

# 第三章 程序的流程控制 —— 语句



# 本章内容

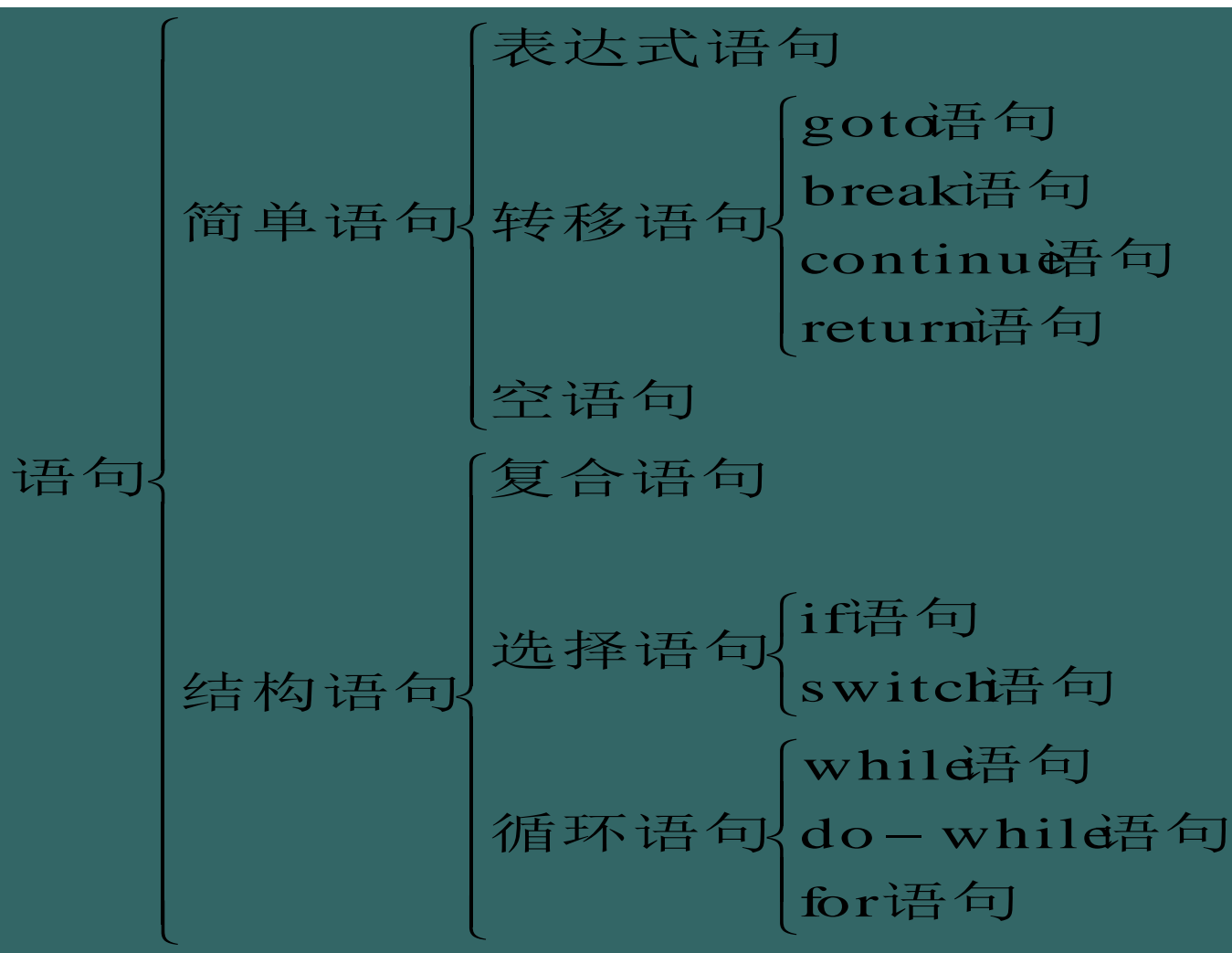
- 语句概述
- 表达式语句
- 复合语句
- 选择语句
- 循环语句
- 转移语句



# 语句概述

- 表达式构成了**数据处理的基本单位**。
- 语句实现对程序执行流程的控制，包括：
  - 顺序控制：按书写次序执行。
  - 分支控制：根据条件选择执行。
  - 循环控制：重复执行。

# C++语句的分类





# 表达式语句

- 在C++表达式后面加上分号
  - 格式：<表达式>;
  - 例如：a + b \* c;
- 连续的多个表达式语句按它们的书写次序（从左到右、从上到下）依次执行。



