第2章 数据类型与表达式

Chapter 2 Data Types & Expression

C++初学者必看的50个建议

- 1.把C++当成一门新的语言学习（和C没啥关系！真的。）；
- 2.看《Thinking In C++》，不要看《C++变成死相》；
- 3.看《The C++ Programming Language》和《Inside The C++ Object Model 》，不要因为他们很难而我们自己是初学者所以就不看；

C++初学者必看的50个建议

- 4.不要被VC、BCB、BC、MC、TC等词汇所迷惑——他们都是集成开发环境，而我们要学的是一门语言；

- 5.不要放过任何一个看上去很简单的小编程问题——他们往往并不那么简单，或者可以引伸出很多知识点；

- 6.会用Visual C++，并不说明你会C++；

2.1 C++的数据类型
C++ Data Types

- 数据（*data*）是对客观事物的符号表示，是信息的载体，是计算机处理的对象。在计算机科学中是指所有能输入到计算机中并由程序识别、存储和加工的信息的总称。
  数据元素（*Data Element*）是数据的基本单位，在计算机程序中通常作为一个整体考虑和处理。数据元素可由若干个数据项组成，数据项是数据的不可分割的最小单位。
- 数据对象（*Data Object*）是性质相同的数据元素的集合，是数据的一个子集。数据对象是一种运行时的概念，可以是外部实体、事物、行为、事件、角色、单位、地点或结构等。总之，可以由一组属性来定义的实体都可以被认为是数据对象

2.1 C++的数据类型

2.1.1 C++的数据对象

C++数据对象分为两大类：程序员定义的数据对象和系统定义的数据对象。

- 程序员定义的数据对象：由程序员通过说明语句*显式*创建和控制的常量、简单变量、数组、文件等。其中常量和简单变量称为基本数据对象。

- 系统定义的数据对象：指由虚拟机建立起来，用于运行事务管理的数据对象，例如运行栈（运行期堆栈）、子程序活动记录、文件缓冲区以及内存空闲区表等。

2.1.2 数据对象的重要属性

类型、名称、位置和值是任何一个数据对象所具有的重要属性。

类型

类型是数据对象的基本属性，例如整型、字符型等。

注意：一个确定的数据对象不可以没有类型！

名称

名称，是数据对象的外部标记，便于实现“按名存取”

注意：任何一个变量不可以没有名字。

2.1.2 数据对象的重要属性

位置

位置是一种绑定，是指数据对象所分配到的内存地址。这种绑定可以由虚拟机的存储管理例程改变，因此属于动态绑定。

值

值是一种绑定，该绑定通常以赋值操作实现，当然其它操作，例如数据对象的输入操作也可以实现值的绑定。

注意：一个数据对象在其生存期中，属性一般不会改变，但绑定是可以动态改变的。

2.1.3 数据类型概述

数据结构（data structure）指数据的组织形式，是一系列性质相同的数据组织成一定的逻辑结构（包括在计算机中的存储结构）并带有自身的一系列操作。

数据类型（data type）是一组性质相同的具有一定范围的值集以及定义于这个值集上的一组操作。数据类型的本质是数据组织和其操作的捆绑性。

数据类型是数据结构在一定编程语言中的描述形式，是在程序设计语言中已经实现了的数据结构。

数据结构可以看做是一种抽象的数据类型。

处理同一类问题，若数据结构不同，算法也会不同。

数据类型概述（2）

1. 数据类型的涵义

数据类型是一个由数据对象以及创建和操纵它们的操作的集合所组成的类。 可以从含义上去理解它，数据类型有以下三个含义：

- 确定数据的值域（数据取什么值）；
- 规定允许施加的运算（操作）；
- 规定数据的存储结构和存储方式。

数据类型概述（3）

2. 数据类型的的重要性

- 每一个语言都有一个原始（标准）数据类型集，此外也应提供定义新数据类型的机制。

- 一个语言所提供的标准数据类型集是否完善，定义新数据类型的机制是否健全，直接反映出该语言的处理能力。

数据类型概述（4）

3．C++的数据类型一览

C++提供的数据类型分成基本类型和非基本类型（构造类型）两大类。

- 基本类型，数据的取值为纯量。例如整型数据在任何时刻其值都为单个整数。

- 非基本类型，包括结构类型、指针类型、空类型（void）、类（Class）等。如下图所示：

下图给出C++的数据类型一览：

- 数据类型
  - 基本类型
    - 整型
      - 短整型（short int） 2字节
      - 整型（int） 4字节
      - 长整型（long int） 4字节
    - 字符型（char） 1字节
    - 浮点型
      - 单精度型（float） 4字节
      - 双精度型（double） 8字节
      - 长双精度（long double） 8字节
    - 布尔型（bool） — 逻辑型 1字节
  - 非基本类型
    - 枚举类型（enum）
    - 数组类型（type[]）
    - 结构体类型（struct）
    - 共用体类型（union）
    - 类类型（class）
    - 指针类型（type *）
    - 引用类型（type &）
    - 空类型（void） — 无值型

