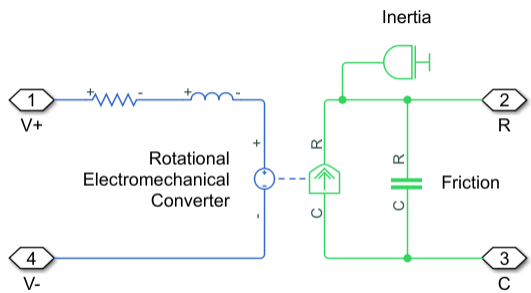
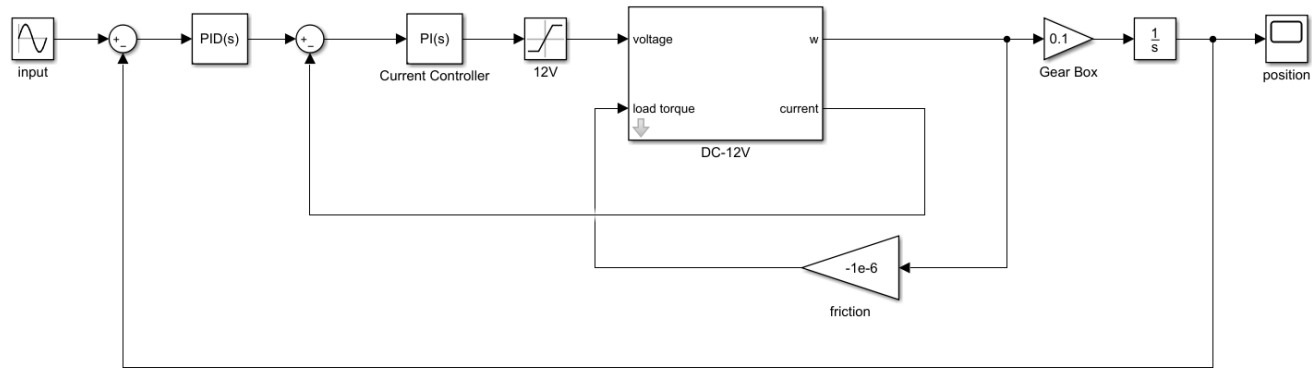


# 第四讲

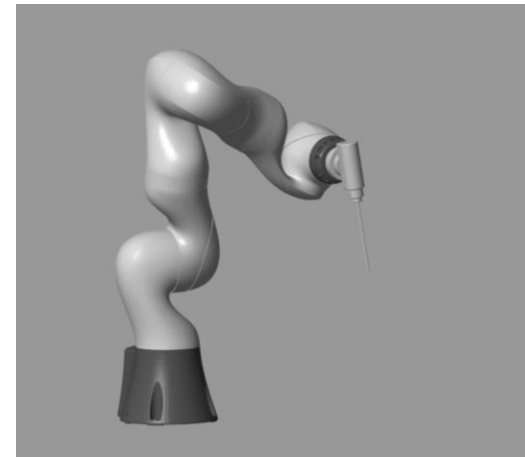
## 结构化编程 structured programming



DC Motor



Servo Joint



Robot Arm

## 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
- 条件表达式
- 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



# 程序设计真的是生命攸关！



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）

如机器人钻孔的步骤：

正确的顺序：目标定位→轨迹生成→关节驱动→开始钻孔

错误的顺序：开始钻孔→关节驱动→轨迹生成→目标定位

错误的顺序：目标定位→开始钻孔→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：

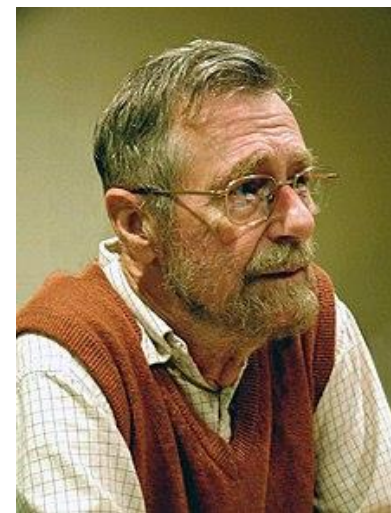
$$\pi = 16 * \left( \frac{1}{5} - \frac{1}{3 * 5^3} + \frac{1}{5 * 5^5} - \frac{1}{7 * 5^7} + \dots \right) - 4 * \left( \frac{1}{239} - \frac{1}{3 * 239^3} + \frac{1}{5 * 239^5} - \frac{1}{7 * 239^7} + \dots \right)$$

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

- 其概念最早由E.W. Dijkstra（荷兰人）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。



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

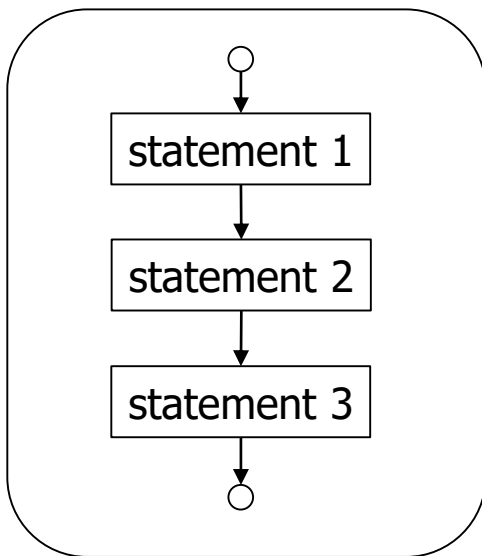
与Knuth并称为我们这个时代最伟大的计算机科学家的人

## C语言是典型的结构化程序设计语言

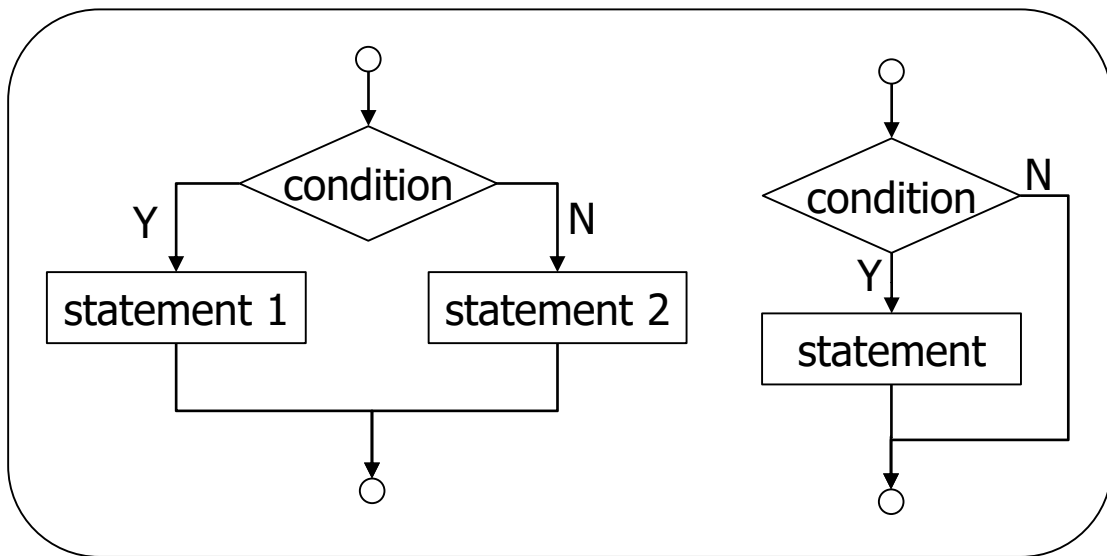


# 结构化程序设计

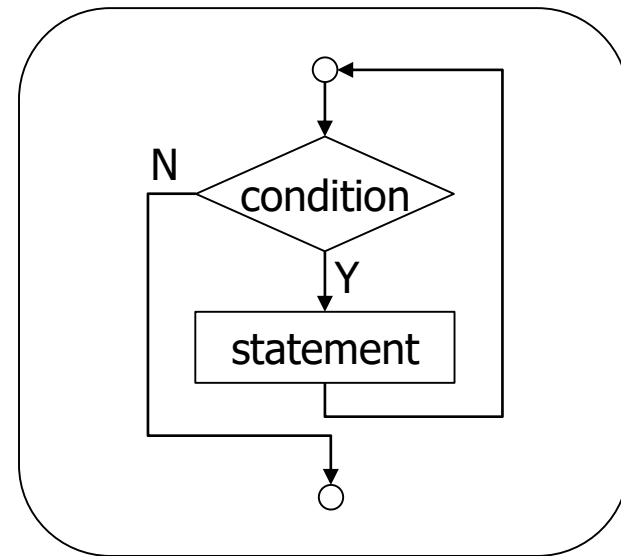
结构化程序的三种控制结构：



顺序 (sequence)



选择 (selection)



重复 (repetition)



# 人的一生是结构化的过程

- **长期来看**的顺序结构:

出生->幼儿园->小学->中学->大学->研究生->工作->结婚->生子->退休->死亡

- **中期来看**的选择结构:

考研or工作? 搞金融or搞IT? 和小张结婚or和小李结婚? ... 人人心中都有一个条件决定自己的选择? 学or不学C语言程序设计?

