第10讲 函数

深圳大学计算机系

学习内容

要编好程序,就要会合理地划分程序中的各个程序块，C++称为函数。

它是完成既定任务的功能（过程）体，它涵盖了数学函数和一般过程．所以基于过程编程本质上就是基于函数编程.

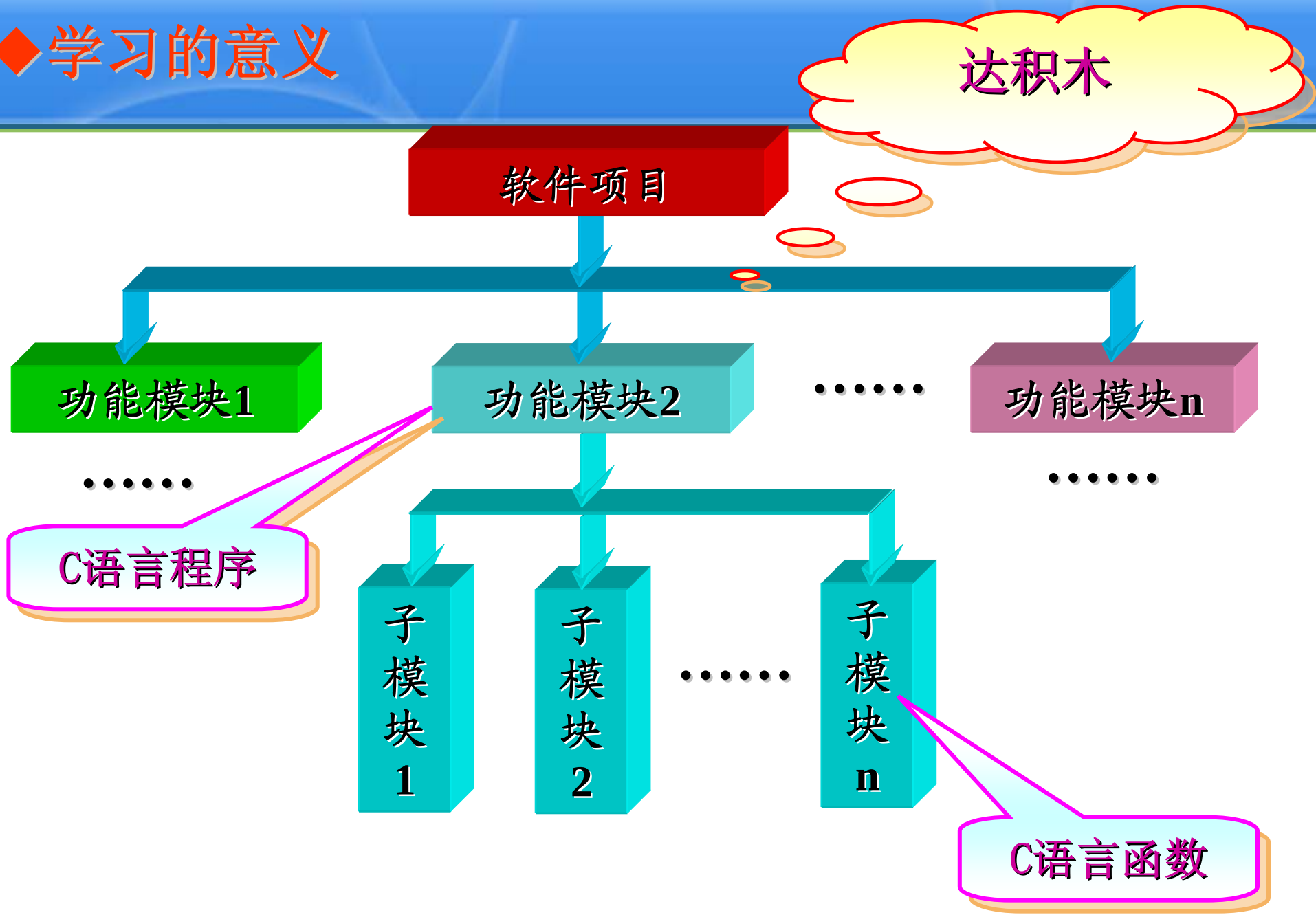
内容：

1. 函数概述
2. 函数原形
3. 全局变量与局部变量
4. 函数调用机制
5. 静态局部变量
6. 递归函数
7. 内联函数
8. 重载函数
9. 默认参数的函数

学习目的

1. 进一步理解多个函数构成一个C程序
2. 进一步了解和熟悉库函数
3. 学会编写自己的函数
4. 理解函数的调用关系
5. 理解函数中参数的传递机制

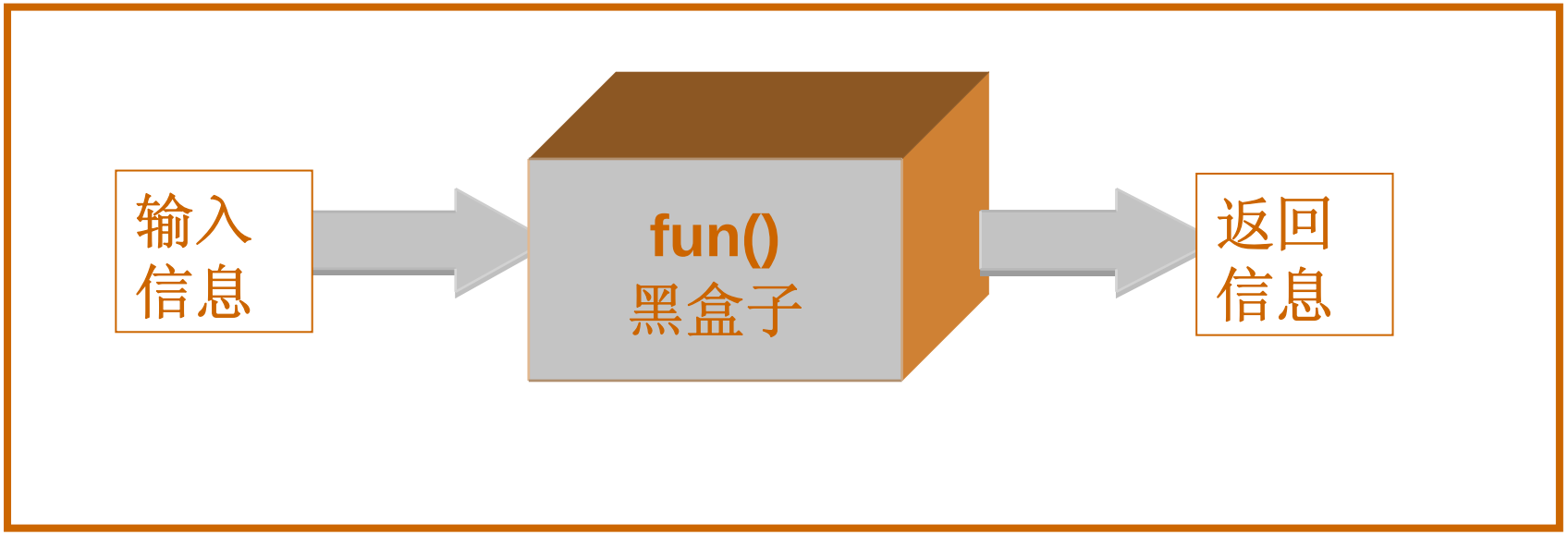
◆学习的意义



5.1 函数概述

函数概述

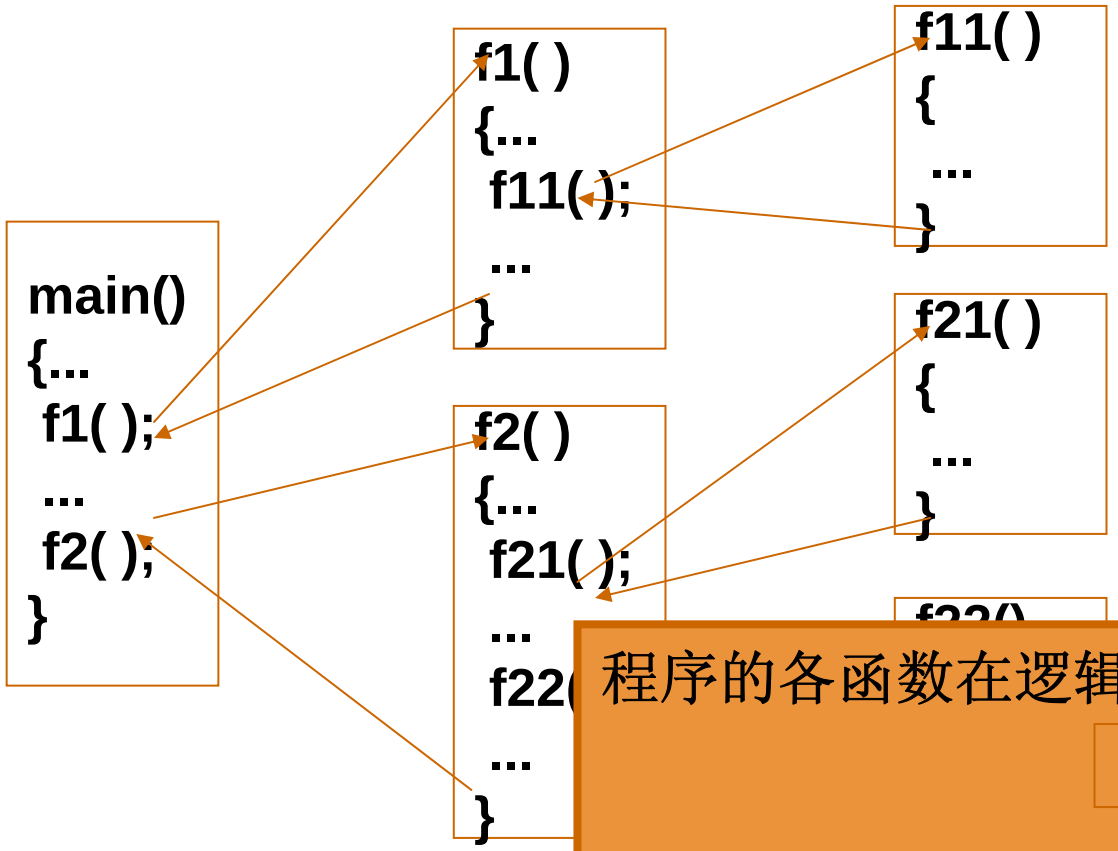
- 模块化程序设计
  - 基本思想：将一个大的程序按*功能*分割成一些*小模块*



