



## 第四讲(Part4)

# 结构化编程

structured programming

# 提纲

## 4.1 什么是结构化程序设计

- 问题解决方法的一般性描述
- 结构化程序设计的定义
- C语言的三类控制结构
- 利用计算机求解问题：算法

## 4.2 条件语句

- 关系运算符
- 逻辑运算符
- if 与 if/else语句
- 条件运算符
- 条件语句中的嵌套
- switch语句

## 4.3 循环语句

- while语句
- for语句
- do while语句
- break与continue语句
- 逗号表达式
- goto语句

## 4.4 模块化编程

- 模块化思想
- 函数封装



新知识点

# 软件故障会导致机毁人亡！



这个自动控制下压机头的系统，名叫机动特性增强系统 (maneuvering characteristics augmentation system, MCAS) 这是波音 737 MAX 的一种操纵辅助系统。它有几个特点：

1. 发现迎角过大后，程序只相信主传感器，不与备份传感器核实。（同样的情况空客的飞机则会交给飞行员处理。）
2. 一旦相信，不通知飞行员，直接操纵机翼。
3. 飞行员手动操作后，仍旧会每五秒自动执行，让飞行员不得不与飞机较劲。
4. 程序开关非常隐蔽。

- 2018年 10 月，印尼狮航一架**波音 737 MAX 8** 喷气式客机撞向印度尼西亚的爪哇海，造成 189 名乘客和机组人员死亡。调查人员称该飞机的**飞行控制软件出现“故障”**。
- 2019年 3 月 10 日，埃塞俄比亚航空公司一架**波音 737 MAX 8** 客机在飞往肯尼亚首都内罗毕途中坠毁。飞机上载有 157 名乘客机组人员（其中有8人来自于中国）。两次飞机出事的症状非常类似，所以有理由怀疑埃航这架飞机发生了同样的**“软件故障”**。
- 在经历了两次空难之后，波音公司承诺，针对全球所有波音 737 Max 型飞机进行软件更新。

## 4.1 什么是结构化程序设计

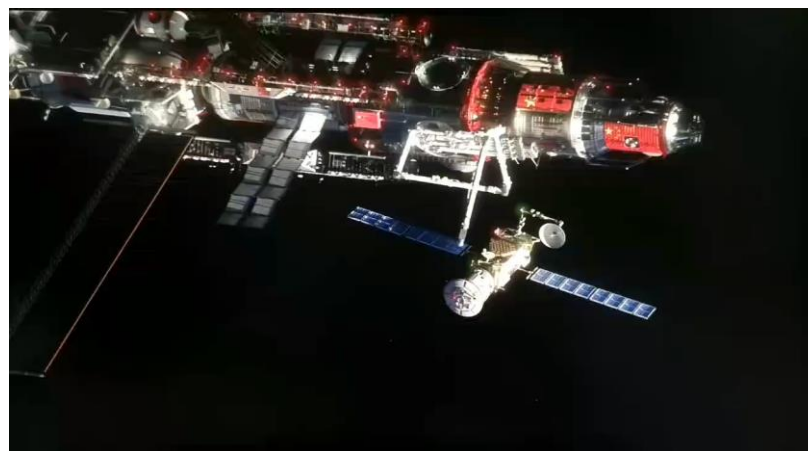
- **顺序和步骤**：当我们完成一个任务时，总是按照一定的逻辑先后顺序（order）采取行动（action）

如空间站机器人的步骤：

正确的顺序：目标定位→轨迹生成→关节驱动→剪电池板

错误的顺序：剪电池板→关节驱动→轨迹生成→目标定位

错误的顺序：目标定位→剪电池板→轨迹生成→关节驱动



- **问题分解**：当我们解决一个复杂问题时，总是将它分解成若干简单问题，并按照合理的顺序依次执行相应的操作去解决各个简单问题，最后得到复杂问题的解

如求解一个复杂数学表达式：

$$\pi = 16 * \left( \frac{1}{5} - \frac{1}{3*5^3} + \frac{1}{5*5^5} - \frac{1}{7*5^7} + \dots \right) - 4 * \left( \frac{1}{239} - \frac{1}{3*239^3} + \frac{1}{5*239^5} - \frac{1}{7*239^7} + \dots \right)$$

求解步骤：寻找通项公式 → 确定求解项数 → 分别计算各项 → 求和 → 得到答案

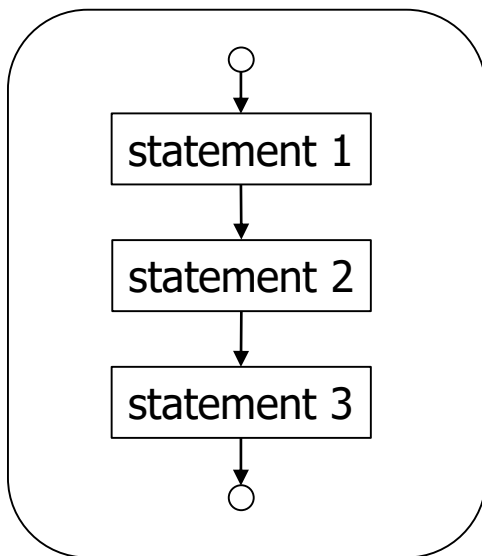
# 结构化程序设计

- 博姆-贾可皮尼理论（也称**结构化程序设计理论**），最早由Bohm和Jacopini在1960年代提出的，是软件发展的一个重要的里程碑。
- 它的主要观点是采用**自顶向下、逐步细化**的程序设计思想和**模块化的设计理念**，使用三种基本控制结构来构造程序，即：任何程序都可由**顺序、选择、重复**三种基本控制结构来构造。
- 结构化编程的理论基础：任何可计算的函数（computable function）都可以通过顺序、选择、重复三种基本控制结构及其**嵌套、组合**来表达。
- 将具有特定功能的结构化语句封装成为**子程序**，子程序间通过三种基本控制结构连接，就实现了模块化编程：复杂问题分解为若干简单问题，每个简单问题通过合理粒度的**代码封装**和**复用**各个击破，最终得到原问题的解。

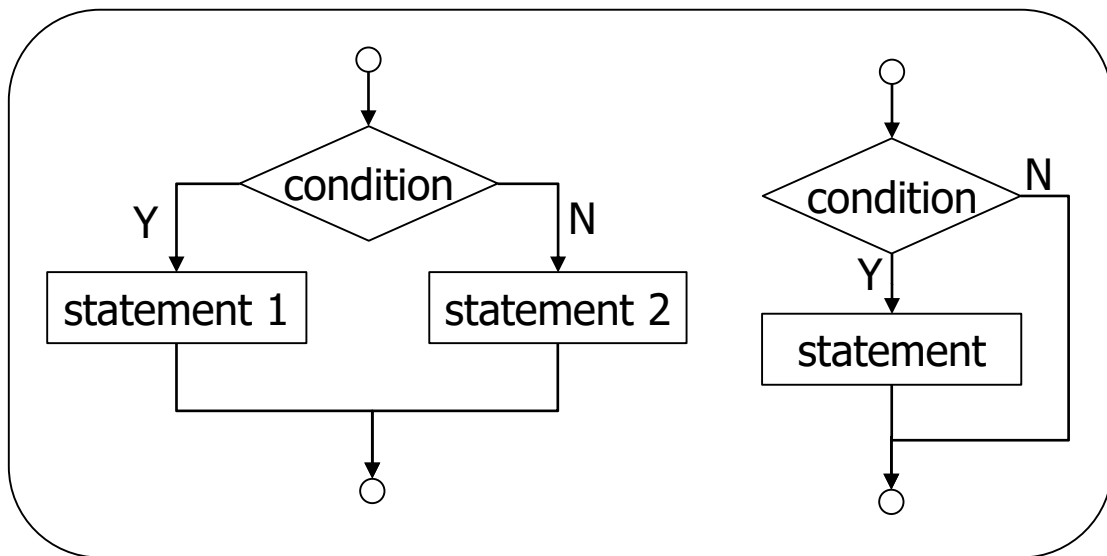
**C语言是典型的结构化程序设计语言**

# 结构化程序设计

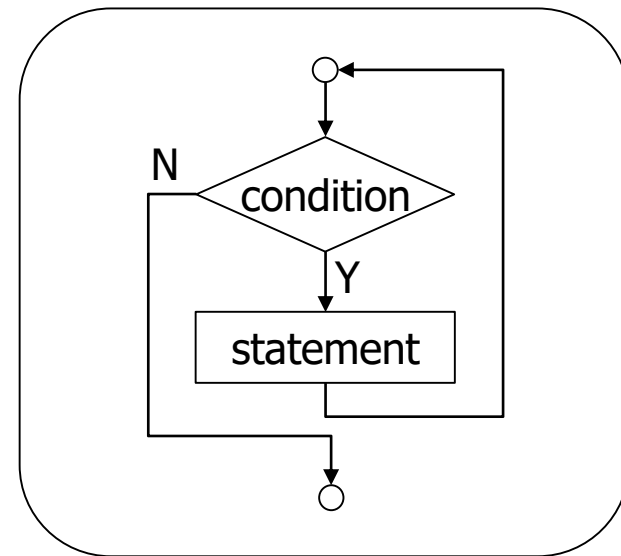
结构化程序的三种控制结构：



**顺序 (sequence)**



**选择 (selection)**



**重复 (repetition)**

# C语言的三类控制结构

- 顺序结构 (Sequence, 除非遇到选择或循环结构, 语句都是顺序执行的)
- 选择结构 (Selection) : if, if/else, switch
- 重复结构 (Repetition)
  - ◆ 又可细分为三种循环语句: while, for, do...while
- 任何C程序都可以用如上七种结构组合而成

## 例4-1: 两数相除

顺序

```
// a divided by b
#include <stdio.h>
int main()
{
    int a, b;
    double d;
    scanf("%d%d", &a, &b);
    d = (double)(a) / b;
    printf ("%d%d"= %f\n", a, b, d);
    return 0;
}
```

选择

```
#include <stdio.h>
int main()
{
    int a, b;
    double d;
    scanf("%d%d", &a, &b);
    if (b == 0)
        printf("divided by zero!\n");
    else
    {
        d = (double)(a) / b;
        printf("%d/%d = %f\n", a, b, d);
    }
    return 0;
}
```

循环

```
#include <stdio.h>
int main()
{
    int a, b;
    double d;
    while (scanf("%d%d", &a, &b) != EOF)
    {
        if (b == 0)
            printf("re-input!\n");
        else
        {
            d = (double)(a) / b;
            printf("%d/%d=%f\n", a, b, d);
        }
        return 0;
    }
}
```

技巧: 写成if(0 == b)更安全! 为什么?

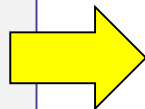
写成if(b == 0)更友好, 符合认知和习惯。

# C语言的三类控制结构：选择

例4-2：求解一元二次方程  $ax^2 + bx + c = 0$  ( $a \neq 0$ )

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

```
#include <stdio.h>
#include <math.h>
int main()
{
    double a= 1, b= 6, c = 5, r1, r2;
    double delta= b* b- 4 * a* c;
    r1= (-b + sqrt(delta)) / (2 * a);
    r2= (-b - sqrt(delta)) / (2 * a);
    printf("r1: %f, r2: %f\n", r1, r2);
    return 0;
}
```



如果  $b = 3$ ; // 此时( $b^2 - 4ac < 0$ )  
结果如何?

```
#include <stdio.h>
#include <math.h>
#define eps 1e-15
int main()
{
    double a, b, c, delta, r1, r2;
    scanf("%lf%lf%lf", &a, &b, &c); // a不为0
    delta = b * b - 4 * a * c;
    if ((delta >= -eps) && (delta <= eps))
        printf("one real root: %f\n", -b/(2 * a));
    else if (delta > 0.)
    {
        r1 = (-b + sqrt(delta)) / (2 * a);
        r2 = (-b - sqrt(delta)) / (2 * a);
        printf("two real roots: %f, %f\n", r1, r2);
    }
    else
        printf("no real roots\n");
    return 0;
}
```



# C语言的三类控制结构：重复（又叫循环）

例4-3：求斐波那契数列第n项：1, 1, 2, 3, 5, 8, 13, 21, ...

```
#include <stdio.h>
int main()
{
    int a = 1, b = 1, n, i;
    scanf("%d", &n);
    for (i = 3; i <= n; i++)
    {
        b = a + b;
        a = b - a;
    }
    printf("%d\n", b);
    return 0;
}
```

