

程序设计基础

Fundamental of Programming

清华大学软件学院

刘玉身

liuyushen@tsinghua.edu.cn

Lecture 3 : Review

- Statements and Compound Statements (语句和复合语句)
- Making Decisions (选择结构)
 - Relational Operators (关系运算符)
 - Logical Operators (逻辑运算符)
 - The **if** Statement (**if**语句)
 - The **if-else** Construct
 - **Nested if** Statements (嵌套**if**语句)
 - The **else-if** Construct
 - The **switch** Statement (**switch**语句)
 - The Conditional Operator (条件运算符)

Lecture 4 : Outline

- **Program Looping (循环结构)**
 - The **for** Statement
 - Relational Operators
 - **Nested for** Loops
 - Increment Operator
 - Program Input
 - **for** Loop Variants
 - The **while** Statement
 - The **do-while** Statement
 - The **break** and **continue** Statement

Lecture 4: Program Looping

- The **for** Statement
- The **while** Statement
- The **do-while** Statement
- The **break** and **continue** Statement
- Examples

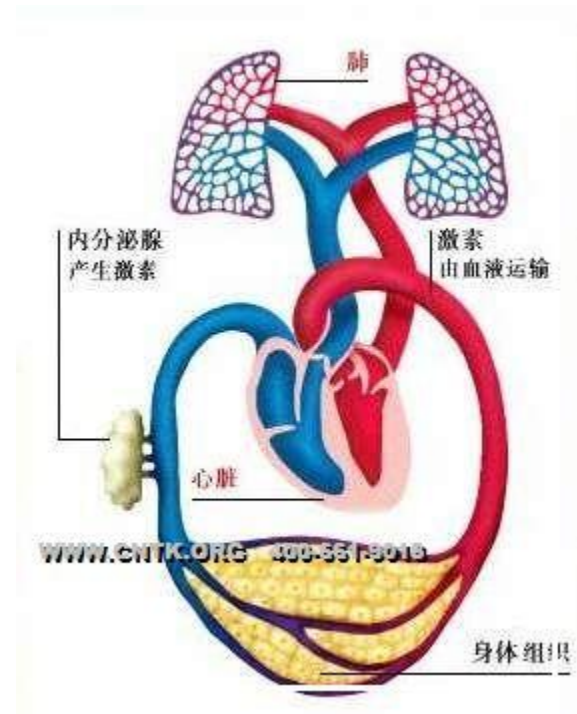
循环控制



顺序



选择



循环

Example: Program Looping

- 循环：重复地去执行某一段代码
- Simple example: displaying a message 100 times:

```
printf("hello !\n");  
printf("hello !\n");  
printf("hello !\n");  
  
...  
printf("hello !\n");  
printf("hello !\n");
```

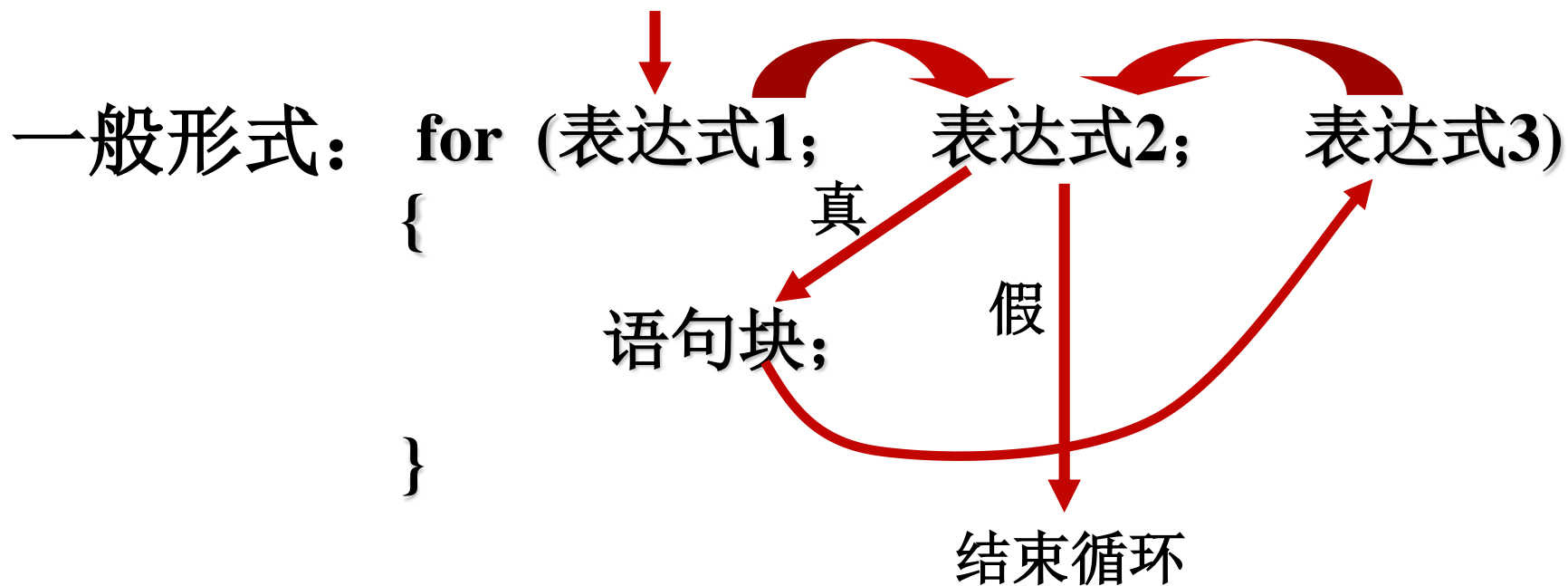
```
Repeat 100 times  
    printf("hello !\n");
```

C语言提供了三种循环语句: **for**, **while**, **do-while**

Lecture 4: Program Looping

- The **for** Statement
- The `while` Statement
- The `do-while` Statement
- The `break` and `continue` Statement
- Examples

for 语句



各部分最少被执行几次？

```
for (init; condition; update)
{
    statements;
}
```

- 事前须初始化、事后要更新
- 先条件测试再执行
- 计数驱动

```
/* Frog lifetime */
int days;
for(days = 155; days > 0; days--)
{
    work_all_day();
    sleep_all_night();
}
die_quietly();
```

分析运行结果1

```
int    i;  
for(i = 100;  i > 0;  i -= 2)  
{  
    printf("%d\n", i) ;  
}
```

100
98
96
...
2

分析运行结果2

多了一个分号

```
sum = 0;
for(i = 1; i <= 10; i = i + 1);
{
    sum = sum + i ;
}
printf("%d\n", sum) ;
```

11

分析运行结果3

```
sum = 0;
for(i = 1; i != 10; i = i + 2)
{
    sum = sum + i ;
}
printf("%d\n", sum) ;
```

死循环!?

Infinite loops

- It's the task of the programmer to design correctly the algorithms so that loops end at some moment !

```
#include <stdio.h>
int main (void)
{
    int i, n = 5, sum =0;
    for ( i = 1; i <= n; n = n + 1 ){
        sum = sum + i;
        printf ("%i %i %i\n", i , sum, n);
    }
    return 0;
}
```



What is wrong here ?
Does the loop end?

分析运行结果4

```
double x ;  
for(x = 0.0; x < 10.0; x = x + 0.2)  
{  
    printf("%.2f\n", x); //10.0是否打印?  
}
```

```
double x ;  
for(x = 0.0; x < 10.0; x = x + 0.2)  
{  
    printf("%.18f\n", x);  
}
```

执行多少次?

