第2章 数据类型与表达式

2.0 预备知识

- ★ 计算机中数据的表示及进制转换
- ★ 字节和位
- ★ 数值的表示方法——原码、反码和补码

★ 计算机中数据的表示

表示数值的方法：**数制**，又称为进位计数制，即按进位制的方法进行计数。常用的：十进制、二进制、八进制、十六进制。

数制	十进制数	二进制数	八进制数	十六进制数
数码	0~9	0~1	0~7	0~9,A~F,a~f
基	10	2	8	16
权	10^0，10^1，10^2，…	2^0，2^1，2^2，…	8^0，8^1，8^2，…	16^0，16^1，16^2，…
表示	十六进制：$81AE=8\times16^3+1\times16^2+10\times16^1+14\times16^0$			
特点	逢十进一	逢二进一	逢八进一	逢十六进一

★ 各种进制之间的转换

① 二进制、八进制、十六进制转换成十进制

方法：按权相加

例 $(111011)_2 = 1\times2^5 + 1\times2^4 + 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0 = (59)_{10}$

例 $(136)_8 = 1\times8^2 + 3\times8^1 + 6\times8^0 = (94)_{10}$

例 $(1F2A)_{16} = 1\times16^3 + 15\times16^2 + 2\times16^1 + 10\times16^0 = (7978)_{10}$

② 十进制转换成二进制、八进制、十六进制

③ 二进制与八进制之间的转换

④ 二进制与十六进制之间的转换

略（自学）

★ 字节和位

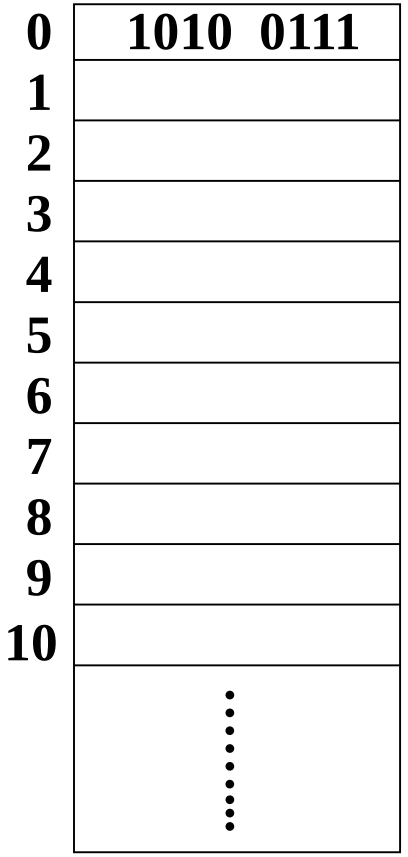
① 内存以字节为单元组成

② 每个字节有一个地址

③ 一个字节由8个二进制位组成

④ 每个二进位的值是0或1

7	6	5	4	3	2	1	0
1	0	1	0	0	1	1	1



★ 数值的表示方法——原码、反码和补码

- **原码：**

  最高位为符号位，其余各位为数值本身的绝对值

- **反码：**

  正数：反码与原码相同

  负数：符号位为1，其余位对原码取反

- **补码：**

  正数：原码、反码、补码相同

  负数：最高位为1，其余位为原码取反，再对整个数加1

(用一字节表示数)

	原码	反码	补码
+7	00000111	00000111	00000111
-7	10000111	11111000	11111001
+0	00000000	00000000	00000000
-0	10000000	11111111	00000000
数的范围	01111111~ 11111111 (-127~+127)	01111111~ 10000000 (-127~+127)	01111111~ 10000000 (-128~+127)

❖负数补码转换成十进制数：最高位不动，其余位取反加1

例 补码：11111001

取反：10000110

加1：10000111=-7

2.1 C++的数据类型

```
// 求两数之和                    (本行是注释行)
#include <iostream>              //预处理命令
using namespace std;             //使用命名空间std
int main( )                      //主函数首部
{                                //函数体开始
    int a,b,sum;                 //定义变量
    cin>>a>>b;                   //输入语句
    sum=a+b;                     //赋值语句
    cout<<"a+b="<<sum<<endl;     //输出语句
    return 0;                    //程序结束
}                                //函数体结束
```

