Xidian University

C语言程序设计

Lec 4 函数

主要内容

- 函数定义
- 函数调用
- 变量作用域
- 函数与递归
- 函数与数组（结合数组讲）
- C语言常用函数

引言

- 例：求一些圆盘的面积，圆盘半径分别为：
  3. 24、2. 13、0. 865、3. 746、12. 3364、8. 421

```
//设圆周率为 3.1416，可写出下面程序：

#include <stdio.h>
int main () {
   printf("radius:%f, area:%f\n", 3.24, 3.24 * 3.24 * 3.1416);
   printf("radius:%f, area:%f\n", 2.13, 2.13 * 2.12 * 3.1415);
   …
}
```

繁琐的东西很容易弄错，不易修改

标准函数有限，需求无限。

引言

- 如果有求圆面积的函数double c_area(double r)

```
int main () {
   printf("radius:%f, area:%f\n", 3.24, c_area(3.24) );
   printf("radius:%f, area:%f\n", 2.13, c_area(2.13) );
   …
}
```

- 如果有打印圆面积的函数 pc_area(double r)

```
int main () {
   printArea(3.24);
   printArea (2.13);
   …
}
```

函数能使程序变短，

变得易写/易理解/易修改

协作大程序

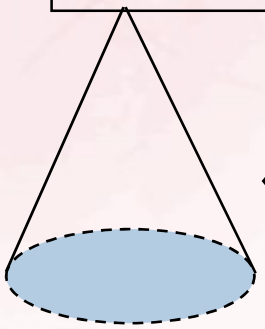
4.1 函数定义

函数定义

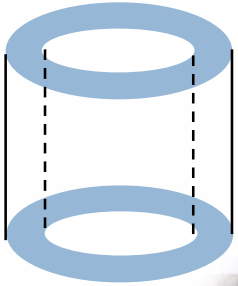
- 把一段计算定义成函数并给以命名，定义后就可以在任何需要的地方通过名字调用。

定义函数 c_area 的程序片段：

```
double c_area (double r) {
    return r * r * 3.1416;
}
```



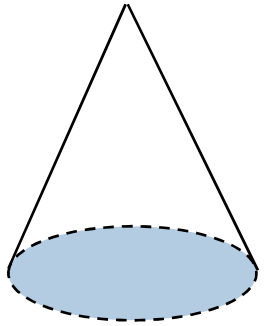
如何求这些与圆有关的物体的面积？



函数定义

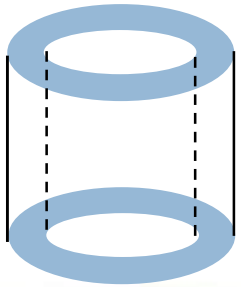
- 半径3. 24高2. 4的圆锥体积：

  2.4 * c_area(3.24) / 3.0



- 外半径5. 3，内半径3. 07，高4. 2的空心圆柱体积：

  (c_area(5.3) - c_area(3.07))*4.2



定义函数的要素

demo_4_circle.c

- 函数头
  - 函数名：使用函数需要的名称，合法标识符
  - 返回值类型—函数计算结果的类型
  - 参数表—完成计算需要的数据（数量和类型）
- 函数体

实现函数功能的代码，由一对大括号包围

```
返回值类型  函数名     参数表
    ↓        ↓         ↓
  double   c_area  ( double r )
```

函数头

```
{
    return r * r * 3.1416;
}
```

函数体

函数返回值

demo_4_circle.c

- 一个函数最多只能有一个返回值，返回值通常是计算结果或者表示计算状态的信息，由调用者使用
- 如果函数有返回值函数必须指定返回值类型，如果函数不需要返回值必须使用void作为函数返回值类型。
- 函数返回值通过return语句返回，return语句一旦执行，整个函数就结束。

函数返回值

demo_4_circle.c

- 一个函数中可以有多条return语句，但只会执行其中一条。
- return语句形式：return 表达式;
- return语句中表达式求值的类型应该和函数返回值类型一致，如果不一致会自动进行类型转换
- 可以 return一个变量

函数定义示例

```
double c_area (double r) {
    return r * r * 3.1416;
}
```

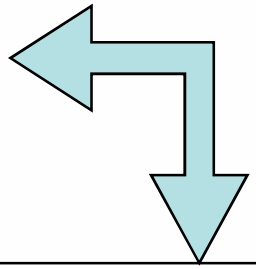
```
void pc_area(double r){
    printf("r = %f, S = %f\n", r, 3.14159265 * r * r);
}
```

```
int max(int a, int b){
    if(a>b)  return a;
    return b;
}
```

```
int compare( int x, int y ){
  if( x == y )    return 0;
  else if( x > y )    return 1;
  else    return -1;
}
```

关于return语句

```
double c_area (double r) {
    return r * r * 3.1416;
}
int main(){
    double v=2.4 * c_area( 3.24 ) / 3.0;
    printf("v=%f\n",v);
}
```



return语句需要先计算后面的表达式，将其结果保存在临时变量中，然后用该临时变量参与表达式运算，运算完成后立刻释放

```
int main(){
    double s = c_area( 3.24 );
    double v=2.4 * s / 3.0;
    printf("v=%f\n",v);
}
```

函数参数表

- 函数可以有0个或多个参数，这些参数称为形式参数
- 每个参数必须指明类型和参数名称
- 函数参数是函数内的局部变量，只在函数体内有效
- 函数参数只有在函数被调用时才有效
- 函数参数的初始值由调用者传入（通过实际参数以值拷贝的方式传入）

形参和实参

- 形参：在函数定义中括号内的标识符，与函数调用时的实参一一对应
- 实参：在调用函数的括号中使用的表达式，它的值被传入函数并赋值给函数的对应形参。

```
#include <stdio.h>
#include <math.h>
//定义函数
double c_area (double r) {
    return pow( r , 2 ) * 3.1416;
}
int main () {
   double v;
   //调用函数
   v=2.4 * c_area( 3.24 ) / 3.0;
   return 0;
}
```

形参

实参

实参

实参

函数定义不能嵌套

```
#include <stdio.h>
#include <math.h>

double c_area (double r) {
    return pow(r, 2) * 3.1416;
}

int main () {
   double v;
   v=2.4 * c_area( 3.24 ) / 3.0;
   return 0;
}
```

```
#include <stdio.h>
#include <math.h>

int main () {
   double c_area (double r) {
      return pow(r, 2) * 3.1416;
   }
   double v;
   v=2.4 * c_area( 3.24 ) / 3.0;

   return 0;
}
```

4.2 函数调用

调用系统函数

- 包含必要的头文件，其本质是将函数原型添加到程序中
- 在需要的地方使用函数，传入类型和数量正确的实际参数，函数返回值可以作为表达式的一部分

函数原型就是函数头部加上分号，其作用是告诉编译器函数应该以什么形式调用

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(){
    double sum=0;
    int n=1;
    while(n<=100){
        sum=sum+ sin(1.0/n) ;
        n=n+1;
    }
    printf("sum=%f\n",sum);
    system("pause");
    return 0;
}
```

调用自定义函数

- 方法1（函数定义放在调用函数之前）：
  - 在需要的地方使用函数，传入类型和数量正确的实际参数，函数返回值可以作为表达式的一部分

```
#include <stdio.h>

double c_area (double r) {
    return r * r * 3.1416;
}

int main () {
   double v;
   printf("radius:%f, area:%f\n",
          3.24,
          c_area(3.24) );
   v=2.4 * c_area( 3.24 ) / 3.0;

   return 0;
}
```

调用自定义函数

- 方法2（函数定义放在调用函数之后）：
  - 在函数调用之前给出函数原型
  - 在需要的地方使用函数，传入类型和数量正确的实际参数，函数返回值可以作为表达式的一部分

demo_4_funcdefine

```
#include <stdio.h>
//函数原型
double c_area (double r);

int main () {
   double v;
   printf("radius:%f, area:%f\n",
          3.24,
          c_area(3.24) );
   v=2.4 * c_area( 3.24 ) / 3.0;

   return 0;
}
double c_area (double r) {
    return r * r * 3.1416;
}
```

函数调用的若干问题

- C语言是一个函数式语言，所有可执行语句都必须放在某个函数体内
- 调用函数的函数称为主调函数，被调用的函数称为被调函数
- 当函数调用发生时，主调函数暂停，程序控制转入被调函数，被调函数执行结束后，主调函数继续

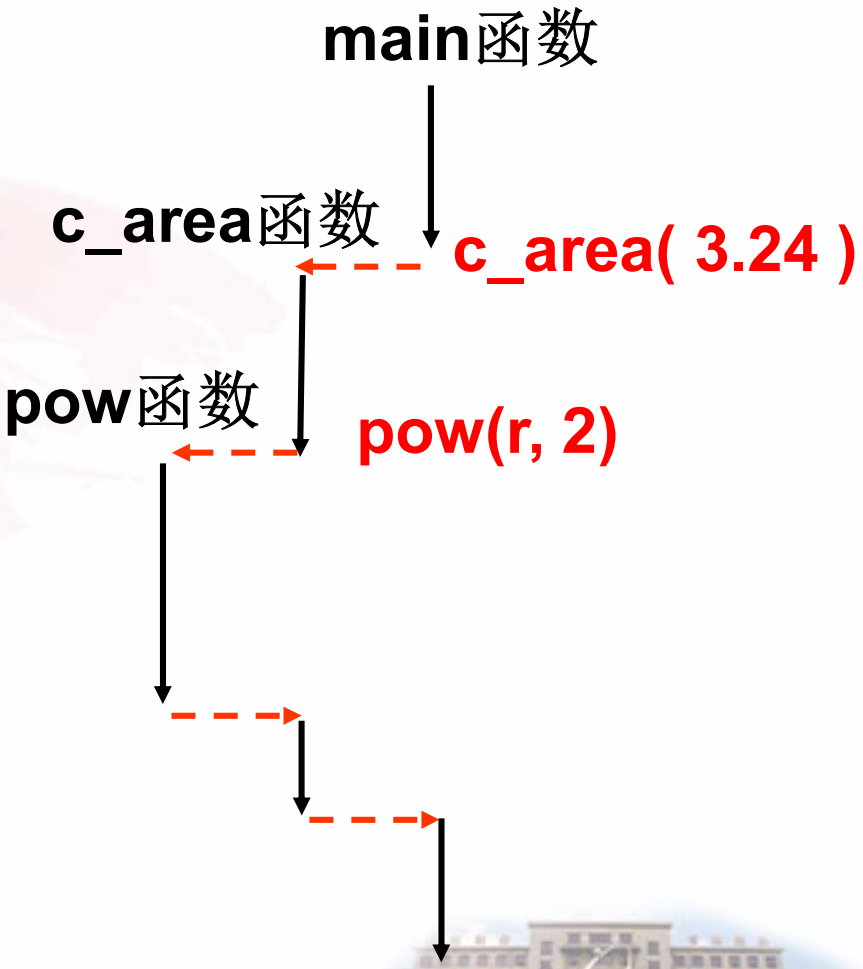
函数调用的若干问题

demo_4_funcdefine

