第9章

函数

函数概述

- **main(),printf(),getchar()**
- 函数：具有某种功能的独立程序段，C程序构建块。
  - 一连串组合在一起并且命名的语句。
  - 本质上：一个自带声明和语句的小程序。
- 函数的优点：
  - 程序分解：划分成小块，这样便于人们理解和修改程序。
  - 代码重用：避免重复编写可多次使用的代码。
  - 代码复用：一个函数能用于多个不同程序中。

函数概述

单函数版

```
int main()
{
    tmt=2; egg=2; salt=5g; soy=5ml;
    con=10g; oil=50ml;//变量声明
    clean tmt; //洗净番茄
    flay tmt;//去皮
    cut tmt;//切块；
    cut con;//葱花切片
    egg=egg+salt;//鸡蛋加少许盐
    mix egg;//打散
    hot oil;//烧热油;
    fry egg;
    ……
    ……
    return();//盛出；
}
```

流水账

多函数版

```
int main()
{
    tmt=2; egg=2; salt=5g; soy=5ml; con=10g;
    oil=50ml;//变量声明
    deal_tmt(tmt); //处理番茄
    deal_egg(egg);//处理鸡蛋
    fry_egg(egg);
    fry_tmt(tmt);
    mix_fry(egg, tmt);
    return();//盛出；
}
deal_tmt(tmt)
{
    clean(tmt);
    flay(tmt);
    cut(tmt);
    return tmt;
}
……
```

分门别类

函数定义与调用

- 定义：对任务的说明，目标、名称、所需材料、具体步骤方法
- 调用：任务执行，分派
- Eg,班级展示材料制作
  - 定义：

    PPT文档 班级展示(同学、班级活动信息，照片)

    {

    文字撰写;

    图片展示;

    }
  - 调用：

    班级展示(扶老奶奶过马路N次，……)；

程序：计算平均值

- 函数average计算两个double类型数值的平均值：

函数定义：

返回类型

函数名

形式参数：
对什么样的数求均值

```
double average(double a, double b)
{
    return (a+b)/2;
}
```

函数体：
具体怎么做

返回给函数调

数学函数f(a,b)=(a+b)/2

程序：计算平均值

- 函数调用：函数名(实际参数)；
  - double x=11.3,y=21.7,z;
  - z = average(x, y);
- x,y：实际参数：
  - 用来给函数提供信息（原料）
  - x和y值复制给形式参数a和b。
  - f(a, b)=(a+b)/2;求f(7,9)
    - a,b：形式参数;7,9：实际参数
    - 求值时实参7,9代入形参a,b;“代入”实质赋值

程序：计算平均值

- 实际参数不一定要是变量；任何正确类型的表达式都可以。
  - `average(5.1, 8.9)`
  - `average(x/2, y/3)`
- **average**函数调用，当做**double**型变量使用
- 打印**x**和**y**平均值的语句为：

```
printf("Average: %g\n",
average(x, y));
```

程序：计算平均值

- 程序**average.c**读取3个数并且计算两两均值

```
Enter three numbers: 3.5 9.6 10.2
Average of 3.5 and 9.6: 6.55
Average of 9.6 and 10.2: 9.9
Average of 3.5 and 10.2: 6.85
```

- 思路：
  - 三个数两两作为参数调用函数 **average(a,b);**

```
#include <stdio.h>
double average(double a, double b)
{
  return (a + b) / 2;
}//函数定义

int main(void)
{
   double x, y, z;

   printf("Enter three numbers: ");
   scanf("%lf%lf%lf", &x, &y, &z);
   printf("Average of %g and %g: %g\n", x, y,
  average(x, y));
   printf("Average of %g and %g: %g\n", y, z,
  average(y, z));
   printf("Average of %g and %g: %g\n", x, z,
  average(x, z));

   return 0;
}
```

程序：显示倒数计数

- 不带返回值的函数——返回类型

函数声明：

```
void print_count(int n)
{
        int i;
        for (i = n; i > 0; --i)
        {
                printf("T minus %d and counting\n", n);
        }
}
```

调用：

```
print_count(10);
```

程序：显示双关语(改进版)

- 无返回值，无参数，
- 格式：**void 函数名（void）**

```
void print_pun(void)
{
    printf("To C, or not to
C: that is the
question.\n");
}
```

- 调用：函数名()

```
print_pun();
```

函数定义

- 函数定义的一般格式：

```
返回类型  函数名(形式参数)
{//函数体
    声明//可选
    语句
}
```

函数定义——返回类型

- 函数返回值的类型
- 规则：
  - 可返回除数组外的其他类型（只能返回一个值）
  - 不返回值：**void**
  - 省略：
    - C89默认int型
    - C99非法

函数定义——形式参数

- 每个形参都需要说明类型；
- 用逗号进行分隔。
- eg，
  - `double average(double a, double b)`
  - `int max(int a, int b)`
- 无参：写明void。

