作业

- Ex1. 假设南京市普通出租车的收费标准是：3公里以内（含3公里）收费9元，超过3公里的部分，每公里收费2.4元；另外每车次加收1元燃油附加费。编程，计算输入的公里数x所对应的车费y，并输出，结果保留一位小数。例如，x为3.45公里，y为9 + (3.45 - 3) * 2.4 + 1 即11.08元，输出11.1元。

```
#include <iostream>
using namespace std;
#include <iomanip>
int main()
{
	double s, p;
	cin >> s;
	if(s > 3)
		p = 10 + (s-3) * 2.4;
	else if(s > 0)
		p = 10;
	cout << fixed << setprecision(1) << p <<endl;
	return 0;
}
```

- Ex2. 设计程序，验证数论中著名的“四方定理”（一个自然数至多只要用四个数的平方和就可以表示）。要求每输入一个自然数，输出其对应的表达式之一，输入0结束程序。
- 例如，3 = 1*1 + 1*1 + 1*1 + 0*0，15 = 3*3 + 2*2 + 1*1 + 1*1，1713 = 24*24 + 22*22 + 22*22 + 13*13。

```
#include<iostream>
#include<cmath>
using namespace std;

int main( )
{       int number;
        cin >> number;
        while(number != 0)
        {
                for(int i = 0; i <= (int)sqrt(number)+1; ++i)
                        for(int j = 0; j <= i; ++j)
                                for(int k = 0; k <= j; ++k)
                                        for(int l = 0; l <= k; ++l)
                                                if(number == i * i + j * j + …)
                                                {       cout << …
                                                        goto T;
                                                }
        T:      cin >> number;
        }
        return 0;
}
```

```
int number = 0;
while(true)
{       cin >> number;
        if(number == 0)
                break;
        bool flag = false;
        for (int i = 0; !flag && i <= int(sqrt(number))+1; ++i)
                for (int j = 0; !flag && j <= i; ++j)
                        for (int k = 0; !flag && k <= j; ++k)
                                for (int l = 0; !flag && l <= k; ++l)
                                        if ((i * i + j * j + k * k + l * l) == number)
                                        {       cout << i << ' ' << j << ' ' << … << endl;
                                                flag = true;
                                        }
}
```

自主训练任务

- 1. 编程，输入整数n，比较1-n的绝对值与n-1的大小，并输出比较结果（例如，|1--3| > -3-1、|1-2| == 2-1等)

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
      int n = 0;
      printf("Please input n: \n");
      scanf("%d", &n);
      if(abs(1-n) > n-1)
            printf("|1 - %d| > %d - 1 \n", n, n);
      else
            printf("|1 - %d| == %d - 1 \n", n, n);
      return 0;
}
```

- 3. 用switch语句实现：从键盘输入一个星期的某一天(用0表示星期天；用1表示星期一；…)，输出其对应的英语单词。

```
int main( )
{     int day;
      scanf("%d" , &day);
      switch (day)
      {     case 0: printf("Sunday \n"); break;
            case 1: printf("Monday \n"); break;
            …
            case 6: printf( "Saturday \n"); break;
            default: printf( "Input error \n");
      }
      return 0;
}
```

- 5. 设计程序，计算从键盘输入的一系列正整数的和，输入-1结束。

```
int main()
{     int n, sum = 0;
      cin >> n;          //输入第一个数

      while(n != -1) //事件控制型循环
      {     sum += n;
            cin >> n; //输入下一个数
      }

      cout << "输入的整数的和是: " << sum << endl;
      return 0;
}
```

- 7. 编程，应用循环流程控制语句计算一个正整数等差数列的和，并输出。数列的项数、首项、公差（>0）由用户从键盘输入。

```
int n=100, min=1, step=1, sum=0;
cin >> n;
cin >> min;
cin >> step;
for(int i = 1; i <= n; ++i)
{
      sum += min + (i-1)*step;
}//迭代
cout << sum <<endl;
```

```
for(int i = min; i <= (n-1)*step + min; i += step)
{
      sum += i;
}//迭代
```

- 9. 设计程序，计算正整数 d1~d2 之间（含d1、d2）不能被整数 d3 整除的整数（用户每输入一组d1、d2、d3的值，就输出一次结果，只要输入0就结束）。

```
int d1, d2, d3;
cin >> d1;
while(d1 != 0)
{
      cin >> d2 >> d3;
      for(int i = d1; i <= d2; ++i)
      {     if(i%d3 != 0)   //    不能被d3整数
                  cout << i << " \t";
      }//穷举
      cin >> d1;
}
```

- 10. 设计程序，用牛顿迭代公式求一个数 x 的立方根，保留两位小数。计算公式为： $x_{n+1} = \frac{1}{3}(2x_n + \frac{x}{x_n^2})$ ，当$|x_{n+1} - x_n| < \varepsilon$(ε为一个很小的数，例如0.000001)时，$x_{n+1}$即为x的立方根。（注意：1/3的结果为0，写成1.0/3可避免整除问题；使用fabs库函数求绝对值时需include <math.h>。）

- **分析：**
  - 该问题是一个重复计算x_{n+1}的过程，该过程直到x_{n+1}-x_n的绝对值小于某个很小的值（假设为10^{-6}）时结束。每次循环操作时，x_n为上一次循环中x_{n+1}的计算结果。第一次循环时，可以把x_n取为x（实际上取任何值都可以，它不影响最终结果，只影响循环次数）。程序可用下面的事件控制的循环来实现：

```
double x, x1, x2, temp;                    //x1和x2分别用于存储xn和xn+1
scanf("%lf", &x);
x1 = x;                                    //第一个值取x
do
{
      x2 = (2*x1 + x/(x1*x1)) / 3;         //计算新的值
      temp = fabs(x2 - x1);
      x1 = x2;                             //记住前一个值
}while(temp >= 1e-6);                      //0.000001
printf("%.2f \n", x2);
```

x 为 0?

注意循环里面语句的顺序

```
double x, x1, x2, temp;                    //x1和x2分别用于存储xn和xn+1
scanf("%lf", &x);
x2 = 1;
do
{
      x1 = x2;                             //记住前一个值
      x2 = (2*x1 + x/(x1*x1)) / 3;         //计算新的值
      temp = fabs(x2 - x1);
}while(temp >= 1e-6);                      //0.000001
printf("%.2f \n", x2);
```

注意循环里面语句的顺序