```
#include <stdio.h>
#include <math.h>

double c_area (double r) {
    return pow(r, 2) * 3.1416;
}

int main () {
   double v;
   v=2.4 * c_area( 3.24 ) / 3.0;
   return 0;
}
```



参数传递机制

```
double c_area (double r) {
    return pow(r, 2) * 3.1416;
}
```

- 形式参数在函数调用时才分配存储空间，并接受实际参数的值
- 实际参数可以为复杂的表达式，在函数调用前获得计算
- 形式参数与实际参数可同名，也可不同名

```
c_area(3.3);
c_area(2*4*a);
c_area(a);
。 。 。
double r=3.3;
c_area(r);
```

参数传递机制

- 参数较多时，实际参数值逐一赋值，它们必须保持数目、类型、顺序的一致
  - 参数的赋值过程单向不可逆，函数内部对形式参数值的修改不会反映到实际参数中
  - 函数参数一般为函数输入集的一部分，函数输出集一般使用返回值表示，只有使用特殊的手段(指针/数组)才可以将函数参数作为函数输出集的一部分

```
int compare( int x, int y )
```

```
int a=1,b=5;
compare(a,b);
```

demo_4_swap.c

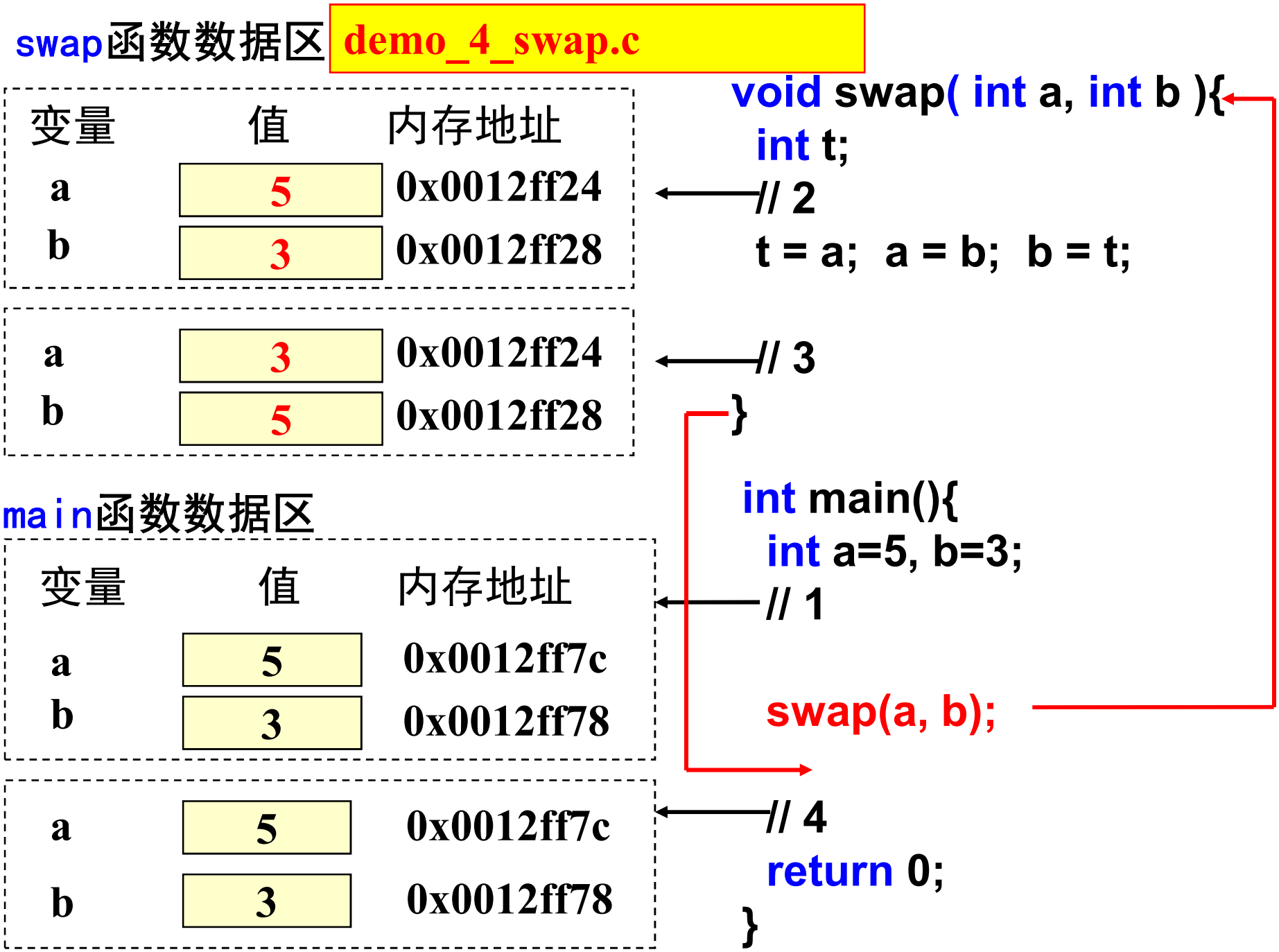
参数传递机制

```
void swap( int a, int b ){
  int t;
  t = a;  a = b;  b = t;
}
int main(){
  int a=5, b=3;

  printf( "before swap:  a= %d; b= %d\n", a, b );
  swap(a, b);
  printf( "after swap:  a= %d; b=%d\n", a, b );
  return 0;
}
```

- swap函数中的a和b与main函数中的a和b是什么关系？
- 两个printf输出的结果是什么？

swap函数数据区 demo_4_swap.c



demo_4_perfect_number.c

函数示例

- 1. 请写一个程序，给出指定整数范围[1，10000]内的所有完数。判断是不是完数用一个函数完成。

```
int isPerfectNumber(int n){
    int i;
    for(i=1,sum=0;i<=n/2;i++){
        if(n%i==0)
            sum+=i;
    }
    return sum==n;
}
```

demo_4_gcd.c

函数示例

- 2. 写一个函数求两个整数的最大公约数

```
int gcd( int m, int n){
    int gcd=1;
    int min=m<n?m:n;
    for(i=2;i<=min;i++){
        if(m%i==0&&n%i==0) //判断i是不是公约数
            gcd=i; //gcd更新为更大的约数
    }
    return gcd;
}
```

demo_4_prime.c

函数示例

- 3. 写一个函数判断一个数是不是素数，用函数返回值表示判断结果（非0表示是素数，0 表示不是素数）

```
int isPrime(int n){
    int i, isPrimeFlag=1;

