



第四章 (Chapter 4)

结构化编程

structured programming

北京航空航天大学

李莹

2024年10月

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

收录于话题
#AK宝典

10个 >

E 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
4149114	REP	131	1696
4151741	AC	4	1700

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

从开学到现在的编程练习中，
你出现了许多的编程错误，
一度让你怀疑自己的智力，
一度让你想要放弃学习编程！
这一切，都在我们的掌控之中！

程序中的错误无法避免，
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。

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
- 三类结构的**堆叠**和**嵌套** ⇒ **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**

知识点不多;
但内容深邃!
培养算法思想!
训练逻辑思维!

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**

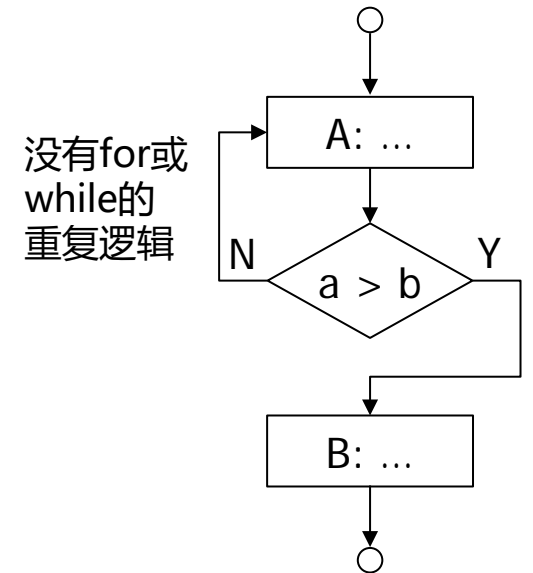
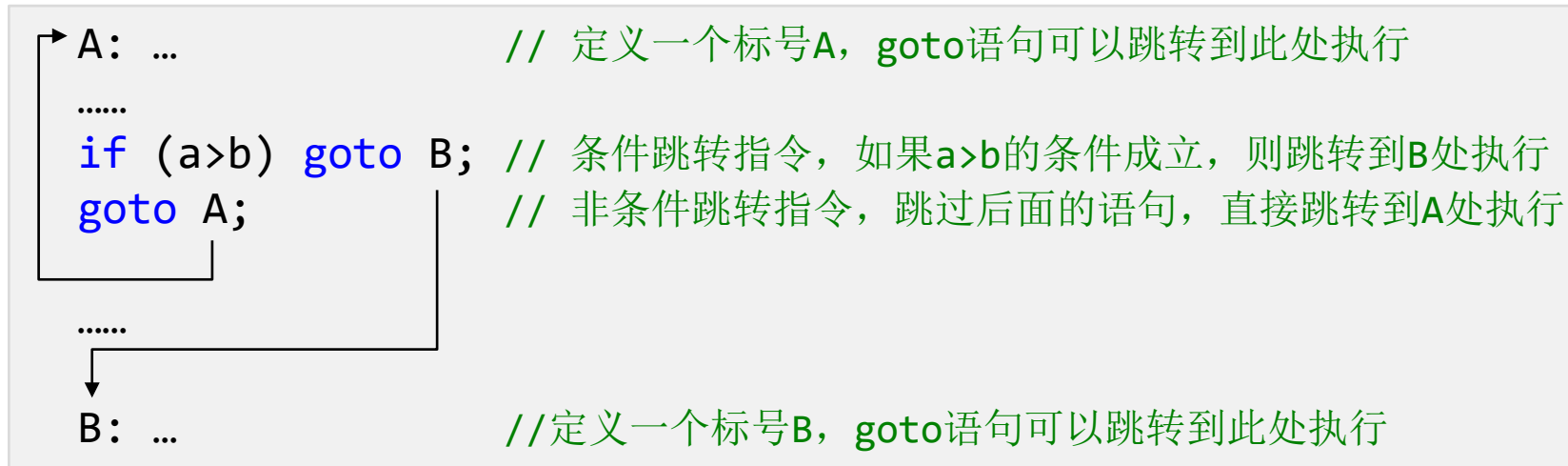
4.4 真正的模块化编程

- 模块化思想
- 函数封装

4.1 从goto到结构化编程

- 程序是指令集合：指令流按顺序执行，通过跳转指令来跳过或者重复执行某些指令
- goto语句是著名的跳转指令

两种类型的跳转指令：非条件跳转和条件跳转



- goto语句破坏程序结构
 - ◆ 滥用goto语句，使得语句间的随意跳转破坏了程序的基本结构
 - ◆ 程序的可读性、可理解性退化到了汇编语言的级别
 - ◆ 造成程序逻辑结构的混乱，很难保证程序的质量

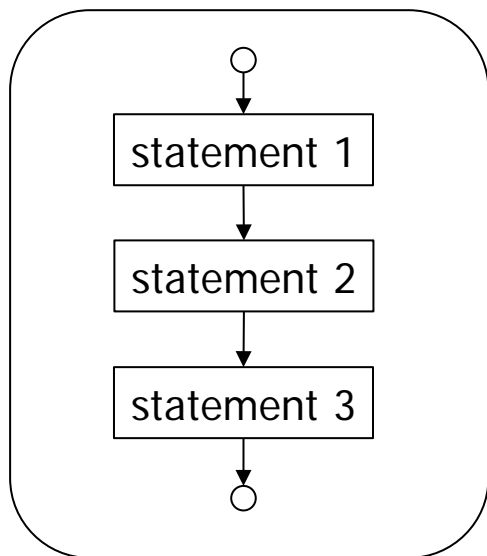
结构化程序设计

- **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。其主要观点是**自顶向下、逐步细化**的程序设计思想和**模块化（结构化）**的设计理念。
- **结构化编程**的理论基础：任何程序都可以通过**顺序、选择、重复**三种基本控制结构及其**嵌套、组合**来构成。
- 将具有特定功能的结构化语句封装成**子程序（函数）**，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。

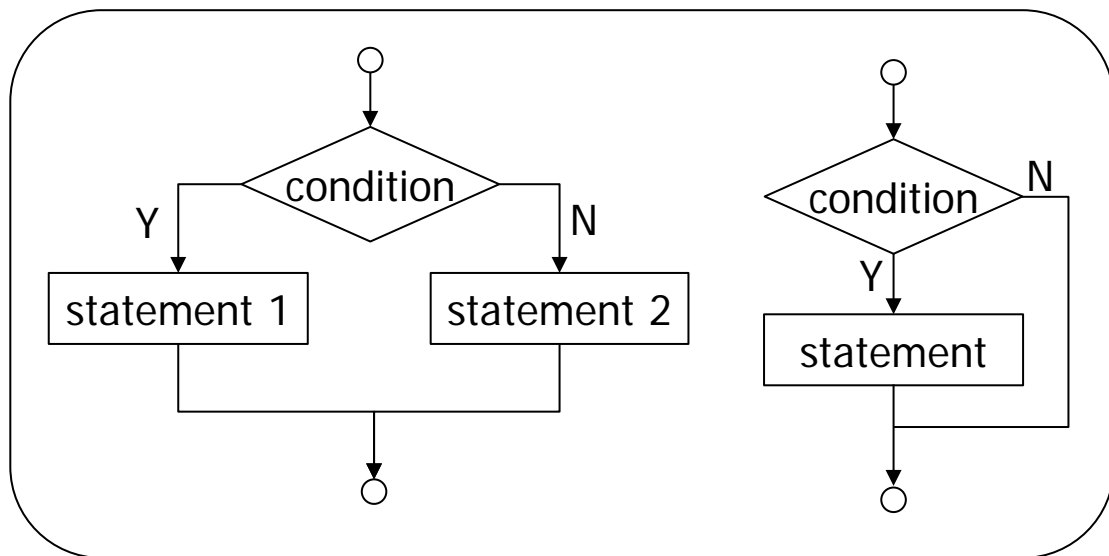
C语言是典型的结构化程序设计语言

结构化程序设计

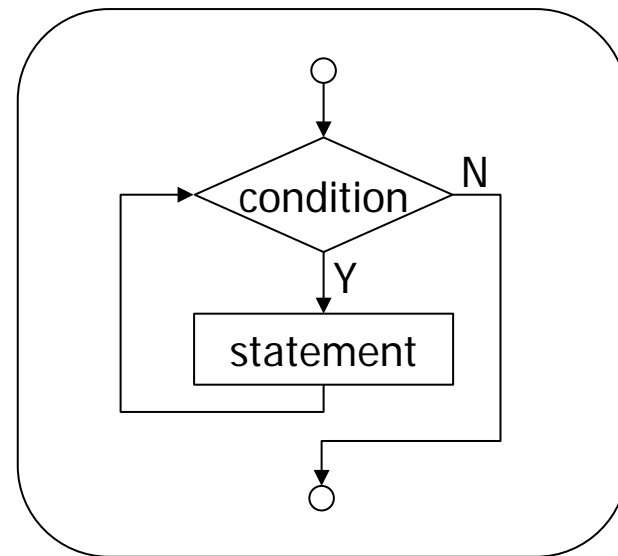
结构化程序的三种控制结构：



顺序 (sequence)



选择 (selection)



重复 (repetition)

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：

要不要学C语言程序设计？要不要每天坚持去<https://accoding.buaa.edu.cn>刷题？考研or工作？

搞金融 or 搞IT or 创业 or 投身航空航天事业？期望月薪8K or 50K or more？

和小张交往or和小李交往？...人人心中都有一个条件决定自己的**选择**？

人的一生是结构化的过程

- 长期来看的**顺序**结构：人的一生是短暂。出生→不虚此行→死亡。
- 中期来看的**选择**结构：人的一生七次改变命运的机会。学会取舍，有舍才有得。

- 短期来看的**重复**结构：每天都很平凡，量变引起质变。

每天在重复：

起床→吃饭→编程/学习→吃饭
↑
睡觉←编程←吃饭←锻炼←编程/学习

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！

```
for(pi=1,sign=1,i=1; i<=n; i++)  
{  
    sign = -sign;  
    pi = pi + sign/(2*i + 1.0);  
} // 计算出神奇的 3.14159265...
```

3.14159265358979323846...

