第二章 C++简单程序设计

清华大学　郑　莉

本章导学

问题与解决

- 计算机的最基本功能是数据处理，因此：
  - 程序要有能力表示与存储各种类型的数据
  - 程序要能够进行各种运算
- C++支持的基本数据类型：整数、实数、字符、逻辑数据
- C++支持的基本运算：算术运算、逻辑运算
- 程序的执行流程不总是顺序的，因此
  - 程序要能够对执行流程进行选择（选择语句、开关语句）
  - 程序要能够用反复同一算法依次处理大批量数据（循环语句）
- 基本数据类型能够表示的数据种类很有限，因此
  - 需要能够在程序中自定义类型（第4章讲类）
  - 枚举类型就是通过列出所有可取值，来定义一种新类型

学习要求

- 学习视频课件
- 阅读教材第2章
- 完成实验二
- 完成在线选择题和编程题
- 将例题在编译环境中调试观察运行情况，并试着修改

目录

C++的特点和程序实例

2.1.1-2.1.3

2.1.1 C++的产生和发展

- C++从C语言发展演变而来，最初的C++被称为“带类的C”；
- 1983年正式取名为C++；
- 于1998年11月被国际标准化组织（ISO）批准为国际标准；
- 2003年10月15日发布第2版C++标准ISO/IEC 14882:2003；
- 2011年8月12日ISO公布了第三版C++标准C++11，C++11标准包含核心语言的新机能，而且扩展C++标准程序库。
- 2014年8月18日ISO公布了C++14，其正式名称为"International Standard ISO/IEC 14882:2014(E) Programming Language C++"。C++14旨在作为C++11的一个小扩展，主要提供漏洞修复和小的改进。

2.1.2 C++的特点

- 兼容C
  - 它保持了C的简洁、高效和接近汇编语言等特点
  - 对C的类型系统进行了改革和扩充
  - C++也支持面向过程的程序设计，不是一个纯正的面向对象的语言
- 支持面向对象的方法
- 支持泛型程序设计方法

2.1.3 C++程序实例—例2-1

```
//2_1.cpp
#include <iostream>
using namespace std;
int main() {
        cout << "Hello!" << endl;
        cout << "Welcome to c++!" << endl;
        return 0;
}
```

运行结果：

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

C++字符集、词法记号、标识符

2.1.4——2.1.5

2.1.4 C++字符集

- 大小写的英文字母：A～Z，a～z
- 数字字符：0～9
- 特殊字符：

!	#	%	^	&	*	_	+	=	-	
~	<	>	/	\	'	"	;	.	,	:
?	(	)	[	]	{	}	|			

2.1.5词法记号

- 关键字　　C++预定义的单词
- 标识符　　程序员声明的单词，它命名程序正文中的一些实体
- 文字　　在程序中直接使用符号表示的数据
- 操作符　　用于实现各种运算的符号
- 分隔符 ()　{}　,　:　;
  用于分隔各个词法记号或程序正文
- 空白符　　空格、制表符（TAB键产生的字符）、垂直制表符、换行符、回车符和注释的总称

标识符的构成规则

- 以大写字母、小写字母或下划线(_)开始。
- 可以由以大写字母、小写字母、下划线(_)或数字0～9组成。
- 大写字母和小写字母代表不同的标识符。
- 不能是C++关键字或操作符。

基本数据类型、常量、变量

2.2.1——2.2.4

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

- 整数类型
  - 基本的整数类型
    - int
  - 按符号分
    - 符号的（signed）和无符号的（unsigned）
  - 按照数据范围分
    - 短整数（short）和长整数（long）、长长整数（ long long ）
  - char类型
    - 字符型，实质上存储的也是整数（详见字符类型）

2.2.1 基本数据类型

- 浮点数类型
  - 单精度
    - float
  - 双精度
    - double
  - 扩展精度
    - long double

2.2.1 基本数据类型

- 字符类型
  - char类型
  - 容纳单个字符的编码
- 字符串类型（详见第6章）
  - C风格的字符串
    - 采用字符数组
  - C++风格的字符串
    - 采用标准C++类库中的String类
- 布尔类型
  - bool类型，只有两个值：true（真）、false（假）
  - 常用来表示关系比较、相等比较或逻辑运算的结果

