NLP 南京大学 自然语言处理研究组 Natural Language Processing Group

南京大学 NANJING UNIVERSITY

程序中的操作描述——程序流程控制

黄书剑

- 程序流程及其描述
- 顺序执行
- 选择执行
- 循环执行
- 无条件转移
- 程序设计风格与结构化程序设计

程序（program）

- **程序是一组连续的相互关联的计算机指令：**

  CPU能执行的指令包括：

  算术运算指令：实现加、减、乘、除等；

  比较指令：比较两个操作数的大小；

  数据传输指令：实现CPU的寄存器、内存以及外设之间的数据传输；

  流程控制指令：用于确定下一条指令的内存地址

  顺序、转移、循环、子程序调用/返回

- **程序是指示计算机处理某项计算任务的任务书，计算机根据该任务书，执行一系列操作，并产生有效的结果。**
  - **计算（compute）：并非单指数值计算，而是指根据已知数据（data）经过一定的步骤（算法）获得结果的过程**

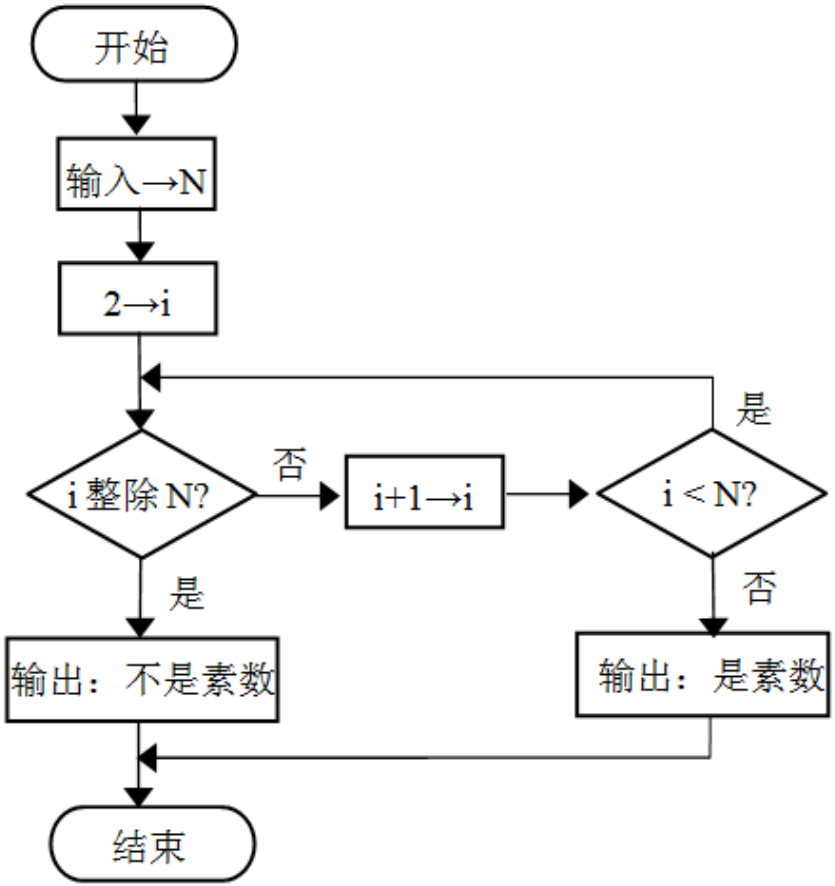
程序 = 算法 + 数据(结构)

程序运行的步骤也需要用语言来描述

算法描述和程序流程图

- 如何求解问题？　判断N（>2）是否为素数（质数）
- 算法的描述
  - 自然语言描述
  - 程序流程图
  - 伪代码
  - 代码等
- 在设计大型、复杂程序的流程控制时，往往先用程序流程图来对程序的流程进行描述，然后再用某种编程语言来写出程序。
- 本课程中所涉及的问题都不是很复杂，因此在进行程序流程设计时不一定采用流程图，但仍然需要有清晰的处理思路。

判断N（>2）是否为素数（质数）的程序流程图



猜数字游戏

- 获得一个100以内的随机整数
- 用户给出猜测
- 程序判断猜测：大、小、相等
- 用户再次猜测
- ……

猜数字游戏2

- 获得一个四位整数
- 用户给出猜测
- 程序判断猜测：
  - 有几个数字值正确且位置正确
    - ……
  - 有几个数字值正确但位置不对
    - ……
- ……

- 如：1234 猜 2341 （0A4B） 1789 （1A0B）

回顾：语句（statement）

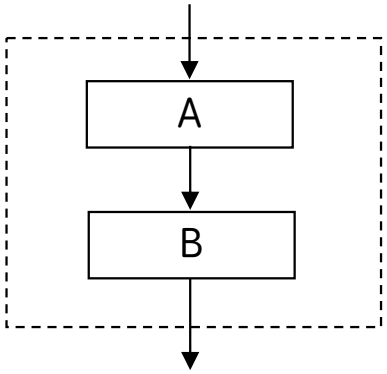
- 表达式末尾加一个分号可以构成语句
  - 如：`d = d + 1;`
  - 最简单的语句是一个分号，即空语句，不执行任何操作
  - 一行可以包含多个语句，语句跨行需要使用续行符
- 可以用一对花括号将多个语句括起来，形成复合语句，一个复合语句可以看作一个语句块

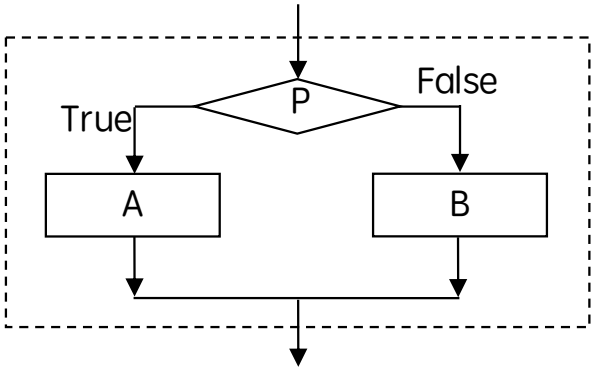
```
{
 sum = sum + PI * d;
 d = d + 1;
}
```

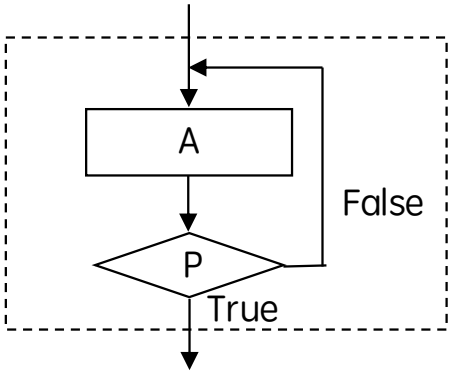
- 一些关键字，如while、return等，与表达式、分号或（子）语句配合也可以构成语句（后续详细介绍）

语句

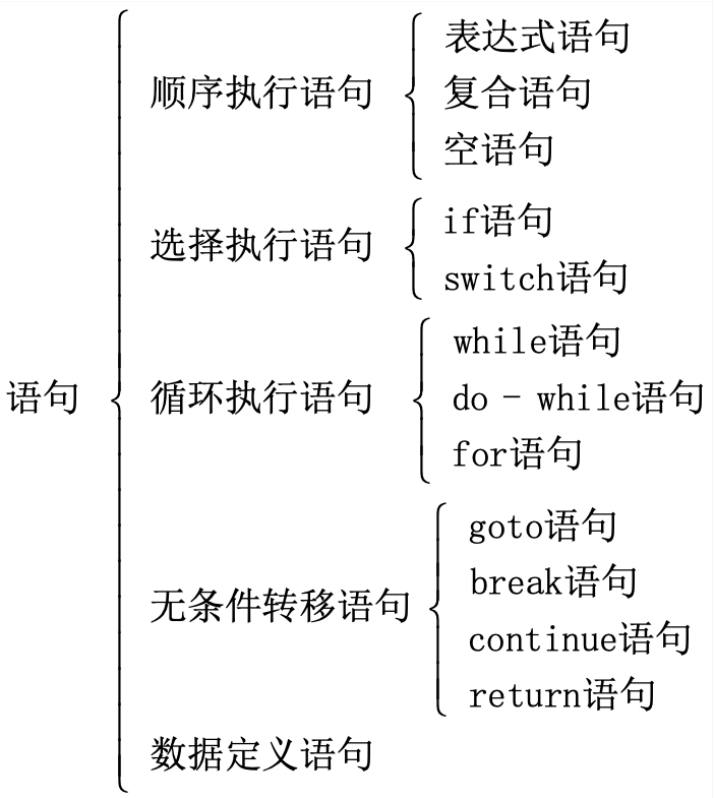
- 在程序中，流程控制是用语句来实现的，它指定了表达式的计算次序。
- 流程控制语句包括：
  - 顺序执行语句：按书写次序依次执行。
  - 选择执行语句：根据条件选择执行。
  - 循环执行语句：重复执行直到某个条件不满足。
  - 无条件转移语句：无条件转移到程序某个位置。







C/C++语句的分类

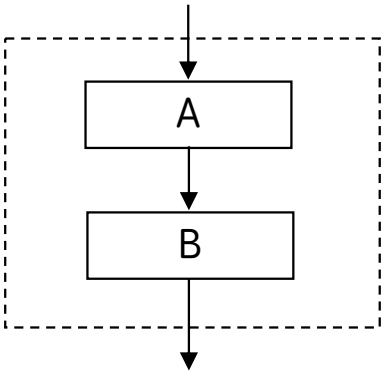

语句
顺序执行语句
表达式语句
复合语句
空语句
选择执行语句
if语句
switch语句
循环执行语句
while语句
do - while语句
for语句
无条件转移语句
goto语句
break语句
continue语句
return语句
数据定义语句

南京大学
NANJING UNIVERSITY

顺序执行

顺序执行

- **一般来说，语句按书写次序顺序执行。**
  - 从左到右、从上到下
- **执行次序可被某些语句改变！**
  - 表达式语句
  - 复合语句
  - 空语句



表达式语句

- 在C/C++表达式的后面加上一个分号";"就可以构成表达式语句，其格式为：

  <表达式>;

例如：

- `a + b * c;`
- `a > b ? a: b;`
- `a++;`
- `x = a + b;`

- 一个表达式语句执行完后将执行紧接在后面的下一个语句。

较常使用的表达式语句

- **常用的表达式语句：**
  - 赋值
  - 自增/自减
  - 输入/输出，等
  - 无返回值的函数调用

例如：

- `x = a+b;  //赋值`
- `x++;  //自增`
- `cin >> a;  //输入`
- `cout << b;  //输出`
- `f(a);  //函数调用`