起床→吃饭→上课睡觉→吃饭
↖
睡觉←抄代码←打游戏

负正能量的重复

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4-1: 两数相除

```
// 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;
}
```

顺序

```
...
{
    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

```
...
{
    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;
}
```

循环

C语言的三类控制结构：选择

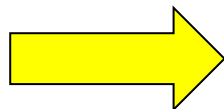
例4-2：求解一元二次方程 $ax^2 + bx + c = 0$ ($a \neq 0$)

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

```
#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;
}
```

顺序：一目了然

选择：
规避
风险



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

```
#include <stdio.h>
#include <math.h>
#define eps 1e-6
int main()
{
    double a, b, c, delta, r1, r2;
    scanf("%lf%lf%lf", &a, &b, &c);
    delta = b * b - 4 * a * c;

    if (fabs(delta) <= eps) // 判断 delta 为 0
    {
        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($1 \leq n \leq 40$): 1, 1, 2, 3, 5, 8, 13, 21, ...

```
#include <stdio.h>
int main()
{
    int fib[41] = {0, 1, 1};
    int n, i;

    scanf("%d", &n);
    for (i = 3; i <= n; i++)
    {
        fib[i] = fib[i-1] + fib[i-2];
        printf("%3d: %d\n", i, fib[i]);
    }
    return 0;
}
```

循环：不知疲倦

输入输出测试

```
10
3: 2
4: 3
5: 5
6: 8
7: 13
8: 21
9: 34
10: 55
```

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：

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

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)

- 三类控制结构写成的简单算法，通过组合，构成具有复杂逻辑的计算过程，这种过程就叫算法。
- **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，包括
 1. 执行的操作
 2. 执行操作的顺序
- **程序设计的核心是算法**，不懂算法的编程者就是码农。
- 优秀的程序员不是码农，是设计师、艺术家！
- **程序 = 算法+数据结构**

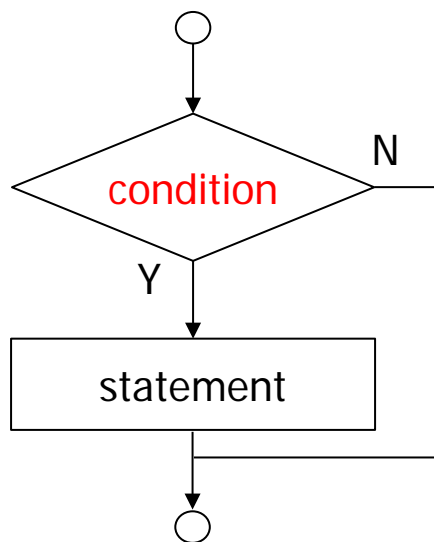
计算机界有这样一种说法：如果说**有一个人因为一句话而得到了图灵奖**，这个人应该就是 Nicklaus Wirth（尼古拉斯·沃斯），这句话就是他提出的著名公式“**算法+数据结构 = 程序**”。这个公式对计算机领域的影响程度足以类似物理学中爱因斯坦的 $E=MC^2$ ，这个公式展示出了程序的本质。
- Niklaus Wirth, 1934年2月15日—2024年1月1日，出生于瑞士，1984年图灵奖得主，Pascal之父，美国国家工程院外籍院士，瑞士工程院院士。

三类控制结构
足以构成任意
复杂的算法



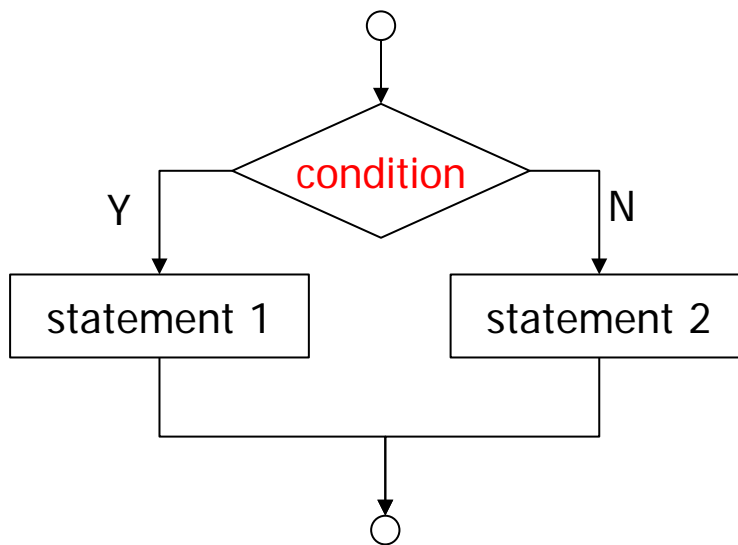
4.2 选择语句（条件语句）

if 选择语句：根据条件值选择是否执行 if 的预计



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

if/else 选择语句：根据条件值选择执行 if 的不同分支



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

1. 任何一个结果为逻辑值的表达式（expression）都可作为 if 的条件（condition），进行 if 分支的选择

【逻辑值，BOOL值，取值为1或0，C语言里用“非0的数值”来模拟逻辑值1】

2. 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)

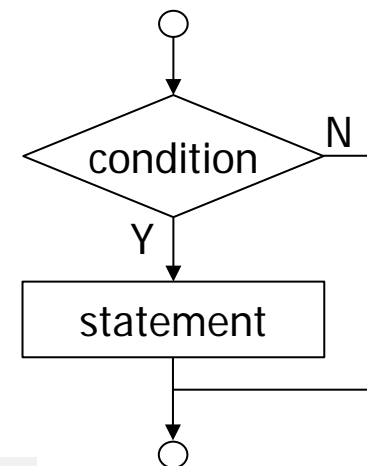
3. statement（语句）可以是单一语句，也可以是{}括起来的复合语句块

if 语句

- if 语句：用于在条件表达式的值为 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(yes)，只有取值为 0 时，才表示条件为假。
【C语言没有BOOL变量类型，通常用整数表示条件的取值，0表示假，非0表示真】



```
if(a + b)  
    <语句>
```

大佬一般这样用，专业、简洁

示例：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。

建议这么写，
因为我们都是一般人。

- 类似地，

```
if(!x)
```



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

一些常用的条件表达式

if 的一些实例

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

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

```
char c;
if (scanf("%c", &c) != EOF && c >= 'a' && c <= 'z')
{
    printf("%c", c);
}
```

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

```
if ((( year%4 == 0) && (year%100 != 0 )) || (year%400 == 0))
{
    leapYear = 1;
}
```


if/else 语句

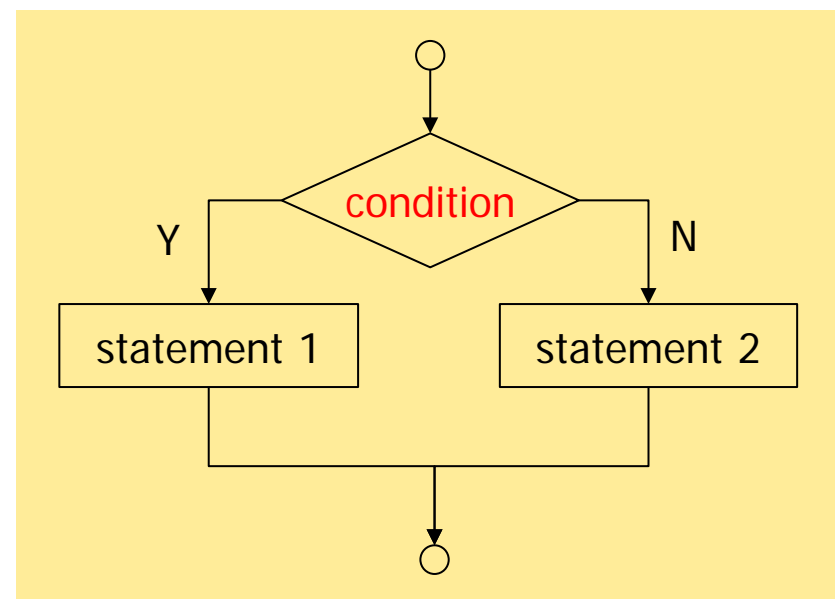
if/else 语句：用于在条件表达式为false (0) 或 true (非0) 时采取不同操作。

■ 语法

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

```
if (/* condition */)   
{  
    /* code */  
}  
else   
{  
    /* code */  
}
```

■ 流程图



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4-4：输出数组x中的前N个数，每行输出5个数

```
for (i = 0; i < N; i++)
{
    if ((i + 1) % 5 != 0)
    {
        printf("%8d", x[i]);
    }
    else
    {
        printf("%8d\n", x[i]);
    }
}
```



```
for (i = 0; i < N; i++)
{
    printf(((i + 1) % 5 != 0 ? "%8d" : "%8d\n", x[i]);
}
```

输出示例（这里的*是通配符，实际是上述代码中x[i]的值，不表示字符星号）

```
* * * * *
* * * * *
* * * * *
* * * * *
```

