

# 函数



# CONTENTS

01

概述：模块,函数,算法

02

函数的定义与说明

03

函数的调用

04

数组与函数

05

变量的作用域

06

变量的存储属性



陕西科技大学

SHAANXI UNIVERSITY OF SCIENCE & TECHNOLOGY

## 一、概述

当你遇到一项一个人无法完成的任务时，你可能会考虑以下几种方案：

- 1、看看有没有别人做过的该任务的经验，我们可以直接借鉴。
- 2、找一些合作伙伴协同完成。

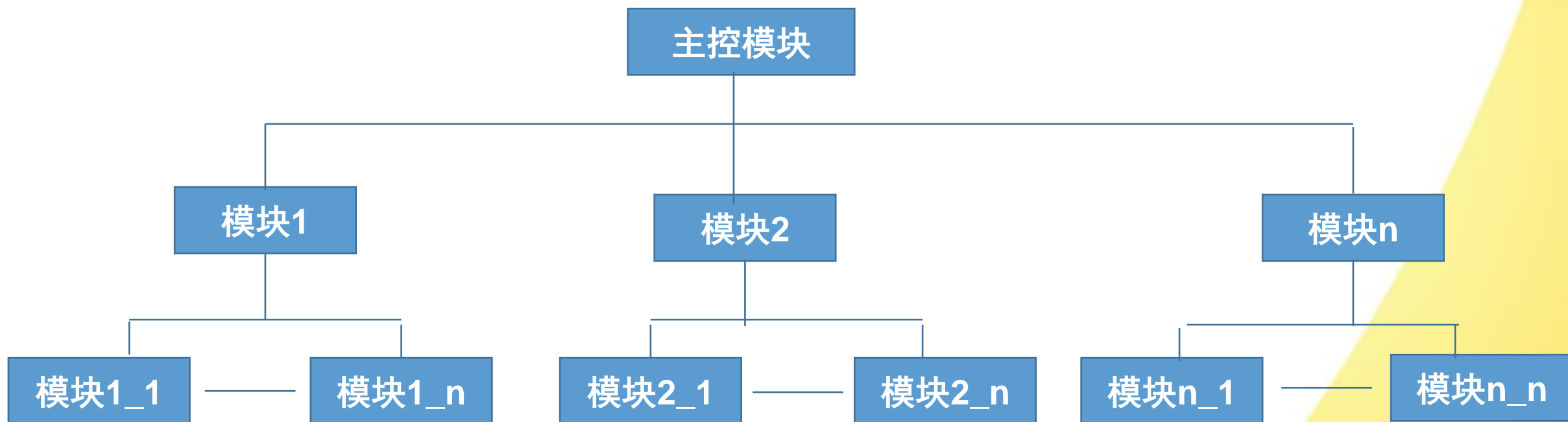
# 模块化设计



# 模块化设计

“自顶向下”的模块化程序设计方法：

将一个大问题分解成多个解决小问题的设计思想。



# 1 模块与函数

## a. 功能模块

求解较小问题的算法和程序称作“功能模块”，各功能模块可以先单独设计，然后将求解所有子问题的模块组合成求解原问题的程序。