```
double x, x1, x2;                          //x1和x2分别用于存储x_n和x_n+1
scanf("%lf", &x);
x2 = 1;
do
{
      x1 = x2;                                    //记住前一个值
      x2 = (2*x1 + x/(x1*x1)) / 3;          //计算新的值
}while(fabs(x2 - x1) >= 1e-6);              //0.000001
printf("%.2f \n", x2);
```

注意循环里面语句的顺序

南京大学 计算机科学与技术系
Department of Computer Science & Technology, NJU

Chapter 2

程序的模块设计方法

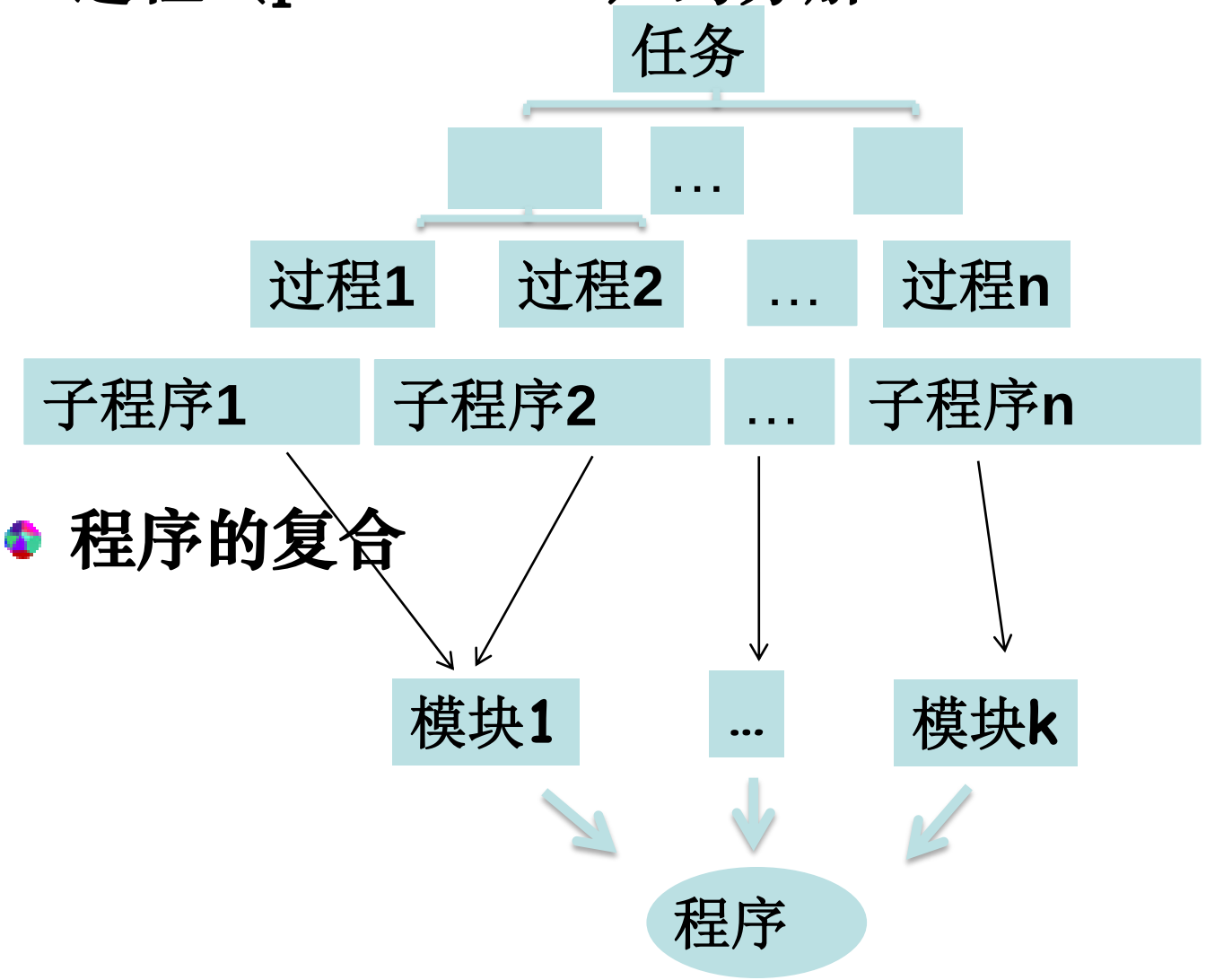
刘奇志

- 简介
- 单模块程序的设计
- 多模块程序的设计
- 程序模块设计的优化

带有子程序的程序如何设计?

模块设计

- **过程（procedure）的分解**



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

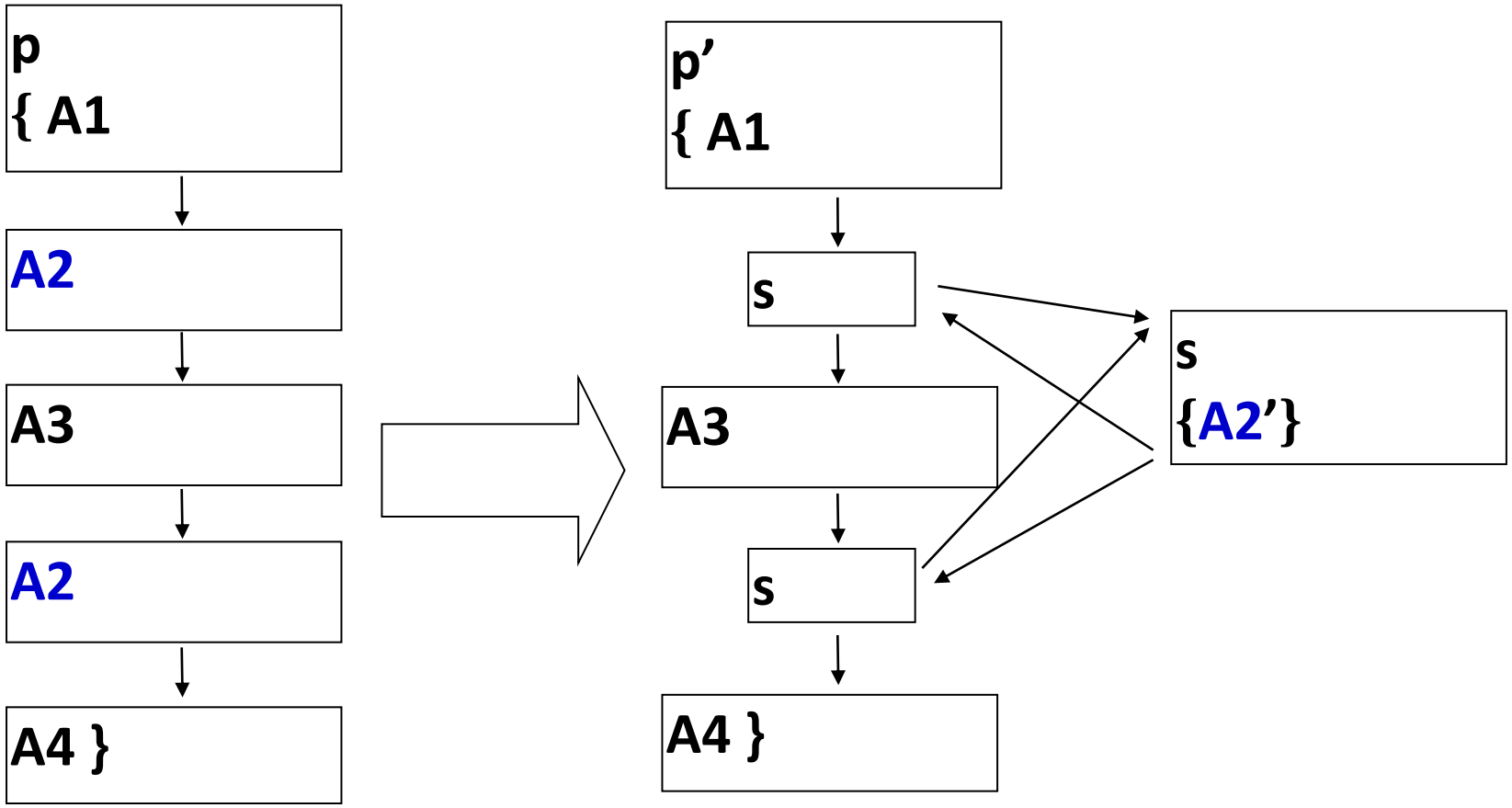
- **程序的复合**

发挥头文件的作用

- 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：

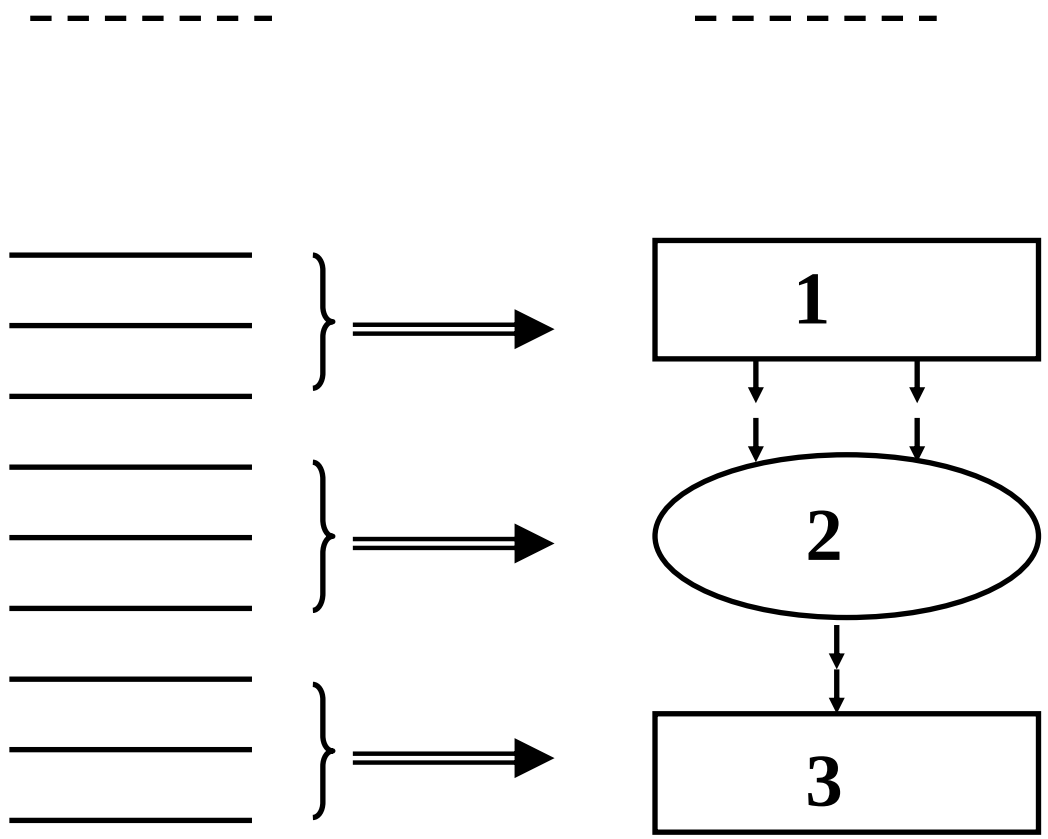
```
int main( )
{
    int a, b, i, j, c, k;
    scanf("%d%d%d%d", &a, &b, &i, &j);
    c = a*a + b*b;
    k = i*i + j*j;
    printf(…;
    return 0;
}
```

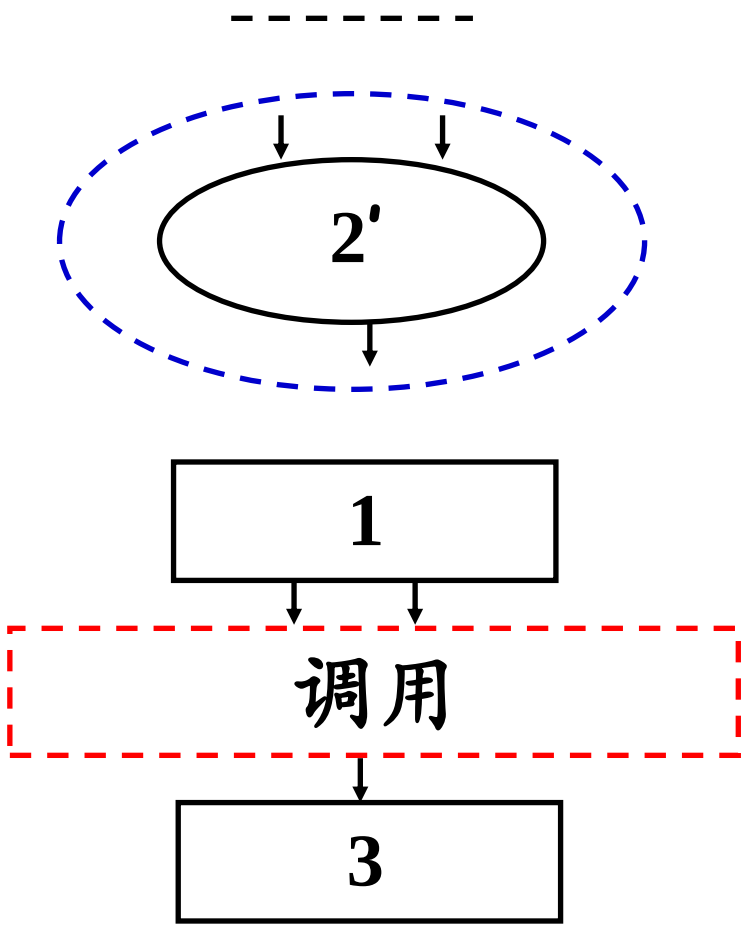
定义

