step by step

进阶

起步：

认知与体验（硬件、软件、程序与C语言）

进阶：

判断与推理（流程控制方法、语句）

抽象与联系（模块设计方法、函数）

表达与转换（基本操作、数据类型）

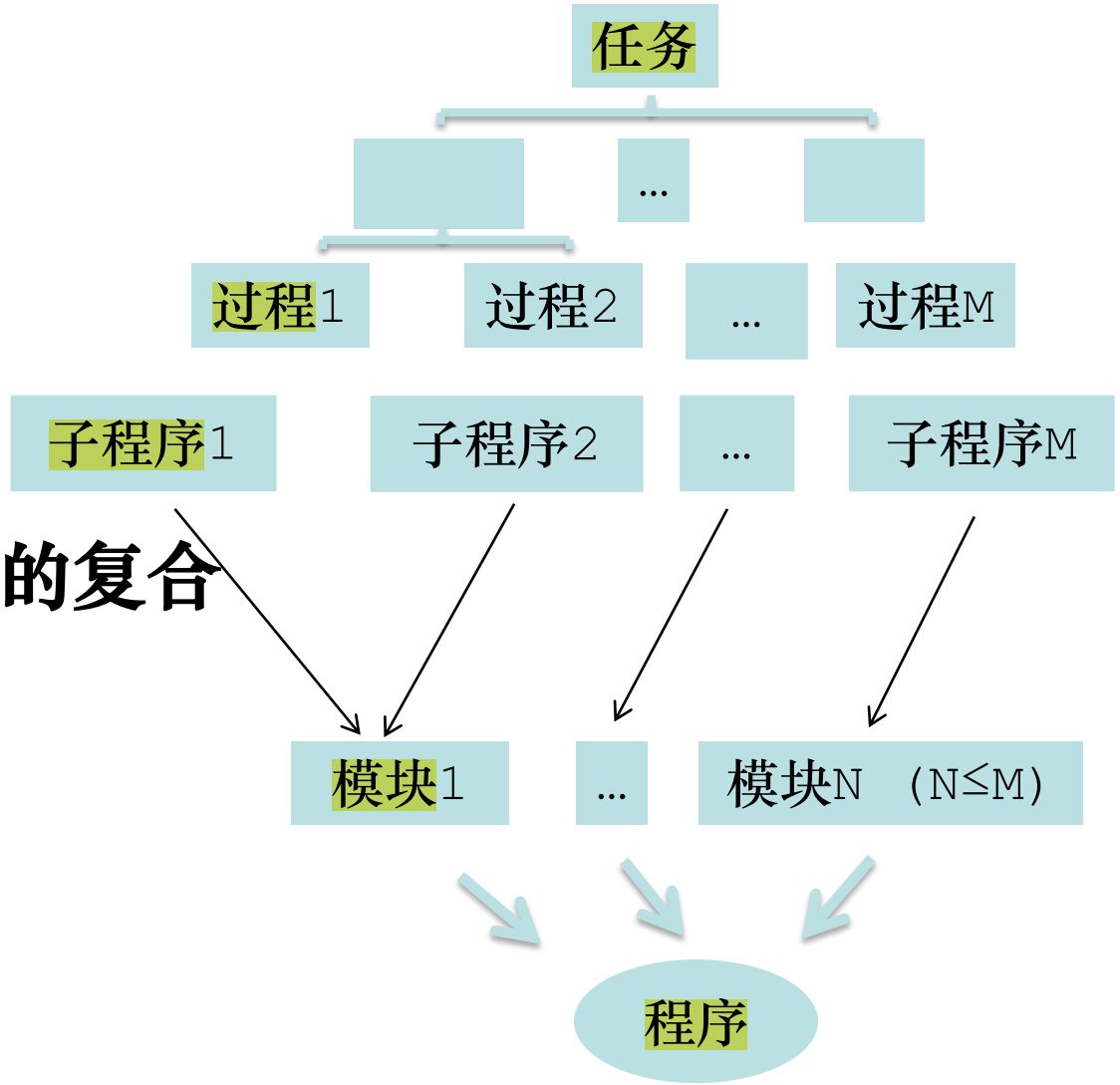
提高：

构造与访问（数组、结构体、指针）

归纳与推广（程序设计的本质）

模块设计

- **过程（procedure）的分解**



- **程序的复合**

从特定的任务实例（3^2+4^2、34^2+65^2）中抽出一般化的功能特征（两个整数的平方和）

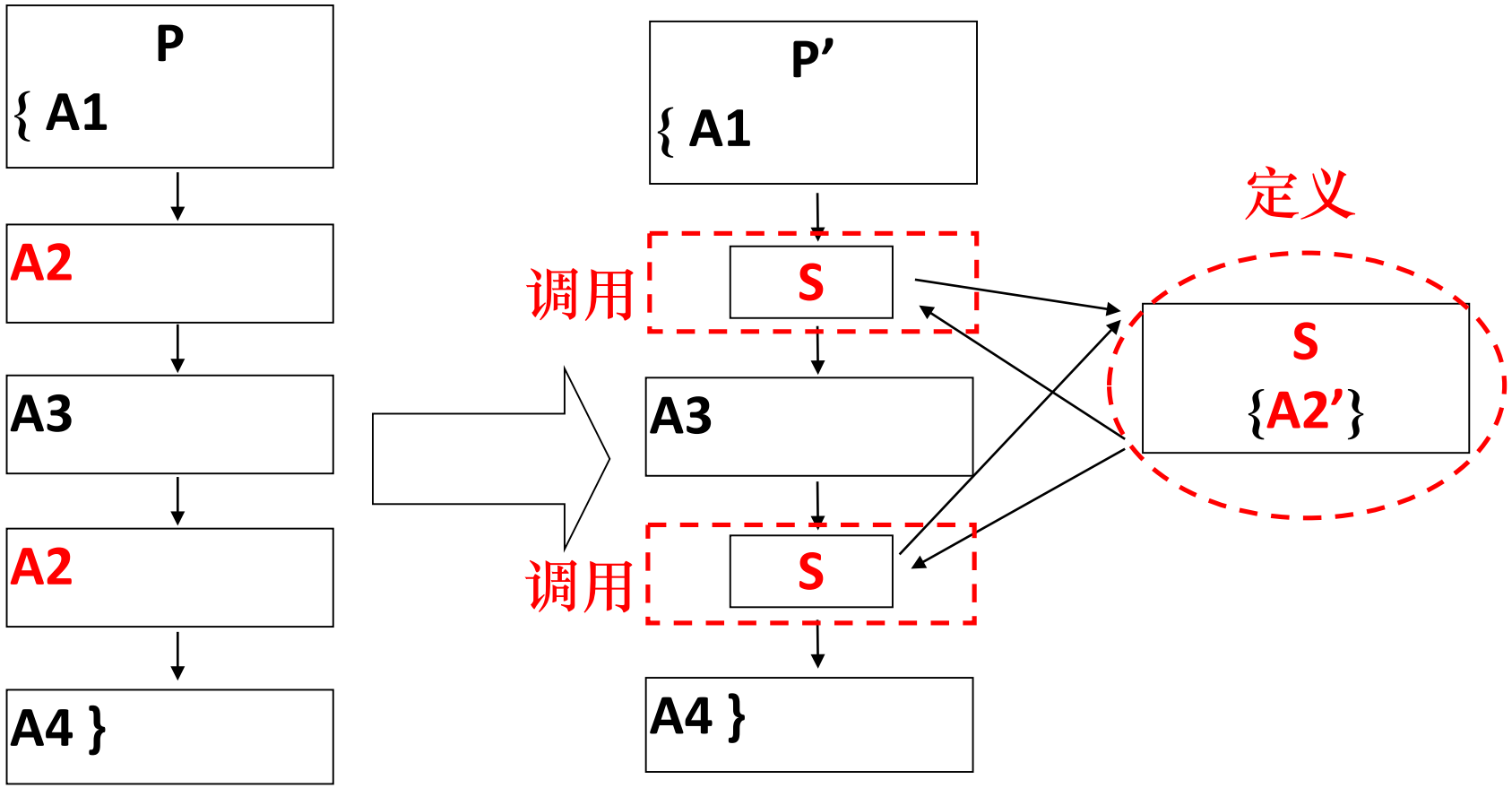
- 过程抽象
- 子程序（subprogram）：封装了一系列操作

发挥头文件的作用，做好每个模块的接口

- 合理安排
- 调用（call）机制：将分布在一个或多个模块（module）中的子程序关联成一个整体

子程序

- 是取了名字的一段程序代码，可以实现一个相对独立的功能



例如：

```
int main( )
{
    int a, b, i, j, c, k;
    scanf("%d%d%d%d", &a, &b, &i, &j);
    c = a*a + b*b;
    k = i*i + j*j;
    printf(…;
    return 0;
}
```

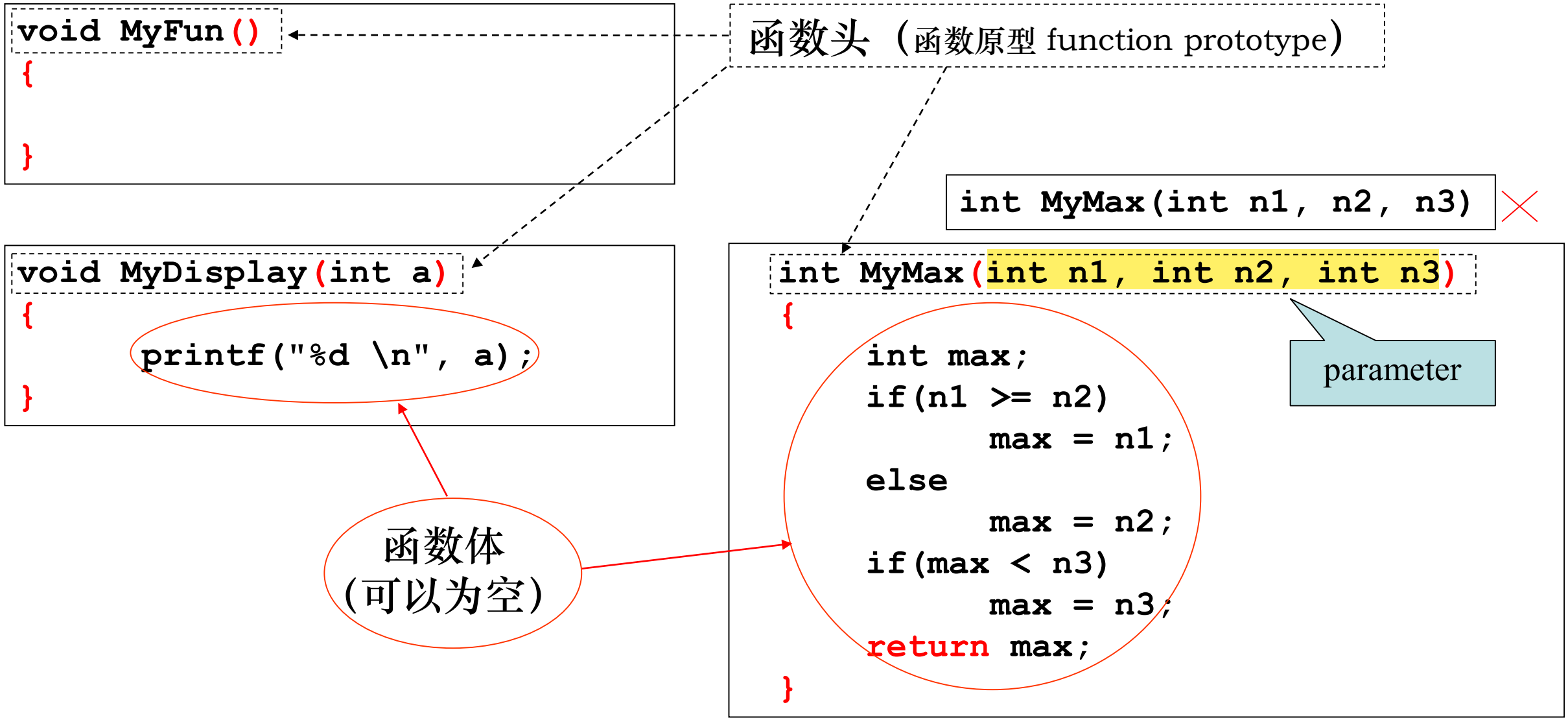
定义

```
int SumSq(int x, int y)
{
      int z;
      z = x*x + y*y;
      return z;
}
```

```
int main( )
 {
    int a, b, i, j, c, k;
      scanf("%d%d%d%d", &a, &b, &i, &j);
      c = SumSq(a, b);
      k = SumSq(i, j);
      printf(…;
      return 0;
}
```

