



中國人民大學

RENMIN UNIVERSITY OF CHINA

信息学院

SCHOOL OF INFORMATION

程序设计荣誉课程

2. 语法1——程序设计基本概念

授课教师：游伟 副教授、孙亚辉 副教授

授课时间：周二14:00 – 15:30，周四10:00 – 11:30（教学三楼3304）

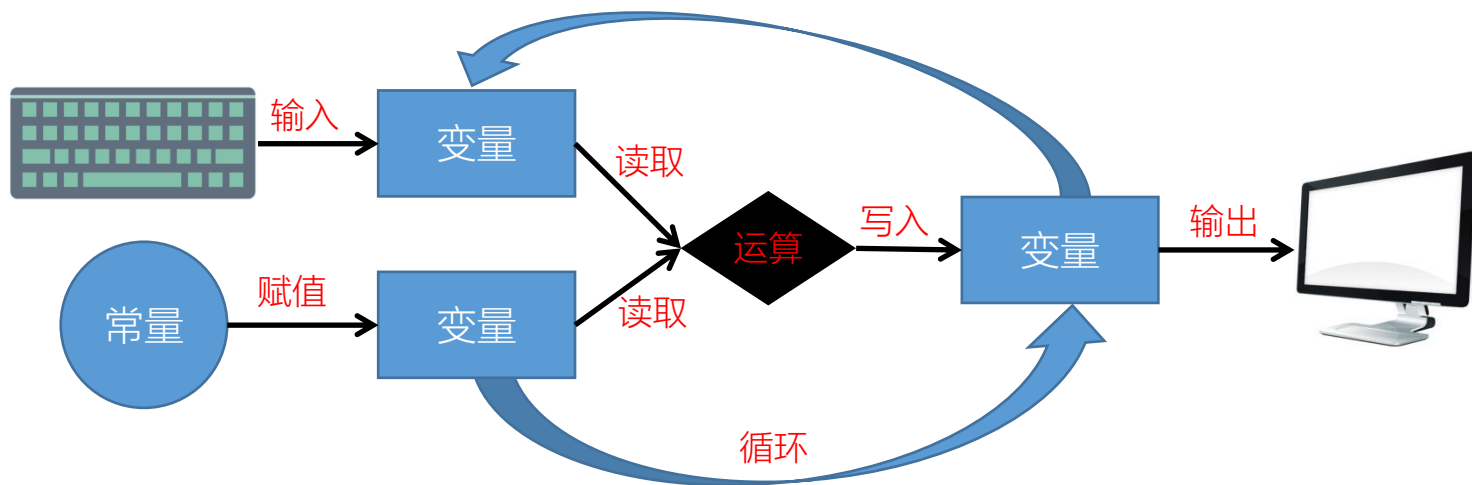
上机时间：周二18:00 – 21:00（理工配楼二层机房）

课程主页：<https://www.youwei.site/course/programming>

引子：21级图灵班选拔卷第四题

有一门简易程序语言（我们称它为**T语言**），支持下列6种基本操作：

- INPUT n: 读入用户**输入**并存放在变量n中
- OUTPUT z: 将变量z的值**输出**到屏幕
- $z \leftarrow c$: 将变量z的值**赋值**为c（即 $z=c$ ），c是一个**常量**或者另一个**变量**的值
- $z \leftarrow \text{ADD}(x,y)$: **读取**变量x和y的值，进行加法**运算**，结果**写入**变量z（即 $z=x+y$ ）
- $z \leftarrow \text{MUL}(x,y)$: 读取变量x和y的值，进行乘法运算，结果写入变量z（即 $z=x*y$ ）
- FOR i : 1 to n {}: **循环**执行{}内的操作n次，变量i初始值为1，每次循环i的值加1



引子：21级图灵班选拔卷第四题

示例1：使用该语言实现计算 2^n ($n \geq 1$)的示例代码如下：

```
1. INPUT  n
2. z←1           // 设置变量z的初始值为1
3. y←2           // 设置变量y的初始值为2
4. FOR i : 1 to n {           // 循环n次
5.   z←MUL(z, y)             // 每次循环执行时，变量z的值都变成原来的2倍
6. }
7. OUTPUT  z
```

示例2：使用该语言实现计算 $1 + 2 + \dots + n$ ($n \geq 3$)的示例代码如下：

```
1. INPUT  n
2. z←0           // 设置变量z的初始值为0
3. FOR i : 1 to n {           // 循环n次，第一轮循环时，变量i的值为1
4.   z←ADD(z, i)             // 每次循环执行时，变量z的值都累加上变量i的当前值
5. }                       // 每次循环后，变量i的值自动加1
6. OUTPUT  z
```

