21世纪高等教育计算机规划教材

COMPUTER

# C语言程序设计习题解析与实验指导（第2版）

The Answer and Practice of C Programming

朱立华 郭剑 主编

俞琼 吴家皋 朱旻如 副主编

- 课后习题分析深入，解析透彻
- 补充习题紧扣考纲，题量丰富
- 实验指导分层设计，指导详尽

人民邮电出版社
POSTS & TELECOM PRESS

21世纪高等教育计算机规划教材

COMPUTER

# C语言程序设计习题解析与实验指导（第2版）

The Answer and Practice of C Programming

朱立华 郭剑 主编
俞琼 吴家皋 朱旻如 副主编

人 民 邮 电 出 版 社
北 京

**图书在版编目（CIP）数据**

C语言程序设计习题解析与实验指导 / 朱立华，郭剑主编. — 2版. — 北京 : 人民邮电出版社，2014.10
21世纪高等教育计算机规划教材
ISBN 978-7-115-37045-7

Ⅰ. ①C… Ⅱ. ①朱… ②郭… Ⅲ. ①C语言－程序设计－高等学校－教学参考资料 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2014)第203506号

## 内 容 提 要

本书是和《C 语言程序设计（第 2 版）》（朱立华 郭剑主编）教材配套使用的参考书，也是可以完全独立于主教材的一本自成体系的实验实习教材。

全书主要分为四部分：第一部分是教材思考与练习解析；第二部分是教材习题参考答案与解析；第三部分参考了最新的二级考试大纲，对应于教材的章节，补充了大量精选习题，并给出了参考答案；第四部分精心设计了 10 次实验，选题体现分层次思想，在每一次实验中精选了具有代表性的实验题目并给出了详细指导。

本书内容紧扣 C 语言程序设计的相关知识点，覆盖面广、题目量大、解析透彻、实验注重能力的培养，是学习 C 语言程序设计的一本实用的参考书。

◆ 主　　编　朱立华　郭　剑
副 主 编　俞　琼　吴家皋　朱旻如
责任编辑　武恩玉
责任印制　彭志环　焦志炜

◆ 人民邮电出版社出版发行　　北京市丰台区成寿寺路 11 号
邮编　100164　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
大厂聚鑫印刷有限责任公司印刷

◆ 开本：787×1092　1/16
印张：19　　2014 年 10 月第 2 版
字数：389 千字　　2014 年 10 月河北第 1 次印刷

定价：42.00 元

读者服务热线：(010)81055256　印装质量热线：(010)81055316
反盗版热线：(010)81055315
广告经营许可证：京崇工商广字第 0021 号

# 前　言

C 语言作为读者学习高级语言程序设计的第一门语言，存在以下两大难点：概念虽然清楚但是不知如何运用，程序虽然能读懂但是自己不会编程。

出现这两大难点的主要原因是：缺乏一定的分析问题和运用知识的能力，缺少足够的实践与练习，开发工具的使用不熟练。

为了帮助读者学好 C 语言，真正掌握用 C 语言解答各种考题，编写出正确甚至完美的程序，本书是与《C 语言程序设计（第 2 版）》（朱立华　郭剑主编）教材配套使用的参考书，重点着力解决以下 3 个问题：

第一，学生如何深入思考。本书的第一部分是主教材章节内容中适时提出的思考题的解析，帮助读者举一反三，深入思考相关的内容，并且在思考过后可以查阅到思考的结论是否正确，存在什么偏差，达到“有问必答”的效果。

第二，概念如何用于解题。本书的第二部分是主教材习题参考答案与解析，通过对主教材后的每一个习题的透彻解析，帮助读者更好地理解和运用所学的理论知识；本书的第三部分参考了最新的全国计算机等级考试二级考试大纲，对应于教材的章节，补充了大量精选习题并给出参考答案，使读者通过更多的练习，大量的训练，巩固所学知识，加强运用知识解题的能力。

第三，理论如何指导实践。本书的第四部分精心设计了 10 次实验，每一次实验精选了具有代表性的、能帮助读者掌握对应知识和技能的题目。本书实验的一大特点是选题体现分层次思想：无“★”标识的题目要求每一个读者都必须完成，这样的训练能使读者掌握基本的编程方法；掌握程度更好的读者可以完成题目前面打“★”的具有较高难度的题目，这样的训练能使读者提高分析问题的能力，能写出尽可能好的程序。为使初学者在编程时少走弯路，帮助他们分析和解决问题，每一个实验题目给出了详细指导，将调试工具的使用要求贯穿在每一次实验中不断训练和强化。