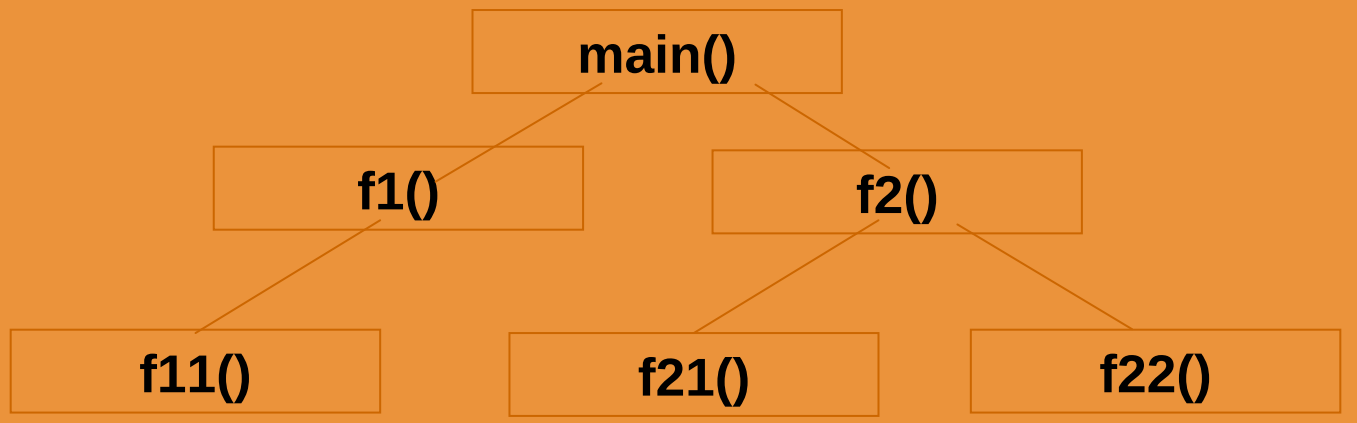
函数概述

- 特点：
  - 各模块*相对独立、功能单一、结构清晰、接口简单*
  - 控制了程序设计的*复杂性*
  - 提高元件的*可靠性*
  - 缩短开发*周期*
  - 避免程序开发的*重复劳动*
  - 易于*维护*和功能*扩充*
- 开发方法：　自上向下，逐步分解，分而治之

函数概述

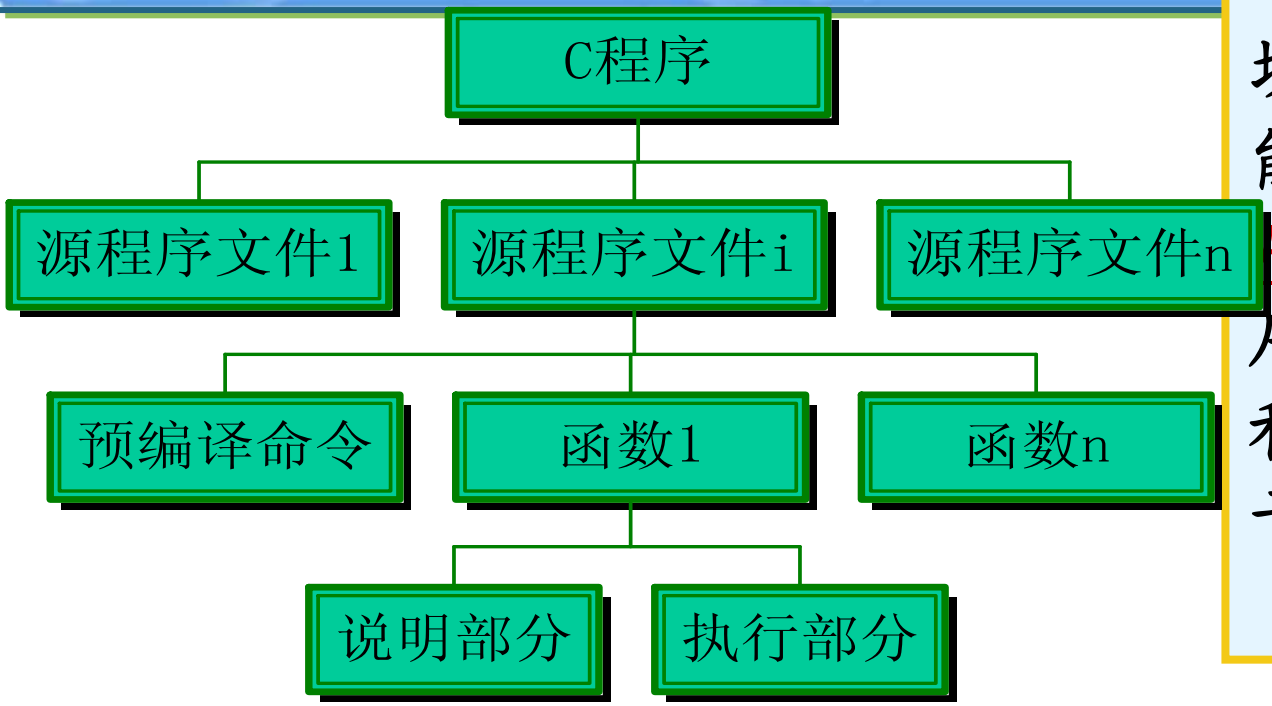


程序的各函数在逻辑关系上形成自上而下的结构



函数

一个较大的程序可分为若干个程序模块，每一个模块用来实现一个特定的功能。在高级语言中用子程序实现模块的功能。子程序由函数来完成。一个C程序可由一个主函数和若干个其他函数构成。



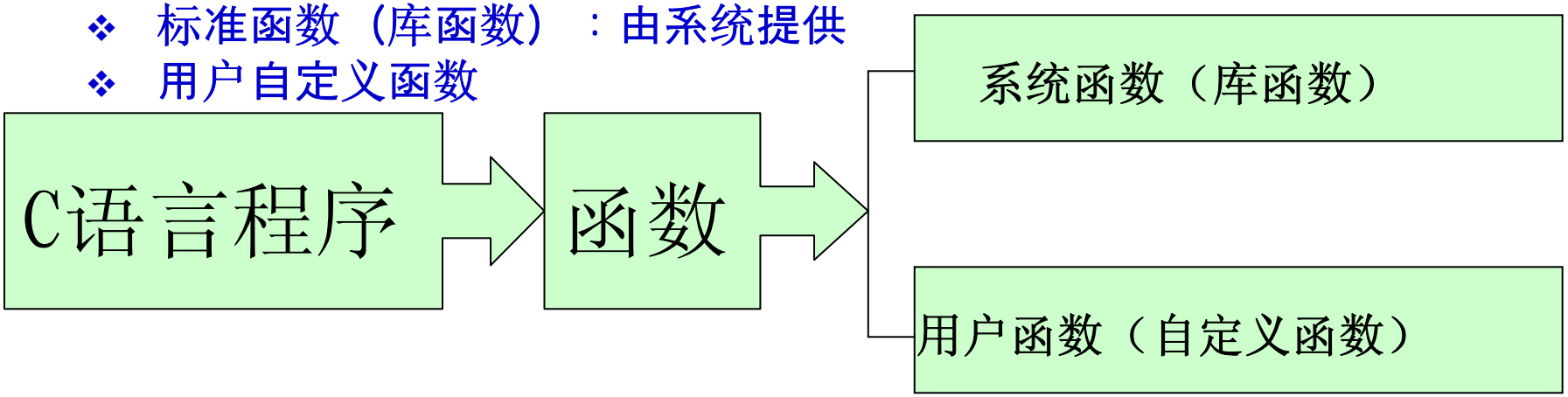
C程序结构

函数间的关系

- C是函数式语言
- 必须有且只能有一个名为main的主函数
- C程序的执行总是从main函数开始，在main中结束
- 函数不能嵌套定义，可以嵌套调用

函数分类

- 从用户角度
  - 标准函数（库函数）：由系统提供
  - 用户自定义函数



- 从函数形式
  - 无参函数
  - 有参函数

使用库函数应注意：

1、函数功能

2、函数参数的数目和顺序，及各参数意义和类型

3、函数返回值意义和类型

4、需要使用的包含文件

函数的定义

函数返回值类型
缺省int型
无返回值void

合法标识符

现代风格：

```
函数类型    函数名（形参类型说明表）
{
        声明部分
        语句部分
}
```

函数体

例　有参函数（现代风格）

```
int max(int x,int y)
{   int z;
    z=x>y?x:y;
    return(z);
}
```

例　有参函数（现代风格）

```
int max(int x, y)
{   int z;
    z=x>y?x:y;
    return(z);
}
```

例　无参函数

```
printstar( )
{  printf("**********\n");  }
```

或

```
printstar(void )
{  printf("**********\n");  }
```

例　空函数

```
dummy( )
{ }
```

函数体为空

练习

请编写函数min,求三个整数的最小值


```
int   min(int x,int y,int z  )
{       int  t;
       if(x<y)     t=x;
       else        t=y;
       if(z<t)     t=z;
       return   t ;
}
```

```
int min(int x,int y,int z)
 {   int r;
      r=x<y?x:y;
      return(r<z?r:z);
 }
```

函数参数和函数的值

```
main()
{    int a,b,c;
     scanf("%d,%d",&a,&b);
     c=max(a,b);
     printf("Max is %d",c);
}
max(int x,int y)
{    int z;
     z=x>y?x:y;
     return(z);
}
```

实参

形参

- 说明：
  - 实参必须有确定的值，求值顺序因系统而定(Turbo C自右向左)
  - 形参与实参个数相等，类型一致，按顺序一一对应
  - 形参与实参类型不同，自动按形参类型转换—函数调用转换
  - 函数调用时才为形参分配内存；调用结束，内存释放

函数类型

1、无参数无返回值的函数

- 定义格式

- 函数用途

此类函数用于完成某项固定的处理任务，执行完成后不向调用者返回函数值。它类似于其它语言的过程。

```
void massage( )          //函数的定义，无参数无返回值
{
  cout<<“This is a massage.\n”;  //函数体，没有声明变量
}
```

函数类型

2、无获取参数有返回值的函数

必须为合法的标识符

```
返回值类型符  函数名 (void)
{
    变量声明部分
    执行部分
}
```

函数体

可以为除数组类型外的任何类型，缺省时，默认为int型

表明无参数，可缺省！

- **函数用途**

此类函数用于完成某项固定的处理任务，执行完成后向调用者返回函数值。

函数类型

2、无获取参数有返回值的函数

例：

```
int sum ( )
{
  int i, tot = 0;
  char key;

  key = getche ( );
  if (key != '0' && key != '1')
       return (-1);
  for (i = (key == '0') ? 2 : 1; i <= 100; i += 2)
       tot += i;
  return ( tot );
}
```

函数sum的功能是：

输入‘0’：计算1～100之间所有偶数之和

输入‘1’：计算1～100之间所有奇数之和

函数类型

3、获取参数但不返回值的函数

至少要有一项，形参之间要用逗号“，”分开

```
void 函数名 (类型符1 形参名1, 类型符2 形参名2, … , 类型符n 形参名n )
{
   变量声明部分
   执行部分
}
```

函数体

形参列表

不允许对形参赋初值

指明形参类型

➢ 函数用途

此类型的函数主要是根据形参的值来进行某种事务的处理。灵活性上要比无形参的函数强，它更能体现调用函数与被调函数之间的数据联系。

函数类型

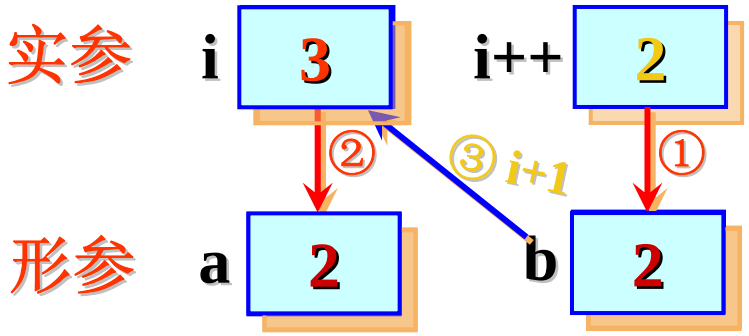
3、获取参数但不返回值的函数

例：

```
#include <stdio.h>
void compare (int a, int b);
void main ( )
{
    int i = 2;
    compare ( i, i++ );
    printf ("i = %d\n", i);
}

void compare ( int a, int b )
{
    printf ("a = %d  b = %d\n", a, b);
    if ( a > b)
        printf ("a > b\n");
    else
     if (a == b)
         printf ("a = b\n");
     else
         printf ("a < b\n");
}
```