调用

函数的定义(definition)



return语句

```
void MyFun()
{
      return;
}
```

```
void MyDisplay(int a)
{
      printf("%d \n", a);
      return;
}
```

```
int main()
{
      return 0;
}
```

- 一个函数中有多个return语句时，执行到哪一个return语句就从哪儿返回

```
int Min(int a, int b)
{
    int temp ;
    if(a < b)
        temp = a;
    else
        temp = b;
    return temp ;
}
```

```
int Min(int a, int b)
{
    if(a < b)
        return a;
    else
        return b;
}
```

函数定义时的注意事项

```
int main( )
{       int n1, n2, n3;
        printf("Please input three integers: \n");
        scanf("%d%d%d", &n1, &n2, &n3);

        int MyMax(int n1, int n2, int n3)
        {       int max;
                if(n1 >= n2)
                        max = n1;
                else
                        max = n2;
                if(max < n3)
                        max = n3;
                return max;
        } //应把函数 MyMax 的定义写在 main 函数的外面

        printf("The max. is: %d \n", MyMax);
        return 0;
}
```

C程序中的函数体里不能再定义函数

```
int MyMax(int n1, int n2, int n3)
{
      int max;
      if(n1 >= n2)
            max = n1;
      else
            max = n2;
      if(max < n3)
            max = n3;
      return max;
}
int main( )
{
      int n1, n2, n3;
      printf("Please input three integers: \n");
      scanf("%d%d%d", &n1, &n2, &n3);
      int max = MyMax(n1, n2, n3);
      printf("The max. is: %d \n", max);
      return 0;
}
```

定义应该各自独立

函数的调用(call)

```
void MyFun()
{

}
```

```
int main( )
{
     MyFun();
     MyDisplay( 7 );
     return 0;
}
```

argument

```
void MyDisplay(int a)
{
     printf("%d \n", a);
}
```

```
int x = MyFun();
printf("%d", MyFun());
```

```
int MyMax(int n1, int n2, int n3)
{
    int max;
    if(n1 >= n2)
        max = n1;
    else
        max = n2;
    if(max < n3)
        max = n3;
    return max;
}
```

```
int main( )
{
    int i = 3, j = 4, k = 5;
    int m = MyMax(i, j, k);
    printf("%d ", m);
    return 0;
}
```

```
int m = int MyMax(int i, int j, int k);
```

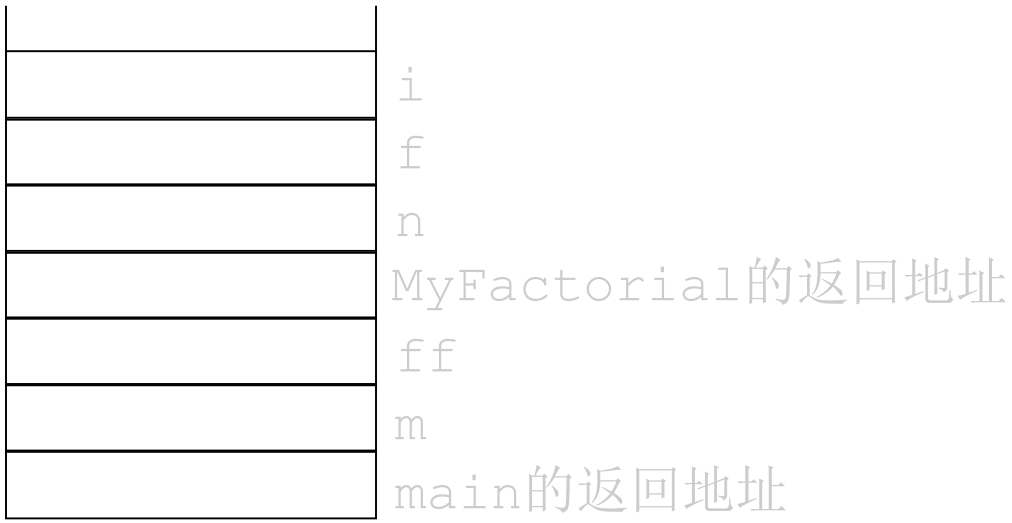
```
int main( )
{
    int i = 3, j = 4, k = 5;
    printf("%d", MyMax(i, j, k));
    return 0;
}
```

例3.1 设计 C 程序，用函数实现 求阶乘问题。

- 分析：求阶乘作为一个独立的功能，用函数实现时，函数的参数为一个整数，函数的返回值为该整数的阶乘。

```
int main( )
{
        int m;
        printf("Input an integer: \n");
        scanf("%d", &m);
        if(m < 0) return -1; //结束整个程序
        int ff = MyFactorial(m);
        printf("Factorial is: %d \n", ff);
        return 0;
}
```

```
int MyFactorial(int n)
{
        int f = 1;
        for(int i = 2; i <= n; ++i)
                f *= i;
        return f;
}
```



例3.2 求输入三个整数中最大值的阶乘，并输出。

```
int main( )
{
      int n1, n2, n3, max, f;
      printf("Input three integers: \n");
      scanf("%d%d%d", &n1, &n2, &n3);
      max = MyMax(n1, n2, n3);
      f = MyFactorial(max);
      printf("Factorial of max. is
      return 0;
}
```

```
int MyFactorial(int n)
{       int f = 1;
        for(int i = 2; i <= n; ++i)
                f *= i;
        return f;
}
```

```
int MyMax(int n1, int n2, int n3)
{       int max;
        if(n1 >= n2)
                max = n1;
        else
                max = n2;
        if(max < n3)
                max = n3;
        return max;
}
```

函数调用结果的不同“身份”（作为赋值操作右操作数或实参）

```
f = MyFactorial(MyMax(n1, n2, n3));
printf("Factorial of max. is: %d \n", f);
```

或

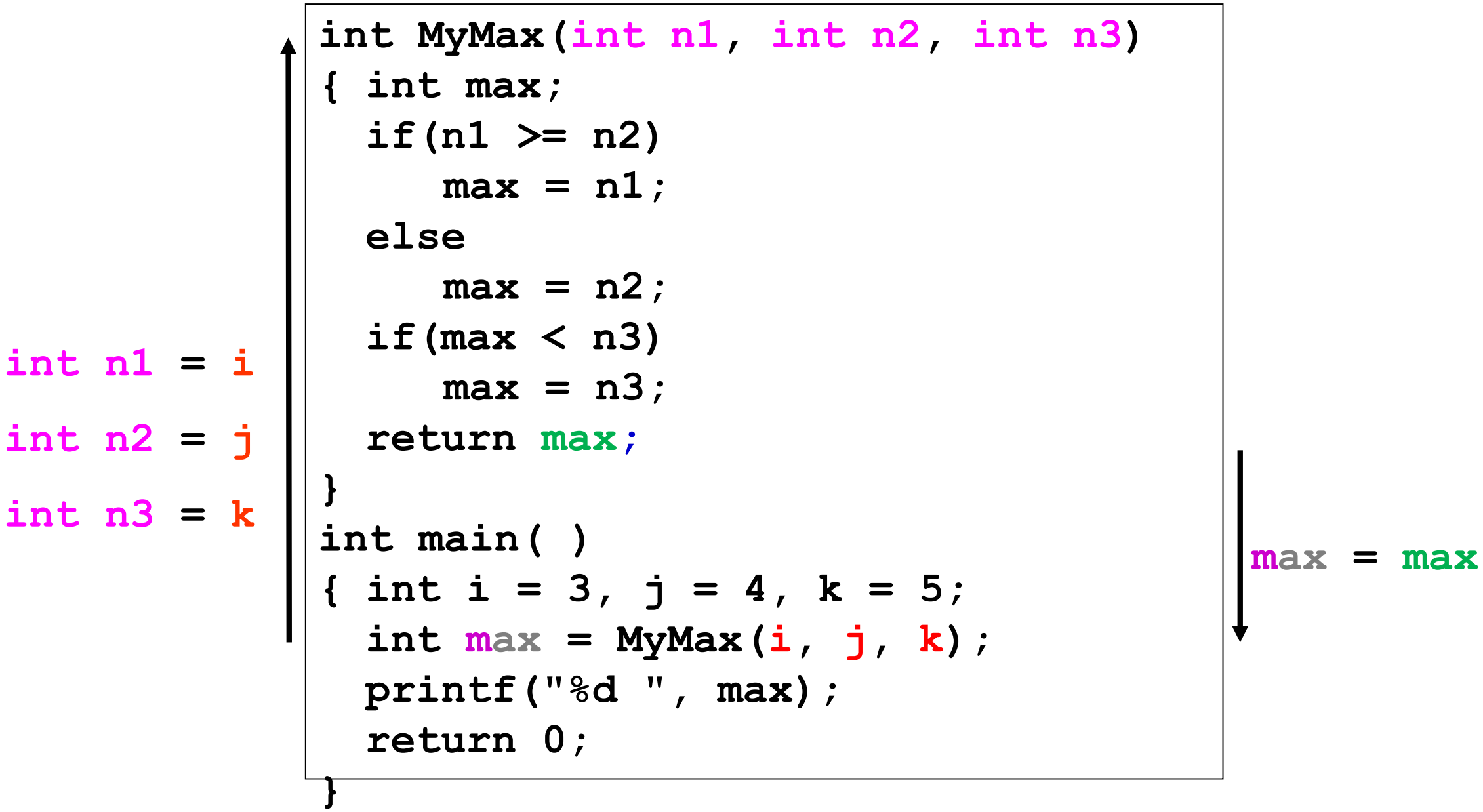
```
max = MyMax(n1, n2, n3);
printf("Factorial of max. is: %d \n", MyFactorial(max));
```

或

```
printf("…is: %d \n", MyFactorial(MyMax(n1, n2, n3)));
```

在上述不同的函数调用形式中，main函数都是先调用 MyMax 函数，在 MyMax 函数执行结束后，再调用 MyFactorial 函数的。

函数间的通讯方式I


int n1 = i
int n2 = j
int n3 = k
int MyMax(int n1, int n2, int n3)
{ int max;
if(n1 >= n2)
max = n1;
else
max = n2;
if(max < n3)
max = n3;
return max;
}
int main( )
{ int i = 3, j = 4, k = 5;
int max = MyMax(i, j, k);
printf("%d ", max);
return 0;
}
max = max

