

§ 3. 程序设计初步

3.1. 面向过程的程序设计和算法

3.1.1. 算法的基本概念

3.1.1.1. 算法的定义

广义定义：为解决某个特定问题而采用的具体的方法和步骤

计算机的定义：对特定问题求解步骤的一种描述，是指令的有限序列，每个指令包含一个或几个基本操作

★ 一个问题可以有多种算法

3.1.1.2. 算法的分类

数值算法： 求解数学值(方程/函数)

大数据量计算，体现运算复杂性(逻辑相对简单)

非数值算法：除数学值外的其它领域(一般用于事务管理领域)

大数据量管理，体现逻辑复杂性(运算相对简单)

3.1.1.3. 算法的基本特征

★ 输入：有0-n个输入

★ 输出：有1-n个输出

★ 确定性：每条指令有确切含义，不产生二义性；对相同输入只能得到相同输出

★ 有穷性：每个算法在有穷步骤内完成；每个步骤都在有穷时间内完成(时间要在合理范围内)

★ 有效性：算法中所有操作都可以通过已实现的基本运算执行有限次数来实现

§ 3. 程序设计初步

3.1. 面向过程的程序设计和算法

3.1.2. 程序的含义及组成

含义：可以在计算机执行的一组相关的指令及数据的集合，用来完成某一特定的任务及功能
组成：

- ┌ 数据的描述（静态）（数据结构）
 在程序中要指定的数据的类型及数据的组织形式
- └ 操作的描述（动态）（算法）
 对动作的描述（操作步骤）

★ 操作的对象是数据，即操作要依赖于数据

★ 数据的描述+操作的描述 = 程序

★ 数据结构 + 算法 = 程序

★ 数据结构+算法+程序设计方法+语言工具+开发环境 = 程序

3.1.3. 程序的三种基本结构

顺序结构：程序按语句排列的先后次序顺序执行

选择结构：程序根据某个条件的逻辑值（真/假）来决定是否执行某些语句

循环结构：反复执行某些语句

特点：

★ 仅有一个入口

★ 仅有一个出口

★ 每一部分均可能被执行

★ 不存在死循环

§ 3. 程序设计初步

3. 1. 面向过程的程序设计和算法

3. 1. 4. 算法的表示

例1: (顺序结构)

输出一个数字的平方

例2: (单分支结构)

当一个成绩小于60分时, 输出“不合格”

例3: (双分支结构)

当一个成绩小于60分时, 输出“不合格”,
否则输出“合格”

例4: (循环结构)

输出10个数字的平方

例5: (综合应用)

100个学生, 对每个学生, 当成绩小于60分时, 输出“不合格”, 否则输出“合格”

§ 3. 程序设计初步

3.1. 面向过程的程序设计和算法

3.1.4. 算法的表示

3.1.4.1. 自然语言表示

用自然文字进行描述

例1: (顺序结构) 输出一个数字的平方

步骤1: 从键盘读入一个数字

步骤2: 求该数的平方

步骤3: 输出该数的平方

例2: (单支结构) 当一个成绩小于60分时,
输出“不合格”

步骤1: 从键盘读入一个数字作为成绩

步骤2: 若该数小于60, 转步骤3, 否则
直接转步骤4

步骤3: 输出“不合格”

步骤4: 结束

例3: (双分支结构) 当一个成绩小于60分时,
输出“不合格”, 否则输出“合格”

步骤1: 从键盘读入一个数字作为成绩

步骤2: 若数小于60, 转步骤3, 否则转步骤4

步骤3: 输出“不合格”, 转步骤5

步骤4: 输出“合格”

步骤5: 结束

例4: (循环结构) 输出10个数字的平方

步骤1: 计数器置0

步骤2: 若计数器大于等于10, 转步骤7

步骤3: 从键盘读入一个数字

步骤4: 求该数的平方

步骤5: 输出该数的平方

步骤6: 计数器加1, 转步骤2

步骤7: 结束

例5: (综合应用) 100个学生, 对每个学生,
当成绩小于60分时, 输出“不合格”,
否则输出“合格”

步骤1: 计数器置0

步骤2: 若计数器大于等于100, 转步骤8

步骤3: 从键盘读入一个数字作为成绩

步骤4: 若该数小于60, 转步骤5, 否则转步骤6

步骤5: 输出“不合格”, 转步骤7

步骤6: 输出“合格”

步骤7: 计数器加1, 转步骤2

步骤8: 结束

§ 3. 程序设计初步

3.1. 面向过程的程序设计和算法

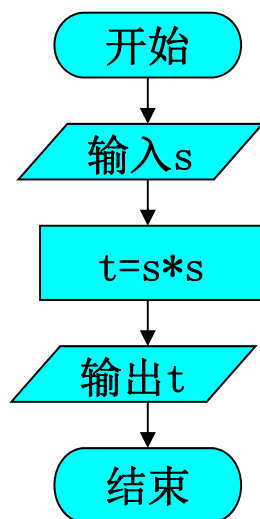
3.1.4. 算法的表示

3.1.4.2. 流程图表示

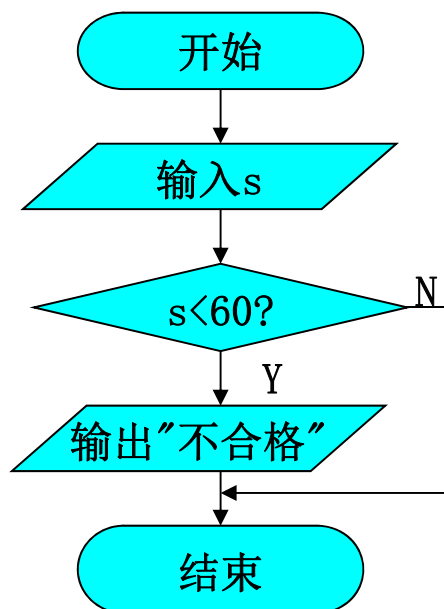
★ 基本图形表示（参考其它书籍）

★ 缺陷：对较大的程序，过于复杂，难以阅读和修改

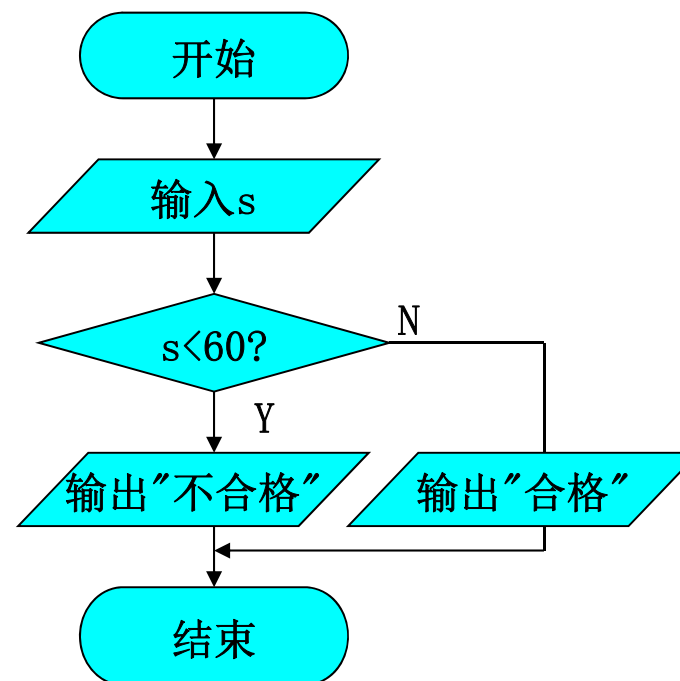
例1：（顺序结构）输出一个数字的平方



例2：（单支结构）当一个成绩小于60分时，输出“不合格”



例3：（双分支结构）当一个成绩小于60分时，输出“不合格”，否则输出“合格”



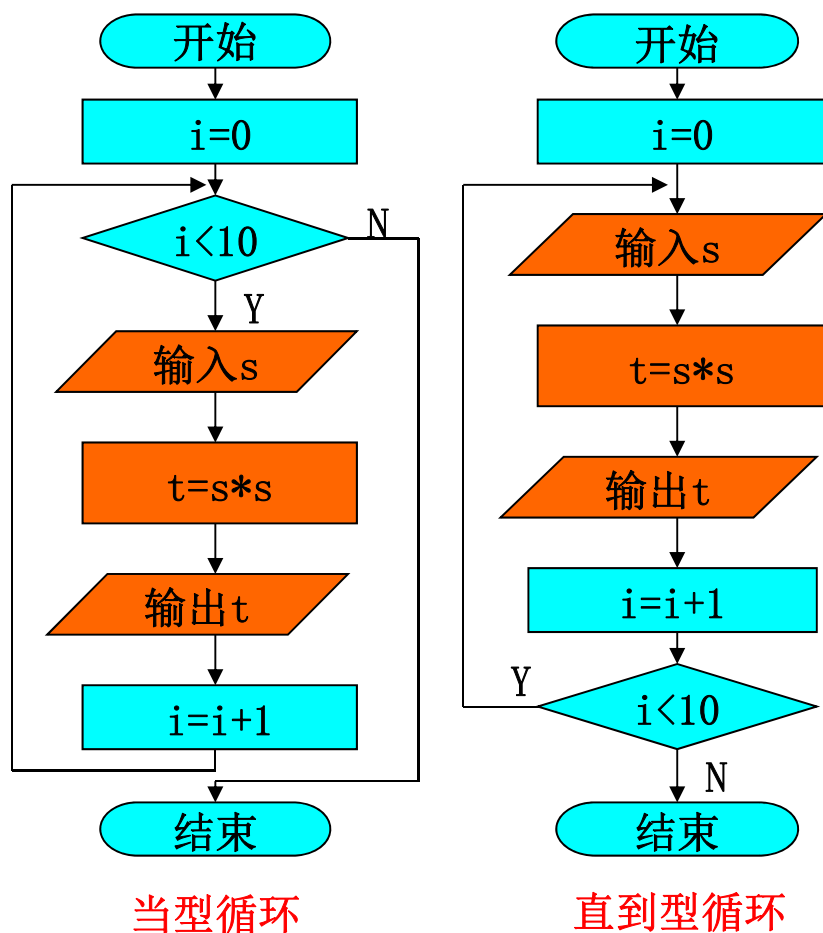
§ 3. 程序设计初步

3.1. 面向过程的程序设计和算法

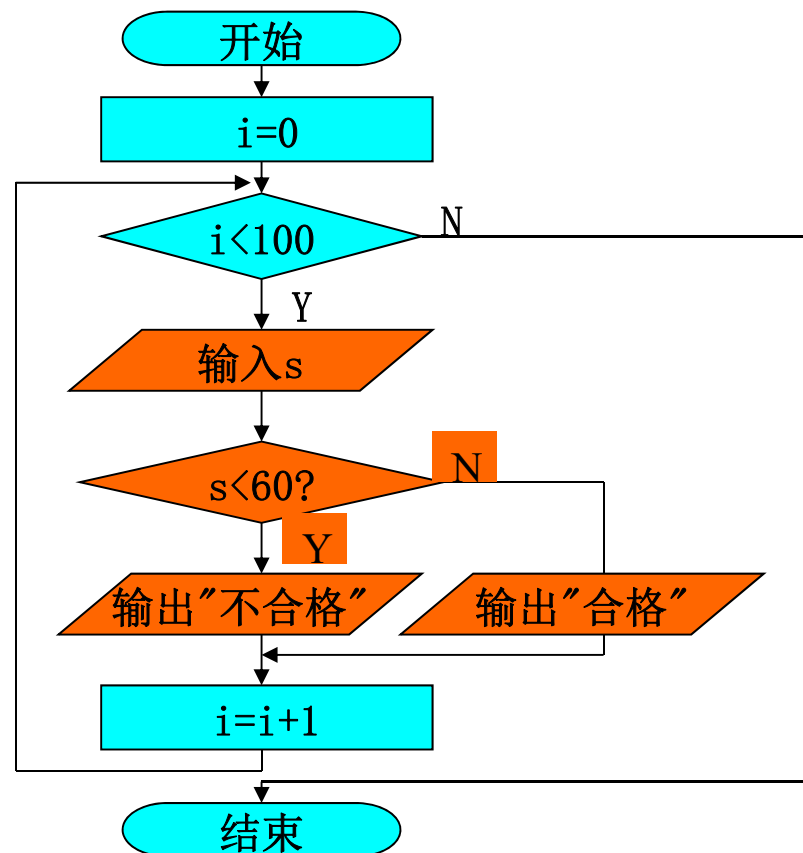
3.1.4. 算法的表示

3.1.4.2. 流程图表示

例4: (循环结构) 输出10个数字的平方



例5: (综合应用) 100个学生, 对每个学生, 当成绩小于60分时, 输出“不合格”, 否则输出“合格”



§ 3. 程序设计初步

3. 1. 面向过程的程序设计和算法

3. 1. 4. 算法的表示

3. 1. 4. 3. 伪代码表示

介于自然语言和计算机程序语言之间的表示

- ★ 比流程图简练，无图形，用文字表示
- ★ 比自然语言直观，无二义性
- ★ 不能直接在计算机上执行

例1: (顺序结构) 输出一个数字的平方

开始	BEGIN
输入s	input s
t = s*s	t = s*s
打印t的值	print t
结束	END

例2: (单支结构) 当一个成绩小于60分时,
输出“不合格”

```
BEGIN
  input s
  if s<60 then
    print "不合格"
END
```

例3: (双分支结构) 当一个成绩小于60分时,
输出“不合格”, 否则输出“合格”

```
BEGIN
  input s
  if s<60 then
    print "不合格"
  else
    print "合格"
END
```

§ 3. 程序设计初步

3.1. 面向过程的程序设计和算法

3.1.4. 算法的表示

3.1.4.3. 伪代码表示

例4: (循环结构) 输出10个数字的平方

BEGIN i=0 while i<10 { input s t = s*s print t i=i+1 } END	BEGIN i=0 do { input s t=s*s print t i=i+1 } while i<10 END
--	---

例5: (综合应用) 100个学生, 对每个学生,
当成绩小于60分时, 输出“不合格”,
否则输出“合格”

BEGIN i=0 while i<100 { input s if s<60 then print “不合格” else print “合格” i=i+1 } END	BEGIN i=0 do { input s if s<60 then print “不合格” else print “合格” i=i+1 } while i<100 END
--	---

§ 3. 程序设计初步

3. 1. 面向过程的程序设计和算法

3. 1. 4. 算法的表示

3. 1. 4. 1. 自然语言表示

3. 1. 4. 2. 流程图表示

3. 1. 4. 3. 伪代码表示

3. 1. 4. 4. 计算机语言表示

★ 用某种具体的程序设计语言来表示，可直接在计算机上运行，即程序

§ 3. 程序设计初步

3.2. C++的程序结构和C++语句

3.2.1. 程序的组成

- ★ 一个程序由若干源程序文件(*.cpp)及头文件(*.h)组成
- ★ 一个源程序文件由预处理指令、全局声明及若干函数组成
- ★ 一个函数由若干语句组成（定义语句、执行语句）

P.44 图3.1

包含的头文件
命名空间
常量定义
函数的定义
全局变量的定义
函数1
...
...
函数n

3.2.2. 语句的种类

P44~45 四种

- ★ 声明语句：定义变量，放在使用该变量的语句前面
- ★ 执行语句：

控制语句： P.44~45 9种

函数和流对象调用语句：

max(a, b); //调用函数max的语句
cout << x << endl; //调用流对象cout的语句

表达式语句：表达式+; 组成

a=3;

- ★ 空语句：只有一个;
- ★ 复合语句：用一对{...}组合而成的语句，里面可以若干声明、执行、空、复合语句

int a; //定义
a=10; //使用

C++中变量必须先定义、后使用

- 1、if、while、break等称为保留字
- 2、C++规定，标识符不能与保留字同名

§ 3. 程序设计初步

3.3. 赋值操作

赋值表达式+;

- ★ C++中赋值语句和赋值表达式有区别
- ★ C++中赋值表达式有值，可以参与表达式的运算

```
int a;  
(a=3)*10    //正确，赋值表达式，可参与运算  
(a=3;)*10   //错误，赋值语句，不能参与运算
```

§ 3. 程序设计初步

3. 4. C++的输入与输出

3. 4. 1. 流的基本概念

流的含义：流是来自设备或传给设备的一个数据流，由一系列**字节**组成，按顺序排列

★ C/C++的原生标准中没有定义输入/输出的基本语句

★ C语言用printf/scanf等函数来实现输入和输出，通过#include <stdio.h>来调用

★ C++通过cin和cout的流对象来实现，通过#include <iostream>来调用

cout：输出流对象 <<：流插入运算符

cin：输入流对象 >>：流提取运算符

P. 465 附录B：

- 1、>>和<<的优先级为6，称为**按位左移/右移运算符**，本章中所称的**流插入/流提取运算符**，本质上是将**按位左移/右移运算符**经过重载（第10章）而得到的
- 2、优先级15中 <<=和>>= 称为**复合按位左移/右移运算符**
- 3、优先级15的最后一个 **!=** 应该是 **|=**

§ 3. 程序设计初步

3. 4. C++的输入与输出

3. 4. 2. 输出流的基本操作

格式: `cout<<表达式1<<表达式2<<...<<表达式n;`

★ 插入的数据存储在缓冲区中, 不是立即输出, 要等到缓冲区满 (不同系统大小不同) 或者碰到换行符 ("`\n`" / `endl`) 或者强制立即输出 (`flush`) 才一齐输出

```
//Linux下编译运行
#include <iostream>
#include <unistd.h> //sleep
using namespace std;

int main()
{
    cout << "12345";
    sleep(5); //秒
    cout << "abcde" << endl;
    return 0;
}
```

```
//Linux下编译运行
#include <iostream>
#include <unistd.h> //sleep
using namespace std;

int main()
{
    cout << "12345" << endl;
    sleep(5); //秒
    cout << "abcde" << endl;
    return 0;
}
```

```
//Linux下编译运行
#include <iostream>
#include <unistd.h> //sleep
using namespace std;
int main()
{
    cout << "12345";
    cout.flush();
    sleep(5); //秒
    cout << "abcde" << endl;
    return 0;
}
```

```
cout << "hello" << endl;
cout << "hello\n";
cout << "hello" << "\n";
cout << "hello" << '\n';
```

换行符的多种形式

```
//Windows下编译运行
#include <iostream>
#include <Windows.h> //Sleep
using namespace std;

int main()
{
    cout << "12345";
    Sleep(1000*5); //毫秒
    cout << "abcde" << endl;
    return 0;
}
```

§ 3. 程序设计初步

3. 4. C++的输入与输出

3. 4. 2. 输出流的基本操作

格式: `cout<<表达式1<<表达式2<<...<<表达式n;`

★ 插入的数据存储在缓冲区中, 不是立即输出, 要等到缓冲区满 (不同系统大小不同) 或者碰到换行符 ("`\n`"/`endl`) 或者强制立即输出 (`flush`) 才一齐输出

★ 默认的输出设备是显示器(可更改, 称输出重定向)

★ 一个`cout`语句可写为若干行, 或者若干语句

```
cout << "This is a C++ program." << endl;
```

```
cout << "This is " << "a C++ " << "program." << endl;
```

```
cout << "This is "  
    << "a C++ "  
    << "program."  
    << endl;
```

一个语句分4行
前三行无分号

```
cout << "This is ";  
cout << "a C++ ";  
cout << "program.";  
cout << endl;
```

4个语句
每行有分号

§ 3. 程序设计初步

3. 4. C++的输入与输出

3. 4. 2. 输出流的基本操作

格式: `cout<<表达式1<<表达式2<<...<<表达式n;`

★ 一个`cout`的输出可以是一行, 也可以是多行, 多个`cout`的输出也可以是一行

```
cout << "hello\nhello"<<endl;
```

```
hello
hello
```

★ 一个插入运算符只能输出一个值

```
#include <iostream>
using namespace std;
```

```
int main()
{
    int a=10, b=15, c=20;
    cout << a << b << c;
    return 0;
}
```

`cout`只输出最原始的数据,
其它内容都需要自行加入

输出: 101520

```
cout << a << ' ' << b << ' ' << c;
```

```
#include <iostream>
using namespace std;
int main()
{
    int a=10, b=15, c=20;

    cout << a, b, c;
    cout << (a, b, c);
    cout << (a, b, c) << endl;
    cout << a, b, c << endl;

    return 0;
}
```

第1-3句`cout`输出什么?
第4句`cout`为什么编译错?

