



北京航空航天大学
BEIHANG UNIVERSITY



程序设计基础

Fundamentals of Programming

北京航空航天大学 程序设计课程组

软件学院 谭火彬

2022年



北京航空航天大学
BEIHANG UNIVERSITY



第四讲 控制结构

Control Flow

- ◆ 结构化程序设计
- ◆ 选择结构：if、if-else、switch
- ◆ 循环结构：for、while、do-while



提纲：控制结构

◆4.1 结构化程序设计

◆4.2 选择结构

- ✓if

- ✓if – else

- ✓switch

◆4.3 循环结构

- ✓while

- ✓for

- ✓do - while



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



程序中存在大量的问题！

老师，我的本地过了，为什么还是WA！

为什么提交上去就是OE啊，有人可以解答一下吗？

REP是什么？没见过啊！

AK宝典 | 观看《AK宝典》，告别WA声一片

原创 士谔宣传媒体中心 北航士谔书院

2-10 19

收录于话题

AK宝典

10个 >



E 提交记录:

ID	结果	时间	内存
4151411	AC	4	1700
4149114	REP	131	1696
4149072	REP	181	1708
4148649	REG	1281	1348
4148643	REG	1676	1392
4148459	REG	1252	1400

F 提交记录:

ID	结果	时间	内存
4151876	WA	11	1732
4151850	WA	18	1728
4151807	WA	1515	7116
4151748	WA	98	5636
4151733	WA	24	5632
4151709	WA	677	1688
4151700	WA	678	1708
4151664	WA	486	1704
4151605	WA	685	1720
4151598	WA	678	1664

程序中的错误是无法避免的，但通过一些方法（规则）可以尽量少犯错误，而且出错后能够定位错误



4.1 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
- ◆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、循环三种基本控制结构来构造
 - ✓结构化编程的理论基础：任何可计算的函数都可以通过顺序、选择、循环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

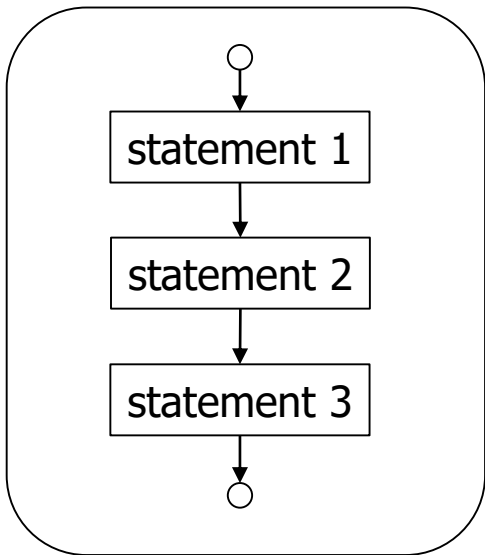
与Knuth并称为我们这个时代最伟大的计算机科学家的人

C语言是典型的结构化程序设计语言

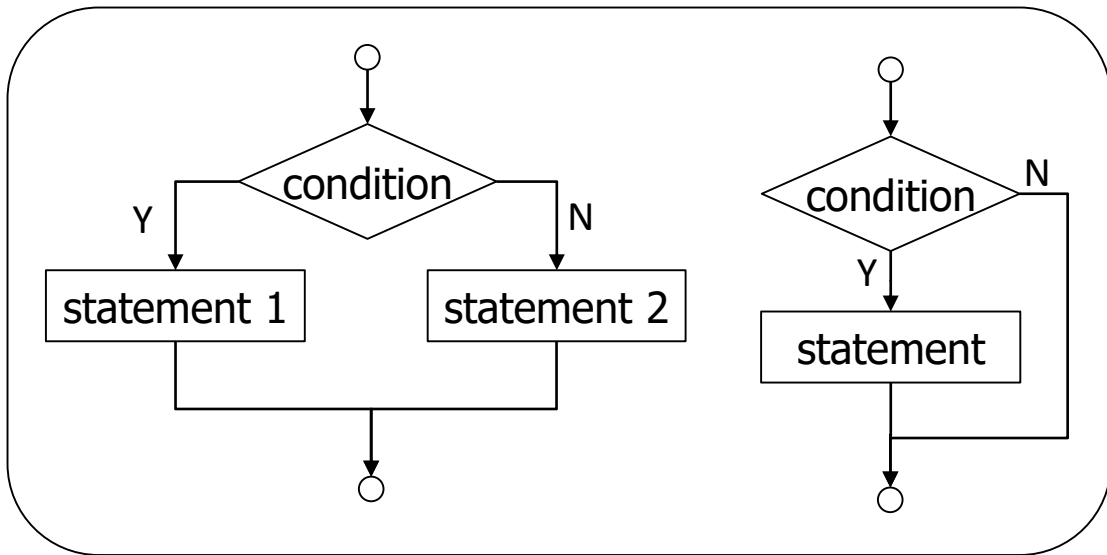


结构化程序设计

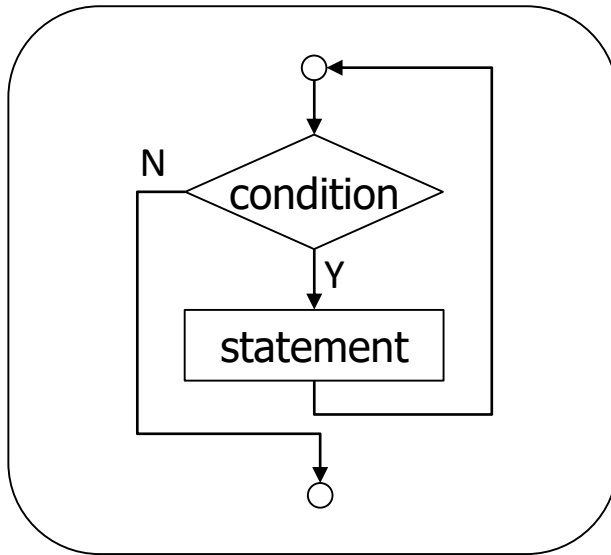
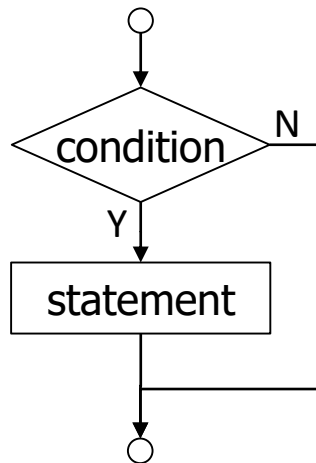
◆ 结构化程序的三种控制结构：



