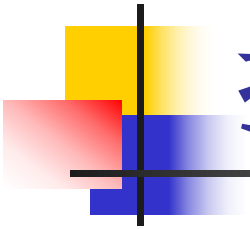
第8讲 函数的设计方法

周水庚

2024-10-24/31



提要

- 函数基础知识
- 函数定义
- 函数调用
- 函数形参
- 函数说明
- 递归函数基础
- 命令行参数
- 函数程序设计实例

提要

- 函数基础知识
- 函数定义
- 函数调用
- 函数形参
- 函数说明
- 递归函数基础
- 命令行参数
- 函数程序设计实例

函数基础知识

- 结构化程序设计中，将复杂的功能分解成若干简单的子功能，并用函数实现子功能，通过调用函数实现子功能要求
- 函数是一个实现指定功能、逻辑上独立的代码段
- 对函数使用者来说，把它看作“黑盒”，只需知道要传送给函数的数据（输入），和函数执行后能得到什么结果（输出）
- 函数还可以定义局部对象，使函数在逻辑上作为程序的一个相对独立单位，不受主函数或其他函数对程序对象命名的影响

函数基础知识（续）

- 函数可带形参，使函数执行时，操作对象、求值方式等可随不同调用的需要而改变
- 函数为程序的层次构造和开发提供支持，使设计新程序能在已有函数基础上构造功能更强的函数和程序
- 一个C程序以 main()函数作为程序的主函数。程序运行时，从它开始执行
- 在C语言中，函数不能嵌套定义，一个函数并不从属于另一个函数
- 除不能调用 main() 函数外，其它函数可以相互调用

函数库

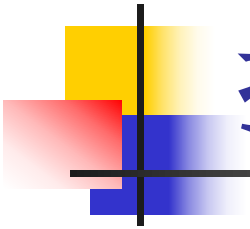
- 把一些公用的、基本的计算功能所对应的函数集中起来，构成一个库，称之为函数库，相应的函数成为库函数
- 函数库中的函数具有预先定义的、标准的输入、输入接口
- C语言中定义了一些基本的标准函数
- 编程环境工具厂商（Microsoft、Borland等）往往提供更多的函数供编程者使用

函数库（续）

- C语言使用头文（header file，即*.h文件）对函数库中的函数进行定义和说明
- 函数库中的函数经编译后，绑定在一起件，构成一个库文件（library file，即*.lib文件）
- C程序调用C语言或者编程环境提供的函数时，要在程序中include相应的头文件；在产生执行文件时，需要与库文件中相应的目标函数代码连接
- 为了使用方便，C语言按功能分类，提供了大量函数库，每个函数库都有自己的头文件

库函数的使用

- 使用相应库函数的程序都要在使用之前写上包含其头文件的预处理命令
- 常用的头文件
  - stdio.h （输入输出库函数）
  - math.h、stdlib.h、float.h （数学库函数）
  - time.h （时间库函数）
  - ctype.h （字符分类和转换库函数）
  - string.h （内存缓冲区和字符串处理库函数）
  - graphics.h （图形处理库函数）
  - malloc.h、stdlib.h （内存动态分配库函数）
  - signal.h、process.h （进程控制库函数）



提要

- 函数基础知识
- 函数定义
- 函数调用
- 函数形参
- 函数说明
- 递归函数基础
- 命令行参数
- 函数程序设计实例

函数定义

- 函数定义的一般形式为

```
类型标识符　函数名(形式参数说明表)
{
   说明和定义部分
   语句序列
}
```

函数定义（续）

- **类型标识符**用于标识函数执行结果返回值的类型
  - 当函数不返回值时，习惯用void来标记
  - 当函数返回int型值时，类型标识符int可以省略
- 函数名是一个标识符，一个C程序有且只有一个main()函数，其它的函数名可以随意命名

函数定义（续）

- 函数名之后括号内的形式参数说明表是按需要而定
  - 没有形参的函数，也就没有形参说明表，常用void代之，但函数名之后的一对圆括号不可省略
  - 当函数有多个形参时，形参说明之间用逗号分隔，每个形参说明指定形参名和类型

函数定义（续）

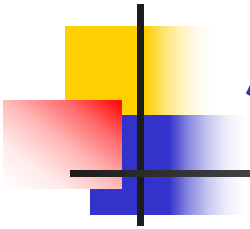
- 最外层花括号“{”和“}”括住的部分是函数体
  - 在函数体的前面部分可有函数需要的程序对象的说明和定义
  - 函数体内定义的变量是局部变量，只能在函数体内引用它们
  - 说明和定义之后是由语句序列组成的代码

例子1

- 求两个数中最小值的函数min()

```
double min(double x, double y) /* 返回 double 型值，有两个形参 x, y，都为 double 型的 */
{ /* 函数返回x和y中的小者的值 */
      return x < y ? x : y;
}
```

- return语句的执行将结束函数的执行
- C语言的return语句有两种形式：
  - return;用于不返回值的函数体中
  - return 表达式；用于有返回值的函数体中

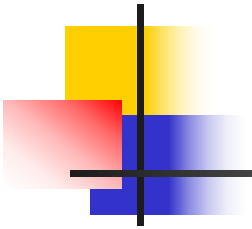


例子2

- 求两个正整数最大公因子的函数gcd()
- 两个正整数a和b的最大公因子有性质：
  - gcd(a, b) = gcd(a-b, b)，如a > b；
  - gcd(a, b) = gcd(a, b-a)，如a < b；
  - gcd(a, b) = a，　　如a = b。

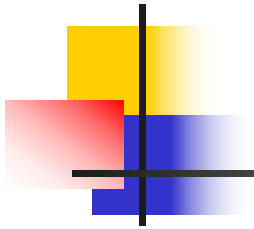
第1解法

```
int gcd(int a, int b)
 {  while( a != b)
       if(a > b) a -= b;
       else b -= a;
    return a;
 }
```



第2解法

- 步骤:
  - [求余数] 求a除b的余数r
  - [判结束] 如r等于0，b为最大公约数
  - [替换] 用b置a，r置b，并回到步骤[求余数]

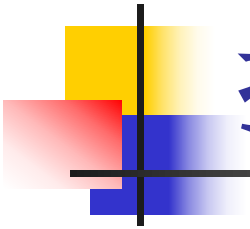


```
int gcd(int a, int b)
 {  int r;
    while(1) {
      if((r = a % b) == 0) break;
      a = b;  b = r;
     }
    return b;
 }
```

```
int gcd(int a, int b)
 {  int r = a;
    do {
        a = b;
        b = r;
        r = a % b;
    } while (r);
    return b;
 }
```

函数定义（续）

- C语言允许在函数名后的圆括号内只给出各形参的名，随后才指定各形参的类型。但这种写法在C++中已不允许
  - double min(x, y)
  - double x, y;
  - {
  - return x < y ? x : y;
  - }
- C语言还允许函数体为空的函数
  - dummy() /* 或 dummy(void) */
  - {
  - }