2.1 C++的数据类型

★数据类型总表

数据类型决定:
1. 数据占内存字节数
2. 数据取值范围
3. 其上可进行的操作

★基本数据类型

类型	符号	关键字	所占位数	数的表示范围
整型	有	(signed)int	32	-2147483648~2147483647
		(signed)short	16	-32768~32767
		(signed)long	32	-2147483648~2147483647
	无	unsigned int	32	0~4294967295
		unsigned short	16	0~65535
		unsigned long	32	0~4294967295
实型	有	float	32	3.4e-38~3.4e38
	有	double	64	1.7e-308~1.7e308
	有	long double	64	1.7e-308~1.7e308
字符型	有	char	8	-128~127
	无	unsigned char	8	0~255

说明:数据类型所占字节数随机器硬件不同而不同,上表以PC机和VC为例

说明：

(1) 整型数据分为长整型(long int)、一般整型(int)和短整型(short int)。

(2) 整型数据的存储方式为按二进制数形式存储，例如十进制整数85的二进制形式为1010101，则在内存中的存储形式：

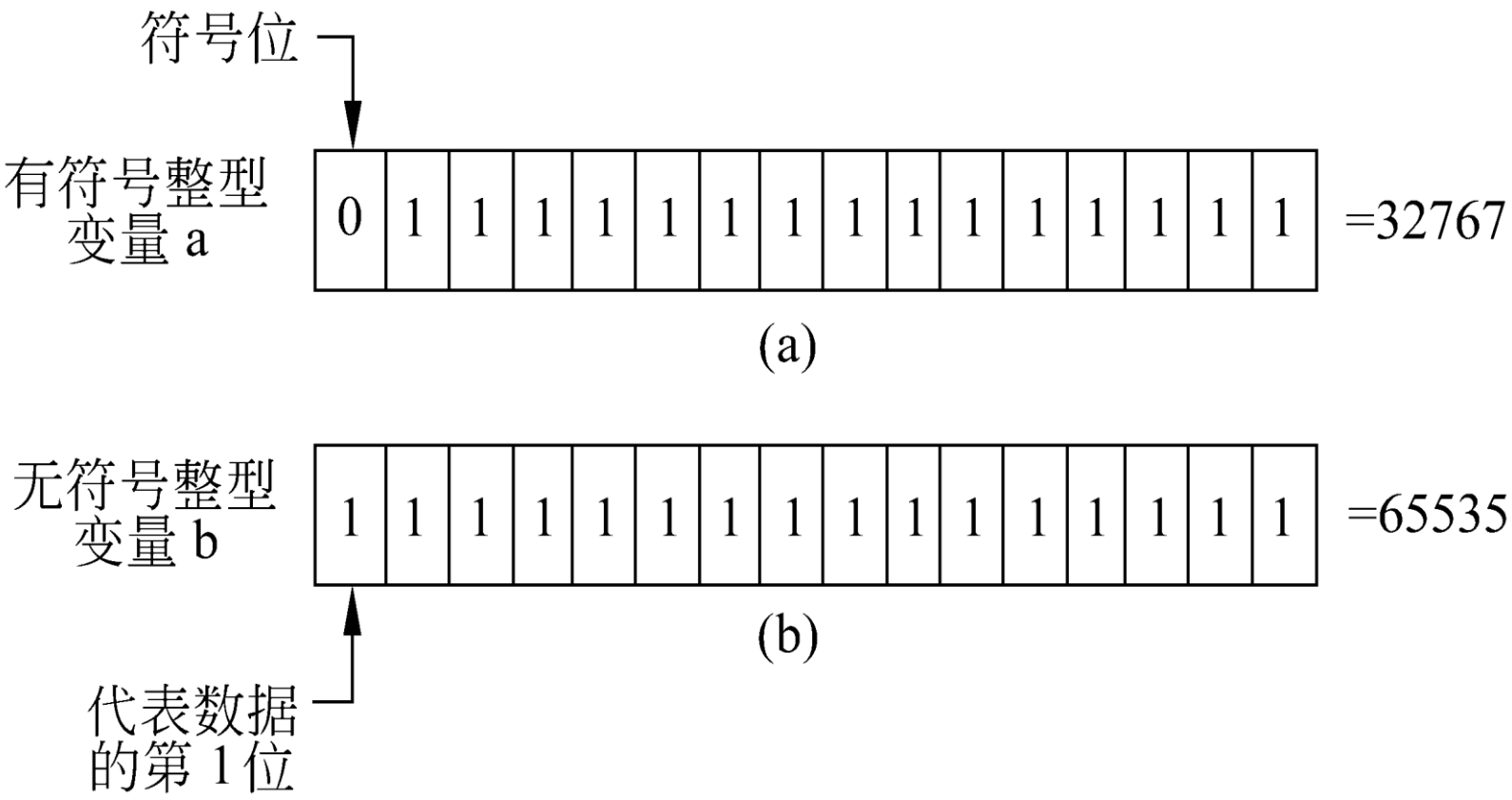
16位：

0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1

32位：

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1

(3) 在整型符号int和字符型符号char的前面,可以加修饰符signed(表示“有符号”)或unsigned(表示“无符号”)。例如短整型数据占两个字节：



(4) 浮点型(又称实型)数据分为单精度

- float：分配4个字节，提供6位有效数字
- double：分配8个字节，提供15位有效数字
- long double：同double。

(5) short和short int等效，long和long int等效，unsigned int和unsigned等效。

2.2 常量

★常量

- 定义：运行时其值不能改变的量（即常数）
- 分类:
  - 整型常量
  - 实型常量
  - 字符常量
  - 字符串常量