循环结构，为何从3开始？

效果等价于 `int tmp = b; b = a + b; a = tmp;`

斐波那契数列（Fibonacci sequence），又称黄金分割数列、因数学家列昂纳多·斐波那契（Leonardoda Fibonacci）以兔子繁殖为例子而引入，故又称为“兔子数列”。在数学上，斐波那契数列由递推的方法定义：

$F(1)=1, F(2)=1, F(n)=F(n-1)+F(n-2) (n \geq 3, n \in \mathbb{N}^*)$

在现代物理、准晶体结构、化学等领域，斐波纳契数列都有直接的应用，为此，美国数学会从1963年起出版了以《斐波纳契数列季刊》为名的一份数学杂志，用于专门刊载这方面的研究成果。

# 利用计算机求解问题：算法(Algorithm)

- **算法**：是用来解决某个问题的计算过程，一般都有输入和输出，包括

1. 执行的操作    2. 执行操作的顺序

- ◆ 任何算法都是由上述三种控制结构组成
- ◆ 算法执行过程不能有二义性，必须在有限时间内结束
- ◆ 算法的提出需要相应的数学模型或物理理论作为支撑
- ◆ **程序设计的核心是算法**，不懂算法的编程者就沦为coder了  
(码农：编码界的劳动密集型体力劳动者)

三类控制结构  
足以构成任意  
复杂的算法

- **程序 = 算法 + 数据结构**

计算机界有这样一种说法：如果说有一个人因为一句话而得到了图灵奖，那么这个人应该就是 Nicklaus Wirth (尼古拉斯·沃斯)，这句话就是他提出的著名公式“算法+数据结构 = 程序”。

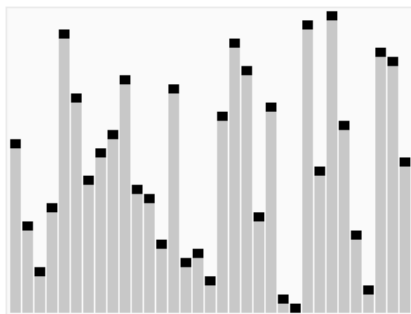
Nicklaus Wirth，1984年获得图灵奖，是Pascal之父。这个公式对计算机领域的影响程度足以类似物理学中爱因斯坦的“ $E=MC^2$ ”，这个公式展示出了程序的本质。



# 一些经典的算法思想

- 分治法

- ◆ 二路归并排序
- ◆ 快速排序
- ◆ 最大子数组



快速排序

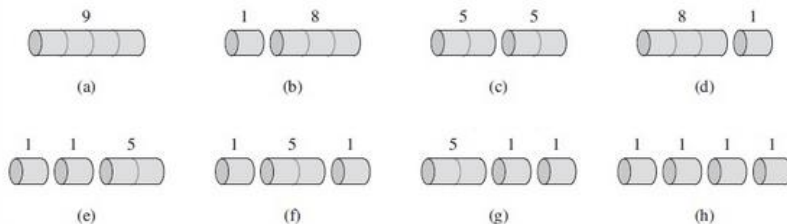
13	-3	-25	20	-3	16	-23	18	20	-7	12	-5	-22	15	-4	7
----	----	-----	----	----	----	-----	----	----	----	----	----	-----	----	----	---

最大子数组

- 动态规划

- ◆ 整数分解
- ◆ 0-1背包
- ◆ 钢条切割
- ◆ 最少硬币找零
- ◆ 矩阵链乘法

长度 $i$ (英寸)	1	2	3	4	5	6	7	8	9	10
价格 $p_i$ (美元)	1	5	8	9	10	17	17	20	24	30



钢条切割

- 贪心算法

- ◆ Kruskal最小生成树
- ◆ 活动选择

- 回溯算法

- ◆ 迷宫问题



最少硬币找零

活动 $i$	1	2	3	4	5	6	7	8	9	10	11
开始时间 $s$	1	3	0	5	3	5	6	8	8	2	12
结束时间 $f$	4	5	6	7	8	9	10	11	12	13	14

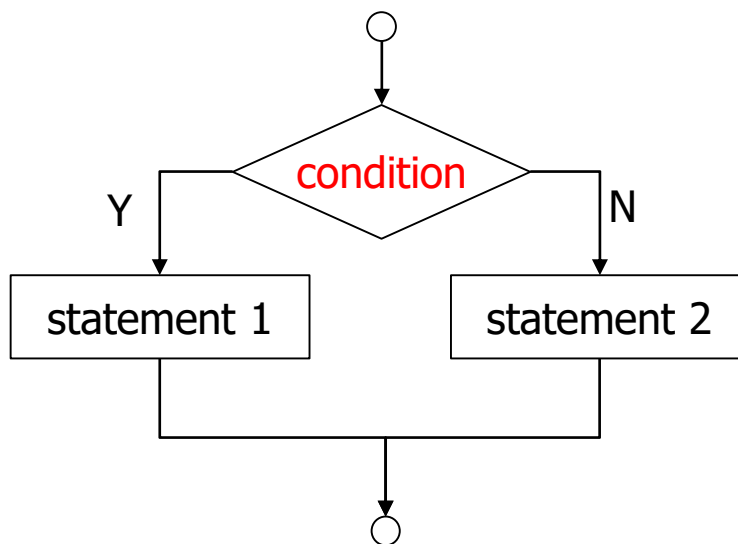
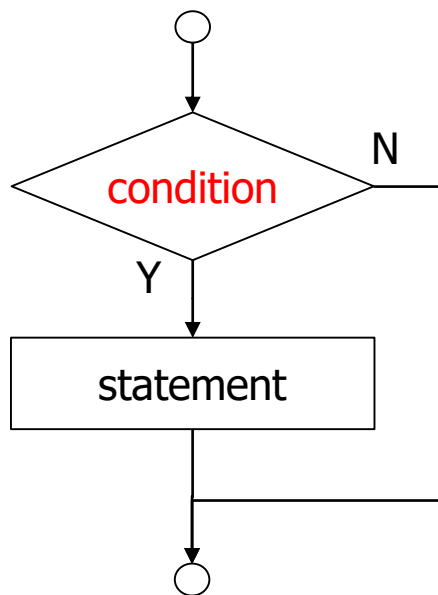
最大兼容活动集

千里之行，始于足下，C语言是理解算法、执行算法的最好工具

## 4.2 条件语句

- if 条件语句
- if/else 条件语句

任何一个表达式都可以作为if 的条件 condition



- 语法

```
if(<expression>)  
    <statement>
```

```
if(<expression>)  
    <statement1>  
else  
    <statement2>
```

1. statement (语句) 可以是单一语句, 也可以是{}括起来的复合语句块

2. 任何一个结果为逻辑值的表达式 (expression) 都可作为if 的条件 (condition)

3. 如果作为条件的表达式的值是0, 则称条件为假(No or False); 否则称条件为真(Yes or True)

# if 条件语句

- if 条件语句：用于在条件为 true (非0) 时采取相关的操作。

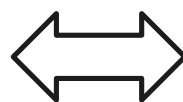
```
if( <表达式> )  
    <语句>
```

- 当<表达式>的取值非0时，均表示条件为真(Y, yes)，只有取值为 0 时，才表示条件为假。

```
if(a + b)  
    <语句>
```

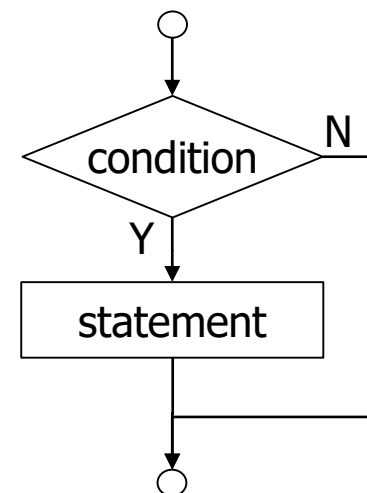
大佬一般这样用，专业、简洁

提示：if(5) 等价于 if(1)



```
if( a + b != 0 )  
    <语句>
```

一般人的做法。  
程序的可读性，可维护性更强些。  
能直观表示编程人员的实际想法。



- 类似地，

```
if(!x)
```



```
if(x == 0)
```

# if 条件语句

- **if** 条件语句：用于在条件为 true (非0) 时采取相关的操作。见如下实例。

```
...  
if(score >= 80 && score < 90)  
    printf("成绩： 良");  
...
```

```
scanf("%d%d", &a, &b);  
if(b == 0) // 另一种写法 if(!b)  
{  
    printf("The denominator is zero, Quit!");  
    return 1;  
}  
d = (double) (a) / b;
```

```
char c;  
  
if( ( ( c = getchar( ) ) != EOF) && (c >= 'a') && (c <= 'z') ) {...}
```

```
if( (a == 1) + (b == 2) + (c == 3) + (d == 4) + (e == 5) + (f == 6) == 3 )  
    printf("Guess it! U r a normal person.\n");
```

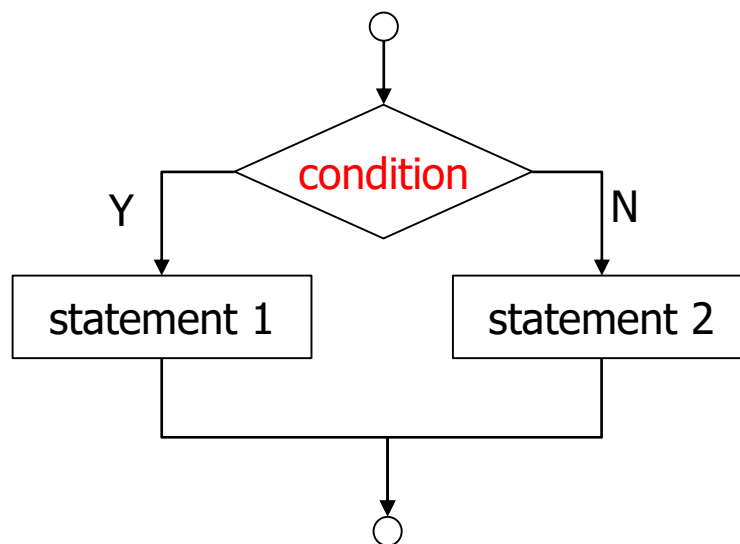
# if/else 条件语句

- if/else 条件语句：用于在条件为false (0) 或 true (非0) 时采取不同操作。

- 语法

```
if( <表达式> )  
    <语句1>  
else  
    <语句2>
```

- 流程图

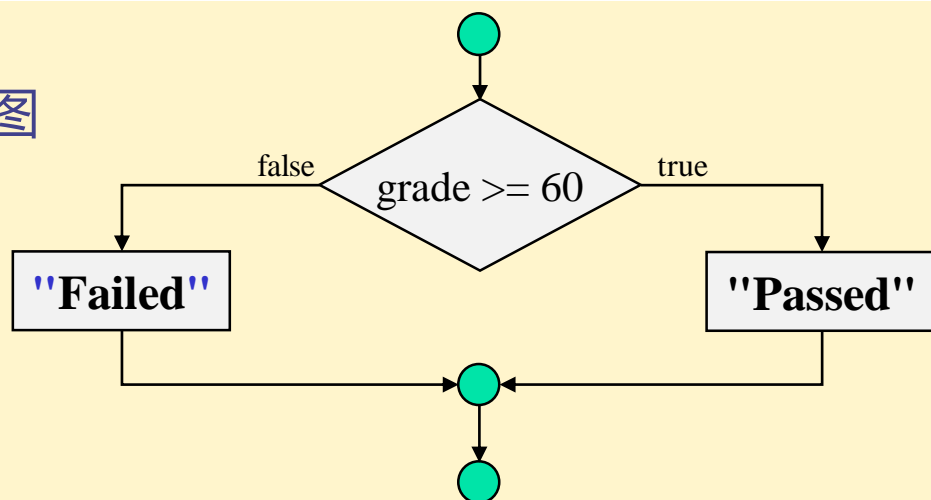


# if/else 条件语句

- if/else 条件语句：用于在条件为false (0) 或 true (非0) 时采取不同操作。

【例4-4】 假设考试成绩大于等于60分为通过，否则为不通过。

流程图



伪代码

```
if student's grade is greater than or equal to 60
    Print "Passed"
else
    Print "Failed"
```

```
#include <stdio.h>
int main()
{
    int grade;
    printf("Input the grade: ");
    scanf("%d", &grade);
    if (grade >= 60)
        printf("Passed!\n");
    else
        printf("Failed!\n");
    return 0;
}
```



# 条件运算符（条件表达式）

- 条件表达式（ `__ ? __ : __` ）与if/else结构关系密切相关

语法格式为： `expression ? expression 1 : expression 2`

含义：当expression表达式为真时，执行表达式expression1并将它的值作为整个表达式的值；否则执行表达式expression2并将它的值作为整个表达式的值

