

程序设计基础

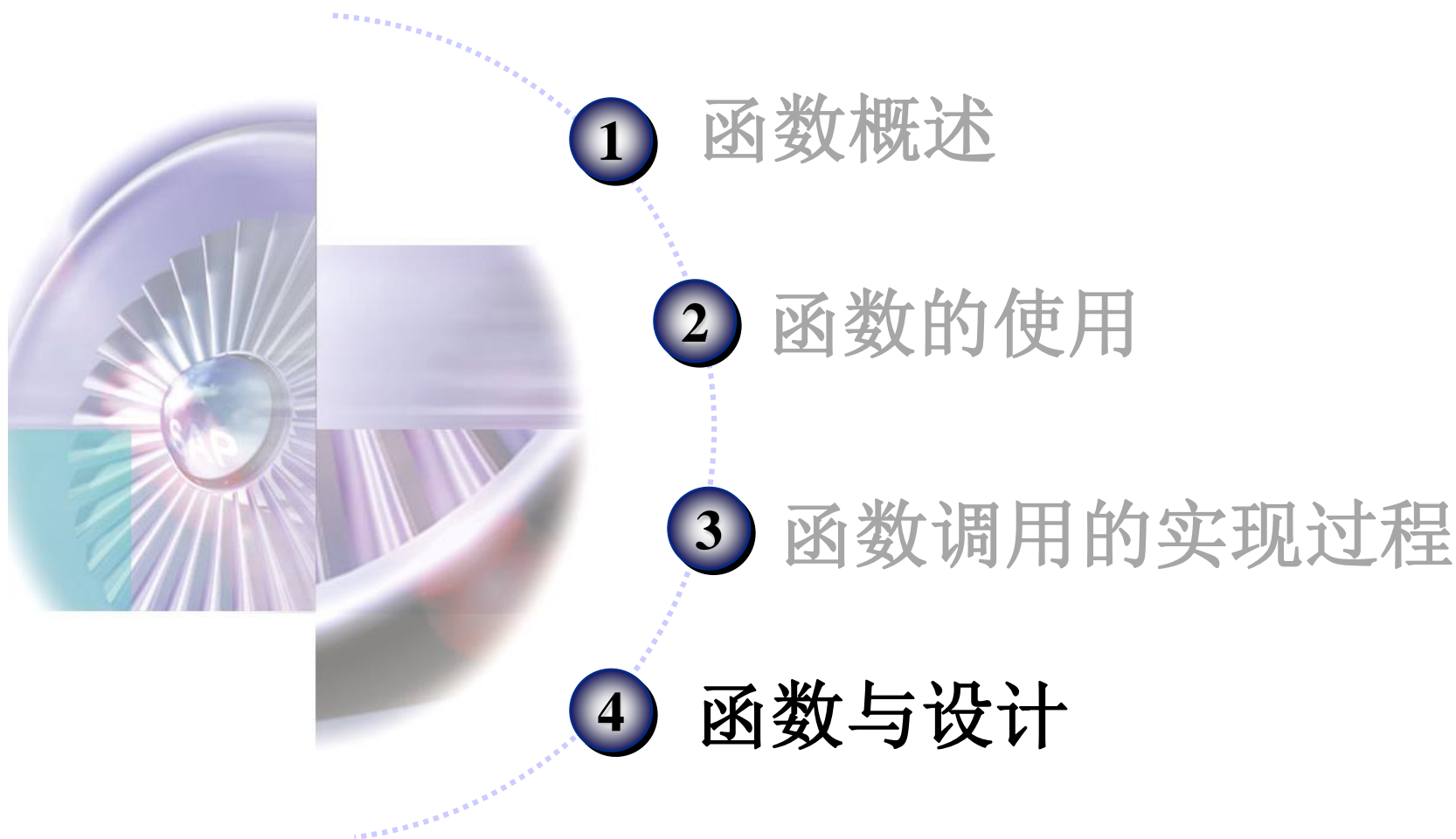
Fundamental of Programming

清华大学软件学院

刘玉身

liuyushen@tsinghua.edu.cn

Lecture 6: 函数

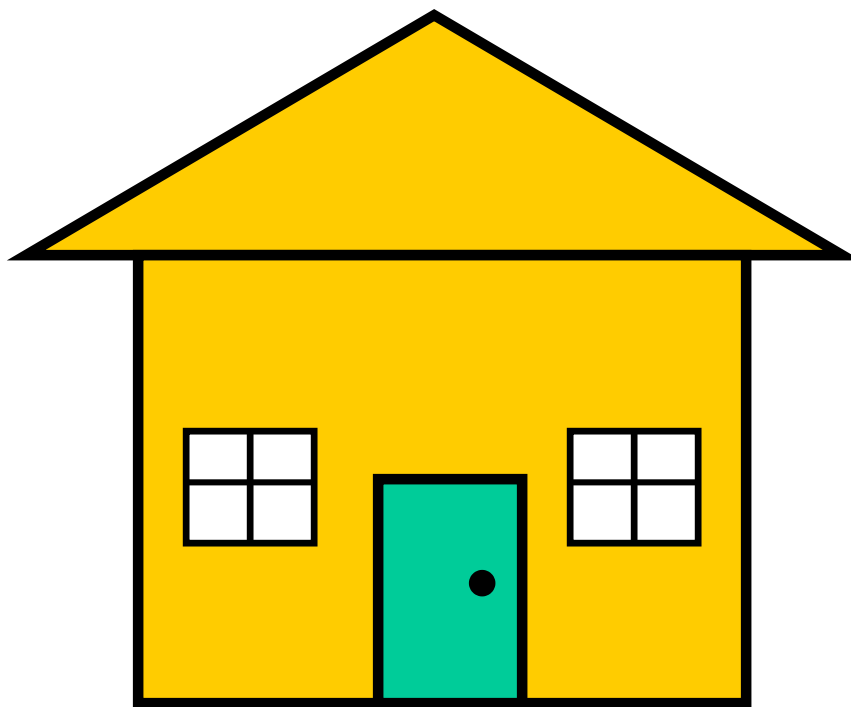
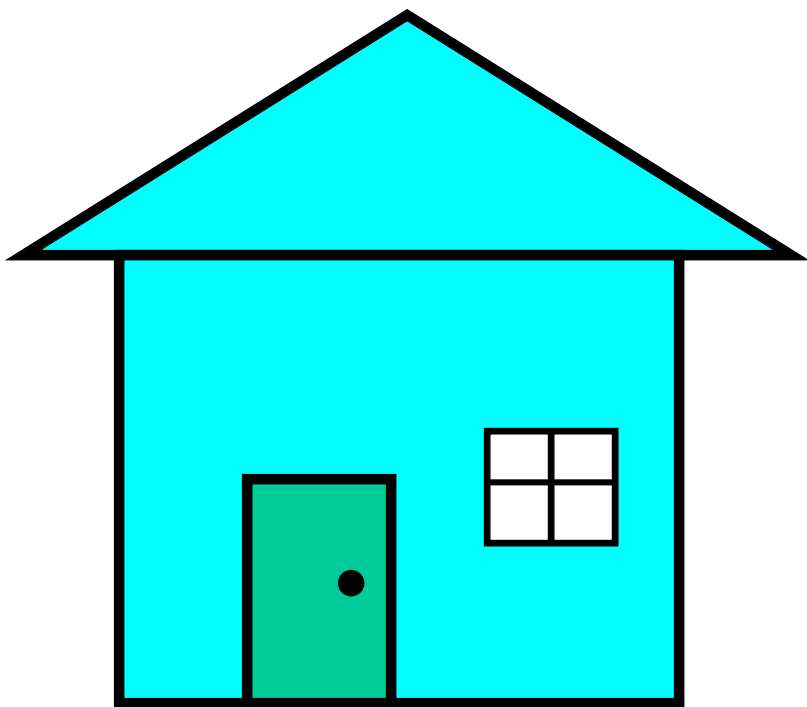


设计过程

如何设计一栋简易的房子？

输入：一张白纸

输出：一张图纸



```
draw_house(color, num_windows)
```

```
    draw body as a colored rectangle;  
    draw roof as a colored triangle
```

```
    if num_windows is one
```

```
        draw door
```

```
        draw window
```