提要

- 函数基础知识
- 函数定义
- **函数调用**
- 函数形参
- 函数说明
- 递归函数基础
- 命令行参数
- 函数程序设计实例

函数调用

- 函数被定义以后，凡要实现函数功能的地方，就可简单地通过函数调用来完成
- 函数调用的一般形式为
  - 函数名（实在参数表）
- 实在参数，简称实参。函数调用时的实参按它们出现的顺序与函数定义中的形参一一对应，并要求实参类型与其对应的形参类型相一致

函数调用（续）

- 函数调用有两种方式
  - **传值调用（call by value）**
    - 把实参的值传给被调用函数的参数（形参）。这时，被调用函数对参数的改变，不影响调用函数实参的原始值
  - **传引用调用（call by reference）**
    - 把实参的地址传给被调用函数的参数（形参）地址。这时，被调用函数对参数的改变，将影响到调用函数实参的原始值

函数调用（续）

- 对double min(double x, double y)的函数调用 w = min(u, v);
  - 函数调用min(u, v)就是对函数min()的调用，它提供了两个实参u和v，分别对应形参x和y
- 如果调用无形参的函数，这时函数的调用形式变为
  - 函数名();
  - 其中，函数名之后的一对圆括号不能省略

函数调用（续）

- 按函数调用在程序中的作用，有两种不同类型的应用
  - **利用函数所完成的功能**。将函数调用作为一个独立的语句。这种应用不要求或无视函数的返回值
    - 如：程序中经常使用的调用格式输入函数scanf()和格式输出函数printf()等
  - **利用函数的返回值**。或用返回值继续进行表达式的计算，或输出函数返回值等

函数调用的执行过程

1. 为形参分配内存空间
2. 计算实参表达式的值，并将值赋给对应的形参
3. 为函数的局部变量分配内存空间
4. 执行函数体内的语句序列
5. 函数体执行完，或执行了**return**语句后，释放为这次函数调用分配的全部内存空间
6. 将函数值（如果有）返回到函数调用处继续执行

```
#include  <stdio.h>
double x, y, d, min(double, double);
int main() {
  printf("Enter x,y.\n"); scanf("%lf%lf",&x,&y);
  d = min(x, y);
  printf("MIN(%.3f, %.3f) = %.3f\n", x, y, d);
  return 0;
}
double min(double a, double b)
{  double temp;
   temp = a > b ? b : a;
   return temp;
}
```

对函数调用的说明

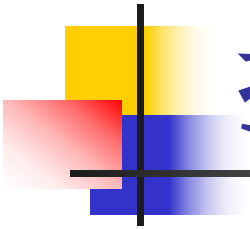
- 当函数执行return语句或执行完函数体的语句序列后，函数的这次调用就执行结束，随之将控制返回到函数调用处继续执行
- 函数的返回值是通过执行return 语句时，计算return之后的表达式值而获得的。如果函数不提供返回值，则return语句不应包含表达式。
- 如果函数有返回值，则应有确定的类型，并在函数定义时指明。若函数定义时不指明返回值类型，且函数有返回值，C语言约定该函数的返回值类型为int型

对函数调用的说明（续）

- 为了明确指明函数不提供返回值，建议在函数定义时，在函数名之前写上**void**。并在这样的函数体内，所有的**return**语句都不应该带表达式
- 当函数执行不带表达式的**return**语句返回时，函数并不是一定不带回值，而是返回一个不确定的值。这样的函数调用不应该利用函数返回值进行计算，否则会产生错误结果
- 函数定义中的**return**语句的表达式类型应与函数定义中指明的返回值类型相一致。如果**return**语句中的表达式类型与函数定义指明的返回值类型不一致时，对于是基本类型情况，则以函数的返回值类型为准，系统会自动进行类型转换

对函数调用的说明（续）

- 在函数未被调用时，函数定义中的形参和函数体中定义的局部变量并不占用存储单元
- 在函数定义中，必须为函数的形参指定数据类型
- 函数体中所使用的形参的初值是由函数调用时对应的实参表达式给定的
- C语言规定，实参表达式对形参的数据传递是“值传递”的，即单向传递
- 对于有多个实参的函数调用情况，C语言不规定实参的求值次序

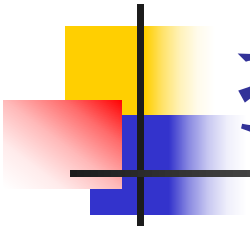


提要

- 函数基础知识
- 函数定义
- 函数调用
- **函数形参**
- 函数说明
- 递归函数基础
- 命令行参数
- 函数程序设计实例

函数形参

- 函数设置形参的目的是让函数被调用时，能从调用处获得信息（数据或指针）
- C语言关于函数形参遵守以下规定
  - 函数调用时，形参从实参获得初值
  - 函数形参可看作函数内部的局部变量，函数体内对形参的修改不会影响实参本身
- 函数形参的作用由形参的类型确定
  - 基本类型、指针类型、数组类型
  - 当函数的形参是某种指针类型或数组类型时，形参从实参处得到某环境变量的指针，函数体就能通过指针形参间接引用环境变量，能改变环境变量的值



提要

- 函数基础知识
- 函数定义
- 函数调用
- 函数形参
- **函数说明**
- 递归函数基础
- 命令行参数
- 函数程序设计实例

函数说明

- 为什么要对函数进行说明？
  - 一个函数要调用另一个函数，应知道有关如何正确调用被调用函数的一些信息
  - 编译程序根据这些信息检查调用的正确性。对不正确的调用给出错误信息；对正确的调用编译出机器代码

调用函数与被调用函数

- 调用函数与被调用函数之间在程序正文中可能会存在以下三种情况
  - 调用同一程序文件中前面已定义的函数
  - 调用处于同一程序文件后面定义的函数
  - 调用别的程序文件中定义的函数
- 对于第一种情况，在函数调用处，被调用函数的详细信息已被编译程序所接受
- 对于后两种情况，因这时被调用函数的信息还未被编译程序所接受，不能检查函数调用的正确性，所以在调用之前需对被调用函数作出说明

调用函数与被调用函数空间关系

```
int f1(…){
…
}

f2(…){
 …
 a=f1(…);
 …
}
```

```
f1(…){
 …
 a=f2(…);
 …

}
int f2(…){
 …
}
```

```
f1(…){
 …
 a=f2(…);
 …
}
```

f1.cpp

```
int f2(…){
 …
}
```

f2.cpp

函数说明形式

- 与调用有关的函数信息包括
  - 函数的返回值类型、函数名和函数有关形参的个数及其类型等
- 只给出函数的调用信息称作函数说明
- 函数说明的一般形式为

  类型　标识符(形参类型表);
  - 其中，形参类型表顺序给出各形参的类型。形参类型表可以为空。为了强调函数没有形参，空形参类型表可以写作void
  - 函数说明对要调用的函数名及其返回值类型和各形参的类型作说明，以便让编译程序预先知道该标识符是函数名、它的返回值类型和它的形参个数及各形参的类型

