第6章 函数（1）

S.IST@XMU

复习回顾

- 一维数组与二维数组
  - 定义、引用、初始化
  - 小心越界！！！
- 字符数组
- 字符串
- 面对同一数组，不同的人有不同的心理感受，你呢？

一个 存放了一半数据 的数组：

- 悲观的程序员觉得数组是半空的；
- 乐观的程序员觉得数组是半满的；
- 新手程序员担心数组空间不够用；
- 专业程序员觉得数组占用空间太大，需要调用realloc()…

字符数组应用举例：字符串比较

- 输入3个字符串,要求找出其中最大者。
- 解题思路第一步：字符串的存放
  - 设一个二维的字符数组str,大小为3×10。每一行存放一个字符串

```
char str[3][10];
```

字符数组应用举例：字符串比较

- 输入3个字符串,要求找出其中最大者。
- 解题思路第二步：**字符串的输入**
  - 可以把str[0],str[1],str[2]看作3个一维字符数组，可以把它们如同一维字符数组那样进行处理

```
for (i=0;i<3;i++)
    gets (str[i]);
```

```
China
Japan
India
```

str[0]	C	h	i	n	a	\0	\0	\0	\0	\0
str[1]	J	a	p	a	n	\0	\0	\0	\0	\0
str[2]	I	n	d	i	a	\0	\0	\0	\0	\0

字符数组应用举例：字符串比较

- 输入3个字符串,要求找出其中最大者。
- 解题思路第三步：字符串的比较
  - 经过两次比较,就可得到值最大者,把它放在一维字符数组string中

```
if (strcmp(str[0],str[1])>0)
    strcpy(string,str[0]);
else
    strcpy(string,str[1]);
if (strcmp(str[2],string)>0)
    strcpy(string,str[2]);
```

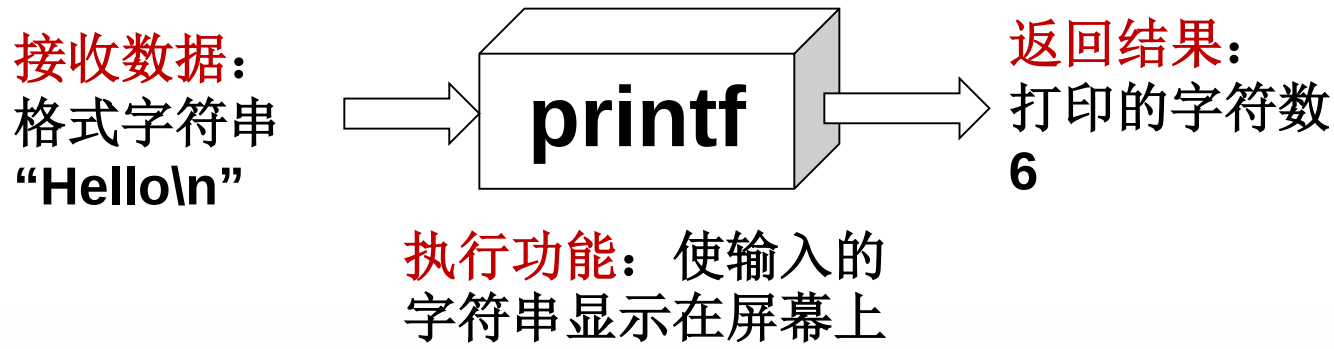
字符数组应用举例：字符串比较

```
1. #include<stdio.h>
2. #include<string.h>
3.
4. int main ( )
5. {
6.     char str[3][10]; char string[10]; int i;
7.     for (i=0;i<3;i++)
8.         gets (str[i]);
9.     if (strcmp(str[0],str[1])>0)
10.         strcpy(string,str[0]);
11.    else
12.         strcpy(string,str[1]);
13.    if (strcmp(str[2],string)>0)
14.         strcpy(string,str[2]);
15.    printf("\nthe largest:\n%s\n",string);
16.    return 0;
17.}
```

```
China
Japan
India

the largest:
Japan
```

函数是什么

- 函数是一个具有一定功能的“黑盒子”。
  - 对外只有两个接口，一个用来接收数据，另一个用来传回数据。
  - 我们只需要把数据送进黑盒子，就能得到传回的结果，至于内部具体如何工作的，我们可以不必关心。
  - 例如：int n; n=printf("Hello\n”);



只定义一个main函数行不行

```
#include <stdio.h>
#define PI 3.1415926
int main()
{   //主函数
    float radius, a;
    radius = 10.0;
    a = PI*radius*radius;

    printf("Area = %f\n",
   a);

    return 0;
}
```

- 逻辑上是可行的
- 实际上是不行的
- 别忘了，printf就是一个main函数之外的函数！

```
int __cdecl printf (
        const char *format,
        ...
        )
/*
 * stdout 'PRINT', 'F'ormatted
 */
{
        va_list arglist;
        int buffing;
        int retval;
        va_start(arglist, format);
        _ASSERTE(format != NULL);
        _lock_str2(1, stdout);
        buffing = _stbuf(stdout);
        retval = _output(stdout,format,arglist);
        _ftbuf(buffing, stdout);
        _unlock_str2(1, stdout);
        return(retval);
}
```

只有main函数有什么问题

- **代码的复用率：**我们无数次地使用printf函数输出，如果每次都从头把它的实现写一遍，你会崩溃吗？（即使会写的话）
- **程序的可读性：**看一段长达上万行的main函数就像读一篇不分章节的长篇小说一样，你觉得舒服吗？
- **排错的效率：**在上万行的main函数中找程序的错误无异于大海捞针，你会抓狂吗？