数据类型	基本类型	整型	短整型（short int）	2字节
			整型（int）	4字节
			长整型（long int）	4字节
		字符型（char）		1字节
		浮点型	单精度型（float）	4字节
			双精度型（double）	8字节
			长双精度型（long double）	8字节
		布尔型（bool） — 逻辑型		1字节
	非基本类型	枚举类型（enum）		
		数组类型（type[]）		
		结构体类型（struct）		
		共用体类型（union）		
		类类型（class）		
		指针类型（type *）		
		引用类型（type &）		
		空类型（void） — 无值型		

关于void类型

- void，空类型，又称无值型或空值型，字节长度为0, 用于声明空值型数据，与其他类型相比，不用返回数据的类型。
- void只能用来声明非数据变量的空参数，和不用返回数据的函数。主要有两个用途:
- 一是明确地表示一个函数不返回任何值;如：

```
void display( [void] )
```

- 一是产生一个同一类型指针(可根据需要动态分配给其内存)。例如:

```
void *buffer;
/* buffer被定义为无值型指针，为NULL */
```

- C#中可空类型功能允许将 null 赋给值类型

关于数据类型的几点说明

- 1. 任一数据都必须属于一种数据类型，必须“先定义，后使用”。
- 2. 数据类型的作用：
  - 确定数据分配空间的大小和所能进行的操作。不同类型的数据在内存中占据不同长度的存储区， 对应确定的值的范围和允许的操作。
- 3. C++的数据分为常量和变量，
- 4. 可构成更复杂的数据结构如表、树、栈等。
- 5. 各类数据的精度、数值范围和在内存中所占的字节数，由C++编译系统决定。

2.2 常量

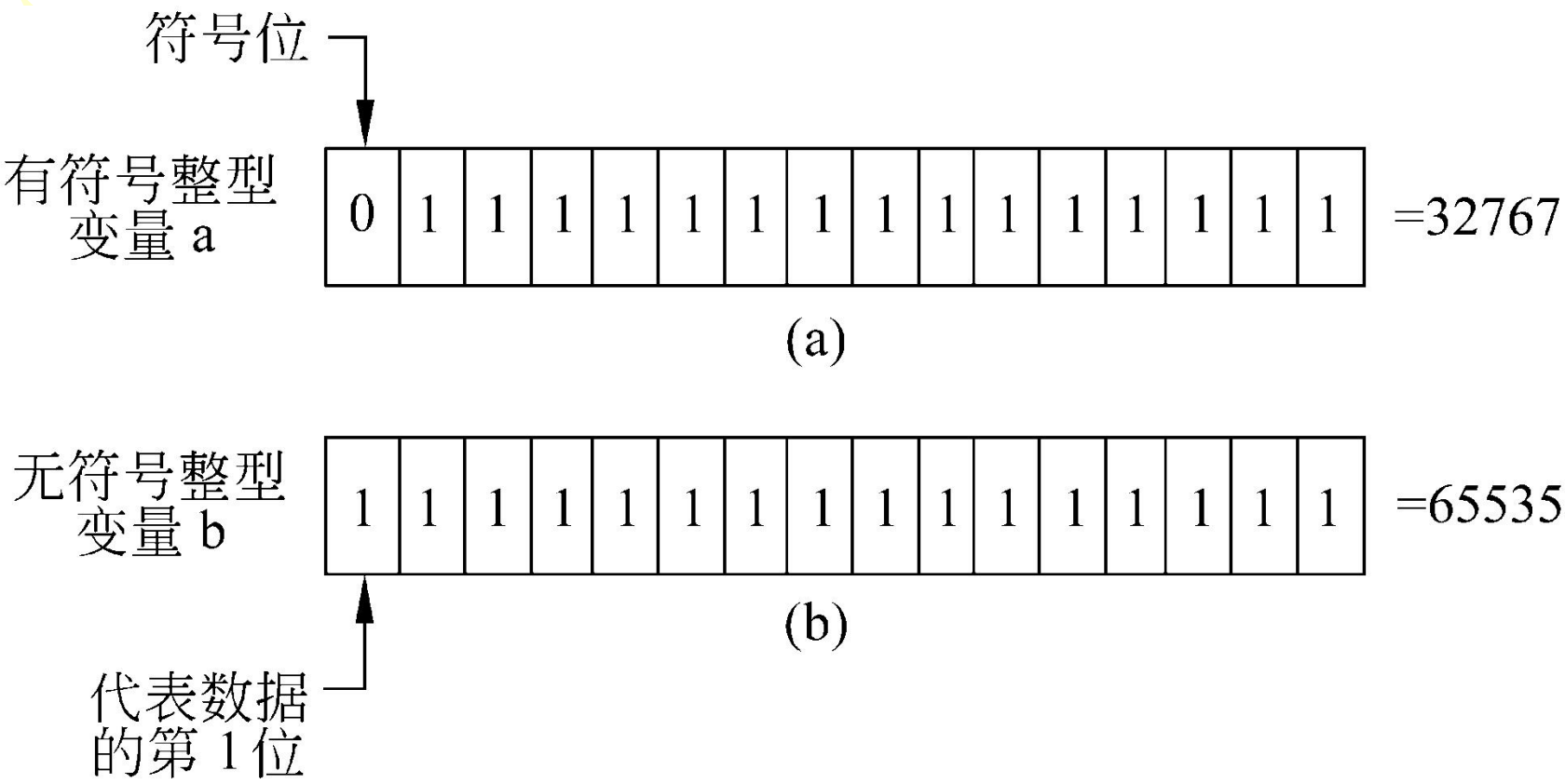
- 常量是指取值在程序的执行过程中始终保持不变的量，又分为文字常量（Literal constant）和常变量（也称“符号常量”）。
- 文字常量包括两大类：
  - 数值型常量（整型、实型）
  - 字符型常量（单字符、字符串）