    for(i=2; i<n; i++){
        if(n%i==0){
            isPrimeFlag=0;
            break;
        }
    }
    return i==n;
    //return isPrimeFlag;
}
```

4.3 变量作用域

demo_4_swap.c

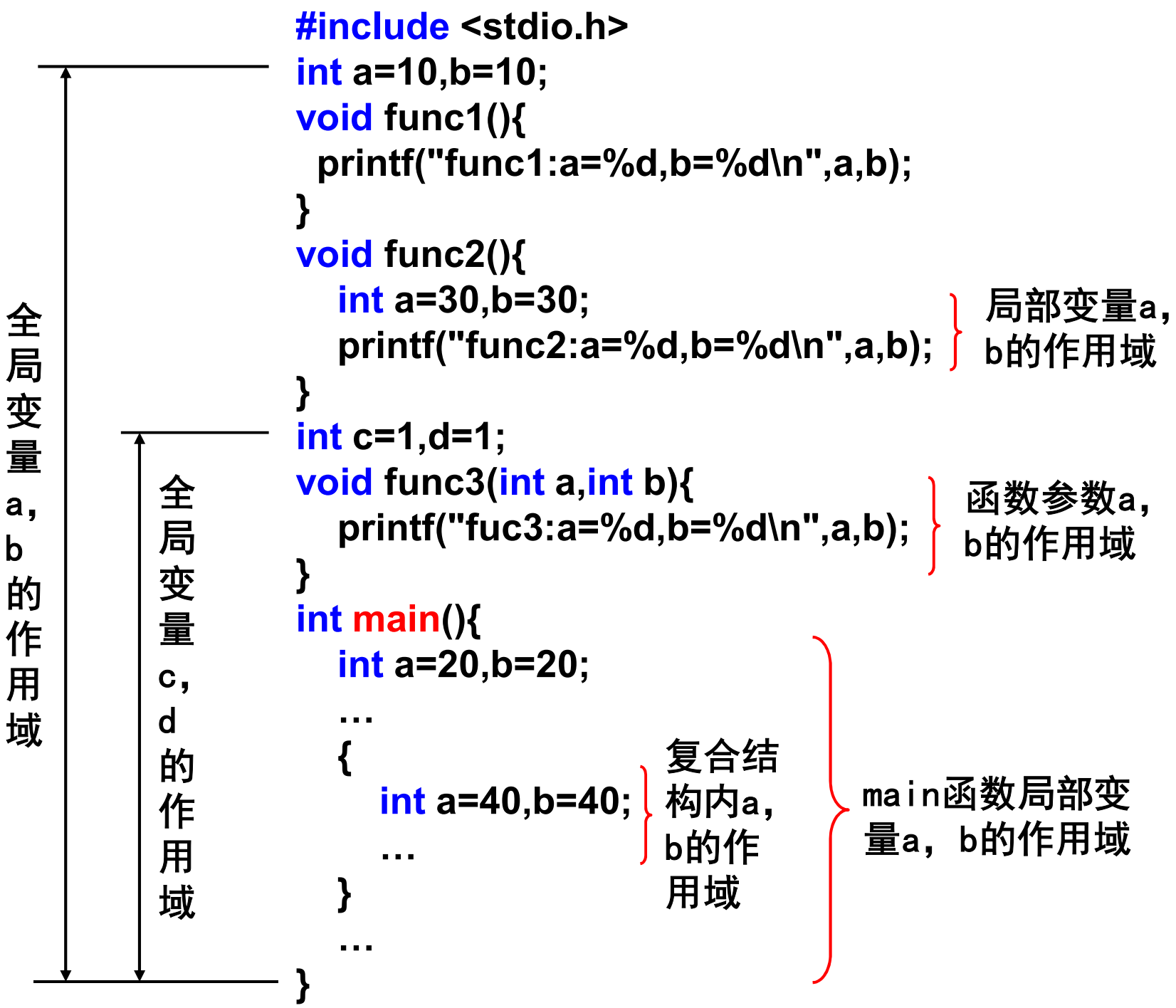
demo_4_swap2.c

swap函数的问题

- 参数传递是赋值传递，修改函数参数（形参）对原来的数据（实参）没有影响，因此不能完成交换数据的任务
- 解决方法
  - 方法1：修改需要交换的数据的作用域，将其变为全局变量
  - 方法2：通过参数传递需要交换的数据的地址

变量作用域

- 作用域：变量的有效范围，也就是变量的生存范围
  - 局部变量：
    - 函数参数，其作用域为整个函数
    - 函数内的变量，其作用域为变量定义位置到函数结束位置
    - 复合语句内的变量，其作用域为变量定义位置到复合语句结束位置
  - 全局变量
    - 定义在所有函数之外的变量，其作用域为变量定义位置到程序结束位置


全局变量a，b的作用域
全局变量c，d的作用域
#include <stdio.h>
int a=10,b=10;
void func1(){
printf("func1:a=%d,b=%d\n",a,b);
}
void func2(){
int a=30,b=30;
printf("func2:a=%d,b=%d\n",a,b);
}
局部变量a，b的作用域
int c=1,d=1;
void func3(int a,int b){
printf("fuc3:a=%d,b=%d\n",a,b);
}
函数参数a，b的作用域
int main(){
int a=20,b=20;
…
{
int a=40,b=40;
…
}
…
}
复合结构内a，b的作用域
main函数局部变量a，b的作用域

demo_4_swap2.c

使用全局变量的swap函数

```
int a=5, b=3;
void swap(){
  int t;

  t = a;  a = b;  b = t;
}
int main(){
  printf( "before swap:  a= %d; b= %d\n", a, b );
  swap();
  printf( "after swap:  a= %d; b=%d\n", a, b );
  return 0;
}
```

4.4 函数与递归

引言

- 函数调用可以嵌套，即在一个函数中调用另一个函数

```
double c_area (double r) {
    return pow(r, 2) * 3.1416;
}
```

- 如果一个函数调用自己就会构成递归调用

递归函数示例

demo_4_fac.c

demo_4_fib.c

- 求n!

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

```
int fac(int n){
    if(n==0)
        return 1;
    return n*fac(n-1) ;
}
int main(){
    printf("3!=%d",fac(3));
}
```

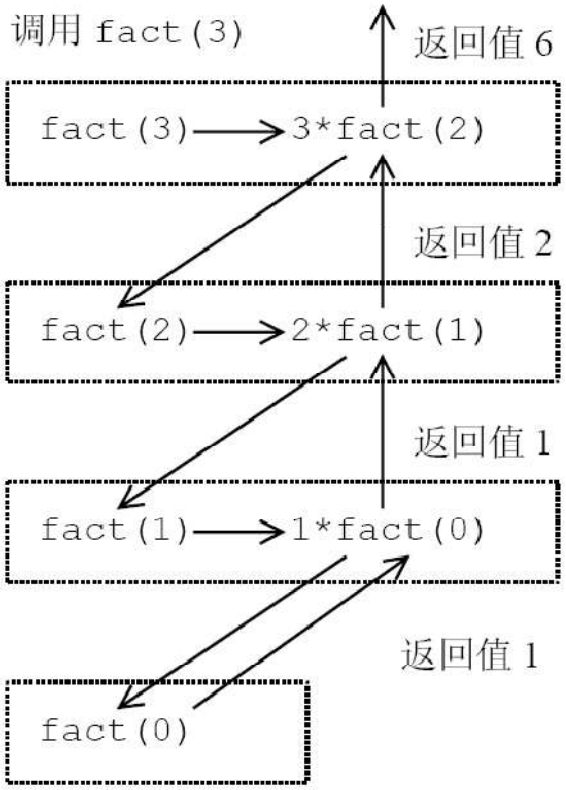
- Fibonacci数列

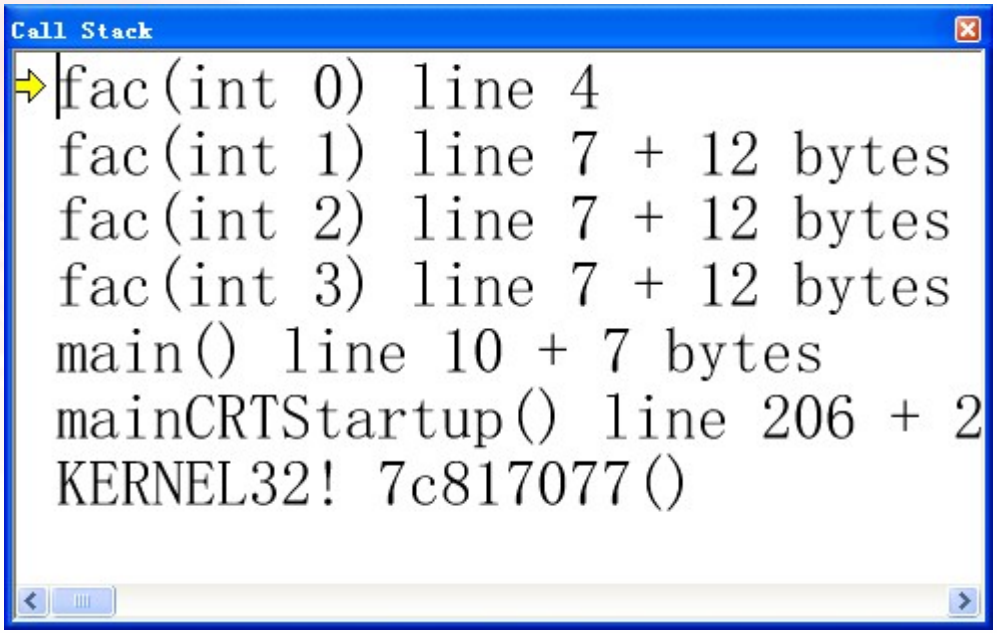
$$F(n)=\begin{cases}1 & n=0\\ 1 & n=1\\ F(n-1)+F(n-2) & n>1\end{cases}$$