函数说明示例

```
#include <stdio.h>
int main()
{ double power(double, int), x;
  int n;
  printf("Enter x and n.\n");
  scanf("%lf %d",&x,&n);
  printf("power(%f, %d) = %f\n",
         x, n, power(x, n));
}
```

```
double power (double x, int n)
{/*x^n=x*x^(n-1),当n为奇数;或x^n = (x^2)^(n/2) ,
当n为偶数 */
   double z, y;
   z = 1.0;   y = x;
   while (n>0) {
      while (n%2 == 0) {
        n /= 2;   y *= y;
      }
      n--;  z *= y;
   }
   return z;
}
```

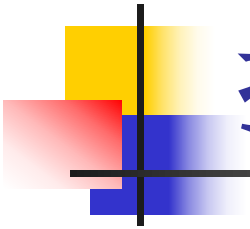
有关函数说明的两种情况-1

- 被调用函数是库函数。程序需使用预处理命令#include将包含有库函数的说明、它们所使用的常量、宏定义、数据类型定义等信息的文件包含进来
  - 习惯称文件名以“.h”为后缀的文件为头文件(header file)
  - 在头文件中通常包含宏定义、数据类型说明和定义、公共变量的外部说明等
  - 在大型C程序设计中，用户程序也会有自己的头文件，如有头文件名为 d_type.h使用包含预处理命令

    #include "d_type.h"

有关函数说明的两种情况-2

- 调用同一程序文件中后面定义的函数，或是调用别的程序文件中定义的函数，应在调用函数之前对被调用函数作函数说明
  - 在C语言中，如被调用函数的返回值是整型或字符型，**也可以不对其返回类型作说明**



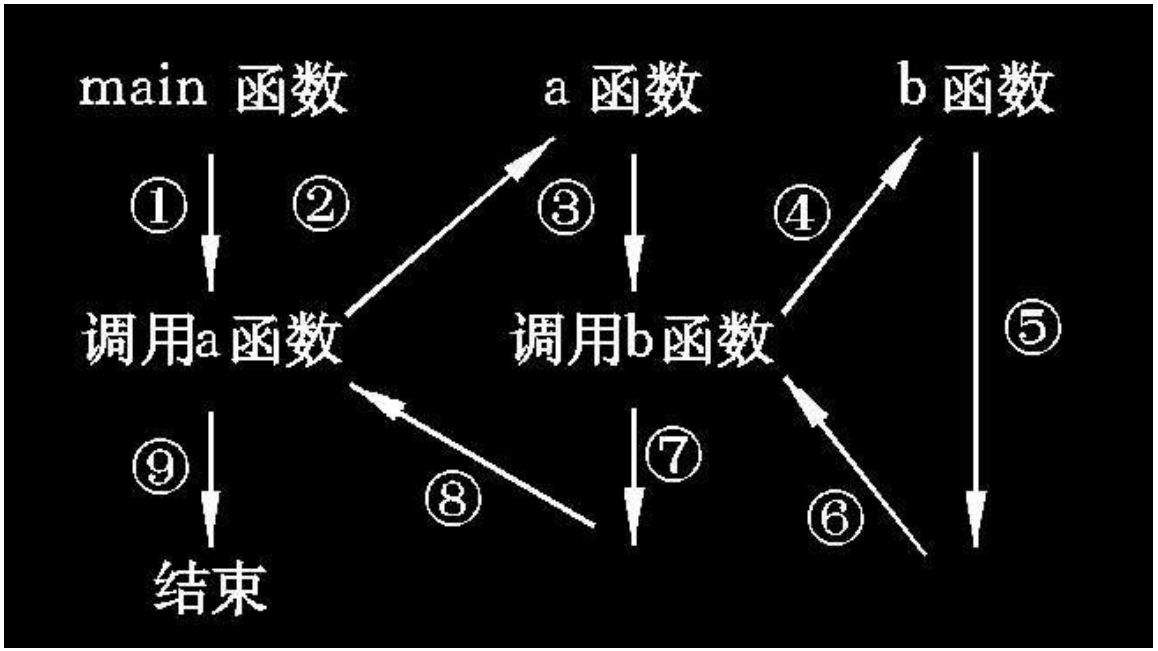
提要

- 函数基础知识
- 函数定义
- 函数调用
- 函数形参
- 函数说明
- **递归函数基础**
- 命令行参数
- 函数程序设计实例

递归函数基础

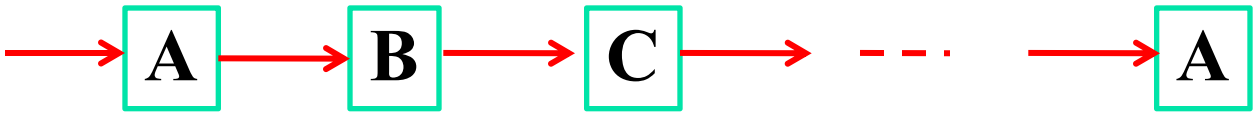
- 一个函数为完成复杂的工作，可以调用其他函数
  - 例如，从主函数出发，主函数调用函数A()，函数A()又调用函数B()，函数B()又调用函数C()，等等。这样从主函数出发，形成一个长长的调用链，就是通常所说的**函数嵌套调用(nested call)**
  - 函数嵌套调用时，有一个重要的特征，**先被调用的函数后返回**。如前面的例子：待函数C()完成计算返回后，B()函数继续计算(可能还要调用其他函数)，待计算完成，返回到函数A()，函数A()计算完成后，才返回到主函数
- **当函数调用链上的某两个函数为同一个函数时，称这种函数调用方式为递归调用(recursive call)**

递归函数基础(2)

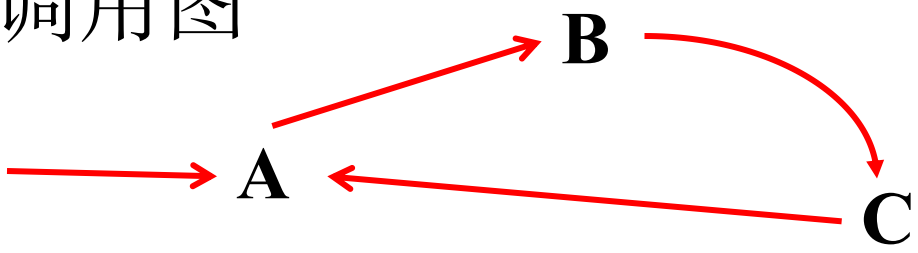


递归函数基础(3)

- 函数调用过程



- 函数调用图



递归的概念

- 递归的定义
  - 若一个对象部分地包含它自己，或用它自己给自己定义，则称这个对象是递归的
  - 若一个函数直接地或间接地调用自己，则称这个函数是递归函数(recursive function)
- 在以下三种情况下，常常用到递归方法
  - 定义是递归的
  - 数据结构是递归的
  - 问题的解法是递归的

递归函数示例-1

阶乘函数:

$$n!=\begin{cases}1, & \text{当 } n=0 \text{ 时}\\ n*(n-1)!, & \text{当 } n\geq 1 \text{ 时}\end{cases}$$