执行结果 (51次):

如何改?

```
0.00
0.20
0.40
0.60
0.80
1.00
.....
9.20
9.40
9.60
9.80
10.00
```

```
0.000000000000000000000000
0.200000000000000000000010
0.400000000000000000000020
0.600000000000000000000090
0.800000000000000000000040
1.000000000000000000000000
.....
9.199999999999999999300
9.3999999999999999998600
9.5999999999999999997900
9.7999999999999999997200
9.9999999999999999996400
```

用整型作为循环变量

```
int i ;  
double x ;  
for(i = 0; i < 50; i = i + 1)  
{  
    x = (double) i / 5.0;  
    printf("%.18f\n", x);  
}
```

执行结果 (50次):

```
0.000000000000000000000000
0.200000000000000000000010
0.400000000000000000000020
0.599999999999999999999980
0.800000000000000000000040
1.000000000000000000000000
.....
9.1999999999999999999999300
9.4000000000000000000000400
9.5999999999999999999999600
9.8000000000000000000000700
```

高斯的难题

德国数学家高斯，在上小学的时候，老师出了一道难题，计算 $1+2+3+\dots+100$ ，高斯很快就在自己的小石板上写出了答案**5050**，老师非常惊讶，高斯怎么算得这么快呢？原来，高斯不是一个数一个数按部就班地加起出来的，而是发现这些数字有一个规律，一头一尾依次两个数相加，它们的和都是一样的： $1+100=101$ ， $2+99=101$ ，一直到 $50+51=101$ ，一共是50个101，所以，他很快就把答案算出来了。

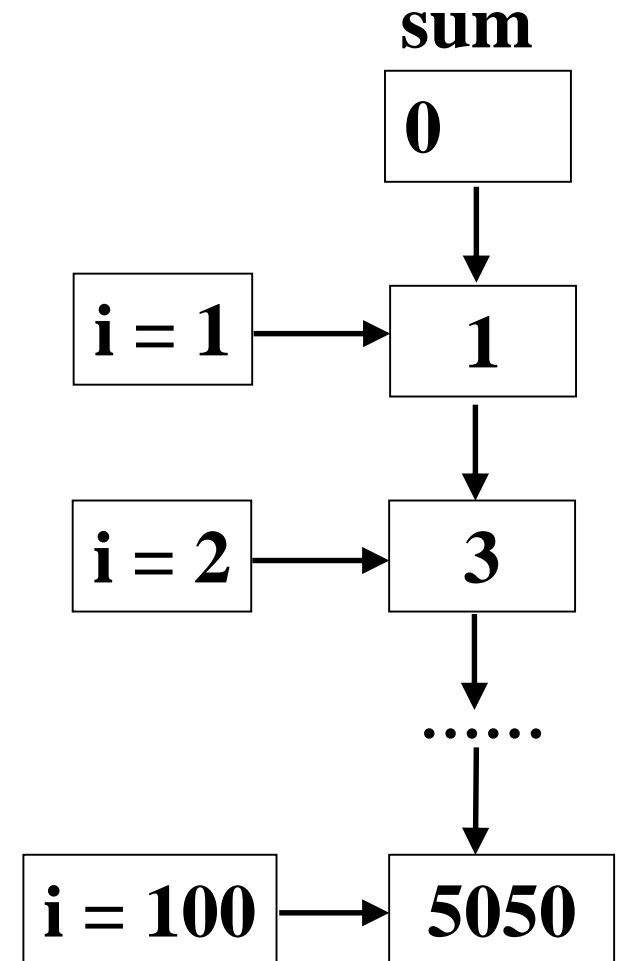
基本思路:

把问题抽象为一种统一的形式，然后采用循环语句来重复地计算。

用一个变量`sum`来保存总和，对于1、2、3、...、100 中的每一个整数 `i`，依次把它加入到`sum`当中，即 `sum = sum + i`。

累加编程模式

```
#include <stdio.h>
int main()
{
    int i, sum;
    sum = 0;
    for ( i = 1; i <= 100; i++)
    {
        sum = sum + i;
    }
    printf ("sum = %d", sum);
    return 0;
}
```



累加编程模式

思考题：在程序中做如下的修改

1. 将原来的`for(i=1; i<=100; i++)`

修改为`for(i=1; i<=100; i=i+2)`

问：这是在计算哪些整数的和？

2. 将原来的`for(i=1; i<=100; i++)`

修改为`for(i=1; i<=100000; i++)`

问：程序执行后能够得出正确结果吗？

为什么？

`sum = 705082704`

用long long整型作为累加和

```
#include <stdio.h>
int main()
{
    long long int i, sum;
    sum = 0;
    for ( i = 1; i <= 100000; i++)
    {
        sum = sum + i;
    }
    printf ("sum = %lld", sum);
    return 0;
}
```

sum = 5000050000

还有无其他做法？

用实数作为累加和

```
#include <stdio.h>
int main()
{
    int i;
    double sum = 0;
    for ( i = 1; i <= 100000; i++)
    {
        sum = sum + i;
    }
    printf ("sum = %.0lf", sum);
    return 0;
}
```

sum = 5000050000

星期几的问题

编写一个程序，首先输入一个整数N，表示某月的第一天是星期N，再输入两个整数from和to，代表该月的某两天（ $\text{from} \leq \text{to}$ ），然后程序将统计出，从from到to之间，有多少天也是星期N。例如，假设10月1日是星期三，以下是程序的一次运行过程：

讨论！

```
3
1 31
从1号到31号，共有5个星期3。
```

1 2 3 4 5 6 7 8 9 10
11 12 13 14 15 16 17 18 19 20
21 22 23 24 25 26 27 28 29 30
31

1. 星期N只出现在红色圆圈内;
2. 需要统计from到to之间, 有多少个红色圆圈;
3. 统计圆圈个数, 就是累加编程模式。

```
#include <stdio.h>
int main()
{
    int N, from, to;
    int i, num;

    scanf("%d %d %d", &N, &from, &to);
    num = 0;
    for(i = from; i <= to; i++)
    {
        if(i % 7 == 1) num ++;
    }
    printf("从%d号到%d号, 共有%d个星期%d。 \n",
           from, to, num, N);
    return 0;
}
```

补充说明: `for` 循环变量

- 可有多多个表达式 (用逗号连接)

`for(i=0, j=10; i<j; i++, j--)`

- 表达式可省略 (但要保留分号)

`i=0;`

`for(; i<10; i++)`

`for(; condition;)` // 等同于 `while` 语句 (不推荐)

- 表达式中可声明变量:

`for(int i=0; i<10; i++)`

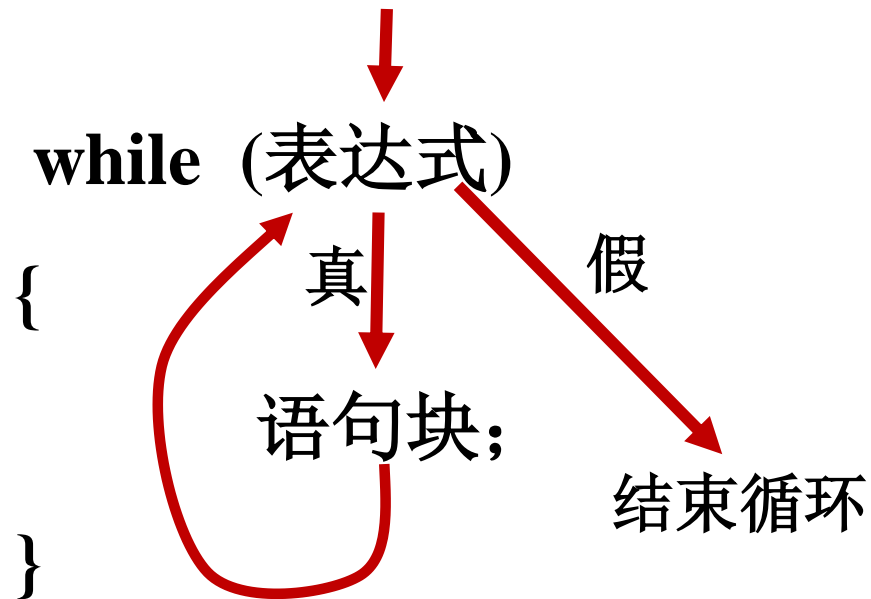
// `i` 的作用域仅是 `for` 循环的循环体内 (不推荐)

Lecture 4: Program Looping

- The `for` Statement
- **The `while` Statement**
- The `do-while` Statement
- The `break` and `continue` Statement
- Examples

while 语句

一般形式:



```
while( condition )
{
    statements;
}
```

- 先条件测试再执行
- 事件驱动（比较计数驱动）

```
/* Frog Feeding */
while ( see_fly() == TRUE )
{
    flick_tongue();
    clamp_mouth();
}
```

脆弱的输入方式

```
char choice;  
  
printf("你是否想借书? (y/n)");  
scanf("%c", &choice);  
.....
```



如果用户输入的是y/n以外的字符呢？

健壮的输入方式

```
char choice;  
  
printf("你是否想借书? (y/n)");  
scanf("%c", &choice);  
while(choice != 'y' && choice != 'n')  
{  
    printf("你是否想借书? (y/n)");  
    scanf("%c", &choice);  
}  
.....
```


分析下列程序的输出结果

```
int x = 1234, y = 0;
while(x != 0)
{
    y = y * 10 + x % 10;
    x = x / 10;
}
printf("%d\n", y);
```

4321

Double Your Money

```
/* 假定你有1000元，每年投资收益为5%  
，经过多少年后，你的钱会翻番？ */  
my_money = 1000.0;  
n = 0;  
while ( my_money < 2000.0 )  
{  
    my_money = my_money * 1.05;  
    n = n + 1;  
}  
printf( "我的钱将会在%d年内翻番", n);
```

1. 多少年? (15)
2. 翻2番(4000)呢?
3. 是否可用for语句实现?

Example

• 反向打印一个十进制的整数

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

```
int main ()
```

```
{
```

```
    int number, right_digit;
```

```
    printf ("请输入一个十进制整数.\n");
```

```
    scanf ("%d", &number);
```

```
    while ( number != 0 ) {
```

```
        right_digit = number % 10;
```

```
        printf ("%d", right_digit);
```

```
        number = number / 10;
```

```
    }
```

```
    return 0;
```

```
}
```



1234
4321

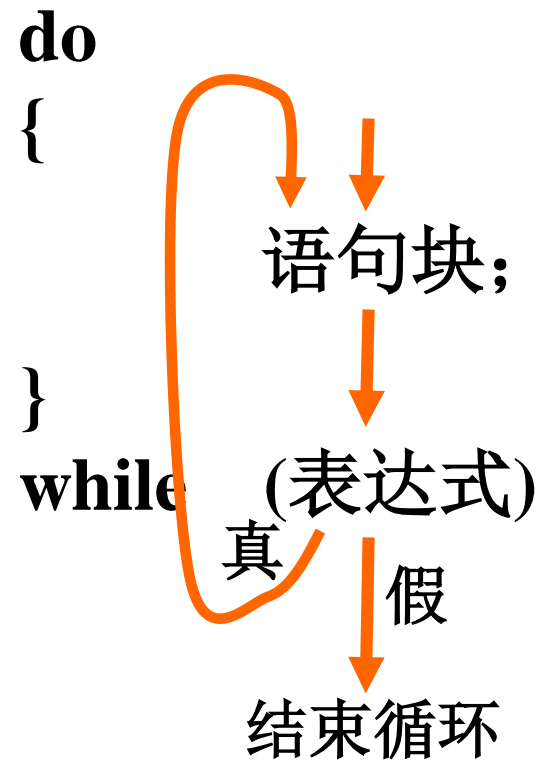
1. 如果输入是0呢？

Lecture 4: Program Looping

- The `for` Statement
- The `while` Statement
- **The `do-while` Statement**
- The `break` and `continue` Statement
- Examples

do-while 语句

一般形式:



```
do
{
    statements;
} while( condition );
```

- 先执行再条件测试
- 事件驱动
- 循环体至少执行一次
- 注意:分号结束

```
/* Frog Feeding */
do
{
    chew_and_mash();
    swallow();
} while (mouth_empty() == FALSE);
```

```
int password;  
  
do  
{  
    printf("请输入密码: ");  
    scanf("%d", &password);  
}while(password != PASSWORD);
```



Example

```
#include <stdio.h>
int main ()
{
    int number, right_digit;
    printf ("请输入一个十进制整数.\n");
    scanf ("%d", &number);
    do
    {
        right_digit = number % 10;
        printf ("%d", right_digit);
        number = number / 10;
    } while ( number != 0 );
    return 0;
}
```

1234
4321

0
0

三种循环语句比较

- **事件驱动**: while语句和do-while语句**结构形式**基本一样, 由“循环条件+循环体”两部分组成;
- **计数驱动**: for语句由“三个表达式+循环体”四部分组成。
- while语句和do-while语句循环**变量初始化**是在它们之前进行的; 而for语句则是在表达式1中实现循环变量初始化。
- 三个语句中都有**循环趋于结束的语句**(如i++), 保证了循环能够正常结束: 在while和do-while语句中, 循环趋于结束的语句都包含在循环体内; for语句则是由表达式3来实现。

循环语句的适用性

- 通常情况下，这三个语句是**通用的**，都能完成循环的功能，如：求 $1+2+3+\cdots+100$ 的和
- for循环语句主要适用于**循环次数已知**的情况，如：求 $n!$
- while语句和do-while语句则适用于**循环次数未知**，但**循环条件确定**的情况
- 与while语句相比，do-while语句更适合于**先执行循环**，后判断循环条件的情况。如：从键盘输入一个整数，统计该数是几位数
- **三个循环语句各有特点，在程序设计的过程中，要从实际的问题出发，选择合适的循环语句，从而提高程序中循环的效率**

Example: while vs for

```
#include <stdio.h>
int main (void)
{
    int n = 1;
    while ( n<= 5 ) {
        printf ("%i\n", n);
        n++;
    }
    return 0;
}
```

```
#include <stdio.h>
int main (void)
{
    int n;
    for ( n=1; n<=5; n++ )
    {
        printf ("%i\n", n);
    }
    return 0;
}
```

Lecture 4: Program Looping

- The `for` Statement
- The `while` Statement
- The `do-while` Statement
- The **`break`** and **`continue`** Statement
- Examples

break 和 continue 语句

break语句的功能:

1. 用来跳出**switch**结构;
2. 用来从循环体内跳出循环体, 即提前结束循环, 接着执行循环下面的语句。

一般形式为: **break**;

```
int days;
double food, fat;
...
for(days=155; days > 0; days--)
{
    work_all_day();
    if(food+fat < 0.01)
        break;
    sleep_all_night();
}
die_quietly();
```