## b. 函数

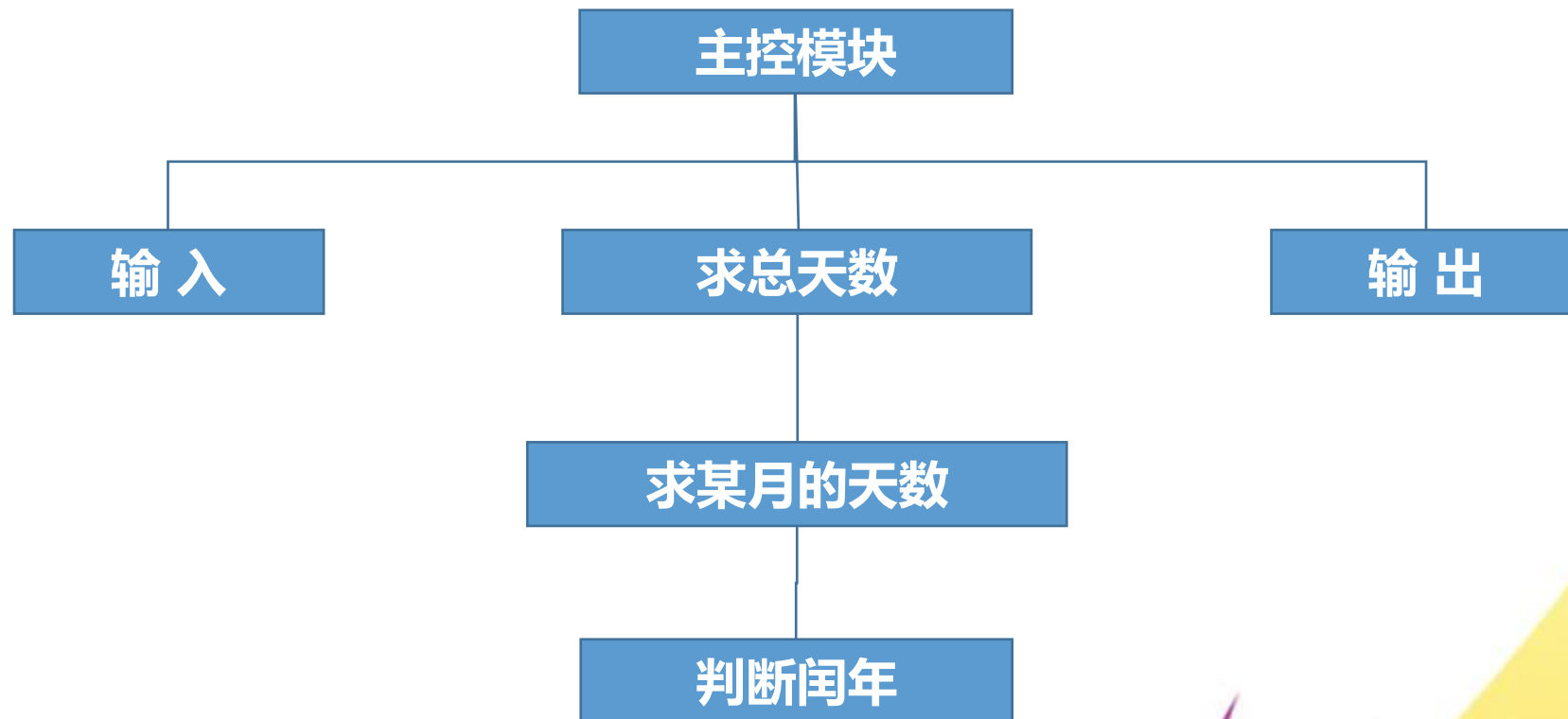
完成相对独立功能的程序。

### 模块化程序设计优点：

- 1、提高代码的复用率
- 2、提高代码的可读性
- 3、简化了代码维护工作
- 4、为团队合作提供了便利

## 【例题1】输入年月日，计算出该日为该年的第几天。

- 当前月以前的每月的天数
- 当月天数
- 闰年的2月天数要多一天



# 程序实现:

(1) 判断闰年。

输入：年份（整数）

输出：是否为闰年（整数：0为不是，1为是）

思路：年份能被4整除并且不能被100整除或者年份能被400整除，则为闰年

```
int leap(int year)
```

```
{int lp; lp=(year%4==0&&year%100!=0||year%400==0)?1:0;
```

```
return lp;
```

```
}
```