选择语句中的嵌套

- 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：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
```

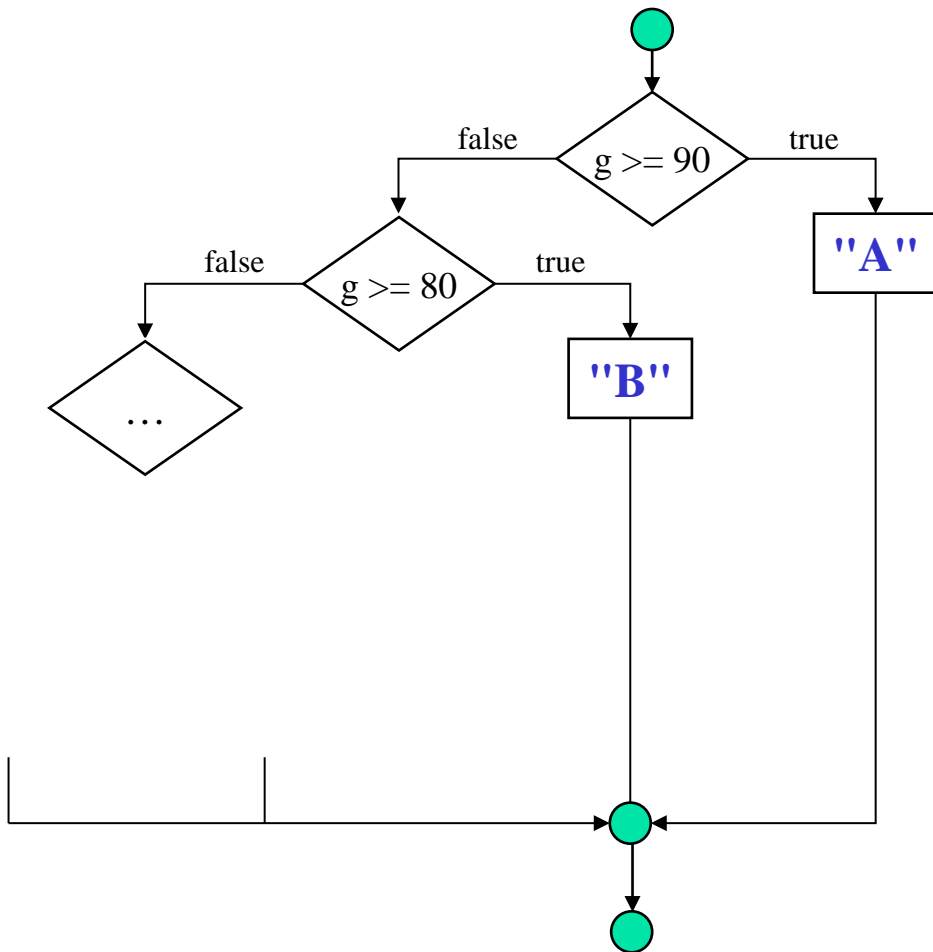
```
            if grade is greater than or equal to 60
```

```
                Print "D"
```

```
            else
```

```
                Print "F"
```

- 完整的流程图？（课后练习）



选择语句中的嵌套

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。

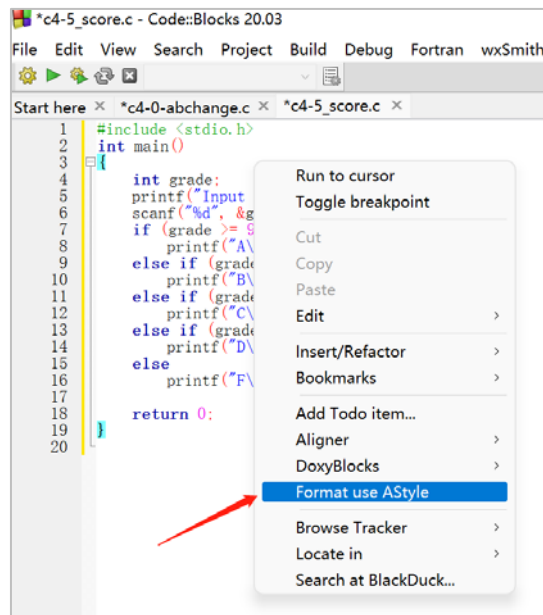
```
if grade is greater than or equal to 90
    Print "A"
else
    if grade is greater than or equal to 80
        Print "B"
    else
        if grade is greater than or equal to 70
            Print "C"
        else
            if grade is greater than or equal to 60
                Print "D"
     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

层次分明

```
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");
```

节省宽度



很多IDE有自动
调整格式功能！

选择语句中的嵌套

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？

注：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");
```

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）

选择语句实例：浮点数的比较

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

分析：先判断 $\text{delta} = b^2 - 4ac$ ，然后求值。

代码中能否写成 `if(delta == 0.0)` ？

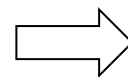
NO

计算机中有些实数的表示不是100%精确，判断实数相等，不要这样用 `if(delta == 0.0)`

怎么解决？

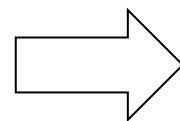
回忆浮点数的精度问题

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



0

```
printf("    %.20f\n", 0.55);  
printf(" + %.20f\n", 0.40);  
for(int i=1; i<27; i++) printf("-");  
printf("\n");  
printf(" = %.20f\n", 0.95);
```



```
0.5500000000000000004000  
+ 0.4000000000000000002000  
-----  
= 0.9499999999999999996000
```

再试试把这几个参数改成15，观察输出情况。

选择语句实例：关系相等与赋值

例4-8：四则运算

```
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) //实数作为输入时，可以直接比较
    r = x / y;
else
{
    printf(" The denominator is 0\n");
    return 1;
}
printf("%6.2f %c %6.2f = %12.4f\n", x, op, y, r);
```

程序是否
有错？

选择语句实例：关系相等与赋值

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。

```
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) //实数作为输入时, 可以直接比较
    r = x / y;
else
{
    printf(" The denominator is 0\n");
    return 1;
}
printf("%6.2f %c %6.2f = %12.4f\n", x, op, y, r);
```

程序是否有错?

```
* 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?

编程提示

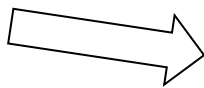
- if (1==gender), 安全性较高的解决方案!
- if (1=gender), 会提示语法错误
- if (gender == 1), 可读性好!
- 当 "a == b" 左右两边都是变量时, 编程人员要格外小心!

这条语句等价于两条语句的逻辑:

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

多路选择结构： switch 语句

多个选择时，太长的 if/else 不够清晰。



```
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");
```

switch 可以方便地进行多项选择，语法为：

```
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 表达式x:
    case 表达式y:
        语句块*** [break;]
    .....
    case 常量表达式n:
        语句块n [break;]
    [default:
        语句块n+1]
}
```

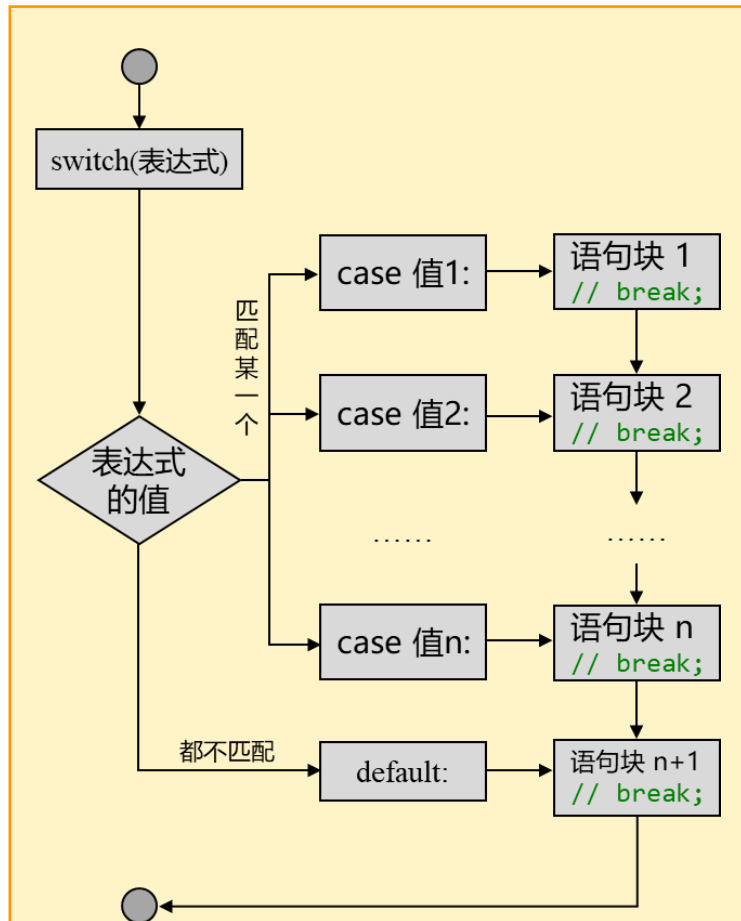
- 表达式与每个 case 标记的表达式比较，若匹配，执行相应的语句
- 连续的 case 之间没有语句，则它们执行同样的语句，如表达式xy
- 与所有 case 都不匹配时，执行 default 对应的语句（有时省掉，作为好习惯，建议保留）
- break 语句使程序跳出 switch, case 中有多个语句时，不必放在{ } 中，因为会按顺序依次执行
- switch 的表达式结果必须是整型值
- case 标记值只能是整型常量或返回常量的表达式（不能是变量）