引子：21级图灵班选拔卷第四题

【问题1】当输入为5时，下列代码的输出结果是多少？ 答案：8（斐波拉契数列）

阅读理解

```
1. INPUT n
2. x←0
3. y←1
4. FOR i : 1 to n {
5.     z←ADD(x, y)
6.     x←y
7.     y←z
8. }
9. OUTPUT z
```

【问题2】下列代码用于实现计算 $1^2 + 2^2 + \dots + n^2$ ($n \geq 3$)，请补全缺失代码

完型填空

```
1. INPUT n
2. z←0
3. FOR i : 1 to n {
4.     x←MUL(i, i)（请在此处补全缺失的代码）
5.     z←ADD(z, x)
6. }
7. OUTPUT z
```

【问题3】请编写代码实现计算 $1! + 2! + \dots + n!$ ($n \geq 3$)

写作

```
1. INPUT n
2. z←0
3. x←1
4. FOR i : 1 to n {
5.     x←MUL(x, i)
6.     z←ADD(z, x)
7. }
8. OUTPUT z
```

目录

1. 简单的C/C++语言程序
2. 数据类型
3. 常量和变量
4. 运算符和表达式
5. 数学函数
6. 语句
7. 数据的输入输出
8. 数据类型与安全

2.1 简单的C/C++语言程序

■ 计算 $1!+2!+\dots+n!$ 的C语言代码 (first.c)

```
1. #include <iostream>                //引用头文件
2. #define SUM_INIT 0                 //定义符号常量SUM_INIT
3. int z = SUM_INIT;                  /* 定义全局变量z, 并进行初始化 */
4. int main(int argc, char **argv) {
5.     int x = 1;                      /* 定义局部变量x, 并进行初始化 */
6.     int i, n;                       /* 定义局部变量i和n, 未进行初始化 */
7.     scanf("%d", &n);                //输入
8.     for (i = 1; i <= n; i++) {       //L8-11行: 循环语句
9.         x = x * i;
10.        z = z + x;
11.    }
12.    printf("%d", z);                 //输出
13.    return 0;                       //函数返回值
14.}
```