(2) 求某月的天数。

输入：年份（整数），月份（整数）

输出：当月的天数（整数）

思路：若为1、3、5、7、8、10、12月则当月天数为31天，  
若为2月，则看年份若为闰年当月天数为29天，否则为28天，  
其他月份当月天数为30天。

```
int month_days(int year, int month)
{ int ds, d;
  switch(month)
  {case 1:
   case 3:
   case 5:
   case 7:
   case 8:
   case 10:
   case 12: d=31; break;
   case 2: d=leap(year)?29:28; break;
   default: d=30;}
  return d;
}
```



(3) 求天数和。

输入：年份（整数），月份（整数），日子（整数）

输出：总天数（整数）

思路：加和月份以前每个月的天数，再加上日子。

```
int days(int year,int month,int day)
```

```
{int i,ds=0;
```

```
for (i=1;i<month;i++)
```

```
    ds=ds+month_days(year,i);
```

```
ds=ds+day;
```

```
return ds;
```

```
}
```

#### 4) 在主函数中分别调用三个函数。

```
void main()
{ int year,month,day,t_day;
  printf("Input year-month-day:\n");
  scanf("%d-%d-%d",&year,&month,&day);
  t_day=days(year,month,day);
  printf("%d-%d-%d is %dth day of the year!\n",year, month,day,t_day);
}
```

注意:

在完整的程序中,前三个函数应放在main( )函数之前。

## 2 模块设计三个原则

### 模块独立

- 功能独立的子功能
- 模块之间的关系简单
- 使用独立变量

### 模块规模适当

- 过大会失去模块化意义
- 过小会增加调用开销，降低效率

### 分解模块要注意层次

- 对问题抽象化
- 设计时细化

## 二、函数定义与使用

### 一、标准库函数

定义在不同的头文件中

用户使用时，必须用#include“头文件”把相应的头文件包含到程序中来。

```
#include <math.h>    /* 包含math.h头文件 */
#include <stdio.h>    /* 包含 stdio.h 头文件 */
main( )
{ double a, b;
  scanf ("%f",&a);    /*调用输入函数，输入变量a的值*/
  b = sin (a);         /*调用sin函数，求sin (a) 的值*/
  printf( "%7.4f", b); } /*调用输出函数，输出变量b的值*/
```

## 二、用户自定义函数

### 1. 函数类型

无参函数

函数的定义无参数说明

有参函数

定义的参数有一个或一个以上的参数

空函数

当定义的函数既无参数也无执行语句。

空函数被调用时，什么也不做立即返回其调用函数。

## 2. 函数定义

### 方式1

函数返回值类型名 函数名(参数列表)

参数类型说明

{ 局部变量说明;  
语句序列; }

如: `int max(a,b)`  
`int a,b;`

### • 方式2

函数返回值类型名 函数名(参数类型说明及参数列表)

{ 局部变量说明;  
语句序列; }

如: `int max(int a,int b)`

## 【例6-3】定义符号函数sign。

```
int sign(x)
/*函数返回值类型未说明，默认为int，建议给出函数类型说明*/
int x;          /*形式参数说明*/
{int y;         /*函数体局部变量*/
  y=x>0?1:(x==0?0:-1);
  return y;      /*返回函数值*/
}
```

**注意：** C语言函数分为两大部分：

函数的说明部分

函数体部分。

# 函数各部分作用

## 1)函数的说明部分

函数说明部分说明函数的类型, 函数名, 参数表及参数类型。

### (1)函数的类型说明

函数的类型即函数的返回值类型。若函数不提供返回值, 则可定义其类型为: void。

例如: void putdata(int a)

### (2) 函数名

函数名又称函数标识符。命名遵循C语言标识符的规定; 函数名要反映函数完成的功能。



### (3)参数表

参数表写在函数名后的( )内，由一个或多个变量标识符及类型标识符组成。

参数表中的变量称为形式参数，简称形参。

若函数没有形参，则称为无参函数，其后“( )”不能省略。

参数必须指定类型。形参的类型说明有两种：

方法1: `int max(a,b)`

`int a,b;`

方法2: `int max(int a, int b)`

省略函数类型名时，C语言默认其为int型。

## 2)函数体

函数体包括变量定义和执行语句序列。函数所完成的工作由函数体中一段程序实现。

函数的返回值用返回语句return返回，形式：

return(表达式);

