



## § 3. 结构化程序设计基础

### 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. 结构化程序设计基础

### 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. 结构化程序设计基础

### 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. 结构化程序设计基础

### 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（课上未讲，简单体会一下即可）

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：（单分支结构）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. 结构化程序设计基础

### 3.1. 面向过程的程序设计和算法

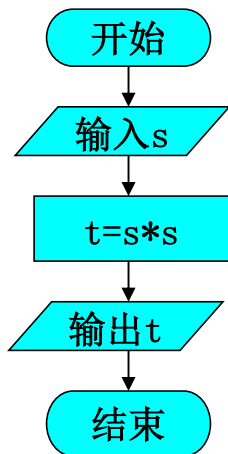
#### 3.1.4. 算法的表示

##### 3.1.4.2. 流程图表示

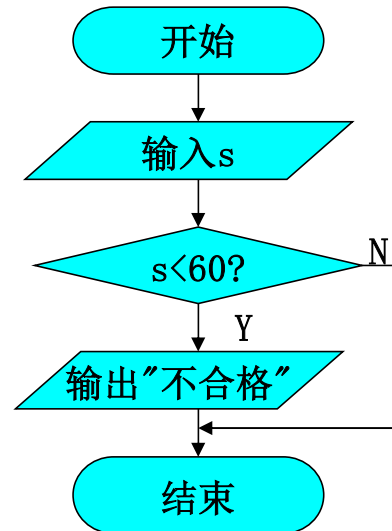
★ 基本图形表示（参考其它书籍）

★ 缺陷：对较大的程序，过于复杂，难以阅读和修改

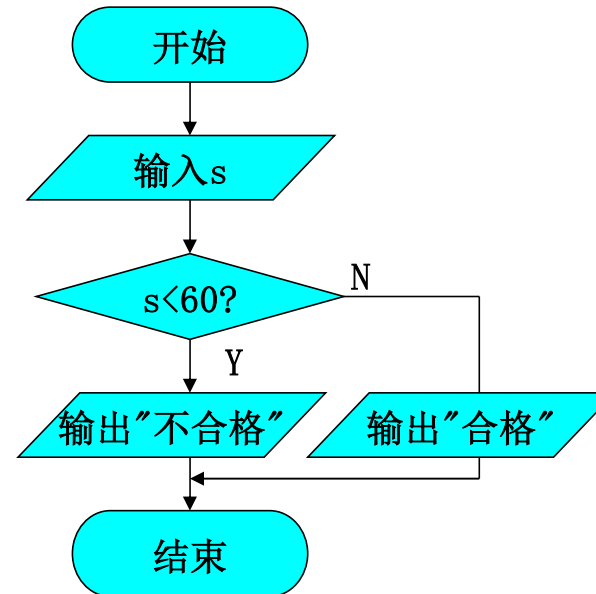
例1：（顺序结构）输出一个数字的平方



例2：（单分支结构）当一个成绩小于60分时吗，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”





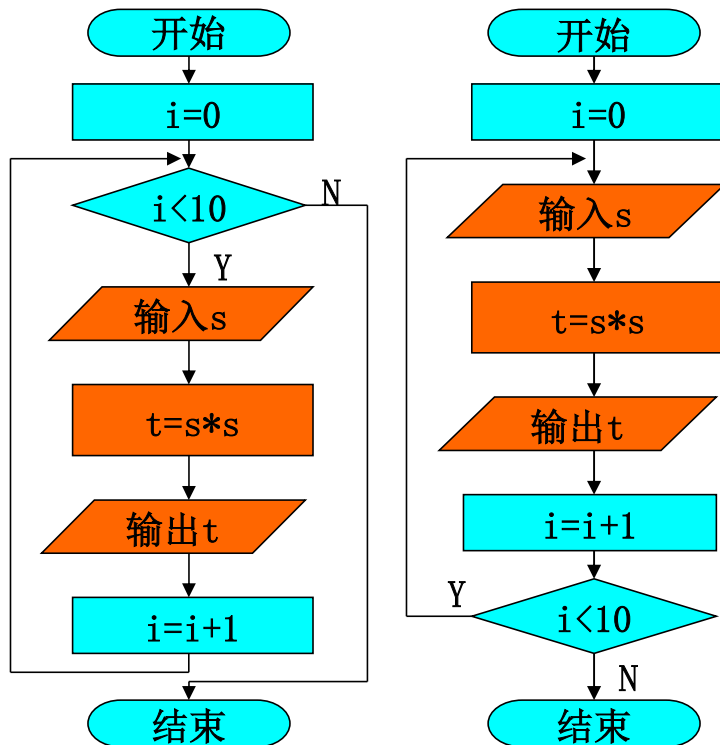
## § 3. 结构化程序设计基础

### 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

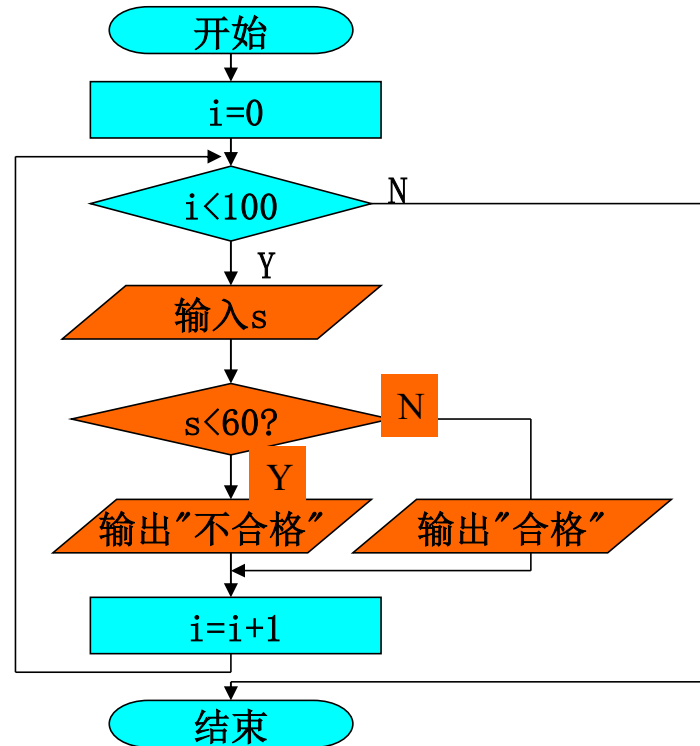


当型循环

思考: 当i的初值为20时, 左右  
两侧的执行结果是否一致?

直到型循环

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. 结构化程序设计基础

### 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: (单分支结构) 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. 结构化程序设计基础

### 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

|                                                                                                |                                                                                                 |
|------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|
| <pre>BEGIN   i=0   while i&lt;10 {     input s     t = s*s     print t     i=i+1   } END</pre> | <pre>BEGIN   i=0   do {     input s     t=s*s     print t     i=i+1   } while i&lt;10 END</pre> |
|------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|

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”

|                                                                                                                                         |                                                                                                                                            |
|-----------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>BEGIN   i=0   while i&lt;100 {     input s     if s&lt;60 then       print "不合格"     else       print "合格"     i=i+1   } END</pre> | <pre>BEGIN   i=0   do {     input s     if s&lt;60 then       print "不合格"     else       print "合格"     i=i+1   } while i&lt;100 END</pre> |
|-----------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|





## § 3. 结构化程序设计基础

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

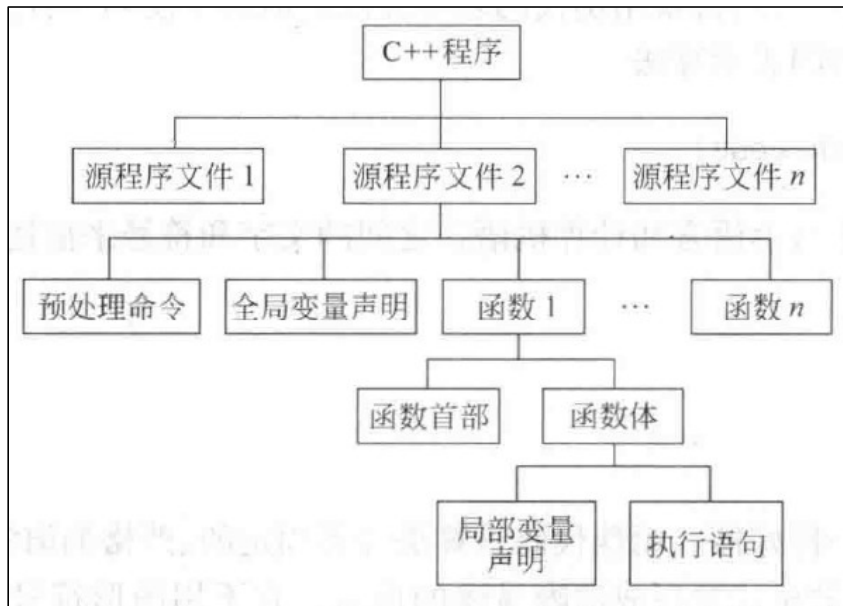


## § 3. 结构化程序设计基础

### 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）



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. 结构化程序设计基础

### 3.2. C++的程序结构和C++语句

#### 3.2.2. 语句的种类

共分为四种

★ 定义/声明语句：定义变量，放在使用该变量的语句前面

```
int a; //定义  
a=10; //使用
```

C++中变量必须先定义、后使用

★ 执行语句：

控制语句： 9种

```
if-else  
switch  
break  
goto  
for  
while  
do-while  
continue  
return
```

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;

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

★ 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



## § 3. 结构化程序设计基础

### 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
```

```
cpp-demo.cpp * x  
cpp-demo (全局范围)  
1  #include <iostream>  
2  using namespace std;  
3  
4  int main()  
5  {  
6      int a;  
7      (a = 3) * 10;    //正确，赋值表达式，值可参与运算  
8      (a = 3;) * 10;  //错误，赋值语句，语句不能参与运算  
9      return 0;  
10 }
```

(7,13): warning C4552: " \* ": 未使用表达式结果  
(8,11): error C2059: 语法错误: ";"  
(8,12): error C2059: 语法错误: ")"  
(8,14): error C2100: 非法的间接寻址

错误的具体含义  
暂时不需要了解



## § 3. 结构化程序设计基础

### 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（也称为**字节流**）

★ 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

下发的附录（D-运算符优先级）

1、>>和<<是优先级第7组，称为**左移/右移运算符**，本处所称的**流插入/流提取运算符**本质上是**将左移/右移运算符**经过**重载**而得到的（重载：后续荣誉课程）

2、优先级第15组中 <<=和>>= 称为**左移/右移并赋值**，也称为**复合按位左移/右移运算符**



## § 3. 结构化程序设计基础

### 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

```
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;  
}
```

```
//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;  
}
```

仔细回想课程 Windows 和 Linux 下的演示, 哪个操作系统下更符合规范?



## § 3. 结构化程序设计基础

### 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 (下面例子中这四种形式的输出是完全一致的!)

|                                                                                                                                            |                   |                                                                                                                                                              |               |
|--------------------------------------------------------------------------------------------------------------------------------------------|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|
| <code>cout &lt;&lt; "This is a C++ program." &lt;&lt; endl;</code>                                                                         |                   |                                                                                                                                                              |               |
| <code>cout &lt;&lt; "This is " &lt;&lt; "a C++ " &lt;&lt; "program." &lt;&lt; endl;</code>                                                 |                   |                                                                                                                                                              |               |
| <code>cout &lt;&lt; "This is "</code><br><code>&lt;&lt; "a C++ "</code><br><code>&lt;&lt; "program."</code><br><code>&lt;&lt; endl;</code> | 一个语句分4行<br>前三行无分号 | <code>cout &lt;&lt; "This is ";</code><br><code>cout &lt;&lt; "a C++ ";</code><br><code>cout &lt;&lt; "program.";</code><br><code>cout &lt;&lt; endl;</code> | 4个语句<br>每行有分号 |



## § 3. 结构化程序设计基础

### 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;
```

再次强调：只输出最原始数据，其它需要自行加入

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

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？

error C2563: 在形参表中不匹配  
error C2568: “<<”: 无法解析函数重载





## § 3. 结构化程序设计基础

### 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

保持char类型不变，  
希望输出65，如何做？

保持char类型不变，  
希望输出65，不准用强制  
类型转换，又该如何做？

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

65

保持int类型不变，  
希望输出A，如何做？



## § 3. 结构化程序设计基础

### 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

|                                                               |                   |                              |               |
|---------------------------------------------------------------|-------------------|------------------------------|---------------|
| <code>cin &gt;&gt; a &gt;&gt; b &gt;&gt; c &gt;&gt; d;</code> |                   |                              |               |
| <code>cin &gt;&gt; a</code>                                   | 1个语句分4行<br>前3行无分号 | <code>cin &gt;&gt; a;</code> | 4个语句<br>每行有分号 |
| <code>&gt;&gt; b</code>                                       |                   | <code>cin &gt;&gt; b;</code> |               |
| <code>&gt;&gt; c</code>                                       |                   | <code>cin &gt;&gt; c;</code> |               |
| <code>&gt;&gt; d;</code>                                      |                   | <code>cin &gt;&gt; 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;` (VS编译出错, `b, c`未初始化; 其他编译器可执行, 观察`bc`的值, 自己动手做)