2.1 简单的C/C++语言程序

■ 计算 $1!+2!+\dots+n!$ 的C++语言代码 (first.cpp)

```
1. #include <iostream>                //引用头文件
2. #define SUM_INIT 0                 //定义符号常量SUM_INIT

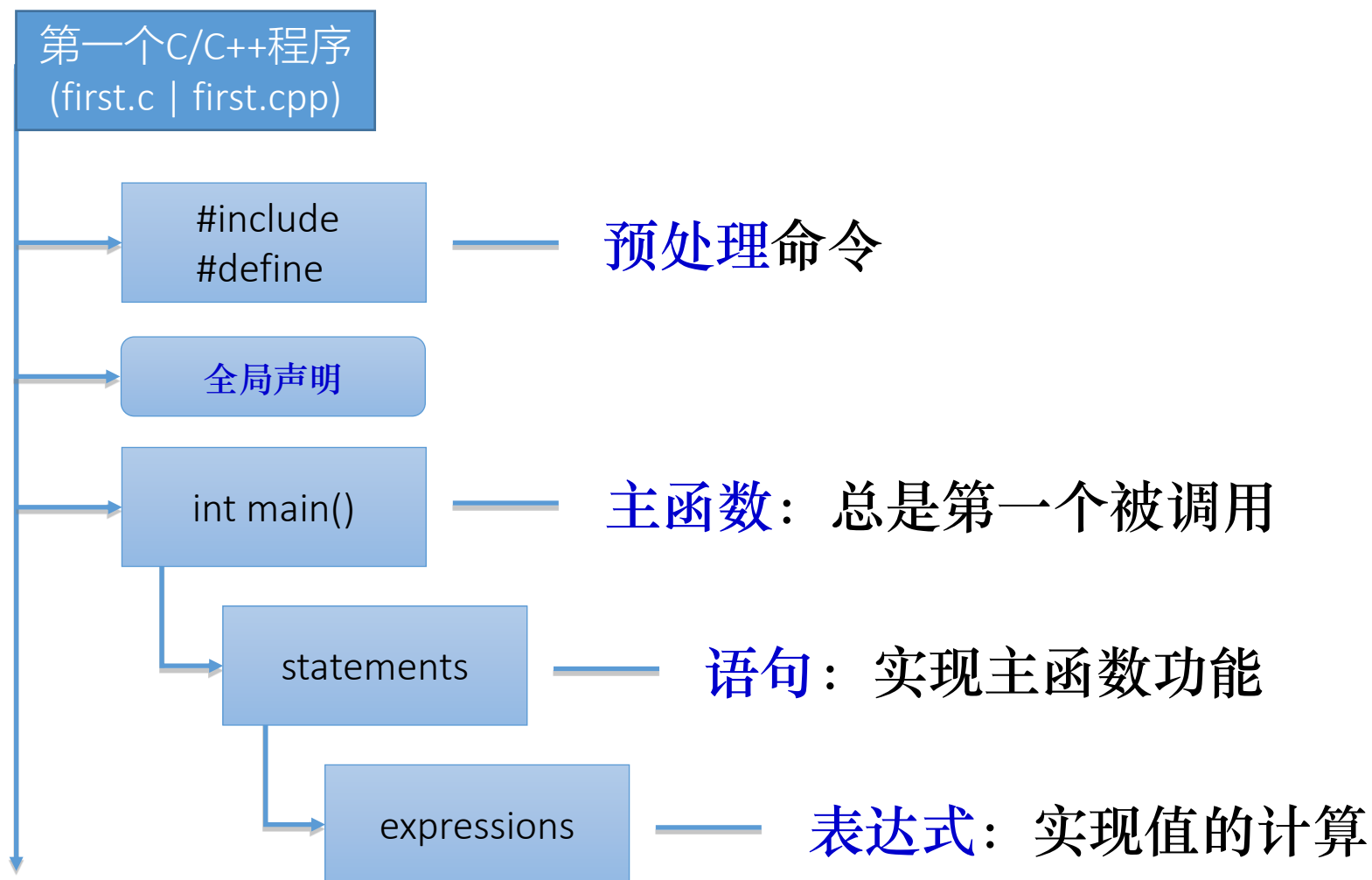
3. int z = SUM_INIT;                  /* 定义全局变量z, 并进行初始化 */

4. int main(int argc, char **argv) {
5.     int x = 1;                      /* 定义局部变量x, 并进行初始化 */
6.     int i, n;                       /* 定义局部变量i和n, 未进行初始化 */

7.     std::cin >> n;                  //输入
8.     for (i = 1; i <= n; i++) {       //L8-11行: 循环语句
9.         x = x * i;
10.        z = z + x;
11.    }

12.    std::cout << z;                 //输出
13.    return 0;                       //函数返回值
14.}
```

2.1.1 C++语言程序结构



注释：//开始的单行与/* */的多行

2.1.1 C++语言程序结构

■ 以符号“#”开头的行，称为预处理命令

- #include称为文件预处理命令，一般用于引用库的头文件
- #define称之为宏定义命令，一般用于定义符号常量

■ 全局声明：在函数之外进行的数据声明

■ 函数定义

- 函数首部

int	main	(int	argc,	char	**argv)
↓	↓	↓	↓	↓	↓
返回值类型	函数名	参数类型	参数名	参数类型	参数名

- 函数体：声明部分定义本函数中所用到的局部变量；执行部分由若干语句组成，指定在函数中所进行的操作

■ main函数

- 一个完整的主函数是一个合法C程序的最基本组成部分
- C程序的执行是从main函数第一条语句开始，到main函数运行结束为止

■ 语句std::cin和std::cout分别被调用来进行输入和输出

- std::cin和std::cout是编译系统提供的一个标准库函数，不是C++语言自身组成部分
- C++语言自身只定义了基本计算、操作、数据类型，以及数据和程序的组织方法
- 大量复杂功能，包括输入/输出都以标准库函数的方式，由具体编译系统提供