- **短期来看**的重复结构:

每天在重复 起床->吃饭->干活/学习->吃饭->干活/学习->吃饭->睡觉

**在正能量的重复中，获得正确的人生选择，度过一个最有价值的顺序人生！**

程序人生



# 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：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

```
// a divided by b
#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

```
// a divided by b
#include <stdio.h>
int main()
{
    int a, b;
    double d;
    while (~scanf("%d%d", &a, &b))
    {
        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？





# C语言的三类控制结构：选择

例：求解二次方程  $ax^2 + bx + c = 0$

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

```
#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>
int main()
{
    double a, b, c, delta, r1, r2;
    scanf("%lf%lf%lf", &a, &b, &c);
    delta = b * b - 4 * a * c;
    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 if (delta == 0)
        printf("one real root: %f\n", -b / (2 * a));
    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：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 tem = b; b = a + b; a = tem;`

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)

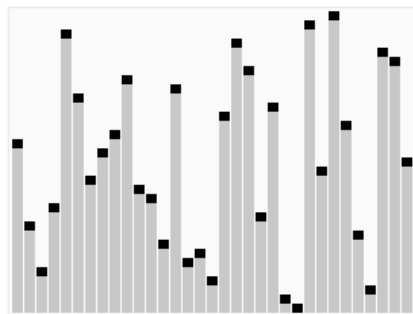
- 采用计算机编程解决某个问题时，需要设计相应的**算法**
  - ◆ 算法：是用来解决某个问题的计算过程，一般都有输入和输出，它包括
    1. 执行的操作
    2. 执行操作的顺序
  - ◆ 任何算法都是由**上述三种控制结构**组成
  - ◆ 算法执行过程**不能有二义性**，必须在**有限时间内**结束
  - ◆ 算法的提出需要相应的数学模型或物理理论作为支撑
    - 理论基础的重要性
    - 数学是工具（对算法的提出具有直接指导意义，例如很多数值算法就是由相应的数学定理直接启发的）
    - 物理是基础（在刘慈欣的“三体”小说中，地球科技就被“智子”锁死了，基础理论不进步，计算机计算力的瓶颈就无法突破）
  - ◆ **程序设计的核心是算法**，不懂算法的编程者就沦为coder了（真正的码农：**编码界的劳动密集型体力劳动者**）
- **结构化程序设计=算法+数据结构**



# 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

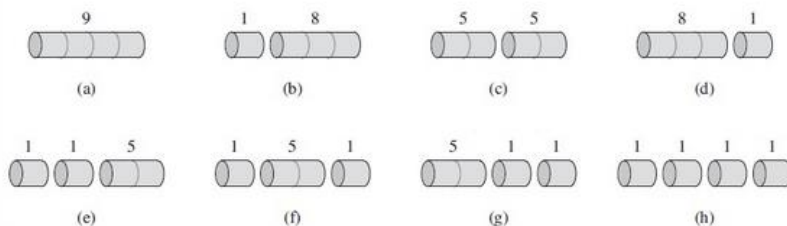
|    |    |     |    |    |    |     |    |    |    |    |    |     |    |    |   |
|----|----|-----|----|----|----|-----|----|----|----|----|----|-----|----|----|---|
| 13 | -3 | -25 | 20 | -3 | 16 | -23 | 18 | 20 | -7 | 12 | -5 | -22 | 15 | -4 | 7 |
|----|----|-----|----|----|----|-----|----|----|----|----|----|-----|----|----|---|

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 |

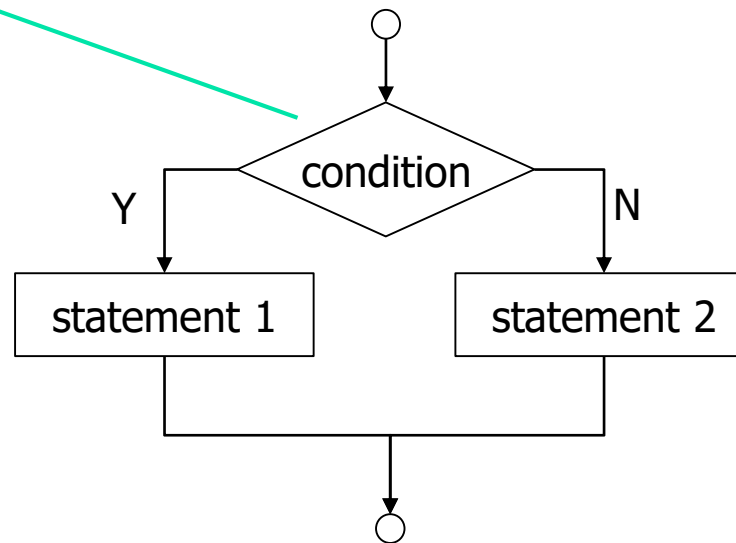
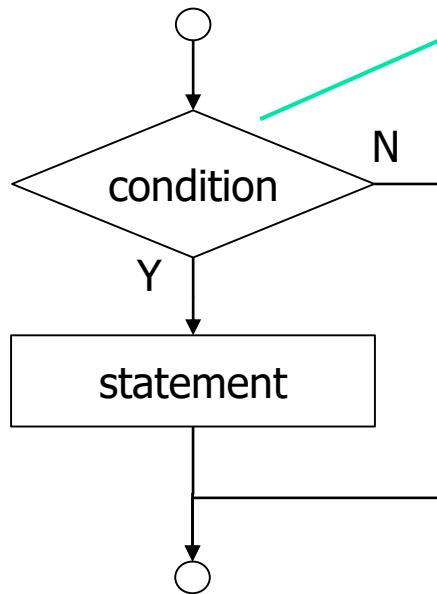
最大兼容活动集

千里之行，始于足下，首先学好C语言，但程序思想最重要

## 4.2 条件语句

- if 条件语句
- if/else 条件语句

任何一个数值表达式都可以作为if 的条件



- 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. 任何一个数值表达式 (expression) 都可以作为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(N or F); 否则称条件为真(T or Y)

# 关系运算符与逻辑运算符【复习】

括号内的表达式作为if语句的“条件”

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

- ◆ 程序中对不同分支的执行取决于给定的**条件**
- ◆ C语言中的条件由**表达式**来描述
- ◆ 表达式可以是一个**关系表达式**（由**关系运算符**描述），也可以是由**逻辑运算符**连接起来的多个关系表达式，甚至可以是**任意一个表达式**
- ◆ 表达式产生确定的值，结果为0（假, false）或非0（真, true）

- ◆ 关系运算符：>, <, >=, <=, ==, !=
- ◆ 逻辑运算符：与 &&, 或 ||, 非 !

例：设 x = 1;

(5 >= 3) || (x == 5) ?

(2 == 3) && (x >= 7) ?

| && |   | A        |   |
|----|---|----------|---|
|    |   | T        | F |
| B  | T | <b>T</b> | F |
|    | F | F        | F |

|   |   | A |          |
|---|---|---|----------|
|   |   | T | F        |
| B | T | T | T        |
|   | F | T | <b>F</b> |

| A  | T | F |
|----|---|---|
| !A | F | T |

电路的串并联？

开关切换？

# 关系运算符与逻辑运算符【复习】

- ◆ 关系运算符:  $>$ ,  $<$ ,  $>=$ ,  $<=$ ,  $==$ ,  $!=$
- ◆ 逻辑运算符: 与  $\&\&$ , 或  $\|\$ , 非  $!$

例: 设 `int x = 1;`

`(x == 5) || (5 >= 3)`          值为 1 (该1占4字节)

`(2 == 3) && (x >= 7)`          值为 0 (该0占4字节)

**\*\*注意:**

1. 在C89中, 没有定义Bool (布尔) 类型, 它用非0值表示真 (True), 用0表示假 (False), 都占4字节。因此, 在C中, 任何一个表达式都可作为条件。
2. C编译器在给出逻辑运算结果时, 以数值非0代表真, 以数值0代表假。
3. C99中, `<stdbool.h>` 有 `bool` 类型, 占1个字节。

**A && B**    若A为0 (假), 则B不必求值, 结果一定为0 (假)。

**A || B**      若A为非0 (真), 则B不必求值, 结果一定为非0 (真)。

如上逻辑表达式的这一重要性质也称为“**短路求值**”, 在程序设计中经常会用到。如在输入数据进行处理时, 经常用如下的表达式:

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