或 return 表达式;

如果函数的类型与return语句的expressions的类型不一致时, 则以函数的类型为准。返回时自动进行数据转换。(见下页例题)

## 例7.3 定义函数power(x,n), 求x的n次方。

函数定义如下:

```
float power( float x, int n)
{ int i;
  float t=1;
  for(i=1;i<=n;i++)
    t = t * x;      /* 1*x*x*...*x 共乘n次 */
  return t; }      /* 返回t的值 */
```

## 7.3 自定义函数的调用

### 7.3.1 函数调用与声明

#### 1. 函数的调用

- 有参数函数的调用形式:

函数名(参数)

- 无参数函数的调用形式:

函数名()

**注意：**当实际参数的个数、次序、类型与对应形式参数的个数、次序、类型不一致时，系统并不提示错误，后果却难以预测。

# 函数调用形式

- 函数语句调用、函数表达式调用和函数参数调用

```
void max(int a,int b,int c)
```

```
{ int y;
```

```
  y=(a>b)?a:b;
```

```
  y=y>c?y:c;
```

```
  printf("max=%d\n",y);}
```

```
void main()
```

```
{int x,y,z,m;
```

```
scanf("%d,%d,%d",&x,&y,&z);
```

```
max(x,y,z); } /*采用函数语句形式调用函数max*/
```

# 函数调用形式

```
int max(int a,int b)
{ int y;
  y=(a>b)?a:b;
  return y;}
void main()
{ int x,y,z,m;
  scanf("%d,%d,%d",&x,&y,&z);
  m=max(x,y); m=max(m,z);
  printf("max=%d\n",m);/*表达式调用形式 */
  m=max(x,y);
  printf("max=%d\n",max(m,z)); /*函数参数调用形式 */}
```

## 2.函数声明

- 函数定义在main()之后，需要进行函数说明。

类型名 函数名(类型1 变量1,类型2 变量2,...,类型n 变量n);

- 说明:

函数声明应与该函数定义的函数类型与名称、形参的个数、类型、次序相一致。

函数声明中的形参名可省略，其形式为:

类型名 函数名（类型1，类型2， ...， 类型n）；

类型名 函数名();

当函数定义在主调函数之前，即先定义，后调用。则调用时函数声明可以省略。

## [例7.4] 编写计算x的n次乘方的程序。

```
#include "stdio.h"
main( )
{ float x, y;
  int n;
  float power (float x, int n);
  scanf("%f,%d", &x, &n);
  y = power(x, n);
  printf("%8.2f",y); }
```

```
float power( float x, int n)
{  int i;
   float t=1;
   for(i=1; i<=n; i++)
       t = t * x;
   return t; }
```



## 7.3.2 形参与实参

- 实参的个数必须与形参相等，且参数顺序、类型要对应一致。  
实参与形参是按位置一一对应地传递数据的。

[例7.5] 编程输入两个数，输出其中较大的。

```
#include "stdio.h"
main( )
{ int a, b, m;
  int max(int, int );
  scanf("%d,%d", &a, &b);
  m = max(a, b);
  printf("max=",m);}
```

```
int max(int x, int y)
{ int t;
  if (x>y) t = x;
  else t = y;
  return t; }
```

若程序运行时输入为: 10,5

便有输出为: 10

- 调用max函数时，实参a把值传给形参x，实参b把值传给形参y，按顺序传递，与形参名称无关。实参a和形参x，实参b和形参y一一对应。

1) 形参y之间值的传递如图4.7所示意。



2) 关于形式参数和实际参数说明如下:

