程序设计基础
(C Programming)

第四讲：程序设计方法-模块化与算法设计

北航计算学院 晏海华

本章目标

- 进一步掌握模块化设计思想
- 掌握常用的数据查找及排序方法
- 掌握全局变量
- 了解递归程序设计思想

问题4.1

【问题描述】

从文件中查找包含给定字符串的行。

【输入形式】

从标准输入中分两行分别输入被查找的文件及要查找的字符串（中间不含空格）。

【输出形式】

在屏幕上输出文件中包含给定字符串的行。

【样例输入】

在键盘输入如下文件名及字符串:

test.txt

the

文件test.txt内容如下：

Now is the time

for all good

men to come to the aid

of their party

【样例输出】

屏幕输出为：

this is the time

men to come to the aid

of their party

问题4.1：问题分析

- **数据结构设计：**分析问题描述，显然需要三个字符数组变量，分别存放文件名、要查找的字符串及从文件中读入的行。

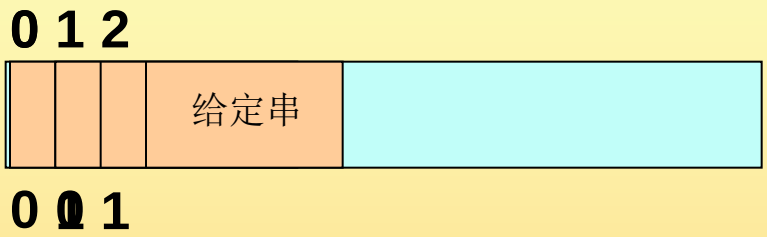
  char filename[32], str[81], line[1024];

  （一个文件名长度通常不超过32个字符；屏幕上一行通常显示80个字符；而1024是一般文件的最大物理行长度。当然这些取决于具体系统实现。）

- **数据输入**
  - 用scanf(“%s...)读入文件名和要查找的串。（中间不能有空格）
  - 从文件中读入一行是简单的方法是用fgets(...)函数。（为何不能用fscanf(fp,“%s...)
- **数据处理：**主要处理就是要从所读入的一行中查找给定的字符串（即从一个字符串中查找另一个字符串）。可用一个单独的函数index实现在一个字符串中查找另一个字符串。（体现模块化思想）

问题4.1：算法设计

- 设int index(char s[ ], char t[ ])函数用来在字符串s中查找字符串t。若找到则返回t在s中出现的位置，否则返回-1。其主要查找算法如下：




```
在字符串s中查找字符串t ：
for(i=0; s[i] != ‘\0’ ; i++)
    for(j=i,k=0; t[k] != ‘\0; j++,k++)
        s[j]和t[k]进行比较
```

遍历输入字符串中每个字符

依次与给定串中每个字符比较。j为s中每次开始比较的位置。

主要算法分析

问题4.1：算法设计（续）

主函数算法如下：

设变量filename,s,line分别用于存储文件名、查找串及文件中一行；

从标准输入中读入文件名和要查找的串到filename和s中；

以读方式打开文件filename；

```
while 文件中还有内容时读一行到line中
    if index(line, s) >= 0
        输出line;
```

如何从文件中读入一行？

```
char  *fgets(char s[ ], int n, FILE  *fp)
```

从fp上读入一行（不超过n-1个字符），放入s 字符数组中。返回s或NULL

行输入/输出*

char *fgets(char s[ ], int n, FILE *fp)

从fp上最多读入n-1个字符，放入s 字符数组中。返回s或NULL。

int fputs(char s[ ], FILE *fp)

把字符串s（不一定含\n）写入文件fp中。返回非负数或EOF。

- fgets在s最后加换行字符（与gets不同）；
- fputs不在输出后加换行字符（与puts不同）；
- fgets能设置字符的最大个数，因此，相比函数gets，它更安全。