函数定义——函数体

- 函数体可以包含声明和语句。
- **average函数：**

```
double average(double a,double b)
{
    double sum; /*declaration*/
    sum = a + b;  /*statement */
    return sum / 2;/*statement*/
}
```

函数定义——内部变量

```
double average(double a, double b){
        double sum;   /*内部变量*/
        sum = a + b;
        return sum / 2;
}

int main(void){
      double x, y, z;//内部变量
      scanf("%lf%lf%f",&x, &y, &z);
      z = average(x,y);
      z=(a+b)/2;//?  ?
      printf("average is %lf\n", sum/2);//?  ?
}
```

函数定义——空函数体

- 返
体为空：

```
vo
{
}
```

- 在
一种临时
措
意义的。
- e

```
int main(void)
{
        prepare();
        deal_egg();
        deal_tmt();
        mix_fry();
        return dish;
}
void prepare()
{
}
void deal_egg(void)
{
}
void deal_egg(void)
{
}
……
```

函数调用

- 函数名(实参列表)：

```
average(x, y)
print_count(i)
print_pun()
```

- （非void ）函数调用返回值当变量使用：
  - 可存于变量，可测试、显示或者其他用途（表达式）：

```
avg = average(x, y);
if (average(x, y) > 0)
     printf("均值为正\n");
printf("均值为 %g\n", average(x, y));
```

函数调用

- 如果不关心返回值，可丢弃非void型函数返回值：

```
average(x, y);//只调用函数，不使用返回值
```

  - printf返回显示的字符的个数。

```
num_chars = printf("Hi, Mom!\n");
printf("Hi, Mom!\n");
```

判定素数——函数版

- 检查一个数是否是素数：

```
Enter a number: 24
Not prime
```

要判断的数：参数

返回值：类型？

- 方法：n逐个除以2到n的平方根，都不能除净则n是素数。

函数体

```
for (d = 2; d < =sqrt(n); d++)
  if (n % d == 0) break;//合数
if (d <= sqrt(n))
  printf("%d is divisible by %d\n", n, d);
else
  printf("%d is prime\n", n);
```

函数定义

```
bool is_prime(int n)
{
   int div;
   if (n <= 1) return false;//0或负数

   for(div= 2; div * div <= n; div++)
     if (n % div == 0) return false;
           //循环非常规结束，提前返回

   return true;//循环常规结束返回
}
```

函数调用

```
int main(void)
{
   int n;

   printf("Enter a number: ");
   scanf("%d", &n);
   if (is_prime(n))
            //等价于is_prime(n)==true
     printf("Prime\n");
   else
     printf("Not prime\n");
   return 0;
}
```

程序练习——标准体重计算函数版

```
#include <stdio.h>
#define FACTOR 0.9f
int main(void){
        int height;
        float weight, stdwt;
        char sex;
        puts("输入性别, 男性用m表示，其它字符表女性.");
        scanf("%c", &sex);
        puts("输入身高(cm).");
        scanf("%d", &height);

        if (sex == 'm')
                stdwt = (height - 100) * FACTOR;
        else
                stdwt = (height - 100) * FACTOR -2.5;
        printf("你的标准体重应是%.1fkg\n.", stdwt);
        return 0;
}
```

程序练习——标准体重计算函数版

```
#include <stdio.h>
#define FACTOR 0.9f
float stdwt_count(性别, 身高){
        变量声明;
        分男女计算;
        返回结果;}
int main(void){
        变量声明;
        提示输入性别、身高
        输入性别、身高
        调用函数stdwt_count计算
        显示结果;
        return 0;
}
```

程序练习——标准体重计算函数版

```
#include <stdio.h>
#define FACTOR 0.9f
float stdwt_count(char sex, int height){
        float stdwt;
        if (sex == 'm')     stdwt = (height - 100) * FACTOR;
        else       stdwt = (height - 100) * FACTOR -2.5;
        return stdwt;}
int main(void){
        int h;
        char sx;
        puts("输入性别, 男性用m表示，其它字符表女性.");
        scanf("%c", &sx);
        puts("输入身高(cm).");
        scanf("%d", &height);
        printf("你的标准体重应是%.1fkg\n.", stdwt_count(sx, h));
        return 0;
}
```

温故而知新——函数

- 函数：具有某种功能的独立程序段。
  - 一连串组合在一起并且命名的语句。
  - 本质上：一个自带[声明]和语句的小程序。
- 函数定义的一般格式：//要完成功能的详细说明

```
返回类型 函数名( 形式参数 )
{//函数体
   声明//可选
   语句
}
```