★ 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. 结构化程序设计基础

### 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=" << k << endl;

    return 0;
}
```

上课认真听  
下课动手做，仔细观察并思考课上所讲

课上的Linux，  
这里值是多少？

运行7次

输入: 123✓ (正确+回车)  
输入: 123 456✓ (正确+空格)  
输入: -123m456✓ (正确+非法字符)

输入: m✓ (直接非法字符)

输入: 54321✓ (超上限)  
输入: -40000✓ (超下限)

输入: ✓ (只按回车)

值  
123  
123  
-123

0  
?  
?  
上限 ?  
下限 ?

输入: ✓ 继续等待输入

- 输入终止条件前如果输入正确，可以得到正确的结果(所有编译器)
- 出现输入错误，则不同编译器表现不同(具体现象已观察到)
  - => 54321已超上限，为什么不取到5432后自动停止？
  - => -40000已超上限，为什么不取到-4000后自动停止？
  - => 有什么隐含规则么？

★ 变量读取后，系统会判断输入数据是否超过变量的范围，若超过则置内部的错误标记并返回一个不可信的值(不同编译器处理不同)

注意: 当`cin.good()`为0 或  
`cin.fail()`为1 时，  
值不可信，不讨论具体值!!!



## § 3. 结构化程序设计基础

### 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

★ 系统会自动根据cin后变量的类型按**合理且最长原则**来读取数据

cin读十进制的规则: 首字符+/-/0~9, 后续字符0-9, 长度不限

**输入54321:**

读到5 - 合理10进制数码, 继续读  
读到4 - 合理10进制数码, 继续读  
读到3 - 合理10进制数码, 继续读  
读到2 - 合理10进制数码, 继续读  
读到1 - 合理10进制数码, 继续读  
读到回车, 结束, 开始处理54321  
=> 超上限

cin读十进制的规则: 首字符+/-/0~9, 后续字符0-9, 长度不限

**输入-40000:**

读到- - 首字符正负号, 合理, 继续读  
读到4 - 合理10进制数码, 继续读  
读到0 - 合理10进制数码, 继续读  
读到0 - 合理10进制数码, 继续读  
读到0 - 合理10进制数码, 继续读  
读到0 - 合理10进制数码, 继续读  
读到回车, 结束, 开始处理-40000  
=> 超下限

字符常量的八进制表示: \开头, 数码0~7, 最多三位

**第2章作业:**

```
cout << strlen("\23456f") << endl;  
cout << strlen("\43456f") << endl; //本行报错
```

**"\23456f"**

读到\ - 转义符开始, 合理, 继续读  
读到2 - 转义符跟8进制, 合理, 第1个, 合理, 继续读  
读到3 - 转义符跟8进制, 合理, 第2个, 合理, 继续读  
读到4 - 转义符跟8进制, 合理, 第3个, 合理, 满3个, 结束  
满3个, 结束, 开始处理\234, **合理**

字符常量的八进制表示: \开头, 数码0~7, 最多三位

**第2章作业:**

```
cout << strlen("\23456f") << endl;  
cout << strlen("\43456f") << endl; //本行报错
```

**"\43456f"**

读到\ - 转义符开始, 合理, 继续读  
读到4 - 转义符跟8进制, 合理, 第1个, 合理, 继续读  
读到3 - 转义符跟8进制, 合理, 第2个, 合理, 继续读  
读到4 - 转义符跟8进制, 合理, 第3个, 合理, 满3个, 结束  
满3个, 结束, 开始处理\434, **不合理, 报错**

error C7744: 转义序列 "\434" 超出范围



## § 3. 结构化程序设计基础

### 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

★ 系统会自动根据cin后变量的类型按**合理且最长原则**来读取数据

★ 变量读取后, 系统会判断输入数据是否超过变量的范围, 若超过则置**内部的错误标记**并返回一个**不可信**的值  
(不同编译器处理不同)

★ cin输入完成后, 通过`cin.good()/cin.fail()`可判断本次输入是否正确

| 输入                      | <code>cin.good()</code> 返回 | <code>cin.fail()</code> 返回 |
|-------------------------|----------------------------|----------------------------|
| <b>正确范围</b> +回车/空格/非法输入 | 1                          | 0                          |
| <b>错误范围</b> +回车/空格/非法输入 | 0                          | 1                          |
| <b>非法输入</b>             | 0                          | 1                          |



# § 3. 结构化程序设计基础

## 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

上课认真听  
下课动手做, 仔细观察并思考课上所讲

课上的Linux,  
这里值是多少?

|                                                                                                                                                                                                                                                                                                                  |                         |              |        |        |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------|--------------|--------|--------|
| <pre>#include &lt;iostream&gt; using namespace std;  int main() {     short k;     cin &gt;&gt; k;     cout &lt;&lt; "k=" &lt;&lt; k &lt;&lt; endl;     cout &lt;&lt; "cin.good()=" &lt;&lt; cin.good() &lt;&lt; endl;     cout &lt;&lt; "cin.fail()=" &lt;&lt; cin.fail() &lt;&lt; endl;      return 0; }</pre> | 运行7次                    | 值            | good() | fail() |
|                                                                                                                                                                                                                                                                                                                  | 输入: 123✓ (正确+回车)        | 123          | 1      | 0      |
|                                                                                                                                                                                                                                                                                                                  | 输入: 123 456✓ (正确+空格)    | 123          | 1      | 0      |
|                                                                                                                                                                                                                                                                                                                  | 输入: -123m456✓ (正确+非法字符) | -123         | 1      | 0      |
|                                                                                                                                                                                                                                                                                                                  | 输入: m✓ (直接非法字符)         | 0            | 0      | 1      |
|                                                                                                                                                                                                                                                                                                                  | 输入: 54321✓ (超上限)        | 上限           | 0      | 1      |
|                                                                                                                                                                                                                                                                                                                  | 输入: -40000✓ (超下限)       | 下限           | 0      | 1      |
|                                                                                                                                                                                                                                                                                                                  | 输入: ✓ (只按回车)            | 输入: ✓ 继续等待输入 |        |        |

★ 变量读取后, 系统会判断输入数据是否超过变量的范围, 若超过则置内部的错误标记并返回一个不可信的值 (不同编译器处理不同)

注意: 当cin.good()为0 或 cin.fail()为1 时, 值不可信, 不讨论具体值!!!



## § 3. 结构化程序设计基础

### 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

上课认真听

下课动手做, 仔细观察并思考课上所讲

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

int main()
{
    unsigned short k;
    cin >> k;
    cout << "k=" << k << endl;
    cout << "cin.good()=" << cin.good() << endl;
    cout << "cin.fail()=" << cin.fail() << endl;
    return 0;
}
```

|                         | 值     | good() | fail() |
|-------------------------|-------|--------|--------|
| 输入: -65536✓ (负数但绝对值超上限) | 65535 | 0      | 1      |
| 输入: -65535✓ (负数但绝对值未超)  | 1     | 1      | 0      |
| 输入: -1234✓ (负数但绝对值未超)   | 64302 | 1      | 0      |
| 输入: -1✓ (负数但绝对值未超)      | 65535 | 1      | 0      |
| 输入: 70000✓ (超上限)        | 65535 | 0      | 1      |

注:

- 1、上面的三个65535中, 有两个不可信 (黄底), 为什么?
- 2、无符号的错误表现与有符号不一样, 规律自行总结 (包括各种整型)



## § 3. 结构化程序设计基础

### 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

上课认真听

下课动手做, 仔细观察并思考课上所讲

```
#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;
}
```

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;
    cin >> k;
    cout << "k=" << k << ' ' << cin.good() << endl;
    return 0;
}
```

Microsoft Visual Studio 调试控制台  
70000  
k=32767 0

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;
    k = 54321; // k = 70000;
    cout << "k=" << k << endl;
    return 0;
}
```

Microsoft Visual Studio 调试控制台  
k=4464

(6,14): warning C4305: "k=": 从 "int" 到 "short" 截断  
(6,9): warning C4309: "k=": 截断常量值





## § 3. 结构化程序设计基础

### 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)

上课认真听

下课动手做，仔细观察并思考课上所讲

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

输入: abc  
输出:  
输入: \n  
输出:  
输入: 'a'  
输出:  
输入: Ctrl+Z  
输出:  
输入: Ctrl+C  
输出:

说明: Ctrl+c表示输入结束, CTRL+c表示终止程序的运行,  
目前只要求了解cin.good()为0即可, 后面有专题

★ 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 << cin.good() << ' ' << f << endl;
    return 0;
}
```

输入:  
123.456789  
输出:  
  
输入: 12e3  
输出:  
  
输入: 1.2e40  
输出:

- 1、good为0不可信
- 2、打印位数与有效位数无关
- 3、打印是正常小数还是科学计数法在输出流中会讲到

★ cin不能跟endl，否则编译错

error C2679: 二元“>>”: 没有找到接受“overloaded-function”类型的右操作数的运算符(或没有可接受的转换)

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. 结构化程序设计基础

### 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;`

★ `cin`读入数据时, 允许输入比期望数据多的数据, 这种情况会将多余的数据保存在缓冲区中, 供下次读取  
(缓冲区未读完之前, 不再从键盘读)

```
#include <iostream>
using namespace std;
int main()
{
    int a, b, c, d;
    cout << "--1--" << endl;
    cin >> a;
    cout << a << endl;
    cout << "--2--" << endl;
    cin >> b;
    cout << b << endl;
    cout << "--3--" << endl;
    cin >> c;
    cout << c << endl;
    cout << "--4--" << endl;
    cin >> d;
    cout << d << endl;
    return 0;
}
```

按下面方式依次执行,  
观察运行现象及结果

1. 每次输入1个数字+回车

--1--后 \_\_\_\_ (停顿/不停顿)  
--2--后 \_\_\_\_ (停顿/不停顿)  
--3--后 \_\_\_\_ (停顿/不停顿)  
--4--后 \_\_\_\_ (停顿/不停顿)

2. 每次输入2个数字+回车

--1--后 \_\_\_\_ (停顿/不停顿)  
--2--后 \_\_\_\_ (停顿/不停顿)  
--3--后 \_\_\_\_ (停顿/不停顿)  
--4--后 \_\_\_\_ (停顿/不停顿)

3. 第一次即输入4个以上数字+回车

--1--后 \_\_\_\_ (停顿/不停顿)  
--2--后 \_\_\_\_ (停顿/不停顿)  
--3--后 \_\_\_\_ (停顿/不停顿)  
--4--后 \_\_\_\_ (停顿/不停顿)  
程序结束时\_\_\_\_还保留在缓冲区中

```
Microsoft
--1--
1
--2--
2
--3--
3
```

```
Microsoft
--1--
1 2
--2--
1 2
--3--
3 4
```

```
Microsoft
--1--
1 2 3 4 5
--2--
2
--3--
3
--4--
4
```

上课认真听  
下课动手做, 仔细观察并思考课上所讲



## § 3. 结构化程序设计基础

### 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;`

★ `cin`读入数据时, 允许输入比期望数据多的数据, 这种情况会将多余的数据保存在缓冲区中, 供下次读取  
(缓冲区未读完之前, 不再从键盘读)

★ `cin`跟多个数据时, 如果到某个数据时`cin.good()`为0/`cin.fail()`为1, 则该值不可信, 后续也不再读取

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

int main()
{
    int a, b, c, d;
    cin >> a;
    cout << cin.good() << ' ' << a << endl;
    cin >> b;
    cout << cin.good() << ' ' << b << endl;
    cin >> c;
    cout << cin.good() << ' ' << c << endl;
    cin >> d;
    cout << cin.good() << ' ' << d << endl;
    return 0;
}
```

1、输入: 1 2 3 4m✓

2、输入: 1 2 3m 4✓

3、输入: 1 2m 3 4✓

4、输入: 1m 2 3 4✓

5、输入: 1 2 3 m✓

6、输入: 1 2 m 4✓

7、输入: 1 m 3 4✓

8、输入: m 2 3 4✓

```
Microsoft
1 2 3m 4
1 1
1 2
1 3
0 0
```

第1个数: 读1, 空格停止, 得1, 正确  
第2个数: 读2, 空格停止, 得2, 正确  
第3个数: 读3, m停止, 得3, 正确  
第4个数: 读m, 得0(不可信), 错误

只给出了例2、7的解释, 其他自行补齐

```
Microsoft Visu
1 m 3 4
1 1
0 0
0 -858993460
0 -858993460
```

第1个数: 读1, 空格停止, 得1, 正确  
第2个数: 读m, 得0(不可信), 错误  
第3个数: 不读, 值随机  
第4个数: 不读, 值随机



## § 3. 结构化程序设计基础

### 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;`

★ `cin`读入数据时, 允许输入比期望数据多的数据, 这种情况会将多余的数据保存在缓冲区中, 供下次读取  
(缓冲区未读完之前, 不再从键盘读)

★ `cin`跟多个数据时, 如果到某个数据时`cin.good()`为0/`cin.fail()`为1, 则该值不可信, 后续也不再读取

思考题: 相似的程序, 为什么`cin.good()`全0? (再次强调: 一定要按程序一步步理解, 不要跳步)

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

int main()
{
    int a, b, c, d;
    cin >> a >> b >> c >> d;
    cout << cin.good() << ' ' << a << endl;
    cout << cin.good() << ' ' << b << endl;
    cout << cin.good() << ' ' << c << endl;
    cout << cin.good() << ' ' << d << endl;

    return 0;
}
```

3、输入: 1 2m 3 4✓

```
Microsoft Visual
1 2m 3 4
0 1
0 2
0 0
0 -858993460
```

上课认真听  
下课动手做, 仔细观察并思考课上所讲



## § 3. 结构化程序设计基础

### 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;`

★ `cin`读入数据时, 允许输入比期望数据多的数据, 这种情况会将多余的数据保存在缓冲区中, 供下次读取  
(缓冲区未读完之前, 不再从键盘读)

★ `cin`跟多个数据时, 如果到某个数据时`cin.good()`为0/`cin.fail()`为1, 则该值不可信, 后续也不再读取

★ `cin`输入时, `Ctrl+c`表示终止程序运行, 之后的所有输出均不可信

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

int main()
{
    int a, b;
    cin >> a;
    cout << cin.good() << ' ' << a << endl;
    cin >> b;
    cout << cin.good() << ' ' << b << endl;

    return 0;
}
```

1、输入: `Ctrl+c`✓  
(表示按`Ctrl+c`组合键, 注意不要有输入法栏, 下同)

2、输入: `123Ctrl+c`✓  
(123不加空格, 直接按`Ctrl+c`)

3、输入: `123 Ctrl+c`✓  
(123加空格, 再按`Ctrl+c`)

4、输入: `123 456 Ctrl+c`✓  
(456加空格, 再按`Ctrl+c`)

输入 输出

这是任意两次的执行结果,  
从无输出到“0 -85xxxxxx”均可能,  
但不可信

只给出了例3的运行截图  
本例一定要每种输入多试几次!!!