- **语句执行必须留下一些"痕迹"！**
  - 改变程序状态
  - `x+y; //`可能没有意义！

复合语句

- 复合语句是由一对花括号括起来的一个或多个语句，又称为块（block）。其格式为：

```
{
    <语句序列>
}
```

  - <语句序列>中的语句可以是任何的C/C++语句，其中包括数据定义和声明语句。
- 一般情况下，复合语句执行完将按照书写次序执行后续的语句，除非在复合语句中包含改变执行流程的语句。
- 语法上，复合语句是一个语句（结构语句），它一般作为其它结构语句（如选择语句和循环语句）的子句（成分语句）或作为函数体。

复合语句举例

```
{
   int a, b;
   scanf("%d%d", &a, &b);
   int max;
   max = (a >= b) ? a : b;
   printf("%d", max);
}
```

空语句

- 根据程序设计的需要，在程序中的某些地方有时需要加上一些空操作，以方便其它流程控制的实现。
- 空语句的格式为：

  ;
- 空语句的作用是用于语法上需要一条语句的地方，而该地方又不需做任何事情。
- 空语句常常作为结构语句的子句。

南京大学
NANJING UNIVERSITY

选择执行

选择执行

- 在程序中，常常需要根据不同的情况来从一组语句中选择一个来执行（分支），这是通过选择语句来完成的。
- C/C++的选择语句包括：
  - if语句
  - switch语句
- 语法上，选择语句属于结构语句。

if 语句

- if语句（又称条件语句）是根据一个条件满足与否来决定是否执行某个语句或从两个语句中选择一个语句执行。
- if语句有两种格式：

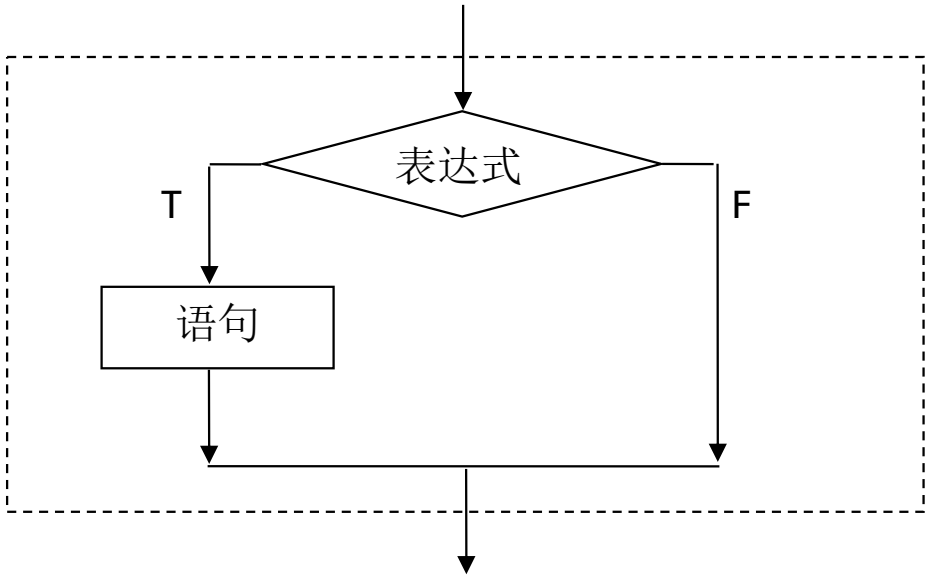
```
if (<表达式>)
    <语句>
```

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

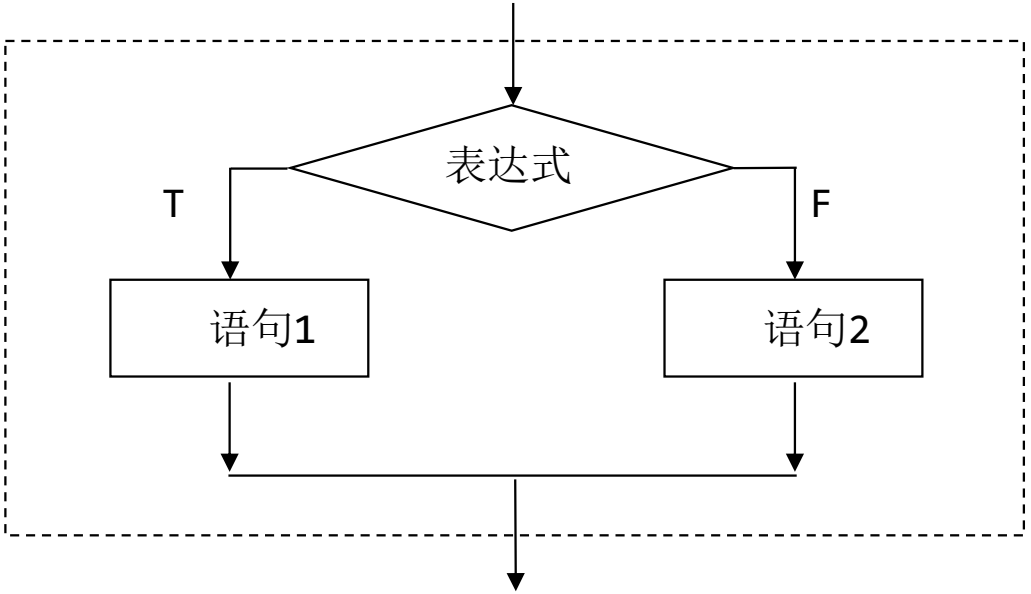
  - 其中，<表达式>可以是任意的C/C++表达式，通常为关系或逻辑表达式，表示条件；
  - <语句>、<语句1>、<语句2>可以是任意的C/C++语句，但必须是一个语句！（结构语句算一个语句）

if语句的含义

```
if (<表达式>)
    <语句>
```




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



if语句的书写

- 保持缩进模式，保持前后一致，不仅可以使程序美观，还有助于查看子句，提高程序的可读性。

```
if(x >= 0)
     y = x * x;
else
     printf("Input error! \n");
```

- 缩进并不决定代码逻辑！
- 如果分支任务含多条语句，则一定要用一对花括号将它们组合成复合语句。

```
if(x >= 0)
{
     y = x * x;
     printf("%f * %f equal %f \n", x, x, y);
}    //复合语句是一个整体，要么都被执行，要么都不被执行
```

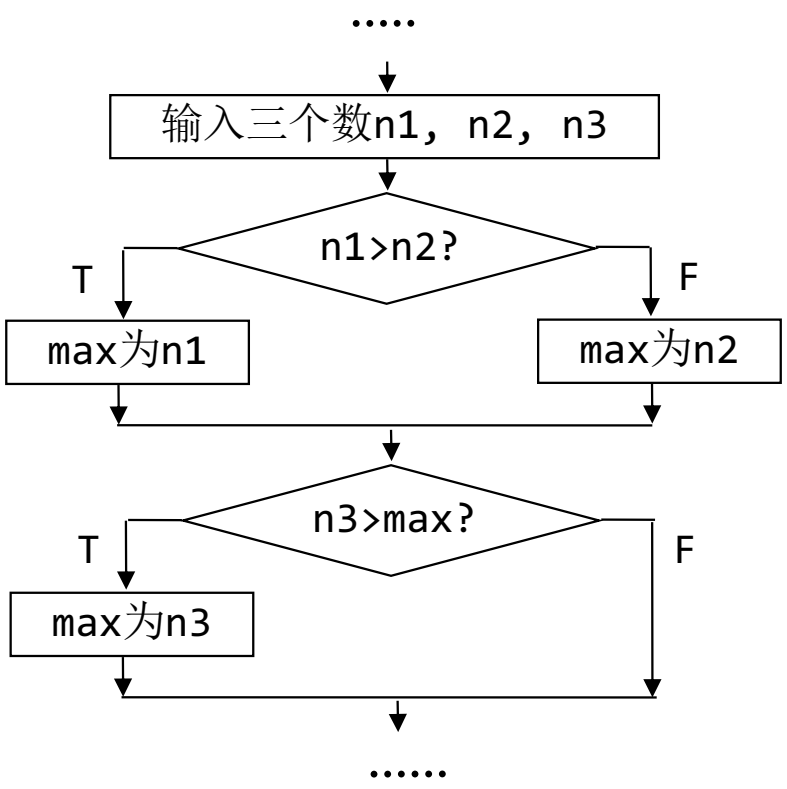
示例1：

- 从键盘输入三个互不相等的整数，计算其中的最大值并将其输出

- 流程?
  - 获取数据
  - 判断三个数的大小
    - 先判断其中两数
    - 大者再与第三数比较

```
int main()
{
   int n1, n2, n3;
   printf("Please enter 3 nums: \n");
   scanf("%d %d %d", &n1, &n2, &n3);
   if(n1 > n2)
      if(n1 > n3)
         printf("The max: %d \n" , n1);
      else
         printf("The max: %d \n" , n3);
   else
      if(n2 > n3)
         printf("The max: %d \n" , n2);
      else
         printf("The max: %d \n" , n3);
   return 0;
}
```