```
if( (c >= 'a') && (c <= 'z') ) { ... }
```

## 逻辑表达式实例 【复习】

- 判断整型变量n的值为一个0到10之间的值：

$(0 \leq n \ \&\& \ n \leq 10)$

注意：

不能写为  $(0 \leq n \leq 10)$

不能得到预期结果

（该表达式是一个合法的逻辑表达式，  
当n为15时，分析一下其值为多少？）

- 判断字符变量c是字母：

$((c \geq 'a') \ \&\& \ (c \leq 'z')) \ || \ ((c \geq 'A') \ \&\& \ (c \leq 'Z'))$

- 判断某年是平年还是闰年（闰年为能被4整除但不能被100整除，或能被400整除）：

$((y \% 4 == 0) \ \&\& \ (y \% 100 != 0)) \ || \ (y \% 400 == 0)$



# 运算符的优先级【再次回顾】

| 优先级 | 运算符    | 名称或含义    | 使用形式                    | 结合方向 | 说明    |
|-----|--------|----------|-------------------------|------|-------|
| 1   | 后置++   | 后置自增运算符  | 变量名++                   | 左到右  |       |
|     | 后置--   | 后置自减运算符  | 变量名--                   |      |       |
|     | []     | 数组下标     | 数组名[整型表达式]              |      |       |
|     | ()     | 圆括号      | (表达式) / 函数名(形参表)        |      |       |
|     | .      | 成员选择（对象） | 对象.成员名                  |      |       |
|     | ->     | 成员选择（指针） | 对象指针->成员名               |      |       |
| 2   | -      | 负号运算符    | -表达式                    | 右到左  | 单目运算符 |
|     | (类型)   | 强制类型转换   | (数据类型)表达式               |      |       |
|     | 前置++   | 前置自增运算符  | ++变量名                   |      | 单目运算符 |
|     | 前置--   | 前置自减运算符  | --变量名                   |      | 单目运算符 |
|     | *      | 取值运算符    | *指针表达式                  |      | 单目运算符 |
|     | &      | 取地址运算符   | &左值表达式                  |      | 单目运算符 |
|     | !      | 逻辑非运算符   | !表达式                    |      | 单目运算符 |
|     | ~      | 按位取反运算符  | ~表达式                    |      | 单目运算符 |
|     | sizeof | 长度运算符    | sizeof 表达式 / sizeof(类型) |      |       |
|     |        |          |                         |      |       |
| 3   | /      | 除        | 表达式/表达式                 | 左到右  | 双目运算符 |
|     | *      | 乘        | 表达式*表达式                 |      | 双目运算符 |
|     | %      | 余数（取模）   | 整型表达式%整型表达式             |      | 双目运算符 |

# 运算符的优先级【再次回顾】

| 优先级 | 运算符 | 名称或含义 | 使用形式             | 结合方向 | 说明    |
|-----|-----|-------|------------------|------|-------|
| 4   | +   | 加     | 表达式+表达式          | 左到右  | 双目运算符 |
|     | -   | 减     | 表达式-表达式          |      | 双目运算符 |
| 5   | <<  | 左移    | 表达式<<表达式         | 左到右  | 双目运算符 |
|     | >>  | 右移    | 表达式>>表达式         |      | 双目运算符 |
| 6   | >   | 大于    | 表达式>表达式          | 左到右  | 双目运算符 |
|     | >=  | 大于等于  | 表达式>=表达式         |      | 双目运算符 |
|     | <   | 小于    | 表达式<表达式          |      | 双目运算符 |
|     | <=  | 小于等于  | 表达式<=表达式         |      | 双目运算符 |
| 7   | ==  | 等于    | 表达式==表达式         | 左到右  | 双目运算符 |
|     | !=  | 不等于   | 表达式!= 表达式        |      | 双目运算符 |
| 8   | &   | 按位与   | 整型表达式&整型表达式      | 左到右  | 双目运算符 |
| 9   | ^   | 按位异或  | 整型表达式^整型表达式      | 左到右  | 双目运算符 |
| 10  |     | 按位或   | 整型表达式 整型表达式      | 左到右  | 双目运算符 |
| 11  | &&  | 逻辑与   | 表达式&&表达式         | 左到右  | 双目运算符 |
| 12  |     | 逻辑或   | 表达式  表达式         | 左到右  | 双目运算符 |
| 13  | ?:  | 条件运算符 | 表达式1? 表达式2: 表达式3 | 右到左  | 三目运算符 |

# 运算符的优先级【再次回顾】

| 优先级 | 运算符 | 名称或含义   | 使用形式        | 结合方向 | 说明       |
|-----|-----|---------|-------------|------|----------|
| 14  | =   | 赋值运算符   | 变量=表达式      | 右到左  |          |
|     | /=  | 除后赋值    | 变量/=表达式     |      |          |
|     | *=  | 乘后赋值    | 变量*=表达式     |      |          |
|     | %=  | 取模后赋值   | 变量%=表达式     |      |          |
|     | +=  | 加后赋值    | 变量+=表达式     |      |          |
|     | -=  | 减后赋值    | 变量-=表达式     |      |          |
|     | <<= | 左移后赋值   | 变量<<=表达式    |      |          |
|     | >>= | 右移后赋值   | 变量>>=表达式    |      |          |
|     | &=  | 按位与后赋值  | 变量&=表达式     |      |          |
|     | ^=  | 按位异或后赋值 | 变量^=表达式     |      |          |
|     | =   | 按位或后赋值  | 变量 =表达式     |      |          |
| 15  | ,   | 逗号运算符   | 表达式,表达式,... | 左到右  | 从左向右顺序运算 |

## 这么多！怎么记忆？

1. 读一遍，能记住多少就记多少。
2. 使用的时候，按常识和感觉，相信自己的直觉。
3. 加括号。
4. 不理它，天天坚持编程（模仿书上的例程），很快，不知为何，就都会了。

## 运算符的优先级【复习】

- 优先级从上到下依次递减，最上面具有最高的优先级，逗号操作符具有最低的优先级。
- 相同优先级中，按结合顺序计算。大多数运算是从左至右计算。
- 基本的优先级需要记住：
  - ◆ 指针最优。单目运算优于双目运算，如正负号。
  - ◆ 先乘除（模），后加减。
  - ◆ 先算术运算，后移位运算，最后位运算，如：  
 $1 \ll 3 + 2 \& 7$  等价于  $(1 \ll (3 + 2)) \& 7$
  - ◆ 逻辑运算最后计算。
- 很多优先级是比较显然的。记住一些常用运算的优先级。记不住怎么办？

提示：善用小括号 ( )

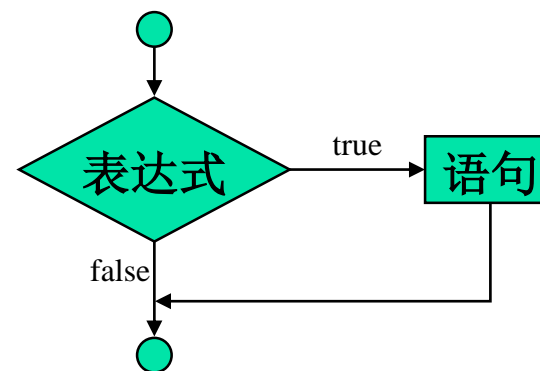
```
char c;
```

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

# 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，只有取值为0时，才表示条件为假。

if(a + b)  
    printf("Hello!\n");

大佬一般这样用，更专业

提示：if(5) 等价于 if(1)



if( **a + b != 0** )  
    printf("Hello!\n");

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

- 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) // 专业写法 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");
```