- 用循环实现阶乘计算的函数

```
long fac(int n)
{ long s;  int  i;
  for(s = 1L, i = 1; i <= n; i++)  s *= i;
  return s;
 }
```

- 用递归实现阶乘计算函数

求解阶乘函数的递归算法

```
long fac (int n )
{  if ( n == 0 )  return 1L;
   else return (long)n*fac (n-1);
 }
```

求解阶乘 4! 的过程


参数 计算 返回
0 0! = 1 1
参数 计算 返回
1 1*Factorial(0) 1
参数 计算 返回
2 2*Factorial(1) 2
参数 计算 返回
3 3*Factorial(2) 6
参数 计算 返回
4 4*Factorial(3) 24
主程序 main
0
1
2
3
4
参数传递
1
1
2
6
24
返回值

递归函数示例-2

- 用递归函数实现数组元素的求和计算

```
int rsum(int *a, int n)
{
  if (n == 0) return 0; /* 若数组没有元素，则返回0 */
  return *a+rsum(a+1, n-1); /* 当前元素与其余元素的和 */
 }
```

递归函数示例-3

- 求解Hanoi塔问题。问题是这样的，有三根针(设分别为A、B、C)，在A针上有n张大小均不相同的金片，大的在下，小的在上。要求按以下规则，把这n张金片从A针搬到C针上
  - 在搬动过程中可使用B针；
  - 每次只允许搬动一张金片；
  - 在搬动过程中，必须保证大金片在下，小金片在上。

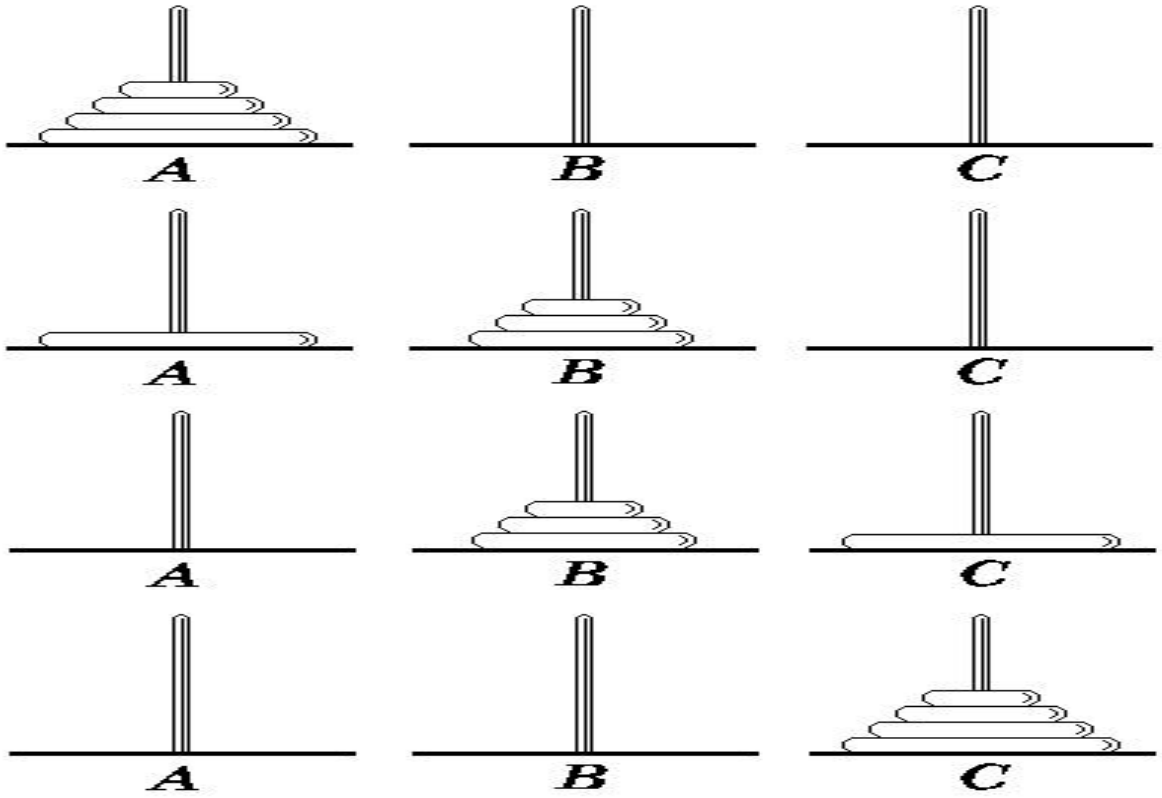
递归函数示例-3(续)

- Hanoi塔问题(续)
- 程序设计分析

```
move(int n, char A, char C, char B)
{ /* 将 n 张金片从 A 针搬到 C 针，搬动过程可使用 B 针 */
  if (n==1) /* 对于只搬动一张金片情况 */
  (1) 将金片 n 从 A 针搬到 C 针;
  else
   { /* 搬 n(>1) 张金片时，按同样算法先搬前 n-1 张金片 */
    (2) 将A 针上的 n-1 张金片搬到 B 针上，中间可使用 C 针;
    (3) 将金片 n 从 A 针搬到 C 针; /* 搬动第 n 张金片 */
     /* 搬 B 针上前 n-1 张金片 */
    (4) 将 B 针上的n-1张金片搬到C针，中间可使用A针;
   }
}
```

递归函数示例-3(续)

Hanoi问题的解法是递归的，是直接递归



递归函数示例-3(续)

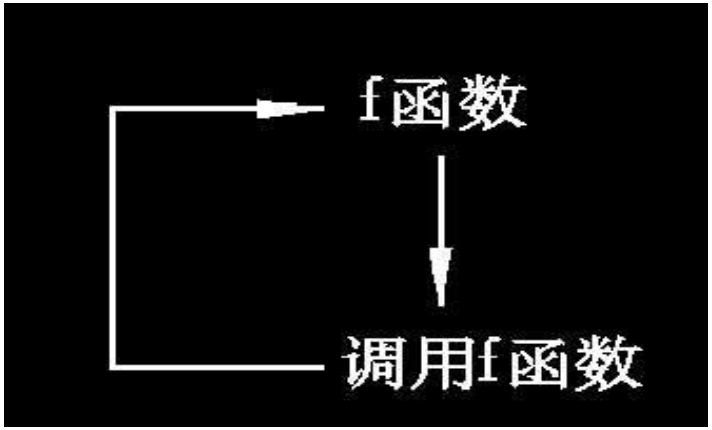
- Hanoi塔问题(续)

```
#include <stdio.h>
void hanoi (int n,char A,char C,char B)
{  //解决汉诺塔问题的算法
  if ( n == 1 )
      printf( " move %c  to %c\n ",A, C);
  else {
      hanoi ( n-1, A, B, C );
      printf( " move %c  to %c\n ",A, C);
      hanoi ( n-1, B, C, A );
    }
}
```