问题4.1：代码

使用scanf的缺点是不能输入带空格的字符串。可换成
gets(s);
来实现查找带空格的字符串（即查找一个句子）。

```
#include <stdio.h>
#define MAXLINE  1000
int index(char s[ ], char t[ ]);
int main( )
{
      char filename[64], s[81], line[MAXLINE];
      FILE *fp;
      scanf("%s", filename);
      scanf("%s", s);
      if((fp = fopen(filename, "r")) == NULL){
           printf("Can't open file %s!\n", filename);
           return 1;
      }
      while(fgets(line, MAXLINE-1, fp) != NULL)
          if(index(line, s) >= 0)
              printf("%s", line);
      return 0;
}
```

```
int index(char s[ ], char t[ ])
{
        int i, j, k;
        for(i =0; s[i] != '\0'; i++){
                for(j=i,k=0;t[k]!='\0'&&s[j]==t[k]; j++,k++)
                        ;
                if(t[k] == '\0')
                        return ( i);
        }
        return ( -1);
}
```

由于打开一个文件操作可能失败(如打开一个读文件不存在)，因此，好的风格应判断fopen函数的返回值，进行错误处理。

注意：前面循环结束时有两种情况：
1. 找到相应子串，即t[k]=='\0'
2. 没有找到，即s[j] != t[k]

因此要依据t[k]=='\0'来判断查找是否成功。

读到文件结束时fgets返回NULL。

问题4.1：测试

当要查找的文件为”test.txt”,查找的串为”the”，
且文件test.txt中内容为：

Now is the time

for all good

men to come to the aid

of their party

则屏幕输出：

this is *the* time
men to come to *the* aid
of *the*ir party

问题4.1：测试（续）

- **其它考虑点：**
  - 要查找的串在一行的头、尾
  - 要查找的串在文件中不存在

问题4.1：思考1

- 问题4.1实现了大小写相关的字符串查找，即字符串”the”和”The”是不同字符串。请实现大小写无关的字符串查找。
- 算法分析：

在比较字符时，可
写既可实现大小写

设函数char tolow
为相应小写字符，

```
int index(char s[ ], char t[ ])
{
        int i, j, k;
        for(i =0; s[i] != '\0'; i++){
                for(j=i,k=0;t[k]!='\0'&&tolower(s[j])== tolower(t[k]);
                        j++,k++)
                        ;
                if(t[k] == '\0')
                        return ( i);
        }
        return ( -1);
}
```

问题4.1：函数tolower实现

- 方法一：

```
char  tolower(char c)
{
    if( c >='A' && c<='Z')
        return 'a' – 'A' + c;
    return c;
}
```

- 方法二：对于象tolower这样功能简单的函数，可以用宏函数来实现。

```
#define tolower(c)    (c>='A'&&c<='Z' ? 'a'-'A'+c:c)
```

预处理程序：define

- 定义函数

宏定义还可带变元（参数）：

#define 标识符(参数1, 参数2,...)　　单词串

如：

#define　max(A,B)　((A)>(B)?(A):(B))

?:为条件运算符

于是语句x = max(p+q, r+s); 被替换为：

x = ((p+q) > (r+s) ? (p+q) : (r+s));

#define isupper(c)　(c >='A' && c<='Z')?1:0

注意：

a. 宏定义名与参数间不能有空格，如max(A,B);
b. 参数应用括号括起来，如(A)>(B)?(A) : (B)

常用标准库函数：字符类型判断和转换*

- #include <ctype.h>

int isalpha(int c)	是否是字母
int isdigit(int c)	是否是数字
int islower(int c)	是否是小写字母
int isupper(int c)	是否是大写字母
int isspace(int c)	是否是空白字符
int tolower(int c)	将大写字母为小写字母
int toupper(int c)	将小写字母为大写字母
…	