运行结果(在VC下)

```
a = 2  b = 2
a = b
i = 3
```



函数类型

4、获取参数并返回值的函数

➢ 定义格式

```
返回值类型符 函数名 (类型符1 形参名1, … … , 类型符n 形参名n )
{
   变量声明部分
   执行部分
}
```

形参列表

函数体

➢ 函数用途

此类型的函数主要是根据形参的值来进行某种事务的处理，同时可将处理后的结果值返回给调用函数 。它最能体现调用函数与被调函数之间的数据联系。

4、获取参数并返回值的函数

例：

```
#include <stdio.h>
int max (in t a, int b);  //函数的原型声明
void main ( )
{
  int a, b, c;
  scanf ("%d%d", &a, &b);
  c = max (a, b);     //函数调用(a、b为实参)
  printf ("the biggest number is : %d\n", c);
}
int max (int a, int b)   //函数定义(a、b为形参)
{
  return (a > b ? a : b);
}
```

5.2 函数原型

函数的调用

```
float add (float x, float y)
{   float z;
    z=x+y;
    return(z);
}
```

怎么调用add函数？

函数的调用

函数调用的一般形式：

函数名（实际参数表）

按函数在程序中出现的位置来分，可以有以下三种函数调用方式：

1．函数语句

例　printstar();

printf(“Hello,World!\n”);//不要求函数带回值

2．函数表达式

例　m=max(a,b)*2;//函数带回一个确定的值

函数的调用

3．函数参数

函数调用作为一个函数的实参。

例如：　　**m = max (a , max (b , c)) ;**

　　printf (“%d”, max (a,b));

函数调用作为函数的参数，实质上也是函数表达式形式调用的一种。

举例:调用add函数

```
#include <stdio.h>
float add(float x, float y)
{   float z;
    z=x+y;
    return(z);
}
main()
{
    float a,b,c;
    scanf("%f,%f",&a,&b);
    c=add(a,b);
    printf("sum is %f",c);
}
```

函数的调用

在一个函数中调用另一函数（即被调用函数）需要具备什么条件呢 ？

(2) 如果使用库函数，用＃ｉｎｃｌｕｄｅ命令将调用有关库函数时所需用到的信息“包含”到本文件中来。

(3) 如果使用用户自己定义的函数，而该函数的位置在调用它的函数（即主调函数）的后面（在同一个文件中），如何调用

举例 对被调用的函数作声明

```
#include <stdio.h>

                          float add(float,float);
main()
{     float add(float,float);
   float a,b,c;
   scanf("%f,%f",&a,&b);
   c=add(a,b);
   printf("sum is %f",c);
}
float add(float x, float y)
{   float z;
   z=x+y;
   return(z);
}
```

```
float   add(float  x,float  y);
float    add(float,float);
```

唯一

多次

函数定义与函数原型有什么不同

函数的定义:对函数功能的确立，包括指定函数名，函数值类型、形参及其类型、函数体等，它是一个完整的、独立的函数单位。

函数原型：把函数的名字、函数类型以及形参的类型、个数和顺序通知编译系统，以便在调用该函数时系统按此进行对照检查。

练习

假设有如下函数定义：

```
Double  func(double a,int b,char c)
{
函数体
}
```

问题:如下几种函数原型的对错？

1、double func(double a, int b,char c);

2、double func(double x, int y,char z);

3、double func(double,int, char);

4、double func(a,b,c);

5、func(double a,int b,char c);

➢ 编写C程序的一般格式

文件包括（如include <stdio.h>等，用于标准库库函数原型声明）
常量定义（根据需要而定，如#define PI 3.1415等）
变量定义（根据需要而定）
用户自定义函数原型声明
main函数
用户自定义函数

➢ 函数的返回

- 形式

- 功能

使程序控制从
值，同时把返值带

例如：

```
void showyes ( )
{
  char key;

  key = getch ( );
  if (toupper(key) != 'Y')
      return;
  printf ("YES! ");
}
```

函数showyes的功能是：
如果输入的字符不是'Y'或'y'，则什么都不输出，直接返回，否则，输出"YES! "

标准库函数，其功能是将小写字符转换成大写字符

➢ 函数的返回

- 说明

函数中可以有多个**return**语句。

```
/*例 求函数y(x)的解*/
int  y_fun(int a)
{  if(a<0)
          return -1;
    else if(a>0)
          return 1;
    else
        return 0;
}
main()
{   int x,y;
    scanf(“%d”,&x);
    y=y_fun(x);
    printf(“x=%d,y=%d\n”,x,y);
}
```

➢ 函数的返回

- 格式

```
#include <stdio.h>
int sum ( );

void main ( )
{
}

int sum
{
   int i,
   for (i
      tot
}
```

因sum函数无return语

注意：如果不将函数调用赋值，返回值将被丢弃！

```
int func ( )
{
  float f = 5;
  f = f / 2;
  return ( f );
}
```

函数将返回2，而不是2.5

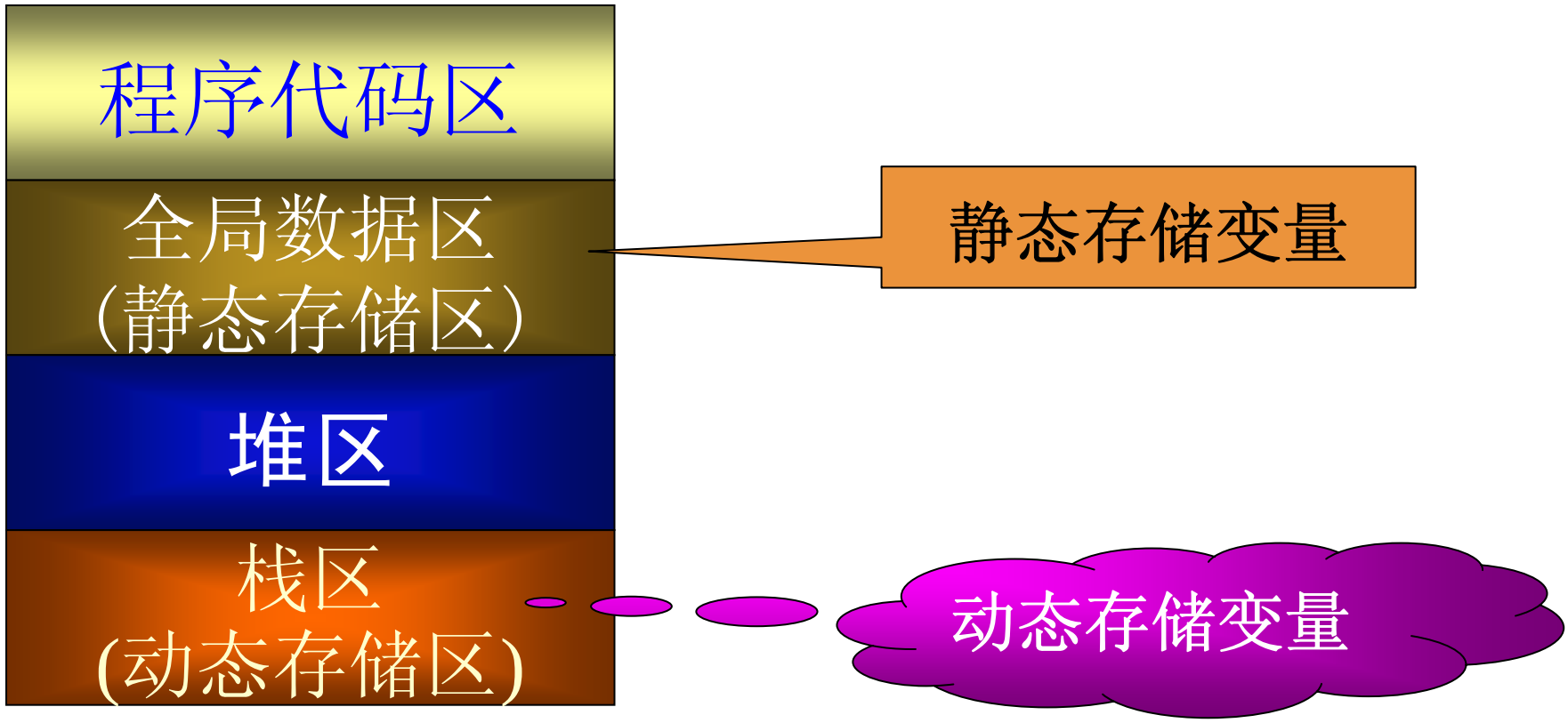
例 函数返回值类型转换

```
int   max(float x, float y)
{   float z;
    z=x>y?x:y;
    return(z);
}
main()
{   float a,b;
    int c;
    scanf("%f,%f",&a,&b);
    c=max(a,b);
    printf("Max is %d\n",c);
}
```

数的返
型转换，
回给调用

5.3 全局变量与局部变量

程序的内存区域



存储分配

变量的作用域和生存期

问题：一个变量在程序的哪个函数中都能使用吗？

变量的作用域

变量的作用域：变量在程序中可以被使用的范围。
根据变量的作用域可以将变量分为局部变量和全局变量。

局部变量及其作用域

- 局部变量/ **Local**： **(**内部变量/ **internal)**在函数或复合语句内定义。

作用域：只在本函数或复合语句内有效。

变量的作用域和生存期

➢ 变量的生存期

变量从被生成到被撤消的这段时间。实际上就是变量占用内存的时间。

按其生存期可分为两种：即动态变量和静态变量

变量只能在其生存期内被引用，变量的作用域直接影响变量的生存期。作用域和生存期是从空间和时间的角度来体现变量的特性。

局部变量作用域和生存期

- **定义**

  在函数内作定义说明的变量，也称为**内部变量**。

- **作用域**

  仅限于函数内，离开函数后不可再引用。

- **生存期**

  从函数被调用的时刻到函数返回调用处的时刻（静态局部变量除外）。

例 局部变量作用域

```
float  f1(int a)
{  int b,c;
   .......
}

char
{   in
    ..
}

mai
{  in
   ...
}
```

a,b,c有效

形参是局部变量

```
int f1 ( int x, int y )
{
  int z;
  z = x > y ? x : y;
  return (z);
}

void f2 ( )
{
  printf ("%d\n", z );
}
```

局部变量

变量x、y、z
的作用域

引用错误!