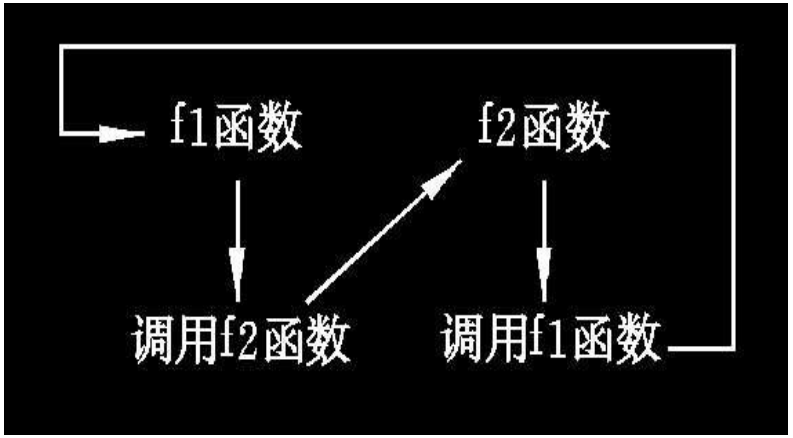
递归函数(续)

- 对于算法间接地由其自身定义情况，称为间接递归。在间接递归情况下，函数调用链中，在同一个函数再次出现之前，有一个或多个其它别的函数
- 递归程序特点
  - 通常，以递归形式定义算法与非递归算法比较，算法结构会更紧凑、清晰
  - 但是，递归函数在递归调用过程中，会占用更多的内存空间和需更多的运行时间
  - 另外，在编写递归函数时，必须防止遗漏递归终止条件和防止振荡式的相互递归调用，否则会产生无限递归调用现象

直接递归 vs. 间接递归



直接递归



间接递归

用递推实现递归的示意程序

- 勒让得多项式的递归定义为

$$p(n,x)=\begin{cases}1, & n=0;\\ x, & n=1;\\ ((2n-1)xp(n-1,x)-(n-1)p(n-2,x))/n, & n>1\end{cases}$$

- 程序设计分析
  - 用递归函数实现，能很容易地写出它的函数定义如下

```
double p(int n, double x)
{  if (n==0) return 1.0;
   if (n==1) return x;
   return ((2.0*n-1.0)*x*p(n-1,x)-(n-1.0)*p(n-2,x))/n;
 }
```

用递推实现递归的示意程序(续)

- 勒让得多项式用递推方法实现
- 程序设计分析
  - 计算p(n,x)的困难是n>1情况
  - 然而，若知道了 p(n-2,x) 和 p(n-1,x) 的话，就能立即按公式计算出 p(n，x)
  - 由于p(0,x)与p(1,x)是能立即得到的
  - 这样就能用递推方法，从p(0,x)与p(1,x)出发，依次计算出p(2,x)，p(3,x)，…
  - 即从幂次低的勒让得多项式到幂次高的勒让得多项式的递推解法

用递推实现递归的示意程序(续)

- 勒让得多项式用递推方法实现，函数定义如下

```
double p(int n, double x)
{ double first, second, third;   int count;
  if (n == 0)  return 1.0;
  if (n == 1)  return x;
  first  = 1.0;  second = x;
  for(count = 2; count <= n; count++)  {
    third  = ((2.0*count-1.0)*x*second - (count-1.0)*first)/count;
    first  = second;   second = third;
   }
  return  third;
}
```

递推(recurrence)vs.递归(recursion)

- 共同点
  - 把复杂问题转化为简单问题进行处理
- 不同点
  - 递推：程序员找到递推关系，程序从最简单问题开始，递推得到复杂问题解
  - 递归：程序员找到递归关系，程序把复杂问题逐步归结为简单处理（编程者完成），再从最简单问题开始，递推得到复杂问题解

递归实例(1)

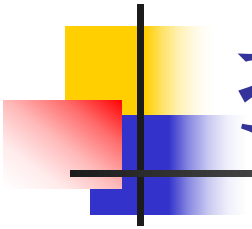
- 递归计算x的y次方

```
int power(int x, int y)
{ if(y==0) return 1;
  return x*power(x, y-1):
}
```

递归实例(2)

- 输入一个正整数，用递归将该整数倒序输出

```
#include<stdio.h>
void reverse(int n)
{ printf("%d", n%10);
   if (n<10) return;
   reverse(n/10);
}
```



提要

- 函数基础知识
- 函数定义
- 函数调用
- 函数形参
- 函数说明
- 递归函数基础
- **命令行参数**
- 函数程序设计实例

命令行参数

- 执行C程序，可以看作是对该程序的main()函数的调用。main()函数执行结束，返回环境。为能从环境向C程序传递信息，启动C程序的命令行可带有任选的参数
- 命令行的一般形式为

  程序名　参数1　参数2　…　参数n

  - 其中程序名和各参数之间用空白符分隔
  - 为能让main()函数读取命令行中的参数，环境将多个参数以两个参数形式传递给main() 函数。其中第一个参数(习惯记作argc)表示命令行中参数的个数(包括程序名)；第二个参数(习惯记作argv)是一个字符指针数组。其中argv[0]指向程序名字符串，argv[1]指向参数1字符串，…，argv[argc-1]指向最后一个参数字符串。
  - 如果argc等于1，则表示程序名后面没有参数

命令行参数示例

- 说明main()函数对参数argc与argv引用方法的程序
  - 打印启动程序时的命令行各参数

```
#include <stdio.h>
int main(int argc, char *argv[] /* 或 char **argv; */ )
{ int k;
  for(k = 1; k < argc; k++)
     printf("%s  %c", argv[k], k < argc-1 ? ' ' : '\n');
  printf("\n\n");
 }
```

  - 如上述程序的执行程序名为 aecho.exe， 执行该程序的命令行为：aecho Hello world!
  - 则程序将输出 Hello world!
  - 根据约定，main()函数的参数argc的值为3；argv[0]，argv[1]，argv[2]分别指向字符串"echo"、"Hello"、"world!"的第一个字符

命令行参数(续)

- 在程序的printf()函数调用中，字符转换格式%c输出一个字符，若是已输出了命令行最后一个参数，该格式将输出一个换行符，若是输出其他参数，则输出一个空白符
- 因函数的数组参数是指向数组首元素的指针变量，所以在主函数 main()中可对argv施行增量运算
  - 例如，在argv[0]指针指向程序名字符串的第一个字符情况下，对argv施增量运算++argv后，argv[0](或*argv)就指向参数1的第一个字符

命令行参数(续)

- 利用在主函数 main()中可对argv施行增量运算的性质，可改写前述示例程序为以下形式

```
#include <stdio.h>
int main(int argc, char **argv)
{  while (--argc > 0)
       printf("%s %c", *++argv,argc > 1 ? ' ' : '\n');
 }
```

  - 这里，++argv使指针argv先加1，让它一开始就指向参数1；逐次增1，使它遍历指向各参数
  - 又利用函数printf()的第一个格式参数是字符串表达式，上述程序对printf()的调用可改写成

```
printf((argc > 1)? "%s   " : "%s\n",*++argv);
```

