处)的 x 坐标为 1,即 stack [0][0]=1,第 0 步(入口处)的 y 坐标为 0,即 stack[0][1]=0,若记入口处坐标为(1,0),则本例执行后输出的可通路径上各点的坐标依次为:(1, 0)→(1, 1)→(1, 2)→(2, 2)→(3, 2)→(3, 1)→(4, 1)→(5, 1)→(5, 2)→(5, 3)→(6, 3)→(6, 4)→(6, 5)→(7, 5)→(8, 5)→(8, 6)→(8, 7)→(8, 8)→(8, 9)。

下面是程序清单:

```
#include  <stdio.h>
#define N1 10
#define N2 10
#define SHOW
int findOnePath(int maze[N1][N2])          /* 在迷宫中用回溯法探寻一条可通路径的函数 */
{
    int stack[N1 * N2][2],top=0;
    int i,x=1,y=0,ok;
    stack[top][0]=x;              /* 当前位置(x,y)为入口位置(1,0),存入路径数组 stack */
    stack[top][1]=y;
    maze[1][0]=2;                                                  /* 入口做上标记 2 */

    while(1)                                        /* 回溯法探路。top 指向路径上的最新点 */
    {
        ok=0;                                                  /* ok 标志表示能否前探一步 */
        if (y+1<=N2-1 && maze[x][y+1]==0) {y=y+1;ok=1;}                  /* 往右试探 */
          else if (x+1<=N1-1 && maze[x+1][y]==0) {x=x+1;ok=1;}           /* 往下试探 */
            else if (y-1>=0 && maze[x][y-1]==0) {y=y-1;ok=1;}            /* 往左试探 */
              else if (x-1>=0 && maze[x-1][y]==0) {x=x-1;ok=1;}          /* 往上试探 */
        if(!ok)              /* 若在当前位置沿 4 个方向都不能前探,则回退一个位置后再探 */
        {
            if (x==1 && y==0)
            {
                printf ("很遗憾! 您回退到了入口或还在入口未动,迷宫无可通路径! \n");
```

```
                return 0;
            }
            printf("回退一步!");
            top--;                                  /* 回退一步,并修改当前位置(x,y) */
            x=stack[top][0];
            y=stack[top][1];
        }
        else
            /* 若沿某一方向可以前探,则将该点坐标存入路径数组 stack,并将该点做上标记 2 */
        {
            printf("前探一步!");
            top++;                          /* 添加新点的位置(x,y)到可通路径数组 stack */
            stack[top][0]=x;
            stack[top][1]=y;
            if (x==N1-2 && y==N2-1)                 /* 到达出口,输出可通路径,并返回 */
            {
                printf("\n* * * * * * * 恭喜* * * * * * * \n");
                printf("您已经探到了迷宫出口! 探得的一条迷宫可通路径为:\n");
                for (i=0;i<top;i++)
                    printf("(%d,%d)->",stack[i][0],stack[i][1]);
                printf("(%d,%d)\n",stack[top][0],stack[top][1]);
                return 1;
            }
            maze[x][y]=2;                           /* 将刚探得的点(x,y)的值标为 2 */
        }
        #ifdef SHOW    /* 条件编译语句旨在输出在探路径,若不要,可将前面的#define SHOW
                          注释掉 */
            printf("正在探寻的可通路径为:\n");
            for (i=0;i<top;i++)
                printf("(%d,%d)->",stack[i][0],stack[i][1]);
            printf("(%d,%d)\n",stack[top][0],stack[top][1]);
        #endif
    }
}
void printMaze(int maze[N1][N2])                    /* 输出迷宫数组的函数 */
{
    int i,j;
    printf("迷宫数组为:\n");
    for(i=0;i<N1;i++)
    {
        for(j=0;j<N2;j++)
            printf("%2d",maze[i][j]);
        printf("\n");
```

```
    }
}
int main()
{
    int    A[N1][N2]={ 1,1,1,1,1,1,1,1,1,1,
                       0,0,0,1,0,0,0,1,0,1,
                       1,1,0,1,0,0,0,1,0,1,
                       1,0,0,0,0,1,1,0,0,1,
                       1,0,1,1,1,0,0,0,0,1,
                       1,0,0,0,1,0,0,0,0,1,
                       1,0,1,0,0,0,1,0,0,1,
                       1,0,1,1,1,0,1,1,0,1,
                       1,1,0,0,0,0,0,0,0,0,
                       1,1,1,1,1,1,1,1,1,1 };
    printMaze(A);
    findOnePath(A);
    return 0;
}
```

在本程序中，所设置的条件编译语句可以将每一步前探或回退后的可通路径显示出来，用以对回溯过程进行跟踪。整个过程可以通过图 9-2 形象地描绘出来。图 9-2(a)演示从入口连续向前探寻 12 步后到达 b 位置。图 9-2(b)演示从 b 位置开始，连续回退 8 步后到达 a 位置。图 9-2(c)演示从 a 位置开始，连续前探 14 步后到达迷宫出口。最终，根据本程序探寻到的从入口到出口的一条迷宫可通路径如图 9-2(c)中的实线箭头所示。

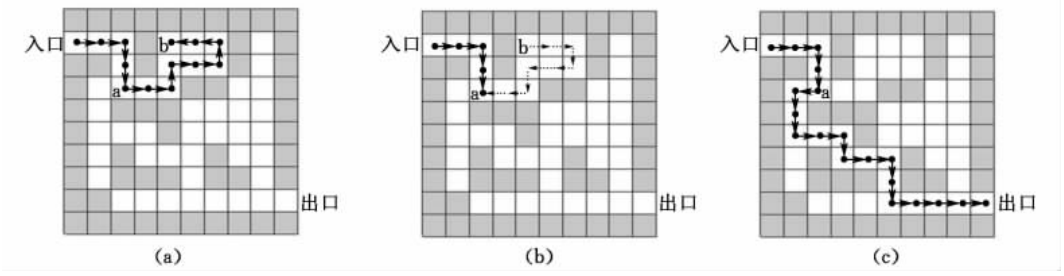

图 9-2　例 9-9 程序实现的前探与回退过程

(a) 前探 12 步至 b 位置；(b) 回退 8 步至 a 位置；(c) 继续前探 14 步至出口

六、贪婪法

贪婪法是一种不追求最优解，只希望得到较为满意解的方法。贪婪法一般可以快速得到满意的解，因为它省去了为找最优解要穷举所有可能而必须耗费的大量时间。贪婪法常以当前情况为基础作最优选择，而不考虑各种可能的整体情况，所以贪婪法不需要回溯。