```
int fib(int n){
    if(n==0||n==1)
        return 1;
    return fib(n-1)+fib(n-2) ;
}
int main(){
    printf("F(5)=%d",fib(5));
}
```

递归函数的调用过程

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





使用递归的条件

- 具有递归定义的形式：自己调用自己
- 有明确的结束条件：对应return一个明确的值，不再自己调用自己

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

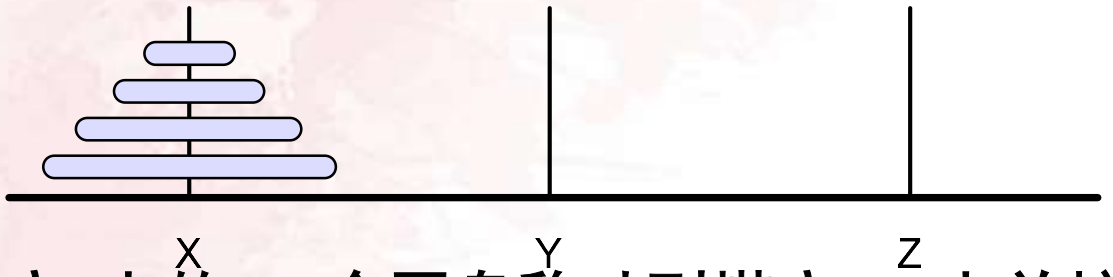
结束条件

递归定义

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

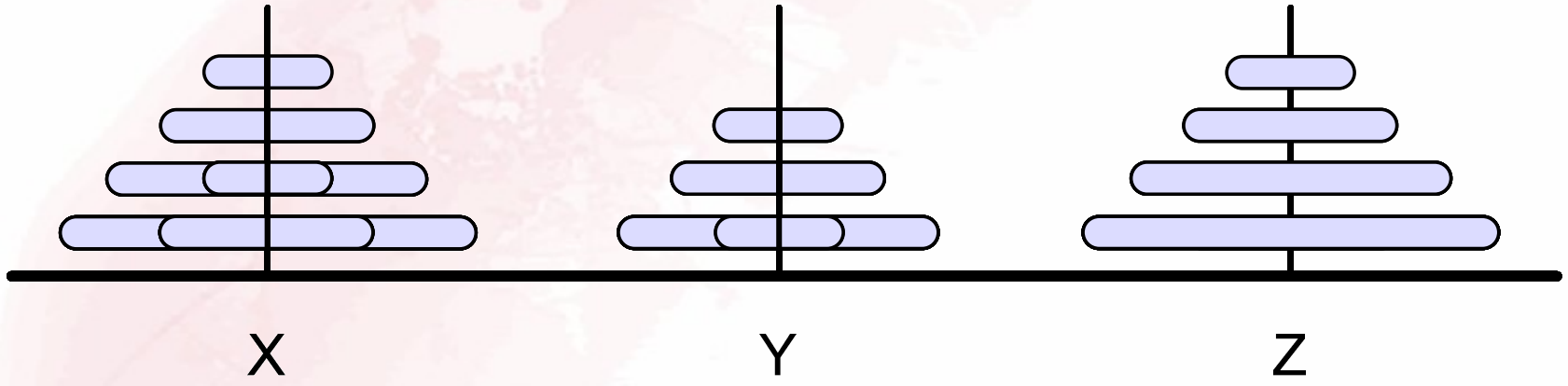
汉诺塔（hanoi）问题

- 假设有三个分别命名为 X、Y 和 Z 的塔座，在塔座 X 上插有 n 个直径大小不同、依小到大分别编号为 1, 2, ..., n 的圆盘，如图所示：



- 要求将塔座X上的 n 个圆盘移动到塔座 Z 上并按相同顺序叠放，圆盘移动时必须遵循下述规则：
  - 每次只能移动一个圆盘；
  - 圆盘可以插在X、Y与Z中的任意塔座上；
  - 任何时刻都不能将较大的圆盘压在较小的圆盘上。
- 如何实现移动圆盘的操作呢？

汉诺塔（hanoi）问题



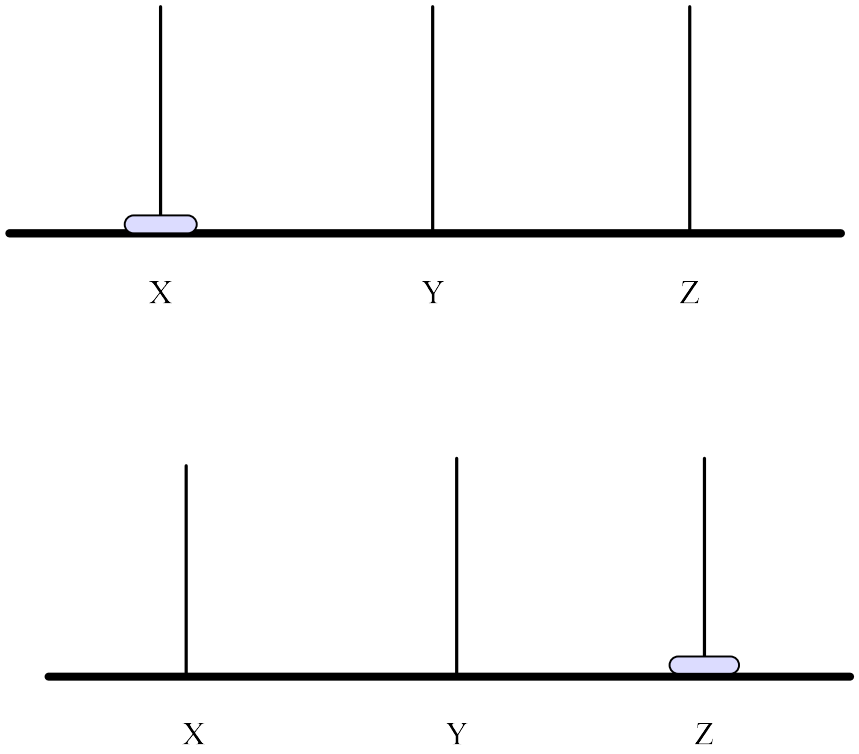

Level 5 / 6 Count 0 / 31 Time 0: 0: 0
TOWER OF HANOI

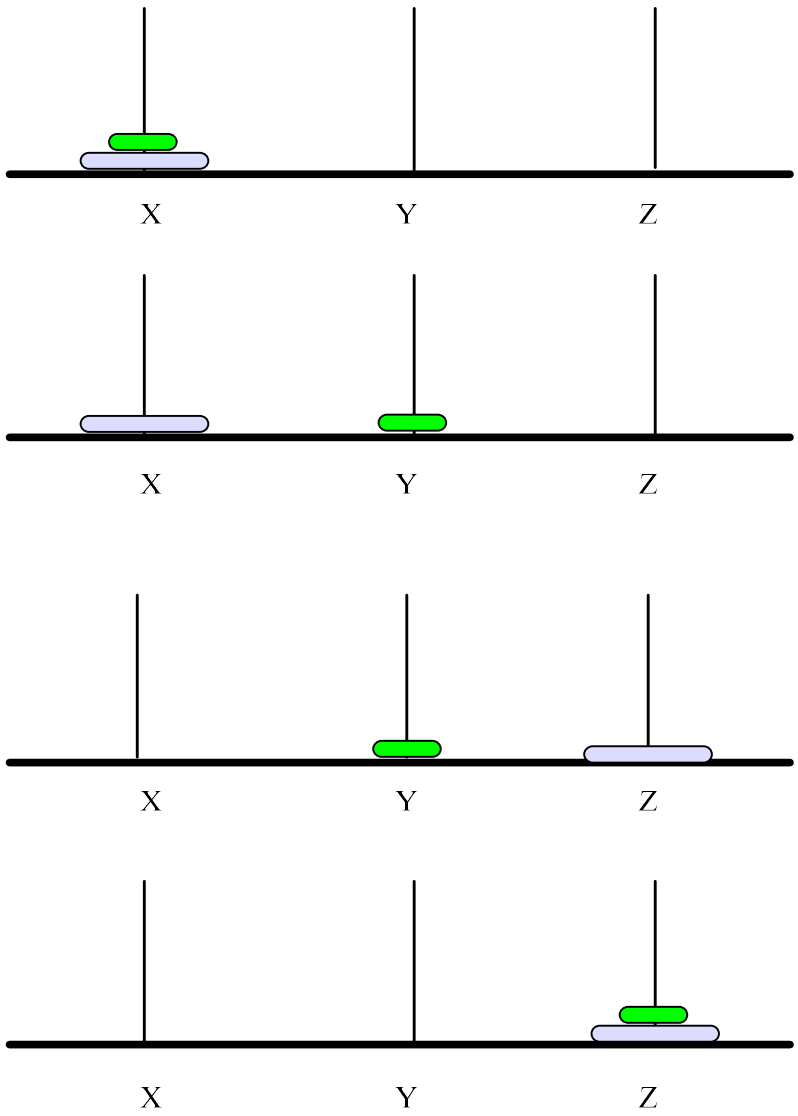

Level 5 / 7 Count 3 / 31 Time 0: 8: 4
TOWER OF HANOI

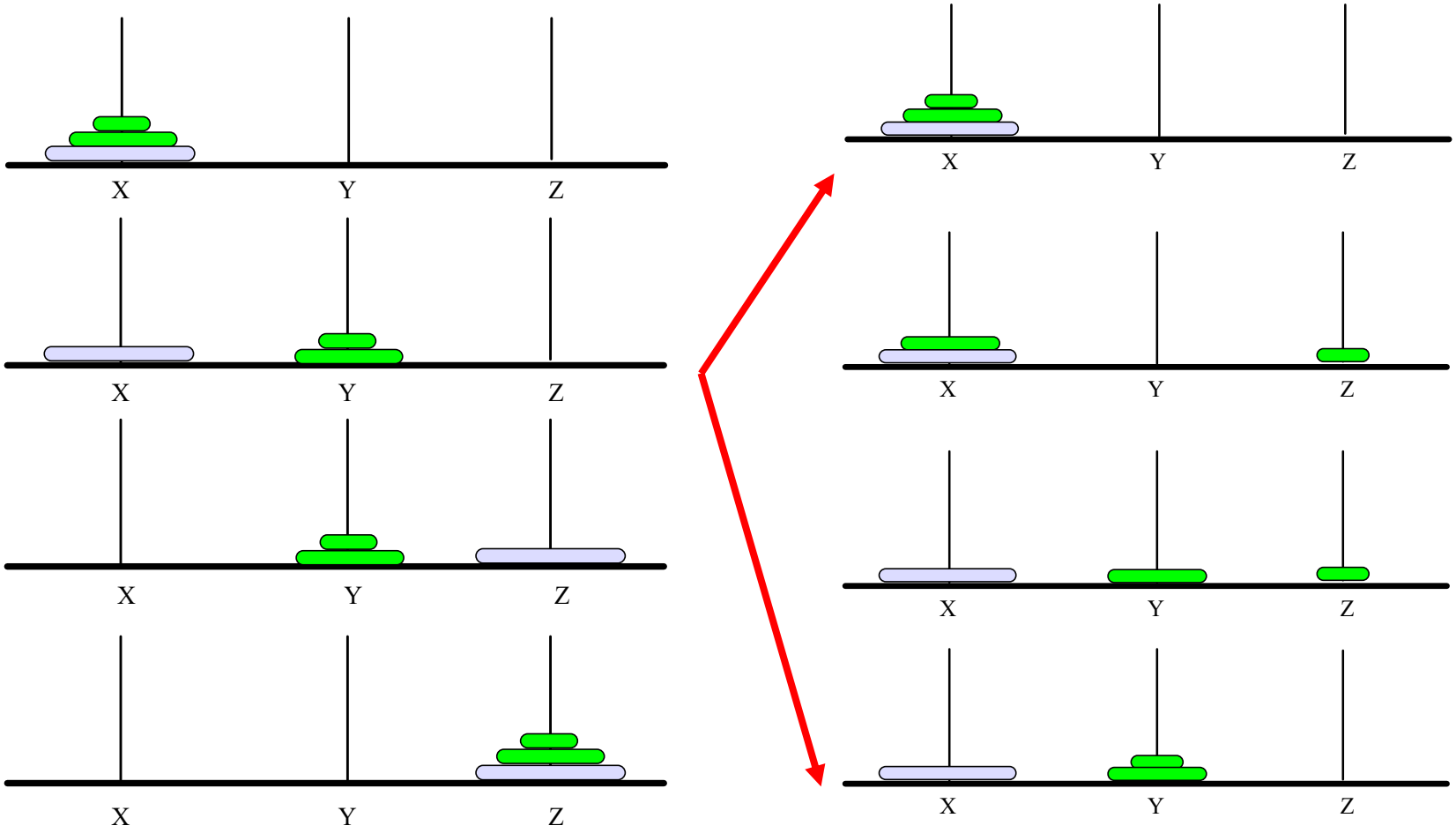

@ Level 3
No N NAME TIME M/C
1 Jake 2.19 7
2 sc45 2.31 7
3 Zzzzzzzz 2.33 7
4 Jake Welcome 2.79 7
5 Shain 3.54 7
6 Dazzer 3.75 7
