计算机程序设计基础(1)
08 模块程序设计（下）

清华大学电子工程系
杨昉
E-mail: fangyang@tsinghua.edu.cn

上节内容

- **模块化**程序设计与函数
  - 模块化程序设计的基本概念
  - 函数的定义：　类型标识符 函数名 (形参名称与类型列表){...}
  - 函数的调用：　函数**向前引用**说明
- 模块间的**参数传递**
  - **形参**与**实参**的结合方式
  - **局部变量**与**全局变量**
  - **动态存储变量**与**静态存储变量**
  - **内部函数**与**外部函数**：static, extern

课程回顾: 函数

类型	定义	说明
函数定义	类型标识符 函数名（形参名称与类型列表） {说明部分 语句部分}	函数**没有从属关系**，**不可嵌套定义**，**互相独立**，可**互相调用**
函数向前引用说明	函数类型 函数名(形参1类型 形参名1,形参2类型 形参名2,…);	也称**函数原型**。函数类型、形参类型和个数需**与原函数一致**,形参名**可省略**
形参与实参结合	数值结合：实参地址和形参地址互相独立，直接将实参值传送给形参并放在形参地址	形参的改变不影响调用程序的实参值，只能**单向传递**，通过**返回值**传递改变
局部变量与全局变量	局部：函数内部定义，函数范围内有效 全局：函数外定义，定义位置到文件结束	**作用域不同**。不同函数局部变量**可重名**,全局变量所有函数**都可访问**
动态变量与静态变量	静态变量：由系统分配固定的存储空间，程序运行期间都不释放;动态变量：根据需要动态分配。调用时分配，调用后释放	存储类别决定了该数据的**存储区域**、**作用域**、**生存期**。默认自动类型，static/extern说明静态/外部变量
内部函数与外部函数	static 类型标识符函数名(形参表) [extern] 类型标识符函数名(形参表)	内部函数只能被**本文件中其他函数**调用;外部函数能被**其他文件中函数**调用

本节目录

8.1 函数（模块）的递归调用

- 直接递归
- 间接递归
- 例题

8.2 编译预处理

- 文件包含命令
- 条件编译命令
- #pragma
- #line

函数（模块）的递归调用

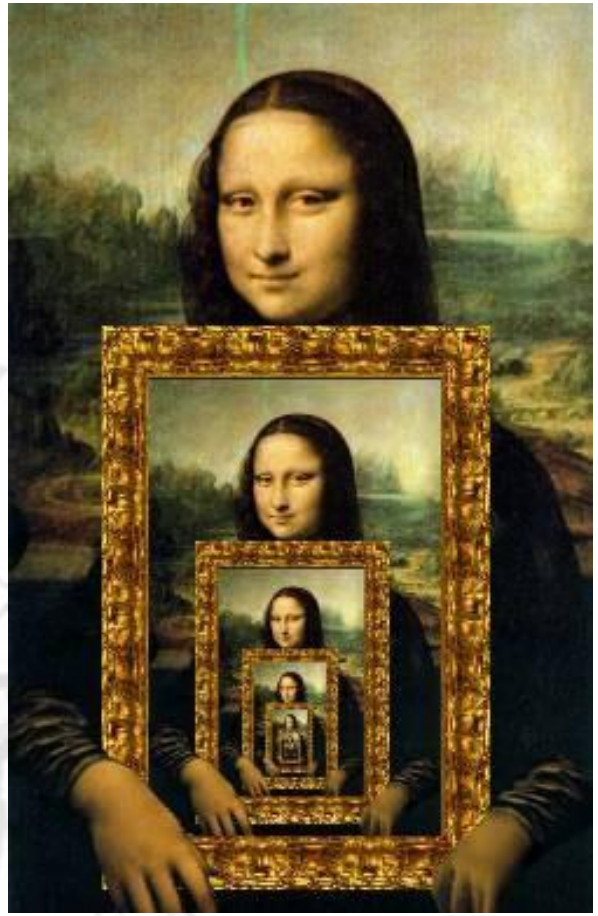
8.1 函数（模块）的递归调用

- 直接递归
- 间接递归
- 例题

函数（模块）的递归调用

●递归的基本思想

- 为降低问题的复杂程度(如问题的规模等)，可以将问题**逐层分解**，最后归结为一些最简单的问题
- 这种逐层分解问题的过程，实际上并没有马上对问题进行求解，而是当解决了最后那些**最简单**的问题后，再沿着原来分解的逆过程逐步进行**综合**
- 优点：使程序**结构清晰**，**可读性**也更强



函数（模块）的递归调用

- 例8-1：对于输入的参数n，依次打印输出自然数**1到n**

  - 最直接的想法：利用**循环语句**

```
1  #include <stdio.h>
2  void wrt(int n){
3    int k;
4    for (k = 1; k <= n; k++)
5      printf("%d\n", k);
6  }
7  void main() {
8    int n = 4;
9    wrt(n);
10 }
```

```
1
2
3
4
请按任意键继续. . .
```

函数（模块）的递归调用

●例8-2：对于输入的参数n，依次打印输出自然数**1到n**

□**递归解决**

□思路：

- 要输出n，需要先输出n−1
- 要输出n−1，需要先输出n−2
- …
- 要输出2，需要先输出1

□假设函数wrt1(n)功能为打印输出自然数1到n，则在执行wrt1(n-1)的基础上，再**打印n**即完成题目要求

□执行到wrt1(1)时，打印1

函数wrt1( )将在函数体内调用wrt1( )

函数（模块）的递归调用

●例8-2：对于输入的参数n，依次打印输出自然数**1到n**

□**递归解决**

□思路：

- 要输出n，需要先输出n-1
- 要输出n-1，需要先输出n-2
- …
- 要输出2，需要先输出1

```
1  #include <stdio.h>
2  void wrt1(int n){
3    if (n > 1)
4      wrt1(n - 1);
5    printf("%d\n", n);
6  }
7  void main() {
8    int n = 4;
9    wrt1(n);
10 }
```

```
1
2
3
4
请按任意键继续. . .
```

函数（模块）的递归调用

- 例8-2：**递归解决**分析
  - 在执行函数wrt1( )时，先判断形参n的值是否大于1，如果是，则将**形参值减1**(即n-1)后作为新的**实参**再次调用函数wrt1( )
  - 这个过程一直进行下去，直到函数wrt1( )的**形参值等于1**，不再执行if后的wrt1( )调用，而打印出1
  - 由于在先前各层的函数调用中，函数wrt1( )实际上没有执行完，即各层中的printf函数尚未执行，这就需要**逐层返回**，以便打印输出各层中的输入参数2，3，…，n

函数（模块）的递归调用

●例8-2：**递归解决**说明

- 在递归函数执行过程中，需要用**栈**操作来记忆各层调用中的**参数**及其**返回地址**
- 在逐层返回时从上次暂停的位置**恢复这些参数**继续进行执行
- 在函数wrt1( )开始执行后，随着各次的递归调用，逐次入栈记忆各层调用中的**输入参数** n，n-1，n-2，…，2，1及**返回地址**
- 在逐层返回时，又依次**出栈**（后进先出，按入栈的相反次序）将这些参数输出

	调用wrt1次数
1	
wrt1返回地址	n-1
2	
wrt1返回地址	n-2
…	
n-3	
wrt1返回地址	3
n-2	
wrt1返回地址	2
n-1	
wrt1返回地址	1