解决方法：结构化程序设计

- 结构化：用函数来组织程序。
  - 如果程序需要做许多事情，就应把它划分成几个函数，每个函数完成一件事情。
- 设想这样一个场景
  - 如果你正要编写一个C程序要实现从键盘上获取一个数字列表，并对其排序，然后打印到屏幕上，你会怎么做？

全写进main函数 vs 结构化

```
int main()
{
 /* 这不是一个可以完整的程序，只
   是框架 */
 .
 .
 /* 这部分代码读取数字列表 */
 .
 .
 /* 这部分代码对数字排序 */
 .
 .
 /* 这部分代码输出排完序的数字*/
 .
 .
 return 0;
}
```

```
int main()
{
  getNums();//调用第二个函数
  sortNums();//调用第三个函数
  printNums();//调用第四个函数
  return 0;
}
getNums() /* 第二个函数 */
{  ... // 实现读取数字列表
  return;
}
sortNums() /* 第三个函数 */
{  ... // 实现数字排序
  return;
}
printNums() /* 第四个函数 */
{  ... // 实现输出排过序的数字
  return;
}
```

关于结构化的结论

- 不要用main()来做所有的事情！
  - 经验：一个函数的代码行最好不超出一个屏幕！
  - 最好是为程序要完成的每项任务编写单独的函数；
  - main()除了调用其他函数之外不应该做太多事情；
  - 每个被main()调用的函数完成任务后返回main()中发生调用的位置；
  - main()继续执行直到程序结束，返回到操作系统。

函数的来源与好处

- 来源1：库函数，如scanf,printf等
- 来源2：自己编写的函数，如getNums，sortNums等
- 好处：
  - 明确问题本身，简化问题的求解。
  - 提高软件的复用率，如printf()的使用。
  - 提高程序的可读性，便于排错和维护。
  - 利于软件后期的升级维护。

一个很有用的库函数rand()

- 一段生成伪随机数的代码

```
1. #include <stdio.h>
2. #include <stdlib.h>                          //包含有rand和srand函数原型
3. #include <time.h>                            //包含有time函数原型

4. int main()
5. {
6.     int i;
7.     unsigned int seed = time(NULL);   //使用系统时间作为种子
8.     srand(seed);                              //调用种子函数初始化种子
9.     for (i=0; i<5; i++)
10.    {
11.        printf("%-8d", rand());          //产生伪随机数
12.    }
13.    printf("\n");
14.    return 0;
15.}
```