- 可选项
  - 返回值 void
  - 形式参数 void
  - 声明
  - 语句
- 调用：任务执行
  - 函数名(实际参数)，实参代入形参

函数声明VS函数定义

- 声明：说明
- 定义：说明及分配资源
- 变量声明、定义等同、统一
- 函数声明、定义不同
  - 声明：说明函数头
  - 定义：定义函数全部

函数可先调用后定义

- C不要求函数定义必须在调用之前。
- 有时没法保证

```
void c(void){ a();}
void b(void){ c();}
void a(void){ b();}
int main(void){
   a();
   return 0;}
```

- 自定义函数可在main之后定义
  - 符合我们解决问题的思维方式：
  - 先整体，后细节

函数声明-先调用后定义之问题

```
#include <stdio.h>

int main(void)
{
   double x, y, z;

   printf("Enter three numbers: ");
   scanf("%lf%lf%lf", &x, &y, &z);
   printf("Average of %g and %g: %g\n", x, y, average(x, y));
   printf("Average of %g and %g: %g\n", y, z, average(y, z));
   printf("Average of %g and %g: %g\n", x, z, average(x,y,z));

   return 0;
}

double average(double a, double b)
{
   return (a + b) / 2;
}
```

编译器：没信息假定返回int型：隐式声明

编译器：
怎么不是int

编译器：
参数匹配否？不清楚

函数声明

- 调用前，先声明(declare)函数
- 使编译器知晓函数前三要素
  - 一般形式：
  - 返回类型 函数名(形式参数) ；

不可或缺

- 在main后完整定义函数
- 函数声明又称函数原型，与其定义（头部）必须一致

函数声明

```
#include <stdio.h>
double average(double a, double b);
/*声明在先 */
int main(void)
{
  double x, y, z;
  printf("Enter three numbers: ");
  scanf("%lf%lf%lf", &x, &y, &z);
  printf("Average of %g and %g: %g\n", x, y,
  average(x, y));
  printf("Average of %g and %g: %g\n", y, z,
  average(y, z));
  printf("Average of %g and %g: %g\n", x, z,
  average(x, z));
  return 0;
}
double average(double a, double b) /* 定义殿后 */
{
  return (a + b) / 2;
}
```

函数声明

- 函数原型可省略形参名字：

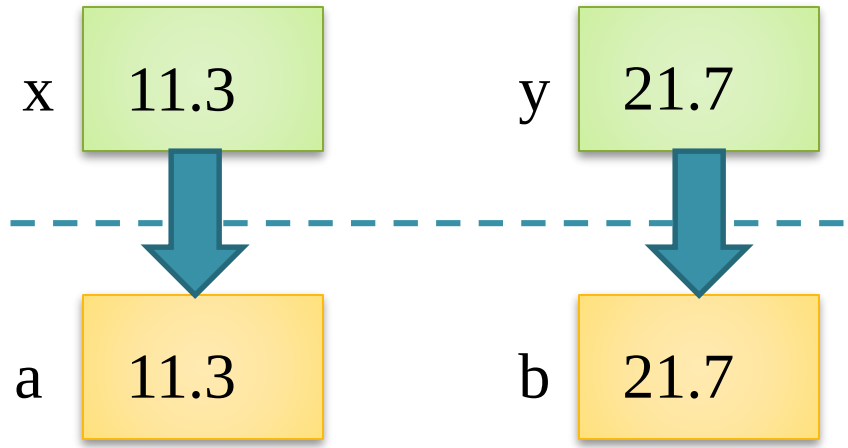
```
double average(double,
double);
```

- C99规定：在调用一个函数之前，必须先对其进行声明或定义。

实际参数→形式参数（赋值，代入）

- 定义或声明：
  - `double average(double a, double b)`
- 调用：
  - `z = average(x, y)`
- 值传递，单向性
  -

本质：
a=x; b=y;



形参是实参副本，形参改变不影响实参，
原件/复印件,eg,pic

实/形参值传递之优点

- 形参当函数内部变量使用（可修改），少用变量

```
int power(int x, int n)//求x的n次幂
{
   int result = 1;

   while (n-- > 0) //用n控制循环：少用变量
      result = result * x;

   return result;
}
```

实/形参值传递之缺点

- 形参改变不影响实参
  - 很难编写某些类型的函数，如希望函数改变实参
  - 如，ps证件照，成绩单
- 如，期望函数实现分解实数为整数和小数部分。

```
void decompose(double x, long
int_part, double frac_part){
      int_part = (long) x;
      frac_part = x - int_part;}
```

实/形参值传递之缺点

```
void decompose(double x, long
int_part, double frac_part){
    int_part = (long) x;
    frac_part = x - int_part;}
```

- 调用：
  - `decompose(3.14159, i, d);`
- 期望
  - `i = 3, d = 0.14159`
- 结果：
  - `int_part=i; frac_part=d`
  - `i,d`不变
- 原因：形参为实参副本
  - 函数对形参操作（修改），不改变实参