switch语句的执行逻辑

```
switch (整数表达式)
```

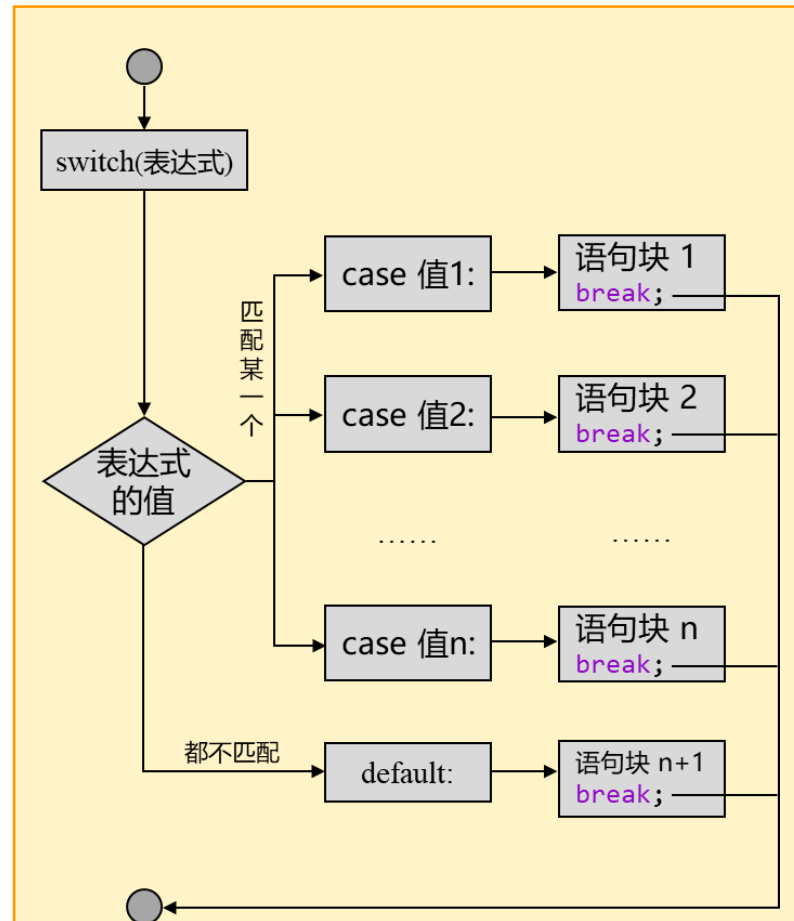
```
{  
    case 常量表达式1: 语句块1 [break;]  
    .....  
}
```

- switch 括号中的表达式其返回值须为**整型值**
- case 中为**常量表达式**，是一个整型值
- case 跳转**只发生一次**
 - switch 表达式的值与找到第一个匹配的 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，不再做后续的case 情况匹配判断
- 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



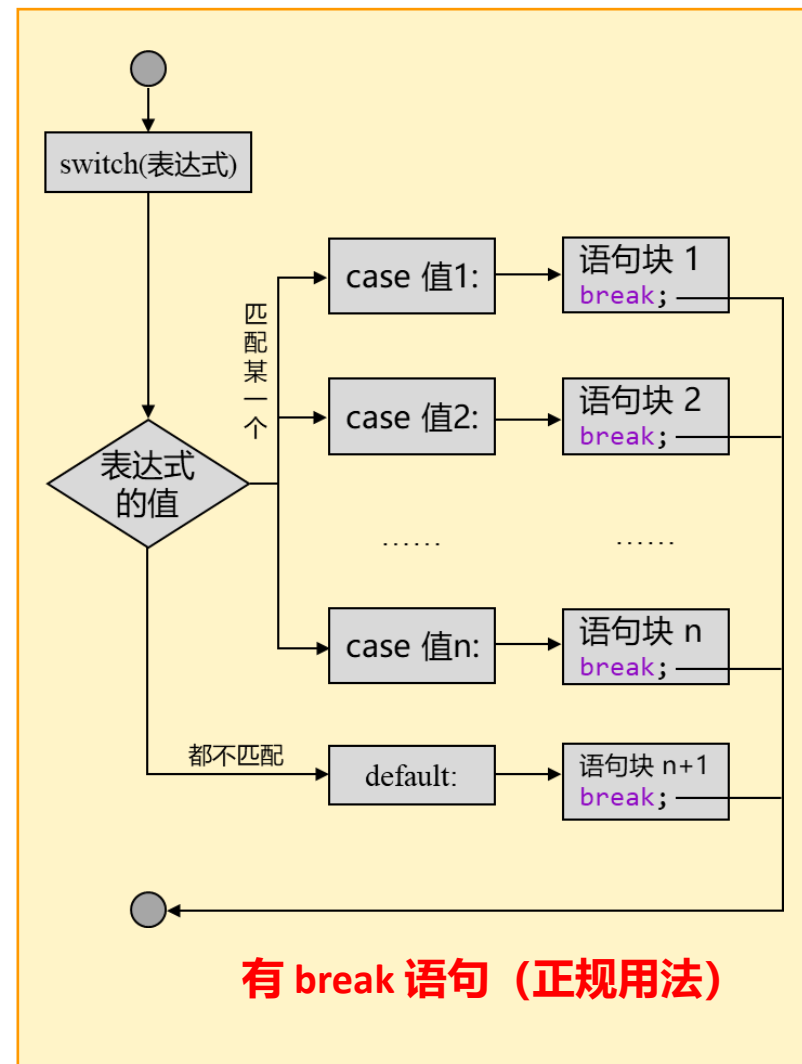
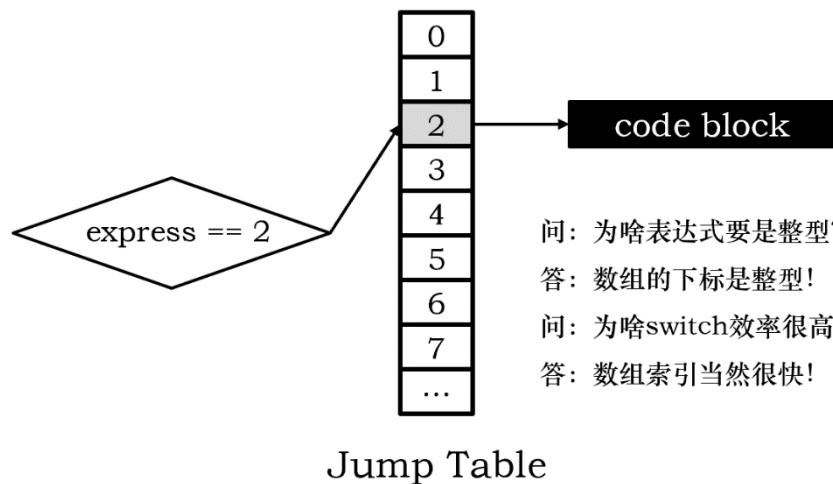
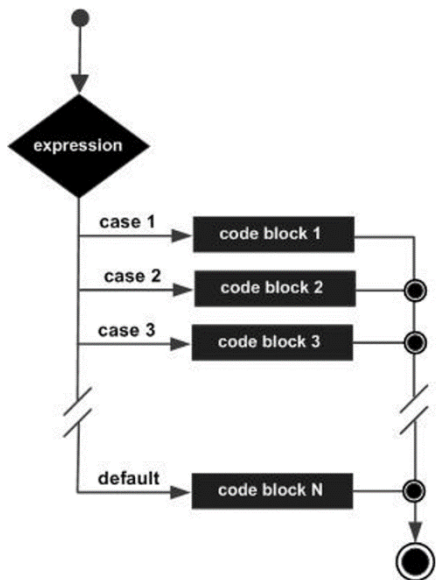
无 break 语句

什么后果？稍后介绍。



有 break 语句（正规用法）

*switch语句的执行逻辑

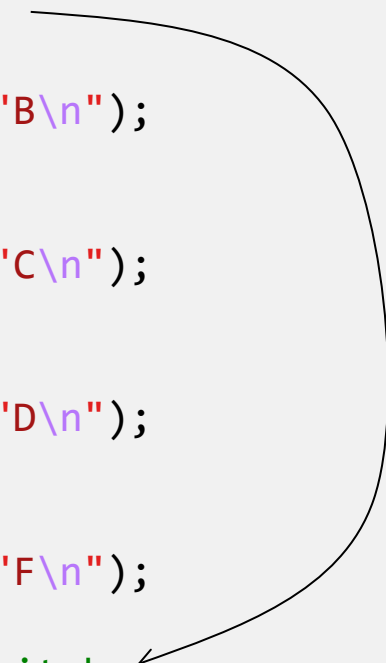


switch语句实例：成绩转换

例4-9：输入一个成绩（百分制整数），转换为等级制（A~F）

- 若连续的 case 之间没有语句，则它们执行同样的语句。
- **break** 语句使程序跳出 switch。
- case 中有多个语句时，不必放在{ }中。

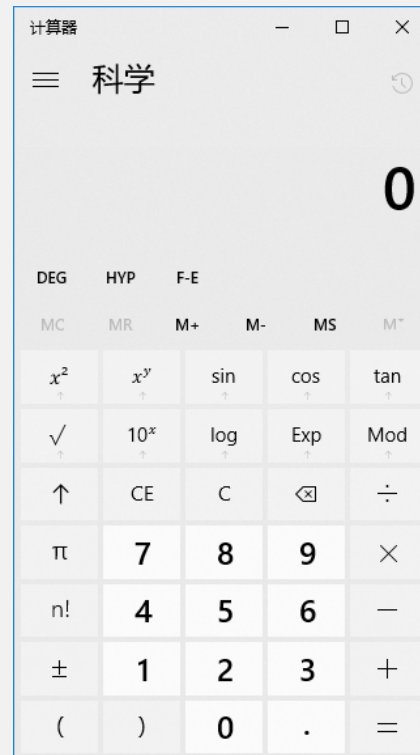
```
scanf("%d", &score);
switch ( score/10 )
{
case 10:    // e.g. 100/10 is 10
case 9:     // e.g. 96/10 is 9
    printf("A\n");
    break;
case 8:
    printf("B\n");
    break;
case 7:
    printf("C\n");
    break;
case 6:
    printf("D\n");
    break;
default:
    printf("F\n");
    break;
} // end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



switch语句实例

讲例4-8的四则运算 (用switch实现)