```
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


1
2
3


2'
1
调用
3

子程序的好处

- 子程序的开发可以相互独立
  - 于是，使多位程序员合作开发一个程序成为可能，每位程序员可以负责一个模块（一个模块对应一个源文件）；
  - 而且，开发一个子程序代码时可以直接调用另一个子程序，使用者只需知道所调用的子程序对应的功能，而不必关心它们具体封装的细节，从而提高开发效率；
  - 即使是一位程序员独自开发一个程序，子程序使程序的结构变得清晰、灵活，也便于程序的扩展，并且便于程序的修改；
  - 子程序往往还可以减少程序中的重复代码，避免代码间的不一致性。
- 子程序是程序模块设计的基础

- 简介
- 单模块程序的设计
  - C语言函数（子程序）基础
    - 函数的定义
    - 函数的调用
    - 函数的声明
  - C语言函数的嵌套调用
    - 嵌套调用及其过程
    - 递归
- 多模块程序的设计
- 程序模块设计的优化

所有子程序在一个模块内

C子程序——函数（function）

- 17世纪末，在德国的数学家莱布尼兹的文章中，首先使用了“function”一词，不过，它表示“幂”、“坐标”、“切线长”等概念。
- 19世纪中期，法国数学家黎曼吸收了莱布尼茨及后来达朗贝尔和欧拉的成果，第一次准确地提出了function的定义：某一个量依赖于另一个量，使后一个量变化时，前一个量也随着变化。
- 清.李善兰：“凡此变数中函（包含）彼变数者，则此为彼之函数”。
- 注：函（象形。今隶误作函。本义:舌[tongue]。后指盛物的匣子、套子 [case]。）

```
f(x,y,z)

average = (n1+n2+n3)/3
average = f(n1,n2,n3)
          y(n1,n2,n3)

myFunction(n1,n2,n3)
{average = (n1+n2+n3)/3}
```

- 计算机最初的任务和目的是进行科学计算
  程序相当于我们给计算机的命令，命令它进行科学计算
  把程序分成若干个函数，每个函数命令计算机完成一项计算任务

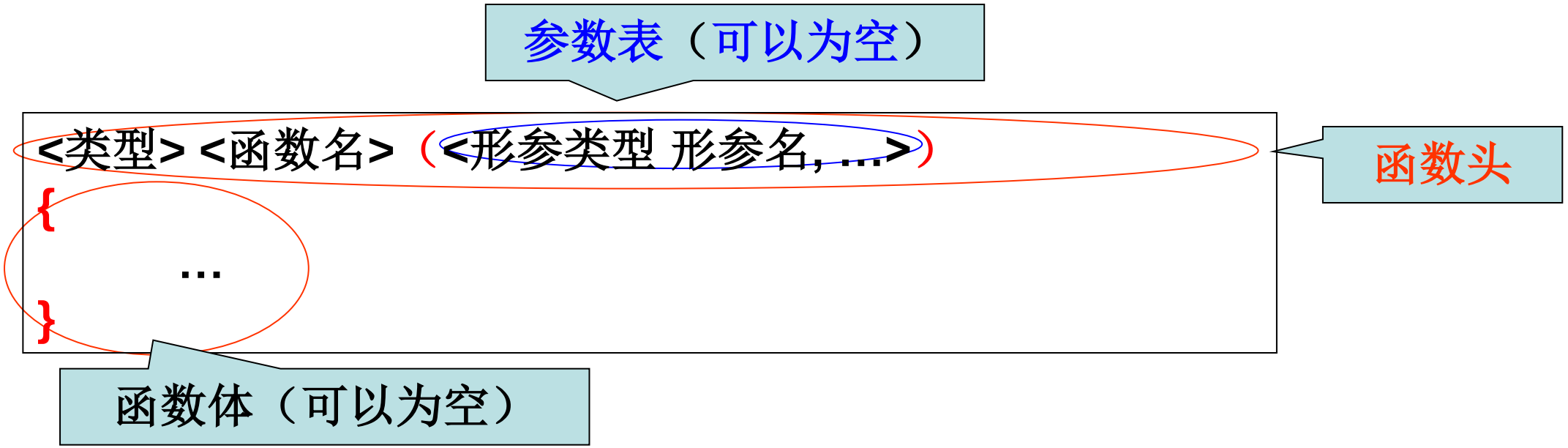
- 在计算机领域，“计算”的涵义被扩展，除科学计算外，还可以是信息处理等计算机可以完成的任何操作

- “函数”的涵义也被扩展，除变量间的依赖关系的描述外，还可以指一组操作的描述
- 其过程除了根据自变量求解应变量值的一系列计算步骤外，还可以指执行一系列计算机可执行的操作步骤
- 执行时，可以有自变量（参数）的参与，也可以没有参数
- 函数的结果除得到应变量的值（返回值）外，还可以没有具体的返回值，仅完成某个功能

函数的定义(definition)

- 定义一个函数，即编写函数的代码，以便实现具体的功能，同时给这段代码取一个函数名，并给出函数的类型，以及各个参数的名称及其类型（即参数的定义）。
  - 一个程序中，同一个函数只能定义一次。参数个数或类型不同的同名函数视为不同的函数。
  - C语言标准规定，任何一个C程序中必须定义一个main函数。本教程前面章节的例子程序都定义了一个main函数。

函数的定义(definition)



```
void myFun()
{

}
```

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

```
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;
}
```

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

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

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

- **<函数名>是标识符的一种，遵循标识符有关规定。**
- **函数的类型写在函数名的前面。**
- **函数定义中的参数叫形式参数，即不知道实际数据的形式上的参数，简称形参。**
  - 一个形参相当于定义的一个变量，其名称与类型写在函数名后面的圆括号中，如果形参个数≥2，则用逗号将它们分开，如果没有形参，就写一个void。
- **这些内容构成函数头。**

- **花括号之间的语句是功能实现代码，可看成一个复合语句，成为函数体。**
- **书写时，最好将花括号与函数头左对齐，并缩进其中的代码。**

return语句

- C语言中的return语句用来结束函数的执行，还可以顺便返回一个返回值。
  - 一个return语句最多只能返回一个值。
  - 如果没有return语句，则函数体的右花括号作为函数执行结束的标志。

  - main函数中的return语句用来结束整个程序的执行。

- 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 myMax(int 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;
}
```

```
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 myFactorial(int max) //错误
      {     int f = 1;
            for(int i = 2; i <= max; i++)
                  f *= i;
            return f;
      }  //应把函数myFactorial的定义写在myMax函数的外面
}
```

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

- 定义应该各自独立。

```
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)//错误
      {     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", max);
      return 0;}
```

```
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);
      printf("The max. is:  %d\n", myMax(n1, n2, n3));
      return 0;}
```

- 定义应该各自独立。

- goto语句不能从一个函数体转到该函数的外部，也不能从一个函数的外部转入该函数体。

函数的调用

- 只有当程序中有相应的调用操作时，一个函数（除main函数外）的函数体才有机会被执行。
- main函数是程序执行的入口，由操作系统调用，不能由其他函数调用。
- 已经定义的函数可以作为被调函数被一个或多个函数调用。
- C语言标准规定了一个函数库，包括一些常用函数的定义，这些标准库函数可以直接被调用。C++语言标准兼容了这个库。

- 当函数体中含有带调用操作时，该函数成为主调函数。

- 任何一个函数（包括main函数）都可以作为主调函数，调用一个或多个被调函数。

参数表（可以为空）

<函数名>（<实际参数, …>）

- 调用操作必须包括**被调函数名**和**圆括号**。
- 圆括号里可以含有参数，这里的参数称为实际参数，即有实际数据的参数，简称实参。
  - 可以是常量，也可以是复杂一点的表达式。
  - 如果实参个数≥2，则用逗号将它们分开，如果没有实参，就什么也不写。
- 实参与形参一一对应，个数相等，类型相符。

函数的调用

```
void myFun()
{

}
```

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

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