书中所有的程序都在 Visual C++ 6.0 下调试通过，每一道编程题都可能存在多种解答，读者可以进一步思考，拓宽自己的思路。

本书由朱立华、郭剑任主编，俞琼、吴家皋、朱旻如任副主编。教材第一部分和第二部分由朱立华、郭剑、吴家皋、朱旻如共同完成；第三部分由俞琼完成，第四部分由朱立华在一版的基础上改编完成，全书由朱立华最后统稿。南京邮电大学 2013 级贝尔学院的薛聪、刘希鹏、李博睿、毛雪宇、熊明亮、潘宏程、伏晓等对补充习题部分作了校对与程序的调试和验证工作，在此深表感谢。

由于编写时间紧，作者水平有限，书中难免存在不准确之处，敬请读者批评指正。

编　者

2014 年 7 月于南京

# 目 录

# 第一部分
# 教材思考与练习解析

# 第 2 章
# 初识 C 语言源程序及其数据类型

1. 例 2.2 的思考题：（1）若本例 scanf 语句中的变量前忘写了取地址符“&”，程序运行结果会怎样？（2）若 scanf 语句中，double 型变量 d 的输入格式写成了" %f"，程序运行结果又会怎样？

**【分析与解答】**

（1）scanf 的输入变量前不加取地址符“&”，程序在编译时会有警告信息：

warning C4700: local variable 'd' used without been uninitialized

warning C4700: local variable 'a' used without been uninitialized

warning C4700: local variable 'ch1' used without been uninitialized

而程序仍能编译成功。但是程序运行后，无论输入什么数据，程序都会异常终止。

因此，用 scanf 输入数据时，输入变量必须给出变量地址，切忌丢失取地址符“&”。

（2）将 double 型变量的输入格式写成"%f"，程序在编译时完全正确，没有任何警告或错误信息。运行此程序，屏幕上首先会显示一条提示信息：

Please input ch1, a, d:

**若用户从键盘输入为：** A␣127␣2.71828183 <回车>

则输出结果为：

A 127 2.71828183

ch1=%c: A

ch1=%d: 65

ch2=%c: a

a=%d: 127

a=%x: 7f

d=%f: -92559604411795182000000000000000000000000000000000000000000000000000000000.000000

f1=%f: 3.140000

f1=%g: 3.14

f1=%e: 3.140000e+000

f2=%10.2f:  -123.46

由此可见，double 型变量 d 的值不是用户输入的 2.71828183，而是一个“乱码”。

因此，用 scanf 输入 double 型变量时，必须使用%lf 或%le，而不能使用%f 或%e。

2. 例 2.3 的思考题：若用户从键盘输入 abc<回车>，上例的运行结果会怎样？

**【分析与解答】**若用户从键盘输入为：abc<回车>

则输出结果为：

a␣␣␣␣␣␣␣␣c

a␣␣␣␣␣␣␣␣c

这时，用户输入的第 1 个字符'a'由函数 getchar 赋给了变量 ch1；用户输入的第 2 个字符'b'被语句 getchar( );忽略了；用户输入的第 3 个字符'c'由下一条语句赋给了变量 ch2，故有上述输出结果。通过此题，读者将更加熟悉函数 getchar 的输入特性。

3. 例 2.4 的思考题：请有兴趣的读者在程序中增加一条修改语句：pi=3.14，并再次编译程序，观察编译器会给出什么错误提示信息。

**【分析与解答】**增加语句：pi=3.14;，重新编译程序时，系统会报错误信息：

error C2166: l-value specifies const object


const 变量为只读变量，不能在程序中进行赋值或修改，否则编译通不过。因此，const 修饰符有助于提高程序的健壮性和安全性。

# 第3章 运算符与表达式

1. 例 3.1 的思考题：若将(-a)%(-2)中的小括号都去掉，则程序编译、运行会有什么结果？

**【分析与解答】**按要求修改后，程序编译正确，运行的结果与修改前完全一样。因为，表达式“(-a)%(-2)”去掉小括号后变成“-a%-2”，根据运算符的优先级，取负数运算符“-”的优先级比取余运算符“%”要高，所以仍与修改前的表达式等价。此题说明正确掌握运算符的优先级的重要性，同时，优先级也不用死记硬背，在任何情况下利用小括号( )可以提升优先级。