```
char op;
double x, y, r, 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 (y != 0.0)
        {
            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);
```



设计一个简单的计
算器(四则运算功能)!

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）
-



循环语句

循环语句是什么？怎么设计更优美？

例： $\pi = 4 * (1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots + \frac{(-1)^n}{(2*n+1)}) = ?$

↓

$$\text{sum} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots + \frac{(-1)^n}{(2*n+1)} = \frac{(-1)^0}{2*0+1} + \frac{(-1)^1}{2*1+1} + \dots + \frac{(-1)^i}{2*i+1} + \dots + \frac{(-1)^n}{(2*n+1)}$$

⇒ $\pi = 4 * \text{sum}$

$i = 0$ 时, $\text{sum} = 1$

$i = 1$ 时, $\text{sum} = \text{sum} + \frac{(-1)^1}{2*1+1}$

$i = 2$ 时, $\text{sum} = \text{sum} + \frac{(-1)^2}{2*2+1}$

$i = 3$ 时, $\text{sum} = \text{sum} + \frac{(-1)^3}{2*3+1}$

...

$i = n$ 时, $\text{sum} = \text{sum} + \frac{(-1)^n}{2*n+1}$

$\pi = 4 * \text{sum}$

scanf("%d", &n);
sum = 1;
sum = sum - 1/(2*1+1.0);
sum = sum + 1/(2*2+1.0);
sum = sum - 1/(2*3+1.0);
sum = sum + 1/(2*4+1.0);
...
// n 为1000时, 写1000行?
// 输入的 n 未知, 程序不能这样写!
// 即使 n 已知, 程序也不应这样写,
// 这样的代码太长, 每行逻辑相同, 循环!

$\pi = 4 * \text{sum};$

scanf("%d", &n);
// i = 0 时, 初始化, 第1项及其符号
sum = 1;
sign = 1;

// 这两步要反复（多次, 循环）执行
sign = -sign; // 下一项的符号跟前一项相反
sum = sum + sign/(2*i+1.0); // 前i+1项和

$\pi = 4 * \text{sum};$

循环语句

循环语句是什么？怎么设计更优美？

例： $\pi = 4 * (1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots + \frac{(-1)^n}{(2*n+1)}) = ?$

↓

$$\text{sum} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots + \frac{(-1)^n}{(2*n+1)} = \frac{(-1)^0}{2*0+1} + \frac{(-1)^1}{2*1+1} + \dots + \frac{(-1)^i}{2*i+1} + \dots + \frac{(-1)^n}{(2*n+1)}$$

⇒ $\pi = 4 * \text{sum}$

$i = 0$ 时, $\text{sum} = 1$

$i = 1$ 时, $\text{sum} = \text{sum} + \frac{(-1)^1}{2*1+1}$

$i = 2$ 时, $\text{sum} = \text{sum} + \frac{(-1)^2}{2*2+1}$

$i = 3$ 时, $\text{sum} = \text{sum} + \frac{(-1)^3}{2*3+1}$

...

$i = n$ 时, $\text{sum} = \text{sum} + \frac{(-1)^n}{2*n+1}$

$\pi = 4 * \text{sum}$

scanf("%d", &n);
sum = 1;
sum = sum - 1/(2*1+1.0);
sum = sum + 1/(2*2+1.0);
sum = sum - 1/(2*3+1.0);
sum = sum + 1/(2*4+1.0);
...
// n 为1000时, 写1000行?
// 输入的 n 未知, 程序不能这样写!
// 即使 n 已知, 程序也不应这样写,
// 这样的代码太长, 每行逻辑相同, 循环!

 $\pi = 4 * \text{sum};$

scanf("%d", &n);
// i = 0 时, 初始化, 第1项及其符号
sum = 1;
sign = 1;

for(i=1; i<=n; i++){
 // sign = sign * (-1)
 sign = -sign; // 下一项的符号跟前一项相反
 sum = sum + sign/(2*i+1.0); // 前i+1项和
}

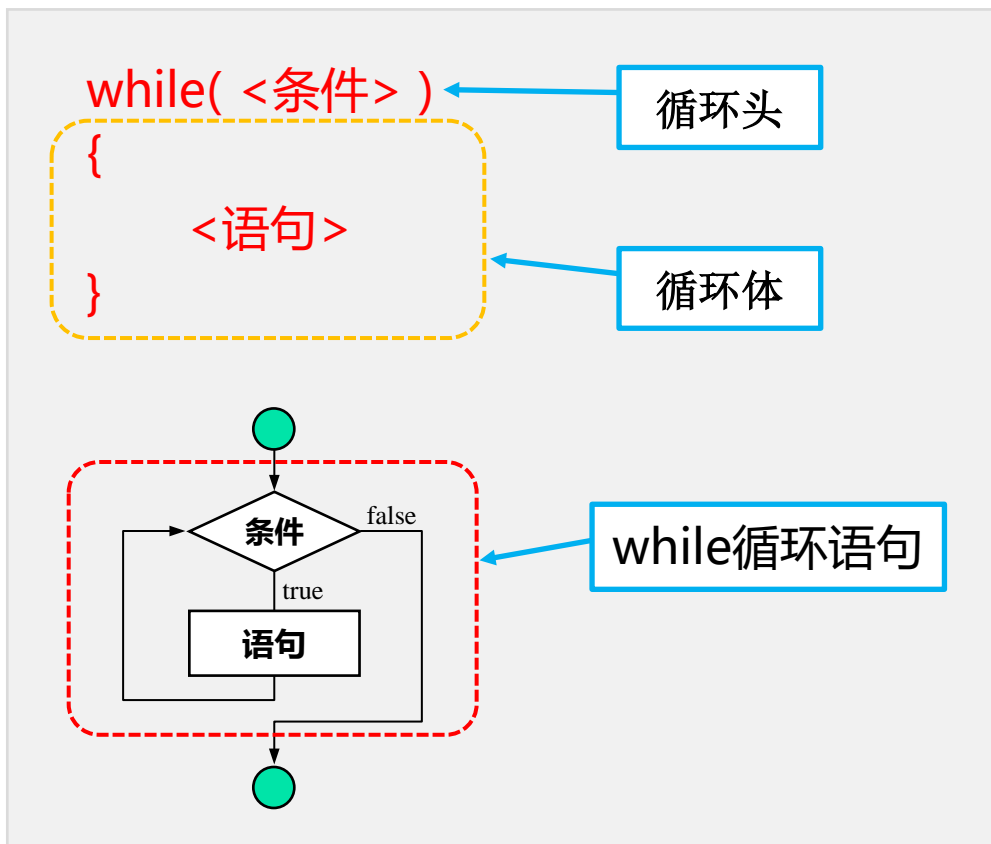
 $\pi = 4 * \text{sum};$

有些问题, 循环的逻辑
从内部开始写更好理解。

循环内部是核心,
循环条件是关键。

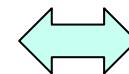
while语句

- 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）：



常见编程错误：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，将导致无限循环（死循环），如：

```
while(a = 3) // 漏了一个等号
{
    .....
}
```



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

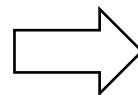
while应用：输入连续整数（输入错误或完成时结束）

例4-11：不间断的输入一个整数，正确输入时继续计算，输入完成时或输入格式有误结束程序。

分析：输入函数 **scanf** 的返回值表示正确输入的数据的个数，根据scanf返回值检测输入格式有误。

- ◆ 正确读入一个整数时，scanf返回值为 1
- ◆ 输入有误（未能正确读入数据）返回 0
- ◆ 读到文件末尾（输入结束）返回 -1

```
int a;
printf("\ninput an integer to go on,\n");
printf("or other char to quit:\n\n");
while(scanf("%d", &a) > 0)
// while(scanf("%d", &a) != EOF)
{
    printf("valid input: %d\n\n", a);
}
printf("invalid input! quit!\n");
```



输入输出样例

input an integer to go on,
or other char to quit:

1
valid input: 1

123
valid input: 123

abc
invalid input! quit!

Q：把第4行的“>0”换成“!= EOF”，执行右边的输入，输出什么？

* while应用：输入正确结束

例4-12：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（该行第一个输入数字不是有效成绩）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 a;  
char ch;  
printf("\ninput a score 0~100:\n");  
printf("(input other char to go on)\n");  
  
while(scanf("%d", &a) == 0 || a < 0 || a > 100)  
{  
    printf("invalid input!\n\n");  
    while(scanf("%c", &ch) == 1 && ch != '\n');  
    //while(getchar() != '\n');  
}  
  
printf("\nGood job! Valid input %d!\n", a);
```

while循环
头最后如果
加了分号

一个特别
用法

```
while(scanf("%c", &ch) == 1 && ch != '\n')  
{  
    ; // do nothing  
}
```

输入输出样例

```
input a score 0~100:  
(input other char to go on)  
123  
invalid input!  
  
-12  
invalid input!  
  
-12 xy  
invalid input!  
  
99  
  
Good job! Valid input 99!
```

* while应用：输入正确结束

例4-12：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（该行第一个输入数字不是有效成绩）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 a;
char ch;
printf("\ninput a score 0~100:\n");
printf("(input other char to go on)\n");

while(scanf("%d", &a) == 0 || a < 0 || a > 100)
{
    printf("invalid input!\n\n");

    while(scanf("%c", &ch) == 1 && ch != '\n');
}

printf("\nGood job! Valid input %d!\n", a);
```

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，则外层循环的scanf()先读入-12到变量a，然后内层循环的scanf()读入且清除-12后面的空格、xy和换行，再开始下一行输入。

如果没有该行？

* while语句的实例

例4-12: 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 (该行第一个输入数字不是有效成绩) 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 a;  char ch;
printf("\ninput a score 0~100:\n");
printf("(input other char to go on)");
// 无效输入时, 条件就为真, 循环, 再输入
while (scanf("%d", &a) == 0 || a < 0 || a > 100)
{
    printf("invalid input!\n\n");

    // while(scanf("%c", &ch) == 1 && ch != '\n');
}
printf("\nGood job! Valid input %d!\n", a);
```

如果注释这一行...

死循环

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!

设计循环时, 要注意循环条件的改变, 避免死循环!