函数（模块）的递归调用

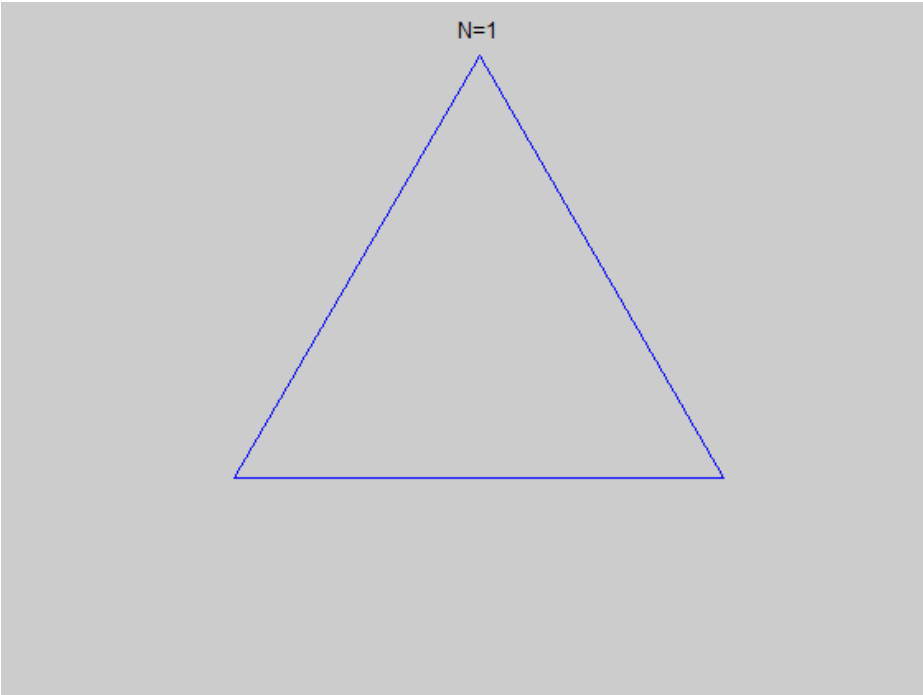

- 例8-3：将wrt1中的**printf语句**与**递归语句**调换顺序
  - 输出结果逆序
  - 因此在编写递归程序时一定要注意这个递归与操作的**时序问题**

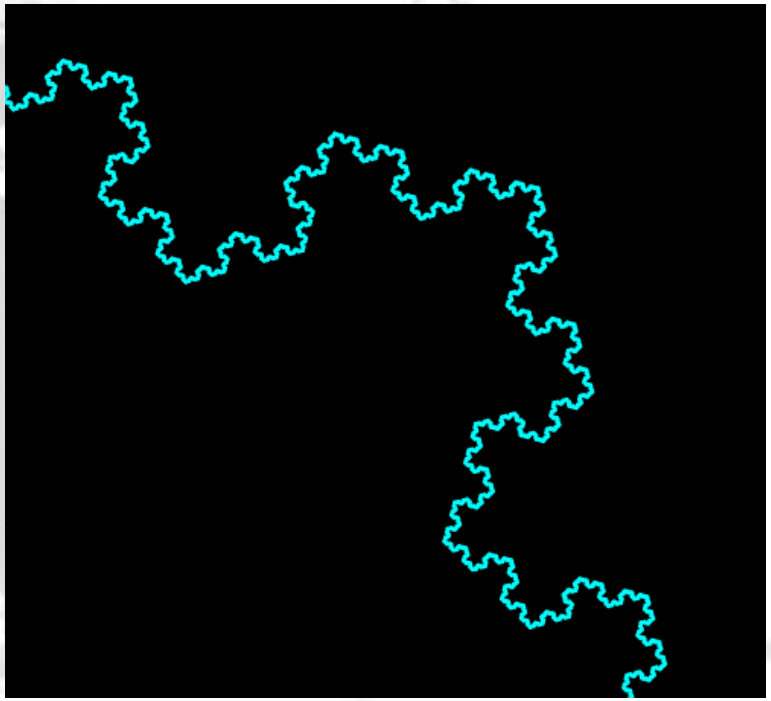
```
1  #include <stdio.h>
2  void wrt1(int n){
3    printf("%d\n", n);
4    if (n > 1)
5      wrt1(n - 1);
6  }
7  void main() {
8    int n = 4;
9    wrt1(n);
10 }
```

```
4
3
2
1
请按任意键继续. . .
```

函数（模块）的递归调用

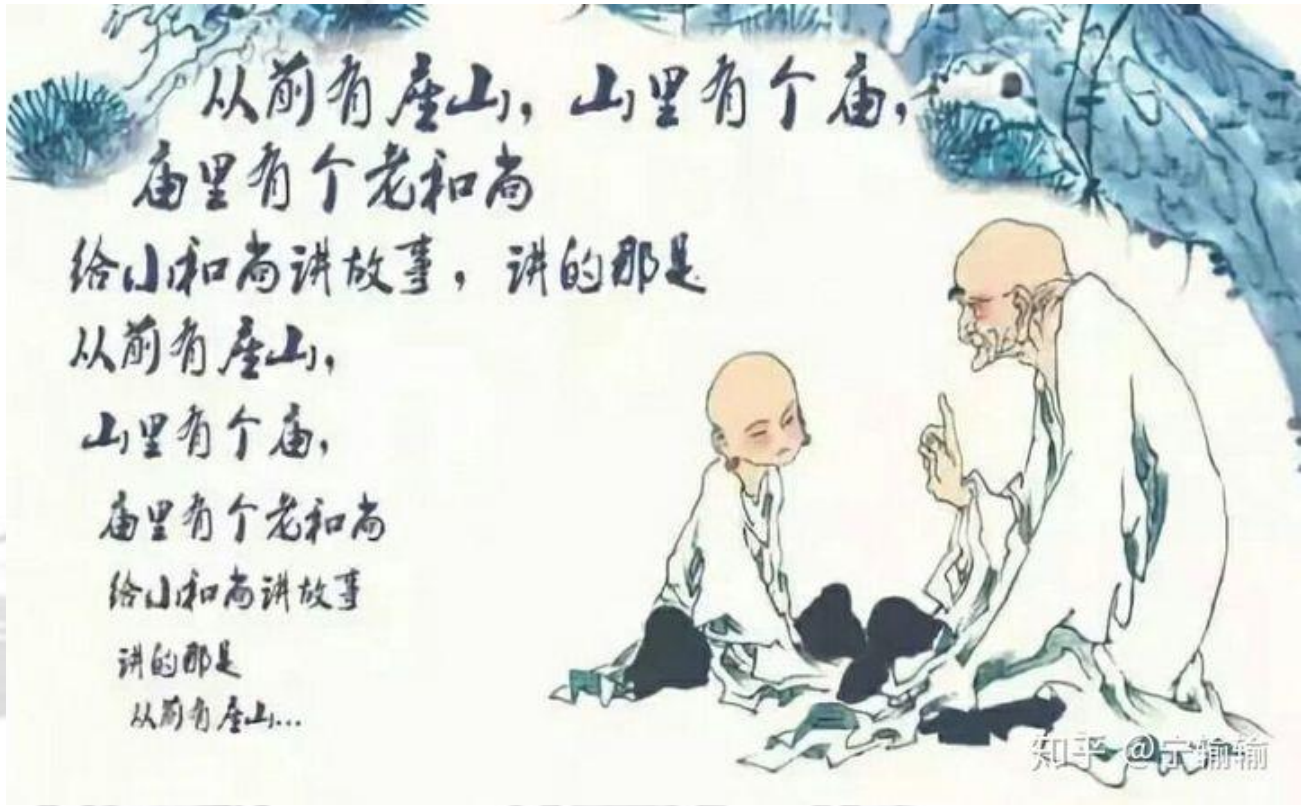
- 科赫雪花
  - 以三角形的每一边中间1/3的部分为底，产生**新的三角形**
  - 对新产生的三角形重复上一步操作





函数（模块）的递归调用

- 递归函数
  - 自己调用自己的函数
  - **直接递归**（自递归函数）
    - 直接调用函数本身
    - 例8-2中的wrt1函数



函数（模块）的递归调用

- 铺砖
  - 想要铺好第1块砖，先铺好第2块砖
  - 想要铺好第n-1块砖，先铺好第n块砖
  - 如果当前砖块为**最后一块**，则铺好后开始**返回**
  - 第1块砖铺好，铺砖结束