- C语言**唯一**的三元（三目）运算符，可以简化if/else描述，下列语句是等价的

```
if(grade >= 60)
    printf("Passed!\n");
else
    printf("Failed!\n");
```

⇔

```
printf( grade >= 60 ? "Passed\n" : "Failed\n" );
```

⇔

```
(grade >=60) ? printf("Passed!\n") : printf("Failed!\n") ;
```

## 条件运算符（条件表达式）

- 条件表达式（\_\_ ? \_\_ : \_\_）与if/else结构关系密切相关
- C语言唯一的三元（三目）运算符，可以简化if/else描述，下列语句是等价的

例：输出数组的LEN个数，每行输出5个数

```
for (i = 0; i < LEN; i++)
{
    if((i+1)%5 != 0)
        printf("%d ", x[i]);
    else
        printf("%d\n", x[i]);
}
```

```
for (i = 0; i < LEN; i++)
{
    printf((i+1)%5 != 0 ? "%d " : "%d\n", x[i] );
}
```

# 条件语句中的嵌套

- 嵌套if...else结构测试多个选择，将一个if...else选择放在另一个if...else选择中。
  - 例：买苹果的挑选过程。
  - 例：成绩分段打印，成绩大于等于90分时打印 A，在80到89分之间打印B，在70到79分之间打印 C，在60到69分之间时打印 D，否则打印 F。
- 伪代码

```
if grade is greater than or equal to 90
```

```
    Print "A"
```

```
else
```

```
    if grade is greater than or equal to 80
```

```
        Print "B"
```

```
    else
```

```
        if grade is greater than or equal to 70
```

```
            Print "C"
```

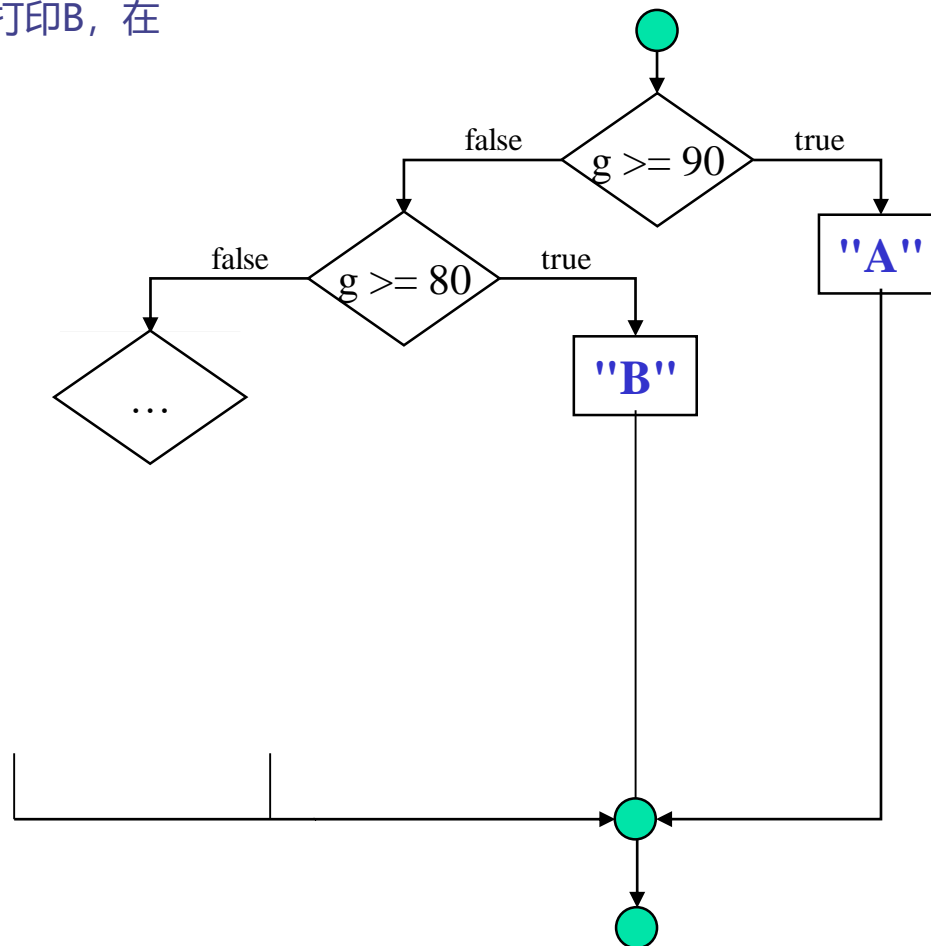
```
        else
```

```
            if grade is greater than or equal to 60
```

```
                Print "D"
```

```
            else
```

```
                Print "F"
```



- 完整的流程图？（课后练习）

# 条件语句中的嵌套

【例4-5】 成绩分段打印，成绩大于等于90分时打印 A，在80到89分之间打印B，在70到79分之间打印 C，在60到69分之间时打印 D，否则打印 F。

```
if grade is greater than or equal to 90
    Print "A"
else
    if grade is greater than or equal to 80
        Print "B"
    else
        if grade is greater than or equal to 70
            Print "C"
        else
            if grade is greater than or equal to 60
                Print "D"
            else
                Print "F"
```

```
int grade;
scanf("%d", &grade);
if(grade >= 90)
    printf("A\n");
else
    if(grade >= 80)
        printf("B\n");
    else
        if(grade >= 70)
            printf("C\n");
        else
            if(grade >= 60)
                printf("D\n");
            else
                printf("F\n");
return 0;
```

```
int grade;
scanf("%d", &grade);
if(grade >= 90)
    printf("A\n");
else if(grade >= 80)
    printf("B\n");
else if(grade >= 70)
    printf("C\n");
else if(grade >= 60)
    printf("D\n");
else
    printf("F\n");
return 0;
```

节省宽度

层次分明

# 条件语句中的嵌套

## 性能提示

- 嵌套 if...else 结构比一系列单项选择 if 结构运行速度快得多，因为它能在满足其中一个条件之后即退出。
- 在嵌套 if...else 结构中，测试条件中 true 可能性较大的应放在嵌套 if...else 结构开头，从而使嵌套 if...else 结构运行更快，比测试不常发生的情况能更早退出。

【例】若成绩好的学生编到实验班，要打印实验班的成绩等级，哪个程序更快？

当然，现在的普通个人电脑的计算能力足够强大，能计算 “ $>2^{30}$ ” 次/秒，右图程序计算速度的区别我们根本感觉不到，但养成好的思维习惯总是好的。比如，同样的逻辑循环执行  $2^{30}$  次呢？

```
grade = 88;  
if (grade >= 90)  
    printf("A\n");  
if ( grade >= 80 && grade < 90)  
    printf("B\n");  
if ( grade >= 70 && grade < 80)  
    printf("C\n");  
if ( grade >= 60 && grade < 70)  
    printf("D\n");  
if (grade < 60)  
    printf("F\n");
```



```
grade = 88;  
if (grade >= 90)  
    printf("A\n");  
else if ( grade >= 80 )  
    printf("B\n");  
else if ( grade >= 70 )  
    printf("C\n");  
else if ( grade >= 60 )  
    printf("D\n");  
else  
    printf("F\n");
```



```
grade = 88;  
if (grade < 60)  
    printf("F\n");  
else if ( grade < 70 )  
    printf("D\n");  
else if ( grade < 80 )  
    printf("C\n");  
else if ( grade < 90 )  
    printf("B\n");  
else  
    printf("A\n");
```

# else匹配问题

## 【例4-6】

在OJ排行榜上,

1. 得分**不低于**300, 且排名前500, 优秀;
2. 若得分**低于**300 (过题数少于3题), 继续努力 (Fail);
3. 得分**不低于**300, 但排名在500以后, 说明题简单, 不表扬不批评 (无输出)。

比赛排名 更新中, 上次更新于 2020-10-05 14:33:34

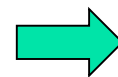
«	<	1	...	4	5	6	7	8	9	10	>	»
---	---	---	-----	---	---	---	---	---	---	----	---	---

排名	用户	得分	罚时	A 955/961	B 873/910	C 927/936	D 535/582
501		590	15:35:30	0:12:32	0:39:43(+1)	1:03:18(+2)	1:51:47(+)
502		587.5	9:01:43	0:04:49(+1)	0:16:24	1:26:20	2:16:09
503		587.5	9:40:51	0:17:57(+1)	0:28:05	0:38:05(+1)	
504		587.5	82:53:26	0:36:28(+3)	0:53:01	0:59:49	
505		581.25	16:06:25	0:12:14	1:04:22(+2)	0:19:16	5:27:51(+)
506		580	9:56:51	0:07:17	0:41:08(+3)	0:46:46	2:13:32
507		577.5	9:06:20	0:05:30	1:19:00(+1)	1:24:21(+1)	1:07:25
508		575	166:24:00	0:07:04	1:50:30(+7)	0:26:07(+1)	1:10:00(+)
509		570	10:07:12	0:04:13	0:20:26	0:33:39(+1)	
510		568.75	75:34:04	0:12:10	1:14:23(+2)	0:38:33	48:11:56(+)

```
int score = 400, rank = 900;
if(score >= 300)
    if(rank <= 500)
        printf("Excellent");
else
    printf("Fail");
```

output?

Fail



```
int score = 400, rank = 900;
if(score >= 300)
    if(rank <= 500)
        printf("Excellent");
else
    printf("Fail");
```

```
// 正确的写法
if(score >= 300)
{
    if(rank <= 500)
        printf("Excellent");
}
else
    printf("Fail");
```

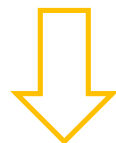
- C语言语法总是把 else 同它之前最近的 if 匹配起来
- C语言不靠缩进决定匹配关系
- C语言缩进是给人看的, 在编译器眼中多个空白字符 (回车, 空格, 换行, 制表符等) 等价于1个空白字符

# 语句块（复合语句）

```
if(studentGrade >=60)
    printf("Passed\n");
else
    printf("Failed\n");
    printf("降级，再读一年.\n");
```



```
if(studentGrade >=60)
    printf("Passed\n");
else
    printf("Failed\n");
    printf("降级，再读一年.\n");
```



应该这样写

```
if(studentGrade >=60)
{
    printf("Passed\n");
}
else
{
    printf("Failed\n");
    printf("降级，再读一年.\n");
}
```

- 选择条件（或循环）下有多条语句时，需要用大括号括起来
- C语言将大括号{}内的多条语句视为逻辑上的一个语句，称为块（block）
- 条件下只有一条语句时，建议也用大括号括起来，便于程序扩展（未来在块里添加语句）

## 条件语句实例：一元二次方程求根

【例4-7 (例2-6扩展)】输入一元二次方程  $ax^2 + bx + c = 0$  的实数系数  $a, b, c$ ，求方程的实数根？

题目分析：根据求根公式

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- 计算机中有些实数的表示不是100%精确。比如：数学上  $dt$  的值为0，但在计算机中实际可能为  $10^{-30}$ ，则程序中  $(dt > 0.0)$  为真（但我们希望为假）， $(dt == 0.0)$  为假（我们希望为真）。

怎么解决？

回忆浮点数的精度问题

**试试：** `printf("%d\n", 0.55 + 0.40 == 0.95);`



# 条件语句实例：一元二次方程求根

注意输入时的拼写、全角、中文符号等问题，如“，；（”等

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

找bug

```
int mian()
{
    double a, b, c, r1, r2, dt;
    printf("输入a b c三个实数（其中a不为0）： ");
    scanf("%lf%lf%lf", &a, &b, &c);
    dt = b*b - 4*a*c;
    if(dt > 0.0)
    {
        r1 = (-b + sqrt(dt)) / (2*a);
        r2 = (-b - sqrt(dt)) / (2*a);
        printf("方程有两个实根: %f, %f\n", r1, r2);
    }
    else if( dt == 0.0 )
        printf("方程有一个实根: %f\n", -b/(2*a));
    else
        printf("方程没有实根\n");
    return 0;
}
```

## 条件语句实例：四则运算

【例4-8】四则运算 输入一个四则运算符op和两个浮点数x和y，输出  $x \text{ op } y = r$  (r是结果) (若输入无效，则输出错误提示符)。


- 计算机中有些实数的表示不是100%精确。最好不要用实数进行 `==` 或 `!=` 的比较。
- 输入中通常应有精度条件限制。比如输入的精度保留到小数点后10位有效数字，则 `(y != 0)` 可改为 `fabs(y) >= 1e-10` (此时需要包括 `<math.h>`。

试试： `printf("%d\n", 0.55 + 0.40 == 0.95);`

再试试： `printf("%d\n", 0.50 + 0.45 == 0.95);`

```
// Four arithmetic operations
#include <stdio.h>

int main()
{
    char op; double x, y, r;
    scanf("%c%lf%lf", &op, &x, &y);
    if(op == '+')      r = x + y;
    else if(op == '-') r = x - y;
    else if(op == '*') r = x * y;
    else if(op == '/' && y != 0.0)
        r = x / y;
    else
    {
        printf("invalid expression: %6.2f %c %6.2f\n", x, op, y);
        return 1;
    }
    printf("%6.2f %c %6.2f = %12.4f\n", x, op, y, r);
    return 0;
}
```



程序的bug!