#include指令说明

三种形式：

#include "c:\cpp\include\myfile.h" ①

#include "myfile.h" ② ③

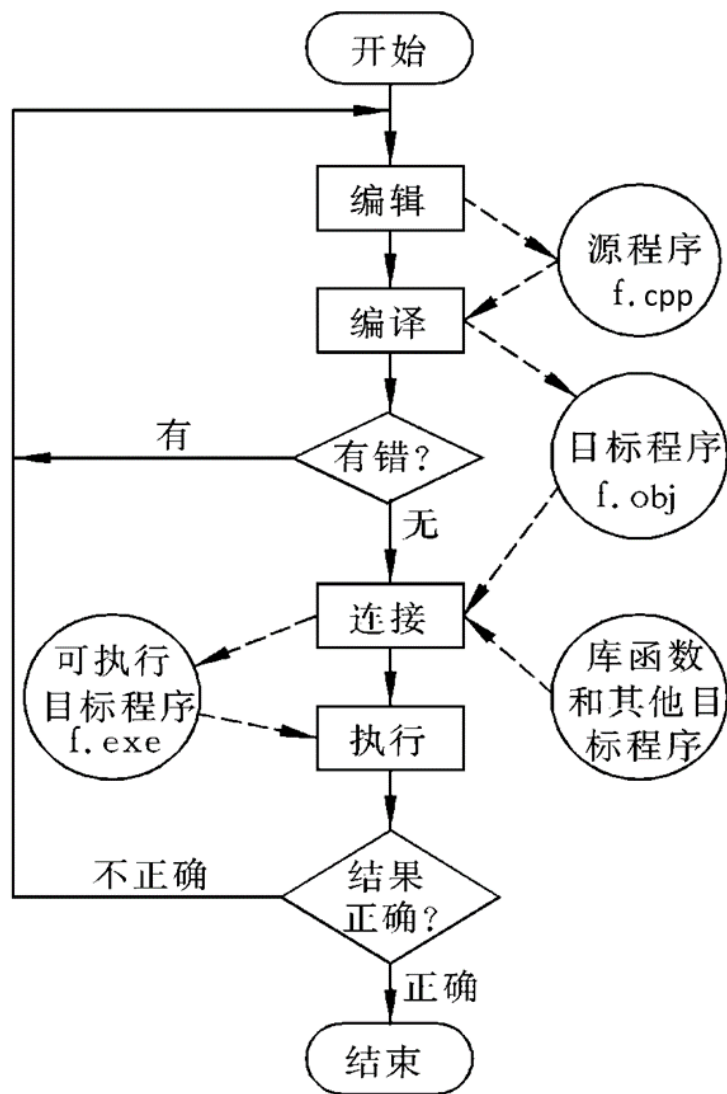
#include <myfile.h> ③

① 按指定路径查找文件

② 源程序文件所在目录

③ C编译系统指定的include目录

2.1.2 C++程序生成、调试和运行



2.2 数据类型

- 计算机中的各种数据是存储在内存空间中
 - 整数、实数、字符.....
 - 不同类型的数据占用大小不同的内存空间
- 数据类型可分为两大类
 - 基本数据类型：整数型，浮点型，字符型
 - 构造数据类型：数组、枚举、结构体、共用体
 - 注：构造数据类型，是由若干个基本数据类型的变量按特定的规律组合构造而成的

■ 无符号整数

$$B2U(\vec{x}) \triangleq \sum_{i=0}^{\omega-1} x_i \cdot 2^i$$

值的范围: $[0, 2^{\omega} - 1]$

■ 有符号整数

■ 补码 (Two's-Complement): 补码 + |负数| = 2^{ω} (模数), 补码=反码+1

$$B2T_{\omega}(\vec{x}) \triangleq -x_{\omega-1} \cdot 2^{\omega-1} + \sum_{i=0}^{\omega-2} x_i \cdot 2^i$$

值的范围: $[-2^{\omega-1}, 2^{\omega-1} - 1]$

2.2.1 数据类型概览

- 有符号(signed)与无符号(unsigned)的区分
- 表示整数使用int, short, long, long long四种类型
- 表示实数使用float, double, long double三种类型
- 表示字符使用char类型

类型	占用内存字节数 (sizeof)	值域	
		signed	unsigned
char	1 ($\omega = 8$)	$[-2^7, 2^7-1]$ ($[-128, 127]$)	$[0, 2^8-1]$ ($[0, 255]$)
short	2 ($\omega = 16$)	$[-2^{15}, 2^{15}-1]$ ($[-32768, 32767]$)	$[0, 2^{16}-1]$ ($[0, 65535]$)
int	4 ($\omega = 32$)	$[-2^{31}, 2^{31}-1]$ ($[-2147483648, 2147483647]$)	$[0, 2^{32}-1]$ ($[0, 4294967295]$)
long	4 ($\omega = 32$) 或 8 ($\omega = 64$)	$[-2^{31}, 2^{31}-1]$ 或 $[-2^{63}, 2^{63}-1]$	$[0, 2^{32}-1]$ 或 $[0, 2^{64}-1]$
long long	8 ($\omega = 64$)	$[-2^{63}, 2^{63}-1]$	$[0, 2^{64}-1]$
float	4 (有效数字: 6)	$\{0, [-3.4 \times 10^{38}, -1.2 \times 10^{-38}], [1.2 \times 10^{-38}, 3.4 \times 10^{38}]\}$	
double	8 (有效数字: 15)	$\{0, [-1.7 \times 10^{308}, -2.3 \times 10^{-308}], [2.3 \times 10^{-308}, 1.7 \times 10^{308}]\}$	
long double	16 (有效数字: 19)	$\{0, [-1.1 \times 10^{4932}, -3.4 \times 10^{-4932}], [3.4 \times 10^{-4932}, 1.1 \times 10^{4932}]\}$	