控制台的输入示例 (缓冲区)

-12 xy[enter]



- (1) 读入整数 -12 到int变量a, 读入正确, scanf返回1, 但数字不符合要求(-12 < 0 正确, 继续循环);
- (2) 下一次循环的读入, 从 -12 后的非空格字符开始解析 (scanf %d, 跳过输入中的空格);
- (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), 即, 缓冲区未清空, 一直在缓冲区这个位置读不动;
- (4) 重复第(2)和第(3)步...

while 应用

例4-13-1：输入一段文本，统计行数。

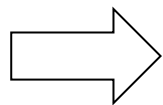
输入



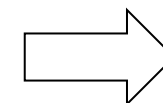
c4-13-1.in - 记事本

文件 编辑 查看

123
abcd
12345
|



```
char ch;  
int n = 0;  
while(scanf("%c", &ch) != EOF)  
    if(ch == '\n')  
        n++;  
  
printf("%d lines", n);
```



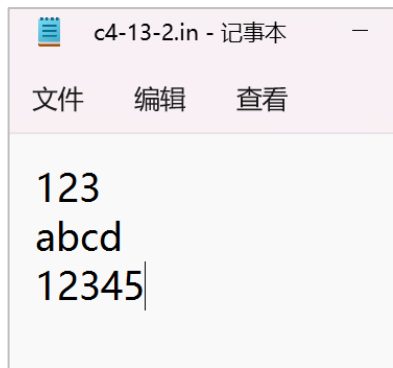
输出

3 lines

while 应用

例4-13-1：输入一段文本，统计行数。

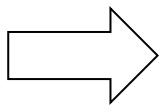
输入



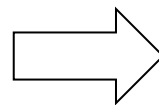
c4-13-2.in - 记事本

文件 编辑 查看

123
abcd
12345



```
char ch;  
int n = 0;  
while(scanf("%c", &ch) != EOF)  
    if(ch == '\n')  
        n++;  
  
printf("%d lines", n);
```



输出

?

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

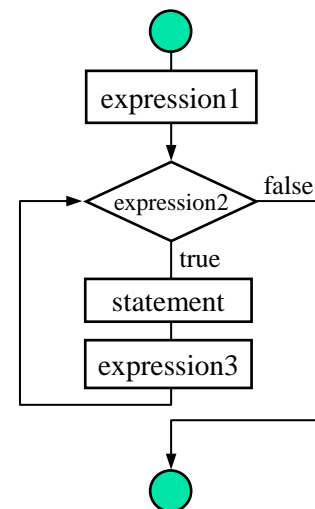
表达式1, 2, 3均可以为空（省略），
但分号不能省略

循环体（循环体只有一条语句时可以用不用大括号）

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

for结构与等价的
while结构

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



for应用：阶乘求余

例4-14：输入正整数 n ($n < 100$)，求 n 的阶乘 r ，如果 r 大于1000003，则用 r 对1000003求余数。

```
int i, n, r, P;
P = 1000003;
scanf("%d", &n);

for (r = i = 1; i <= n; i++)
{
    r *= i; // r = r * i
    if (r > P)
        r %= P; // r = r % P
}
```

Q：不要把 n 的阶乘 $r = n!$ 先计算出来，再计算 $r \% P$ ，为什么？

while循环

```
int i, n, r, P;
P = 1000003;
scanf("%d", &n);

r = i = 1;
while (i <= n)
{
    r *= i;
    if (r > P)
        r %= P;
    i++;
}
```

```
for (r = i = 1; i <= n; i++)
{
    r *= i;
}
r %= P;
```



A： n 的阶乘 $n!$ 可能很大，超过 `int` 范围，导致数据范围溢出。对中间结果进行 `%` 运算，能保证最终结果不变，数据还不会溢出。利用这个性质：

$(a*b)\%P = ((a\%P)*(b\%P))\%P$

证明思路： $(n!)\%P = ((n\%P)*((n-1)!\%P))\%P$

for应用：自然数的倒数和

例4-15：输入正整数 n ($n < 10^8$)，求 $\sum_{i=1}^n 1/i$ 。

```
int i, n;  
double ans;  
scanf("%d", &n);  
  
for(ans = 0.0, i = 1; i <= n; i++)  
{  
    ans += 1.0/i;  
}
```

逗号运算符，

- **逗号运算符：将多个表达式连接在一起，作为一个表达式**

- ◆ 常见的应用场景：for循环头中，分号之间只能写一个表达式，如果存在多个独立操作，需要将这多个表达式合并成一个表达式
- ◆ 将两个初始化操作合并为一个表达式

```
for (ans = 0.0, i = 1; i <= n; i++)
```

- **逗号运算符构成一个单独的逗号表达式**

- ◆ 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);
```

等价于

```
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

for应用：最大公约数（辗转相除法）

例4-16：最大公约数(Greatest Common Divisor, GCD)，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： $\text{GCD} = \text{gcd}(a, b)$

◆**定理**： $\text{gcd}(a, b) = \text{gcd}(b, a \% b)$

算法：

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

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

赋值的顺序别搞错了

证明（略）

(1) 先证明 $\text{gcd}(a, b)$ 是 b 和 $a \% b$ 的公约数

设 $g = \text{gcd}(a, b)$, $g1 = \text{gcd}(b, a \% b)$

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 整除（商为 $x - m * y$ ），

又因 g 是 b 的约数，所以， g 是 b 和 $a \% b$ 的公约数，即 g 能整除 $g1$ 。

(2) 再证明 $g1$ 也能整除 g 。同理。

例： $\text{gcd}(40, 24)$

→ $\text{gcd}(24, 16)$

→ $\text{gcd}(16, 8)$

→ 8

for应用：最大公约数（辗转相除法）

例4-16：输入两个非负整数 a 和 b（int范围），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. gcd(a, b): if b is 0, gcd(a, b) is a, stop.
2. gcd(a, b): 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) 】

```
int a, b, gcd;
scanf("%d%d", &a, &b); // 设输入b不为0
for (gcd = a%b; gcd != 0; gcd = a%b)
{
    a = b;
    b = gcd;
}
gcd = b;
printf("GCD is: %d\n", gcd);
```

思考题：如果输入b为0时，gcd(a, b)是什么？代码如何完善？

for应用：最大公约数（辗转相除法）

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

例4-16：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。

```
#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);
        return 0;
    }
    for(; (r = a%b) != 0; )
    {
        a = b;
        b = r;
    }
    gcd = b;
    prn(gcd);
    return 0;
}
```

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

可看出，此时用 while 语句使得结构更清晰些？
下一章用函数会更清晰！



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

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应用：最小公倍数(Least Common Multiple, LCM)

例4-17：输入两个正整数 a 和 b，计算它们的最小公倍数 LCM（设a、b、LCM都不超过 int 范围）。

算法：a 能整除 x，且 b 能整除 x，则 x 是 a 和 b 的公倍数；
满足上述性质的最小 x 为 a 和 b 的 LCM。

// 朴素的方法（穷举算法）

```
int a, b, LCM, x;
scanf("%d%d", &a, &b);
x = (a > b) ? a : b; // x = max(a,b)
while( !((x % a) == 0 && (x % b) == 0) )
{
    x++;
}
LCM = x;
printf("%d", LCM);
```

a 能整除 x，且 b 能整除 x

// 高效的算法

```
int a, b, GCD, LCM;
scanf("%d%d", &a, &b);
LCM = a * b; // 这行不能少

while ((GCD = a % b) != 0)
{
    a = b;
    b = GCD;
}
GCD = b;
LCM = LCM/GCD;
//不能写成(a*b)/GCD，因a和b已被修改（在计算GCD的过程中）
printf("%d", LCM);
```

for应用： 养老保险金 【课后阅读】

例4-18： **养老保险金** 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 = 0.006, 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);
```

for应用：复利计算【课后阅读】

例4-19：复利计算 150 年前，Bob 因为躲避战争，逃难到瑞士，在瑞士银行存了10000美元（假设那时候的利率较高，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：与上一个例子相同，同样是利率计算问题，不过此处本金是一次性的，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 表示本金，r 表示年利率，则第 n 年的资产总额：

$$\begin{aligned} S_n &= S_{n-1} + S_{n-1} * r \\ &= S_{n-1} * (1+r) \\ &= S_1 * (1+r)^n \\ &= \text{base} * (1+r)^n \end{aligned}$$

```
int base = 10000, n, i;
double rate = 0.075, sum = base;
for (i = 1; i <= 150; i++)
{
    sum = sum * (1 + rate);
}
printf("\n%12.2f $\\n", sum);
```

for 应用实例

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

例4-20: 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)

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

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

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:

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

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?

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 (通项) 相同, 如何让代码更优美?