储

局部变量说明

允许在不同的函数中使用相同的变量名，它们代表不同的对象，分配不同的单元，互不干扰，也不会发生混淆。

```
#include <stdio.h>
void subf ( );
void main ( )
{
 int a, b;
  a=3, b=4;
  printf ("main: a = %d, b = %d\n", a, b);
  subf ( );
  printf ("main: a = %d, b = %d\n", a, b);
}
```

变量名相同

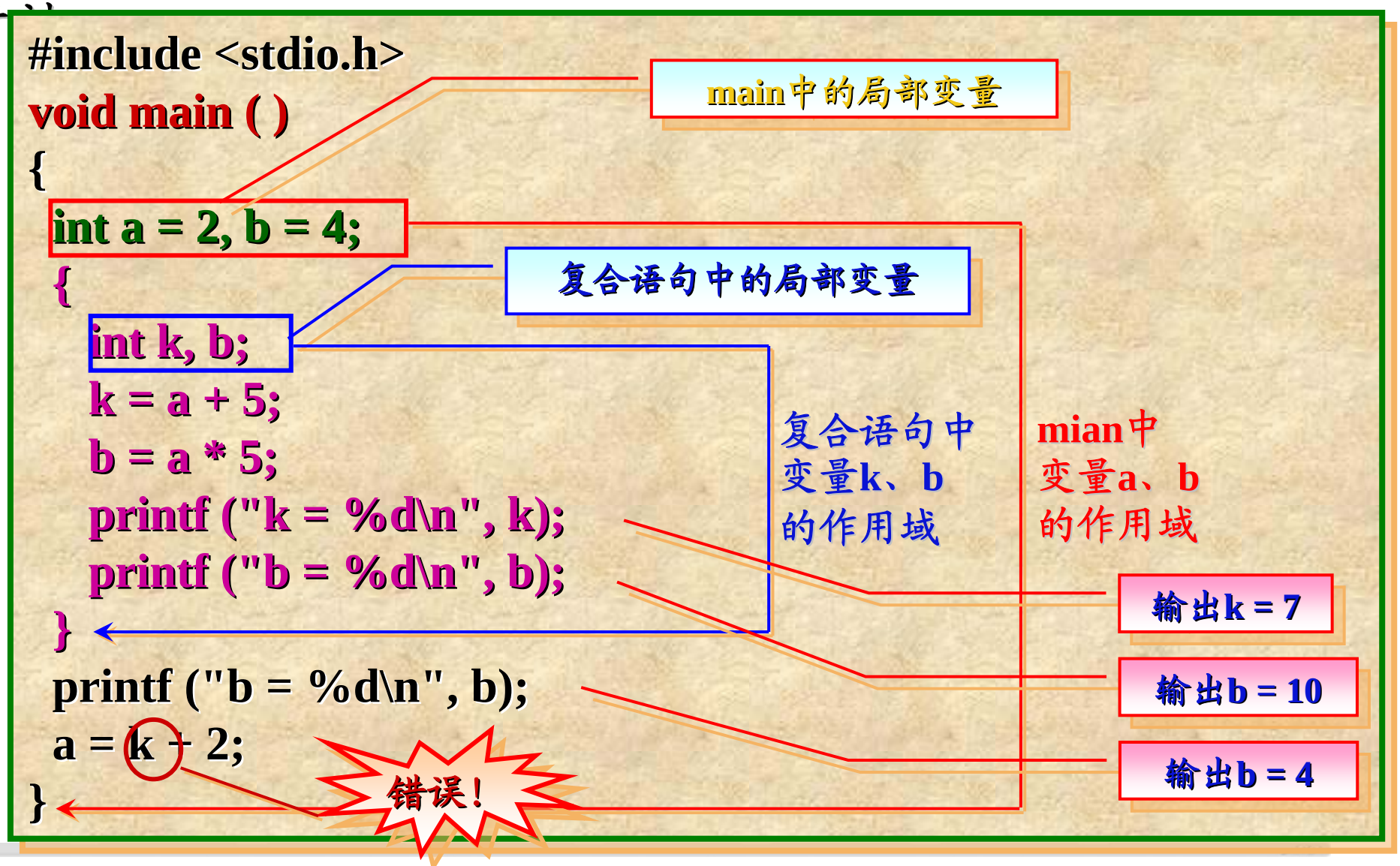
```
void subf ( )
{
  int a, b;
   a = 6, b = 7;
   printf("subf: a = %d,
              b = %d\n",a,b);
}
```

运行结果：

```
main: a = 3, b = 4
subf:  a = 6, b = 7
main: a = 3, b = 4
```

局部变量说明

在复合语句中定义的变量也是局部变量，其作用域只在复合语句范围内。其生存期是从复合语句被执行的时刻到复合语句执行完毕的时



全局变量作用域和生存期

- 定义

  在函数外部作定义说明的

于哪一个函数，而属于一个

- 作用域

  从定义变量的位置开始到

的

-

全

```
#include <stdio.h>
#include <math.h>
int sign ( );

//计算数n的平方根
float sqr ( )
{
  if ( n > 0 )
     return (sqrt(n));
  else
     return (-1);
}
```

错误！

```
float n = 0;

void main ( )
{
  int s;
  float t;
  scanf ("%f", &n);
  s = sign ( );  //取符号
  t = sqr ( );   //取平方根
  printf ("s = %d  t = %f ", s, t);
}

//取数n的符号
int sign ( )
{
  int r = 0;
  if (n > 0)  r = 1;
  if (n < 0)  r = -1;
  return ( r );
}
```

定义全局变量，并赋初值

局部变量

局部变量s、t的作用域

局部变量

局部变量r的作用域

全局变量n的作用域

全局变量说明

对全局
一般形

extern

```
void gx ( ), gy ( );
void main ( )
{
  extern  int x, y;
  printf ("1: x = %d\ty = %d\n", x, y);
  y = 246;
  gx ( );  gy ( );
}
extern int x, y;
void gx ( )
{
  x = 135;
  printf ("2: x = %d\ty = %d\n", x, y);
}
int x = 0, y = 0;
void gy ( )
{
  printf ("3: x=%d\ty=%d\n", x, y);
}
```

全局变量说明

全局变量说明

全局变量定义

说明后
的作用域

说明后
的作用域

未说明前
的作用域

运行结果:

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

全局变量和局部说明

- 全局变量若在定义时未赋初值，则系统自动赋初值0，赋初值只一次；
- 局部变量未赋初值，其内容不可预料，赋初值可以多次。

exc10-1.cpp

```
//**********************
//**    ch5_1.cpp    **
//**********************

#include <iostream.h>

int func1();
int func2();

void main()
{
  func1();
  cout <<func2() <<endl;
}
```

```
int func1()
{
  int n=12345;
  return n;
}

int func2()
{
  int m;
  return m;}
```

运行结果：

12345

5.4 函数调用机制

C++函数调用过程

1.建立被调函数的栈空间;

2.保护调用函数的运行状态和返回地址

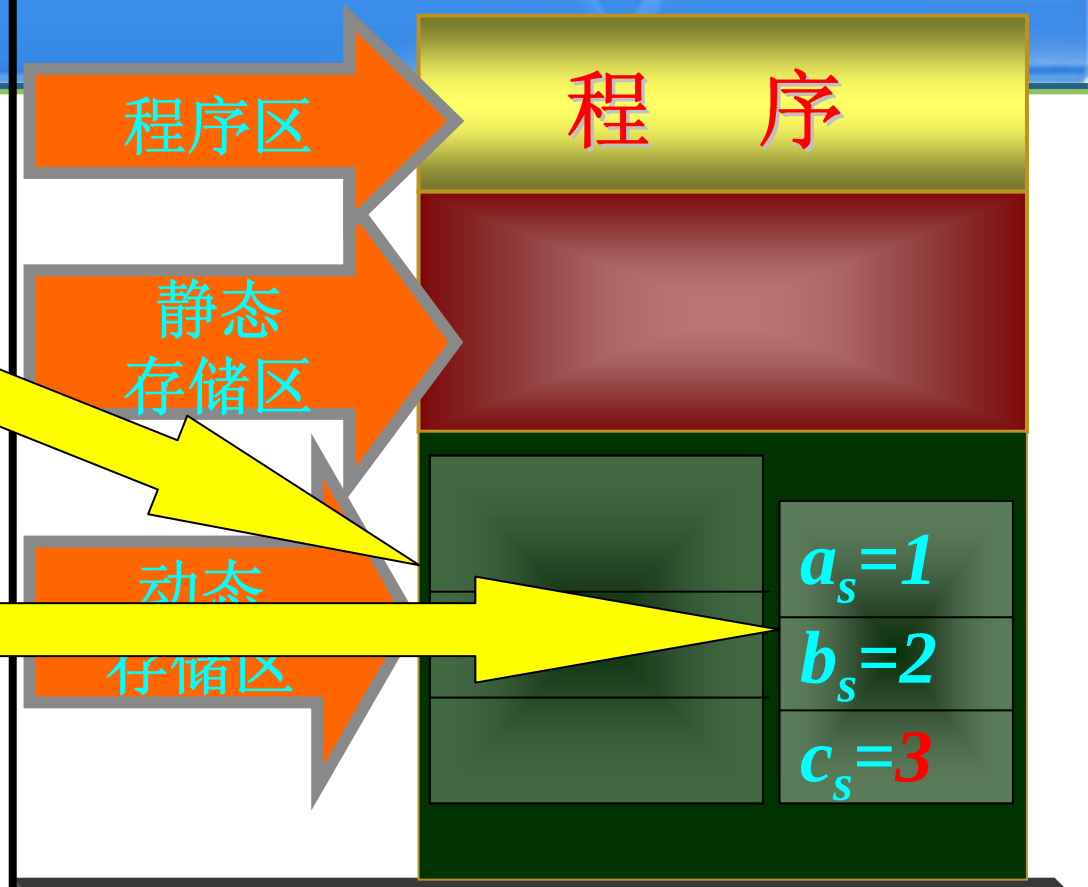
3.传递参数

4.将控制转交被调函数

观察下列程序运行时变量的存储情况

```
main()
{ int a,b,c;
  printf("Enter a,b\n");
  scanf("%d%d",&a,&b);
  c=sum(a,b);
  printf("Sum=%d\n",c);
}
sum(int a,int b)
{ int c=0;
  c=a+b;
  return(c);
}
```

运行结束



Enter a,b:
1　2 <回车>
Sum=3

函数参数的传递方式

- 值传递方式/ Pass by value
  - 方式：函数调用时,为形参分配单元,并将实参的值复制到形参中；调用结束，形参单元被释放，实参单元仍保留并维持原值
  - 特点：
    - 形参与实参占用不同的内存单元
    - 单向传递

地址传递好处：
1.节约内存
2.提高效率
3.同步更新

- 地址传递 / Pass by reference
  - 方式：函数调用时，将数据的存储地址作为参数传递给形参
  - 特点：
    - 形参与实参对应同样的存储单元
    - “双向”传递
    - 实参和形参必须是地址常量或变量

例：交换两个数(值传递方式)