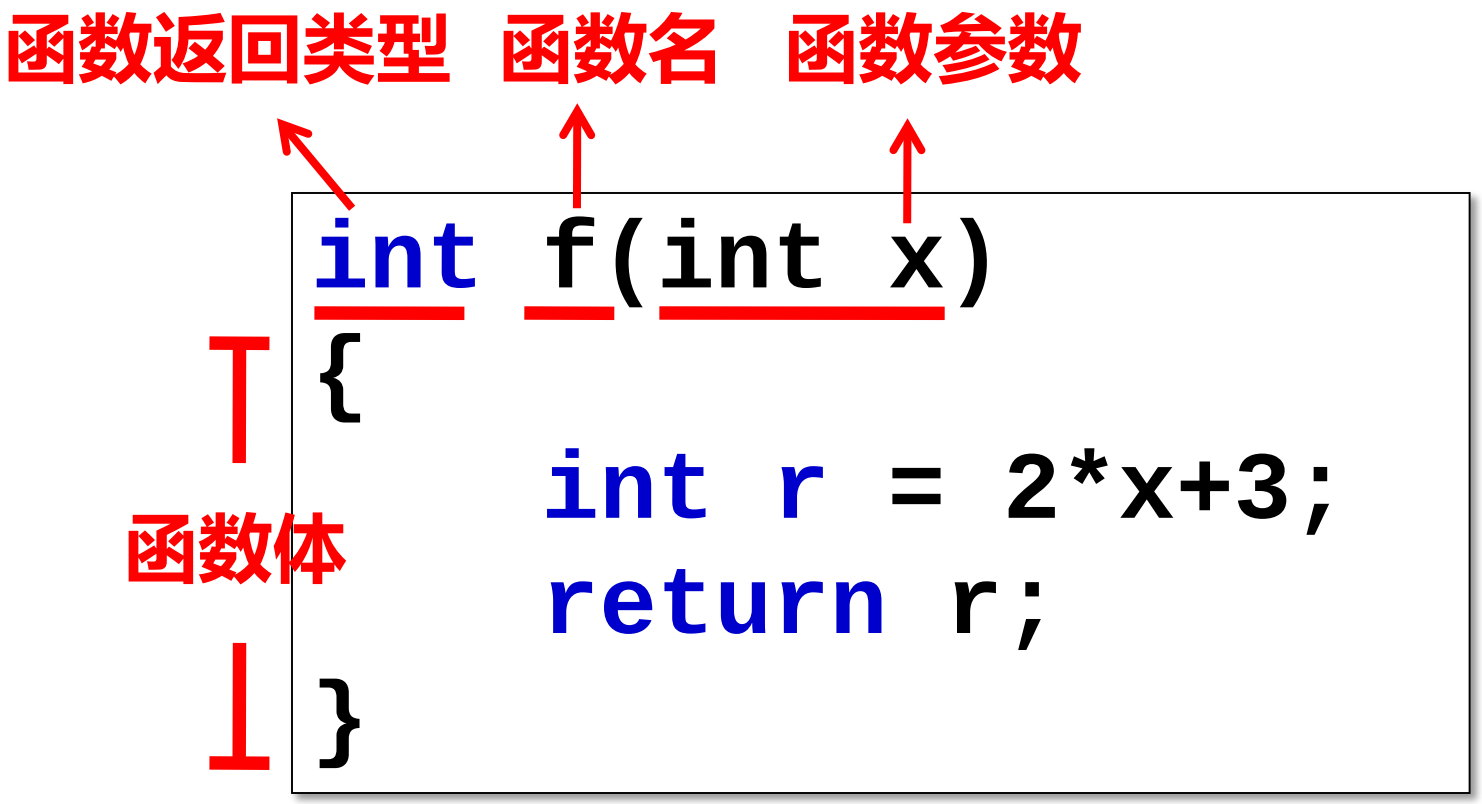
参考这段代码，
你可以写出随机点名程序吗？

通过数学函数理解C语言函数

- 在数学中，假设有函数y=f(x)，描述为y=2x+3，其中x和y均为整数。
- 对于C语言来说，就是定义以下函数

```
int f(int x)
{
    int r = 2*x+3;
    return r;
}
```

C语言函数的组成



为什么需要函数定义

- 若不知道函数的定义，怎么调用函数呢？
- 函数定义需要将以下信息通知编译系统
  - 函数参数的个数与类型（以便从调用者处接收数据）
  - 函数的名字（以便接受调用）
  - 函数实现的功能（以便完成特定的任务）
  - 函数的返回值类型（以便向调用者传回结果）

函数的定义

- 函数定义的一般形式：

```
类型标识符 函数名（形参列表）
{
        函数体;
}
```

- ◆形参列表，函数体和返回类型标识符都是**可选项**。
- ◆**类型标识符**表示的是函数返回值的类型，也就意味着函数将传回一个哪种类型的数值。
- ◆函数也**可以没有返回值**，此时就用void来修饰返回类型标识符。

没有返回值的函数定义举例

```
#include <stdio.h>

void greeting()
{
    printf("Good morning!\n");
}

int main()
{
    greeting();

    return 0;
}
```

函数分类：库函数与自定义函数

- 从用户使用的角度看，函数有两种
  - 库函数，它是由系统提供的，用户不必自己定义而直接使用它们。应该说明，不同的C语言编译系统提供的库函数的数量和功能会有一些不同，当然许多基本的函数是共同的。
  - 用户自己定义的函数。它是用户自己编写的，用以解决用户专门需要的函数。

函数分类：无参函数与有参函数

- 依据是否需要接收数据，函数分两类
  - **无参函数**。如getchar，无参函数一般用来执行指定的一组操作。无参函数可以带回或不带回函数值，但一般以不带回函数值的居多。
  - **有参函数**。如scanf，在调用函数时，主调函数在调用被调用函数时，通过参数向被调用函数传递数据，一般情况下，执行被调用函数时会得到一个函数值，供主调函数使用。

定义函数的方法：无参函数

```
void greeting()
{
    printf("Hello\n");
}
```

1.定义无参函数

定义无参函数的一般形式为:

```
类型名  函数名()
{
    函数体
}
```

```
类型名  函数名(void)
{
    函数体
}
```

包括声明部分和语句部分

定义函数的方法：无参函数

```
void greeting()
{
    printf("Hello\n");
}
```

1.定义无参函数

定义无参函数的一般形式为:

类型名 函数名()

{

函数体

}

指定函数值的类型

类型名 函数名(void)

{

函数体

}

定义函数的方法：有参函数

2.定义有参函数

定义有参函数的一般形式为:

类型名 函数名（类型标识符1 变量名1，类型标识符2 变量名2，...）

{

函数体

}

```
1. int add(int a, int b)
2. {
3.     int res;
4.     res = a + b;
5.     return res;
6. }
```

定义函数的方法：空函数

3. 定义空函数

定义空函数的一般形式为：

```
类型名 函数名（ ）
{
}
```

```
int add(int a, int b)
{
    return 0;
}
```

```
void idle()
{

}
```