# 条件语句实例：四则运算

## 【例4-8】四则运算

### 程序的严重问题！

- 如果if(op == '-')写成if(op = '-'), 后果会怎么样?
- 把逻辑相等 if(a == b) 的比较写成了赋值 if(a = b) (等价于if(a)) , 是一个**逻辑错误** (编译不会提示) ! 而且, 还很隐蔽地修改了变量的值! 这是**双重错误**!

一种修改方式 if( '-' == op ) , 即, 比较的**常量在左边**! 编译器就会提示我们的粗心。

```
// Four arithmetic operations
#include <stdio.h>
#include <math.h>

int main()
{
    char op;  double x, y, r;
    scanf("%c%lf%lf", &op, &x, &y);
    if(op == '+')      r = x + y;
    else if(op = '-')  r = x - y;
    else if(op == '*') r = x * y;
    else if(op == '/' && fabs(y) >= 1e-10)
        r = x / y;
    else
    {
        printf("invalid expression: %6.2f %c %6.2f\n", x, op, y);
        return 1;
    }
    printf("%6.2f %c %6.2f = %12.4f\n", x, op, y, r);
    return 0;
}
```

\* 2 5  
2.00 - 5.00 = -3.0000

输入\* 2 5, 应该输出 2\*5 = 10, 怎么是 2-5 = -3 呢?

# 相等关系"=="与赋值"="运算符的进一步说明

- 赋值

```
int a;    a = 6; // 变量赋值
```

- 相等关系运算

```
int a, b, c;  
b = 5;    c = 7;  
a = (b == c); // ?
```

- 用==运算符进行赋值或用=运算符表示关系相等是个逻辑错误

```
int gender = 0; // 1表示male, 0表示female  
if ( gender = 1 ) // ??  
    printf("male\n");  
else  
    printf("female\n");
```

输出?

等价于两条语句:

**gender = 1, then if ( gender )**  
逻辑错误, 还隐蔽地修改了变量的值。

## 💡 编程提示

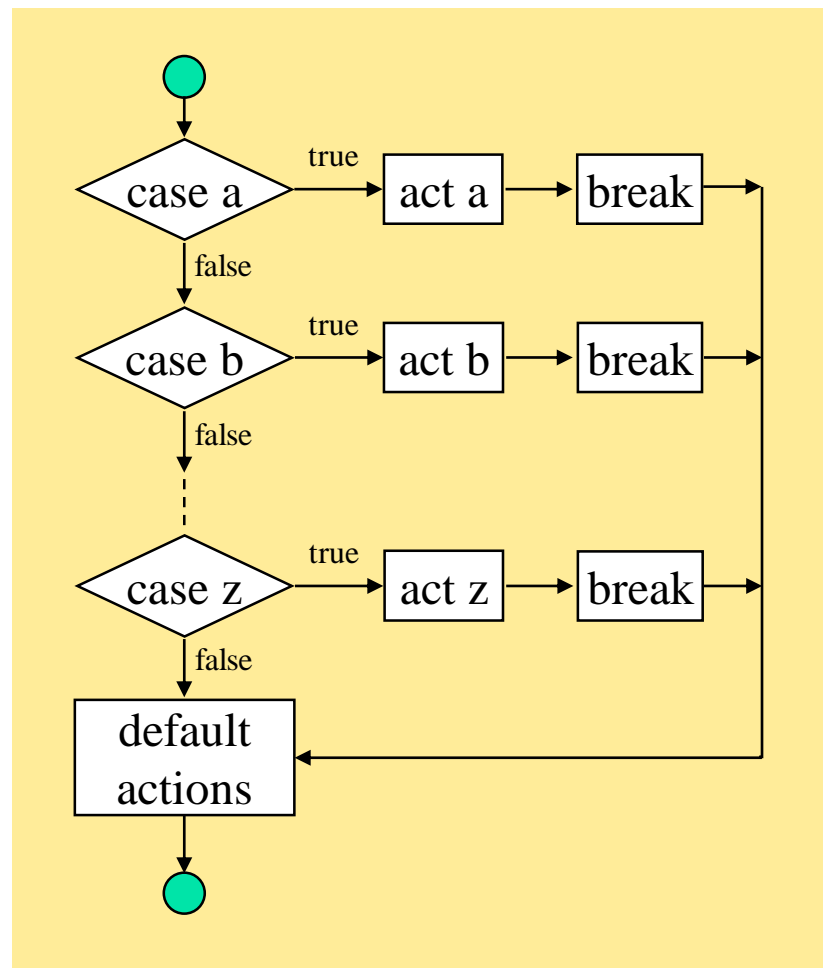
if ( 1 == gender ), 好的解决方案  
if ( 1 = gender ), 会提示语法错误  
当 "a == b" 左右两边都是变量时,  
编程人员要格外小心!

# switch语句

switch可以方便地进行多项选择，语法为：

```
switch(<expression>)  
{  
  case <value 1>: <statement 1>  
  case <value 2>: <statement 2>  
  ...  
  [default: <default statement >]  
}
```

- switch括号中的控制表达式与每个case标记比较，若与某个标记匹配，就执行相应的语句。若连续的case之间没有语句，则它们执行同样的语句。与所有case都不匹配时，执行default对应的语句（有时省掉，作为好习惯，建议保留）。break语句使程序跳出switch，case中有多个语句时，不必放在{ }中，因为会按顺序依次执行。
- switch的表达式结果必须是整型值。
- case标记值只能是整型常量或返回常量的表达式（不能是变量，例4-9）。



# switch语句实例：成绩转换

【例4-9】输入一个成绩（百分制整数），转换为等级制（A~F）

```
#include <stdio.h>
int main()
{
    int score;
    printf("Input score(0~100): ");
    scanf("%d", &score);

    if(!(score >= 0 && score <= 100))
    {
        printf("invalid input! quit!");
        return 1;
    }
    // <未完，转右边>
```

- 若连续的case之间没有语句，则它们执行同样的语句。
- 与所有case都不匹配时，执行default对应的语句。
- **break语句使程序跳出switch。**
- case中有多个语句时，不必放在{ }中。
- switch只用于测试整型值。

```
switch ( score/10 )
{
    case 10: // e.g. 100/10 is 10
    case 9:  // e.g. 96/10 is 9
        printf("A\n");
        break;
    case 8:
        printf("B\n");
        break;
    case 7:
        printf("C\n");
        break;
    case 6:
        printf("D\n");
        break;
    default:
        printf("F\n");
        break;
} // end switch

return 0;
}
```

## switch语句实例：成绩转换

【例4-10】输入一个成绩（百分制整数），90分以上为A，70~89为B，60~69为C，60以下为F。

```
...
switch ( score/10 )
{
case 10:
case 9:
    printf("A\n");
    break;
case 8:
case 7:
    printf("B\n");
    break;
case 6:
    printf("C\n");
    break;
default:
    printf("F\n");
    break;
}
...
```



**提示：**

**逻辑清晰，**

**节省空间。**

**各有优缺点，程序员根据自己的喜好来选择用哪种风格。**

```
...
switch ( score/10 )
{
case 10: case 9:
    printf("A\n");
    break;
case 8: case 7:
    printf("B\n");
    break;
case 6:
    printf("C\n");
    break;
default:
    printf("F\n");
    break;
}
...
```

# switch语句实例：四则运算

## 【例4-11】四则运算



**设计一个简单的  
计算器！**

```
...
int main()
{
    char op;  double x, y, r;
    scanf("%c%lf%lf", &op, &x, &y);
    switch(op)
    {
        case '+':    r = x + y;    break;
        case '-':    r = x - y;    break;
        case '*':    r = x * y;    break;
        case '/':
            if(fabs(y) > epsilon) // 表达的含义 0 != y
            {
                r = x / y;
                break;
            }
        default:
            printf("invalid expression: %f %c %f\n", x, op, y);
            return 1;
    }
    printf("%6.2f %c %6.2f = %12.4f\n", x, op, y, r);
    return 0;
}
```



## 4.3 循环语句

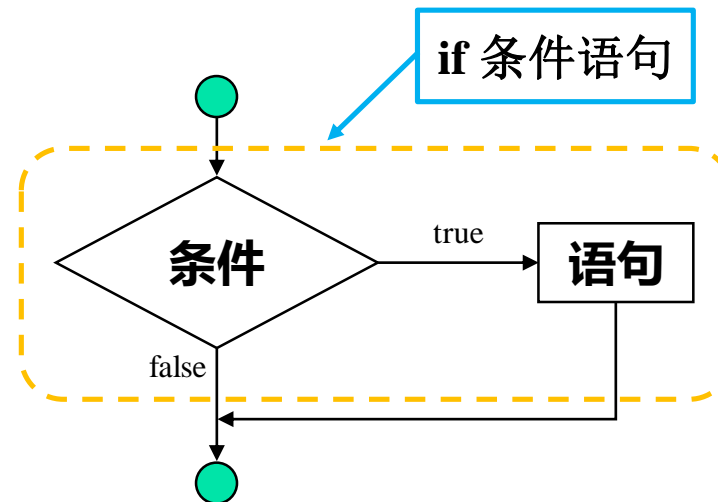
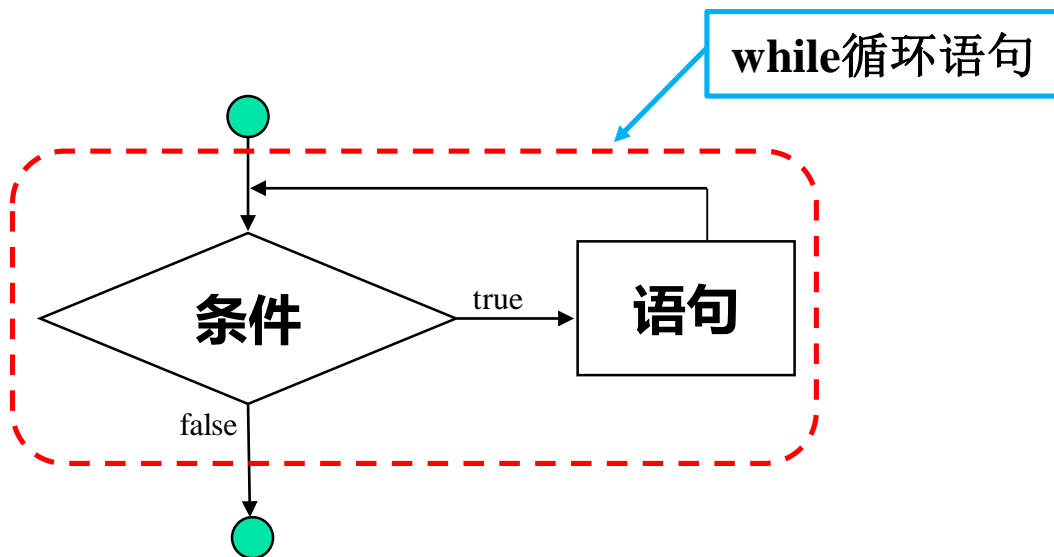
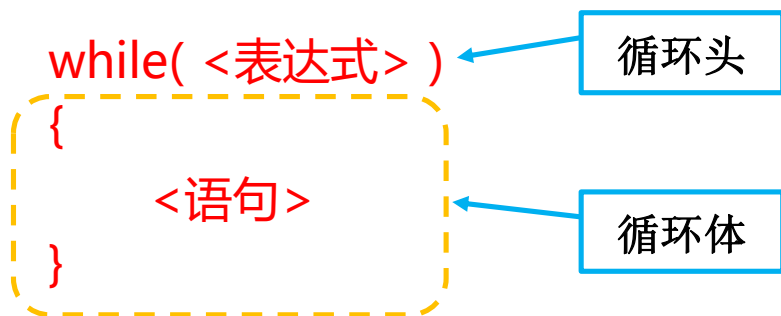
### 计算机的快速重复计算能力：天下武功，唯快不破

- 问题1：求pi （计算公式为： $4*(1-1/3+1/5-1/7 \dots + (-1)^n/(2*n+1) + \dots)$  ）  
“永不疲倦”地计加下去！
- 问题2：依次输入 $10^8$ 个同学的成绩，求平均分、最高分、最低分、.....
- 问题3：程序输入有错时给出提示，并要求再次输入，直到输入正确后开始执行相应操作
- 二分法求解方程
- 矩阵（图像）处理
- 字符串处理（在某篇文章中查找某个词）
- .....