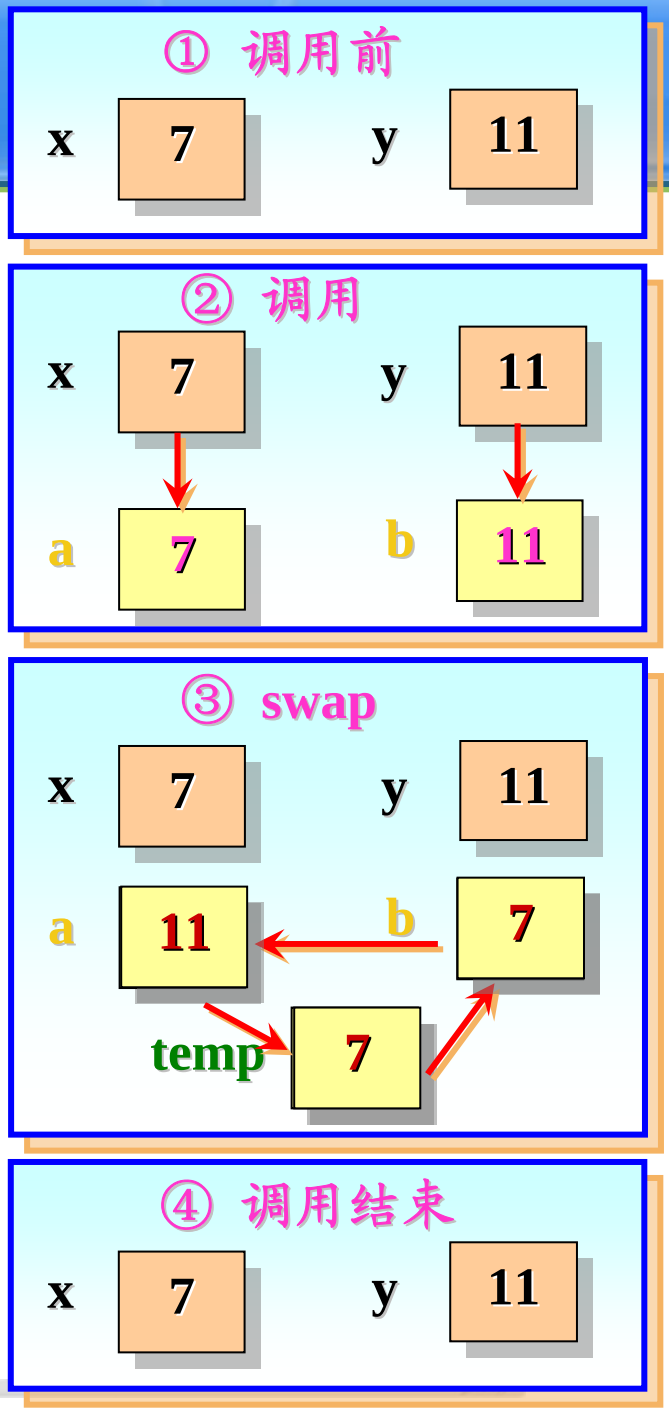
```
#include <stdio.h>
void swap (int a, int b);
void main ( )
{
    int x = 7, y = 11;
    printf ("before swapped: ");
    printf ("x=%d, y=%d\n", x, y);
    swap (x, y);
    printf ("after  swapped: ");
    printf ("x=%d, y=%d\n", x, y);
}

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

Why?

运行结果:

before swapped: x = 7, y = 11

after swapped: x = 7, y = 11



地址传递方式

- **方式：**
  函数调用时，将数据的存储地址作为参数传递给形参
- **特点：**
  ① 形参与实参占用同样的存储单元
  ② 双向传递
  ③ 实参和形参必须是地址常量或变量

用数组名作为函数参数时还应注意以下几点：

✓ 形参数组和实参数组的类型必须一致，否则将引起错误。

✓ 形参数组和实参数组的长度可以不相同，因为在调用时，只传送首地址而不检查形参数组的长度。

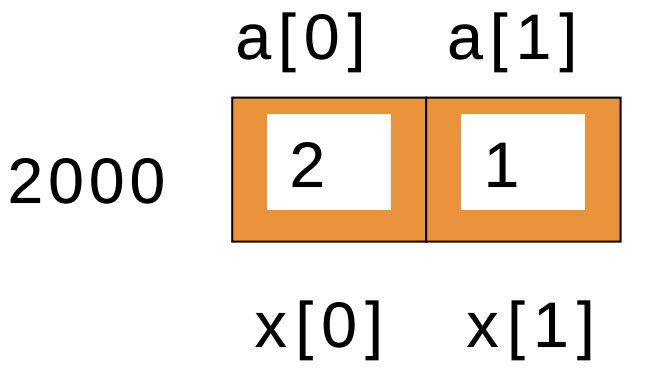
✓ 多维数组也可以作为函数的参数。在函数定义时对形参数组可以指定每一维的长度，也可省去第一维的长度。

除了用数组名作为函数参数来实现参数的地址传递以外，其实还有一种应用**更广的地址传递方法，那就是用指针变量来作为函数的形参**。

例　数组名作参数（交换两个数）

形参用数组定义，⇔int x[ ] 是地址变量

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



地址传递

例　数组元素与 数组名作函数参数比较

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

```
#include <stdio.h>
swap(int a,int b)
main()
{int a[2]={7,11};;
swap(a[0], a[1]);
  printf("a[0]=%d\na[1]=%d\n",a[0],a[1]);
}
swap(int a,int b)
{   int temp;
    temp=a; a=b; b=temp;
}
```

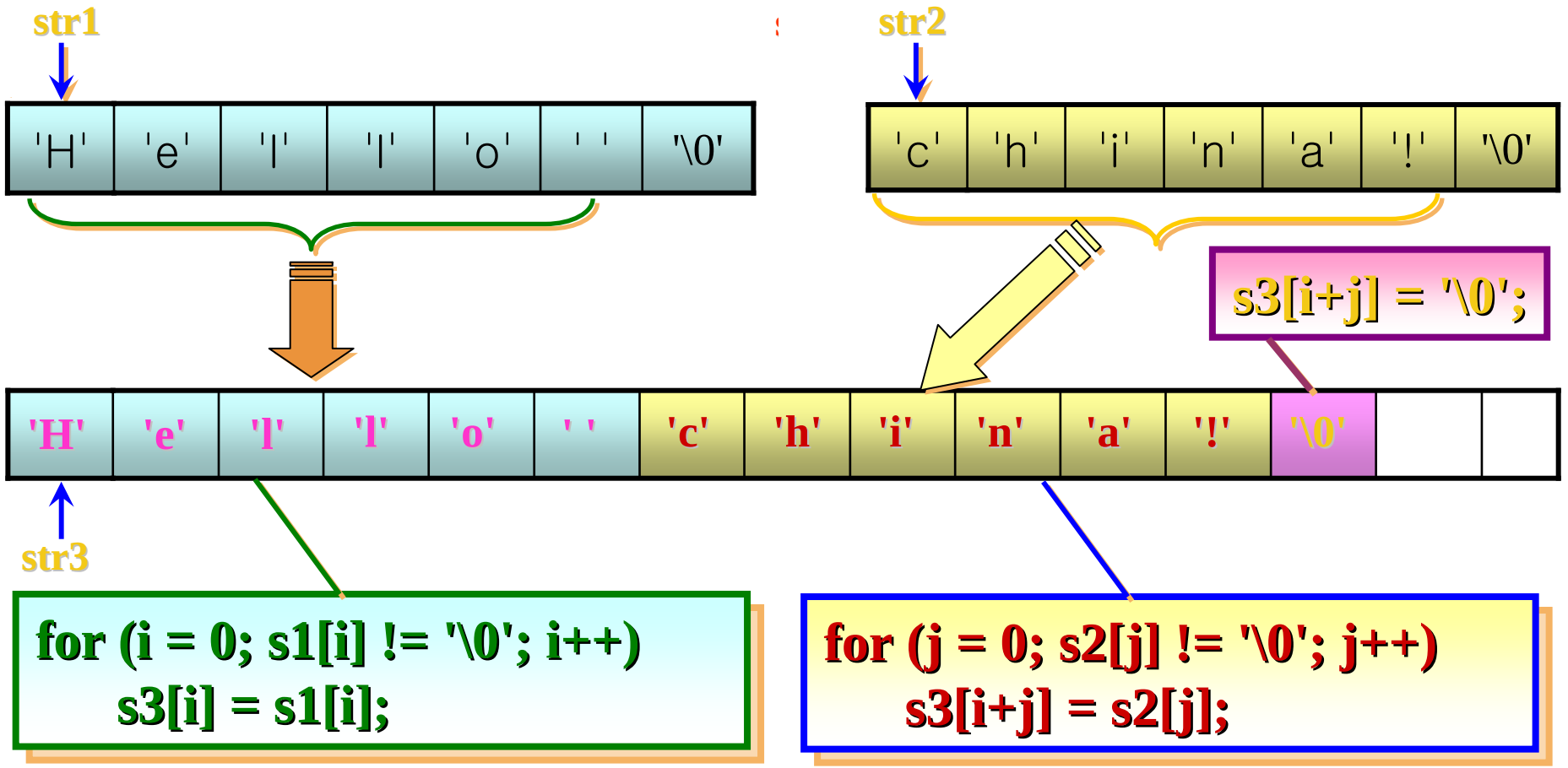
例：将任意两个字符串连接成一个字符串（数组名作为函数参数实现地址传递方式）

```
#include <stdio.h>
void mergestr (char s1[ ], char s2[ ], char s3[ ]);
void main ( )
{
    char str1[ ] = {"Hello "};
    char str2[ ] = {"china!"};
    char str3[40];
    mergestr (str1, str2, str3);
    printf ("%s\n", str3);
}
void mergestr (char s1[ ], char s2[ ], char s3[ ])
{
    int i, j;
    for (i = 0; s1[i] != '\0'; i++)  //将s1复制到s3中
        s3[i] = s1[i];
    for (j = 0; s2[j] != '\0'; j++)  //将s2复制到s3的后边
        s3[i+j] = s2[j];
    s3[i+j] = '\0';              //置字符串结束标志
}
```

运行结果：

Hello china!

调用前 → 调用 → 连接 → 补\0 → 调用结束



多维数组名作函数参数

用多维数组名作为函数实参和形参。在被调函数中对形参数组定义时可以指定每一维的大小 。对于二多维数组,原型中声明整数数组参数的形式只能省略左边的方括号。

例：**max_value (int a r r a y [ ] [4]);**

练习　有1个3X4的矩阵，求所有元素中的最大值.


```
#include <stdio.h>
void  main （）
 { max_value ( int  a  r  r a y  [ ]  [4] );
  int a[3][4]={{1,3,5,7},{2,4,6,8},{15,17,34,12}};
  printf（″ max value is ％d ＼n″ ,
              max_value(a) ）;
 }
```