顺序 (sequence)



选择 (selection)



循环 (repetition)

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



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？

◆短期来看的循环结构：

每天在循环 起床->吃饭->干活/学习->吃饭->干活/学习->吃饭->睡觉

在正能量的循环中，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语言中的三种控制结构

- ◆ **顺序**结构：除非遇到选择或循环结构，语句都是顺序执行的
- ◆ **选择**结构：if, if/else, switch
- ◆ **循环**结构：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

```
#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;
}
```

顺序

```
#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;
}
```

选择

```
// 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;
}
```

循环



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）

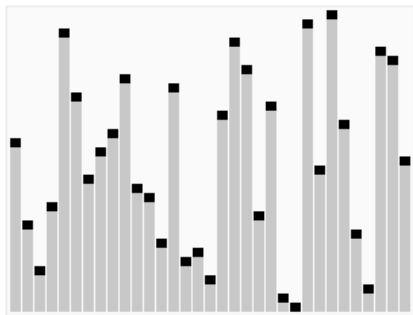
◆结构化程序设计=算法+数据结构



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

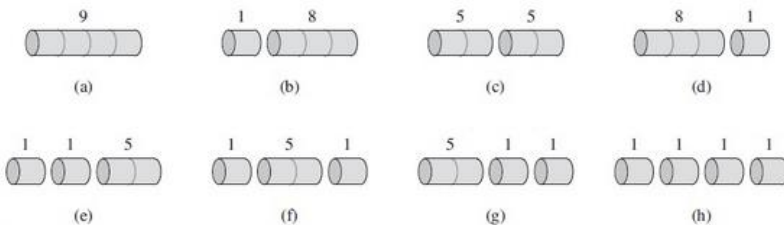
13	-3	-25	20	-3	16	-23	18	20	-7	12	-5	-22	15	-4	7
----	----	-----	----	----	----	-----	----	----	----	----	----	-----	----	----	---

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



最少硬币找零

◆ 贪心算法

- ✓ Kruskal最小生成树
- ✓ 活动选择

◆ 回溯算法

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

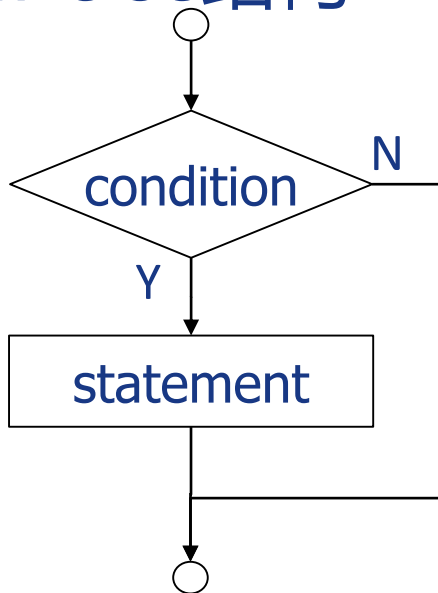


4.2 选择结构

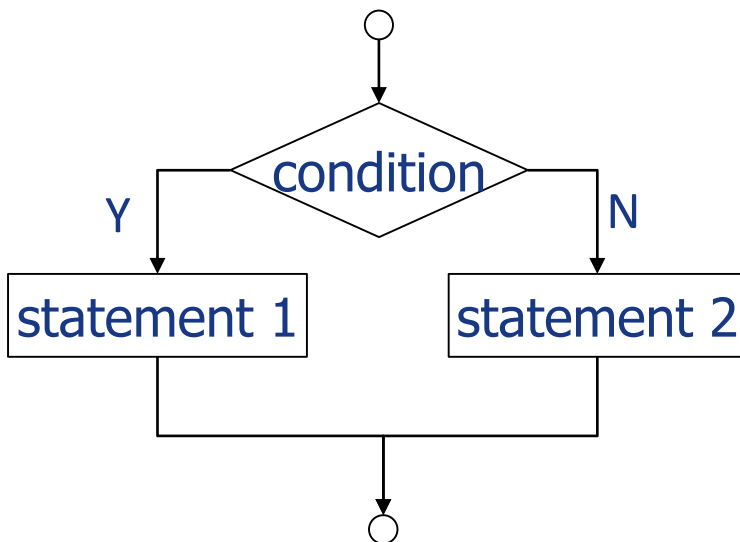
◆根据条件选择执行语句（执行0-1次）

✓if结构

✓if-else结构



```
if(<condition>)  
    <statement>
```



```
if(<condition>)  
    <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；否则称条件为真



条件表达式

- ◆当<条件表达式>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");
```



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

提示：if(5) 等价于 if(1)

建议做法
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

```
if (!x)
```



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




常见的条件表达式

◆示例1：判断数n是否大于1并小于等于10： $1 < n \leq 10$

```
(1 < n <= 10) //错误?
```

```
( (n > 1) && (n <= 10) )
```

◆示例2：判断字符变量ch是否是字母

```
((ch >= 'a') && (ch <= 'z')) || ((ch >= 'A') && (ch <= 'Z'))
```