# while语句

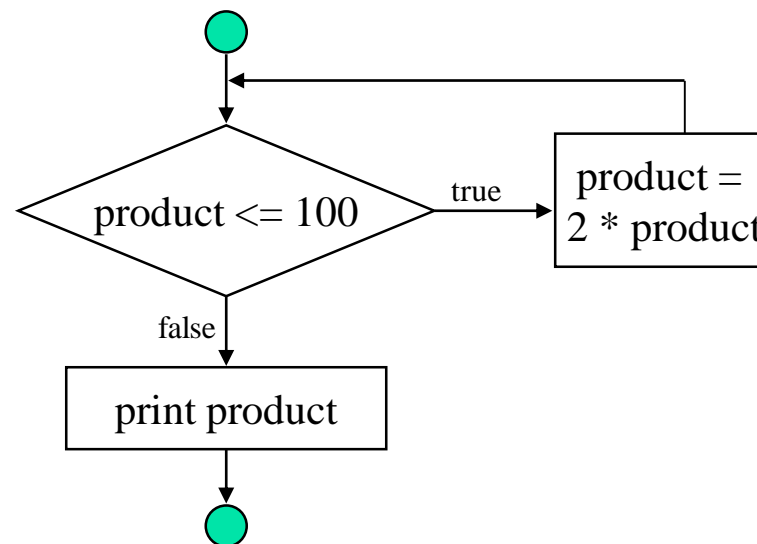
- 循环(loop)语句，也称为重复结构(repetition structure)：使程序在一定条件下重复操作。也是一种条件语句：当条件满足时重复执行循环体中的语句（与选择语句的区别？）
- 语法格式（循环体只有一条语句时可以用不用大括号）：



# while语句

【例4-12】设程序要打印第一个大于100的2的指数值，假设初始变量为 2。(  $2^7 = 128$  )

- 流程图
- C代码片段示例



```
int product = 2;
while (product <= 100)
    product = 2 * product;
printf("%d", product);
```

## 常见编程错误:

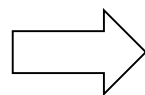
**while**循环体中，若没有使循环条件变为假的动作，或满足条件的**break**或**goto**语句；或没有外部的读入数据使得循环条件为假，将导致**无限循环（死循环）**，如

```
while(3)
{
    .....
}
```

# while语句

## while语句的特殊形式

```
while(expression) ;
```



```
while(expression)
{
    ; // do nothing
}
```

例：清空输入缓冲区直到新的一行开始

```
while((c = getchar()) != '\n');
```

# while语句的实例

**【例4-13】** 输入一行，该行的正确输入是一个百分制成绩（0~100），若输入有误则要求在新一行重新输入，直到输入正确为止。（while( )嵌套用法）

```
#include <stdio.h>
int main()
{
    int a;
    printf("\ninput a score 0~100: ");

    // 无效输入时，条件就为真
    while(scanf("%d", &a) == 0 || (a < 0 || a > 100))
    {
        printf("invalid input: %d\n", a);
        printf("input a score 0~100: ");
        while(getchar() != '\n');
    }
    printf("Good job! Valid input %d!\n", a);
    return 0;
}
```

注意：本行很重要！

scanf()从输入设备上正确读入后，才能进入下一个输入状态。


例如：输入 -12 xy，则scanf()先读入-12到a，然后getchar()读入且清除-12后面的空格、xy和换行，再开始下一行输入。

若没有该行，第一次读入-12，第二次scanf()时，会跳过%d要求的整数前的空格，读到字符 'x' 的位置，此时发现是无效读入，scanf返回0，循环，又读到 'x'，……，死循环！

# while语句的实例

**【例4-13】** 输入一行，该行的正确输入是一个百分制成绩（0~100），若输入有误则要求在新行重新输入，直到输入正确为止。（while( )嵌套用法）

```
#include <stdio.h>
int main()
{
    int a; printf("\ninput a score 0~100: ");
    // 无效输入时，条件就为真
    while(scanf("%d", &a) == 0 || (a < 0 || a > 100))
    {
        printf("invalid input: %d\n", a);
        printf("input a score 0~100: ");
        while(getchar() != '\n');
    }
    ...
}
```



```
while(getchar() != '\n')
{
    ;
}
```

注意：本行很重要！

scanf()从输入设备上正确读入后，才能进入下一个输入状态。

例如：输入 -12 xy，则scanf()先读入-12到a，然后getchar()读入且清除-12后面的空格、xy和换行，再开始下一行输入。

若没有该行，第一次读入-12，第二次scanf()时，会跳过%d要求的整数前的空格，读到字符 'x' 的位置，此时发现是无效读入，scanf返回0，循环，又读到 'x'，……，死循环！

# while语句的实例

【例4-13】输入一行，该行的正确输入是一个百分制成绩（0~100），若输入有误则要求在新行重新输入，直到输入正确为止。（while( )嵌套用法）

```
#include <stdio.h>
int main()
{
    int a;
    printf("\ninput a score 0~100: ");

    // 无效输入时，条件就为真
    while( scanf("%d", &a) == 0 || (a < 0 || a > 100))
    {
        printf("invalid input: %d\n", a);
        printf("input a score 0~100: ");
        // while(getchar() != '\n');
    }
    ...
```

如果注释这一行...

死循环

控制台显示的输入（缓冲区）：

-12 xy[enter]



- (1) 读入整数 -12 到int变量a，读入格式正确（scanf返回1），但数字不符合要求；
- (2) 下一次循环的读入，从 -12 后的非空格字符开始解析（scanf %d前的空格被跳过）；
- (3) 字符 'x' 试图读入到int变量a，但读入不正确（不匹配），读入位置不移动（位于'x'处）；
- (4) 重复第(2)和第(3)步...

# while语句的实例

**【例4-14】** 输入行数统计 输入一段文本，统计行数。

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

int main()
{
    int n=0;
    char s[100];
    while(gets(s) != NULL)
        n++;

    printf("You input %d lines.\n", n);
    return 0;
}
```

很有用的一条语句！



思考

统计文本中的单词  
数的程序如何写？



# for语句

- for结构与while结构类似。也是一种广泛使用的循环结构

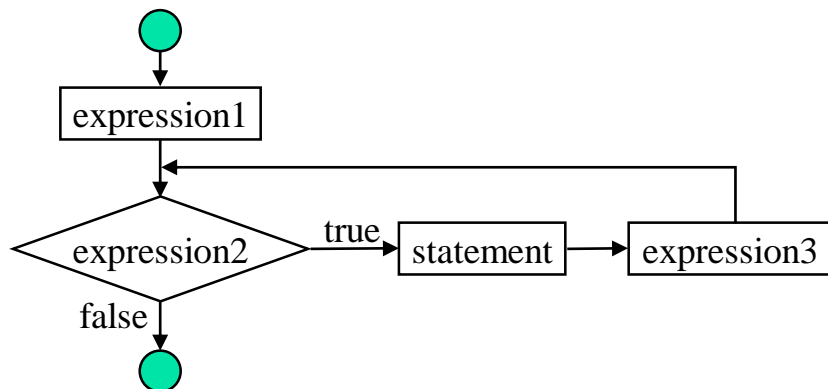
```
for( <表达式1>; <表达式2>; <表达式3> )  
{  
    <语句>  
}
```

循环头

表达式1, 2, 3均可以为空（省略），  
但分号不能省略

循环体（循环体只有一条语句时可以不用  
大括号）

```
for ( expression1; expression2; expression3 )  
{ statement  
}
```



for结构与等价的while结构

```
expression1;  
while ( expression2 )  
{  
    statement  
    expression3;  
}
```

# for语句实例

【例4-15】最大公约数 输入两个非负整数a和b，用辗转相除法求它们的最大公约数。

辗转相除算法gcd(a, b)

定理:  $\gcd(a, b) = \gcd(b, a \% b)$

算法:

1. if b is 0, gcd(a, b) is a, stop.
2. if  $a \% b$  is 0, gcd(a, b) is b, stop.
3. let  $r \leftarrow a \% b$ ,  $a \leftarrow b$ ,  $b \leftarrow r$ , go to step 2.

证明

(1) 先证明gcd(a, b)是b和a%b的公约数  
设  $g = \gcd(a, b)$ ,  $g1 = \gcd(b, a \% b)$   
则  $a = g * x$ ,  $b = g * y$ ,  
令  $a / b = m$ ,  
则  $a \% b = a - m * b$   
 $= g * x - m * g * y = g * (x - m * y)$ ,  
即  $a \% b$  能被 g 整数 (商为  $x - m * y$ ),  
即 g 也是 b 和  $a \% b$  的公约数,  
即  $g1$  能被 g 整除。

(2) 再证明 g 能被  $g1$  整除, 同理。

## for语句实例

【例4-15】最大公约数 输入两个非负整数a和b，用辗转相除法求它们的最大公约数。

**辗转相除算法gcd(a, b):**

1. if b is 0, gcd(a, b) is a, stop.
2. if  $a \% b$  is 0, gcd(a, b) is b, stop.
3. let  $r \leftarrow a \% b$ ,  $a \leftarrow b$ ,  $b \leftarrow r$ , go to step 2.

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

```
#include <stdio.h>
#define prn(x) printf("%d", x)
int main()
{
    int a, b, r;
    scanf("%d%d", &a, &b);
    if(b == 0)
    {
        prn(a);
        return 0;
    }
    for(r = a % b; r != 0; r = a % b)
    {
        a = b;
        b = r;
    }
    prn(b);
    return 0;
}
```

# for语句实例

【 gcd(a, b) = gcd(b, a%b) 】

【例4-15】最大公约数 用辗转相除法求两个整数a和b的最大公约数。

```
#include <stdio.h>
#define prn(x) printf("%d", x)
int main()
{
    int a, b, r;
    scanf("%d%d", &a, &b);
    if(b == 0)
    {
        prn(a);
        return 0;
    }
    for(r = a%b; r != 0; r = a%b)
    {
        a = b;
        b = r;
    }
    prn(b);
    return 0;
}
```

```
while((r = a%b) != 0)
{
    a = b;
    b = r;
}
```

可看出，此时用while语句使得结构更清晰些？



```
for(; (r = a%b) != 0; )
{
    a = b;
    b = r;
}
```

注意：for循环头的三个表达式都可以省略，即写成for(;;)，此时第一个第三个表达式表示不执行操作，第二个表达式表示一个非0（即为真）的条件取值。这是死循环？如何退出？

# for语句实例

## 【例4-16】三位的水仙花数 求出所有水仙花数

(水仙花数：一个三位数，其各位数字的立方之和等于该数。如153是水仙花数，因为  $153 = 1^3 + 5^3 + 3^3$ )

💡 还记得水仙花数吗？

你能发明木仙花、金仙花、火仙花、土仙花数吗？地仙花数、天仙花数？

```
#include <stdio.h>

int main()
{
    short n, n3, n2, n1;
    printf("Daffodil Number:\n");
    for(n = 100; n < 1000; n++)
    {
        n3 = n/100;
        n2 = (n%100)/10; // (n/10)%10
        n1 = n%10;

        if(n == n3*n3*n3 + n2*n2*n2 + n1*n1*n1)
            printf("%5d\n", n);
    }
    return 0;
}
```

# for语句实例

**【例4-17】复利计算** 120年前，你的祖爷爷在瑞士银行存了100,000美元（年利率0.075，假设那时候的利率比较高，后来世界发生战争，很多人忘了此事），现在银行找到你（你是该笔钱的法定继承人），你有多少钱了？（把每年有多少钱也显示出来）

分析：

$$a_1 = p + p * r = p * (1 + r)$$

$$a_2 = a_1 + a_1 * r = a_1 * (1 + r)$$

$$= p * (1 + r) * (1 + r) = p * (1 + r)^2$$

$$a_3 = a_2 + a_2 * r = a_2 * (1 + r)$$

$$= p * (1 + r)^2 * (1 + r) = p * (1 + r)^3$$

...

$$a_n = a_{n-1} * (1 + r) = p * (1 + r)^n$$

```
int i, yr, money;  
double rate, s;
```

```
printf("\n本金(>=0): ");  
scanf("%d", &money);  
printf("\n存入年限: ");  
scanf("%d", &yr);  
printf("\n年利率: ");  
scanf("%lf", &rate);
```

```
s = money;
```

```
for(i=1; i<=yr; i++)  
{  
    s = s*(1+rate);  
    printf("%4d yr later, you have: %f$\n", i, s);  
}
```

# for语句实例

【例4-18】保险理财（退休金）硕士毕业工作后，每年交10000元，交35年。退休后每年领12万退休金（不考虑通胀的原因）。开保险公司是否划算？

分析：

设2030年底， $s=0$ 元

(1) 第1年初，2031年初，存入 $m$ 元

(2) 第2年初（1年后），2032年初，有 $s=(s+m)*(1+r)$ 元【等号右边的 $s$ 是上一年的账户余额， $m$ 是本年新存入的， $r$ 是利率】

(3) 第3年初，2023年初，有 $s=(s+m)*(1+r)$ 元

...

