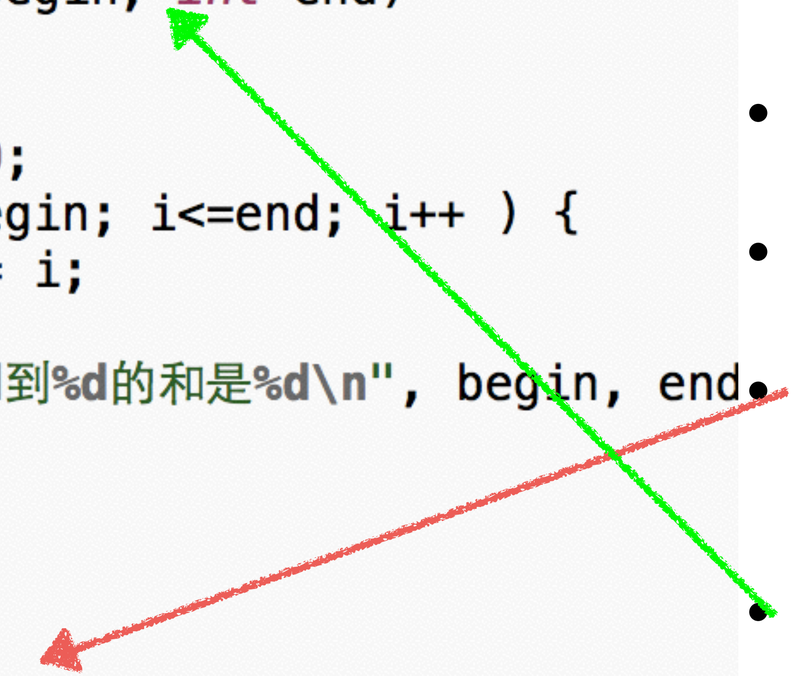
函数原型

函数先后关系

```c
void sum(int begin, int end)
{
    int i;
    int sum =0;
    for ( i=begin; i<=end; i++ ) {
        sum += i;
    }
    printf("%d到%d的和是%d\n", begin, end, sum);
}

int main()
{
    sum(1,10);
    sum(20,30);
    sum(35,45);

    return 0;
}
```



- 像这样把sum()写在上面，是因为：
- C的编译器自上而下顺序分析你的代码
- 在看到sum(1,10)的时候，它需要知道sum()的样子
- 也就是sum()要几个参数，每个参数的类型如何，返回什么类型
- 这样它才能检查你对sum()的调用是否正确

如果不知道

- 也就是把要调用的函数放到下面了
- 旧标准会假设你所调用的函数所有的参数都是int，返回也是int
- 如果恰好不对...

```
    int a,b,c;
    a = 5;
    b = 6;
    c = max(10,12);
    printf("%d\n", c);
    max(12,13);

    return 0;
}

double max(double a, double b)
{
```

```
/Users/wengkai/cc/7.4.c:15:8: error: conflicting types for 'max'
double max(double a, double b)
       ^
```

函数原型

- 函数头，以分号“;”结尾，就构成了函数的原型
- 函数原型的目的是告诉编译器这个函数长什么样
  - 名称
  - 参数（数量及类型）
  - 返回类型
- 旧标准习惯把函数原型写在调用它的函数里面
- 现在一般写在调用它的函数前面
- 原型里可以不写参数的名字，但是一般仍然写上

函数原型

```
double max(double a, double b);

int main()
{
    int a,b,c;
    a = 5;
    b = 6;
    c = max(10,12);
    printf("%d\n", c);
    max(12,13);

    return 0;
}

double max(double a, double b)
```

根据原型判断

实际的函数头

参数传递

调用函数

- 如果函数有参数，调用函数时必须传递给它数量、类型正确的值
- 可以传递给函数的值是表达式的结果，这包括：
  - 字面量
  - 变量
  - 函数的返回值
  - 计算的结果

```
int a,b,c;
a = 5;
b = 6;
c = max(10,12);
c = max(a,b);
c = max(c, 23);
c = max(max(23,45), a);
c = max(23+45, b);
```

类型不匹配?

- 调用函数时给的值与参数的类型不匹配是C语言传统上最大的漏洞
- 编译器总是悄悄替你把类型转换好，但是这很可能不是你所期望的
- 后续的语言，C++/Java在这方面很严格

传过去的是什么?