函数的声明(declaration)及其好处

```
int MyMax(int n1, int n2, int n3);//函数的声明

int main( )
{       int n1, n2, n3;
        printf("Please input three integers: \n");
        scanf("%d%d%d", &n1, &n2, &n3);
        int max = MyMax(n1, n2, n3);  //函数的调用
        printf("The max. is: %d \n", max);
        return 0;
}
int MyMax(int n1, int n2, int n3) //函数的定义
{       int max;
        if(n1 >= n2)
                max = n1;
        else
                max = n2;
        if(max < n3)
                max = n3;
        return max;
}
```

全局变量

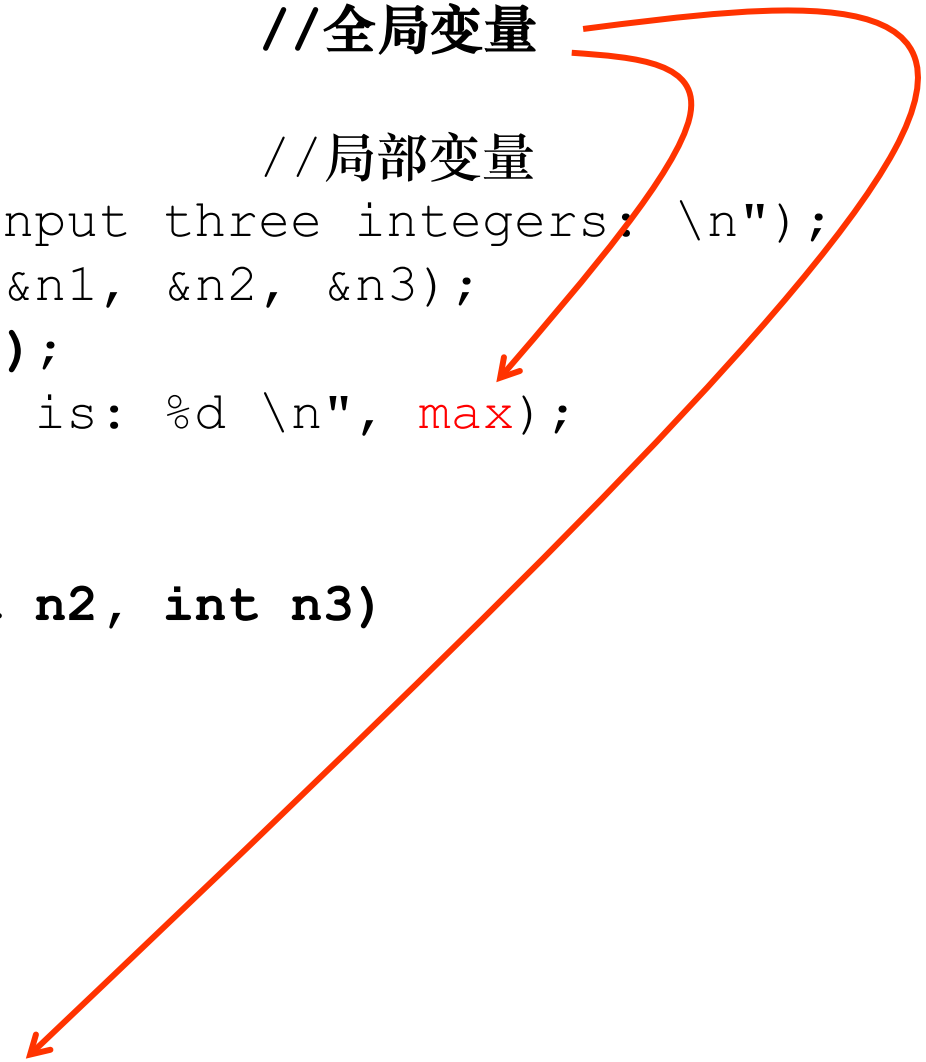
```
void MyMax(int n1, int n2, int n3);
int max = 0;                          //全局变量
int main( )
{       int n1, n2, n3;               //局部变量
        printf("Please input three integers: \n");
        scanf("%d%d%d", &n1, &n2, &n3);
        MyMax(n1, n2, n3);
        printf("The max. is: %d \n", max);
        return 0;
}
void MyMax(int n1, int n2, int n3)
{
        if(n1 >= n2)
                max = n1;
        else
                max = n2;
        if(max < n3)
                max = n3;
}
```

全局变量的声明（declaration）

```
void MyMax(int n1, int n2, int n3);
extern int max;                              //全局变量的声明
int main( )
{       int n1, n2, n3;
        printf("Please input three integers: \n");
        scanf("%d%d%d", &n1, &n2, &n3);
        MyMax(n1, n2, n3);
        printf("The max. is: %d \n", max);
        return 0;
}
int max = 0;                                 //全局变量的定义
void MyMax(int n1, int n2, int n3)
{
        if(n1 >= n2)
                max = n1;
        else
                max = n2;
        if(max < n3)
                max = n3;
}
```

函数间的通讯方式II

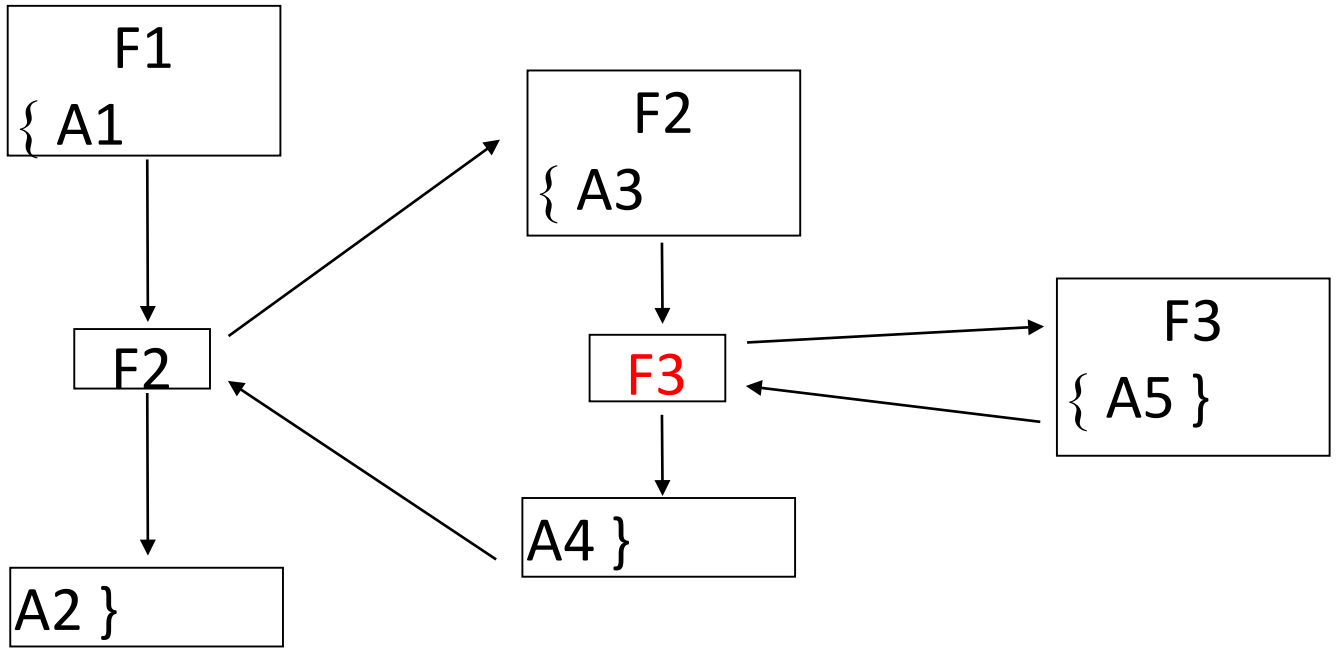
```
void MyMax(int n1, int n2, int n3);
int max = 0;                          //全局变量
int main( )
{       int n1, n2, n3;               //局部变量
        printf("Please input three integers: \n");
        scanf("%d%d%d", &n1, &n2, &n3);
        MyMax(n1, n2, n3);
        printf("The max. is: %d \n", max);
        return 0;
}
void MyMax(int n1, int n2, int n3)
{
        if(n1 >= n2)
                max = n1;
        else
                max = n2;
        if(max < n3)
                max = n3;
}
```



函数的副作用

函数的嵌套调用

- 即被调函数的定义中含有函数的调用操作



被调函数 F2 中含有函数 F3 的调用操作

不含嵌套调用

- 例3.2 求输入三个整数中最大值的阶乘，并输出。

```
int main( )
{
      int n1, n2, n3, max, f;
      printf("Input three integers: \n");
      scanf("%d%d%d", &n1, &n2, &n3);
      max = MyMax(n1, n2, n3);
      f = MyFactorial(max);
      printf("Factorial of max. is
      return 0;
}
```

```
int MyFactorial(int n)
{       int f = 1;
        for(int i = 2; i <= n; ++i)
                f *= i;
        return f;
}
```

```
int MyMax(int n1, int n2, int n3)
{       int max;
        if(n1 >= n2)
                max = n1;
        else
                max = n2;
        if(max < n3)
                max = n3;
        return max;
}
```

含嵌套调用

- 例3.2’求输入三个整数中最大值的阶乘，并输出。

```
int main( )
{
      int n1, n2, n3, f;
      printf("Input three integers: \n");
      scanf("%d%d%d", &n1, &n2, &n3);
      f = MyFactorialNew(n1, n2, n
      printf("Factorial of max. is
      return 0;
}
```