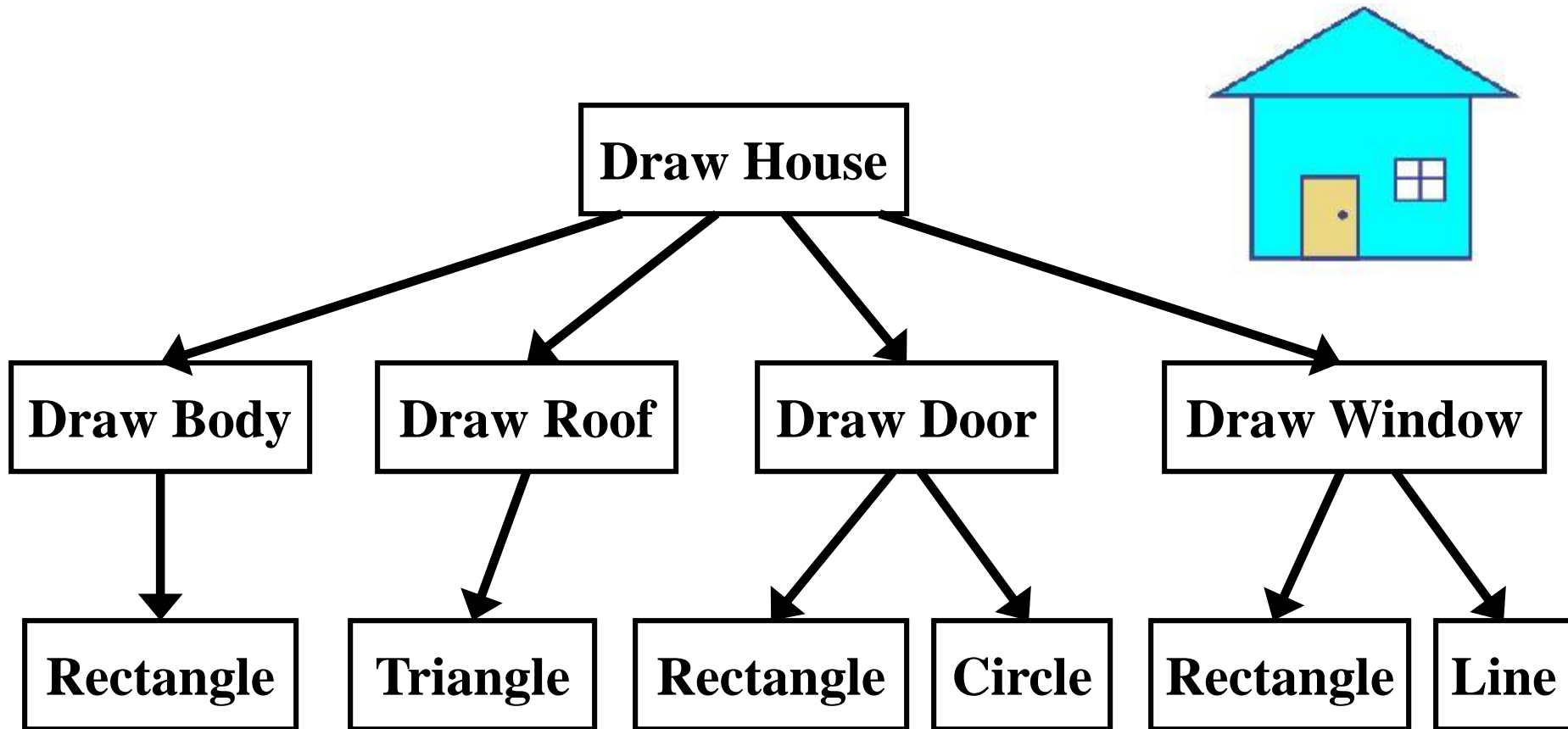
```
    if num_windows is two
```

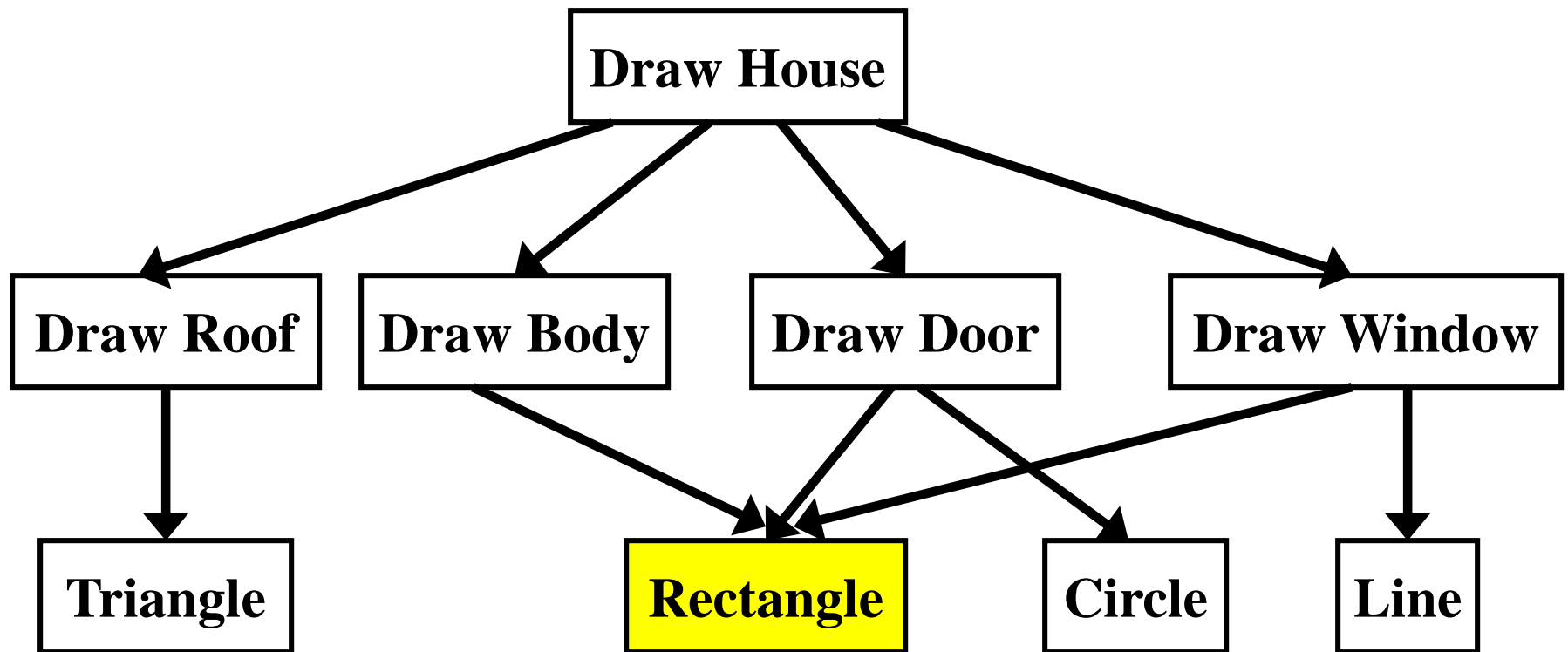
```
        draw door
```

```
        draw window
```

```
        draw window
```

函数分解





分析、设计与编程

- ◆ 分析问题
- ◆ 设计一个 “big-picture” 解决方案
 - ☺ 功能分解、相互关系。
- ◆ 设计各个函数
 - ☺ 基于现有的函数
- ◆ 编程实现

示例3: 质数对

问题描述:

一个加密算法需要用到一对质数，符合要求的质数对必须满足：

- ① 两个数不相同，且都是质数；
- ② 在这两个质数之间没有其他质数。

编写一个程序，输入任意两个正整数，判断它们是否是有效的质数对。

整数1	整数2	是否有效	原因
15	17	无效	15 不是质数
13	19	无效	17 是质数
17	19	有效	
31	29	有效	
31	31	无效	两个数相同