## § 3. 结构化程序设计基础

### 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;`

★ `cin`读入数据时, 允许输入比期望数据多的数据, 这种情况会将多余的数据保存在缓冲区中, 供下次读取  
(缓冲区未读完之前, 不再从键盘读)

★ `cin`跟多个数据时, 如果到某个数据时`cin.good()`为0/`cin.fail()`为1, 则该值不可信, 后续也不再读取

★ `cin`输入时, `Ctrl+c`表示终止程序运行, 之后的所有输出均不可信

★ `cin`输入时, `Ctrl+z`表示输入结束, 后续数据均不可信

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

int main()
{
    int a, b;
    cin >> a;
    cout << cin.good() << ' ' << a << endl;
    cin >> b;
    cout << cin.good() << ' ' << b << endl;

    return 0;
}
```

1、输入: `Ctrl+z` ✓  
(表示按`Ctrl+z`组合键, 注意  
不要有输入法栏, 下同)

2、输入: `Ctrl+z123` ✓

3、输入: `Ctrl+z123 456` ✓

4、输入: `123Ctrl+z456` ✓

5、输入: `123 456Ctrl+z` ✓

```
Microsoft Visual Studio 调试控制台
^Z123 456
0 -858993460
0 -858993460
```

读到`Ctrl+z`后, 置`cin.good()`为0,  
不再读取任何数据

只给出了例3的运行截图  
上课认真听  
下课动手做, 仔细观察并思考课上所讲



## § 3. 结构化程序设计基础

### 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

| 控制符                         | 作用                                                                              |
|-----------------------------|---------------------------------------------------------------------------------|
| dec                         | 设置整数为10进制                                                                       |
| hex                         | 设置整数为16进制                                                                       |
| oct                         | 设置整数为8进制                                                                        |
| setbase(n)                  | 设置整数为n进制 (n=8, 10, 16)                                                          |
| setfill(c)                  | 设置填充字符，c可以是字符常量或字符变量                                                            |
| setprecision(n)             | 设置实数的精度为n位。在以一般十进制形式输出时，n代表有效数字。<br>在以fixed(固定小数位)形式和scientific(指数)形式输出时，n为小数位数 |
| setw(n)                     | 设置字段宽度为n                                                                        |
| setiosflags ios::fixed      | 设置浮点数以固定的小数位数显示                                                                 |
| setiosflags ios::scientific | 设置浮点数以科学计数法（即指数形式）显示                                                            |
| setiosflags ios::left       | 输出数据左对齐                                                                         |
| setiosflags ios::right      | 输出数据右对齐                                                                         |
| setiosflags ios::skipws     | 忽略前导的空格                                                                         |
| setiosflags ios::uppercase  | 在以科学计数法输出E和十六进制输出字母X时，以大写表示                                                     |
| setiosflags ios::showpos    | 输出正数时，给出“+”号                                                                    |
| resetiosflags(*)            | 终止已设置的输出格式状态，括号内为具体内容(本处用*替代)                                                   |

★ 常用格式控制符见上表



## § 3. 结构化程序设计基础

### 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

★ 格式控制符有两类

- 设置后始终有效，直到用另一个同类设置取代为止
- 设置后仅一次性有效，需要时进行重复设置即可

```
#include <iostream>
#include <iomanip> //使用格式化输出需加此头文件
using namespace std;
```

```
int main()
{
    int a=123, b=456;
    cout << hex << "0x" << a << " 0x" << b << endl;
    cout << dec;
    cout << '*' << setw(10) << b << ',' << b << '*' << endl; //加*的目的?

    return 0;
}
```

hex : 进制转换符，始终有效  
setw(): 设置宽度，仅一次性有效

★ 很多细节的内容，通过作业来掌握





## § 3. 结构化程序设计基础

### 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');
```

这四个都在屏幕上输出A

上课认真听

下课动手做，仔细观察并思考课上所讲

★ 加#include <cstdio>或#include <stdio.h>

(目前两编译器均可不要)

★ 返回值是int型，是输出字符的ASCII码，可赋值给字符型/整型变量

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

```
int main()  
{  
    char ret1;  
    cout << (ret1 = putchar('A')) << endl;  
    int ret2;  
    cout << (ret2 = putchar('B')) << endl;  
    return 0;  
}
```

为什么是这个输出？想明白！

写测试程序验证以下问题：如何证明putchar()的返回值是int而不是char？是否有多种方法？

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

```
int main()  
{  
    cout << putchar('A') << endl;           //方法1  
    cout << sizeof( putchar('A') ) << endl;   //方法2-1  
    cout << typeid( putchar('A') ).name() << endl; //方法2-2  
    return 0;  
}
```

引申问题：为什么方法1输出了A/执行了putchar？方法2未输出A/未执行putchar？



## § 3. 结构化程序设计基础

### 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>

(目前两编译器均可不要)

★ 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;
}
```

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

左右程序

- 1、假设键盘输入: a, 则输出: ?
- 2、左右蓝色框中语句是否等价?

写测试程序验证以下问题:

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

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

上课认真听

下课动手做, 仔细观察并思考课上所讲



## § 3. 结构化程序设计基础

### 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

上课认真听

下课动手做, 仔细观察并思考课上所讲

```
#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 (前面cin时用int演示过, 同理)

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. 结构化程序设计基础

### 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

上课认真听

下课动手做，仔细观察并思考课上所讲

```
#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Dev C++中

getch() ⇔ \_getch()

getche() ⇔ \_getche()



## § 3. 结构化程序设计基础

### 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: 1、熟练使用C++的cin/cout

2、能看懂C的printf/scanf

3、除非明确要求, C++作业不允许使用printf/scanf



## § 3. 结构化程序设计基础

### 3.5. 编写顺序结构的程序

例：求一元二次方程的根（上：C++方式，下：C方式）

```
#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是系统提供的开方函数，需包含<math.h>或<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

```
Microsoft Visual Studio 调试控制台
1 -3 1
x1=2.61803
x2=0.381966
```

```
Microsoft Visual Studio 调试控制台
1 -2 1
x1=1
x2=1
```

```
Microsoft Visual Studio 调试控制台
1 1 1
x1=-nan(ind)
x2=-nan(ind)
```

```
#define _CRT_SECURE_NO_WARNINGS //C方式使用scanf需要加
#include <stdio.h> //C方式的基本输入输出头文件
#include <math.h> //C方式不能是cmath
```

```
int main()
{
    double a, b, c, x1, x2;
    scanf("%lf %lf %lf", &a, &b, &c); //输入方式不同
    x1 = (-b + sqrt(b*b-4*a*c)) / (2*a);
    x2 = (-b - sqrt(b*b-4*a*c)) / (2*a);
    printf("x1=%f\n", x1); //输出方式不同
    printf("x2=%f\n", x2); //输出方式不同
    return 0;
}
```



## § 3. 结构化程序设计基础

### 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;
}
```

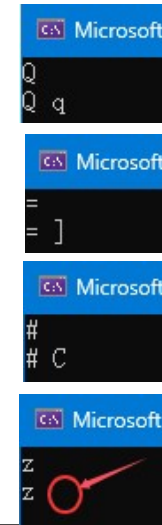
c1=getchar();  
scanf("%c", &c1);

c2 = c1 + 'a' - 'A';

printf("%c %c", c1, c2);

cout << c1 << ' ' << c2 << endl;

- 1、执行时无提示即等待输入
- 2、若输入的不是大写字母出错
- 3、为什么输入z没有输出？



上课认真听  
下课动手做，仔细观察并思考课上所讲



## § 3. 结构化程序设计基础

### 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：

< <= > >= 优先级相同（优先级第8组）

== != 优先级相同（优先级第9组）

优先级和结合性：附录D

##### 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 //比较算术表达式和常量的值

(a=20) > (b=30) //比较两个赋值表达式的值

关系表达式的值：

“真” - 1

“假” - 0





## § 3. 结构化程序设计基础

### 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

```
int a=1, b=2, c;  
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;  
}
```

```
#include <iostream>  
using namespace std;  
int main()  
{  
    int a=1, b=2, c=3, d;  
    d = a>b>c;  
    cout << d << endl;  
    d = a<b<c;  
    cout << d << endl;  
    d = b>a<c;  
    cout << d << endl;  
    return 0;  
} //VS下有三个warning
```

Microsoft  
0  
1  
1

```
#include <iostream>  
using namespace std;  
int main()  
{  
    int a=3, b=2, c=1, d;  
    d = a>b>c;  
    cout << d << endl;  
    d = a<b<c;  
    cout << d << endl;  
    d = b>a<c;  
    cout << d << endl;  
    return 0;  
} //VS下有三个warning
```

Microsoft  
0  
1  
1

d=1  
d=0  
d=0

### 1、左右两个程序编译时均有warning

“>”：在操作中使用类型“bool”不安全  
“<”：在操作中使用类型“bool”不安全  
“<”：在操作中使用类型“bool”不安全

### 2、为什么左右结果一样？

### 3、a>b>c 正确的求解方式是什么？

### 4、C/C++的实际求值过程是怎样的？

注：一定要搞明白，为什么右侧程序的运行结果不是蓝色!!!



## § 3. 结构化程序设计基础

### 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

★ 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. 结构化程序设计基础

### 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

//第2章 IEEE754 作业中的例子

例1: float f=100.25;

32bit机内表示 0100 0010 1100 1000 1000 0000 0000 0000

尾数是 100 1000 1000 0000 0000 0000

尾数转换为十进制小数是 0.56640625

尾数表示的十进制小数是 1.56640625 (加整数部分的1后)

$1.56640625 \times 2^6 = 100.25$  (未体现出误差)

例2: float f = 1234567.7654321;

32bit机内表示 0100 1001 1001 0110 1011 0100 0011 1110

尾数是 001 0110 1011 0100 0011 1110

尾数转换为十进制小数形式是 0.1773755503845214844

尾数表示的十进制小数形式是 1.1773755503845214844 (加1后)

$1.1773755503845214844 \times 2^{20} = 1234567.75$  (体现出误差)

```
#include <iostream>
using namespace std;
int main()
{
    float f1=100.25;
    cout << (f1-100.25) << endl;
    cout << (f1==100.25) << endl;

    float f2=1234567.7654321;
    cout << (f2-1234567.7654321) << endl;
    cout << (f2==1234567.7654321) << endl;
    return 0;
}
```

demo.cpp(9,28): warning C4305: “初始化”: 从“double”到“float”截断

第2章 IEEE754 作业中的例子

- 1、为什么第5行double赋值给float无warning, 但第9行有?
- 2、如何理解==的结果是 1/0 ?
- 3、在不了解IEEE754的情况下, 是否有通用的方法避免误差?

Microsoft Visual Studio 调试控制台

```
0
1
-0.0154321
0
```

```
#include <iostream>
#include <cmath> //fabs需要, VS可不加
using namespace std;
int main()
{
    float f1=100.25;
    cout << (fabs(f1-100.25) < 1e-6) << endl;

    float f2=1234567.7654321;
    cout << (fabs(f2-1234567.7654321) < 1e-1) << endl;
    cout << (fabs(f2-1234567.7654321) < 1e-2) << endl;
    return 0;
}
```

fabs是求绝对值的系统函数

通用方法:

当两数相减的绝对值小于某个值则认为相等

Microsoft

```
1
1
0
```

问: 为什么cout时要加红色的一对括号?



## § 3. 结构化程序设计基础

### 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

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> //VS可不加
using namespace std;
int main()
{
    float b = 1.1; //有warning
    cout << (b - 1.1) << endl;           2.38419e-08
    cout << (b == 1.1) << endl;           0
    cout << (fabs(b - 1.1) < 1e-6) << endl; 1

    float c = 1.0; //无warning
    cout << (c - 1.0) << endl;           0
    cout << (c == 1.0) << endl;           1
    cout << (fabs(c - 1.0) < 1e-6) << endl; 1

    return 0;
}
```

```
#include <iostream>
#include <cmath> //VS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;
} //有warning
```

warning C4305: “初始化”: 从“double”到“float”截断  
warning C4305: “初始化”: 从“double”到“float”截断

1、为什么不等为数判断== 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. 结构化程序设计基础

### 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

结论:

- 实数输出时可以任意指定位数, 但超过有效位数限定的值不可信

```
#include <iostream>
#include <iomanip>
#include <cmath> //VS可不加
using namespace std;

int main()
{
    cout << setiosflags(ios::fixed); //指定fixed输出

    double d1 = 123.456789012345678;
    cout << setprecision(15) << d1 << endl;
    cout << setprecision(20) << d1 << endl;

    float f1 = 123.456789012345678;
    cout << setprecision(15) << f1 << endl;
    cout << setprecision(20) << f1 << endl;

    cout << endl;

    double d2 = 123.456789123456789;
    cout << setprecision(15) << d2 << endl;
    cout << setprecision(20) << d2 << endl;

    float f2 = 123.456789123456789;
    cout << setprecision(15) << f2 << endl;
    cout << setprecision(20) << f2 << endl;

    return 0;
} //VS有两个warning
```

```
warning C4305: “初始化”: 从“double”到“float”截断
warning C4305: “初始化”: 从“double”到“float”截断
```

Microsoft Visual Studio 调试控制台

```
123.456789012345681
123.45678901234568058953
123.456787109375000
123.456787109375000000000

123.456789123456787
123.45678912345678668316
123.456787109375000
123.456787109375000000000
```



## § 3. 结构化程序设计基础

### 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. 结构化程序设计基础

### 3.6. 关系运算和逻辑运算

#### 3.6.2. 逻辑常量和逻辑变量 (C++特有, C无)

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 << cin.good() << ' ' << k << ' ' << (int)k << endl;
```

```
    return 0;
```

```
}
```

输入:0      输出: 1 0 0

1            1 1 1

123          0 1 1

true        0 0 0

false       0 0 0

这三种都是错误的输入方法  
不同编译器下表现可能不同,  
虽然值看似符合预期, 但不可信

```
#include <iostream>
using namespace std;
int main()
{
    bool f;

    cout << typeid(bool).name() << endl;
    cout << sizeof(bool) << endl;
    cout << sizeof(f) << endl;
    cout << sizeof(true) << endl;
    cout << sizeof(false) << endl;
    cout << sizeof("true") << endl;
    cout << sizeof("false") << endl;
    return 0;
}
```

最后两项是什么?