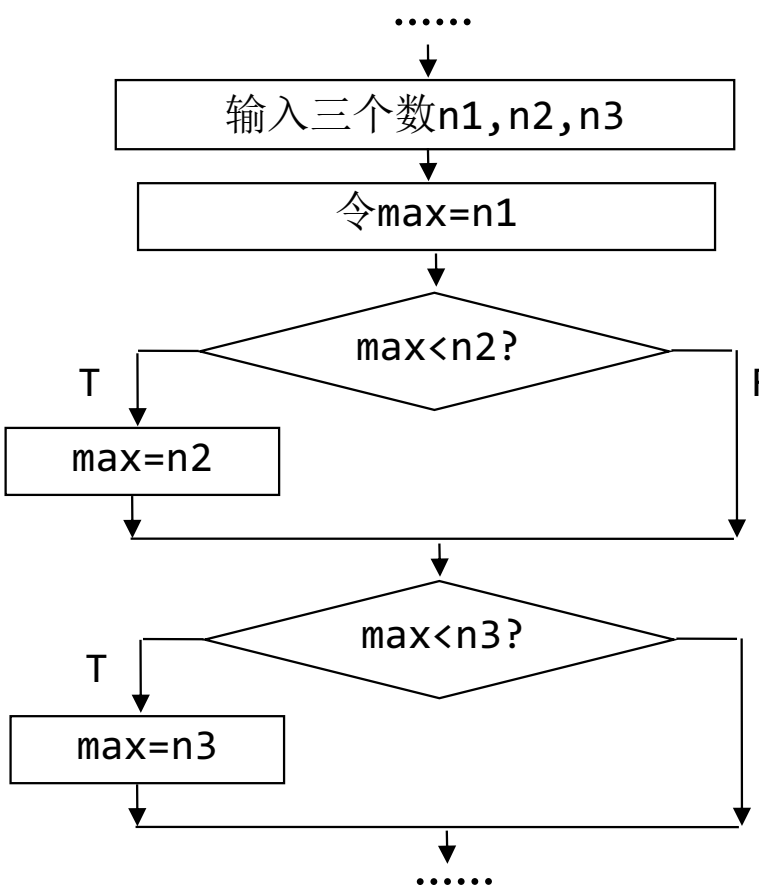
逻辑清晰，但代码略微繁琐！
使用变量max，分离输出过程

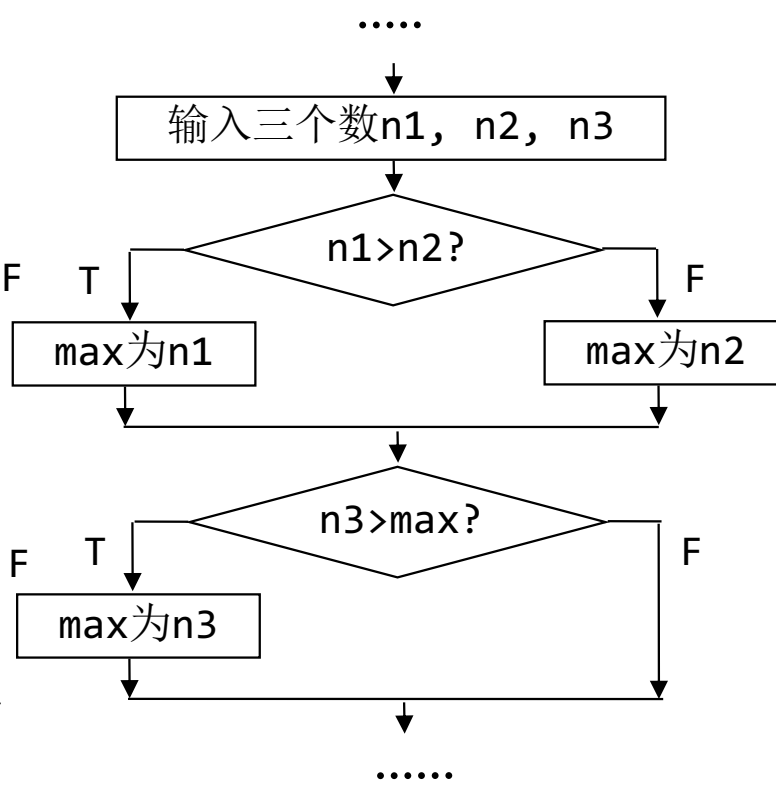



```
int main()
{
   int n1, n2, n3, max;
   printf("Please enter 3 nums: \n");
   scanf("%d%d%d", &n1, &n2, &n3);
   if(n1 > n2)
      max = n1;
   else
      max = n2;
   if(n3 > max)
      max = n3;
   printf("The max: %d \n" , max);
   return 0;
}
```

```
max = n1;
if(max < n2)
      max = n2;
if(max < n3)
      max = n3;
```

```
max = n1;
if(max < n2)
      max = n2;
if(max < n3)
      max = n3;
```





- 使用逻辑运算符组合多个条件

```
if(n1 > n2 && n1 > n3)
   max = n1;
if(n2 > n1 && n2 > n3)
   max = n2;
if(n3 > n1 && n3 > n2)
   max = n3;
```

嵌套的if语句

- 多个条件依次进行判断时，可能涉及条件的重复判断
- 此时，可以用嵌套的if语句来表示
  - 即if语句的成分语句也可以包含if语句
  - 此时，应使用更多缩进，表明语句间的层次关系

```
if (A)
    ...
if (!A)
    ...
...
```

单个分支v.s.两个分支
(相当于if-else)

```
if (A)
    ...
if (!A && B)
    ...
...
```

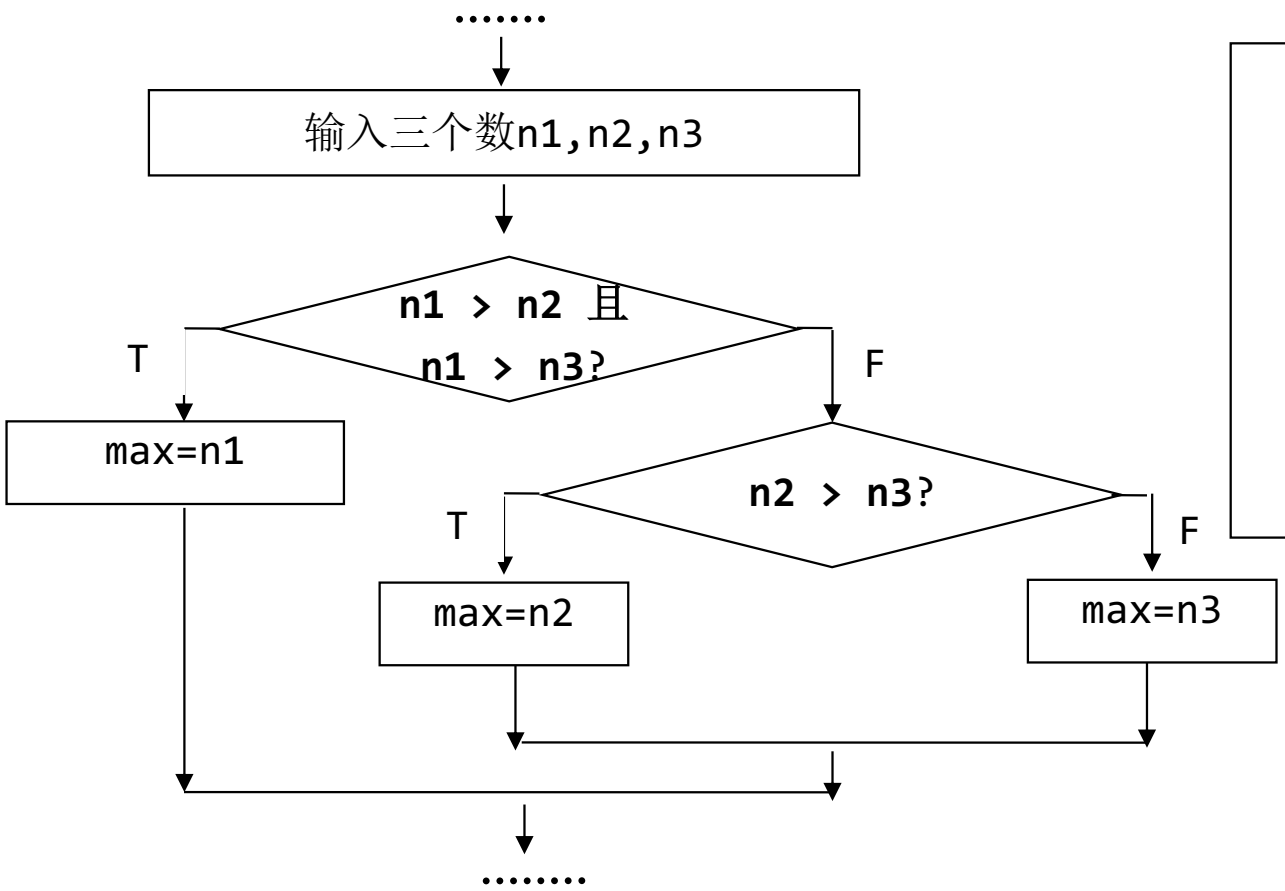
条件A可能会被判断多次

```
if (A)
    ...
else
   if (B)
       ...
   else
```

```
if(n1 > n2 && n1 > n3)
   max = n1;
if(n2 > n1 && n2 > n3)
   max = n2;
if(n3 > n1 && n3 > n2)
   max = n3;
```

```
if(n1 > n2 && n1 > n3)
    max = n1;
else
    if(n2 > n1 && n2 > n3)
       max = n2;
    else
       max = n3;
```

注意，此时一个分支复合语句整体作为else后的语句




```
if(n1 > n2 && n1 > n3)
    max = n1;
else
    if(n2 > n3)
        max = n2;
    else
        max = n3;
```

```
int main()
{
   int n1, n2, n3, max;
   printf("Please enter 3 nums: \n");
   scanf("%d%d%d", &n1, &n2, &n3);
   if(n1 > n2)
      max = n1;
   else
      max = n2;
   if(n3 > max)
      max = n3;
   printf("The max: %d \n" , max);
   return 0;
}
```

```
max = n1;
if(max < n2)
      max = n2;
if(max < n3)
      max = n3;
```

```
if(n1 > n2 && n1 > n3)
    max = n1;
else
    if(n2 > n3)
       max = n2;
    else
       max = n3;
```

if 语句的歧义问题

- 程序中，当两种不同形式的if语句嵌套时，理解时会产生分歧。

```
max = n1;
if(n1 > n2)
    if(n3 > n1)
        max = n3;
else    //这里else是对应 n3 > n1 不成立的情况
        //      不是对应 n1 > n2 不成立的情况
    ......
```

```
max = n1;
if(n1 > n2)
    if(n3 > n1)
        max = n3;
    else    //这里else是对应 n3 > n1 不成立的情况
            //    不是对应 n1 > n2 不成立的情况

        ......
```

缩进并**不改变**程序的逻辑。

C/C++语言规定，**else子句与上面最近的、没有与else子句配对的if子句配对**，而不是和较远那个if子句配对。

if 语句的歧义问题

- **程序中，当两种不同形式的if语句嵌套时，理解时会产生分歧。**

```
max = n1;
if(n1 > n2)
{
    if(n3 > n1)
        max = n3;
}
else    //这里else是对应 n3 > n1 不成立的情况
        //不是对应 n1 > n2 不成立的情况
    ......
```

```
max = n1;
if(n1 > n2)
    if(n3 > n1)
        max = n3;
    else
        ;
else    //这里else是对应 n3 > n1 不成立的情况
        //不是对应 n1 > n2 不成立的情况
    ......
```

如果须表达另一种语义，则需：

使用复合语句（用括号标识）

或

使用完整的if else结构!