◆示例3：判断gcd不是a和b的公约数

```
(!(a % gcd == 0) && (b % gcd == 0))
```

◆示例4：判断year年是否是闰年（能被4整除且不能被100整除；或者能被400整除）

```
((year%4 == 0) && (year%100 != 0)) || (year%400 == 0)
```



条件运算符

- ◆ 条件运算符 (`_ ? _ : _`)
- ◆ 语法格式为: `condition ? expression 1 : expression 2`
 - ✓ 当`condition`表达式为真时, 执行表达式`expression 1`并将它的值作为整个表达式的值; 否则执行表达式`expression 2`并将它的值作为整个表达式的值
- ◆ C语言唯一的三目运算符, 可以简化if/else描述

```
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");
```



条件语句的嵌套

◆C04-01：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

如果成绩大于等于90

输出A

否则

如果成绩大于等于80

输出B

否则

如果成绩大于等于70

输出C

否则

如果成绩大于等于60

输出D

否则

输出F



C04-01: 成绩分段打印

如果成绩大于等于90

输出A

否则

如果成绩大于等于80

输出B

否则

如果成绩大于等于70

输出C

否则

如果成绩大于等于60

输出D

否则

输出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");
```

```
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");
```

提示: 更好的书写风格?

构成了 if ... else if ... else if ...else结构!



条件语句的嵌套

◆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，它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，从而尽早退出判断

◆如：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？

```
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的匹配问题

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

```
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



复合语句块

```
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，**一定要用大括号括起来**
- ◆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**语句块**



一元二次方程

◆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?

```
#include <stdio.h>
#include <math.h>
int main(){
    double a, b, c, r1, r2, dt;
    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("two real root: %f, %f\n", r1, r2);
    }
    else if (0.0 == dt)
        printf("one real root: %f\n", -b / (2 * a));
    else
        printf("no real root \n");
}
```

找bug



浮点数的精度问题

```
#include <stdio.h>
#include <math.h>
int main(){
    double a, b, c, r1, r2, dt;
    double eps = 1e-9;
    scanf("%lf%lf%lf", &a, &b, &c);
    dt = b * b - 4 * a * c;
    if (fabs(dt) < eps)
        printf("one real root: %f\n", -b / (2 * a));
    else if (dt > 0.0) {
        r1 = (-b + sqrt(dt)) / (2 * a);
        r2 = (-b - sqrt(dt)) / (2 * a);
        printf("two real root: %f, %f\n", r1, r2);
    }
    else
        printf("no real root \n");
}
```

思考:

1. 如何判断dt是否为0?
2. 调整了if判断的顺序: 先判断是否为0, 再考虑大于、小于0的情况, 为什么?



C04-02: 四则运算

- ◆C04-02 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，则输出错误提示信息)



C04-02: 四则运算

```
#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)
    else {
        printf("invalid expression: %c\n", op);
        return 1;
    }
    printf("%6.2f %c %6.2f = %12.4f\n", x, op, y, r);
    return 0;
}
```

程序的严重问题!
(不是小瑕疵)

◆ 如果if(op == '-')写成if(op = '-') ?

◆ 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



多路选择：switch语句

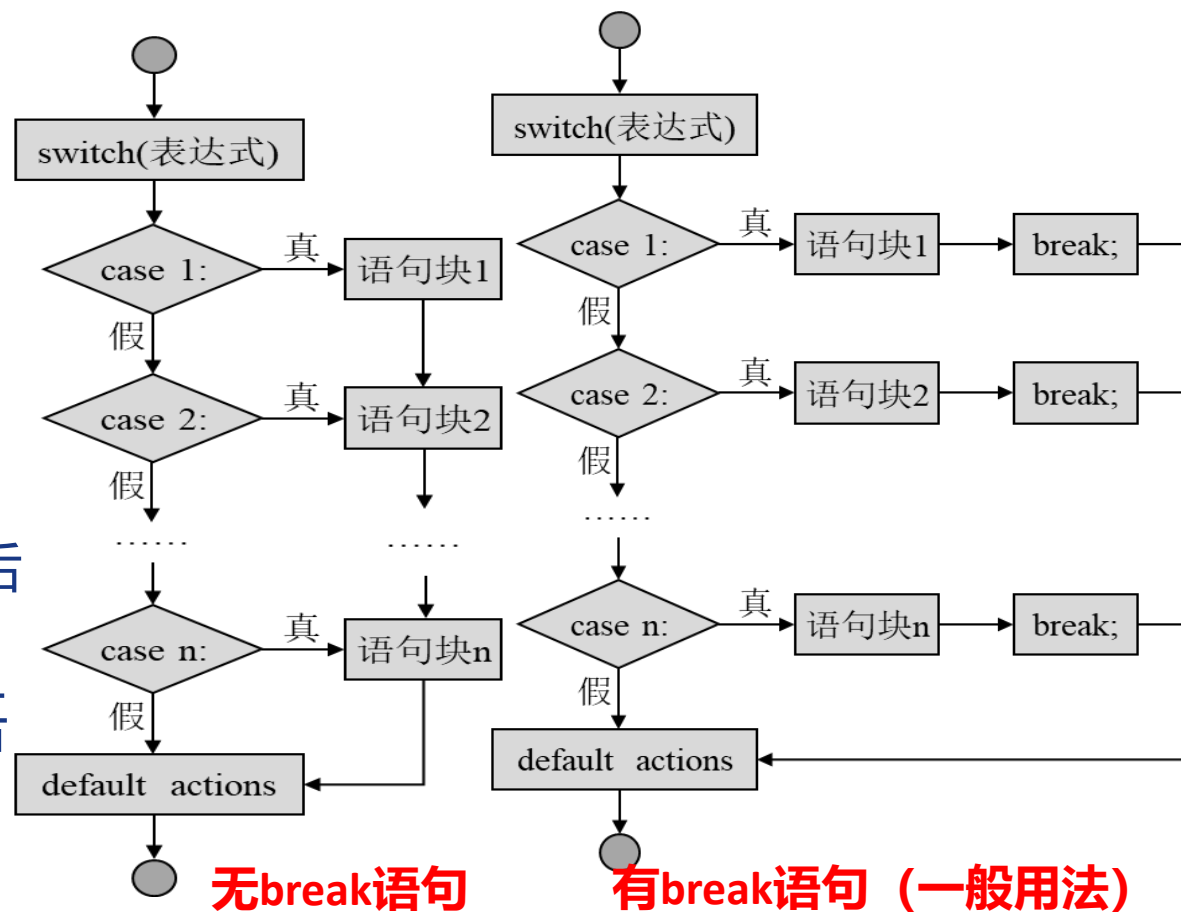
- ◆ if-else只能实现双向选择，通过嵌套 if-else-if结构可实现多路选择
- ◆ switch语句可以实现针对整数表达式的多路选择
- ◆ 语法格式如下（[]表示里面的内容可以省略，实际代码不需要写[]）

```
switch (整数表达式)
{
    case 常量表达式1: 语句块1 [break;]
    case 常量表达式2: 语句块2 [break;]
    .....
    .....
    case 常量表达式n: 语句块n [break;]
    [default: 语句块n+1]
}
```