函数（模块）的递归调用

●递归函数

□间接递归

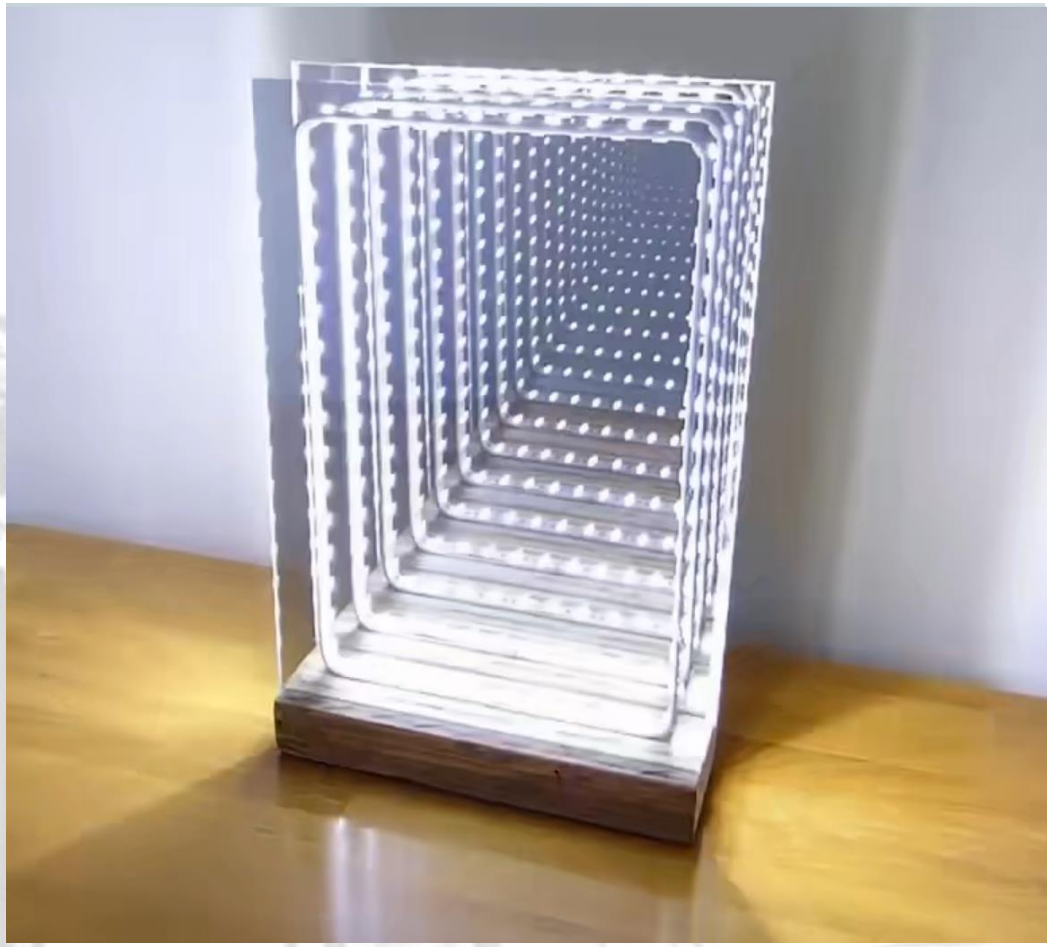
- 函数通过调用别的函数调用自身
- 右侧f1通过调用f2间接调用自身

间接递归例

```
1  int f2(int x);
2  int f1(int x) {
3    int y, z;
4    …
5    z = f2(y);
6    …
7    return(z * z);
8  }
9  int f2(int x) {
10   int a, b;
11   …
12   b = f1(a);
13   …
14   return(3 * b);
15 }
```

函数（模块）的递归调用

- 当你往镜子前面一站，镜子里面会出现你的像；但你试过两面镜子一起照吗?
- 如果A、B两面镜子相互面对面放着，你站在两面镜子中间，两面镜子里就会有你的千百个“化身”
- A镜子中有B镜子里的像，B镜子中也有A镜子里的像，这也是一种**间接递归**



程杰. 大话数据结构. 清华大学出版社, 2011.

函数（模块）的递归调用


- 编写递归程序的关键
  - 找出**递归关系**和**初始值**
  - 方法之一是利用归纳法，把一个问题归纳总结出**递归式**，加上**初始条件**，从而编写出递归函数
  - 对于单变量的递归问题f(n)，步骤是
    - 当n＝1或0时，可以得到f(n)的值；
    - 假设x小于等于n-1时，都可以得到f(x)的值；
    - 对于n，找出f(n)与f(n-1)，f(n-2)，……的关系式：

      f(n)=(f(n-1)，f(n-2)，……)

    - 开始编写递归程序

函数（模块）的递归调用


- 递归程序一般格式

```
if ( 最简单情况 ) {
        直接得到问题答案
}
else {
        将问题转化为简单一些的一个或多个子问题;
        以递归的方式逐个求解这些子问题;
        通过子问题的解组装出原问题的解;
}
```

函数（模块）的递归调用

- 例8-4：年龄问题
  - 有5个人，第5个人说他比第4个人大2岁
  - 第4个人说他比第3个人大2岁，第3个人说他比第2个人大2岁
  - 第2个人说他比第1个人大2岁，第1个人说他10岁
  - 求第5个人多少岁

$$age(n) = \begin{cases} 10, & n = 1 \\ age(n-1) + 2, & n > 1 \end{cases}$$

虽然问题很简单，递归程序怎么写?

函数（模块）的递归调用


●例8-4：年龄问题

$$age(n) = \begin{cases} 10, & n = 1 \\ age(n-1) + 2, & n > 1 \end{cases}$$

```
1  #include <stdio.h>
2  void main() {
3    int age(int);
4    printf("The age is %d\n", age(5));
5  }
6  int age(int n) {
7    int a;
8    if (n == 1) a = 10;
9    else a = age(n - 1) + 2;
10   return (a);
11 }
```

```
The age is 18
请按任意键继续. . .
```

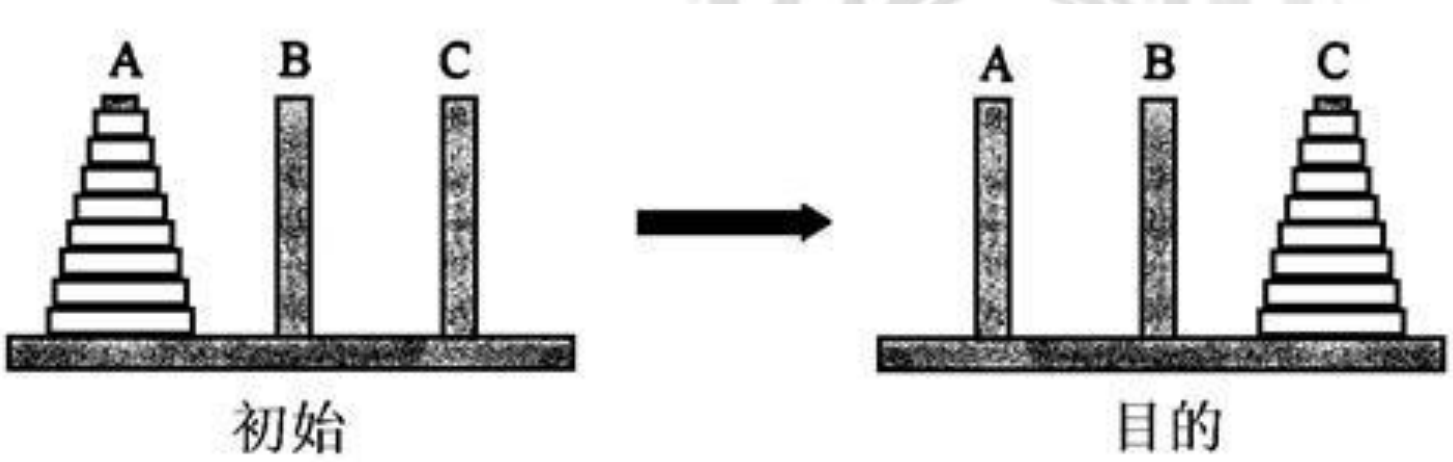
函数（模块）的递归调用

- Hanoi塔问题
  - 河内塔(Towers of Hanoi)是法国人M.Claus (Lucas)于1883年从泰国带至法国的，河内为越南的首都
  - 据说创世纪时有一座波罗教塔，是由三支钻石棒（Pag）所支撑
  - 开始时神在第一根棒上放置**64个**由上至下依由小至大排列的金盘子（Disc）
  - 神命令僧侣将所有的金盘子从第一根棒移至第三根棒，且搬运过程中遵守**每次只能移动一个**、**大盘子在小盘子之下**的原则
  - 当金盘子全数搬运完毕之时，此塔将毁损，也是**世界末日来临**之时