7 Juliana 5.78 7
8 Anne 6.5 8
9 margaret 6.61 7
10 Dav 8.03 7
11 Rocky 8.66 9
12 Ff 9.5 7
13 MamaMia! 9.73 7
14 Jose 13.94 10
15 Sandra 14.48 7
16 Edi 14.5 15
17 Wall 14.84 11
18 Gryffindor 14.85 10
19 Matt 15.26 7
No N NAME TIME M/C
20 Sabanack 15.81 13
21 Lexie 18.58 8
22 Lilian 18.78 9
23 Tonyjapan 19.19 15
24 Kelly 20.54 13
25 Lele 20.57 12
26 Asdf 23.89 17
27 Ryan 24.76 11
28 Ashtaysar 24.81 11
29 Vsfde 25.45 7
30 Meggles 25.51 12
31 Sarah 28.41 23
32 Hudson 30.56 9
33 Osvaldo 31.38 11
34 Ash 32.69 21
35 Prem 34.22 26
36 Isaac 34.57 15
37 Hayden 38.83 13
38 Siwaly 43.66 11

汉诺塔（hanoi）问题

- 待解决的问题
  - Q1：是否存在某种简单情形，问题很容易解决
  - Q2：是否可将原始问题分解成性质相同但规模较小的子问题，且新问题的解答对原始问题有关键意义

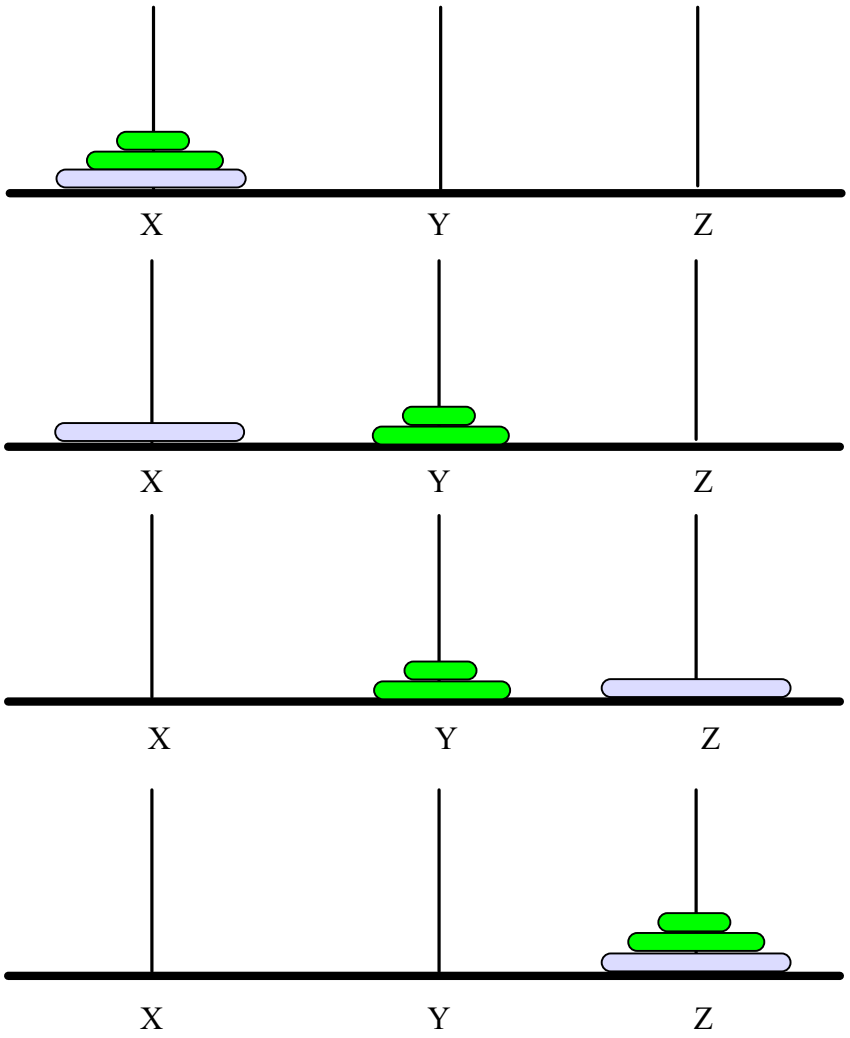

X
Y
Z
X
Y
Z


X
Y
Z
X
Y
Z
X
Y
Z
X
Y
Z


X
Y
Z
X
Y
Z
X
Y
Z
X
Y
Z
X
Y
Z
X
Y
Z
X
Y
Z
X
Y
Z

汉诺塔（hanoi）问题

- 解决方案
  - A1：只有一个圆盘时是最简单情形
  - A2：对于 n > 1，考虑 n – 1 个圆盘，如果能将 n – 1 个圆盘移动到某个塔座上，则可以移动第 n 个圆盘
  - 策略：首先将 n – 1 个圆盘移动到塔座 Y 上，然后将第 n 个圆盘移动到 Z 上，最后再将 n – 1 个圆盘从 Y 上移动到 Z 上



from temp to

demo_4_hanoi.c

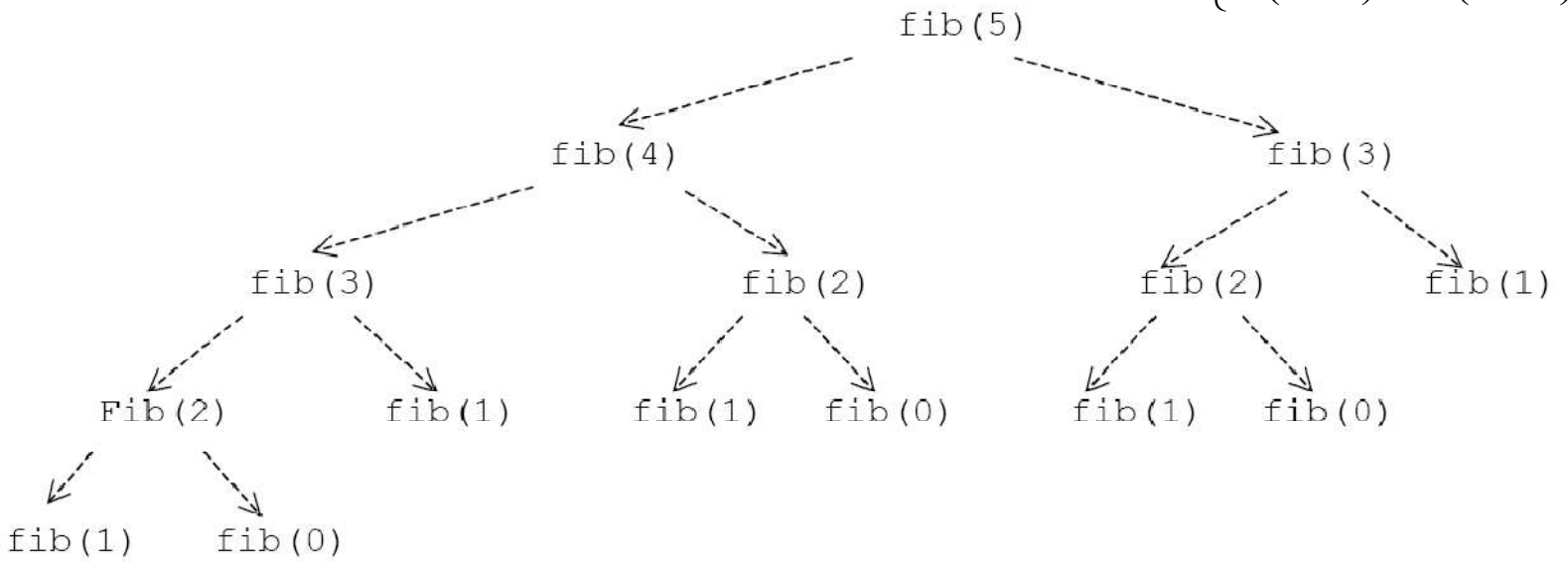
汉诺塔（hanoi）问题