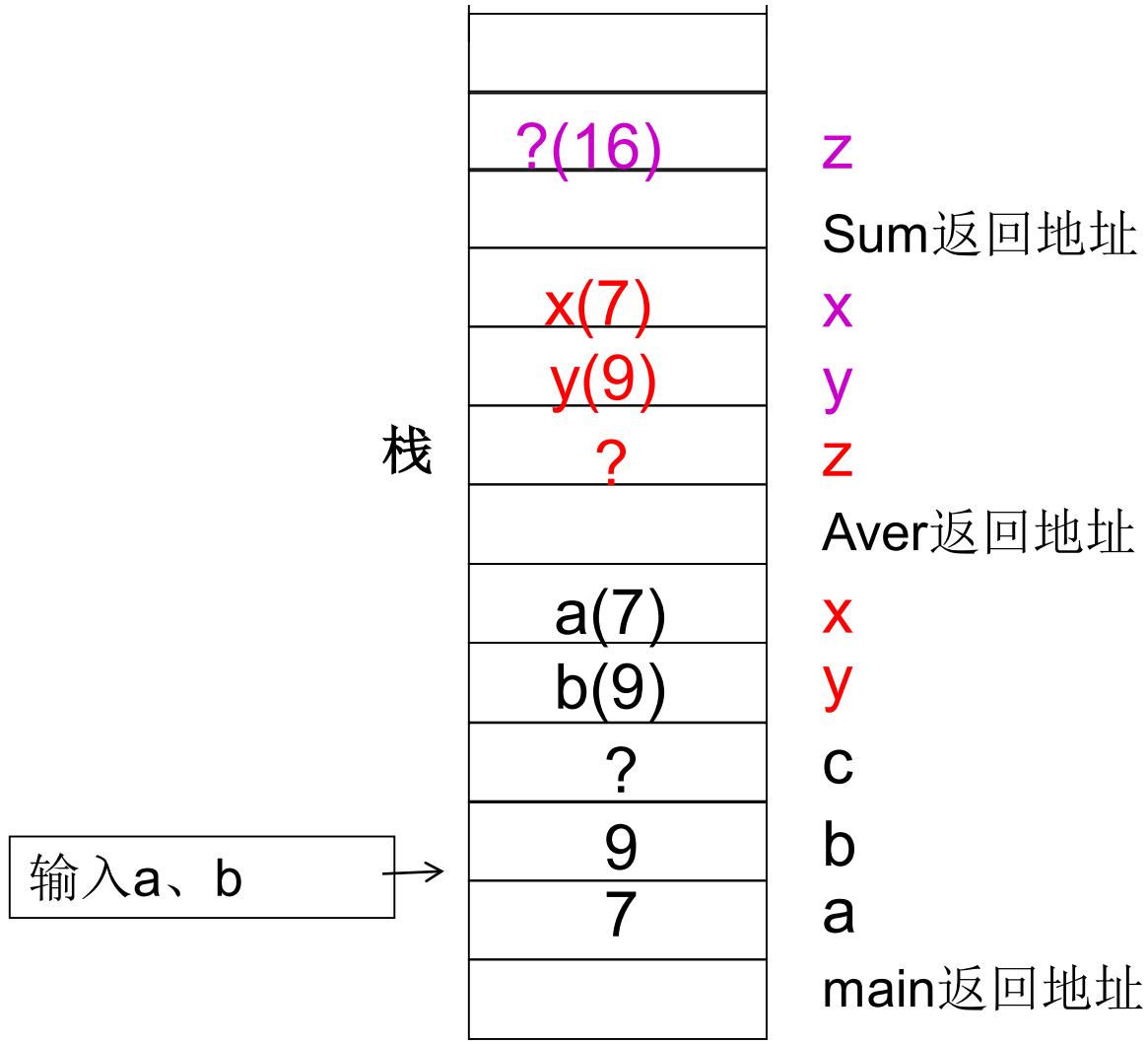
```
f = MyFactorial(MyMax(n1, n2, n3));
```

```
int MyFactorialNew(int n1, int n2, int n3)
{       int n = MyMax(n1, n2, n3);
        int f = 1;
        for(int i = 2; i <= n; ++i)
                f *= i;
        return f;
}
```

```
int MyMax(int n1, int n2, int n3)
{       int max;
        if(n1 >= n2)
                max = n1;
        else
                max = n2;
        if(max < n3)
                max = n3;
        return max;
}
```

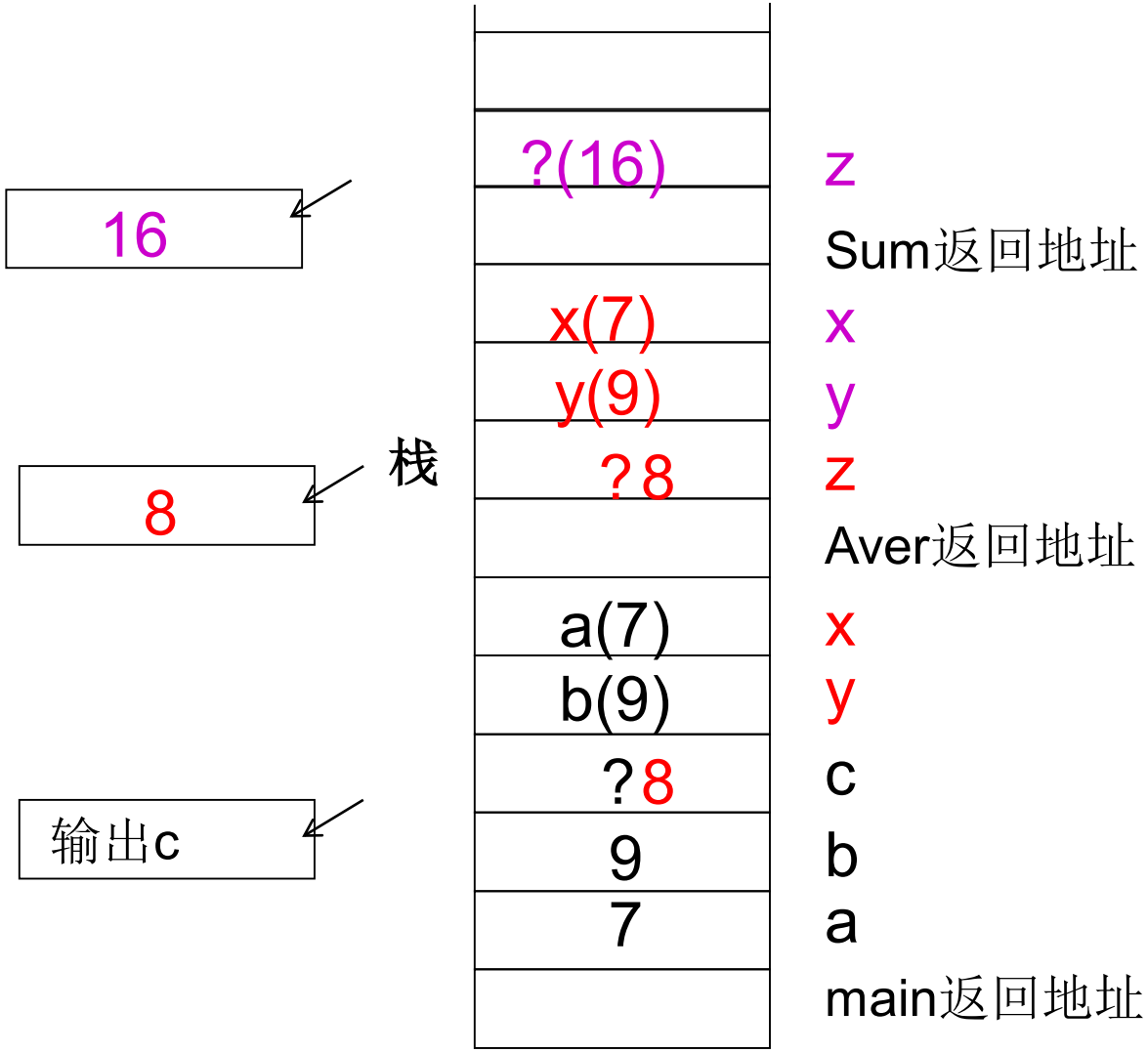
C函数嵌套调用的过程

```
int Sum(int x, int y)
{   int z;
    z = x + y;
    return z;
}

int Aver(int x, int y)
{   int z;
    z = Sum(x, y) / 2;
    return z;
}

int main()
{   int a, b, c;
    scanf("%d%d",&a, &b);
    c = Aver(a, b);
    printf("%d", c);
    return 0;
}
```



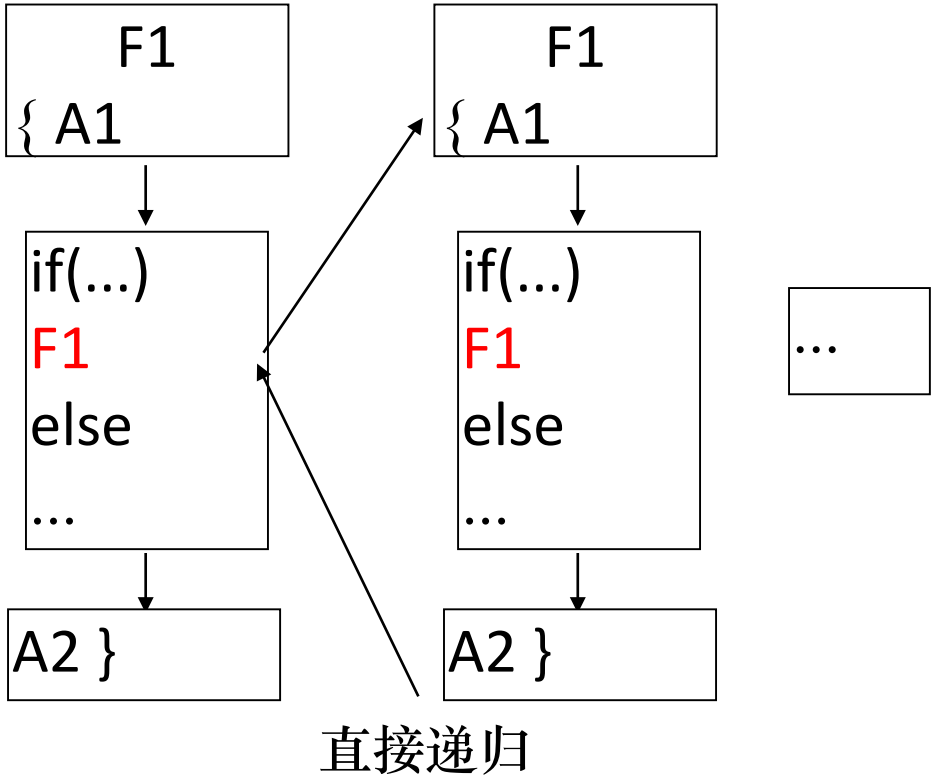
C函数嵌套调用过程（续）

```
int Sum(int x, int y)
{   int z;
    z = x + y;
    return z;
}

int Aver(int x, int y)
{   int z;
    z = Sum(x, y) / 2;
    return z;
}

int main()
{   int a, b, c;
    scanf("%d%d",&a, &b);
    c = Aver(a, b);
    printf("%d", c);
    return 0;
}
```

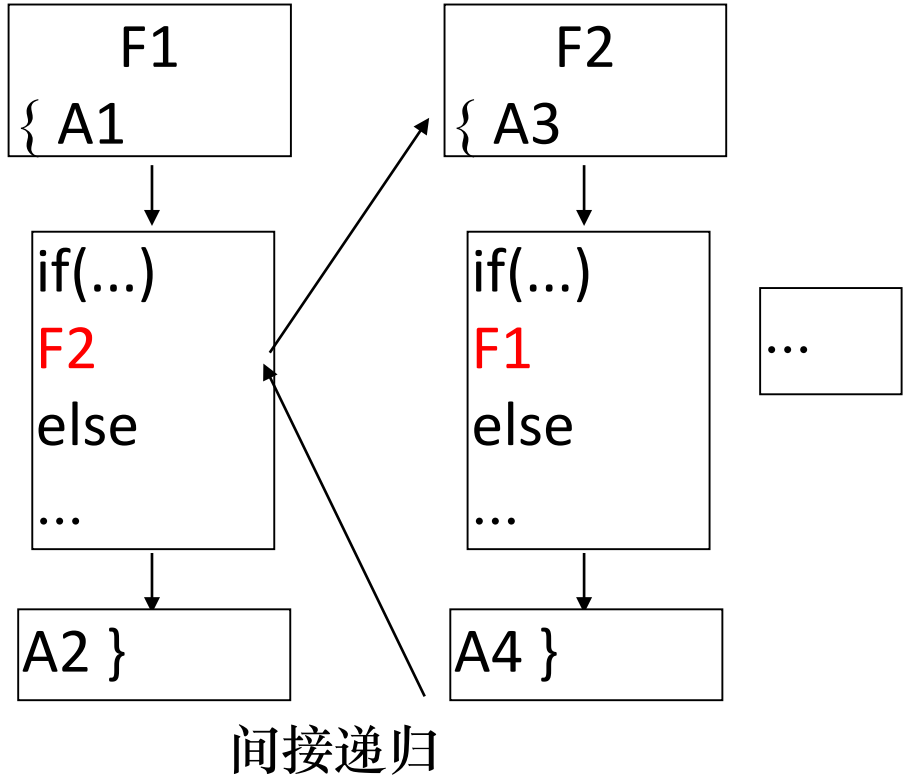


函数的递归（recursion）调用

- 即被调函数的定义中含有本函数的调用操作



直接递归



间接递归

- 例3.3 设计 C 程序，用递归调用的函数实现 求阶乘问题。
  - 分析：

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

```
int MyFactorial(int n)
{
        int f = 1;
        for(int i = 2; i <= n; ++i)
                f *= i;
        return f;
}
```