它们都是用宏函数实现的。

问题4.1：思考1（代

```
#include <stdio.h>
#define MAXLINE  1000
#define tolower(c)  (c>='A'&&c<='Z' ? 'a'-'A'+c
int index(char s[ ], char t[ ]);
int main( )
{
    char filename[64], s[81], line[MAXLINE];
    FILE *fp;
      scanf("%s", filename);
      scanf("%s", s);
      if((fp = fopen(filename, "r")) == NULL){
                printf("Can't open file %s!\n", filename);
                return 1;
      }
      while(fgets(line, 81, fp) != NULL)
       if(index(line, s) >= 0)
           printf("%s", line);
    return 0;
}
```

```
int index(char s[ ], char t[ ])
{
        int i, j, k;
        for(i =0; s[i] != '\0'; i++){
                for(j=i,k=0;t[k]!='\0'&&tolower(s[j])== tolower(t[k]);
                        j++,k++)
                          ;
                if(t[k] == '\0')
                          return ( i);
        }
        return ( -1);
}
```

问题4.2

【问题描述】

某班有不超过200名的学生，从文件中输入某班学生成绩，对输入成绩按由高到低进行排序，并输出到另一个文件中。

【输入形式】

从文件scorelist.in中读入学生成绩，学生成绩以整数形式按行存放。注意，学生成绩数目不确定。

【输出形式】

将排序结果按行写到文件sorelist.out中。

【样例输入】

若文件scorelist.in中有如下成绩:

58
75
62
86
98

【样例输出】

程序运行结束后文件scorelist.out中内容为：

98
86
75
62
58

问题4.2：问题分析

- **数据结构分析**

```
int  scorelist[200];  /* 题目要求最多只有200名学生*/
```

- **数据输入：**如何从文件中读入数目不确定的数据（本题要求最多不超过200）？即如何判断输入结束或已读到文件尾？
  - 方法一：

```
while(!feof(in))    /*函数feof用来测试是否已读写到文件尾,若到文件尾,则返回1,否则返回0。*/
   fscanf(in, "%d ", &scorelist[n++]);
```

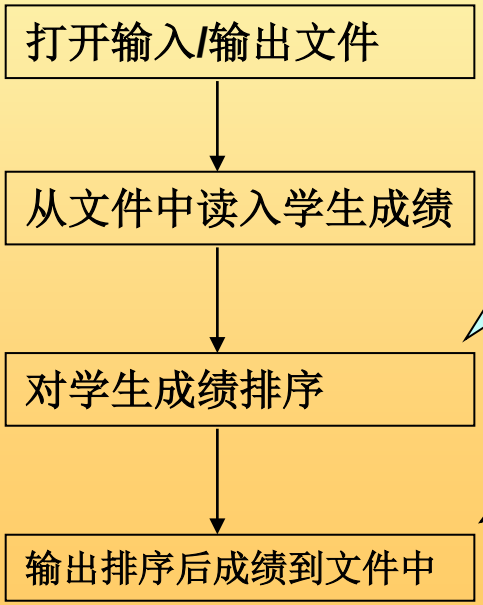
  - 方法二：

```
while(fscanf(in,"%d",&scorelist[n] )>0)   /* 函数fscanf返回成功读入数据的个数*/
   n++;
```

- **数据处理：**即对输入数据进行排序。单独设一个函数对整数集进行排序（模块化）。

问题4.2：算法分析

- 问题可分解为如下几个步骤：



算法：

```
FILE *in, *out;
in = fopen("scorelist.in", "r");
out = fopen("scorelist.out","w");
```

```
anf(in, "%d", &scorelist[n++]);
```