```
bool
1
1
1
1
5
6
```



## § 3. 结构化程序设计基础

### 3.6. 关系运算和逻辑运算

#### 3.6.2. 逻辑常量和逻辑变量 (C++特有, C无)

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

★ 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
```

256 = 0..01 00000000 (4字节)  
即低8bit为0  
因此, 本例证明bool赋值时  
不是多字节赋少字节的规则,  
而是“非0为真0为假”

★ C方式 (源程序后缀为.c) 加 #include <stdbool.h>后,  
也可使用bool/true/false

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

int main()
{
    bool k;
    k=123;
    cout << k << ' ' << (int)k << endl;
    k=0;
    cout << k << ' ' << (int)k << endl;
    k=256;
    cout << k << ' ' << (int)k << endl;
    return 0;
}
```

warning C4305: “=”: 从“int”到“bool”截断  
warning C4305: “=”: 从“int”到“bool”截断





## § 3. 结构化程序设计基础

### 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

|    |      |                    |
|----|------|--------------------|
| && | A&&B | 均为真，取值为真（优先级第13组）  |
|    | A  B | 有一个为真，值为真（优先级第14组） |
| !  | !A   | 取反（单目运算符）（优先级第3组）  |

优先级与结合性：附录D

##### 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

逻辑运算的真值表（取值时）：

| 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    |

逻辑运算的真值表（表达式参与运算时）：

| 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    |



## § 3. 结构化程序设计基础

### 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

1、按优先级结合性得到的求解顺序

2、再结合本文档下页的短路运算

得到的求解顺序

问：应该是哪个？如何验证？



## § 3. 结构化程序设计基础

### 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;
}
```

|       |       |
|-------|-------|
| $m=1$ | $n=1$ |
| $m=0$ | $n=1$ |



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. 结构化程序设计基础

### 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. 4. 按位与/或运算

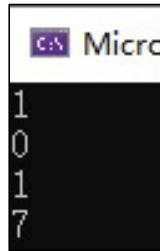
附录D:

按位与: & 两数按对应bit位与, 对应bit位均为1则取值为1(真) (优先级第10组)

按位或: | 两数按对应bit位或, 对应bit位有1个为1则值为1(真) (优先级第12组)

```
#include <iostream>
using namespace std;
int main()
{
    short a=3, b=4;
    cout << (a && b) << endl;
    cout << (a & b) << endl; //按位&
    cout << (a || b) << endl;
    cout << (a | b) << endl; //按位|

    return 0;
}
```



```
0000 0000 0000 0011 (a)
& 0000 0000 0000 0100 (b)
-----
0000 0000 0000 0000 (0)
```

```
0000 0000 0000 0011 (a)
| 0000 0000 0000 0100 (b)
-----
0000 0000 0000 0111 (7)
```

常见错误:  
&&/|| 写成&/|, 某些情况下也表现正确

本例: a||b写成a|b, 结果均为逻辑真

问题:

1、除&和|外, 是否还有其它位运算符?

还有 ^ (异或) ~ (取反) >> (右移) << (左移)

2、两数长度不等时(例: short | int), 该如何运算?

低位对齐, 高位补符号位/0

★ 不再展开, 后续面向对象等课程再学习, 有兴趣可以自行了解



## § 3. 结构化程序设计基础

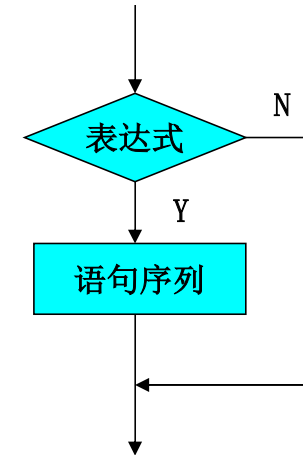
### 3.7. 选择结构和if语句

#### 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;  
}
```

```
1  #include <iostream>  
2  using namespace std;  
3  
4  int main()  
5  {  
6      int i;  
7      cout << "请输入成绩(0-100)" << endl;  
8      cin >> i;  
9      if (i < 60;) {  
10         cout << "不及格" << endl;  
11     }  
12     cout << "程序结束" << endl;  
13     return 0;  
14 }  
15
```

```
warning C4552: "<" : 未使用表达式结果  
error C2429: 语言功能 "if/switch 中的 init-statement" 需要编译器标志 "/std:c++17"  
error C2059: 语法错误: ")"  
error C2143: 语法错误: 缺少 ";" (在 "{" 的前面)
```

- ★ 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; 输入:37 输出:不及格
    return 0;                    程序结束
}
```

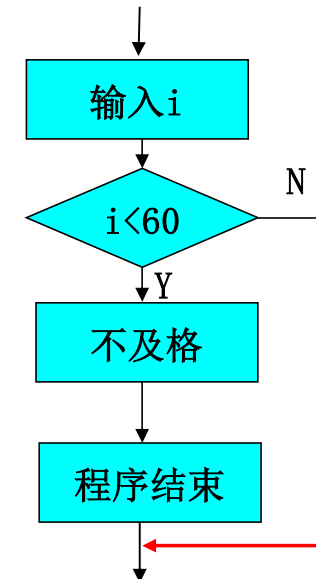
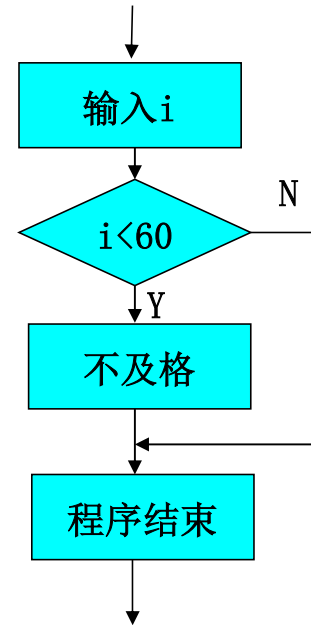
输入:74 输出:程序结束

```
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;
}
```

注意：括号位置对程序正确性影响很大

输入:37 输出:不及格  
程序结束

输入:74 输出: /





## § 3. 结构化程序设计基础

### 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

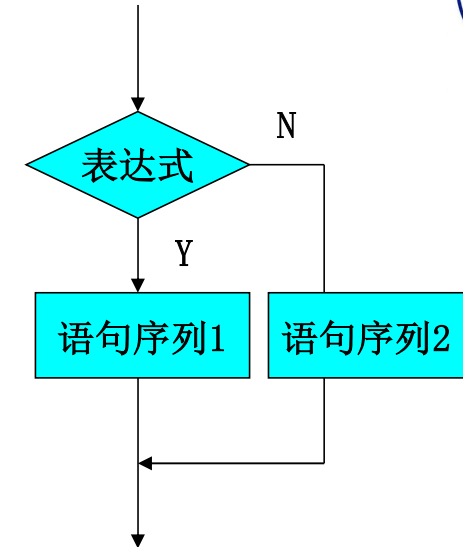
##### 3.7.1.2. 双支语句

```
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



```
1  #include <iostream>  
2  using namespace std;  
3  
4  int main()  
5  {  
6      int i;  
7      cout << "请输入成绩(0-100)" << endl;  
8      cin >> i;  
9      if (i < 60) {  
10         cout << "不及格" << endl;  
11     }  
12     cout << "Hello" << endl;  
13     else {  
14         cout << "及格" << endl;  
15     }  
16     cout << "程序结束" << endl;  
17     return 0;  
18 }  
19
```

人的思维：整体看，cout插在if-else中间，cout多余  
编译器思维：从上到下看，单if、并列的cout，因此else非法





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. 结构化程序设计基础

### 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

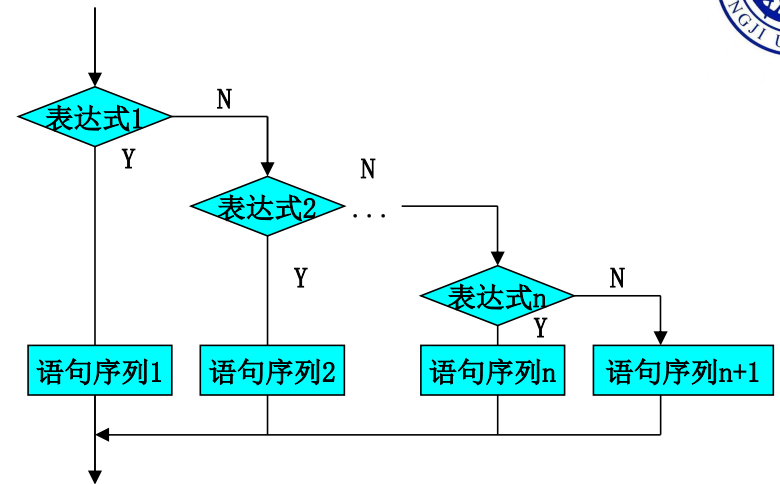
##### 3.7.1.3. 多支语句

- ★ 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？

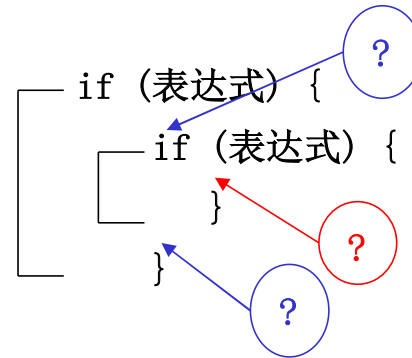
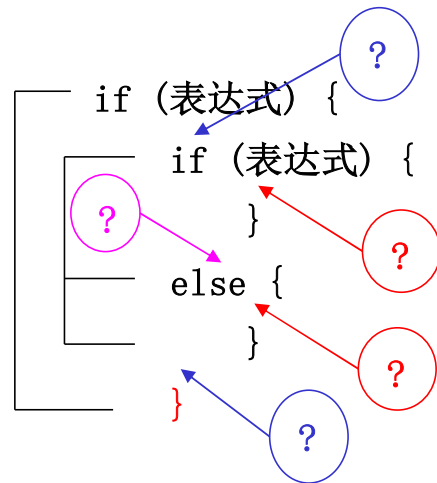
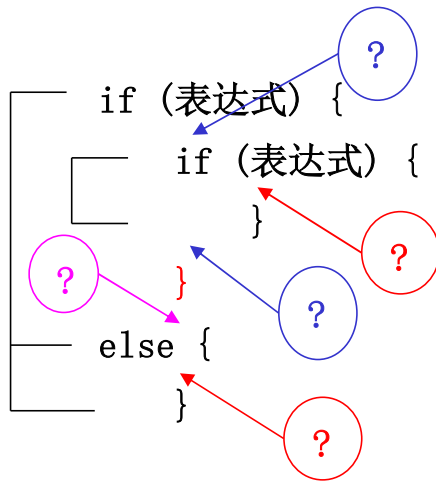


## § 3. 结构化程序设计基础

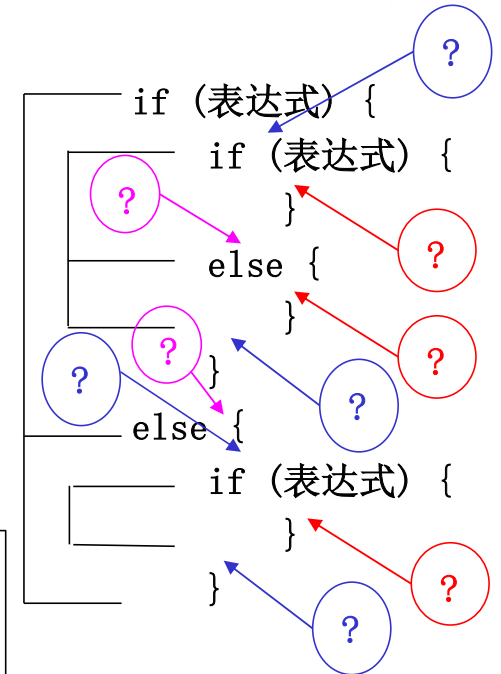
### 3.7. 选择结构和if语句

#### 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:

```
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

遇3, 2出, 匹配





例:

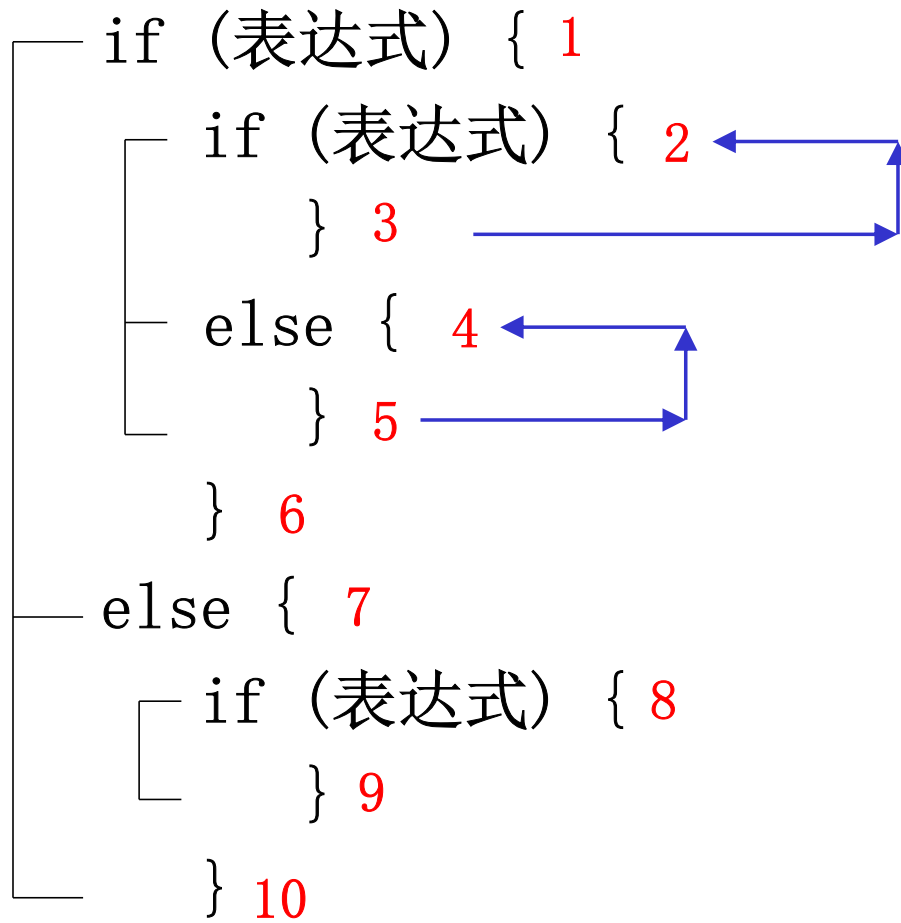
```
if (表达式) { 1
  if (表达式) { 2
    } 3
  else { 4
    } 5
  } 6
else { 7
  if (表达式) { 8
    } 9
  } 10
```