或

```
int MyFactorialR(int n)
{
        if(n==0 || n==1)
                return 1;
        else
                return n * myFactorialR(n-1);
}
```

递归调用函数的执行过程

```
int MyFactorialR(int n)
{
        if(n==0 || n==1)
                return 1;
        else
                return n * myFactorialR(n-1);
}
```

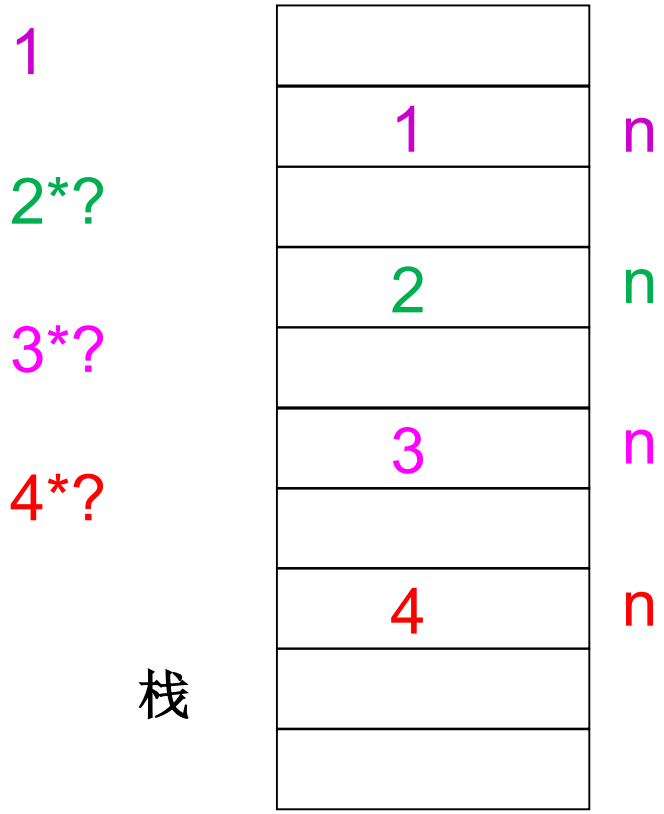
```
int main()
{       printf("%d", myFactorialR(4));
        return 0;
}
```

实例一 => 4 * myFactorialR(3)

实例二 => 3 * myFactorialR(2)

实例三 => 2 * myFactorialR(1)

实例四 => 1



递归调用函数的执行过程

```
int MyFactorialR(int n)
{
        if(n==0 || n==1)
                return 1;
        else
                return n * myFactorialR(n-1);
}
```

```
int main()
{       printf("%d", myFactorialR(4));
        return 0;
}
```

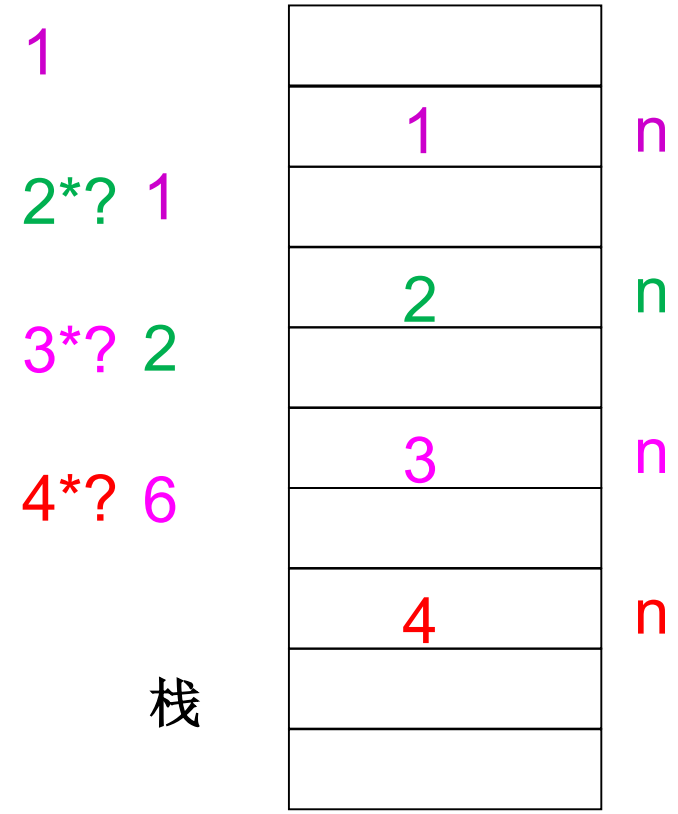
24

实例一 => 4 * 6

实例二 => 3 * 2

实例三 => 2 * 1

实例四 => 1

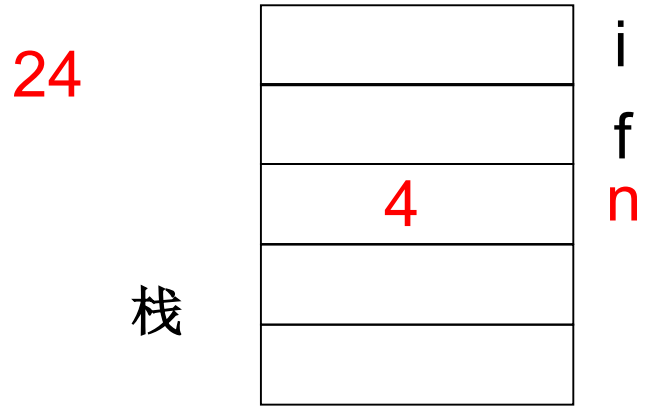


迭代法函数的执行过程

```
int MyFactorial(int n)
{
        int f = 1;
        for(int i = 2; i <= n; ++i)
                f *= i;
        return f;
}
```

```
int main()
{       printf("%d", myFactorial(4));
        return 0;
}
```

24



斐波那契Fibonacci数列：有一对兔子，从出生后第3个月起每个月生一对兔子，小兔子长到第3个月后每个月又生一对兔子，假设所有兔子都不死，求第n个月的兔子总对数。

1，1，2，3，5，8，13，...

Fibonacci 数的定义：

$$fib(n) = \begin{cases} 1 & (n=1) \\ 1 & (n=2) \\ fib(n-2) + fib(n-1) & (n \geq 3) \end{cases}$$

```
int main
{
    int n;
    scanf("%d", &n);
    printf("第 %d 个月有 %d 对兔子.\n", n, temp);
    return 0;
}
```

```
int MyFib(int n)
{
int fib1 = 1, fib2 = 1, temp = 1;
for(int i = 3; i <= n; ++i)
{       temp = fib1 + fib2;
        fib1 = fib2 ;
        fib2 = temp;
}
return temp;
}
```

```
int MyFibR(int n)
{
    if(n == 1 || n == 2)
       return 1;
    else
       return MyFibR(n-2) + MyFibR(n-1);
}
```

推广：

$$fib(n) = \begin{cases} 1 & (n < m) \\ fib(n-m+1) + fib(n-1) & (n \geq m) \end{cases}$$

```
int Fib(int n, int m)
{
    if(n < m)
       return 1;
    else
       return Fib(n-m+1, m) + Fib(n-1, m);
}
```

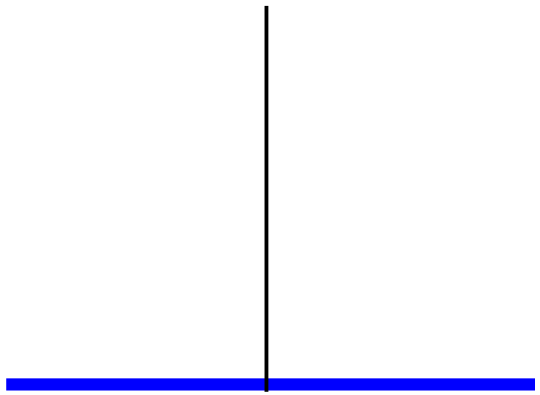
有一头小母牛，从出生后第 4 年起每年生一头母牛，小母牛长到第 4 年后每年又生一头母牛，假设所有母牛都不死，求第 n 年的母牛头数。

1，1，1，2，3，4，6，9,…

```
int FibCow(int n)
{   int fib1=1, fib2=1, fib3=1, temp;
    if(n < 4)
        return 1;
    else
    {
        for(int i = 4; i <= n; ++i)
        {   temp = fib1 + fib3;
            fib1 = fib2;
            fib2 = fib3;
            fib3 = temp;
        }
        return temp;
    }
}
```

- **例3.4 设计 C 程序，用递归调用的函数求解 河内塔（Tower of Hannoi，Tower of Brahma，Lucas’ Tower）问题。**

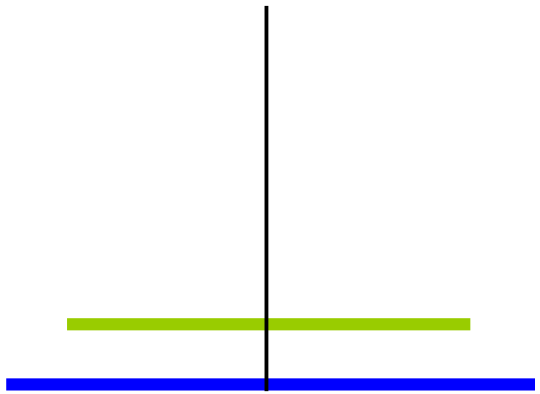
A → C



原来-A

借助-B

目标-C



原来-A

借助-B

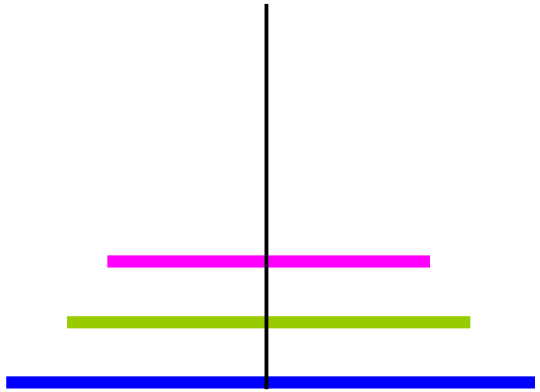
目标-C

A → B
A → C
B → C

A → C
A → B
C → B

A → C

B → A
B → C
A → C



原来-A

借助-B

目标-C

原来-A
借助-A

目标-B
原来-B

借助-C
目标-C

- 分析：

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

$$= 2 \cdot (2 \cdot (2 \cdot (\ldots) + 1) + 1) + 1$$
$$= 2^n - 1$$

（1）当 n 为 1 时：x → z

（2）当 n 大于 1 时：

① n-1 个盘子：x → y （借助 z ）

② 第 n 个盘子：x → z

③ n-1 个盘子：y → z （借助 x ）

Hanoi('A', 'B', 'C', n);

```
void Hanoi(char x, char y, char z, int n)
{
    if(n == 1)
        printf("%c → %c \n", x, z);
    else
    {
        Hanoi(x, z, y, n-1);
        printf("%c → %c \n", x, z);
        Hanoi(y, x, z, n-1);
    }
}
```

```
Hanoi('A', 'B', 'C', 3);
```

```
void Hanoi(char x, char y, char z, int n)
{   if(n == 1)
        printf("%c → %c \n", x, z);
    else
    {
        Hanoi(x, z, y, n-1);
        printf("%c → %c \n", x, z);
        Hanoi(y, x, z, n-1);
    }
}
```

```
printf("A → C\n");
printf("A → B\n");
printf("C → B\n");
```

```
printf("A → C\n");
```

```
printf("B → A\n");
printf("B → C\n");
printf("A → C\n");
```

```
Hanoi('A', 'C', 'B', 2);
printf("%c → %c \n", x, z);
Hanoi('B', 'A', 'C', 2);
```

```
Hanoi('A', 'B', 'C', 1);
printf("A → B\n");
Hanoi('C', 'A', 'B', 1);
```

```
printf("A → C\n");
```

```
Hanoi('B', 'C', 'A', 1);
printf("B → C\n");
Hanoi('A', 'B', 'C', 1
```