2. 例 3.3 的思考题：若有 int a=1,b=1;，则运算 a=(a++)+b；后，a、b 的值各为多少？

**【分析与解答】**运行上述语句后，我们得到 a=3，b=1。

此题的关键是正确的理解表达式语句“a=(a++)+b;”的含义。根据“后++”的定义，该语句相当于 “a=a+b; a=a+1;”。又因为 a、b 的初值为 1 时，所以该表达式运算结束时，变量 a=3，b=1，同时整个表达式的结果就是变量 a 的值为 3。

# 第 4 章 程序流程控制

1. 例 4.1 的思考题：该程序有一个前提假设，即输入的 3 条边一定能构成一个三角形。但在实际情况中，用户的输入是不可预知的。如果 3 条边不能构成一个三角形，程序的运行结果是什么？如果用户输入 0 或者负数，程序的运行结果又将如何？测试下列数据的输出结果，并思考代码的缺陷和改进方法。

1　2　3

1　2　4

0　1　2

-1　3　4

**【分析与解答】**以上 4 组数据输入后对应的输出结果如表 1-1 所示。

表 1-1　　VC++ 6.0 环境下的测试结果

| 测试数据 | | | 结果输出 |
|---|---|---|---|
| 1 | 2 | 3 | area = 0.000000 |
| 1 | 2 | 4 | area = -1.#IND00 |
| 0 | 1 | 2 | area = -1.#IND00 |
| -1 | 3 | 4 | area = 0.000000 |

从测试结果可以看出，计算机很“笨”，只会执行给定的代码。它并不知道输入数据是否合法，也不会主动进行判别，由此会导致许多错误的结果。为避免这种情况的发生，目前我们可以增加一些提示信息，告诉用户合法的数据输入格式是什么。当然这还不够，我们还需要对用户的输入数据进行判别，并进行针对性地处理，如让用户重新输入，或者对错误的数据进行纠正等，以此增强代码的容错性。这些知识可以在后续的章节中学到。

2. 例 4.13 的思考题：能否对该代码进一步优化，将代码由两重循环削减为一重循环？

**【分析与解答】**在教材中，三层循环变为两层循环的关键是，当 a、b 确定时，c 的值也就确定了，它可以用 a 和 b 来表示，不必再从 0 尝试至 100，从而减少一层循环。

因此，如果再削减一层循环，意味着当某一个变量（例如 a）确定时，b 和 c 的值可以通过变量 a 计算出来，不必再经过循环来尝试寻找。事实上，这是可以做到的。对于方程组：

$$\begin{cases} a+b+c=100 \\ 5a+3b+c/3=100 \end{cases}$$

我们可以得到：

$$\begin{cases} b = (100 - 7a) / 4 \\ c = (300 + 3a) / 4 \end{cases}$$

由此代码改进如下：

```
#include <stdio.h>
int main( )
{
    int a, b, c;
    for ( a = 0 ; a <= 14 ; a++ )
    {
            b = ( 100 - 7 * a ) / 4;
            c = ( 300 + 3 * a ) / 4;
            if ( a + b + c == 100 )
            {
                printf( "%d, %d, %d\n", a, b, c );
            }
        }
        return 0;
}
```


此代码中 a 的取值范围是[0, 14]，这是由 b = ( 100 - 7a ) / 4 所知，a 的值最大为 14。

# 第 5 章
# 函数的基本知识

1. 例 5.3 的思考题：请读者在此函数的基础上稍加修改，依次定义表 5-1 中另外两个功能类似的函数：（1）函数 void DrawLine( int n ) 实现画出一条由 *n* 个减号组成的横线；（2）函数 void DrawLine( int n , char c) 实现画出一条由 *n* 个指定字符组成的横线。