函数（模块）的递归调用

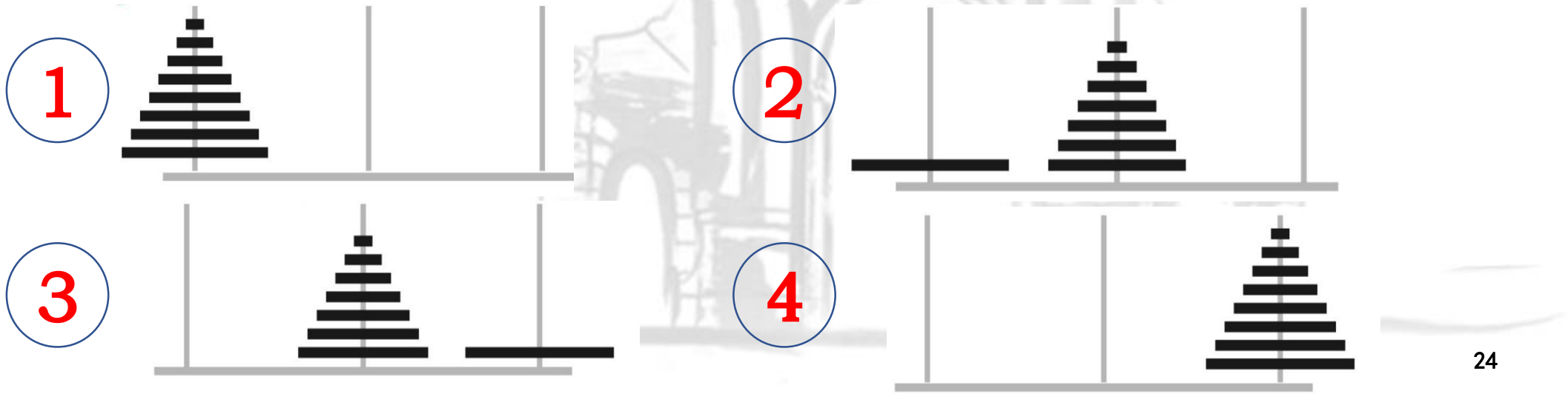
●例8-5：Hanoi塔问题

- □有n个盘子在A处，盘子从大到小，**最上面的盘子最小**，现在要把这n个盘子从A处搬到C处，**每次只能搬一个**，可以在B处暂存，但任何时候**不能**出现大的压在小的上面的情况



函数（模块）的递归调用

- (1) 若n=1，则可以把盘子直接从A处搬到C处
- (2) 假设n-1时，知道如何搬
- (3) 当n时，根据(2)的假设，可以先把前n-1个盘子从A处通过C处搬到B处，再把第n个盘子直接从A处搬到C处，最后把前n-1个盘子从B处通过A处搬到C处，则完成了全部盘子的搬动



函数（模块）的递归调用

```c
#include <stdio.h>
int Step = 1;
void move(int n, char a, char c){
  printf("Step %2d: Disk %d %c ---> %c\n", Step, n, a, c);
  Step++;
}
void Hanoi(int n, char a, char b, char c) { /* 递归程序*/
  if (n > 1){
    Hanoi(n - 1, a, c, b);   /* 先将前n-1个盘子从a通过c搬到b */
    move(n, a, c);           /* 将第n个盘子从a搬到c */
    Hanoi(n - 1, b, a, c);   /* 再将前n-1个盘子从b通过a搬到c */
  } else
    move(n, a, c);           /* 将第1个盘子从a搬到c */
}
void main(){
  int n;
  scanf("%d", &n);
  Hanoi(n, 'A', 'B', 'C');
}
```

```
4
Step  1: Disk 1 A ---> B
Step  2: Disk 2 A ---> C
Step  3: Disk 1 B ---> C
Step  4: Disk 3 A ---> B
Step  5: Disk 1 C ---> A
Step  6: Disk 2 C ---> B
Step  7: Disk 1 A ---> B
Step  8: Disk 4 A ---> C
Step  9: Disk 1 B ---> C
Step 10: Disk 2 B ---> A
Step 11: Disk 1 C ---> A
Step 12: Disk 3 B ---> C
Step 13: Disk 1 A ---> B
Step 14: Disk 2 A ---> C
Step 15: Disk 1 B ---> C
请按任意键继续. . .
```

看似复杂的问题迎刃而解!

函数（模块）的递归调用

- Hanoi塔一共需要搬运多少次?
  - 数学归纳
    - $n = 1$时，搬动$c_1 = 2^1 - 1 = 1$
    - 假定$n - 1$个盘子时，需要搬动$c_{n-1} = 2^{n-1} - 1$
    - n个盘子时，需要搬动$c_n = 2^{n-1} - 1 + 1 + 2^{n-1} - 1 = 2^n - 1$
  - $n = 64$时，$c_{64} = 2^{64} - 1 = 1.8447 \times 10^{19}$
    - 若每秒搬一次，按照每年365天计算，需要

$1.8447 \times 10^{19}/(365 * 24 * 3600) = \mathbf{5.8494 \times 10^{11}}$（年）

函数（模块）的递归调用

- 例8-6：上楼梯问题
  - 用递归方法编写函数f(n)：一共有n个台阶，某人上楼一步可以跨1个台阶，也可以跨2个台阶，问有多少种走法
- 分析：
  - 当n=1时，共1种走法；f(1)=1;
  - 当n=2时，可以一步走一个或两个台阶，共2种走法；f(2)=2;
  - 假设已经知道n-1时的走法，那么当n时，可归结为两种情况
    - 一步1个台阶，剩n-1个台阶
    - 一步2个台阶，剩n-2个台阶

递归式：f(n)=f(n-1)+f(n-2)

函数（模块）的递归调用

●例8-6：上楼梯问题

□用递归方法编写函数f(n)：一共有n个台阶，某人上楼一步可以跨1个台阶，也可以跨2个台阶，问有多少种走法

```
1  #include <stdio.h>
2  int f(int n){
3    if (n == 1)
4      return 1;
5    else if (n == 2)
6      return 2;
7    else
8      return f(n - 1) + f(n - 2);
9  }
10 void main(){
11   int n;
12   scanf("%d", &n);
13   printf("T=%d\n", f(n));
14 }
```

递归式：f(n)=f(n-1)+f(n-2)

```
3
T=3
请按任意键继续. . .
```

```
10
T=89
请按任意键继续. . .
```

```
30
T=1346269
请按任意键继续. . .
```

函数（模块）的递归调用


●例8-6：上楼梯问题讨论

□边界初始值界定错误，遗漏了n==2时候值

```
1  #include <stdio.h>
2  int f(int n){
3    if (n == 1)
4      return 1;
5    else
6      return f(n - 1) + f(n - 2);
7  }
8  void main(){
9    int n;
10   scanf("%d", &n);
11   printf("T=%d\n", f(n));
12 }
```

□当 n == 2 时，会有

f(2) = f(1) + f(0)

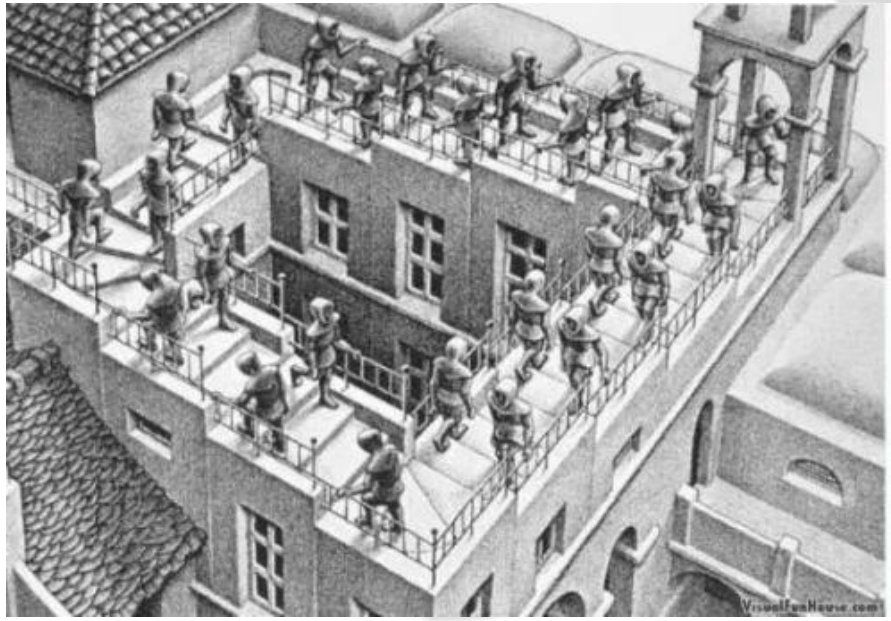
□但左边代码会继续调用

f(0) = f(-1) + f(-2)