```
int main( )
{
      myDisplay( 7 );
      return 0;
}
```

```
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;
}
```

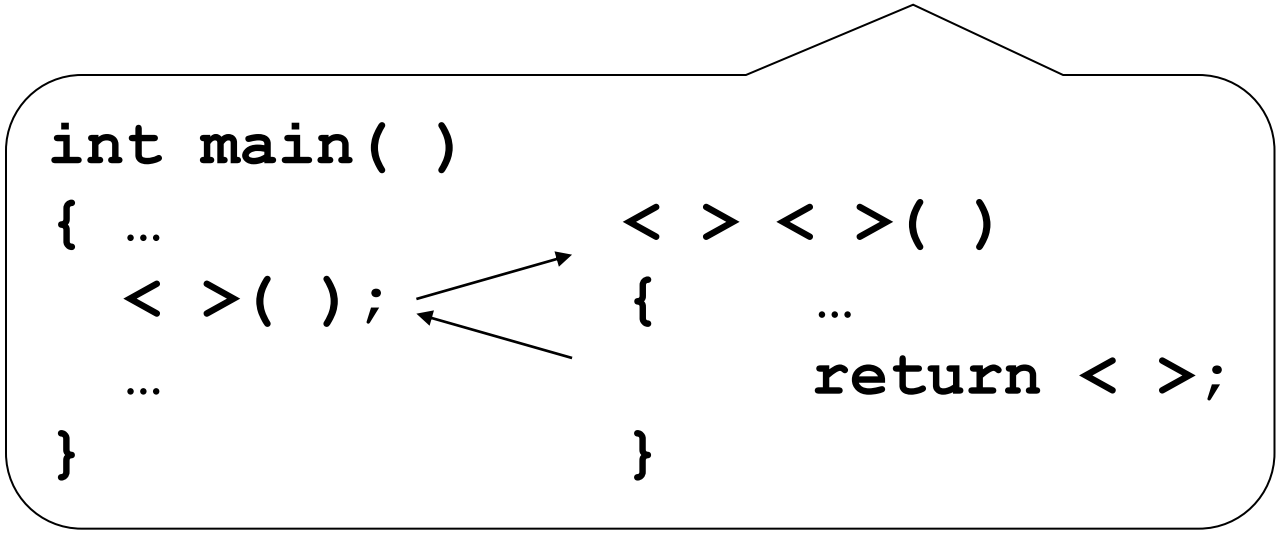
```
int main( )
{
    int i = 3, j = 4, k = 5;
    int m = myMax(i, j, k);
    printf("%d ", m);
    return 0;
}
```

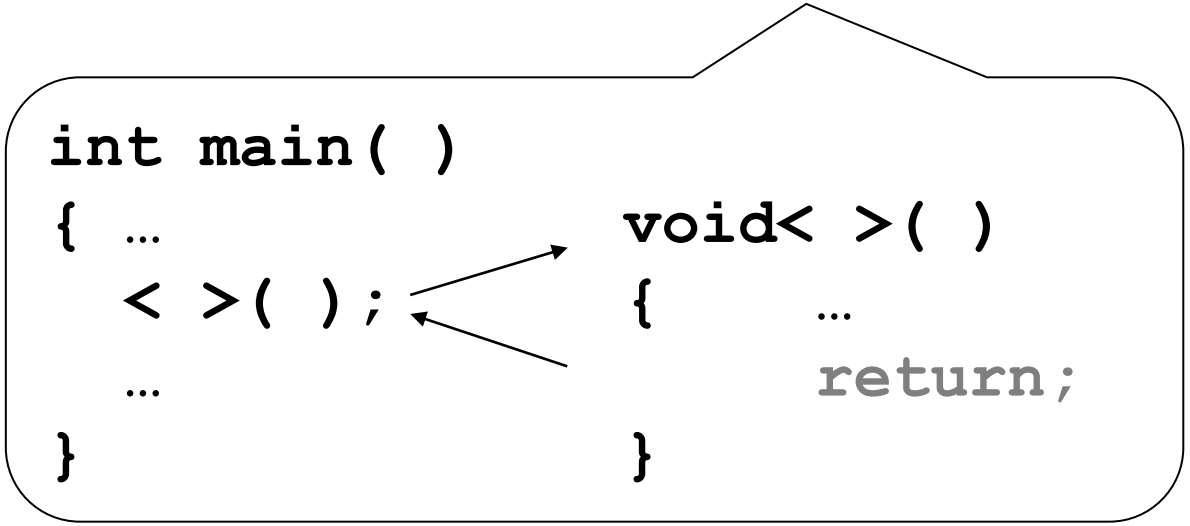
```
int main( )
{
    int i = 3, j = 4, k = 5;
    printf("%d", myMax(i, j, k));
    return 0;
}
```

- 没有返回值的函数，其调用操作只能单独成句。例如，
  - `myFun();`
  - `myDisplay(7);`

- 有返回值的函数，其调用操作可以作为另一个基本操作的操作数或另一个函数调用操作的实际参数。例如，
  - `int max = myMax(i, j, k);`
  - `printf("%d", myMax(i, j, k));`

- 当程序执行到调用操作时，系统先将实参（如果有）通过类似赋值的方式传给对应的形参，然后记下当前函数的代码执行到何处（即函数返回地址），再执行被调函数的代码。
- 被调函数执行结束后，返回值（如果有）被返回到主调函数里的被调函数名处，并继续执行主调函数余下的代码。





- 例2.1用函数实现例1.2中的求阶乘问题。
- 分析：求阶乘问题可以分解为一个独立的功能，用函数实现时，函数的参数为一整数，函数的返回值为该整数的阶乘。

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

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

```
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;
}
```

- 例2.2 求输入三个整数中最大值的阶乘，并输出。
- 分析：求最大值和求阶乘相对独立，可以分别用函数实现，求最大值函数的参数有三个。

```
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: %d \n", f);
    return 0;
}
```

函数调用的不同“身份”

```
f = myFactorial(myMax(n1, n2, n3)); //作为实参
printf("Factorial of max. is: %d \n", f);
```

或

```
max = myMax(n1, n2, n3); //作为操作数
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。

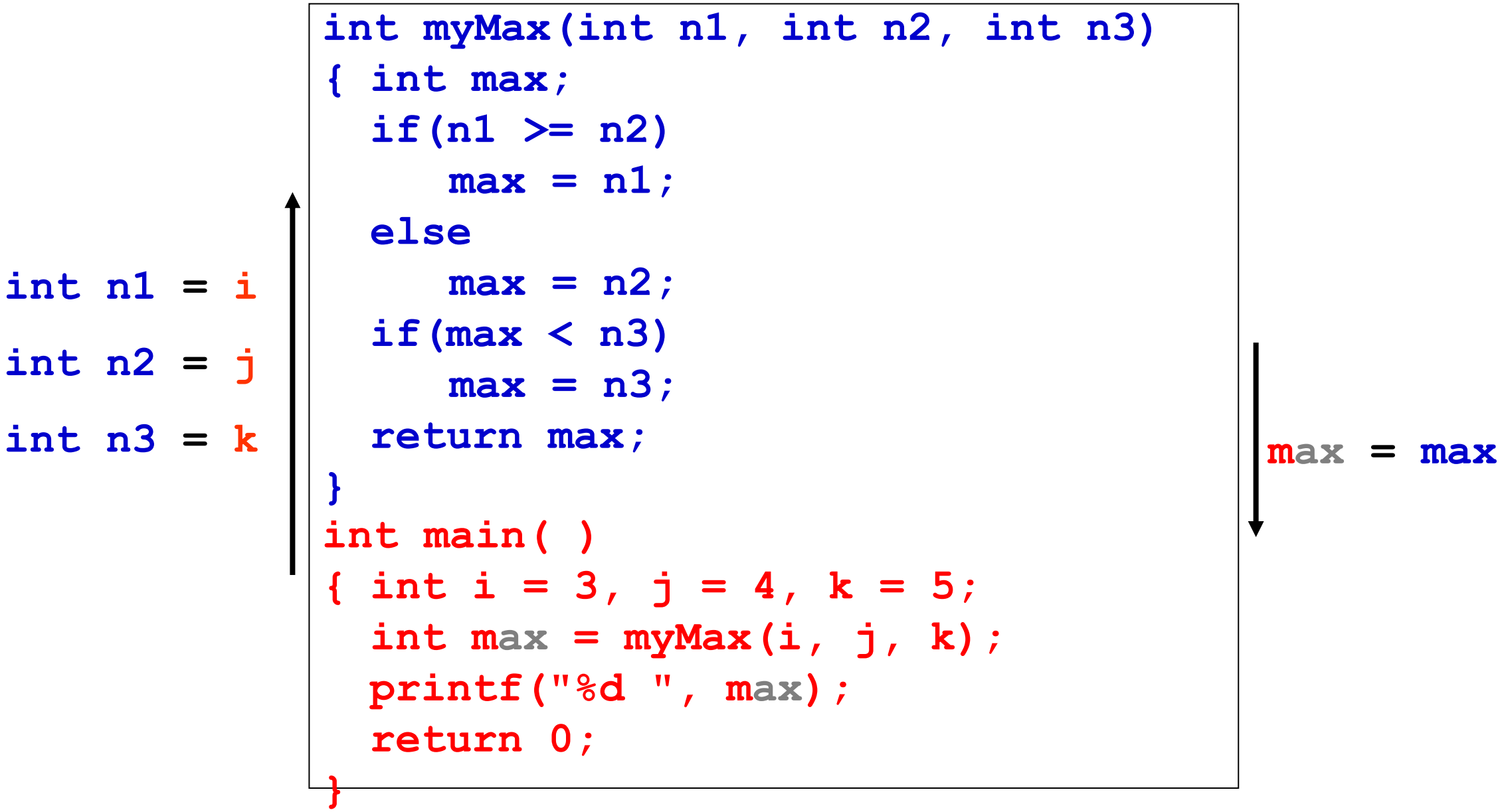
函数的参数

- 形式参数与实际参数
  - 形参是定义时，函数名后面括号中的变量名；
  - 实参是调用时，函数名后括号中的表达式。
- 形参与实参的内存分配
  - 未出现函数调用时，形参不占内存（尽管已经定义），只有发生函数调用时，形参才被分配内存单元，调用结束后，形参所占的内存被释放；
  - 实参有确定的值，与形参占不同的内存单元。实参与形参可以是同名变量，但仍然占据不同的内存单元。
- 形参与实参的对应关系
  - 调用时，实参的值单向传给形参；
  - 实参与形参一一对应，个数相等，类型相符。

函数的返回值

- 函数返回值的类型以所定义的函数类型为准；
- 函数的返回值是通过函数中的return语句获得的；
- 没有return语句时（被调函数不需要返回什么有用的值），函数会返回一不确定的值，这时可以用void将函数定义成空类型；
- main函数通过返回值把整个程序的执行情况告诉调用者（一般是操作系统），返回0表示正常结束，否则返回-1。操作系统通过接收到的这个值来判断程序是否正常执行完毕（如果作批处理，后面的操作就可以通过前面的返回值来决定该如何做了）。C语言标准规定main函数应返回一个整数，而大多数操作系统都约定接收到0表示任务正常执行完毕，-1表示出现异常。

函数的参数与返回值

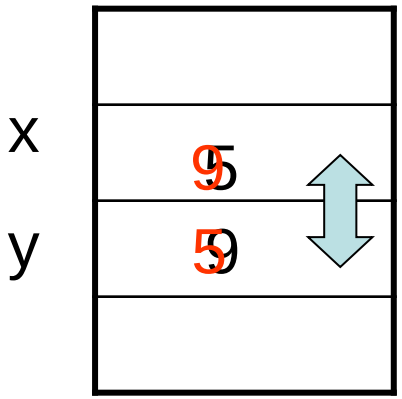

int n1 = i
int n2 = j
int n3 = k
int myMax(int n1, int n2, int n3)
{ int max;
if(n1 >= n2)
max = n1;
else
max = n2;
if(max < n3)
max = n3;
return max;
}
int main( )
{ int i = 3, j = 4, k = 5;
int max = myMax(i, j, k);
printf("%d ", max);
return 0;
}
max = max