函数feof用来测试是否已读写到文件尾，若到文件尾,则返回1,否则返回0。
函数fscanf用来从文件中读数据。与标准

算法：

设一个函数专门用来对学生成绩进行排序，函数原型为：

```
void sortScore(int list[], int len )
```

算法：

```
for(i=0; i<n; i++)
    fprintf(out,"%d ",scorelist[i]);
```

函数fprintf用来输出数据到文件中。与标准输出printf不同的是第一个参数为文件指针。

问题 4.2：算法分析- 选择排序（续）

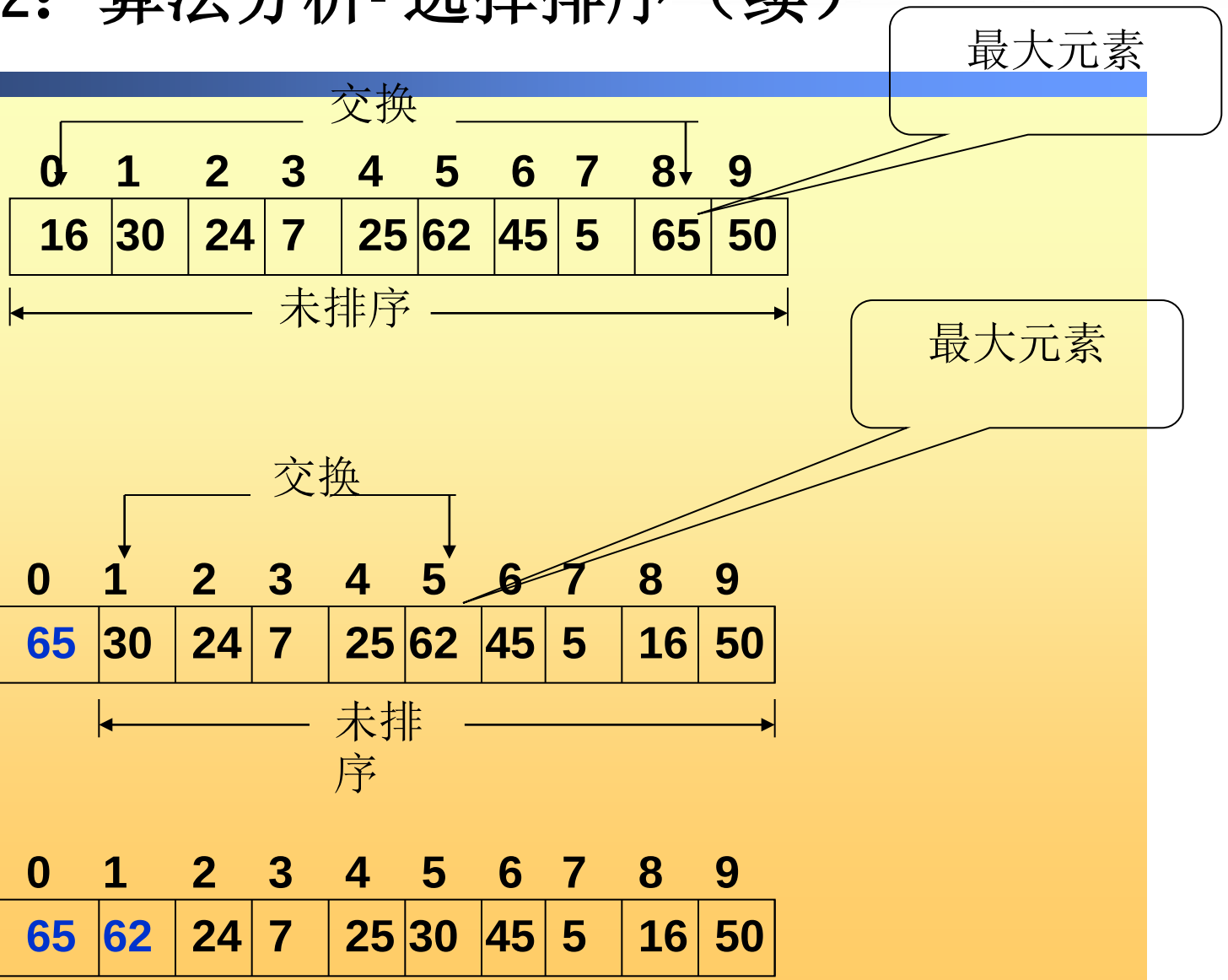
- 有许多经典的算法用来对数据进行排序，如选择排序（selection sort）、插入排序（insertion sort）和快速排序（quick sort）等。有关排序算法及分析主要在《数据结构》课程中讲授。
- 在问题4.2中将使用常用的**选择排序**算法对学生成绩进行排序。