§ 3. 程序设计初步

3. 4. C++的输入与输出

3. 4. 2. 输出流的基本操作

格式: `cout<<表达式1<<表达式2<<...<<表达式n;`

★ 系统会自动判断输出数据的格式

```
#include <iostream>
using namespace std;
int main()
{
    char ch = 65;
    cout << ch << endl;
    return 0;
}
```

A

```
#include <iostream>
using namespace std;
int main()
{
    int ch = 65;
    cout << ch << endl;
    return 0;
}
```

65

§ 3. 程序设计初步

3. 4. C++的输入与输出

3. 4. 3. 输入流的基本操作

格式: `cin >> 变量1 >>变量2 >> ... >>变量n;`

- ★ 键盘输入的数据存储在缓冲区中, 不是立即被提取, 要等到缓冲区满 (不同系统大小不同) 或碰到回车符才进行提取
- ★ 默认的输入设备是键盘 (可更改, 称**输入重定向**)
- ★ 一行输入内容可分为若干行, 或者若干语句

P. 48-49例

<code>cin >> a >> b >> c >> d;</code>			
<code>cin >> a</code>	1个语句分4行 前3行无分号	<code>cin >> a;</code>	4个语句 每行有分号
<code>>> b</code>		<code>cin >> b;</code>	
<code>>> c</code>		<code>cin >> c;</code>	
<code>>> d;</code>		<code>cin >> d;</code>	

- ★ 一个提取运算符只能输入一个值

例: `int a, b, c;` 希望键盘输入3个整数, 则:

`cin >> a >> b >> c;` (正确)

`cin >> a, b, c;` (VS2017编译出错, b, c未初始化; 其他编译器可执行, 观察bc的值)

- ★ 提取运算符后必须跟变量名, 不能是常量/表达式等

例: `int a=1, b=1, c=1;`

`cin >> a+10;` (编译时语法错)

`cin >> (a, b, c);` (编译正确, 运行后假设输入10 20 30, 发现仅c得值) 为什么?

§ 3. 程序设计初步

3. 4. C++的输入与输出

3. 4. 3. 输入流的基本操作

格式: `cin >> 变量1 >>变量2 >> ... >>变量n;`

★ 输入终止条件为回车、空格、非法输入

```
#include <iostream>
using namespace std;
int main()
{
    short k;
    cin >> k;
    cout << k << endl;
    return 0;
}
```

运行5次

输入: 123✓

输入: 123 456✓

输入: 123m✓

输入: m✓

输入: ✓

?

§ 3. 程序设计初步

3. 4. C++的输入与输出

3. 4. 3. 输入流的基本操作

格式: cin >> 变量1 >>变量2 >> ... >>变量n;

★ 系统会自动判断输入数据, 若超过变量范围则错误

```
#include <iostream>
using namespace std;
int main()
{
    char c1, c2;
    int a;
    float b;
    cin >> c1 >> c2 >> a >> b;
    cout << c1 << ' ' << c2 << ' ' << a << ' ' << b << endl;
    return 0;
}
```

P. 49 例

输入: 1234~56.78

输出:

输入: 1~2~34~56.78

输出:

```
#include <iostream>
using namespace std;
int main()
{
    short k;
    k = 54321; //k=70000;
    cout << "k=" << k << endl;
    return 0;
}
```

k=-11215 (warning)

54321-65536

k=4464 (warning)

70000-65536

注意: 输入时给出超范围值
赋值时给出超范围值
情况不相同!!!

```
#include <iostream>
using namespace std;
int main()
{
    short k;
    cin >> k;
    cout << k << endl;
    return 0;
}
```

输入: 12345✓
输入: -123✓
输入: 54321✓
输入: 70000✓

?

```
#include <iostream>
using namespace std;
int main()
{
    unsigned short k;
    cin >> k;
    cout << k << endl;
    return 0;
}
```

输入: 12345✓
输入: 54321✓
输入: -123✓
输入: 70000✓

?

§ 3. 程序设计初步

3. 4. C++的输入与输出

3. 4. 3. 输入流的基本操作

格式: `cin >> 变量1 >>变量2 >> ... >>变量n;`

★ 字符型变量只能输入图形字符(33-126)，不能以转义符方式输入
(单双引号、转义符全部当作单字符)

```
#include <iostream>
using namespace std;
int main()
{
    char ch;
    cin >> ch;
    cout << (int)ch <<endl;
    return 0;
}
```

输入: abc

输出:

输入: \n

输出:

输入: 'a'

输出:

★ 浮点数输入时，可以是十进制数或指数形式，只取有效位数(4舍5入)

```
#include <iostream>
using namespace std;
int main()
{
    float f;
    cin >> f;
    cout << f <<endl;
    return 0;
}
```

输入: 123.456789

输出:

输入: 12e3

输出:

★ `cin`不能跟`endl`，否则编译错

特别提示: 有时候一大批错误指向系统文件，原因并不是系统没装好/感染病毒/被破坏…

§ 3. 程序设计初步

3. 4. C++的输入与输出

3. 4. 4. 在输入输出流中使用格式化控制符

C++缺省输入/输出格式是默认格式，为满足一些特殊要求，需要对数据进行格式化

P. 50 表3. 1

★ 第5项

`setprecision` 设置浮点数的精度为n位。在以一般十进制小数形式输出时，**n代表有效数字**。
在以`fixed`(固定小数位数)和`scientific`(指数)形式输出时，**n为小数位数**

★ 倒数第4项

`setiosflags(ios::skipws)` 忽略前导的空格

● 对`cin`有效，且缺省状态下有效

★ 加`#include <iomanip>`，否则编译错

★ 浮点数的固定小数及科学计数法输出**互斥**，若同时使用，则必须先`resetiosflag`第1个，第2个才生效

例：先使用`setiosflags(ios::fixed)`输出，完成后必须`resetiosflags(ios::fixed)`，再使用`setiosflags(ios::scientific)`才生效，否则无法科学计数法输出，**反之亦然**

§ 3. 程序设计初步

3. 4. C++的输入与输出

3. 4. 4. 在输入输出流中使用格式化控制符

//由P. 50双精度例子而来的完整程序

```
#include <iostream>
```

```
#include <iomanip> //格式化输出必须包含此头文件
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    double a=123.456789012345;
```

```
    cout << a << endl;
```

```
    cout << setprecision(9) << a << endl;
```

```
    cout << setprecision(6);
```

```
    cout << setiosflags(ios::fixed) << a << endl;
```

```
    cout << setiosflags(ios::fixed) << setprecision(8) << a << endl;
```

```
    cout << resetiosflags(ios::fixed); //使ios::fixed无效(必须有)
```

```
    cout << setiosflags(ios::scientific) << setprecision(6) << a << endl;
```

```
    cout << setiosflags(ios::scientific) << setprecision(4) << a << endl;
```

```
    return 0;
```

```
}
```

(4) 缺

(6) 若不加，则宽度为8

1、每行最后加endl

2、(5)fixed和(6)scientific间加resetiosflag(ios::fixed)
否则最后两行无效

3、去除 resetiosflag(ios::fixed) 后，观察不同编译器下
最后两行的输出有何不同

123.457 默认有效位数6

123.456789 有效位数9位

恢复默认格式 6位有效数字

123.456789 小数点后6位

123.45678901 小数点后8位

去掉ios::fixed设置

1.234568e+02 小数点后6位

1.2346e+00 小数点后4位

§ 3. 程序设计初步

3. 4. C++的输入与输出

3. 4. 4. 在输入输出流中使用格式化控制符

//由P. 51整数例子而来的完整程序

```
#include <iostream>
```

```
#include <iomanip> //格式化输出必须包含此头文件
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    int b=123456;
```

```
    cout << b << endl;
```

```
    cout << hex << b << endl;
```

```
    cout << setiosflags(ios::uppercase) << b << endl;
```

```
    cout << dec; // (3) (4) 间补此句, 设为10进制, 否则仍为16进制
```

```
    cout << setw(10) << b << ',' << b << endl;
```

```
    cout << setfill('*') << setw(10) << b << endl;
```

```
    cout << setfill('*') << setw(10) << setiosflags(ios::left) << b << endl;
```

```
    cout << setiosflags(ios::showpos) << b << endl;
```

```
    return 0;
```

```
}
```

1、每行最后加endl

2、补1句恢复10进制的语句

3、加一句输出指定宽度, 左对齐的例子

cout只输出最原始的数据, 其它内容都需要自行加入
cout << "0x" << hex << b << endl;

123456

1e240

1E240

setw只对
1个有效

123456, 123456

****123456 总10, 右对齐填*

123456**** 总10, 左对齐填*

+123456 showpos显示+

由本例可知, 格式控制有两种情况:

- 设置后始终有效, 直到再次设置 (hex等)

- 设置仅一次有效 (setw等)

具体的可自行尝试

§ 3. 程序设计初步

3. 4. C++的输入与输出

3. 4. 4. 在输入输出流中使用格式化控制符

★ 思考:

(1) 指定的输出宽度比实际宽度小时, 如何处理?

例: `cout << setw(3) << 123456 << endl;`

`cout << setw(3) << "hello" << endl;`

(2) 负数的符号是否计入指定输出的总宽度?

例: `cout << setw(15) << -123.45 << endl;` //前面有几个空格?

`cout << setw(15) << setiosflags(ios::left) << -123.45 << '*' << endl;`
// *前几个空格?

§ 3. 程序设计初步

3. 4. C++的输入与输出

3. 4. 4. 在输入输出流中使用格式化控制符

★ 倒数第4项:

setiosflags(ios::skipws) 忽略前导的空格

- 对cin有效
- 在缺省情况下是有效的(缺省不用设)
- 前导空格就是指输入过程中起分隔作用的空格

```
#include <iostream>
using namespace std;
int main()
{   int a, b;

    cin >> a >> b;
    cout << a << endl;
    cout << b << endl;

    return 0;
}
```

假设键盘输入为: 12 34

则: a=12 b=34

(忽略前导空格, 则输入中的空格被忽略, 34赋给b)

```
#include <iostream>
using namespace std;
int main()
{   int a, b;
    cin.unsetf(ios::skipws); //使skipws失效
    cin >> a >> b;
    cout << a << endl;
    cout << b << endl;

    return 0;
}
```

假设键盘输入为: 12 34

则: a=12 b=不确定值

(不忽略前导空格, 则空格被认为是非法输入, b未得值)

§ 3. 程序设计初步

3. 4. C++的输入与输出

3. 4. 5. 字符的输入和输出

3. 4. 5. 1. 字符输出函数putchar

形式: putchar(字符变量/常量)

功能: 输出一个字符

```
char a='A';
```

```
putchar(a);
```

```
putchar('A');
```

```
putchar('\x41');
```

```
putchar('\101');
```

★ 加#include <cstdio>或#include <stdio.h>(VS2017不需要, 但其它编译器需要)

§ 3. 程序设计初步

3. 4. C++的输入与输出

3. 4. 5. 字符的输入和输出

3. 4. 5. 2. 字符输入函数getchar

形式: `getchar()`

功能: 输入一个字符(给指定的变量)

★ 加`#include <cstdio>`或`#include <stdio.h>`(VS2017不需要, 但其它编译器需要)

★ 返回值是`int`型, 是输入字符的ASCII码, 可赋值给字符型/整型变量

```
#include <iostream>
#include <cstdio>
using namespace std;
int main()
{
    char ch;
    ch = getchar();
    cout << ch << endl;
    return 0;
}
```

假设键盘输入: a
则输出 : a

```
#include <iostream>
#include <cstdio>
using namespace std;
int main()
{
    char ch;
    cout << (ch = getchar()) << endl;
    return 0;
}
```

写测试程序验证一下问题:

问题1: 如何证明`getchar()`的返回值是`int`而不是`char`? 是否有多种方法?

问题2: 在不允许定义`char`型变量的前提下, 如何使`getchar()`的输出为字符

§ 3. 程序设计初步

3. 4. C++的输入与输出

3. 4. 5. 字符的输入和输出

3. 4. 5. 2. 字符输入函数getchar

形式: `getchar()`

功能: 输入一个字符(给指定的变量)

★ 输入时有回显, 输入后需按回车结束输入(若直接按回车则得到回车的ASCII码)

★ 可以输入空格, 回车等cin无法处理的非图形字符, 但仍不能处理转义符

```
#include <iostream>
#include <cstdio>
using namespace std;
int main()
{
    cout << getchar() << endl;
    return 0;
}
```

输入: 

输出: 32

输入: 

输出: 10

输入: `\n`

输出: 92

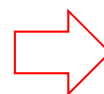
```
#include <iostream>
#include <cstdio>
using namespace std;
int main()
{
    cout << "--Step1--" << endl;
    cout << getchar() << endl;
    cout << "--Step2--" << endl;
    cout << getchar() << endl;
    cout << "--Step3--" << endl;
    cout << getchar() << endl;
    cout << "--Step4--" << endl;
    cout << getchar() << endl;
    return 0;
}
```

★ 调试程序时, 可以用`getchar()`来延迟结束

(再次强调, 本课程的作业严禁在程序最后用

`getchar()` / `system("pause")` / `cin>>a` 等来暂停)

★ `cin/getchar` 等每次仅从输入缓冲区中取需要的字节, 多余的字节仍保留在输入缓冲区中供下次读取



按下面方式执行3次, 观察运行现象及结果

1. 每次输入一个回车
2. 每次输入一个字母并按回车
3. 第一次即输入4个以上字母并按回车

§ 3. 程序设计初步

3. 4. C++的输入与输出

3. 4. 5. 字符的输入和输出

3. 4. 5. 3. 字符输入函数_getch与_getche

```
#include<iostream>
#include<conio.h> // _getch()/_getche() 用到的头文件
using namespace std;
int main()
{
    char ch;
    ch = _getch(); // 换成为_getche()/getchar() 对比
    cout << (int)ch << endl;

    return 0;
} // 注意：测试时不能是中文输入法
```

★ 几个字符输入函数的差别

getchar : 有回显，不立即生效，需要回车键

_getche : 有回显，不需要回车键

_getch : 无回显，不需要回车键

★ 在CodeBlocks和Dev C++中

getch() ⇔ _getch()

getche() ⇔ _getche()

★ Linux下支持getch()/getche()需要额外的curses库，不再讨论

§ 3. 程序设计初步

3. 4. C++的输入与输出

3. 4. 6. C语言的格式化输入与输出函数

★ 格式化输出: printf

★ 格式化输入: scanf

下发相关资料并参考其它书籍，结合作业进行自学

要求: 熟练使用C++的cin/cout

能看懂C的printf/scanf

§ 3. 程序设计初步

3.5. 编写顺序结构的程序

P.54-55 例3.5

```
#include <iostream>
#include <cmath>
using namespace std;

int main()
{   float a, b, c, x1, x2;
    cin >> a >> b >> c;
    x1 = (-b + sqrt(b*b-4*a*c) )/(2*a);
    x2 = (-b - sqrt(b*b-4*a*c) )/(2*a);
    cout << "x1=" << x1 << endl;
    cout << "x2=" << x2 << endl;
    return 0;
}
```

- 1、sqrt是系统提供的开根号函数，需要包含 <cmath>
- 2、 b^2 的表示方法 $b*b$
- 3、 $4ac$ 的表示方法 $4*a*c$
- 4、 $2*a$ 必须加()
- 5、在 $b^2-4ac<0$ 的情况下，出错
- 6、程序执行时无任何提示即等待输入

§ 3. 程序设计初步

3.5. 编写顺序结构的程序

例：从键盘输入一个大写字母，要求改为小写字母输出

```
#include <iostream>
using namespace std;
int main()
{
    char c1, c2;
    cin >> c1;
    c2 = c1 + 32;
    cout << c1 << ' ' << c2 << endl;

    return 0;
}
```

The diagram illustrates the execution flow of the C++ program. Arrows indicate the sequence of operations: from the input statement to the character conversion, then to the output statement, and finally to the return statement. Specific code snippets are highlighted in red boxes to emphasize key operations.

```
graph TD
    A[cin >> c1;] --> B[c2 = c1 + 'a' - 'A';]
    B --> C[cout << c1 << ' ' << c2 << endl;]
    C --> D[return 0;]
```

Highlighted code snippets:

- `c1=getchar();`
- `scanf("%c", &c1);`
- `c2 = c1 + 'a' - 'A';`
- `printf("%c %c", c1, c2);`

- 1、程序执行时无任何提示即等待输入
- 2、若输入的不是大写字母出错

§ 3. 程序设计初步

3.6. 关系运算和逻辑运算

3.6.1. 关系运算和关系表达式

3.6.1.1. 关系运算

含义：将两个值进行比较，取值为“真”（用1表示）

“假”（用0表示）

3.6.1.2. 关系运算符

种类：P. 56

< <= > >= 优先级相同（高 7级）

== != 优先级相同（低 8级）

优先级和结合性：P. 465 附录B

3.6.1.3. 关系表达式

含义：用关系运算符将两个表达式(算术、逻辑、赋值、关系)连接起来，称为关系表达式

$10+20 > 30$

$(a=20) > (b=30)$

关系表达式的值：

“真” - 1

“假” - 0

§ 3. 程序设计初步

3.6. 关系运算和逻辑运算

3.6.1. 关系运算和关系表达式

3.6.1.3. 关系表达式

关系表达式的值:

“真” - 1

“假” - 0

```
a=1 b=2
c=a>b    c=0
c=a<b    c=1
```

掌握用程序验证的方法

```
#include <iostream>
using namespace std;
int main()
{   int a=1, b=2, c;
    c = a>b;
    cout << c << endl;
    c = a<b;
    cout << c << endl;
    return 0;
}
```

```
a=1 b=2 c=3
d=a>b>c    d=0
d=a<b<c    d=1
d=b>a<c    d=1
```

```
a=3 b=2 c=1
d=a>b>c    d=1    d=0
d=a<b<c    d=0    d=1
d=b>a<c    d=0    d=1
```

- (1) $a>b>c$ 正确的求解方式是什么?
- (2) 一定要搞明白, 为什么结果不是蓝色而是红色的!!!

★ 关系表达式的值可以作为整型参与运算

```
int a, b, c;
cin >> a >> b;
```

$c = a > b;$ 赋值表达式, 值为0/1

$10 + (a <= b) * 2$ 算术表达式, 值为10/12

§ 3. 程序设计初步

3.6. 关系运算和逻辑运算

3.6.1. 关系运算和关系表达式

3.6.1.3. 关系表达式

关系表达式的值:

★ 关系表达式的值可以作为整型参与运算

★ 实数参与关系运算时要考虑误差

```
#include <iostream>
#include <cmath> //VS2017可不加
using namespace std;
int main()
{
    int a=1;
    cout << (a==1) << endl;      1

    float b=1.1f;
    cout << (b==1.1) << endl;    0

    double c=1.1;
    cout << (c==1.1) << endl;    1

    return 0;
}
```

结论:

- 用==判断实型数是否相等, 某些情况下可能与预期结果不符合, 因此**禁用**
- fabs()函数是通用的保证实型数误差方法, 但是使用时不要超过有效位数限定

```
#include <iostream>
#include <cmath> //VS2017可不加
using namespace std;
int main()
{
    float b=1.1f;
    cout << (b==1.1) << endl;      0
    cout << (fabs(b-1.1)<1e-6) << endl; 1

    float c=1.0f;
    cout << (c==1.0) << endl;      1
    cout << (fabs(c-1.0)<1e-6) << endl; 1

    return 0;
}
```

- (1) fabs是求绝对值的系统函数
- (2) cout时为什么要加红色的括号?

```
#include <iostream>
#include <cmath> //VS2017可不加
using namespace std;
int main()
{
    double f1=123.456789012345678;
    double f2=123.456789123456789;
    cout << (f1==f2) << endl;      0
    cout << (fabs(f1-f2)<1e-6) << endl; 1
    cout << (fabs(f1-f2)<1e-7) << endl; 0

    float g1=123.456789012345678;
    float g2=123.456789123456789;
    cout << (g1==g2) << endl;      1
    cout << (fabs(g1-g2)<1e-6) << endl; 1
    cout << (fabs(g1-g2)<1e-7) << endl; 1

    return 0;
} //warning
```

有warning
红色部分已超过float的有效数字

- 1. 为什么不等的数判断==返回1?
- 2. 为什么小数点后第7位不同但判断1e-7返回1?