整型数据的存储方式

- 二进制位（bit，简写为b）是数据的最小单位。
- 8个二进制位称为一个字节（Byte，简写为B）。
- 若干个字节组成字（Word）。一个字由所包含的位数称为字长。常用的字长有8位、16位、32位和64位等。
- 整型数据的存储方式按二进制数形式存储。数的存储长度与数的实际长度（二进制位数，没有限制）无关。
- 数的符号用最高位（左边第一位）来表示，并约定：0代表正数，1代表负数。

整型数据的存储方式

- 按二进制数形式存储



浮点数据的表示方法

浮点数可写成小数形式或指数形式。由于指数部分的存在，使得同一个浮点数可以用不同的指数形式来表示，数字部分中小数点的位置是浮动的。

浮点数的规格化表示：

数字部分必须小于1；

小数点后面第一个数字必须是一个非0数字；

参见：C++程序设计教程(第二版)，钱能，清华大学出版社, 3.3

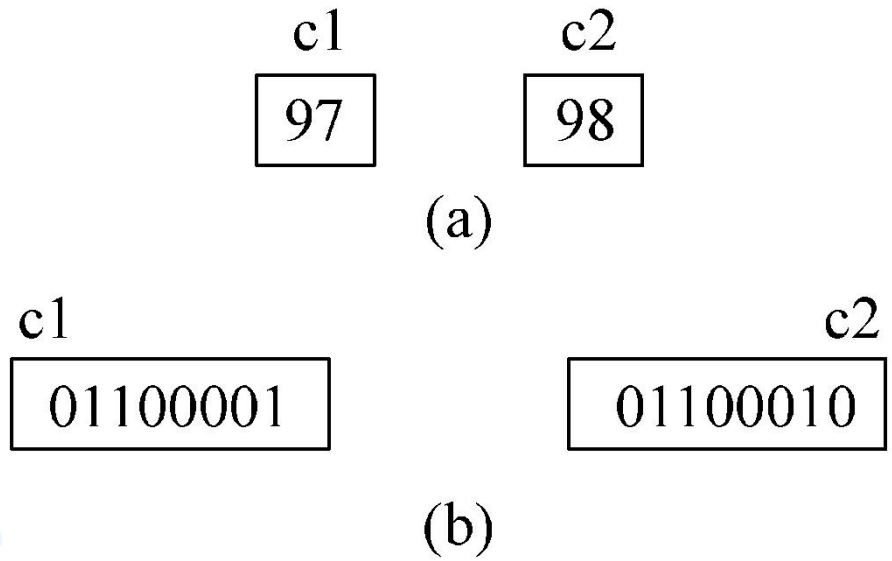
浮点数据的表示方法

- 浮点数一般记作：
- $N = M \times R^P$
- M即数字部分又称为尾数(mantissa)，是一个纯小数；R称为基数（radix）；P称为指数或阶码(exponent) 。
- 存储空间被划分为两部分，分别存放尾数和阶码。