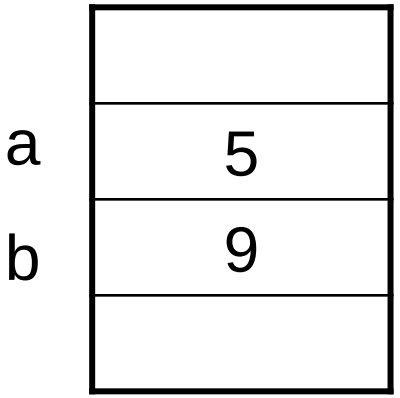
思考：mySwap能否实现a和b的交换?

```
void mySwap(int x, int y)
{
   int temp;
   temp = x;
   x = y;
   y = temp;
}
```

```
int x = a
int y = b
```

```
int main ( )
{
      int a = 5, b = 9;
      mySwap(a, b);
      printf("%d, %d", a, b);
      return 0;
}
```





函数间的通讯方式 I

- 函数间有多种通讯方式
  - 传值方式(把实参的副本复制给形参)
  - 利用函数返回值传递数据

函数的声明（declaration）

- 所谓函数的声明，即以语句形式给出函数的原型（function prototype）。
- 函数原型包括函数名、函数返回值的类型、以及形参的类型和名字（函数头），形参的名字可以没有。

- C程序中调用的所有函数都要有定义。
- 如果被调函数的定义在主调函数的定义之后或在其他文件中，则需要在调用前对被调函数进行声明（declaration）。

```
…
int myFactorial(int n); //函数的声明
int myMax(int n1, int n2, int n3); //函数的声明
int main( )
{ ……//有函数的调用
}
int myFactorial(int n) //函数的定义
{ ……
}
int myMax(int n1, int n2, int n3) //函数的定义
{   ……
}
```

```
…
int myFactorial(int n) //函数的定义
{ ……
}
int myMax(int n1, int n2, int n3) //函数的定义
{   ……
}
int main( )
{ ……//有函数的调用
}
```

- 如果被调函数的定义在本文件主调函数的定义之前，则可以不声明。
- 将main函数的定义写在前面，被调函数的定义写在后面，并在main函数的定义前给出被调函数的声明，这样可以使程序的层次结构显得更为清晰。
- 函数声明可以减轻程序员调试程序的工作量，因为如果调用方式与声明的类型不一致，编译器会尽早指出错误。
- 一个程序中同一个函数的声明可以有多个。

```
void mySwap(int x, int y);

int main ( )
{     int a = 5, b = 9;
      mySwap(a, b);
      printf("%d, %d", a, b);
      return 0;
}

void mySwap(int x, int y)
{     int temp;
      temp = x;
      x = y;
      y = temp;
}
```

声明

调用

定义

定义

局部变量与全局变量

- 在C中，根据变量的定义位置，把变量分成：
  - 局部变量：在复合语句（包括函数体）中定义的变量，它们只能在定义它们的复合语句中使用（定义点之后）。函数的形式参数也可看成是局部变量。
  - 全局变量：在函数外部定义的变量，它们一般能在程序中的所有函数中使用（静态的全局变量除外）。

例

- 数列求和程序。

```
#include <stdio.h>

int s = 0;  //全局变量
void mySum(int);

int main( )
{
    int n;
    printf("Please input an integer: \n");
    scanf("%d", &n);
    mySum(n);
    printf("s = %d \n", s);
    return 0;
} //局部变量n
```

```
void mySum(int n)
{
    int sum = 0;
    for(int i = 1; i <= n; i++)
    {
        sum += i;
    }//局部变量i
    s = sum;
} //局部变量sum、n
```

- 用全局变量改写下面的程序，实现数据交换。

```
void mySwap(int x, int y)
{
      int temp = x;
      x = y;
      y = temp;
}

int main ( )
{     int a = 5, b = 9;
      mySwap(a, b);
      printf("%d, %d", a, b);
      return 0;
}
```

```
int a, b;
void mySwap()
{
      int temp = a;
      a = b;
      b = temp;
}

int main ( )
{
      a = 5, b = 9;
      mySwap();
      printf("%d, %d", a, b);
      return 0;
}
```

函数间的通讯方式Ⅱ

- 函数间有多种通讯方式
  - 传值方式(把实参的副本复制给形参)
  - 利用函数返回值传递数据

  - 通过全局变量传递数据
    (函数副作用问题)

全局变量的声明（declaration）

- 即以语句的形式列出全局变量名及其类型，并在前面加关键字extern。
  - 如果全局变量的定义在使用该全局变量的函数体之后，或在其他文件中，则需要在使用前对全局变量进行声明。
  - 执行全局变量的声明时，系统并不为全局变量分配内存空间，所以声明全局变量时不可以赋值，也可以对一个全局变量进行多次声明。
  - 相同类型的多个全局变量可以并列声明。

例

- 数列求和程序。

```
extern int s;
void mySum(int n)
{
    int sum = 0;
    for(int i = 1; i <= n; i++)
    {
        sum += i;
    }
    s = sum;
}
```

```
#include <stdio.h>

int s = 0;
int main( )
{
    int n;
    printf("Please input an integer: \n");
    scanf("%d", &n);
    mySum(n);
    printf("s = %d \n", s);
    return 0;
}
```

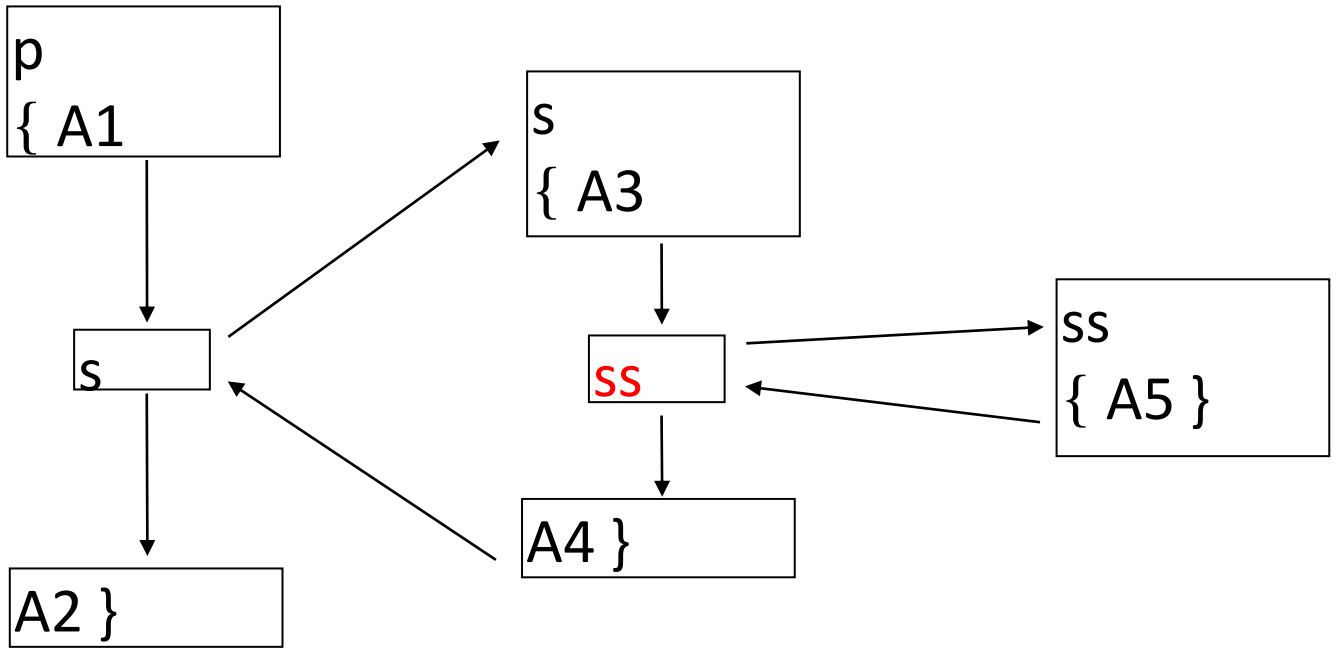
函数的副作用

- 全局变量可以实现函数之间的数据共享
- 通过全局变量来实现两个函数之间的数据传递不是一个好的程序设计风格，使用不当会带来诸多问题，特别是会引起设计者未意识到的函数副作用问题。
- 函数副作用：函数中改变了非局部量的值。

- 简介
- 单模块程序的设计
  - C语言函数（子程序）基础
    - 函数的定义
    - 函数的调用
    - 函数的声明
  - C语言函数的嵌套调用
    - 嵌套调用及其过程
    - 递归
- 多模块程序的设计
- 程序模块设计的优化

嵌套调用

- **子程序有时候也比较复杂，需要进一步分解，于是出现嵌套调用现象，即被调子程序中出现子程序的调用操作。**



被调子程序s中出现子程序ss的调用操作

对于C程序，

- 子程序的嵌套调用即函数的嵌套调用，子程序的递归调用即函数的递归调用。

C函数的一般嵌套调用

- 被调函数体中含有调用另一个函数的操作。比如：

```
…
int main( )
{
    …
    f = myFactorialNew(n1, n2, n3);
    …
}
int myFactorialNew(int n1, int n2, int n3)//被调函数
{
    int max = myMax(n1, n2, n3);
    …
}
int myMax(int n1, int n2, int n3)
{   …
}
```