实参的类型转换

- 实参、形参类型不匹配。
- 本质赋值，削足适履
  - 形参=实参；
  - 形参：鞋，实参：脚
  - 实参隐式地转换成相应形参类型。
  - 例如：int实参——》double形参，实参自动转换成double类型。

数组型实际参数

- 函数只管数组名字而不管长度
  - 一个参数只能传递两个信息：类型、名字
  - 数组为实参，函数只能获取类型、名字
- 定义时可不说明数组的长度：

```
int f(int a[])   //不指定长度
{   …
}
```

- 如果函数需要，则必须把长度作为额外的参数进行传递

```
int f(int a[], int len)//指定长度
{   …
}
```

数组型实际参数

- 例子：

```
int sum_array(int a[], int n)
{
  int i, sum = 0;
  for (i = 0; i < n; i++)
    sum += a[i];
  return sum;
}
```

- sum_array函数原型：

```
int sum_array(int a[], int n);
```

- 可以忽略形式参数的名字：

```
int sum_array(int [], int);
```

数组型实际参数

- 调用：实参只给数组名

```
#define LEN 100
int main(void)
{
   int b[LEN], total;
   …
   total = sum_array(b, LEN);
   //注意：只传数组名，不要[]
}
```

数组型实际参数

- 数组长度通过额外参数传递
  - 可能出现长度参数与实际数组长度不相符。
- 函数无法检测参数传递的数组长度是否正确（相符）。
  - 传递长度<=实际数组长度：处理部分数组
  - 传递长度> 实际数组长度：数组下标越界
- 只传递数组名（地址）长度另传之好处
  - 提高复用性（函数可处理多种长度的数组）
  - 实参数组不整体传递给形参，不整体操作数组，传递效率

数组型实际参数

- 数组作函数参数，形参数组和实参数组统一
  - `int sum_array(int a[], int n)`
  - `total = sum_array(b, LEN);`
  - a[]与b[]是同一个数组
- why?
  - 同样参数值单向传递
  - 赋值a=b；数组名(地址)值单向传递
  - 形参、实参数组地址相同，同一个数组
  - 函数改变形参数组，即改变实参数组。
- 与变量参数不同
  - 变量：函数处理副本，eg.成绩册副本
  - 数组：函数通过地址处理原件，eg.成绩册存放地址

数组型实际参数

- 数组清零函数：

```
void store_zeros(int a[], int
n)
{
   int i;
   for (i = 0; i < n; i++)
     a[i] = 0;
}
```

- 调用：

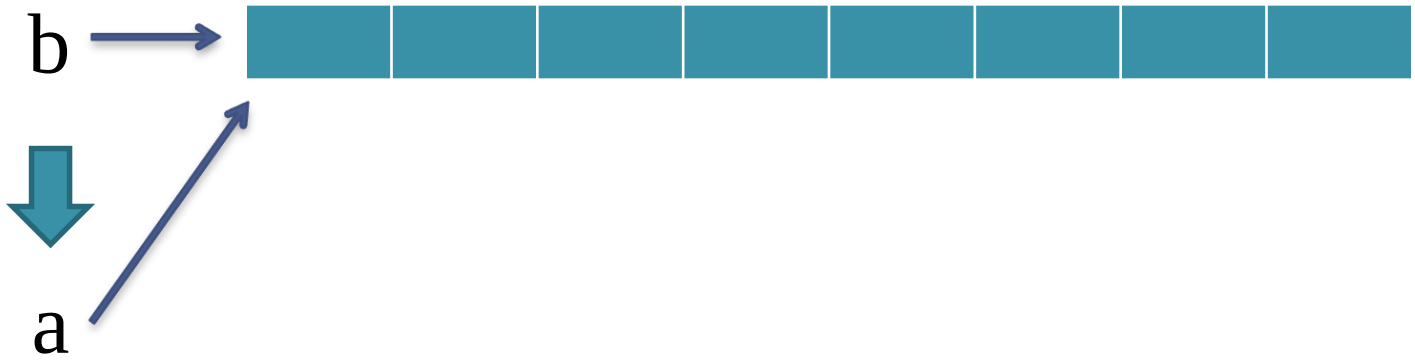
```
store_zeros(b, 100);
```

实现对数组b清零

数组型实际参数

- 数组名：数组地址

```
void store_zeros(int a[], int n);
int b[N];
store_zeros(b, 100);
```

