



UNIT 02

C++简单程序设计

武汉大学计算机学院程序设计课程组

主讲人：常 军

E-MAIL: chunsc@163.com

电 话: 18986211771

QQ 群: 1020712774



思考两个问题？

数值数据： 1, 2, ... 1.23, ... -2.3, ...	文本： 武汉大学 Program
图形图像： 照片 视频	音频： 说话 音乐

计算机中的数据和数学中的
数据有何相同和不同之处？

结构化程序设计为什么只有
三种基本结构？



本讲提纲

C++语言概述

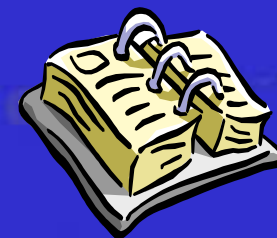
基本数据类型和表达式 (课前自学)

数据的输入与输出

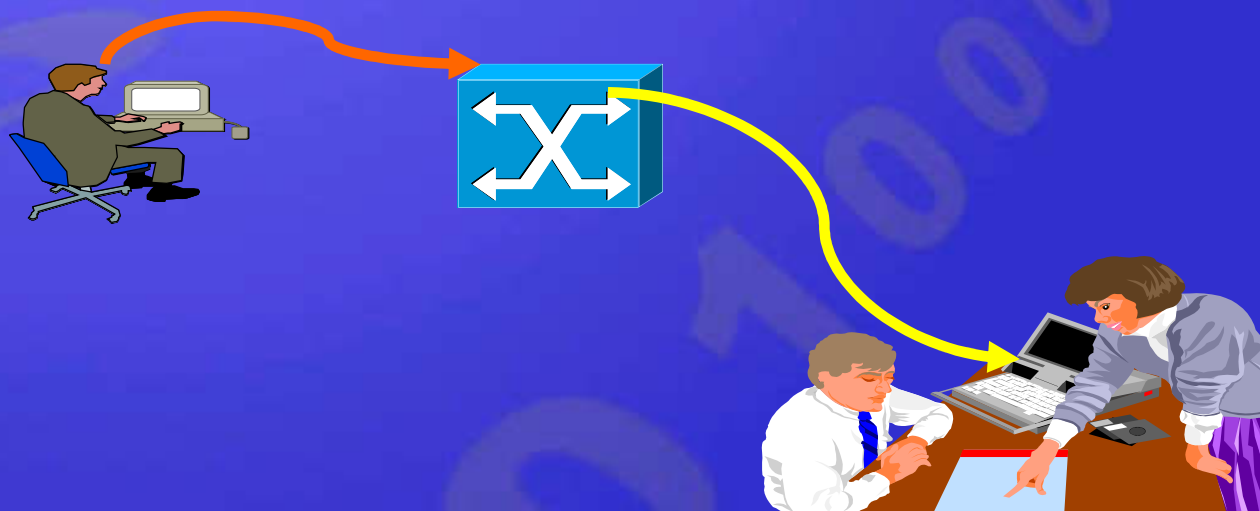
算法的基本控制结构 (课前浏览预习)

类型别名与类型推断

深度探索



1. C++语言概述



2.1.0 常用面向对象语言

混合型面向对象程序设计语言C++

纯面向对象程序设计语言Java

可视化程序设计语言Visual Basic

最纯正的面向对象语言Smalltalk

最早的面向对象程序设计语言Simula

.....

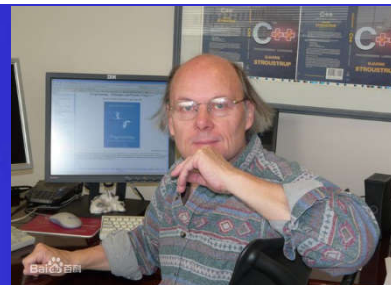


2.1.1 C++ 发展历程

- **第一阶段：从80年代到1995年。** 这一阶段C++语言基本上传统类型上的面向对象语言，并且凭借着接近C语言的效率，在工业界使用的开发语言中占据了相当大份额；
- **第二阶段：从1995年到2000年。** 这一阶段由于标准模板库(STL)和后来的Boost等程序库的出现，泛型程序设计在C++中占据了越来越多的比重性。
同时由于Java、C#等语言的出现和硬件价格的大规模下降，C++受到了一定的冲击；
- **第三阶段：从2000年至今。** 由于以Loki、MPL等程序库为代表的产生式编程和模板元编程的出现，C++出现了发展历史上又一个新的高峰，这些新技术的出现以及和原有技术的融合，使C++已经成为当今主流程序设计语言中最复杂的一员



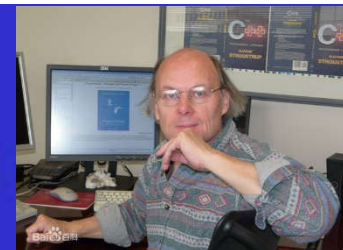
2.1.1 C++ 发展历程 (典型事件)



- ① 1967 年, Simula 语言中第一次出现了面向对象 (OO) 的概念, 但由于当时软件规模还不小, 技术也还不太成熟, 面向对象的优点并未发挥出来。
- ② 1980 年, Smalltalk-80 出现后, 面向对象技术才开始发挥魅力。
- ③ 1979 年, Bjarne Stroustrup 借鉴 Simula 中 "Class" 的概念, 开始研究增强 C 语言, 使其支持面向对象的特性。B.Stroustrup 写了一个转换程序 "Cfront" 把 C++ 代码转换为普通的 C 代码, 使它在各种各样的平台上立即投入使用。1983 年, 这种语言被命名为 C++
- ④ 1986 年, B.Stroustrup 出版了 《The C++ Programming Language》第一版, 这时 C++ 已经开始受到关注, B.Stroustrup 被称为 C++ 之父 (Creator of C++)。



2.1.1 C++ 发展历程 (典型事件)



- ⑤ 1989 年, 负责 C++ 标准化的 ANSI X3J16 挂牌成立。1990 年, B.Stroustrup 出版了《The Annotated C++ Reference Manual》(简称 ARM), 由于当时还没有 C++ 标准, ARM 成了事实上的标准。
- ⑥ 1990 年, Template(模板) 和 Exception(异常) 加入到了 C++ 中, 使 C++ 具备了泛型编程(Generic Programming)和更好的运行期错误处理方式。
- ⑦ 1991 年, 负责 C++ 语言国际化的技术委员会工作组 ISO/IEC JTC1/SC22/WG21 召开了第一次会议, 开始进行 C++ 国际化的工作。从此, ANSI 和 ISO 的标准化工作保持同步, 互相协调。
- ⑧ 1993 年, RTTI(运行期类型识别) 和 Namespace(名字空间) 加入到 C++ 中。1994 年, C++ 标准草案出台。B.Stroustrup 出版了《The Design and Evolution of C++》(简称 D&E)。



2.1.1 C++ 发展历程 (典型事件)

- ⑨ 本来，C++ 标准已接近完工，这时 STL(标准模板库) 的建议草案被提交到标准委员会，对 STL 标准化的讨论又一次推迟了 C++ 标准的出台。
- ⑩ 1998 年，ANSI 和 ISO 终于先后批准 C++ 语言成为美国国家标准和国际标准。
- ⑪ 2000 年，B.Stroustrup 推出了《The C++ Programming Language》特别版(Special Edition)，书中内容根据 C++ 标准进行了更新。



C&C++语言

初步印象



2.1.2 C++语言特点

最大特点之一：C++是混合型面向对象程序设计语言

四种“子语言”

- ① **C子语言**：C++支持C语言的几乎全部功能，在语法上与C语言仅有极微妙的差别。
- ② **面向对象的C++**：C++作为一门面向对象的语言而闻名。
- ③ **泛型编程语言**：C++强大（但容易失控的）模板功能使它能在编译期完成许多工作，从而大大提高运行期效率。
- ④ **STL (C++标准模板库)**：随着STL的不断发展，它已经逐渐成为C++程序设计中不可或缺的部分，其效率可能比一般的代码低些，但是其安全性与规范性使它大受欢迎。



2.1.2 C++ 语言特点

- ① **C语言超集**，保持对C的兼容。既保持了C语言的简洁、高效和接近汇编语言等特点，又克服了C语言的缺点，其编译系统能检查更多的语法错误，因此，C++比C语言更安全。代码质量高、速度快、可移植性好；
- ② **强类型语言**，编译阶段就能发现程序潜在错误，不会将错误带到运行阶段；
- ③ **表达能力强**，C++的多继承是JAVA, C#等语言所没有的；
- ④ **支持运算符重载**，对象的运算更易表达且表达更加自然。
- ⑤ **抽象能力强**，函数模板和类模板提供更高级别的抽象；C++设计成直接的和广泛的支援多种程序设计风格（程序化程序设计、资料抽象化、面向对象程序设计、泛型程序设计）；
- ⑥ **内存管理高效**，C++提供自动和人工回收两种方式；
- ⑦ **异常处理加强**，支持对象类型的异常；
- ⑧ **支持名字空间**，更加有利于大型软件工程项目；
- ⑨ **非纯面向对象的语言**，同时支持对象和模块描述程序结构。



2.1.3 C++ 程序实例 — 例2-1

```
//2_1.cpp
#include <iostream> //头文件包含
using namespace std; //引入C++标准名字空间std
// 一个完整的C++程序有且仅有一个主函数main
int main() {
    cout << "Hello!" << endl; //输出语句，标准输出流cout
    cout << "Welcome to c++!" << endl;
    return 0; /*返回0，表示程序执行正常结束*/
}
```

运行结果：

```
Hello!
Welcome to c++!
```

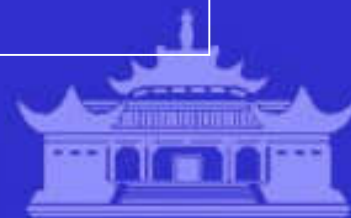
2.1.3 C++ 程序实例

头文件:

`iostream`, C++头文件, 注意不用.h扩展名;
仍然可以用C头文件, 例如 `#include <stdio.h>`

名字空间:

- ◆ 标准C++引入的可以由程序员命名的作用域
- ◆ 每个命名空间中可以放入一些相关的实体, 例如变量、函数、对象、类, 以解决同名冲突问题
- ◆ `std`: C++标准类库的一个名字空间
- ◆ `using`指令: 例如`using namespace std`, 作用是引入`std`中的实体名到该程序中



2.1.3 C++ 程序实例——注释

注释是程序员为程序语句所做的说明，是提高程序可读性的一种手段。注释一般分为两种：
序言性注释和注解性注释。

C++提供两种注释形式：

- ◆ 一种用“//”开头，直至行末，用于**单行注释**
- ◆ 另一种用“/*”和“*/”括起任意文字，用于**多行注释**。



2.1.3 C++ 程序实例 1

```
//2_1.cpp
#include <iostream> //头文件包含
using namespace std; //引入C++标准名字空间std
// 一个完整的C++程序有且仅有一个主函数
int main()
{
    cout << "Hello!" << endl; //输出语句, 标准输出流cout
    cout << "Welcome to c++!" << endl;
    return 0; /*返回0, 表示程序执行正常结束*/
}
```



2.1.3 C++ 程序实例

编译预处理命令：
文件包含

```
#include <iostream>           //编译预处理，文件包含iostream
using namespace std;          //引入C++标准名字空间std
int main()                     //主函数
{
    char name[20];             //定义字符数组
    cout<<"please input your name:"; //输出提示信息
    cin>>name;                 //从键盘输入
    cout<<"Hello,"<<name<<"!"<<endl; //输出问候信息
    return 0;
}
```

有且仅有一个
main函数

标准I/O流：**cin**、**cout**



词法

语言的组词



2.1.4、字符集

C++语言中可用到的字符集有：

- ◆ **数字**：0、1、.....、9。
- ◆ **字母**：注意C++程序中严格区分大小写字母，如A和a是不同的字符。
- ◆ **空白符**：空格符、制表符、换行符和换页符统称为空白符。它们主要用于分隔单词，一般无其它特殊意义。
- ◆ **图形符号**：图形（可见）符号，即 ! “ # % & ‘ () * + , - . / ; : < = > ? [\] ^ { | } ~
主要用作各种运算符。