□这会导致无限调用，进入**死循环**

函数（模块）的递归调用

- 例8-7：上楼梯问题拓展
  - 用递归方法编写函数f(n,m)：一共有n个台阶，某人上楼可能一步可以跨1个台阶，也可以跨2个台阶，最多一步跨m个台阶，问有多少种不同走法



函数（模块）的递归调用


- 例8-7：上楼梯问题拓展分析
  - 当n=1或者m=1时，共1种走法
  - 当n<m时，一步最多n个台阶，因此f(n, m)=f(n, n)
  - 当n=m时，可以一步走m个台阶，共1种走法,也可以不一步走m个台阶，有f(n, m-1)种走法；f(n, m)=f(n, m-1)+1
  - 当n>m时，可以最后一步走1个台阶，有f(n-1，m)种走法；也可以一步走2个台阶，有f(n-2 ,m)种走法……；直到一步走m个台阶，有f(n-m, m)种走法。因此递归式为

$$f(n, m)=f(n-1, m)+f(n-2, m)+\cdots\cdots+f(n-m, m)$$

函数（模块）的递归调用

●例8-7：上楼梯问题拓展

```
3 2
T=3
请按任意键继续. . .
```

```
10 2
T=89
请按任意键继续. . .
```

```
30 4
T=201061985
请按任意键继续. . .
```

递归式：

$n = m,\quad f(n,m) = f(n,m-1) + 1$

$n < m,\quad f(n,m) = f(n,n)$

$n > m,$

$f(n,m) = f(n-1,m) + f(n-2,m) + \cdots + f(n-m,m)$

```
1  #include <stdio.h>
2  int f(int n, int m){
3    if (n == 1 || m == 1) return 1;
4    else if (n<m) return f(n, n);
5    else if (n==m) return f(n,m-1)+1;
6    else{
7      int s = 0, i;
8      for (i = 1; i <= m; i++)
9        s += f(n - i, m);
10     return s;
11   }
12 }
13 void main(){
14   int m, n;
15   scanf("%d%d", &n, &m);
16   printf("T=%d\n", f(n, m));
17 }
```

函数（模块）的递归调用


●例8-8： **组合数**问题：用递归方法编写函数：从1，2，……,n的n个整数中取k个的全组合（有C_n^k种组合）的个数

- 当k=1时，共n种取法；$C_n^1 = n$;
- 当n=k时，即从k个数中取k个，共1种取法；$C_n^n = 1$;
- 假设已经知道如何从n-1个数中取k个的方法，那么当从n个中取k个时，可以归结为两种情况
  - 从前n−1个数中**取k个**
  - 先从前n−1个数中**取k−1个**，再加上第n个数，一共取k个

递归式：$C_n^k = C_{n-1}^k + C_{n-1}^{k-1}$

函数（模块）的递归调用

●例8-8：组合数问题

递归式：$C_n^k = C_{n-1}^k + C_{n-1}^{k-1}$

```
Input n and k:5 3
C(5,3)=10 called=11
请按任意键继续. . .
```

```
Input n and k:33 7
C(33,7)=4272048 called=1812383
请按任意键继续. . .
```

```
Input n and k:49 6
C(49,6)=13983816 called=3424607
请按任意键继续. . .
```

```
1  #include <stdio.h>
2  int cc = 0; /* 统计函数被调用的次数*/
3  int combine(int n, int k){
4    cc++;
5    if (n > k && k > 1)
6      return combine(n-1,k-1) + combine(n-1,k);
7    else if (k == 1)
8      return n;
9    else if (n == k)
10     return 1;
11 }
12 void main(){
13   int n, k, c;
14   printf("Input n and k:");
15   scanf("%d%d", &n, &k);
16   c = combine(n, k);
17   printf("C(%d,%d)=%d called=%d\n",n,k,c,cc);
18 }
```

函数（模块）的递归调用

- 例8-9：摆苹果问题
  - 用递归方法编写函数：把m个同样的苹果放在n个同样的盘子里，允许有的盘子空着不放，问共有多少种不同的放法?
  - 当m=7，n=3时，视5,1,1和1,5,1为同一种放法

函数（模块）的递归调用


●例8-9：摆苹果问题思路

- 设$f(m,n)$为m个苹果n个盘子的摆法数
- 当m=1或n=1时，只有1种摆法，$f(m,n)=1$
- 如果$n>m$，必定有n-m个盘子要空着，去掉它们对摆法数不产生影响；即n>m时，$f(m,n)=f(m,m)$
- 当$n \le m$时，不同的摆法可以分成两类：
  - 至少一个盘子空着，相当于$f(m,n)=f(m,n-1)$；
  - 所有盘子都有苹果，每个盘子至少要先放一个苹果，即$f(m,n)=f(m-n,n)$。在这种情况下，m = n时，$f(m,n)=1$

函数（模块）的递归调用

●例8-9：摆苹果问题

```
1  #include <stdio.h>
2  int f(int m, int n){
3    if (m == 1 || n == 1)
4      return 1;
5    else if (m == n)
6      return f(m, n - 1) + 1;
7    else if (n > m)
8      return f(m, m);
9    else
10     return f(m, n - 1) + f(m - n, n);
11 }
12 void main(){
13   int m, n, c;
14   printf("Input m and n:");
15   scanf("%d%d", &m, &n);
16   c = f(m, n);
17   printf("C=%d\n", c);
18 }
```

递归式：

$$n > m, f(m,n) = f(m,m)$$
$$n < m, f(m,n) = f(m,n-1) + f(m-n,n)$$
$$n = m, f(m,n) = f(m,n-1) + 1$$

```
Input m and n:5 3
C=5
请按任意键继续. . .
```

```
Input m and n:10 4
C=23
请按任意键继续. . .
```

```
Input m and n:50 10
C=62740
请按任意键继续. . .
```

课堂练习

练习1：下面的代码通过递归计算n阶勒让德多项式$P_n(x)$的结果，请据此计算$P_3(2)$ （**提示：写出递推式**）

$$P_n(x)=\begin{cases}1, & n=0\\ x, & n=1\\ \dfrac{(2n-1)xP_{n-1}(x)-(n-1)P_{n-2}(x)}{n}, & n\geq 1\end{cases}$$

```
1  float pn(int n,float x){
2    float p;
3    if (n==0)
4      p=1;
5    else if(n==1)
6      p=x;
7    else{
8      p=((2*n-1)*x*pn(n-1,x)-
9         (n-1)*pn(n-2,x))/n;
10   }
11   return (p);
12 }
```

课堂练习

练习1：下面的代码通过递归计算n阶勒让德多项式$P_n(x)$的结果，请据此计算$P_3(2)$ （**提示：写出递推式**）

```
1  #include<stdio.h>
2  float pn(int,float);
3  void main(){
4    int n;
5    float x,p;
6    printf("Please enter n and x:\n");
7    scanf("%d%f", &n, &x);
8    p=pn(n,x);
9    printf("P_n(x)=%f", p);
10 }
```

```
11 float pn(int n,float x){
12   float p;
13   if (n==0)
14     p=1;
15   else if(n==1)
16     p=x;
17   else{
18     p=((2*n-1)*x*pn(n-1,x)-
19        (n-1)*pn(n-2,x))/n;
20   }
21   return (p);
22 }
```

```
Please enter n and x:
3 2.0
P_n(x)=17.000000请按任意键继续
```

函数（模块）的递归调用

- 例8-10：二分查找的递归实现
  - 在有序数组中查找特定元素
  - 查找过程：

1. 从数组的**中间元素**开始搜索，如果该元素正好是目标元素，则搜索过程结束，否则执行下一步
2. 如果目标元素大于/小于中间元素，则在数组大于/小于中间元素的**一半区域**查找，然后重复步骤1的操作。
3. 如果某一步**数组为空**，则表示找不到目标元素