多个if语句的锯齿格式

- if语句的成分语句也可以是if语句
  - 此时，应使用更多缩进，形成锯齿形式（左图）
- 为了减少文本的缩进量，大部分语言都支持else if组合形式
  - 此时，缩进可以得到有效的控制（右图）：

```
if (...)
    ...
else
   if (...)
      ...
   else
      if (...)
         ...
      else
         if (...)
            ...
         else
            ...
```

```
if (...)
    ...
else if (...)
    ...
else if (...)
    ...
else if (...)
    ...
else
    ...
```

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

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

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

示例2：

- 用求根公式求一元二次方程$ax^2+bx+c=0$的根，并输出

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

- 简要流程
  - 输入a、b、c
  - 计算x的两个值

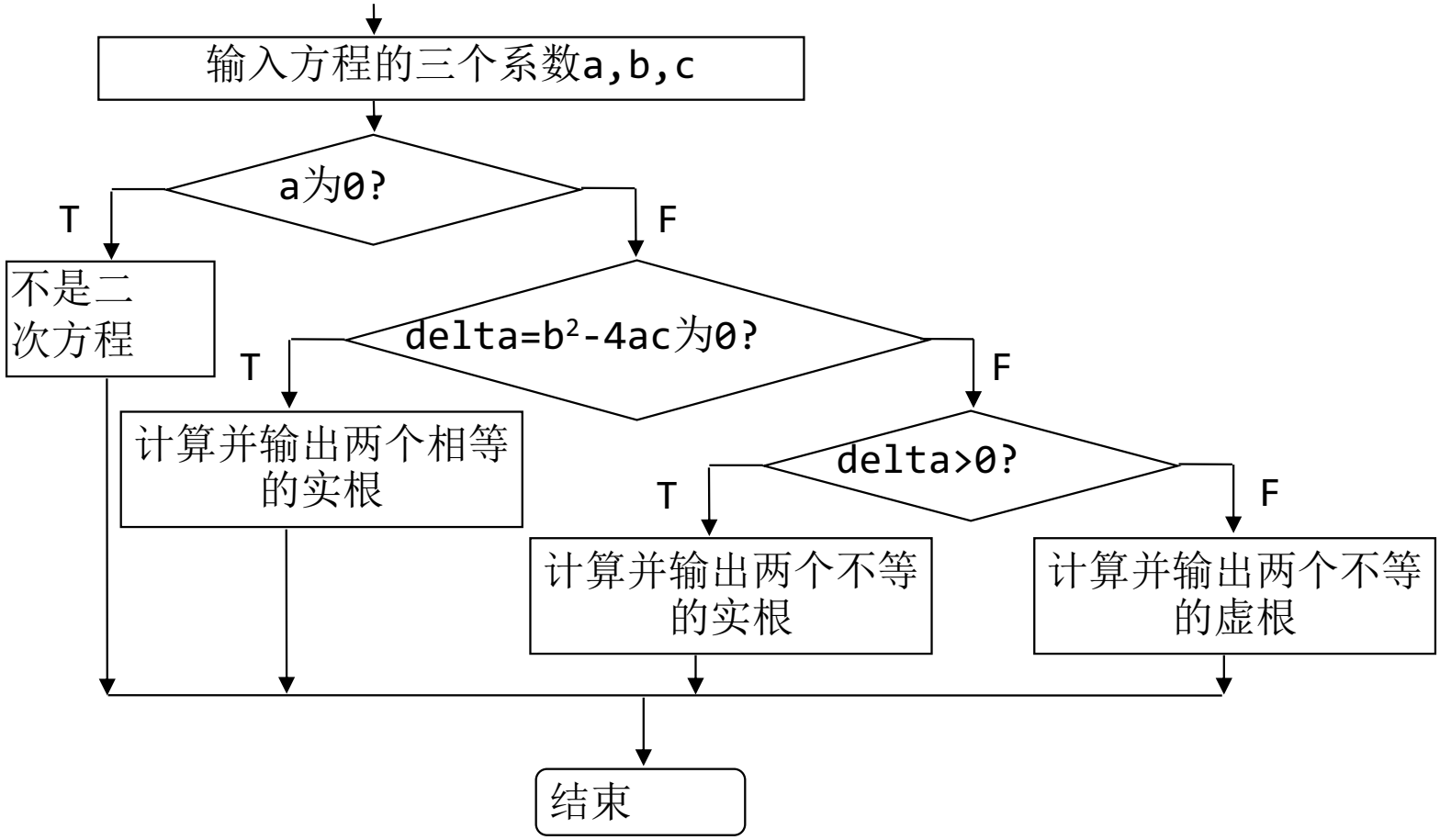
示例2：

- 用求根公式求一元二次方程$ax^2+bx+c=0$的根，并输出

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

- 简要流程
  - 输入a、b、c
  - 判断：
    - a是否为0
    - 方程是否有实数解、两根是否相同
  - 计算方程的解

- **用求根公式求一元二次方程$ax^2+bx+c=0$的根，并输出**



```
…
#include <cmath>
int main( )
{
    double a, b, c, delta, p, q;
    printf("Please input three coefficients of the equation: \n");
    scanf("%lf%lf%lf", &a, &b, &c);
    if(a == 0)                                          // 这里切勿写成if(a = 0)
        printf("It isn't a quadratic equation! \n");
    else if((delta = b*b - 4*a*c) == 0)
        printf("x1 = x2 = %f \n", -b / (2 * a));
    else if(delta > 0)
    {
        p = -b / (2*a);
        q = sqrt(delta) / (2*a);                        //sqrt函数在cmath头文件中定义
        printf("x1 = %f, x2 = %f \n", p + q, p - q);
    }
```

```
    else
    {
        p = -b / (2 * a);
        q = sqrt(-delta) / (2 * a);
        printf("x1 = %f + %fi, x2 = %f - %fi \n", p, q, p, q);
    }
    return 0;
}
```

在输出两个**复数**根时，采用先分别输出实部和虚部的方法，再添加+、-、i字符来表示复数。

```
Please input three coefficients of the equation:
1.2
2.1
3.4
x1 = -0.88+1.44i,  x2 = -0.88-1.44i
```

多路选择语句

- 程序中有时需要从两个（组）以上的语句中选择一个（组）来执行。虽然用嵌套的if语句能够实现，但不简洁。
- C/C++提供了一条多路选择语句：switch语句（又称开关语句），它能根据某个表达式的值在多组语句中选择一组语句来开始执行。

switch语句的格式和含义

```
switch (<整型表达式>)
{
        case <整型常量表达式1>: <语句序列1>
        case <整型常量表达式2>: <语句序列2>
                :
        case <整型常量表达式n>: <语句序列n>
        [default: <语句序列n+1>]   //可以省略
}
```

- 每一组语句序列的最后一个语句往往是break语句。

- switch语句的含义是：先计算<整型表达式>的值，
  - 然后执行某个与该值相等的case分支中的语句序列。
  - 如果没有与该值相等的case分支，则执行default分支的语句序列。如果没有default分支，则什么也不做。

示例1：等级转化为分数

```
char grade;
grade = getchar();
switch(grade)
{
   case 'A': printf("90-100 \n"); break;
   case 'B': printf("80-89 \n"); break;
   case 'C': printf("70-79 \n"); break;
   case 'D': printf("60-69 \n"); break;
   case 'F': printf("0-59 \n"); break;
   default: printf("error \n"); break;
}
```

示例2：日期转化为单词

数字对应每周的一天（0：星期天；1：星期一；…），根据输入数字输出其对应的英语单词

```
int week;
scanf("%d", &week);
switch(week)//该行没有分号
{
   case 0: printf("Sunday \n"); break;
   case 1: printf("Monday \n"); break;
   case 2: printf("Tuesday \n"); break;
   case 3: printf("Wednesday \n"); break;
   case 4: printf("Thursday \n"); break;
   case 5: printf("Friday \n"); break;
   case 6: printf("Saturday \n"); break;
   default: printf("error \n"); break;
}        //该行没有分号
```

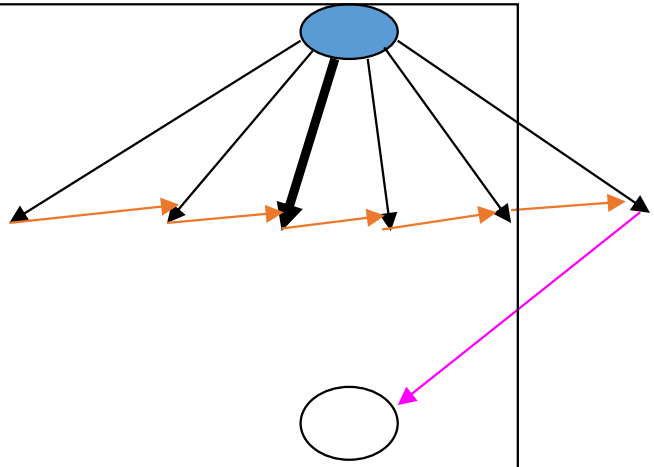
```
 if(week == 0)
   printf("Sunday \n ");
 else if(week == 1)
   printf("Monday \n ");
…
 else
   printf("error \n ");
```

switch语句中使用break语句

- 在执行switch语句的某个分支时，需要用break语句结束该分支的执行。
- 在switch语句的一个分支的执行中，如果没有break语句（最后一个分支除外），则该分支执行完后，将继续执行紧接着的下一个分支中的语句序列。

```
switch(week)
 {
    case 1: printf("Monday \n");
    case 2: printf("Tuesday \n");
    case 3: printf("Wednesday \n");
    case 4: printf("Thursday \n");
    case 5: printf("Friday \n");
    default: printf("error \n");
}
```

3

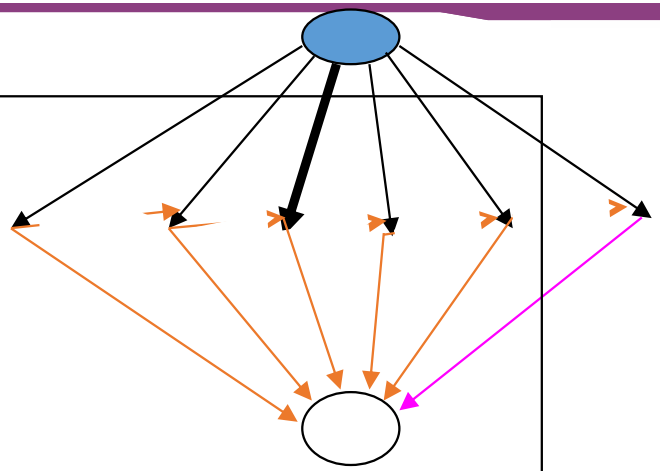


```
Wednesday
Thursday
Friday
error
```