2.1.5、词法记号

词法记号是最小的词法单元，C++词法记号包括：
标识符、关键字、文字、操作符（运算符）、分
隔符、空白



2.1.5、词法记号——标识符

标识符：在C++语言中要使用的对象，如符号常量、变量、函数、标号、数组、文件、数据类型和其他各种用户定义的对象，**标识符就是这些语法对象的名字。**

标识符的构成规则：

- **标识符由三类字符构成**：英文大小写字母；数字0.....9；下划线。
- **必须由字母或下划线开头**；后面可以跟随字母、数字或下划线。
- C++语言**区分大小写**，即大小写字母有不同的含义，例如：**num**，**Num**，**NUM**为3个不同的标识符。



2.1.5、词法记号——标识符

例如：以下均是合法的标识符：

sum, a2, j5k3, suml_ave, _123

以下均是不合法的标识符：

8i /*错在以数字开头*/

w.5 /*错在出现小数点“.”*/

good bye /*错在中间有空格*/

使用标识符的注意事项：

- 标识符不能与关键字同名，不能与库函数或自定义函数同名。
- 在选择标识符时，尽量避免使用容易混淆的字符。比如：字母l和数字1、字母o和数字0、字母z和数字2、字母b和数字6等。

2.1.5、词法记号——关键字、文字

关键字： 又称保留字，指具有特定含义、专门用作系统的特定成分的一类标识符。不能用作一般标识符，即不允许用作变量名或函数名等。

标准 C++ 语言关键字参见教材第 22 页说明

注意关键字 C++ 语言的关键字都是小写的。例如 `else` 是关键字，但 `ELSE` 就不是关键字

文字： 在程序中直接使用符号表示的数据，包括数字、字符、字符串和布尔文字。



2.1.5、词法记号——操作符、分隔符

操作符（运算符）： C++语言中含有相当丰富的运算符。运算符与变量、函数一起组成表达式，表示各种运算功能。运算符由一个或多个字符组成。

分隔符： 分隔符在语法起间隔各个词法单元的作用。包括
() { } , : ;



2.1.5、词法记号——空白

空白：空白是空格、制表符（Tab键产生的字符）、垂直制表符、换行符、回车符、注释的总称。

空白符用于指示词法记号的开始和结束位置。

C++程序可以不必严格按行书写，凡是出现空格的位置都可以出现换行符。



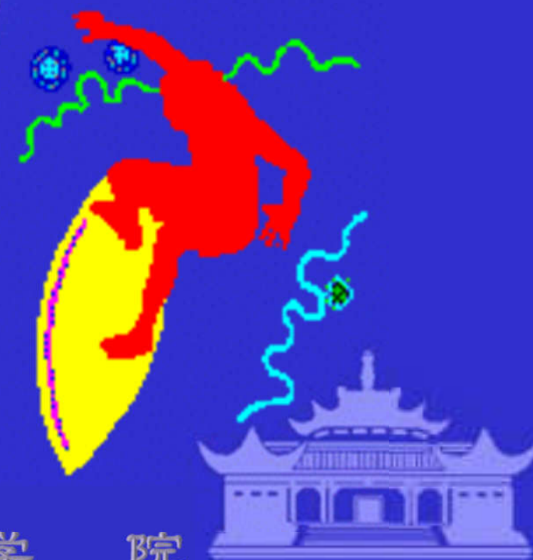
语法

语言的组句

怎么表达 数据 + 算法



2. 基本数据类型 和表达式



C++ 数据类型概述



C++ 数据类型概述

基本数据类型：对应于计算机的基本存储单元和使用这些单元去保存数据的一些常用方式。

自定义数据类型，又称复合数据类型：基于其他数据类型（基本数据类型或复合数据类型）定义的类型。



2.2.1 基本数据类型

C++能够处理的基本数据类型

- 整数类型
- 浮点数类型
- 字符类型
- 布尔类型

程序中的数据

- **常量**：在源程序中直接写明的数据，其值在整个程序运行期间不可改变，这样的数据称为常量。
- **变量**：在程序运行过程中允许改变的数据，称为变量。



2.2.1 基本数据类型

类型名	长度 (字节)	取值范围
bool	1	false, true
char	1	-128~127
signed char	1	-128~127
unsigned char	1	0~255
short (signed short)	2	-32768~32767
unsigned short	2	0~65535
int (signed int)	4	-2147483648~2147483647
unsigned int	4	0~4294967295
long (signed long)	4	-2147483648~2147483647
unsigned long	4	0~4294967295
float	4	$\pm 3.4\text{E}\pm 38$
double	8	$\pm 1.7\text{E}\pm 308$
long double	8	$\pm 1.7\text{E}\pm 308$



负数采用补码表示

2.2.1 基本数据类型

各类型长度以具体编译系统为准

有
符
号
整
型

整
型

短整型: short int

2个字节长度, 数据范围-32768~32767

整型: int

2 or 4个字节

长整型: long int

4 or 8个字节, 4字节的数据范围-2 147 483 648~2 147 483 647

长长整型: long long int

至少有8个字节



2.2.1 基本数据类型

各类型长度以具体编译系统为准

无符号整型

无符号短整型: unsigned short int

2个字节长度, 数据范围0~65535

无符号整型: unsigned int

2 or 4 个字节

无符号长整型: unsigned long int

4 or 8 个字节, 4字节的数据范围0~4 284 967 295

无符号长长整型: unsigned long long int

至少有8 个字节



2.2.1 基本数据类型

有
符
号

char

1个字节长度，数据范围-128~127

字
符
型

C++语言将字符型看作是1个字节的整数

无
符
号

unsigned char

1个字长，数据范围0~255



2.2.1 基本数据类型

浮点型: float

可以保存7位精度, $3.4E-38 \sim 3.4E38$

双精度浮点型: double

可以保存15位精度, $1.7E-308 \sim 1.7E308$

长双精度浮点型: long double

可以保存15位精度, $1.7E-308 \sim 1.7E308$



2.2.1 基本数据类型

布尔型，又称逻辑型： `bool`

VC++等环境中占1个字节，不同编译系统中占据字节数可能不同

数据取值：`false` 或者 `true` ，分别表示假、真

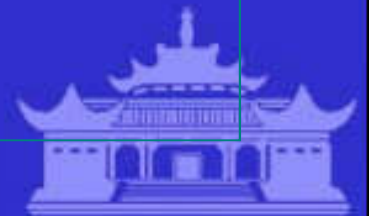


2.2.2 常量：数学中的常量和计算机中的常量

数学中的常量：取值不变的量。

计算机程序中的常量：

- ◆ 程序中取值不变的量；
- ◆ 常量的数据类型：根据字面形式可将常量区分为不同的数据类型。
 - 整型常量
 - 浮点型常量
 - 字符型常量
 - 字符串常量
 - 布尔常量



2.2.2 常量：整型常量（表示方法）

十进制整数：不带任何修饰的整数常量

123 -99

八进制整数：以0开头的整数常量

0123 067

068 ✕

十六进制整数：以0X或者0x开头的整数常量

0Xa123 0x67fe



2.2.2 常量：整型常量（数据类型）

规则： 编译程序把数值常量表示成最小兼容类型。

10 int

常量后缀： L，长型； U，无符号。

35000L long int

10000U unsigned int



2.2.2 常量：浮点型常量（表示方法）

小数表示法：

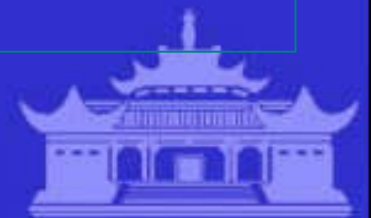
123.45 -99.0 0.234 .456

科学表示法：用字母e表示十进制指数中的10，前面为尾数（小数形式或者整数），后面为阶码（整数）。

0.123E12 0.123×10^{12}

67e-2 67×10^{-2}

6.8E2.3 ✕



2.2.2 常量：浮点型常量（数据类型）

规则： 编译程序把数值常量表示成最小兼容类型，无任何修饰系统默认为double类型。

123.23 double

1.0 double

0.987654321 double

常量后缀： F，浮点型； L，长型。

3.5L long double

123.23F float

1.23E-2F float



2.2.2 常量：字符型常量

字符常量：一对单引号中的单个字符。

'1' '+' '%' 'a'

转义字符，又称反斜线字符常量：

多数可打印字符适用于直接放在一对单引号的方法。但是少数（回车键等）不能通过键盘放在字符常量中，为此，C++采用特殊的反斜线字符常量。

\b	退格	\t	水平制表	\v	垂直制表	\a	报警
\f	换页	\"	双引号	\?	问号		
\n	换行	\'	单引号	\N	8进制常量		
\r	回车	\\	反斜线	\xN	16进制常量		



2.2.2 常量：字符串常量

字符串常量：一对双引号中的一系列字符。

"123.23 " " This is a test. " "

字符串结束标记：编译程序在编译源程序时自动在每个字符串末尾放空字符'\0',作为字符串结束标记。

"A" 和 'A' 的区别



C风格字符串常量

通过添加前缀可以改变字符常量或者字符串常量的类型，前缀及其含义如下表所示：

前缀	含义	类型
u	Unicode 16 字符	char16_t
U	Unicode 32字符	char32_t
L	宽字符	wchar_t
u8	UTF-8（仅用于字符串字面常量）	char

Unicode是一个巨大的字符集，给世界上所有的字符定义了一个唯一编码。其仅仅规定了每个符号的二进制代码，没有制定细化的存储规则。UTF-8、UTF-16、UTF-32才是Unicode的存储格式定义

在ANSI/ISO 9899--1990也就是美国国家为程序设计语言C指定的标准(也称为ANSI C)中是这样定义的：

用多个字节来代表的字符称之为宽字符，而Unicode只是宽字符编码的一种实现，宽字符并不一定是Unicode



2.2.2 常量：布尔常量

布尔常量

false

true



2.2.3 变量：数学中的变量 和计算机中的变量

数学中的变量：取值可变的量。

计算机程序中的变量：

- ◆ 程序中取值可变的量；
- ◆ *内存空间中已分配、并且已命名的位置*，数据类型决定系统分配给变量的内存空间的大小，以及该内存空间中数据的二进制编码规则



2.2.3 变量：变量的定义形式

【限定词】 类型 对象名称 【=初始值】 ；

【……】： 可选项

限定词： 类型限定词或者存储类型限定词。

类型： 有效的C++数据类型。

对象名称： 一个或多个用逗号间隔的标识符。

int a, b, c;

/*说明a, b, c为整型变量*/

char cc;

/*说明cc为字符变量*/

double x, y;

/*说明x, y为双精度实型变量*/

说明： 变量必须先定义，后使用



2.2.3 变量：变量的定义位置

所有函数外： 定义点之后的所有函数可以使用；全局变量或共有变量。

函数内： {的后面，函数内定义的变量只有该函数可以使用，局部变量或者私有变量。

函数参数定义： 形式参数，属于局部变量。



2.2.3 变量：变量的初始化

定义的同时初始化：

```
int i1=3, i2=4;  
float f1, f2=3.5;
```

初始化和赋值的区别？

定义后，单独赋值：

```
int i1,i2;          /* 定义整型变量i1和i2 */  
i1=3;  
i2=4;              /* 为i1赋初值为3，i2赋初值为4 */
```



2.2.3 变量：变量的初始化

C++定义了多种变量初始化的形式，例如：

`int units_sold = 0;` //我们最熟悉的初始化形式

`int units_sold = {0};` //列表初始化形式

`int units_sold {0};` //第二种列表初始化形式

`int units_sold (0);` //第四种初始化形式



2.2.3 变量：变量的初始化

这四种变量初始化的形式有什么区别吗

我们看一个范例程序，例如：

```
1  #include <iostream>
2
3  int main()
4  {
5      long double ld=3.1415926536;
6      int a{ld}, b={ld};
7      int c(ld),d=ld;
8      std::cout<<a<<','<<b<<std::endl;
9      std::cout<<c<<','<<d<<std::endl;
10     std::cin.ignore();
11     return 0;
12 }
13
```