整型常量（整常数）

三种形式：

- 十进制整数：由数字0~9和正负号表示. 如 123, -456, 0
- 八进制整数：由数字0开头,后跟数字0~7表示. 如0123, 011
- 十六进制整数：由0x开头,后跟0~9,a~f,A~F表示. 如 0x123, 0Xff

问题：

0123 = (83)10

0x123 = (291)10

0Xff = (255)10

```
#include<iostream>
using namespace std;
int main()
 {      int a=0xff;
        cout<<a<<endl;
        return 0;
 }    // 255
```

整型常量（整常数）

- 三种形式：
- 整型常量的类型
  - 根据其值所在范围确定其数据类型
  - 在整常量后加字母l或L，认为它是long int 型常量

例 30000 为short int型
65536 为 int 型

实型常量（实数或浮点数）

- 表示形式：
  - 十进制数形式：（必须有小数点）

    如 0.123，.123，123.0，0.0，123.
  - 指数形式：（e或E前必须有数字；指数必须为整数）

    如12.3e3 , 123E2, 1.23e4 , e-5 , 1.2E-3.5
- 实型常量的类型
  - 默认double型
  - 在实型常量后加字母f或F，认为它是float 型

字符常量

- 定义:用单引号括起来的单个普通字符或转义字符.

如 ‘a’ ‘A’ ‘?’ ‘\n’ ‘\101’

- 字符常量的值：该字符的ASCII码值

如 ‘A’——65，‘a’——97， ‘0’——48，‘\n’——10

- 转义字符:反斜线后面跟一个字符或一个代码值表示

如 ‘\101’ ----------‘A’ ‘\012’ ----------’\n’ ‘\376’ ----------’■’ ‘\x61’ ----------’a’ ‘\60’ ----------’0’ ‘\483’ ----------(错误)

字符常量

❖**转义字符**：反斜线后面跟一个字符或一个代码值表示

转义字符及其含义：

转义字符	含义	转义字符	含义
\n	换行	\t	水平制表
\v	垂直制表	\b	退格
\r	回车	\f	换页
\a	响铃	\\	反斜线
\‘	单引号	\“	双引号
\ddd	3位8进制数代表的字符	\xhh	2位16进制数代表的字符