开关

```
switch(week)
 {
   case 1: printf("Monday \n");       break;
   case 2: printf("Tuesday \n");      break;
   case 3: printf("Wednesday \n");    break;
   case 4: printf("Thursday \n");     break;
   case 5: printf("Friday \n");       break;
   default: printf("error \n");       break;
}
```

3



```
Wednesday
```

示例3：计算某年某月的天数

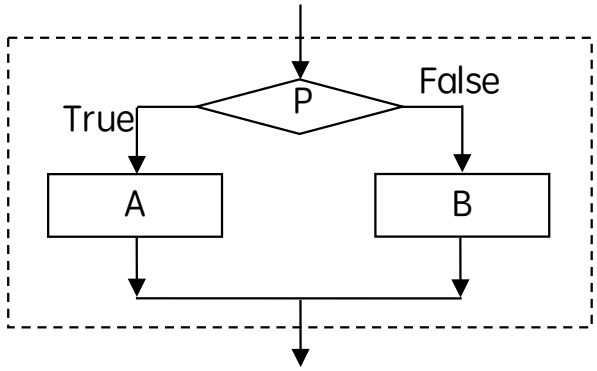
```
switch (month)
{
   case 1: case 3: case 5: case 7: case 8: case 10: case 12:
      days = 31;
      break;
   case 4: case 6: case 9: case 11:
      days = 30;
      break;
   case 2:
      if (year%400 == 0 || (year%4 == 0 && year%100 != 0))
         days = 29;
      else
         days = 28;
}
```

示例4：switch语句的嵌套

```
switch(x)
{
   case 0: printf("xy = 0 \n "); break;  // 外层分支
   case 1:
        switch(y)
        {
                case 0: printf("xy = 0 \n"); break;      // 内层分支
                case 1: printf("xy = 1 \n"); break;      // 内层分支
                default: printf("xy = %f \n", y);       // 内层分支
        } ◀
        break;                                          // 外层分支
   default: printf("error! \n ");                       // 外层分支
} ◀
```

选择执行的综合训练

- 选择执行使得程序针对不同数据的执行可以不同
  - 选择不同的分支执行



- 猜数字判断
  - 给定两个两位数，判断他们的匹配程度
    - 几个数字正确但位置不对
    - 几个数字正确且位置正确
  - 如果给定的是两个三位数？两个四位数？

南京大学
NANJING UNIVERSITY

循环执行

程序的多次执行

- 如何屏幕上输出n个星号?
  - 如果n=3?
  - 如果n=100? 如果n的个数在运行时指定?
- 如何猜数字可以猜n次?
  - 如果可以猜2次?
  - 如果可以猜5次? 猜10次?
- 如何计算用户输入的n个整数的和?
  - 如果输入2次、3次、10次?
- 程序中相同的流程可能需要重复多次!
  - 少量的重复可能可以通过复制和增加代码完成
  - 但多次重复和未知数量的重复难以处理

问题求解

- 如何判断n是否为素数?
- 如何编程计算整数n的所有因子?
  - 2? 3? 4? … n-1?
- 如何编程计算n的阶乘？
  - n!=n*(n-1)*(n-2)*...*2*1
- 如何按照迭代过程计算a的三次方根
  - x_n -> x_{n+1}

$$x_{n+1} = \frac{1}{3}(2x_n + \frac{a}{x_n^2})$$

- 程序中相近的代码可能需要重复多次!
  - 每次执行的程序可能还需要发生部分变化

问题求解的迭代法和穷举法

- **迭代法：**
  - 思路：将求解过程分解为重复或类似的一系列子过程
    - 从一个初始结果开始
    - 按照某种规则从当前结果计算到下一阶段的中间结果
    - 重复上述过程直至求解完成

$$x_{n+1} = \frac{1}{3}(2x_n + \frac{a}{x_n^2})$$

n!

- **穷举法：**
  - 思路：按照某种顺序逐一验证 "所有"可能的解
    - 取一个可能的解
    - 判断该解是否满足指定的条件
      - 如满足则该解是该问题的一个解，否则不是
    - 重复上述过程直至找到解或者所有可能都被验证

判断素数

求因子

- 计算机问题求解的迭代法和穷举法常常通过重复操作来实现：
  - 对相同的操作重复执行多次，每一次操作的数据有所不同。

- 求n的阶乘（从1开始迭代乘入更大的数）
  - f = 1，对 i = 2~n，重复执行：
    - f = f * i;（或，f *= i;）
- 求n的所有因子（从2开始尝试所有的可能因子）
  - 对 i = 2~n-1，重复执行：
    - 判断 n % i == 0

循环执行

- 上述需要重复执行的程序代码一般由循环语句来描述

- 循环一般由四个部分组成：
  - 循环初始化：为重复执行的语句提供初始数据
  - 循环条件：描述重复操作需要满足的条件
  - 循环体：描述要重复执行的操作
  - 下一次循环准备：为下一次循环更新数据（包括重复操作以及循环条件判断所需要的数据），它常常会隐式地包含在循环体中。

循环的组成示例：

- 求n的阶乘
  - 循环初始化： f = 1, i = 2
  - 循环条件：i = 2~n （ i<=n ）
  - 循环体：f = f * i
  - 下一次循环准备：i = i + 1

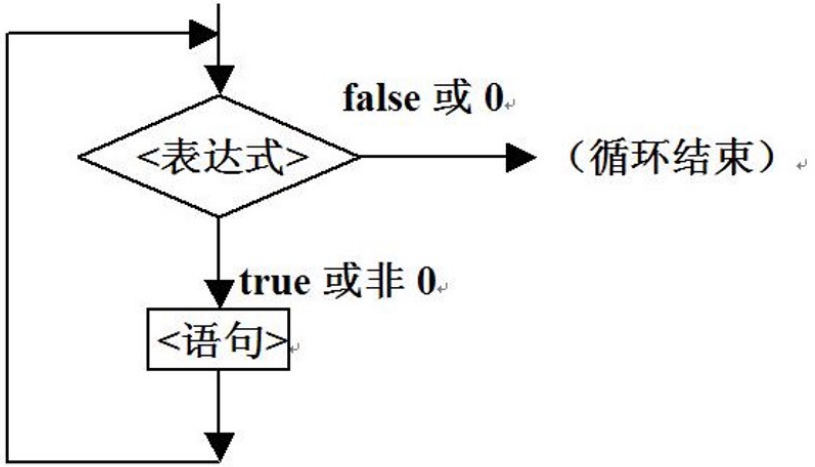
  - 循环初始化： f = 1, i = n
  - 循环条件：i = n~2 （ i>1 ）
  - 循环体：f = f * i
  - 下一次循环准备：i = i - 1

循环语句

- C/C++提供了三种实现重复操作的语句（称为循环语句）：
  - while语句
  - do-while语句
  - for语句
- 语法上，循环语句属于结构语句。

while 语句

- **格式：** `while (<表达式>)`
  `<语句>`
  - <表达式>可以为任意表达式，一般为关系或逻辑表达式
  - <语句>为循环体，可以是任意的一个C/C++语句（包括结构语句）
- **含义**



示例：用while语句求n!

```
#include <stdio.h>
int main()
{
    int n;
    scanf("%d", &n);
    int i = 1, f = 1;      //循环初始化
    while (i <= n)  //循环条件
    {
        f *= i;       //重复操作
        i++;          //下一次循环准备
    }                 //循环体(复合语句)
    printf("factorial of %d = %d \n", n, f);
    return 0;
}
```

← 注意，i=2，结果仍然正确

← 注意，while一行本身并不是语句，而是与后续的复合语句共同构成一个结构语句，所以该行后面没有分号。

示例：求1~n的自然数之和

```
int main()
{
   int n;
   scanf("%d", &n);
   int i = 1, sum = 0;
   while(i <= n)
   {
      sum = sum + i;
      i = i + 1;
   }
   printf("Sum. of integers 1-%d: %d\n", n, sum);
   return 0;
}
```

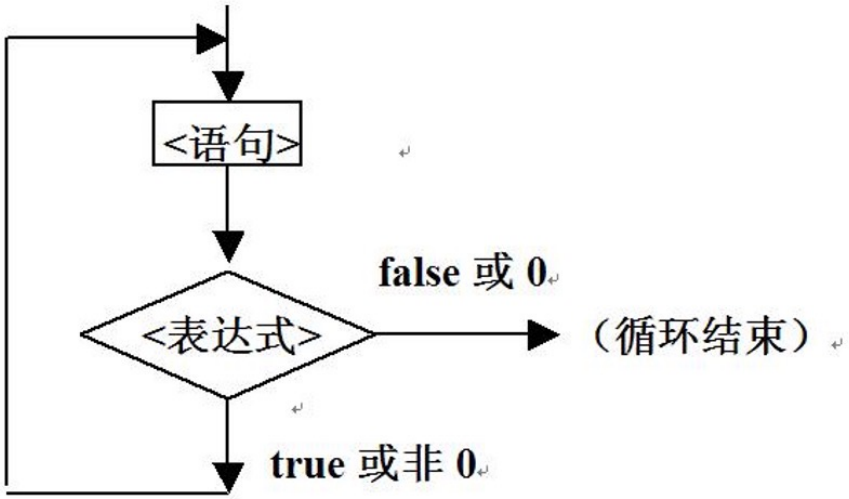
do-while 语句

- 格式： do

  <语句>

  while (<表达式>);

  - <表达式>可以为任意表达式，一般为关系或逻辑表达式
  - <语句>循环体，可以是任意的一个C/C++语句（包括结构语句）
- 含义：



示例：用do-while语句求n!

```
#include <stdio.h>
int main()
{
    int n;
    scanf("%d", &n);
    int i = 1, f = 1; //循环初始化
    do                //循环体
    {
        f *= i;       //重复操作
        i++;          //下一次循环的准备
    } while (i <= n); //循环条件
    printf("factorial of %d = %d \n", n, f);
    return 0;
}
```

```
int i = 2, f = 1;    //循环初始化
while (i <= n)  //循环条件
{
    f *= i;     //重复操作
    i++;        //下一次循环准备
}               //循环体(复合语句)
```

为什么i从1而不是2开始?