用花括号初始化，一旦初始值存在丢失信息的风险时，编译器会警告！

第6行警告，第7行就没有！

编译时系统提示：

	E:\01-本科教学\02-C++程序设计\01-课件-ppt\02-2...	In function 'int main()':
6	10	E:\01-本科教学\02-C++程序设计\01-课件-ppt\02-2017... [Warning] narrowing conversion of 'ld' from 'long double' to 'int' inside {} [-Wnarrowing]
6	18	E:\01-本科教学\02-C++程序设计\01-课件-ppt\02-2017... [Warning] narrowing conversion of 'ld' from 'long double' to 'int' inside {} [-Wnarrowing]

2.2.3 变量：变量的初始化

默认初始化，是指定义变量没有指定初值时，则变量被默认初始化，即变量被赋予了“默认值”。

基本数据类型的变量未被初始化，它的值由定义的位置决定。定义在函数之外的，默认初值为0，定义在函数之内的，则不被初始化！

未初始化的变量取值不确定，使用它可能引发运行时故障！



2.2.3 变量：限定词

【限定词】 类型 对象名称 【=初始值】 ；

const类型限定词：const型的变量可以初始化，但不能被程序修改。const定义的就是符号常量。

```
const int a=10;
```

定义int类型变量a，其初始值为10。程序不能修改a的值。

volatile类型限定词：volatile型的变量可能以没有在程序中明确说明的方式改变。多数编译程序认为没有出现在赋值语句左侧的变量不会改变。volatile型的变量告知编译程序，volatile型的变量例外，因此编译系统会自动优化。例如用变量保存系统的实时时钟值。



2.2.3 变量：变量的存储类别

存储类别：

- ◆ **局部变量，默认：**局部动态生存期，函数形参和不加static说明的局部变量。
- ◆ **register：**寄存器类别，用于说明自动存储期的变量，以达到目标代码优化的目的。
- ◆ **extern：**外部类别，说明需要使用在程序的其他地方具有外部链接的对象，被说明的对象必须是静态生存期的变量。
- ◆ **static：**静态生存期，可用于全局变量和局部变量。
静态全局变量：设置为内部链接（本文件可访问）。
静态局部变量：具有静态生存期。



全局变量：默认extern类别；

局部变量：默认，局部动态生存期。



2.2.3 变量：变量的存储类别

存储类别：

- ◆ **thread_local**：具有线程存储生存期，是Thread的局部变量，用于编多线程程序。只能用于修饰命名空间以及块作用域中的变量，或者已经被指定为static的变量。
- ◆ **mutable**：只能用于类数据成员，并且不能与const或者static同时使用，不能用来修饰引用变量。mutable关键字是为了突破const关键字的限制，被mutable关键字修饰的成员变量永远处于可变的狀態，即使是在被const修饰的成员函数中。



2.2.3 变量：赋值及左值和右值

< 左值 > = < 右值 >

左值和右值的本意如下（现在的C++对其有扩展）

左值：可以出现在赋值运算符左边的值；是有地址的值；

右值：可以出现在赋值运算符右边的值；

① `x = 5;`



② `5 = x;`



- ◆ C++中有些运算符仅仅需要操作数需要右值，例如“+”的两侧、赋值运算符的右边；
- ◆ 有些运算符仅仅需要左值，例如赋值运算符“=”的左边；
- ◆ 有些运算符需要操作数同时有右值与左值的角色，例如++或--



2.2.4 符号常量

符号常量在声明时一定要赋初值，而在程序中间不能改变其值。

✓ **const** 数据类型说明符 常量名=常量值;

或:

数据类型说明符 **const** 常量名=常量值;

例: **const float PI = 3.1415926;**



*constexpr*与常量表达式

常量表达式：值不能改变的表达式

例：`const float PI = 3.1415926;`
`const int size=get_size();` //size不是常量表达式

◆ **constexpr**:被修饰的变量暗含**const**属性

例：`constexpr int size=get_size();`
//当`get_size()`是**constexpr**函数时，编译能通过



2.2.5、运算符和表达式： 表达式的基本特征

表达式，就像英文中一个完整的语句：

- ◆ **动词**：操作符或函数；要完成的操作或动作。
- ◆ **名词**：变量或常量，运算对象。
- ◆ **一个结果**：表达式的值。



2.2.5、运算符和表达式： 运算符和算元

运算符：操作符。C++语言中：数学运算符，以及大量特殊运算符。

操作数：表达式中，一个操作的输入值。

$3+a$ +运算符；3、a操作数

算元：所需要的操作数的数目。

C++语言包括：一元运算符；二元运算符；三元运算符



2.2.5、运算符和表达式： 优先级别和括号

圆括号：组合符号（既不是运算符，也不是操作数），改变计算的先后次序。

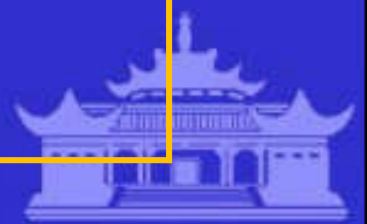
优先级别：定义每个运算符的优先级和结合性。结合性法则负责管理优先级别相同的运算符。

例如， $(3 * Z / 10)$

$*$ / 优先级别相同，结合性都是从左到右。

副作用：C++中少数运算符在表达式计算完成后，可能改变操作数变量的取值。

例如，增量运算符、赋值运算符



2.2.5、运算符和表达式：

C++语言中的运算符

结合律和运算符		功能	用法
左	::	全局作用域	:: name
左	::	类作用域	class :: name
左	::	命名空间作用域	namespace :: name
左	.	成员选择	object.member
左	->	成员选择	pointer->member
左	[]	下标	expr[expr]
左	()	函数调用	name(expr_list)
左	()	类型构造	type(expr_list)
右	++	后置递增运算	lvalue++
右	--	后置递减运算	lvalue--
右	typeid	类型 ID	typeid(type)
右	typeid	运行时类型 ID	typeid(expr)
右	explicit cast	类型转换	cast_name<type>(expr)

2.2.5、运算符和表达式： C++语言中的运算符（续）

结合律和运算符		功能	用法
右	++	前置递增运算	++lvalue
右	--	前置递减运算	--lvalue
右	~	位求反	~expr
右	!	逻辑非	!expr
右	-	一元负号	-expr
右	+	一元正号	+expr
右	*	解引用	*expr
右	&	取地址	&lvalue
右	()	类型转换	(type) expr
右	sizeof	对象的大小	sizeof expr

2.2.5、运算符和表达式：

C++语言中的运算符（续）

结合律和运算符		功能	用法
结合律和运算符		功能	用法
右	sizeof	类型的大小	sizeof(type)
右	Sizeof...	参数包的大小	sizeof...(name)
右	new	创建对象	new type
右	new[]	创建数组	new type[size]
右	delete	释放对象	delete expr
右	delete[]	释放数组	delete[] expr
右	noexcept	能否抛出异常	noexcept (expr)
左	->*	指向成员选择的指针	ptr->*ptr_to_member
左	.*	指向成员选择的指针	obj.*ptr_to_member
左	*	乘法	expr * expr
左	/	除法	expr / expr
左	%	取模（取余）	expr % expr
左	+	加法	expr + expr
左	-	减法	expr - expr
左	<<	向左移位	expr << expr
左	>>	向右移位	expr >> expr

2.2.5、运算符和表达式： C++语言中的运算符（续）

结合律和运算符		功能	用法
左	<	小于	expr < expr
左	<=	小于等于	expr <= expr
左	>	大于	expr > expr
左	>=	大于等于	expr >= expr
左	==	相等	expr == expr
左	!=	不相等	expr != expr
左	&	位与	expr & expr
左	^	位异或	expr ^ expr
左		位或	expr expr
左	&&	逻辑与	expr && expr
左		逻辑或	expr expr
右	? :	条件	expr ? expr : expr
右	=	赋值	lvalue = expr
右	*, /=, % =	复合赋值	lvalue += expr 等
右	+=, -=		
右	<<=, >>=		
右	&=, =, ^=		
右	throw	抛出异常	throw expr
左	,	逗号	expr, expr

2.2.5、运算符和表达式： C++语言中的运算符

基本运算符

单目运算符

算术运算符（先乘除，后加减）

移位运算符

关系运算符

逻辑运算符（！除外）

条件运算符

赋值运算符

逗号运算符

高

低



2.2.5、运算符和表达式：算术运算符

运算符	作用
-	减法、负号
+	加法
*	乘法
/	除法
%	模除
--	减量
++	增量

/ 除法

○ 两个整数除法的结果是整数，整数除法的商。

5/2 结果 2

-5/2 结果 -2

% 模除，整数除法的余数。

○ 二元运算符，操作数均为整数

7% 4 结果 3 -7% 4 结果 -3

-7%-4 结果 -3 7%-4 结果 3

0% 5 结果 0

○ %运算的符号只取决于第一个运算数的符号。



2.2.5、运算符和表达式：算术运算符 —— 增量、减量运算符

增量运算符： ++ 操作数增量一个单位
即： $x = x + 1$ 与 ++x 一样

前缀增量运算： ++x

对操作数完成增量运算；

使用增量后操作数的值完成表达式的运算

C++标准没有规定，
二者执行先后次序

后缀增量运算： x++

使用操作数的原值完成表达式的运算；

对操作数完成增量运算

C++标准没有规定，
二者执行先后次序

2.2.5、运算符和表达式：算术运算符 —— 增量、减量运算符

减量运算符： `--` 操作数减量一个单位
即： `x=x-1` 与 `x--` 一样

前缀减量运算： `--x`

对操作数完成减量运算；

使用减量运算后操作数的值完成表达式的计算

C++标准没有规定，
二者执行先后次序

后缀减量运算： `x--`

使用操作数的原值完成表达式的计算；

对操作数完成减量运算

C++标准没有规定，
二者执行先后次序

2.2.5、运算符和表达式：算术运算符

——不要滥用增量、减量运算符

不要滥用增/减量运算符：增量运算符有副作用，会改变运算分量的值但是如果使用不当，会带来意想不到的结果

1. 对于代码 `int i = 3; i = i++;` 不同编译器给出不同的结果，有的为3，有的为4，哪个是正确的？

答：没有正确答案；这个表达式无定义（无从判断该引用（左边的*i*）是旧值还是新值）。

注意，`i++` 和 `++i` 都不同于 `i+1`。如果你要使*i* 自增1，使用 `i=i+1`, `i+=1`, `i++` 或 `++i`，而不是任何组合。

2. 如：若 `i=3`，则表达式 `(i++) + (i++) + (i++)` 的结果应为多少？

答：有的系统从左到右完成上述运算，即表达式结果为 `3+4+5=12`。

另外一些系统（如 Turbo C、MS C）则先计算表达式的值，再自加3次*i*，即表达式结果为 `3+3+3=9`。

请使用类似 `i+(i+1)+(i+2)` 这样的表达式

2.2.5、运算符和表达式： 赋值运算符

对象名称=表达式

对象名称：变量或指针，赋值目标，要求有左值（可放在赋值运算符的左边⇒有用户访问的存储空间）。

```
int a, b, c;           /*说明a, b, c为整型变量*/  
a=12;  
b=c=a;                /*多重赋值*/  
a+=a-=a*a;            /*表达式的结果是多少? */
```

```
int a, b, c;           /*说明a, b, c为整型变量*/  
23=a; ×  
23是常量，没有左值。
```

2.2.5、运算符和表达式： 赋值运算符——赋值中的类型

对象名称=表达式

对象名称与表达式类型不一致时

赋值的类型转换规则：赋值右部（表达式）的值转换为赋值左部（赋值目标，对象名称）的类型。

```
int x;
```

```
float y = 3.5;
```

```
x = y;
```

则运行后，变量x的值为3，int类型。



2.2.5、运算符和表达式：

赋值运算符——赋值中的类型转换

整型类别数据之间的转换

短→长

规则：转换前后的数据取值不变。无符号数，高位补0；有符号数，高位补符号位。

例如：

```
unsigned char c;  
short int x;  
c='\376';    /*(八进制数376) */  
x=c;
```

变量c 11111110



变量x

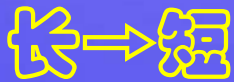
00000000

11111110



2.2.5、运算符和表达式： 赋值运算符——赋值中的类型转换

整型类别数据之间的转换



规则：截取低位赋值，丢弃高位。

例如：

```
unsigned char c;
```

```
short int x;
```

```
x=0xff76;
```

```
c=x;
```

变量c

0	1	1	1	0	1	1	0
---	---	---	---	---	---	---	---

变量x

1	1	1	1	1	1	1	1	0	1	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---



2.2.5、运算符和表达式：

赋值运算符——复合赋值运算符

对象名称 运算符=表达式

赋值语句的变异：等价于

对象名称=对象名称 运算符(表达式)

$x-=10;$

等价于

$x=x-10;$



2.2.5、运算符和表达式： 逗号运算符

Exp1 , Exp2

Exp1 , Exp2,,Expn

1. 计算Exp1的值；
2. 计算Exp2的值；
3. 以此类推，以最后一项Expn的值作为表达式的结果。

例： 1) $a=3*5, a*4$ 结果为60
 2) $(a=3*5, a*4), a+5$ 结果为20



2.2.5、运算符和表达式：

关系和逻辑运算符

关系运算符		逻辑运算符	
运算符	作用	运算符	作用
<	小于	&&	与
>	大于		或
<=	小于等于	!	非
>=	大于等于	优先级别 ! > >= < <= == != && 	
==	等于		
!=	不等		



2.2.5、运算符和表达式： 关系与逻辑运算符

关系运算符：关系指各值之间的关系

$3 > 5$

$'a' \leq 'b'$

$1.2 == 2.5$

$x == 0$

逻辑运算符：逻辑指如何组合各值之间的关系。

$(x > 0) \&\&(x < 10)$

$!(x == 0)$

“真” (true) 和 “假” (false)：非零值为true，零值为假。

逻辑或关系表达式的结果：假值为false，真值为true.