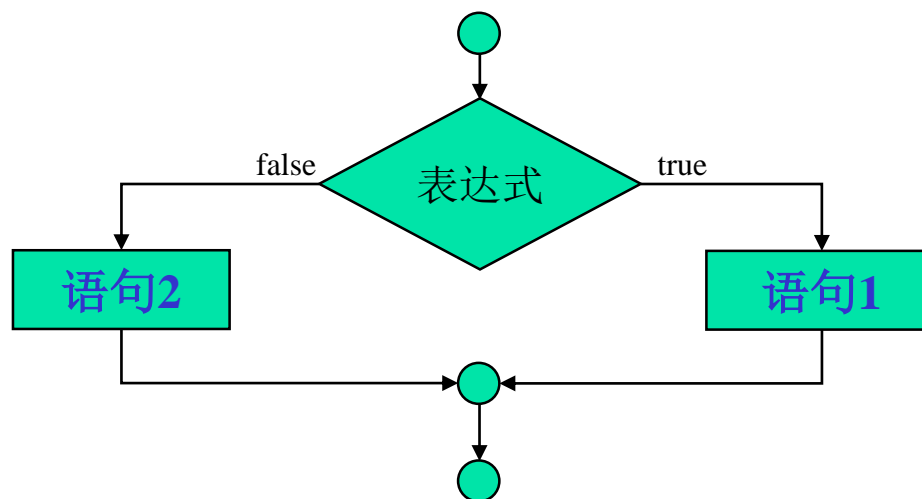
# if/else 条件语句

- 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



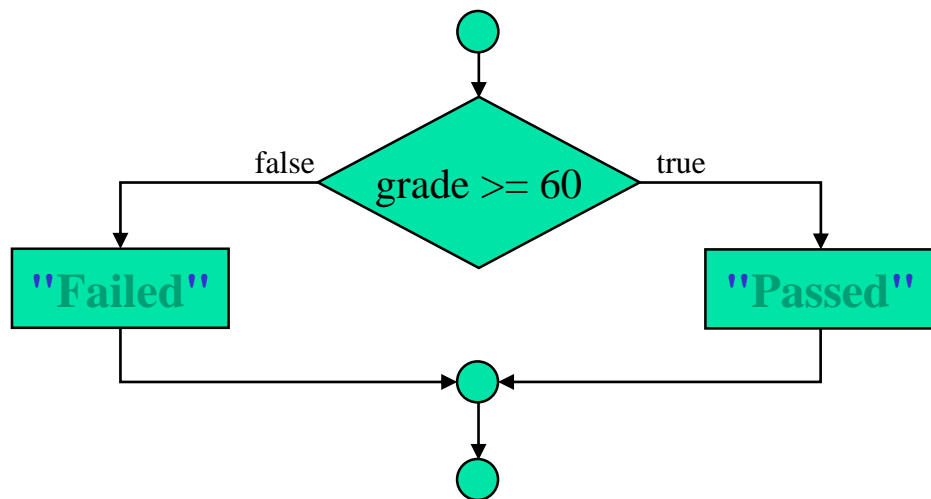
# if/else 条件语句

- 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】 假设考试成绩大于等于60分为通过，否则为不通过。

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

- 条件运算符 ( `__ ? __ : __` ) 与if/else结构密切相关

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");
```

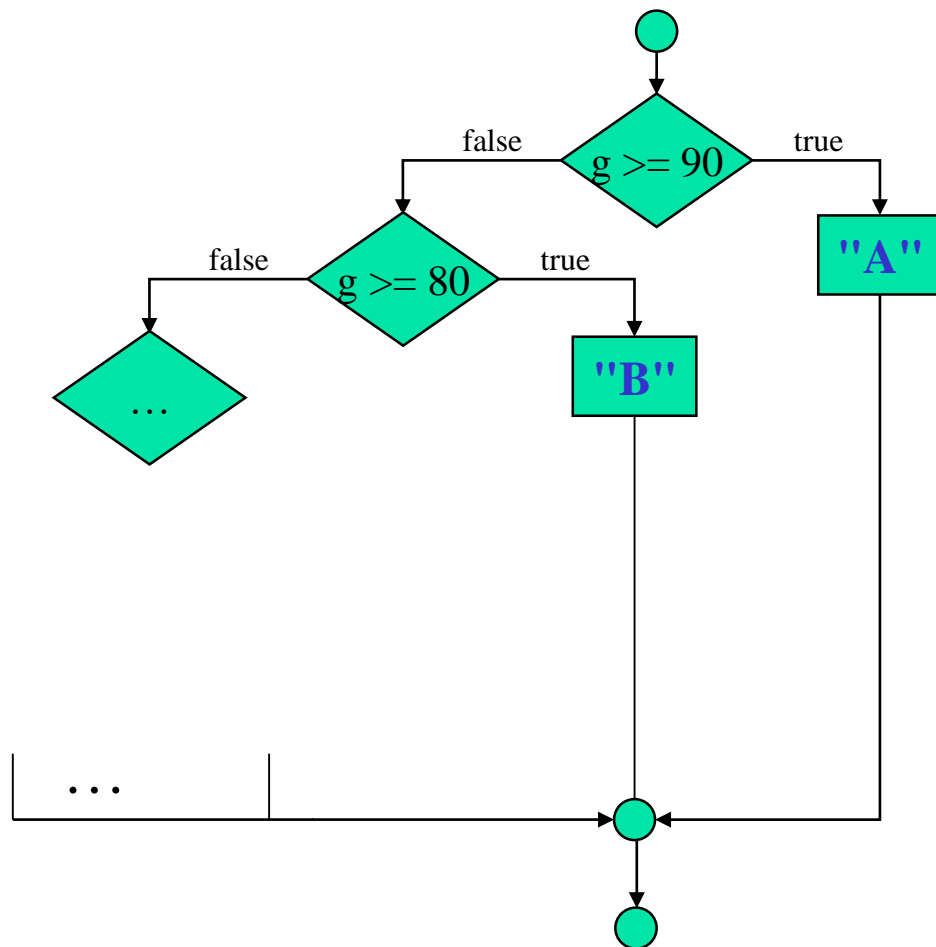
# 条件语句中的嵌套

- 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1：买苹果的挑选过程。
  - ◆ 例2：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】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"
```

```
int grade;
printf("Input the 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;
printf("Input the 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;
```

提示：更好的书写风格？

# 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，现在的计算机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");
```



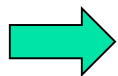
```
grade = 88;
if (grade < 60)
    printf("F\n");
else if ( grade < 70 )
    printf("D\n");
else if ( grade < 80 )
    printf("C\n");
else if ( grade < 90 )
    printf("B\n");
else
    printf("A\n");
```

## else 匹配问题

```
x = 6;   y = 2;
if(x > 5)
    if(y > 5)
        printf("x and y are > 5\n");
else
    printf("x is <= 5\n");
```

output?

x is <= 5



```
x = 6;
y = 2;
if(x > 5)
    if(y > 5)
        printf("x and y are > 5\n");
    else
        printf("x is <= 5\n");
```

```
x = 6;   y = 2;
if(x > 5){
    if(y > 5)
        printf("x and y are > 5\n");
}
else
    printf("x is <= 5\n");
```

- 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");
```



```
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）

# 不太好的代码习惯

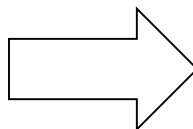
```
/*
Author: [REDACTED]
Result: AC Submission_id: 1976971
Created at: Tue Oct 29 2019 15:30:14 GMT+0800 (CST)
Problem: 2583 Time: 15 Memory: 1708
*/

#include<stdio.h>
int a[1000000];
int main(void)
{
    int n,i,j,goal;

    while(~scanf("%d",&n))
    {
        j=32;
        if(n!=-2147483648)
        {
            if(n>=0)
            {
                n=n;
            }
            if(n<0)
            {
                a[1]=1;
                n=-n;
            }

            while(j>1)
            {
                goal=n%2;
                a[j]=goal;
                n=n/2;
                j-=1;
            }
            for(i=1;i<32;i++)
            {
                printf("%d",a[i]);
            }
            printf("%d\n",a[32]);
        }
        if(n== -2147483648)
        {
            printf("100000000000000000000000000000000\n");
        }
        n=0,i=0,a[1]=0;
    }
    return 0;
}
```

不太好



```
#include <stdio.h>
int a[1000000];
int main(void)
{
    int n, i, j, goal;

    while(~scanf("%d", &n))
    {
        j=32;
        if(n!=-2147483648)
        {
            if(n<0)
            {
                a[1]=1;
                n=-n;
            }
            while(j>1)
            {
                goal=n%2;
                a[j]=goal;
                n=n/2;
                j-=1;
            }
            for(i=1; i<32; i++)
                printf("%d",a[i]);
            printf("%d\n",a[32]);
        }
        if(n== -2147483648)
            printf("100000000000000000000000000000000\n");
        n=0,i=0,a[1]=0;
    }
    return 0;
}
```

略好

## 不太好的代码习惯