+	3	.314159
数符	指数	尾数
float	8	23
double	11	52

字符数据的存储形式

与整数的存储形式类似。将字符数据（1Byte）的ASCII代码以二进制形式存放到存储单元中。



对字符数据进行算术运算相当于对它们的ASCII码进行算术运算。

例2.1 将字符赋给整型变量。

```
#include <iostream>
using namespace std;
int main（ ）
{
    int  i,j;                  //i和j是整型变量
    i='A';                     //将一个字符常量赋给整型变量i
    j='B';                     //将一个字符常量赋给整型变量j
    cout<<i<<' '<<j<<'\n';    //输出整型变量i和j的值，'\n' 是换行符
    return 0;
}
```

例2.2 字符数据与整数进行算术运算。

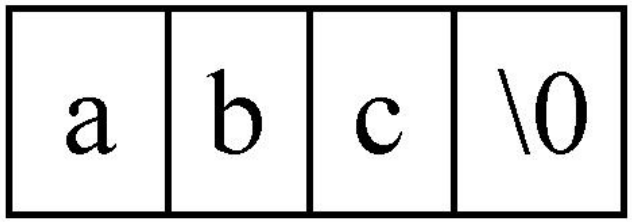
```
#include <iostream>
using namespace std;
int main（ ）
{
    char c1,c2;
    c1='a';
    c2='b';
    c1=c1-32;
    c2=c2-32;
    cout<<c1<<' '<<c2<<endl;
    return 0;
}
```

字符串常量的存储形式

编译系统会在字符串最后自动加一个'\0'作为字符串结束标志。但'\0'并不是字符串的一部分，它只作为字符串的结束标志。如

```
cout<<"abc"<<endl;
```

输出3个字符abc，而不包括'\0'。



注意："a"和'a'代表不同的含义，"a"是字符串常量，'a' 是字符常量。前者占两个字节，后者占1个字节。请分析下面的程序片段：

```
char c;              //定义一个字符变量
c='a';
c="a";
//错误，c只能容纳一个字符
```

思考：字符串常量"abc\\n"包含几个字符?

转义字符

- “\”必须与后面的字符组成一个合法的转义字符。
- 如果希望将“\”字符作为一个字符，则应写为“\\”。

- cout<<"abc\\\n"<<endl;
- 输出是： abc\，然后换行。

- cout<<"I say \"Thank you！ \"\n";
- 输出是： I say "Thank you！ "

续行符

•如果在一个字符串中最后一个字符为“\”，则表示它是续行符，下一行的字符是该字符串的一部分，且在两行字符串间无空格。如

```
cout<<“We must study C \
++ hard!“;
```

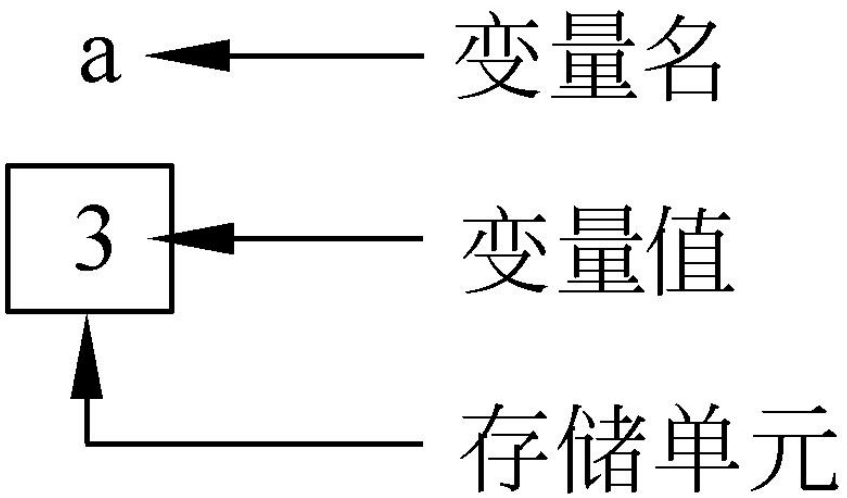
输出是: We must study C++ hard!