问题 4.2：算法分析- 选择排序（续）

- 选择排序的核心思想是：
  - 找到数组中未排序部分的最大元素；
  - 然后将其移到未排序部分的最前端（从大至小排序）；
  - 重复上述步骤，直到所有元素都排好序。

  即首先在整个数组中查找最大元素，将其换到第一个位置；然后从数组中第二个元素开始查找最大元素，以此类推。下面以图示来说明：

问题 4.2：算法分析- 选择排序（续）



问题 4.2：算法分析- 选择排序（续）

- **选择排序包括以下步骤：**
  1. 找到最大元素
  2. 将最大元素移到未排序部分的第一个位置上

```
index = 0;
for(i=0; i<N; i++)
    if(array[index] < array[i])
        index = i;
```

通过交换两个元素即可。如：

```
tmp = array[i];
array[i] = array[index];
array[index] = tmp;
```

问题 4.2：代码实现（排序函数）

```
void sortArray(int array[], int n)
{
    int i,j,tmp, index;
    for(i=0; i<n; i++) {
        index = i;
        for (j=i; j<n; j++)
            if(array[index] < array[j])
                index = j;
        tmp = array[i];
        array[i] = array[index];
        array[index] = tmp;

    }
}
```

找最大元素

将最大元素移到未排序部分的头部

问题 4.2：代码实现（主函数）

```
#include <stdio.h>
#define NUM 200
void sortArray(int array[], int n);
int main()
{
    int scorelist[NUM], i, n=0;
    FILE *in, *out;
    if((in = fopen("scorelist.in","r")) == NULL){
                printf("Cann't open file scorelist.in!\n");
                return 1;
    }
    if((out = fopen("scorelist.out","w")) == NULL){
                printf("Cann't open file scorelist.out!\n");
                return 1;
    }
    while(!feof(in))
                fscanf(in,"%d",&scorelist[n++] );
    sortArray(scorelist, n);
   for(i=0; i<n; i++)
                fprintf(out, "%d\n",scorelist[i]);
   fclose(in); fclose(out);
   return 0;
}
```

一种模块化更好的实现:

```
#include <stdio.h>
#define NUM 200
int readList(int array[ ]);
void sortArray(int array[ ], int n);
void writeList(int array[ ], int n);
int main()
{
   int scorelist[NUM], n;
   n = readList(scorelist);
   sortArray(scorelist, n);
   writeList(scorelist, n);
   return 0;
}
int readList(int array[ ])
{
   FILE *in;
   int n=0;
   if((in = fopen("scorelist.in","r")) == NULL){
      printf("Cann't open file scorelist.in!\n");
      exit(1);
    }
   while(!feof(in))
      fscanf(in,"%d",&array[n++] );
   fclose(in);
   return n;
}
void writeList(int array[ ], int n)
{
   FILE *out;
   int i;
   if((out = fopen("scorelist.out","w")) == NULL){
      printf("Cann't open file scorelist.out!\n");
      exit(1);
    }
    for(i=0; i<n; i++)
       fprintf(out, "%d\n",array[i]);
    fclose(out);
}
```

问题 4.2：测试及常见问题

若文件**scorelist.in**内容为：

58
75
62
86
98

程序运行后文件**scorelist.out**内容为：

98
86
75
62
58

- 若文件scorelist.in文件尾有一个回车，则会发生什么现象？如何调试？

问题 4.2：修改主函数

```
#include <stdio.h>
#define NUM 200
void sortArray(int array[], int n);
int main()
{
    int scorelist[NUM], i, n=0;
    FILE *in, *out;
    if((in = fopen("scorelist.in","r")) == NULL){
                printf("Cann't open file scorelist.in!\n");
                return 1;
    }
    if((out = fopen("scorelist.out","w")) == NULL){
                printf("Cann't open file scorelist.out!\n");
                return 1;
    }
    while(fscanf(in,"%d",&scorelist[n] )>0)
                n++;
    sortArray(scorelist, n);
   for(i=0; i<n; i++)
                fprintf(out, "%d\n",scorelist[i]);
   return 0;
}
```