```
int i, j, k, sign = -1;
double n, d, s1 = 0, s2 = 0, eps;
eps = 1e-11; // pow(10, -11)

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

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

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

do while语句【略】

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);
prn(a);
```

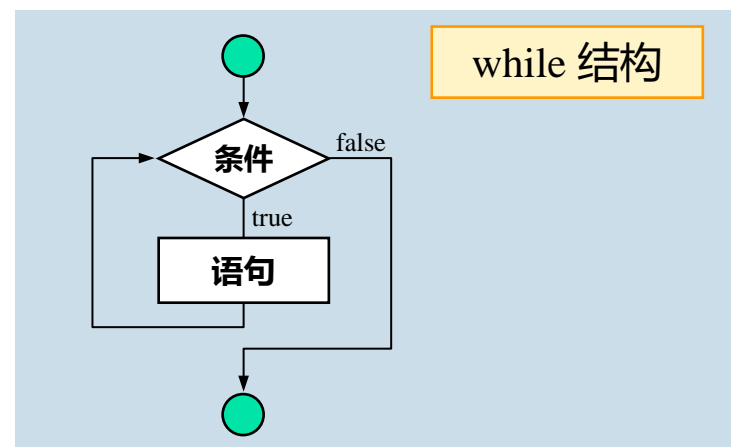
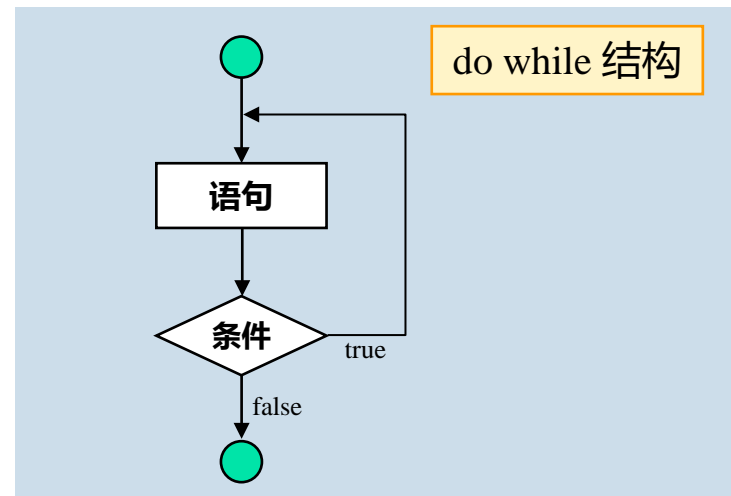


```
while((r = a%b) != 0)
{
    a = b, b = r;
}
prn(b);
```



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

**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。**



选择合适的循环结构

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

【例】求 2~100 间的所有偶数平方和

```
for (s = 0, i = 2; 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。

循环结构的嵌套

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4-21: **大 V 手势** 期末考试了, 给自己来一个大大的 “V” 字吧。输入正整数 n ($n < 30$), 打印出一个占 n 个字符高度的 “V” 形图案。如:

输入 n 等于 1 时, 输出为:

V

输入 n 等于 3 时, 输出为:

找规律：n=4时，第3行， $2n-3=5$ 个字符，不含最后的/

```
int n, i, j;
scanf("%d", &n);
for(i=1; i<=n; i++) // 共n行，从上往下，分别是第1到第n行
{
    for(j=1; j<=2*n-i; j++)// 第i行，有2n-j个符号，不含最后一个符号 /
    {
        if(i == j)           // 第i行第i列，为符号 \
            printf("\\");    // 注意转义符输出
        else
            printf(" ");      // 第i行其他列，为空格
    }
    printf("/\n");            // 第i行最后一个符号为 / ，然后换行
}
```

2020春

期末A题

A

1514/1556

循环结构的嵌套【书上的写法，自行阅读】

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4-21：大V手势 期末考试了，给自己来一个大大的“V”字吧。输入正整数 n ($n < 30$)，打印出一个占 n 个字符高度的“V”形图案。如：

输入 n 等于 1 时，输出为：

V

输入 n 等于 3 时，输出为：

V

换一种顺序分析，
最下面是第1行：
第 i 行，有5个步骤

2020春

期末A题

A

1514/1556

```
int n, i, j;
scanf("%d", &n);
for (i = n; i >= 1; i--) // 共n行，从下往上，分别是第 1 到第 n 行
{
    // 第 i 行的5个步骤
    for (j = 1; j <= n-i; j++) // 1. 输出 n-i 个空格
        printf(" ");
    printf("\\"); // 2. 输出反斜杠，注意转义符用法
    for (j = 1; j <= 2*i-2; j++) // 3. 输出 2i-2 个空格，注意数数
        printf(" ");
    printf("/"); // 4. 输出斜杠
    printf("\n"); // 5. 换行
}
```

循环结构的嵌套：整数分解

例4-22：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 (≤ 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

最早可能的分解，
从1开始。
最后可能的分解，
从 $s/2$ 开始。

分析：

S1: for $i = 1 \dots s/2$?

S2: for $j = i+1 \dots$

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

S3: if $\text{sum} == s$

print $s = \sum_i j$

go to S1, and loop $i+1 \dots s/2$

$s = 99$
 $= s/2 + (s/2 + 1)$
 $= 49 + 50$

```
int s, i, j, k, st, sum, got = 0;
scanf("%d", &s);
for(i = 1; i <= s/2; i++)
{
    //s=s/2+(s/2+1), s最后可能的分解，因此循环最多到 s/2
    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+1; k <= j; k++)
                printf(" + %d", k);
            putchar('\n');
            got++;
        }
    }
}
printf("cases: %d\n", got);
```

2024Spr, E1- **H** 拆解立方数

时间限制：1000ms 内存限制：65536kb

通过率：237/325 (72.92%) 正确率：237/1080 (21.94%)

循环结构的嵌套：阶乘之和（简单，自行阅读）

例4-23：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)

```
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);
```

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;
}
```

思考：若输入 n 为 1000，中间计算结果对 100007 取模，程序程序如何写？

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);
```

遇到死循环怎么办

死循环？如何结束？如：

```
sum = 0;  
i = 0;  
while(1)  
{  
    sum += i;  
    i++;  
}  
...
```

```
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
```

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);
}

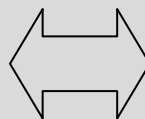
..... // other codes
```

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++;
}
```

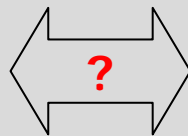


```
for (int x = 1; x <= 10; x++)
{
    printf("%d ", x);
}
```

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: ?
```

输出 ?



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

输出 ?

循环的“非常规”控制实例（break 和 continue 应用）

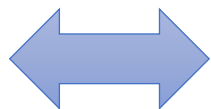
【例】自然数累加 自然数 1~n 进行累加

常规做法

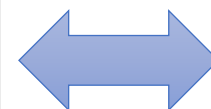
```
int n, i, sum;  
scanf("%d", &n);  
i = 1;  
sum = 0;
```

多种不同方法，
实现相同功能。

```
for(i=1; i<=n; i++)  
{  
    sum += i;  
}
```



```
while(1)  
{  
    if(i > n)  
        break;  
    sum += i;  
    i++; // 这行一定不可少  
}
```



```
for( ; ; i++)  
{  
    if(i > n)  
        break;  
    sum += i;  
}
```


break语句实例

例4-24：连续正整数 ...输出最长的序列... (详细题意参见【例4-22】)

2024Spr, E1- **H** 拆解立方数

时间限制: 1000ms 内存限制: 65536kb

通过率: 237/325 (72.92%) 正确率: 237/1080 (21.94%)

分析：在【例4-22】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？

输入15，三种分解：

15 = 1+2+3+4+5

15 = 4+5+6

15 = 7+8

```
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+1; k <= j; k++)
                printf(" + %d", k);
            putchar('\n');
        }
    }
    if (sum == s)
        break;
}
if (sum != s)
    printf("NONE\n");
```

跳出哪个循环？

思考：如果不用 break，
程序该如何写？

【例4-24】连续正整数 ...输出最长的序列... (详细题意参见【例4-22】)

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+1; 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; 0 == flag && 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+1; k <= j; k++)
                printf(" + %d", k);
            putchar('\n');
        }
    }
    if (sum == s)
        flag = 1; // found
}

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

```
int s, i, j, k, st, sum = 0;
scanf("%d", &s);
for (i = 1; sum != s && 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+1; k <= j; k++)
                printf(" + %d", k);
            putchar('\n');
        }
    }
}

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

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），或循环逻辑更为复杂。
- 使用 break，有时会使得程序很简洁、增强程序的可读性。

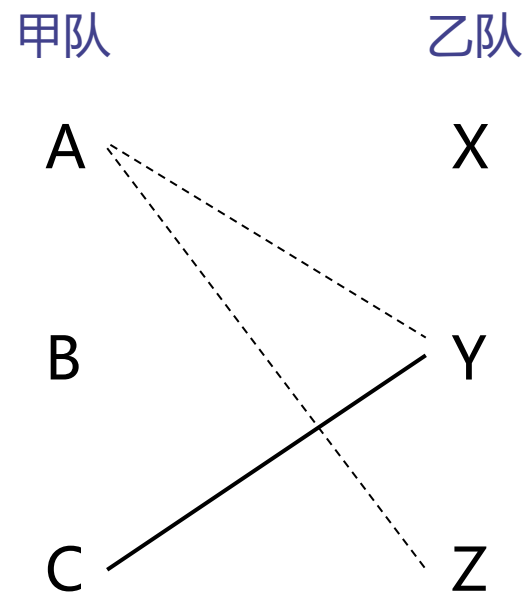
continue应用：朴素的AI—安排比赛

例4-25：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。

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



continue应用：朴素的AI—安排比赛

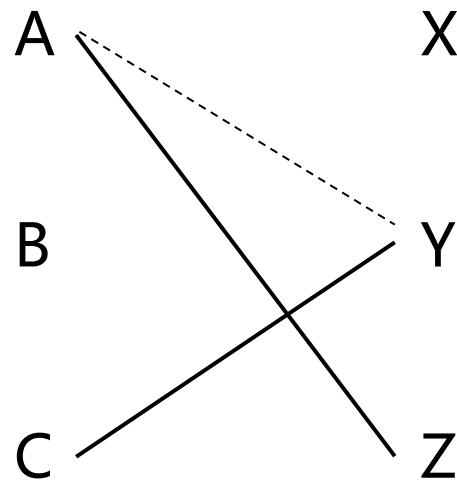
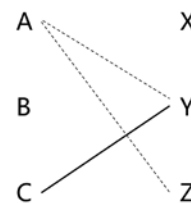
例4-25：乒乓球赛 甲队的 A、B、C 与乙队的 X、Y、Z 比赛，
已知 A 不与 X 对阵，C 不与 X 和 Z 对阵，输出对阵名单。

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



continue应用：朴素的AI—安排比赛

例4-25：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。

根据题意，人的推理很容易，很简单：

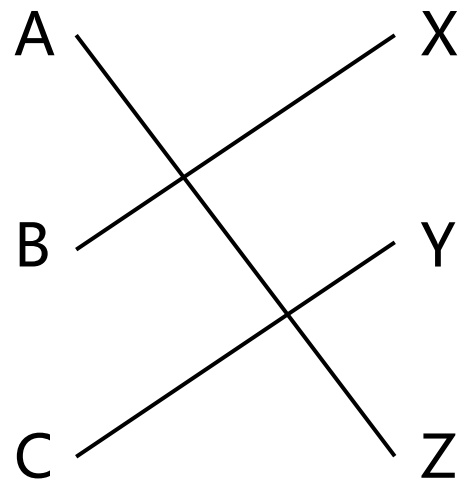
(1) A 不对阵 X，则 A 可能对阵 Y 或 Z

(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应用：朴素的AI—安排比赛

例4-25：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。

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);
        }
```

比赛冲突规则解释：

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)

对阵选手不重复

排除不符要求的组合

人的推理、人的智能可能更“逻辑”些。

计算机怎么样推理的呢？计算机的智能是按某种规则“枚举”（枚举过程中进行筛选），这就是**计算机的AI**。计算机的计算速度很快，这种枚举很多时候比人“强大”。

该例程即是一种计算机推理过程。

continue应用：朴素的AI—安排比赛

例4-25：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);
        }
```


continue小结

- 不用 `continue`，也能跳过循环后面的语句继续执行，但可能需要设计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

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

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.