$12 > 20$ 结果为false

$'a' > 'b'$

结果为false

$12 < 20$ 结果为true

$'a' < 'b'$

结果为true

2.2.5、运算符和表达式： 逻辑运算符（短路原则）

$a \& \& b$

当a为false时，可提前计算表达式结果为false，因此不再处理b。

例如，设变量int m,n,a,b的值均为0，则执行表达式 $(m=a>b) \& \& (n=a \geq b)$ 后，m、n 的值分别为（ 0 ）和（ 0 ）。

$a \parallel b$

当a为true时，可提前计算表达式结果为true，因此不再处理b。

例如，设变量int m,n,a,b的值均为0，则执行表达式 $(m=a \geq b) \parallel (n=a \geq b)$ 后，m、n 的值分别为（ 1 ）和（ 0 ）。



2.2.5、运算符和表达式： 条件运算符

Exp1 ?Exp2:Exp3

1. 计算Exp1的值；
2. 如果Exp1的值为真，计算Exp2的值作为表达式的结果；
3. 如果Exp1的值为假，计算Exp3的值作为表达式的结果。

x=10;

y=(x>9)?100:200;



2.2.5、运算符和表达式： 编译时运算符，*sizeof*()

***sizeof*(操作数)**

***sizeof* 变量名**

- **操作数**：变量、数据类型名。操作数为类型名必须用圆括号，操作数为变量可以不必用圆括号。
- **编译时一元运算符**；
- **返回操作数对应的数据类型的字节数。**



2.2.5、运算符和表达式： 位运算符

运算符	作用
&	按位与
	按位或
^	异或（没有进位的二进制加法运算）
~	求1的补
>>	右移
<<	左移

位运算运算规则

		&		^
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0



2.2.5、运算符和表达式：

位运算符——位逻辑运算符

例：如果m为0x137f，n为0xf731，则m和n在16位计算机内的二进制表示形式如下：

m	0001 0011 0111 1111
n	1111 0111 0011 0001

m&n=	0001 0011 0011 0001	=0x1331
m n=	1111 0111 0111 1111	=0xf77f
m^n=	1110 0100 0100 1110	=0xe44e
~m=	1110 1100 1000 0000	=0xec80

与逻辑运算符区别：位逻辑运算符是针对二进制位，而逻辑运算符是针对整个表达式。

4&6 结果为 4

4&&6 结果为 1



2.2.5、运算符和表达式：

位运算符——左移位运算符

<<：左移1位相当于乘2运算。

`x=0x23;`

0000 0000 0010 0011

`x<<=2;`

0000 0000 1000 **1100**

x的结果为0x8C

<<：左移1位相当于乘2运算。

`x=0xff23;`

1111 1111 0010 0011

`x<<=2;`

1111 1100 1000 **1100**

x的结果为0xFC8C



2.2.5、运算符和表达式：

位运算符——位移运算符

>>：右移1位相当于除以2运算。

<code>x=0x23;</code>	<code>0000 0000 0010 0011</code>
<code>x>>=2;</code>	<code>0000 0000 0000 1000</code>
<code>x</code> 的结果为0x8	

>>：右移1位相当于除以2运算。

<code>x=0xff23;</code>	<code>1111 1111 0010 0011</code>
<code>x>>=2;</code>	<code>??11 1111 1100 1000</code>
<code>x</code> 的结果为？	

答：取决于x为有符号数或无符号数。无符号数右移高位补0，有符号数高位补符号位。



2.2.5、运算符和表达式：

混合运算时数据类型的转换——隐含转换

一些二元运算符（算术运算符、关系运算符、逻辑运算符、位运算符和赋值运算符）要求两个操作数的类型一致。

在算术运算和关系运算中如果参与运算的操作数类型不一致，编译系统会自动对数据进行转换（即隐含转换），基本原则是将低类型数据转换为高类型数据。

char, short, int, unsigned, long, unsigned long, float, double

低  高



2.2.6、运算符和表达式：

混合运算时数据类型的转换——隐含转换

条件	转换
有一个操作数是long double型。	将另一个操作数转换为long double型。
前述条件不满足，并且有一个操作数是double型。	将另一个操作数转换为double型。
前述条件不满足，并且有一个操作数是float型。	将另一个操作数转换为float型。
前述条件不满足（两个操作数都不是浮点数）。 有一个操作数是unsigned long long型。	将另一个操作数转换为unsigned long long型。
有一个操作数是long long型，另一个操作数是unsigned long型	两个操作数都转换为unsigned long long型。
前述条件不满足，并且有一个操作数是unsigned long型。	将另一个操作数转换为unsigned long型。
前述条件不满足，并且有一个操作数是long型，另一个操作数是unsigned int型。	将两个操作数都转换为unsigned long型。
前述条件不满足，并且有一个操作数是long型。	将另一个操作数转换为long型。
前述条件不满足，并且有一个操作数是unsigned int型。	将另一个操作数转换为unsigned int型。
前述条件都不满足	将两个操作数都转换为int型。



2.2.6、运算符和表达式：

混合运算时数据类型的转换——隐含转换

- ◆ 当把一个非布尔类型的算术值赋给布尔类型时，算术值为0则结果为false，否则结果为true。
- ◆ 当把一个布尔值赋给非布尔类型时，布尔值为false则结果为0，布尔值为true则结果为1
- ◆ 当把一个浮点数赋给整数类型时，结果值将只保留浮点数中的整数部分，小数部分将丢失。
- ◆ 当把一个整数值赋给浮点类型时，小数部分记为0。如果整数所占的空间超过了浮点类型的容量，精度可能有损失。



2.2.5、运算符和表达式：

混合运算时数据类型的转换——隐含转换

- ◆ 当参与运算的操作数必须是bool型时，如果操作数是其它类型，编译系统会自动将非0数据转换为true，0转换为false。
- ◆ 位运算的操作数必须是整数，当二元位运算的操作数是不同类型的整数时，也会自动进行类型转换，
- ◆ 赋值运算要求左值与右值的类型相同，若类型不同，编译系统会自动将右值转换为左值的类型。



2.2.5、运算符和表达式：

混合运算时数据类型的转换——显式转换

语法形式（3种）：

1. 类型说明符(表达式)
2. (类型说明符)表达式
3. 类型转换操作符<类型说明符>(表达式)

类型转换操作符可以是：const_cast、dynamic_cast、reinterpret_cast、static_cast

显式类型转换的作用是将表达式的结果类型转换为类型说明符所指定的类型。

例：int(z), (int)z, static_cast<int>(z) 三种完全等价



2.2.6 语句

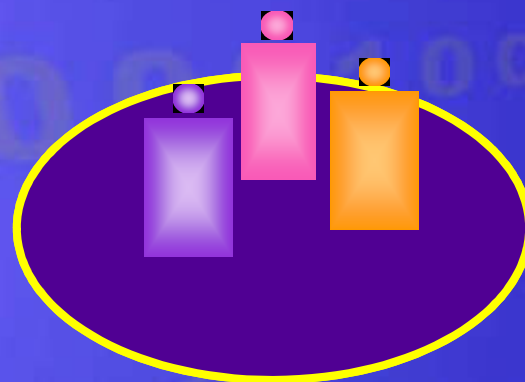
C++中的语句有如下种类

- **空语句**：只有一个语句结束符 “;”
- **声明语句**：例如，变量声明
- **表达式语句**：在表达式末尾添加语句结束符构成表达式语句

例如：a=3;

- **流程控制语句**（详见2.4节）：选择语句、循环语句、跳转语句
- **标号语句**：在语句前附加标号，通常用来与跳转语句配合
- **复合语句**：用 “{}” 括起来的多条语句





3. 数据的输入与输出



2.3.1 I/O 流

在C++中，将数据从一个对象到另一个对象的流动抽象为“流”。流在使用前要被建立，使用后要被删除。

从流中获取数据的操作称为**提取**操作，向流中添加数据的操作称为**插入**操作。

数据的输入与输出是通过I/O流来实现的，cin和cout是预定义的流类对象。cin用来处理标准输入，即键盘输入。cout用来处理标准输出，即屏幕输出。



2.3.2 预定义的插入符和提取符

“<<”是预定义的插入符，作用在流类对象cout上便可以实现最一般的屏幕输出。

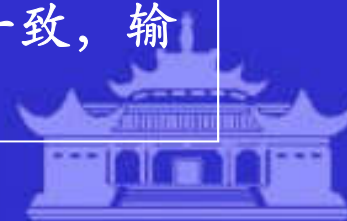
```
cout<<E1<<E2<<...<<Em;
```

- ◆ “<<”是预定义的插入运算符，E1、E2、...、Em均为表达式
- ◆ 功能是计算各表达式的值，并将结果输出到屏幕当前光标位置处。

键盘输入是将提取符作用在流类对象cin上。

```
cin>>V1>>V2>>...>>Vn;
```

- ◆ “>>”是预定义的提取运算符，V1、V2、...、Vn都是变量。
- ◆ 功能是暂停执行程序，等待用户从键盘输入数据，各数据间用空格或Tab键分隔，输入数据类型要与接受变量类型一致，输完后，按Enter回车键结束。



2.3.2 预定义的插入符和提取符

```
// C++  
cout<<"Welcome!";  
cin>>a;  
cout<<a<<endl;  
#include<iostream>
```

```
/* C */  
printf("Welcome!");  
scanf("%d",&a);  
printf("%d\n",a);  
#include<stdio.h>
```

- ◆ C++换行符可用**endl**，也支持C语言中的'\n'换行符
- ◆ C++语言中，标识符大小写敏感。
- ◆ 关键字是特殊的标识符，不能用作变量、常量等的名称