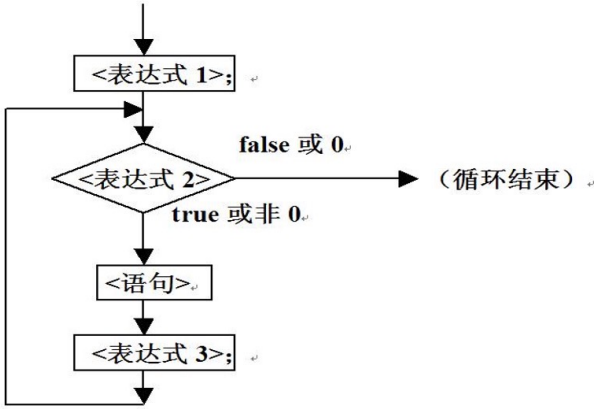
for 语句

- **格式:**

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

  - <表达式1>、<表达式2>和<表达式3>为任意表达式，通常情况下，<表达式1>为赋值表达式，<表达式2>为关系或逻辑表达式，<表达式3>为自增/自减的算术表达式。
  - <语句>为循环体，可以是任意一个C/C++语句（包括结构语句）。

- **含义:**



用for语句求n!

```
#include <stdio.h>
int main()
{
   int n,i,f;
   scanf("%d", &n);
   for (i = 2,f = 1 ;  i <= n   ;  i++)
   {
      f *= i;    //重复操作
   }              //循环体
   printf("factorial of %d = %d \n", n, f);
   return 0;
}
```

循环初始化　循环条件　下一次循环准备

- 在for语句中，<表达式1>、<表达式2>和<表达式3>均可以省略。但是，省略某个表达式并不代表循环相应部分缺失！
  - <表达式1>省略 表示for语句本身不便提供循环初始化，这时，循环初始化在for语句之前进行；
  - <表达式2>省略 表示其值为true或1，这时，一定要在循环体中判断循环条件并以某种其它方式（如：通过break语句）退出循环；
  - <表达式3>省略 表示for语句未显式给出下一次循环准备，该项工作一定是在循环体中给出的。

功能相同的两种写法

```
for (i = 2,f = 1 ;  i <= n   ;  i++)
{
   f *= i;    //重复操作
}              //循环体
```

```
i = 2, f = 1;     //循环初始化
for (; i <= n ;) //仅给出循环条件
{
   f *= i;
   i++;         //下一次循环准备
}                  //循环体
```

```
i = 2, f = 1;     //循环初始化
while(i <= n) //仅给出循环条件
{
   f *= i;
   i++;         //下一次循环准备
}
```

功能相同的两种写法

```
for (i = 2,f = 1 ;  i <= n   ;  i++)
{
   f *= i;    //重复操作
}              //循环体
```

```
i = 2, f = 1;     //循环初始化
for (; i <= n ;) //仅给出循环条件
{
   f *= i;
   i++;          //下一次循环准备
}                //循环体
```

```
i = 2, f = 1;     //循环初始化
for (; ;)
{
   f *= i;
   i++;          //下一次循环准备
   if(i > n) //原循环条件不成立
      break;
}                //循环体
```

- <表达式1>可以是带有初始化的变量定义，例如：

```
for (int i = 1 ; i <= 10; i++)
        <语句>
printf("%d", i)  //??
```

  - i的有效范围？
    - C++国际标准规定，i只能在定义它的循环语句中使用，出了循环不能使用i。
    - 有的C++实现允许出了循环也能使用i（如vc++6.0）
    - 如果需要在后续代码中使用i，建议在循环前进行定义！

循环的种类

- 计数控制的循环
  - 循环前就知道循环的次数，循环时重复执行循环体直到指定的次数。
  - 用于计数的变量称为"循环控制变量"。
  - 循环的执行次数不依赖于循环体的执行结果。
- 事件控制的循环
  - 循环前不知道循环的次数，循环的终止是由循环体的某次执行导致循环的结束条件得到满足而引起的。
  - 循环的执行次数要依赖于循环体的执行结果。

三种循环语句的使用原则

- 三种循环语句在表达能力上是等价的，在解决某个具体问题时，用其中的一种可能会比其它两种更加自然。
- 一般来说，
  - 计数控制的循环一般用for语句；
  - 事件控制的循环一般用while或do-while语句，其中，如果循环体至少要执行一次，则用do-while语句。
  - 由于for语句能清晰地表示"循环初始化"、"循环条件"以及"下一次循环准备"，因此，一些非计数控制的循环也常用for语句实现。

示例：计算键盘输入的一系列整数之和（计数控制）

```
#include <iostream>
using namespace std;
int main()
{
    int n;
    cout << "请输入整数的个数: ";
    cin >> n;
    cout << "请输入" << n << "个整数: ";
    int sum=0;
    for (int i=1; i<=n; i++) //i是循环控制变量!
    {
        int a;
        cin >> a;
        sum += a;
    }
    cout << "输入的" << n << "个整数的和是: " << sum << endl;
    return 0;
}
```

示例：计算键盘输入的一系列整数之和（事件控制）

```
#include <iostream>
using namespace std;
int main()
{
   int a,sum=0;
   cout << "请输入若干个整数（以0结束）：";
   cin >> a;
   while (a != 0)
   {
      sum += a;
      cin >> a;
   }
   cout << "输入的整数的和是: " << sum << endl;
   return 0;
}
```

示例：计算键盘输入的一系列整数之和（事件控制）

```
#include <iostream>
using namespace std;
int main()
{
   char ch;
   do
   {
      cout << "请输入Yes或No (y/n) : ";
      cin >> ch;
   }
   while (ch != 'y' && ch != 'n');
   if (ch == 'y')
      ......
   else
      ......
}
```

"死循环"

- 在循环体或for语句的<表达式3>中一定要有能改变循环条件中操作数值的操作，并逐步使得循环条件有不满足的趋势，否则将会出现"死循环"：循环永远结束不了！

```
for (i = 2,f = 1;  i = n;  i++)
{
   f *= i;    //重复操作
}              //循环体
```

```
for (i = 2,f = 1;  i <= n  ;  i--)
{
   f *= i;    //重复操作
}              //循环体
```

示例：利用循环提高程序的鲁棒性

```c
#include <stdio.h>

int main()
{
   double r;
   scanf("%lf", &r);
   double s = 3.14*r*r;
   ……
   return 0;
}
```

输入负数?

```c
#include <stdio.h>

int main()
{
   double r;
   scanf("%lf", &r);
   while(r <= 0)
   {
      scanf("%lf", &r);
   }
   double s = 3.14*r*r;
   ……
   return 0;
}
```

事件控制的循环

```c
#include <stdio.h>
int main()
{
    double r = 0;
    scanf("%lf", &r);
    while(r <= 0)
    {
        scanf("%lf", &r);
    }
    double s = 3.14*r*r;
    ……
    return 0;
}
```

```c
#include <stdio.h>
int main()
{
    double r = 0;
    if(! scanf("%lf", &r) )
        return -1;
    while(r <= 0)
    {
        if(! scanf("%lf", &r) )
            return -1;
    }
    double s = 3.14*r*r;
    ……
    return 0;
```

为了防止这种情况，可添加判断和提前终止程序语句。

对于非字符型变量 r，程序运行时若输入字符，r 获取不到输入数据，scanf 库函数将返回 0（正常情况返回 1），导致之后的 scanf 库函数不再被执行，从而有可能导致死循环。

基于循环的程序设计举例

- 循环是一个使用非常频繁的程序设计技术，一个程序如果没有循环，则该程序一般做不了太复杂的事情，往往是循环操作使得程序变得复杂和功能强大起来。
- 对初学者而言，程序设计往往是一件很困难的工作，尤其是在如何发现和组织循环方面。

示例：求第n个费波那契(Fibonacci)数

```
//1,1,2,3,5,8,13,...
#include <iostream>
using namespace std;
int main()
{
   int n;
   cin >> n;
   int fib_1=1;          //第一个Fibonacci数
   int fib_2=1;          //第二个Fibonacci数
   for (int i=3; i<=n; i++)
  {
      fib_2 = fib_1 + fib_2; //计算和记住新的Fibonacci数
      fib_1 = fib_2 - fib_1; //记住前一个Fibonacci数
   }
   cout << "第" << n << "个费波那契数是: " << fib_2 << endl;
   return 0;
}
```

示例：用牛顿迭代法求三次方根

- 计算 $\sqrt[3]{a}$ 的牛顿迭代公式为：

$$x_{n+1} = \frac{1}{3}(2x_n + \frac{a}{x_n^2})$$

- 取x_0为a（任何值都可以，但影响收敛速度！）
- 依次计算x_1、x_2、…
- 直到：$|x_{n+1}-x_n| < \varepsilon$（ε为一个很小的数，可设为$10^{-6}$）时为止
- x_{n+1}即为三次方根的值

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{   const double EPS=1e-6;      //一个很小的数
    double a,x1,x2;             //x1和x2分别用于存储最新算出的两个值
    cout << "请输入一个数: ";
    cin >> a;
    x1 = a;                     //第一个值取a
    x2 = (2*x1+a/(x1*x1))/3;    //计算第二个值
    while (fabs(x2-x1) >= EPS)
    {  x1 = x2;                   //记住前一个值
       x2 = (2*x1+a/(x1*x1))/3;  //计算新的值
    }
    cout << a << "的立方根是: " << x2 << endl;
    return 0;
}
```

上面程序中，"x2 = (2*x1+a/(x1*x1))/3;"写了两次，容易造成不一致错误！

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{   const double EPS=1e-6;          //一个很小的数
    double a,x1,x2;                 //x1和x2分别用于存储最新算出的两个值
    cout << "请输入一个数: ";
    cin >> a;
    x2 = a;                         //第一个值取a
    do
    {  x1 = x2;                     //记住前一个值
       x2 = (2*x1+a/(x1*x1))/3;     //计算新的值
    } while (fabs(x2-x1) >= EPS);
    cout << a << "的立方根是: " << x2 << endl;
    return 0;
}
```

循环流程的嵌套

- 循环流程也可以嵌套，即循环体中又含有循环流程。

```
for(int i = 1; i <= 9; ++i)
{
   for(int j = 1; j <= 9; ++j)
      printf("%d \t", j);
   printf("\n");
}
```

```
#include <iomanip>
cout << setw(8) << i*j;
```