健壮的输入方式 (while again)

```
char choice;  
  
while( 1 )  
{  
    printf("你是否想借书? (y/n)");  
    scanf("%c", &choice);  
    if ( choice == 'y' || choice == 'n' ) break;  
}  
.....
```

循环条件永真模式

统计正数和负数的个数

问题描述:

输入一组整数，当输入 0 时表示输入结束，
然后统计这组整数中正数和负数的个数。

12 -4 -25 47 0



问题分析：

1. “输入一组整数，直到输入0”，可用**循环条件为永真的编程模式**；
适合不断地处理一组数据，直到满足某个退出条件的问题。

2. “统计正数和负数的个数”，即为**累加编程模式**。

```
#include <stdio.h>
int main( )
{
    int Value, PosNum, NegNum;
    printf("统计一组整数中正数和负数的个数，输入0结束。\\n");
    PosNum = 0;
    NegNum = 0;
    while (1)
    {
        scanf("%d", &Value);
        if(Value == 0) break;
        else if(Value < 0) NegNum++;
        else PosNum++;
    }
    printf("正数个数: %d, 负数个数: %d", PosNum, NegNum);
    return 0;
}
```

12 -4 -25 47 0
正数个数: 2,
负数个数: 2

作业习题(3)——字符统计

- **问题描述**：编写一个程序，不断输入字符直到遇到‘#’为止。然后输出读入的**空格’，换行符’\n’和其它字符个数**。
- **问题分析**：不断地处理输入的字符（并统计XX的个数），直到遇到‘#’退出。
- **解决方法**：采用循环条件为永真的编程模式

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

```
int main()
```

```
{
```

```
    int spaceNum = 0, newlineNum = 0, otherNum = 0;
```

```
    char c;
```

```
    while(1)
```

```
    {
```

```
        scanf("%c", &c);
```

```
        if(c == '#')
```

```
            break;
```

```
        else if(c == ' ')
```

```
            spaceNum++;
```

```
        else if(c == '\n')
```

```
            newlineNum++;
```

```
        else
```

```
            otherNum++;
```

```
    }
```

```
    printf("%d %d %d\n", spaceNum, newlineNum, otherNum);
```

```
    return 0;
```

```
}
```

Chapter 3. # 209

while (...)

{

....

while (...)

{

.....

break;

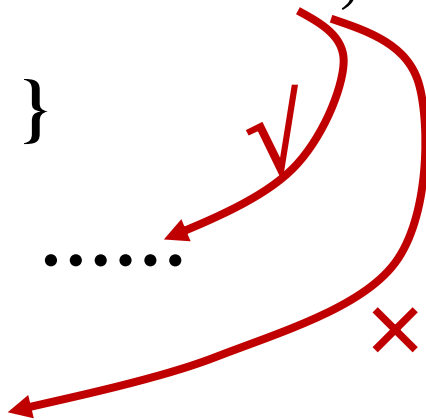
}

.....

}

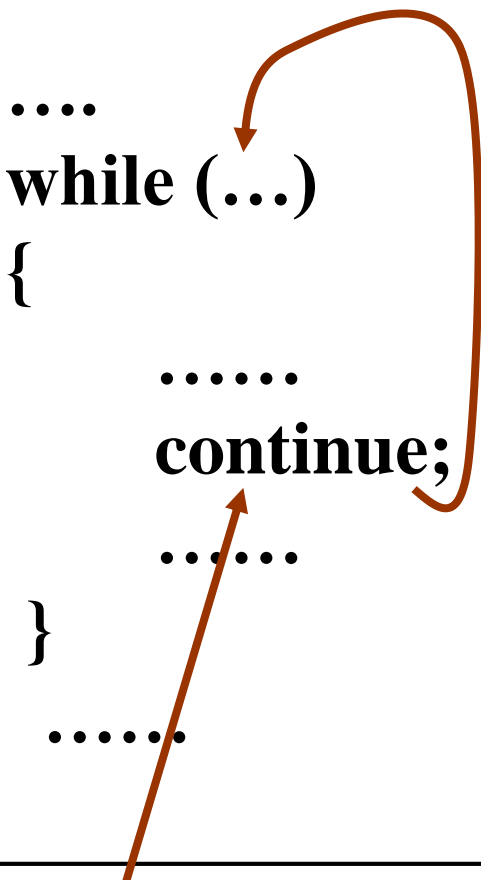
.....

跳出最近的循环。



continue语句

```
while (...)  
{  
    ....  
    while (...)  
    {  
        .....  
        continue;  
        .....  
    }  
    .....  
}
```



结束本次循环，即跳过循环体中尚未执行的语句，直接回到循环条件的判别。

分析下列程序的输出结果

```
sum = 0;
for (i = 0; i < 10; ++i)
{
    if (i == 4) break;
    if (i == 2) continue;
    sum += i;
}
printf("%d", sum);
```

4

sum = 0 + 1 + 3

Lecture 4: Program Looping

- The `for` Statement
- The `while` Statement
- The `do-while` Statement
- The `break` and `continue` Statement
- **Examples**

程序举例

例1：求最大公约数 (Greatest Common Divisor, GCD)

设有两个整数 x, y ，既能整除 x 又能整除 y 的正整数称为 x 和 y 的公约数，其中最大者称为 x 和 y 的最大公约数。



问题分析

1. 什么叫整除？ k 能整除 x ，这意味着什么？
2. 如果 k 是 x 和 y 的公约数，这又意味着什么？
3. 如何求最大公约数？

若整数“ a ”除以不等于0的整数“ b ”，商为整数，且余数为零，我们就说 **a 能被 b 整除**（或说 **b 能整除 a** ），记作 $b|a$ ，读作“ b 整除 a ”或“ a 能被 b 整除”。

如：2整除100，或100能被2整除

算法设计(1): 逐一测试法

1. 找到 x 和 y 当中的较小者，不妨设为 y ；
2. 从 y 开始，从大到小地对每一个比 y 小的整数进行测试，如果该数既能整除 x 又能整除 y ，则说明它就是最大公约数；
3. 是一种枚举法（或穷举法）；
4. 适宜for循环语句实现。

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

```
int main()
```

```
{
```

```
    int x, y, temp, k;
```

```
    printf("请输入两个整数: ");
```

```
    scanf("%d %d", &x, &y);
```

```
    if(x < y)
```

```
    {
```

```
        temp = x; x = y; y = temp; //交换x和y的值
```

```
    }
```

```
    for(k = y; k > 0; k--)
```

```
    {
```

```
        if((x % k == 0) && (y % k == 0)) break;
```

```
    }
```

```
    printf("最大公约数为: %d\n", k);
```

```
    return 0;
```

```
}
```

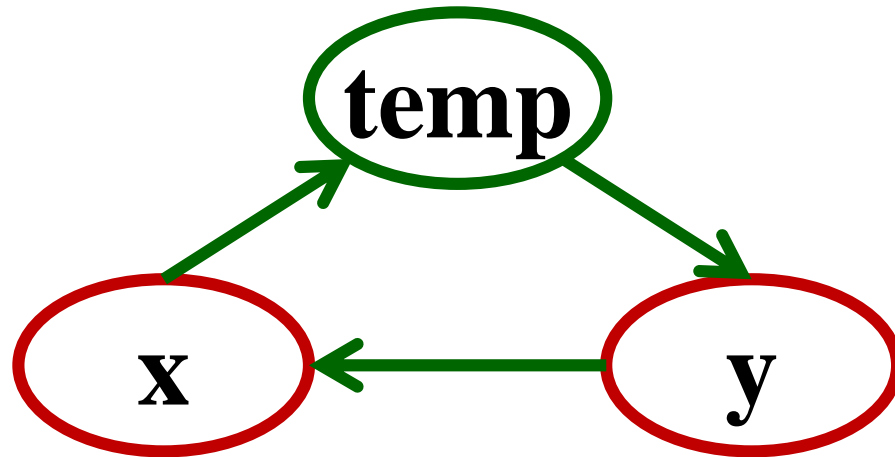
请输入两个整数: 6 8
最大公约数为: 2

程序有无问题?

if (y==0)?