```

/*
Author: [REDACTED]
Result: AC Submission_id: 1976971
Created at: Tue Oct 29 2019 15:30:14 GMT+0800 (CST)
Problem: 2583 Time: 15 Memory: 1708
*/

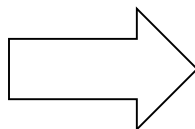
#include<stdio.h>
int a[1000000];
int main(void)
{
    int n,i,j,goal;

    while(~scanf("%d",&n))
    {
        j=32;
        if(n!=-2147483648)
        {
            if(n>=0)
            {
                n=n;
            }
            if(n<0)
            {
                a[1]=1;
                n=-n;

                while(j>1)
                {
                    goal=n%2;
                    a[j]=goal;
                    n=n/2;
                    j--1;
                }
                for(i=1;i<32;i++)
                {
                    printf("%d",a[i]);
                }
                printf("%d\n",a[32]);
            }
            if(n== -2147483648)
            {
                printf("10000000000000000000000000000000\n");
            }
            n=0, i=0, a[1]=0;
        }
    }
    return 0;
}

```

不太好

[illegible]



## 条件语句实例：一元二次方程求根

【例3-4 (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

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

程序的小瑕疵！

- 计算机中有些实数的表示不是100%精确。比如：dt的值应该为0，但在计算机中实际可能为 $10^{-40}$ ，则  $(dt > 0.0)$  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);`

//// file: c3-4 solve equation.c

#include <stdio.h>

#include <math.h>

注意输入时的半角、全角等问题，尤其是“，；（”等符号。

int main()

{

double a, b, c, r1, r2, dt;

printf("输入a b c三个实数（其中a不能为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3-6】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)。

程序的小瑕疵！

- 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>`)。

```
// Four arithmetic operations
// file: c3-6_4arith_op.c
#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;
}
```

试试：

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

再试试：

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

# 条件语句实例：四则运算

## 【例3-6】四则运算

程序的严重问题！  
(不是小瑕疵)

- 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
// file: c3-6_4arith_op.c
#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); // b == c的关系运算结果为 0, 此值赋给变量 a
```

- 用==运算符进行赋值或用=运算符表示相等是个逻辑错误

```
int gender = 0; // 为1时表示性别为male,为0时表示为female
```

```
if ( gender = 1 ) // gender = 1 and if ( gender ). 逻辑错误, 还隐蔽地修改了变量的值。
```

```
    printf("male\n");
```

```
else
```

```
    printf("female\n");
```

输出?

### 编程提示

if ( 1 == gender ), 好的解决方案

if ( 1 = gender ), 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!

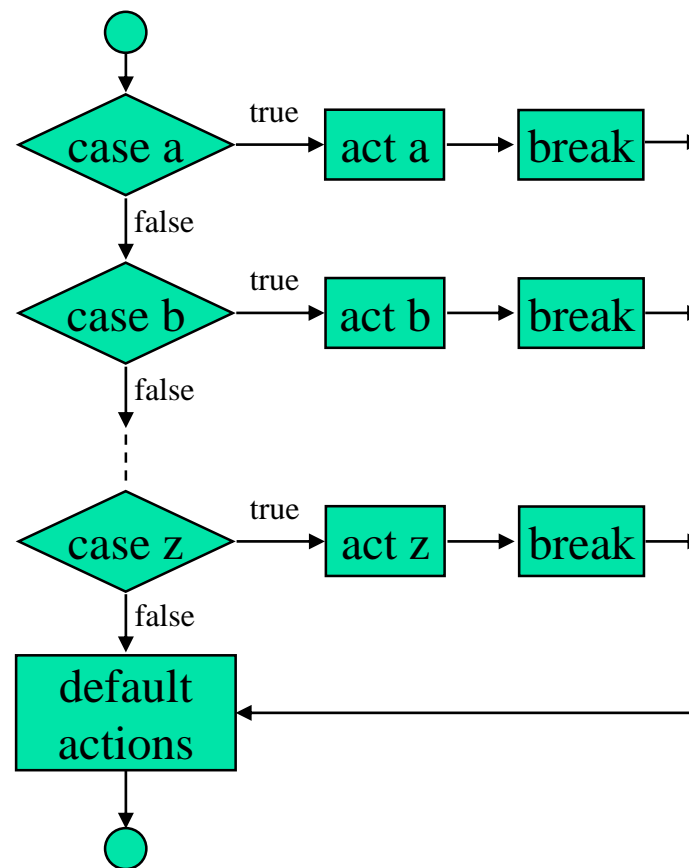
# switch语句

- if/else是双向选择结构，switch可以进行多项选择。

switch  
语法：

```
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语句使程序退出switch，case中有多个语句时，不必放在{ }中。
- switch只用于测试常量整型表达式。



# switch语句实例：成绩转换

【例a3-1】输入一个成绩（百分制整数），转换为等级制（A~F）

```
// file: a3-1_score_rank.c
#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语句使程序退出switch。**
- case中有多个语句时，不必放在{ }中。
- switch只用于测试整型值。

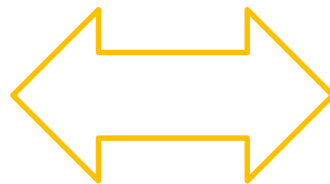
```
switch ( score/10 ){
case 10: // e.g. 100/10 is 10
case 9:  // e.g. 96/10 is 9
    printf("A\n");
    break;
case 8:
    printf("B\n");
    break;
case 7:
    printf("C\n");
    break;
case 6:
    printf("D\n");
    break;
default:
    printf("F\n");
    break;
} // end switch

return 0;
}
```

## switch语句实例：成绩转换

【例a3-1e】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：**  
左边逻辑清晰，  
右边节省空间。  
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3-6-1】四则运算

注意与【例3-6】比较

switch控制结构里嵌套  
(包括) 了一个if结构。  
控制结构允许嵌套。

```
// file: c3-6-1 4arith_op.c
```

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

```
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(0 != y) // fabs(y) > epsilon
```

```
            {
```

```
                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;
```

```
}
```

恭喜你！  
已经会设计简单的  
计算器了！





## 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）
- .....



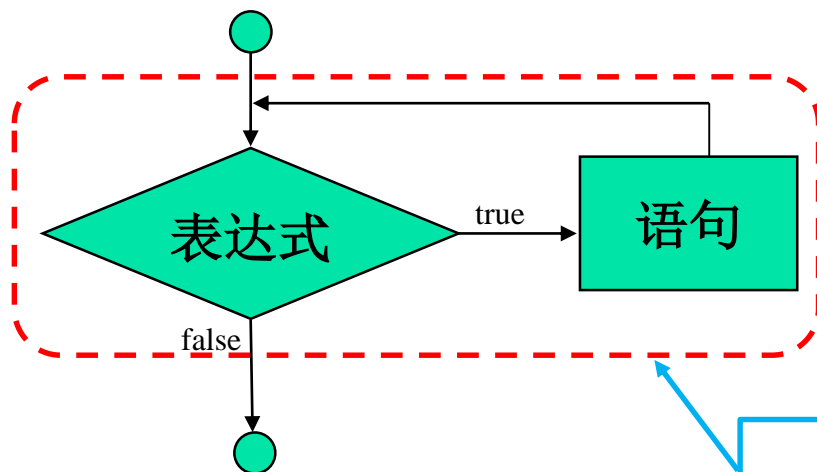
# while语句 【复习】

- 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）：

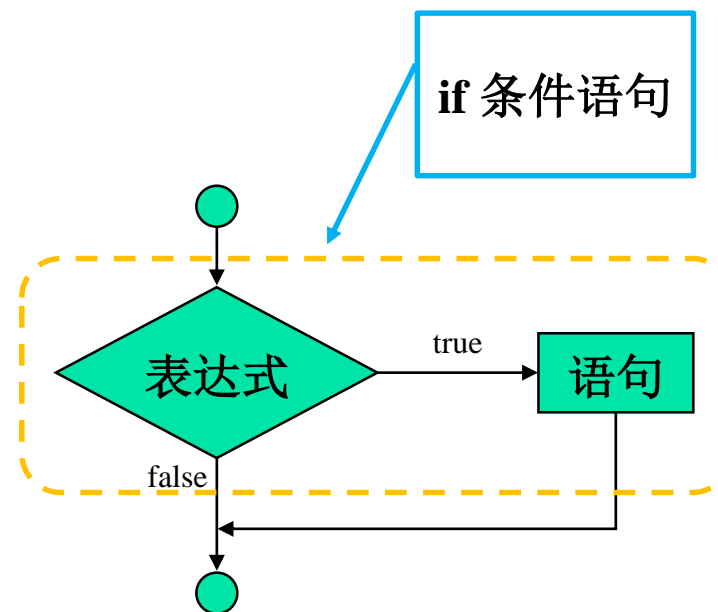
```
while( <表达式> )  
{  
    <语句>  
}
```