```
int i, yr, money_year;
double rate, s;
printf("\n每年购买金额(>=0): ");
scanf("%d", &money_year);
printf("\n购买年限: ");
scanf("%d", &yr);
printf("\n年利率(0~0.2): ");
scanf("%lf", &rate);

s = 0;
for(i=1; i<=yr; i++)
{
    s = (s+money_year)*(1+rate);
    printf("after %4d years, you have: %15.2f$\n", i, s);
}
```

💡 退休后，保险公司把你的部分利息返给你，你的本金永远归保险公司了。

## for语句实例

$$16 * \frac{1}{k * 5^k} - 4 * \frac{1}{k * 239^k} > \text{eps}$$

【例4-19-1】pi的计算（至少精确到小数点后 x 位，x 是某一个给定的正整数，比如 x = 11）

$$\pi = 16 \arctan \frac{1}{5} - 4 \arctan \frac{1}{239}$$

$$\pi = 16 * \left( \frac{1}{5} - \frac{1}{3 * 5^3} + \frac{1}{5 * 5^5} - \frac{1}{7 * 5^7} + \dots \right) - 4 * \left( \frac{1}{239} - \frac{1}{3 * 239^3} + \frac{1}{5 * 239^5} - \frac{1}{7 * 239^7} + \dots \right)$$

马青公式由英国天文学教授约翰·马青(John Machin, 1686–1751)于1706年发现，他利用这个公式计算到了100位的圆周率。

分析：

pi = 16\*A - 4\*B, A的通项  $\frac{(-1)^i}{(2i+1)*5^{(2i+1)}}$ , B的通项  $\frac{(-1)^i}{(2i+1)*239^{(2i+1)}}$

与  $4 * (1 - 1/3 + 1/5 - 1/7 + \dots + (-1)^n / (2*n+1) + \dots)$  相比，哪个收敛速度快？

```
int i, j, k, sign = -1;
double n1, n2, d1, d2, d;
double s1 = 0, s2 = 0, eps = 1e-11;

for(i = 0, d = 1; d > eps; i++)
{
    sign = -sign;
    k = 2*i + 1;
    for(j = 0, n1 = n2 = 1.0; j < k; j++)
    {
        n1 *= 5;
        n2 *= 239;
    }
    d1 = 1.0 / (k*n1);
    d2 = 1.0 / (k*n2);
    s1 += d1*sign;
    s2 += d2*sign;
    d = 16*d1 - 4*d2;
}

printf("\nPi is: %.20f\n", 16*s1 - 4*s2);
```



## for语句实例

【例4-19-2】pi的计算。另一种计算方式：不合并同次项，各项单独计算。

$$\pi = 16\arctan\frac{1}{5} - 4\arctan\frac{1}{239}$$

$$\pi = 16 * \left( \frac{1}{5} - \frac{1}{3*5^3} + \frac{1}{5*5^5} - \frac{1}{7*5^7} + \dots \right) - 4 * \left( \frac{1}{239} - \frac{1}{3*239^3} + \frac{1}{5*239^5} - \frac{1}{7*239^7} + \dots \right)$$

分析：

$\pi = 16*A - 4*B$ ，A的通项  $\frac{(-1)^i}{(2i+1)*5^{(2i+1)}}$ ，B的通项  $\frac{(-1)^i}{(2i+1)*239^{(2i+1)}}$

精度条件：  $16 * \frac{1}{k*5^k} > \text{eps}/2$  且  $4 * \frac{1}{k*239^k} > \text{eps}/2$

因为  $a \leq \text{eps}/2$ ，且  $b \leq \text{eps}/2 \Rightarrow a+b \leq \text{eps}$

课后练习：如何进一步优化，把内层嵌套的for循环去掉。

思考题：两个for的逻辑结构（通项计算）相同，如何让代码更优美？

```
int i, j, k, sign = -1;
double n, d, s1 = 0, s2 = 0, eps=1e-11;
int count4 = 0, count16 = 0;
```

```
for(i = 0, d = 1; 16*d > eps/2; i++)
{
    sign = -sign;
    k = 2*i + 1;
    for(j = 0, n = 1.0; j < k; j++)
        n *= 5;
    d = 1.0 / (k*n);
    s1 += d*sign;
}
```

```
sign = -1;
for(i = 0, d = 1; 4*d > eps/2; i++)
{
    sign = -sign;
    k = 2*i + 1;
    for(j = 0, n = 1.0; j < k; j++)
        n *= 239;
    d = 1.0 / (k*n);
    s2 += d*sign;
}
```

```
printf("\nPi: %.20f\n", 16*s1 - 4*s2);
```

# 逗号表达式

## 【例】pi的计算

```
for( i = 0, d = 1 ; d > eps; i++)  
{ ... }
```

- 逗号表达式：由逗号分割的多个表达式。
- 在语法上看成是一个整体，但逻辑上是多个表达式。
- 也称为顺序表达式，子表达式按照**从左至右**的顺序求值，逗号表达式的**值等于最右边子表达式的值**，如：

```
r = (a = x, b = y, c = z);
```

≡

```
a = x;
```

```
b = y;
```

```
c = z;
```

```
r = c;
```

### 逗号表达式的使用建议：

💡 合理使用

💡 尽量少用

💡 避免滥用

# do while语句

do...while结构与while结构相似。do...while结构执行循环体之后再测试循环条件，至少执行循环体一次。

```
do
{
    <语句>
} while (<表达式>);
```

【例】辗转相除求 a 和 b 的最大公约数

```
do
{
    r = a%b;
    a = b;    b = r;
} while(r != 0)
```

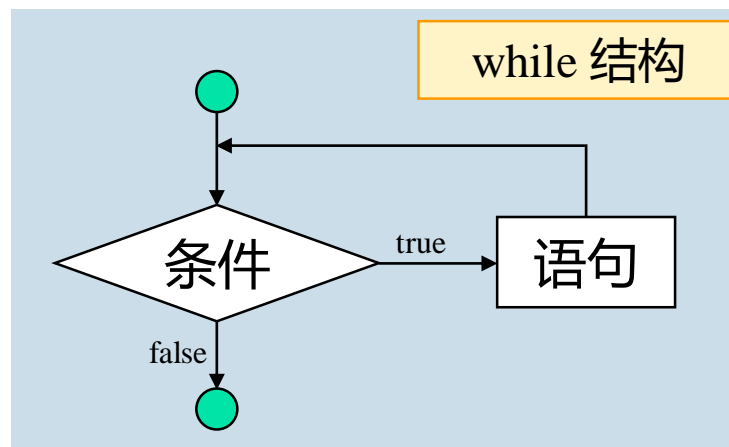
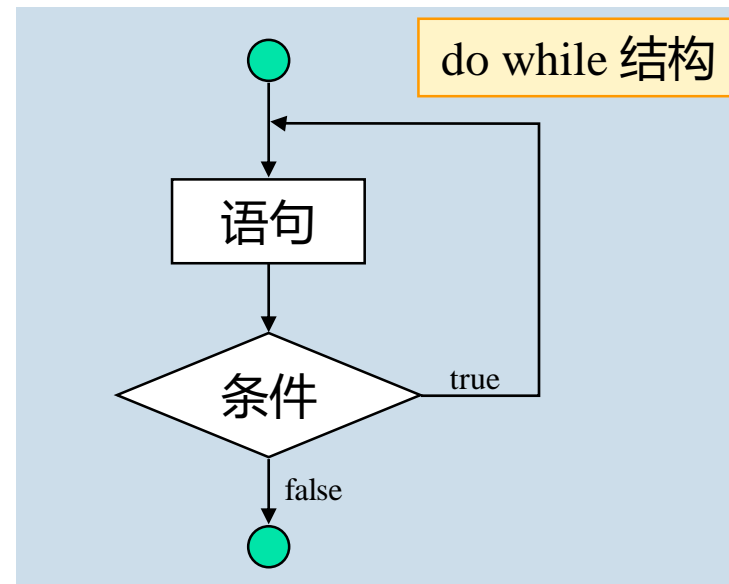


```
while((r = a%b) != 0)
{
    a = b;
    b = r;
}
```



do while 用的时候并不多，其完全能被while取代。但有时候也有其方便之处。

```
for(; (r = a%b) != 0; )
{
    a = b;    b = r;
}
```



# 循环语句的嵌套

**循环嵌套：**一个循环语句的循环体中包含一个或多个循环语句。（选择语句与循环语句也能相互嵌套）（前面已有较多实例）

**【例4-20】平行四边形图案** 输入整数 $n$ (1~20), 输出高为 $n$ 、宽度为 $2n$ 、斜边斜率为 $45^\circ$ 的平行四边形图案。如：

输入 $n$ 等于1时，输出为：

\*\*

输入 $n$ 等于2时，输出为：

\*\*\*\*

\*\*\*\*

**小知识：** putchar(c) 的用法

```
int i, j, n;
scanf("%d", &n);
for(i=0; i<n; i++)
{
    for(j=0; j<n-i-1; j++)
        putchar(' ');
    for(j=0; j<2*n; j++)
        putchar('*');
    putchar('\n');
}
```

# 循环语句的嵌套实例

**【例4-21】连续正整数** 有些正整数可表示 $n$  ( $n \geq 2$ )个连续正整数之和，如9可以表示为 $4+5$ ，或 $2+3+4$ 。输入 $x$  ( $\leq 10000$ )，找出和为 $x$ 的所有正整数序列。输出格式的规律如下所示。例如：

输入15，输出为：

$15 = 1+2+3+4+5$

$15 = 4+5+6$

$15 = 7+8$

cases: 3

输入16，输出为：

cases: 0

分析：

s1: for  $i = 1 \dots s/2$  ( // 1 2 3..  $s/2$  )

s2: for  $j = i \dots$

$\text{sum} = \sum_i j$  (when  $\text{sum} < s$ )

s3: if  $\text{sum} == s$

print  $s = \sum_i j$

go to s1, and loop  $i+1 \dots s/2$

```
int s, i, j, k, st, sum, got = 0;
scanf("%d", &s);
for (i = 1; i <= s / 2; i++)
{
    sum = i;
    for (j = i + 1; sum < s; j++)
    {
        sum += j;
        if (sum == s)
        {
            st = i; // st means start
            printf("\n%d = %d", s, st++);
            for (k = st; k <= j; k++)
                printf(" + %d", k);
            putchar('\n');
            got++;
        }
    }
}
printf("cases: %d\n", got);
```

# 循环语句的嵌套实例

**【例4-22】阶乘之和** 输入正整数 $n$  ( $n \leq 20$ ), 求  $1!+2!+\dots+n!$  (结果用long long类型)

```
#include <stdio.h>
int main()
{
    int i, j, n;
    long long r, sum = 0;
    scanf("%d", &n);
    for(i=1; i<=n; i++)
    {
        for(r=j=1; j<=i; j++)
            r *= j; // r is j!
        sum += r;
    }
    printf("sum: %lld\n", sum);
    return 0;
}
```

二重循环优化为单层循环，理论有重要意义。虽然在该例中并不能感觉到计算速度的提升（如  $10^{-6}s$  up to  $10^{-9}s$ ，我们感觉不到，但提升了1000倍！）

```
...
for(r=i=1; i<=n; i++)
{
    r *= i;    // r is i!
    sum += r;
}
...
```

请进一步测试一下，输入 $n$ 为多大时，结果就不再正确了？为什么？测试一些加入求余运算的应用。

## 循环语句的嵌套

- C语言对嵌套的层数没有限制。
- 矩阵处理通常是2~4层循环嵌套。
- 通常不建议嵌套层数太深，例如超过5层（不含）时程序的逻辑就很费解。
- 在多重循环中，一个循环长（循环次数很多），一个循环短，为了提高效率，通常把短的循环放在最外层，长的循环放在内层，以减少CPU跨切循环层的次数。