§ 3. 程序设计初步

3.6. 关系运算和逻辑运算

3.6.1. 关系运算和关系表达式

3.6.1.3. 关系表达式

关系表达式的值：

★ 关系表达式的值可以作为整型参与运算

★ 实数参与关系运算时要考虑误差

★ 注意=和==的区别!!!

```
int a;
```

```
a==10;  关系表达式，值为0/1
```

```
a=10;   赋值表达式，值为10
```

§ 3. 程序设计初步

3.6. 关系运算和逻辑运算

3.6.2. 逻辑常量和逻辑变量

逻辑常量: true / false

逻辑变量: bool 变量名

定义后赋值: 定义时赋初值:

bool f; bool f=false;

f=true;

★ 在内存中占1个字节, 表示为整型值, 取值只有0/1 (true=1/false=0) sizeof(bool) => 1

★ cin时只能输入0/1, 输出时按整型量进行处理

```
#include <iostream>
using namespace std;
int main()
{
    bool k;
    cin >> k;
    cout << k << ' ' << (int)k << endl;
    return 0;
}
```

输入:	输出:
0	0 0
1	1 1
123	不确定值*2
true	不确定值*2
false	不确定值*2

§ 3. 程序设计初步

3.6. 关系运算和逻辑运算

3.6.2. 逻辑常量和逻辑变量

逻辑常量: true / false

逻辑变量: bool 变量名

★ 在内存中占1个字节, 表示为整型值, 取值只有0/1 (true=1/false=0) sizeof(bool) => 1

★ cin时只能输入0/1, 输出时按整型量进行处理

★ 赋值及运算时, 按“非0为真零为假”的原则进行

```
#include <iostream>
using namespace std;
int main()
{
    bool k;
    k=123; //VS2017有warning
    cout << k << ' ' << (int)k << endl;
    return 0;
}
```

输出: 1 1

```
#include <iostream>
using namespace std;
int main()
{
    bool k;
    k=0; //无warning
    cout << k << ' ' << (int)k << endl;
    return 0;
}
```

输出: 0 0

★ 可按整型值 (0/1) 参与表达式的运算

```
bool f=true;
```

```
int a=10;
```

```
a=a+f;
```

```
cout << a << endl; 11
```

§ 3. 程序设计初步

3.6. 关系运算和逻辑运算

3.6.3. 逻辑运算和逻辑表达式

3.6.3.1. 逻辑运算

含义：将多个关系表达式或逻辑量共同进行逻辑运算，取值为“真”/“假” 1/0

3.6.3.2. 逻辑运算符

种类： && A&&B 均为真，取值为真 （12级）

P. 58 || A||B 有一个为真，值为真(13级)

 ! !A 取反(单目运算符) （3级）

优先级与结合性：P. 465 附录B

3.6.3.3. 逻辑表达式

含义：将多个表达式或逻辑量用逻辑运算符连接起来

逻辑表达式的值：

★ 取值

真 1

假 0

★ 表达式参与运算时

非0 真

0 假

a	b	!a	!b	a&&b	a b
真	真	假	假	真	真
真	假	假	真	假	真
假	真	真	假	假	真
假	假	真	真	假	假

a	b	!a	!b	a&&b	a b
非0	非0	0	0	1	1
非0	0	0	1	0	1
0	非0	1	0	0	1
0	0	1	1	0	0

逻辑运算的真值表：
P. 58 表3.2

逻辑运算的真值表：
P. 60 表3.3(有错)

§ 3. 程序设计初步

3.6. 关系运算和逻辑运算

3.6.3. 逻辑运算和逻辑表达式

3.6.3.3. 逻辑表达式

含义：将多个表达式或逻辑量用逻辑运算符连接起来

例：a=4 b=5

a && b =1

a || b =1

!a || b =1

!a && b =0

例：4 && 0 || 2 = 1

例：5>3 && 2 || 8<4 - !0 = 1

自行给出求值顺序

§ 3. 程序设计初步

3.6. 关系运算和逻辑运算

3.6.3. 逻辑运算和逻辑表达式

3.6.3.3. 逻辑表达式

含义：将多个表达式或逻辑量用逻辑运算符连接起来

逻辑表达式的值：

★ 取值

★ 表达式参与运算时

★ 仅当必须执行下一个逻辑运算符才能求出解时，才执行该运算符，否则不执行(短路运算)

$a \&\& b \&\& c$ 若 $a=0$ ，值必为0， b, c 不求解

$a || b || c$ 若 $a=1$ ，值必为1， b, c 不求解

★ 容易犯的错误：表示 t 在 $(70, 80]$ 之间

错误： $70 < t \leq 80$!!!!!

正确： $t > 70 \&\& t \leq 80$

★ 常见的等价表示

$a == 0 \quad \Leftrightarrow \quad !a$

$a != 0 \quad \Leftrightarrow \quad a$

```
#include <iostream>
using namespace std;
int main()
{
    int a=1, b=2, c=3, d=4, m=1, n=1;
    cout << "m=" << m << " n=" << n << endl;
    (m=a>b)&&(n=c>d); //m=0, n不再求解, 保持原值
    cout << "m=" << m << " n=" << n << endl;
    return 0;
}
```

$m=1 \quad n=1$
 $m=0 \quad n=1$

例：若某年是闰年，则符合下列两个条件之一

(1) 被4整除，不被100整除

(2) 被4整除，又被400整除

各种形式的表示：

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

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

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

$!(year \% 4) \&\& (year \% 100) \ || \ !(year \% 400)$

$!(year \% 4) \&\& year \% 100 \ || \ !(year \% 400)$

真：闰年

假：非闰年

例：若某年不是闰年，则符合下列两个条件之一

(1) 不被4整除

(2) 被100整除，不被400整除

条件1: $(year \% 4 != 0)$

条件2: $(year \% 100 == 0) \&\& (year \% 400 != 0)$

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

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

真：非闰年

假：闰年

为什么条件2的整体括号(蓝色)可以省略？

§ 3. 程序设计初步

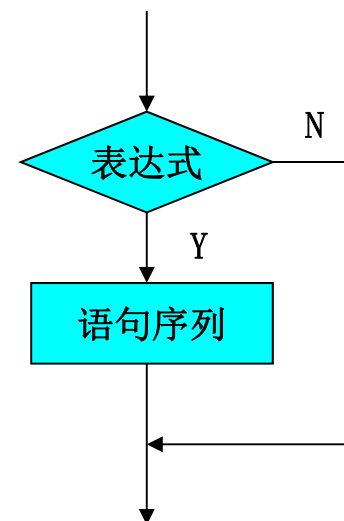
3.7. 选择结构和if语句

3.7.1. if语句的三种形式

3.7.1.1. 单支语句

```
if (表达式) {  
    语句序列;  
}
```

- ★ 当表达式为真时执行语句序列，为假则不执行
- ★ 表达式可以是任意类型，但按逻辑值求解(非0为真0为假)
- ★ 表达式后无; (表达式和表达式语句的区别)



编译错

```
if (表达式;) {  
    语句序列;  
}
```

- ★ 若语句序列中只有一个语句，{}可省

```
if (i<60) {  
    cout << "不及格" << endl;  
}  
等价于  
if (i<60)  
    cout << "不及格" << endl;
```

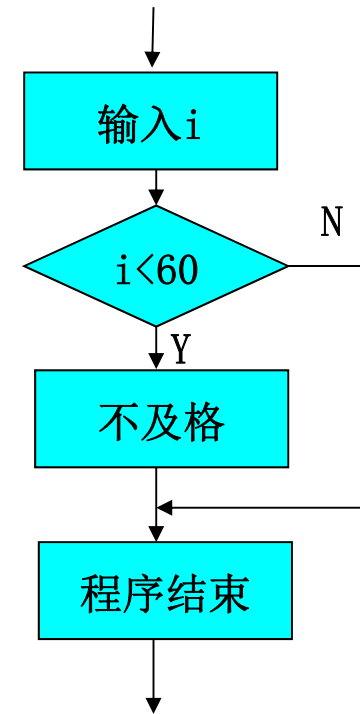
- ★ 整个单支语句可以作为一个语句来看待(复合语句)

例：输入一个成绩，若不及格，则打印提示信息

```
int main()
{
    int i;
    cout<<"请输入成绩(0-100)"<<endl;
    cin >> i;
    if (i<60) {
        cout << "不及格" << endl;
    }
    cout << "程序结束" << endl;
    return 0;
}
```

输入:37 输出:不及格
程序结束

输入:74 输出:程序结束

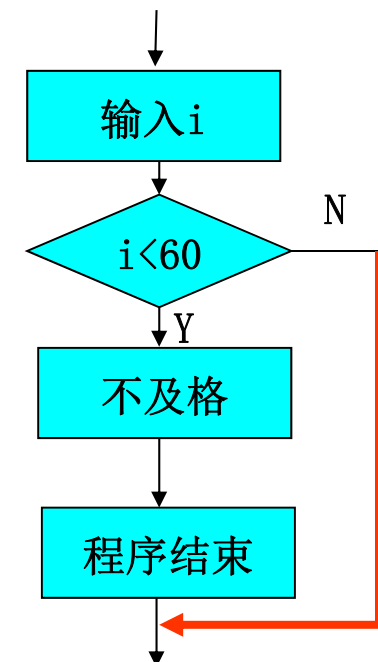


```
int main()
{
    int i;
    cout<<"请输入成绩(0-100)"<<endl;
    cin >> i;
    if (i<60) {
        cout << "不及格" << endl;
        cout << "程序结束" << endl;
    } //注意：括号位置已变!!!
    return 0;
}
```

输入:37 输出:不及格
程序结束

输入:74 输出: /

注意：括号位置对程序
正确性影响很大



§ 3. 程序设计初步

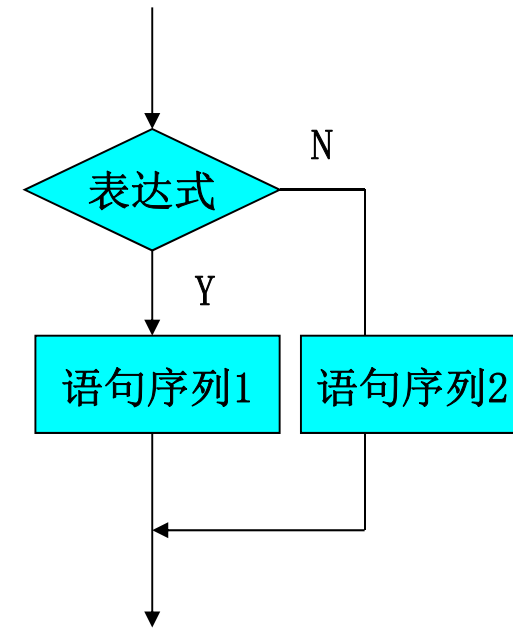
3.7. 选择结构和if语句

3.7.1. if语句的三种形式

3.7.1.1. 单支语句

3.7.1.2. 双支语句

```
if (表达式) {  
    语句序列1;  
}  
else {  
    语句序列2;  
}
```



- ★ 当表达式为真时执行语句序列1，为假则执行语句序列2
- ★ 表达式可以是任意类型，但按逻辑值求解(非0为真0为假)
- ★ 表达式后无;
- ★ 若语句序列1、2中只有一个语句，{}可省
- ★ 整个双支语句可以作为一个语句来看待，
中间不允许插入任何的其它语句

```
if (i<60) {  
    ...;  
}  
cout << "... " << endl;  
else {  
    ...;  
}
```

编译错，提示else没有对应的if

例：输入一个成绩，根据分数是否及格打印相应的提示信息

```
int main()
{
    int i;
    cout << "请输入成绩(0-100)" << endl;
    cin >> i;
    if (i<60)
        cout << "不及格" << endl;
    else
        cout << "及格" << endl;
    cout << "程序结束" << endl;
    return 0;
}
```

注意：若没有括号，则只有第1句
语句属于if-else

```
int main()
{
    int i;
    cout << "请输入成绩(0-100)" << endl;
    cin >> i;
    if (i<=60)
        cout << "不及格" << endl;
    else
        cout << "及格" << endl;
    cout << "程序结束" << endl;
    return 0;
}
```

即使能保证输入正确[0..100]，
仍有一个数据的运行结果是错误的

程序设计中的几个很重要的概念：

- 1、部分测试数据的正确性不代表程序一定是正确的，只是错误没有暴露出来而已
- 2、错误的发现可能需要相当长时间，时间不是证明没有错误的借口
- 3、程序的测试很重要，测试的目的是为了证明程序有错误，而不是为了证明程序是正确的
- 4、复杂程序无法保证完全正确，因此如何快捷方便地更正错误很重要

§ 3. 程序设计初步

3.7. 选择结构和if语句

3.7.1. if语句的三种形式

3.7.1.1. 单支语句

3.7.1.2. 双支语句

3.7.1.3. 多支语句

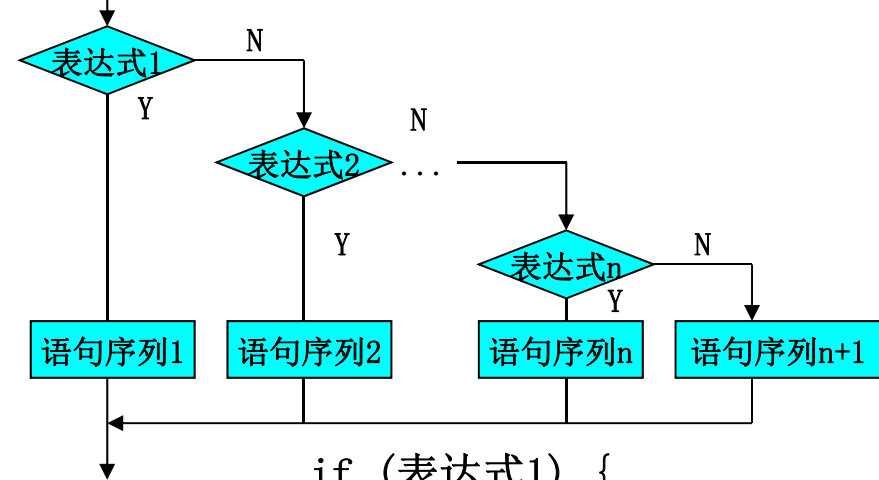
★当表达式1为真时，执行语句序列1
为假时，则判断表达式2
当表达式2为真时，执行语句序列2
为假时，则判断表达式3
.
.
当表达式n为真时，执行语句序列n
为假时，则执行语句序列n+1

★ 表达式可以是任意类型，按逻辑值求解（非0为真0为假）

★ 表达式后无；

★ 若语句序列中只有一个语句，{}可省

★ 整个多支语句可以作为一个语句来看待，
中间不允许插入任何的其它语句



```
if (表达式1) {  
    语句序列1;  
}  
else if (表达式2) {  
    语句序列2;  
}  
...  
else if (表达式n) {  
    语句序列n;  
}  
else {  
    语句序列n+1;  
}
```

例：输入一个分数，根据所处的分数段0-59, 60-69, 70-79, 80-89, 90-100分别打印“优”、“良”、“中”、“及格”、“不及格”，其它则打印“输入错误”

```
int main()
{
    int i;
    cout << "请输入成绩(0-100)" << endl;
    cin >> i;
    if (i>=90 && i<=100)
        cout << "优" << endl;
    else if (i>=80 && i<90)
        cout << "良" << endl;
    else if (i>=70 && i<80)
        cout << "中" << endl;
    else if (i>=60 && i<70)
        cout << "及格" << endl;
    else if (i>=0 && i<60)
        cout << "不及格" << endl;
    else
        cout << "输入错误" << endl;
    cout << "程序结束" << endl;
    return 0;
}
```

问题1：
能否改为 $i \leq 89$
哪个更好？

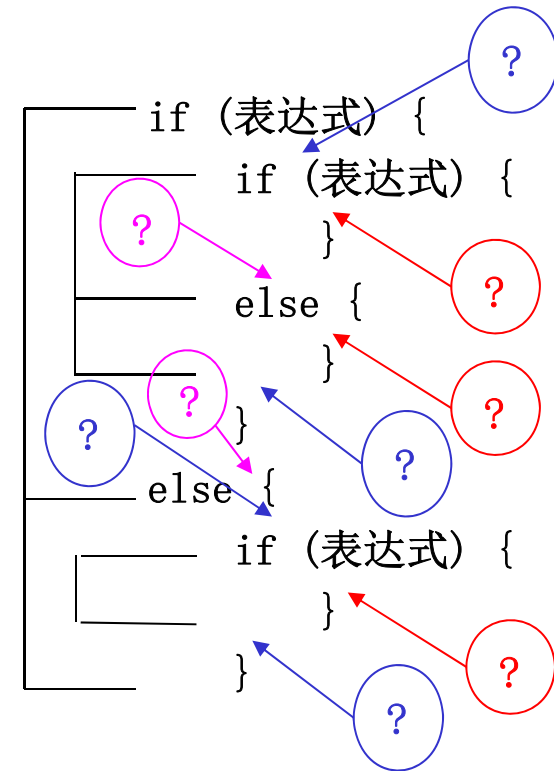
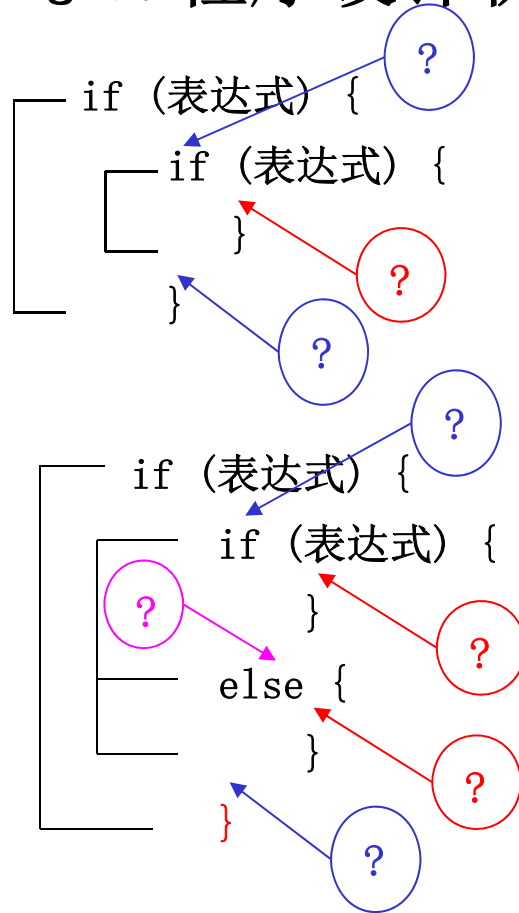
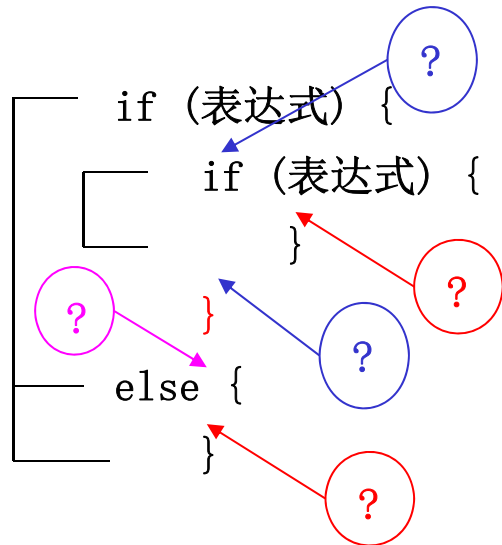
问题2：
能否改为 $i \leq 90$
运行是否正确

§ 3. 程序设计初步

3.7. 选择结构和if语句

3.7.1. if语句的三种形式

3.7.2. if语句的嵌套



各红色? 的语句在什么情况下执行
各蓝色? 的语句在什么情况下执行
各粉色? 处若有语句, 是否正确?