2.3.3 简单的I/O 格式控制

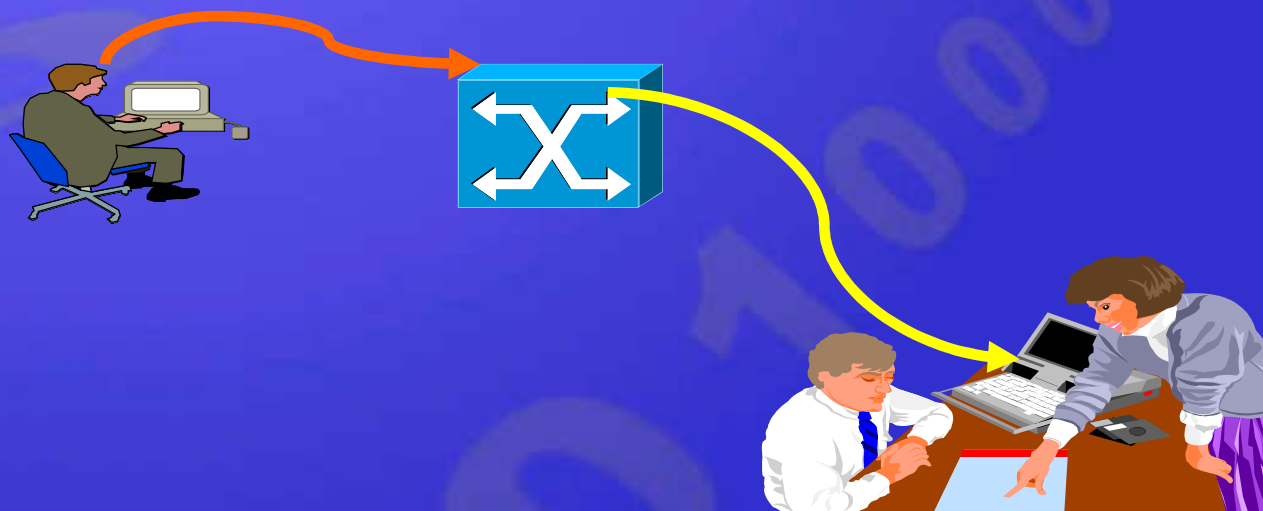
常用的I/O流类库操纵符

操纵符名	含 义
dec	数值数据采用十进制表示
hex	数值数据采用十六进制表示
oct	数值数据采用八进制表示
ws	提取空白符
endl	插入换行符，并刷新流
ends	插入空字符
setprecision(int)	设置浮点数的小数位数字 (包括小数点)
setw(int)	设置域宽

例: `cout << setw(5) << setprecision(3) << 3.1415;`



4. 算法的基本控制结构



2.4.1 用if语句实现选择结构

——单分支

if语句用于在程序中有条件地执行某一语句序列，它有如下单分支和双分支两种基本语法格式。

单分支形式如下所示：

if(条件表达式) 单条语句;

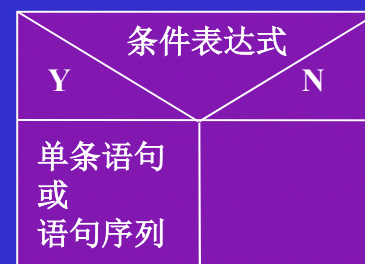
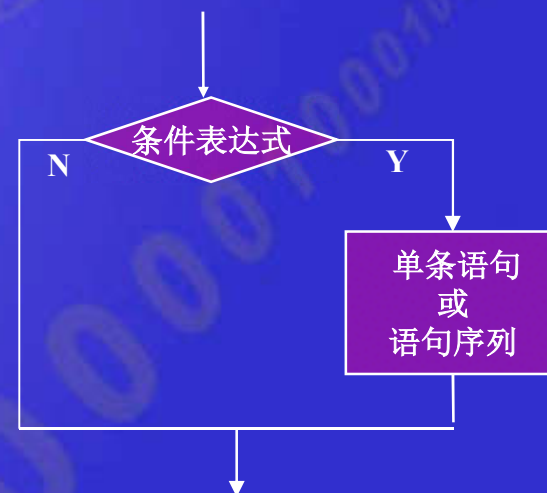
或者：

if(条件表达式)

{

语句序列;

};



2.4.1 用if语句实现选择结构

——双分支

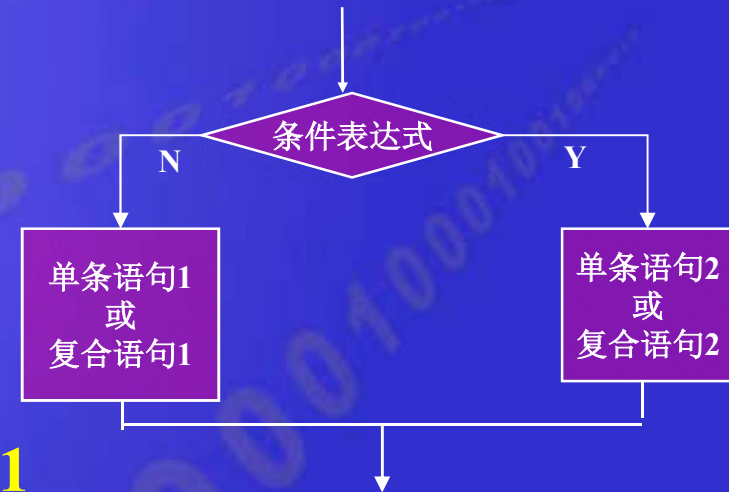
if语句双分支形式如下所示:

if (条件表达式)

单条语句1 或者 复合语句1

else

单条语句2 或者 复合语句2



条件表达式	
Y	N
单条语句1 或 复合语句1	单条语句2 或 复合语句2



2.4.1 用if语句实现选择结构

例2-2 输入一个年份，判断是否闰年

```
//2_2.cpp
#include <iostream>
using namespace std;
int main() {
    int year;
    bool isLeapYear;
    cout << "Enter the year: ";
    cin >> year;
    isLeapYear = ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0));
    if (isLeapYear)
        cout << year << " is a leap year" << endl;
    else
        cout << year << " is not a leap year" << endl;

    return 0;
}
```

2.4.1 用if语句实现选择结构

例如：grade表示百分制的成绩，将grade转换为五分制，转换规则是：100分对应“A++”，低于60分对应“F”，60~69对应“D”，70~79对应“C”，80~89对应“B”，90~99对应“A”。

```
const vector<string> scores = {"F", "D", "C", "B", "A", "A++"};
```

```
string lettergrade; //定义五分制成绩的变量
```

```
//.....定义grade，读入grade数据
```

```
if ( grade < 60 )
```

```
    lettergrade = scores[0];
```

```
else
```

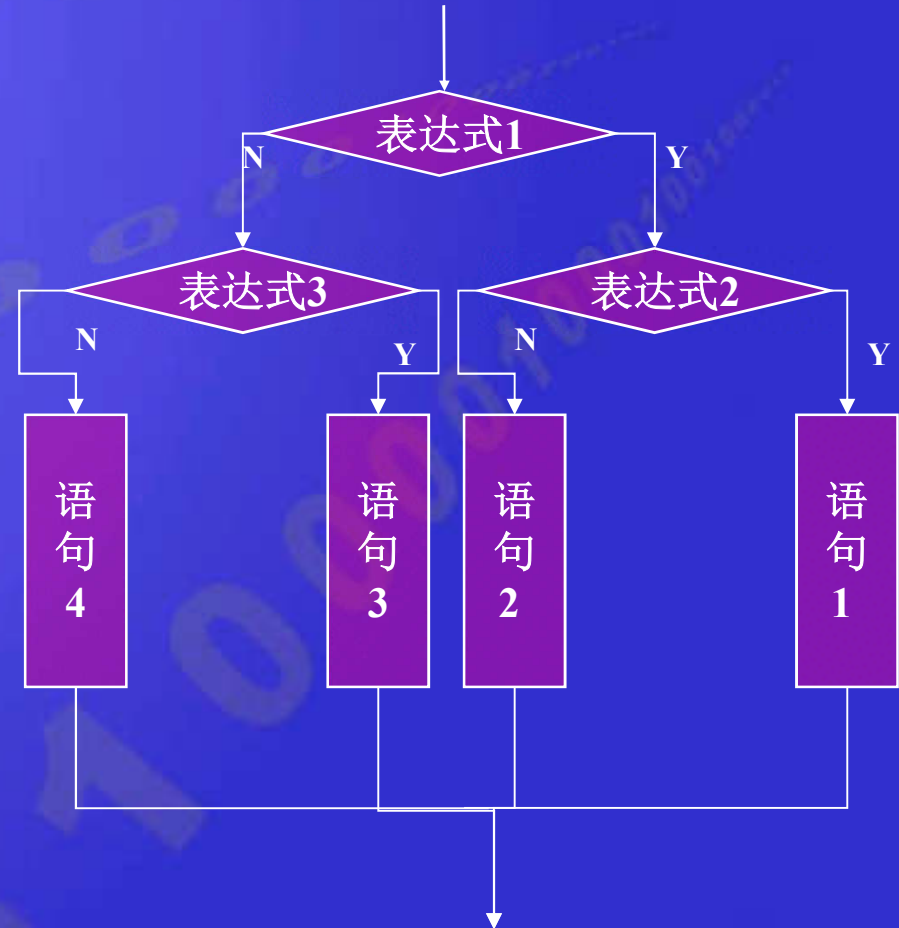
```
    lettergrade = scores[(grade-50)/10];
```



2.4.2、多重选择结构——嵌套的if语句

嵌套if实例一

```
if(表达式1)
    if(表达式2)
        语句1
    else
        语句2
else
    if(表达式3)
        语句3
    else
        语句4
```



2.4.2 多重选择结构——嵌套的if结构

```
#include<iostream>
using namespace std;
int main() {
    int x, y;
    cout << "Enter x and y:";
    cin >> x >> y;
    if (x != y)
        if (x > y)
            cout << "x > y" << endl;
        else
            cout << "x < y" << endl;
    else
        cout << "x = y" << endl;
    return 0;
}
```

例2-3：输入两个整数，比较两个数的大小。

运行结果1：

Enter x and y:5 8

x < y

运行结果2：

Enter x and y:8 8

x = y

运行结果3：

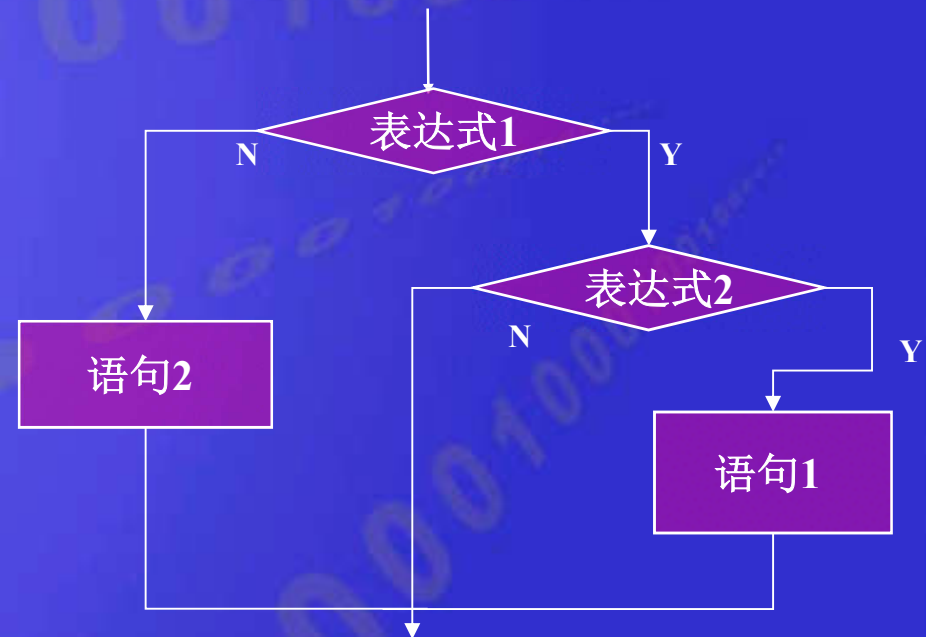
Enter x and y:12 8

x > y

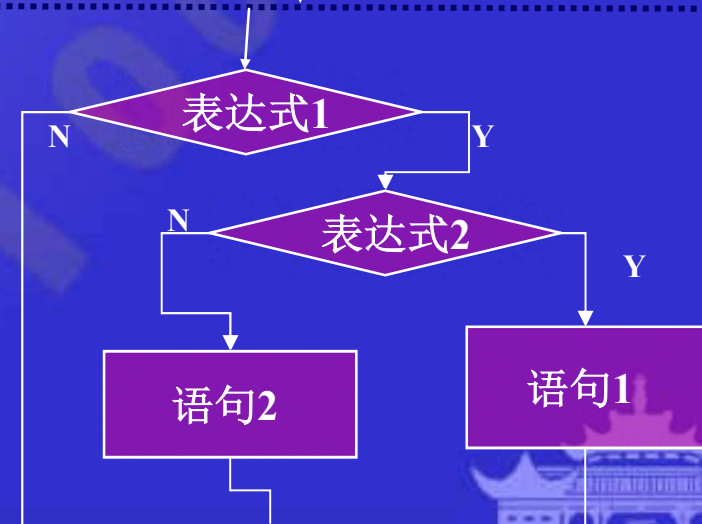


2.4.2 多重选择结构——嵌套的if结构

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



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



2.4.2 多重选择结构——*if...else if*语句

*if...else if*语句

*if...else if*语句用于进行多重判断，其语法形式如下：

if (条件表达式1) 语句1;

else if (条件表达式2) 语句2;