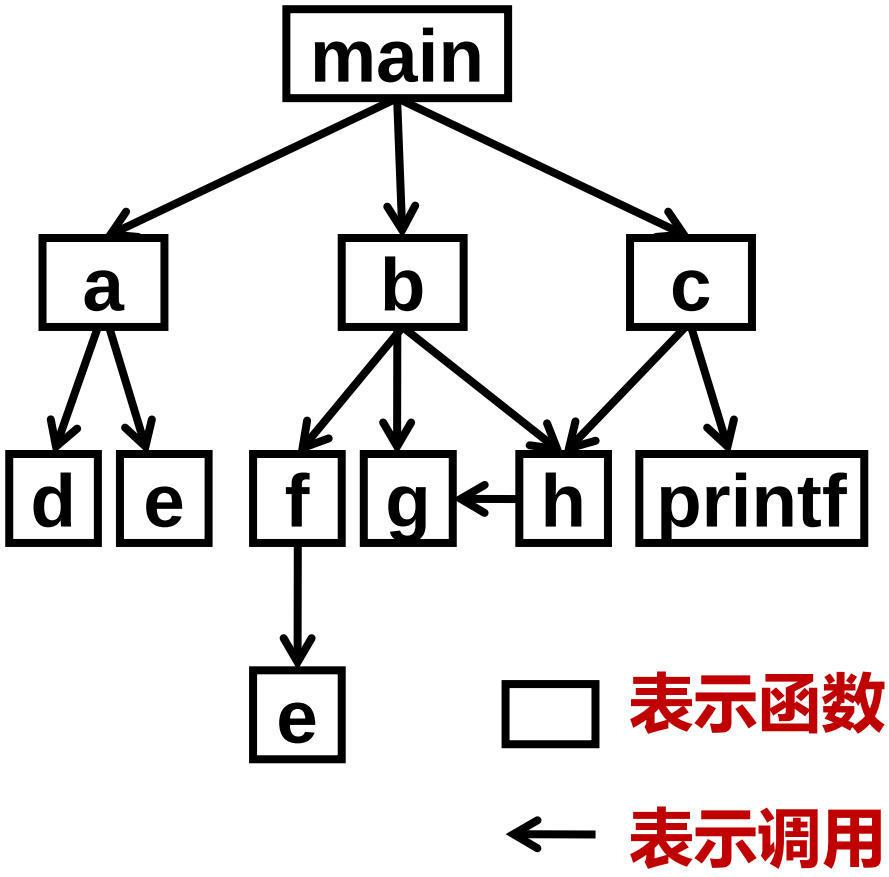
- **诀窍**：先用空函数占一个位置，以后逐步扩充
- **好处**：程序结构清楚，可读性好，以后扩充新功能方便，对程序结构影响不大

关于调用函数

- 我们需要知道：
  - 函数调用的形式
  - 函数调用时的数据传递
  - 函数调用的过程
  - 函数的返回值

函数之间的调用关系

- 主函数调用其他函数，其他函数也可以互相调用，但是最好**别调用主函数！**
  - **不妨试试会发生什么**
- 同一个函数可以被一个或多个函数调用任意多次



函数调用的一般形式

- 函数调用的一般形式为：

  函数名（实参表列）

- 如果是调用无参函数，则“实参表列”可以没有，但括号不能省略
- 如果实参表列包含多个实参，则各参数间用逗号隔开

函数调用形式一：调用语句

- 按函数调用在程序中出现的形式和位置来分，可以有以下3种函数调用方式：

1．函数调用语句

- 把函数调用单独作为一个语句

如 printf_star();

这时不要求函数传回值，只要求函数完成一定的操作

函数调用形式二：函数表达式

- 按函数调用在程序中出现的形式和位置来分，可以有以下3种函数调用方式：

2．函数表达式

- 函数调用出现在另一个表达式中

如 `c=max(a,b);`

这时要求函数传回一个确定的值以参加表达式的运算

函数调用形式三：函数参数

- 按函数调用在程序中出现的形式和位置来分，可以有以下3种函数调用方式：

3．**函数参数**

- 函数调用作为另一函数调用时的实参

如 `m = max(a,max(b,c));`

其中max(b,c)是一次函数调用，它的值作为max另一次调用的实际参数

函数调用时的数据传递

- **形式参数和实际参数**
  - ◆在调用有参函数时，主调函数和被调用函数之间有数据传递关系
  - ◆**定义函数时**函数名后面的变量名称为“形式参数”（简称“形参”）
  - ◆主调函数中**调用一个函数时**，函数名后面参数称为“实际参数”（简称“实参”）
    - 实际参数可以是常量、变量或表达式

形参和实参间的纠葛

- **形参和实参是彼此完全独立**的！哪怕看起来变量名称和类型完全一样！
- 在调用函数过程中，系统会把实参的值拷贝一份，传递给被调用函数的形参，或者说，**形参从实参得到一个值**
- **该值在函数调用期间有效**，可以参加被调函数中的运算

函数调用的例子：找大数

- 输入两个整数，要求输出其中值较大者。要求用函数来找到大数。
- 解题思路：设计函数定义
  - (1) **函数名**：应是见名知意，今定名为max
  - (2) **返回类型**：由于给定的两个数是整数，返回主调函数的值（即较大数）应该是整型
  - (3) **参数**：max函数应当有两个参数，以便从主函数接收两个整数，因此参数的类型应当是整型