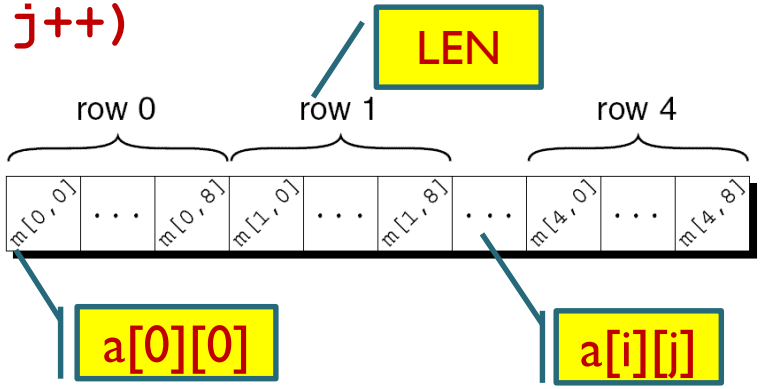


形/实参依然值传递：
形参数组地址改变不改实参数组地址

数组型实际参数

- 形式参数是多维数组，则只有第一维的长度可以省略

```
#define LEN 10
int sum_two_dimensional_array(int a[][LEN],
int n){
  int i, j, sum = 0;
  for (i = 0; i < n; i++)
    for (j = 0; j < LEN; j++)
      sum += a[i][j];
  return sum;}
```



- 实现函数内数组元素寻址
  - 通过下标找到元素
  - [i][j]计算&a[i][j]
  - 类比：根据行列确定队列中同学序号
  - 设&a[0][0]=p;
  - &a[i][j]=p+(i*LEN+j)*sizeof(type)
- 类似修路：
  - 第一维是长度，高维相当于宽度

函数小结

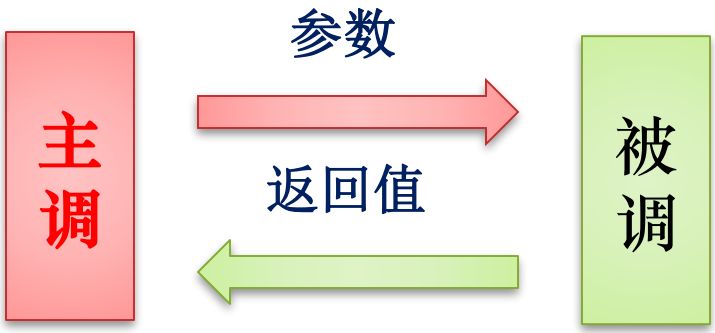
- 函数定义并列—独立之体现

```
int main(){
……}
int average(int a, int b){
……}
```

- 通过函数调用发生关联

```
int main(){
    z=average(x,y);printf(……);
}
```

- 函数与外界通信方法：参数传递和返回值（单向）



复合字面量 (C99)

- 调用sum_array，数组做实参：

```
int b[] = {3, 0, 3, 4, 1};
total = sum_array(b, 5);
```

- 先声明并初始化数组b
- 如果b不作它用，仅为调用sum_array来创建它，则有些浪费。

复合字面量 (C99)

- 通过指定其包含的元素而创建的没有名字的数组。

```
(int [4]){1, 9, 2, 1}
```

- 数组长度说明：可有可无

```
(int []){1, 9, 2, 1}
```

- 做参数

```
total = sum_array((int []){3,
0, 3, 4, 1}, 5);
```

return语句

- 函数终止
- 非void的函数必须return值。
- 格式：

```
return 表达式 ;
```

- 表达式：常量、变量或复杂表达式

```
return 0;
return status;
return n >= 0 ? n : 0;
```

return语句类型转换

- **return**表达式类型和函数返回类型不匹配：削足适履
  - 表达式隐式转换成返回类型
  - 本质上是赋值
- 例如：
  - 返回**int**型，但**return double**表达式，系统将会把表达式的值转换成**int**型。

return语句

- 常规返回：
  - 函数末尾return expr;
  - 返回类型void，执行完最后语句返回
- 非常规返回：
  - 函数中间return expr，通常if判断
  - 返回类型void，return后无expr。eg.

```
void print_int(int i)//输出非负整数
{
   if (i < 0) return;//条件退出函数
   printf("%d", i);
}
```

- 跳出
  - break跳出循环
  - return跳出函数

程序终止

- `return`
- `exit`

程序终止

- 正常情况下，main的返回类型是int

```
int main(void)
{
    …
}
```

- 老标准省略main返回类型，默认int类型，在C99中是非法的。
- 函数不返回值或无参数时可省略void，但最好指明。

程序终止

- main返回值：状态码，用于程序终止时检测程序状态：给用户的反馈
  - 正常终止：return 0
  - 异常终止：返回非0
    - eg，数组下标越界if(i>=LEN)return 1;
    - if(divisor==0) return 1;
    - 惯常用法：#define FAILURE 0xFFFFFFFF

exit函数

- 终止程序另一种方法<stdlib.h>
- 正常终止：exit(0);
- 异常终止：exit(1);
- 因为0有些模糊，使用EXIT_SUCCESS和EXIT_FAILURE（<stdlib.h>宏），值通常分别为0和1。

```
exit(EXIT_SUCCESS);
exit(EXIT_FAILURE);
```

exit函数vs return

- 形式

```
int main(void){
        a();
        …
        return 0;
}
void a(void){
        b();
        …
}
int b(void){
        if(...) return FAILURE;
        if(...) exit(FAILURE);
}
```