Switch语句的执行逻辑

- ◆ switch括号中为整数表达式，其返回值应为**整数（或字符类型）**
- ◆ case中为**常量表达式**，是一个常量值
- ◆ case跳转**只发生一次**
 - ✓ 即在switch(表达式)时，与依次与后面的case标记比较，找到第一个匹配的case，即跳转执行后面的语句块
 - ✓ 执行语句块后，如果没有break则继续执行后面的语句块，不再做后续判断
- ◆ 与所有case都不匹配时，执行default语句（可省掉，一般建议写上）
- ◆ break语句使程序退出switch，case中有多个语句时，不必放在{ }中





C04-03: 成绩分段打印(switch实现)

◆C04-03: 成绩大于70到79分之间打印

```
int score;
switch (score / 10) {
case 10: //高位为100
case 9:  //高位为9
    printf("A\n");
    break; //跳出switch
case 8:  //高位为8
    printf("B\n");
    break; //跳出switch
case 7:  //高位为7
    printf("C\n");
    break; //跳出switch
case 6:  //高位为6
    printf("D\n");
    break; //跳出switch
default: //其他情况
    printf("F\n");
    break; //跳出switch
}
```

在



C04-04: 四则运算 (switch实现)

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

```
int main(){
```

```
    char op;
```

```
    double x, y, r;
```

```
    double eps = 10e-9;
```

```
    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)
```

```
            {
```

```
                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;
```

```
}
```



C04-05：今天星期几？

◆综合应用：输入某一天（输入格式为yyyymmdd，如：1999年10月1日应输入为19991001），输出该天是星期几的英文缩写？

◆问题分析：如何获得某天是星期几？

✓方案一：已知某一天是星期几，一天天类推后续的日期的星期

➢如：已知1900年1月1日是星期一，根据相差的天数求解之前或之后日期的星期

✓算法：蔡勒 (Zeller) 公式

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

✓W：星期，0-星期日，1-星期一，...

✓C：世纪值，即：年/100 的值，公元前为负数，如公元前253则为-3

✓Y：年的后两位，公元前则为 $c \bmod 100 + 100$

✓M：月，取值为3-14，其中1、2月需换算为前一年13、14月，即2003年1月1日需按2002年的13月1日计算

✓D：日



C04-05: 今天星期几

```
#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;
        //限制日期范围, 不考虑公元前
        if (!(longday >= 101 && longday <= 99991231))
        {
            printf("Wrong input format, try again!\n");
            continue;
        }
    }
}
```



C04-05: 今天星期几 (续)

```
y = longday / 10000;
m = (longday % 10000) / 100;
d = longday % 100;
if (m < 3)
{ // 1、2月换成前一年13、14月
    y = y - 1;
    m = m + 12;
}

c = y / 100; //世纪
y = y % 100; //年份后两位
           //蔡勒公式求星期
w = (y + y / 4 + c / 4 - 2 * c + (26 * (m + 1)) / 10 + d - 1) % 7;
if (w < 0)
    w += 7; //如果是负数，则+7
```




C04-05: 今天星期几 (续)

```
printf("The day is: ");
switch (w) //打印星期
{
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;
}
```



4.3 循环结构

◆循环结构

- ✓ 星期计算除了用Zeller公式，还可以从1900年1月1日一天天算后续日期是星期几，而且能很快计算出来，这是简单的算法思路，这依赖于循环结构
- ✓ 借助于循环结构，可以让计算机毫无怨言地重复做任何事情，这是计算机的优势：**快速的重复运算**

◆常见的循环问题

- ✓ 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108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
- ✓ 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）

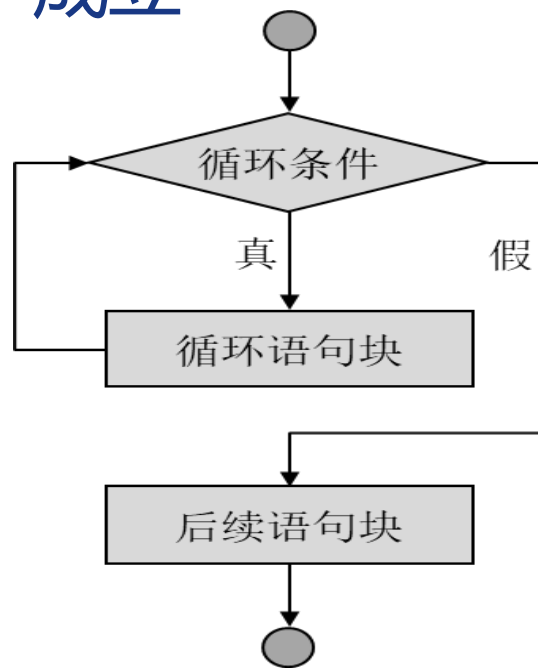


while循环结构

◆while循环

- ✓当循环条件成立时，执行循环体内的“循环语句块”，执行完成后，重新判断循环条件，如果循环条件还成立则重复执行“循环语句块”，如此循环，直到循环条件不成立

```
while (循环条件) //循环头
{
    循环语句块    //循环体
}
后续语句块
```