交换两个变量的值

“交换两个变量的值” 是一种基本的编程模式



思考：

1. 交换两个变量的值而不用临时变量？
2. 编写一个子函数交换两个变量的值？ `swap(x, y)`？
3. 交换3个或多个变量的值？

算法设计(2): 辗转相除法

1. 两个整数的最大公约数等于其中较小的数和两数的相除余数的最大公约数;
2. 是一种递归的思想:

$$\text{gcd}(a, b) = \text{gcd}(b, a \% b)$$

3. 适宜while循环语句实现

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

```
int main ()
```

```
{
```

```
    int x, y, temp;
```

```
    printf("请输入两个整数: ");
```

```
    scanf("%d %d", &x, &y);
```

```
    while ( y != 0 ) {
```

```
        temp = x % y;
```

```
        x = y;
```

```
        y = temp;
```

```
    }
```

```
    printf("最大公约数为: %d\n", x);
```

```
    return 0;
```

```
}
```

请输入两个整数: 6 8
最大公约数为: 2

1. 没有x和y的比较?
2. 是否适宜采用do-while实现?
3. 两种算法的复杂度?
4. 递归实现?

程序设计的鲁棒性

- 逐一测试法

```
请输入两个整数: 0 6  
最大公约数为: 0
```

- 辗转相除法

```
请输入两个整数: 0 6  
最大公约数为: 6
```


函数调用的实现

```
#include <stdio.h>
int Gcd(int a, int b);
int main ()
{
    int x, y, res;
    printf("请输入两个整数: ");
    scanf("%d %d", &x, &y);

    res = Gcd(x, y);

    printf("最大公约数为: %d\n", res);
    return 0;
}
```

递归方法

// 方法1: 迭代形式

```
int Gcd(int a, int b)
{
    int temp;
    while(b != 0)
    {
        temp = a % b;
        a = b;
        b = temp;
    }
    return a;
}
```

//方法2: 递归实现

```
int Gcd(int a, int b)
{
    if(b == 0)
        return a;
    return Gcd(b, a % b);
}
```

例2：实数解码

问题描述：

编码规则

- 用4个字节（整数）来描述一个实数，低28位为**有效数字**，高4位为**指数部分**；
- 如果指数部分大于8，则进行**除法**；
- 如果指数部分小于8，则进行**乘法**；

输入一个编码后的数据（十六进制整数），输出经过解码以后的实数。

样例

16进制编码数据	有效数字	指数	解码数据
0x 0 476909D	0x476909D	0x 0	74879133
0x 1 19FDEC7	0x19FDEC7	0x 1	272544710
0x 9 502786A	0x502786A	0x 9	8404797.8
0x A 668424F	0x668424F	0x A	1074960.15
0x B 0000F32	0x0000F32	0x B	3.89

问题分析

1. 如何把28位的“有效数字”和4位的“指数部分”剥离出来;
2. 如何将一个数乘以 10^N 或除以 10^N ?

位运算符 (Bitwise Operators)

1. **&** **按位与 (bitwise AND)**: 将两个运算量的每一个位进行逻辑与操作;
2. **|** **按位或 (bitwise inclusive OR)**: 将两个运算量的每一个位进行逻辑或操作;
3. **^** **按位异或 (bitwise exclusive OR)**: 相同为0, 不同为1;
4. **~** **取反(one's complement (unary))**: 对一个二进制数按位取反。

只适用于字符型和整数型数据, 即
(signed/unsigned) char, short, int, long

0011 (整数3)
& 0101 (整数5)

0001 (整数1)

0011 (整数3)
| 0101 (整数5)

0111 (整数7)

00111001 (整数57)
^ 00101010 (整数42)

00010011 (整数19)

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

```
int main (void)
```

```
{
```

```
    int i = 0xAB00;
```

```
    int j = 0xABCD;
```

```
    int n;
```

```
    n = i & j;
```

```
    printf("%0#X\n", n);
```

```
    n = i | j;
```

```
    printf("%0#X\n", n);
```

```
    n = i ^ j;
```

```
    printf("%0#X\n", n);
```

```
    return 0;
```

```
}
```

0XAB00
0XABCD
0XCD


```

int x, count = 0;
int num = 9999; // 0x270F
x = num;
while(x)
{
    count ++;
    x = x & (x - 1);
}
printf("%d", count);

```

输出结果是: 8

1

0x270f

0x270e

0x270e

2

0x270e

0x270d

0x270c

3

0x270c

0x270b

0x2708

4

0x2708

0x2707

0x2700

5

0x2700

0x26ff

0x2600

6

0x2600

0x25ff

0x2400

7

0x2400

0x23ff

0x2000

8

0x2000

0x1fff

0x0000

位运算符补充说明

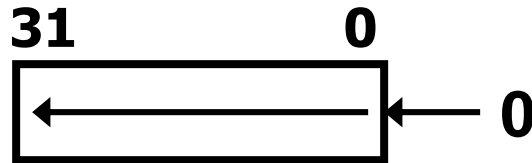
1. **&** : 常用于屏蔽某些二进制位;
2. **|** : 常用于将某些二进制位置为1;
3. **^** : 当两个操作数的对应位不相同位置为1 ;

&、**|** 应与 **&&**、**||** 区分, 如:

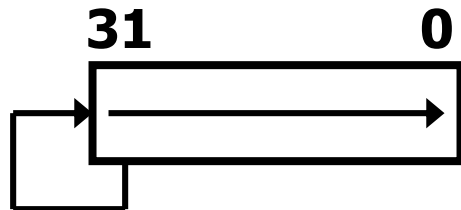
```
int n, x = 1, y = 2;  
n = x & y;           // n等于0  
n = x && y;          // n等于1
```

移位运算符 (shift operators)

1. 左移运算 (\ll) : 左移后, 低位补0, 高位舍弃, 如 $a \ll 2$;



2. 右移运算 (\gg) : 右移后, 低位舍弃, 高位补“符号位”。



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

```
int main (void)
```

```
{
```

```
    int n, x = 7; // 二进制: x = 0111
```

```
    n = x << 2; // 二进制: n = 11100
```

```
    printf("%d\n",n);
```

```
    n = x >> 2; // 二进制: n = 0001
```

```
    printf("%d\n",n);
```

```
    return 0;
```

```
}
```

28

1

移位运算符补充

- 移位时，移出的位数全部丢弃，移出的空位补入的数**与左移还是右移有关**。
- 如果是左移(<<)，则规定补入的数全部是0；
- 如果是右移(>>)，还与被移位的数据是否带**符号**有关。
 - 若是**unsigned**数，则高位补入的数全部为0；
 - 若是**signed**数，则高位补入的数全部等于原数的最左端位上的原数(即**原符号位**)。