例 9-10　找零钱问题。当前使用的货币面额分别是 100、50、20、10、5、2、1 元。在购物找钱时，怎样找钱才能使找零的张数最少？请根据找零的金额输出找零的方案。

解题思路　用贪婪法解决这个问题，在找钱时，为使找回的零钱的张数最少，不考虑找

零钱的所有可能方案，而是从最大面值的币种开始，按递减的顺序考虑各币种，先尽量用大面值的币种，当不足大面值币种的金额时才去考虑下一种较小面值的币种。实现程序如下：

```
#include  <stdio.h>
int main()
{
    int money,k,i,change[20],total;      /* money:将被找零的总额;change:零钱数组 */
    int denomination[7] ={100,50,20,10,5,2,1};      /* denomination:纸币面额数组 */
    printf("输入欲找零钱的总额[1-99]:");
    scanf("%d",&money);
    for(i=0;i<7;i++)
        if(money>=denomination[i])
            break;                                  /* 跳过面额比总额大的纸币单位 */
    k=0;
    total=money;
    while(i<7&&total!=0)
    {
        if(total>=denomination[i])
        {
            change[k]=denomination[i];
            total=total- change[k];
            k++;
        }
        else
        {
            i++;
        }
    }
    printf("总额为%d 元的找零钱方案为:",money);
    printf("%d=%d",money,change[0]);
    for(i =1;i <k;i++)
        printf("+%d",change[i]);
    printf("\n 此贪婪法求得的最少找零张数为%d 张。\n",k);
    return 0;
}
```

本程序的一次运行结果如下：

```
输入欲找零钱的总额[1-99]:98↙
总额为 98 元的找零钱方案为:98= 50+20+20+5+2+1
此贪婪法求得的最少找零张数为 6 张。
```

七、查找算法

本节主要介绍最简单的 2 种查找方法：顺序查找和二分查找。顺序查找用于一般的数据查找，二分查找用于有序数组的查找。

1. 顺序查找

顺序查找一般用于在数组(或链表)中查找指定的元素(或结点)。要查找的数一般称为关键字,顺序查找就是将给定的关键字逐个与数组元素(或链表结点)进行比较,如果有与关键字相等的数组元素(或链表结点),则查找成功并输出有关信息,否则查找失败并提示没有匹配值。例 9-11 是一个简单的顺序查找算法实例。

例 9-11 在数组中查找指定的元素并输出其位置。

```
#include  <stdio.h>
int search(int a[],int n,int key)
{
    int i;
    for(i=0;i<n;i++)
        if(a[i]==key)
            return i;
    return -1;
}
int main()
{
    int position,Key;
    int A[10]={2,8,13,27,55,89,73,4,11,36};
    printf("请输入要查找的关键字:");
    scanf("%d",&Key);
    position=search(A,10,Key);
    if(position!=-1)
        printf("关键字是数组中的元素 A[%d]\n",position);
    else
        printf("关键字不是数组中的元素! \n");
    return 0;
}
```

本程序的一次运行结果如下:

```
请输入要查找的关键字:11↙
关键字是数组中的元素 A[8]
```

2. 二分查找

二分查找要求数组是有序数组,如果不是有序数组,必须先排序,然后才能使用二分查找法。二分查找的算法思想如下:

① 把查找范围对分为两部分,看要查找的值是否属于中点。

② 如果要查找的值不属于中点,则看这个值落在哪一部分。

③ 因为已经判断过中点,所以可在该部分中排除中点,然后对剩余部分继续使用二分法。

④ 逐步缩小查找范围,如果查找到要求的值,则输出相关信息,否则输出查找失败信息。

例 9-12　在一个有序的数列中查找指定的数。

解题思路　采用的是二分查找法：假设有序数列按递增的顺序存放在一个数组中。查找时，先查找中间的数，若待查的数与中间数相等，则查找成功；若待查的数比中间数小，则到数组的前半部分查找；否则待查数比中间数大，则到数组的后半部分查找。这样一直查下去，如果找到了要查的数，则查找成功，否则查找失败，表明有序数列中不存在要查找的数。

此法要用到 3 变量：front、tail 和 mid。分别指示被查找的那一部分有序数列的头、尾和中间位置。如在有序数列(11,22,33,44,55,66,77,88,99)中查找 66，则二分查找过程如图 9-3 所示。