数被

n调用

温故而知新——函数

- 函数：具有某种功能的独立程序段
  - find()、insert()、delete()、output()
- main()vs其他函数
  - main()主函数
  - 其他函数：子函数
- main()组织其他函数完成程序工作
  - casear密码，coder()，decoder()

温故而知新——函数

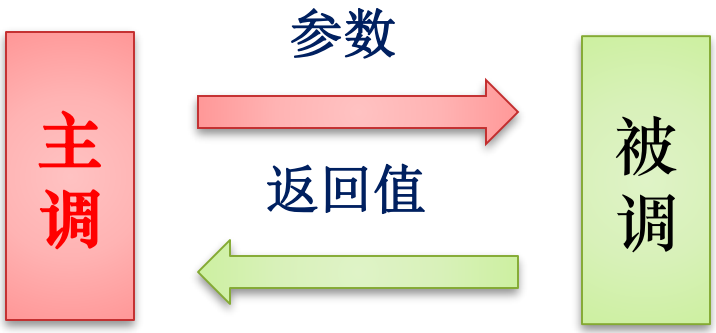
- 函数定义并列—独立之体现

```
int main(){
……}
int average(int a, int b){
……}
```

- 通过函数调用发生关联

```
int main(){
    z=average(x,y);printf(……);
}
```

- 函数与外界（其他函数）通信方法：参数传递和返回值



温故而知新——数组做参数

- 只传递数组名（地址），长度另传
  - 提高复用性（函数可处理多种长度的数组）
  - 实参数组不整体传递给形参，不整体操作数组，传递效率
- 函数定义：

```
type func(type ary[], int len)
int sum_two_dimensional_array(int a[][LEN], int n)
```

- 函数调用：

```
func(ac_ary, LEN);
sum_two_dimensional_array(array, N);
```

温故而知新——程序退出

- `return expr;`
  - 函数返回
- `exit (expr);`
  - 程序退出
  - 反馈程序状态码

递归

- 函数调用自身(recursive)
- 例如求n！：
  - $n! = n \times (n-1)!$//递归表达式，
  - 1!=1//初始（终止）条件
- 求解过程：
  - 先n!递推到(n-1)!直至1!
  - 再2!,3!回归到n!
  - eg. 19!=19*18!,18!=18*17!……2!=2*1
- 递归如下：

```
int fact(int n)
{
  if (n <= 1) //初值，1! =1
    return 1;
  else //递推
    return n * fact(n - 1);
}
```

递归终止条件

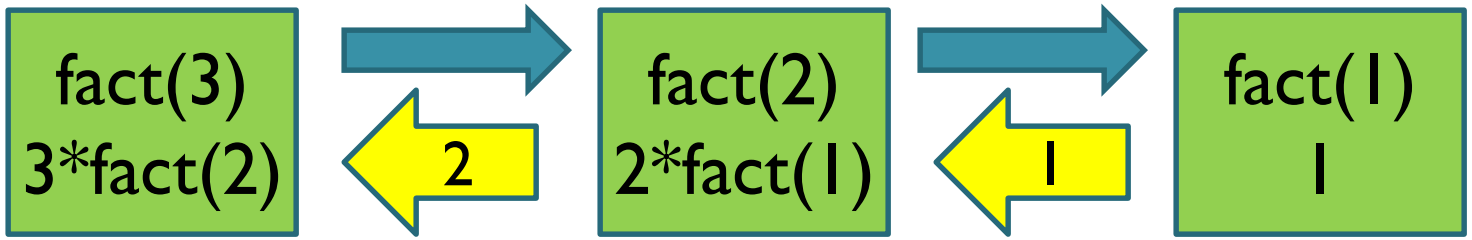
递归表达式

```
int fact(int n){
    if (n <= 1)  return 1;
    else return n * fact(n - 1);
}
```

递归跟踪：eg,n=3

```
i = fact(3);
```

- n=3，不<=1，调用fact(n-1)：fact(2)
- n=2，不<=1，调用fact(1)
- n=1，fact(1)返回1，fact(2)返回2×fact(1) = 2，fact(3) 返回 3 × 2 = 6。



递归

- 递归计算x^n：
- *递归表达式：* $x^n = x \times x^{n-1}$：

```
int power(int x, int n)
{

return
n == 0 ? 1 : x * power(x, n - 1);
```

程序练习：递归——斐波拉契

- f0=0, f1=1, f2=1,
- fn=fn-1+fn-2;