2.3 变量

2.3.1 什么是变量

在程序运行期间其值可以改变的量称为变量。

一个变量有一个名字，并在内存中占据一定的存储单元，在该存储单元中存放变量的值。



2.3.2 标识符命名规则

标识符：用来标识变量、符号常量、函数、数组、类等实体名字的有效字符序列称为标识符（*identifier*）。

标识符的命名规则是：

- 1） 只能由英文字母、数字和下划线三种字符组成。
- 2） 必须以英文字母或下划线开头。
- 3） 标识符长度由编译系统规定。一般取32个字符，超过的字符不被识别。

标识符命名注意事项

- 1） C++区分字母的大小写。
- 2） 尽量不要以下划线开头，避免可能和系统定义名字冲突。
- 3） 标识符尽可能选取有意义的词汇，以便“见名知意”。
- 4） 一般符号常量用大写，变量名和函数名用小写。
- 5） 避免用易混淆的字符，如0Oo,1Il,2Zz。

编程风格之命名规则

- 1. 变量名的命名规则
- （1）匈牙利命名法[Hungarian]：
- ①变量：开头字母用变量的类型，其余部分用变量的英文意思或其英文意思的缩写,要求单词的第一个字母应大写。
- 即： 变量名=变量类型+变量的英文意思（或缩写）
- bool 用b开头 bIsParent
- byte 用by开头 byFlag
- ②指针变量：命名的基本原则为：
- “p”+变量类型前缀+命名
- ③struct、union、class变量：要求定义的类型用大写，并要加上前缀。
- ④const 变量：要求在变量的命名规则前加入c_,即：c_+变量命名规则；

（2）骆驼命名法[camelCase]： 小写字母开头，下一个单词的首字母大写，如

studentName

（3）帕斯卡命名法[PascalCase]：首字母大写，如 **string UserName;**

（4）下划线命名法：单词间用下划线来连接（C风格），如

number_of_student

简单说来就是类和结构体使用帕斯卡命名法，而函数、变量、枚举使用骆驼命名法。

- **2. 函数的命名规范：**
- 函数的命名应该尽量用英文表达出函数完成的功能。遵循动宾结构的命名法则，函数名中动词在前,并在命名前加入函数的前缀，函数名的长度不得少于8个字母。
- 例如：
- **long getDeviceCount(……);**

几个特殊关键字的说明

- 1） 关于空类型（void）
  - void类型一是表示函数无返回值；二是设置空指针。
- 2） 类型修饰符（signed、unsigned、short、long）
- 3） 存取修饰符（const、volatile）
  - 常数型、暂态型
  - 控制对变量访问或修改的方式。const型变量在程序执行期间不可改变，volatile型变量的值可由程序中没有显式说明的方式改变。
- 4） 存储类型符（extern、static、register、auto）
  - 外部、静态、寄存器、自动

2.3.3 定义变量

变量强制定义的目的

（1）先定义，后使用，能保证程序中变量名使用得正确。例如：

int student；//声明部分

statent=30；//执行语句错写成statent

在编译时检查出statent未经定义，作为错误处理。便于用户发现错误，避免变量名使用时出错。

（2） 每一个变量被指定为一确定类型，在编译时就能为其分配相应的存储单元。

如指定 a 和 b 为int型，一般的编译系统对其各分配4个字节，并按整数方式存储数据。

（3） 指定每一变量属于一个特定的类型，这就便于在编译时，据此检查该变量所进行的运算是否合法。

例如，整型变量a和b，可以进行求余运算：

a%b

%是“求余”，得到a/b的余数。如果将a和b指定为实型变量，则不允许进行“求余”运算，在编译时会给出有关的出错信息。

2.3.4 为变量赋初值

变量初始化：在定义变量时对它赋予一个初值。

初值可以是常量或一个有确定值的表达式。如：

```
float a, b=5.78*3.5, c=2*sin(2.0);
cout<<a<<" "<<b<<" "<<c<<endl;
```

如果对变量未赋初值，则该变量的初值是一个不可预测的值，即该存储单元中当时的内容是不知道的。

2.3.5 常变量（*constant variable*）

用常量说明符const给文字常量命名所得的标识符就称为“标识符常量”。

因为标识符常量的说明和引用形式很像变量，所以也称“常变量”。

常变量：变量的值在程序运行期间不能改变，可理解为只读变量（*read-only-variable*）。例如，

```
const int a=3;
```

在定义常变量时必须同时对它初始化，此后它的值不能再改变。

```
const int a; a=3;
//错误，常变量不能被赋值即常变量不能出现在赋值号的左边。
```

符号常量和常变量

都是以标识符形式出现的常量。

C：#define PRICE 30

称为符号常量，没有类型，在内存中并不存在以符号常量命名的存储单元。

C++：const PRICE=30

称为常变量，具有类型，在内存中存在着以它命名的存储单元，可以用sizeof测量其长度。

"Use const whenever you need."