字符常量

例 转义字符举例

```
#include<iostream>
using namespace std;
int main()
{ cout<< "\101 \x42 C\n";
   cout<< "I say:\"How are you?\"\n";
   cout<< "\\C Program\\\n";
   cout<< "Turbo \'C\'"<<endl;
   return 0;
}
```

运行结果：(屏幕显示)

A B C
I say:”How are you?”
\C Program\
Turbo ‘C’

- 字符数据的使用方法
  - 字符数据和整型数据之间可以运算。
  - 字符数据与整型数据可以互相赋值。

- 字符串常量:

用双引号括起来的一串字符

例："CHINA" 

"a" 

'a' 

所以：char c; c="a"; 

```
#include<iostream>
using namespace std;
int main()
{
    int num=10;
    char ch;
    ch =num+56;
    cout<< num <<ch<<endl;
    return 0;
}
```

结果：10B

★符号常量

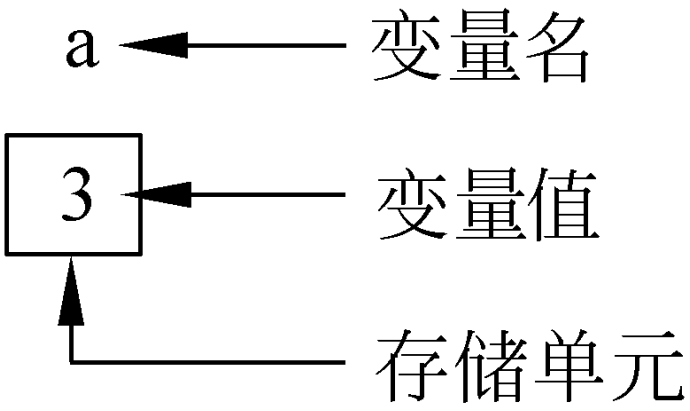
- ●符号常量：用标识符代表常量
  - ◆定义格式： #define 符号常量标识符 常量
  - ◆一般用大写字母
  - ◆是宏定义预处理命令，不是C语句

```
#include<iostream>
#define  PRICE  30
using namespace std;
int main()
{    int num,total;
     num=10;
     total=num*PRICE;
     cout<< "total= " <<total<<endl;
     return 0;
}
```

运行结果：
total=300

2.3 变量

在程序运行期间其值可以改变的量称为**变量**。



变量名规则

- 定义：用来标识变量、常量、函数等的字符序列
- 组成：
  - 只能由字母、数字、下划线组成，
  - 且第一个字母必须是字母或下划线
  - 大小写敏感，如 sum与Sum不同
  - 不能使用关键字，如 int, char, main等

例:判断下列标识符号合法性

sum Sum M.D.John day Date 3days

student_name #33 lotus_1_2_3

char a>b _above $123

定义变量

- 变量必须“先定义，后使用”
- 定义变量的一般形式:

  变量类型 变量名表列;

  例： **float a,b,c,d,e;**
- 定义变量时指定它的初值。

  例： **float a=83.5,b,c=64.5,d=81.2,e;**

❖对多个变量赋予同一初值，必须分别指定，不能写成

```
float a=b=c=4.5;
```

而应写成

```
float a=4.5,b=4.5,c=4.5;
```

常变量

❖在定义变量时，如果加上关键字const，则变量的值在程序运行期间不能改变，此变量称为常变量。

例： const int a=3;

❖在定义常变量时必须同时对它初始化(即指定其值)

❖常变量不能出现在赋值号的左边。

例：const int a=5;

a=3; //常变量不能被赋值

❖可以用表达式对常变量初始化

例：

```
const int b=3+6,c=3*cos(1.5);
```

注意：

使用了系统函数cos，必须包含头文件

#include <cmath> 或 #include <math.h>

2.4 C++的运算符

- 算术运算符

  +、-、*、/、 %、++、 --

- 关系运算符

  >、<、 ==、>=、<=、!=

- 逻辑运算符

  &&、||、 !