C04-06: 求大于n的最小的2的指数值

◆输入整数n，输出大于n的最小的2的指数值

◆问题分析：

✓为了求2的指数值，需要重复进行乘2的操作，从2开始，一直累乘，直到第一个大于n的数出现

✓循环条件是得到的结果小于等于n

```
int n, product = 2;
scanf("%d", &n);

while (product <= n) //当product小于等于n时，循环执行
    product = 2 * product;

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



特殊的while循环

◆while循环体可以为空

```
while(循环条件)
{
}
```



```
while(循环条件);
```

◆示例

- ✓清空输入缓冲区直到新的一行开始
- ✓即将当前一行的输入全部忽略

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



C04-07: 最大公约数

◆输入两个正整数，求它们的最大公约数

✓注意：与第一章的方法不同

➤此处从最小的可能开始找

➤第一章从最大的可能开始找

```
#include <stdio.h>
int main()
{
    int a, b, gcd, i = 1;
    scanf("%d%d", &a, &b);
    //从1开始，求最大公约数
    while(i <= a && i <= b)
    {
        if(a%i == 0 && b%i == 0)
            gcd = i;
        i++;
    }
    printf("gcd is: %d\n", gcd);

    return 0;
}
```



复习：while循环输入多组数据

◆输入若干行数据，每行两个数a和b，求a+b的值

```
#include <stdio.h>

int main()
{
    int a, b;
    int sum;
    //循环读入数据，直到没有数据可读入(EOF)
    while(scanf("%d%d",&a, &b)!=EOF){
        sum = a + b;
        printf("%d\n", sum);
    }

    return 0;
}
```

**提示：scanf()函数有返回值，若输入成功，则返回输入的数据项数，成功读到文件结束末尾时返回 EOF (-1)，输入错误返回0
本地模拟输入结束方式：输入完成后回车，再输入ctrl+Z**



C04-08: 测试不间断输入

- ◆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
- ◆scanf输入是如何判断出错
 - ✓返回值：返回的值即为成功输入的个数，-1表示输入结束

```
#include <stdio.h>
int main() {
    int a;
    printf("\ninput an integer: ");
    while(scanf("%d", &a) > 0) //返回值应等于参数个数
    {
        printf("valid input: %d\n", a);
        printf("\ninput an integer: ");
    }
    printf("invalid input! quit!\n");
    return 0;
}
```

注意：当scanf输入出错时，出错的内容仍在缓冲区，后续读取会继续尝试读入！



C04-09: 控制输入数据

◆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

```
#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()从输入出错后，内容还在输入缓冲区，后续读入还在出错的位置继续！

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！



C04-10: 输入行数统计

◆输入行数统计： 输入一段文本，统计行数

```
#include <stdio.h>

int main()
{
    int c, n=0; //初始化行数为0
    //输入没有结束，循环
    while((c = getchar()) != EOF)
        if(c == '\n') n++; //回车则行数加1

    printf("输入文本行数: %d\n", n);
    return 0;
}
```

注意：输入的最后一行文本必须要有回车，否则统计行数出错！



循环条件的注意事项

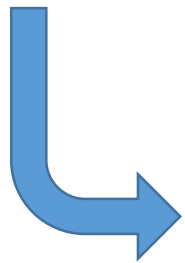
- ◆ 控制变量（或循环计数器）的名称
- ◆ 控制变量的初始值
- ◆ 测试控制变量终值的条件（即是否继续循环）
- ◆ 每次循环时控制条件修改



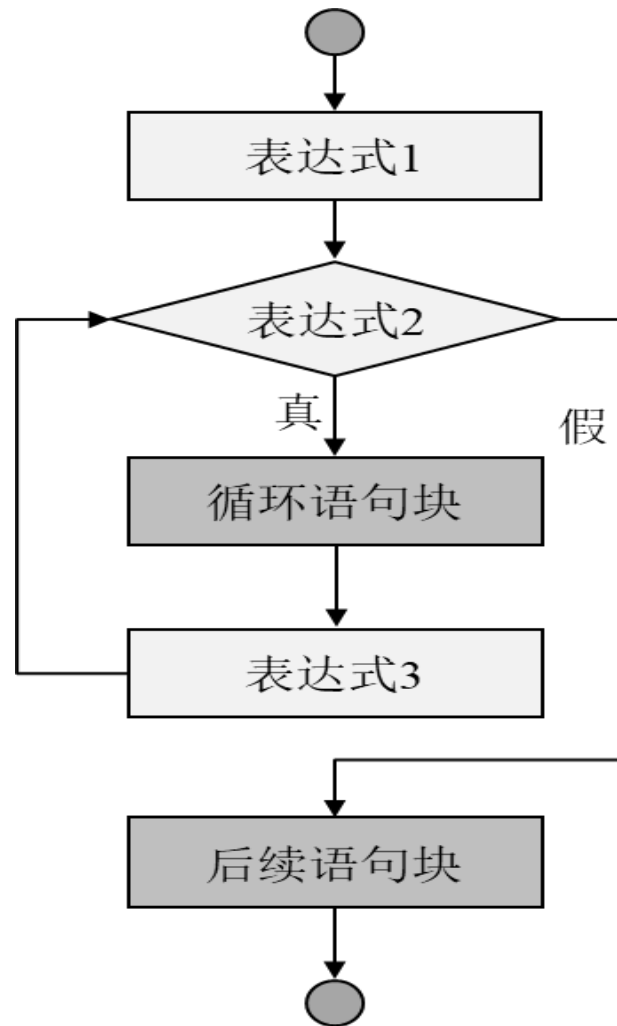
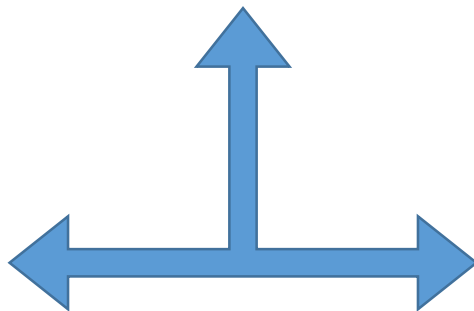
for循环结构