...

else if (条件表达式*n*) 语句*n*;

else 语句*n*+1;



2.4.2 多重选择结构——*if...else if*语句

【例】 以下程序将用户输入的分数转换成等级：A (≥ 90)，B (80~89)，C (70~79)，D (60~69)，E (< 60)。

```
#include <iostream>
using namespace std;
int main()
{
    float x;
    cout << "分数: "; cin >> x;
    if (x >= 90) cout << "A" << endl;
    else if (x >= 80) cout << "B" << endl;
    else if (x >= 70) cout << "C" << endl;
    else if (x >= 60) cout << "D" << endl;
    else cout << "E" << endl;
    return 0;
}
```



2.4.2 多重选择结构——*switch* 语句

```
switch(expression){  
    case constant1:  
        statement sequence  
        break;  
    case constant2:  
        statement sequence  
        break;  
    .....  
    default:  
        statement sequence  
}
```

1. **break**: 跳出case分支的跳转语句，必不可少。
3. **expression**: 字符型、枚举类型或整型表达式；
4. **case constant**: case后面只能为常量表达式；各个case常量必须各异。
5. 当表达式的值与case后面的常量表达式值相等时就执行此case后面的语句。
6. case只能判断相等。遇第一个相等的case常量分支之后，顺序向下执行，不再进行相等与否的判断。

2.4.2 多重选择结构——*switch* 语句

- ◆ 每个case语句只是一个入口标号，并不能确定执行的终止点，因此每个case分支的最后应该加break语句，用来结束整个switch结构，否则会从入口点开始一直执行到switch结构的结束点。
- ◆ 当若干分支需要执行相同操作时，可以使多个case分支共用一组语句。

如：

case 1:

case 2:cout<<“case1和2”<<endl;

注意：可以使用break退出switch语句的执行。如：

case 1:cout<<“case1”<<endl; break;

case 2: cout<<“case2”<<endl ;



2.4.2 多重选择结构——switch语句

补充例题：switch语句使用错误范例。

//break语句在switch中的应用，错误版本。

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

```
int main(void)
{
    int x;

    cin>>x;

    switch(x) {
        case 5: cout<<"excellent";
        case 4: cout<<"Good";
        case 3: cout<<"Pass";
        case 2: cout<<"Fail";
        default: cout<<"Poor";
    }

    return 0;
}
```

Why?

3 ✓
PassFailPoor

2.4.2 多重选择结构—— switch 语句

例题：switch 语句正确使用范例。

//break 语句在 switch 中的应用，正确版本。

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

```
int main(void)
{
```

```
    int x;
    cin>>x;
```

```
    switch(x) {
```

```
        case 5: cout<<"excellent"; break;
```

```
        case 4: cout<<"Good"; break;
```

```
        case 3: cout<<"Pass"; break;
```

```
        case 2: cout<<"Fail"; break;
```

```
        default: cout<<"Poor";
```

```
    }
```

```
    return 0;
```

```
}
```

3 ✓
Pass

2.4.2 多重选择结构——*switch* 语句

例2-4：输入一个0~6的整数，转换成星期输出。

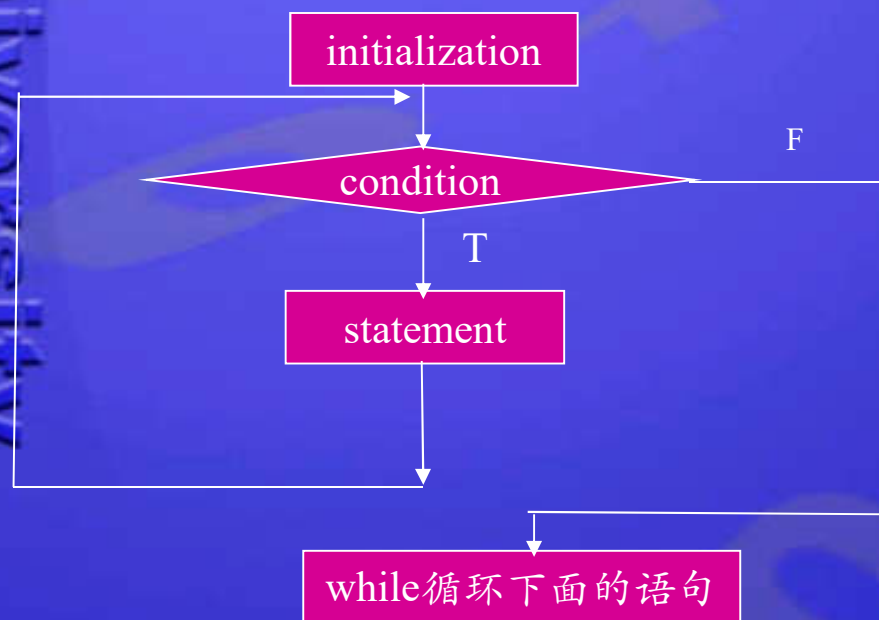


例2-4 (续)

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

2.4.3、循环结构——while语句

```
initialization;  
while(condition)  
    statement;
```



1. **initialization**: 初始化, 一般为赋值语句;
2. **condition**: 循环条件, 循环一直执行直到条件为假为止;
3. **statement**: 循环体, 单个语句、块语句、空语句;

2.4.3 循环结构——while语句

例2-5 求自然数1~10之和

分析：本题需要用累加算法，累加过程是一个循环过程，可以用while语句实现。



例2-5 (续)

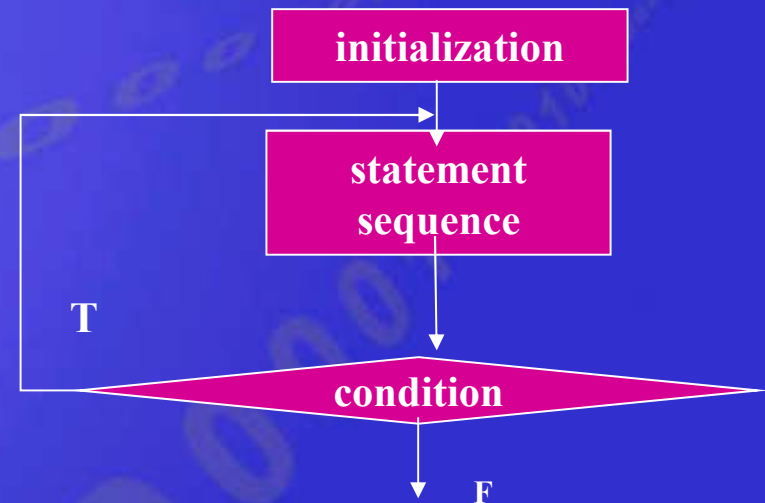
```
#include <iostream>
using namespace std;
int main() {
    int i = 1, sum = 0;
    while (i <= 10) {
        sum += i; //相当于sum = sum + i;
        i++;
    }
    cout << "sum = " << sum << endl;
    return 0;
}
```

运行结果：
sum = 55



2.4.3 循环结构——*do-while*语句

```
initialization;  
do{  
    statement sequence  
}while(condition);
```



1. **initialization**: 初始化，一般为赋值语句；
2. **condition**: 循环条件，循环一直执行直到条件为假为止；
3. **statement sequence**: 循环体，语句序列；



2.4.3 循环结构——do-while语句

```
#include <iostream>
using namespace std;
int main() {
    int n, right_digit, newnum = 0;
    cout << "Enter the number: ";
    cin >> n;
    cout << "The number in reverse order is ";
    do {
        right_digit = n % 10;
        cout << right_digit;
        n /= 10; //相当于n=n/10
    } while (n != 0);
    cout << endl;
    return 0;
}
```

例2-6：输入一个数，将各位数字翻转后输出

运行结果：

Enter the number: 365

The number in reverse order is 563

//2_7.cpp

```
#include <iostream>
```

```
using namespace std;
```

```
int main() {
```

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

```
    do {
```

```
        sum += i;
```

```
        i++;
```

```
    } while (i <= 10);
```

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

```
    return 0;
```

```
}
```

*例2-7用do-while语句编程，
求自然数1~10之和*



对比下面的程序

程序1：

```
#include <iostream>
using namespace std;
int main() {
    int i, sum = 0;
    cin >> i;
    while (i <= 10) {
        sum += i;
        i++;
    }
    cout<< "sum= " << sum
        << endl;
    return 0;
}
```

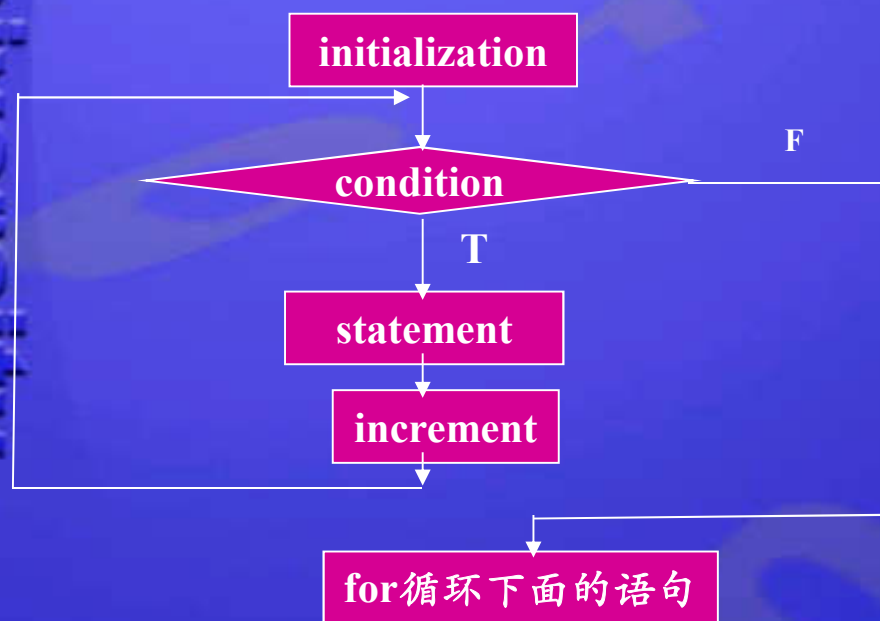
程序2：

```
#include <iostream>
using namespace std;
int main() {
    int i, sum = 0;
    cin >> i;
    do {
        sum += i;
        i++;
    } while (i <= 10);
    cout << "sum=" << sum
        << endl;
    return 0;
}
```



2.4.3 循环结构——传统的for语句

```
for(initialization; condition; increment)
    statement;
```



1. **initialization**: 初始化，一般为赋值语句；
2. **condition**: 循环条件，循环一直执行直到条件为假为止；condition缺省时表示true。
3. **statement**: 循环体，单个语句、块语句、空语句；
4. **increment**: 修改控制变量。



2.4.3 循环结构——传统的for语句

最好不要用浮点数类型的变量作为循环控制变量:

例如, 以下的循环结构可能出现死循环现象

```
for(x=0; x!=12.3;)
```

```
cin>>x;
```

为什么? 因为计算机中的浮点数存在天生的可表示误差, 换言之, 计算机表示的是有限位数的实数!