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

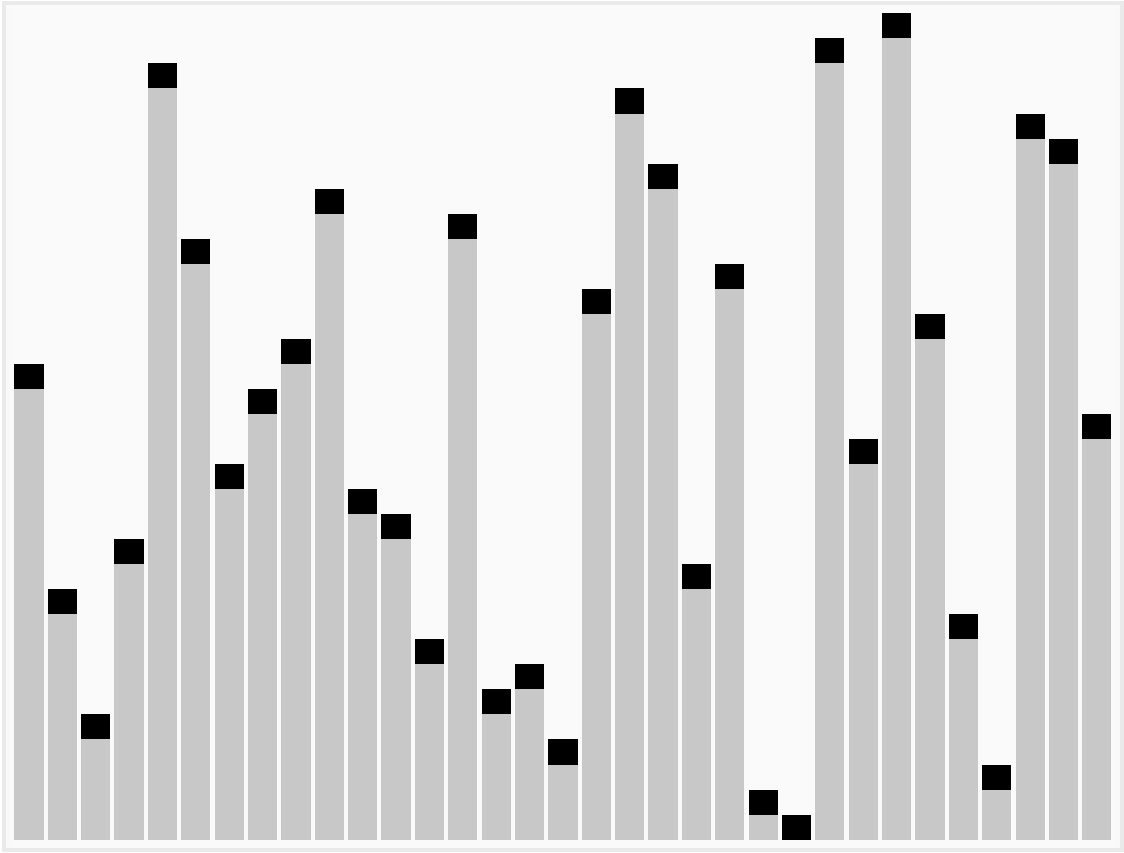
快速排序算法

- 递归对要求函数调用自身两次或多次的复杂算法非常有帮助。
- 递归常用于分治法(divide-and-conquer)。
  - 把一个大问题划分成多个较小的问题，
  - 然后采用相同的算法分别解决这些小问题。
- 分治法的经典示例：快速排序算法