二分查找的算法流程如图 9-4 所示。根据该算法流程编写的二分查找程序如下。

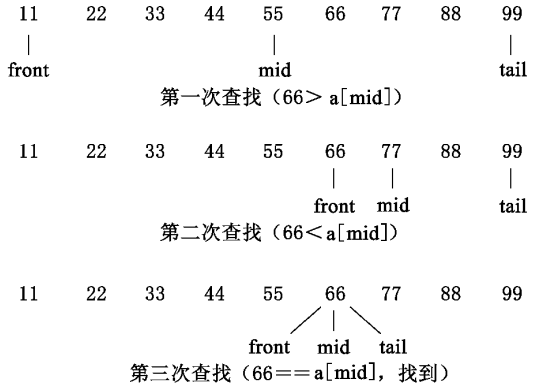

图 9-3　二分查找 66 的过程

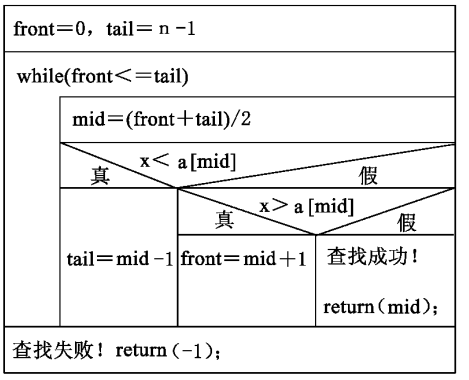

图 9-4　二分查找法流程图

```
#include  <stdio.h>
int bisearch(int a[],int n,int key)
                /* 二分法查找函数:在数组 a 中查找 key,找到则返回其位置,n 是 a 数组长度 */
{
    int front=0,tail=n-1,mid;
    while(front<=tail)
    {
        mid=(front+tail)/2;
        if (key<a[mid])
            tail=mid-1;
        else
            if (key>a[mid])
                front=mid+1;
            else
                return(mid);
    }
    return(-1);
}
```

```
int main()
{
    int position,Key;
    int A[9]={11,22,33,44,55,66,77,88,99};
    printf("请输入要查找的关键字:");
    scanf("%d",&Key);
    position=bisearch(A,9,Key);
    if(position!=-1)
        printf("关键字是数组中的元素 A[%d]\n",position);
    else
        printf("关键字不是数组中的元素! \n");
    return 0;
}
```

本程序的一次运行结果如下：

```
请输入要查找的关键字:66↙
关键字是数组中的元素 A[5]
```

八、排序算法

排序算法有直接插入排序、折半插入排序、希尔排序、冒泡排序、快速排序、简单选择排序、堆排序、归并排序等，在数据结构课程中将会详细阐述。本节主要介绍最简单的 2 种排序方法：冒泡排序和简单选择排序。

1. 冒泡排序

对 n 个数进行升序排列，冒泡排序的算法思想如下：

① 从第 1 个数开始，第 1 个数与第 2 个数进行比较，若为逆序，则交换。然后比较第 2 个数与第 3 个数，若为逆序，则交换。依此类推，直至第 n－1 个数与第 n 个数比较并处理后为止，则完成第一趟冒泡排序。此时，最大的数已经排好在第 n 个位置上。

② 对前 n－1 个数按上述同样的方法进行第二趟冒泡排序。这样，次大的数将排好在第 n－1 个位置上。

③ 对前 n－2 个数按上述同样的方法进行第三趟冒泡排序。这样，倒数第三大的数将排好在第 n－2 个位置上。

④ 重复上述过程，总共进行 n－1 趟排序，则排序结束。

例 9-13 输入 5 个数，用冒泡排序方法将 5 个数按升序排列。

图 9-5 形象地描述了 5 个数进行 4 趟排序的过程。从图 9-5 可以看出，每相邻两个数进行比较，轻(小)的往上冒(像水中的气泡)，重(大)的往下沉(像落水的石头)。“冒泡”法估计是因为这个特点而得名。

从图 9-6 中的统计数据可以推出：对于 N 个数的排序，需进行 N－1 趟排序，第 i 趟排序需进行 N－i 次两两比较。

冒泡排序算法的流程图如图 9-7 所示(用两层嵌套循环实现)。根据冒泡排序算法流程图，编写例 9-13 的程序如下：

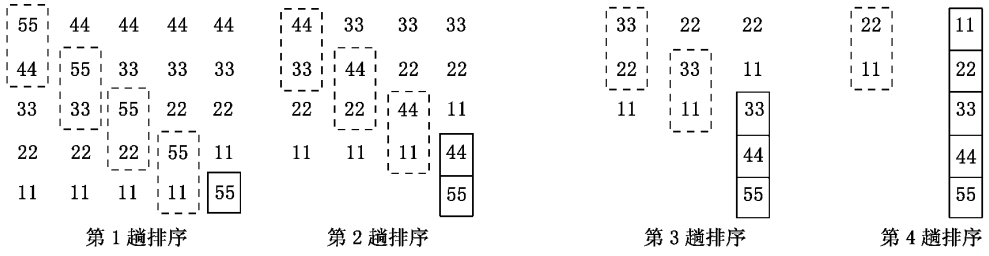

图 9-5　“气泡法”排序示意图

5 个数排序	两两比较次数	未排序数个数	已排序数个数
第 1 趟排序	4	4	1
第 2 趟排序	3	3	2
第 3 趟排序	2	2	3
第 4 趟排序	1	0	5

图 9-6　对应于图 9-5 的统计数据

```
#include  <stdio.h>
#define N 5
void bubbleSort(int a[],int n)                          /* 冒泡排序函数 */
{   int i,j,t;                                          /* i,j 做循环变量,t 做交换用的临时变量 */
    for(i=1;i<N;i++)                                    /* 第 i 趟比较,共 N-1 趟 */
        for(j=0;j<N-i;j++)                              /* 第 i 趟中两两比较 N-i 次 */
            if (a[j]>a[j+1])                            /* 两相邻数若为逆序则交换 */
            {
                t =a[j];
                a[j] =a[j+1];
                a[j+1] =t;
            }
}
int main()
{
    int i,A[N];
    printf("请输入%d 个整数:\n",N);
    for(i=0;i<N;i++)
        scanf("%d",&A[i]);                              /* 输入 N 个整数 */
    bubbleSort(A,N);
    printf("这%d 个整数按从小到大的顺序排列为:\n",N);
    for(i=0;i<N;i++)
        printf("%5d",A[i]);
    return 0;
}
```

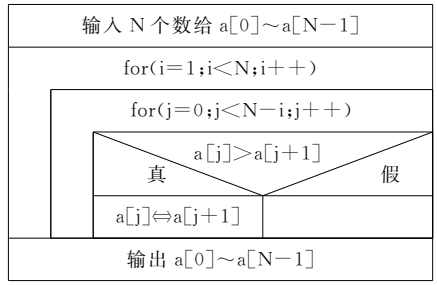

图 9-7　冒泡排序算法流程图

2. 简单选择排序

对 n 个数进行升序排列，简单选择排序的算法思想如下：

① 首先通过 n－1 次比较，从 n 个元素中找出值最小的元素，将它与第一个元素交换（第一趟排序）。

② 再通过 n－2 次比较，从剩余的 n－1 个元素中找出值次小的元素，将它与第二个元素交换（第二趟排序）。

③ 重复上述操作，共进行 n－1 趟排序后，排序结束。

例 9-14 输入 10 个数，用简单选择排序方法将 10 个数按升序排列。

解题思路 对 n 个数进行排序的过程为：首先从 n 个整数中选出值最小的整数，将它交换到第一个元素位置，再从剩余的 n－1 个整数中选出值次小的整数，将它交换到第二个元素位置，重复上述操作 n－1 次后，排序结束。