- ★ {} 的匹配原则: 自上而下, 忽略 { , 以 } 为准向上匹配未配对的 {
- ★ {} 的匹配可用栈理解, 遇 { 进栈, 遇 } 则栈顶 { 出栈并匹配为一对, 若到最后仍有 { 或 } 未配对则语法错
- ★ if/else 匹配原则: 以 {} 成对为基准, 将 else 与它上面最近的 if 配对
(if 不一定有 else, else 一定要有 if, 因此和 {} 的匹配方式不完全相同)

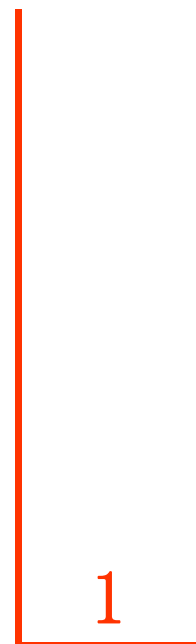
例:

```
if (表达式) { 1
  if (表达式) { 2
    } 3
  else { 4
    } 5
  } 6
else { 7
  if (表达式) { 8
    } 9
  } 10
```

初始: 空

例:

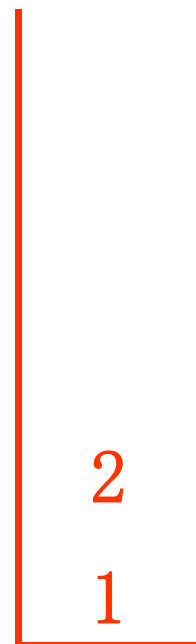
```
if (表达式) { 1
  if (表达式) { 2
    } 3
  else { 4
    } 5
  } 6
else { 7
  if (表达式) { 8
    } 9
  } 10
```



1进栈

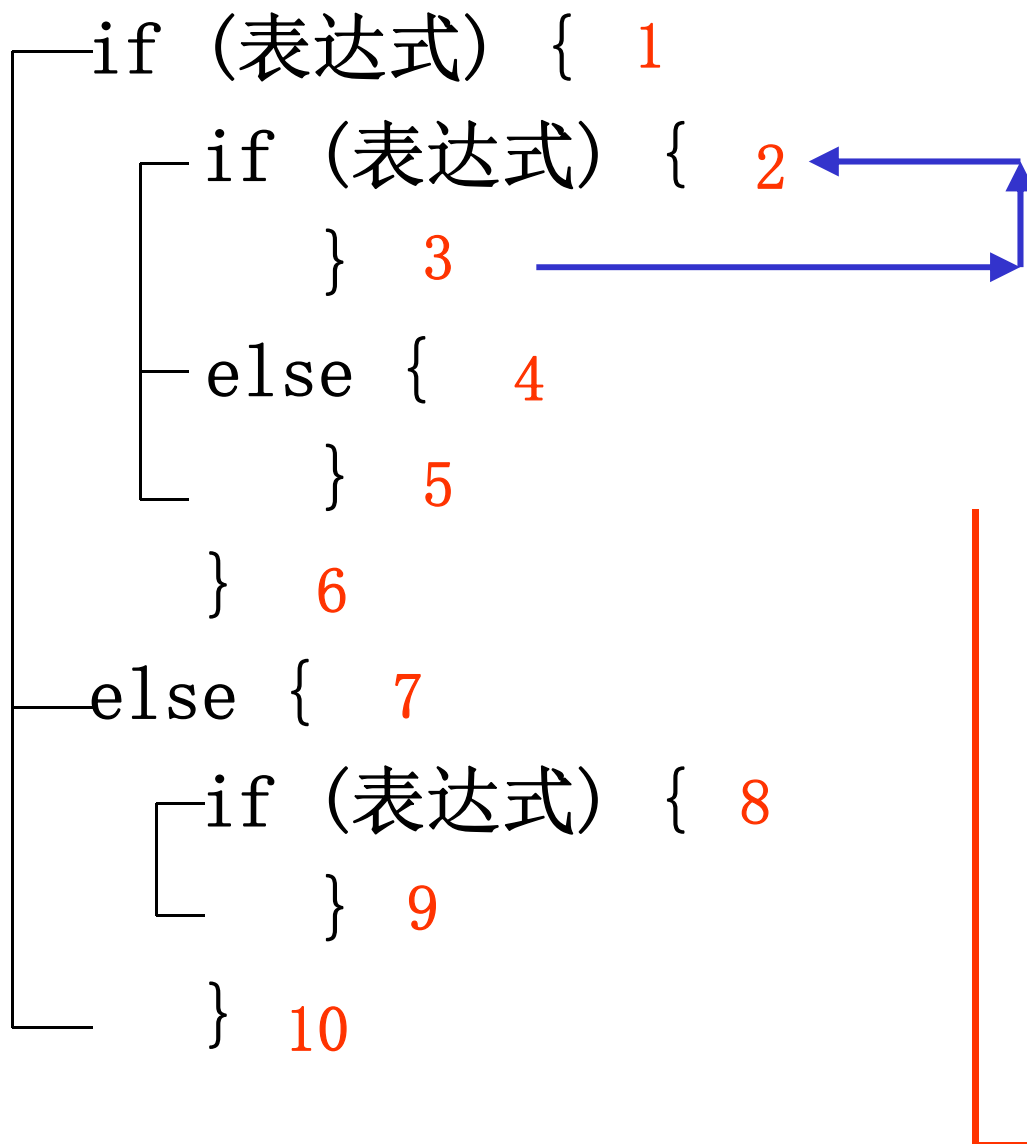
例:

```
if (表达式) { 1
  if (表达式) { 2
    } 3
  else { 4
    } 5
  } 6
else { 7
  if (表达式) { 8
    } 9
  } 10
```



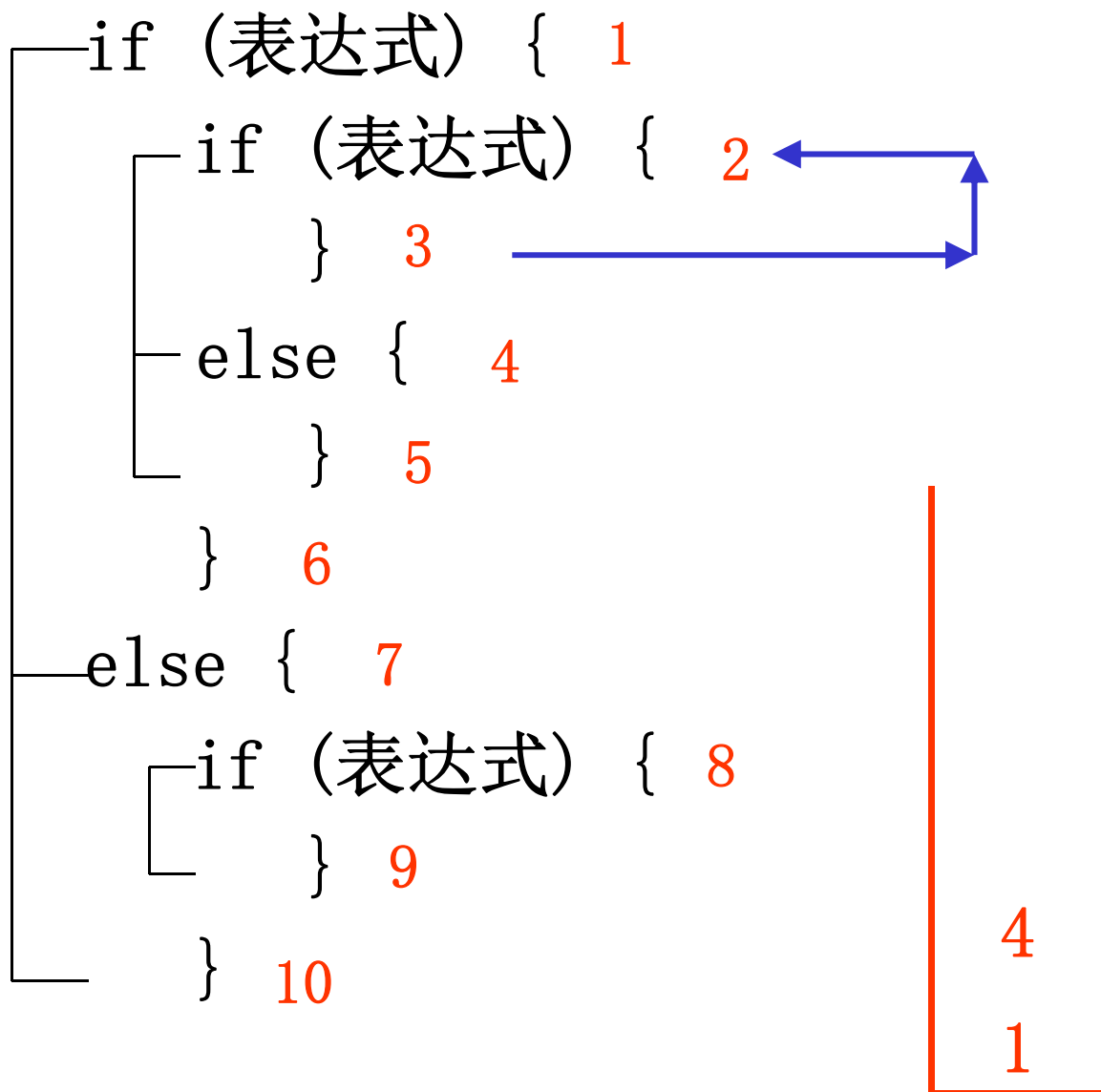
2进栈

例:



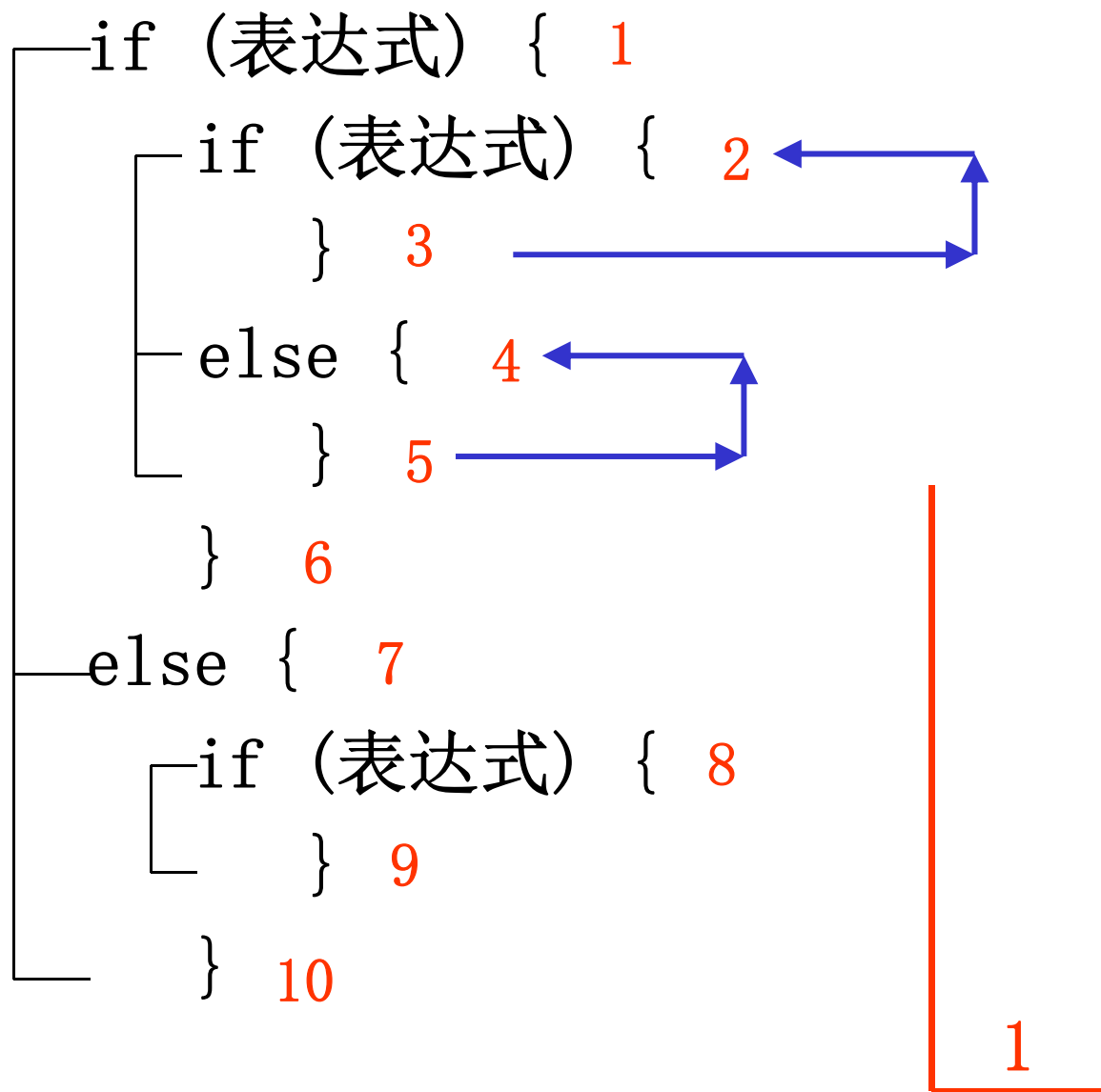
遇3, 2出, 匹配

例:



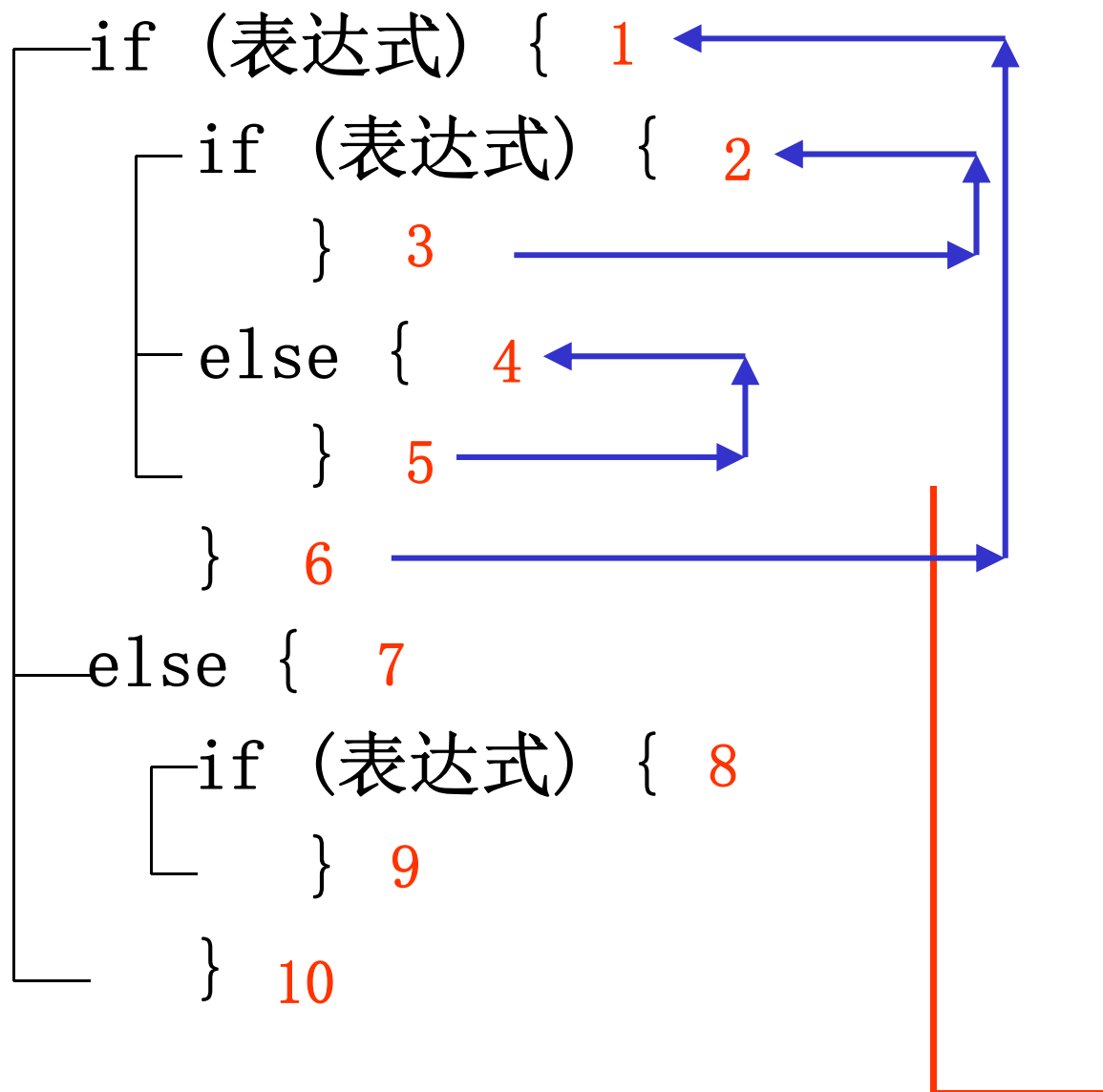
4进栈

例:



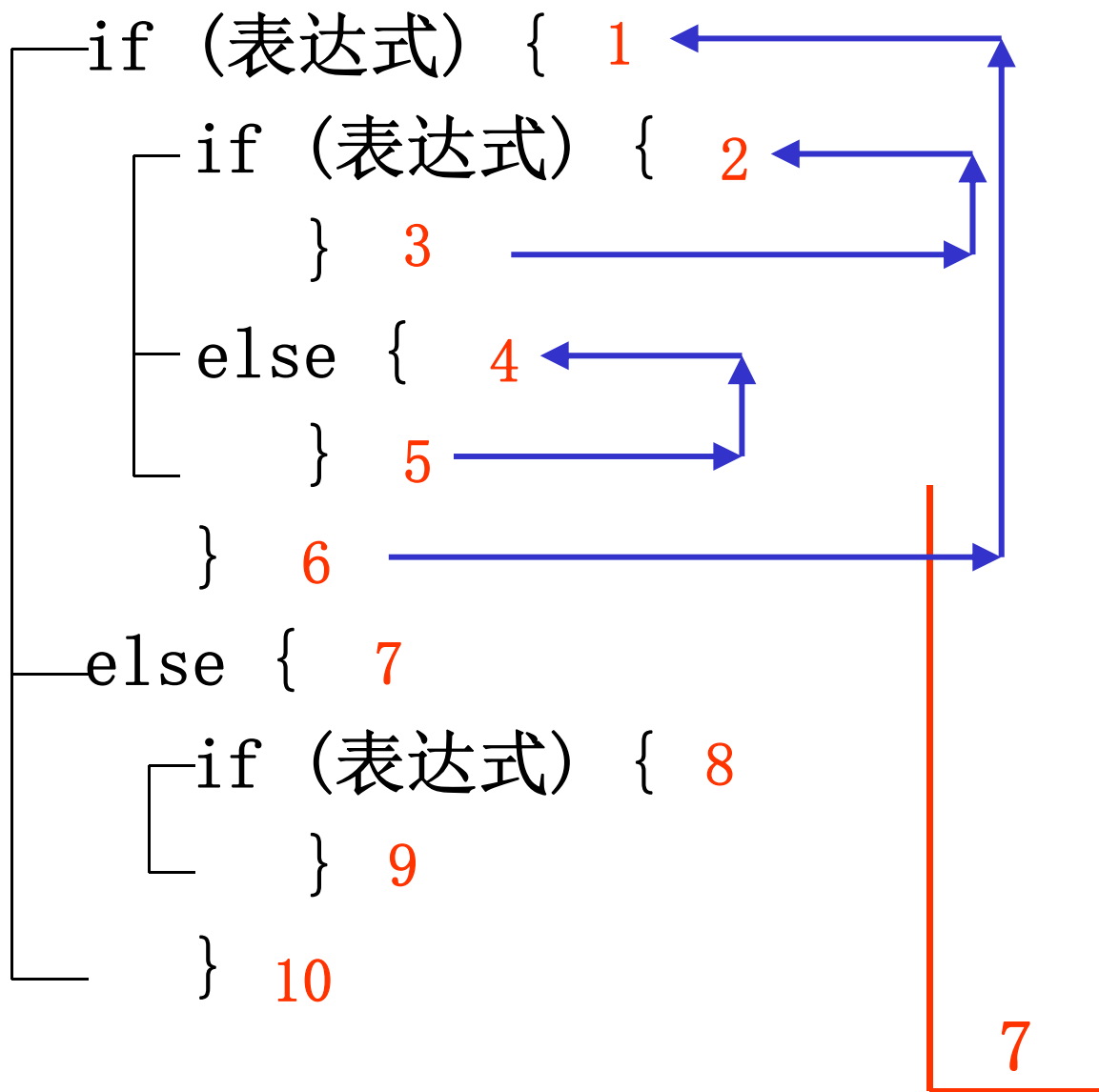
遇5, 4出, 匹配

例:



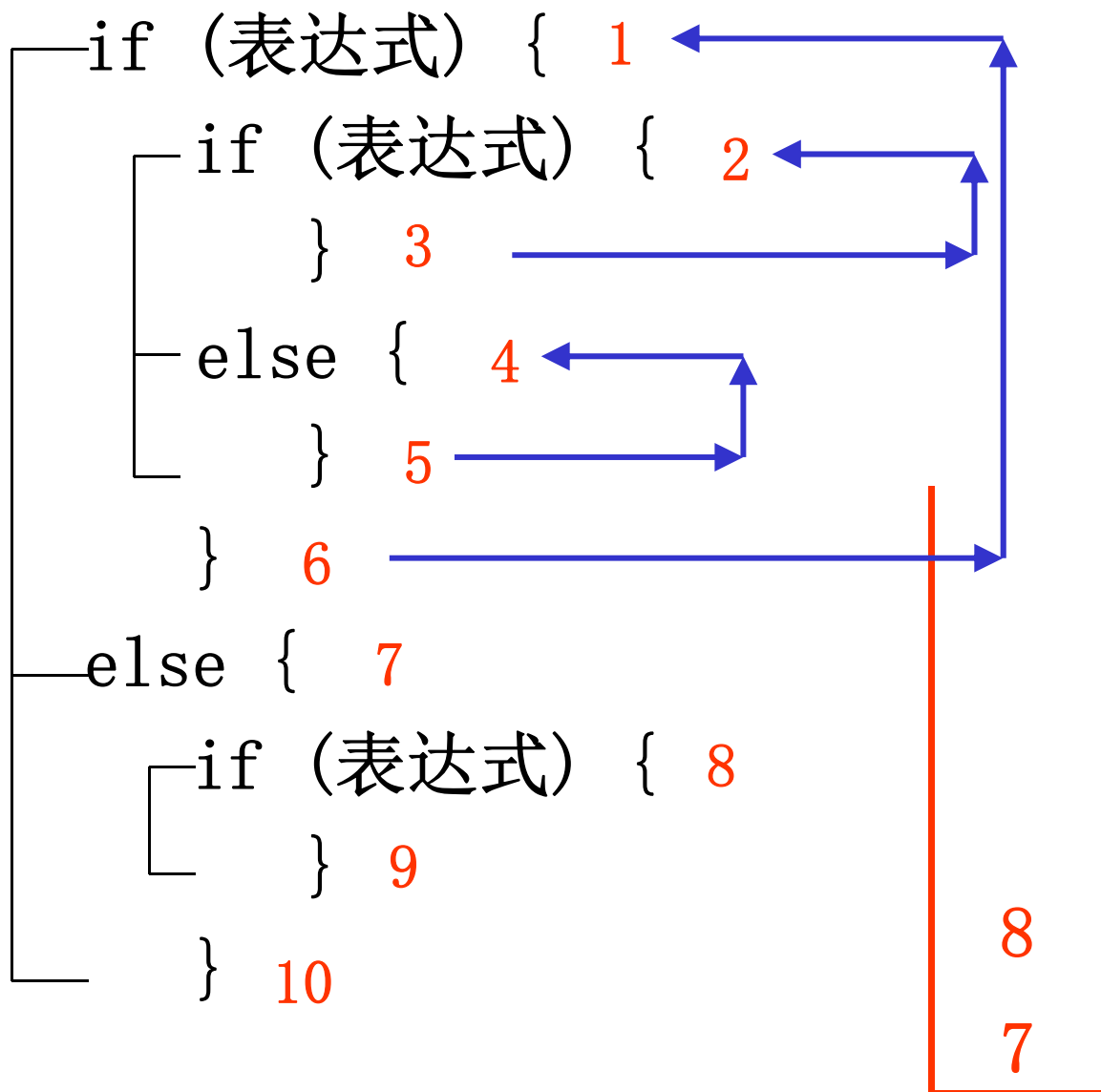
遇6, 1出, 匹配

例:



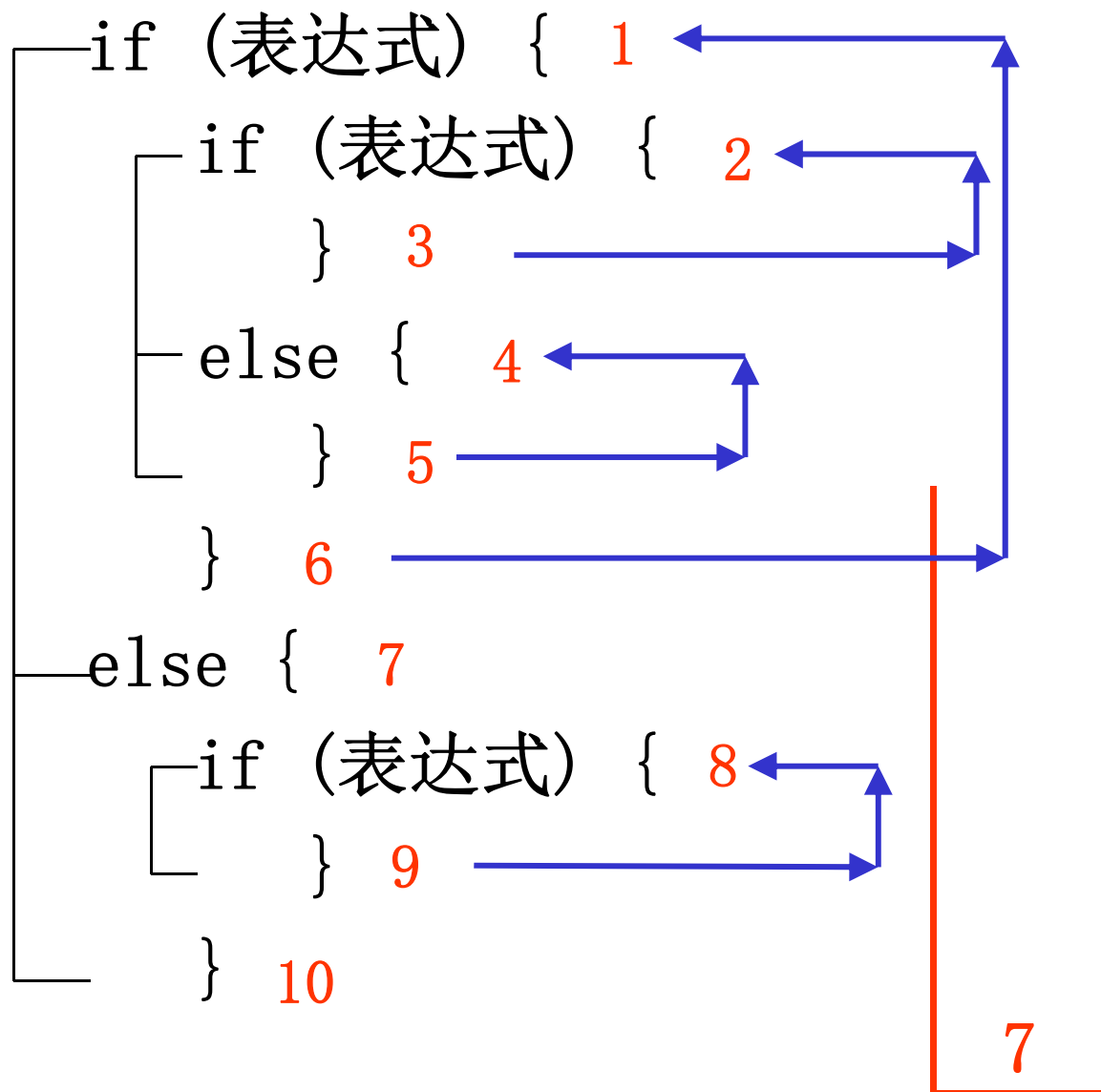
7进栈

例:



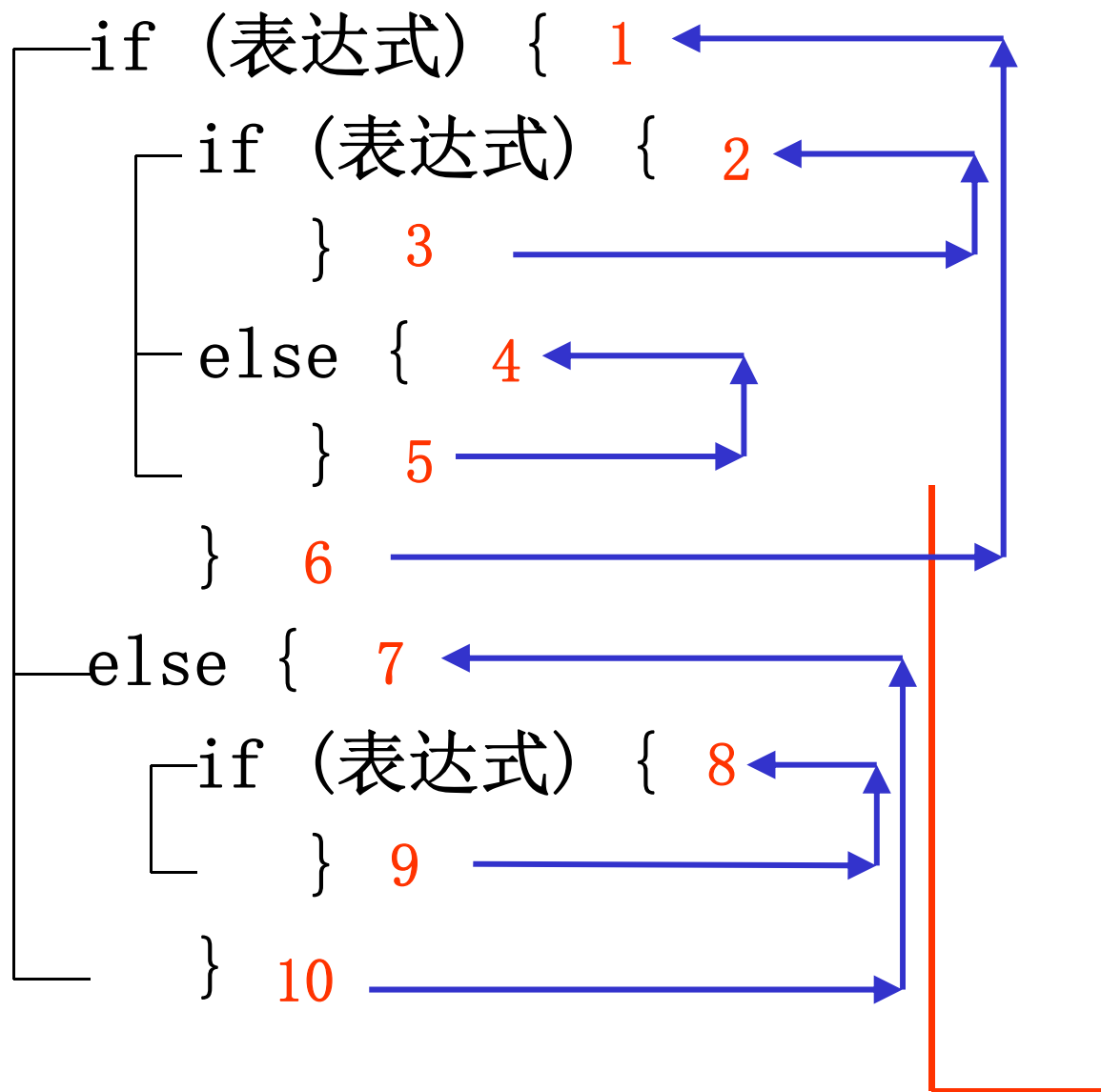
8进栈

例:



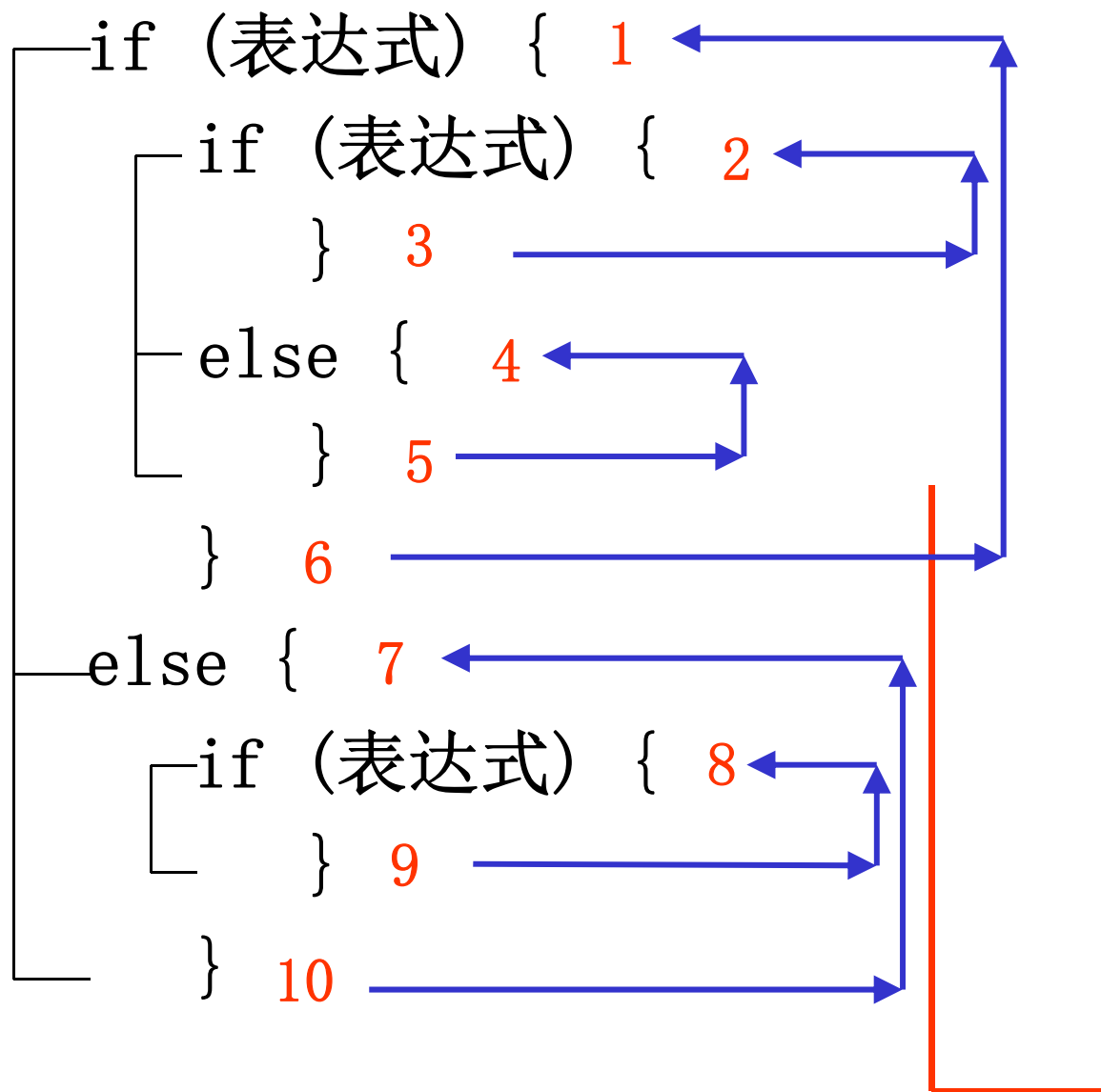
遇9, 8出, 匹配

例:



遇10, 7出, 匹配

例:



栈空, 外面也无未匹配的}
正确结束

例：输入一个分数，根据所处的分数段0-59, 60-69, 70-79, 80-89, 90-100分别打印“优”、“良”、“中”、“及格”、“不及格”，其它则打印“输入错误”

```
if (i>=80)
{
    if (i>=90)
        printf("优\n");
    else
        printf("良\n");
}
else
{
    if (i>=60)
    {
        if (i>=70)
            printf("中\n");
        else
            printf("及格\n");
    }
    else
        printf("不及格\n");
}
```

单个语句都加 {}

```
if (i>=80) {
    if (i>=90) {
        printf("优\n");
    }
    else {
        printf("良\n");
    }
}
else {
    if (i>=60) {
        if (i>=70) {
            printf("中\n");
        }
        else {
            printf("及格\n");
        }
    }
    else {
        printf("不及格\n");
    }
}
```

★ 假设输入正确

§ 3. 程序设计初步

3.7. 选择结构和if语句

3.7.3. 条件运算符和条件表达式

3.7.3.1. 引入

if - else语句中语句序列均为一个赋值语句且给同一变量赋值的，可以使用条件运算符

3.7.3.2. 形式

表达式1 ? 表达式2 : 表达式3

★ C/C++中唯一的一个三目运算符(14级)

★ 表达式1按逻辑值求解，若为真，求解表达式2并使整个条件表达式的值为表达式2的值，
否则，求解表达式3并使整个条件表达式的值为表达式3的值

★ 表达式1、2、3的类型可以不同

```
例: int a,b,max;  
cin >> a >> b;  
if (a>b)  
    max = a;  
else  
    max = b;  
max = a > b ? a : b;
```

```
例: int a,b;  
cin >> a >> b;  
if (a>b)  
    cout << "max=" << a << endl;  
else  
    cout << "max=" << b << endl;  
a > b ? cout << "max=" << a << endl :  
        cout << "max=" << b << endl;  
cout << "max=" << (a>b?a:b) << endl;  
printf("max=%d", a>b?a:b);
```

§ 3. 程序设计初步

3.7. 选择结构和if语句

3.7.4. 多分支选择结构和switch语句

3.7.4.1. 作用

替代多重if语句的嵌套，增强可读性

3.7.4.2. 形式

见右侧

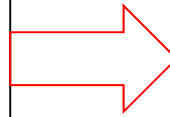
3.7.4.3. 使用

- ★ 表达式可以是任何类型，最终取值为整型即可
- ★ 当整型表达式的取值与整型常量表达式1-n中的任意一个相等时，执行对应的语句序列；否则执行default后的语句序列（**不能是实数，因为无法直接判断相等**）
- ★ 各整型常量表达式的值应各不相同，但顺序无要求
- ★ 各语句序列的最后一句应是break；否则连续执行下一case语句，最后一个可省
- ★ 语句序列不必加 {}
- ★ 多个case可以共用一组语句
- ★ 不能完全替代多重if语句的嵌套

```
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;  
}
```


例：输入一个分数，根据所处的分数段0-59, 60-69, 70-79, 80-89, 90-100分别打印“优”、“良”、“中”、“及格”、“不及格”，其它则打印“输入错误”。

```
#include <iostream>
using namespace std;
int main()
{
    int i;
    cout << "请输入成绩 (0-100) " << endl;
    cin >> i;
    if (i>=90 && i<=100)
        cout << "优" << endl;
    else if (i>=80 && i<90)
        cout << "良" << endl;
    else if (i>=70 && i<80)
        cout << "中" << endl;
    else if (i>=60 && i<70)
        cout << "及格" << endl;
    else if (i>=0 && i<60)
        cout << "不及格" << endl;
    else
        cout << "输入错误" << endl;
    cout << "程序结束" << endl;
    return 0;
}
```



```
#include <iostream>
using namespace std;
int main()
{
    int i;
    cout<<"请输入成绩 (0-100)"<<endl;
    cin >> i;
    switch(i/10) {
        case 10:
        case 9:
            cout<<"优"<<endl;
            break;
        case 8:
            cout<<"良"<<endl;
            break;
        case 7:
            cout<<"中"<<endl;
            break;
        case 6:
            cout<<"及格"<<endl;
            break;
        case 5:
        case 4:
        case 3:
        case 2:
        case 1:
        case 0:
            cout<<"不及格"<<endl;
            break;
        default:
            cout<<"输入错误"<<endl;
            break;
    }
    cout<<"程序结束"<<endl;
    return 0;
}
```