2.4.3 循环结构——传统的for语句

**例2-8：输入一个整数，
求出它的所有因子。**

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

    cout << "Enter a positive integer: ";
    cin >> n;
    cout << "Number " << n << " Factors ";

    for (int k = 1; k <= n; k++)
        if (n % k == 0)
            cout << k << " ";
    cout << endl;
    return 0;
}
```

运行结果1：

Enter a positive integer: 36

Number 36 Factors 1 2 3 4 6 9 12 18 36

运行结果2：

Enter a positive integer: 7

Number 7 Factors 1 7

2.4.4 循环结构与选择结构的嵌套

举例

```
#include <iostream>
using namespace std;
int main() {
    for (int n = 100; n <= 200; n++) {
        if (n % 3 != 0)
            cout << n;
    }
    return 0;
}
```



2.4.4 循环结构与选择结构的嵌套

例2-10 读入一系列整数，统计出正整数个数 i 和负整数个数 j ，读入0则结束。

分析：

- ✓ 需要读入一系列整数，但是整数个数不定，要在每次读入之后进行判断，因此使用while循环最为合适。循环控制条件应该是 $n \neq 0$ 。由于要判断数的正负并分别进行统计，所以需要在循环内部嵌入选择结构。



例2-10 (续)

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

int main() {
    int i = 0, j = 0, n;
    cout << "Enter some integers please (enter 0 to quit): " << endl;
    cin >> n;
    while (n != 0) {
        if (n > 0) i += 1;
        if (n < 0) j += 1;
        cin >> n;
    }
    cout << "Count of positive integers: " << i << endl;
    cout << "Count of negative integers: " << j << endl;
    return 0;
}
```

2.4.5 循环结构——范围for语句

范围for语句，C++11新引入的更简单的for语句，用于遍历容器或其他序列的所有元素。其一般形式为：

for (declaration : expression) 语句;

其中，**expression**必须是一个序列，例如花括号括起来的初始值列表、数组、**vector**、**string**等类型的对象。

declaration定义一个变量，这个变量就是循环变量。最简单的方法就是用**auto**定义循环变量。

范围for循环的执行过程为：每次迭代都重新定义循环变量；并将其初始化为序列中的下一个值；然后执行语句；知道**expression**中所有元素都处理完毕。



2.4.5 循环结构——范围for语句

例如，以下的代码段的功能是把v的每个元素值都翻倍：

```
vector v = {0,1,2,3,4,5,6,7,8,9};
```

```
//如果需要对v的元素执行写操作，
```

```
//那么范围for的循环变量必须是引用类型
```

```
for( auto &r : v ) //对于v的每个元素
```

```
    r *= 2;
```

等价于传统for语句：

```
for( auto beg = v.begin(), end = v.end(); beg != end; ++ beg){
```

```
    auto &r = *beg;    r *= 2;
```

```
}
```



2.4.6、其他控制语句——*break*语句

`break`语句用于终止离它最近的`while`、`do while`、`for`或`switch`语句，从这些语句之后的第一条语句开始执行。例如：

```
string buf;
while (cin >> buf && !buf.empty() ){
    switch(buf[0]){
        case '-': //处理到第一个空白为止
            for( auto it = buf.begin() +1; it != buf.end(); ++it){
                if( *it == ' ')
                    break; // #1 离开for循环
            }
            //break #1将控制权转移到这里
            //剩余的 '-' 处理
            break; // #2, 离开switch语句
        case '+': //...
    } //结束switch语句
    // break #2将控制权转移到这里
} // 结束while
```



2.4.5 循环结构——*continue*语句

`continue`语句用于终止最近的循环中的当前迭代，并立刻开始下一次迭代。`continue`仅用于`while`、`do while`和`for`语句内部。例如，下面的代码段功能是：每次从标准输入读取一个单词，循环中只对以下画线开头的单词感兴趣：

```
string buf;  
while (cin >> buf && !buf.empty() ){  
    if ( buf[0] != '_' )  
        continue;  
    //...处理以下画线开头的单词  
} // 结束while
```



2.4.5 循环结构——goto 语句

goto 语句实现从 goto 语句无条件跳转到同一个函数内的另一条语句处。其一般形式是：

goto label;

其中，label 是语句标号，由标识符和冒号组成。

例如：

begin:

int sz = get_size();

if (sz < 0)

goto begin;



现在开始课堂练习！

练习内容：选择结构、循环结构



用if语句实现选择结构

【例】 以下错误的语句为_____。

- A. `if (x>y);`
- B. `if (x=y)&&(x!=0) x+=y;`
- C. `if (x!=y) cin>>x; else cin>>y;`
- D. `if (x<y) {x++; y++;}`

答：if语句的条件必须包含在一个括号中。本题答案为：B。



多重选择结构——嵌套的if结构

【例】为了避免在嵌套的条件语句if-else中产生二义性，C++语言规定：else子句总是与_____配对。

A.缩排位置相同的if

B.同一行上的if

C.其之后最近的if

D.其之前最近的if

答：本题答案为：D。



多重选择结构——嵌套的if结构

【例】 以下程序的输出结果是_____。

```
#include <iostream>
```

```
int main()
```

```
{ int x=2,y=-1,z=2;
```

```
  if(x<y)
```

```
    if(y<0) z=0;
```

```
    else z+=1;
```

```
    std::cout << z << std::endl;
```

```
    return 0;
```

```
}
```

A.3

B.2

C.1

D.0

答： $x < y$ 为假，直接执行cout语句。

本题答案为：B。



多重选择结构——switch 语句

【例】有以下程序：

```
#include <iostream>
using namespace std;
int main()
{ int a=15,b=21,m=0;
  switch(a%3)
  {
    case 0:m++;break;
    case 1:m++;
      switch(b%2)
      {
        default:m++;
        case 0:m++;break;
      }
  }
  cout << m << endl;
  return 0;
}
```

程序运行后的输出结果是__。

- A. 1 B. 2
C. 3 D. 4

答：a%3=0，执行m++和cout语句。本题答案为：A。



多重选择结构—— switch 语句

例如：输入一段文字，分别统计其中五个小写元音字母出现的次数。代码段如下：

```
unsigned aCnt=0, eCnt=0, iCnt=0, oCnt=0, uCnt=0;
char ch;
while ( cin >> ch){
    switch (ch){
        case 'a': ++aCnt; break;
        case 'e': ++eCnt; break;
        case 'i': ++iCnt; break;
        case 'o': ++oCnt; break;
        case 'u': ++uCnt; break;
```



多重选择结构——*switch* 语句

例如：输入一段文字，分别统计其中五个小写元音字母出现的次数。代码段如下：

```
cout << "Number of vowel a: \t" << aCnt << '\n'  
    << "Number of vowel e: \t" << eCnt << '\n'  
    << "Number of vowel i: \t" << iCnt << '\n'  
    << "Number of vowel o: \t" << oCnt << '\n'  
    << "Number of vowel u: \t" << uCnt << '\n';
```



循环结构——while语句

【例】设有以下程序段：

```
int x=0,s=0;
```

```
while(!x!=0) s+=++x;
```

```
std::cout << s << std::endl;;
```

则_____。

A.运行程序段后输出0

B.运行程序段后输出1

C.程序段中的控制表达式是非法的

D.程序段执行无限次

答：x=0，!x值是1，!x!=0为真，执行s+=++x，++x返回1，x=1，s=s+1=1；x=1，!x是0，!x!=0为假，不再执行循环语句。本题答案为：B。



循环结构——do-while语句

【例】有以下程序段：

```
int n=0,p;  
do {  
    std::cin>>p; n++;  
} while(p!=12345 && n<3);
```

此处do-while循环的结束条件是__。

- A. p的值不等于12345并且n的值小于3
- B. p的值等于12345并且n的值大于等于3
- C. p的值不等于12345或者n的值小于3
- D. p的值等于12345或者n的值大于等于3

答：do-while循环的结束条件为! (p!=12345 && n<3)，即p==12345 || n>=3。

本题答案为：D。



循环结构——传统的for语句

【例】以下程序的输出结果是__。

```
#include <iostream>
int main()
{ int a=0,i;
  for(i=1;i<5;i++)
  { switch(i)
    {
      case 0:
      case 3: a+=2;
      case 1:
      case 2: a+=3;
      default: a+=5;
    }
  }
  std::cout << a << std::endl;
  return 0;
}
```

A.31

B.13

C.10

D.20

答：

i=1，执行a+=3和a+=5语句，
a=8；

i=2，执行a+=3和a+=5语句，
a=16；

i=3，执行a+=2、a+=3和a+=5
语句，a=26；

i=4，执行a+=5语句，a=31。

本题答案为：A。



循环结构——传统的for语句

【例】下面程序的功能是：计算1到10之间奇数之和及偶数之和，请填空。

```
#include <iostream>
```

```
int main()
```

```
{    int a,b,c,i;
```

```
    a=c=0;
```

```
    for (i=0;i<10;i+=2)
```

```
    {    a+=i;
```

```
        _____;
```

```
        c+=b;
```

```
    }
```

```
    std::cout << "偶数之和=" << a << std::endl;
```

```
    std::cout << "奇数之和=" << c << std::endl;
```

```
    return 0;
```

```
}
```

答：for循环中i扫描所有偶数，b扫描所有奇数。本题答案为：**b=i+1。**



本次课堂练习结束!



5. 类型别名与类型推断



什么是自定义数据类型

C++语言不仅有丰富的系统预定义的基本数据类型，而且允许用户进行数据类型的自定义。自定义的数据类型有结构类型、联合类型、枚举类型、数组类型和类等。



2.5.1、typedef声明——定义类型别名

```
typedef type name;
```

定义类型别名：

- type: 原有数据类型名;
- name: 类型别名;
- 仅仅定义类型别名, 没有定义新的数据类型。

例 **INTEGER a,b,c;**
REAL f1,f2;

```
typedef int INTEGER;  
typedef float REAL;
```



```
int a,b,c;  
float f1,f2;
```

说明:

- 1.typedef 没有创造新数据类型
- 2.typedef 是定义类型,不能定义变量
- 3.typedef 与 define 不同

2.5.1、typedef声明——定义步骤

typedef定义类型步骤

- ① 按定义变量方法先写出定义体 如 `int i;`
- ② 将变量名换成新类型名 如 `int INTEGER;`
- ③ 最前面加typedef 如 `typedef int INTEGER;`
- ④ 用新类型名定义变量 如 `INTEGER i,j;`



2.5.1、typedef声明

例 定义数组类型

- ① `int a[100];`
- ② `int ARRAY[100];`
- ③ `typedef int ARRAY[100];`
- ④ `ARRAY a,b,c;`

\Leftrightarrow `int a[100],b[100],c[100];`



2.5.1、typedef声明

例 定义函数指针类型

① `int (*p)();`

② `int (*POWER)();`

③ `typedef int (*POWER)();`

④ `POWER p1,p2;`

\Leftrightarrow `int (*p1)(),(*p2)();`



2.5.1、typedef声明

例 定义结构体类型

```
① struct date  
{ int month;  
  int day;  
  int year;  
}d;
```

例 定义结构体类型

```
② struct date  
{ int month;  
  int day;  
  int year;  
}DATE;
```



2.5.1、typedef声明

例 定义结构体类型

```
③ typedef struct date  
    { int month;  
      int day;  
      int year;  
    }DATE;
```

例 定义结构体类型

```
④ DATE birthday, *p;
```

```
⇔ struct date  
    { int month;  
      int day;  
      int year;  
    }birthday, *p;
```



2.5.1、typedef声明

```
例 typedef struct club
{   char name[20];
    int size;
    int year;
}GROUP;
typedef GROUP *PG;
PG pclub;
```

GROUP为结构体类型
PG为指向GROUP的指针类型

⇔ GROUP *pclub;
⇔ struct club *pclub;



2.5.1 、 *typedef*声明——C++11新用法

方法二，C++11的新方法：别名声明

using 别名 = 类型名;

例如：

using SI = Sales_item; //SI是Sales_item的别名



2.5.1、typedef声明——C++11新用法

需注意的问题：当类型别名指代复合类型或常量时，那么把类型别名用到声明语句中时，可能产生意想不到的后果。

例如，类型别名指代的是指针时：

```
typedef char *pstring;
```

```
const pstring cstr = 0;
```

错误的理解：const char * cstr=0;

pstring本身是指针，const修饰指针，而不是char的。

正确的理解：cstr是指向char类型的常量指针！而不是指向常量字符的指针！

```
const pstring *ps;
```

ps是指针！*ps（也就是ps指向的对象）是指向char类型的常量指针！

2.5.2 *auto* 类型与 *decltype* 类型

auto 类型说明符：由编译器分析表达式类型。例如：

auto val=val1+val2; //val的类型由val1+val2决定

auto i=0,j=1; //正确：i、j都是int类型

auto size=0,pi=3.14; //错误：size和pi类型不一致

decltype:当我们定义一个变量与某一表达式类型相同，但是不想用该表达式初始化该变量时使用。例如：

decltype (i) j=2; //j初始值为2，类型和i一致



6. 深度探索



2.6.1 变量的实现机制

站在目标代码的角度上看问题

变量具有两个重要属性——数据类型和变量名

- ✓ 它们都用文字表示，便于人们理解，但不便于CPU识别，因此它们不能出现在目标代码中；
- ✓ 下面将讨论它们在目标代码中的表示方式。



变量名

源代码中，变量依靠变量名来标识；

- ✓ 目标代码中，变量依靠地址来标识，每个变量的地址互不相同。

```
int a, b;  
int main() {  
    a++;  
    b++;  
    return 0;  
}
```



```
incl 0x80495f8  
incl 0x80495fc
```