4  
1

4进栈



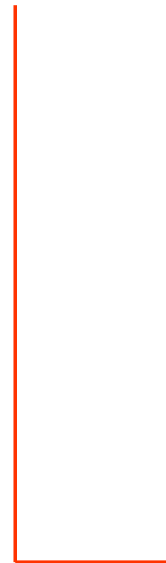
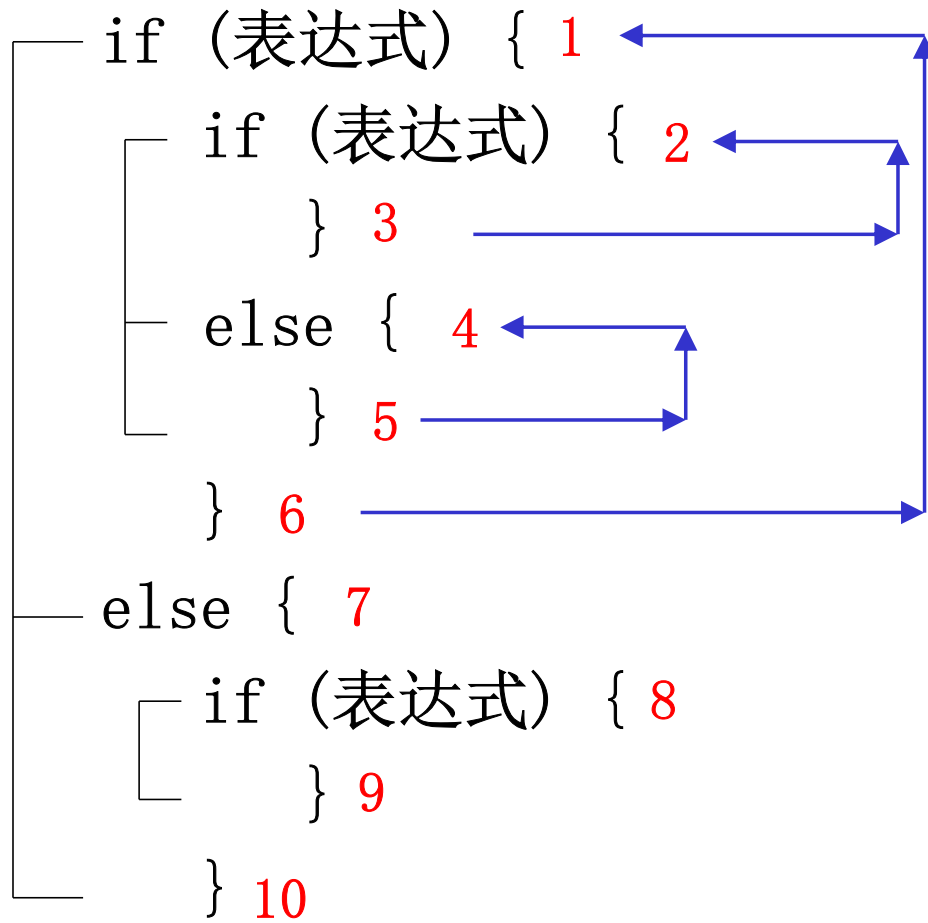
例:



遇5, 4出, 匹配



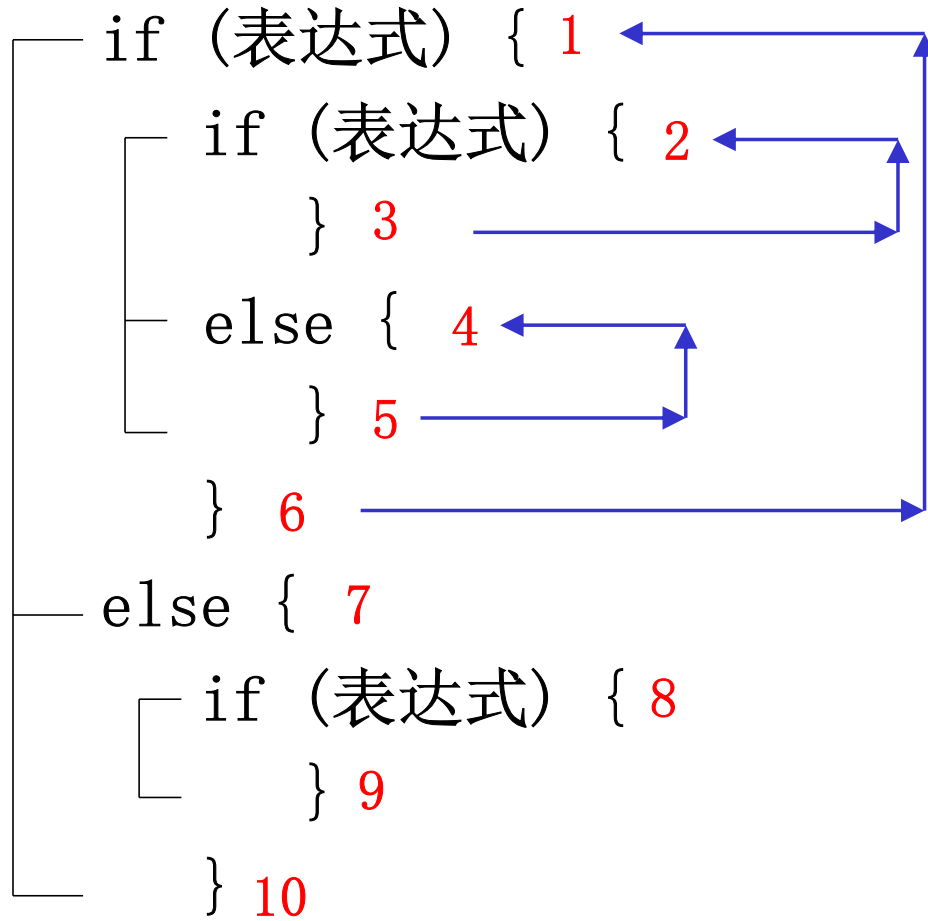
例:



遇6, 1出, 匹配



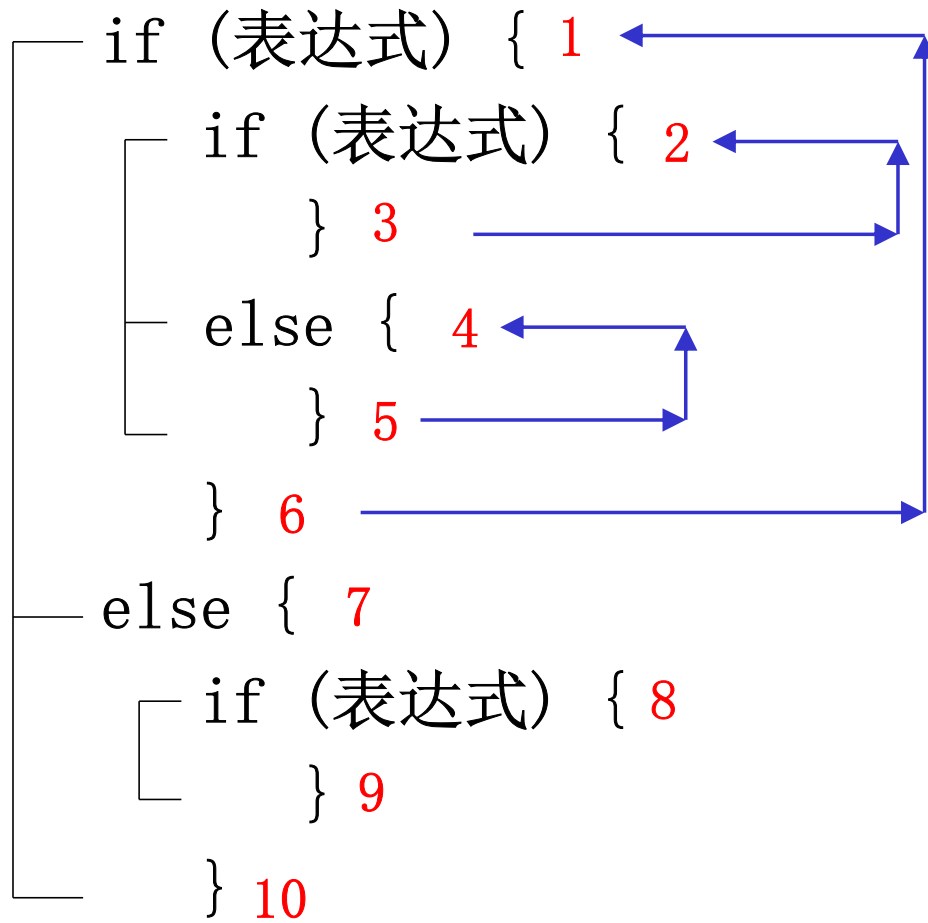
例:



7进栈



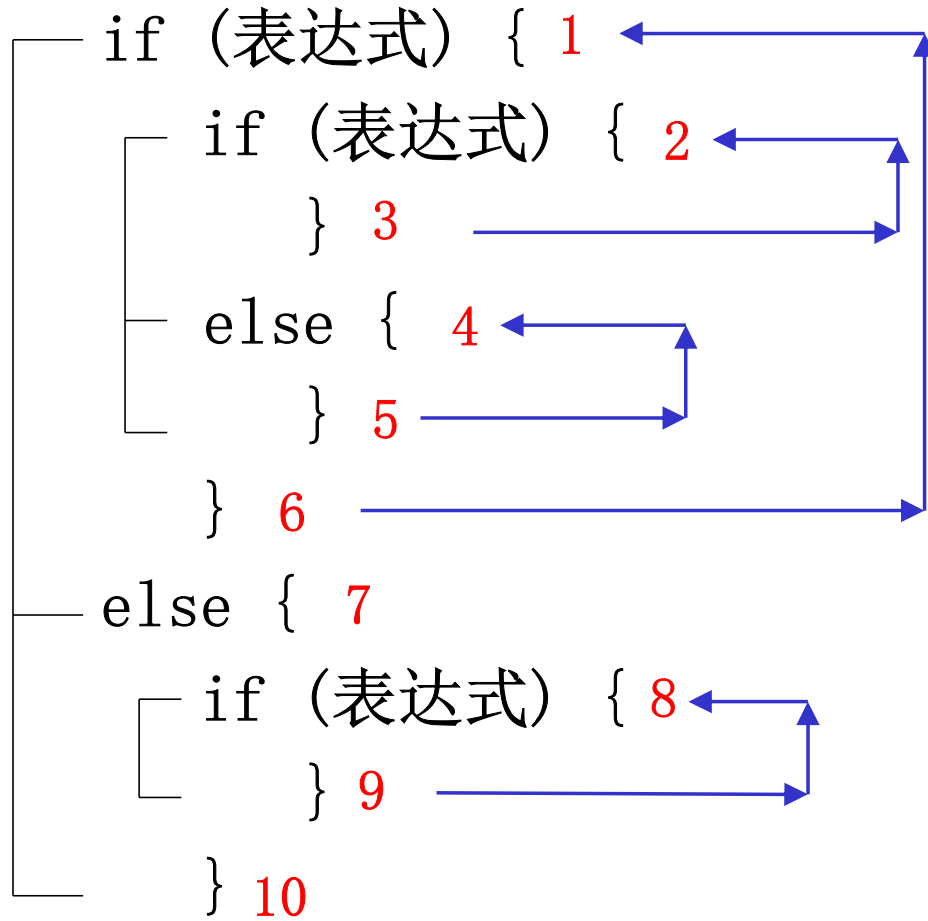
例:



8进栈



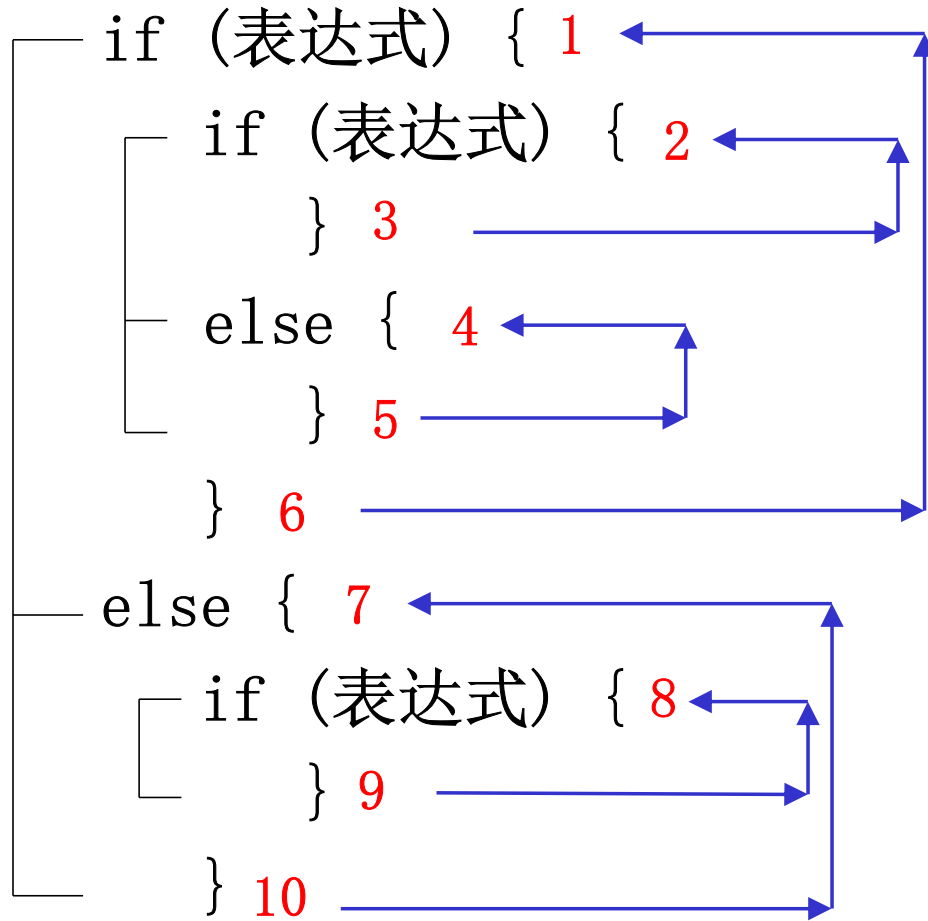
例:



遇9, 8出, 匹配



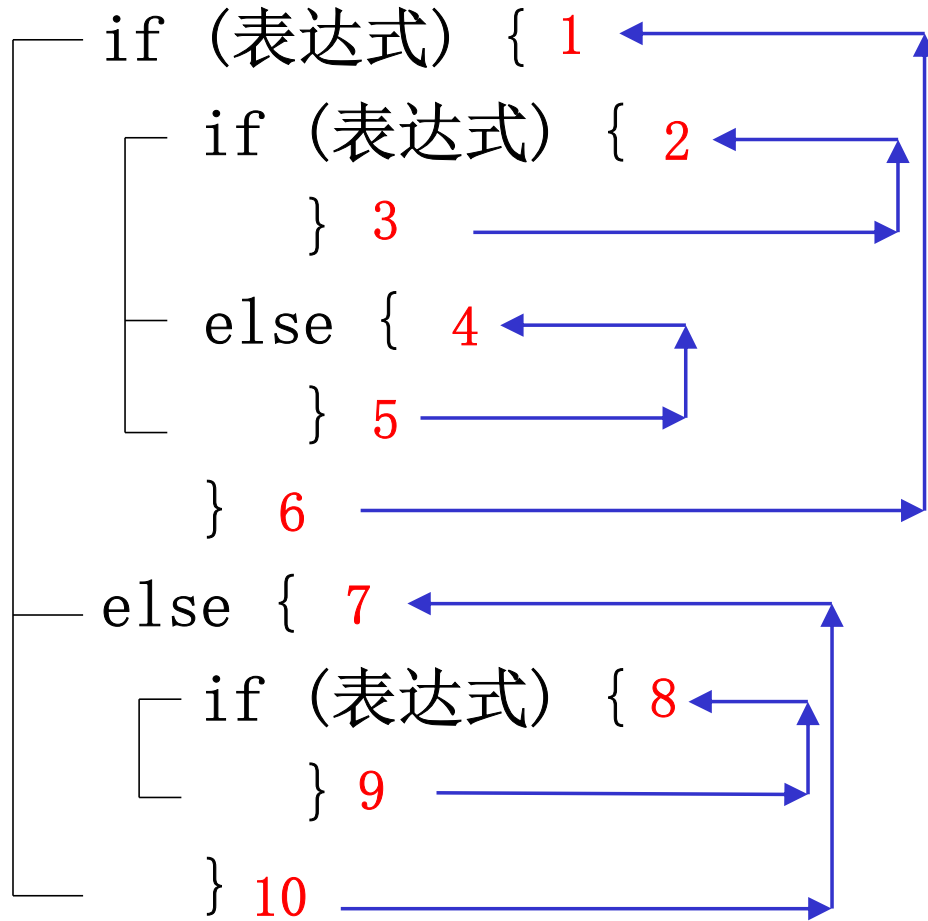
例:



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”

```
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. 结构化程序设计基础

### 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(附录D: 优先级第15组)

★ 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

```
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);
```

```
#include <iostream>
using namespace std;
int main()
{
    int a = 1, b = 2;

    //编译报错
    a == 1 ? "Hello" : 123;

    //编译报错
    a > b ? cout << a : printf("%d", b);

    //编译正确
    a == 1 ? 'A' : 123;
    return 0;
}
```

★ 表达式1、2、3的类型可以不同，但2、3的类型必须相容  
(否则无法确定条件表达式的值类型)

```
error C2446: “.”: 没有从“int”到“const char [6]”的转换
message : 从整型强制转换为指针类型要求 reinterpret_cast、C 样式强制转换或函数样式强制转换
error C2678: 二进制“?”: 没有找到接受“std::basic_ostream<char, std::char_traits<char>>”类型的左操作数的运算符(或没有可接受的转换)
message : 可能是“内置 C++ operator?(int, int)”
message : 尝试匹配参数列表“(std::basic_ostream<char, std::char_traits<char>>, int)”时
```



## § 3. 结构化程序设计基础

### 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

```
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;  
}
```

##### 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{}

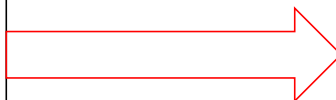
★ 多个case可以共用一组语句

★ 不能完全替代多重if语句的嵌套

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. 结构化程序设计基础

### 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

★ 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

```
double x = **;  
switch(int(x)) {  
  
    ...  
}
```

```
switch(i/10) { 正确  
    case 10:  
    case 9:  
        cout<<"优"<<endl;  
        break;  
    case 8:  
        cout<<"良"<<endl;  
        break;  
    ...  
}
```

```
switch(i/10) { 正确  
    case 8:  
        cout<<"良"<<endl;  
        break;  
    case 10:  
    case 9:  
        cout<<"优"<<endl;  
        break;  
    ...  
}
```

```
const int k = 1; 正确  
  
switch(i/10) {  
    case 3+7:  
    case 8+k:  
        cout<<"优"<<endl;  
        break;  
    ...  
}
```

★ 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(但不建议)

```
switch(i/10) {  
    ...  
    case 8:  
        cout << "良" << endl;  
    case 7:  
        cout << "中" << endl;  
        break;  
    ...  
}
```

```
Microsoft Visual Studio 调试控制台  
请输入成绩(0-100)  
83  
良  
中  
程序结束
```

```
Microsoft Visual Studio 调试控制台  
请输入成绩(0-100)  
78  
中  
程序结束
```

```
const int k = 1; 错误  
  
switch(i/10) {  
    case 8:  
    case 7+k:  
        cout<<"优"<<endl;  
        break;  
    ...  
}
```

error C2196: case 值“8”已使用

```
int k = 1; 错误  
  
switch(i/10) {  
    case 3+7:  
    case 8+k:  
        cout<<"优"<<endl;  
        break;  
    ...  
}
```

error C2131: 表达式的计算结果不是常数  
message : 因读取超过生命周期的变量而失败  
message : 请参见“k”的用法  
error C2051: case 表达式不是常量



## § 3. 结构化程序设计基础

### 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. 结构化程序设计基础

### 3.7. 选择结构和if语句

#### 3.7.5. 编写选择结构的程序

例：键盘输入一个整数当年份，判断该年是否为闰年

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

int main()
{
    int year;
    bool leap;
    cin >> year;
    if (year%4==0) {
        if (year%100==0) {
            if (year%400==0)
                leap=true; ——— 被4、100、400整除
            else
                leap=false;
        }
        else
            leap=true; ——— 被4、100整除，不被400整除
    }
    else
        leap=false; ——— 不被4整除

    if (leap)
        cout << year << " is a leap year" << endl;
    else
        cout << year << " is not a leap year" << endl;
    return 0;
}
```

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

int main()
{
    int year;
    bool leap;
    cin >> year;

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

    if (leap)
        cout << year << " is a leap year" << endl;
    else
        cout << year << " is not a leap year" << endl;
    return 0;
}
```

可以省略两组括号





## § 3. 结构化程序设计基础

### 3.7. 选择结构和if语句

#### 3.7.5. 编写选择结构的程序

例：键盘输入一个整数当年份，判断该年是否为闰年

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

```
int main()
{
```

```
    int year;
    bool leap;
    cin >> year;
```

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

```
    if (leap)
```

```
        cout << year << " is a leap year" << endl;
```

```
    else
```

```
        cout << year << " is not a leap year" << endl;
```

```
    return 0;
```

```
}
```

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

```
int main()
{
```

```
    int year;
    bool leap;
    cin >> year;
```

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

改写1

```
    else
        leap=false;
```

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

改写2

```
    if (leap)
```

```
        cout << year << " is a leap year" << endl;
```

```
    else
```

```
        cout << year << " is not a leap year" << endl;
```

```
    return 0;
```

```
}
```

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");
```



## § 3. 结构化程序设计基础

### 3.7. 选择结构和if语句

#### 3.7.5. 编写选择结构的程序

例：键盘输入一个整数当年份，判断该年是否为闰年

```
//最简版本
#include <iostream>
using namespace std;

int main()
{
    int year, leap;
    cin >> year;

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

    cout << year << (leap ? "is" : "is not") << " a leap year."<<endl;

    return 0;
}
```

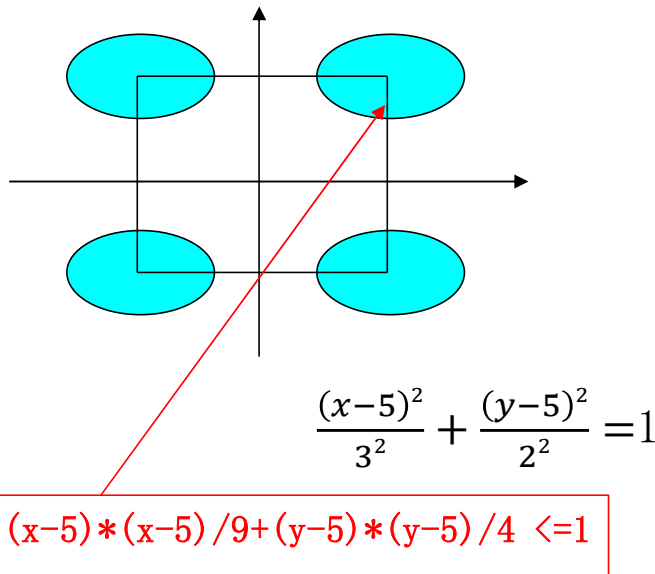


## § 3. 结构化程序设计基础

### 3.7. 选择结构和if语句

#### 3.7.5. 编写选择结构的程序

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



```
#include <iostream>
using namespace std;
int main()
{
    double x, y;           //尽量别用int
    cout << "请输入坐标: "; //不加endl, 则光标停在本行等待输入
    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;
}
```

注意:  
复制粘贴时要改对



## § 3. 结构化程序设计基础

### 3.7. 选择结构和if语句

#### 3.7.5. 编写选择结构的程序

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、垃圾格式

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



## § 3. 结构化程序设计基础

### 3.7. 选择结构和if语句

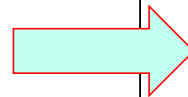
#### 3.7.5. 编写选择结构的程序

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;
}
```



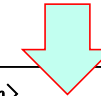
```
#include <iostream>
using namespace std;
int main()
{
    int n, k;
    cin >> n >> k;

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

    return 0;
}
```

限制：  
1、不允许用if-else，如何实现？

两种方法

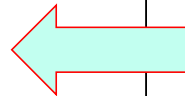


```
#include <iostream>
using namespace std;
int main()
{
    int n, k;
    cin >> n >> k;

    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  
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 + (n%k > 0) << endl;
    cout << n/k + bool(n%k) << endl;
    cout << n/k + !(n%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？

三种方法

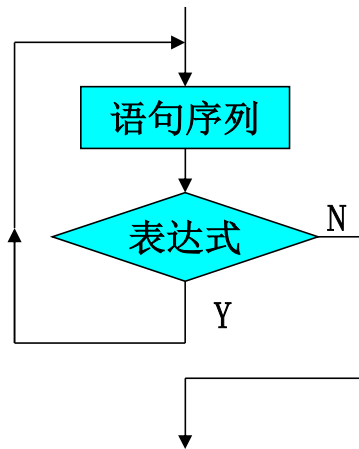
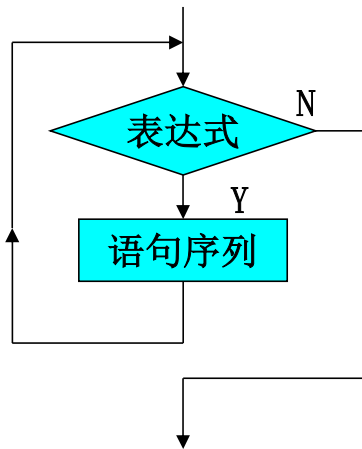


## § 3. 结构化程序设计基础

### 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. 结构化程序设计基础

### 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)

强调:

★ 学习goto的目的, 是了解, 读别人程序时能看懂

★ 本课程禁止在任何时候、任何方式下使用goto语句

```
#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. 结构化程序设计基础

### 3.8. 循环结构和循环语句

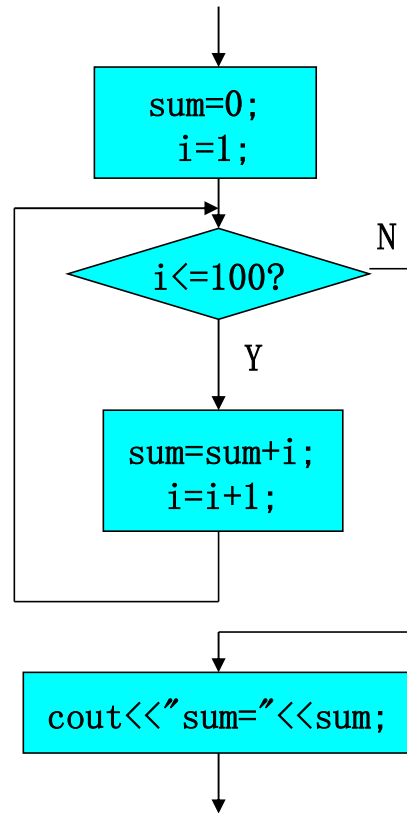
#### 3.8.1. GOTO语句

与if语句一起构成循环:

例: 求 $1+2+\dots+100$ 的和

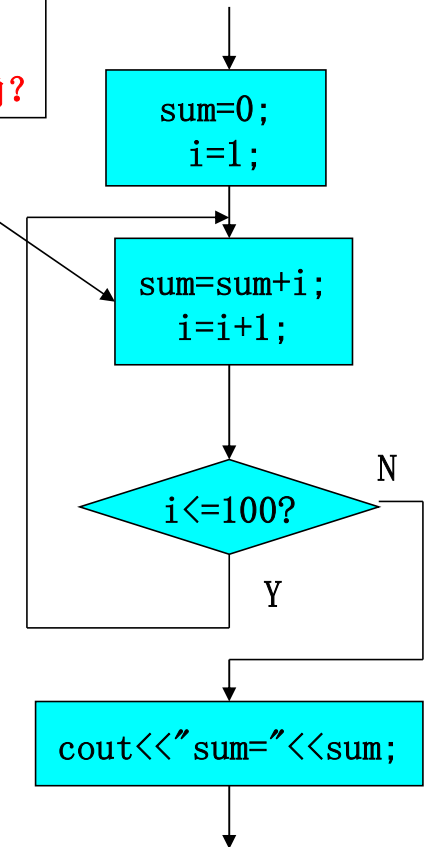
1、当型