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	-2^{31}~2^{31}-1
unsigned int	4	0~2^{32}-1
long（signed long）	4	-2^{31}~2^{31}-1
unsigned long	4	0~2^{32}-1
long long	8	-2^{63}~2^{63}-1
unsigned long long	8	0~2^{64}-1
float	4	绝对值范围3.4×10^{-38}~3.4×10^{38}
double	8	绝对值范围1.7×10^{-308}~1.7×10^{308}
long double	8	绝对值范围1.7×10^{-308}~1.7×10^{308}

注：表中各类型的长度和取值范围，以面向IA32处理器的msvc12和2 gcc4.8为准

2.2.1 基本数据类型

- ISO C++标准并没有明确规定每种数据类型的字节数和取值范围，它只是规定它们之间的字节数大小顺序满足：
  - (signed/unsigned)signed char ≤ (unsigned) short int ≤ (unsigned) int ≤ (unsigned) long int ≤ long long int

2.2.2 常量

- 所谓常量是指在程序运行的整个过程中其值始终不可改变的量，也就是直接使用符号（文字）表示的值。例如：12，3.5，‘A’都是常量。

整数常量

- 以文字形式出现的整数；
- 十进制
  - 若干个0~9的数字，但数字部分不能以0开头，正数前边的正号可以省略。
- 八进制
  - 前导0+若干个0~7的数字。
- 十六进制
  - 前导0x+若干个0~9的数字及A~F的字母（大小写均可）。
- 后缀
  - 后缀L（或l）表示类型至少是long，后缀LL（或ll）表示类型是long long，后缀U（或u）表示unsigned类型。

浮点数常量

- 以文字形式出现的实数。
- 一般形式：
  - 例如，12.5，-12.5等。
- 指数形式：
  - 例如，0.345E+2，-34.4E-3
  - 整数部分和小数部分可以省略其一
- 浮点常量默认为double型，如果后缀F（或f）可以使其成为float型，例如：12.3f。

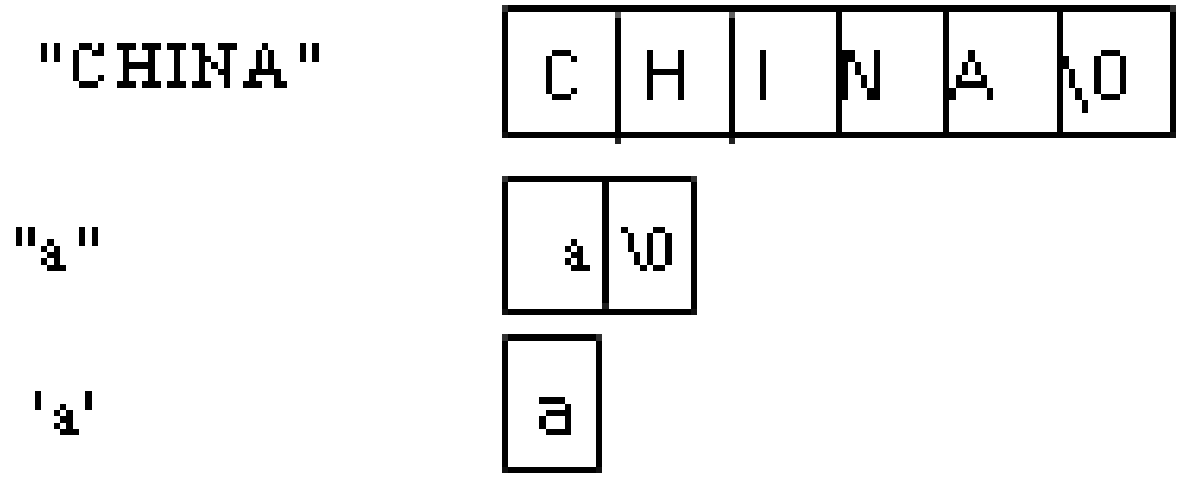
字符常量

- 单引号括起来的一个字符，如：'a', 'D', '?', '$ ‘
- C++转义字符列表（用于在程序中表示不可显示字符）

字符常量形式	ASCII码（十六进制）	含义
\a	07	响铃
\n	0A	换行
\t	09	水平制表符
\v	0B	垂直制表符
\b	08	退格
\r	0D	回车
\f	0C	换页
\\	5C	字符“\”
\”	22	双引号
\’	27	单引号
\?	3F	问号