◆将while循环中初值、终值和循环条件的修改都放在循环头

```
for (表达式1; 表达式2; 表达式3) //循环头
{
    循环语句块 //循环体
}
后续继语句块
```



```
表达式1
while (表达式2)
{
    循环语句块
    表达式3
}
后续继语句块
```





for循环头：3个表达式均可省略

for (表达式1; 表达式2; 表达式3)
语句块

for (; 表达式2; 表达式3)
语句块

for (表达式1; ; 表达式3)
语句块

for (表达式1; 表达式2;)
语句块

for (; ;)
语句块

注意：每个表达式都可以省略，但分号必须保留！用于区分3个表达式！



for循环头：使用逗号运算符在表达式中写多个语句

◆逗号运算符：由逗号分割的多个表达式

- ✓语法上看成是一个整体，**是一个表达式**；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

◆for循环头的每部分，可以通过逗号运算符连接多个表达式

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



```
a = x;  
b = y;  
r = c = z;
```

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



简单的for循环示例

◆N个自然数累加和

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



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

◆N的阶乘

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

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;
```



C04-11: 斐波那契数列

- ◆ **斐波那契** (Fibonacci) 数列, 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:
 - $F(1)=1, F(2)=1, F(n)=F(n-1)+F(n-2) (n \geq 3, n \in N^*)$
 - ✓ 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



C04-11: 斐波那契数列

- ◆C04-11: 求斐波那契数列第n项 ($n > 3$)
- ◆问题分析: 从第1项、第2项开始, 按照公式, 依次求解第3、4、...、n项

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



C04-11-b: 斐波那契数列

◆输出斐波那契数列的前n项 ($n < 50$)

```
#include <stdio.h>
#define LEN 50
int main()
{
    long long fibo[LEN];
    int n, i;
    scanf("%d", &n);
    fibo[0] = 111; //long long类型
    fibo[1] = 111; //long long类型
    for(i=2; i<n; i++){
        fibo[i] = fibo[i-1] + fibo[i-2];
    }
    for(i=0; i<n; i++)
        printf("%lld\n", fibo[i]);
    return 0;
}
```

**注意：此处使用long long类型，
因为：斐波那契数列增长很快，40几项就超出整数范围，100项超出long long范围！！！！**



C04-12: 最大公约数 (辗转相除法)

- ◆ 输入两个整数 a 和 b ，用辗转相除法求它们的最大公约数
 - ✓ 支持 0 或负数
 - ✓ 比之前的枚举法，效率更高，时间更短
- ◆ 定理： $\gcd(a, b) = \gcd(b, a \% b)$

先证明：

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 整除 (商为 $x - m * y$)，

即： g 也是 b 和 $a \% b$ 的公约数，

即： $g \leq g_1 = \gcd(b, a \% b)$



C04-12: 最大公约数 (辗转相除法)

算法：辗转相除算法gcd(a, b)

1. 如果 $b==0$ ，则：gcd(a, b)为a，结束
2. $\text{if}(a\%b==0)$ ，则：gcd(a, b)为b，结束
3. 否则： $a = b, b = a\%b$ ，回到第2步

思考：如何求最小公倍数？慢方法？快方法？

```
#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;
}
```



更多的实现方式

```
#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;
}
```

提示：此处while结构更清晰

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



C04-13: 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 $153 = 1^3 + 5^3 + 3^3$

```
#include <stdio.h>
int main(){
    int 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;
}
```



C04-14: 复利计算

◆120年前，Bob因为躲避战争，逃难到瑞士，在瑞士银行存了10000美元（假设利率较高，年利率0.075），后来Bob忘记了这笔钱，现在银行找到Bob的法定继承人Alice，问：Alice可获得多少钱？

分析：固定本金，每年计算利息，然后加到本金上，本金加利息就是下一年的本金；这是一个典型的利上生利问题。设base (p) 表示本金，r表示年利率，则第n年的资产总额：

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

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

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

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

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

...

```
int base = 10000, n, i;
double rate = 0.075, sum = base;
for (i = 1; i <= 120; i++)
{
    sum = sum * (1 + rate);
}
printf("\n%12.2f $\\n", sum);
```



C04-15: 保险理财

◆假如每月交固定额度的养老保险金base，连续交n年，n年后总共有多少保险金（假设每月利息为0.6%，且这样的利率一直保存不变）？

分析：

设sum表示养老保险金总和（本金+利息，sum初始值为0），base为每月初新存入金额，rate为每月利率，则：

第一月底的保险金为 $sum = (sum + base) * (1 + rate)$ ；
第二月底的保险金为 $sum = (sum + base) * (1 + rate)$ ；
依次类推，可以通过循环方式计算n年后（12*n月）的养老保险金。

由于循环执行次数是确定的，适合采用for中的计数控制结构