```
int s, i, j, k, st, sum, got = 0;
scanf("%d", &s);
for (i = 1; i <= s / 2; i++)
{
    sum = i;
    for (j = i + 1; sum < s; j++)
    {
        sum += j;
        if (sum == s)
        {
            st = i; // st means start
            printf("\n%d = %d", s, st++);
            for (k = st; k <= j; k++)
                printf(" + %d", k);
            putchar('\n');
            got++;
        }
    }
}
printf("cases: %d\n", got);
```

## 循环语句的比较与小结

- while, for, do while三种循环本质上相同。什么时候用哪种方式?

除了跟编程人员的喜好有关，也有一些常规的套路。

- 计数器控制循环常用for。标志控制循环常用while。do while 用得比较少。

【例】n个自然数累加

```
for(i = 0, r = 0; i <= n; i++)  
    r += i;
```

【例】求2~100间的所有偶数平方和

```
for(i = 2, s = 0; i <= 100; i += 2)  
    s += i * i;
```

用for进行计数控制  
循环更清晰。

【例】求gcd

```
for(; (r = a%b) != 0; )  
{  
    a = b;  
    b = r;  
}
```



```
while((r = a%b) != 0)  
{  
    a = b;  
    b = r;  
}
```

用while进行标志控  
制循环更自然。



## 循环语句的比较与小结

死循环？ 如何结束？ 如：

```
while(1)
```

```
{
```

```
.....
```

```
}
```

```
for( ; ; )
```

```
{
```

```
.....
```

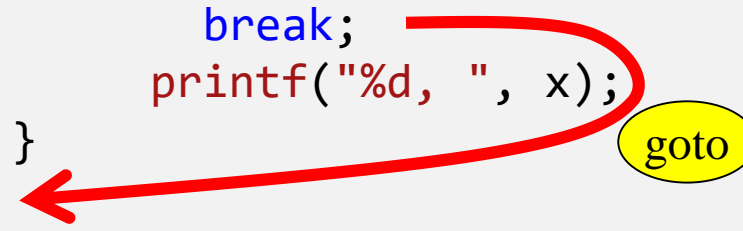
```
}
```

# break和continue：循环语句的“非常规”控制

break 和 continue 语句能快速改变程序的执行流程。

break 语句在 while, for, do/while 或 switch 结构中执行时，使程序立即退出这些结构，从而执行该结构后面的第一条语句，常用于提前从循环退出或跳过switch结构的其余部分。

```
for (x = 1; x <= 10; x++)  
{  
    if (x == 5)  
        break;  
    printf("%d, ", x);  
}  
..... // other codes
```

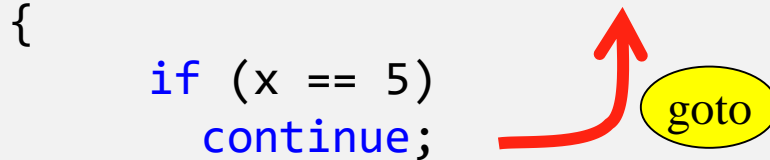


goto

输出？：

continue 语句在 while, for 或 do/while 结构中执行时跳过该结构体的其余语句，进入下一轮循环。

```
for ( x = 1; x <= 10; x++)  
{  
    if (x == 5)  
        continue;  
    printf("%d, ", x);  
}  
..... // other codes
```



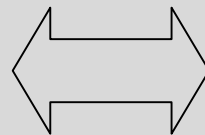
goto

输出？

# break和continue

while结构在大多数情况下可以取代for结构，但如果while结构中的递增表达式在continue语句之后，则会出现例外。

```
int x = 1;
while ( x <= 10 )
{
    printf("%d ", x);
    x++;
}
```

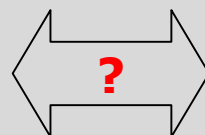


```
for ( int x = 1; x <= 10; x++ )
{
    printf("%d ", x);
}
```

输出：1 2 3 4 5 6 7 8 9 10

```
while ( x <= 10 )
{
    if( x == 5 )
        continue;
    printf("%d ", x);
    x++;
}
printf("OK");
```

输出？：



```
for (x = 1; x <= 10; x++ )
{
    if( x == 5 )
        continue;
    printf("%d ", x);
}

printf("OK");
```

输出？

## 循环的“非常规”控制实例（break 和 continue 应用）

### 【例】自然数累加 自然数1~n进行累加

```
int i = 1, r = 0;
scanf("%d", &n);
while(i <= n)
{
    r += i;
    i++;
}
```



```
int i = 1, r = 0;
scanf("%d", &n);
while(1)
{
    if(i > n)
        break;
    r += i;
    i++;
}
```



```
int i, r;
scanf("%d", &n);
for(i=1, r = 0; ; i++)
{
    if(i > n)
        break;
    r += i;
}
```

# break语句实例

【例4-23】连续正整数 ...输出最长的序列... (详细题意参见【例4-21】)

分析:

在【例4-21】的基础上改写。

第一个满足条件的序列就是最长的序列。  
为什么?

break只跳出其所在的那一层的循环 (或 switch)，有多层循环，不同层次在不同条件都涉及退出循环时，需要用多个break语句。

思考：如果不用break，程序该如何修改？

```
scanf("%d", &s);
for(i=1; i<=s/2; i++)
{
    sum = i;
    for(j=i+1; sum<s; j++)
    {
        sum += j;
        if(sum == s)
        {
            st = i; // st means start
            printf("\n%d = %d", s, st++);
            for(k=st; k<=j; k++)
                printf(" + %d", k);
            putchar('\n');
        }
    }
    if(sum == s)
        break;
}

if(sum != s)
    printf("NONE\n");
```

跳出哪个循环?

## 【例4-23】连续正整数 ...输出最长的序列... (详细题意参见【例4-21】)

用break

```
scanf("%d", &s);
for(i=1; i<=s/2; i++)
{
    sum = i;
    for(j=i+1; sum<s; j++)
    {
        sum += j;
        if(sum == s)
        {
            st = i; // st means start
            printf("\n%d = %d", s, st++);
            for(k=st; k<=j; k++)
                printf(" + %d", k);
            putchar('\n');
        }
    }
    if(sum == s)
        break;
}
if(sum != s)
    printf("NONE\n");
```

不用break, 就需要设置循环标记 (或其他办法)

```
int s, i, j, k, st, sum = 0, flag = 0;

scanf("%d", &s);
for (i = 1; i <= s / 2 && 0 == flag; i++)
{
    sum = i;
    for (j = i + 1; sum < s; j++)
    {
        sum += j;
        if (sum == s)
        {
            st = i; //st means start
            printf("\n%d = %d", s, st++);
            for (k = st; k <= j; k++)
                printf(" + %d", k);
            putchar('\n');
        }
    }
    if (sum == s)
        flag = 1; // found
}

if (sum != s)
    printf("NONE\n");
return 0;
```

```
int s, i, j, k, st, sum = 0;

scanf("%d", &s);
for (i = 1; i <= s / 2 && sum != s; i++)
{
    sum = i;
    for (j = i+1; sum < s; j++)
    {
        sum += j;
        if (sum == s)
        {
            st = i; //st means start
            printf("\n%d = %d", s, st++);
            for (k = st; k <= j; k++)
                printf(" + %d", k);
            putchar('\n');
        }
    }
}

if (sum != s)
    printf("NONE\n");
return 0;
```

## break小结

---

- 不用break，也能跳出循环和switch等控制结构，实现复杂的程序执行流程，但需要设置特殊的循环控制标记（flag）。
- 使用break，有时会使得程序很简洁、增强程序的可读性。

## continue语句实例

**【例4-24】乒乓球赛** 甲队的A、B、C与乙队的X、Y、Z比赛，已知A不与X对阵，C不与X和Z对阵，输出对阵名单。（不同的对阵在不同场馆同时进行）

根据题意，人的推理很容易，很简单：

(1) A不对阵X，则A可能对阵Y或Z

(2) C不对阵X和Z，则C只能对阵Y

(3) 由(1)和(2)，A只能对阵Z

(4) 最后，只有B和X能对阵

计算机怎么样推理的呢？



# continue语句实例

**【例4-24】乒乓球赛** 甲队的A、B、C与乙队的X、Y、Z比赛，已知A不与X对阵，C不与X和Z对阵，输出对阵名单。（不同的对阵在不同场馆同时进行）

算法：

用变量A、B、C表示选手  
A、B、C，每个变量可在  
'X'、'Y'、'Z'范围内取值，  
取值表示其对阵选手。

```
char A,B,C;
for (A='X';A<='Z';A++)
    for (B='X';B<='Z';B++)
        for (C='X';C<='Z';C++)
        {
            if (A == B || A == C || B == C)
                continue;
            if (A != 'X' && C != 'X' && C != 'Z');
                printf("A vs %c\nb vs %c\nc vs %c\n", A,B,C);
        }
return 0;
```

解释

示例：假如A = 'Y'，如果A == B，即选手A和B的值都是'Y'，意味着 A vs Y, B vs Y，一个队员Y不可能同时跟A和B比赛，这样的情况需要排除掉，在可能的情况中寻找新的组合 (continue派上用场了)

人的推理、人的智能可能更“逻辑”些。

计算机怎么样推理的呢？计算机的智能更多时候是“穷举”（穷举过程中根据规则进行筛选），这就是**计算机的AI**。计算机的计算速度很快，这种穷举很多时候比人“强大”。该例程即是一种计算机推理过程。

## continue语句实例

【例4-24】乒乓球赛 甲队的A、B、C与乙队的X、Y、Z比赛，已知A不与X对阵，C不与X和Z对阵，输出对阵名单。（不同的对阵在不同场馆同时进行）

算法：

用变量A、B、C表示选手A、B、C，每个变量可在'X'、'Y'、'Z'范围内取值，取值表示其对阵选手。

```
char A,B,C;
for (A='X';A<='Z';A++)
    for (B='X';B<='Z';B++)
        for (C='X';C<='Z';C++)
        {
            if (A == B || A == C || B == C)
                continue;
            if (A != 'X' && C != 'X' && C != 'Z');
                printf("A vs %c\nb vs %c\nc vs %c\n", A,B,C);
        }
return 0;
```

思考题：如果注释掉这条语句，程序实现什么功能？

## continue语句实例

**【例4-25\*】数据求和** 输入多组数 $\langle x_i, y_i \rangle$ ，求所有满足  $a < x_i < b$ ,  $c < y_i < d$ ，且  $x_i$  和  $y_i$  没有整除关系的  $x_i$  之和与  $y_i$  之和。（比如 $a, b, c, d$ 分别为1, 15, 1, 30）

输入

1 2

3 7

6 5

20 3

?

输出

9 12

## continue语句实例

【例4-25\*】数据求和 输入多组数 $\langle x_i, y_i \rangle$ ，求所有满足  $a < x_i < b$ ,  $c < y_i < d$ ，且  $x_i$  和  $y_i$  没有整除关系的  $x_i$  之和与  $y_i$  之和。

输入

1 2

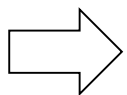
3 6

7 4

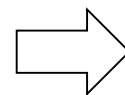
12 3

5 9

101 65



```
int a, b, c, d, x, y, sum_x = 0, sum_y = 0;
a=1; b=100; // 根据实际需要进行初始化
c=1; d=200;
while(scanf("%d%d", &x, &y) != EOF)
{
    if(x<a || x>b || y<c || y>d)
        continue;
    if(x !=0 && y%x == 0)
        continue;
    if(y !=0 && x%y == 0)
        continue;
    sum_x += x;    sum_y += y;
}
printf("sum_x = %d, sum_y = %d", sum_x, sum_y);
```



输出

?

思考题：

如果不使用continue，  
程序该如何实现？

## continue小结

---

- 不用continue，也能跳过循环后面的语句继续执行，但可能需要设置比较复杂的逻辑表达式来决定是否执行相应语句。
- 使用continue，有时会使得程序很简洁、增强程序的可读性。

# 控制语句综合实例

【例4-26\*】查询某天是星期几 输入某一天，查询该天是星期几。输入格式为yyyymmdd (如2021年11月1日应输入为20211101) 。

本例逻辑简单，  
但很综合，用了  
while, break,  
continue, if,  
switch 等三类  
控制结构的所有  
知识。