C风格字符串常量

- 一对双引号括起来的字符序列
- 字符串与字符是不同的，它在内存中的存放形式是：按串中字符的排列次序顺序存放，每个字符占一个字节，并在末尾添加 ‘\0’ 作为结尾标记。



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

2.2.3 变量

- 变量定义：
  - 数据类型　变量名1, 变量名2, ..., 变量名n;
- 在定义一个变量的同时，也可以对它初始化。C++语言中提供了多种初始化方式，例如：
  - int a = 0;
  - int a(0);
  - int a = {0};
  - int a{0};
  - 其中使用大括号的初始化方式称为列表初始化，列表初始化时不允许信息的丢失。例如用double值初始化int变量，就会造成数据丢失。
- C++基本数据类型没有字符串类型，C++标准库中有string类（第6

2.2.4 符号常量

- 常量定义语句的形式为：
  - 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；
- 例如，我们可以定义一个代表圆周率的符号常量：
  - const float PI = 3.1415926；
- 注意，符号常量在定义时一定要赋初值，而在程序中间不能改变其值。

程序举例

2.2.4之后的

补充2-1：读入并显示整数

- 主要知识点：
  - 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
    - 在运行过程中从计算机的外部设备（例如键盘、硬盘）读取的，这些数据的值在程序运行过程中允许改变，这样的数据称为变量
  - 从键盘输入数据
    - iostream类的对象cin的>>操作，可以从标准输入设备（通常是键盘）读入数据
  - 数据的存储
    - 为了存储数据，需要预先为这些数据分配内存空间。
    - 变量的定义就是在给变量命名的时候分配内存空间。

补充2-1：读入并显示整数（续）

```
#include <iostream>
using namespace std;
int main()
{
    int radius;
    cout<<"Please enter the radius!\n";
    cin>>radius;
    cout<< “The radius is:” <<radius<< ‘\n’ ;
    cout<<"PI is:"<<3.14<<'\n
    cout<<"Please enter a different radius!\n";
    cin>>radius;
    cout<<"Now the radius is changed to:"
        <<radius<<'\n
    return 0;
}
//观察：通过调试功能跟踪观察变量的值
```

补充2-2：为常量命名

主要知识点：符号常量

```
#include <iostream>
using namespace std;
int main()
{   const double pi(3.14159);
     int radius;
     cout<<"Please enter the radius!\n";
     cin>>radius;
     cout<<"The radius is:"<<radius<<'\n';
     cout<<"PI is:"<<pi<<'\n';
     cout<<"Please enter a different radius!\n";
     cin>>radius;
     cout<<"Now the radius is changed to:"<<radius<<'\n';
     cout<<"PI is still:"<<pi<<'\n   ';
     //cin>>pi;
     return 0;
}
//观察：通过调试工具跟踪观察符号常量。
//思考：能给常量输入新值吗？如定义pi时不初始化会怎样？
```

补充2-2（续）

运行结果：

```
Please enter the radius!
2
The radius is:2
PI is:3.14159
Please enter a different radius!
3
Now the radius is changed to:3
PI is still:3.14159
```

补充2-3：变量的初始化

- 主要知识点：变量的初始化
  - 虽然变量的值是可以在运行时获得的，但是在定义变量时也可以进行初始化，而且应该提倡进行初始化；
  - 未经初始化的变量，其值可能是随机的。如果误用了未经初始化也没有给予确定值的变量，就会引起错误。

补充2-3：变量的初始化（续）

```
#include <iostream>
using namespace std;
int main()
{
    const double pi(3.14159);
    int radius(0);
    cout<<"The initial radius is:"<<radius<<'\n';
    cout<<"PI is:"<<pi<<'\n  '; 。
    cout<<"Please enter a different radius!\n";
    cin>>radius;
    cout<<"Now the radius is changed to:"<<radius<<'\n';
    cout<<  “PI is still:”  <<pi<<  ‘\n’  ;
    return 0;
}     //观察：通过调试工具跟踪观察变量。
```