程序如下：

```
#include  <stdio.h>
#define N 10
void smp_seleSort(int a[], int n)                          /* 简单选择排序 */
{
    int i,j,k,temp;
    for(i=0;i<n-1;i++)                                     /* 共 n-1 趟排序 */
    {
        k=i;
        for(j=i+1;j<n;j++)                                 /* 找出关键字最小的元素的位置 */
            if(a[j]<a[k])
                k=j;
        if(i!=k)                                           /* 交换 */
        {
            temp=a[i];
            a[i]=a[k];
            a[k]=temp;
        }
    }
}
int main()
{
    int i,A[N];
    printf("请输入%d 个整数:\n",N);
    for(i=0;i<N;i++)
        scanf("%d",&A[i]);
    smp_seleSort(A,N);                                     /* 调用简单选择排序函数 */
    printf("排序后的输出为:\n");
    for(i=0;i<N;i++)
        printf("%5d",A[i]);
```

```
    return 0;
}
```

该程序的一次运行结果如下：

```
请输入 10 个整数：
23  43   8   5   46  13   4   32   57   88
排序后的输出为：
     4     5     8   13   23   32   43   46   57   88
```

第二节　模块化程序设计实例

一、模块化程序设计基础

C 语言是结构化程序设计语言，它的程序设计特点就是以函数为主要模块的模块化程序设计。

1. 函数和模块

用 C 语言这种结构化程序设计语言进行程序设计，一般把逻辑功能完全独立或相对独立的程序部分设计成函数，且每个函数只完成一个功能。这样，一个函数就是程序的一个功能模块。函数的使用既符合只有一个入口和出口的结构化程序设计原则，也特别适合开发多功能大型程序。

函数实际上就是封装起来的一段有名字的程序代码，这种封装将函数的内部与外部分开。从外部看，函数的调用者关心的是函数能实现什么样的功能，而不必关心函数体的实现。这样，函数的定义者和调用者可以不是同一个人，这种特点适合多人共同开发多功能大型程序。

2. 文件

C 程序分为后缀为“.c”和“.h”文件的两类文件。“.c”文件为包含实际程序代码的基本程序文件，“.h”文件是为基本程序文件提供必要信息的辅助性文件。

“.h”文件称为头文件，头文件内容的安排可遵循如下原则：

① 头文件中只写那些不实际生成代码，不导致实际分配存储空间的描述。例如，可写函数原型声明、不写函数定义；可以包含标准库头文件；可用 extern 声明外部变量，但不定义变量。

② 可以包括各种公用的类型定义。例如，公用的结构和联合类型。

③ 可以包括各种公用的宏定义。

④ 只用文件包含命令(#include)包含“.h”头文件，不用它包含“.c”程序文件。

一个实用的系统一般都会由多个文件组成，文件的组织可遵循如下原则：

① 首先根据程序的规模和实现的功能模块，将整个程序规划成一个或多个“.c”文件。一般一个文件可以包含一个或多个函数模块。例如，主函数通常单独建立一个文件，其中也可以包含相关的菜单选择函数；与输入和输出有关的功能可以放在一个文件中，但如果输入和输出都比较复杂，则可各自放在一个文件中。

② 然后根据源程序的文件数量和功能，设计相应的头文件。如果源文件比较复杂，完全可能需要为每个源文件设计一个头文件。

头文件编写的注意事项：

① 把所有公用的类型定义、公用的结构、联合和枚举声明、公用的宏定义放在适当的头文件中，提供给各个文件参考。

② 如果只有一个文件需要某个标准头文件，则不要将它放在公用的头文件中，应让这个源程序文件直接包含它，以提高编译效率。

③ 对于在一个源程序文件中定义，而在其他文件中使用的东西，需要在相应头文件中声明(函数原型或变量的外部声明)。

④ 要避免对头文件的重复编译。头文件中可以包含其他头文件，或一个程序文件中包含多个头文件，都可能引起对同一个头文件的重复包含问题。因此，要使用预处理命令来避免。

二、模块化程序设计实例

上一节介绍了模块化程序设计的基础知识，本节将以一个学生成绩管理系统为例，说明多文件模块化程序设计。

例 9-15 编写一个小型学生成绩管理系统，主要实现如下功能：

① 用结构数组存储记录。每一条记录包括一个学生的学号、姓名、3 门课成绩等。

② 输入功能：可以一次完成若干条记录的输入。

③ 显示功能：完成全部学生记录的显示。

④ 查找功能：完成按姓名查找学生记录，并显示。

⑤ 排序功能：能按平均分排序或按学号排序。

⑥ 插入功能：在指定位置插入一条学生记录。

⑦ 删除功能：能删除指定记录。

⑧ 能将学生成绩信息存在文件中，或能将文件中的成绩信息导入。

要实现题目要求的所有功能，必须编写多个函数模块，如果按本章第一节中介绍的基础知识组织成多文件工程，则适合多人进行合作开发。下面就具体介绍设计过程。

1. 菜单设计

设计如下所示的“主菜单”：

```
* * * * * * * * * * * * * * * 主菜单 * * * * * * * * * * * * * * *
                    1. 输入记录
                    2. 显示所有记录
                    3. 对所有记录进行排序
                    4. 按姓名查找记录并显示
                    5. 插入记录
                    6. 删除记录
                    7. 将所有记录保存到文件
                    8. 从文件中读入所有记录
                    9. 退出
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

请选择操作(1～9)：

2. 数据存储方式及定义

学生的成绩等信息用结构数组进行存储。建立学生信息数据结构 struct student，结构包括学号、姓名、成绩、总分、平均分、名次。结构类型及结构类型数组定义如下。