命令行参数示例-1

- 假定启动程序时给出的命令行参数是一串整数，程序将全部整数求和后输出

```
#include  <stdio.h>
#include  <math.h>
int main(int argc, char **argv)
{  int k, s;
   for(s = 0, k = 1; k < argc; k++)
        s += atoi(*++argv); /* 从数字字符串译出整数 */
   printf("\t%d\n", s);
   return 0;
 }
```

命令行参数(续)

- 利用指针数组作main()函数的形参，向程序传递命令行参数字符串，其中字符串的长度可以不同，参数的个数也可以任意，这是指针数组作函数参数最为方便灵活的应用
- 命令行可带任选的参数，程序(主要main()函数）应能正确识别某参数的出现，及该参数的意义。通常，C程序将命令行参数分成两部分
  - 一部分是一定要出现的参数，它们通常被安排在命令行参数表的最后部分，并对这些参数出现顺序也有明确的约定
  - 另一部分是可任选的参数，因它们中的某些或全部可以不出现，为使程序能正确识别出现的参数及其意义，C程序习惯用以下形式表示一个可任选的参数

    -字符 参数

命令行参数(续)

- C程序习惯用以下形式表示一个可任选的参数

  -字符　参数

  - 其中负号表示一个任选参数，紧接的字符是任选参数的标志符。
  - 其后的参数可以没有。如有，就作为命令行的下一个参数。
  - 特别是对于标志符之后的参数没有的情况，多个参数标志符紧接在一起，合用一个任选参数开始符(负号字符)

命令行参数程序示例

- 输入正文，从中寻找包含特定字符串的行。其中特定字符串由命令行的参数指定。程序另设两个供任选的参数
  - 参数1　-x印出所有那些不包含特定字符串的行；无-x，则印出所有那些包含特定字符串的行
  - 参数2　-n在输出字符行之前，冠以该行在正文中的行号；否则，输出时不带行号
  - 另约定，特定字符串参数是命令行的最后一个参数；任选参数的出现顺序可以任意。如两个任选参数都出现时，可合在一起同时指定，如-xn或-nx。一个任选参数最多只能出现一次。
  - 运行本程序的命令行的一般形式为

```
find -x -n  pattern  filename
```

命令行参数程序示例(续)

```
#define LINES   500
#define MAXLINE 300
#include <stdio.h>
#include <string.h>
#include <malloc.h>
int getline(FILE *fp, char *s, int lim); /* 从文件读出一个串到s,
返回串的长度*/
int index(char *s, char *t); ) /* 判断t是否出现在s中，并确定出现
位置*/
int main(int argc, char **argv)
{  char *lineptr[LINES];    int   lineno[LINES];    int   len, m;
   int   nlines  = 0;    int   n = 0;    int   err = 0;
   int   inverse = 0;    int   num     = 0;
   FILE *fp;
   char line[MAXLINE], *s;
```

命令行参数程序示例(续)

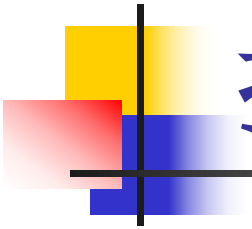
```
while (err == 0 && --argc > 0 && (*++argv)[0] == '-')
 /* *++argv是指向任选参数串的首字符指针，(*++argv)[0]是任选参数字符串的首字符*/
    for(s = argv[0]+1; *s != '\0'; s++)
        switch (*s) {
           case 'x' :  if (inverse) err = 1;  /*限制x最多只能出现1次*/
                       else inverse = 1;/* inverse=1表示x出现*/
                       break;
           case 'n' :  if (num) err = 1; /*  限制n最多只能出现1次*/
                       else num = 1; /* num=1表示n出现*/
                       break;
           default  :  err = 1;
        }
/*正确情况下，至此argc应为2*/
if (err || argc != 2) {  /*如有使用不当*/
        printf("Usage:find -x -n pattern filename\n");    return 1;
 }
```

命令行参数程序示例(续)

```
if ((fp = fopen(*(argv+1), "r")) == NULL) {
    fprintf(stderr, "Can't open %s. \n", *(argv+1));    return 2;
 }
while ((len = getline(fp, line, MAXLINE)) > 0)  {
    nlines++; /*记录行数*/
    if ((index(line, *argv) >= 0) != inverse)
       if ((s = (char *)malloc(len+1)) == NULL)  break;
       else {  lineno[n] = nlines;   strcpy(s, line);   lineptr[n++] = s;
        }
 }
fclose(fp);
for(m = 0; m < n; m++)  {  if (num)   printf("%4d: ", lineno[m]);
       printf("%s\n", lineptr[m]);
  }
return 0;
}
```

命令行参数程序示例(续)

```
int getline(FILE *fp, char *s, int lim)
/* 从文件读出一个串到s，返回串的长度*/
{ int c;    char   *ps = s;
  while (ps < s+lim-1 && (c = fgetc(fp)) != EOF) {
     *ps++ = c;   if (c == '\n') break;
   }
  *ps = '\0';   return ps-s;
}
int index(char *s, char *t) /* 判断t是否出现在s中，并确定出现位置*/
{ int i, len1 = strlen(s), len2 = strlen(t);   char *ps, *pt;
  for(i = 0; i <= len1-len2; i++) {
     for(ps = s+i, pt = t; *pt != '\0' && *ps == *pt; ps++, pt++);
     if (*pt == '\0')   return i;
   }
  return -1;
}
```