```
int base, n, i;
double rate = 0.005, sum = 0;
scanf("%d%d", &base, &n);
for (i = 1; i <= 12 * n; i++)
{
    sum = (sum + base) * (1 + rate);
}
printf("\n%15.2f $\n", sum);
```




C04-16: PI的计算

◆PI的计算：马青公式（精确到小数点后第e位）

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

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

◆ 问题分析：

$$\text{pi} = 16 * A - 4 * B$$

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



C04-16: PI的计算

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

循环结束条件由精度控制

精度条件:

$16 * 1 / (k * 5^k) > \text{eps} / 2$ 且 $4 * 1 / (k * 239^k) > \text{eps} / 2$
因为 $a \leq \text{eps} / 2$, 且 $b \leq \text{eps} / 2$, 即: $a + b \leq \text{eps}$

思考:

- 1) 此例程可进一步优化, 内层嵌套的for循环可以去掉!
- 2) 两个for循环很类似? 可以简化吗?

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

```
int main(){
```

```
    int i, j, k, sign = -1;
```

```
    double n, d, s1 = 0, s2 = 0, eps = 1e-15;
```

```
    for(i = 0, d = 1; 16*d > eps/2; i++) {
```

```
        sign = i%2 == 0 ? 1 : -1; //符号位
```

```
        k = 2*i + 1;
```

```
        for(j = 0, n = 1.0; j < k; j++)
```

```
            n *= 5;
```

```
        d = 1.0 / (k*n);
```

```
        s1 += d*sign;
```

```
    }
```

```
    for(i = 0, d = 1; 4*d > eps/2; 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);
```

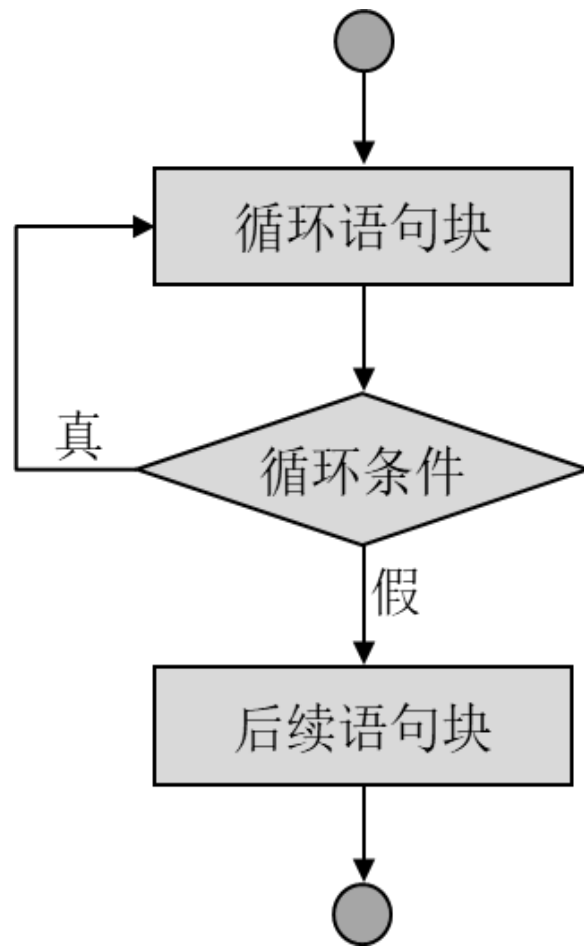
```
}
```



do-while循环结构

- ◆ do-while循环：类似于while，不过先执行循环体，再判断循环条件
- ◆ do-while循环体至少会执行一次

```
do  
{  
    循环语句块 //循环体  
} while (循环条件);  
后续继语句块
```





各种循环结构：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;
}
```



选择合适的循环结构

- ◆ while, for, do while 三种循环控制语句本质上相同
- ◆ 选择合适的循环结构
 - ✓ for 中循环控制变量的初始化和修改都放在语句头部分，形式较简洁，它比较适用于循环次数确定的情况
 - ✓ while 中循环控制变量的初始化一般放在 while 语句之前，循环控制变量的修改一般放在循环体中，形式上不如 for 语句简洁，但它比较适用于循环次数不确定的情况；
 - ✓ do...while 中循环控制变量的初始化一般放在 do 语句之前，循环控制变量的修改一般放在循环体中，但它先执行完循环体、再判断循环条件
- ◆ 根据实际情况选择合适的循环结构



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
- ◆C04-17：输出平行四边形图案

输入整数 n (1~20)，输出高为 n 、宽度为 $2n$ 、斜边斜率为 45° 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：



C04-17: 输出平行四边形图案

```
#include <stdio.h>
int main(){
    int i, j, n;
    scanf("%d", &n);
    for (i = 0; i < n; i++)//打印一行
    {
        for (j = 0; j < n - i; j++)//打印前面的空白字符
            putchar(' ');
        for (j = 0; j < 2 * n; j++)//打印后面的星号*
            putchar('*');
        putchar('\n');//打印回车
    }
    return 0;
}
```

思考：如果最后一行不要求输出换行，
该怎么处理？

提示：putchar()的用法

```
8
      *****
     *****
    *****
   *****
  *****
 *****
*****
```



C04-18: 整数分解

◆整数分解：找出和为x的所有正整数序列

- ✓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 $2+3+4$

◆问题分析

- ✓从1开始累加到目标数
- ✓再从2开始累加到目标数
- ✓如此继续，直到 $n/2$

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



C04-18: 整数分解 (续)

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 $i+1 \dots s/2$

```
#include <stdio.h>
int main(){
    int s, i, j, k, st, sum, got = 0;
    scanf("%d", &s);
    for(i=1; i<=s/2; i++){
        sum = i; //从1开始, 一个个累加
        for(j=i+1; sum<s; j++){//累加不超过s
            sum += j;
            if(sum == s) {//找到累加相等的值
                st = i; // st表示开始的序列, 打印
                printf("%d = %d", s, st++);
                for(k=st; k<=j; k++) printf(" + %d", k);
                putchar('\n');
                got++; //计算次数
            }
        }
    }
    printf("cases: %d\n", got);
}
```