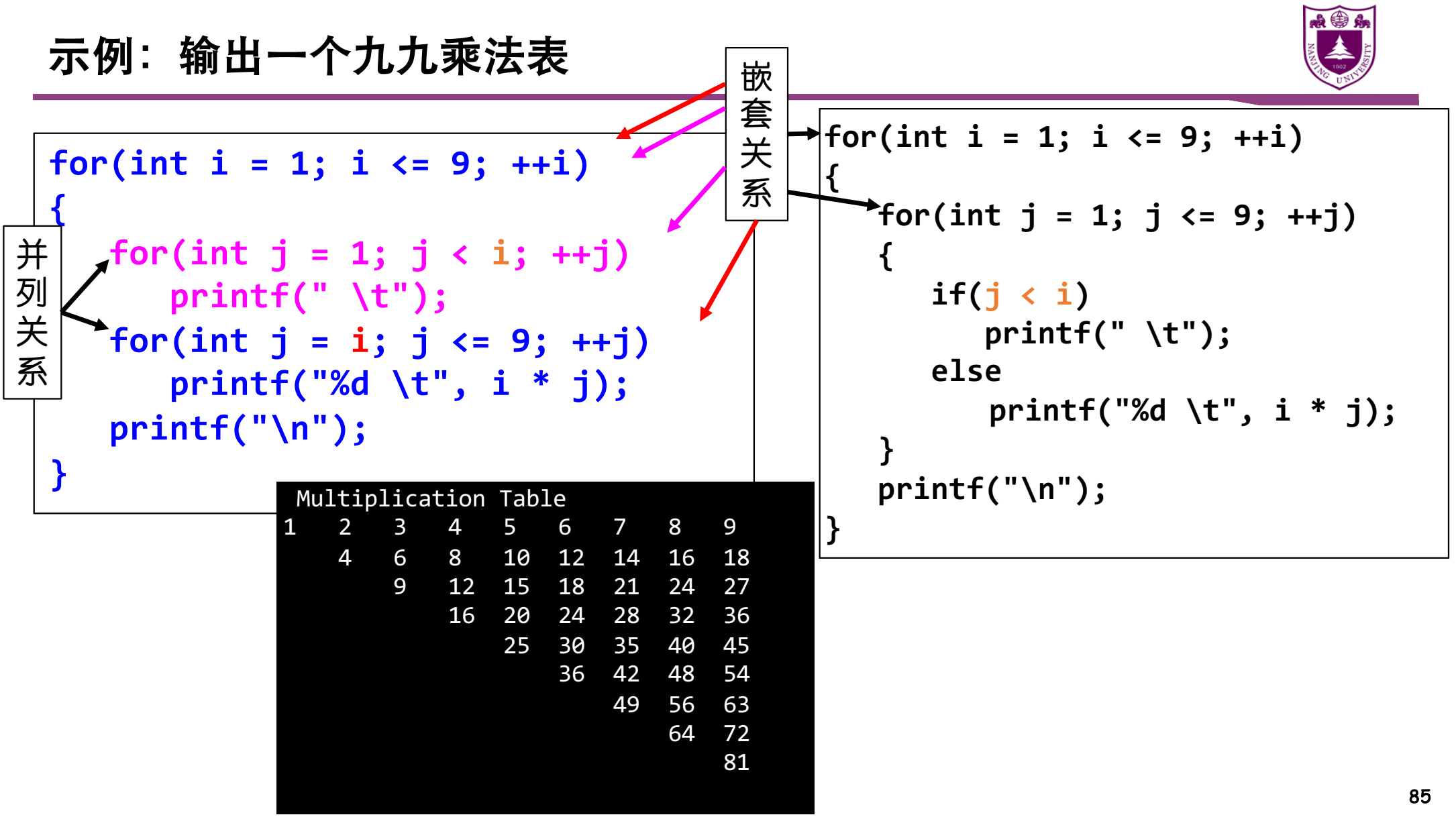
```
1   2   3   4   5   6   7   8   9
1   2   3   4   5   6   7   8   9
1   2   3   4   5   6   7   8   9
1   2   3   4   5   6   7   8   9
1   2   3   4   5   6   7   8   9
1   2   3   4   5   6   7   8   9
1   2   3   4   5   6   7   8   9
1   2   3   4   5   6   7   8   9
1   2   3   4   5   6   7   8   9
1   2   3   4   5   6   7   8   9
1   2   3   4   5   6   7   8   9
```

示例：输出一个九九乘法表

```
…
int main()
{
    printf(" Multiplication Table \n");
    for(int i = 1; i <= 9; ++i)
    {
        for(int j = 1; j <= 9; ++j)
            printf("%d \t", i * j);
        printf("\n");
    }
    return 0;
}
```

```
 Multiplication Table
1   2   3   4   5   6   7   8   9
2   4   6   8   10  12  14  16  18
3   6   9   12  15  18  21  24  27
4   8   12  16  20  24  28  32  36
5   10  15  20  25  30  35  40  45
6   12  18  24  30  36  42  48  54
7   14  21  28  35  42  49  56  63
8   16  24  32  40  48  56  64  72
9   18  27  36  45  54  63  72  81
```

示例：输出一个九九乘法表



示例：求每个输入的正整数的阶乘并输出（输入0停止）

```
int main()
{
    int n, i = 2, f = 1; //f 要初始化!
    printf("Please input an integer:\n");
    scanf("%d", &n);
    while(i <= n)
    {
        f *= i;
        ++i;
    }
    printf("factorial of %d is: %d \n", n, f);
    return 0;
}
```

```
int main()
{       int n, i, f;
        scanf("%d", &n);
        while(n != 0)
        {
            i = 2, f = 1;
            while(i <= n)
            {
                f *= i;
                ++i;
            }
            printf("%d", f);
            scanf("%d", &n);
        }
        return 0;
}
```

外循环是**事件控制型循环**，即其循环条件是"n != 0"这个事件是否发生(while/do-while更适合)；

内循环是**计数控制型循环**，其循环条件是计数变量 i 是否达到边界值 n (for更适合)。

循环优化问题

- 基础实现也能完成给定的计算任务！
- 与基础实现相比，优化往往来源于对问题和对程序的深入思考！

- 算法的优化：减少循环次数
- 避免在循环中重复计算不变的表达式

示例：编程求出小于n的所有素数（穷举法）

```
#include <iostream>
using namespace std;
int main()
{   int n;
    cout << "请输入一个正整数: "
    cin >> n;                          //从键盘输入一个正整数
    for (int i=2; i<n; i++)            //循环: 分别判断i=2、3、...、n-1是否为素数
    {
        int j=2;
        while (j < i && i%j != 0) //循环: 分别判断i是否能被2 ~ i-1整除
            j++;
        if (j == i) //i是素数
            cout << i << "  ";
    }
    cout << endl;
    return 0;
}
```

注意：1、上面的for循环中，偶数没有必要再判断它们是否为素数；
2、上面的while循环中j没有必要到i-1，只需要到：sqrt(i)

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{   int n;
    cin >> n;                                  //从键盘输入一个数
    if (n <= 2) return -1;
    cout << 2 << ",";                          //输出第一个素数
    for (int i=3; i<n; i+=2)                    //循环：分别判断3、5、...、是否为素数
    {   int j=2;
        while (j<=sqrt(i) && i%j!=0) //循环：分别判断i是否为素数
            j++;
        if (j > sqrt(i))                       //i是素数
            cout << i << ",";
    }
    cout << endl;
    return 0;
}
```

注意：上面程序中的sqrt(i)被重复计算！

```
......
int j = 2,k=sqrt(i);
while ( j <= k && i%j != 0)
   j++;
if (j > k) //i是素数。
......
```

注意：对有些循环优化，编译器能自动实现！

是否有更高效的找到素数的方法?

6倍原理（大于5的素数为6n+1或6n+5; 搜索因子时仅需要搜6n+1或6n+5）

筛法求素数（排除法，素数的倍数都不是素数）

try: 你能找到的最大的素数是多少?

哥德巴赫猜想和陈氏定理

- 1742年哥德巴赫："任一大于2的整数都可写成三个质数之和"
  - 当时仍把1看做质数
- 现代版本："任一大于5的整数都可写成三个质数之和"
  - 2 + (n − 2); 3 + (n − 3)
- 欧拉版本："任一大于2的偶数都可写成两个质数之和"
  - a + b (a, b为最大素因子个数)
  - 1930s起：华罗庚、王元、陈景润、潘承洞等
  - 1966/1973年陈景润的陈氏定理：1 + 2

- 能否设计一个程序验证一下一定范围内的陈氏定理?

更多详情可查阅在线百科中相关介绍

示例："百元买百鸡"问题

- 我国古代数学家张丘建在《算经》一书中曾提出过著名的"百钱买百鸡"问题：
  - 鸡翁一，值钱五；鸡母一，值钱三；鸡雏三，值钱一；
  - 百钱买百鸡，则翁、母、雏各几何？

- 用100元钱买100只鸡：
  - 公鸡5元1只
  - 母鸡3元1只
  - 小鸡1元3只（必须整3只买！）
  - 有几种买法？（公鸡、母鸡、小鸡各几只？）

问题抽象

- 用roosters、hens、chickens分别表示公鸡、母鸡和小鸡的数量，尝试下面所有可能的值：
  - 0 ⩽ roosters ⩽ 100
  - 0 ⩽ hens ⩽ 100
  - 0 ⩽ chickens ⩽ 100
- 如果它们满足下面的条件：
  - roosters + hens + chickens = 100（只）
  - chickens%3 = 0
  - roosters*5 + hens*3 + chickens/3 = 100（元）

则它们是一个解。

程序

```
const int MONEY_TOTAL=100; //钱的总数: 100元
const int NUM_TOTAL=100; //鸡的总数: 100只
......
cout << ""百元买百鸡"问题所有可能的解如下: \n";

for (int r=0; r <= NUM_TOTAL; r++) //r: 公鸡的数量
{  for(int h=0; h <= NUM_TOTAL; h++ ) //h: 母鸡的数量
    { for(int c=0; c <= NUM_TOTAL; c++ ) //c: 小鸡的数量
       { if (r+h+c == NUM_TOTAL && c%3 == 0
             && r*5+h*3+c/3 == MONEY_TOTAL)
          {  cout << "公鸡 " << r << "只, ";
             cout << "母鸡 " << h << "只, ";
             cout << "小鸡 " << c << "只\n";
          }
       }
    }
}
......
```

优化