- 位运算符

  <<、>>、&、|、∧、~

- 赋值运算符 (＝及其扩展赋值运算符)
- 条件运算符 (？:)
- 逗号运算符 (,)
- 指针运算符 (*)
- 引用运算符和地址运算符 (&)
- 求字节数运算符 (sizeof)
- 强制类型转换运算符 ((类型) 或类型())
- 成员运算符 (.)
- 指向成员的运算符 (－>)
- 下标运算符 ([ ])
- 其他 (如函数调用运算符 ( ))

- 学习运算符应注意：（如：5/2）
  - 运算符功能
  - 与运算量关系
    - 要求运算量个数
    - 要求运算量类型
  - 运算符优先级别
  - 结合方向
  - 结果的类型

2.5 算术运算符和表达式

- 基本算术运算符： + - * / %
  - 结合方向：从左向右
  - 优先级： - ----> * / % -----> + -
    (3) (4) (5)

说明：

- “-”为单目运算符时,右结合
- 两整数相除，结果为整数
- %要求两侧均为整型数据

例 5/2 = 2
-5/2.0 = -2.5

运行结果：
1
-1

```
#include <iostream>
using namespace std;
int main( )
{
    cout<<5%2<<endl;
    cout<<-5%2<<endl;
    return 0;
}
```

例：

```
#include <iostream>
#define   PRICE    12.5
using namespace std;
void   main()
      {   int   num=3;
          float   total;
          char   ch1,ch2='D';

          total=num*PRICE;
          ch1=ch2-'A'+'a';
          cout<<"total="<<total<<",ch1="<<ch1<<endl;
      }
```

宏定义

变量定义

输出结果

运行结果：

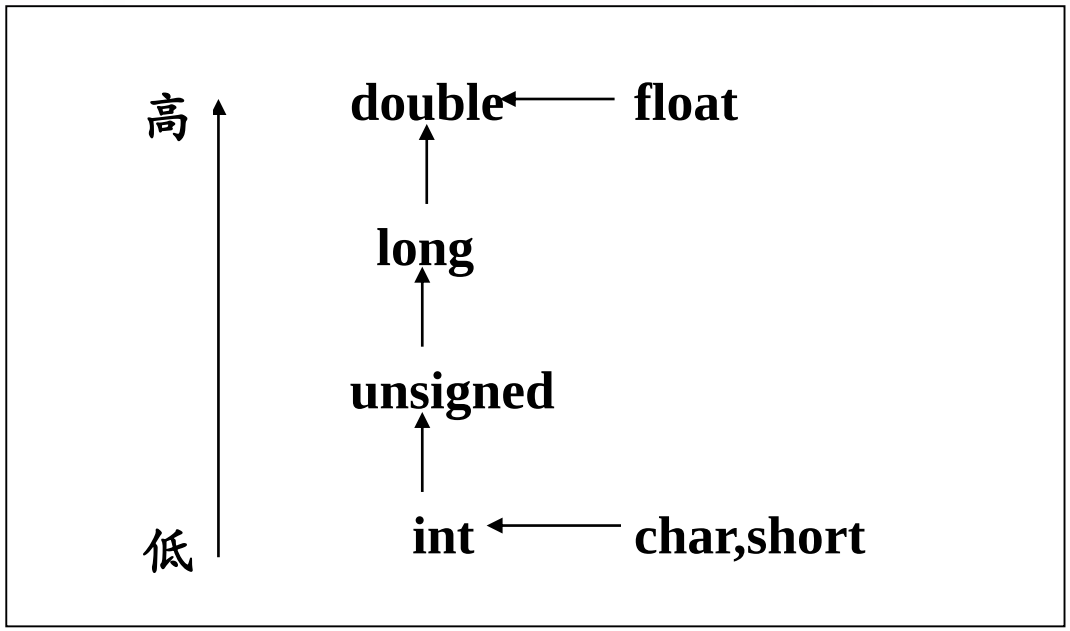
total=37.5, ch1=d

表达式中各类数值型数据间的混合运算

隐式转换

- 什么情况下发生
  - 运算转换------不同类型数据混合运算时
  - 赋值转换------把一个值赋给与其类型不同的变量时
  - 输出转换------输出时转换成指定的输出格式
  - 函数调用转换------实参与形参类型不一致时转换
- 运算转换规则:

  不同类型数据运算时先自动转换成同一类型

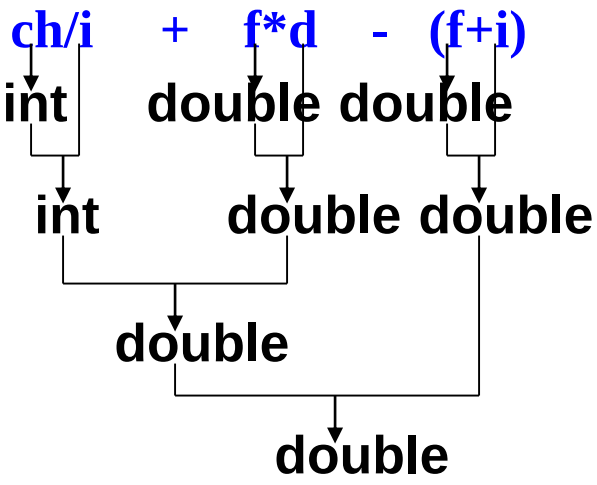


说明：

← 必定的转换

↑ 运算对象类型不同时转换

例
```
char ch;
int i;
float f;
double d;
```



- 自增、自减运算符++ --
  - 作用：使变量值加1或减1
  - 种类：
    - 前置：++i, --i （先执行i+1或i-1，再使用i值）
    - 后置：i++, i-- （先使用i值，再执行i+1或i-1）

```
例 j=3;  k=++j;                    //k=4,j=4
   j=3;  k=j++;                    //k=3,j=4
   j=3;  cout<<++j<<endl;          //4
   j=3;  cout<<j++<<endl;          //3
   a=3; b=5;c=(++a)*b;             //c=20,a=4
   a=3; b=5;c=(a++)*b;             //c=15,a=4
```

自增、自减运算符++ --

- 说明：
  - ++ -- 不能用于常量和表达式,如5++，(a+b)++
  - ++ -- 结合方向： 自右向左
  - 优先级：

++ --(后置) ---> - ++ --(前置)---> * / % ---> + -

(2) (3) (4) (5)

例 -i++ ⇔ -(i++)

```
i=3;  cout<<-i++<<endl;   //-3
```

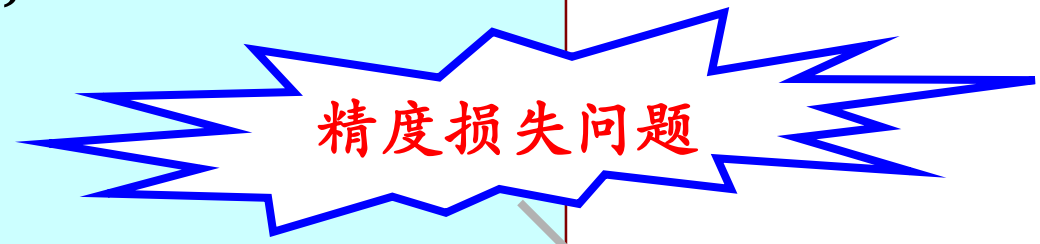
- 显式转换（强制转换）
  - 一般形式：（类型名）（表达式）

例：(int)(x+y)　　(int)x+y　　(double)(3/2)　　(int)3.6

  - 强制转换得到所需类型的中间变量，原变量类型不变

```
#include <iostream>
using namespace std;
int main()
{     float  x;
      int  i;
      x=3.6;
      i=(int)x;
      cout <<"x="<<x<<"i="<<i<<endl;
      return 0;
}
```

结果：x=3.6，i=3



较高类型向较低类型转换时可能发生

2.6赋值运算符与赋值表达式

例
```
a=3;
d=func();
c=d+2;
```

- 简单赋值运算符
  - 符号： =
  - 格式： 变量标识符=表达式
  - 作用：将一个数据（常量或表达式）赋给一个变量
- 复合赋值运算符
  - 种类：+= -= *= /= %= 《= 》= &= ^= |=
  - 含义： exp1 op= exp2 ⇔ exp1 = exp1 op exp2

a+=3 ⟺ a=a+3

x*=y+8 ⟺ x=x*(y+8)