无符号整型变量移位示例

```
#include <stdio.h>
int main()
{
    unsigned short b, a = 0111; // 对应二进制数为: 0 000 000 001 001 001
    b = a;
    printf("%#o\n", b);
    b = a << 3; // 结果为01110, 对应二进制数为: 0 000 001 001 001 000
    printf("%#o\n", b);
    b = a >> 4; // 结果为04, 对应二进制数为: 0 000 000 000 000 100
    printf("%#o\n", b);
    return 0;
}
```

0111
01110
04

有符号整型变量移位示例

```
#include <stdio.h>
int main()
{
    // 1. 负数的补码等于它的绝对值的二进制形式，按位取反再加1
    // 2. 有符号整数的最高位被用作符号位。符号位: 0代表正数; 1代表负数
    // Ref: P.21, 《计算机语言与程序设计》 湛卫军
    short int b, a = -4; // 对应二进制数为: 1 111 111 111 100
    b = a;
    printf("%d\n", b);
    b = a << 3; // 结果为-32, 对应二进制数为: 1 111 111 111 100 000
    printf("%d\n", b);
    b = a >> 4; // 结果为-1, 对应二进制数为: 1 111 111 111 111 111
    printf("%d\n", b);
    b = a << 15; // 结果为0, 对应二进制数为: 0 000 000 000 000 000
    printf("%d\n", b);
    return 0;
}
```

-4
-32
-1
0

负数的二进制表示方法

- 在计算机中，负数以**补码**形式表达；
- 负数的补码等于它的**绝对值**的二进制形式，**按位取反，再加1**；
- 以 **(short) -4** 为例，求其**补码**
 - **1000 0000 0000 0100** //**源码**
 - **1111 1111 1111 1011** //**反码**
 - **1111 1111 1111 1100** //**补码 (最后一位加1)**

注意

- **正数**：原码=反码=补码（该数的二进制）；
- **负数**：反码=原码取反(符号位为1)
- **负数**：补码=反码+1(符号位为1)
 - 即负数的补码为对该数的原码（除符号位外）各位取反，然后在最后一位加1。

C 语言提供的按位运算符 (Bitwise Operators):

➤ & | ^ ~

➤ << >>

➤ &= |= ^= >>= <<=

Why补码?

- **符号整数 (以1 byte为例)**
 - 最高位是符号位: 0代表正数; 1代表负数
 - 如果不用补码: 1000 0001 (-1), 0000 0001 (+1)
 - 那么就有两个零: +0 和 -0

```
unsigned int seed = -1; // seed = 1? = 4294967295?
```

问题分析（2）

1. 对一个无符号整数，如0x**1**9FDEC7，如何把28位“有效数字”（0x**1**9FDEC7）剥离出来？

value & 0x0FFFFFFF

2. 如何把4位“指数部分” (0x1)剥离出来？

value >> 28

3. 如何将一个数乘以 10^N 或除以 10^N ？

```
#include <stdio.h>
int main()
{
    unsigned int value;
    int N, i;
    double result;

    scanf("%X", &value);
    N = value >> 28;
    result = value & 0xFFFFFFFF;
    if(N > 8)
    {
        N = N - 8;
        for(i = 1; i <= N; i++) result /= 10;
    }
    else
    {
        for(i = 1; i <= N; i++) result *= 10;
    }
    printf("%.2f\n", result);
    return 0;
}
```

输入: 0x119FDEC7
N = 0x1
value = 0x19FDEC7
result = 272544710.00

测试...

位运算符应用示例

(1) **&** 常用于二进制**取位操作**.

例1: **x & 1** 取二进制的**最末位**.

例2: **x & 1** 可以用来判断一个整数的奇偶.

二进制的**最末位为0**表示该数为偶数, **最末位为1**表示该数为奇数.

例3: **x & 7** 取二进制**末三位**.

1101 101 → 101

例4: **x & (x+1)** 把二进制右边连续的**1**变成**0**.

10010 1111 → 10010 0000

(2) | 常用于二进制特定定位上的无条件赋值.

例1: $(x | 1)$ 把二进制最末位强行变成1.

$10110\ 0 \rightarrow 10110\ 1$

例2: $(x | 1 - 1)$ 把二进制最末位变成0.

$10110\ 1 \rightarrow 10110\ 0$

例3: $x | (x+1)$ 把二进制右起第一个0变成1.

$100101111 \rightarrow 100111111$

例4: $x | (x-1)$ 把二进制右边连续的0变成1.

$11011\ 000 \rightarrow 11011\ 111$

(2) ^ 常用于对二进制的特定一位进行取反操作.

例1: $(a \wedge b) \wedge b$ 等于 a

可对一个数字加密和解密, 如:

```
int x = 2351;
```

```
int keys = 20131022;
```

```
加密: x ^ keys 等于 20129249;
```

```
解密: 20129249 ^ keys 等于 2351;
```

(3) ~ 把二进制中的0和1全部取反,

要格外小心, 需要注意整数类型有没有符号

(4) << 左移.

例1: $a \ll n$ 等于 $a * 2^n$

通常认为比乘法更快

(5) >> 右移.

例1: $a \gg n$ 等于 $a / 2^n$

用 >> 代替除法运算可以使程序效率大大提高

例2：谁做的好事？

问题描述：

清华附中有四位同学中的一位做了好事，不留名，表扬信来了之后，校长问这四位是谁做的好事。

- A说：不是我。
- B说：是C。
- C说：是D。
- D说：他胡说。

已知三个人说的是真话，一个人说的是假话。现在要根据这些信息，找出做了好事的人。

注：本例来自于吴文虎老师的讲义，在此表示感谢。

问题分析

1. 一个人：某个人做了好事，不知是谁；
2. 四句话：“该人是/不是某某”，如何用程序语言描述每个人说的话？
3. 三真一假：如何描述一句话是真是假？
4. 如何找到该人？

在声明变量时，让 **thisman** 表示要找的人，
定义他为字符型变量。

char thisman;

下面，把四个人说的四句话写成关系表达式。

让 “==” 的含义为 “是”

让 “!=” 的含义为 “不是”

A说：不是我。写成关系表达式为 (thisman != 'A')

B说：是C。 写成关系表达式为 (thisman == 'C')

C说：是D。 写成关系表达式为 (thisman == 'D')

D说：他胡说。写成关系表达式为 (thisman != 'D')

相应字符的ASCII码值为：

字符	'A'	'B'	'C'	'D'
ASCII码值	65	66	67	68

思路分析(1):

如何找到该人，一定是“先假设某人是做好事者，然后到每句话中去测试看有几句是真话”。“有三句是真话就确定是该人，否则换下一人再试”。比如，先假定是A同学，让thisman='A'，代入到四句话中：

A说： thisman != 'A'; 'A' != 'A' 假，值为0。

B说： thisman == 'C'; 'A' == 'C' 假，值为0。

C说： thisman == 'D'; 'A' == 'D' 假，值为0。

D说： thisman != 'D'; 'A' != 'D' 真，值为1。

显然，不是'A'做的好事（3假1真）。

思路分析(2):

再试B同学，让 `thisman = 'B'`；代入到四句话中

A说: `thisman != 'A'; 'B' != 'A'` 真，值为1。

B说: `thisman == 'C'; 'B' == 'C'` 假，值为0。

C说: `thisman == 'D'; 'B' == 'D'` 假，值为0。

D说: `thisman != 'D'; 'B' != 'D'` 真，值为1。

显然，不是'B'所为（2假2真）

思路分析(3):