单模块（Single module）

多模块（Multiple modules）

起步：

认知与体验（硬件、软件、程序与C语言）

进阶：

判断与推理（流程控制方法、语句）

抽象与联系（模块设计方法、函数）

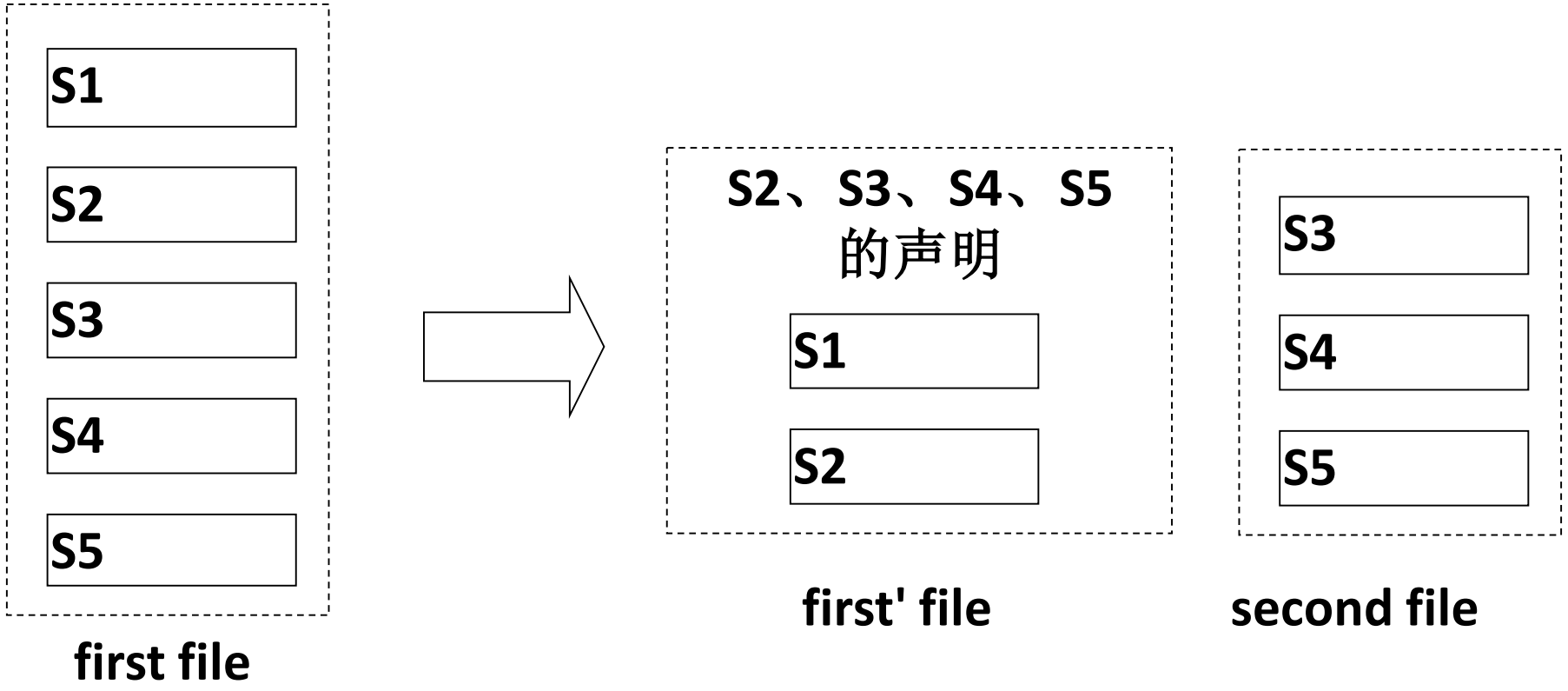
表达与转换（基本操作、数据类型）

提高：

构造与访问（数组、结构体、指针）

归纳与推广（程序设计的本质）

- **一个源文件通常被看成一个模块，是一个独立的编译单元。**



S1调用S2、S3，S2调用S4、S5

例3.5 甲乙两人共同开发一个程序，实现求最大数的阶乘功能。

```
//first.c 甲
#include <stdio.h>
int myMax(int, int, int);
extern int myFactorial(int);  //声明乙编写的函数
int main()
{    int n1, n2, n3, max, f;
     printf("Please input three integers: \n");
     scanf("%d", &n1, &n2, &n3);
     max = myMax(n1, n2, n3);
     f = myFactorial(max);
     printf …
     return 0;
}

int myMax(int n1, int n2, int n3)
{     …     }
```

```
//second.c 乙
int myFactorial(int n)
{    int f = 1;
     for(int i=2; i <= n; ++i)
            f *= i;
     return f;
}
```

文件包含预处理（preprocess）命令与头文件（head file）

```c
//first.c 甲
#include <stdio.h>
int myMax(int, int, int);
#include "second.h" //包含乙编写的头文件
int main()
{    int n1, n2, n3, max, f;
     printf("Please input three integers: \n");
     scanf("%d", &n1, &n2, &n3);
     max = myMax(n1, n2, n3);
     f = myFactorial(max);
     printf …
     return 0;
}

int myMax(int n1, int n2, int n3)
{     …     }
```

```c
//second.h
extern int myFactorial(int);
```

```c
//second.c 乙
int myFactorial(int n)
{    int f = 1;
     for(int i=2; i <= n; ++i)
          f *= i;
     return f;
}
```

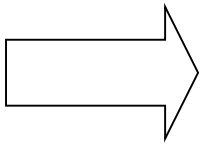
- 头文件里还可以放全局变量的声明等内容

//xlib.c

```
int x, y;
int F()
{     x = 1000;
      y = x*x;
      return y;
}
```

//my.c

```
extern int x, y;
int F();
int G()
{     int z;
      F();
      z = x + y;
      return z;
}
```



//xlib.c

```
int x, y;
int F()
{     x = 1000;
      y = x*x;
      return y;
}
```

//my.c

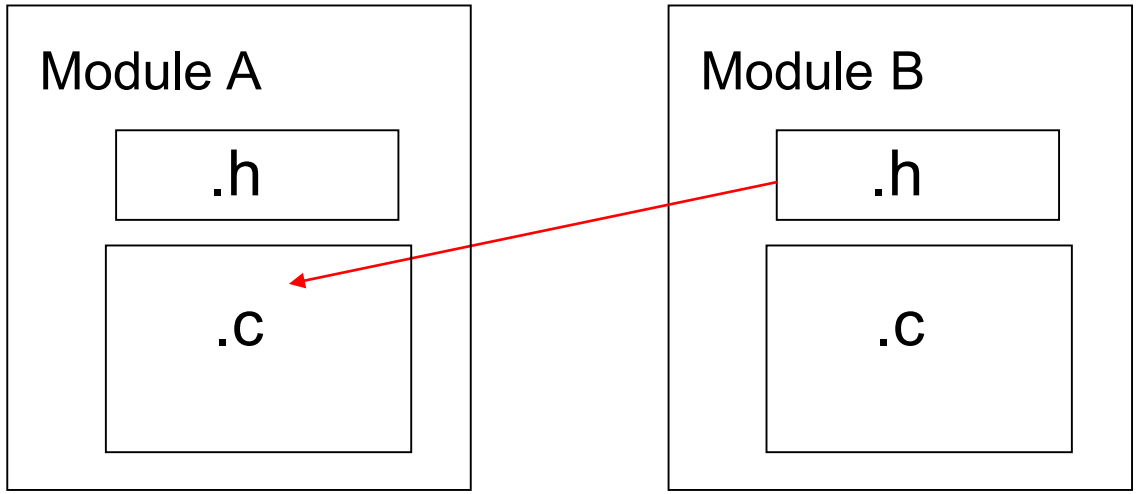
```
#include "xlib.h"
int G()
{     int z;
      F();
      z = x + y;
      return z;
}
```

//xlib.h

```
extern int x, y;
int F();
```

模块 = 接口 + 实现

- 接口（interface，.h 文件）
  - 在本模块中定义的、提供给其他模块使用的函数等程序实体的声明
- 实现（implementation，.c 文件）
  - 函数等程序实体的定义



标识符的属性

- **作用域（scope）**
  - 链接（linkage）

标识符的有效范围

- **存储期（storage duration）**

文件作用域

```
// second.c
extern int s; //从声明之后有效
void MySum(int n) //从定义开始有效
{
   int sum = 0;
   for(int i=1; i <= n; ++i)
        sum += i;
   s = sum;
}
```

```
// first.c
#include <stdio.h>
int s = 0; //从定义开始有效
extern void MySum(int); //从声明之后有效
int main()
{     int n;
      printf("Please input an integer: \n");
      scanf("%d", &n);
      if(n <= 0) goto L1;
      MySum(n);
L1:   printf("s = %d \n",  s);
      return 0;
}
```

文件作用域

```
// second.c
static void MySum(int n) //从定义开始有效
{
        int sum = 0;
        for(int i=1; i <= n; ++i)
                sum += i;
        printf("%d", sum);
}
```

```
// first.c
#include <stdio.h>

static int s = 0; //从定义开始有效
int main()
{     int n;
      printf("Please input an integer: \n");
      scanf("%d", &n);
      if(n <= 0) goto L1;
      s = ... ;
L1:   printf("s = %d \n", s);
      return 0;
}
```

文件作用域

```
// first.c
#include <stdio.h>
int s = 0; //从定义开始有效
void MySum(int); //从声明之后有效
int main()
{
    int n;
    printf("Please input an integer: \n");
    scanf("%d", &n);
    if(n <= 0) goto L1;
    MySum(n);
L1: printf("s = %d \n", s);
    return 0;
}
```

```
void MySum(int n)
{
    int sum = 0;
    for(int i=1; i <= n; ++i)
        sum += i;
    s = sum;
} //写在 first.c 文件后部
```

块作用域

```
// first.c
#include <stdio.h>
int s = 0;
void MySum(int);
int main()
{
    int n;                  //n
    printf("Please input an integer: \n");
    scanf("%d", &n);
    if(n <= 0) goto L1;
    MySum(n);
L1: printf("s = %d \n", s);
    return 0;
}
```

```
void MySum(int n)                       //n
{
    int sum = 0;                        //sum
    for(int i=1; i <= n; ++i)           //i
    {     sum += i;          }
    s = sum;
} //写在 first.c 文件后部
```