问题分析

1. 需要定义几个函数？
2. 函数原型是什么？
3. 质数函数如何实现？
4. 如何判断一对整数是否有效？需要做哪些事情？

```
#include <stdio.h>
#include <math.h>
int IsPrime(int m);
int main()
{
    int num1, num2, i, temp;
    int result = 1;

    scanf("%d %d", &num1, &num2);
    if((num1 == num2) || !IsPrime(num1) ||
        !IsPrime(num2))
        result = 0;
    else
    {
        if(num1 > num2)
        {
            temp = num1;
            num1 = num2;
            num2 = temp;
        }
    }
}
```

```
        for(i = num1+1; i < num2; i++)
        {
            if(IsPrime(i))
            {
                result = 0;
                break;
            }
        }
    }
    if(result == 1)
        printf("%d和%d是有效质数对", num1, num2);
    else
        printf("%d和%d是无效质数对", num1, num2);
    return 0;
}
int IsPrime(int m)
{
    .....
}
```

判断整数 m 是否为质数的方法：让 m 被 2 到 \sqrt{m} 除，若 m 能被其中任何一个数整除，则说明它不是一个质数；否则的话，说明它是一个质数。

```
int IsPrime(int m)
{
    int j, sq;

    sq = (int)sqrt((double)m);
    for(j = 2; j <= sq; j++)
    {
        if(m % j == 0) break;
    }
    if(j > sq)
        return 1;
    else
        return 0;
}
```

示例4: 循环右移

问题描述:

编写一个程序，读入一组整数（不超过20个），当用户输入0时，表示输入结束。接下来再输入一个正整数M，然后程序将把这组整数循环右移M次，然后把循环右移的结果打印出来。

所谓循环右移，就是把每个数组元素往右移动一格，然后把最右边的那个元素移回到最左边。例如，对于一组整数“100 400 200 300”，把它循环右移一次的结果是“300 100 400 200”；把它循环右移两次的结果是“200 300 100 400”。

问题分析

1. 如何循环右移M位？
2. 能否进行问题分解？
3. 如何循环右移1位？

如何将每个数组元素循环右移1位？

0	2
1	1
...	...
N-2	4
N-1	3

```
temp = a[N-1];  
for( k = N-1; k > 0; k-- )  
{  
    a[k] = a[k-1];  
}  
a[0] = temp;
```

```
#include <stdio.h>
void shift(int a[], int N);
int main()
{
    int N=0, b[20], i, M;

    while(1)
    {
        scanf("%d", &b[N]);
        if(b[N] == 0) break;
        else N++;
    }
    scanf("%d", &M);
    for(i = 1; i <= M; i++) shift(b, N);
    for(i = 0; i < N; i++) printf("%d ", b[i]);
    return 0;
}
```

实参用数组名

形参用数组定义 \Leftrightarrow int a[]

```
void shift(int a[], int N)
{
    int temp, k;

    temp = a[N-1];
    for(k = N-1; k > 0; k--)
    {
        a[k] = a[k-1];
    }
    a[0] = temp;
}
```

```

#include <stdio.h>
void shift(int a[], int N);
int main()
{
    int N=0, b[20], i, M;
    while(1)
    {
        scanf("%d", &b[N]);
        if(b[N] == 0) break;
        else N++;
    }
    scanf("%d", &M);
    for(i = 1; i <= M; i++) shift(b, N);
    for(i = 0; i < N; i++) printf("%d ", b[i]);
    return 0;
}

```

```

void shift(int a[], int N)
{
    int temp, k;
    temp = a[N-1];
    for(k = N-1; k > 0; k--)
    {
        a[k] = a[k-1];
    }
    a[0] = temp;
}

```

Question?

在函数调用时，传地址而不传值

a a[0] a[1] a[2] a[3] a[4]

1	2	3	4	5
----------	----------	----------	----------	----------

b b[0] b[1] b[2] b[3] b[4]

main的栈帧

```

#include <stdio.h>
void shift(int a[], int N);
int main()
{
    int N=0, b[20], i, M;
    while(1)
    {
        scanf("%d", &b[N]);
        if(b[N] == 0) break;
        else N++;
    }
    scanf("%d", &M);
    for(i = 1; i <= M; i++) shift(b, N);
    for(i = 0; i < N; i++) printf("%d ",b[i]);
    return 0;
}

```

```

void shift(int a[], int N)
{
    int temp, k;
    temp = a[N-1];
    for(k = N-1; k > 0; k--)
    {
        a[k] = a[k-1];
    }
    a[0] = temp;
}

```

QuickWatch

Expression: Reevaluate

Add Watch

Value:

Name	Value	Type
b	0x003af990 {1, 2, 3, 0, -858993460}	int[20]
[0]	1	int
[1]	2	int
[2]	3	int
[3]	0	int
[4]	-858993460	int
[5]	858993460	int

Close Help

QuickWatch

Expression: Reevaluate

Add Watch

Value:

Name	Value	Type
a	0x003af990 {1}	int *
	1	int

Close Help

```

#include <stdio.h>
void shift(int a[], int N);
int main()
{
    int N=0, b[20], i, M;
    while(1)
    {
        scanf("%d", &b[N]);
        if(b[N] == 0) break;
        else N++;
    }
    scanf("%d", &M);
    for(i = 1; i <= M; i++) shift(b, N);
    for(i = 0; i < N; i++) printf("%d ",b[i]);
    return 0;
}

```

1 2 3 0

1

3 1 2

1 2 3 0

1

3 1 2

```

void shift(int a[10], int N)
{
    int temp, k;
    temp = a[N-1];
    for(k = N-1; k > 0; k--)
    {
        a[k] = a[k-1];
    }
    a[0] = temp;
}

```

```

void shift(int a[100], int N)
{
    int temp, k;
    temp = a[N-1];
    for(k = N-1; k > 0; k--)
    {
        a[k] = a[k-1];
    }
    a[0] = temp;
}

```

数组名作函数参数

- 数组名作函数参数说明

- 地址传递

- 在主调函数与被调函数分别定义数组, 且类型应一致

- 形参数组大小(多维数组第一维)可不指定

- 形参数组名是 “地址变量”

与“指针”的关系?