补充2-4：整数变量的定义与输出

- 主要知识点：有符号整数与无符号整数的差别
  - 无符号整数unsigned short取值范围为0～65535；
  - 有符号整数short的取值范围为-32768～32767；
  - 如果超出数值类型取值范围，则会出现数值溢出。

补充2-4：整数变量的定义与输出（续）

〈不讲，自学〉

```
#include <iostream>
using namespace std;
int main()
{
   short int i;
   unsigned short int j;
   j = 50000;
   i = j;
   cout <<   “Short int is:” <<i<<endl;
   cout <<   “Short unsigned int is: ” <<j<<endl;
   return 0;
}
//观察思考：将正数50000赋值给变量i以后，输出i的结果是什么？为什么？输出j的结果是什么？为什么？
```

补充2-5：不同类型整数的最值

- 主要知识点：整数类型变量
  - 标准C++中有6种整数类型，它们是short、int、long、unsigned short、unsigned int 和unsigned long。
  - C++头文件limits中定义了一系列符号常量来表示这些最值。

补充2-5：不同类型整数的最值（续）

〈不讲，自学〉

```
#include <iostream>
#include <iostream>
#include <climits>
using namespace std;
int main()
{  cout <<" Min of short is: " <<SHRT_MIN<<endl;
   cout <<" Max of short is: " <<SHRT_MAX<<endl;
   cout <<" Min of int is: " <<INT_MIN<<endl;
   cout <<" Max of int is: " <<INT_MAX<<endl;
   cout <<" Min of long is: " <<LONG_MIN<<endl;
   cout <<" Max of long is: " <<LONG_MAX<<endl;
   cout <<" Max of unsigned short is: " <<USHRT_MAX<<endl;
   cout <<" Max of unsigned int is: " <<UINT_MAX<<endl;
   cout <<" Max of unsigned long is: " <<ULONG_MAX<<endl;
   return 0;
}
//观察：输出的值
```

补充2-6：整型文字常量的应用

- 标准C++中整数文字常量有下列形式。
  - 3种进位计数制——八进制、十进制、十六进制
  - 无符号整数文字常量（后缀u或U）
  - 长整数文字常量（后缀l或L）
- 各种进制整数的输入与输出
  - 程序中在输入与输出数据时可以使用操纵符来指出进制。dec是十进制操作符，oct是八进制操作符，hex是十六进制操作符。

补充2-6：整型文字常量的应用（续）

〈不讲，自学〉

```
#include <iostream>
#include <limits>
using namespace std;
int main()
{
   int isample, osample, hsample;
   unsigned long ulsample;
   cin>>isample>>oct>>osample>>hex>>hsample;
   cout<<isample<<';'<<oct<<osample<<';   '
         <<hex<<hsample<<endl;
   isample=123;
   osample=0173;
   hsample=0x7B;
  ulsample=4294967295UL;
```

补充2-6（续）

〈不讲，自学〉

```
cout<<dec<<isample<<';'<<oct<< isample
     <<';'<<hex<< isample <<endl;
  cout<<dec<< osample<<';' <<oct<<osample
     << ';' <<hex<< osample <<endl;
  cout<< dec<<hsample << ';' <<oct<< hsample<< ';'
     <<hex<<hsample<<endl;
  cout<<dec<< ulsample <<';'<<oct<<ulsample<<';'
     <<hex<<ulsample<<endl;
  return 0;
}
//观察：123、0173、0x7B是同一个数的不同进制表示形式，输入和输出时都可以采用不同的进制。
```

补充2-7：不同类型浮点数的应用

- 主要知识点：浮点类型文字常量
  - 默认情况下浮点文字常量的类型是double，
  - float的浮点文字常量，需要后缀f或者F
  - long double的浮点文字常量，后缀l或者L
  - C++标准没有规定每一种浮点类型的字节数，如果需要知道字节数，可以用sizeof运算得到。