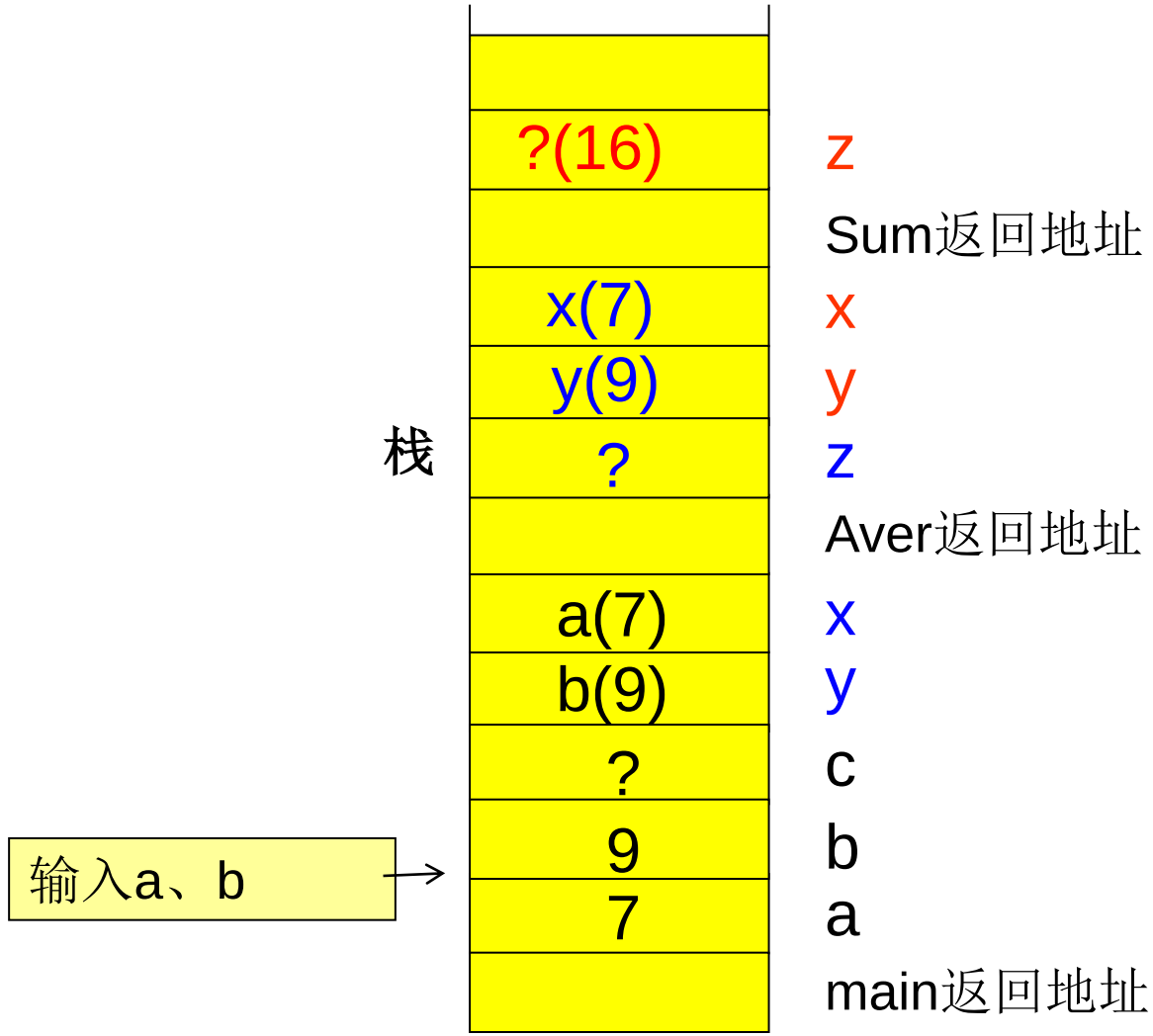
```
f = myFactorial(myMax(n1, n2, n3));
//不是嵌套调用，而是函数调用的另一种
//“身份”：作为实参，不是作为操作数
```

注意：函数myFactorialNew与函数myFactorial的区别（参数个数、被调用次序）。

```
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;
}
```

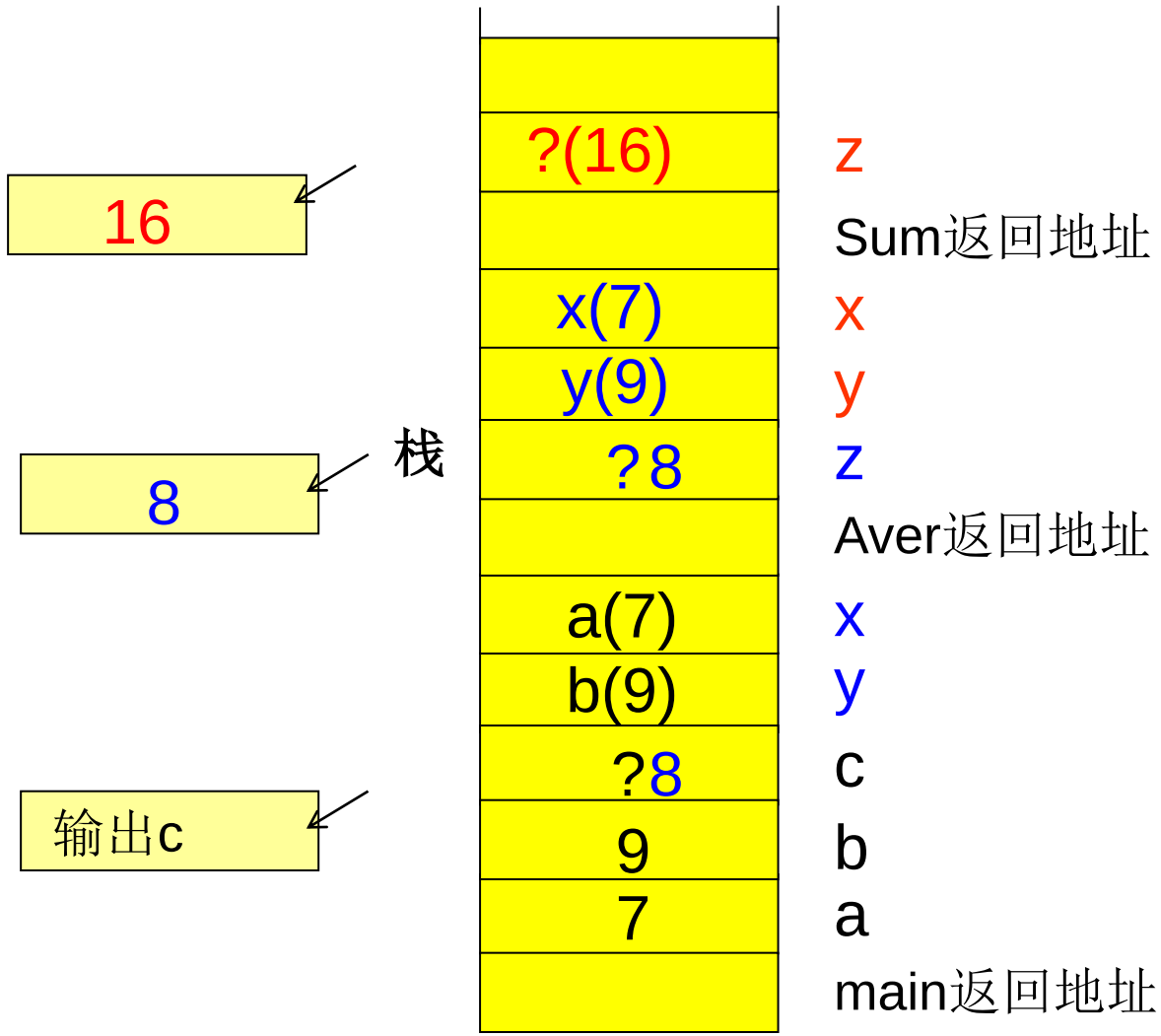
C函数嵌套调用的过程

```
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;
}
```



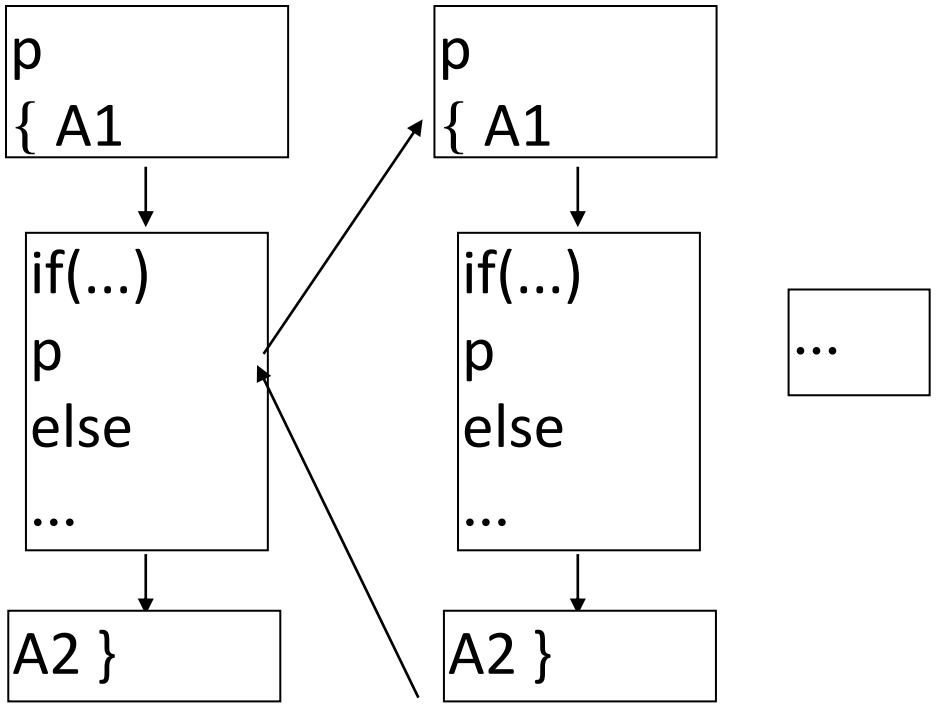
C函数嵌套调用过程（续）

```
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;
}
```