```
max_value （ int array［ ］ ［4］ ）
 { int i， j， k， 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）；
}
```

运行结果如下：

Max value is 34

5.5 静态局部变量

静态变量（static类别）

除形参外，局部变量和全局变量都可以定义为静态变量。

静态变量

- 局部静态变量（或称内部静态变量）
- 全局静态变量（或称外部静态变量）

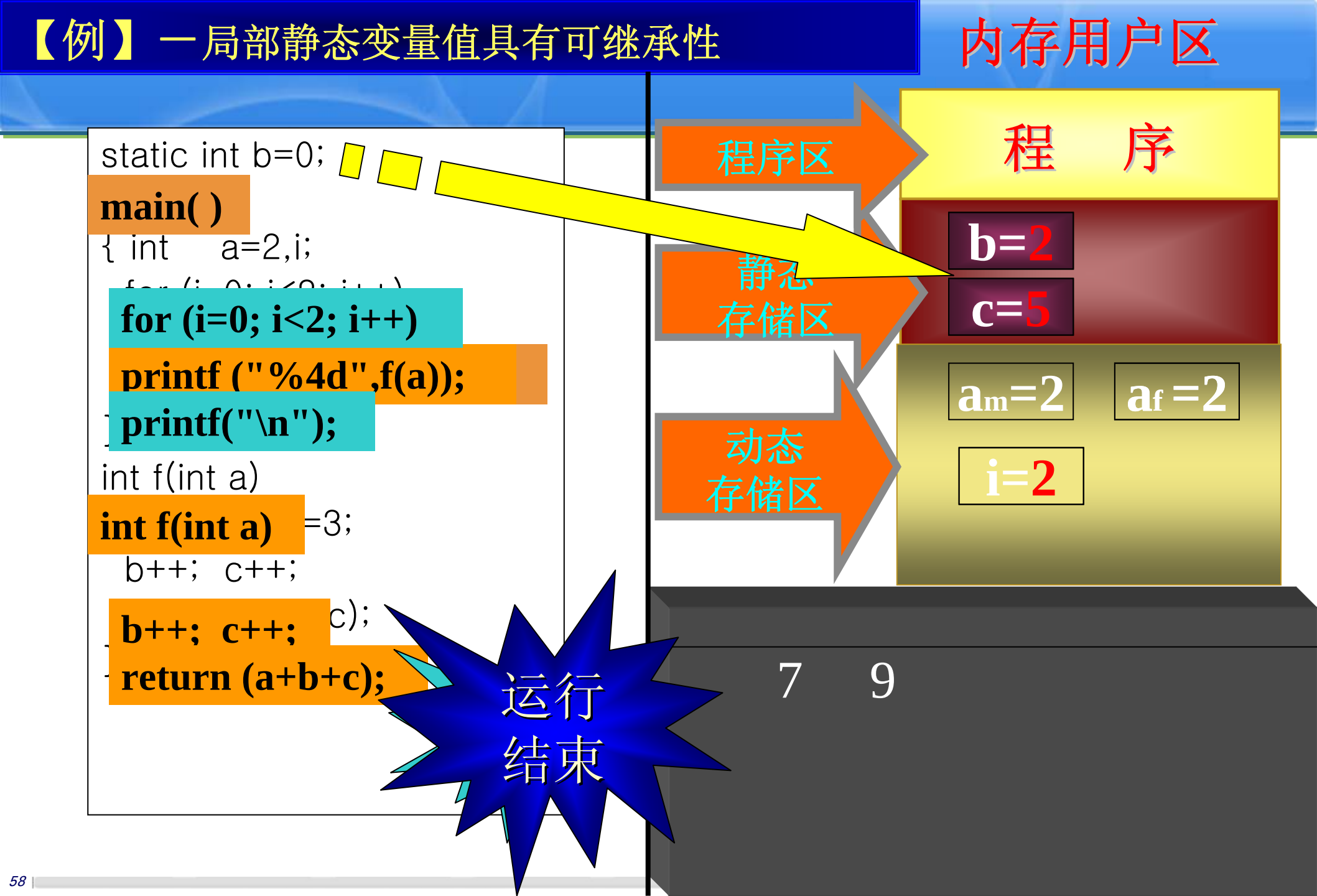
不能省

```
static int a;
main( )
{ float x,y;
  … }
f( )
{ static int b=1;
  ……
}
```

- static int a; → 静态全局变量
- float x,y; → 局部变量
- static int b=1; → 静态局部变量

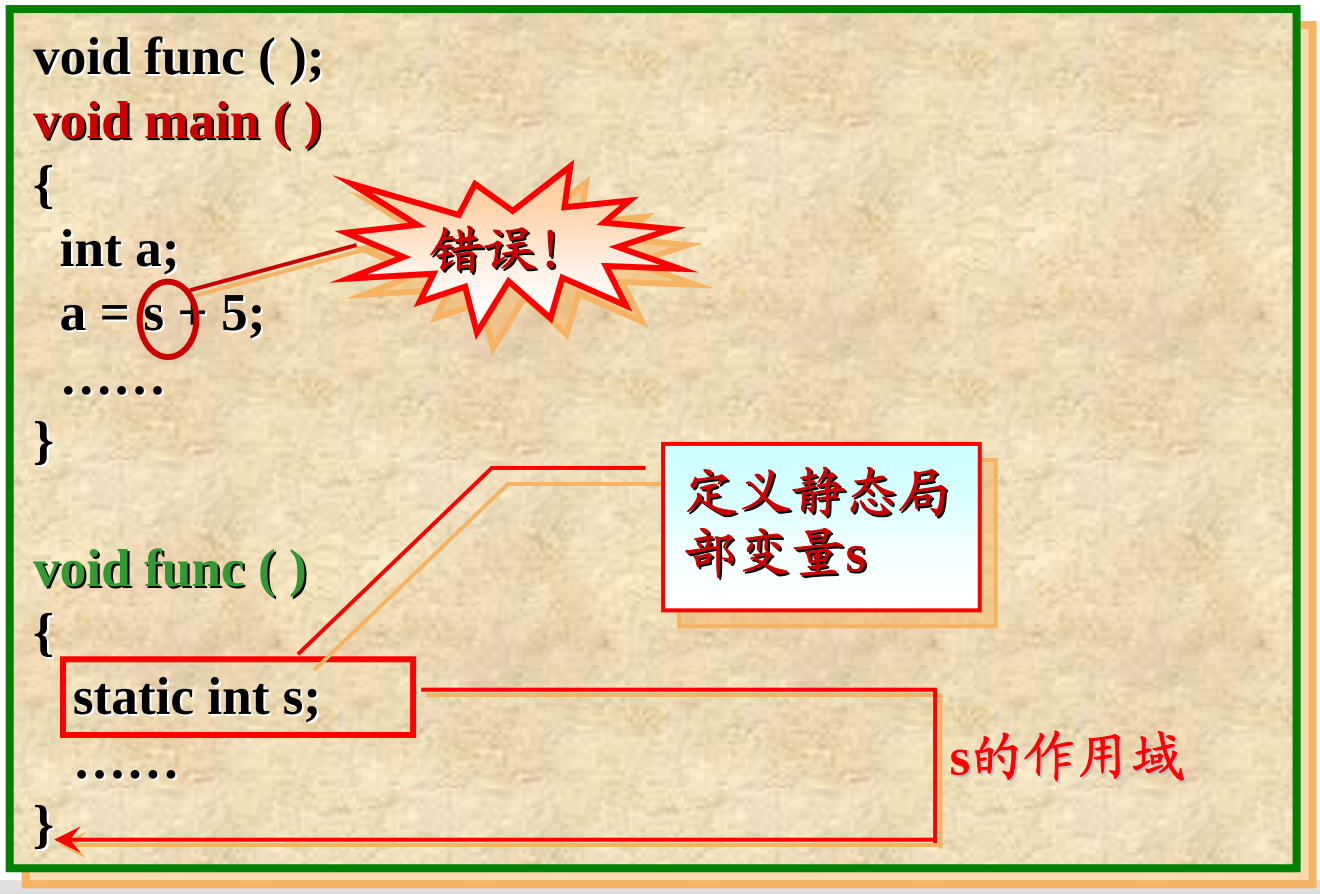
【例】—局部静态变量值具有可继承性

```
static int b=0;
main( )
{ int     a=2,i;
  for (i=0; i<2; i++)
  printf ("%4d",f(a));
  printf("\n");
}
int f(int a)
int f(int a)      =3;
  b++;  c++;
  b++;  c++;      c);
  return (a+b+c);
}
```



静态局部变量与局部变量之比较

- 静态局部变量生存期长，为整个源程序。局部变量生存期短。
- 静态局部变量作用域只在定义它的函数中有效



静态局部变量与局部变量之比较

- 静态局部变量若在定义时未赋初值，则系统自动赋初值0
- 静态局部变量赋初值只一次，而局部变量赋初值可能多次

```
#include <iostream.h>  //  ch5_2.cpp
void func();
int n=1;    //全局变量
void main()
{  static int a;    // 静态局部变量
  int b= -10;      // 局部变量
  cout <<"a:" <<a
       <<"  b:" <<b
       <<"  n:" <<n <<endl;
  b+=4;
  func();
  cout <<"a:" <<a
       <<"  b:" <<b
       <<"  n:" <<n <<endl;
  n+=10;
  func();
}
```

```
void func()
{
  static int a=2;    // 静态局部变量
  int b=5;           // 局部变量
  a+=2;
  n+=12;
  b+=5;
  cout <<"a:" <<a
       <<"  b:" <<b
       <<"  n:" <<n <<endl;
}
```

运行结果：

```
a:0 b: -10 n:1
a:4 b: 10 n:13
a:0 b: -6 n:13
a:6 b: 10 n:35
```

exc10-2

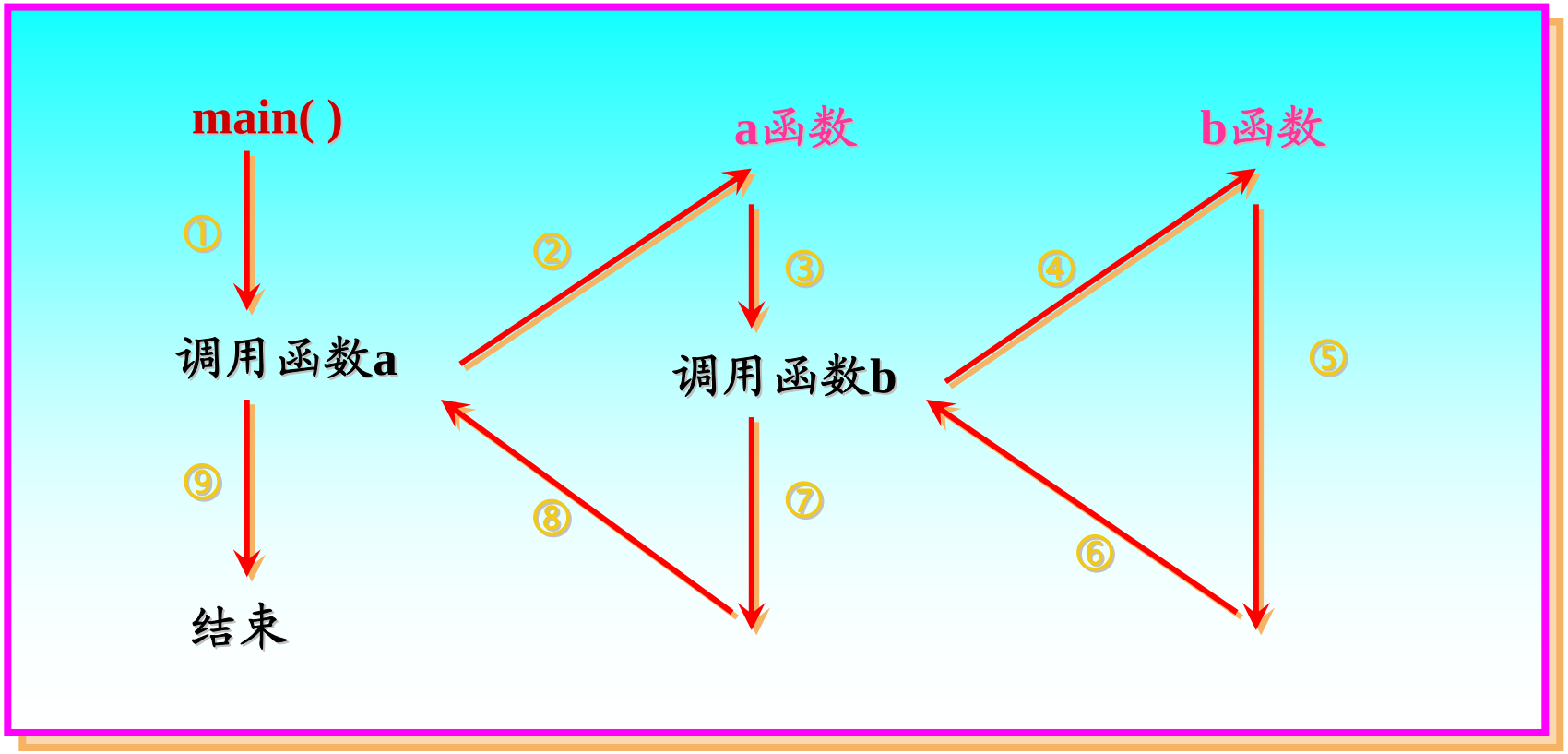
练习 读程序，写结果。