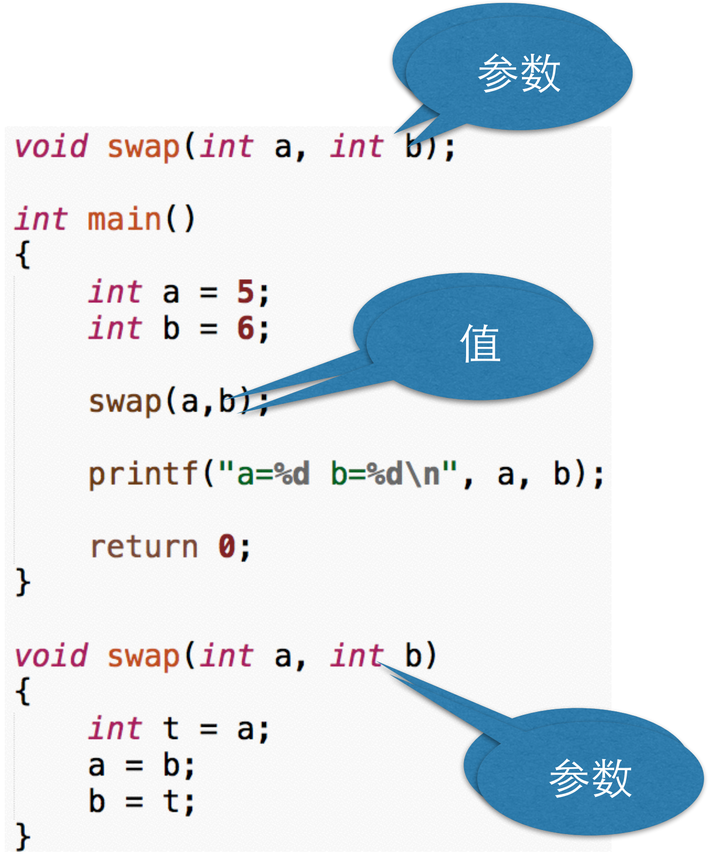
```c
void swap(int a, int b);

int main()
{
    int a = 5;
    int b = 6;

    swap(a,b);

    printf("a=%d b=%d\n", a, b);

    return 0;
}

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

C语言在调用函数时，永远只能传值给函数

- 这样的代码能交换a和b的值吗?

传值

- 每个函数有自己的变量空间，参数也位于这个独立的空间中，和其他函数没有关系
- 过去，对于函数参数表中的参数，叫做“形式参数”，调用函数时给的值，叫做“实际参数”
- 由于容易让初学者误会实际参数就是实际在函数中进行计算的参数，误会调用函数的时候把变量而不是值传进去了，所以我们不建议继续用这种古老的方式来称呼它们
- 我们认为，它们是参数和值的关系




```
void swap(int a, int b);

int main()
{
    int a = 5;
    int b = 6;

    swap(a,b);

    printf("a=%d b=%d\n", a, b);

    return 0;
}

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

本地变量

本地变量

- 函数的每次运行，就产生了一个独立的变量空间，在这个空间中的变量，是函数的这次运行所独有的，称作本地变量
- 定义在函数内部的变量就是本地变量
- 参数也是本地变量

变量的生存期和作用域

- 生存期：什么时候这个变量开始出现了，到什么时候它消亡了
- 作用域：在（代码的）什么范围内可以访问这个变量（这个变量可以起作用）
- 对于本地变量，这两个问题的答案是统一的：大括号内——块

本地变量的规则

- 本地变量是定义在块内的
  - 它可以是定义在函数的块内
  - 也可以定义在语句的块内
  - 甚至可以随便拉一对大括号来定义变量
- 程序运行进入这个块之前，其中的变量不存在，离开这个块，其中的变量就消失了
- 块外面定义的变量在里面仍然有效
- 块里面定义了和外面同名的变量则掩盖了外面的
- 不能在一个块内定义同名的变量
- 本地变量不会被默认初始化
- 参数在进入函数的时候被初始化了

其他细节

没有参数时

- void f(void);
- 还是
- void f();
  - 在传统C中，它表示f函数的参数表未知，并不表示没有参数

逗号运算符?

- 调用函数时的逗号和逗号运算符怎么区分?
- 调用函数时的圆括号里的逗号是标点符号，不是运算符
  - f(a,b)
  - f((a,b))

函数里的函数?

- C语言不允许函数嵌套定义

这是什么?

- int i,j,sum(int a, int b);
- return (i);

关于main

- int main()也是一个函数
- 要不要写成int main(void)?
- return的0有人看吗?
  - Windows：if errorlevel 1 …
  - Unix Bash：echo $?
  - Csh：echo $status