提要

- 函数基础知识
- 函数定义
- 函数调用
- 函数形参
- 函数说明
- 递归函数基础
- 命令行参数
- **函数程序设计实例**

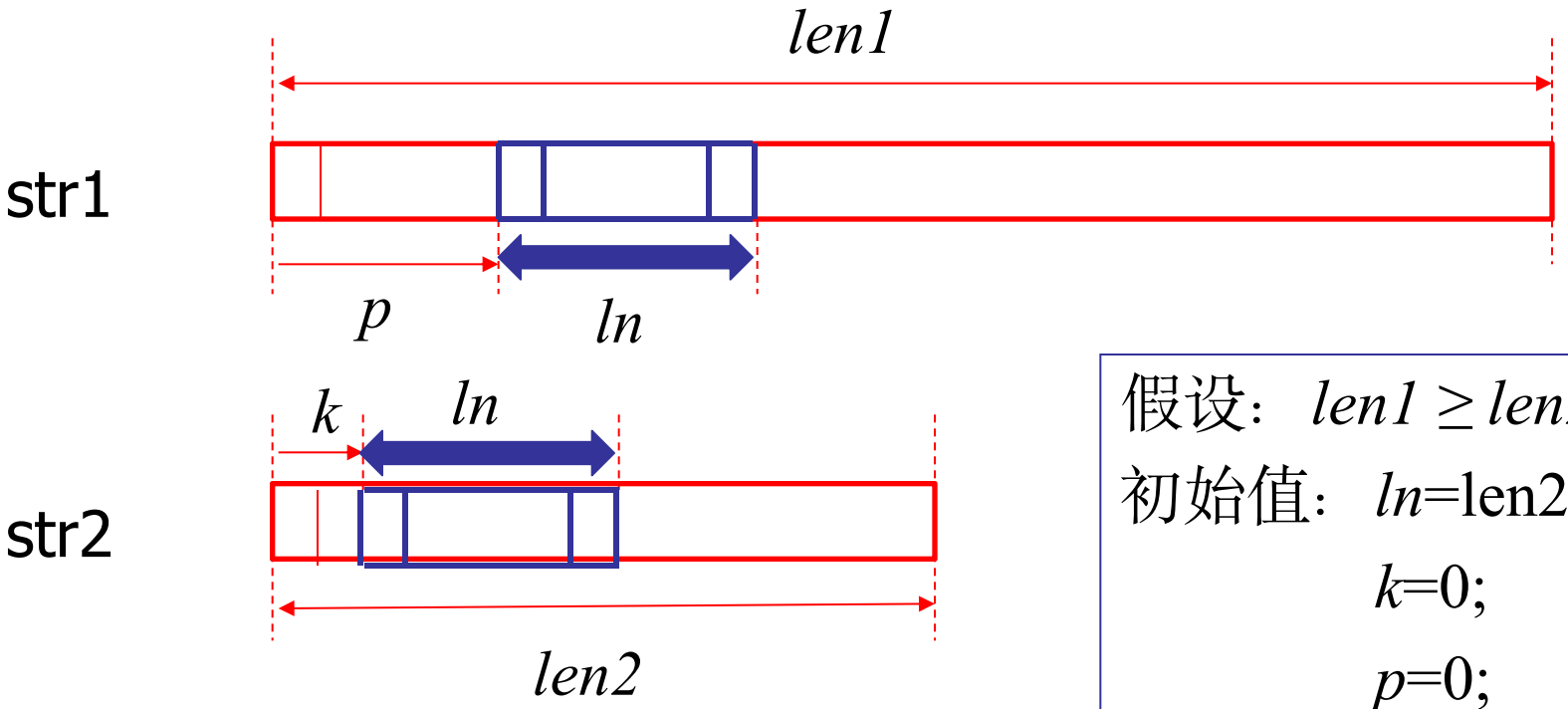
函数程序设计实例-1

- 编制求整数是几位十进位数的函数
  - 设函数模型为int digit(int n)，函数的功能是求形参n是几位十进位数。引入变量c计数n的位数，则反复将n除以10，直至n等于0的重复次数，就能推算出n的十进位的位数
  - 函数开始时预置计数器c为0，循环的工作部分是让c增1和n除以10，循环直至n除以10后为0结束
  - 如下面的函数digits()的定义

```
int digits(int n)
{  int c = 0;
   do {   c++;   n /= 10;
    } while (n);
   return c;
 }
```

函数程序设计实例-2

- 编制从两个已知非空字符串中找出最长公共子串的长度和最长公共子串的个数的函数
  - 设函数模型为　int commStr(char str1[], char str2[], int lenpt[])
  - 函数的形参str1和str2是已知字符串首字符的指针，形参lenpt用于存储找到的最长子串的长度，函数返回找到的最长公共子串的个数
  - 函数算法的基本想法是对字符串s2的各种可能子串，在字符串s1中找是否有同样的子串
  - 记两个字符串str1和str2的长度分别为len1和len2。假定len1 >= len2。则它们最长的公共子串长度不会超过len2。设要寻找的子串长为ln，ln的初值为len2，即以str2字符串的最长子串开始寻找，当在str1中没有长为ln的相同子串时，就让ln减1，即用str2更短的子串重新寻找。若找到，则该子串就可作为函数要找的str1和str2的一个最长子串。若直至找遍了str2的所有可能的子串，还未在str1中找到相同的子串，则说明str1和str2没有公共子串



假设：$len1 \geq len2$;

初始值：ln=len2;

k=0;

p=0;

函数程序设计实例-2(续)

- 编制从两个已知非空字符串中找出最长公共子串的的长度和最长公共子串的个数的函数(续)

```
#include <stdio.h>
int commStr(char str1[], char str2[], int lenpt[])
/*函数返回最长公共子串的个数, 通过指针参数传回最长公共子串的长度*/
{  int len1, len2, ln, count, i, k, p;   char *st;
   if((len1=strlen(str1))<(len2=strlen(str2))){  /* 把str2作为短的串*/
         st = str1;  str1 = str2;    str2 = st;
         ln = len1;  len1 = len2;  len2 = ln;
    }
   count = 0;
```

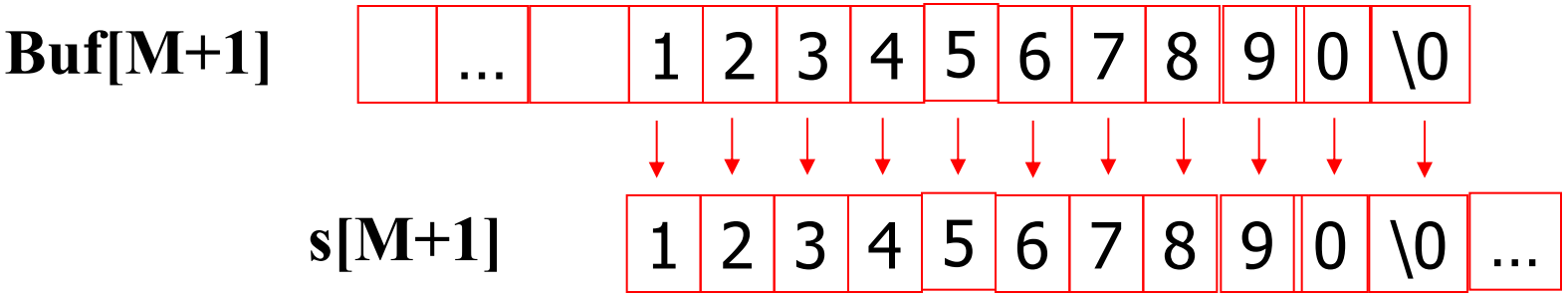
函数程序设计实例-2(续)

```
for(ln=len2;ln>0;ln--) {  /* 找长ln的公共子串 */
    for(k = 0; k + ln <= len2; k++) {
        /* 自str2[k]开始，长为ln子串与str1的子串比较 */
        for(p = 0; p + ln <= len1; p++) { /* str1中的子串自str1[p]开始,
两子串对应字符逐一比较 */
            for(i = 0; i < ln; i++)    if (str2[k+i] != str1[p+i]) break;
            if (i == ln)    count++;   /* 找到一个最长公共子串 */
        }
    }
    if (count) break;  /* 如找到过，退出寻找循环 */
}
lenpt[0] = ln;    return count;
}
int main()
{  int c, len[2];   c=commStr("Abc1AbcsAbcd123", "123bAbc",
len);
   printf("有 %d 个长为 %d 的公共子串\n", c, len);
}
```