- **Valid examples:**

```
void function(int array_values[100][50]);
```

```
void function(int array_values[][50]);
```

- **Invalid examples:**

```
void function(int array_values[100][]);
```

```
void function(int array_values[][]);
```

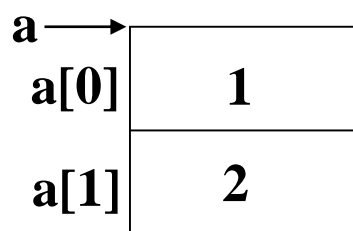

例：数组元素与 数组
名作函数参数比较

值传递

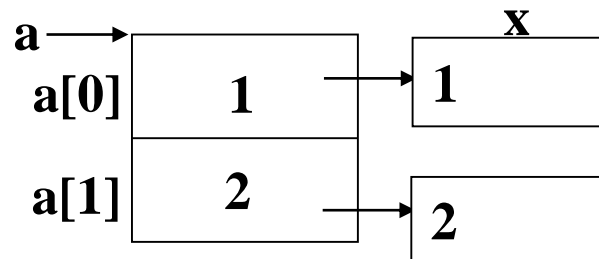
```
#include <stdio.h>
void swap2(int x, int y){
    int z;
    z=x; x=y; y=z;
}
int main( ) {
    int a[2]={1, 2};
    swap2(a[0],a[1]);
    printf("a[0]=%d\n a[1]=%d\n",a[0],a[1]);
}
```

a[0]=1

a[1]=2

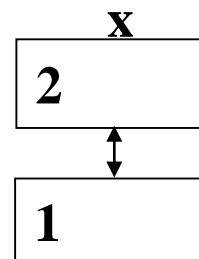


调用前

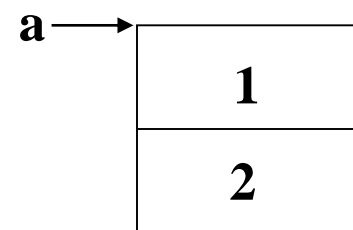


调用

y



y
交换



返回

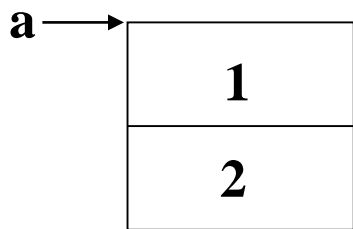
例：数组元素与 数组
名作函数参数比较

地址传递

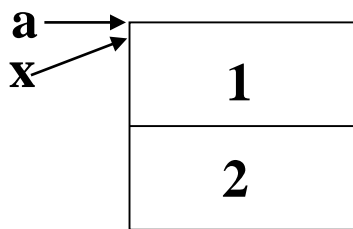
```
#include <stdio.h>
void swap2(int x[]) {
    int z;
    z=x[0]; x[0]=x[1]; x[1]=z;
}
int main() {
    int a[2]={1, 2};
    swap2(a);
    printf("a[0]=%d\n a[1]=%d\n",a[0],a[1]);
}
```

a[0]=2

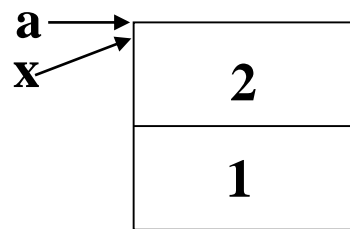
a[1]=1



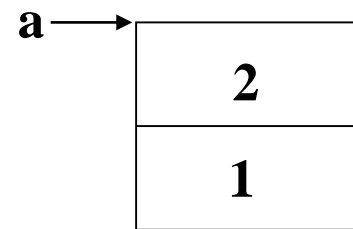
调用前



调用



交换



返回

例: 求二维数组中最大元素值

```
#include <stdio.h>
int MaxValue (int array[3][4]) {
    int i, j, max;
    max = array[0][0];
    for(i=0; i<3; i++)
        for(j=0; j<4; j++)
            if(array[i][j] > max)
                max = array[i][j];
    return max;
}
int main( ) {
    int a[3][4]={1, 3, 5, 7}, {2, 4, 6, 8}, {15, 17, 34, 12}};
    printf("max value is %d\n", MaxValue(a));
}
```