```
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;
}
```



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;
}
```



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





## § 3. 结构化程序设计基础

### 3.8. 循环结构和循环语句

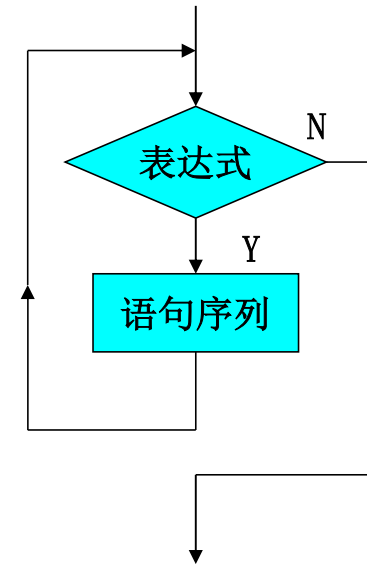
#### 3.8.2. 用while语句构成循环

##### 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;
```

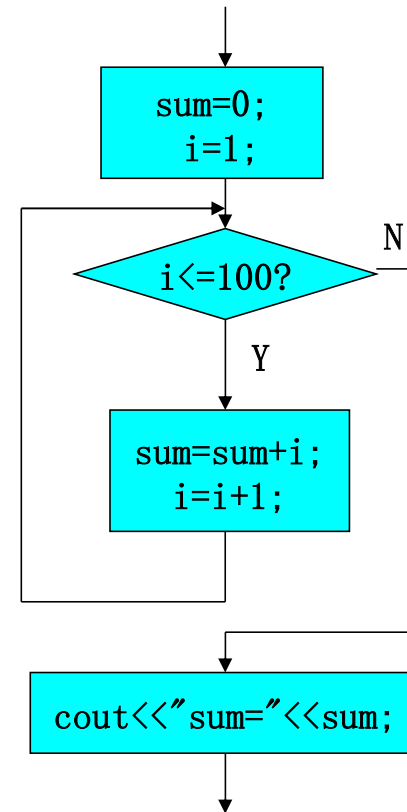
```
    while(i<=100) {
        sum=sum+i;
        i++;
    }
```

```
    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++;
    }

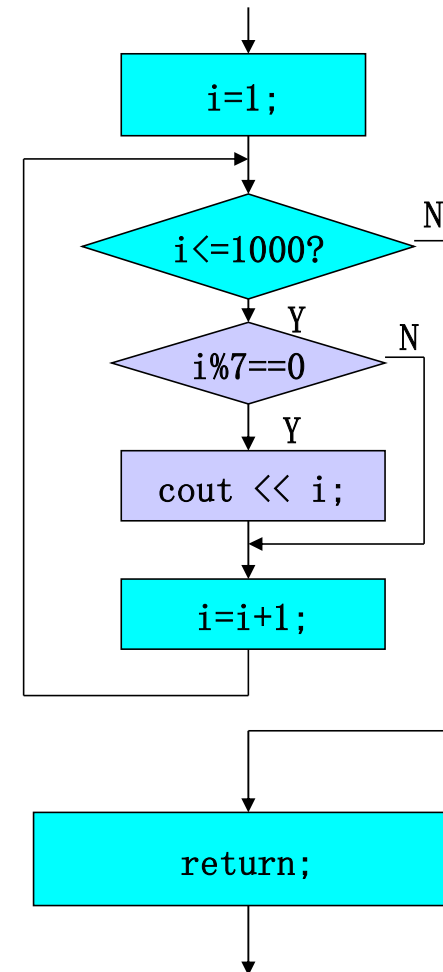
    return 0;
}
```

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

int main()
{
    int i=7;

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

    return 0;
}
```





## § 3. 结构化程序设计基础

### 3.8. 循环结构和循环语句

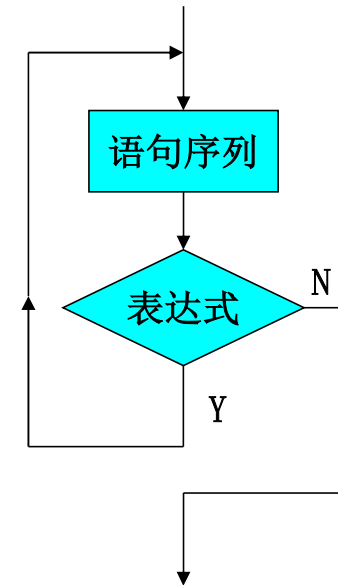
#### 3.8.3. 用do-while语句构成循环

##### 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

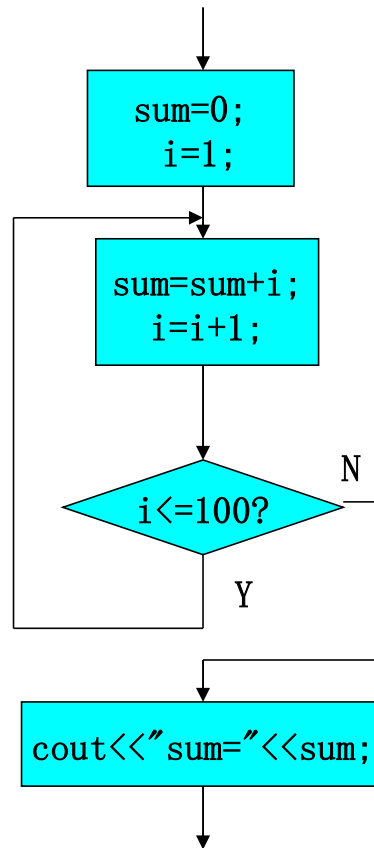
```
#include <iostream>
using namespace std;

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

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

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

    return 0;
}
```





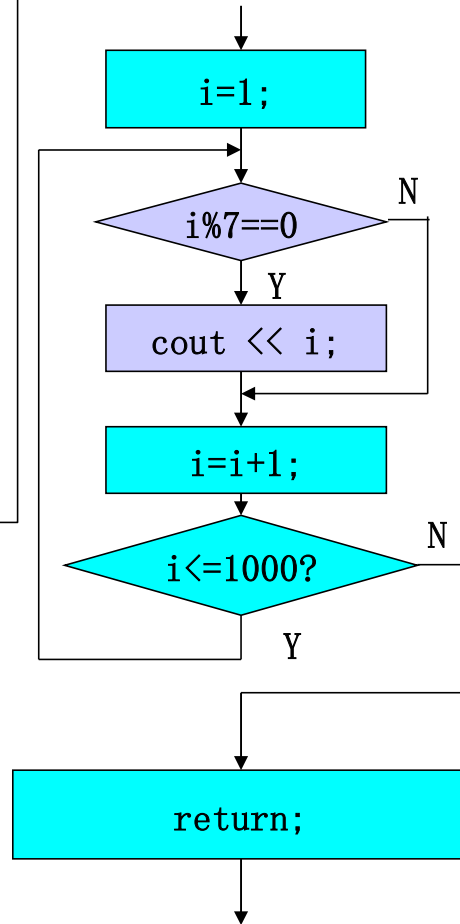
例：打印1-1000内7的倍数

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

int main()
{
    int i=1;

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

    return 0;
}
```



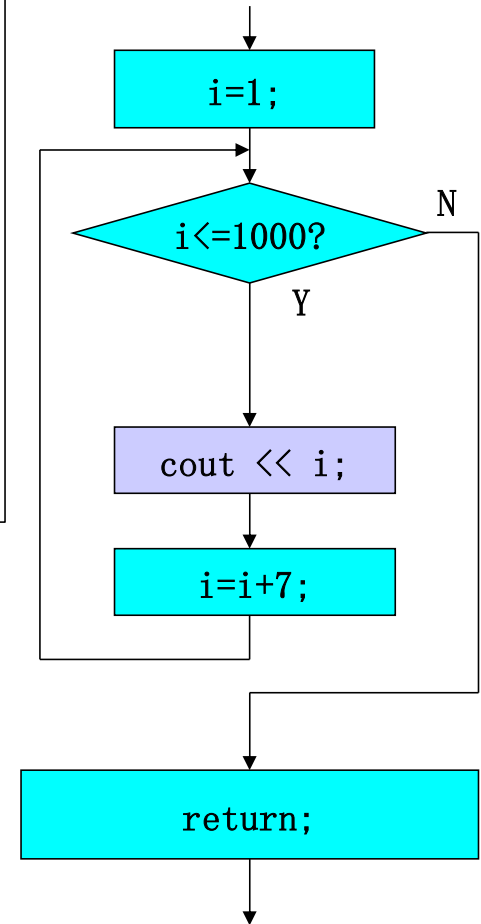
```
#include <iostream>
using namespace std;

int main()
{
    int i=7;

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

    return 0;
}
```

更简洁高效的做法，  
只要数学上等价即可





## § 3. 结构化程序设计基础

### 3.8. 循环结构和循环语句

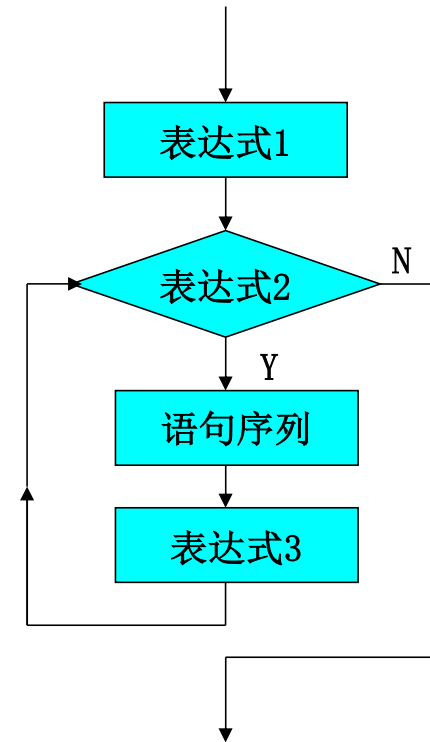
#### 3.8.4. 用for语句构成循环

##### 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. 结构化程序设计基础

### 3.8. 循环结构和循环语句

#### 3.8.4. 用for语句构成循环

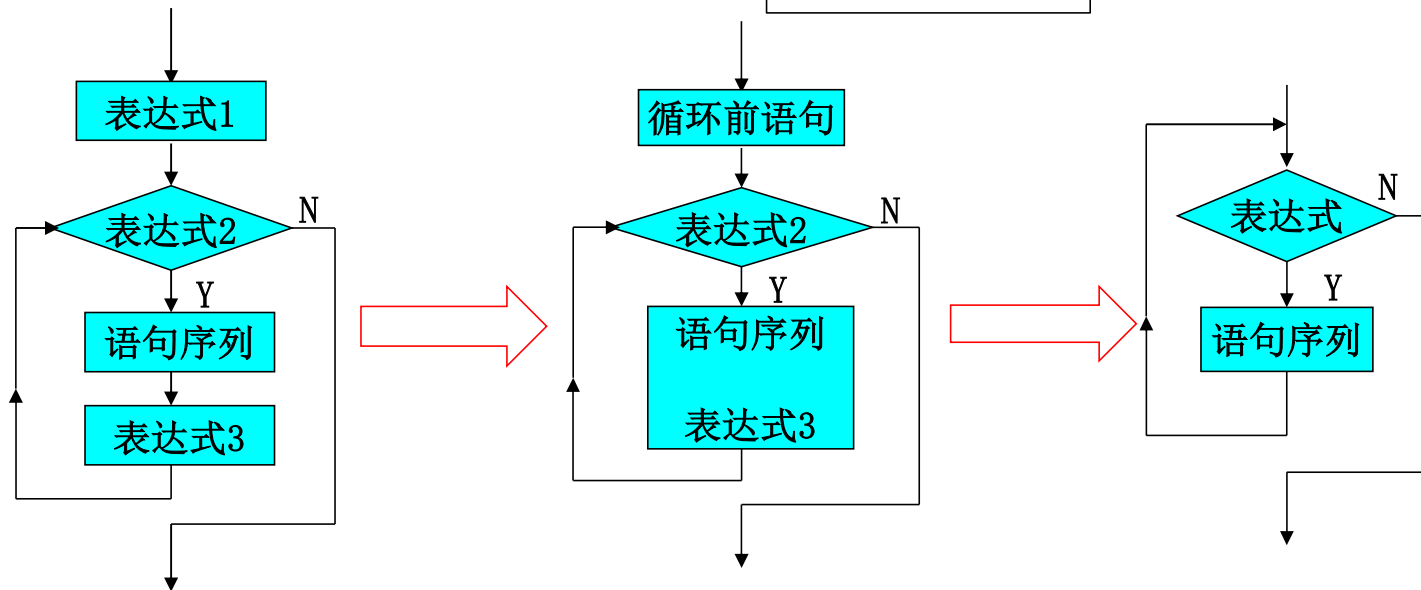
##### 3.8.4.1. 形式

##### 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. 结构化程序设计基础

### 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;  
    cout << "i=" << i << " sum=" << sum << endl;  
    return 0;  
}
```

Microsoft Visual Studio 调试控制台  
i=141 sum=10011

为什么循环结束后要--i ?