补充2-7：不同类型浮点数的应用（续）

〈不讲，自学〉

```
#include <iostream>
#include <limits>
using namespace std;
const float PI_FLOAT = 3.1415926f;
const double PI_DOUBLE = 3.1415926;
const long double PI_LDOUBLE = 3.1415926;
int main()
{
    float nRadiusFloat = 5.5f, nAreaFloat;
    double nRadiusDouble = 5.5, nAreaDouble;
    long double nRadiusLDouble = 5.5, nAreaLDouble;
    nAreaFloat = PI_FLOAT* nRadiusFloat* nRadiusFloat;
     nAreaDouble = PI_DOUBLE* nRadiusDouble*
                   nRadiusDouble;
```

补充2-7（续）

〈不讲，自学〉

```
nAreaLDouble = PI_DOUBLE* nRadiusDouble*
                              nRadiusDouble;
 cout <<  “nAreaFloat = ” << nAreaFloat <<” ,
          sizeof(nAreaFloat) = ” << sizeof(nAreaFloat)<< endl;
 cout <<  “nAreaDouble = ” << nAreaDouble
         <<” , sizeof(nAreaDouble) = ”
         << sizeof(nAreaDouble)<<endl;
 cout <<  “nAreaLDouble = ” << nAreaLDouble
         <<” , sizeof(nAreaLDouble) = ”
         << sizeof(nAreaLDouble)<<endl;
return 0;
}
//观察运行结果
```

补充2-8：字符数据的应用

- 主要知识点：字符常量、字符串常量、转义序列
  - 字符常量的一般形式是单引号括起来的一个字符；
  - 一些不可显示的字符，无法通过键盘输入，可以用转义序列来表示。

补充2-8：字符数据的应用（续）

〈不讲，自学〉

```
#include <iostream>
using namespace std;
int main()
{
   cout << 'A'<< ' '<<'a'<<endl;     //输出普通字符
   cout << "one\ttwo\tthree\n";     //使用水平制表符
   cout << "123\b\b45\n";           //使用退格符
   cout << "Alert\a\n";             //使用响铃键
   return 0;
}
//观察运行结果
```

补充2-9：bool类型的变量

- 主要知识点：bool类型的应用
  - 在算术运算表达式里，当表达式需要整数时，bool值将被转化为int，true为1，false 为0。
  - 如果需要将整数转换为bool值，那么0转换为false，所有非0值都为true。

补充2-9：bool类型的变量（续）

〈不讲，自学〉

```
#include <iostream>
using namespace std;
int main()
{
    bool bV1= true, bV2= false;
    cout <<" bool value bV1 = " <<bV1<<endl;
    cout <<" bool value bV2 = " <<bV2<<endl;
    int nV1=bV1, nV2 = 0;
    bV1 = nV2;
    cout <<" int value nV1 = " << nV1<<endl;
    cout <<" bool value bV1 = " <<bV1<<endl;
    return 0;
}
//观察程序运行过程中内存中bool变量的值
```

算术运算符与赋值运算

2.2.5（1）

〈只出文字〉

算术运算符与算术表达式

- 基本算术运算符

  + - * /(若整数相除，结果取整)

  %（取余，操作数为整数）

- 优先级与结合性

  先乘除，后加减，同级自左至右

- ++, --（自增、自减）

  例：i++; --j;

〈只出文字〉

赋值运算符和赋值表达式
——简单的赋值运算符"="

- 举例

  n = n + 5
- 表达式的类型

  赋值运算符左边对象的类型
- 表达式的值

  赋值运算符左边对象被赋值后的值

〈只出文字〉

赋值运算符和赋值表达式
——复合的赋值运算符

- 有10种复合运算符：

  +=, -=, *=, /=, %=,

  <<=, >>=, &=, ^=, |=

- 例

  a += 3 等价于 a = a + 3

  x *= y + 8 等价于 x = x * (y + 8)

逗号运算、关系运算和逻辑运算、条件运算

<只出人像>2.2.5（5）

〈只出文字〉

逗号运算和逗号表达式

- 格式

  表达式1，表达式2
- 求解顺序及结果
  - 先求解表达式1，再求解表达式2
  - 最终结果为表达式2的值
- 例

  a = 3 * 5，a * 4　最终结果为60

〈只出文字〉

关系运算与关系表达式

- 关系运算是比较简单的一种逻辑运算，优先次序为：

  < <= > >= == !=

  优先级相同（高） 优先级相同（低）

- 关系表达式是一种最简单的逻辑表达式

  其结果类型为 bool，值只能为 true 或 false。

- 例如：a > b，c <= a + b，x + y == 3

〈只出文字〉

逻辑运算与逻辑表达式

- 逻辑运算符

  !(非)　　&&(与)　　||(或)