函数（模块）的递归调用

- 例8-10：二分查找的递归实现

```
5
请按任意键继续. . .
```

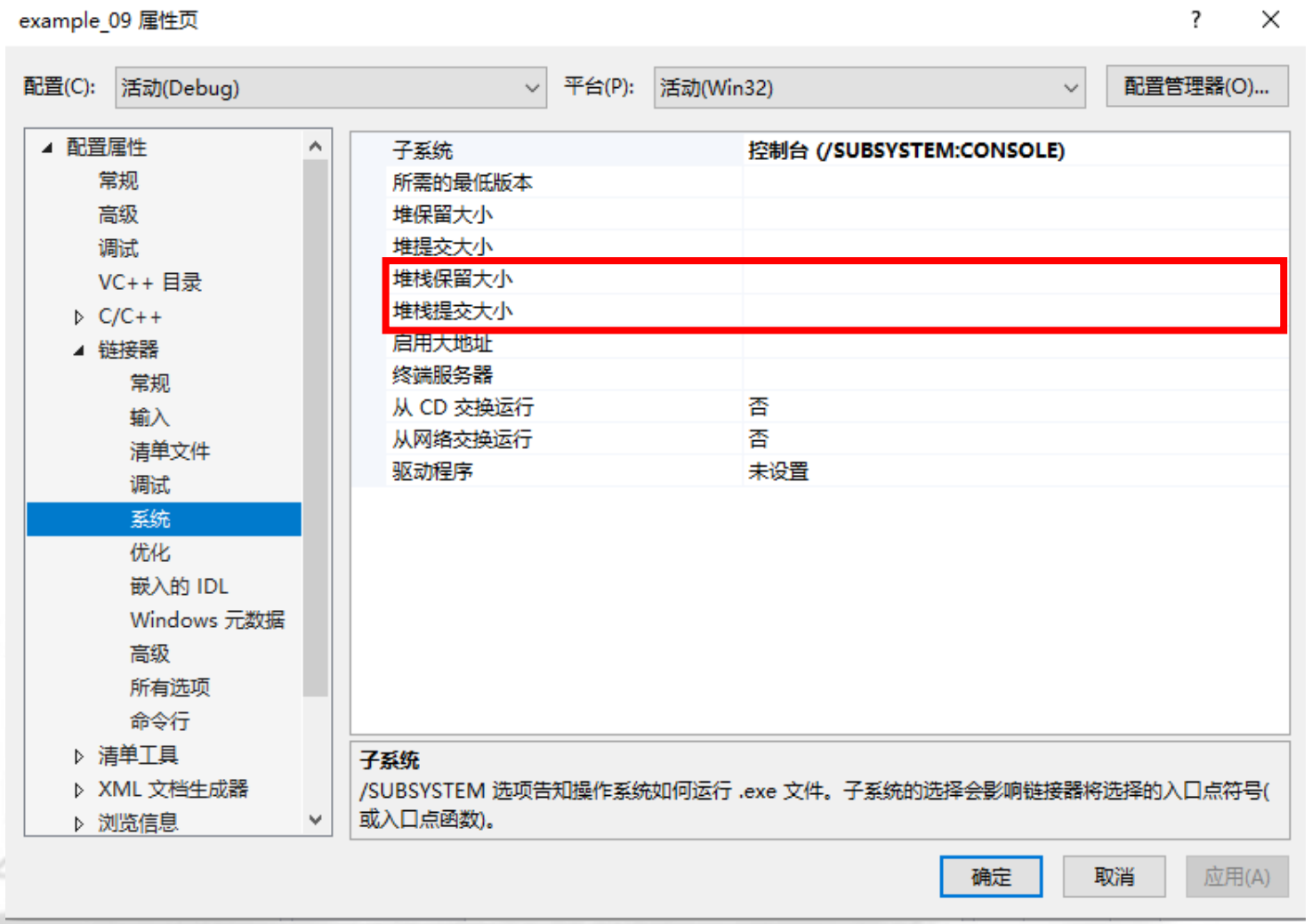
```
1  #include <stdio.h>
2  int search(int m[], int key, int low, int high){
3    int mid = (low + high) / 2;
4    if (low > high)  return -1;
5    if (key == m[mid]) return mid;
6    else if (key < m[mid])
7      return search(m, key, low, mid - 1);
8    else
9      return search(m, key, mid + 1, high);
10 }
11 void main(){
12   int a[] = { 1,2,5,7,9,12,16 };
13   int i, len;
14   len = sizeof(a) / sizeof(a[0]) - 1;
15   i = search(a, 12, 0, len);
16   printf("%d\n", i);
17 }
```

函数（模块）的递归调用

- 设置栈的大小
  - 为了防止运行时递归次数过多引起**堆栈溢出**错误，需要把编译系统默认的**栈的大小加大**
  - 方法：打开“**项目**”菜单中“**xxx属性**”（这里xxx是你的项目名，本题是example_09）对话框，找到“**配置属性**”中“**链接器**”中“系统”页
  - 修改下列任一属性：
    - 堆栈提交大小
    - 堆栈保留大小

函数（模块）的递归调用

- 设置栈的大小
  - 修改“堆栈保留大小”为500000000（500MB）



归纳思维（递归）：大道至简

- 编程时需要学会合理且灵活地拆解问题：寻找到过程和子过程之间的**相似关系**
- 在汉诺塔问题中，无需知晓所有圆盘搬运的过程，只需学会搬运一个圆盘时的规律并不断地调用，就会找到结果
- 程序设计的这一思想体现在生活中的方方面面，也是**解决复杂问题**常用的思路

移动n个圆盘从A到C

```
Void Hanoi(int disk, char A, char B,
char C){
    if ( 只有一个圆盘 )
        直接移动到C;
    先把n-1个圆盘从A移动到B;
    将最后1个圆盘从A移动到C;
    最后把n-1个圆盘从B移动到C;
}
```

8.2 编译预处理

- 文件包含命令
- 条件编译命令
- #pragma
- #line

编译预处理

- 编译预处理
  - C语言编译系统首先对程序模块中的**编译预处理命令**进行处理
  - 编译预处理命令一般是在**函数体的外面**
- C 语言提供的编译预处理命令
  - #define　　宏定义
  - #include　　文件包含命令
  - #if…　　条件编译命令
  - #pragma
  - #line

文件包含命令

- 文件包含
  - 一个源文件可以将另一个指定的源文件包括进来
- 文件包含命令
  - #include <文件名> 或 #include "文件名"
  - 将指定文件中的全部内容读(插入)到该命令所在的位置后一起被编译

文件包含命令：作用

- 文件包含命令作用
  - 将指定文件中的内容读(插入)到该命令所在位置后一起被编译
  - 在对文件file2.c进行编译处理时，将首先对其中的#include命令进行“文件包含”处理
  - 将文件file1.c中的全部内容插入到文件file2.c中的#include "file1.c" 处代替此行

```
1  //file1.c
2  int x, y, z;
3  float a, b, c;
4  char c1, c2;
```

```
1  //file2.c
2  #include "file1.c"
3  void main(){
4     ...
5  }
```

编译预处理后

```
1  int x, y, z;
2  float a, b, c;
3  char c1, c2;
4  void main(){
5     ...
6  }
```

文件包含命令

- #include "文件名"
  - 编译系统首先在源程序文件**所属的文件目录**中寻找所包含的文件。如果没有找到，再按系统规定的**标准方式**检索其他目录
- #include <文件名>
  - 编译系统按系统规定的**标准方式**检索文件目录寻找所包含的文件
  - 使用双撇号的#include命令的检索路径**包含**了使用尖括号的#include命令的检索路径
  - **#include <stdio.h>**一般不使用双引号，**防止**源程序文件所属的文件目录中包含的与stdio.h同名的**自定义文件**被引用

文件包含命令


●头文件

- □在C编译系统中，以.h为扩展名的文件
- □对相应**函数的原型**与**符号常量**等进行了说明和定义
  - 如果要在程序中使用C编译系统提供的**库函数**，则在源程序的开头应包含相应的头文件
  - 例如，如果在一个程序模块中要用到**输入或输出函数**时，则在该程序模块前要用#include <stdio.h>
- □课本**附录B**中，列出了一些常用头文件中所包含的库函数

文件包含命令：注意事项

- 当#include命令指定的文件中的内容改变时，包含这个头文件的所有源文件都应该**重新进行编译**
- 一个#include命令只能指定**一个**被包含文件，如果需要包含多个文件，则要用多个#include命令实现
- 被包含的文件应该是**源文件**，不能是经编译后的目标文件
- 文件包含可以**嵌套使用**，即被包含文件中可以使用#include
- **不能出现递归包含**，也就是A文件用#include命令文件包含B文件，则B文件不能通过#include命令直接或间接再包含A文件