循环头

循环体



while循环语句



if 条件语句

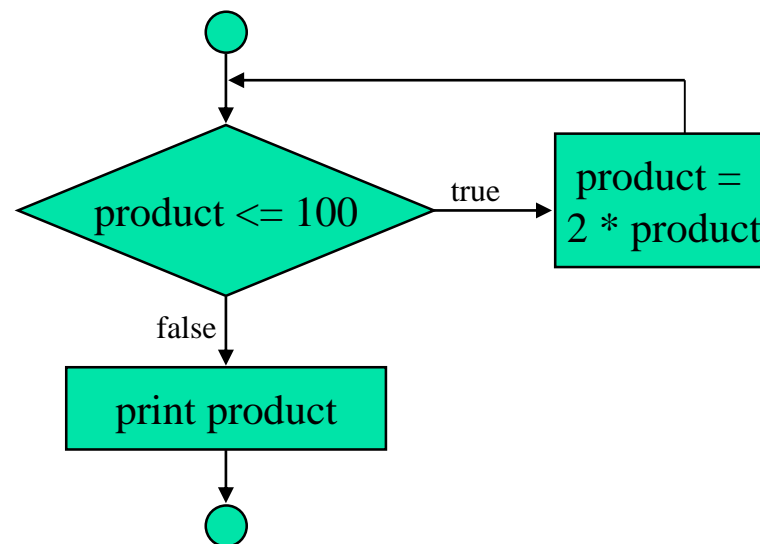
# while语句【复习】

- 循环(loop)语句：  
使程序在一定条件下重复（反复）操作。

【例】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：

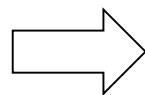
**while**循环体中，若没有使循环条件变为假的动作，或满足条件的**break**或**goto**语句；或没有外部的读入数据使得循环条件为假，将导致**无限循环（死循环）**，如

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

# while 语句

## while 语句的特殊形式

```
while(expression) ;
```



```
while(expression)
{
}
```

例: `while((c = getchar()) != '\n');`

清空输入缓冲区直到新的一行开始

## while语句的实例（读入多行数据） 【复习】

- 在OJ上练习时，有时会遇到要求输入多组数据的情况，输入结束用文件末尾EOF控制
- 如果是键盘输入，在windows平台下，输入回车后按下Ctrl + Z时，scanf会读入EOF（Ctrl+Z后还需要回车才能起作用）
- 如果是文件输入，则读到文件结尾时scanf返回EOF（OJ的测试数据其实是保存到文件里，然后采用输入重定向的方法使得你的程序中的scanf实际从文件中读入数据）

```
int a, b;  
while(~scanf("%d%d", &a, &b))  
    printf("%d + %d = %d\n", a, b, a + b);
```

```
C:\Users\17419\Desktop\sytestC\a.exe  
1 2  
1 + 2 = 3  
5  
21  
5 + 21 = 26  
^Z  
  
Process returned 0 (0x0)  
execution time : 31.024 s  
Press any key to continue.
```

# while语句的实例（计数器控制循环）【复习】

## 【例4-2】最大公约数

输入两个正整数，求它们的最大公约数。

循环次数 $\min(a, b)$ .  
枚举法。第一讲介绍了另一种类似的实现方式，请对比分析。

当然，还有更好的算法能计算gcd，后文介绍。

```
#include <stdio.h>

int main()
{
    int a, b, gcd, i = 1;
    scanf("%d%d", &a, &b);

    while(i <= a && i <= b)
    {
        if(a%i == 0 && b%i == 0)
            gcd = i;
        i++;
    }
    printf("gcd is: %d\n", gcd);

    return 0;
}
```

# while语句的实例（标志控制循环）【复习】

【例a4-2】求平均分 编写一个通用程序，用于计算每一门课考试的平均成绩，但每一门课修课的人数不一样。不用计数器控制重复。

- 标志控制重复(sentinel -controlled repetition)
  - ♦ 用特殊的标志值(sentinel value)控制循环结束，也称为不确定重复(indefinite repetition)。
- 自上而下逐步完善的程序设计思想

```
// a4-2 average-sentinel.c  
.....  
scanf("%d", &grade);  
while ( grade != -1 )  
{  
    total += grade;  
    g_counter++;  
    scanf("%d", &grade);  
}  
.....
```

**Initialize variables**

**Input, Sum, Count the quiz grades**

**Calculate and print the class average**



**Initialize total to zero**

**Initialize counter to zero**

**Input the first grade (possibly the sentinel)**

**While the user has not yet entered the sentinel**

**Add the grade into the running total**

**Add one to the grade counter**

**Input the next grade (possibly the sentinel)**

**If the counter is not equal to zero**

**Set the average to the total divided by counter**

**Print the average**

**else**

**Print "No grades were entered"**

# while语句的实例（标志控制循环）

**【例a4-3】** 测试不间断的输入整数，直到输入格式有误或强制结束输入时结束程序（windows下 Ctrl+C命令强制结束输入并退出，Ctrl+Z命令强制结束输入）。



## 思考

scanf读入错误返回什么？  
读入到文件结束返回什么？

注意：scanf()从输入设备上正确读入后，读入的字符才从输入缓冲区删除，否则一直留在缓冲区中，这可能会影响scanf的再次读入。见下一个例子。

```
// a4-3 test while scanf.c
#include <stdio.h>

int main()
{
    int a;
    printf("\ninput an integer: ");
    while(scanf("%d", &a) > 0)
    {
        printf("valid input: %d\n", a);
        printf("\ninput an integer: ");
    }

    printf("invalid input! quit!\n");
    return 0;
}
```



# while语句的实例（标志控制循环）

**【例4-3】** 数据输入 输入一行，该行的正确输入是一个3位正整数，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）

```
// c4-3_test_while_scanf.c
#include <stdio.h>
int main()
{
    int a;
    printf("\ninput a 3-digit number: ");

    // 无效输入时，条件就为真
    while(scanf("%d", &a) == 0 || (a < 100 || a > 999)){
        printf("invalid input: %d\n", a);
        printf("input a 3-digit number: ");
        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xy，开始下一行输入。

若没有该行，第一次读入12，第二次scanf()读到 'x' 时发现是无效读入，循环，又读到 'x' ，.....，死循环！



思考

该代码还是可能进入死循环，when?

# while语句的实例（标志控制循环）

**【例4-3】** 数据输入 输入一行，该行的正确输入是一个3位正整数，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）

```
int main(){
    int a; printf("\ninput a 3-digit number: ");

    // 无效输入时，条件就为真
    while(scanf("%d", &a) == 0 || (a < 100 || a > 999)){
        printf("invalid input: %d\n", a);
        printf("input a 3-digit number: ");
        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xy，开始下一行输入。若没有该行，第一次读入12，第二次scanf()读到'x'时发现是无效读入，循环，又读到'x'，.....

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



思考

该代码还是可能进入死循环，when?

## while语句的实例（标志控制循环）（很实用的例子）

**【例4-3】** 数据输入 输入一行，该行的正确输入是一个3位正整数，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）

```
int main(){
    int a; printf("\ninput a 3-digit number: ");

    // 无效输入时，条件就为真
    while(scanf("%d", &a) == 0 || (a < 100 || a > 999)){
        printf("invalid input: %d\n", a);
        printf("input a 3-digit number: ");
        while(getchar() != '\n');
    }
    ...
}
```

如果没有这一行...

键盘缓冲区:  
12 xy\n

a

- 1) 读入整数12到变量a;
- 2) 读入格式正确( scanf返回1 ), 但数字不符合要求, 新的读入从12后的空格字符开始解析;
- 3) 试图读入一个整数到变量a;
- 4) 读入不正确(不匹配), 读入位置不移动(位于x处), 下次又在同样位置尝试读入...