或

```
#include <stdio.h>
#define NUM 200
void sortArray(int array[], int n);
int main()
{
    int scorelist[NUM], i, n=0;
    FILE *in, *out;
    if((in = fopen("scorelist.in","r")) == NULL){
                printf("Cann't open file scorelist.in!\n");
                return 1;
    }
    if((out = fopen("scorelist.out","w")) == NULL){
                printf("Cann't open file scorelist.out!\n");
                return 1;
    }
    while(!feof(in))
                fscanf(in,"%d  " ,&scorelist[n++] );
    sortArray(scorelist, n);
   for(i=0; i<n; i++)
                fprintf(out, "%d\n",scorelist[i]);
   fclose(in); fclose(out);
   return 0;
}
```

问题 4.2：另一个常用排序方法

```
void sortArray(int array[], int n)
{
    int i, j, tmp;
    for(i=0; i<n; i++)
        for(j=i; j<n; j++){
            if(array[i] < array[j]){
                tmp = array[i];
                array[i] = array[j];
                array[j] = tmp;
            }
        }
}
```

从本质上看它还是一种选择排序

有关排序

- 从问题4.2可知如何用一种排序方法（如选择排序）对数值类（字符、整数、浮点数）数据集进行排序。
- 如何对字符串数据集排序（如英文单词、人名等）？
  - 如何存储字符串数据？指针数组、二维字符数组等
  - 如何比较字符串大小？字符串比较函数strcmp()

外部变量

- **外部变量（global variable ）：在函数外面定义的变量。**
  - 作用域（scope）为整个程序，即可在程序的所有函数中使用。
  - 外部变量有隐含初值0。
  - 生存期（life cycle）：外部变量（存储空间）在程序执行过程中始终存在（静态存储分配）。

外部变量说明（extern）

- C程序可以分别放在几个文件上，每个文件可作为一个编译单位分别编译。外部变量只需在某个文件上定义一次，其它文件若要引用此变量时，应用extern加以说明。（外部变量定义时不必加extern关键字。
- 在同一文件中，若 的外部（在函数之外 以extern说明。

```
/* t1.c */
int  N;
main()
{
      …
      N =  …
      …
}
```

```
/* t2.c */
exte
fun
{

}
```

```
extern int  N;
main()
{
      …
      N =  …
      …
}
int N=0;
void fun()
{ …
```

外部变量说明（extern）（续）

- 例如，对问题4.2的代码实现中，如果外部变量N不在程序头部定义，则需要用extern加以说明。

```
…
extern int N;
int main()
{
…
}
int N = 0;
void insertData(int array[], int data)
{
…
}
```

外部变量说明

外部变量定义

递归（Recursion）

- 通过调用自身解决问题的过程称为递归。递归是解决某些复杂问题的有效方法。如：

$$x^n = \begin{cases} 1 & n=0 \\ x * x^{n-1} & n>0 \end{cases}$$

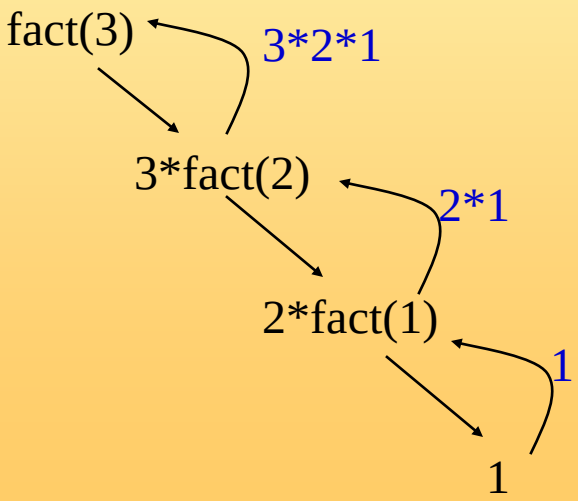
$$n! = \begin{cases} 1 & n=0 \\ n*(n-1)! & n>0 \end{cases}$$