# 复合语句

- 复合语句是由花括号括起来的一条或多条语句，又称为块（block）。语法上，复合语句可看作是一个语句。
  - 格式：{ <语句序列> }
  - <语句序列>中的语句可以是任何的C++语句，其中包括数据定义和声明语句。



## 复合语句举例

```
{ int a,b;  
  cin >> a >> b;  
  int max;  
  if (a >= b) //选择语句  
      max = a;  
  else  
      max = b;  
  cout << max << endl;  
}
```



# 选择语句

- 根据不同的情况来从一组语句中选择一个来执行（分支）。
  - if语句
  - switch语句





## if 语句

- 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 (<表达式>) <语句>
  - if (<表达式>) <语句1> else <语句2>
  - 其中的<语句>、<语句1>、<语句2>必须是一个语句！（复合语句算一个语句。）

# if语句的锯齿格式

- 写if语句时，最好采用“锯齿”格式。但如果嵌套层次很深，将使得程序正文严重偏向右边。这时，可以按下面格式书写：

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

等价于：

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



## 避免不必要的测试

```
if (score >= 90)
    cout << "优";
if (score >= 80 && score < 90)
    cout << "良";
if (score >= 70 && score < 80)
    cout << "中";
if (score >= 60 && score < 70)
    cout << "及格";
if (score < 60)
    cout << "不及格";
```

# if 语句的歧义问题

- if (<表达式1>) if (<表达式2>) <语句1>  
else <语句2>
  - **else子句与它前面最近的、没有else子句的if配对。**  
if (<表达式1>) if (<表达式2>) <语句1>  
else <语句2>
  - 加上花括号（复合语句）：  
if (<表达式1>) { if (<表达式2>) <语句1> }  
else <语句2>



# switch 语句

- 多路选择语句（又称开关语句）
  - 根据某个表达式的值在多组语句中选择一组语句来执行。
  - 每一组语句的最后一个语句往往是break语句。

# switch语句的格式

**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>]  
}
```



## switch语句中使用break语句

- 在执行某个分支时，用break语句结束该分支的执行。
- 如果没有break语句（最后一个分支除外），则继续执行紧接着的下一个分支中的语句序列。在其它一些语言（如：Pascal）的多路选择语句中，一个分支执行完后将自动结束多路选择语句的执行。



# Switch语句例子