2.5 算术运算符与算术表达式

+、－、*、/

%（模或求余运算符，%两侧均应为整型数据）

两个整数相除的结果为整数，若结果为负数，则舍入的方向是不固定的，多数编译系统采取“向零取整”的方法。

例如，即5/3的值等于1，-5/3的值等于-1。

Visual C++ 6.0， Dev C++ 5.0系统上得到结果-1，有的C++系统则给出结果-2。

2.5.2 运算符的优先级与结合性

用算术运算符和括号将运算对象或操作数（常量、变量、函数等）连接起来的、符合C++语法规则的式子，称C++算术表达式。例如：

a*b/c-1.5+'a'

在求解表达式时，先按运算对象两侧运算符的优先级别高低顺序执行；

如a-b*c，相当于a-（b*c）。

若运算对象两侧的运算符优先级别相同，则按运算符结合性 (自左至右，左结合性；自右至左，右结合性)规定的方向处理。如

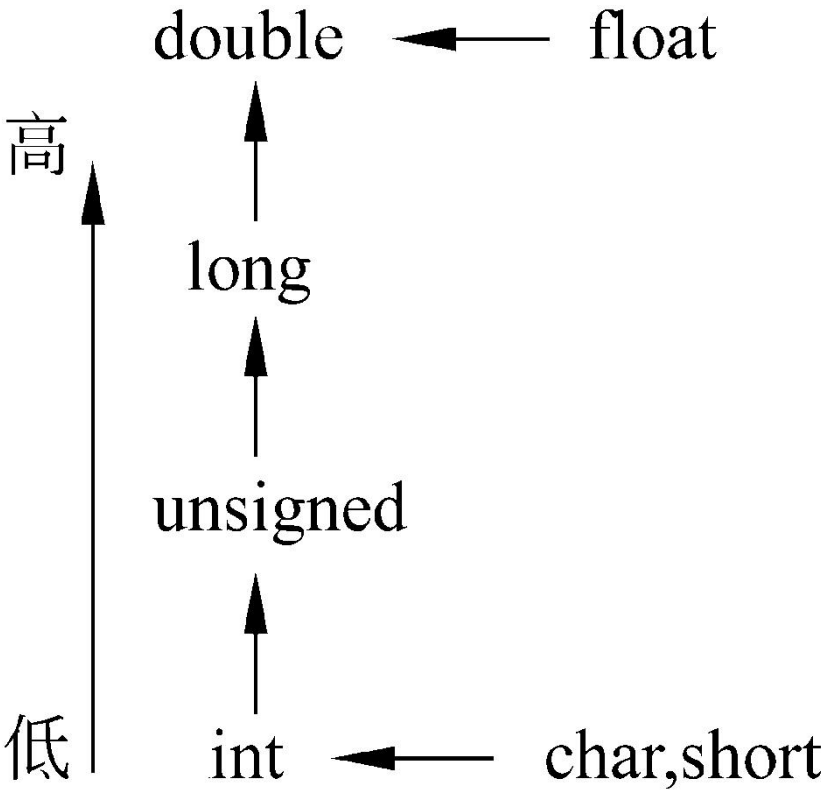
x-y+z；//先执行x-y，再执行+z。

x=y=z；//先执行y=z，再执行x=(y=z) 。

参见：附录B。

2.5.3 各类数值型数据间的混合运算

自动类型转换：在进行运算时，不同类型的数据要先转换成同一类型，然后进行运算。



设 i 为int， f 为float， d 为double， e 为long。

表达式： 10+'a'+i*f-d/e

运算次序为：

①10+'a'，' a '转换成整数97，运算结果为int。

②i*f，i与f都转换成double型，运算结果为double。

③107+i*f，整数107转换成双精度数，结果为double。

④d/e，变量e转换成double型，d/e结果为double。

⑤将10+‘a’+i*f的结果与d/e的商相减，结果为double。

强制类型转换参见2.5.5。

2.6.2 赋值过程中的类型转换

如果赋值运算符两侧的类型不一致，但都是数值型或字符型时，在赋值时会自动进行类型转换。

（1） 将浮点型数据（包括单、双精度）赋给整型变量时，舍弃其小数部分。

（2） 将整型数据赋给浮点型变量时，数值不变，但以指数形式存储到变量中。

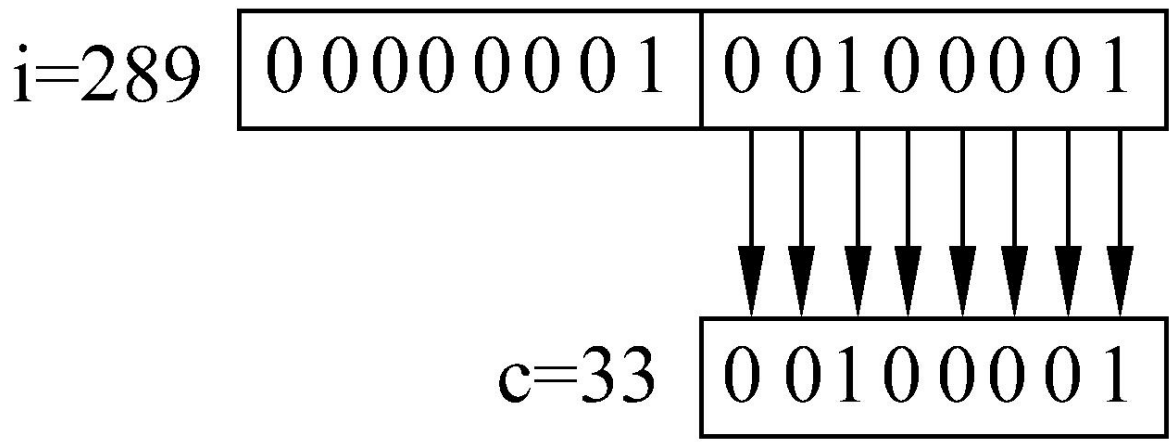
（3） 将一个double型数据赋给float变量时，要注意数值范围不能溢出。为什么？

（4） 字符型数据赋给整型变量，将字符的ASCII码赋给整型变量。