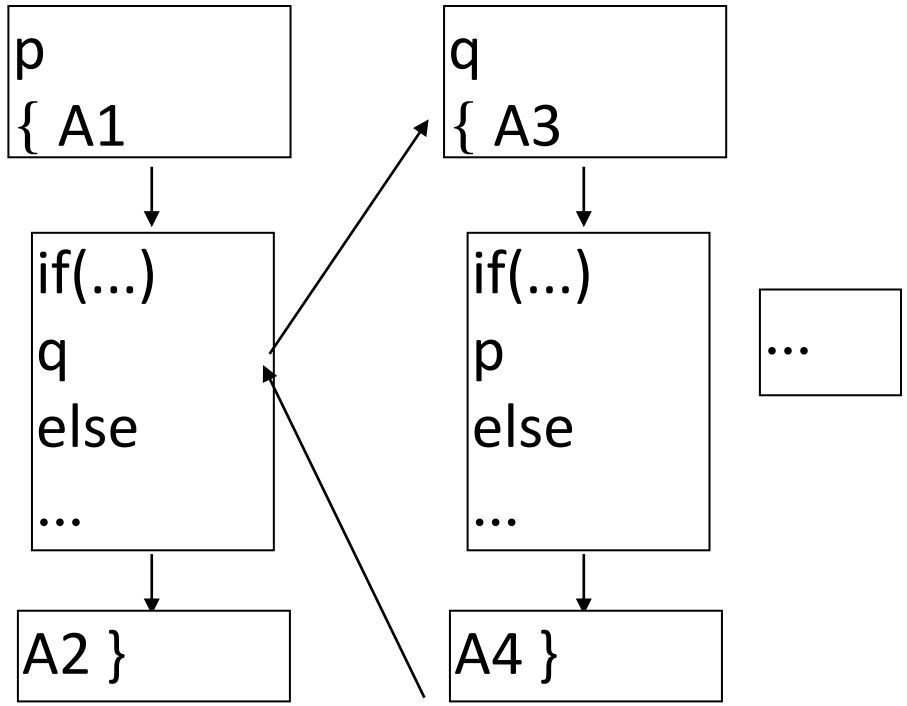


递归调用

- 如果子程序p中直接或间接包含自身的调用操作，则出现递归调用现象，它是嵌套调用的一种特殊形式。



直接递归



间接递归

- 这种情况下，子程序中通常含有分支流程，分别处理递归调用操作和调用结束操作。其中，直接递归形式更为常见，它为某些**带有重复性操作的任务**提供了一种比采用循环流程更为自然、简洁的实现方式。

C函数的递归（recursion）调用

- 本课件称**函数体中含本函数调用操作**的函数为**递归函数**，并只讨论**直接**递归函数。

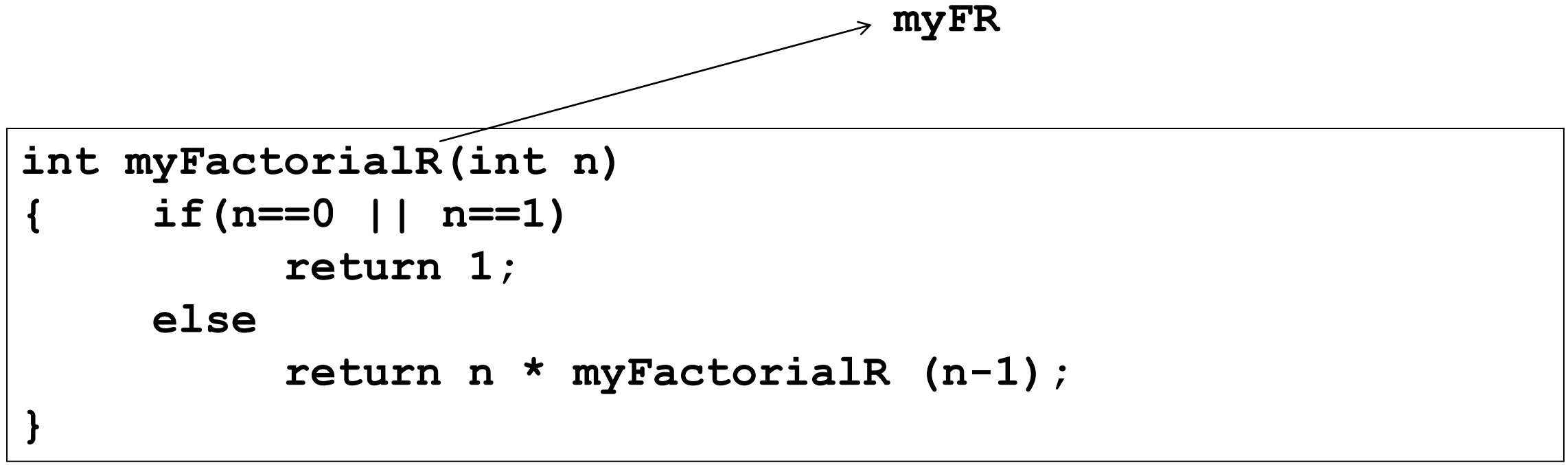
- **在定义递归函数时，要对两种情况给出描述：**
  - **递归条件。指出何时进行递归调用，它描述了问题求解的一般情况。**
  - **结束条件。指出何时不需递归调用，它描述了问题求解的特殊情况或基础情况。**

例2.3

- 用递归函数求一个非负整数的阶乘。

- 分析：
  - 一个正整数的阶乘，可以用其本身乘以其前一个正整数的阶乘，以此类推，直到最终计算1的阶乘为1，另外0的阶乘为1。
  - 阶乘的计算过程可以看作是多个相同操作的反复执行，只不过每次的操作数不同而已。所以，除了循环，还可以用递归函数的形式来实现。

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


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

递归函数的执行过程

```
int main()
{     myFR(4);
      return 0;
}
```

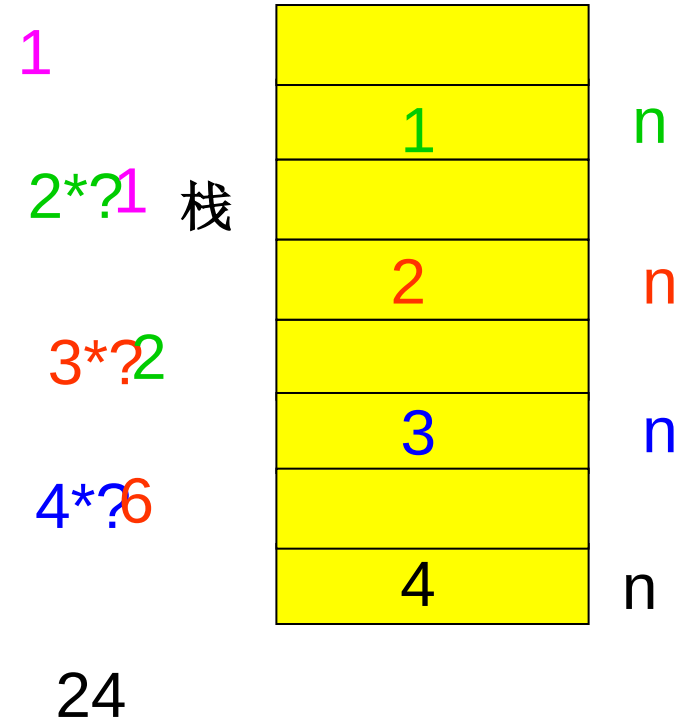
```
int myFR(int n)
{     if(n==0 || n==1)
            return 1;
      else
            return n * myFR(n-1);
}
```

```
4*myFR(3);
```

```
4*3*myFR(2);
```

```
4*3*2*myFR(1);
```

```
4*3*2*1;
```



- Fibonacci数列第n项求值程序可以用一个独立的函数来实现。

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

```
int myFib(int n)
{
    int fib_1 = 1, fib_2 = 1;
    for(int i = 3; i <= n; i++)
    {    int temp = fib_1 + fib_2;
         fib_1 = fib_2 ;
         fib_2 = temp;
    }
    return fib_2;
}
```

例2.4

- 可以看出，Fibonacci数列中每一项的求解是一个同质问题。
- 所以可以用递归函数实现上面的myFib函数。

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

- **小牛问题：若一头小母牛，从出生起第四个年头开始每年生一头母牛，按此规律，第n年有多少头母牛?**

year	未成熟母牛头数	成熟母牛头数	母牛总头数
1	1	0	1
2	1	0	1
3	1	0	1
4	+1	1	2
5	+1 +1	1	3
6	+1 +1 +1	1	4
7	+1 +1 +1 +1	1 +1	6
8	+1 +1 +1 +1 +1 +1	1 +1 +1	9

Fn = Fn-1 + Fn-3.
类似问题的规律：
Fn = Fn-1 + Fn-(m-1)
（n>m）
从出生起第m个年头开始每年生一头母牛

```
int myGetCowR(int year)
{
  if( year < 4 )
      return 1;
  else
      return myGetCowR(year-1) + myGetCowR(year-3);
}
```