## while语句的实例（标志控制循环）

**【例4-4】** 输入行数统计 输入一段文本，统计行数（一行文本结束后须加一个回车换行）。

```
// c4-4_count_line_num.c
#include <stdio.h>

int main()
{
    int c, n=0;
    while((c = getchar()) != EOF)
        if(c == '\n') 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？

# 循环的要点

- 控制变量（或循环计数器）的名称(name)
- 控制变量的初始值(initial value)
- 测试控制变量终值(final value)的条件（即是否继续循环）
- 每次循环时控制条件修改

```
...  
    const int COUNT = 10;  
    int g_counter = 1;  
...  
    while ( g_counter <= COUNT ) {  
        scanf("%d", &grade);  
        total += grade;  
        g_counter++;  
    }  
    average = (double) total / COUNT;  
...
```

## 编程要点与技巧

- 用整数值控制计数循环
- 特殊标记的控制条件
- 缩排控制结构中的语句
- 程序中应有适当的空行
- 嵌套结构不宜太多
- 缩排和空行使程序具有二维效果，增加可读性

# for语句 【复习】

- for结构与while结构类似。也是一种广泛使用的循环结构
- 语法格式（循环体只有一条语句时可以用不用大括号）：

**for( <表达式1>; <表达式2>; <表达式3> )**

**{**

**<语句>**

**}**

循环头

循环体

表达式1, 2, 3均可以为空（省略），但分号不能省略

- for结构的一般格式与等价的while结构

**for ( expression1; expression2; expression3 )**

**{ statement**

**}**

⇔

**expression1**

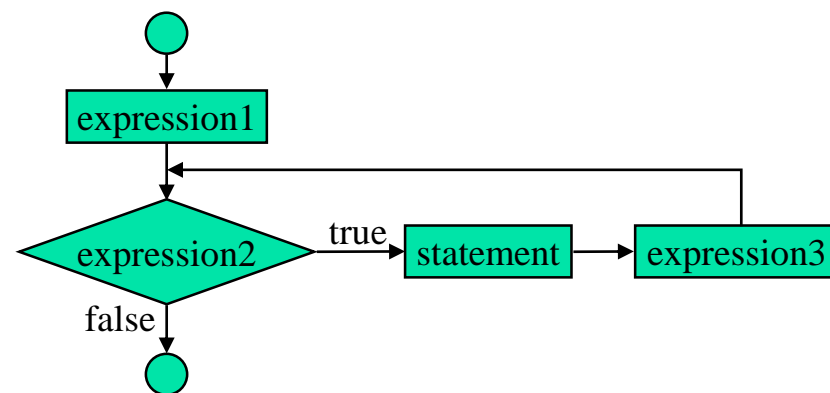
**while ( expression2 )**

**{**

**statement**

**expression3;**

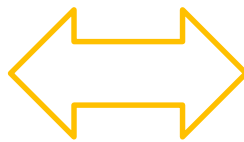
**}**



# for语句实例

## 【例1】n个自然数累加

```
int i, r = 0;  
for(i = 1; i <= n; i++)  
    r += i;
```



```
int i = 1, r = 0;  
while(i <= n){  
    r += i;  
    i++;  
}
```

## 【例2】求n的阶乘

```
int i, r, n;  
for(r = i = 1; i <= n; i++)  
    r *= i;
```

## 【例3】求2~100间的所有偶数平方和

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

## for语句实例

【例4-2-1】最大公约数 输入两个整数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:  $\gcd(a, b) = \gcd(b, a \% b)$

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  $a \leftarrow b$ ,  $b \leftarrow a \% b$ , go to step 2.

证明

先证明  $\rightarrow$

设  $g = \gcd(a, b)$ , 则  $g * x = a$ ,  $g * y = b$

,

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  $g \leq g_1 = \gcd(b, a \% b)$

$\leftarrow$  部分的证明, 同理。



# for语句实例

**【例4-2-1】最大公约数** 输入两个整数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  $a \leftarrow b$ ,  $b \leftarrow a \% b$ , go to step 2.

**【gcd(a, b) = gcd(b, a%b)】**

注意：题意中讲得是输入整数，则可能是0或负数。编程初学者特别注意题意。

```
#include <stdio.h>

int main()
{
    int a, b, r;
    scanf("%d%d", &a, &b);
    if(b == 0) {
        printf("gcd is: %d\n", a);
        return 0;
    }
    for(r = a%b; r != 0; r = a%b) {
        a = b;
        b = r;
    }
    printf("gcd is: %d\n", b<0 ? -b : b);

    return 0;
}
```

# for语句实例

【gcd(a, b) = gcd(b, a%b)】

【例4-2-1】最大公约数 辗转相除法求两个数的最大公约数。

```
#include <stdio.h>

int main()
{
    int a, b, r;
    scanf("%d%d", &a, &b);
    if(b == 0) {
        printf("gcd is: %d\n", a);
        return 0;
    }
    for(r = a%b; r != 0; r = a%b) {
        a = b;
        b = r;
    }
    printf("gcd is: %d\n", b<0 ? -b : 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。死循环？如何退出？

## for语句实例

### 【例4-6】三位的水仙花数 求出所有水仙花数

(【例3-2】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？  
你能发明土仙花、木仙花、金仙花数吗？  
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;
        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 a4-4】保险理财（退休金） 每年交10000元，交35年，退休后每月应领多少钱妥当（不考虑通货膨胀的关系）？

分析：

为什么是 $s+m$ ？本金 $s$ ，年初新存入 $m$ ，因此这一年的本金是 $s+m$

2017年底， $s=0$ 元

1. 第1年初，2018年初，存入 $m$ 元
2. 第2年初，2019年初，有 $s=(s+m)*(1+r)$ 元【等号右边的 $s$ 是上一年的原始本金， $m$ 是上一年新存入的， $r$ 是利率】
3. 第3年初，2020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: ");
scanf("%lf", &rate);

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

💡 开保险公司赚钱吗？

## for语句实例

**【例 a4-5】复利计算** 120年前，你的祖爷爷在瑞士银行存了10000美元（年利率0.075）（后来世界发生战争，很多人忘了此事），现在银行找到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一下）

分析：

$$a1 = p + p * r = p * (1 + r)$$

$$a2 = a1 + a1 * r = a1 * (1 + r)$$

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

$$a3 = a2 + a2 * r = a2 * (1 + r)$$

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

...

```
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, u have: %f$\n", i, s);
```

```
}
```

## for语句实例

$$16 \cdot \frac{1}{k \cdot 5^k} > e \Rightarrow \frac{1}{k \cdot 5^k} > \frac{e}{16} = \text{eps}$$

**d** > **eps**

【例 4-7】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 = 15）

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

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：

$$\pi = 16 \cdot A - 4 \cdot B$$

$$A \text{ 的通项 } \frac{(-1)^i}{(2i+1) \cdot 5^{(2i+1)}}, \quad B \text{ 的通项 } \frac{(-1)^i}{(2i+1) \cdot 239^{(2i+1)}}$$

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

课后练习：同学们仔细研读该例。此例程可进一步优化，内层嵌套的for循环可以去掉。

```
int i, j, k, sign = -1, x = 15;
double n, d, s1 = 0, s2 = 0, eps, e = (1e-x)/2;
```

```
eps = e/16.0;
for(i = 0, d = 1; d > eps; i++) {
    sign = i%2 == 0 ? 1 : -1;
    k = 2*i + 1;
    for(j = 0, n = 1.0; j < k; j++)
        n *= 5;
    d = 1.0 / (k*n);
    s1 += d*sign;
}
```

```
eps = e/4.0;
for(i = 0, d = 1; d > eps; i++) {
    sign = i%2 == 0 ? 1 : -1;
    k = 2*i + 1;
    for(j = 0, n = 1.0; j < k; j++)
        n *= 239;
    d = 1.0 / (k*n);
    s2 += d*sign;
}
```

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

# 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。

- 语法格式

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

## 【例4-2-1】辗转相除求最大公约数

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

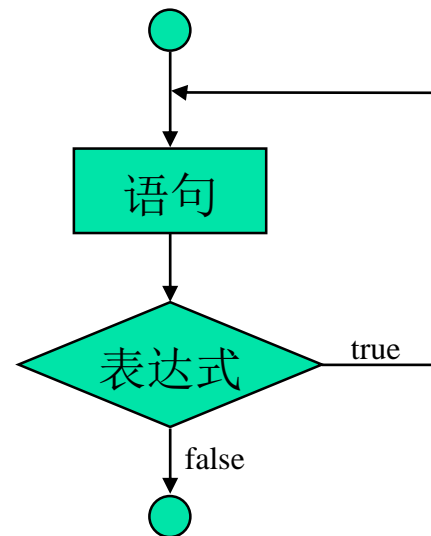


```
while((r = a%b) != 0) {  
    a = b;  
    b = r;  
}
```