- 形式参数在函数被调用时才被分配内存。当函数执行完毕返回时, 形式参数占用的内存空间便被释放。
- 实参可以是变量、常量和表达式。

如:  $y = \text{power}(x, 4);$      $y = \text{power}(x, i * 2);$

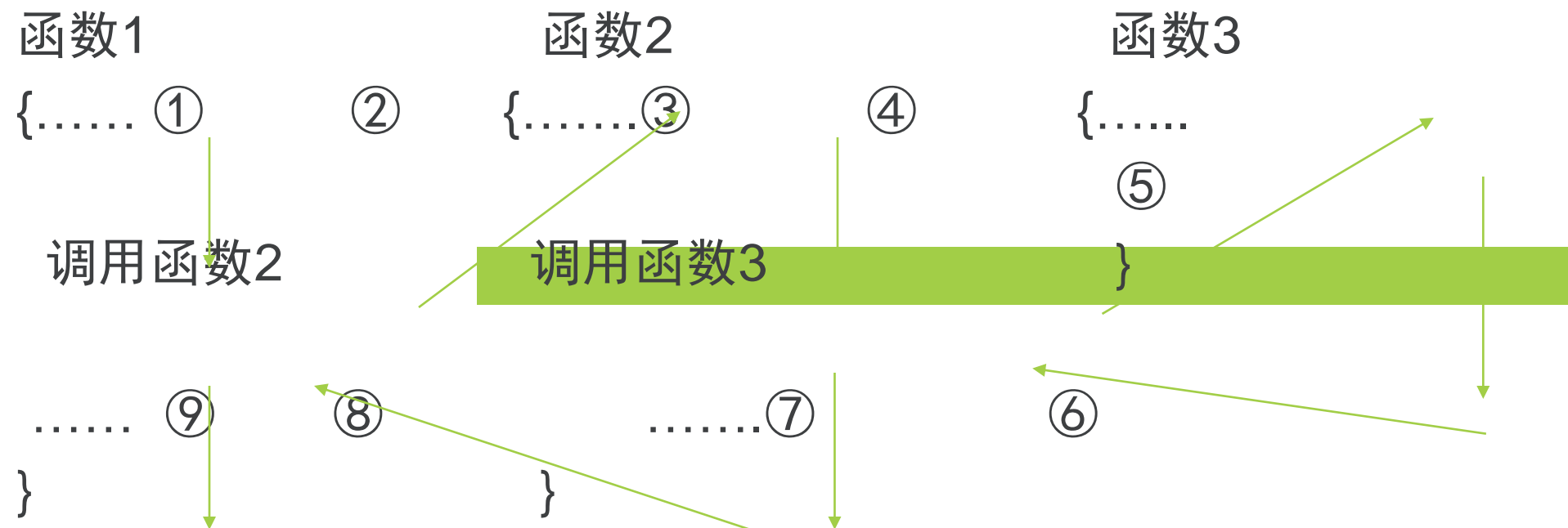
但实参必须有确定的值。

3) 形参和实参的类型必须相容。

4) 形参和实参之间的关系是: **单向的值的传递**

## 7.3.3 函数的嵌套调用

调用一个函数的过程中又调用了另一个函数，这种调用称为函数的嵌套调用。



## 【例6-9】 求方程 $ax^2+bx+c=0(a\neq 0)$ 的根。

```
#include <stdio.h>
#include <math.h>
void main()
{ float a,b,c,x1,x2;
  int dict(float,float,float);
  float root(float,float,float,int);
  printf("Input a,b,c:");
  scanf("%f,%f,%f",&a,&b,&c);
  if (dict(a,b,c))
  { x1=root(a,b,c,1); /*调用函数root*/
    x2=root(a,b,c,0);
    printf("实根x1=%f,x2=%f\n",x1,x2);}
  else printf("无实根!\n");
}
```