再试C同学，让`thisman = 'C'`；代入到四句话中

A说: `thisman != 'A'`; `'C' != 'A'` 真，值为1。

B说: `thisman == 'C'`; `'C' == 'C'` 真，值为1。

C说: `thisman == 'D'`; `'C' == 'D'` 假，值为0。

D说: `thisman != 'D'`; `'C' != 'D'` 真，值为1。

显然，就是'C'做了好事（**3真1假**），这时，可以理出头绪，要用所谓的**枚举法**，**一个一个地去试**，四句话中有三句为真，该人即为所求。

从编写程序的角度看，实现枚举最好用循环结构。

```
// thisman 分别赋值为'A', 'B', 'C', 'D'
for(thisman = 'A'; thisman <= 'D'; thisman++)
{
    sum = (thisman != 'A')           // A的话是否为真
        + (thisman == 'C')           // B的话是否为真
        + (thisman == 'D')           // C的话是否为真
        + (thisman != 'D');           // D的话是否为真

    if (sum == 3)
    {
        printf("This man is %c\n", thisman);
        break;
    }
}
```


例4：猜数字游戏

问题描述：

电脑随机产生一个**数字不重复的四位数**，由玩家来猜，每猜一次，电脑将显示形如“*A*B”的结果，A代表位置正确数字也正确，B代表数字正确但位置不正确，例如：2A2B。总共有10次机会。

问题分析

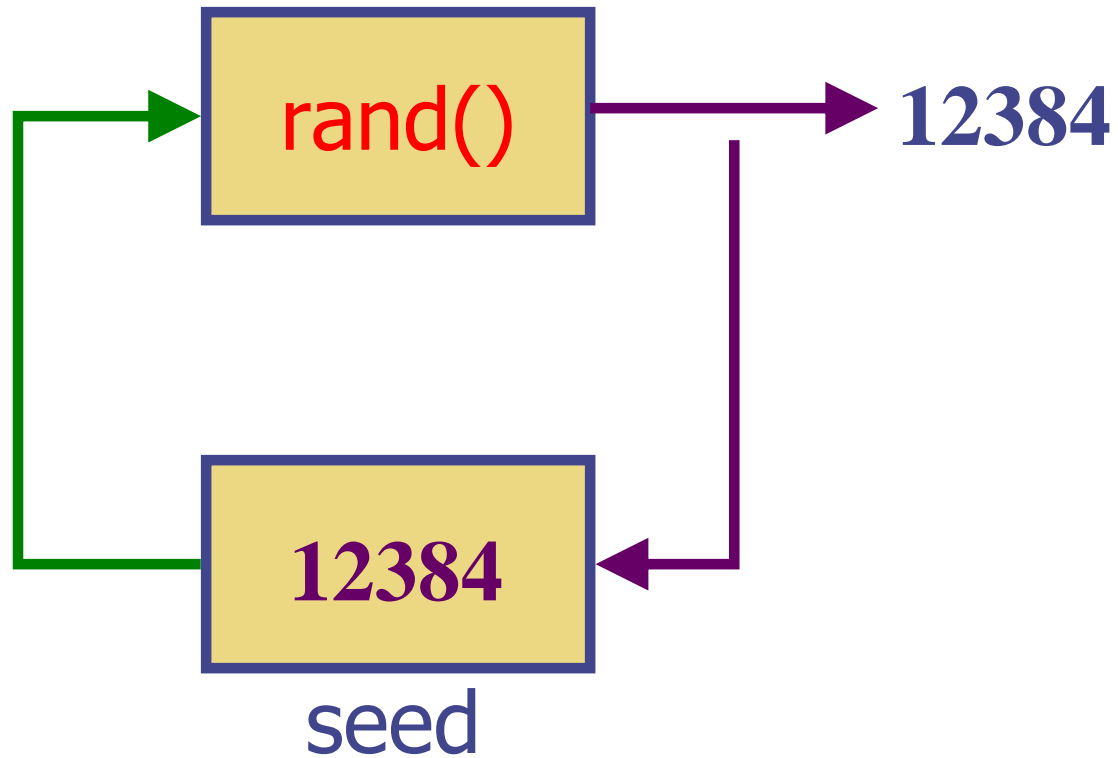
1. 如何随机产生一个数字不重复的四位数？
2. 对于玩家猜测的一个四位数，如何计算相应的A和B的数量？
3. 如何实现“总共有十次机会”？

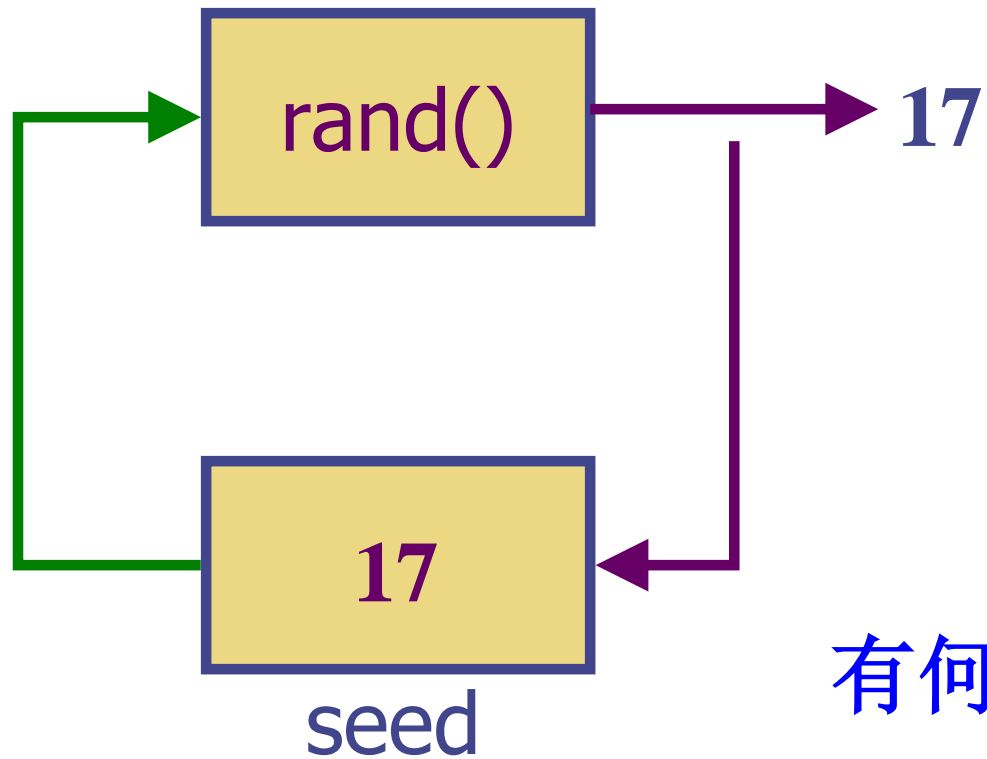
1. 随机数

1. 随机数：随机过程，如仍硬币、掷骰子，不确定事件；
2. 计算机：代码的执行是确定的；
3. 伪随机：用一个确定的过程来生成，从统计意义上类似随机数，且难以预测。

在ANSI C中生成随机数(三步)