快速排序算法

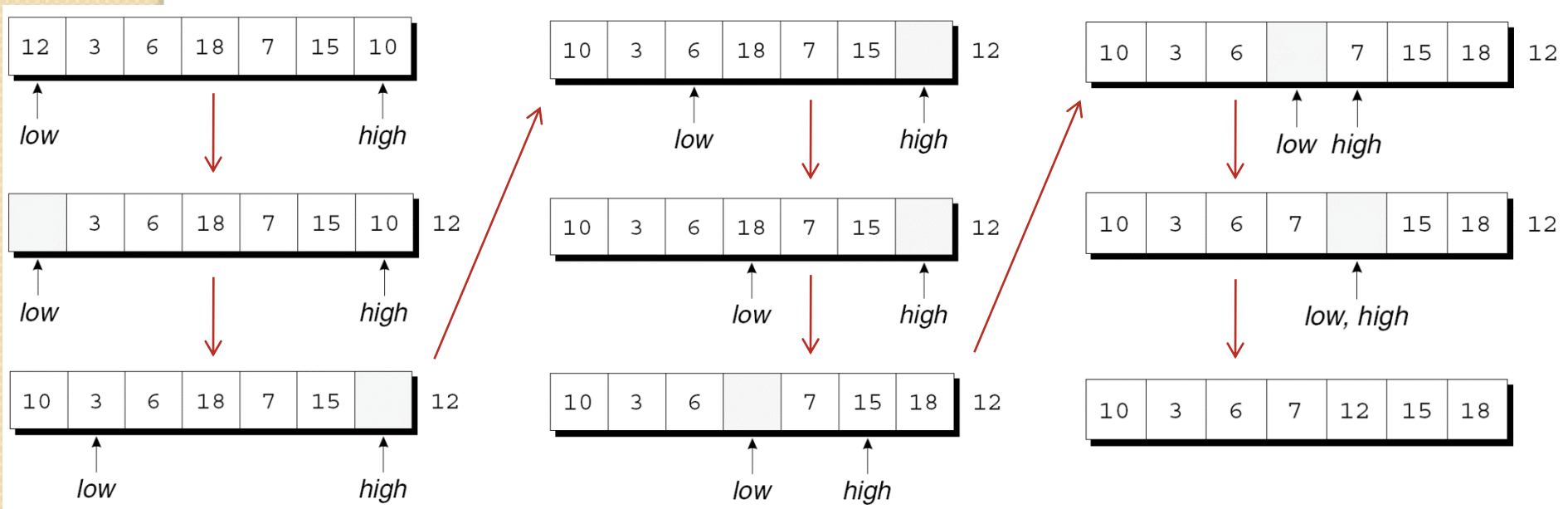
2006年11月10日家长开放日

快排gif



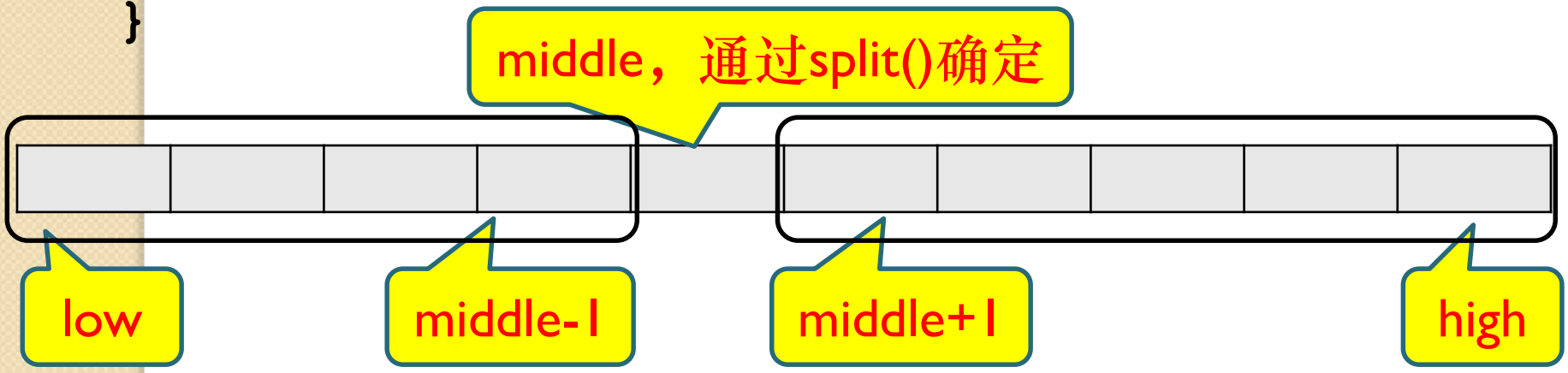
快速排序算法—分割

- 首元素做分割元素e，位置空缺出来
- 从尾首两个方向交替分别查找小于、大于e的元素。用两下标，如high，low
  - 尾：找小于e的元素（应排e前）
  - 首：找大于e的元素（应排e后）
- 找到后移动到分割元素或空缺的位置



```
#define N 10

void quicksort(int a[], int low, int high);
int split(int a[], int low, int high);

int main(void)
{
  int a[N], i;

  printf("Enter %d numbers to be sorted: ", N);
  for (i = 0; i < N; i++)   scanf("%d", &a[i]);

   quicksort(a, 0, N - 1);

  printf("In sorted order: ");
  for (i = 0; i < N; i++)   printf("%d ", a[i]);
  printf("\n");

  return 0;
}
```

```
void quicksort(int a[], int low, int high)
{
    int middle; //分割元素位置

    if (low >= high) return;//递归终止
    middle = split(a, low, high);
    quicksort(a, low, middle - 1);
    quicksort(a, middle + 1, high);
}
```



```
int split(int a[], int low, int high)
{
  int part_element = a[low]; //分割元素

  for (;;) {
    while (low < high && part_element <= a[high])
      high--; //移动high，两种情况结束循环
    if (low >= high) break;//分割元素位置确定
    a[low++] = a[high]; //移动元素
    while (low < high && a[low] <= part_element)
      low++; //移动low
    if (low >= high) break;
    a[high--] = a[low];
  }

  a[high] = part_element; //移动分割元素（就位）
  return high; //返回分割元素位置
}
```