## dict()和root()

```
int dict(float a,float b,float c)
{int f;
if (b*b-4*a*c>=0) f=1;
  else f=0
return f; }
float root(float a,float b,float c,int flag)
{float d,x;
d=dict(a,b,c); /*调用函数 dict*/
if (d)
  x=flag?(-b+sqrt(d))/(2*a):(-b-sqrt(d))/(2*a);
return x;}
```

# 嵌套过程

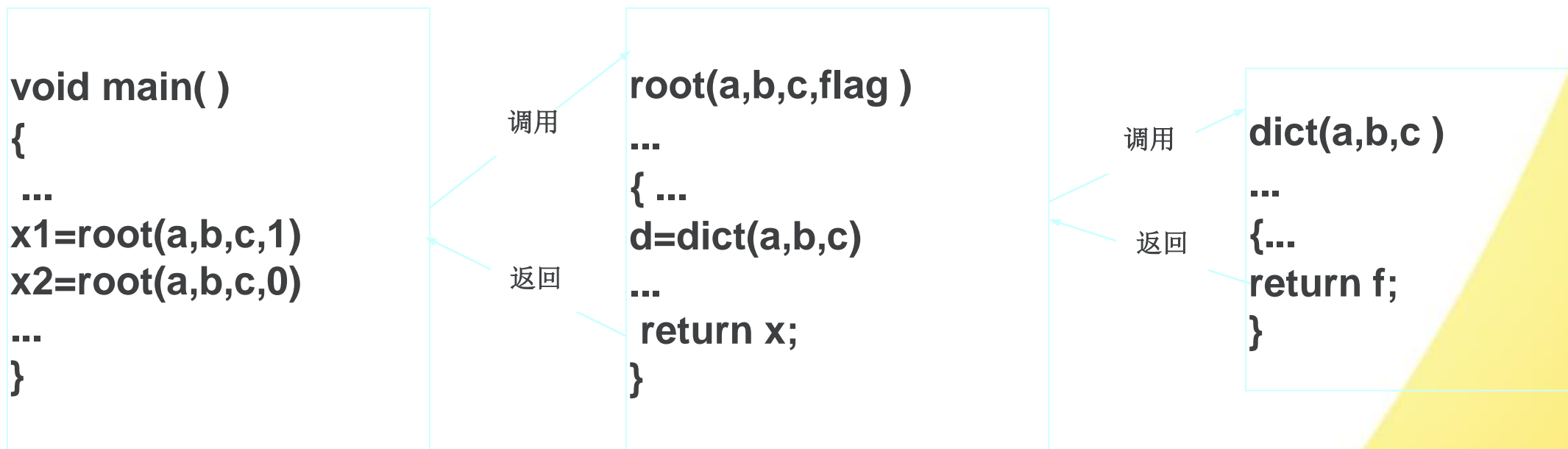


图6-3 嵌套调用过程

## 7.3.4 递归调用(重点)

- 函数调用函数本身，称为函数的递归调用。递归调用形式如下：

### 1) 直接递归

```
void a( )  
{.....  
    a ( ); .....  
}
```

### 2) 间接递归

```
void a( )  
{ .....  
    b ( );  
    ..... }  
void b( )  
{ .....  
    a ( );  
    ..... }
```



# 用递归算法计算n!

- 讨论:

采用递归的方法计算。n!的递归定义形式的:

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

- 编程:

if (初始条件)

表达式;

else 递推表达式;