本程序在哪两个数据区间会得到错误结果？

例：输入一个分数，根据所处的分数段0-59, 60-69, 70-79, 80-89, 90-100分别打印“优”、“良”、“中”、“及格”、“不及格”，其它则打印“输入错误”。

```
#include <iostream>
using namespace std;
int main()
{
    int i;
    cout << "请输入成绩 (0-100) " << endl;
    cin >> i;
    if (i>=90 && i<=100)
        cout << "优" << endl;
    else if (i>=80 && i<90)
        cout << "良" << endl;
    else if (i>=70 && i<80)
        cout << "中" << endl;
    else if (i>=60 && i<70)
        cout << "及格" << endl;
    else if (i>=0 && i<60)
        cout << "不及格" << endl;
    else
        cout << "输入错误" << endl;
    cout << "程序结束" << endl;
    return 0;
}
```

```
#include <iostream>
using namespace std;
int main()
{
    int i;
    cout<<"请输入成绩 (0-100)"<<endl;
    cin >> i;
    if (i>=0 && i<=100) {
        switch(i/10) {
            case 10:
            case 9:
                cout<<"优"<<endl;
                break;
            case 8:
                cout<<"良"<<endl;
                break;
            case 7:
                cout<<"中"<<endl;
                break;
            case 6:
                cout<<"及格"<<endl;
                break;
            default:
                cout<<"不及格"<<endl;
                break;
        }
    }
    else
        cout << "输入错误" << endl;

    cout<<"程序结束"<<endl;
    return 0;
}
```

保证输入区间的
正确性

case 5:
case 4:
case 3:
case 2:
case 1:
case 0:

§ 3. 程序设计初步

3.7. 选择结构和if语句

3.7.4. 多分支选择结构和switch语句

3.7.4.3. 使用

★ 不能完全替代多重if语句的嵌套

例：输入一个分数，根据所处的分数段0-59, 60-69, 70-84, 85-100分别打印“优”、“良”、“及格”、“不及格”，其它则打印“输入错误”。

★ 若用 `switch(score/10)`，则部分分数无法区分

★ 若用 `switch(score)`，要写 101 个case

★ 若分数精确到小数点后，则无法用switch

//if的实现方式

```
if (score>=0 && score<60)
    cout << "不及格" << endl;
else if (score>=60 && score<70)
    cout << "及格" << endl;
else if (score>=70 && score <85)
    cout << "良" << endl;
else if (score>=85 && score<=100)
    cout << "优" << endl;
else
    cout << "输入错误" << endl;
```

§ 3. 程序设计初步

3.7. 选择结构和if语句

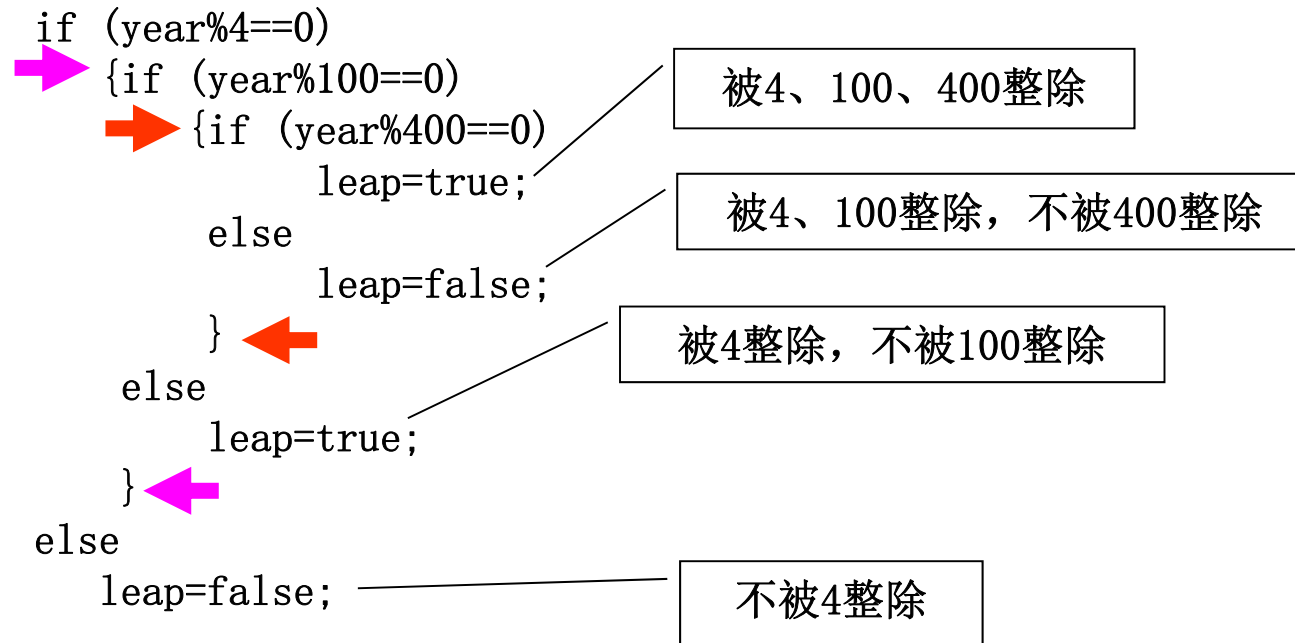
3.7.5. 编写选择结构的程序

P. 67-69

例3.8 闰年判断

例3.9 运费分段计算

例3.8: 闰年判断



§ 3. 程序设计初步

3.7. 选择结构和if语句

3.7.5. 编写选择结构的程序

P. 67-69

例3.8 闰年判断

例3.9 运费分段计算

例3.8：闰年判断

```
if (year%4==0)
    {if (year%100==0)
        {if (year%400==0)
            leap=true;
        else
            leap=false;
        }
    else
        leap=true;
    }
else
    leap=false;
```

```
if (year%4==0) {
    if (year%100==0) {
        if (year%400==0)
            leap=true;
        else
            leap=false;
    }
    else
        leap=true;
}
else
    leap=false;
```

可以省略两组括号

§ 3. 程序设计初步

3.7. 选择结构和if语句

3.7.5. 编写选择结构的程序

P. 67-69

例3.8 闰年判断

例3.9 运费分段计算

例3.8: 闰年判断 (P. 68 改写)

```
if (year%4!=0)  
    leap=false;
```

不被4整除

```
else if (year%100!=0)  
    leap=true;
```

被4整除, 不被100整除

```
else if (year%400!=0)  
    leap=false;
```

被4、100整除, 不被400整除

```
else  
    leap=true;
```

被4、100、400整除

例3.8: 闰年判断

```
if ((year%4==0 && year%100!=0) || (year%400==0))  
    leap=true;  
else  
    leap=false;
```

可简化为以下形式:

```
leap = year%4==0 && year%100!=0 || year%400==0;
```

```
if (leap)  
    cout << year << " is"  
else  
    cout << year << " is not"  
cout<<" a leap year."<<endl;
```

可简化为以下3种形式:

```
cout << year << (leap ? " is" : " is not") << " a leap year." << endl;  
printf("%d %s a leap year.\n", year, leap ? "is" : "is not");  
printf("%d is%s a leap year.\n", year, leap ? "" : " not");
```

//例3.8可改写为

```
#include <iostream>
using namespace std;
int main()
{   int year, leap;
    cout << "Please enter year:";
    cin >> year;
    leap=year%4==0 && year%100!=0 || year%400==0;
    cout << year << (leap ? " is" : " is not")
        << " a leap year." << endl;

    return 0;
}
```

cout不加endl,
这样输入接在后面



例3.9: 运费计算

```
if (s>=3000) c=12;  
else c=s/250;
```

```
switch(c)
```

```
{ case 0:d=0;break;
```

[0..250)

```
case 1:d=2;break;
```

[250..500)

```
{ case 2:
```

[500..750)

```
case 3:d=5;break;
```

[750..1000)

```
case 4:
```

[1000..1250)

```
case 5:
```

[1250..1500)

```
case 6:
```

[1500..1750)

```
case 7:d=8;break;
```

[1750..2000)

```
case 8:
```

[2000..2250)

```
case 9:
```

[2250..2500)

```
case 10:
```

[2500..2750)

```
case 11:d=10;break;
```

[2750..3000)

```
case 12:d=15;break;
```

>=3000

```
}
```

程序不能正确处理负数
[-249..-1]: d=0
<=-250 : d为未定值

//if语句的实现方法

```
if (s>=0 && s<250)
```

```
d=0;
```

```
else if (s>=250 && s<500)
```

```
d=2;
```

```
else if (s>=500 && s<1000)
```

```
d=5;
```

```
else if (s>=1000 && s<2000)
```

```
d=8;
```

```
else if (s>=2000 && s<3000)
```

```
d=10;
```

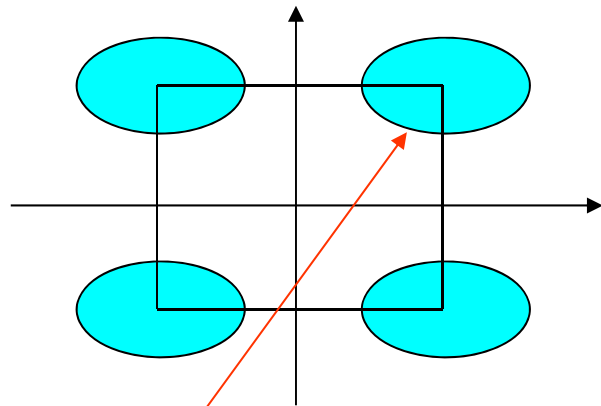
```
else if (s>=3000)
```

```
d=15;
```

```
else
```

```
cout << "输入值不合法" << endl;
```

例:四个椭圆塔, 圆心分别为(5, 5), (-5, 5), (-5, -5), (5, -5), 长半径为3, 短半径为2, 这4个塔的高度为10m, 塔以外无建筑物(高度为0), 编写程序, 输入任一点的坐标, 求该点的建筑高度



$$\frac{(x-5)^2}{3^2} + \frac{(y-5)^2}{2^2} = 1$$

$$(x-5)*(x-5)/9+(y-5)*(y-5)/4 <=1$$

```
#include <iostream>
using namespace std;
int main()
{
    float x, y; //尽量别用int
    cout << "请输入坐标: "; //不加endl, 输入在本行
    cin >> x >> y;

    if ( ((x-5)*(x-5)/9+(y-5)*(y-5)/4<=1) ||
         ((x-5)*(x-5)/9+(y+5)*(y+5)/4<=1) ||
         ((x+5)*(x+5)/9+(y+5)*(y+5)/4<=1) ||
         ((x+5)*(x+5)/9+(y-5)*(y-5)/4<=1) )
        cout << "高度为10" << endl;
    else
        cout << "高度为0" << endl;

    return 0;
}
```

例：对n个人分班，每班k(k>0)人，最后不足k人也编为一个班，问要分几个班？
键盘输入n, k的值，输出分班数(尝试使用if语句)

```
#include<iostream>
using namespace std;
int main()
{
double o, p, m, b;
int c;
cin>>o>>p;
b=o/p;
c=o/p;
if (b-c>0) m=c+1;
else m=c;
cout<<m;
return 0;
}
```

作者原意：

虽然输入类型为double，但希望键盘输入为整数
(o=n p=k)

b是浮点除法

c是整数除法

b-c>0 表示除不尽 => o不是p的整数倍

程序存在的问题：

- 1、格式
- 2、o、p变量用浮点数，不能保证输入的正确性
- 3、浮点数有误差，导致b-c>0不可信
- 4、m应该是int型

例：对n个人分班，每班k(k>0)人，最后不足k人也编为一个班，问要分几个班？
键盘输入n, k的值，输出分班数(尝试使用if语句)

```
#include <iostream>
using namespace std;
int main()
{
    int n, k;
    cin >> n >> k;
    if (n%k==0)
        cout << n/k << endl;
    else
        cout << n/k+1 << endl;

    return 0;
}
```

限制：
1、不允许用if-else
如何实现？

```
#include <iostream>
using namespace std;
int main()
{
    int n, k;
    cin >> n >> k;

    cout << (n%k == 0 ? n/k : n/k+1) << endl;
    return 0;
}
```

限制：
1、不允许用if-else
2、不允许用条件表达式
3、不允许用bool，不允许用关系、逻辑运算符
如何实现？

```
#include <iostream>
using namespace std;
int main()
{
    int n, k;
    cin >> n >> k;

    cout << (n+k-1)/k << endl;
    return 0;
}
```

限制：
1、不允许用if-else
2、不允许用条件表达式
如何实现？

```
#include <iostream>
using namespace std;
int main()
{
    int n, k;
    cin >> n >> k;

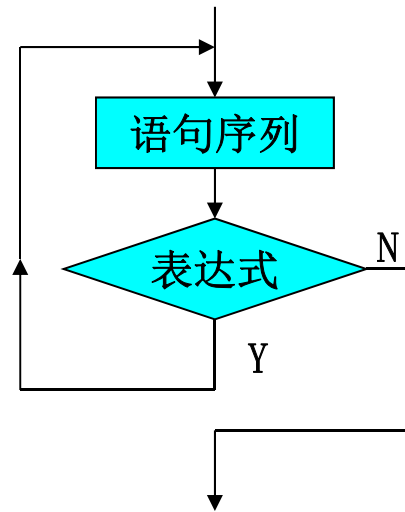
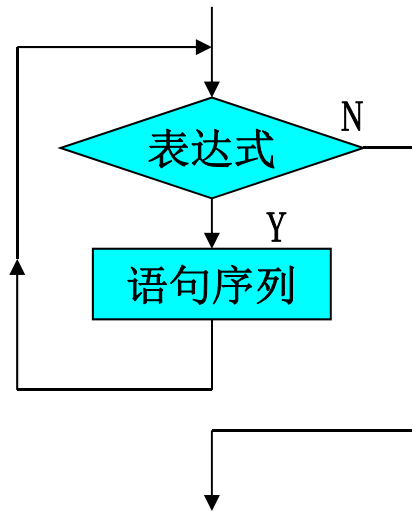
    cout << n/k + (n%k > 0) << endl;
    cout << n/k + bool(n%k) << endl;
    cout << n/k + !!(n%k) << endl;
    return 0;
}
```

三种方法

§ 3. 程序设计初步

3.8. 循环结构和循环语句

- ★ 当型：先判断，后执行（可能一次都不执行）
- ★ 直到型：先执行，后判断（至少执行一次）



§ 3. 程序设计初步

3.8. 循环结构和循环语句

3.8.1. GOTO语句

形式:

goto 语句标号;

语句标号的组成:

语句标号:

(命名规则同变量: 以字母或下划线开始, 由字母、数字、下划线组成)

用途:

无条件跳转到语句标号处

缺点:

使程序的流程无规律, 可读性差(建议少用或不用)

```
#include <iostream>
using namespace std;
int main()
{
    LOOP:
        cout << "Hello";
        goto LOOP;

        return 0;
}
//程序执行会陷入死循环
```

§ 3. 程序设计初步

3.8. 循环结构和循环语句

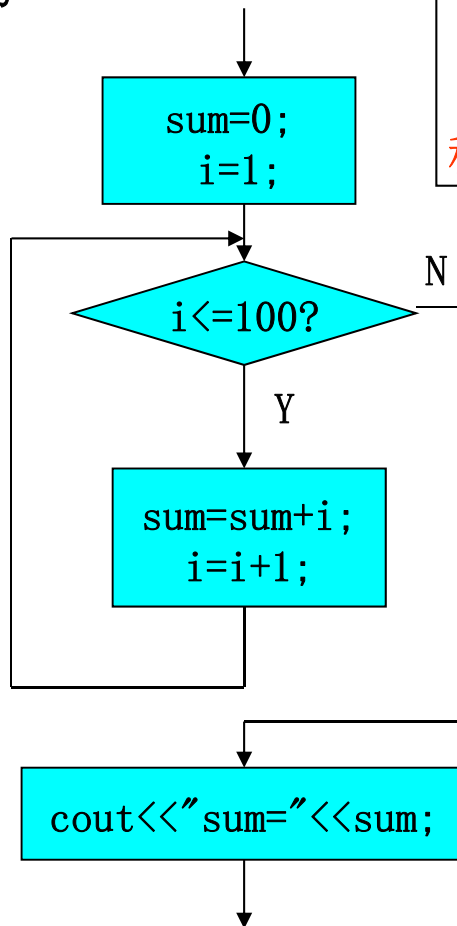
3.8.1. GOTO语句

与if语句一起构成循环:

例: 求 $1+2+\dots+100$ 的和

1、当型

```
int main()
{
    int i, sum;
    i=1;
    sum=0;
    LABEL1:
    if (i<=100) {
        sum=sum+i;
        i++;
        goto LABEL1;
    }
    cout<<"sum="<<sum<<endl;
    return 0;
}
```

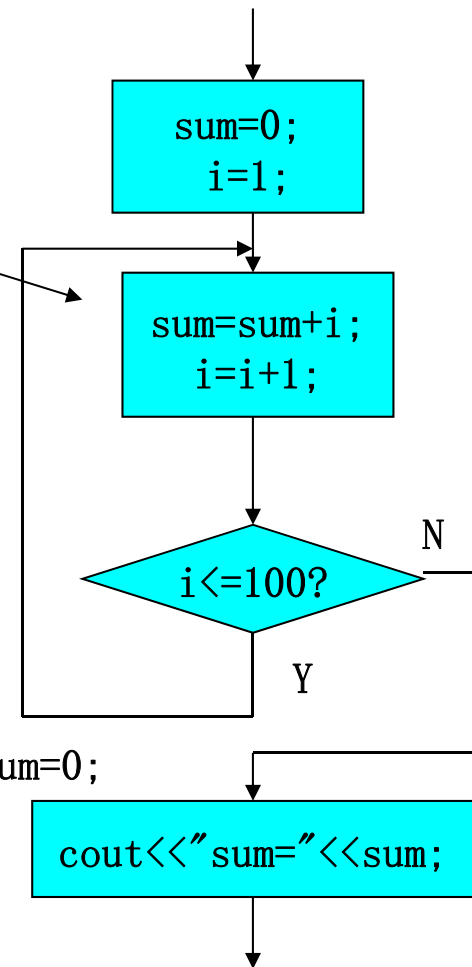


问题: 若调整为
 $i=i+1;$
 $sum=sum+i;$
程序如何变动才正确?