文件包含命令：嵌套调用


●例8-11：文件包含命令嵌套调用

- □源文件中调用A.h
- □A.h中调用B.h

```
1  //A.h
2  #include"B.h"
3  #define A 100
```

```
1  //B.h
2  #define B 200
```

```
1  #include <stdio.h>
2  #include"A.h"
3  void main() {
4    int x = A, y = B;
5    printf("%d\n", x);
6    printf("%d\n", y);
7  }
```

调用stdio.h使用尖括号

调用A.h与B.h使用双引号

```
100
200
请按任意键继续...
```

文件包含命令：注意事项

- 注意
  - 由#include命令所指定的文件中可以有**任何语言成分**
  - 因此，通常可以将经常使用的、具有公用性质的**符号常量**、**带参数的宏定义**以及**外部变量**等集中起来放在这种头文件中，以尽量避免一些重复操作
  - **不要**用#include命令引用.c文件，不符合写代码常规
    - .c文件中是代码具体实现，同一文件中#include两个.c文件，很可能造成**变量重复定义**等错误

条件编译命令：原理

- 条件编译
  - 对C源程序中的部分内容只在**满足一定条件**时才进行编译；或者当满足条件时**对一部分语句**进行编译，而当条件不满足时对**另一部分语句**进行编译
  - 使同一个源程序在不同的编译条件下能够产生不同的**目标代码文件**
  - 便于程序在不同平台（操作系统）上的**移植**（porting），使程序具有**通用性**和**普适性**

条件编译命令：预编译处理命令

- #ifdef，#else，#endif
  - 如果“标识符”已经**定义**过（一般是指用#define命令定义），则程序段1参加编译，而程序段2不参加编译
  - 如果“标识符”**没有定义**过，程序段2参加编译，而程序段1不参加编译
  - 其中程序段1和程序段2均可以包含任意条语句（**不需要用花括号括起来**）

```
#ifdef 标识符
    程序段1
#else
    程序段2
#endif
```

条件编译命令：预编译处理命令

- #ifdef，#endif
  - 如果“标识符”已经定义过（一般是指用#define命令定义），则程序段1参加编译，否则程序段1不参加编译
  - 其中程序段1可以包含任意条语句（不需要用花括号括起来）

```
#ifdef 标识符
    程序段1
#endif
```

条件编译命令：代码示例

- 例8-12：字母转换
  - 对键盘输入的字符，如果定义LOW，则将大写字母转换成小写字母，其他字符不变
  - 否则小写转大写字母

```
input ch:A
a
请按任意键继续. . .
```

```
1  #define LOW 1
2  #include <stdio.h>
3  void main(){
4    char ch;
5    printf("input ch:");
6    scanf("%c", &ch);
7    #ifdef LOW
8      if (ch >= 'A' && ch <= 'Z' )
9      ch = ch - 'A' + 'a'; /*大写字母转换成小写字母*/
10   #else
11     if (ch >= 'a' && ch <= 'z' )
12     ch = ch - 'a' + 'A'; /*小写字母转换成大写字母*/
13   #endif
14   printf("%c\n", ch);
15 }
```

标识符

条件编译命令：代码示例

- 例8-12：分析
  - 程序开头有一个宏定义命令：#define LOW 1
  - 定义了一个常量LOW，这个常量表示什么**无关紧要**
  - 使用#define LOW也可以起到相同的作用，LOW未定义为任何内容，但是LOW已经**被定义**了，这会使得#ifdef LOW 为**真**
  - 因此，条件编译命令中的程序段1被保留并**参加编译**，而程序段2被**舍弃**

条件编译命令：代码示例

●例8-12：以下代码在编译中等价

```
1  #define LOW 1
2  #include <stdio.h>
3  void main(){
4    char ch;
5    printf("input ch:");
6    scanf("%c", &ch);
7    #ifdef LOW
8      if (ch >= 'A' && ch <= 'Z' )
9        ch = ch - 'A' + 'a' ;
10   #else
11     if (ch >= 'a' && ch <= 'z' )
12       ch = ch - 'a' + 'A' ;
13   #endif
14     printf("%c\n", ch);
15 }
```

相当于编译

```
1  #define LOW 1
2  #include <stdio.h>
3  void main(){
4    char ch;
5    printf("input ch:");
6    scanf("%c", &ch);
7    if (ch >= 'A' && ch <= 'Z' )
8      ch = ch - 'A' + 'a' ;
9    printf("%c\n", ch);
10 }
```

条件编译命令：代码示例

- 例8-13：小写字母转换大写字母
  - 去掉宏定义，则LOW未被定义，#else部分被编译
  - 程序将键盘输入的小写字母转换成大写字母（其他字符不变）输出

```
input ch:a
A
请按任意键继续. . .
```

```
1  #include <stdio.h>
2  void main(){
3    char ch;
4    printf("input ch:");
5    scanf("%c", &ch);
6    #ifdef LOW
7      if (ch >= 'A' && ch <= 'Z' )
8        ch = ch - 'A' + 'a' ;
9    #else
10     if (ch >= 'a' && ch <= 'z' )
11       ch = ch - 'a' + 'A' ;
12   #endif
13     printf("%c\n", ch);
14 }
```

条件编译命令 VS 选择结构

- 选择结构
  - 选择结构中的各程序段不管最后是否被执行，都需要进行**编译**，形成的**目标程序很长**
  - 在实际运行时，选择结构要对条件进行**判断**后才能决定执行哪个程序段，因而**运行时间长**
- 条件编译命令
  - 在编译过程中根据条件对部分程序进行编译，**减少目标程序长度**
  - 在执行过程中不再测试条件，**减少运行时间**
  - 当条件编译段比较多时，会大大**提高程序的运行效率**

条件编译命令：预编译处理命令

- #ifndef，#else，#endif
  - 如果“标识符”**没有定义**过，则程序段1参加编译，而程序段2不参加编译
  - 如果“标识符”已经**定义**过，程序段2参加编译，而程序段1不参加编译
  - 其中程序段1和程序段2均可以包含任意条语句（**不需要用花括号括起来**）

```
#ifndef 标识符
    程序段1
#else
    程序段2
#endif
```

条件编译命令：预编译处理命令

- #ifndef，#endif
  - 如果“标识符”**没有定义**过，则程序段1参加编译，否则程序段1不参加编译
  - 其中程序段1可以包含任意条语句（**不需要用花括号括起来**）

```
#ifndef 标识符
    程序段1
#endif
```

条件编译命令：代码示例


●例8-14：小写字母转换大写字母(将#ifdef改成#ifndef)

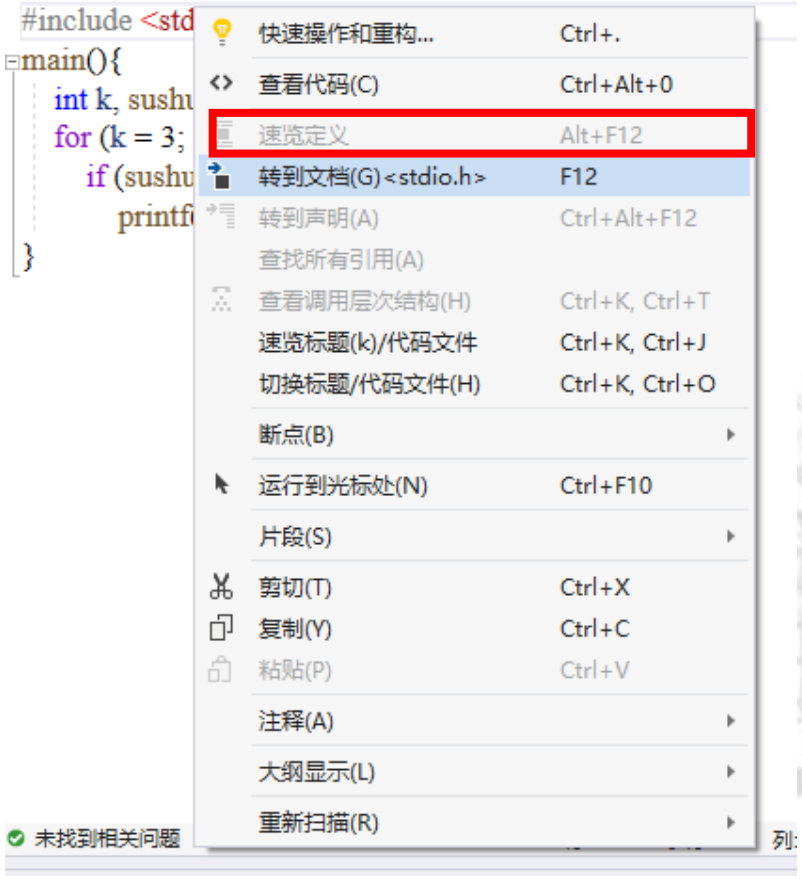
```
1  #define LOW 1
2  #include <stdio.h>
3  void main(){
4    char ch;
5    printf("input ch:");
6    scanf("%c", &ch);
7    #ifndef LOW
8      if (ch >= 'A' && ch <= 'Z' )
9        ch = ch - 'A' + 'a'; /*大写字母转换成小写字母*/
10   #else
11     if (ch >= 'a' && ch <= 'z' )
12       ch = ch - 'a' + 'A'; /*小写字母转换成大写字母*/
13   #endif
14   printf("%c\n", ch);
15 }
```