地址



数据类型

任何数据在内存中都是用二进制串的形式表示的；

✓ 一串二进制数，只有确定了类型，才有真实的含义。

✓ 例：10111111,10000000,00000000,00000000

解释为int型：-1,082,130,432

解释为unsigned型：3,212,836,864

解释为float型：-1.0



目标代码体现数据类型的方式

类型的特性蕴含于操作之中；

- ✓ 对不同数据类型，源代码中形式上相同的操作，会被转化为目标代码中的不同操作。

```
int a;  
short b;  
char c;  
int main() {  
    a++;  
    b++;  
    c++;  
    return 0;  
}
```



```
incl 0x80495f8  
incw 0x80495fc  
incb 0x80495fe
```

不同的操作



变量的声明和定义

变量的声明

- ✓ 将一个标识符声明为变量，告诉编译器这个标识符表示变量，同时还指出了它的数据类型；
- ✓ 只有确定了一个变量的数据类型，变量参与的操作才具有完整的意义，编译器才能将该变量参与的表达式翻译为合适的操作。

变量的定义

- ✓ 不仅确定了一个标识符表示变量，以及该变量的类型，还确定了变量地址的分配位置。



2.6.2 C++表达式的执行原理

什么是寄存器？

- ✓ CPU内部的存储单元
- ✓ 读写速度非常快
- ✓ 数量很少

IA-32有eax、ebx、ecx、edx、esp、ebp、esi、edi八个通用寄存器

大部分CPU指令都需要读写寄存器。

复杂的表达式，分步执行，每条指令只能做一次基本运算，中间结果暂存在寄存器中。



2.6.2 C++表达式的执行原理 (续)

```
int a, b, c, d;
```

```
int e;
```

```
int main() {
```

```
    a = 4;
```

```
    b = 2;
```

```
    c = 1;
```

```
    d = 10;
```

```
    e = (a + b) * (c - d);
```

```
    return 0;
```

```
}
```

把4存入地址0x80495d8中

```
movl $0x4,0x80495d8
```

```
movl $0x2,0x80495dc
```

```
movl $0x1,0x80495e0
```

```
movl $0xa,0x80495e4
```

0x80495d8: a变量

0x80495dc: b变量

0x80495e0: c变量

0x80495e4: d变量

0x80495e8: e变量



2.6.2 C++表达式的执行原理 (续)

```
int a, b, c, d;
```

```
int e;
```

```
int main() {
```

```
    a = 4;
```

```
    b = 2;
```

```
    c = 1;
```

```
    d = 10;
```

```
    e = (a + b) * (c - d);
```

```
    return 0;
```

```
}
```

$ecx \leftarrow ecx + eax$

$eax \leftarrow eax - eax$

0x80495d8: a变量

0x80495dc: b变量

0x80495e0: c变量

0x80495e4: d变量

0x80495e8: e变量

把0x80495dc地址的值到eax寄存器中

mov 0x80495dc,%eax

mov 0x80495d8,%ecx

add %eax,%ecx

mov 0x80495e4,%edx

mov 0x80495e0,%eax

sub %edx,%eax

imul %ecx,%eax

mov %eax,0x80495e8

$eax \leftarrow (eax * eax)$ 的低32位



2.6.3 数值类型的计算误差问题

计算误差问题：受字长限制，计算机无法表示无限位数的整数或实数；实数还存在表示精度的问题。一旦操作数过大、过小或者两个操作数相差过大，可能产生不准确或者完全错误的结果。

计算误差的类别：

- ◆ **上溢：**运算结果过大，超过数据类型的可表示范围之外的一种错误状态
- ◆ **下溢：**数据小于最小可表示数值时产生的一种错误状态；整数不会下溢；
- ◆ **可表示误差：**计算机无法表示无限数位的实数



2.6.3 数值类型的计算误差问题

例题 整型数据溢出示例，32767加1运算。

```
//整型数据溢出范例。  
#include <iostream>  
using namespace std;  
int main(void)  
{
```

```
    short int a,b;
```

```
    a=32767;
```

```
    b=a+1;
```

```
    cout<<"a="<<a<<" , b="<<b<<
```

```
    return 0;
```

```
    }/*main函数结束*/
```

WHY?

32767+1 结
果为-32768

解答:

32767	0111 1111 1111 1111
+ 1	0000 0000 0000 0001
	1000 0000 0000 0000

a=32767,b=-32768

浮点数的可表示误差

例题 浮点数可表示误差示例。

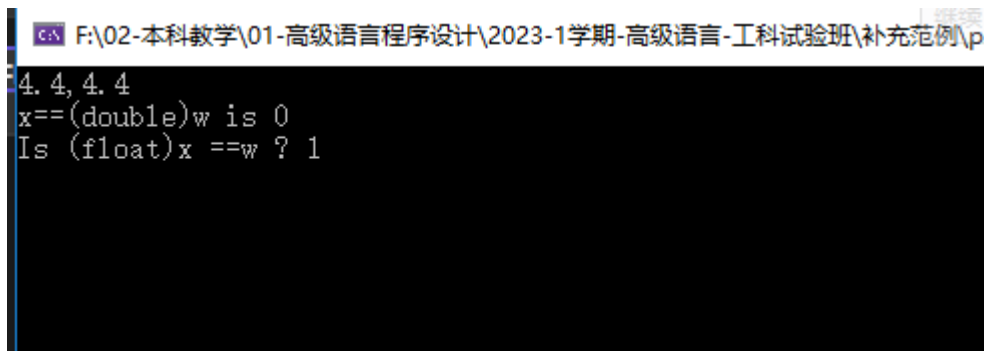
//浮点数可表示误差范例。源文件：LT2-6-1.cpp

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

```
int main(void)
{
    float w = 4.4;
    double x = 4.4;
```

```
    cout<<x<<","<<w<<endl;
    cout<<"x==(double)w is " <<(x == (double)w)<<endl;
    cout<<"Is (float)x ==w ? " <<((float)x == w)<<endl ;
```

```
    cin.ignore ();
    return 0;
}/*main函数结束*/
```



The screenshot shows a terminal window with the following output:

```
4.4, 4.4
x==(double)w is 0
Is (float)x ==w ? 1
```

浮点数的可表示误差

例题 浮点数可表示误差示例。

引出一个问题：应该如何比较浮点数是否相等？

○解答是：如果两个浮点数之间的差值非常小，小于问题的精度要求，就可认为二者相等。

例如，判断变量w和x是否相等，可以用表达式：

`fabs(w - x) < epsilon`

其值，**`fabs()`**是计算绝对值的库函数，**`epsilon`**是自行设定的一个误差值，例如**`1e-4`**。



浮点数的上溢

例题 浮点数上溢示例，计算35的阶乘。

//浮点数上溢范例，计算35的阶乘。

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

```
int main(void)
```

```
{
```

```
int n;
```

```
float factf = 1.0;
```

```
for (n = 1; n <= 35; n++)
```

```
factf = factf * n;
```

```
cout<<" 35的阶乘是"<< factf<<endl ;
```

```
return 0;
```

```
/*main函数结束*/
```

Microsoft Visual Studio 调试控制台

35的阶乘是inf

F:\02-本科教学\01-高级语言程序设计\2023-1学期-高级语言程序设计 12248)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“按任意键关闭此窗口...”

浮点数的上溢

例题 浮点数上溢示例，计算35的阶乘。

//浮点数上溢范例，计算35的阶乘。

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

```
int main(void)
```

```
{
```

```
int n;
```

```
double factf = 1.0;
```

```
for (n = 1; n <= 35; n++)
```

```
factf = factf * n;
```

```
cout<<" 35的阶乘是"<< factf<<endl ;
```

```
return 0;
```

```
}/*main函数结束*/
```

Microsoft Visual Studio 调试控制台

35的阶乘是1.03331e+40

F:\02-本科教学\01-高级语言程序设计\2023-1学期-高级语言程序设计 8088)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“在调试停止时自动关闭控制台”。
按任意键关闭此窗口。...

浮点数的下溢

例题 浮点数下溢示例，不断用1.0除以10，显示 10^{-43} 到 10^{-47} 的结果。

//浮点数下溢范例，计算10的-43~-47次方。

```
#include <iostream>
using namespace std;
int main(void){
    int n;
    float frac = 1.0;
    for (n = 1; n <= 42; n++)
        frac = frac / 10;
    for (n = 43; n <= 47; n++) {
        frac = frac / 10;
        cout<< n<<"\t"<< frac<<endl;
    }
    return 0;
}/*main函数结束*/
```

Microsoft Visual Studio 调试控制台

```
43      9.94922e-44
44      9.80909e-45
45      1.4013e-45
46      0
47      0
```

F:\02-本科教学\01-高级语言程序
程 13460) 退出，代码为 0。
要在调试停止时自动关闭控制台，
按任意键关闭窗口...

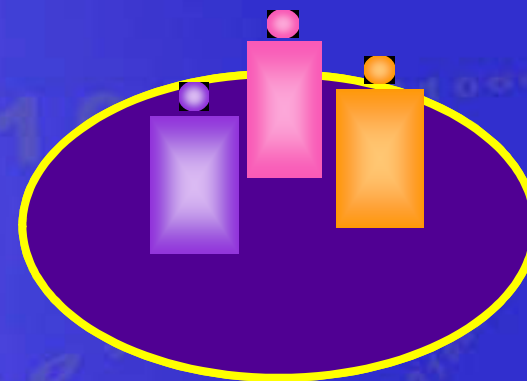
结果为0，浮点数下溢

可计算误差小结：数据类型的选择

C++语言中那么多数据类型，我该定义变量为哪种类型？

- 合适的范围：足够表示所有可能出现的数据取值；
- 满足精度要求；
- 尽量少占据内存空间：满足上述条件的前提下，尽可能少占据内存空间。





本讲小结



本讲主要知识点

1. 主要内容

- ① C++语言概述
- ② 基本数据类型和表达式
- ③ 数据的输入与输出
- ④ 算法的基本控制结构
- ⑤ 自定义数据类型

2. 达到的目标：掌握C++语言的基本概念和基本语句，能够编写简单的程序段。



第2讲 课后练习

作业：

2-23、若 $a=1, b=2, c=3$ ，下列各式的结果是什么？

- | | |
|----------------|-----------------|
| (1) $a b-c$ | (2) $a^b \& -c$ |
| (3) $a \& b c$ | (4) $a b \& c$ |

2-24、若 $a=1$ ，下列各式的结果是什么？

- | | | | |
|------------|----------------|-----------|--------------|
| (1) $!a a$ | (2) $\sim a a$ | (3) a^a | (4) $a >> 2$ |
|------------|----------------|-----------|--------------|

2-28、用穷举法找出1~100的质数并显示出来。分别用while、do...while、for循环实现。

2-31、在程序中定义一个整型变量，赋以1~100的值，要求用户猜这个数，比较两个数的大小，把结果提示给用户，直到猜对为止。分别用while、do...while、for循环实现。



第2讲 课后练习

作业：

2-27 编写一个完整的程序，运行时向用户提问“你考试考了多少分？（0~100）”，接收输入后判断其等级显示出来。规则如下：

- 1) 如果高于85分，则输出“优”；
- 2) 如果低于85但高于75，则输出“良”；
- 3) 如果低于75但高于60，则输出“中”；
- 4) 如果低于60分，则输出“差”。



第2讲 课后练习

第五版

作业： 2-23、2-24、2-27、2-28、
2-31

建议自学的课后练习：

2-2、2-11、2-14、2-17、2-18、
2-19、2-22、2-32、2-33



本讲结束