- 1、要产生随机数需要在程序开头加入头文件
`#include <stdlib.h>`
- 2、用`srand()`函数设置种子seed
- 3、`rand()`是产生随机数的函数，
`int rand(void);`
它可生成0至`RAND_MAX`的整数，`RAND_MAX = 0x7FFF = 32767`





有何问题？

随机数序列重复

第一次

第二次

第三次

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

int main( )
{
    int i, x;

    for(i = 0; i < 10; i++)
    {
        x = rand( );
        printf("%d\n", x);
    }

    return 0;
}
```

41

18467

6334

26500

19169

15724

11478

29358

26962

24464

41

18467

6334

26500

19169

15724

11478

29358

26962

24464

41

18467

6334

26500

19169

15724

11478

29358

26962

24464

如何改进? 103

1、需要修改seed的初始值

2、通过srand()函数来实现，

```
void srand(unsigned int seed);
```

3、可以用当前时间来作为初始值

```
srand((unsigned)time(NULL));
```

如果在调用rand()之前没有调用过srand(seed), 效果将和调用了srand(1)再调用rand()一样

当前时间作为种子

第一次 第二次 第三次

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main( )
{
    int i, x;

    srand((unsigned)time( NULL ));
    for(i = 0; i < 10; i++)
    {
        x = rand( );
        printf("%d\n", x);
    }

    return 0;
}
```

28946	29393	31679
24006	21977	9222
12469	2260	22572
7994	27868	29436
18018	22091	565
16047	29721	25920
31399	31821	15080
19861	12040	27332
8191	9286	296
26646	3561	25290

如果 `srand(time(NULL));` warning C4244: 'argument' :
conversion from 'time_t' to 'unsigned int', possible loss of data

标准库rand()函数实现

- 标准库的rand函数使用线性同余算法。
- 《The C Programming Language 2nd》

```
/* rand:  return pseudo-random integer on 0..32767 */
int rand(void)
{
    next = next * 1103515245 + 12345;
    return (unsigned int)(next/65536) % 32768;
}

/* srand:  set seed for rand() */
void srand(unsigned int seed)
{
    next = seed;
}
```

问题：随机数分布在0--RAND_MAX中（32767）。如果你的算法需要的是300000多个的随机数，那么使用rand函数会产生重负次数近30次的随机数！

标准库rand()函数实现

```
void __cdecl srand (unsigned int seed)
{
    _getptd()->_holdrand = (unsigned long) seed;
}

int __cdecl rand (void)
{
    _ptiddata ptd = _getptd();

    return( ((ptd->_holdrand = ptd->_holdrand * 214013L
        + 2531011L) >> 16) & 0x7fff );
}
```

如何修改取值范围？

1、rand生成的随机数位于[0, RAND_MAX]，
如何生成任意范围[a, b]内的随机数？

2、
$$d = \text{rand}() / (\text{double})(\text{RAND_MAX} + 1);$$
$$a + d * (b - a + 1);$$

EX 1: 随机整数在[a, b)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main()
{
    int i, x, a, b;
    a = 3; b = 10;
    srand((unsigned)time(NULL));

    for(i = 0; i < 10; i++)
    {
        x = rand()%(b-a) + a;
        printf("%d\n", x);
    }
    return 0;
}
```

6	8
8	4
3	6
4	7
9	3
5	10
3	5
8	5
5	5
6	5

程序有无问题？

[a, b)?

$x = \text{rand}() \% (b - a + 1) + a;$

EX 2: 随机浮点数[0, 1)

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main()
{
    int i;
    double x;
    srand((unsigned)time(NULL));
    for(i = 0; i < 10; i++)
    {
        x = rand()/(double)(RAND_MAX+1);
        printf("%lf\n", x);
    }
    return 0;
}
```

0.088409

0.425537

0.713043

0.506104

0.003418

0.852112

0.873260

0.839996

0.153442

0.159546

如何随机生成一个数字不重复的四位数？

- 1、从10个数字中选4个构成排列；
- 2、逐一生成每一位，且与前面的各位不同；
- 3、生成一个四位的随机数，再过滤有重复者；
- 4、.....

2. 计算A和B

已知目标数和猜测数，如何计算相应的A和B的数量？

- 假设目标(Target)数为T1 T2 T3 T4，
猜测(Guess)数为G1 G2 G3 G4；
- 一种方法：T_i和G_i相比，相同则A加1；不相同则把T_i和剩余三个猜测位相比，若有相同者，则B加1。

3. 总共有十次机会

如何实现？

累加编程模式！

算法思路

1. 随机产生一个数字不重复的四位数，将它拆分为四位数字T1 T2 T3 T4；
2. 让用户输入一个数字不重复的四位数，将它拆分为四位数字G1 G2 G3 G4；
3. 计算相应的A和B的数量；
4. 如果结果为4A0B，则成功；否则，将猜测的次数加1，如果不超过10，则转第2步；否则猜测失败。

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main()
{
    int Target, T1, T2, T3, T4;
    int Guess, G1, G2, G3, G4;
    int nA, nB, NumGuess;
    double d;
```

```
// 随机产生一个数字不重复的四位数
srand((unsigned)time( NULL ));
while(1)
{
    d = rand() / (double)(RAND_MAX+1);
    Target = 1000 + (d * 9000); //1000~9999

    // 将它拆分为四位数字
    T1 = Target / 1000;
    T2 = (Target / 100) % 10;
    T3 = (Target / 10) % 10;
    T4 = Target % 10;
    if((T1 != T2) && (T1 != T3) && (T1 != T4) &&
        (T2 != T3) && (T2 != T4) && (T3 != T4))
        break;
}
```

1234
T1=1
T2=2
T3=3
T4=4

```
NumGuess = 0;
while(1)
{
    printf("输入一个不重复的四位数: ");
    scanf("%d", &Guess);

    // 将它拆分为四位数字
    G1 = Guess / 1000;
    G2 = (Guess / 100) % 10;
    G3 = (Guess / 10) % 10;
    G4 = Guess % 10;

    // 计算相应的A和B的数量
    nA = 0;  nB = 0;
    if(T1 == G1) nA++;
    else if((T1==G2) || (T1==G3) || (T1==G4)) nB++;
    if(T2 == G2) nA++;
    else if((T2==G1) || (T2==G3) || (T2==G4)) nB++;
    if(T3 == G3) nA++;
    else if((T3==G1) || (T3==G2) || (T3==G4)) nB++;
    if(T4 == G4) nA++;
    else if((T4==G1) || (T4==G2) || (T4==G3)) nB++;
}
```

```
// 判断结果
if(nA == 4 && nB == 0)
{
    printf("恭喜你猜对了! 答案是: %d\n", Target);
    break;
}
else
{
    NumGuess++;
    if(NumGuess >= 10)
    {
        printf("十次都没猜中, 你玩完了! \n");
        printf("答案是: %d\n", Target);
        break;
    }
    else
        printf("%dA %dB\n", nA, nB);
}
}
return 0;
}
```

输入一个不重复的四位数: 1234

1A 0B

输入一个不重复的四位数: 3456

0A 2B

输入一个不重复的四位数: 8571

2A 1B

输入一个不重复的四位数: 8537

恭喜你猜对了! 答案是: 8537

Lecture 4 - Summary

- **Topics covered:**
 - **for**语句
 - **while**语句
 - **do-while**语句
 - **break**和**continue**语句
 - 编程模式
 - 累加编程模式、循环条件永真模式、交换两个变量的值
 - 程序举例
 - 位运算、移位运算、随机数