## 例程序：

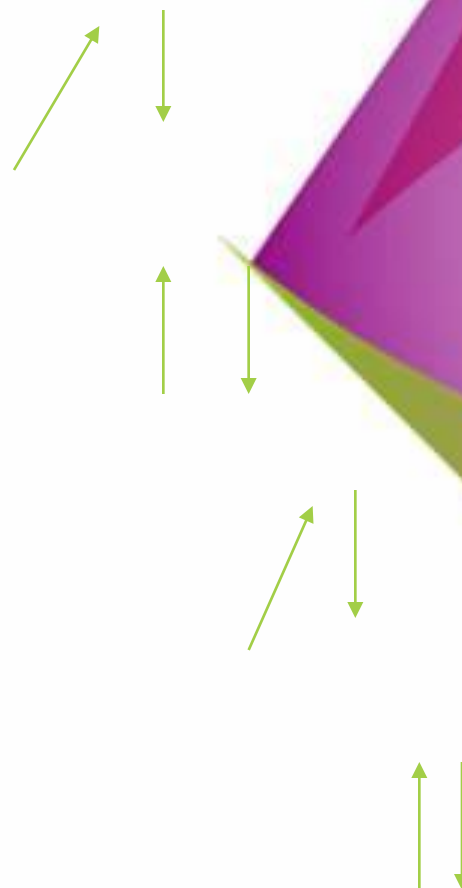
```
#include <stdio.h>
long fac(unsigned n)
{ long f;
  if (n==0)
    f=1; /*递归结束条件*/
  else f=n*fac(n-1);
  return f;
}
```

```
main( )
{ long y;
  int n;
  scanf("%d", &n);
  y = fac(n);
  printf("%d!=%ld\n", n, y);
}
```

分析:当程序输入3时

⑧          ①  
          3\*fac(2)  
⑦          ②  
          2\*fac(1)  
⑥          ③  
          1\*fac(0)  
⑤          ④  
              1

y=fac(3)



## [例7.11] 汉诺塔游戏

- 汉诺塔(Tower of Hanoi)游戏。底座上有三根针，第一根针上放着从大到小64个金片。游戏的目标是把所有金片从第一根针通过第二根针移到第三根针上。移动过程中大的金片不能压在小的金片上。
- 把 $n(n>1)$ 个金片从第一根针a上移到第三根针c的问题分解成如下步骤:
  - (1) 将 $n-1$ 个金片从a经过c 移动到b。
  - (2) 将第 $n$ 个金片移动到c。
  - (3) 再将 $n-1$ 个盘子从b经过a移动到c。

```

void hanoi(int n, int a, int b, int c)
{ if (n==0) return;      /* 0个金片不处理 */
  if (n==1) printf("%d ->%d\n",a,c); /* n=1时, 直接将金片从a移动到c */
  else
  { hanoi(n-1,a,c,b);    /* 先将n-1个金片从a经过c 移动到b */
    printf("%d ->%d\n",a,c); /* 将第n个金片从a移动到c */
    hanoi(n-1,b,a,c); } /* 再将n-1个金片从b经过a移动到c */
}

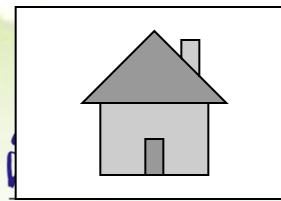
```

主函数如下：

```

#include <stdio.h>
main( )
{ int n;
  printf("Input n:");
  scanf("%d",&n);
  hanoi(n,1,2,3);/*n个金片从第一根针经过第二根针移动到第三根针上*/ }

```



## 7.4 变量的作用域及存储特性

- [例]

```
void f1( )  
{ int t=2; a *= t; b /= t; }  
main()  
{ int a, b;  
  printf(" Enter a,b:");  
  scanf("%d,%d", &a, &b);  
  f1( ); /* 调用函数f1( ) */  
  printf ("a=%d,b=%d", a, b); }
```

- 编译程序会提示出错: Undefined symbol 'a' 和 Undefined symbol 'b' 。为什么?