*例4-26：查询某天是星期几

2024年10月				
一	二	三	四	五
30 廿八	1 廿九	2 三十	3 九月	4 初二
7 初五	8 寒露	9 初七	10 初八	11 重阳节
14 十二	15 十三	16 十四	17 十五	18 十六
21 十九	22 二十	23 霜降	24 廿二	25 廿三

2005年10月						
一	二	三	四	五	六	日
26 廿三	27 廿四	28 廿五	29 廿六	30 廿七	1 廿八	2 廿九
3 九月	4 初二	5 初三	6 初四	7 初五	8 寒露	9 初七
10 初八	11 重阳节	12 初十	13 十一	14 十二	15 十三	16 十四
17 十五	18 十六	19 十七	20 十八	21 十九	22 二十	23 霜降

```
scanf("%d", &longday); // 20240321
```

```
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");  
    ...  
}
```

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**，比如，2024年10月17日应输入为 **20241017**。

本代码虽然长，但逻辑简单，很综合，用了 **while**, **break**, **continue**, **switch**, **if** 等三类控制结构的所有知识。

```
// 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;
```

```
// 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;
```

```
    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;
} // end of switch
} // end of while
```

4.4 再论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4-16-goto版：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：

step1: if b is 0, gcd(a, b) is a, stop.

step2: if a%b is 0, gcd(a, b) is b, stop.

step3: 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);

    // step1
    if(b == 0)
    {
        prn(a < 0 ? -a : a);
        return 0;
    }

    step2:
    if(a%b == 0)
    {
        prn(b < 0 ? -b : 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4-16-goto版：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：

- step1: if b is 0, gcd(a, b) is a, goto step4.
 - step2: if a%b is 0, gcd(a, b) is b , goto step4.
 - step3: let $r \leftarrow a \% b$, $a \leftarrow b$, $b \leftarrow r$, go to step 2.
 - step4: stop
- 【 gcd(a, b) = gcd(b, a%b) 】

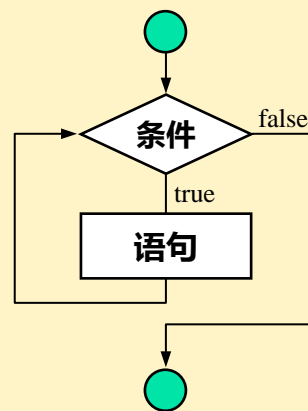
为什么不建议用 goto 语句？用 goto 语句编程，有时跟写伪代码一样易写，易读！但就是太灵活了，可能容易失控！

另一个
goto版

```
scanf("%d%d", &a, &b);  
  
// step1  
if(b == 0)  
{  
    prn(a < 0? -a : a);  
    goto step4;  
}  
  
step2:  
if(a%b == 0)  
{  
    prn(b < 0 ? -b : b);  
    goto step4;  
}  
  
// step3  
r = a%b;  
a = b;  
b = r;  
goto step2;  
  
step4:  
return 0;
```

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

```
#include <stdio.h>  
int main()  
{  
    int n=0;  
    char ch;  
    printf("input a text:\n");  
  
loop: ← 语句标号  
    if(~scanf("%c", &ch))  
    {  
        n++;  
        goto loop;  
    }  
    printf("\nchar\'# of this text: %d\n", n);  
    return 0;  
}
```

输入
输出
示例

```
命令提示符  
D:\alac\example\chap4>c4-28_goto < c4-28_goto.c  
input a text:  
  
char'# of this text: 229
```

** 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！
- 慎用 goto，因为这容易使程序陷入逻辑混乱，破坏结构化设计风格，可能带来错误或隐患，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。

```
...  
goto statement;  
int a, b; // 变量a和b被goto跳过，但后面可能用  
int sum; // 同上  
double x; // 同上  
...  
statement:  
...
```

作业：请写几个具体的例子，来说明用 goto 的“好”与“不好”？

本讲小结

- 结构化编程的基本三种结构：顺序，选择，循环
- `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【结合 `goto`】）。
- 用好 `switch`, `break`, `continue` 能更快捷实现复杂逻辑。
- 循环结束的条件判断与改变是循环的关键。
- 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。
- 熟练掌握控制结构（程序执行流程），理解程序的**算法思想**，培养**逻辑思维**。

补充：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）**

```
// c: century-1, y: year, m: month, w: week, d: day
int c, y, m, w, d, longday = 1;

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;

    // 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;
        ...
    }
}
```

顺序结构

选择结构

if
if/else
switch

重复结构

for
while
do/while

工欲善其事必先利其器

Visual Studio Code (vscode) 的代码自动补全功能，真的很强大！

```
if (/* condition */)
{
    /* code */
}
else
[ ]
```

```
switch (expression)
{
case /* constant-expression */:
    /* code */
    break;

default:
    break;
}
```

```
while (/* condition */)
{
    /* code */
}
```

```
for (size_t i = 0; i < count; i++)
{
    /* code */
}
```

```
do
{
    /* code */
} while (/* condition */);
```

补充阅读：continue应用，朴素的AI（2）*

例：某场球赛的联赛即将开始，北方区有4个队N1~N4，南方区有4个队S1~S4，第一轮比赛时北方区某个队和南方区某个队进行比赛，实行淘汰赛，获胜的4个队伍再进行比赛。

在第一轮比赛安排中，已知：N1不能与S1和S2比赛，N2不能与S1和S2比赛，N3不能S1比赛，N4不能与S3和S4比赛。请问，第一轮比赛有几种比赛对阵安排？请依序输出每一种对阵情况。

<div>S \ N</div>	1	2	3	4
1	×	×	×	
2	×	×		
3				×
4				×

补充阅读：continue应用，朴素的AI（2）*

例：某场球赛的联赛即将开始，北方区有4个队N1~N4，南方区有4个队S1~S4，第一轮比赛时北方区某个队和南方区某个队进行比赛，实行淘汰赛，获胜的4个队伍再进行比赛。

在第一轮比赛安排中，已知：N1不能与S1和S2比赛，N2不能与S1和S2比赛，N3不能S1比赛，N4不能与S3和S4比赛。请问，第一轮比赛有几种比赛对阵安排？请依序输出每一种对阵情况。

问题分析：人的推理过程非常简单，根据已知条件，可列出不能对阵的比赛情况，如图所示。容易推出：

- S1只能与N4对阵（然后S1那一行就去掉，S1不能与其他N#对阵，N4那一列，同理）。
- 然后S2就只能与N3对阵（S2那一行，N3那一列，同理）。
- S3可以与N1或N2对阵，相应地，S4就可以与N2或N1对阵。
- 人具有智能，这种推理反映了人的逻辑推理能力。计算机如何推理呢？计算机的强大在于计算速度快，通过**遍历所有情况**，不符合的情况跳过，符合的情况就输出结果，这可以看成是计算机的推理过程，或计算机的人工智能(Artificial Intelligence, AI)

<div>S N</div>	1	2	3	4
1	<div>×</div>	<div>×</div>	<div>×</div>	<div>✓</div>
2	<div>×</div>	<div>×</div>	<div>✓</div>	<div></div>
3			<div></div>	<div>×</div>
4			<div></div>	<div>×</div>

与【例4-25】相同

补充阅读：continue应用，朴素的AI (2) *

```
int N[5]= {0}, count=0;
for(N[1]=1; N[1]<=4; N[1]++)
    for(N[2]=1; N[2]<=4; N[2]++)
        for(N[3]=1; N[3]<=4; N[3]++)
            for(N[4]=1; N[4]<=4; N[4]++)
            {
                if( (N[1]==N[2]) + (N[1]==N[3]) + (N[1]==N[4])
                    + (N[2]==N[3]) + (N[2]==N[4]) + (N[3]==N[4]) >= 1 )
                    continue; // 一个区的某队不能与另一区的两个队同时比赛
                if (N[1] >= 3 && N[2] >= 3 && N[3] != 1 && N[4] <= 2) // 对阵规则
                {
                    printf("%d: ", ++count);
                    printf("N1 vs S%d, N2 vs S%d, N3 vs S%d, N4 vs S%d\n",
                        N[1],N[2],N[3],N[4]);
                }
            }
}
```

S\N	1	2	3	4
1	×	×	×	
2	×	×		
3				×
4				×

```
1: N1 vs S3, N2 vs S4, N3 vs S2, N4 vs S1
2: N1 vs S4, N2 vs S3, N3 vs S2, N4 vs S1
```