函数调用的例子：找大数

先编写max函数：

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

函数调用的例子：找大数

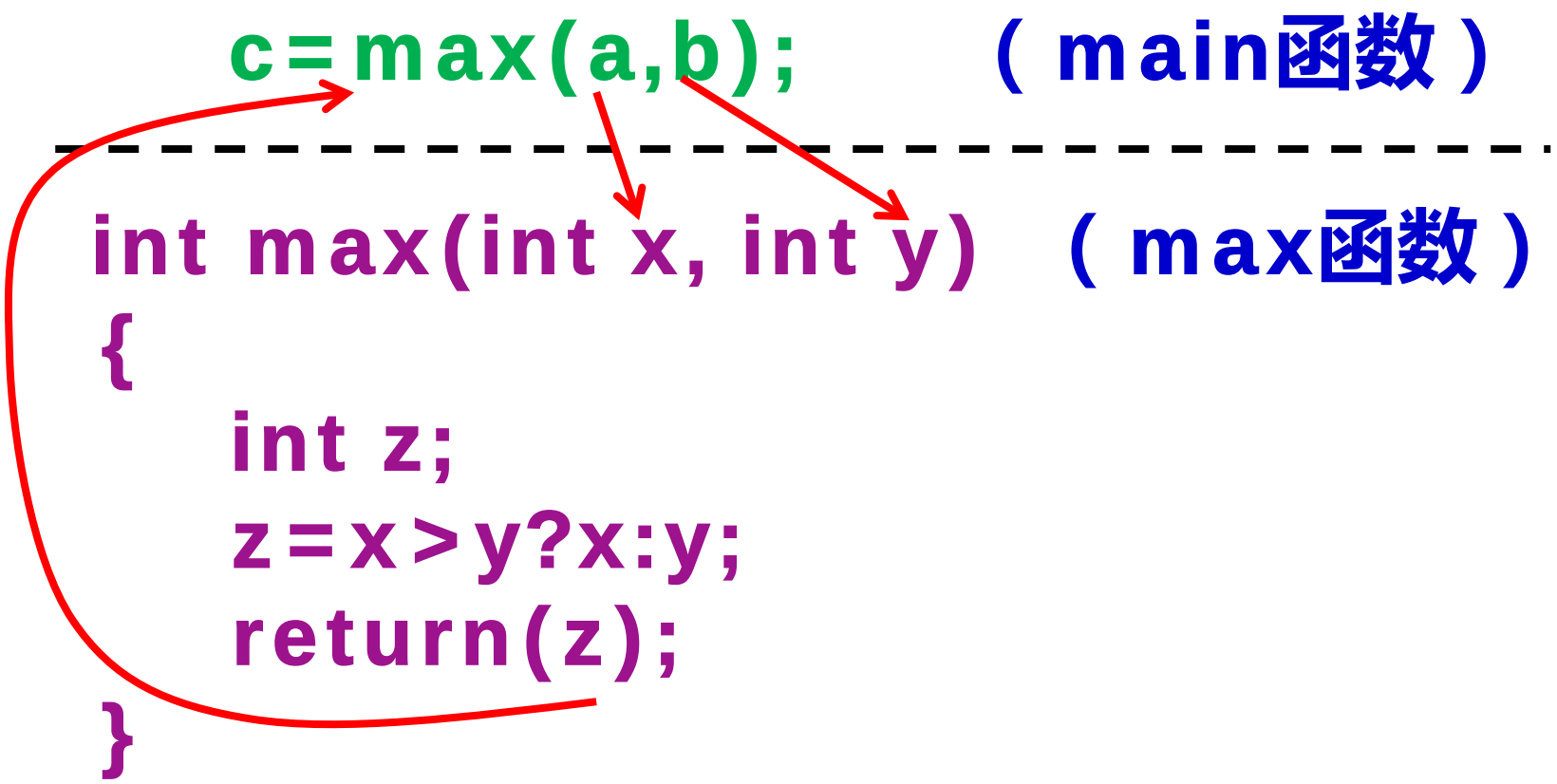
然后在max函数后面，再编写主函数

```
1. int main()
2. {
3.     int a,b,c;
4.     printf("two integer numbers: ");
5.     scanf("%d,%d",&a,&b);
6.     c = max(a,b);
7.     printf("max is %d\n",c);
8. }
```

```
two integer numbers:12,-34
max is 12
```