  优先次序：　高　　→　　低

- 逻辑表达式

  例如：(a > b) && (x > y)

  其结果类型为 bool，值只能为 true 或 false

〈只出文字〉

逻辑运算与逻辑表达式(续)

- “&&” 的“短路特性”

  表达式1 && 表达式2
  - 先求解表达式1
  - 若表达式1的值为false，则最终结果为false，**不再求解表达式2**
  - 若表达式1的结果为true，则求解表达式2，以表达式2的结果作为最终结果
- “||”也具有类似的特性

〈只出文字〉

条件运算符与条件表达式

- 一般形式

  表达式1？表达式2：表达式3

  表达式1 必须是 bool 类型

- 执行顺序
  - 先求解表达式1，
  - 若表达式1的值为true，则求解表达式2，表达式2的值为最终结果
  - 若表达式1的值为false，则求解表达式3，表达式3的值为最终结果
- 例： x = a > b ? a : b;

〈只出文字〉

条件运算符与条件表达式（续）

- 注意：
  - 条件运算符优先级高于赋值运算符，低于逻辑运算符
  - 表达式2、3的类型可以不同，条件表达式的最终类型为 2 和 3 中较高的类型。
- 例： x = a > b ? a : b;

①

②

Sizeof运算、位运算

<只出人像>2.2.5（4）

〈只出文字〉

sizeof 运算符

- 语法形式
  sizeof (类型名)
  或 sizeof 表达式
- 结果值：
  “类型名”所指定的类型，或“表达式”的结果类型所占的字节数。
- 例：

```
sizeof(short)
sizeof  x
```

〈只出文字〉

位运算——按位与（&）

- 运算规则
  - 将两个运算量的每一个位进行逻辑与操作
- 举例：计算 3 & 5

```
  3：   00000011
  5：   00000101
3&5:    00000001
```

- 用途：
  - 将某一位置0，其他位不变。例如：
    将char型变量a的最低位置0: a = a & 0xfe;
  - 取指定位。
    例如：有char c; int a;
    取出a的低字节，置于c中：c=a & 0xff;

〈只出文字〉

位运算——按位或（|）

- 运算规则
  - 将两个运算量的每一个位进行逻辑或操作
- 举例：计算 3 | 5

  3：　0 0 0 0 0 0 1 1

  5：　0 0 0 0 0 1 0 1

  3|5:　0 0 0 0 0 1 1 1
- 用途：
  - 将某些位置1，其他位不变。
    例如：将 int 型变量 a 的低字节置 1 ：

    a = a | 0xff;

〈只出文字〉

位运算——按位异或（^）

- 运算规则
  - 两个操作数进行异或：
    若对应位相同，则结果该位为 0，
    若对应位不同，则结果该位为 1，
- 举例：计算 071^052

```
    071:     0 0 1 1 1 0 0 1
    052:   0 0 1 0 1 0 1 0
071^052:   0 0 0 1 0 0 1 1
```

〈只出文字〉

位运算——按位异或（^）（续）

- 用途：
  - 使特定位翻转（与0异或保持原值，与1异或取反）

例如：要使 01111010 低四位翻转：

```
  01111010
^ 00001111
  01110101
```

〈只出文字〉

位运算——取反（ ~ ）

单目运算符，对一个二进制数按位取反。

例： 025：0000000000010101

~025：1111111111101010

〈只出文字〉

位运算——移位

- 左移运算（<<）
  左移后，低位补0，高位舍弃。
- 右移运算（>>）
  右移后，
  低位：舍弃
  高位：无符号数：补0
  有符号数：补“符号位”

