函数入门：

在C语言的编程中，其实用到很多的函数，而每一个函数就可以理解为一个独立的模块， 因此C语言也称为模块化编程。

我们封装函数应该尽可能左到：低耦合，高内聚

在软件设计中通常用耦合度和内聚度作为衡量模块独立程度的标准。划分模块的一个准则是高内聚低耦合。从模块粒度来看，高内聚：尽可能类的每个成员方法只完成一件事（最大限度的聚合）； 低耦合：减少类内部，一个成员方法调用另一个成员方法。从类角度来看， 高内聚低耦合：减少类内部，对其他类的调用；从功能块来看 高内聚低耦合：减少模块之间的交互复杂度（接口数量，参数数据）即横向：类与类之间、模块与模块之间；纵向：层次之间；尽可能，内容内聚，数据耦合。

降低耦合度的方法

1、少使用类的继承，多用接口隐藏实现的细节。 Java面向对象编程引入接口除了支持多态外， 隐藏实现细节也是其中一个目的。

2、模块的功能化分尽可能的单一，道理也很简单，功能单一的模块供其它模块调用的机会就少。（其实这是高内聚的一种说法，高内聚低耦合一般同时出现）。

3、遵循一个定义只在一个地方出现。

4、少使用全局变量。

5、类属性和方法的声明少用public，多用private关键字。

6、多用设计模式，比如采用MVC的设计模式就可以降低界面与业务逻辑的耦合度。

7、尽量不用"硬编码"的方式写程序，同时也尽量避免直接用SQL语句操作数据库。

8、最后当然就是避免直接操作或调用其它模块或类（内容耦合）；如果模块间必须存在耦合，原则上尽量使用数据耦合，少用控制耦合，限制公共耦合的范围，避免使用内容耦合。

增强内聚度方法

1、模块只对外暴露最小限度的接口，形成最低的依赖关系。

2、只要对外接口不变，模块内部的修改，就不得影响其他模块。

3、删除一个模块，应当只影响有依赖关系的其他模块，而不应该影响其他无关部分。

对于函数的使用者来说，应该尽可能简单地去使用该函数接口，使用者只管往函数中输入需要的数据， 通过该函数获得一个结果即可。

例如使用美图软件进行修图。你只需要输入美图的级别1-10 级， 该软件则会输出一个美化之后的图像给你。

函数的定义：

函数头： 函数对外公开的接口信息。 比如：void *calloc(size_t nmemb, size_t size);

- 函数的返回值， 该函数运行结束后会返回什么东西给你， 比如： void *
- 函数名， 命名规则跟变量一致。应该尽量顾名思义。比如： calloc
- 参数列表，告诉用户该函数需要输入的数据以及类型，有可能有多个也可能没有， 比如(size_t nmemb, size_t size)

语法：

```
返回值类型 函数名 ( 参数1 , 参数2 ,参数3 , ... , 参数N )
{
 // 函数体

 return 返回值 ;
}
```

示例：

```
#include <stdio.h>

// 函数声明
int add ( int , int ); // 函数声明中可以把形参的名字省略


int main(int argc, char const *argv[])
{
    int x = 100 ;
     int y = 250 ;

     int tmp = 0 ;

     // 调用add函数， 把 x 和 y 的值 传递过去，
     //  x , y 是实参， 作为形参 a , b 初始值
     //   a = x  , b = y ;
     // 使用 tmp 来接受 add函数的返回值
     tmp = add(x,y);

     printf("x + y = %d \n" , tmp );

     return 0;
```

```
}


// 设计一个函数 ，接收 两个整型参数， 并返回 它们的和
int    add ( int a , int b ) // a , b 属于函数add 的局部变量
{
    int tmp = a + b ;

    return tmp ;
}
```

练习：

使用函数来实现求 3个整型中最大的并返回。

使用函数来实现交换两个变量的值（指针）。

总结：

- 当函数的返回值类型为 void 的时候， 表示该函数不会有返回的值
- 当参数列表中为void的时候， 表示该函数不需要参数
- 当返回值为具体的类型的时候 ， 函数的结尾就应该返回一个对应类型的数据
- return 只能够携带一个数据进行返回
- 当返回值类型为 void 的时候，可以不写return,如果写的话一般就直接写 return ;

形参与实参：

概念：

函数调用的时候传递的值，称为实参 max(123.034 , 'C' , "87") ;