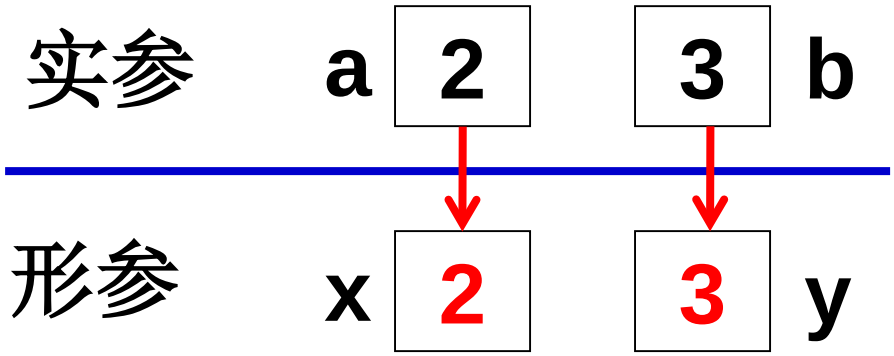
实参可以是常量、变量或表达式

数据传递的过程



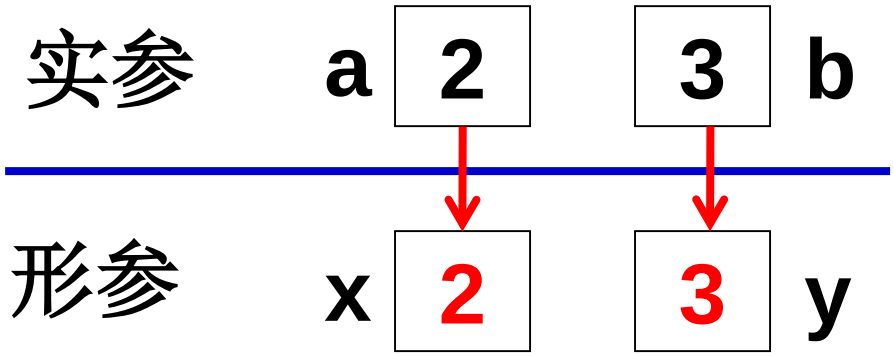
函数调用的过程：形参的变化

- 在定义函数中指定的形参，在**未出现函数调用时**，它们并不占内存中的存储单元。
- 在**发生函数调用时**，函数max的形参被临时分配内存单元。



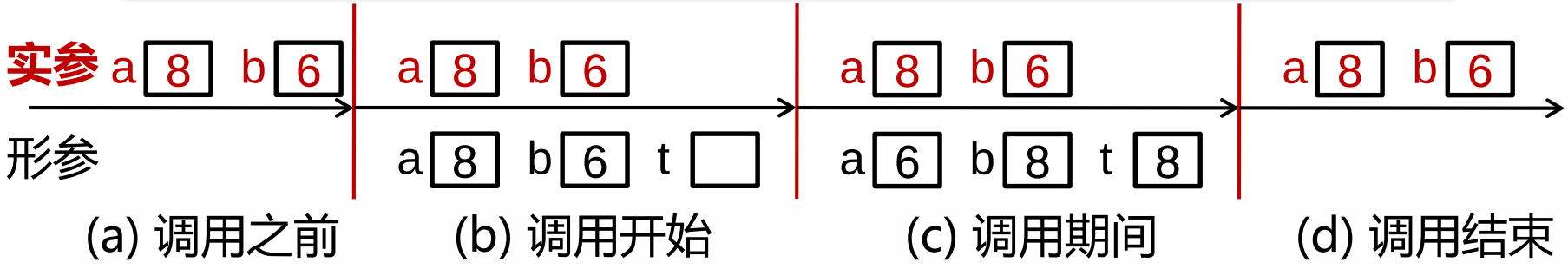
函数调用的过程：形参vs.实参

- **调用结束**，形参单元被释放，实参单元仍保留并维持原值，没有改变
- 如果在执行一个被调用函数时，**形参的值发生改变，不会改变**主调函数的**实参的值**



一个失败的函数

```
1.  #include  <stdio.h>
2.  void  sort(int a, int b)    // 定义sort函数, a , b 是形参
3.  {
4.      int t;              //定义在sort函数内使用的变量
5.      if (a>b)
6.      {
7.          t=a; a=b; b=t;              //交换形参 a , b 中的值
8.      }
9.  }               //函数调用完毕，释放形参a , b和变量t所占的存储单元
10. int main ( )
11. {
12.     int a=8, b=6 ;
13.     sort(a, b);  //调用函数sort , a 和 b 是实参
14.     printf("%d %d", a, b);
15. }               //程序运行结束，释放实参 a 和 b 所占的存储单元
```



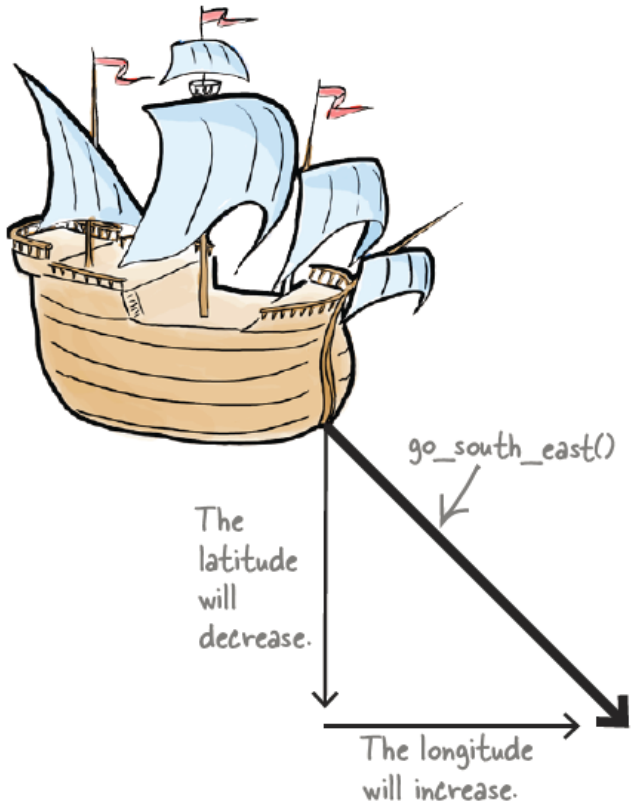
另一个失败的函数

```
#include <stdio.h>

void go_south_east(int lat, int lon)
{
   lat = lat - 1;
   lon = lon + 1;
}

int main()
{
   int latitude = 32;
   int longitude = -64;
   go_south_east(latitude, longitude);
   printf("Avast! Now at: [%i, %i]\n", latitude, longitude);
   return 0;
}
```

Pass in the latitude and longitude.

Decrease the latitude.

Increase the longitude.



失败的执行结果

```
#include <stdio.h>

void go_south_east(int lat, int lon)
{
   lat = lat - 1;
   lon = lon + 1;
}

int main()
{
   int latitude = 32;
   int longitude = -64;
   go_south_east(latitude, longitude);
   printf("Avast! Now at: [%i, %i]\n", latitude, longitude);
   return 0;
}
```

Pass in the latitude and longitude.

Decrease the latitude.

Increase the longitude.

WTF? The ship is still in the same place.

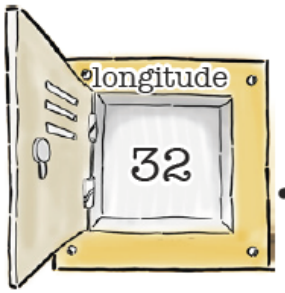
Where's The Fightin'?