```
#include <stdio.h>
int main(){
// c: century-1, y: year, m:month, w:week, d:day
int c, y, m, w, d, longday = 1;

printf("Query what day a certain date is\n");
printf("Note: the format of the day is like 20120101\n");
printf("The input is between 101 and 99991231\n\n");

while(1) {
    printf("\nInput date (or -1 to quit): ");
    scanf("%d", &longday);

    if(-1 == longday) break;
    if(!(longday >= 101 && longday <= 99991231)) {
        printf("Wrong input format, try again!\n");
        continue;
    }

    y = longday/10000;
    m = (longday%10000)/100;
    d = longday%100;
```

```
// 接左边
// Zeller formula
if(m<3) {
    y --;
    m += 12;
}

c = y/100;
y = y%100;
w = (c/4 -2*c + y+ y/4
      + (26*(m+1))/10 + d - 1)%7;
if(w<0) w+=7;

printf("The day is: ");
switch(w)
{
case 0:
    printf("Sun\n");
    break;
case 1:
    printf("Mon\n");
    break;
```

```
// 接左边
case 2:
    printf("Tue\n");
    break;
case 3:
    printf("Wed\n");
    break;
case 4:
    printf("Thu\n");
    break;
case 5:
    printf("Fri\n");
    break;
case 6:
    printf("Sat\n");
    break;
}
return 0;
}
```

# Zeller公式:

## 计算任意一天是星期几

$$W = \left( \left\lfloor \frac{C}{4} \right\rfloor - 2C + Y + \left\lfloor \frac{Y}{4} \right\rfloor + \left\lfloor \frac{26(M+1)}{10} \right\rfloor + D - 1 \right) \bmod 7$$

Where

$W$ : the day of week. (0 = Sunday, 1 = Monday, ..., 5 = Friday, 6 = Saturday)

$C$ : the zero-based century. (=  $\lfloor \text{year}/100 \rfloor = \text{century} - 1$ )

$Y$ : the year of the century. (=  $\begin{cases} \text{year mod } 100, & M = 3, 4, \dots, 12, \\ (\text{year} - 1) \bmod 100, & M = 13, 14. \end{cases}$ )

$M$ : the month. (3 = March, 4 = April, 5 = May, ..., 14 = February)

$D$ : the day of the month.

**NOTE:** In this formula January and February are counted as **months 13 and 14 of the previous year**. E.g. if it is 2010/02/02, the formula counts the date as 2009/14/02.

```
scanf("%d", &longday); // 20211014
```

```
y = longday/10000;  
m = (longday%10000)/100;  
d = longday%100;
```

```
// Zeller formula
```

```
if(m<3)  
{  
    y--;  
    m += 12;  
}  
c = y/100;  
y = y%100;
```

```
w = (c/4 - 2*c + y + y/4 + (26*(m+1))/10 + d - 1)%7;
```

```
if(w<0) w += 7;
```

```
printf("The day is: ");  
switch(w)  
{  
case 0:  
    printf("Sun\n");  
    break;  
case 1:  
    printf("Mon\n");  
    ...  
}
```

博姆-贾可皮尼理论，也称结构化程序理论：

任何复杂的程序都可以用顺序、选择和循环  
这三种基本结构来表达。

—— C. Bohm and G. Jacopini

程序中本没有循环（while、for等），因为只用if就可以实现任何复杂的逻辑！但有人说循环好，于是循环就流行起来了！



## \*\* goto 语句

goto语句和语句标号一起使用，使程序逻辑跳转到标号位置处。

【例4-15-goto】最大公约数 用辗转相除法求整数a和b的最大公约数。

辗转相除算法gcd(a, b)的伪代码：

1. if b is 0, gcd(a, b) is a, stop.
2. if a%b is 0, gcd(a, b) is b, stop.
3. let  $r \leftarrow a \% b$ ,  $a \leftarrow b$ ,  $b \leftarrow r$ , go to step 2.

【 gcd(a, b) = gcd(b, a%b) 】

按伪代码的逻辑直接写一个程序。更好理解？

没有用循环，但实现了循环逻辑

```
#include <stdio.h>
#define prn(x) printf("%d", x)
int main()
{
    int a, b, r;
    scanf("%d%d", &a, &b);
    printf("gcd is: ");

    // step1
    if(0 == b)
    {
        prn(a);
        return 0;
    }

    step2:
    if(0 == a%b)
    {
        prn(b);
        return 0;
    }

    // step3
    r = a%b;
    a = b;
    b = r;
    goto step2;
}
```

语句标号

## \*\* goto语句 【古老的语句，强烈不建议用，但有时也挺好用？】

【例4-15-goto】最大公约数  
用辗转相除法求整数a和b的最大公约数。

辗转相除算法gcd(a, b)的伪代码：

1. if b is 0, gcd(a, b) is a, stop.
2. if a%b is 0, gcd(a, b) is b, stop.
3. let  $r \leftarrow a \% b$ ,  $a \leftarrow b$ ,  $b \leftarrow r$ , go to step 2.

【 gcd(a, b) = gcd(b, a%b) 】

为什么不建议用goto语句？用goto语句编程，有时跟写伪代码一样易写，易读！但就是太灵活了，可能容易失控！

```
#include <stdio.h>
#define prn(x) printf("%d", x)
int main()
{
    int a, b, r;
    scanf("%d%d", &a, &b);
    printf("gcd is: ");

    // step1
    if(0 == b)
    {
        prn(a); return 0;
    }

    step2:
    if(0 == a%b)
    {
        prn(b); return 0;
    }

    // step3
    r = a%b;
    a = b;
    b = r;
    goto step2;
}
```

goto版

```
#include <stdio.h>
#define prn(x) printf("%d", x)
int main()
{
    int a, b, r;
    scanf("%d%d", &a, &b);
    printf("gcd is: ");

    if(0 == b)
    {
        prn(a<0?-a:a);
        return 0;
    }
    for(; (r = a%b) != 0; )
    {
        a = b;
        b = r;
    }
    prn(b);
    return 0;
}
```

非goto版

## **\*\* goto语句 【古老的语句，强烈不建议用】**

1968年，E. W. Dijkstra (1930年5月11日~2002年8月6日，1972年获得图灵奖) 首先提出“GOTO语句是有害的”论点，向传统程序设计方法提出了挑战，从而引起了人们对程序设计方法讨论的普遍重视。



Edsger Wybe Dijkstra  
1930/5/11-2002/8/6  
1972年图灵奖获得者

- 1 提出“goto有害论”
- 2 提出信号量和PV原语
- 3 解决了“哲学家聚餐”问题
- 4 最短路径算法和银行家算法
- 5 Algol 60的设计者和实现者
- 6 THE操作系统的设计者和开发者

与Knuth并称为我们这个时代最伟大的计算机科学家的人

## \*\* goto语句 【古老的语句，强烈不建议用】

- goto语句可能使程序逻辑混乱，使程序难以理解和维护。很多语言已经取消了goto语句，但C语言仍然保留（有时候真的很方便）。
- goto语句能使得程序快速跳转，如直接跳出多重循环（不用多个break）。

```
{ ...  
    { ...  
        { ...  
            { ...  
                goto ERR;  
            }  
        }  
    }  
}  
ERR:  
...
```

```
int main()  
{  
    int n=0;  
    printf("input a string:\n");  
loop: ← 语句标号  
    if(getchar() != '\n')  
    {  
        n++;  
        goto loop;  
    }  
    printf("\n# of this line: %d\n", n);  
    return 0;  
}
```

输入输出示例

```
C:\a1ac\example\c4-31_goto.exe  
input a string:  
0123456789abcde  
  
# of this line: 15
```

## \*\* goto语句 【古老的语句，强烈不建议用】

- 建议尽量不要用goto语句！因为 for, while, do/while 能解决goto 能解决的所有问题，让程序呈现结构化特征！
- 建议不要用goto，因为这容易使程序陷入逻辑混乱、破坏结构化设计风格、带来错误或隐患（如右边的代码片段）。
- 但错误不在goto语句本身，真正的错误是程序员造成的。就像有的人因为金钱铤而走险，并非金钱的错误，是人的贪念造成的。

```
...  
goto statement;  
int a, b;    // 被goto跳过  
int sum;    // 被goto跳过  
double x;   // 被goto跳过  
...  
statement:  
...
```

# 结构化编程小结

- 程序设计、软件开发就像建筑设计、工程施工一样，既是工程，更是艺术。
- 建筑的领域已经数千年，仍然具有很大魅力，软件开发领域要比建筑领域年轻，才短短几十年，有非常广泛的发展空间，也会有许多前所未有的挑战。
- 3类控制结构（7种语句结构）的程序设计具有结构化设计思想
- 结构化的程序设计方法便于理解，有利于程序扩展。

顺序结构

选择结构

if  
if/else  
switch

重复结构

while  
do/while  
for

# 结构化编程的三种控制形式

- 结构化编程提倡简单性，Bohm和Jacopini证明，实现结构化编程只需要三种控制形式：
  - ◆ 顺序(sequence)
  - ◆ 选择(selection)
  - ◆ 重复(repetition)
- 选择结构的三种实现方法：
  - ◆ if（单项选择）；if/else（双向选择）；switch（多项选择）
- 简单的if结构即可提供任何形式的选择（即任何能用 if/else 结构和 switch 结构完成的工作，也可以组合简单 if 结构来实现）
- 重复结构的三种实现方法：
  - ◆ while语句；do/while语句；for语句
- 简单的while语句即可提供任何形式的重复（即任何能用 do/while 和 for语句完成的工作，也可以用 while语句来实现）

顺序结构

选择结构

if

if/else

switch

重复结构

while

do/while

for

# 三种选择结构的关系

简单的 **if** 语句即可提供任何形式的选择（即任何能用if/else 语句和 switch语句完成的工作，也可以组合简单的 **if**语句来实现）

选择结构

if  
if/else  
switch

```
if (expression)
    statement1
else
    statement2
```



```
if(expression)
    statement1
if(!expression)
    statement2
```

```
switch(expression)
{ case a:
    a_actions
    break;
  case b:
    b_actions
    break;
  case c:
    c_actions
    break;
  ...
}
```



```
if(expression == a)
    a_actions
if(expression == b)
    b_actions
if(expression == c)
    c_actions
...
```



# 三种循环结构的关系

简单的 **while** 语句即可提供任何形式的重复（即任何能用 do/while 和 for 语句完成的工作，也可以通过简单的 **while** 语句来实现）。

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



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

```
for(expression1; expression2; expression3)
{
    statement
}
```



```
expression1;
while(expression2)
{
    statement
    expression3;
}
```

重复结构

while  
do/while  
for

# 结构化编程的一些要素

- C程序所需的任何控制形式均可用

下列形式表示：

- ◆ 顺序
- ◆ 选择
- ◆ 重复（循环）

- 这些控制结构有两种组合方式：**嵌套**和**堆叠**

- 嵌套：任何一种控制结构逻辑上可以作为一条语句，嵌套到其他控制结构中
- 堆叠：任何一种控制结构逻辑上可以作为一条语句，堆叠到顺序结构中

- 结构化编程的特点是自顶向下、模块化、强调解决问题步骤的逻辑性
- **函数是结构化编程中实现模块化的重要形式**（下一章学习）

顺序结构

选择结构

if  
if/else  
switch

重复结构

while  
do/while  
for

## 本讲小结

---

- if, if/else, switch, while, for, do while可以完成所有复杂的逻辑，甚至只需要 if 和 while 就够了，但需要设置更复杂的逻辑表达式（甚至只需要 if 就够了）。
- 用好 switch, break, continue 能更快捷实现复杂逻辑。
- 熟悉逗号表达式，条件表达式。
- 循环语句来源于goto语句，又“抛弃”了goto语句。

## 课堂练习

下列程序的作用是什么（实现什么功能）？

```
int i = 1, r = 0, n;  
scanf("%d", &n);  
while(1)  
{  
    if(i > n)  
        break;  
    r += i;  
    i++;  
}
```

## 课堂练习

下列程序运行后，对左侧的输入，输出是什么？

输入：

2 6

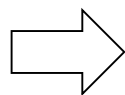
6 4

5 9

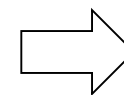
3 6

5 106

7 2



```
#include <stdio.h>
int main()
{
    int a, b, c, d, x, y, sum_x = 0, sum_y = 0;
    a=1; b=100;
    c=1; d=200;
    while(scanf("%d%d", &x, &y) != EOF)
    {
        if(x<a || x>b || y<c || y>d) continue;
        if(x !=0 && y%x == 0) continue;
        if(y !=0 && x%y == 0) continue;
        sum_x += x;    sum_y += y;
    }
    printf("%d, %d", sum_x, sum_y);
    return 0;
}
```



输出？