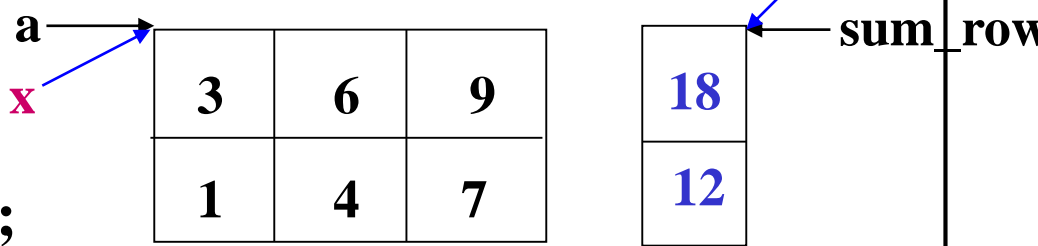
多维形参数组第一维维数
可省略,第二维必须相同
⇔ `int array[][4]`

max value is 34

例：求二维数组中各行元素之和

```
#include <stdio.h>
void get_sum_row (int x[ ][3], int result[ ], int row, int col) {
    int i, j;
    for(i=0; i<row; i++) {
        result[i] = 0;
        for(j=0; j<col; j++)
            result[i] += x[i][j];
    }
}

int main( ) {
    int a[2][3] = {3, 6, 9, 1, 4, 7};
    int sum_row[2], row = 2, col = 3, i;
    get_sum_row(a, sum_row, row, col);
    for(i=0; i<row; i++)
        printf("The sum of row[%d]=%d\n", i+1, sum_row[i]);
}
```



3	6	9
1	4	7

18
12

The sum of row[1]=18
The sum of row[2]=12

形参数组可实现多值返回
如： result[]

几点说明：

1. 数组名作形、实参数时，应分别在主、被调函数中对其声明
2. 形、实参数的数组类型要一致，大小一般相等，以保证数据的全部传送；
3. 当形参数组大小未指定时，用一实参将数组长度传递给形参以便对数组进行操作；
4. 数组名作参数时，传递的是地址，对形参数组的操作实际上也是对实参数组的操作。

字符串指针作函数参数

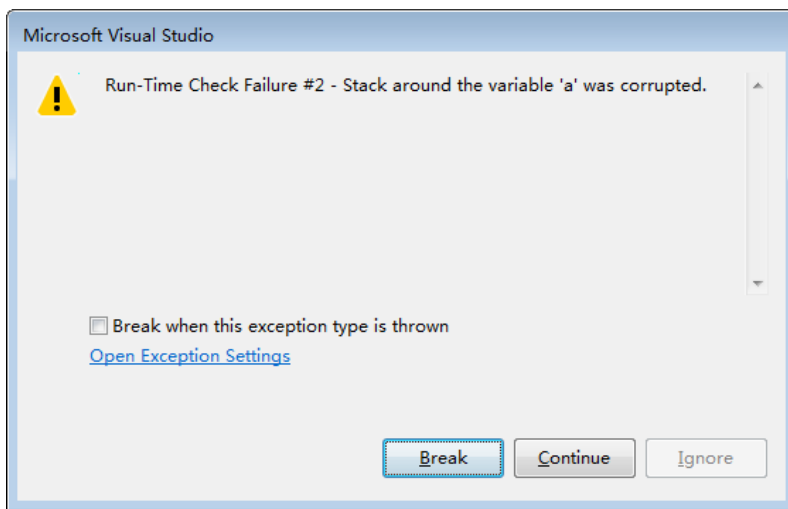
采用地址传递的办法，在被调用的函数中可以改变字符串的内容，在主调函数中可以得到改变了的字符串。

方式1：形参和实参都用字符数组作参数

```
#include <stdio.h>
void strcpy(char from[], char to[])
{
    int i=0;
    while (from[i]!='\0')
    {
        to[i]=from[i];
        i++;
    }
    to[i]='\0';
}
```

```
int main()
{
    char a[ ] = "I am a teacher.";
    char b[ ] = "you are a student.";
    printf("str_a=%s\nstr_b=%s\n", a, b);
    strcpy(a,b);
    printf("\nstr_a=%s\nstr_b=%s\n", a, b);
    return 0;
}
```

```
str_a=I am a teacher.
str_b=you are a student.
str_a=I am a teacher.
str_b=I am a teacher.
```



strcpy()中实参a和b互换会怎样？

```
#include <stdio.h>
void strcpy(char from[], char to[])
{
    int i=0;
    while (from[i]!='\0')
    {
        to[i]=from[i];
        i++;
    }
    to[i]='\0';
}
```

```
int main()
{
    char a[ ] = "I am a teacher.";
    char b[ ] = "you are a student.";
    printf("str_a=%s\nstr_b=%s\n", a, b);
    strcpy(b,a);
    printf("\nstr_a=%s\nstr_b=%s\n", a, b);
    return 0;
}
```

```
str_a=I am a teacher.
str_b=you are a student.
str_a=you are a student.
str_b=you are a student.
```

方式2：形参用字符指针变量，实参用数组名或字符指针。

```
#include <stdio.h>
void strcpy(char *from, char *to)
{
    for (; *from!='\0'; from++, to++)
        *to = *from;
    *to='\0';
}
int main()
{
    char a[ ] = "I am a teacher.";
    char b[ ] = "you are a student.";
    printf("str_a=%s\nstr_b=%s\n", a, b);
    strcpy(a, b);
    printf("\nstr_a=%s\nstr_b=%s\n", a, b);
    return 0;
}
```

```
str_a=I am a teacher.
str_b=you are a student.
str_a=I am a teacher.
str_b=I am a teacher.
```


字符数组 \longleftrightarrow 字符串指针

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


```
int main()
```

```
{  
    char* a = "hello";  
    char b[] = "world!";  
    a = b;  
    puts(a);  
    puts(b);  
    return 0;  
}
```

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

```
int main()
```

```
{  
    char* a = "hello";  
    char b[] = "world!";  
    b = a;  
    puts(a);  
    puts(b);  
    return 0;  
}
```