函数作用域

```
// first.c
#include <stdio.h>
int s = 0;
void MySum(int n);
int main()
{
    int n;
    printf("Please input an integer: \n");
    scanf("%d", &n);
    if(n <= 0) goto L1;
    MySum(n);
L1: printf("s = %d \n", s); //L1
    return 0;
}
```

```
void MySum(int n)
{
    int sum = 0;
    for(int i=1; i <= n; ++i)
        sum += i;
    s = sum;
} //写在 first.c 文件后部
```

函数原型作用域

```
// first.c
#include <stdio.h>
int s = 0;
void MySum(int n);        //n
int main()
{
    int n;
    printf("Please input an integer: \n");
    scanf("%d", &n);
    if(n <= 0) goto L1;
    MySum(n);
L1: printf("s = %d \n", s);
    return 0;
}
```

```
void MySum(int n)
{
    int sum = 0;
    for(int i=1; i <= n; ++i)
        sum += i;
    s = sum;
} //写在 first.c 文件后部
```

- **内层作用域的标识符会“挤走”外层作用域的同名标识符。**

```
int myFactorial(int n)
{
    int f = 1;
    for(int i=2; i <= n; ++i)
    {
        int f = f*i;
    }
    return f;
}
```

//for语句里重新定义了一个 f

//这里的 f 仍为1

- **在内层作用域中若要使用与其同名的外层作用域中的标识符，则需要用全局域选择符（::）进行限制。**

```
int f = 1;
int myFactorial(int n, int f)
{
      for(int i=2; i <= n; ++i)
      {
            ::f *= i;
      }
      return f * ::f;
}
```

标识符的属性

- **作用域（scope）**
  - 链接（linkage）

标识符的有效范围

不同子程序中的同名标识符之间的关系

- **存储期（storage duration）**

链接

```
//first.c-by 甲
#include <stdio.h>
int m = 2;   //具有外部链接或内部链接属性，被 second.c 中的使用外部链接
extern int MyFact (int); //要求定义的 MyFact 应具有外部链接属性
static int MyMax(int, int, int); //要求定义的 MyMax 应具有内部链接属性
int main()
{       int n1, n2, n3;
        scanf("%d%d%d", &n1, &n2, &n3);       //n1, n2, n3, max, f无链接
        int max = MyMax(n1, n2, n3);  //链接下面定义的 MyMax
        int f = MyFact (max);    //链接 second.c 中定义的 MyFact
        printf("The factorial of max. is: %d \n", f);
        return 0;
}
static int MyMax(int n1, int n2, int n3)//仅具有内部链接属性，被上面的调用内部链接
{       int max;
        if(n1 >= n2)          // n1, n2无链接
                max = n1;     // max无链接
        else
                max = n2;
        if(max < n3)          // n3无链接
                max = n3;
        return max;
```

```
//second.c-by 乙
int MyFact(int n) //具有外部链接或内部链接属性，被 first.c 中的调用外部链接
{
      extern int m;                    //要求定义的 m 应具有外部链接属性
      int f = 1;
      for(int i=2; i <= n; ++i)        //i, f, n无链接
            f *= i;
      return m*f;                      //链接 first.c 中定义的 m
}
int MyMax(int n1, int n2, int n3)//具有外部链接或内部链接属性，没有被链接过
{
      int max;
      if(n1 >= n2)
            max = n1;
      else
            max = n2;
      if(max < n3)
            max = n3;
      return max;
}
```

链接

```
//first.c-by 甲
#include <stdio.h>
int m = 2;  //具有外部链接或内部链接属性，被 second.c 中的使用外部链接
extern int MyFact (int); //要求定义的 MyFact 应具有外部链接属性
static int MyMax(int, int, int); //要求定义的 MyMax 应具有内部链接属性
int main()
{       int n1, n2, n3;
        scanf("%d%d%d", &n1, &n2, &n3);      //n1, n2, n3, max, f无链接
        int max = MyMax(n1, n2, n3);  //链接下面定义的 MyMax
        int f = MyFact (max);   //链接 second.c 中定义的 MyFact
        printf("The factorial of max. is: %d \n", f);
        return 0;
}
static int MyMax(int n1, int n2, int n3)//仅具有内部链接属性，被上面的调用内部链接
{       int max;
        if(n1 >= n2)          // n1, n2无链接
                max = n1;     // max无链接
        else
                max = n2;
        if(max < n3)          // n3无链接
                max = n3;
        return max;
```

标识符的属性

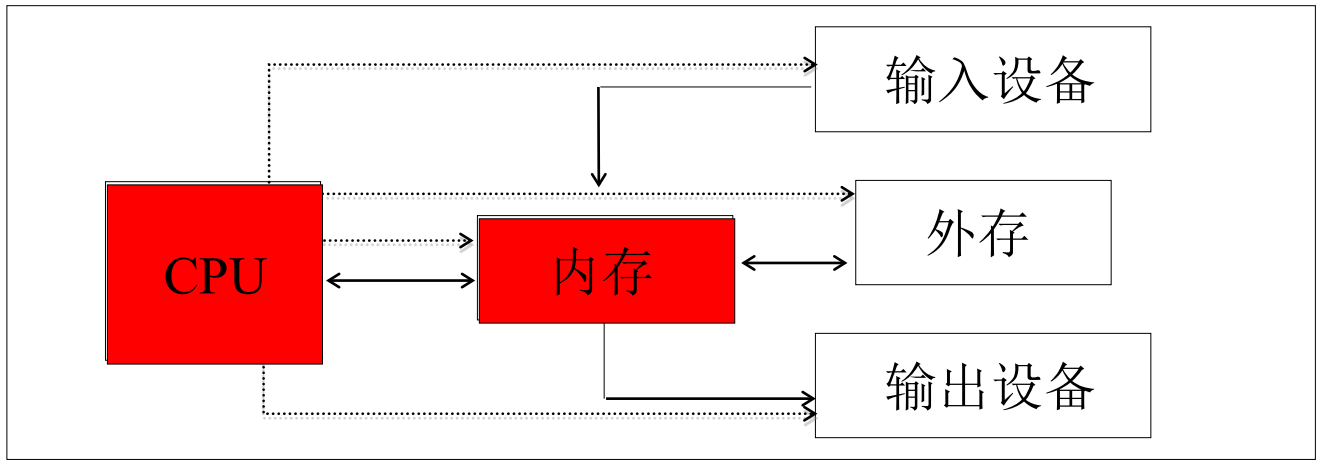
- **作用域（scope）**
  - 链接（linkage）

标识符的有效范围

不同子程序中的同名标识符之间的关系

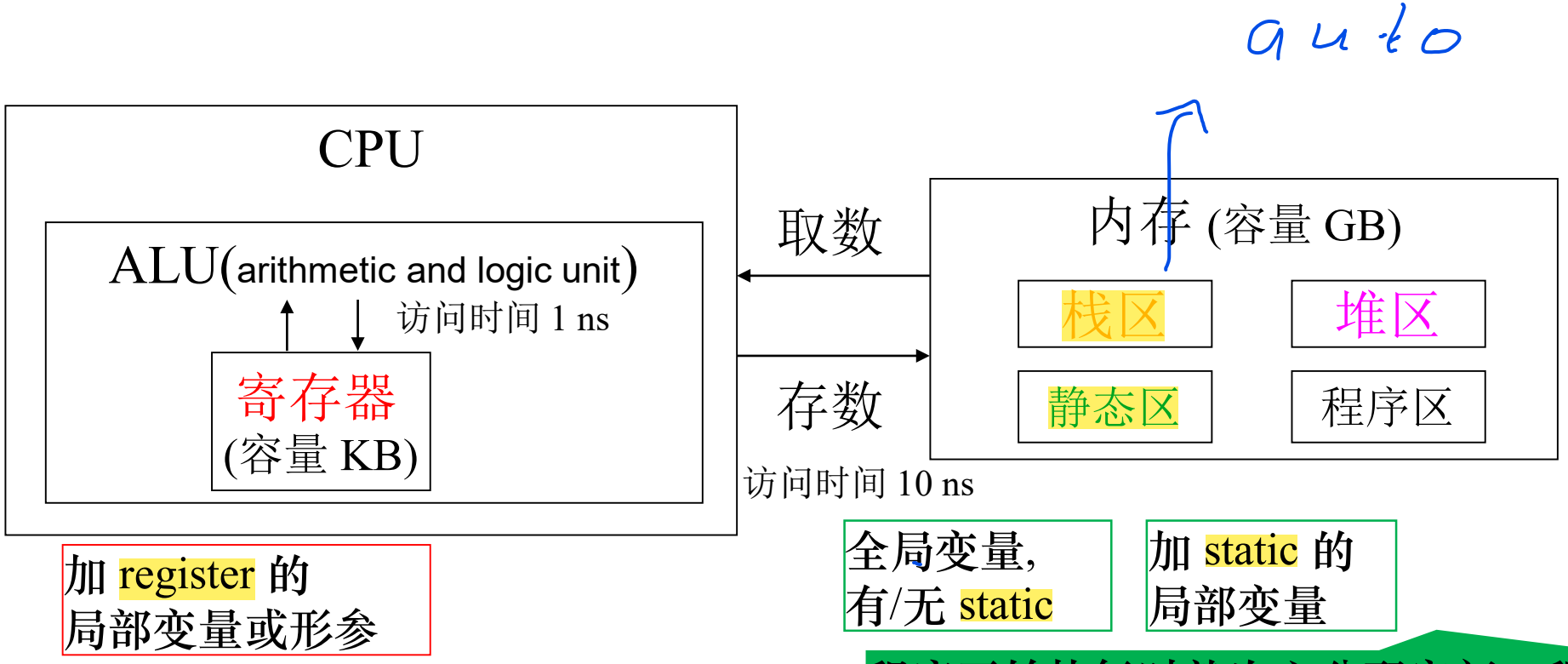
- **存储期（storage duration）**

数据的生存寿命（lifetime）



存储期

- 在程序执行期间，不同存储位置中的数据存储期不同



例3.6 输出任意三个不同的整数。

```
#include <stdio.h>
int MyRand(void);
int main()
{
        int m;
        m = MyRand();
        printf("The m is: %d \n", m);
        m = MyRand();
        printf("The m is: %d \n", m);
        m = MyRand();
        printf("The m is: %d \n", m);
        return 0;
}
int MyRand(void)
{
        static int s = 1; //静态变量
        s = (7 * s + 19) % 3;
        return s;
}
```

看不到.