```
/* 定义结构类型 */
typedef struct student
{
    char no[11];
    char name[15];
    float score[N];
    float sum;
    float average;
    int order;
}STUDENT;
/* 定义结构数组 */
STUDENT stu[30];
```

3. 函数设计

(1) 输入函数

函数原型：int input(STUDENT *stud,int n)，

功能：利用 for 循环语句和 scanf、gets、getchar 函数完成对结构数组的输入，存放 n 个学生的信息(包括学号、姓名、性别、成绩等)。

参数说明：

STUDENT *stud：接收主函数传过来的数组首地址。

int n ：整型形参变量，接收 main()传过来的数组长度。

(2) 显示函数

函数原型：void print(STUDENT *stud,int n)；

功能：将 stud 指向的 n 条学生记录显示出来。

(3) 排序函数

函数原型：void sort(STUDENT *stud,int n)；

功能：将 stud 指向的 n 条学生记录按照学号或者总分进行排序。

(4) 查找函数

函数原型：void search(STUDENT *stud,int n)；

功能：在 stud 指向的 n 条学生记录中查找指定的学生记录。

(5) 插入函数

函数原型：int insert(STUDENT *stud,int n)；

功能：在 stud 指向的 n 条学生记录中插入一条新的记录。

(6) 删除函数

函数原型：int Delete(STUDENT *stud,int n)；

功能：在 stud 指向的 n 条学生记录中删除一条记录。

(7) 保存数据到文件

函数原型:void fileWrite(STUDENT *stud,int n);

功能:将 stud 指向的 n 条学生记录存入指定的文件。

(8) 从文件导入数据

函数原型:int fileRead(STUDENT *stud);

功能:将文件中的数据导入 stud 指向的结构体数组中,并返回读入记录的条数。

4. 文件设计

表 9-1 给出了程序的文件和函数组成。

表 9-1　　文件及函数组成

文件名	函数原型或其他定义	功能说明
sms_struct.h	系统头文件	引用系统库函数
	常量定义	学生的成绩门数
	结构体定义	定义存储学生信息的结构类型
	声明全部函数原型	以便让工程中文件引用相关函数
sms_main.c	int main( )	总控函数
	int menu_select( )	主菜单选择函数
sms_input.c	int input(STUDENT *stud,int n)	输入一条或多条记录
sms_print.c	void print(STUDENT *stud,int n)	显示所有记录
sms_sort.c	void sort(STUDENT *stud,int n)	排序总控函数
	void sort_sum(STUDENT *stud,int n)	根据总分 sum 的值按降序排列
	void sort_no(STUDENT *stud,int n)	根据学号 no 的值按升序排列
	void swap(STUDENT *stud1,STUDENT *stud2)	交换 2 条记录
sms_search.c	void search(STUDENT *stud,int n)	按姓名查找记录
sms_insert.c	int insert(STUDENT *stud,int n)	在指定位置插入一条记录
sms_Delete.c	int Delete(STUDENT *stud,int n)	按姓名删除相应记录
sms_fileRW.c	void fileWrite(STUDENT *stud,int n)	将所有记录全部存入指定文件
	int fileRead(STUDENT *stud)	将文件中所有记录全部导入结构数组

(1) 头文件 sms_struct.h

成绩管理系统中公共接口的头文件内容如下:

```
/* sms_struct.h */
#ifndef sms_stuct_INCLUDED                          /* 防止下面的内容被重复包含 */
#define sms_stuct_INCLUDED
#include<stdio.h>                                   /* 包含 printf()、scanf()等函数 */
#include<string.h>                                  /* 包含 strlen()、strcpy()等函数 */
#include<stdlib.h>                                  /* 包含 atoi()函数 */
#define N 3                                         /* 定义常数,表示成绩门数 */
typedef struct student                              /* 定义结构体类型 */
```

```
{
    char no[11];
    char name[15];
    float score[N];
    float sum;
    float average;
    int order;
}STUDENT;
/* 函数声明 */
int input(STUDENT *stud,int n);                                    /* 输入记录 */
void print(STUDENT *stud,int n);                                   /* 显示记录 */
void sort(STUDENT *stud,int n);                                    /* 排序记录 */
void search(STUDENT *stud,int n);                                  /* 查找记录 */
int insert(STUDENT *stud,int n);                                   /* 插入记录 */
int Delete(STUDENT *stud,int n);                                   /* 删除记录 */
void fileWrite(STUDENT *stud,int n);                               /* 存储记录 */
int fileRead(STUDENT *stud);                                       /* 导入记录 */
#endif
```

（2）主函数文件 sms_main.c

```
/* sms_main.c */
#include "sms_struct.h"
int menu_select()                                                  /* 菜单选择模块 */
{
    char s[3];  int c;
    printf("\n      * * * * * * * * 主菜单* * * * * * * \n");
    printf("            1. 输入记录\n");
    printf("            2. 显示所有记录\n");
    printf("            3. 对所有记录进行排序\n");
    printf("            4. 按姓名查找记录并显示\n");
    printf("            5. 插入记录\n");
    printf("            6. 删除记录\n");
    printf("            7. 将所有记录保存到文件\n");
    printf("            8. 从文件中读入所有记录\n");
    printf("            9. 退出\n");
    printf("         * * * * * * * * * * * * * * * * * * \n\n");
    do
    {
        printf("      请选择操作(1- 9):");
        scanf("%s",s);
        c=atoi(s);
    }while(c<0||c>9);                                  /* 选择项不在 0- 9 之间,则重输 */
    return(c);                                 /* 返回选择项,主程序根据该值调用相应的函数 */
}
```