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

```
#include <string.h>
```

```
int main()
```

```
{  
    char* a = "hello";  
    char b[] = "world!";  
    strcpy(b, a);  
    puts(a);  
    puts(b);  
    return 0;  
}
```

```
world!  
world!
```

```
error C2440: '=' : cannot convert  
from 'char *' to 'char [7]'
```

```
hello  
hello
```

变量的存储属性

- 变量是对程序中数据的**存储空间**的抽象

► 变量的属性

- ✓ **数据类型**: 变量所持有的数据的性质(操作属性)
- ✓ **存储类型**
 - 存储器类型: 寄存器、静态存储区、动态存储区
 - 生存期: 变量在某一时刻存在 (**静态变量**与动态变量)
 - 作用域: 变量在某区域内有效 (局部变量与**全局变量**)

变量的存储属性

► 变量的存储类型

- ① **auto** —— 自动型
- ② **register** —— 寄存器型
- ③ **static** —— 静态型
- ④ **extern** —— 外部型

如: **int** sum;
auto **int** a, b, c;
register **int** i;
static **float** x, y;

► 变量定义格式:

<存储类型> <数据类型> <变量表>;

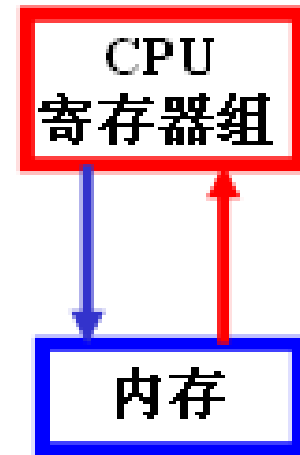
(1) auto 变量

➤ 局部变量——内部变量

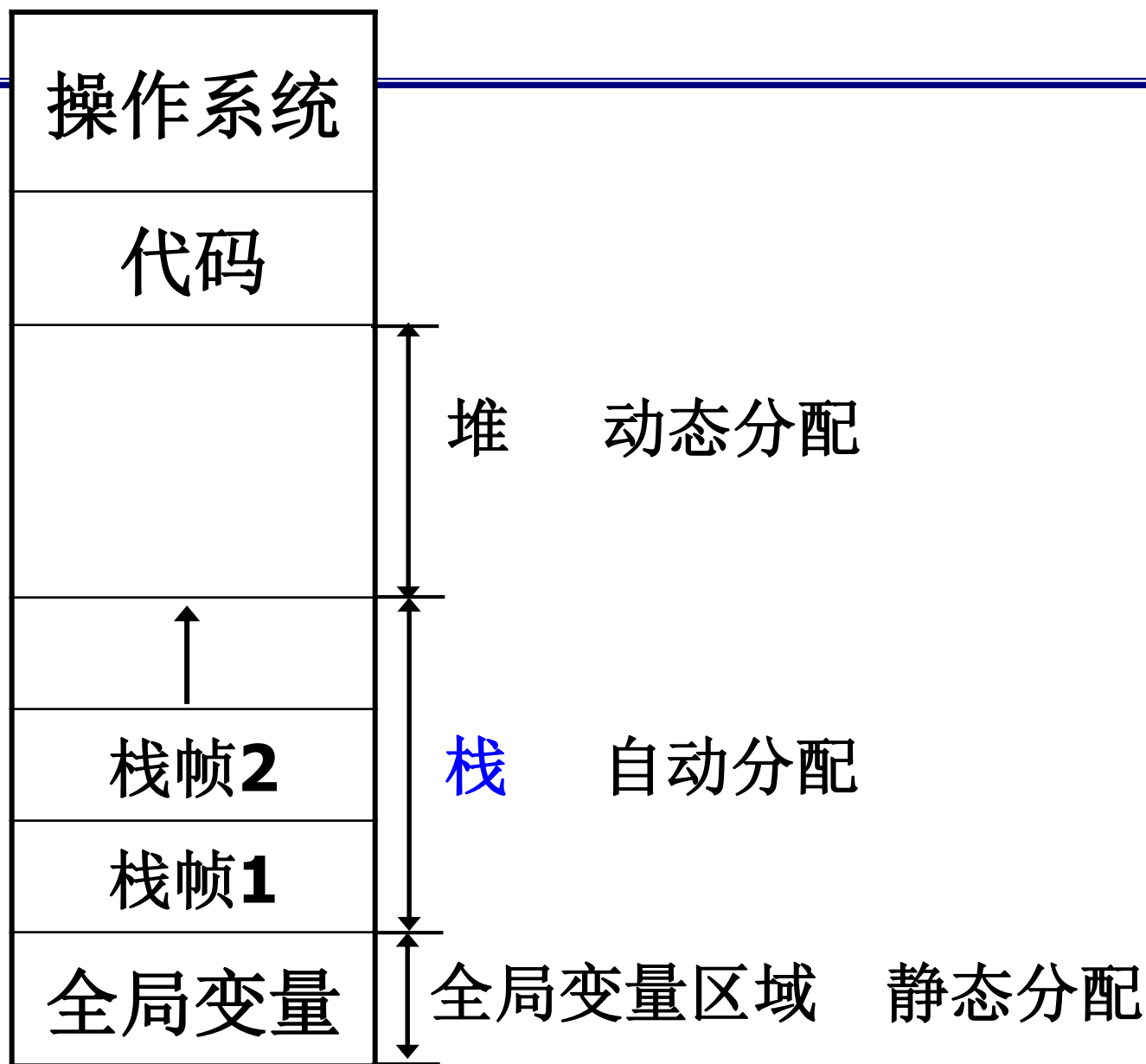
- 定义：在函数内定义, 只在本函数内有效
- main中定义的变量只在main中有效
- 不同函数中同名变量, 占不同内存单元
- 形参属于局部变量
- 可定义在复合语句 ({ }) 中有效的变量。模块以左花括号开始, 以右花括号结束。
- 局部变量可用存储类型: auto register static
(默认为auto)
- 局部变量默认为auto型, 因此auto就几乎很少使用了