- 说明:
  - 结合方向: 自右向左
  - 优先级: 15
  - 左侧必须是变量，不能是常量或表达式
  - 赋值转换规则:使赋值号右边表达式值自动转换成其左边变量的类型

例 3=x-2*y;
a+b=3;

```
例  int i;
    i=2.56;   //结果i=2;
```

```
例  float f;
    int i=10;
    f=i;
则  f=10.0
```

- 说明:
  - 结合方向: 自右向左
  - 优先级: 15
  - 左侧必须是变量，不能是常量或表达式
  - 赋值表达式的值与变量值相等,且可嵌套

```
例:  a=b=c=5                //表达式值为5，a,b,c值为5
     a=5+(c=6)              //表达式值11，c=6,a=11
     a=(b=10)/(c=2)         //表达式值5，a=5,b=10,c=2
```

```
例: a=12;
    a+=a-=a*a       //a=-264 等价于a=a+(a=a-(a*a))
```

例 将有符号数据传送给无符号变量。

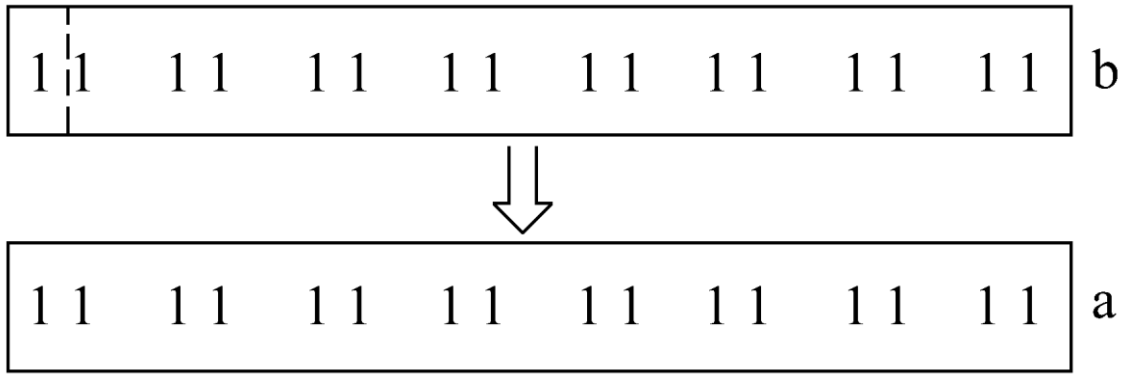
```
#include <iostream>
using namespace std;
int main( )
{ unsigned short a;
  short int b=-1;
  a=b;
  cout<<" a=" <<a<<endl;
  return 0;
}
```

不同类型的整型数据间的赋值归根到底就是一条：

按存储单元中的存储形式直接传送！

运行结果为：

a=65535



2.7 逗号运算符和表达式

- 形式：表达式1,表达式2,……表达式n
- 结合性: 从左向右
- 优先级: 17
- 逗号表达式的值：等于表达式n的值

```
例   a=3*5,a*4          //a=15,表达式值60
     a=3*5,a*4,a+5      //a=15,表达式值20
例   x=(a=3,6*3)        //赋值表达式，表达式值18，x=18
     x=a=3,6*a          //逗号表达式,表达式值18,x=3
```

2.7 逗号运算符和表达式

- 形式：表达式1,表达式2,……表达式n
- 结合性: 从左向右
- 优先级: 17
- 逗号表达式的值：等于表达式n的值
- 用途：常用于循环for语句中

```
#include <iostream>
using namespace std;
int main()
{   int x,y=7;    float z=4;
    x=(y=y+6,y/z);
    cout<<"x="<<x<<endl;
    return 0;
}
```

运行结果：x=3

作业：

$P_{37\sim38}$ 4~7