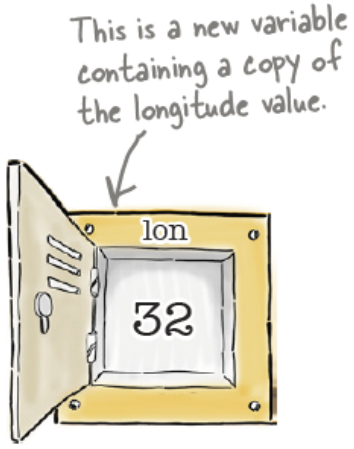

```
File Edit Window Help Savvy?
> gcc southeast.c -o southeast
> ./southeast
Avast! Now at: [32, -64]
>
```

老外解释函数失败的原因

The code broke because of the way that C calls functions.

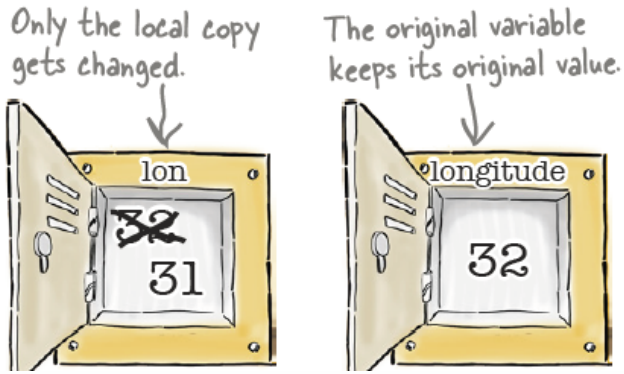
1. Initially, the `main()` function has a local variable called `longitude` that had value 32.





2. When the computer calls the `go_south_east()` function, it **copies the value** of the `longitude` variable to the `lon` argument. This is just an assignment from the `longitude` variable to the `lon` variable. When you call a function, you don't send the *variable* as an argument, just its *value*.

3. When the `go_south_east()` function changes the value of `lon`, the function is just changing its local copy. That means when the computer returns to the `main()` function, the `longitude` variable still has its original value of 32.



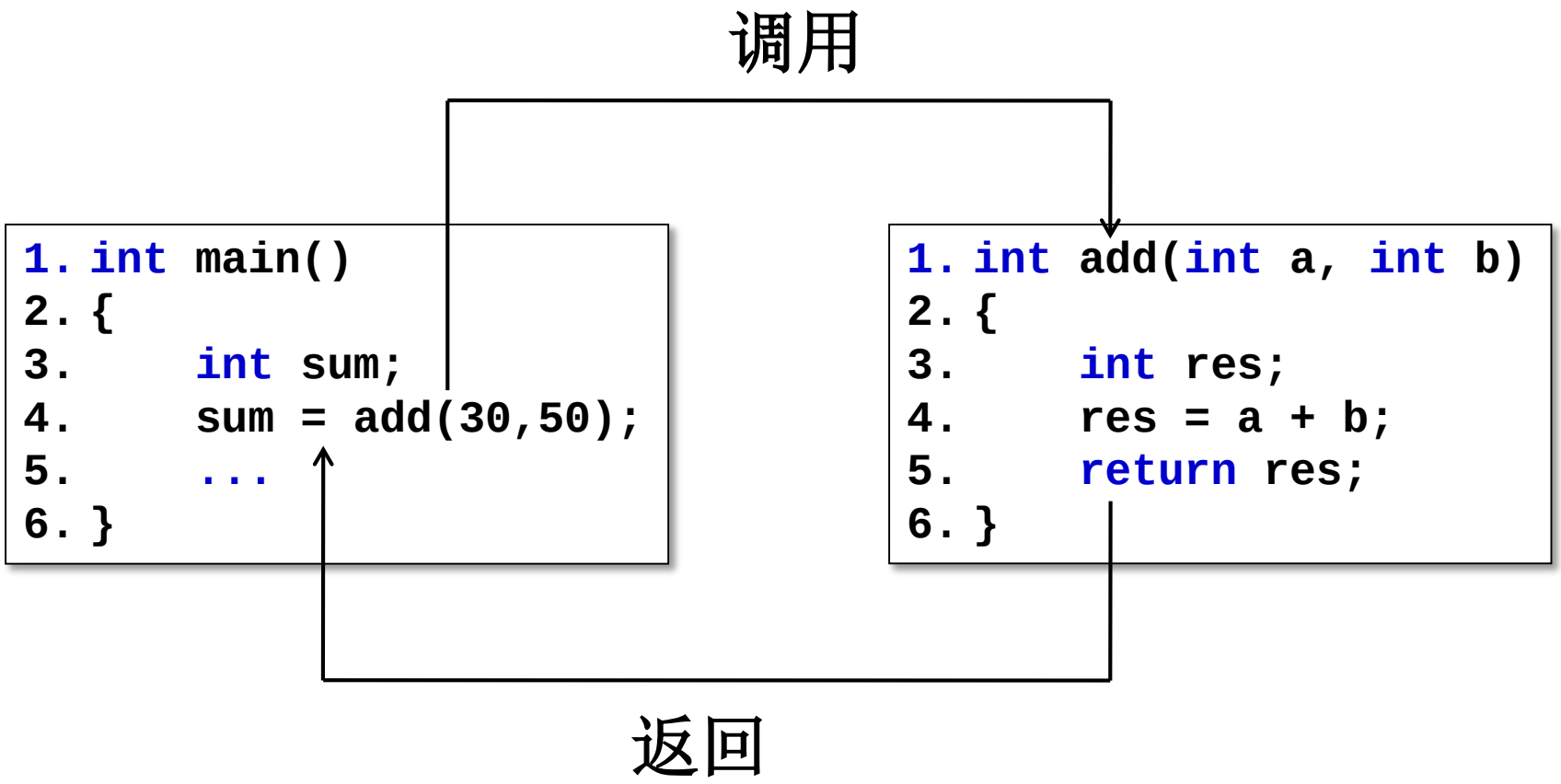
But if that's how C calls functions, how can you ever write a function that updates a variable?