```
int main()                                                    /* * * * * * 主函数* * * * * * */
{
    int n=0;
    STUDENT student[20];                                          /* 定义结构数组 */
    for(;;)                                                       /* 无限循环 */
    {
        switch(menu_select())          /* 调用主菜单函数,返回值整数作开关语句的条件 */
        {
            case 1: n=input(student,n);break;                     /* 新建记录 */
            case 2: print(student,n);break;                       /* 显示记录 */
            case 3: sort(student,n);break;                        /* 排序记录 */
            case 4: search(student,n);break;                      /* 查找记录 */
            case 5: n=insert(student,n);break;                    /* 插入记录 */
            case 6: n=Delete(student,n);break;                    /* 删除记录 */
            case 7: fileWrite(student,n);break;                /* 记录保存到文件 */
            case 8: n=fileRead(student);break;                       /* 读文件 */
            case 9: exit(0);                                      /* 程序结束 */
        }
    }
    return 0;
}
```

（3）输入记录文件 sms_input.c

```
/* sms_input.c */
#include "sms_struct.h"
int input(STUDENT *stud,int n)                                /* 录入学生记录的模块 */
{
    int i,j;
    float s;
    char sign;
    i=0;
    while(sign!='n'&&sign!='N')
    {
        printf("\n 请按如下提示输入相关信息.\n\n");
        printf("输入学号:");
        scanf("%s",stud[n+i].no);                                 /* 输入学号 */
        printf("输入姓名:");
        scanf("%s",stud[n+i].name);                               /* 输入姓名 */
        printf("输入%d 个成绩:\n",N);                          /* 提示开始输入成绩 */
        s=0;                                                    /* 总分 s,初值为 0 */
        for(j=0;j<N;j++)                                /* 输入 N 门成绩,需要循环 N 次 */
        {
            do {
                printf("score[%d]:",j);                       /* 提示输入第几门成绩 */
```

```
            scanf("%f",&stud[n+i].score[j]);                    /* 输入成绩 */
            if(stud[n+i].score[j]>100||stud[n+i].score[j]<0)
                printf("非法数据,请重新输入! \n");      /* 确保成绩在 0-100 之间 */
            }while(stud[n+i].score[j]>100||stud[n+i].score[j]<0);
                s=s+stud[n+i].score[j];                         /* 累加各门成绩 */
        }
        stud[n+i].sum=s;                                        /* 将总分保存 */
        stud[n+i].average=(float)s/N;                           /* 求出平均值 */
        stud[n+i].order=0;                                  /* 未排名次前此值为 0 */
        printf ("该学生的总分为:%4.2f\n\t 平均分为:%4.2f\n", stud[n+i].sum,
            stud[n+i].average);
        printf("=====>提示:继续添加记录? (Y/N)");
        getchar();/* 把键盘缓冲区前面输入的回车键给读掉,不然后面 sign 读的是回车符 */
        scanf("%c",&sign);
        i++;
    }
    return(n+i);
}
```

（4）显示记录文件 sms_print.c

```
/* sms_print.c */
#include "sms_struct.h"
void print(STUDENT *stud,int n)                                 /* 显示记录模块 */
{
    int i=0;                                                    /* 统计记录条数 */
    if(n==0)
    {
        printf("\n 很遗憾,空表中没有任何记录可供显示! \n");
    }
    else
    {
        printf("* * * * * * * * * *   STUDENT  * * * * * * * * * * \n");
        printf("记录号  学号  姓名  成绩 1  成绩 2  成绩 3  总分  平均分  名次\n");
        printf("- - - - - - - - - - - - - - - - - - - - - - - - - - \n");
        while(i<n)
        {
            printf ("%-4d %-11s%-15s%5.2f%7.2f%7.2f %9.2f %6.2f %3d \n", i+1,
                stud[i].no,stud[i].name,stud[i].score[0],stud[i].score[1],
                stud[i].score[2], stud[i].sum, stud[i].average, stud[i].
                order);
            i++;
        }
        printf("* * * * * * * * * * * * * * * * * * * * * * * * * * \n\n");
    }
```

```
}
```

（5）记录排序文件 sms_sort.c

```
/* sms_sort.c */
#include "sms_struct.h"
void swap(STUDENT *stud1,STUDENT *stud2)                    /* 交换 2 条记录 */
{
    int i;
    char temp1[15];
    float temp2;
    float temp3;
    strcpy(temp1,stud1->no);
    strcpy(stud1->no,stud2->no);
    strcpy(stud2->no,temp1);
    strcpy(temp1,stud1->name);
    strcpy(stud1->name,stud2->name);
    strcpy(stud2->name,temp1);
    for(i=0;i<3;i++)
    {
        temp2=stud1->score[i];
        stud1->score[i]=stud2->score[i];
        stud2->score[i]=temp2;
    }
    temp3=stud1->sum;stud1->sum=stud2->sum;stud2->sum=temp3;
    temp3=stud1->average;stud1->average=stud2->average;stud2->average=temp3;
}
/* 排序模块,实现根据总分 sum 的值按降序排列 */
void sort_sum(STUDENT *stud,int n)
{
    int i,j;
    int maxPosition;
    for(i=0;i<n-1;i++)
    {
        maxPosition=i;
        for(j=i+1;j<n;j++)
        {
            if(stud[j].sum >stud[maxPosition].sum)
            {
                maxPosition =j;
            }
        }
        if(maxPosition!=i)
        {
            swap(&stud[i],&stud[maxPosition]);
```

```
        }
    }
    i=0;
    while(i<n)
    {
        stud[i].order=i+1;
        i++;
    }
    printf("按总分从高到低排名成功!!! \n");                    /* 排序成功 */
}
/* 排序模块,实现根据学号 no 的值按升序排列 */
void sort_no(STUDENT *stud,int n)
{
    int i,j;
    int minNoPosition;
    for(i=0;i<n-1;i++)
    {
        minNoPosition=i;
        for(j=i+1;j<n;j++)
        {
            if(atoi(stud[j].no)<atoi(stud[minNoPosition].no))
            {
                minNoPosition =j;
            }
        }
        if(minNoPosition!=i)
        {
            swap(&stud[i],&stud[minNoPosition]);
        }
    }
    printf("按学号从低到高排序成功!!! \n");                    /* 排序成功 */
}
void sort(STUDENT *stud,int n)                                 /* 排序总控模块 */
{
    char s[3];
    int c;
    printf("\n      * * * * * * * * * * 排序菜单* * * * * * * * * * \n");
    printf("            1. 按学号排序\n");
    printf("            2. 按总分排序\n");
    printf("          * * * * * * * * * * * * * * * * * * \n\n");
    do
    {
        printf(" 请选择操作(1- 2):");
```