```
void MoveHanoi( unsigned int n,  //圆盘数
                char from,  //源塔座，初始值如 ‘x’
                char tmp,   //过渡塔座，初始值如‘y’
                char to ){   //目标源塔座，初始值如‘z’
   if( n == 1 ) //递归结束条件
      //将圆盘1从 from 移动到 to
   else{
      //将 n – 1 个圆盘从 from 以 to 为中转移动到 tmp;  //递归
      //将圆盘 n 从 from 移动到 to
      //将 n - 1个圆盘从 tmp 以 from 为中转移动到 to;   //递归
   }
}
```

递归的缺点

- 函数调用需要额外的空间（栈）来完成，在调用次数很多的情况下会降低程序效率
- 递归调用中的重复计算

$$F(n)=\begin{cases}1 & n=0\\ 1 & n=1\\ F(n-1)+F(n-2) & n>1\end{cases}$$



demo_4_fib.c

Fibonacci数列的两种求解方法

```
//使用递归求Fibonacci数列
int fib_recursion(int n){
    if(n==0 || n==1)
        return 1;
    return fib_recursion(n-1)
            + fib_recursion(n-
2);
}
```

```
//使用循环求Fibonacci数列
int fib_loop(int n){
    int fn,fn_1=1,fn_2=1,i;
    if(n==0||n==1)    return 1;
    for(i=2;i<=n;i++){
        fn=fn_1+fn_2; //计算f(n)
        fn_2=fn_1;      //更新f(n-2)
        fn_1=fn;        //更新f(n-1)
    }
    return fn;
}
```

递归到循环的转换常常需要借助于高级的程序设计技术和一定的数据结构才能完成

兔子繁殖问题（ Fibonacci数列）

- 例1：假设有一对兔子，一个月后成长为大兔子，从第二个月开始，每对大兔子生一对小兔子。不考虑兔子的死亡，求第n个月的兔子总数

月份	小兔子	大兔子	总数
0	1	0	1
1	0	1	1
2	1	1	2
3	1	2	3
4	2	3	5
5	3	5	8

兔子繁殖问题（ Fibonacci数列）

- 递推过程

F(0)=1 F(1)=1

F(n-2)=M1+M2

F(n-1)=M2 + (M2+M1)

F(n)=(M2+M1)+(M2+M1+M2)

⬇

F(n)=F(n-2)+F(n-1)

兔子繁殖问题（ Fibonacci数列）

- 例2：假设有一对兔子，从出生后第3个月起每个月都生一对兔子，小兔子长到第三个月后每个月又生一对兔子，假如兔子都不死，问每个月的兔子总数为多少？

月份	月龄<1	月龄<2	月龄>2	总数
0	1	0	0	1
1	0	1	0	1
2	1	0	1	2
3	1	1	1	3
4	2	1	2	5
5	3	2	3	8

兔子繁殖问题（ Fibonacci数列）

- 例2：　　F(0)=1　F(1)=1

F(n-2)=M1+M2+M3

F(n-1)=(M3+M2)+M1+(M3+M2)

F(n)=(M3+M2+M1)+(M3+M2)+(M1+M2+M3)

⬇

F(n)=F(n-2)+F(n-1)

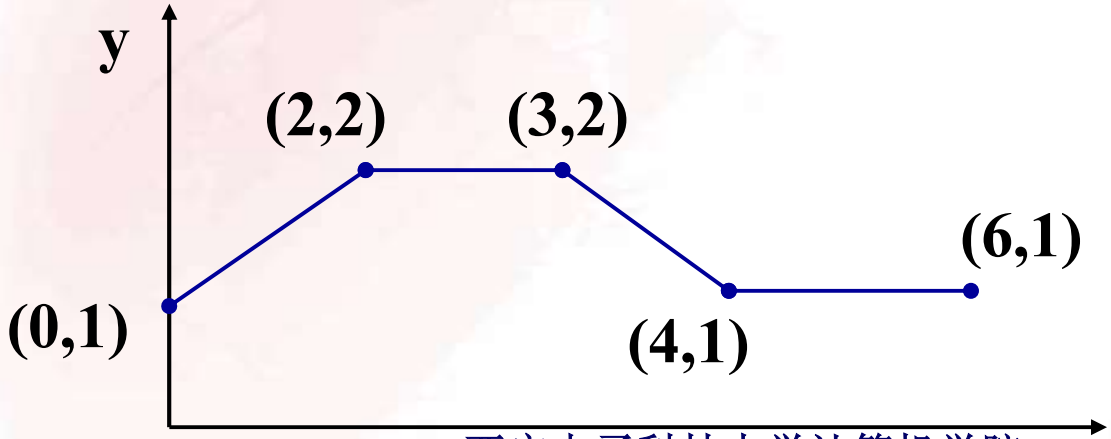
此处先讲数组，然后再讲后续的内容

4.5 数组与函数

- 需要讨论的问题
  - 数组元素能否作为函数参数？
  - 数组能否作为函数参数？
  - 数组能否作为函数返回值？

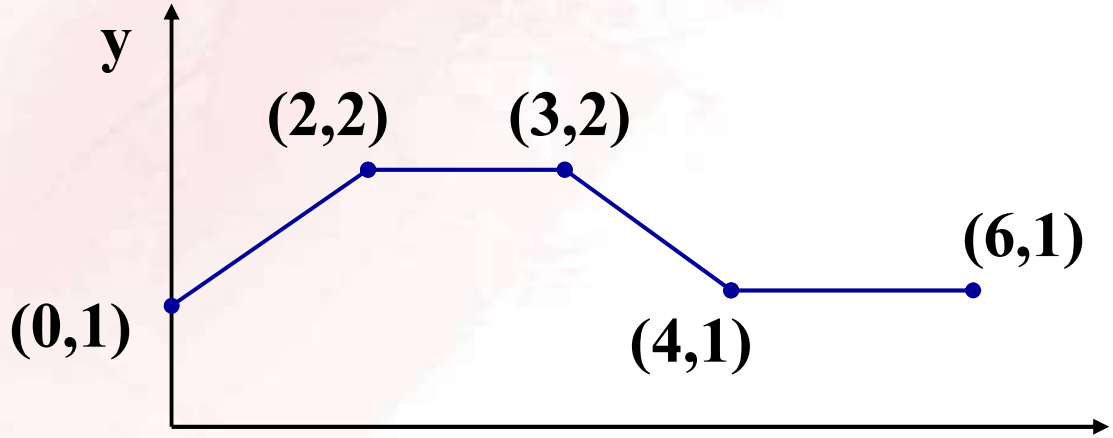
数组元素作为函数参数

- 数组元素是一个变量，可以出现在任何普通变量能出现的地方
  - 作为表达式的一部分参与运算
  - 作为函数调用的实际参数
- 示例：求下图所示的折线长度



数组元素作为函数参数

- 示例：求下图所示的折线长度
  - 折线长度=多条线段长度之和
  - 求线段长度的公式：$\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$
  - 可以利用循环完成求和

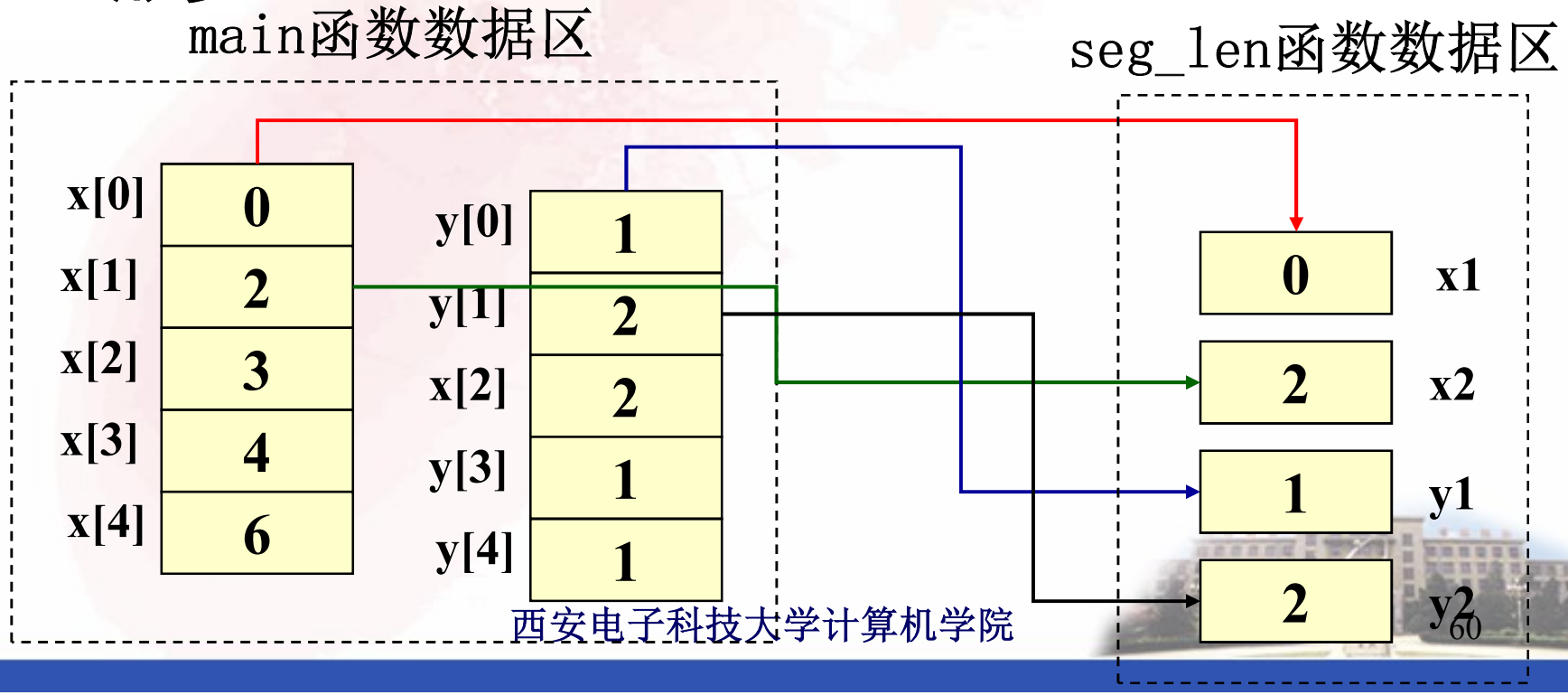


demo_4_len.c