**【分析与解答】**例 5.3 所定义函数的首部为：void DrawLine ( ) ，函数实现了通过 printf 画出固定长度（由 for 循环控制，循环终值为 30）、固定字符（'-'）组成的一条横线。如果要实现（1）所要求的由 *n* 个减号组成的横线，则体现为将循环控制的终值由 30 改为形式参数 *n*；（2）在（1）的基础上，再将原来固定输出的减号改变为第二形式参数 c 所代表的字符。根据分析，这两个函数的实现代码如下：

```
(1) void DrawLine (int n )
    {
        int i;
        for (i=1;i<=n;i++)
            printf("-");
        printf("\n");
    }
```

```
(2) void DrawLine( int n , char c)
    {
        int i;
        for (i=1;i<=n;i++)
            printf("%c",c);
        printf("\n");
    }
```

2. 例 5.4 的思考题：请读者修改本题的主函数，调用 JudgePrime 函数求出所有的 3 位质数并按每行 5 个的形式输出。

**【分析与解答】**将主函数中读入一个 m 改为用 m 作为循环控制变量，因为要求的是 3 位质数，所以 m 的范围是[100，999]，但因为 100 是偶数不是质数，因此 m 的初值从 101 开始，并且变化步长为 2，也就是说，仅考虑所有的 3 位奇数，偶数都不用考虑，这样可以减少循环次数。换行可以用一个累加器，每输出一个质数加 1，然后用除 5 余数为 0 控制换行即可。完整的程序代码如下：

```
#include<stdio.h>
#include<math.h>                    /*sqrt 函数定义在此文件中*/
/*函数功能：判断一个正整数是否为质数
函数参数： 一个整型形式参数
函数返回值： int 型，用值 1 表示 n 是质数，值 0 表示不是质数
*/
int JudgePrime(int n)               /* 形式参数 n 将接收正整数*/
{
```

```
    int i,k ;
    int judge=1;                    /*judge 存判断结果，未判断时默认为是质数*/
    if ( n==1 )                     /*如果参数为 1，不是质数*/
            judge=0 ;               /*为 judge 变量赋值为 0，表示不是质数*/
    k = (int) sqrt ( n );                   /*k 保存了需要扫描的除数终值*/
    for (i = 2; judge && i<=k ; i++) /*i 作除数，从 2 到除数终值扫描，若 judge 已为 0，
则停止循环*/
            if (n % i == 0)                 /*n 被某除数整除，则不是一个质数*/
                judge=0 ;                   /*为 judge 变量赋值为 0，表示不是质数*/
    return judge;                           /*返回 judge 的值，如果保持为 1，则是质数*/
}

int main ( )
{
    int m;
int count=0;                                /*统计已输出的质数个数，控制换行*/
    for (m=101;m<1000;m=m+2)                /*只需要考虑所有的三位奇数*/
    if ( JudgePrime (m ) )          /*判断质数函数的调用*/
    {
            printf ( "%6d" , m ) ;
            count++;                        /*输出质数的个数加 1*/
            if (count%5==0)                 /*控制每 5 个换一行*/
                printf("\n");
      }
    return 0 ;
}
```

3. 例 5.6 的思考题：如果仅声明了函数 gcd，而忘记后面再定义该函数，编译时会出现什么现象？

**【分析与解答】**只声明不定义 gcd 函数，如果只是做编译，不会报错，显示“0 errors, 0 warnings”，但是如果继续链接运行程序，将会出现如下报错：“error LNK2001: unresolved external symbol _gcd”，意即“不确定的外部符号_gcd”，如果连接程序不能在所有的库和目标文件内找到所引用的函数、变量或标签，将产生此错误消息。这里发生这一错误的原因就是主函数所引用的函数 gcd 的定义不存在。因此函数声明之后，必须要定义函数。