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

```
int main()  
{  
    int i=1, sum=0;  
    while(sum<=10000)  
        sum+=i++;  
    --i;  
    cout << "i=" << i << " sum=" << sum << endl;  
    return 0;  
}
```

Microsoft Visual Studio 调试控制台  
i=141 sum=10011



## § 3. 结构化程序设计基础

### 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

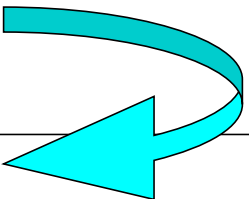
```
i=1;  
for (;i<=100;i++)  
    sum=sum+i;
```

```
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. 结构化程序设计基础

### 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. 结构化程序设计基础

### 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，  
如果因此而导致循环不需要包含任何语句，则再加一个;表示不需要语句序列(;也可以理解为空语句)

```
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=sum+i++);
```

//分号单独一行，表示为空语句形式，更直观

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

★ 最后加一个分号，表示  
for循环不需要语句序列



## § 3. 结构化程序设计基础

### 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，  
如果因此而导致循环不需要包含任何语句，则要加一个;表示不需要语句序列(;也可以理解为空语句)

上课认真听  
下课动手做，仔细观察并思考课上所讲

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

有分号

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

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

无分号

```
int main()
{
    int i, sum;
    for(i=1, sum=0; i<=100; sum+=i++)
        cout << "sum=" << sum << endl;
    return 0;
} //虽然cout形式上无缩进，仍然是for的子句
```

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

有分号

```
int main()
{
    int i, sum;
    for(i=1, sum=0; i<=100; sum+=i++);
        cout << "sum=" << sum << endl;
    return 0;
} //虽然cout形式上缩进了，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++)
        ; //空语句可单独一行，表达更清晰一些
    cout << "sum=" << sum << endl;
    return 0;
}
```



## § 3. 结构化程序设计基础

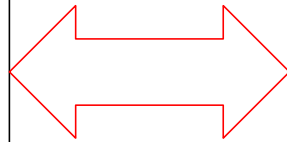
### 3.8. 循环结构和循环语句

#### 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. 结构化程序设计基础

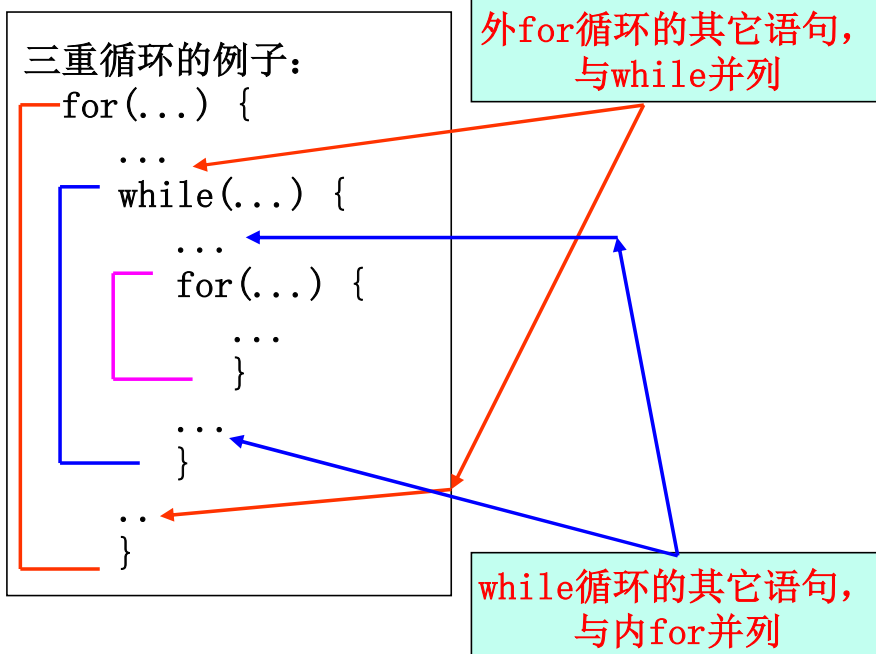
### 3.8. 循环结构和循环语句

#### 3.8.5. 循环的嵌套

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





## § 3. 结构化程序设计基础

### 3.8.5. 循环的嵌套

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

|                                                                                                              |                                                                                                                                                                                                                                                                                                                                                   |
|--------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>for(i=1;i&lt;=100;i++)<br/>    for(j=1;j&lt;=100;j++)<br/>        cout &lt;&lt; i*j &lt;&lt; ' ';</pre> | <pre>for(i=1;i&lt;=100;i++) {<br/>    cout &lt;&lt; i &lt;&lt; endl;<br/>    for(j=1;j&lt;=100;j++) {<br/>        cout &lt;&lt; i*j &lt;&lt; endl;<br/>        for(k=1;k&lt;=100;k++)<br/>            cout &lt;&lt; i*j*k &lt;&lt; ' ';<br/>        cout &lt;&lt; j*i &lt;&lt; endl;<br/>    }<br/>    cout &lt;&lt; i &lt;&lt; endl;<br/>}</pre> |
| cout 语句执行了 <u>10000</u> 遍                                                                                    | 红语句及for j执行了_____遍<br>蓝语句及for k执行了_____遍<br>粉语句执行了_____遍                                                                                                                                                                                                                                                                                          |



## § 3. 结构化程序设计基础

### 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

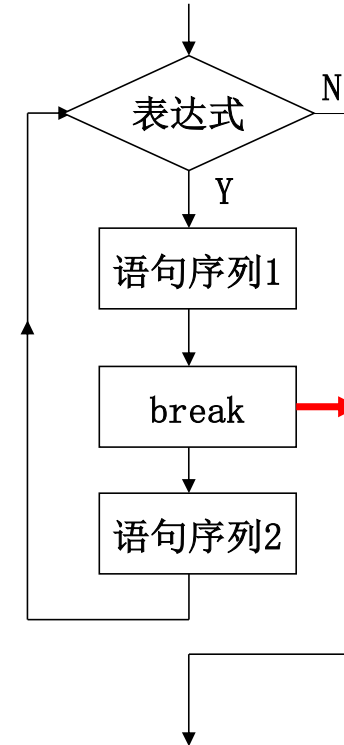
例：i从1开始累加，加到10000为止，打印出i及累加的和

```
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;
}
```

```
for(i=1, sum=0;; i++) {
    sum=sum+i;
    if (sum>10000)
        break;
}
//不需要i--, 为什么?
```

```
int main()
{
    int i=1, sum=0;
    while(sum<=10000)
        sum+=i++;
    i--;
    printf("i=%d sum=%d", i, sum);
    return 0;
}
```

```
while(1) {
    sum=sum+i++;
    if (sum>10000)
        break;
}
i--;
```



按程序执行逻辑逐步  
分析，注意细节!!!



## § 3. 结构化程序设计基础

### 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. 结构化程序设计基础

### 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

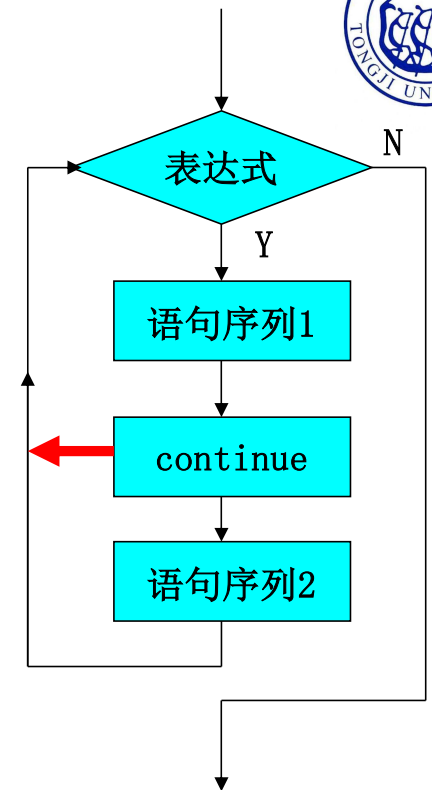
##### 3.8.6.2. continue语句

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





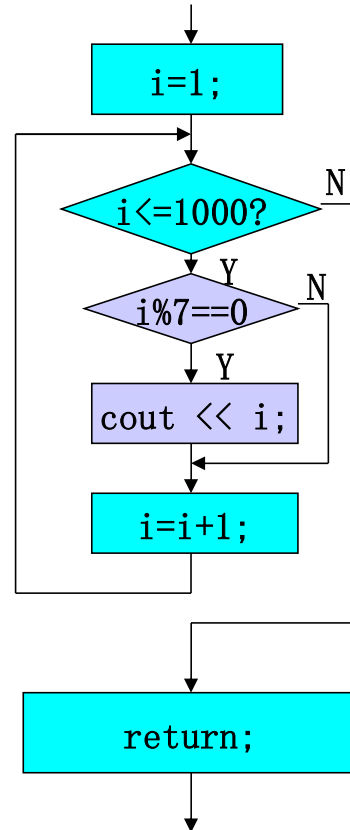
例：打印1-1000内7的倍数

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

```
int main()
{
    int i=1;

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

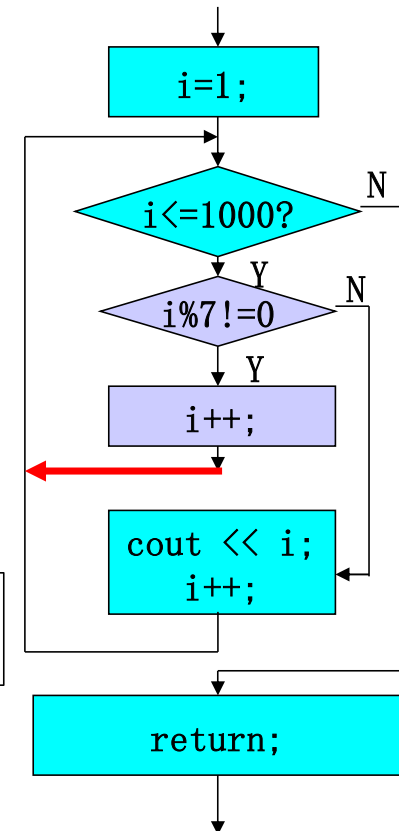
    return 0;
}
```



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

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

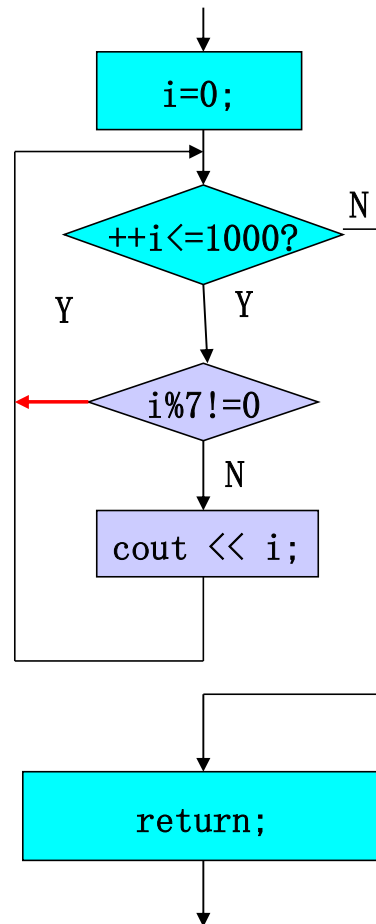
仅为了举例continue的使用，逻辑复杂，不建议



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. 结构化程序设计基础

### 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;  
}
```

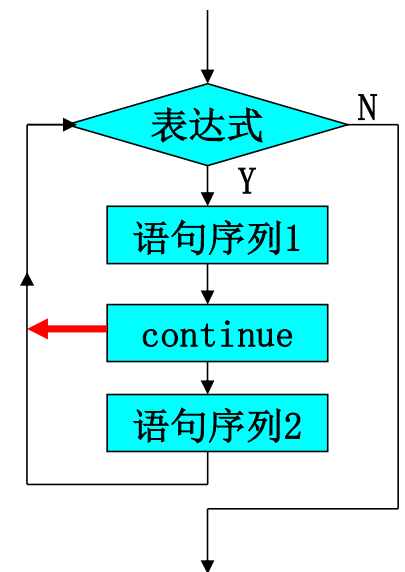
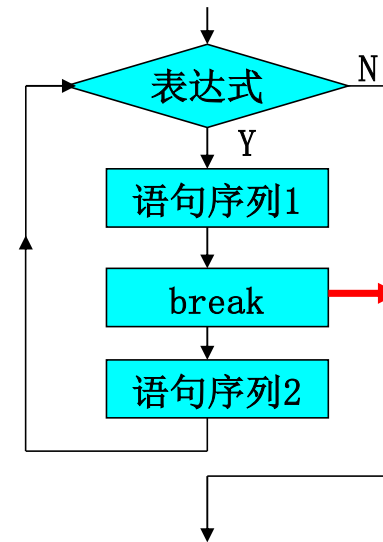
例：给出下列程序的运行结果

```
#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. 结构化程序设计基础

### 3.8. 循环结构和循环语句

#### 3.8.7. 编写循环结构的程序

例：用公式  $\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots$  求  $\pi$  值(到最后一项绝对值 $<10^{-7}$ 为止)

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

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

**//核心算法**

```
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  
}
```





```
#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;
    LARGE_INTEGER tick, begin, end;
    QueryPerformanceFrequency(&tick);    //取计数器频率
    QueryPerformanceCounter(&begin);      //取初始硬件定时器计数
    while(fabs(t) > 1e-6) {
        pi=pi+t;
        n=n+2;
        s=-s;
        t=s/n;
    }

    QueryPerformanceCounter(&end); //获得终止硬件定时器计数
    /* 执行到此, 打印pi的值 */
    pi=pi*4;
    cout << "n=" << setprecision(10) << n << endl;
    cout << "pi=" << setiosflags(ios::fixed) << setprecision(9) << pi << endl;

    cout << "计数器频率: " << tick.QuadPart << "Hz" << endl;
    cout << "时钟计数 : " << end.QuadPart - begin.QuadPart << endl;
    cout << setprecision(6) << (end.QuadPart - begin.QuadPart)/double(tick.QuadPart) << "秒" << endl;
    return 0;
}
```

调整两处蓝色箭头处的值,  
对比不同精度的执行时间

(1) n, t, pi为double型

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

(2) n, t, pi为float型

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



## § 3. 结构化程序设计基础

### 3.8. 循环结构和循环语句

#### 3.8.7. 编写循环结构的程序

例：求 Fibonacci 数列的前40项

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

int main()
{
    int i, f1=1, 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
    }

    return 0;
}
```

本程序的不足之处：无法回溯  
(当f1表示第7个月后，第5个月的值无法再现)



## § 3. 结构化程序设计基础

### 3.8. 循环结构和循环语句

#### 3.8.7. 编写循环结构的程序

例：找出100-200间的全部素数

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    int m, k, i, line=0;
    bool prime;

    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) 则VS有警告
        for(i=2; i<=k; i++)
            if (m%i==0) {
                prime=false;
                break;
            }

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

for(i=2; prime && i<=k; i++)  
if (m%i==0)  
 prime=false;  
//是否可以改成这种形式?

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

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

m=103-200，看一下两者区别，  
为什么？哪个是正确的？

```
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    int m, k, i, line=0;

    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;
            line++;
            if (line%10==0)
                cout << endl;
        }
    } //end of for

    return 0;
}
```

改写：  
不用逻辑变量prime，  
直接用i和k的关系来判断  
想清楚，为什么!!!

i循环的退出有两个可能  
1、不满足i<=k(是, 且i>k)  
2、满足m%i==0(否, 且i<=k)