```
#include <iostream>
using namespace std;
int main()
{   int day;
    cin >> day;
    switch (day)
    {   case 0: cout << "Sunday"; break;
        case 1: cout << "Monday"; break;
        case 2: cout << "Tuesday"; break;
        case 3: cout << "Wednesday"; break;
        case 4: cout << "Thursday"; break;
        case 5: cout << "Friday"; break;
        case 6: cout << "Saturday"; break;
        default: cout << "Input error";
    }
    cout << endl;
    return 0;
}
```



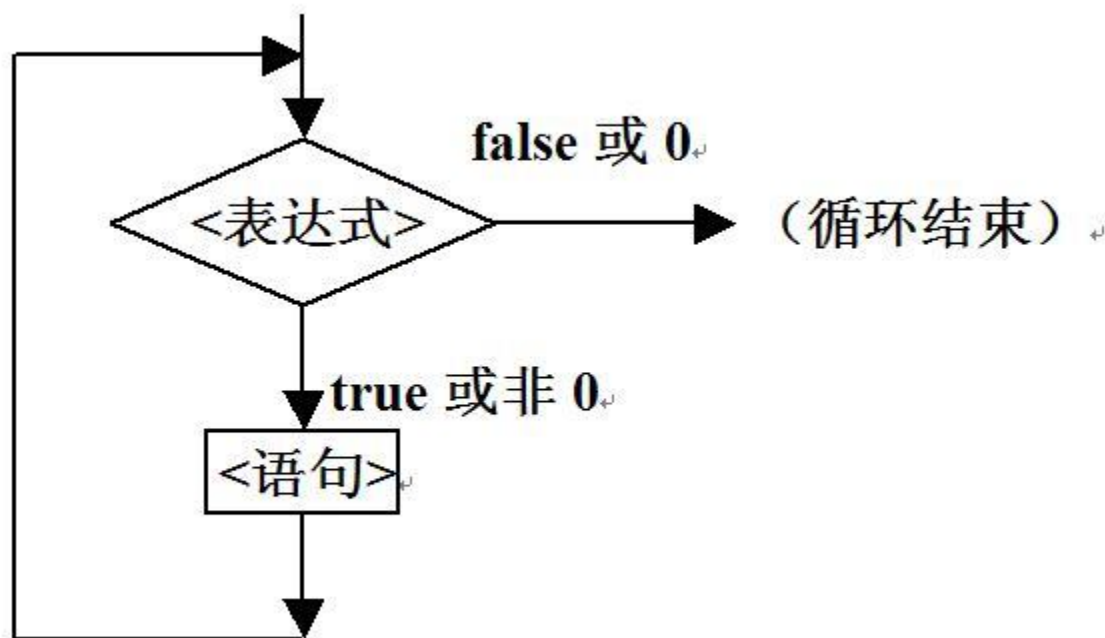


# 循环（重复）语句

- 解决重复操作
- 由四个部分组成：
  - 循环初始化
  - 循环条件
  - 循环体
  - 下一次循环准备
- C++ 三种循环语句：
  - while语句
  - do-while语句
  - for语句

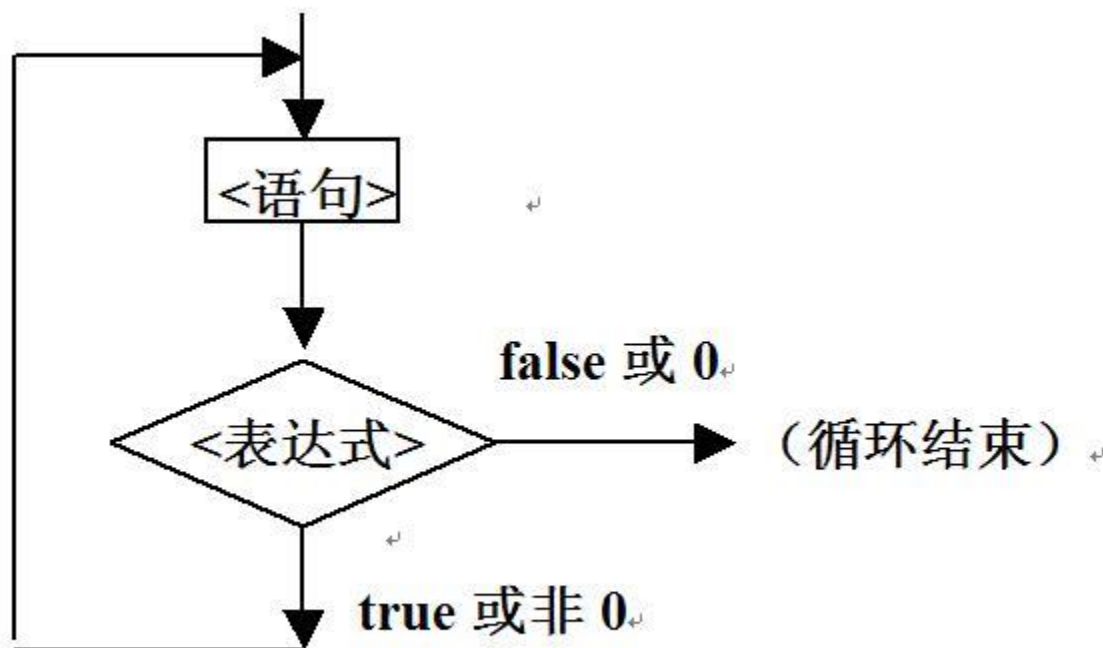
# while 语句

while (<表达式>) <语句>



# do-while 语句

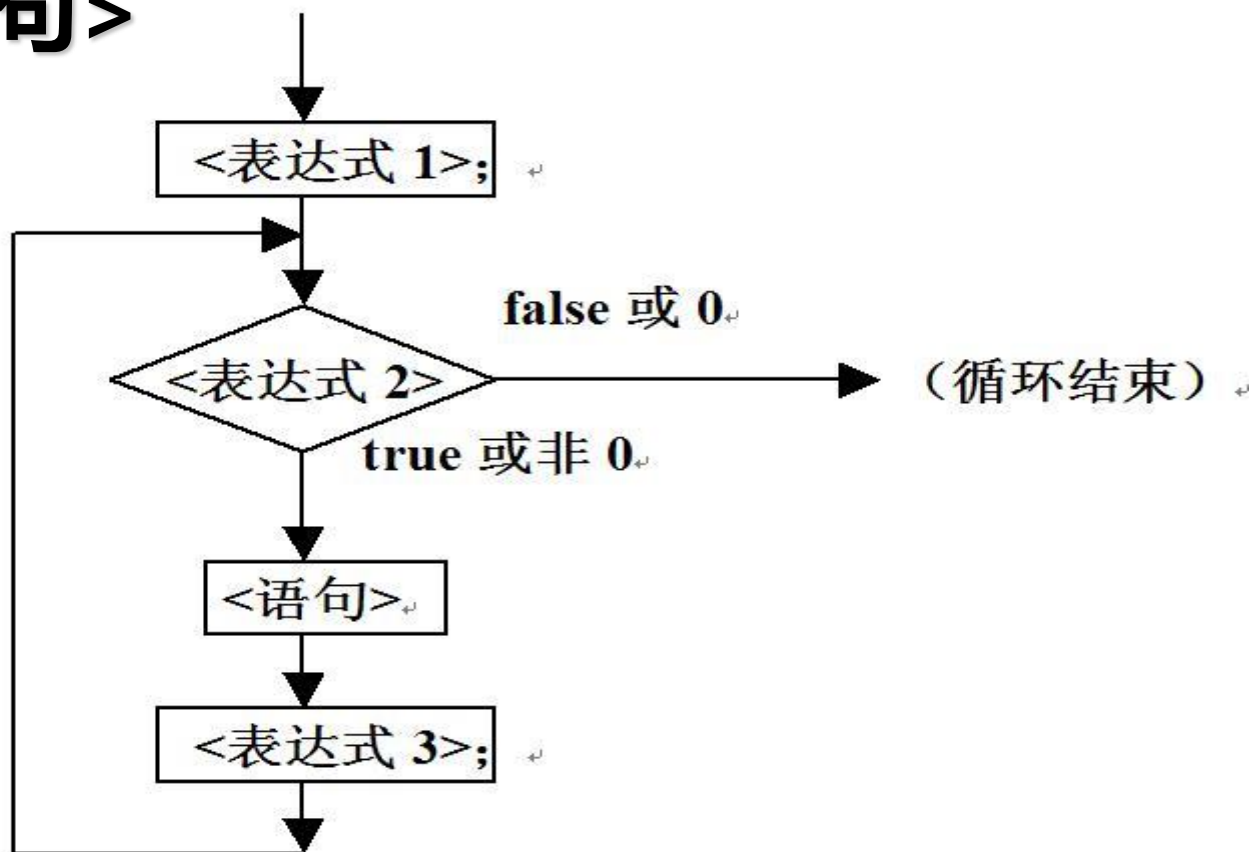
do <语句> while (<表达式>;



# for 语句

for (<表达式1>;<表达式2>;<表达式3>)

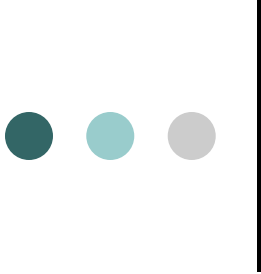
<语句>





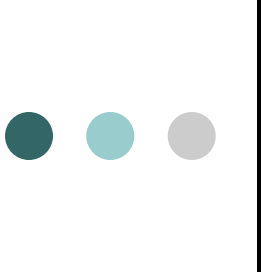
# 三种循环语句的使用原则

- 三种循环语句在表达能力等价。
- 使用原则：
  - 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用for语句实现。



**例：计算从键盘输入的一系列整数的和，要求首先输入整数的个数。（计数控制的循环）**