```
#include <stdio.h>
int fac(int n);
void main()
    {int i;
       for(i=1;i<=5;i++)
         printf("%d!=%d\n",i, fac(i));
}
int fac(int n)
    {static int f=1;
       f=f*n;
       return(f);
     }
```

5.6 递归函数

函数的嵌套调用机制

C++规定：函数定义不可嵌套，但可以嵌套调用函数



——函数嵌套调用的示意图

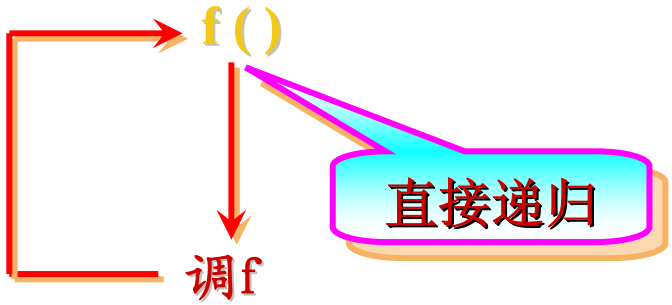
递归函数

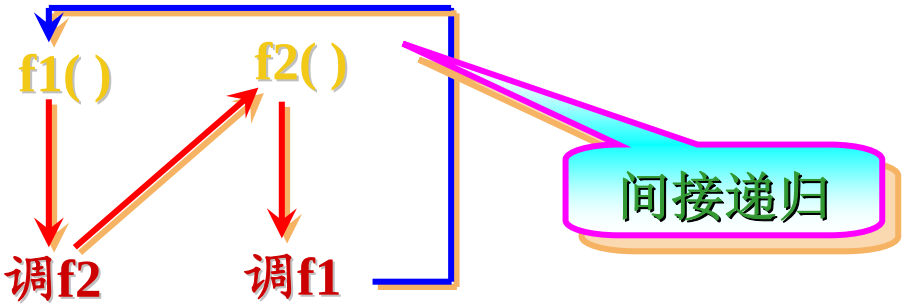
- 定义：函数直接或间接的调用自身的函数。

```
int  f (int x)
{
   int y, z;
   ……
   z = f (y);
   …….
   return (2*z);
}
```

```
int  f1 (int x)
{
   int y,z;
   ……
   z = f2 (y);
   …….
   return (2*z);
}
```

```
int  f2 (int t)
{
   int a,c;
   ……
   c = f1 (a);
   …….
   return (3+c);
}
```





- 说明

- C++编译系统对递归函数的自调用次数没有限制
- 每调用函数一次，在内存堆栈区分配空间，用于存放函数变量、返回值等信息，所以递归次数过多，可能引起堆栈溢出。

函数的递归调用(Recursion)

1. 递归过程必须解决两个问题

   递归计算的公式

   递归结束的条件

2. 递归过程的算法描述：

```
if （递归结束条件）
    return （递归结束条件下的返回值）；
else
    ……
    return （递归计算公式）；
```

3. 递归调用函数的调用方法和一般函数的调用方法完全相同

函数的递归调用(Recursion)

例：利用递归函数计算*n*的阶乘。

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

问题分析：

1、由数学知识可知：当n>1时，n!=n*(n-1)!；当n=0或 n=1时，n!=1。

2、可见，该问题可以用递归方法编程解决。

3、该程序分为两个函数，主函数和求阶乘函数fac。

算法要点：

定义一个函数fun(int n)求n!，在主函数中输入n 的值，调用fun(n)。

数学公式的C++语言表示：

```
if(n==1||n==0)
   return (1);
else
    return(n * fun(n－1));
```

例 求n的阶乘

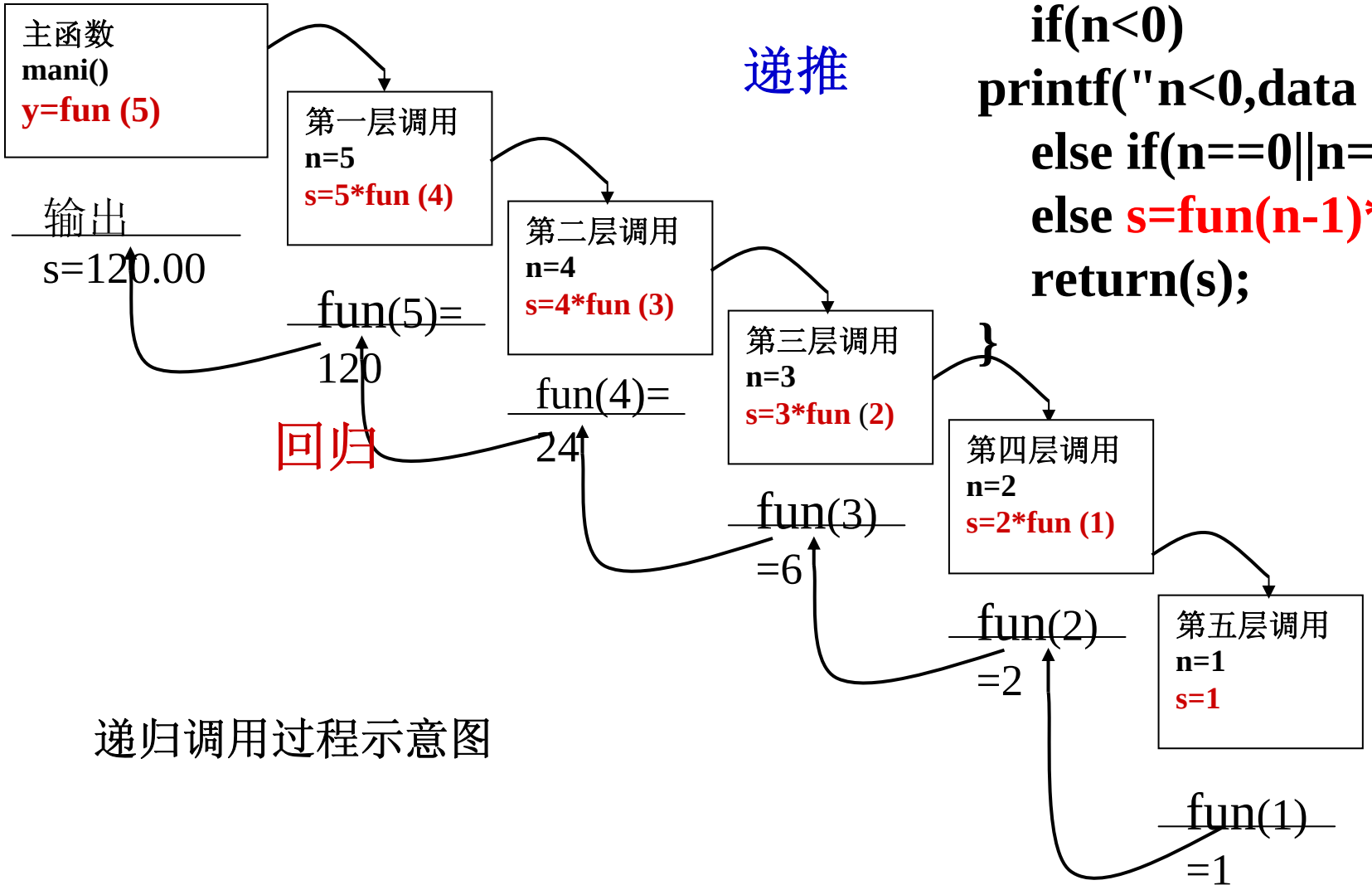
$$n!=\begin{cases}1 & (n=0,1)\\ n\cdot(n-1)! & (n>1)\end{cases}$$

exc10-2

```
/*ch5_3.c*/
#include <stdio.h>
int fun(int n);
main()
{   int n, y;
    printf("Input a integer number:");
    scanf("%d",&n);
    y=fun(n);
    printf("%d! =%15d\n",n,y);
}
int fun(int n)
{   int s;
    if(n<0)  printf("n<0,data error!");
    else if(n==0||n==1)  s=1;
    else s=fun(n-1)*n;
    return(s); }
```

例 求5的阶乘

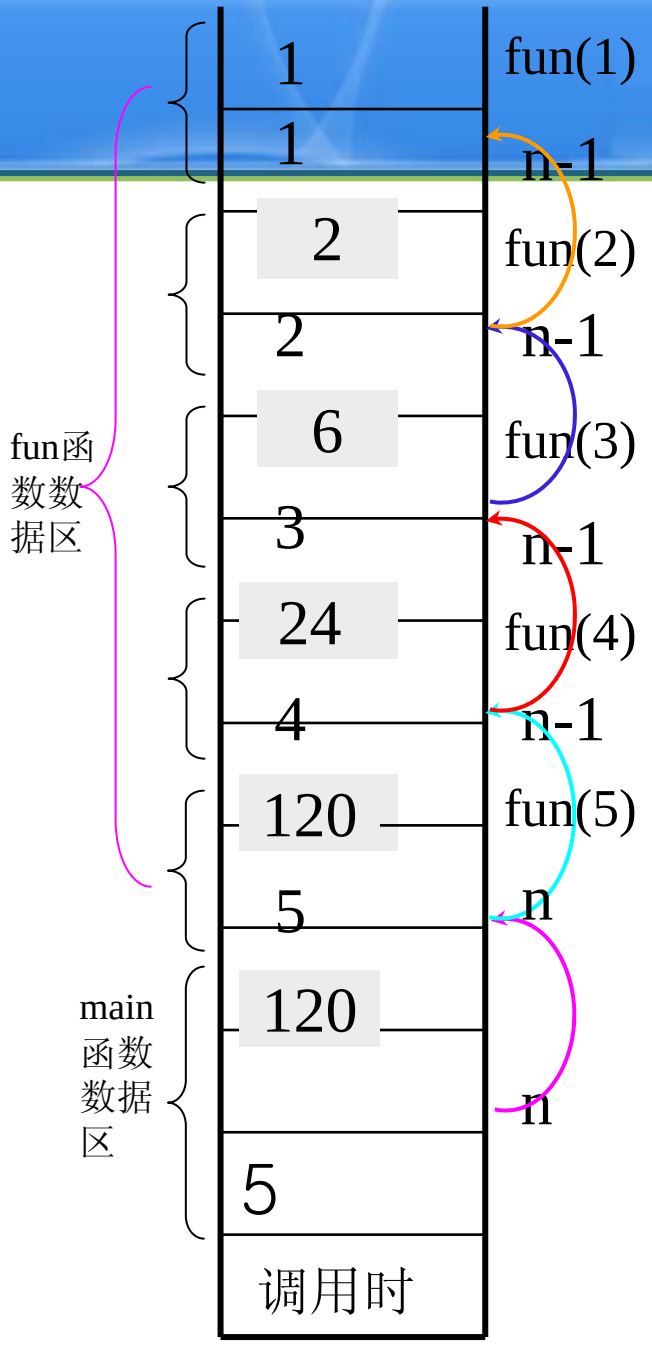
```
int fun(int n)
{   int s;
    if(n<0)
printf("n<0,data error!");
    else if(n==0||n==1)  s=1;
    else s=fun(n-1)*n;
    return(s);
}
```



递归调用过程示意图

求5！执行如下：

120


1
1
fun(1)
n-1
2
2
fun(2)
n-1
6
3
fun(3)
n-1
24
4
fun(4)
n-1
120
5
fun(5)
n
120
n
5
调用时
fun函数数据区
main函数数据区

递归函数非递归化

思考：

将求n!的递归函数非递归化