函数程序设计实例-3

- 编写函数int trans(unsigned n, int d, char s[])，将无符号整数n翻译成d(2 <= d <= 16)进制的字符串s。函数的返回值是该d 进制的字符串长度。当d>10时，d进制的数字位值可能要大于9，约定大于9的数字位值分别用字符A、B、C、D、E、F标记数字位值10、11、12、13、14、15
  - 求n整除d的余数，就能得到n的d进制数的最低位数字，然后n/=d
  - 重复上述步骤，直至n为0，能依次得到n的d进制数的最低位数字至最高位数字
  - 再由各位数字取出相应字符，就得到n的d进制数的字符串表示

一个长整数：1234567890



函数程序设计实例-3(续)

- 编写函数int trans(unsigned n, int d, char s[])(续)

```
#include  <stdio.h>
#define  M sizeof(unsigned int)*8
int trans(unsigned n, int d, char s[])
{  char digits[] = "0123456789ABCDEF";
   char buf[M+1];    int j, k = M;
   if (d < 2 || d > 16) {
      s[0] = '\0';    return 0;
    }
   buf[k] = '\0';
   do {  buf[--k] = digits[n%d];    n /= d;
    } while (n);
   for(j = 0; s[j++] = buf[k++]; );
   return j-1;
}
```

函数程序设计实例-4

- 编制判定正整数n的d进制表示形式是否是回文数的函数。回文数就是自左向右读和自右向左读是相同的数。例如，n = 232,其十进制表示是回文数；n = 27，其二进制表示11011是回文数
  - 设函数circle(int n, int d)是要判n的d进制表示是否是回文数。实现这个判定有两种方法:
    - 一是顺序译出n的d进制表示的各位数字，然后将其首末对应位数字两两比较，若对应位数字都相同，则n的d进制表示是回文数。
    - 二是在顺序从低位到高位译出n的d进制各数字位的同时，把译出的各位数字看作是另一个d进制数字的高位至低位，并将其重新转换成整数。若转换所得整数与n相等，则n的d进制表示是回文数

函数程序设计实例-4(续)

- 判n的d进制表示是否回文数函数circle(int n, int d)(续)

下面是参照第二种方法编写的函数

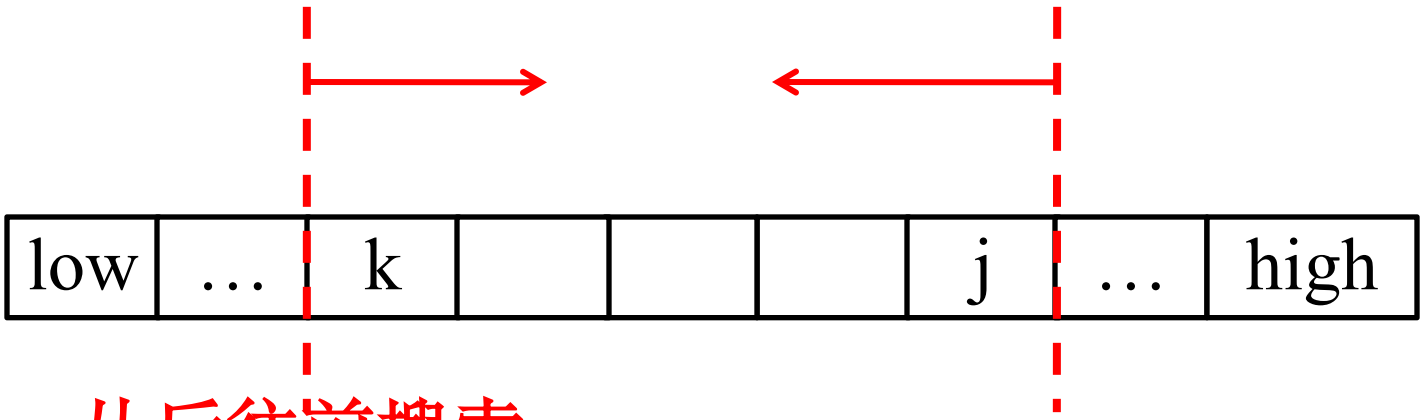
```
int circle(int n, int d)
{  int s = 0, m = n;
   while (m) {
     s = s*d + m%d;    m /= d;
    }
   return s == n;
}
```

函数程序设计实例-5

- **编制函数** int part(int a[], int low, int high)
- 对数组元素a[low] ~ a[high]以a[low]为基准，对数据作划分，使划分后新的a[low] -- a[k-1]中的元素均小于a[k]；a[k+1] -- a[high]中的元素均大于等于a[k]，函数返回k值
  - 令函数要调整存储位置的元素是a[k]至a[j]，变量k和j就是未确定存储位置元素区间的下界和上界，它们的初值分别为low和high
  - 首先将a[low]暂存于变量temp，然后是一个循环，循环内是交替地执行从后向前考察元素的循环和从前向后考察元素的循环

函数程序设计实例-5

- temp=a[low]; 初始化： k=low; j=high



- 从后往前搜索
  - If a[j] ≥temp, j--; otherwise, a[k]和a[j]互换； k++
- 从前往后搜索
  - If a[k]<temp, k++; otherwise, a[k]和a[j]互换； j--

函数程序设计实例-5(续)

- 编制函数 int part(int a[], int low, int high)(续)
  - 在从后向前考察循环中，当考察元素a[j]不比temp小时，让j减1。当发现元素a[j]比temp小时，结束从后向前的循环，并将a[j]移至a[k]，和让k增1
  - 在从前向后考察循环中，当考察元素a[k]比temp小时，让k增1。当发现a[k]不比temp小时，结束从前向后的考察循环，并将a[k]移至a[j]，和让j减1
  - 上述循环直至调整区间不再有元素时结束。最后将暂存于temp中的值回填到a[k]中，返回k值

函数程序设计实例-5(续)

- 编制函数 int part(int a[], int low, int high)(续)

```
int part(int a[], int low, int high)
{   int k = low, j = high, temp = a[low];
    do {
       while ( j > k && a[j] >= temp )   j--;
       if (k < j)   a[ k++ ] = a[j];
       while ( k < j && a[k] < temp )   k++;
       if (k < j)   a[ j-- ] = a[k];
     } while ( k < j);
    a[k] = temp;
    return k;
 }
```

函数程序设计实例-5(续)

- 编制函数 int part(int a[], int low, int high)(续)
  - 以下是利用函数part()实现求解数组a中第k个元素(自0开始计数）的函数int kth()和示意主函数

```
int kth(int a[], int k, int n)
 {  int m = 0, h = n-1, t;
    do {
       t = part(a, m, h);
       if ( t > k )  h = t - 1 ;
       else
          if ( t < k ) m = t + 1 ;
     } while ( t != k );
    return a[k];
  }
```

```
int a[] = {0, 1, 9, 7, 6, 4, 5, 2, 3, 8};
void main()
{  int i;
   printf("Enter i  "); scanf("%d", &i);
   printf("%d  %d\n", i, kth(a, i, 10));
 }
```

第8讲 小结

- 函数基础知识
- 函数定义
- 函数调用
- 函数形参(指针类型、数组类型、字符指针)
- 函数说明
- 递归函数基础(直接递归、间接递归、线性递归、递推)
- 命令行参数
- 函数程序设计实例