```
        scanf("%s",s);
        c=atoi(s);
    }while(c<0||c>2);                                   /* 选择项不在 0- 2 之间重输 */
    switch(c)
    {
        case 1: sort_no(stud,n);break;                  /* 调用按总分排序函数 */
        case 2: sort_sum(stud,n);break;                 /* 调用按学号排序函数 */
    }
}
```

（6）查找记录文件 sms_search.c

```
/* sms_search.c */
#include "sms_struct.h"
void search(STUDENT *stud,int n)                        /* 查找记录模块 */
{
    int i=0;
    char s[15];                                         /* 存放姓名的字符数组 */
    printf("请输入您要查找的学生姓名:\n");
    scanf("%s",s);                                      /* 输入姓名 */
    while(i<n && strcmp(stud[i].name,s))
    {
        i++;
    }
    if(i==n)
        printf("\n 您要查找的是%s,很遗憾,查无此人! \n",s);
    else                                                /* 显示找到的记录信息 */
    {
      printf("* * * * * * * * * * Found * * * * * * * * * * * \n");
      printf(" 学号  姓名  成绩 1  成绩 2  成绩 3  总分  平均分  名次\n");
      printf("- - - - - - - - - - - - - - - - - - - - - - \n");
      printf ("%-11s%-15s%5.2f%7.2f%7.2f %9.2f %6.2f %3d \n", stud[i].no,stud[i].
            name,stud[i].score[0],stud[i].score[1],stud[i].score[2],stud[i].sum,
            stud[i].average,stud[i].order);
      printf("* * * * * * * * * * * * * * * * * * * * * * \n");
    }
}
```

（7）插入记录文件 sms_insert.c

```
/* sms_insert.c */
#include "sms_struct.h"
int insert(STUDENT *stud,int n)                         /* 在指定位置插入记录 */
{
    int i=0,j;
    float s;
    int position;
```

```
printf("请输入插入记录的位置:\n");
scanf("%d",&position);                                    /* 输入插入记录的位置 */
while(position<0 || position>n)
{
    printf("输入位置有误,请重新输入插入记录的位置:\n");
    scanf("%d",&position);
}
//将插入位置开始的所有记录向后移动
for(i=n-1;i>=position;i-- )
{
    strcpy(stud[i+1].no,stud[i].no);
    strcpy(stud[i+1].name,stud[i].name);
    stud[i+1].score[0]=stud[i].score[0];
    stud[i+1].score[1]=stud[i].score[1];
    stud[i+1].score[2]=stud[i].score[2];
    stud[i+1].sum=stud[i].sum;
    stud[i+1].average=stud[i].average;
    stud[i+1].order=stud[i].order;
}
//录入记录并插入
i=position;
printf("\n 请按如下提示输入相关信息.\n\n");
printf("输入学号:");
scanf("%s",stud[i].no);                                   /* 输入学号 */
printf("输入姓名:");
scanf("%s",stud[i].name);                                 /* 输入姓名 */
printf("输入%d 个成绩:\n",N);                              /* 输入成绩 */
s=0;                                                      /* 总分 s,初值为 0 */
for(j=0;j<N;j++)                                          /* 输入 N 门成绩,需要循环 N 次 */
{
    do{
        printf("score[%d]:",j);                          /* 提示输入第几门成绩 */
        scanf("%f",&stud[i].score[j]);                    /* 输入成绩 */
        if(stud[i].score[j]>100||stud[i].score[j]<0)
            printf("非法数据,请重新输入! \n");            /* 确保成绩在 0-100 之间 */
      }while(stud[i].score[j]>100||stud[i].score[j]<0);
      s=s+stud[i].score[j];                               /* 累加各门课程成绩 */
}
stud[i].sum=s;                                            /* 将总分保存 */
stud[i].average=(float)s/N;                               /* 求出平均值 */
stud[i].order=0;                                          /* 未排名次前此值为 0 */
printf("\n 已经在位置%d 成功插入新记录! \n",position);
return n+1;
```

```
}
```

（8）删除记录文件 sms_Delete.c

```
/* sms_Delete.c */
#include "sms_struct.h"
int Delete(STUDENT *stud,int n)                                    /* 删除记录模块 */
{
    int i=0;
    char k[5];                                          /* 定义字符串数组,用来确认删除信息 */
    char s[15];                                                    /* 存放学号 */
    printf("请输入要删除学生的姓名:\n");
    scanf("%s",s);                                          /* 输入要删除记录的姓名 */
    while(i<n && strcmp(stud[i].name,s))
    {
        i++;
    }
    if(i==n)
        printf("\n 您要删除的是%s,很遗憾,查无此人! \n",s);
    else                                                       /* 显示找到的记录信息 */
    {
      printf("* * * * * * * * * * Found * * * * * * * * * * * * \n");
      printf(" 学号  姓名  成绩 1  成绩 2  成绩 3  总分  平均分  名次\n");
      printf("- - - - - - - - - - - - - - - - - - - - - - - - - - - - \n");
      printf ("%-11s%-15s%5.2f%7.2f%7.2f %9.2f %6.2f %3d \n", stud[i].no,stud[i].
            name,stud[i].score[0],stud[i].score[1],stud[i].score[2],stud[i].
            sum,stud[i].average,stud[i].order);
      printf("* * * * * * * * * * * * * * * * * * * * * * * * \n");
      do{
          printf("您确实要删除此记录吗? (y/n):");
          scanf("%s",k);
      }while(k[0]!='y'&&k[0]!='n');
    if(k[0]!='n')                                              /* 删除确认判断 */
    {
        for(;i<n;i++)
        {
            strcpy(stud[i].no,stud[i+1].no);
            strcpy(stud[i].name,stud[i+1].name);
            stud[i].score[0]=stud[i+1].score[0];
            stud[i].score[1]=stud[i+1].score[1];
            stud[i].score[2]=stud[i+1].score[2];
            stud[i].sum=stud[i+1].sum;
            stud[i].average=stud[i+1].average;
            stud[i].order=stud[i+1].order- 1;
        }
```