```
#include <iostream>          //预处理命令
using namespace std;
int main( )                  //主函数
{                            //主函数体开始
    float f; double d;       //变量声明
    d=123456789e40;
    f=d;
    cout<<"float="<<f<<'\n';
    cout<<"double="<<d<<'\n';
    return 0;                //程序正常结束，返回零值
}                            //主函数结束
```

（5） 将一个int、short或long型数据赋给一个char型变量，只将其低8位原封不动地送到char型变量（发生截断）。例如

```
short int i=289;
char c;
c=i;            //将一个short int数据赋给一个char变量
```



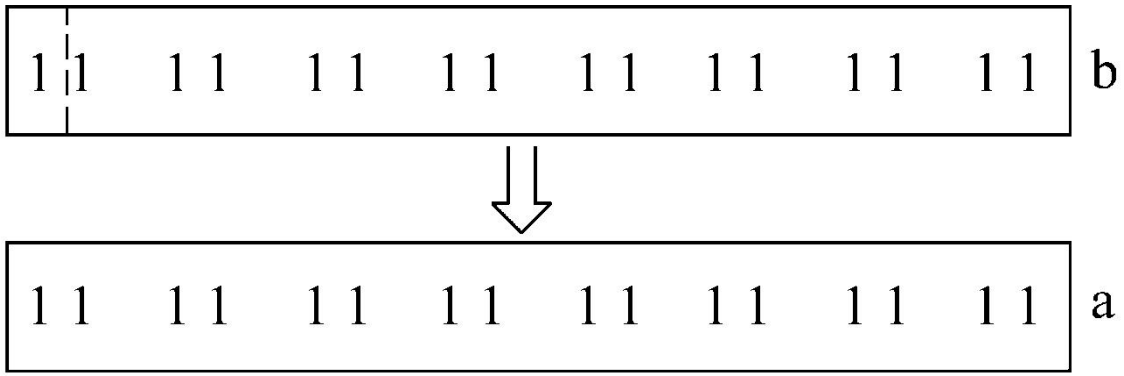
（6） 将signed（有符号）型数据赋给长度相同的unsigned（无符号）型变量，将存储单元内容原样照搬（连原有的符号位也作为数值一起传送）。

例2.5 将有符号数据传送给无符号变量。

```
#include <iostream>
using namespace std;
int main()
{
    unsigned short a;
    short int b=-1;
    a=b;
    cout<<"a="<<a<<endl;
    return 0;
}
```

运行结果为65535。

原因：-1的补码形式为1111111111111111。如果b为正值，且在0~32767之间，则赋值后数值不变。



不同类型的整型数据间的赋值归根结底就是：按存储单元中的存储形式直接传送。

- 下面程序的运行结果是_____。

```
#include < iostream.h >
void main( )
{
    char a='0',b='9';
    float x;
    x=(b-a)/('F'-'B');
    cout<<(int)(3.14*x);
}
```