递归（续）

例：求n!

```
#include <stdio.h>
int fact(int n);
main( )
{
        printf("3!=%d, 5!=%d\n", fact(3), fact(5));
}
int fact(int n)
{
        if( n <= 1)
             return ( 1);
        else
            return ( n * fact(n-1));
}
```

在C语言中，一个函数直接或间接调用自己称为递归。



递归（续）

- 递归算法十分简洁，编译后得到的目标代码也很短，但它并不节省（实际上还要增加）运行时所需的时间和空间，因为它必须维持一个要处理的值的栈。 此外，递归算法并不是语言必须的，不用它同样可以实现相应功能，如上例中，递归函数fact可用非递归方法 实现：

```
int fact(int n)
{
    int f ;
    for(f=1; n>0;  n--)
        f *= n;
     return ( f);
}
```

问题4.3： 一个经典递归程序示例：hanoi（汉诺塔）游戏*

如果要移动64个盘子,则移动次

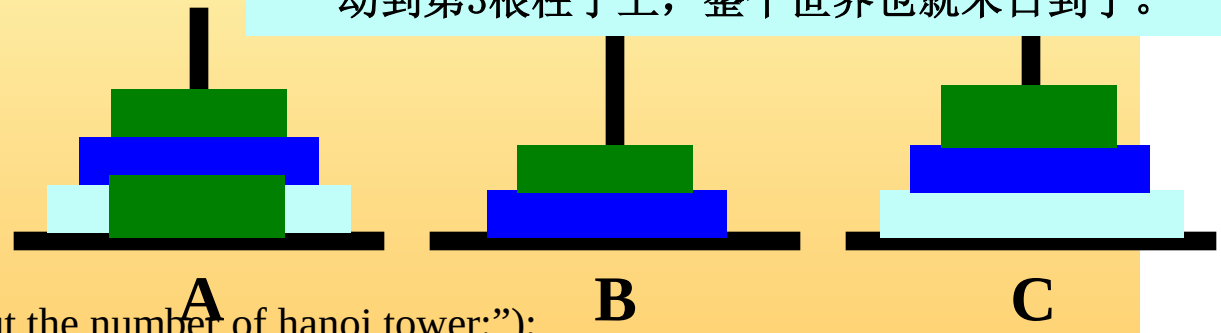
例：汉诺塔(hanoi tower)游戏

```
void hanoi( int n, char x, char y, char z
{
    if( n > 0 ) {
        hanoi(n-1, x, z, y);
        printf("MOVE %d: %c → 
        hanoi(n-1, y, x, z);
    }
}
main( )
{
    int n;
    printf("Please input the number of hanoi tower:");
    scanf("%d", &n);
    hanoi(n, 'A', 'B', 'C');
}
```

该游戏是印度Brahama寺庙僧侣们的一项工作。传说在开创世界之初， Brahama寺庙拥有3根钻石的柱子，其中一根柱子上有64个金子做的盘子。64个盘子从下至上按由大到小的顺序叠放。僧侣的工作是把这64个盘子从第1根柱子移动到第3根柱子，移动规则为：

1. 每次只能移动一个盘子；
2. 移动的盘子必须放在其中的一根柱子上；
3. 大盘子在移动过程中不能放在小盘子上。

僧侣们被告知一旦他们把所有的盘子从第1根柱子上移动到第3根柱子上，整个世界也就末日到了。



递归（续）：递归问题总结*

- **通常包含如下特性的问题适合应用递归方法解决:**
  - ✓ 问题包含一个(或多个)基本实例，如 0! = 1
  - ✓ 问题的解可以简化为包含比当前问题更简单一步的问题的解，并且最终问题解可归结到基本实例，如 n! = n*(n-1)!, 0!=1。

本讲结束