运算优先级、类型转换

2.2.5（5）

〈只出文字〉

运算符优先级

优先级	运算符	结合性
1	[] () . -> 后置 ++ 后置 --	左→右
2	前置 ++ 前置 -- sizeof & * +（正号） -（负号） ~ ！	右→左
3	（强制转换类型）	右→左
4	.* ->*	左→右
5	* / %	左→右
6	+ -	左→右
7	<< >>	左→右
8	< > <= >=	左→右
9	== !=	左→右
10	&	左→右
11	^	左→右
12	|	左→右
13	&&	左→右
14	||	左→右
15	? :	右→左
16	= *= /= %= += -= <<= >>=&= ^= |=	右→左
17	,	左→右

〈只出文字〉

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 (unsigned) short (unsigned) int (unisgned) long (unsigned) long long float double

低 ——————————————————————————→ 高

〈只出文字〉

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。

〈只出文字〉

混合运算时数据类型的转换

- 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。

〈只出文字〉

混合运算时数据类型的转换——显式转换

- 语法形式（3种）：
  - 类型说明符(表达式)
  - (类型说明符)表达式
  - 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

2.3 数据的输入和输出

<只出人像>

〈只出文字〉

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。

〈只出文字〉

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上便可以实现项标准输出设备输出。
  - cout << 表达式 << 表达式...
- 标准输入是将提取符作用在流类对象cin上。
  - cin >> 表达式 >> 表达式...
- 提取符可以连续写多个，每个后面跟一个表达式，该表达式通常是用于存放输入值的变量。例如：
  - int a, b;
  - cin >> a >> b;

〈只出文字〉

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
setsprecision(int)	设置浮点数的小数位数（包括小数点）
setw(int)	设置域宽

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

if语句

<只出人像>2.4.1-2.4.2

2.4.1 用if语句实现选择结构

```
//2_2.cpp       例2-2输入一个年份，判断是否闰年
#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;
} //调试观察选择执行情况
```

〈只出文字〉

If语句的语法形式

if (表达式) 语句

例：if (x > y) cout << x;

if (表达式) 语句1 else 语句2

例：if (x > y) cout << x;

else cout << y;

if (表达式1) 语句1

else if (表达式2) 语句2

else if (表达式3) 语句3

…

else 语句 n

2.4.2 多重选择结构——嵌套的if结构

例2-3：输入两个整数，比较两个数的大小。

```
#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;
}
```

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

〈只出文字〉

嵌套的if结构（续）

- 语法形式

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

- 注意

  语句 1、2、3、4 可以是复合语句，每层的 if 与 else 配对，或用 { } 来确定层次关系。

switch语句

〈只出人像〉

例2-4：输入一个0～6的整数，转换成星期输出

```
#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;
}
```

〈只出文字〉

switch语句的语法

- 一般形式

```
switch (表达式)
  { case 常量表达式 1：语句1
    case 常量表达式 2：语句2
                ┆
    case 常量表达式 n：语句n
    default :        语句n+1
  }
```

- case分支可包含多个语句，且不用{ }。
- 表达式、判断值都是int型或char型。
- 若干分支执行内容相同可共用一组语句。

- 执行顺序

以case中的常量表达式值为入口标号，由此开始顺序执行。因此，每个case分支最后应该加break语句。

循环结构——while语句

〈只出人像〉2.4.3（1）

例2-5（续）求自然数1～10之和

```
#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;
}
```

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。

运行结果：
sum = 55

〈只出文字〉

while语句的语法

- 形式

  while（表达式）语句

可以是**复合语句**，其中必须含有改变条件表达式值的语句。

- 执行顺序

  先判断表达式的值，若为 true 时，执行语句。

do-while 语句

<只出人像>2.4.3（2）

例2-6：输入一个数，将各位数字翻转后输出

```
#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;
}
```

运行结果：

```
Enter the number: 365
The number in reverse order is  563
```

〈只出文字〉

do-while 语句的语法形式

- 一般形式

  do　语句 ← 可以是复合语句，其中必须含有改变条件表达式值的语句。

  while (表达式)

- 执行顺序

  先执行循环体语句，后判断条件。
  表达式为 true 时，继续执行循环体

- 与while语句的比较：

  while 语句执行顺序
  先判断表达式的值，为true时，再执行语句

例2-7用do-while语句编程，求自然数1~10之和

```
//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;
}
```

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;
}
```

for语句

〈只出人像〉2.4.3（3）

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
```