2.6.3 复合的赋值运算符

在赋值符“＝”之前加上其他运算符，可以构成复合的运算符。例如，

a+=3　　等价于 a=a+3

x*=y+8　　等价于 x=x*（y+8）

x%=3　　等价于 x=x%3

凡是二元（双目）运算符（即应有两个量参与运算），都可以与赋值符一起组合成复合赋值符：

＋＝，－＝，*＝，／＝，％＝；

<<=，>>=，&=，∧=，|= 5种是有关位运算的。

C++采用复合运算符，一是为了简化程序，使程序精炼，二是为了提高编译效率（“逆波兰”式）。

逆波兰表达式（Reverse Polish Notation，RPN）

- 对于由两个运算数和一个运算符组成的表达式，习惯上是将运算符写在两个运算数中间，这叫做中缀形式，例如A+B 。
- 计算机处理表达式时，常把运算符放在两个运算数的后面或前面。
  - 1.把运算符放在两个运算数的前面，例如 +AB，则称做前缀形式，也叫做波兰式 。
  - 2.把运算符放在两个运算数的后面，例如 AB+ ，称为后缀形式，也叫做逆波兰表达式。

- 这个知识点在数据结构和编译原理这两门恐怖的课程中都有介绍，下面是一些例子：

  正常的表达式　　逆波兰表达式

- a+b　　a,b,+
- a+(b-c)　　a,b,c,-,+
- a+(b-c)*d　　a,d,b,c,-,*,+

  那么a=1+3 就写成 a=1,3 +了。

  http=(smtp+http+telnet)/1024 写成什么呢？

  http=1024,smtp,http,telnet,+,+,/

- 逆波兰表示的表达式的优点在于它得到的是不加括号的表示式，从表示式中就能确定表达式的运算先后顺序（在解析表达式时就解决了优先级的问题）。
  主要在编译原理上应用，用了两个堆栈：
  一个运算数堆栈
  一个操作数堆栈
  每遇到一个操作符就从栈顶取出相应个数的的操作数，结果再压入栈中，充分利用了栈的特性，不用判断优先级了。
- 在编译表达式值的处理等方面应用中经常用到。

2.6.4 赋值表达式

它的一般形式是：

<变量> <赋值运算符> <表达式>

赋值运算符左侧的标识符称为“左值”（left value）。

赋值运算符右侧的表达式称为“右值”（right value）。

赋值表达式中的“表达式”，又可以是一个赋值表达式。

例如：

```
a=5+(c=6);
a=(b=4)+(c=6) ;
a=(b=10)/(c=2);
```

赋值表达式作为左值时应加括号。

请分析下面的赋值表达式：

（a=3*5）=4*3

赋值表达式也可以包含复合的赋值运算符。

例如　　a+=a-=a*a

设a的初值为12，则有：

① 先进行“a-=a*a”的运算，

它相当于a=a-a*a=12-144＝-132。

② 再进行“a+=-132”的运算，

它相当于a=a+（-132）＝-132-132＝-264

赋值操作还可以以表达式形式出现在其他语句（如输出语句、循环语句等）中。这是C++语言灵活性的一种表现。

注意：用cout语句输出一个赋值表达式的值时，要将该赋值表达式用括号括起来，写成“cout<<（a=b）;”

如果写成“cout<<a=b;”将会出现编译错误。

2.7 逗号运算符与逗号表达式

C++提供一种特殊的运算符——逗号运算符。用它将两个表达式连接起来。如

3+5,6+8

称为逗号表达式，又称为“顺序求值运算符”。逗号表达式的一般形式为：

表达式 1 ，表达式 2

逗号表达式的求解过程是：先求解表达式1，再求解表达式2。整个逗号表达式的值是表达式2的值。例如逗号表达式：a=3*5,a*4

```
#include <iostream>
using namespace std;
int main()
{
        int a,x,y;
        x=((a=3*5,a*4),a+5);
        //cout<<a<<'\n';
        //x=(a=3,6*3);
        //y=a=3,6*a;
        cout<<x<<'\n'<<a<<'\n';
        //cout<<y<<'\n'<<a<<'\n';
        //cout<<(3*5,43-6*5,67/3)<<endl;
        return 0;
}
```

- 以下符号中不能作为标识符的是：（ ）

  A、_256　　B、 void

  C、 scanf　　D、 Struct

- C++程序的执行总是从哪里开始的？（ ）

  A、main函数　　B、 第一行

  C、 头文件　　D、 函数注释

小 结

- 数据的存储方式
- 标识符的命名规则
- 符号常量和常变量
- 运算符的优先级与结合性
- 自动进行类型转换

思考

- 若a为double型的变量，表达式a=1,a+5,a++的值为_____。

- 1？
- 7？

作　业

- 课本习题
- **P42 第2章 5、6、7、8**
- 分别输出整数和浮点数各类型的字节长度和位长，输出形式如：
  long int: 4 byte 32bit
- 第三章预习：
- 什么是过程化（结构化）编程？
- 如何进行过程化（结构化）编程？