```
double seg_len(double x1,double y1,double x2,double y2){
    return sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));
}
int main(){
    double x[5]={0,2,3,4,6}; //点的x坐标
    double y[5]={1,2,2,1,1}; //点的y坐标
    double polyline_len=0;  //记录折线的长度
    int i;
    for(i=0;i<4;i++){
        polyline_len+=seg_len(x[i], y[i], x[i+1], y[i+1]);
    }
    printf("polyline length = %f\n",polyline_len);
    return 0;
}
```

数组元素作为函数参数时的传递

- 函数调用时实参是值传递
- 数组元素作为实参就是把数组元素的值传递给形参



用函数求折线段长度

- 数组可以表示一组数据，而且可以单独操作每个元素，考虑用数组作为函数参数
- 用数组作为函数参数需要解决的问题

1. 定义函数时如何定义数组参数？

2. 调用函数时如何传入实际的数组？

3. 实参数组和形参数组的关系？

用函数求折线段长度

```
double poly_len( … ){
    …
}

int main(){
    double co_x[5],co_y[5];
    double len=poly_len(…);
    printf("polyline length = %f\n", len);
    return 0;
}
```

如何定义数组参数，需要定义数组大小吗？

如何传入实际数组，数组元素值和元素个数如何传递？

回顾数组的特点

取数组首地址的三种方法
- 取数组名对应的值a
- 取数组名的地址&a
- 取第一个元素的地址&a[0]

- 数组名是数组首地址，数组元素连续存放
- 通过下标和首地址可以计算任何一个数组元素的地址，即使下标超出数组范围

```
&a[i] = a + i×sizeof(a[0])
```

- 只要将数组首地址传入函数，那么在函数内就可以访问到所有数组元素
- 为了使数组元素访问不超出数组范围，应该明确给出数组大小

demo_4_len_add.c

用函数来求折线段长度

```
double poly_len( double x[], double y[], int n){
    double len=0.0;
    int i;
    for(i=0;i<n-1;i++)
        len+=sqrt(pow(x[i]-x[i+1],2)+pow(y[i]-y[i+1],2));
    return len;
}

int main(){
    double co_x[5]={0,2,3,4,6};
    double co_y[5]={1,2,2,1,1};
    double len=poly_len( co_x, co_y, 5);
}
```

这确实是一个数组吗？

传递数组首地址的好处是避免复制大量元素

用数组作为函数参数的本质

如果x是一个数组，那么
x = &x = &x[0]

```
double poly_len( double x[], double y[], int n){
    …
}
```

实际的结果是：
x=0x0012fc60
&x=0x0012fbfc
&x[0]=0x0012fc60

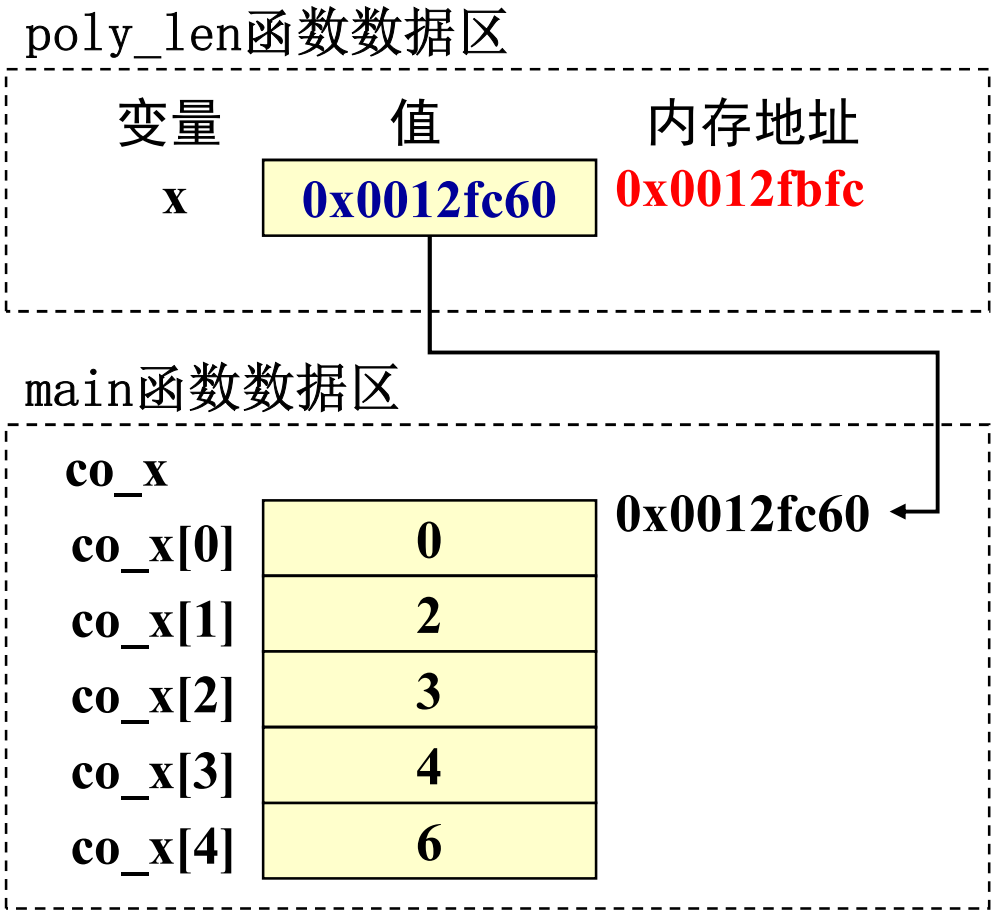
x是函数内的局部变量，x是用指针变量表示的，x的值是数组首地址，是函数调用时传入的实际数组的首地址

```
double poly_len( double *x, double *y, int n){
    …
}
```

```
double poly_len( double x[], double y[], int n){
}

 double poly_len( double *x, double *y，  int n){
 }
```

```
int main(){
   …
   poly_len( co_x, co_y, 5 )
   …
}
```



- **取数组首地址的三种方法**
  - 取数组名对应的值a
  - 取数组名的地址&a
  - 取第一个元素的地址&a[0]

sizeof运算符 （不是库函数）

```
int main(){
    …
    poly_len( co_x, co_y, sizeof(co_x)/sizeof(co_x[0] ) );
    …
}
```

数组大小一旦改变，这个值也要改变，能不能计算出数组元素个数？

sizeof运算符用于计算一种类型或一个变量所占据的存储空间，如果对数组变量应用sizeof运算符得到的是整个数组所占据的存储空间

⇩

sizeof(数组)=sizeof(单个元素)×数组元素个数

⇩

数组元素个数=sizeof(数组)/sizeof(单个元素)

demo_4_sizeof.c

sizeof运算符

```
sizeof(char)=1;
sizeof(int)=4;
sizeof(long)=4;
sizeof(float)=4;
sizeof(double)=8;
int a; sizeof(a)=4;
```

```
int a[3]={1,2,3};
sizeof(a)=12;

double b[]={1.0,2.0,3.0};
sizeof(b)=24;
sizeof(b[0])=8
数组的元素个数=
    sizeof(b)/sizeof(b[0])
或：sizeof(b)/sizeof(double)
```

sizeof运算符