4. 例 5.8 的思考题：在本例程序的基础上稍作修改，用递归方法实现，对一个正整数 n 各位数字逆序输出。例如，输入 175，则输出 571。

**【分析与解答】**该题的最后一个测试用例，当输入的进制数为 10 时，输出结果是 n 按位分离正序输出，如果想要逆序输出，则先输出本层余数，再进行递归调用可。相比于例 5.8，这里的 B 固定是 10，因此不必要设为一个形式参数了。实现逆置的完整程序源代码如下：

```
#include <stdio.h>
void MultiBase(int n);              /*递归函数的原型声明*/
int main( )
{
      int n;
```

```
    do
    {    scanf("%d",&n);
}while (n<=0);                    /*直到读入的 n 为正整数*/
    printf("change result:\n");
    MultiBase(n);                 /*调用递归函数进行数制转换*/
    printf("\n");
    return 0;
}
/*函数功能：用递归方法将一个正整数 n 逆置输出
 函数参数： 1 个整型形式参数 n 对应于待转换的十进制整数
 函数返回值：无类型
*/
void MultiBase(int n)
{
    int m;
    if(n)                         /* n!=0 则递归调用，而 n==0 时终止递归*/
    {
        m=n%10;                   /*求本层的余数*/
        printf("%d",m);           /*余数原样输出*/
        MultiBase(n/10);          /*递归调用*/
    }
}
```

5. 例 5.9 的思考题：对本例分别作以下几种修改，每次只修改一处，然后恢复原样，再作下一次修改，请上机编程，观察程序在编译、运行时将会出现什么现象，并分析原因。

（1）在 sumDigit 函数体内增加一语句：

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

（2）在 sumDigit 函数体 return 前增加语句：

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

（3）删除 sumDigit 函数开头变量 sum 的定义。

（4）在 sumDigit 函数体内增加变量定义：

int count=0;

（5）将全局变量 count 的定义位置移到 main 函数后 sumDigit 之前。

**【分析与解答】**

（1）在 sumDigit 函数体内增加一语句：printf("a=%d\n",a);，重新编译之后，报错：error C2065: 'a' : undeclared identifier，意即 a 是未定义的标识符，因为变量 a 是 main 函数中定义的局部变量，其作用域仅限于 main 函数，因此到了 sumDigit 函数中，它就是一个未定义的标识符了。

（2）在 sumDigit 函数体 return 前增加语句：printf("b=%d\n",b);，重新编译之后，报错：error C2065: 'b' : undeclared identifier，意即 b 是未定义的标识符，因为变量 b 是在 sumDigit 的 for 循环体复合语句中定义的局部变量，其作用域仅限于该复合语句，本函数中的其余位置不是其作用域，因此 return 之前 b 是不可以被访问的局部变量了。

（3）删除 sumDigit 函数开头变量 sum 的定义，重新编译之后，报错：error C2065: 'sum' : undeclared identifier，意即 sum 是未定义的标识符，错误原因与（1）类似，删除之后，sumDigit

函数中就没有变量 sum 了，另一个在主函数中定义的局部变量 sum 不能在 sumDigit 函数中使用。

（4）在 sumDigit 函数体内增加变量定义：int count=0;，那么，在 sumDigit 函数体内起作用的就是这个新定义的局部变量，同名的全局变量在此函数中没有作用域，程序正常编译及运行，但是输出结果不一样，main 函数中输出的 count 永远为 0，函数 sumDigit 的输出结果是正确的。此问题主要涉及全局变量在同名局部变量所在的作用域内暂时隐身无作用。

（5）将全局变量 count 的定义位置移到 main 函数后 sumDigit 之前，重新编译之后，报错：error C2065: 'count' : undeclared identifier，因为放到这个位置之后 ，虽然 count 依然是全局变量，但是其作用域却是从定义点之后开始的全部程序，因为主函数在这一单子定义之后，要不然就好了。

6. 例 5.10 的思考题：

（1）如果将函数 fun 中的 static int f= 1;改为 int f= 1;，重新运行程序，结果是什么？请解释原因；

（2）如果将 main 函数中的循环 for(i =1;i <=5;i++)改为 for(i =3;i <=5;i++)，那么输出结果是 3 到 5 的正确阶乘吗？请解释原因；

（3）如果将 main 函数中的循环 for(i =1;i <=5;i++)改为 for(i =1;i <=5;i=i+2)，那么输出结果是 1、3、5 的正确阶乘吗？请解释原因；

（4）如果将 main 函数中的循环 for(i =1;i <=5;i++) printf("%d != %d\n" , i , fun(i)); 改为 printf("%d != %d\n" , 5, fun(5));，那么输出结果是 5 的正确阶乘吗？请解释原因；