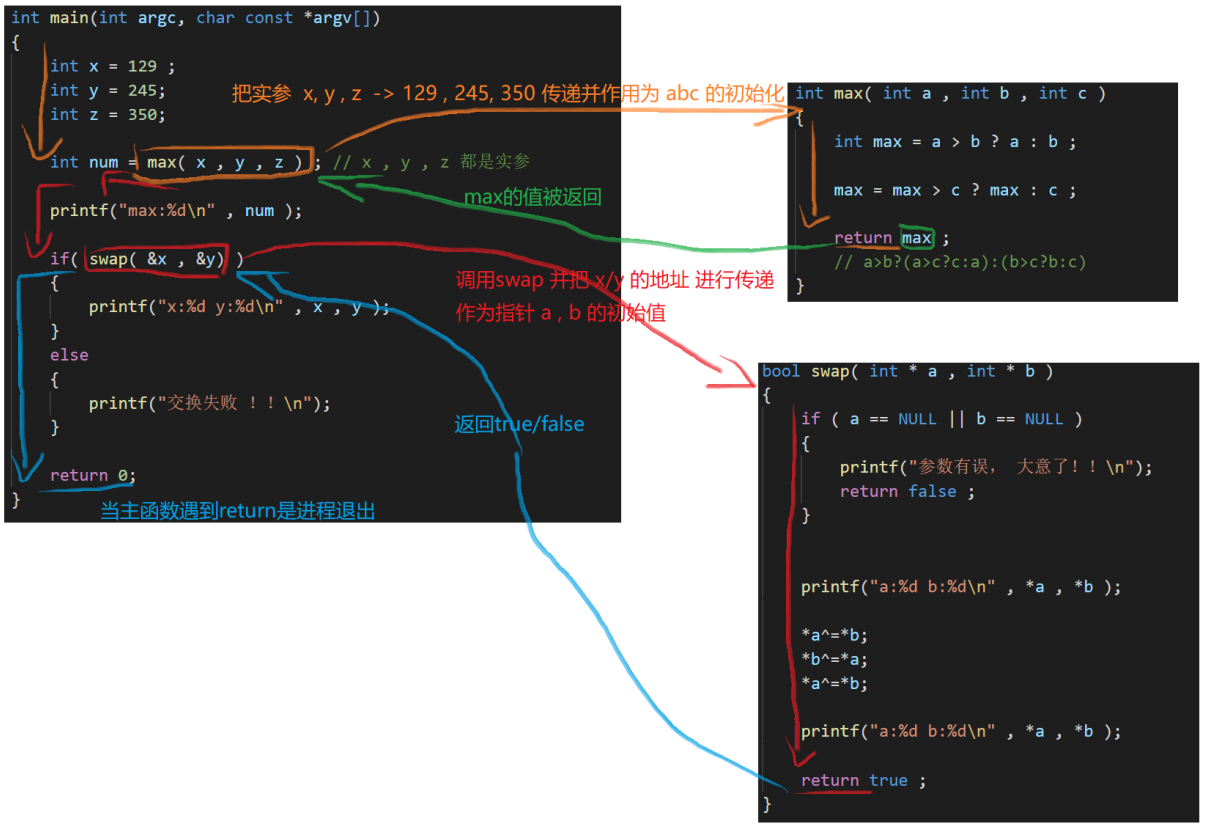
函数定义当中出现的参数列表，称为形参 max(float a , char b , char * c)

实参与形参的关系：

- 实参与形参应该是一一对应的。(顺序+类型)
- 形参的值是由实参进行初始化。
- 形参与实参是处于两个完全不相关的栈空间中。彼此是独立的

函数调用的过程：

函数调用的时候，进程会进行上下文切换，当指向完被调用的函数后,将会切换会被调用的语句，继续往下执行。



局部变量与栈内存：

局部变量：　　被函数体的一对大括号所包含的变量，称为局部变量（在函数体内部定义）

局部变量的特点：

- 属于函数内部的变量， 所存储的位置是该函数所拥有的栈空间
- 局部变量不会被其它函数所访问，因此不同的函数内部可以拥有完全一样的两个变量名字， 但是从内存来看它们是完全独立的。
- 当函数退出的时候， 局部变量所占用的内存，将会被系统回收，因此局部变量也称为临时变量。
- 形参当中的变量，虽然没有在大括号的范围内， 但是也是属于该函数的局部变量。

栈内存的特点：

- 当有一个函数被调用的时候，栈内存将会增长一段， 用来存放该函数的局部变量。
- 当函数退出时，他所占用的栈内存将会被释放回收。
- 系统分配栈空间遵循从上往下增长的原则。
- 栈空间的内存相对来说比较少， 不建议用来存放大量的数据。

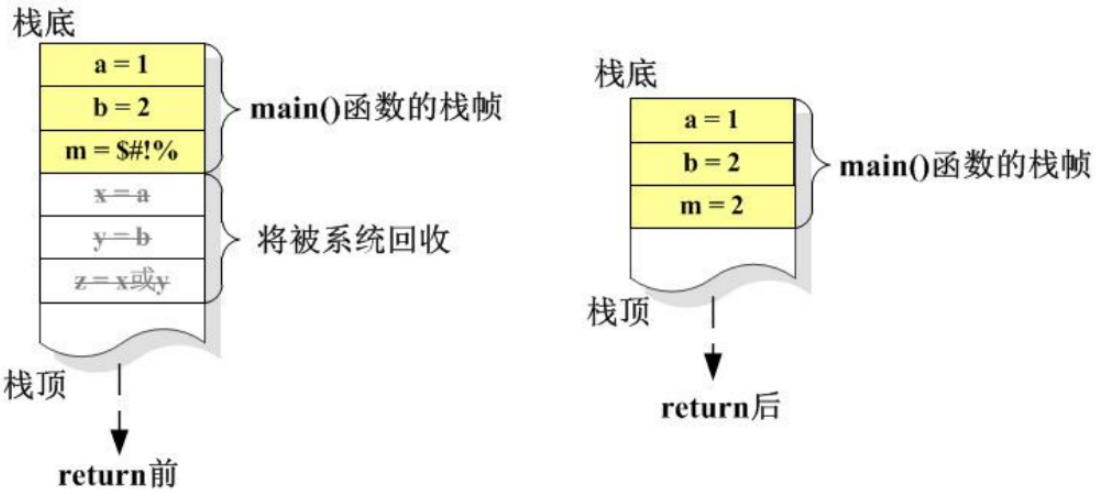

栈底
a = 1
b = 2
m = $#!%
x = a
y = b
z = x或y
main()函数的栈帧
将被系统回收
栈顶
return前
栈底
a = 1
b = 2
m = 2
main()函数的栈帧
栈顶
return后

图 2-21 函数的返回