```
double poly_len( double x[], double y[], int n){
    …
}
```

sizeof(x) = 4
sizeof(x)/sizeof(x[0]) = 0

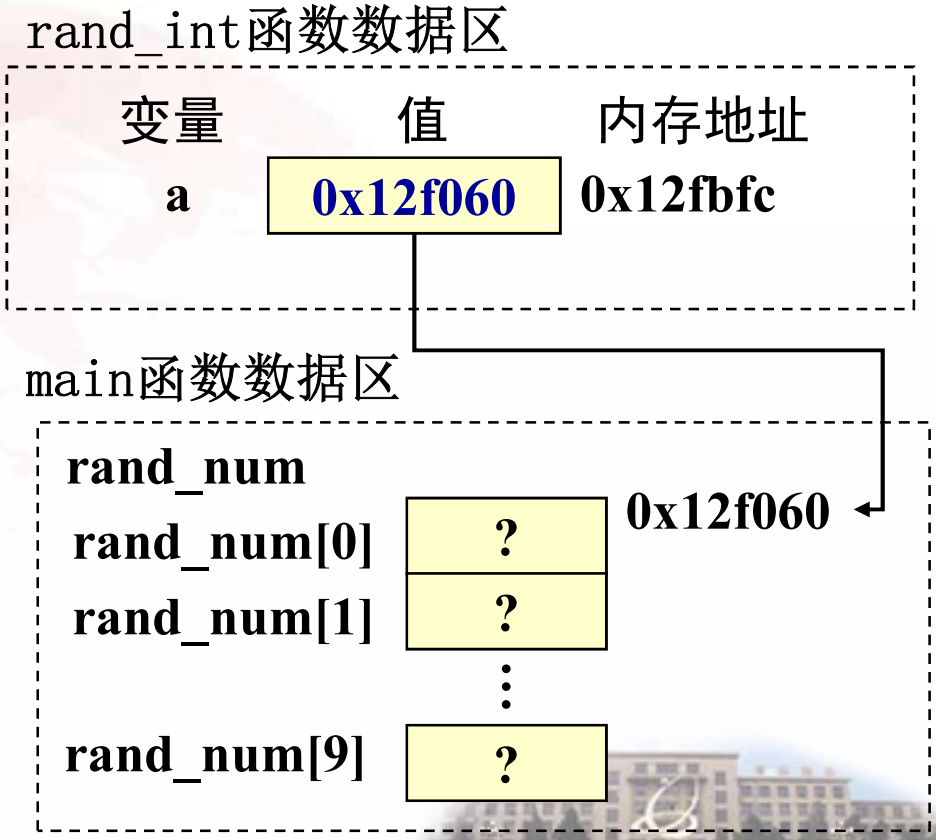
这从另一侧面证明函数参数中的数组是以指针形式实现的，而不是一个真正的数组。

demo_4_rand_array.c

数组作为函数的输出参数

- 用函数随机产生若干整数，保存在数组中

```
void rand_int( int a[], int n ){
    int i;
    srand(time(0));
    for( i = 0; i < n; i++ )
        a[i] = rand()%100;
}
int main(){
    int rand_num [10];
    rand_int(rand_num, 10);
}
```



函数形参数组和实参数组是同一块内存，因此在函数内修改形参数组就相当于修改了实参数组

数组与函数返回值

- 数组不能作为函数返回值

在函数内如何访问作为返回值的数组？

```
int [] rand_int(int n ){
    …
}
int main(){
    int rand_num [10];
    rand_num = rand_int(10);
}
```

数组可以整体赋值吗？编译错误

4.6 C语言常用函数

数学函数

- #include <math.h>
  - double sin(double rad);
  - double sqrt(int n);
  - double pow(double,double);
  - double fabs(double);
  - int abs(int)
  - double log(double x);
  - double log10(double x);

输入输出函数

- #include <stdio.h>
  - printf—格式化输出
  - scanf—格式化输入
  - getchar—输入一个字符
  - putchar—输出一个字符

demo_4_chario.c

输入输出函数

- int getchar()—从标准输入流（stdin）读取一个字符
  - 缓冲输入，需要按下回车后才能获取到值
  - 正常情况下，返回值表示读入的字符
  - 如果返回值是EOF（-1）表示读错误或到了流结束位置
- int getch()；直接读取键值，不用输入回车键
- int putchar(int ch)—将字符ch写入标准输出流（stdout）
- 示例：将从键盘输入的一行小写字符转换成大写字符。

时间函数

- #include <time.h>
  - time_t time(time_t *timer)—获得从1970/1/1 0时0分0秒至今的秒数
    - time_t可以看作整数类型
  - clock_t clock();—获得从程序开始运行至今处理器经过的时钟数（可以看成毫秒数）
    - clock_t可以看作整数类型
  - CLOCKS_PER_SEC—表示每秒有多少个时钟的常数

时间函数

对这一段程序计时

```
#include <stdio.h>
#include <time.h>
int main(){
    int start,finish;
    double time;
    start=clock();
    …
    …
    finish=clock();
    time=(finish-start)*1.0/CLOCKS_PER_SEC;
    ….
    return 0;
}
```

随机数函数

- #include <stdlib.h>
  - int rand()—产生一个[0,RAND_MAX]范围内的伪随机数
    - RAND_MAX是一个系统常数，可以直接使用
  - void srand(unsigned int seed)—设置伪随机数序列的种子
    - 如果不设定随机数系列的种子，同一个程序两次运行得到的随机数完全相同
    - 通常以时间作为随机数种子

demo_4_rand.c

随机数函数

- 产生5个随机数

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int i;
    printf( "RAND_MAX is %d.\n", RAND_MAX );
    printf( "Five numbers generated as follows:\n" );
    for( i = 0; i < 5; i++ )
        printf( “%d ", rand() );
    printf( "\n" );
    return 0;
}
```

demo_4_rand.c

随机函数

- 用时间做种子产生5个随机数

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main(){
    int i;
    printf( "Five numbers generated as follows:\n" );
    srand( (int)time(NULL) );
    for( i = 0; i < 5; i++ )
        printf( "%d; ", rand() );
    printf( "\n" );
    return 0;
}
```

demo_4_rand_range.c

随机函数

- 生成5个［low, high]范围内的随机数

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main(){
    int i, low=10, high=20;
    srand( (int)time(NULL) );
    for( i = 0; i < 5; i++ )
        printf( "%d; ",  low+(rand()%(high-low)) );
    printf( "\n" );
    return 0;
}
```

字符串处理函数

- 需要包含头文件<string.h>
- 常用字符串处理函数
  - int strlen(char s[])：计算字符串长度，返回值标志字符串长度

```
char name[]="xidian";
int n=strlen(name);
```

⟹ n=6

  - strcat(char dst[],char src[])：将字符串src连接到字符串dst尾部

```
char s[50]="c language";
char t[]=" programming";
strcat(s,t);
```

⟹ s="c language programming"

字符串处理函数

- 常用字符串处理函数
  - strcpy(char dst[],char src[])—将字符串src复制到dst中

```
char name1[]="xidian",name2[20];
strcpy(name2,name1);
//结果，name2:"xidian"
```

  - strncpy(char dst[],char src[], int n)—将src中最多n个字符复制到dst中

```
char s1[]="c language", s2[6];
strncpy(s2,s1,5);
//结果，s2:"c lan"
```

字符串处理函数

demo_4_strcmp.c

- 常用字符串处理函数
  - strcmp （ char s[],char t[] ）—比较字符串s和t，返回值表示比较结果，0表示相同，s>t返回值正值(通常是1)，s<t返回负值（通常是-1），大小写敏感

```
char s[]="c language",t[]=" C LANGUAGE";
int n=strcmp(s,t);
//结果，n=1
```

字符串I/O函数

demo_4_scanf_gets.c

- 用scanf输入字符串

```
char str[100];
scanf("%s", str);
```

遇到空格或换行符表示字符串输入结束，因此不能输入包含空格的字符串

- 用gets输入字符串

gets函数原型：`char *gets( char buffer[] );`

```
char str[100];
gets (str);
```

遇到换行符表示字符串输入结束，可以输入包含空格的字符串，返回值为NULL表示文件结束或出错

字符串I/O

demo_4_puts_printf.c

- 用printf 输出字符串

```
char s1[]="hello ";
char s2[]="world";
printf("%s", s1);
printf("%s", s2);
```

输出结束不会换行，
因此输出结果为
hello world

- 用puts 输出字符串

puts函数原型：int puts(const char string[]);

```
char s1[]="hello ";
char s2[]="world";
puts( s1 );
puts( s2 );
```

输出结束自动换行，
因此输出结果为

```
hello
world
```

字符串函数使用实例

- 写一个程序，通过标准输入读入多行字符串，输出其中最长的一个，如果最长行不止一个，则输出其中的第一行。输入的行为“***end**”时表示输入结束

demo_4_maxline.c

字符串函数使用实例

```
while (1) {
    输入新行;
    if(是结束行)break;

    if (新行比以前记录的最长行更长)
    {
        记录新行及其长度;
    }
}

输出所记录的最长行;
```

用gets输入一行文本

用strcmp比较 “***end**”

用strlen求一行长度

用strcpy保存新行

用printf/puts输出结果

小结

- 定义函数的要素
  - 函数头
    - 函数名—使用函数需要的名称，合法的标识符
    - 返回值类型—函数计算结果的类型
    - 参数表—完成计算需要的数据（数量和类型）
  - 函数体

```
返回值类型  函数名     参数表
    ↓        ↓         ↓
double   c_area  ( double r )
```

函数头

```
{
    return r * r * 3.1416;
}
```

函数体

小结

- 函数定义中的形参和函数调用中的实参

```
#include <stdio.h>
#include <math.h>

double c_area (double r) {
    return pow(r, 2) * 3.1416;
}

int main () {
   double v;
   v=2.4 * c_area( 3.24 ) / 3.0;

   return 0;
}
```

形参

实参

实参

实参

形参是定义函数时用来表示函数输入数据类型和名称的变量定义

实参是调用函数时实际传入的数据，可以是变量、常数、表达式或其他函数调用返回的值

小结

- 局部变量
  - 在函数内部定义的变量、形式参数、复合结构内定义的变量
- 全局变量
  - 在所有函数之外定义的变量
- 递归函数
  - 函数直接或间接调用自己形成递归

作业