## 7.4.1 变量的作用域

1.变量按作用域：分为全局变量和局部变量

2.区别：

	全局变量(外部变量)	局部变量(内部变量)
定义位置	函数体外	函数体内
作用域	从定义处到文件结束	从定义处到本函数结束
举例	所有在函数体外定义的变量	(1) 所有在函数体内定义的变量 (2)形式参数
注意	与局部变量同名的处理	不同函数中同名局部变量互不干扰

## [例]程序

```
#include <stdio.h>
int a,b; /*a, b为全局变量*/
void f1( )
{ int t1,t2;
  t1 = a * 2;
  t2 = b * 3;
  b = 100;
  printf ("t1=%d,t2=%d\n", t1, t2); }
main()
{ a=2; b=4;
  f1( );
  printf ("a=%d,b=%d", a, b); }
```

程序输出结果为:

t1=4,t2=12

a=2,b=100



## 将程序改为：

```
#include <stdio.h>
int a=2,b=4; /*a,b为全局变量*/
void f1( )
{ int t1,t2;
  t1 = a * 2;
  t2 = b * 3;
  b = 100;
  printf ("t1=%d,t2=%d\n", t1, t2); }
main()
{ int b=4;
  f1( );
  printf ("a=%d,b=%d", a, b); }
```

程序输出结果为：

**t1=4,t2=12**

**a=2, b=4**

结论：全局变量与局部变量同名时，局部变量的作用域屏蔽全局变量

## 7.4.2 变量的存储特性

### 1. 变量按存在时间分: 静态变量, 动态变量

静态存储变量: 生存期为程序执行的整个过程, 在该过程中占有固定的存储空间, 也称永久存储。

动态存储变量: 只生存在某一段时间内。


例如: 函数的形参、函数体或分程序中定义的变量, 只有当程序进入该函数或分程序时才分配存储空间, 函数/分程序执行完后, 变量的存储空间又被释放。

### 2. 变量属性: 数据类型, 存储特性

完整的变量定义:

[存储特性] [数据类型] 变量名;

### 3.变量的存储特性



自动型	auto
静态型	static
寄存器型	register
外部型	extern

#### 1) auto型

有形式参数、函数内变量、分程序变量。

进入程序自动分配内存，不长期占用内存。

#### 2) static 型

①局部静态变量 ②全局静态变量

长期占用内存

## [例]

```
f(int a)
{int b=0; static int c=3;
  b++;c++;
  printf("%5d%5d%5d",a,b,c);
  return(a+b+c);
}
main()
{ int a=2,k;
  for(k=0;k<3;k++)
    printf("%5d\n",f(a));
}(看L4_11)
```

### 3) register型

- 将使用频率高的变量定义为register型,可以提高运行速度。



- 寄存器变量只限于**整型**、**字符型**、**指针型**的局部变量。寄存器变量是动态变量，仅允许说明两个寄存器变量。

例如: `register int d;`  
`register char c;`

## 4) extern型

- 引用: extern 类型 变量名;

如果某个模块文件中要用到另一个模块文件中的全局变量, 要用extern说明。

- 例如:

程序模块file1.c中定义了全局变量

```
int s ;
```

另一程序file2.c的函数fun1()需要使用这个变量s。在file2.c的fun1()对s进行外部变量说明:

```
fun1()
```

```
{ extern int s ; /*表明变量s是在其他文件定义的*/
```

```
..... }
```

- 定义时分配内存,其他文件引用时不再分配内存。

练习:

```
main()
```

```
{int i=1;
```

```
static int a=10;
```

```
register int b=5;
```

```
printf("i=%d,a=%d,b=%d\n",i,a,b);
```

```
other();
```

```
printf("i=%d,a=%d,b=%d\n",i,a,b);
```

```
}
```

```
other()
```

```
{int i;
```

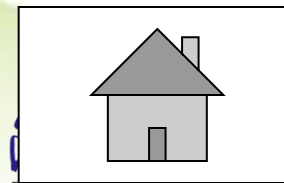
```
static int a;
```

```
i=6;
```

```
a=100;
```

```
printf("i=%d,a=%d\n",i,a);}(看L4_12)
```

47



大学

SHAANXI UNIVERSITY OF SCIENCE & TECHNOLOGY

# THANKYOU