（5）如何在 main 函数中稍加修改，达到求 sum= $\sum_{i=1}^{5} i!$ 的效果。

**【分析与解答】**

（1）如果将函数中的 static int f=1; 改为 int f=1;，重新运行程序，结果如下：

1 != 1

2 != 2

3 != 3

4 != 4

5 != 5

**原因：**去掉 static 之后，f 就是自动局部变量，不具有“记忆功能”，每一次调用都是重新分配存储空间并且都被重新初始化为 1，因此表达式 f=f*n;实际上相当于 f=n;，因此主函数中 5 次调用所提供的实参依次是 1 至 5，于是输出的计算阶乘的结果也就依次是 1 至 5。

（2）如果将 main 函数中的循环 for(i =1;i <=5;i++)改为 for(i =3;i <=5;i++)，f 仍然恢复成静态局部变量，那么输出结果如下：

3 != 3

4 != 12

5 != 60

**原因：**每一个数字阶乘的输出结果是正确结果的一半，这是因为循环不从 1 开始，实际上在求 3 的阶乘时，直接得到的是 3 的结果，而不是 1*2*3 的结果，这样就变成实际结果的一半；后

面的 4！是在 3！的基础上乘以 4 得到的，所以结果也是实际结果的一半，5！结果原理与前面的相同。

（3）如果将 main 函数中的循环 for(i =1;i <=5;i++)改为 for(i =1;i <=5;i=i+2)，那么输出结果如下：

1 != 1

3 != 3

5 != 15

**原因：**循环的控制变量 i 的变化步长不是 1，但是求 n 的阶乘的基本方法是从 1 一直乘到 n，如果变化步长为 2，那么那些偶数就没有参与运算，因此上面的输出只有 1!是对的，其余的下一项的阶乘都少了中间相隔的那个偶数倍。

（4）如果将 main 函数中的循环 for(i =1;i <=5;i++) printf("%d != %d\n",i,fun(i));改为 printf("%d != %d\n",5,fun(5));，那么输出结果如下：

5 != 5

因为不经过循环，就没有计算从 1 到 5 依次相乘的过程，只是计算了 1*5 的结果，因此答案不正确。从本例可以看到，递归程序虽然简洁，但是读者一定要保证起点和终点合理。

（5）为了求阶乘的和，一定需要引入一个求和变量并且初始化为 0，将 fun 函数每次调用结果直接累加到求和变量中，主函数中只需要用一层循环，修改后的主函数代码如下：

```
int main()
{
    int i,sum=0;                    /*定义局部变量 i*/
    for(i =1;i <=5;i++)             /*循环，依次求 1 到 5 的阶乘*/
            sum+= fun(i);
    printf("sum= %d\n" , sum);
    return 0;
}
```

# 第6章 数 组

1. 例 6.1 的思考题：在程序的 return 语句之前增加一条语句：score1=score2;，重新编译程序，观察错误提示并解释原因。

**【分析与解答】**在 VC++ 6.0 中重新编译程序，会报一个错：error C2106: '=' : left operand must be l-value，意思是赋值号左边必须是一个变量，现在赋值号左边是数组名 score1，数组名本质上是一个地址常量而不是变量，这个知识会在第 7 章作详细介绍，所以直接对数组名赋值是错误的，如果要实现两个数组的所有元素值对应完全一样，就应该用本例的方法，通过循环方式对元素逐个赋值，因为数组的元素都是基类型的变量，能用在赋值号左边。

2. 例 6.4 的思考题：如果二维数组的行、列数完全相同，即表示方阵的情况下，（1）程序要做怎样的改动就能实现方阵的转置？（2）只用一个数组是否就能实现？

**【分析与解答】**

（1）例 6.4 中，如果二维数组行列相同，只需要修改 define 处的定义，即将原程序中的：#define ROW 3

#define COL 4

修改成现在矩阵的行列数。且由于行列数相同，因此只要定义一个符号常量：

#define SIZE 4

就能满足要求，符号常量的使用使得程序更具有通用性。

（2）当数组行、列相同，就是一个方阵，用一个数组就可以表示了，参考程序如下：

```
#include <stdio.h>
#include <time.h>
#include <stdlib.h>
#define  SIZE 4
int main()
{   int array[SIZE][ SIZE];
      int i,j;
    int temp;
    srand(time(NULL));
    for(i=0;i< SIZE;i++)
    {
          for(j=0;j< SIZE;j++)
              array[i][j]=rand()%100+1;       /*产生 100 以内随机值给数组元素赋值*/
    }
```