```
        printf("\n 已经成功删除姓名为 %s 的学生的记录！\n",s);
      }
    }
    return n-1;
}
```

（9）记录存储与导入文件 sms_fileRW.c

```
/* sms_fileRW.c */
#include "sms_struct.h"
void fileWrite(STUDENT *stud,int n)                    /* 保存数据到文件的函数模块 */
{
    FILE *fp;                                          /* 定义指向文件的指针 */
    int i=0;
    char outfile[20];
    printf("请输入导出文件名,例如:G:\\f1\\score.txt:\n");
    scanf("%s",outfile);
    if((fp=fopen(outfile,"wb"))==NULL)   /* 为输出打开一个二进制文件,如没有则建立 */
    {
      printf("Can not open file\n");
      exit(1);
    }
    while(i<n)
    {
      fwrite(&stud[i],sizeof(STUDENT),1,fp);           /* 写入一条记录 */
      i++;
    }
    fclose(fp);                                        /* 关闭文件 */
    printf("- - - 所有记录已经成功保存至文件%s 中！- - - \n",outfile);
}
int fileRead(STUDENT *stud)                            /* 从文件导入记录的函数模块 */
{
    int i;
    FILE *fp;                                          /* 定义指向文件的指针 */
    char infile[20];
    printf("请输入导入文件名,例如:G:\\f1\\score.txt:\n");
    scanf("%s",infile);                                /* 输入文件名 */
    if((fp=fopen(infile,"rb"))==NULL)                  /* 打开一个二进制文件,为读方式 */
    {
        printf("文件打开失败！\n");
        return 0;
    }
    i=0;
    while(!feof(fp))                                   /* 循环读数据直到文件尾结束 */
    {
```

```
        if(1!=fread(&stud[i],sizeof(STUDENT),1,fp))
            break;                                    /* 如果没读到数据,跳出循环 */
        i++;
    }
    fclose(fp);
    printf("已经成功从文件%s 导入数据!!! \n",infile);
    return i;
}
```

习　　题

一、常用算法设计题

1. 迷宫问题。在指定的迷宫中找出从入口到出口的所有可通路径。

2. 砝码问题。一位商人有 4 块砝码,各砝码重量不同且都是整磅数,而且用这 4 块砝码可以在天平上称 1～40 磅之间的任意重量(砝码可以放在天平的任一端),请问这 4 块砝码各重多少?

3. 抓贼问题。警察审问四名偷窃犯罪嫌疑人。已知,这四人当中仅有一名是窃贼,还知道这四个人中每人要么是诚实的,要么总是说谎。他们给警察的回答是:

甲说:“乙没有偷,是丁偷的。”

乙说:“我没有偷,是丙偷的。”

丙说:“甲没有偷,是乙偷的。”

丁说:“我没有偷。”

请根据这四个人的回答判断谁是窃贼。

4. 子串定位问题。子串定位运算的功能是返回子串 t 在主串 s 中首次出现的位置,如果 s 中未出现 t,则返回－1。函数名为 index(s,t)。例如,主串为“abcdefbc”,子串为“bc”,则子串在主串中首次出现的位置为 2。

5. 约瑟夫问题。设有 n 个人围坐在一个圆桌周围(从 1 到 n 依次编号),现从第 s 个人开始报数,数到第 m 的人出列,然后从出列的下一个人重新开始报数,数到第 m 的人又出列…如此重复直到所有的人全部出列为止。对于任意给定的 n,s 和 m,求出这 n 个人员的出列次序。设 n＝8,s＝1,m＝4,则其出列顺序为:4→8→5→2→1→3→7→6。

6. n 皇后问题。在 n×n 的方阵棋盘上,试放 n 个皇后,每放一个皇后,必须满足该皇后与其他皇后互不攻击(即不在同一行、同一列、同一对角线上),求出所有可能解。

7. 背包问题。有一个背包,能装入的物品总重量为 S,设有 N 件物品,其重量分别为 W1,W2,…,WN。希望从 N 件物品中选择若干件物品,所选物品的重量之和恰能放入该背包,即所选物品的重量之和等于 S。试编程求解。

8. 过桥问题。有 N(N≥2)个人在晚上需要从 X 地到达 Y 地,中间要过一座桥,过桥需要手电筒(而他们只有 1 个手电筒),每次最多两个人一起过桥(否则桥会垮)。N 个人的过桥时间按从小到大的顺序依次存入数组 t[N]中,分别为:t[0], t[1], …, t[N－1]。过桥的速度以慢的人为准! 注意:手电筒不能丢过桥! 问题是:编程求这 N 个人过桥所花的最短时间。

二、模块化程序设计题

1. 用链表结构存储记录，编写一个小型学生成绩管理系统，可参照例 9-15 进行模块化程序设计。

2. 编写一个重要数据管理系统，具体要求如下。

① 现在每个人在不同网站都有用户名和密码等信息，还有银行卡卡号及密码信息，众多的信息经常忘记，因此我们可以编写一个重要数据管理系统，将自己需要保护的数据加密存储在指定的文件中。

② 程序执行时，首先要进行密码检测，以不让非法用户使用本程序。标准密码预先在程序中设定，也可预先加密存储在专门的文件中。程序运行时，若用户的输入密码和标准密码相同，则显示“口令正确!”并转去执行后续程序；若不相等，重新输入，3 次都不相等则显示“您是非法用户!”并终止程序的执行。

③ 管理系统的日常管理功能可参照例 9-15 进行模块化程序设计。需要保护的数据包括编号，账号位置，账号描述，账号名及密码等信息。

④ 对重要数据进行日常管理(包括查询、添加、删除、修改等)时是要求明文显示的，而存入文件是要求加密存储的。