〈只出文字〉

for语句（续）

语法形式

```
for（初始语句；表达式1；表达式2）语句
```

循环前先求解　为true时执行循环体　每次执行完循环体后求解

for语句还有另一种更加简洁的写法，称为范围for语句，语法形式为：

```
for（声明：表达式）
    语句
```

这种形式的for语句主要用于遍历一个容器中的序列，将在第6、10章详细介绍

嵌套的循环、其他控制语句

<只出人像>2.4.4

〈只出文字〉

2.4.4 循环结构与选择结构的嵌套

例2-10

- 输入一系列整数，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!=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;
}
```

〈只出文字〉

2.4.5 其他控制语句

- break语句
  使程序从循环体和switch语句内跳出，继续执行逻辑上的下一条语句。不宜用在别处。
- continue 语句
  结束本次循环，接着判断是否执行下一次循环。
- goto 语句
  goto语句的作用是使程序的执行流程跳转到语句标号所指定的语句。

自定义类型

<只出人像>2.5

〈只出文字〉

2.5.1 类型别名

- 可以为一个已有的数据类型另外命名
  - typedef 已有类型名 新类型名表
    - typedef double Area, Volume;
    - typedef int Natural;
    - Natural i1,i2;
    - Area a;
    - Volume v;
  - using 新类型名 = 已有类型名;
    - using Area = double;
    - using Volume = double;

〈只出文字〉

2.5.2 枚举类型

- 只要将需要的变量值一一列举出来，便构成了一个枚举类型。
- C++包含两种枚举类型：不限定作用域的枚举类型和限定作用域的枚举类。
- 不限定作用域枚举类型声明形式如下：

  enum 枚举类型名 {变量值列表};
  - 例如：

    enum Weekday
    {SUN, MON, TUE, WED, THU, FRI, SAT};
- 关于限定作用域的enum类将在第4章和第5章详细介绍。

〈只出文字〉

不限定作用域枚举类型说明

- 对枚举元素按常量处理，不能对它们赋值。例如，不能写：SUN = 0;
- 枚举元素具有默认值，它们依次为： 0,1,2,……。
- 也可以在声明时另行指定枚举元素的值，如：

```
enum Weekday{SUN=7,MON=1,TUE,WED, THU,FRI,SAT};
```

- 枚举值可以进行关系运算。
- 整数值不能直接赋给枚举变量，如需要将整数赋值给枚举变量，应进行强制类型转换。

〈只出文字〉

例2-11

- 设某次体育比赛的结果有四种可能：胜（WIN）、负（LOSE）、平局（TIE）、比赛取消（CANCEL），编写程序顺序输出这四种情况。
  - 分析：由于比赛结果只有四种可能，所以可以声明一个枚举类型，声明一个枚举类型的变量来存放比赛结果。

例2-11（续）

```
#include <iostream>
using namespace std;
enum GameResult {WIN, LOSE, TIE, CANCEL};
int main() {
    GameResult result;
    enum GameResult omit = CANCEL;
    for (int count = WIN; count <= CANCEL; count++) {
      result = GameResult(count);
      if (result == omit)
        cout << "The game was cancelled" << endl;
      else {
        cout << "The game was played ";
        if (result == WIN)      cout << "and we won!";
        if (result == LOSE)       cout << "and we lost.";
        cout << endl;
      }
    }
    return 0;
}
```

运行结果

```
The game was played and we won!
The game was played and we lost.
The game was played
The game was cancelled
```

2.5.3 auto类型与decltype类型

- auto让编译器通过初始值自动推断变量的类型。
  - 例如：

    auto val = val1 + val2;

    val的类型取决于表达式val1+val2的类型，如果val1+val2是int类型，那么val将是int类型；如果val1+val2是double类型，那么val将是double类型。
- 定义一个变量与某一表达式的类型相同，但并不想用该表达式初始化这个变量，这时我们需要decltype变量
  - 例如：

    decltype(i) j = 2;

    表示 j 以 2 作为初始值，类型与 i 一致。

第2章 小结

- 主要内容
  - C++语言概述、基本数据类型和表达式、数据的输入与输出、算法的基本控制结构、自定义数据类型
- 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。