```
const int MONEY_TOTAL=100;
const int NUM_TOTAL=100;
......
cout << ""百元买百鸡"问题所有可能的解如下: \n";

for (int r=0,r_max=MONEY_TOTAL/5; r <= r_max; r++)
{  for(int h=0,h_max=(MONEY_TOTAL-r*5)/3; h<=h_max; h++)
    {
       int c=NUM_TOTAL-r-h;
       if (c%3 == 0 && r*5+h*3+c/3 == MONEY_TOTAL)
       {  cout << "公鸡 " << r << "只, ";
          cout << "母鸡 " << h << "只, ";
          cout << "小鸡 " << c << "只\n";
       }
    }
}
......
```

无条件转移控制

- 除了有条件的选择语句（if和switch）外，C++还提供了无条件的转移语句：
  - break
  - continue
  - return
  - goto

break语句

- break语句的格式：

  break;

- break语句的含义有两个：
  - 结束switch语句的某个分支的执行
  - 退出包含它的最内层循环语句（由于循环可以嵌套）
- 在循环体中只要执行了break语句，就立即跳出（结束）循环，循环体中跟在break语句后面的语句将不再执行，程序继续执行循环之后的语句。
- 在循环体中，break语句一般作为某个if语句的子句，用于实现进一步的循环控制。

- 对于事件控制的循环，有时循环条件比较复杂，这时，可以在循环条件表达式中给出主要的循环控制描述，而在循环体中进行特殊情况的循环控制。

示例：求输入10个整数的和，遇到负数或0就终止。

```
int d, sum = 0, i = 1;
while(i <= 10)
{
   scanf("%d", &d);
   if(d <= 0) break;//结束循环流程
   sum += d;
   ++i;
}
printf("sum: %d \n", sum);
```

```
scanf("%d", &d);
if(d > 0)
{
   sum += d;
   ++i;
}  //是否等价?
```

这次运行，循环只完整地执行了2次.

这次运行，循环只完整地执行了4次.

注意：使用break语句后，循环的执行次数可能少于预计的次数（折断）

功能相同的两种写法（回顾）

```
for (i = 2,f = 1 ;  i <= n   ;  i++)
{
   f *= i;    //重复操作
}              //循环体
```

```
i = 2, f = 1;     //循环初始化
for (; i <= n ;) //仅给出循环条件
{
   f *= i;
   i++;          //下一次循环准备
}                //循环体
```

```
i = 2, f = 1;     //循环初始化
for (; ;)
{
   f *= i;
   i++;          //下一次循环准备
   if(i > n) //原循环条件不成立
      break;
}                //循环体
```

break语句给出了一种中断循环执行的方式

- 对于多重循环，内层循环语句循环体中的break语句只能用于结束内层循环语句的执行。

```
while (...) //外层循环
{  ...
    for (...) //内层循环
    { ......
        ... break; //转到紧接在for循环语句后面的语句
        ......
    }
    ...
}
```

示例：输出矩形中的部分元素

```
for(int i = 1; i <= 9; ++i)
{
   for(int j = 1; j <= 9; ++j)
   {
      if(i * j > 10)   break;   //并非一旦乘积超过10就结束程序
      printf("%d \t", i * j);
   }
   printf("\n");
}
```

```
1   2   3   4   5   6   7   8   9
2   4   6   8   10
3   6   9
4   8
5   10
6
7
8
9
```

如何跳出两重循环?

```
1   2   3   4   5   6   7   8   9
2   4   6   8   10
```

```
1   2   3   4   5   6   7   8   9
2   4   6   8   10
```

```
for(int i = 1; i <= 9; ++i)
{    int j;
     for(j = 1; j <= 9; ++j)
     {
        if(i * j > 10)     break;
        printf("%d \t", i * j);
     }▲
     if(i * j > 10)
        break;
     printf("\n");
}▲
```

```
int flag = 1; //使用标志变量辅助判断
for(int i = 1; i <= 9 && flag; ++i)
{
    for(int j = 1; j <= 9 && flag; ++j)
    {
        if(i * j > 10)
            flag = 0;
        else
            printf("%d \t", i * j);
    }
    printf("\n");
}
```

continue语句

- continue语句的格式如下：

  ```
  continue;
  ```

- continue语句只能用在循环语句的循环体中，其含义是：立即结束当前循环，准备进入下一次循环。
  - 对于while和do-while语句，continue语句将使控制转到循环条件的判断；
  - 对于for语句，continue语句将使控制转到<表达式3>的计算。
- 在循环体中，continue语句一般作为某个if语句的子句，用于实现进一步的循环控制。

示例：求输入10个整数的和，遇到负数或0就忽略。

```
int d, sum = 0, i = 1;
while(i <= 10)
{
   scanf("%d", &d);
   if(d <= 0) continue;//接续下一次
   sum += d;
   ++i;
}
printf("sum: %d \n", sum);
```

```
scanf("%d", &d);
if(d > 0)
{
   sum += d;
   ++i;
}  //是否等价?
```

```
1
2
10
11
sum=63
```

循环至少要完整地执行10次.

注意：使用continue语句后，循环的执行次数不变。
但循环体中的部分语句不再执行（接续）。

goto语句

- goto语句的格式如下：

  goto <语句标号>;

  - <语句标号>为标识符，其定义格式为：

    <语句标号>: <语句>

- goto的含义是：程序转移到带有<语句标号>的语句

用goto语句求n!

```
#include <iostream>
using namespace std;
int main()
{  int n;
   cin >> n;
   int i=1,f=1;
loop:
   f *= i;
   i++;
   if (i <= n) goto loop;
   cout << "factorial of " << n << "=" << f << endl;
   return 0;
}
```

- 在使用goto语句时，应该注意：
  - 不能用goto语句从一个函数外部转入该函数的内部（函数体），也不能用goto语句从一个函数的内部转到该函数的外部。
  - 允许用goto语句从内层复合语句转到外层复合语句或从外层复合语句转入内层复合语句。
  - goto语句不能掠过带有初始化的变量定义。

```
int main()
{  ......
   goto L1;  //Error
   ......
   while (...)
   {  int x=0; //带有初始化的变量定义
L1:  ...
       ......
       goto L2; //Error
       ......
   }
   .....
   int y=10; //带有初始化的变量定义
L2:......
   ......
}
```

用goto语句实现break语句的功能

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

```
   while (...)
   {  ......
      ... goto L;
      ......
   }
L: ...
```

用空语句和goto语句实现continue语句的功能

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

```
while (...)
{  ......
   ... goto end;
   ......
end:
   ;
}
```

示例：键盘输入非零整数，求其中正数的平方根。

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{  int n;
   double square_root;
   cout << "请输入若干整数（以0结束）：";
   for ( cin >> n; n != 0; cin >> n)
   {
      if (n < 0) continue; //准备进入下一次循环
      square_root = sqrt((double)n);
      cout << n << "的平方根是: " << square_root << endl;
   }
   return 0;
}
```

- 上面的循环虽然可以写成：

```
for (cin >> n ; n != 0; cin >> n)
{
    if (n >= 0)
    {
        square_root = sqrt((double)n);
       cout << n << "的平方根是:  " << square_root << endl;
    }
}
```

但当循环体复杂以后，就比较麻烦！

```
while (...)
{ ......
  if (b1) continue;
  ......
  if (b2) continue;
  ......
  if (b3) continue;
  ......
}
```

上述程序，如果不用continue会是什么样的？

如果循环程序中不用break会是什么样的？

```
while (...)
{ ......
  if (!b1)
  {
     ......
    if (!b2)
    {
       ......
       if (!b3)
       {
          ......
       }
    }
  }
}
```

程序设计风格

- 程序设计风格通常是指对程序进行静态分析所能确认的程序特性，它涉及程序的易读性和可维护性。
  - 采用一致/有意义的标识符为程序实体（如：变量、函数等）命名
  - 使用符号常量
  - 为程序书写注释
  - 采用代码的缩进格式，等等
- 除此之外，结构化程序设计也是一种良好程序设计风格的典范。

结构化程序设计

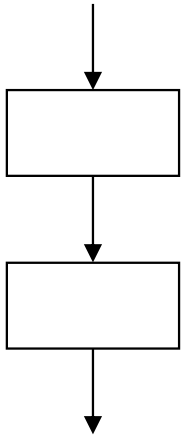
- 结构化程序设计（Structured Programming，简称SP）是指"按照一组能够提高程序易读性与易维护性的规则进行程序设计的方法"
- SP不仅要求所编出的程序结构良好，而且还要求程序设计过程也是结构良好的，后者是前者的基础。

结构化程序设计

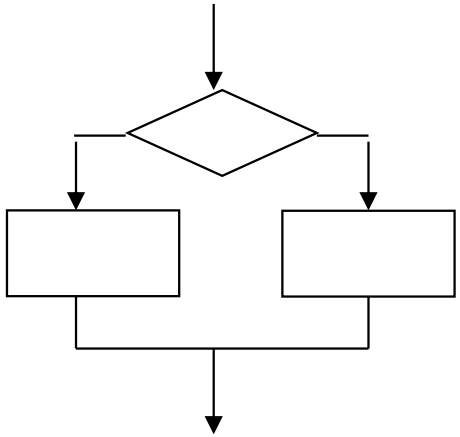
- 对程序而言，"结构良好"是指：
  - 每个程序单位应具有单入口、单出口的性质（降低耦合）。
  - 不包含不会停止执行的语句，程序在有限时间内结束。
  - 程序中没有无用语句，程序中所有语句都有被执行的机会。
- 对程序设计过程而言，"结构良好"是指
  - 采用分解和抽象的方法来完成程序设计任务，
  - 具体体现为："自顶向下、逐步精化"的程序设计过程。

结构化程序设计（续）

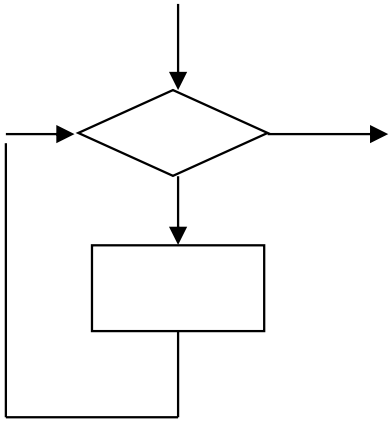
- 结构化程序设计通常可用三种基本结构来实现：



（顺序）



（选择）



（循环）

- 上面三种结构都具有单入口、单出口的性质，从而减少程序各部分之间的耦合度（纠缠度），便于程序的设计、理解与维护，保证程序的正确性。

关于goto语句

- goto语句会使得程序的静态结构和动态结构不一致，导致程序难以理解、可靠性下降和不容易维护。有时会导致程序效率的下降。
- 从结构化程序设计的角度讲， goto语句会破坏程序中的每一个结构所具有的单入口/单出口的性质。
- 实际上，goto语句的使用可以分成两类：
  - 向前的转移（forward）：可用分支结构实现
  - 往回的转移（backward）：可用循环结构实现
- 尽量不要使用goto语句!

小结

- **教学要求：**
  - 了解程序流程控制的基本原理
  - 掌握利用计数和事件循环进行计算机问题求解的方法

- **实践：**
  - 利用程序流程控制求解计算机问题
  - 培养良好的编程风格

- **阅读：教材第三章相关内容**