2、直到型

```
int main()
{
    int i=1; sum=0;
    LABEL1:
    sum+=i;
    i++;
    if (i<=100)
        goto LABEL1;
    cout<<"sum="<<sum<<endl;

    return 0;
}
```



§ 3. 程序设计初步

3.8. 循环结构和循环语句

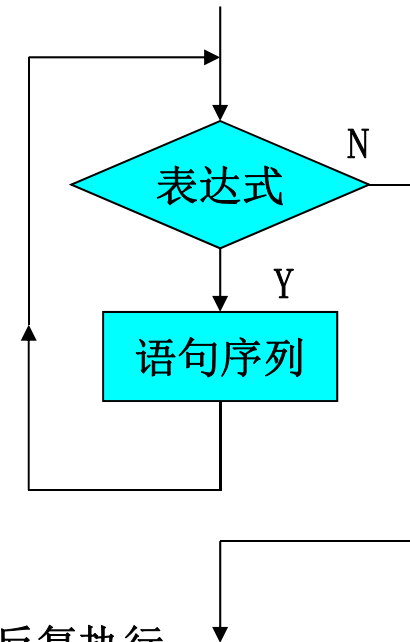
3.8.2. 用while语句构成循环

3.8.2.1. 形式

```
while (表达式) {  
    语句序列;  
}
```

3.8.2.2. 使用

- ★ 先判断，后执行
- ★ 表达式可以是任意类型，按逻辑值求解（非0为真0为假），为真时反复执行
- ★ 语句序列中只有一个语句时，{}可省
- ★ 语句序列中应有改变表达式取值的语句，否则死循环



例：求 $1+2+\dots+100$ 的和

```
#include <iostream>
using namespace std;
```

```
int main()
{
```

```
    int i=1, sum=0;
```

```
    while(i<=100) {
        sum=sum+i;
        i++;
    }
```

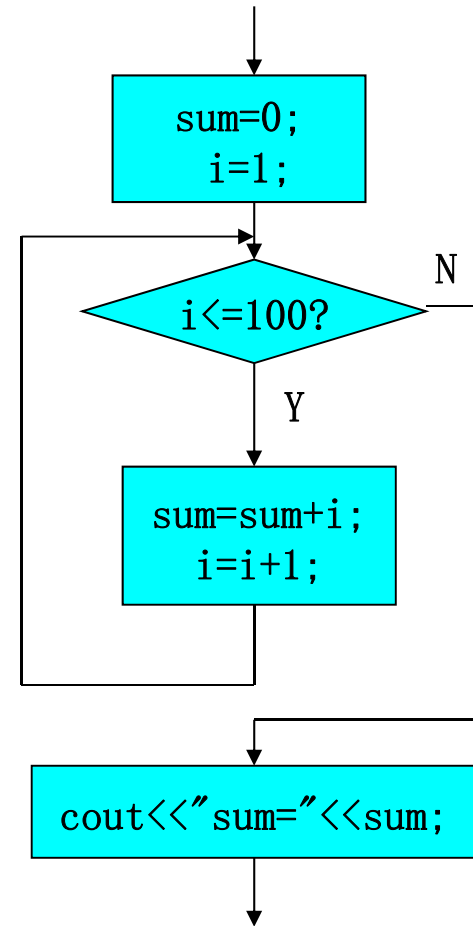
```
    cout<<"sum="<<sum<<endl;
```

```
    return 0;
```

```
}
```



```
while(i<=100)
    sum+=i++;
```



例：打印1-1000内7的倍数

```
#include <iostream>
using namespace std;

int main()
{
    int i=1;

    while(i<=1000) {
        if (i%7==0)
            cout << i << ' ';
        i++;
    }

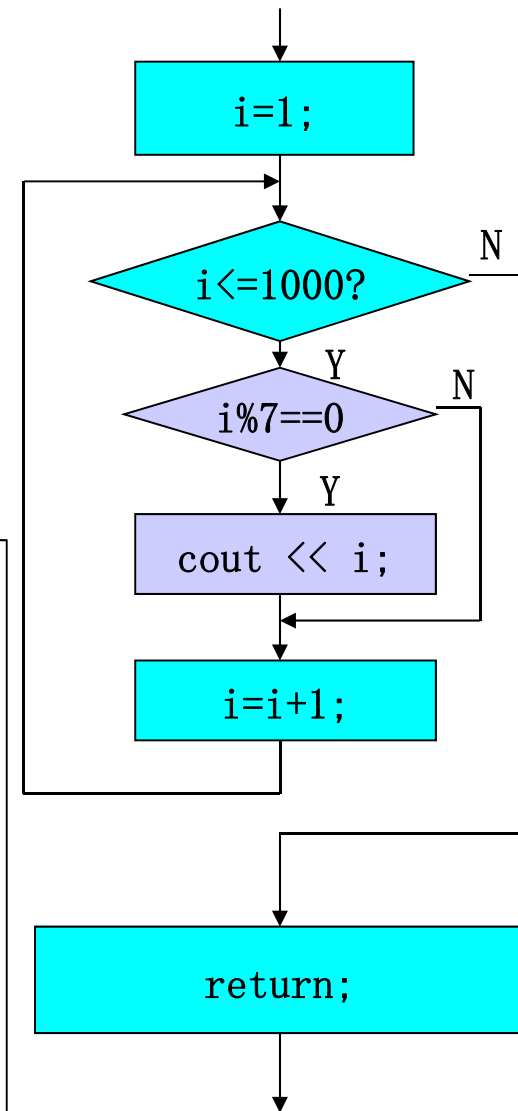
    return 0;
}
```

```
#include <iostream>
using namespace std;

int main()
{
    int i=7;

    while(i<=1000) {
        cout << i << ' ';
        i+=7;
    }

    return 0;
}
```



§ 3. 程序设计初步

3.8. 循环结构和循环语句

3.8.3. 用do-while语句构成循环

3.8.3.1. 形式

```
do {  
    语句序列;  
} while(表达式);
```

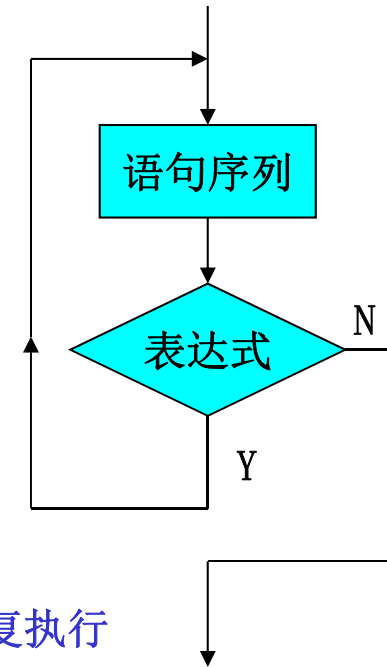
3.8.3.2. 使用

★ 先执行，后判断

★ 表达式可以是任意类型，按逻辑值求解（非0为真0为假），为真时反复执行

★ 语句序列中只有一个语句时，{}可省

★ 语句序列中应有改变表达式取值的语句，否则死循环



例：求 $1+2+\dots+100$ 的和

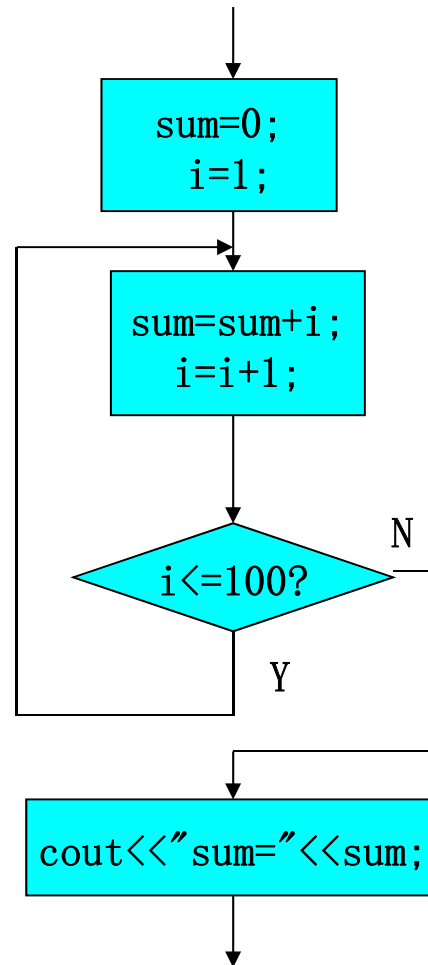
```
#include <iostream>
using namespace std;
```

```
int main()
{
    int i=1, sum=0;

    do {
        sum=sum+i;
        i++;
    } while(i<=100);

    cout<<"sum="<<sum<<endl;

    return 0;
}
```



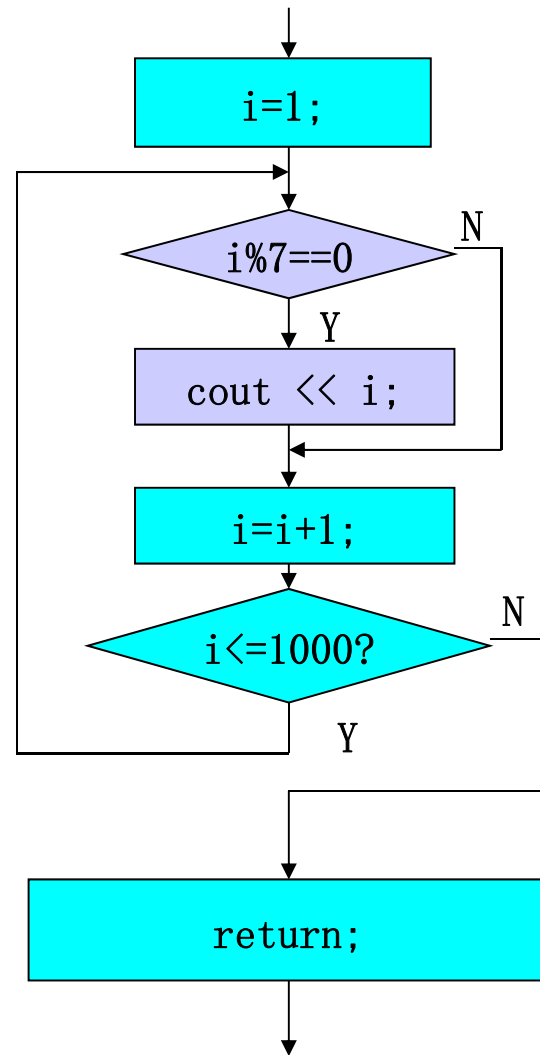
例：打印1-1000内7的倍数

```
#include <iostream>
using namespace std;

int main()
{
    int i=1;

    do {
        if (i%7==0)
            cout << i << ' ';
        i++;
    } while(i<=1000);

    return 0;
}
```



§ 3. 程序设计初步

3.8. 循环结构和循环语句

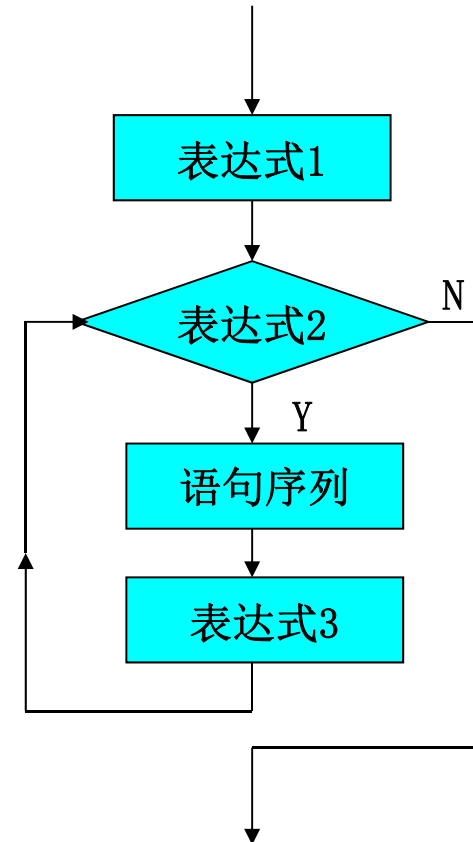
3.8.4. 用for语句构成循环

3.8.4.1. 形式

```
for(表达式1; 表达式2; 表达式3) {  
    语句序列;  
}
```

3.8.4.2. for语句的执行过程

- ① 求解表达式1 (初值)
- ② 以逻辑值求解表达式2，为真则执行循环体，
(当型) 为假则结束循环体的执行
- ③ 执行完循环体后，求解表达式3，重复②
(语句序列或表达式3中应有改变表达式2的求解条件)



§ 3. 程序设计初步

3.8. 循环结构和循环语句

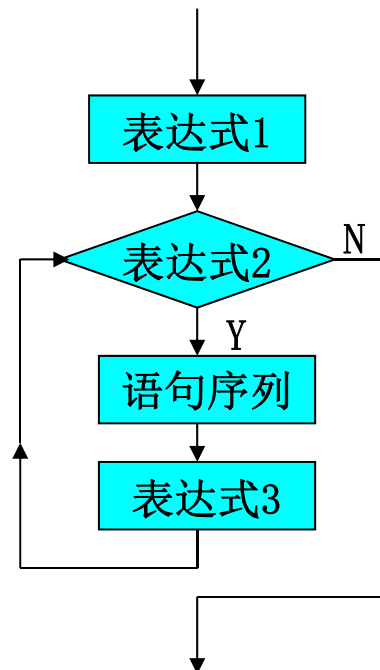
3.8.4. 用for语句构成循环

3.8.4.1. 形式

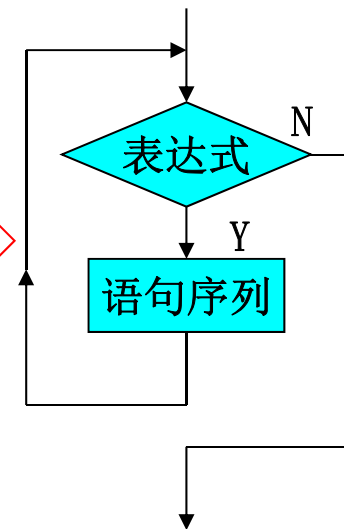
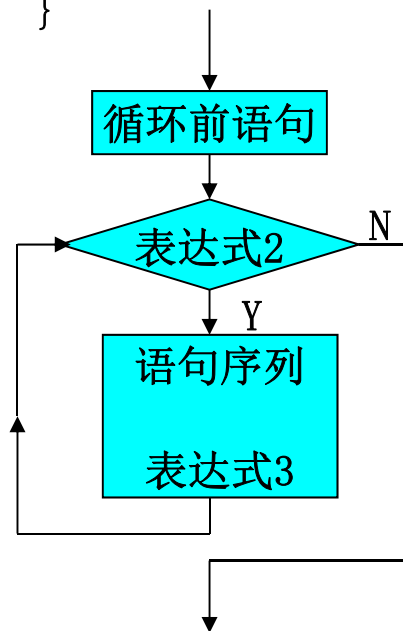
3.8.4.2. for语句的执行过程

3.8.4.3. 与while语句的互换性

```
for(表达式1;表达式2;表达式3) {  
    语句序列;  
}
```



```
表达式1;  
while(表达式2) {  
    语句序列;  
    表达式3;  
}
```



例：求 $1+2+\dots+100$ 的和

```
#include <iostream>
using namespace std;
int main()
{
    int i=1, sum=0;

    while(i<=100) {
        sum=sum+i;
        i++;
    }

    cout<<"sum="<<sum<<endl;

    return 0;
}
```

```
#include <iostream>
using namespace std;
int main()
{
    int i, sum=0;

    for(i=1;i<=100;i++)
        sum=sum+i;

    cout<<"sum="<<sum<<endl;

    return 0;
}
```


§ 3. 程序设计初步

3.8. 循环结构和循环语句

3.8.4. 用for语句构成循环

3.8.4.4. for语句的基本使用形式

```
for(循环变量赋初值; 循环条件; 循环变量增值) {  
    语句序列;  
}
```

★ 对已知循环结束条件(或循环次数)的循环表达方式较直观

例：i从1开始累加, 加到10000为止, 打印出i及累加的和

```
#include <iostream>  
#include <cstdio>  
using namespace std;  
  
int main()  
{  
    int i, sum;  
    for(i=1, sum=0; sum<=10000; i++)  
        sum=sum+i;  
    i--;  
    printf("i=%d sum=%d", i, sum);  
    return 0;  
}
```

```
#include <iostream>  
#include <cstdio>  
using namespace std;  
  
int main()  
{  
    int i=1, sum=0;  
    while(sum<=10000)  
        sum+=i++;  
    i--;  
    printf("i=%d sum=%d", i, sum);  
    return 0;  
}
```

§ 3. 程序设计初步

3.8. 循环结构和循环语句

3.8.4. 用for语句构成循环

3.8.4.5. for语句的扩展使用

★ 表达式1可省，在for语句前给变量赋值

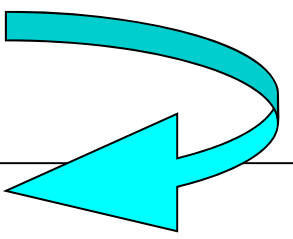
```
i=1;  
for (;i<=100;i++)  
    sum=sum+i;
```

```
for (i=1; i<=100; i++)  
    sum=sum+i;
```

★ 若表达式2省略，则永真（死循环）

可以在语句序列中设置相应条件以退出

```
for(i=1;;i++) {  
    ...  
    if (i>100)  
        ...  
}
```



§ 3. 程序设计初步

3.8. 循环结构和循环语句

3.8.4. 用for语句构成循环

3.8.4.5. for语句的扩展使用

★ 表达式3可省，另外设法改变表达式2的取值

```
for(i=0; ++i<=100;)
    sum=sum+i;
```

```
for(i=1; i<=100;) {
    sum=sum+i;
    i++;
}
```

```
for(i=0; i++<100;)
    sum=sum+i;
```

```
for(i=1; i<=100;)
    sum+=i++;
```

```
for (i=1; i<=100; i++)
    sum=sum+i;
```

★ 省略表达式1、3，完全等同于while语句的形式

```
i=1;
for(; i<=100;) {
    sum=sum+i;
    i++;
}
```

```
i=1;
while(i<=100) {
    sum=sum+i;
    i++;
}
```

★ 三个表达式全省，相当于永真

```
for(;;) {
    ...
}
```

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

§ 3. 程序设计初步

3.8. 循环结构和循环语句

3.8.4. 用for语句构成循环


3.8.4.5. for语句的扩展使用

★ 表达式1、3可以是简单表达式，也可以是多个简单表达式组合形式的逗号表达式


```
int i, sum=0;
for(i=1;i<=100;i++)
    sum=sum+i;
```

```
int i, sum;
for(i=1, sum=0; i<=100; i++)
    sum=sum+i;
```

```
int i, sum;
for(i=1, sum=0; i<=100; sum=sum+i, i++);
```




```
int i, sum;
for(i=1, sum=0; i<=100; sum+=i++);
```




```
#include <iostream>
using namespace std;

int main()
{
    int i, sum;
    for(i=1, sum=0; i<=100; sum+=i++);
        cout << "sum=" << sum << endl;
    return 0;
} //cout形式上缩进，但仍然和for是平级的
```



```
#include <iostream>
using namespace std;
int main()
{
    int i, sum;
    for(i=1, sum=0; i<=100; sum+=i++)
        ; //空语句方式，表达更清晰
    cout << "sum=" << sum << endl;
    return 0;
}
```



§ 3. 程序设计初步

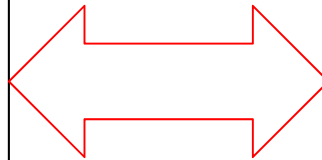
3.8. 循环结构和循环语句

3.8.4. 用for语句构成循环

3.8.4.5. for语句的扩展使用

★ 表达式2可以是任何类型，但按逻辑值求解

```
char c;  
c=getchar();  
while(c!='\n') {  
    cout<<c;  
    c=getchar();  
}
```



```
char c;  
for(;;(c=getchar())!='\n';cout<<c)  
    ;
```

for的表达式2要分三步理解

虽然简单，但可读性差，建议初学者不把与循环变量无关的内容放入for语句

§ 3. 程序设计初步

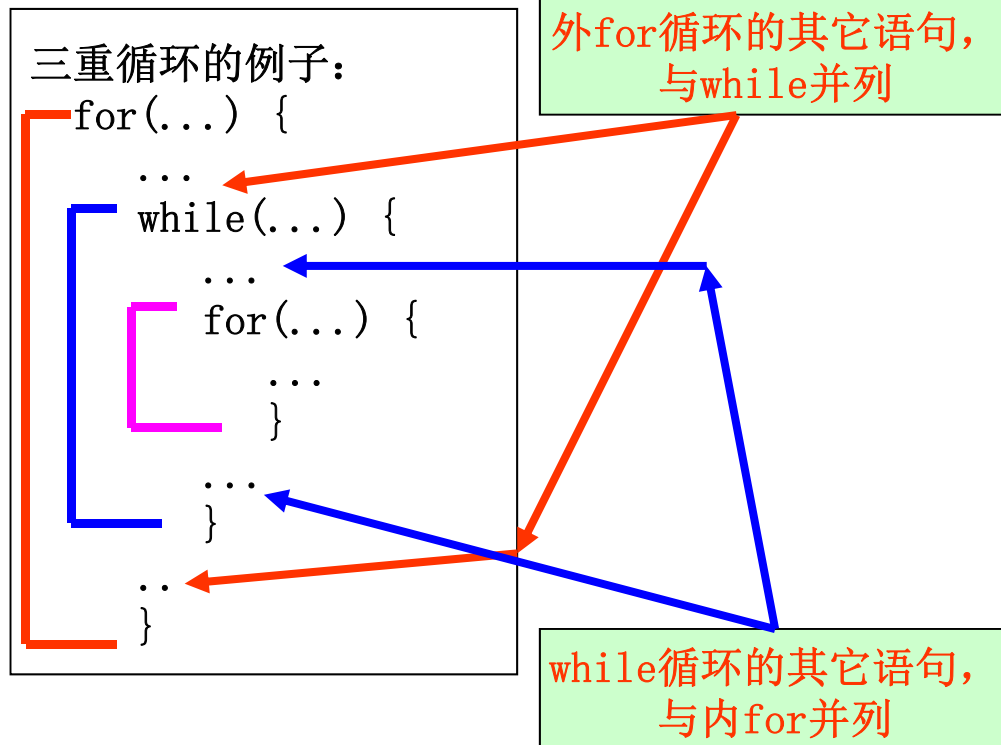
3.8. 循环结构和循环语句

3.8.5. 循环的嵌套

P. 79-80 多种形式

★ {} 的匹配原则

与if/else{}的匹配类似，“{”进栈，遇到”}”出栈匹配



<pre>while(...) { while(...) { ... } }</pre>	<pre>do { do{ ... } while(...); } while(...);</pre>
<pre>for(...) { for(...) { ... } }</pre>	<pre>while(...) { do(...) { ... } while(...); }</pre>
<pre>for(...) { while(...) { ... } }</pre>	<pre>do (...) { for(...) { ... } } while(...);</pre>