标识符

```
input ch:a
A
请按任意键继续. . .
```

条件编译命令：条件编译命令作用

- 查看**编译系统**提供的.h文件的内容
  - 在相应的.h文件名上按鼠标右键，会弹出下拉菜单：




```
//
// stdio.h
//
//      Copyright (c) Microsoft Corporation. All rights reserved.
//
// The C Standard Library <stdio.h> header.
//
#pragma once
#ifndef _INC_STDIO // include guard for 3rd party interop
#define _INC_STDIO

#include <corecrt.h>
#include <corecrt_wstdio.h>

_CRT_BEGIN_C_HEADER

/* Buffered I/O macros */

#define BUFSIZ  512
```

条件编译命令：条件编译命令作用

- 条件编译还用来防止多个文件引用同一个头文件时，出现**多重定义**同一个外部变量或说明的问题。
- 打开stdio.h，有如下语句

```
1 #ifndef _INC_STDIO // include guard for 3rd party interop
2 #define _INC_STDIO
```

- 若多个C文件都包含头文件stdio.h，则每个C文件编译时，首先判断_INC_STDIO是否已经宏定义过了，若没定义过，则宏定义_INC_STDIO
- 一旦某个文件引用过stdio.h，则_INC_STDIO已经宏定义过。若此文件再包含文件stdio.h，则**不会重复定义**_INC_STDIO

条件编译命令：预编译处理命令

- #if，#else，#endif
  - 如果常量表达式的值为**“真”（值非0）**，则程序段1参加编译，而程序段2不参加编译
  - 如果常量表达式的值为**“假”（值为0）**，程序段2参加编译，而程序段1不参加编译
  - 其中程序段1和程序段2均可以包含任意条语句（**不需要用花括号括起来**）

```
#if 常量表达式
    程序段1
#else
    程序段2
#endif
```

条件编译命令：预编译处理命令

- #if，#endif
  - 如果常量表达式的值为“真”（值非0），则程序段1参加编译，否则程序段1不参加编译
  - 其中程序段1可以包含任意条语句（不需要用花括号括起来）

```
#if 标识符
    程序段1
#endif
```

条件编译命令：预编译处理命令

- #if，#elif，#else，#endif
  - 如果常量表达式1的值为**“真”（值非0）**，则程序段1参加编译，其余程序段不参加编译
  - 如果常量表达式2的值为**“真”（值非0）**，则程序段2参加编译，其余程序段不参加编译
  - ……
  - 最终是从n+1个程序段中选择一段参加编译，生成相应可执行程序代码

```
#if 常量表达式1
    程序段1
#elif 常量表达式2
    程序段2
……
#elif 常量表达式n
    程序段n
#else
    程序段n+1
#endif
```

条件编译命令：代码示例

- 例8-15: 大写字母转换小写字母(将#ifdef改成#if)

```
1  #define LOW 1
2  #include <stdio.h>
3  void main(){
4    char ch;
5    printf("input ch:");
6    scanf("%c", &ch);
7    #if LOW
8      if (ch >= 'A' && ch <= 'Z')
9      ch = ch - 'A' + 'a'; /*大写字母转换成小写字母*/
10   #else
11     if (ch >= 'a' && ch <= 'z')
12     ch = ch - 'a' + 'A'; /*小写字母转换成大写字母*/
13   #endif
14   printf("%c\n", ch);
15 }
```

```
input ch:A
a
请按任意键继续. . .
```

条件编译命令：预编译处理命令

- #undef 标识符
  - 将已经定义的标识符变为**未定义**

```
1  #define WIDTH 80
2  #define ADD( X, Y ) ((X) + (Y))
3  …
4  #undef WIDTH
5  #undef ADD
```

#pragma命令

- 一般形式：**#pragma token-string**
- 作用：指示编译器如何进行编译
  - 比如如何处理某文件被多次include，如何进行内存存放处理
  - 本课程将仅讲once、warning的使用
- token-string有多种
  - alloc_text, auto_inline, bss_seg, check_stack, code_seg, const_seg, comment, component, data_seg, function, hdrstop, include_alias, init_seg1, inline_depth, inline_recursion, intrinsic, message, **once**, optimize, pack, pointers_to_members1, setlocale, vtordisp1, **warning**

#pragma命令

- **#pragma once**
  - 让编译器把指定的文件只包含一次，防止此文件被多次引用出现的重复定义等错误。通常放在头文件的开始处
  - 打开stdio.h，开头有如下语句
    - 其作用是当一个文件多次include文件stdio.h时，让编译器把文件stdio.h只文件包含一次，防止多次引用出现的重复定义等错误

```
1  // stdio.h
2  //
3  //      Copyright (c) Microsoft Corporation. All rights reserved.
4  //
5  // The C Standard Library <stdio.h> header.
6  //
7  #pragma once
```

#pragma命令

- #pragma warning(disable:4996)
  - 作用是将**4996类警报**置为失效，让编译器不再显示这类警告

```
1  #include<stdio.h>
2  void main(){
3     int x;
4     scanf("%d", &x);
5     printf("%d\n",x);
6  }
```

  - 编译结果：
    - error C4996: 'scanf': This function or variable may be unsafe. Consider using scanf_s instead. To disable deprecation, use _CRT_SECURE_NO_WARNINGS. See online help for details.

#pragma命令

- #pragma warning(disable:4996)
  - 程序开头加上

```
1  #include<stdio.h>
2  #pragma warning(disable:4996)
3  void main(){
4    int x;
5    scanf("%d", &x);
6    printf("%d\n",x);
7  }
```

  - 将不会再出现这类错误

#line命令

- 一般形式：**#line 数字["文件名"]**
  - "文件名"是任选项
- 作用
  - 让编译器编译显示错误信息时，改变当前所显示的**行号**和**文件名**，便于调试
  - 例8-16：在文件test.c中，插入#line 151
    - 从此行后，编译信息显示将是test.c的151行开始的计数，实际上尽管#line 151所在的行可能是第1行。
  - 例8-17：文件test.c中，插入#line 151 "copy.c "
    - 从此行后，编译信息显示将是copy.c的151行开始的计数

本节总结

- 函数（模块）的递归调用
  - 直接递归、间接递归
  - **递归式+初始条件**
- 编译预处理
  - 文件包含命令　　#include<stdio.h>
  - 条件编译命令　　#ifdef, #else, #endif
  - #pragma　　#pragma once
  - #line　　#line 151

本节作业

- 作业8
  - 课本第八章习题6,10
- 上机实验
  - 课本第八章习题11（要求写实验报告）

附加作业（一）

- 约瑟夫环
  - 有n个人逆时针围成圆圈，标号依次是1，2，…，n。此时从1号开始逆时针计数，数到k的人令其出局（k为给定自然数），后面的人从1开始重新计数。例如，当n=6，k=2时，出局的人依次为2、4、6、3、1，最终胜利的人是5号。
  - 请编写C程序，对于任意给定n和k，求胜利者的编号，要求使用递归思路编写。

附加作业（二）

- 辗转相除法求最大公约数
  - 辗转相除原理：两个整数的最大公约数等于其中较小的数和两数相除余数的最大公约数。
  - 基于上述原理，请用C语言编写求两整数最大公约数的算法，要求使用递归思路。

附加作业（三）

- 子集问题
  - 集合中每个元素各不相同，对于某一给定集合，我们希望知道它所有的子集。注意，空集和集合本身也是子集。
  - 请编写C程序，对于任意给定整数集合，列出其所有子集，要求使用递归思路编写。


THANKS