(2) register 变量

- CPU内部有寄存器组可用来存放数据，若把数据声明为寄存器（register）类型，则将该类型的数据存放在寄存器中，
- 其优点在于：减少数据与内存之间的交换频率，**提高程序的效率和速度。**



内存分布状况



(3) static 变量

- 用static声明局部变量
- 应用：若希望函数调用结束后，其值不消失，下次调用函数时继续使用，则用static对变量加以声明。

```
#include <stdio.h>
void auto_static (void)
{
    int autoVar = 1;
    static int staticVar = 1;
    printf ("automatic = %i, static = %i\n",
           autoVar, staticVar);

    autoVar++;
    staticVar++;
}
int main (void)
{
    int i;
    for ( i = 0; i < 5; ++i )
        auto_static ();
    return 0;
}
```

automatic = 1, static = 1
automatic = 1, static = 2
automatic = 1, static = 3
automatic = 1, static = 4
automatic = 1, static = 5

局部static变量的声明

- (1) 分配在静态区，**程序运行结束**释放存储单元。
- (2) 仅开始时**赋初值一次**（**未赋初值时为0**），以后每次调用函数时，变量不再赋值，前次操作的结果被保留。
- (3) 局部动态变量若未赋初值，其值是不确定的，所分配的存储单元是不固定的；
而局部静态变量未赋初值，其值为0（字符型变量的值为 `'\0'`），所分配的存储单元是固定的。
- (4) 局部静态变量**在函数调用结束后虽存在，但其它函数不能引用它。**

➤ 使用局部静态变量有如下几种情况

- 需要保留上一次调用结束时的值

- 初始化后变量只被引用而不改变其值，则用静态局部变量较方便，如：`staticVar++`;

- 缺点：从程序运行开始到结束一直占用内存，这样会浪费系统资源。

示例：打印 1~5的阶乘值

```
#include <stdio.h>
int fac(int n)
{
    static int f = 1;
    f = f * n;
    return (f);
}

int main()
{
    int i;
    for(i = 1; i <= 5; i++)
        printf("%d! = %d\n", i, fac(i));
    return 0;
}
```

1! = 1
2! = 2
3! = 6
4! = 24
5! = 120

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

```
int fac(int n)
```

```
{
```

```
    auto int f = 1;
```

```
    f = f * n;
```

```
    return (f);
```

```
}
```

```
int main()
```

```
{
```

```
    int i;
```

```
    for(i = 1; i <= 5; i++)
```

```
        printf("%d! = %d\n", i, fac(i));
```

```
    return 0;
```

```
}
```

将static 改为auto

1! = 1

2! = 2

3! = 3

4! = 4

5! = 5

(4) extern 变量

➤ 全局变量——外部变量

- 定义：在函数外定义, 可为本文件所有函数共用
- 有效范围：从定义变量的位置开始到本源文件结束, 及有extern说明的其它源文件
- 外部变量说明：**extern 数据类型 变量表;**
- 外部变量定义与外部变量说明不同
- 若外部变量与局部变量同名, 则外部变量被屏蔽
- 外部变量可用存储类型：**缺省** 或 **static**

例：用extern扩展外部变量作用域

```
void gx(), gy();
int main() {
    extern int x, y;
    printf("1: x=%d\ty=%d\n", x, y);
    y = 246;
    gx();
    gy();
}
void gx(){
    extern int x, y;
    x=135;
    printf("2: x=%d\ty=%d\n", x, y);
}
int x, y;
void gy(){
    printf("3: x=%d\ty=%d\n", x, y);
}
```

1:	x=0	y=0
2:	x=135	y=246
3:	x=135	y=246

例: 引用其它文件中的外部变量

```
int global;  
extern float x;  
int main()  
{ int local;  
    .  
    .  
    .  
}
```

file1.c

```
extern int global;  
func2()  
{ .  
  .  
  .  
}
```

file2.c

```
float x;  
func3()  
{ extern int global;  
  .  
  .  
  .  
}
```

file3.c

变量存储类型小结

- 局部变量默认为auto型
- register型变量个数受限, 且不能为long, double, float型
- 局部static变量具有全局寿命和局部可见性
- 局部static变量具有可继承性
- extern不是变量定义, 可扩展外部变量作用域

不用死记硬背, 理解上述原则, 重点理解 **static** 即可!

选择题

函数f定义如下，执行语句 “sum = f(5)+f(3);” 后，sum的值应为__C__。

```
int f( int m ) {  
    static int i = 0; int s = 0;  
    for( ; i <= m; i++ )  
        s += i;  
    return s;  
}
```

A、 21

B、 16

C、 15

D、 8

Lecture 6 - Summary

- **Topics covered:**
 - **Defining a Function**
 - **Arguments and Local Variables**
 - **Automatic Local Variables**
 - **Returning Function Results**
 - **Declaring a Function Prototype**
 - **Functions and Arrays**
 - **Arrays as parameters**
 - **Global Variables and Static Variables**