§ 3. 程序设计初步

3.8.5. 循环的嵌套

P. 79-80 多种形式

★ {} 的匹配原则

与if/else{}的匹配类似，“{”进栈，遇到”}”出栈匹配

★ 外层循环每执行一次，内层循环都要执行一遍

★ 各种分支语句、循环语句之间可相互任意嵌套，只要 {} 的匹配理解没有问题，就是正确的

```
for(i=1;i<=100;i++)  
    for(j=1;j<=100;j++)  
        cout << i*j << ' ';
```

cout 语句执行了10000遍

```
for(i=1;i<=100;i++) {  
    cout << i << endl;  
    for(j=1;j<=100;j++) {  
        cout << i*j << endl;  
        for(k=1;k<=100;k++)  
            cout << i*j*k << ' ' ;  
        cout << j*i << endl;  
    }  
    cout << i << endl;  
}
```

红语句及for j执行了_____遍

蓝语句及for k执行了_____遍

粉语句执行了_____遍

§ 3. 程序设计初步

3.8. 循环结构和循环语句

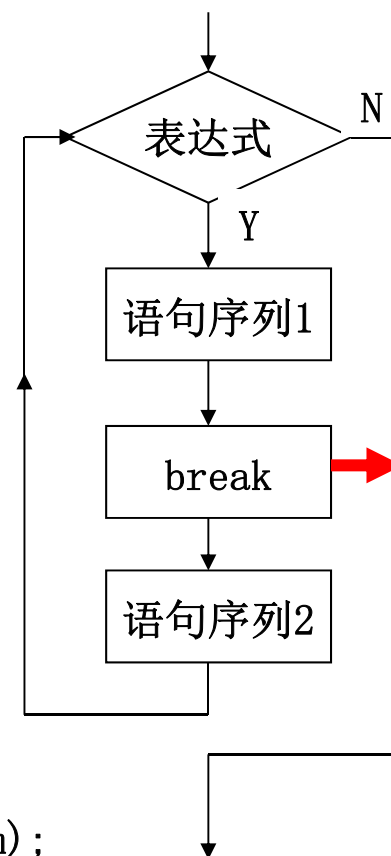
3.8.6. 改变循环控制的语句

3.8.6.1. break语句 (前面switch/case中用过)

作用：提前结束循环体的循环

例：i从1开始累加，加到10000为止，打印出i及累加的和

<pre>int main() { int i, sum; for(i=1, sum=0; sum<=10000; i++) sum=sum+i; i--; printf("i=%d sum=%d", i, sum); return 0; }</pre>	<pre>int main() { int i=1, sum=0; while(sum<=10000) sum+=i++; i--; printf("i=%d sum=%d", i, sum); return 0; }</pre>
<pre>for(i=1, sum=0;; i++) { sum=sum+i; if (sum>10000) break; } //不需要i--</pre>	<pre>while(1) { sum=sum+i++; if (sum>10000) break; } i--;</pre>



§ 3. 程序设计初步

3.8. 循环结构和循环语句

3.8.6. 改变循环控制的语句

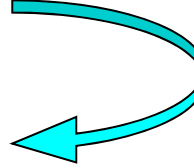
3.8.6.1. break语句（前面switch/case中用过）

作用：提前结束循环体的循环

★ 无条件结束循环体，因此必须和if/else语句一起使用才能体现实际的意义

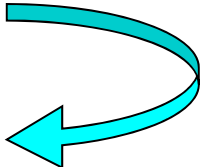
★ 当多重循环嵌套时，break仅跳出本循环

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

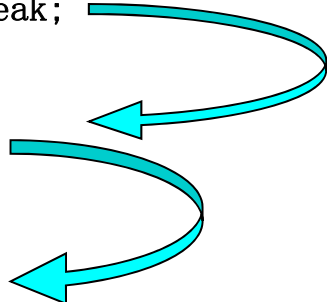


★ 若出现循环和switch语句的嵌套，则break的位置决定了跳转的位置

```
for(...) {  
    switch(...) {  
        case ....:  
            ...  
            break;  
            ...  
    }  
    ...  
}
```



```
switch(...) {  
    case ...:  
        for(...) {  
            ...  
            break;  
        }  
    ...  
    break;  
    case ...  
}
```



§ 3. 程序设计初步

3.8. 循环结构和循环语句

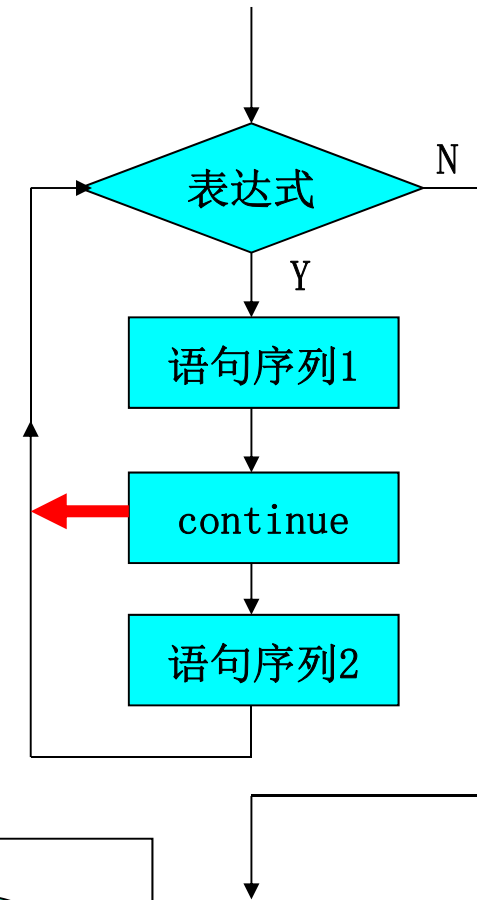
3.8.6. 改变循环控制的语句

3.8.6.1. break语句

3.8.6.2. continue语句

作用：结束本次循环，进行下一次是否执行循环的判断

- ★ 无条件结束本次循环，因此必须和if/else语句一起使用才能体现实际的意义
- ★ 若出现循环和switch语句的嵌套，则continue只对循环体有效
- ★ for语句中若出现continue，则先执行表达式3，再去判断表达式2是否应该继续执行



```
switch(...) {  
    case ...:  
        for(...) {  
            ...  
            continue;  
        }  
        ...  
        break;  
    case ...  
}
```

```
for(...) {  
    switch(...) {  
        case ...:  
            ...  
            continue;  
        ...  
    }  
    ...  
}
```

例：打印1-1000内7的倍数

```
#include <iostream>
using namespace std;
```

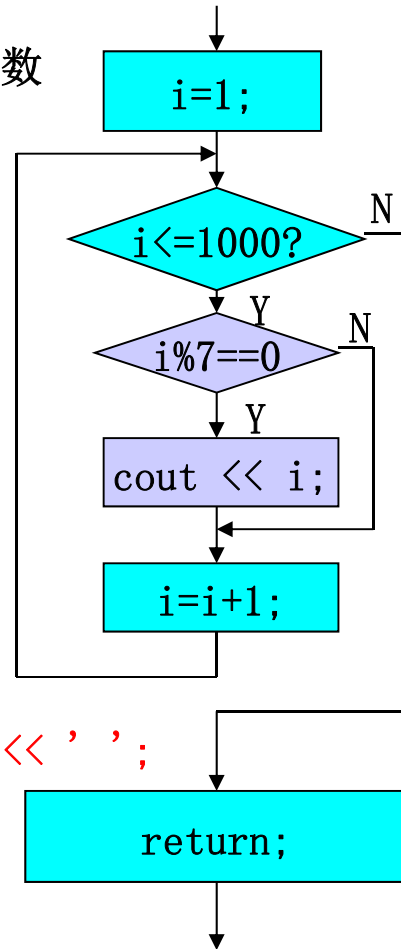
```
int main()
{
```

```
    int i=1;
```

```
    while(i<=1000) {
        if (i%7==0)
            cout << i << ' ';
        i++;
    }
```

```
    return 0;
```

```
}
```



```
#include <iostream>
using namespace std;
```

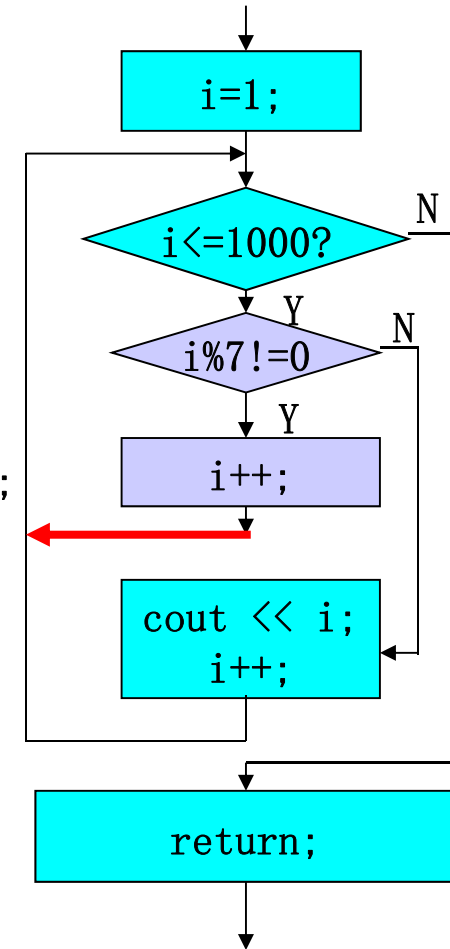
```
int main()
{
```

```
    int i=1;
    while(i<=1000) {
        if (i%7!=0) {
            i++;
            continue;
        }
```

```
        cout << i << ' ';
        i++;
    }
```

```
    return 0;
```

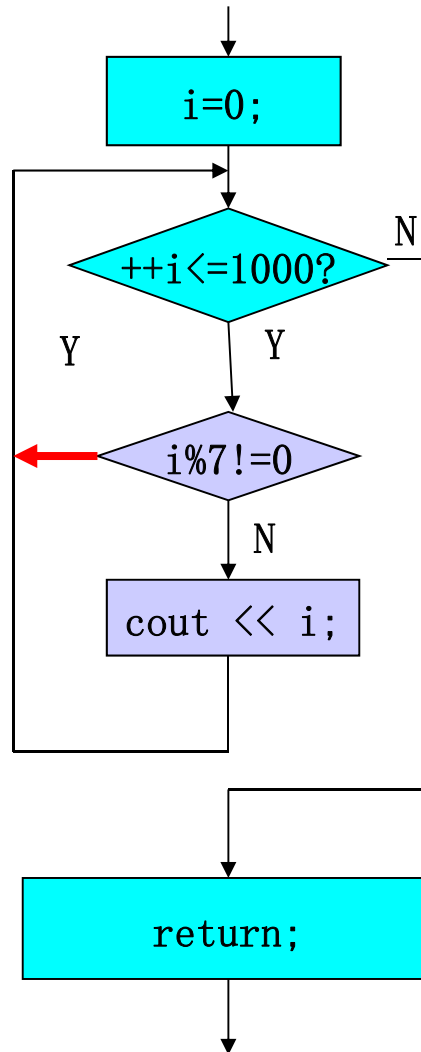
```
}
```



例：打印1-1000内7的倍数

```
#include <iostream>
using namespace std;

int main()
{
    int i=0;
    while(++i<=1000) {
        if (i%7!=0)
            continue;
        cout <<i << ' ';
    }
    return 0;
}
```



```
#include <iostream>
using namespace std;

int main()
{
    int i;
    for(i=1;i<=1000;i++) {
        if (i%7!=0)
            continue;
        cout << i << ' ';
    }
    return 0;
}
```

§ 3. 程序设计初步

3.8.6. 改变循环控制的语句

3.8.6.3. break和continue的比较

```
while(表达式1) {  
    语句序列1;  
    break/continue;  
    语句序列2;  
}
```

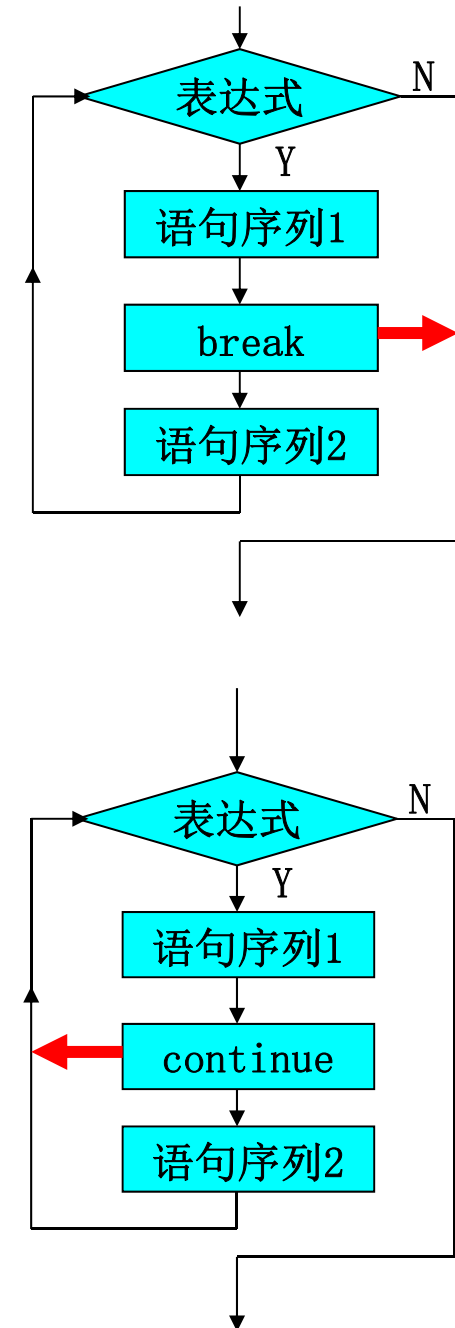
例：给出下列程序的运行结果

```
#include <iostream>  
using namespace std;  
  
int main()  
{  
    int i=0, sum=0;  
    while(i<1000) {  
        i++;  
        break;  
        sum=sum+i;  
    }  
    cout << "i=" << i  
        << " sum=" << sum;  
        << endl;  
    return 0;  
}
```

?

```
#include <iostream>  
using namespace std;  
  
int main()  
{  
    int i=0, sum=0;  
    while(i<1000) {  
        i++;  
        continue;  
        sum=sum+i;  
    }  
    cout << "i=" << i  
        << " sum=" << sum;  
        << endl;  
    return 0;  
}
```

?



§ 3. 程序设计初步

3.8. 循环结构和循环语句

3.8.7. 编写循环结构的程序

P. 75-80

例3.12: 用公式求 π 值

例3.13: 求 Fibonacci 数列的前40项

例3.14: 找出100-200间的全部素数


例3.15: 密码转换

§ 3. 程序设计初步

例3.12: 用公式求 π 值

```
#include <cmath> //所有数学类函数对应的头文件
```

```
while(fabs(t)>1e-7) {  
    pi = pi + t;    //pi为累加和  
    n  = n + 2;     //n为分母  
    s  = -s;        //分子在正负1间变化  
    t  = s/n;       //每项的值  
}
```


$$\frac{1}{1} + \frac{-1}{3} + \frac{1}{5} + \frac{-1}{7}$$

例3.12: 调整两处蓝色箭头处的值, 对比不同精度的执行时间

```
#include <iostream>
#include <iomanip>    //格式输出
#include <cmath>      //fabs
#include <windows.h> //取系统时间
using namespace std;
int main()
{
    int s=1;
    double n=1, t=1, pi=0;
    long t_start, t_end;

    t_start = GetTickCount(); //取当前系统时间, 单位毫秒, 精度55ms
    while(fabs(t) > 1e-6) {
        pi = pi + t;
        n = n + 2;
        s = -s;
        t = s/n;
    }
    t_end = GetTickCount(); //取当前系统时间

    /* 执行到此, 求得 pi/4 的值 */
    pi=pi*4;
    cout << "n=" << setprecision(10) << n << endl;
    cout << "pi=" << setiosflags(ios::fixed) << setprecision(9) << pi << endl;

    /* 以秒为单位输出循环的执行时间 */
    eout << "time=" << (t_end - t_start)/1000.0 << endl;
    return 0;
}
```

(1) n, t, pi为double型

精度为1e-6:	pi=	n=
1e-7:	pi=	n=
1e-8:	pi=	n=
1e-9:	pi=	n=

(2) n, t, pi为float型

精度为1e-6:	pi=	n=
1e-7:	pi=	n=
1e-8:	为什么无结果?	

有删除线的代码去掉后, 可以在Linux下编译

c++ -o demo demo.cpp : 编译

time ./demo : 观察运行时间

§ 3. 程序设计初步

例3.13: 求 Fibonacci 数列的前40项

```
f1 = f2 = 1;    //初值
for(i=1; i<=20; i++) { //每次2数, 20次=40个
    cout << setw(12) << f1 << setw(12) << f2;
    if (i%2==0)
        cout << endl; //每2次(4个)加换行
    f1 = f1 + f2;    //f1为第3/5/7/... 个月
    f2 = f1 + f2;    //f2为第4/6/8/... 个月
}
```

例3.14: 找出100-200间的全部素数

```
for(m=101; m<=200; m+=2) { //偶数没必要判断
    prime=true;           //对每个数, 先认为是素数
    k=int(sqrt(m));        // k=sqrt(m) 也可 (有警告)
```

任意 $m\%i==0$ 就不是素数,
循环不必再继续执行

整个循环完成, $m\%i==0$
都未满足, 才认为是素数

```
for(i=2; i<=k; i++)
    if (m%i==0) {
        prime=false;
        break;
    }
```

```
for(i=2; prime && i<=k; i++)
    if (m%i==0)
        prime=false;

//是否可以改成这种形式?
```

```
if (prime) {
    cout << setw(5) << m;
    n=n+1;        //计数器, 只为了加输出换行
}
if (n%10==0)     //每10个数输出一行
    cout<<endl;
}
```

```
if (prime) {
    cout << setw(5) << m;
    n=n+1;
    if (n%10==0)
        cout<<endl;
}
```

//是否应该改成这种形式?
令 $m=103-200$, 看运行差别
哪个代码正确?

例3.14: 找出100-200间的全部素数(改写, 取消prime)

```
for(m=101; m<=200; m+=2) {  
    k=int(sqrt(m));  
    for(i=2; i<=k; i++)  
        if (m%i==0)  
            break;  
    if (i>k) {  
        cout << setw(5) << m;  
        n=n+1;  
        if (n%10==0)  
            cout << endl;  
    }  
}
```

不要直接 $i \leq \sqrt{m}$
为什么?

i循环的退出有两个可能
1、不满足 $i \leq k$ (是, 且 $i > k$)
2、满足 $m \% i == 0$ (否, 且 $i \leq k$)
哪种可能性表示是素数?

§ 3. 程序设计初步

例3.15: 密码转换

```
while((c=getchar())!='\n') {  
    if ((c>='a' && c<='z') || (c>='A' && c<='Z')) {  
        c+=4;  
        if (c>'Z' && c<='Z'+4 || c>'z')  
            c-=26; //对WXYZ, 要转成ABCD  
    }  
}
```

if语句中的c
必定是字母

- 1、对于c>'Z'的情况，除WXYZ+4外，还包括所有小写字母，因此必须再加上 c<='Z'+4
- 2、对于c>'z'的情况，只有wxyz+4，因此不需要c<='z'+4

```
cout<<c; //如if不满足(非字母)则输出原字符  
}          如if 满足          则输出加密字符
```