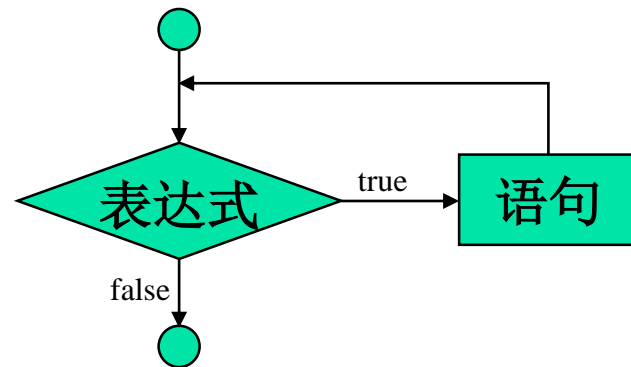


```
for(; (r = a%b) != 0; ) {  
    a = b;  
    b = r;  
}
```

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



while语句



## 循环语句的选择

- while, for, do while 三种循环控制语句本质上相同。
- 什么时候用哪种方式?
- 习惯上, 计数器控制循环用for, 标志控制循环用while

跟编程人员的喜好有关。

【例】n个自然数累加

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

【例】求2~100间的所有偶数平方和

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

【例】文本的行数统计

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



```
for( ; (c = getchar()) != EOF ; )
    if(c == '\n') n++;
```


总觉得有点不自然?



# 逗号表达式

## 【例 4-7】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;**

**r = c = z;**

**提示：合理使用逗号表达式，尽量少用，避免滥用。**

# 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 4-9】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

```
#include <stdio.h>
int main(){
    int i, j, n;
    scanf("%d", &n);
    for(i=0; i<n; i++) {
        for(j=0; j<n-i; j++)
            putchar(' ');
        for(j=0; j<2*n; j++)
            putchar('*');
        putchar('\n');
    }
    return 0;
}
```

如果最后一行不求输出换行，该怎么处理？

## 循环语句的嵌套实例

**【例 4-10】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("%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 4-11】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);
    // printf("%d\n", sizeof(r));
    // printf("r: %lld\n", r);
    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;
}
...
```

同学们可以进一步测试一下，输入 $n$ 为多大时，结果就不再正确了？

# 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）时程序的逻辑就比较费解了。

```
// 【例 4-10】连续正整数
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("%d = %d", s, st++);
            for(k=st; k<=j; k++)
                printf(" + %d", k);
            putchar('\n');
            got++;
        }
    }
}
printf("cases: %d\n", got);
```

# 循环语句的选择与控制

- while, for, do while 三种循环控制语句本质上相同。
- 什么时候用哪种方式？

跟编程人员的喜好有关！

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



```
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？

注意：

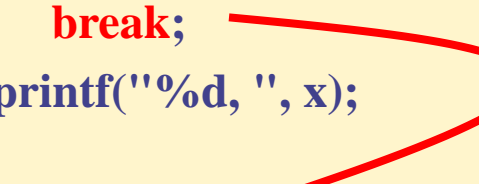
- for循环头的三个表达式都可以省略，即写成 for(;;)，此时第一个第三个表达式表示不执行操作，第二个表达式表示一个非0（即为真）的条件取值。
- 死循环？如何退出？如：

```
while(1){  
    .....  
}
```

# break和continue: 循环语句的“非常规”控制

- break和continue语句改变控制流程。
- 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

输出: 1, 2, 3, 4,

- 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
```



输出: 1, 2, 3, 4, 6, 7, 8, 9, 10,

# break 和 continue 语句

- 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: ?
```

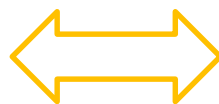
```
for (x = 1; x <= 10; x++ )
{
    if( x ==5 )
        continue;
    printf("%d ", x);
}
printf("OK");
输出: ?
```



## 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(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 4-10-1】连续正整数 ...如果有多个序列满足条件，则输出最长的序列...

分析：

在【例4-10】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。

```
#include <stdio.h>
int main() {
    int s, i, j, k, st, sum;
    scanf("%d", &s);
    for(i=1; i<=s/2; i++) { // 1
        sum = i;
        for(j=i+1; sum<s; j++) { // 2
            sum += j;
            if(sum == s) {
                st = i;
                printf("%d = %d", s, st++);
                for(k=st; k<=j; k++) // 3
                    printf(" + %d", k);
                putchar('\n');
                break; // 跳出哪个循环？
            }
        } // 2 end
        if(sum == s) break; // 跳出哪个循环？
    } // 1 end
    if(sum != s) printf("NONE\n");
    return 0;
}
```

其实这个 **break** 不需要的，因为当满足 **break** 时，在对应该层循环的 **sum < s** 不会满足，自然会退出循环。

## continue语句实例

**【例 4-12】**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);
        }
```

解释

示例：假如A='Y'，如果A==B，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)

根据题意，人的推理很简单、很容易得，C vs Y, A vs Z, B vs X。  
但计算机怎么样推理的呢？该例程即是一种计算机推理，这就是所谓的一种AI（但从程序可以看出，人的智能可能更“逻辑”些，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），计算机的计算速度很快，这种穷举很多时候就比人“强大”）。

## continue语句实例

**【例 4-13】数据求和** 输入多组数 $\langle x_i, y_i \rangle$ ，求所有满足 $a \leq x_i \leq b$ ， $c \leq y_i \leq d$ ，且 $x_i$ 和 $y_i$ 不能互相整除的 $x_i$ 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 4-13】数据求和** 输入多组数 $\langle x_i, y_i \rangle$ ，求所有满足 $a \leq x_i \leq b$ ， $c \leq y_i \leq d$ ，且 $x_i$ 和 $y_i$ 不能互相整除的 $x_i$ 之和与 $y_i$ 之和。

```
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) == 2) {
    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);
```

输入

1 2

3 6

7 4

12 3

5 9

输出

?

## \*控制语句综合实例

【例 a4-6】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1999年10月1日应输入为19991001）。

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

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

```
        if(m<3) {
            y = y-1;
            m = m+12;
        }

        c = y/100;
        y = y%100;
        // Zeller formula
        w = (y + y/4 + c/4 - 2*c
            + (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;
```

```
            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;
    }
}
```

## 控制语句综合实例

Zeller公式：计算任意一天是星期几

【例 a4-6】 查询某天是星期几 输入某一天，查询改天是星期几。输入格式为 yyyyymmdd（如1999年10月1日应输入为19991001）。

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

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

$M$ : 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.

## **\*\*goto语句 【古老的语句了，强烈不建议用】**

- 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。
- 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语句可能使得程序逻辑混乱，使得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）。
- **建议尽量不要用goto语句！！！！**

语句标号

```
int main(){  
    .....  
flagA:  
    <语句1>  
    ...  
    goto flagA;  
    ...  
    return 0;  
}
```



# 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;
}
```

循环语句

**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主要形式（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

- 掌握顺序结构、选择结构、重复结构的含义及作用
- 熟练应用关系表达式、逻辑表达式表述问题
- 理解并会应用逻辑与（&&）和逻辑或（||）的“短路求值”特点
- 熟练掌握if, if/else, switch三种选择语句的语法格式和执行过程
- 掌握自顶而下、逐步完善的程序设计思想进行较简单问题的求解
- 掌握while、for、do while语句的语法格式和相互之间的异同
- 熟练掌握break、continue的区别并会正确使用
- 熟练掌握循环中计数器的作用及其使用方法
- 熟练掌握循环中标志控制的实现方式及注意事项
- 掌握逗号表达式的特点及表达式求解次序
- 理解嵌套的循环语句的执行流程并会正确应用
- 熟练掌握结构化程序设计的基本结构（顺序、选择、重复），并能使用相关控制流语句完成这三种基本结构的程序设计。

## 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？

```
#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) == 2) {
        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);
    return 0;
}
```

输入：

2 6

6 4

5 9

3 6

4 5

7 2

输出？



## for语句实例

**【例 a4-4】 保险理财（退休保险金）** 每年交10000元，交35年，退休后每月应领多少钱妥当（不考虑通货膨胀的关系）？

```
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: ");
scanf("%lf", &rate);

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

💡 每年缴费2万，缴费35年，退休后每个月给你发8千，保险公司会赔钱吗？（年利率5%）