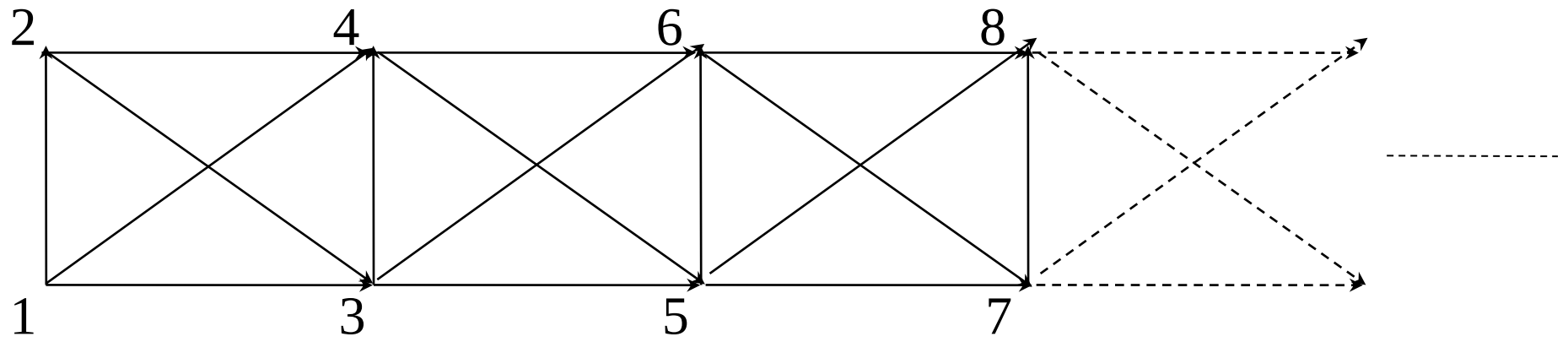
```
int myGetCowR(int year, int m)
{
    if( year < m )
        return 1;
    else
        return myGetCowR(year-1) + myGetCowR(year-m+1);
}
```

用循环（迭代法）实现

```
int myGetCow(int year)
{
  int r1 = 1, r2 = 1, r3 = 1;
  if(year < 4)
          return 1;
  else
  {       for(int i = 4; i <= year; i++)
          {    int temp = r1 + r3;
               r1 = r2;
               r2 = r3;
               r3 = temp;
          }
          return r3;
  }
}
```

例2.5

- 路径问题。
- 根据图写一个递归函数：int Path(int n)；计算从结点1到结点n（n大于1）共有多少条不同的路径。



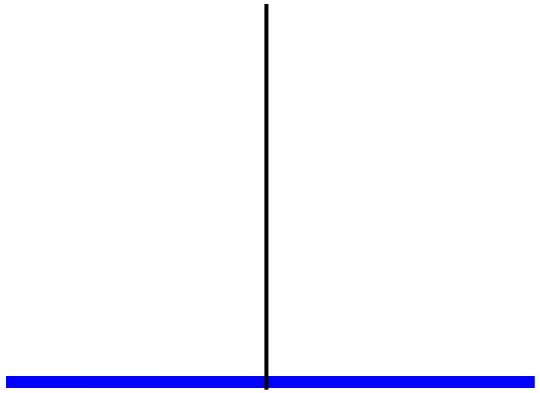
- 分析：从图可以看出，
- n为大于1的奇数时，可经过前2个结点到达，
- n为大于2的偶数时，可经过前3个结点到达。
- 所以，要想求从结点1到结点n的路径，必须先依次计算前面从结点1到结点n-1、n-2、n-3的路径数。
- 可以采用递归函数实现这个问题。

```
int Path(int n)
{
  if( n < 1 )
     return -1;
  else if(n == 1)
     return 1;
  else if(n == 2)
     return 1;
  else if(n % 2 == 1)
     return Path(n - 1) + Path(n - 2);
  else
     return Path(n - 1) + Path(n - 2) + Path(n - 3);
}
```

例2.6

- 河内塔问题。
- 河内塔（Tower of Hannoi），又叫梵塔（Tower of Brahma）或卢卡斯塔（Lucas’ Tower），是法国数学家Édouard Lucas于1883年根据一个印度的古老传说而发明的谜题：
- 假设有A、B、C三根杆子，杆子A上有n个穿孔圆盘，圆盘的尺寸从下到上依次变小；要求按规则把杆子A上的所有圆盘移到杆子C上，移动时可借助杆子B，也可以将从某个杆子移走的圆盘重新移回该杆子；规则是：一次只能将某个杆子最上面的一个圆盘移到另一个杆子上，且每个杆子上的圆盘尺寸只能越来越小（自下而上）；问最少要移动多少次?

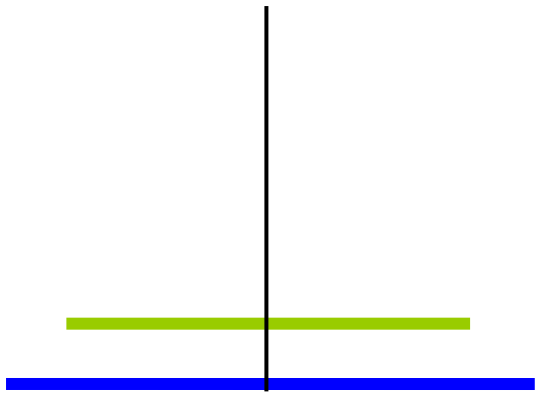
A → C



原来-A

借助-B

目标-C



原来-A

借助-B

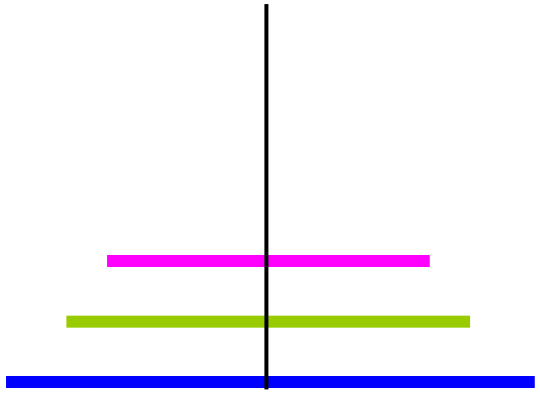
目标-C

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：**
  - 求 H(n) （总移动次数）

（1）当n为1时，只要把圆盘从A移至C就可以了，即移动 1 次。

（2）当n大于1时，该问题可以分解成三个子问题：

① 把n-1个圆盘从A按规则移到B（借助C），即移动 H(n-1) 次；

② 把第n个圆盘从A直接移到C，即移动 1 次；

③ 把n-1个圆盘从B按规则移到C（借助A），即移动 H(n-1) 次。

其中子问题①和③与原问题相同，只是盘子的个数少了一个，子问题②是移动一个盘子的简单问题。所以可以用递归函数实现。

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

- 这个谜题的答案是2^n-1次。按一秒钟移动一次计算，如果有3个圆盘，需要7秒钟，如果有20个圆盘，需要一百多万秒，如果有64个圆盘，则需要五千多亿年！

- 现改变需求，要求编写C程序，输入n，输出移动步骤。

- **分析：**
  - 求 n 个圆盘 A→C 的移动步骤

（1）当n为1时，只要把圆盘从A移至C就可以了，即：A→C。

（2）当n大于1时，该问题可以分解成三个子问题：

① 把n-1个圆盘从A按规则移到B（借助C），即 n-1 个圆盘 A→B 的移动步骤；

② 把第n个圆盘从A直接移到C，即：A→C；

③ 把n-1个圆盘从B按规则移到C（借助A），即 n-1 个圆盘 B→C 的移动步骤。

其中子问题①和③与原问题相同，只是盘子的个数少了一个，子问题②是移动一个盘子的简单问题。所以可以用递归函数实现。

```
…
void Hanoi(char x, char y, char z, int n);
int main( )
{
    int n;
    printf("Input n: ");
    scanf("%d", &n);
    Hanoi('A', 'B', 'C', n);
    return 0;
}
```

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

- 第二个参数始终是移动圆盘的“跳板”

```
void Hanoi(char x, char y, char z, int n)
            //把n个圆盘从x表示的杆子移至z表示的杆子
{
    if(n == 1)
          printf("%c → %c \n", x, z);
         //把第1个圆盘从x表示的杆子移至z表示的杆子
    else
    {     Hanoi(x, z, y, n-1);
         //把n-1个圆盘从x表示的杆子移至y表示的杆子
         printf("%c → %c \n", x, z);
         //把第n个圆盘从x表示的杆子移至z表示的杆子
         Hanoi(y, x, z, n-1);
         //把n-1个圆盘从y表示的杆子移至z表示的杆子
    }
}
```

原来　　借助　　目标

```
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
```

- 递归
  - 一个问题的求解，依赖于一个小规模同质问题的求解

递归函数 vs. 循环流程

- 数据操作：
  - 循环流程是在同一组变量上进行重复操作；
  - 递归函数则是在不同的变量组（属于递归函数的不同实例）上进行重复操作。

- 递归函数的优势：
  - 为某些带有重复性操作的任务提供了一种比采用循环流程更为自然、简洁的实现方式。

- 递归函数的缺陷：
  - **由于递归函数表达的重复操作是通过函数调用来实现的，所以需要额外开销，栈空间的大小会限制递归的深度，从而降低了递归函数的可行性。**
  - **有时会出现重复计算。**

```
int fib(int n)
{ if(n == 1 || n == 2)
    return 1;
  else
    return fib(n-2) + fib(n-1);
}
// 计算fib(n)时要计算fib(n-1)和fib(n-2),
// 计算fib(n-1)时要计算fib(n-2)和fib(n-3)
```

- 可用“动态规划”（Dynamic Programming）解决重复计算问题：即把计算过的内容保存下来（比如保存在数组中），需要时不再计算，直接用保存的结果。这是一种以空间换时间的策略。

小结

- 程序的模块设计
  - 分解与复合：过程抽象、子程序；合理安排、调用
- 单模块程序的设计
  - C语言函数（子程序）基础
    - 函数的概念
    - 函数的定义
    - 函数的调用
    - 函数的参数与返回值
    - 函数的声明
    - 函数的副作用
  - C语言函数的嵌套调用
    - 嵌套调用及其过程
    - 递归

- 要求：
  - 会运用C语言函数实现独立的计算任务，并被main函数或其他函数调用
    - 在函数中运用顺序、分支、循环流程，设计变量
    - 一个程序代码量≈30行，
  - 能够用递归函数实现一个递归算法
  - 能够分析递归函数的功能与结果
  - 能够用模块化思想设计程序：
    - 仔细阅读并分析需求
    - 分解出关键子任务
    - 尝试从大规模问题中分析出小规模同质问题
  - 能够调试、判断程序中的逻辑错误
  - 继续保持良好的编程习惯

Thanks !