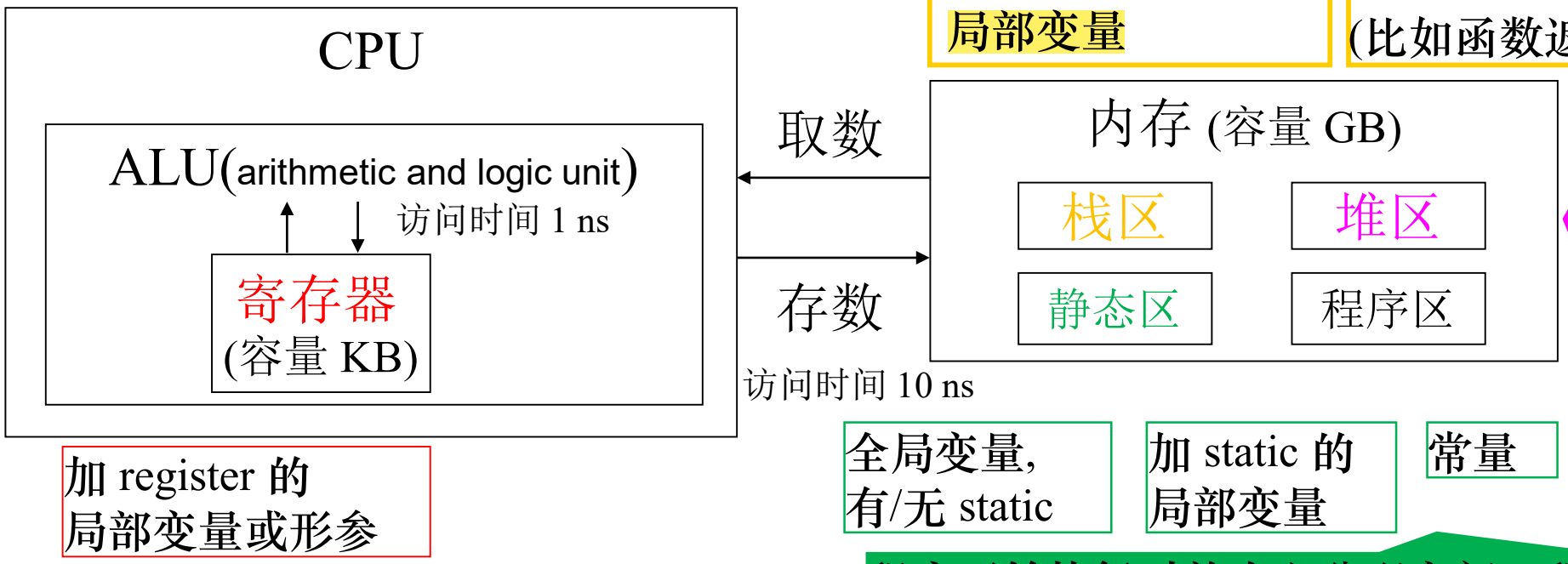
存储期

- 在程序执行期间，不同存储位置中的数据存储期不同

执行到定义它们的语句块时才为之分配空间，所属语句块执行结束，即被收回。

函数的形参
(定义时不可加auto,
不可初始化);
函数调用相关信息
(比如函数返回地址)

无修饰或加auto的局部变量



没有定义，执行到分配/收回语句才分配/收回空间

加 register 的
局部变量或形参

全局变量,
有/无 static

加 static 的
局部变量

常量

程序开始执行时就为之分配空间，程序结束才收回。默认值为0(最好显式初始化)。

跟模块设计有关的优化

宏定义
内联函数
条件编译

起步：

认知与体验（硬件、软件、程序与C语言）

进阶：

判断与推理（流程控制方法、语句）

抽象与联系（模块设计方法、函数）

表达与转换（基本操作、数据类型）

提高：

构造与访问（数组、结构体、指针）

归纳与推广（程序设计的本质）

宏（Macro）定义

```
#include <stdio.h>

#define PI 3.142

void BallSize(int r)
{
  printf("Diameter: %f \n", 2*PI);
  printf("Perimeter: %f \n", 2*PI*r);
  printf("Area: %f \n", PI*r*r);
  printf("Volume: %f \n", 4*PI*r*r*r/3);
}
```

符号常量（manifest constant）

```
// BallSize.h
#include <stdio.h>
#define PI 3.142
```

```
#include "BallSize.h"
int main()
{
      ……
      BallSize(m);
      return 0;
}
```

```c
#include <stdio.h>

#define QSum(x, y) x*x+y*y
int main()
{
    int m, n;
    scanf("%d%d", &m, &n);
    printf("%d", QSum(m, n)); //printf("%d", m*m+n*n);
}
```

带参数的宏定义

```
#include <stdio.h>

#define QSum(x, y)  ((x)*(x)+(y)*(y))
int main()
{
      int m, n;
      scanf("%d%d", &m, &n);
      printf("%f", 1.0/QSum(m+n, m-n));
}
```

带参数的宏定义

```
printf("%f", 1.0/m+n*m+n+m-n*m-n); //?
```

```
printf("%f", 1.0/((m+n)*(m+n)+(m-n)*(m-n)));
```

内联（inline）函数

```
inline double QSum(double x, double y)
{
  return x*x + y*y;
}
```

```
int main()
{
      int m, n;
      scanf("%d%d", &m, &n);
      printf("%f", 1.0/((m+n)*(m+n)+(m-n)*(m-n)));
}
```

条件编译

```
#define ABC

#ifdef ABC
      <代码片段1>
#else
      <代码片段2>
#endif

      <其余代码>
#undef ABC
```

```
#if defined(ABC)
```

```
#define ABC

#ifndef ABC
      <代码片段1>
#else
      <代码片段2>
#endif

      <其余代码>
#undef ABC
```

```
#if !defined(ABC)
```

用来避免重复包含头文件

main.cpp

```
#include <stdio.h>

#include "module1.h"

#include "module2.h"

int main()
{
      //……
}
```

module2.h

```
#include "module1.h"
double myFun(double, double);
```

module1.h

```
#define N 100
double mySin(double);
```

用来避免重复包含头文件

main.cpp

```
#include <stdio.h>
#define N 100
double mySin(double);

#include "module2.h"

int main()
{
      //……
}
```

module2.h

```
#include "module1.h"
double myFun(double, double);
```

module1.h

```
#define N 100
double mySin(double);
```

用来避免重复包含头文件

main.cpp

```
#include <stdio.h>
#define N 100
double mySin(double);
#define N 100
double mySin(double);
double myFun(double, double);
int main()
{
      //……
}
```

module2.h

```
#include "module1.h"
double myFun(double, double);
```

module1.h

```
#define N 100
double mySin(double);
```

用来避免重复包含头文件

main.cpp

```
#include <stdio.h>

#include "module1.h"

#include "module2.h"

int main()
{
      //……
}
```

module2.h

```
#include "module1.h"
double myFun(double, double);
```

module1.h

```
#ifndef MODULE1
#define MODULE1
#define N 100
double mySin(double);
#endif
```

这样，在一个源文件中如果多次包含上面的 **module1.h** 文件，系统只会对第一次包含的内容进行处理。

用来避免重复包含头文件

main.cpp

```
#include <stdio.h>
#define MODULE1
#define N 100
double mySin(double);
#include "module2.h"

int main()
{
      //……
}
```

module2.h

```
#include "module1.h"
double myFun(double, double);
```

module1.h

```
#ifndef MODULE1
#define MODULE1
#define N 100
double mySin(double);
#endif
```

这样，在一个源文件中如果多次包含上面的 **module1.h** 文件，系统只会对第一次包含的内容进行处理。

用来避免重复包含头文件

main.cpp

```
#include <stdio.h>
#define MODULE1
#define N 100
double mySin(double);

double myFun(double, double);
int main()
{
      //……
}
```

module2.h

```
#include "module1.h"
double myFun(double, double);
```

module1.h

```
#ifndef MODULE1
#define MODULE1
#define N 100
double mySin(double);
#endif
```

这样，在一个源文件中如果多次包含上面的 **module1.h** 文件，系统只会对第一次包含的内容进行处理。

用于多环境的程序编写

```
#define OS 'W'

#if OS == 'W'
   ……          //适合于 Windows 环境的代码
#elif OS == 'U'
   ……          //适合于 UNIX 环境的代码
#elif OS == 'M'
   ……          //适合于 macOS 环境的代码
#else
   ……          //适合于其他环境的代码
#endif
   ...          //与环境无关的公共代码
```

```
#ifdef OS
#if defined (OS)
#ifndef <OS>
#if !defined (OS)
```

预定义标识符

用于程序的调试

```
#include <stdio.h>
void BallSize(int r);

#define DEBUG

int main()
{
    double r;
    scanf("%lf", &r);
    #ifdef DEBUG
  // printf("%.2f \n", r);  //调试输出
    #endif
    BallSize(r);
    return 0;
}
```

调试结束，注释掉此行

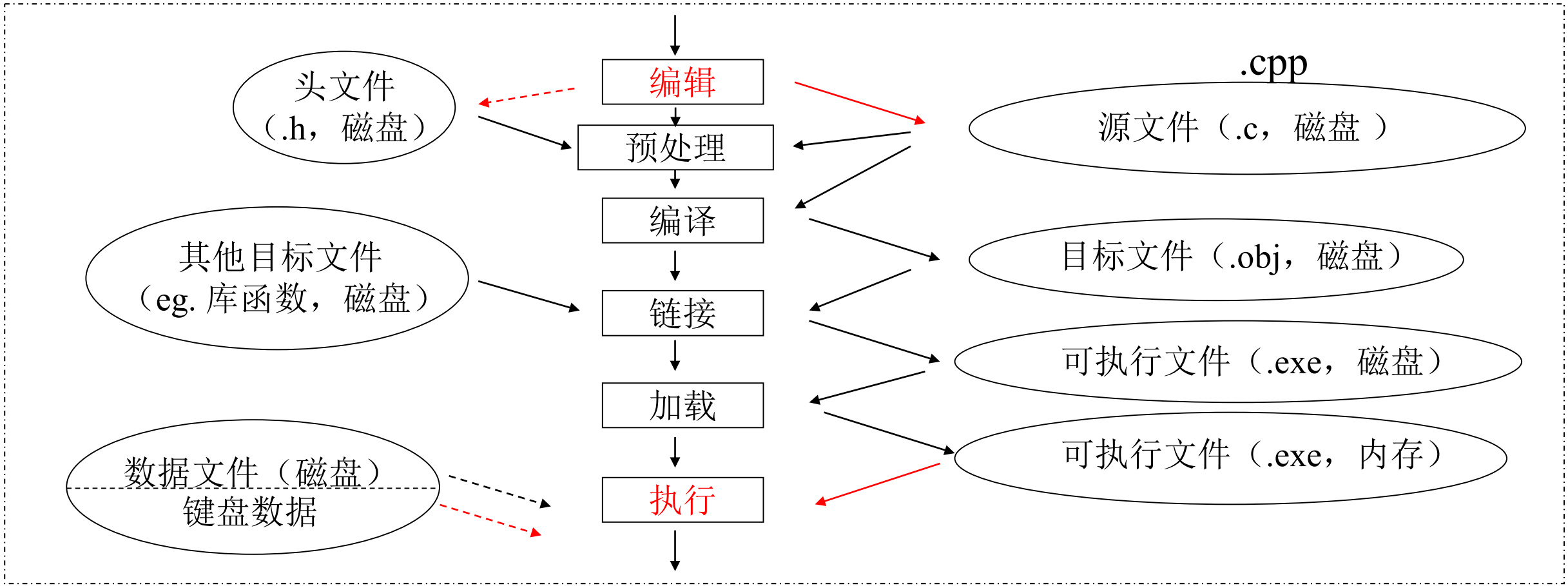
```
#define PI 3.142
void BallSize(int r)
{
    printf("Diameter: %f \n", 2*PI);
    printf("Perimeter: %f \n", 2*PI*r);
    printf("Area: %f \n", PI*r*r);
    printf("Volume: %f \n", 4*PI*r*r*r/3);
}
```

- 演示

C程序的开发步骤与集成开发环境（IDE: Integrated Development Environment）

- **编辑** → **编译** → **链接** → **执行** → **输出结果**

edit → pre-process → compile → link (Build) → load → execute/run → output



Thanks!