```
long fac(int k )   /*求k的阶乘*/
{      long rtn=1;
       int i;
       for(i=1;i<=k;i++)
       rtn*=i;
       return(rtn);
 }
```

练习

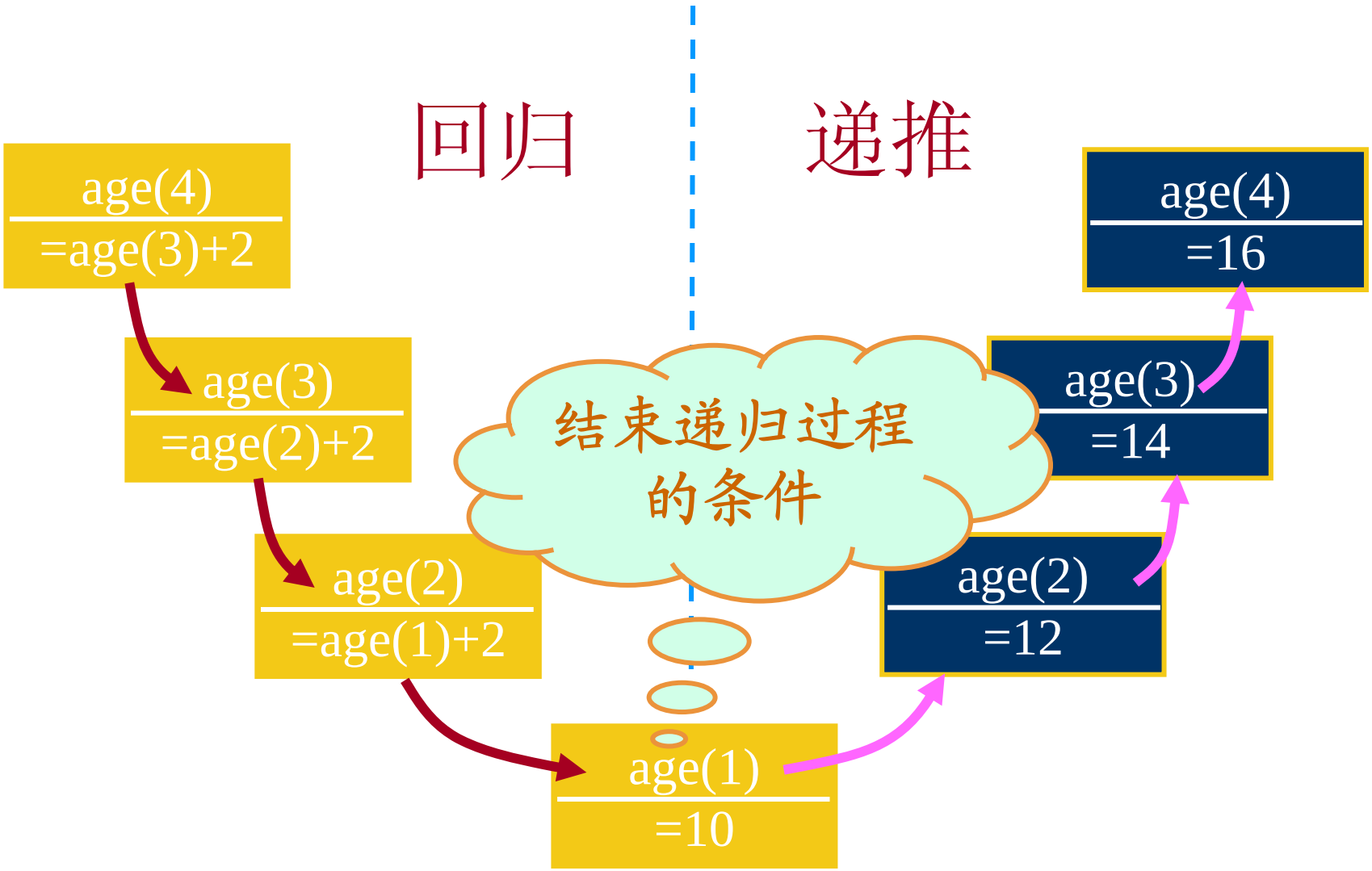
有4个人坐在一起，问第4个人多少岁？他说比第3个人大２岁。问第3个人岁数，他说比第2个人大２岁。问第2个人，又说比第1个人大２岁。最后问第１个人，他说是１０岁。请问第4个人多大。

age（4）＝age（3）＋2
age（3）＝age（2）＋2
age（2）＝age（1）＋2
age（1）＝10

可以用数学公式表述如下：

$$age(n) = \begin{cases} 10 & n=1 \\ age(n-1)+2 & n>1 \end{cases}$$

练习


回归
递推
age(4)
=age(3)+2
age(3)
=age(2)+2
age(2)
=age(1)+2
age(1)
=10
结束递归过程
的条件
age(4)
=16
age(3)
=14
age(2)
=12

练习

编程实现：

```
/*求年龄的递归函数*/
int age(int n)
{
    int c;
    if(n==1) c=10;
    else c=age(n-1)+2;
    return(c);
}
```

```
/*主函数求第四个人的年龄*/
#include < stdio.h>
void main()
{
  printf("Age=%d\n",age(4));
}
```

运行结果：Age=16

练习

编程实现：

```
/*求年龄的递归函数*/
int age(int n)
{
    int c;
    if(n==1) c=10;
    else c=age(n-1)+2;
    return(c);
}
```

```
/*主函数求第四个人的年龄*/
#include < stdio.h>
void main()
{
  printf("Age=%d\n",age(4));
}
```

运行结果：Age=16

练习:阅读程序训练

```
# include<stdio.h>
int f(int m, int n)
{  if(m%n==0) return n;
     else return f(n, m%n);
}

void main()
{  printf("%d\n", f(840, 48));
}
```

输出结果为： **24**

5.7 内联函数

内联函数（Inline Functions）

- 做法：将一些反复被执行的简单语句序列做成小函数
- 用法：在函数声明前加上inline关键字
- 作用：不损害可读性又能提高性能

内联函数（Inline Functions）

频繁调用的函数:
用昂贵的开销换取
可读性

```
//Ch5_4
 #include<iostream>
int isnumber(char);    //函数声明
void main()
{  char c;
  while((c=cin.get())!='\n')
     if( isnumber(c) )      //调用一个小函数
       cout <<"you entered a digit\n";
     else
       cout <<"you entered a non_digit\n";
}
int isnumber(char ch)      //函数定义
{
  return (ch>='0' && ch<='9')? 1:0;
}
```

内联函数（Inline Functions）

内嵌代码：开销虽少，但可读性差

```
//Ch5_4
 #include<iostream>
int isnumber(char);    //函数声明
void main()
{  char c;
  while((c=cin.get())!='\n')
    if(ch>='0' && ch<='9')? 1:0;
      cout <<"you entered a digit\n";
    else
      cout <<"you entered a non_digit\n";
}
```

内联函数（Inline Functions）

内联方式：开销少，可读性也佳

内联标记放在函数声明的前面

```
#include<iostream>
inline int isnumber(char);    //函数声明
void main()
{  char c;
  while((c=cin.get())!='\n')
    if( isnumber(c) )      //调用一个小函数
      cout <<"you entered a digit\n";
    else
      cout <<"you entered a non_digit\n";
}
int isnumber(char ch)      //函数定义
{  return (ch>='0' && ch<='9')? 1:0;
}
```

内联函数的使用经验

- 函数体适当小，且无循环或开关语句，这样就使嵌入工作容易进行，不会破坏原调用主体．如：排序函数不能内联
- 程序中特别是在循环中反复执行该函数，这样就使嵌入的代码利用率较高．如：上例中的isnumber函数
- 程序并不多处出现该函数调用，这样就使嵌入工作量相对较少，代码量也不会剧增

5.8 重载函数

函数重载（Function Overload）

- 函数重载：一组概念相同，处理对象（参数）不同的过程，出于方便编程的目的，用同一个函数名字来命名的技术称为函数重载。
- 参数默认：一个函数，既可以严谨和地道的调用，也可以省略参数，轻灵地调用，达到此种方便编程目的的技术称为参数默认。
- 重载与参数默认：它们都是通过参数的变化来分辨处理任务的不同．如果参数决定了不同的处理过程，则应重载，否则参数默认更简捷一些。

函数重载（Function Overload）

重载是不同的函数，以参数的类型，个数和顺序来分辨。

```
void print(double);
void print(int);

void func(){
  print(1);          // void print(int);
  print(1.0);        // void print(double);
  print('a');        // void print(int);
  print(3.1415f);    // void pirnt(double);
}
```

5.9 默认参数的函数

默认参数的函数

参数默认是通过不同参数来分辨一个函数调用中的行为差异。

```
void delay(int a = 2);    // 函数声明时

int main(){
  delay();                // 默认延迟 2 秒
  delay(2);               // 延迟 2 秒
  delay(5);               // 延迟 5 秒
}
void delay(int a){        // 函数定义时
  int sum=0;
  for(int i=1; i<=a; ++i)
  for(int j=1; j<3500; ++j)
  for(int k=1; k<100000; ++k) sum++;
}
```

小结

1、函数的分类

- ✓ 标准库函数：由C系统提供的函数；
- ✓ 用户自定义函数：由用户自己定义的函数；
- ✓ 有返回值的函数：向调用者返回函数值，应说明函数类型（即返回值的类型）；
- ✓ 无返回值的函数：不返回函数值，说明为空（void）类型；
- ✓ 有参函数：主调函数向被调函数传送数据；
- ✓ 无参函数：主调函数与被调函数间无数据传送；
- ✓ 内部函数：只能在本源文件中使用的函数；
- ✓ 外部函数：可在整个源程序中使用的函数。

2、函数定义的一般形式

[extern/static] 类型说明符 函数名（[形参列表]）{ 声明部分 执行部分 }

小结

3、函数说明的一般形式

[extern] 类型说明符 函数名（[形参列表]）；

4、函数调用的一般形式

函数名（[实参列表]）

5、函数的参数分为形参和实参两种，形参出现在函数定义中，实参出现在函数调用中，发生函数调用时，将把实参的值传送给形参。

6、函数的值是指函数的返回值，它是在函数中由**return**语句返回的。

7、C语言中，不允许函数嵌套定义，但允许函数的嵌套调用和函数的递归调用。

小结

8、程序中特别是在循环中反复执行的函数，用内联函数使嵌入的代码利用率较高。

9、函数重载：一组概念相同，处理对象（参数）不同的过程，出于方便编程的目的，用同一个函数名字来命名的技术。

10、参数默认是通过不同参数来分辨一个函数调用中的行为差异。