编译环境决定实参的求值顺序

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

2. int f(int a, int b)
3. {
4.     return a;
5. }

6. int main()
7. {
8.     int i=1;
9.     printf("%d\n", f(i,++i));
10.}
```

- 有些编译器实参的求值顺序是从右到左（如VC），左侧的运行结果是 2；
- 有些编译器实参的求值顺序是从左到右，左侧运行结果是 1；
- 写程序必须规避二义性，永远不要写这种奇葩的语句！！！

函数的返回值


调用
1. int main()
2. {
3. int sum;
4. sum = add(30,50);
5. ...
6. }
1. int add(int a, int b)
2. {
3. int res;
4. res = a + b;
5. return res;
6. }
返回

return语句的使用原则(一)

1. 函数返回值是通过return返回的，该语句可将一个确定的值返回到主调函数

2. 若被调函数是无返回值的，那么函数体内可以有return语句也可以没有。但如果被调函数是有返回值的，return语句就必须有。

3. 一个函数中可以包含多个return语句，执行到第一条return时，函数将返回主调函数，此后的其他语句不会被继续执行。

return语句的使用原则(二)

4. 对于不包含return语句的函数，函数体内的语句将逐条执行，直到表示函数结尾的大括号处。

5. return后面可以是一个常量，或变量，或表达式，但**必须具有一个确定的值**。

6. 定义函数时，对函数类型的说明应该和return语句返回的值类型一致。**凡不加类型说明的函数，一律默认为返回值的类型为整数**。

若返回值类型和函数类型不同

- 情况一：并无影响

```
1. int func()
2. {
3.     return 'a';
4. }
5. int main()
6. { //ret的值会等于97
7.     int ret = func();
8.     ...
9. }
```

- 情况二：精度损失

```
1. int func()
2. {
3.     return 3.14;
4. }
5. int main()
6. { //ret的值会等于3.000000
7.     float ret = func();
8.     ...
9. }
```

- 情况三：编译出错

```
1. int func()
2. {
3.     return "0";
4. }
```

用实例检验函数返回值

```
1. #include <stdio.h>
2. int main()
3. {
4.     int max(float x,float y);
5.     float a,b;  int c;
6.     scanf("%f,%f,",&a,&b);
7.     c = max(a,b);
8.     printf("max is %d\n",c);
9.     return 0;
10.}
11.int max(float x, float y)
12.{
13.    float z;
14.    z=x>y?x:y;
15.    return(z) ;
16.}
```

变为2

1.5 2.6

2.6 ⇒ 2

```
1.5,2.6
max is 2
```

这个程序为什么是错误的？

```
#include <stdio.h>
#define PI 3.1415926
int main()
{   //主函数
    float radius, a;
    radius = 10.0;
    a=circle_area(radius);//调用函数1
    show_area(a);       //调用函数2
    return 0;
}
float circle_area(float r)
{   // 函数1：计算圆面积的函数
    float tmp = PI*r*r;
    return tmp;
}
void show_area(float a)
{   // 函数2：输出圆面积的函数
    printf("Area = %f\n", a);
}
```

对被调用函数的声明

- 在一个函数中调用另一个函数需要具备如下条件：
  - (1) 被调用函数必须是已经定义的函数（是库函数或用户自己定义的函数）
  - (2) 如果使用自己定义的函数，而该函数的位置在调用它的函数后面，应该声明
  - (3) 如果使用库函数，应该在本文件开头加相应的#include指令

```
1.  #include <stdio.h>
2.  #define PI 3.1415926
3.  float circle_area(float r)
4.  {    // 计算圆面积的函数
5.       float tmp = PI*r*r;
6.       return tmp;
7.  }
8.  void show_area(float r);
9.  int main()
10. {    //主函数
11.      float radius, a;
12.      radius = 10.0;
13.      a=circle_area(radius);//调用函数1
14.      show_area(a);   //调用函数2
15.      return 0;
16. }
17. void show_area(float a)
18. {    // 输出圆面积的函数
19.      printf("Area = %f\n", a);
20. }
```

作业 2017/11/17

- **按下列要求编写程序，提交手写源代码**

1. 输入不超过200字符的一段话，输入一个单词，统计这段文本中这个单词出现的次数（区分大小写）。样例输入：粉色部分的话和单词If，样例输出：The word "If" appears 1 time in the text.（注意，如果出现的次数多于一次，上面的输出中time应该为复数形式，即times）

If you give someone a program, you will frustrate them for a day;
if you teach them how to program, you will frustrate them for a lifetime!

2. 利用字符数组，允许输入两个100位以内的整数，并输出其加、减运算结果。

- **上机练习（不用交）：本讲义例程，教材第六章编程题**