C04-19: 阶乘之和

◆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++) { //求和
        for (r = j = 1; j <= i; j++) //求阶乘
            r *= j;
        sum += r;
    }
    printf("sum: %lld\n", sum);
    return 0;
}
```

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



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



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



循环语句的非常规控制

◆C语言中提供了break和continue语句来改变循环控制流程

- ✓break语句在while、for、do...while或switch结构中使用，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
- ✓continue语句在while、for或do...while结构中执行时，用于跳过当前循环体中的后续的语句，进入下一次循环判断

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

输出: 1, 2, 3, 4,

```
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,



使用非常规控制后，程序的执行逻辑发生变化


◆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

◆自然数1~n进行累加

✓此处不建议使用break语句，使用正常的循环条件控制退出循环

```
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;
}
```



C04-20: 整数分解 (最长序列)

◆整数分解：找出和为x的所有正整数序列，如果有多个序列满足条件，则输出**最长序列**

◆问题分析

- ✓哪个是最长序列？
- ✓从1开始找的话，第一个就是最长序列
- ✓找到第1个序列后，直接退出循环

输入15，输出为：

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

$$15 = 4+5+6$$

$$15 = 7+8$$

cases: 3



C04-20: 整数分解 (最长广子)

提示: 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;
}
```




C04-21: 乒乓球赛

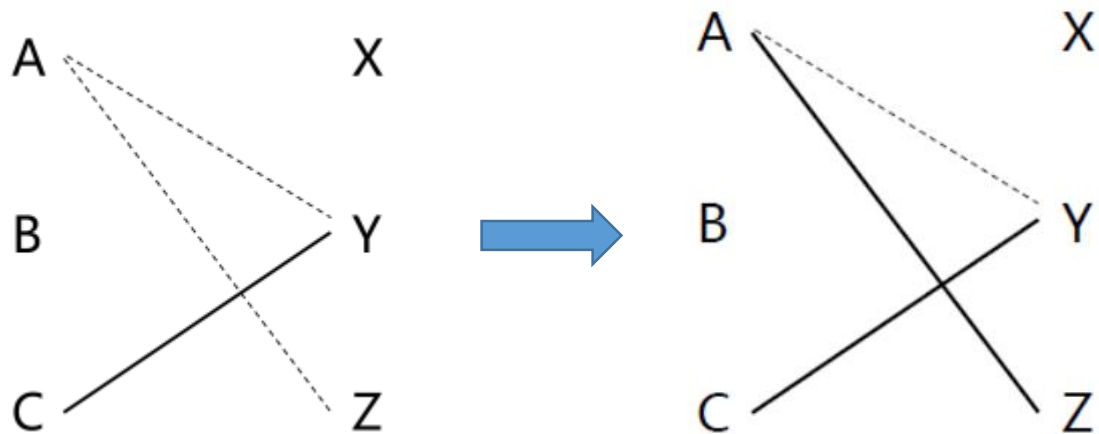
◆乒乓球赛：甲队的A、B、C与乙队的X、Y、Z比赛，已知A不与X对阵，C不与X和Z对阵，输出对阵名单

◆人的推理过程

- ✓A不对阵X，则A可能对阵Y或者Z
- ✓C不对针X和Z，则C只能对阵Y
- ✓由此可知，A只能对阵Z
- ✓最后：B只能和X对阵

◆计算机如何推理？

- ✓只会最简单的方法，逐一可能尝试
- ✓最简答的算法：枚举所有的可能，之后排除不符合要求对阵，剩余即为对阵





C04-21: 乒乓球赛

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) //对阵选手不重复
                continue;
            if (A != 'X' && C != 'X' && C != 'Z') //排除题目要求的组合
                printf("A vs %c\nB vs %c\nC vs %c\n", A, B, C);
        }
```



C04-22: 数据求和

◆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

✓如：a, b, c, d分别为1, 15, 1, 30

```
int a, b, c, d, x, y, sum_x = 0, sum_y = 0;
a = 1; b = 100; c = 1; d = 200; // 根据实际需要进行初始化
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 7

6 5

20 3

输出

9 12



关于循环的非常规控制

- ◆ break和continue破坏了程序的结构化特征
- ◆ 大多数情况下，都可以不需要使用非常规控制语句
 - ✓ 可以通过设置特殊的循环控制标记、修改循环条件等方式来控制退出，但可能是程序结构变得复杂
 - ✓ 当然，一般switch语句中要用到break跳出
- ◆ 使用非常规控制，有时候会使得程序更简洁、增强程序的可读性



古老的goto语句

◆ goto语句：使程序控制流程直接跳转到指定位置（标号）

- ✓ 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）
- ✓ 然而，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！！！！

```
#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');
                goto found;
            }
        } // 2 end
    } // 1 end
    if (sum != s) printf("NONE\n");
found:
    return 0;
}
```



总结：结构化编程中的三种控制结构

◆ Bohm和Jacopini证明，结构化编程只需要三种控制结构

- ✓ 顺序(sequence)
- ✓ 选择(selection)
- ✓ 循环(repetition)

◆ 选择结构的三种实现方法：

- ✓ if（单项选择）； if/else（双向选择）； switch（多路选择）
- ✓ 简单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，均可以组合简单if 结构来实现

◆ 循环结构的三种实现方法：

- ✓ while语句； for语句； do/while语句
- ✓ 简单的while语句即可提供任何形式的循环
 - 即任何能用for 和 do/while语句完成工作，均可以用 while语句来实现

顺序结构

选择结构

if
if-else
switch

循环结构

for
while
do-while