```
#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++)
    {   int a;
        cin >> a;
        sum += a;
    }
    cout << "输入的" << n << "个整数的和是: " << sum << endl;
    return 0;
}
```



**例：计算从键盘输入的一系列整数的和，要求输入以-1结束。（事件控制的循环）**

```
#include <iostream>
using namespace std;
int main()
{int a,sum=0;
  cout << "请输入若干个整数（以-1结束）： ";
  cin >> a;
  while (a != -1)
  {    sum += a;
      cin >> a;
  }
  cout << "输入的整数的和是： " << sum << endl;
  return 0;
}
```



**例 从键盘接收字符，一直到输入了字符y(Y)或n(N)为止。（事件控制的循环）**

```
#include <iostream>
#include <cstdlib>
using namespace std;
int main()
{   char ch;
    do
    {   cout << "请输入Yes或No (y/n) : ";
        cin >> ch;
        ch = tolower(ch);
    } while (ch != 'y' && ch != 'n');
    if (ch == 'y')
        .....
    else
        .....
    return 0;
}
```





# 循环优化问题

- 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

# 例：编程求出小于n的所有素数（质数）（1）

```
#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循环没有必要到i-1，只需要到：sqrt(i)

## 例：编程求出小于n的所有素数（质数）（2）

```
#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！



## 例：编程求出小于n的所有素数（质数）（3）

.....

```
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，编译器能实现！**



# 转移语句

- 除了if和switch外，C++还提供无条件的分支语句：
  - break
  - continue
  - goto (尽量不使用)
  - return (第4章)



# break语句

- 两个功能：
  - 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



# continue语句

- continue语句 **只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：**先计算<表达式3>，然后计算<表达式2>**，并根据<表达式2>的计算结果来决定是进入下一次循环还是结束循环。