注: 以上是在64位Ubuntu系统上的结果, 不同平台可能略有差异。可以用sizeof操作符查看占用内存字节数。

2.2.2 整数类型

■ 多字节数据的存储与排列顺序

- 一个整数类型数据可能会占用多个存储单元
- 需要考虑以下问题：
 - 变量的地址是其最大地址还是最小地址？
 - 多个字节在存储单元中存放的顺序如何？
- 示例：int i = -65535，存放在100号单元（占100 ~ 103），则用“取数”指令访问100号单元取出i时，必须清楚i的4个字节是如何存放的

最高有效字节				最低有效字节			
MSB				LSB			
变量i	103	102	101	100	小端 (little endian)		
	FF	FF	00	01			
	100	101	102	103	大端 (big endian)		

$$65535 = 2^{16} - 1$$

$$[-65535]_{\text{补}} = \text{FFFF0001H}$$

大端方式 (Big Endian)：MSB所在的地址是数的地址 (e.g., MIPS)

小端方式 (Little Endian)：LSB所在的地址是数的地址 (e.g., Intel 80x86)

有些机器两种方式都支持，可通过特定控制位来设定采用哪种方式

2.2.2 整数类型

■ 多字节数据的存储与排列顺序

```
1. #include <stdio.h>

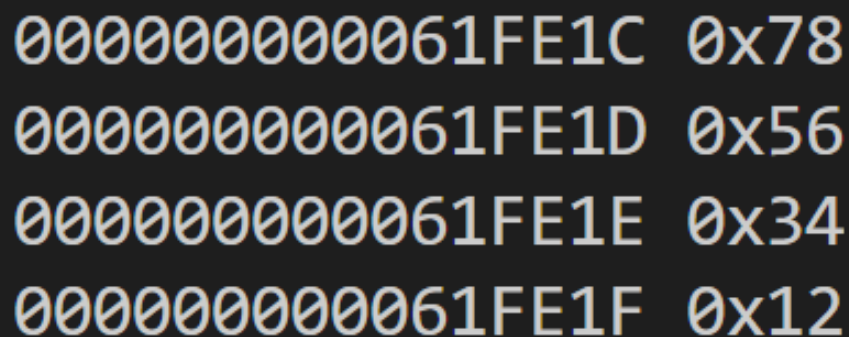
2. int main(int argc, char **argv) {
3.     int i = 0x12345678;
4.     unsigned char *p;

5.     p = &i;
6.     printf("%p 0x%x\n", p, *p);

7.     p++;
8.     printf("%p 0x%x\n", p, *p);

9.     p++;
10.    printf("%p 0x%x\n", p, *p);

11.    p++;
12.    printf("%p 0x%x\n", p, *p);
13.}
```



00000000000061FE1C	0x78
00000000000061FE1D	0x56
00000000000061FE1E	0x34
00000000000061FE1F	0x12

说明：左侧代码验证多字节数据的存储和排列顺序

- C++语言的优势：可以用指针直接访问内存数据
- 使用指针p，依次指向变量i的每一个字节
- 打印指针p指向的地址以及该地址的内容（单字节）
- 可以看到在Intel 80x86平台上，使用的是小端方式

- 有符号整数

- 原码 (Sign-Magnitude) : 除符号位以外, 其余位保持原样

$$B2S_{\omega}(\vec{x}) \triangleq (-1)^{x_{\omega-1}} \cdot \left(\sum_{i=0}^{\omega-2} x_i \cdot 2^i \right) \quad \text{值的范围: } [-2^{\omega-1} + 1, 2^{\omega-1} - 1]$$

- 移码 (Frame Shift) : 移码 = 补码 + $2^{\omega-1}$

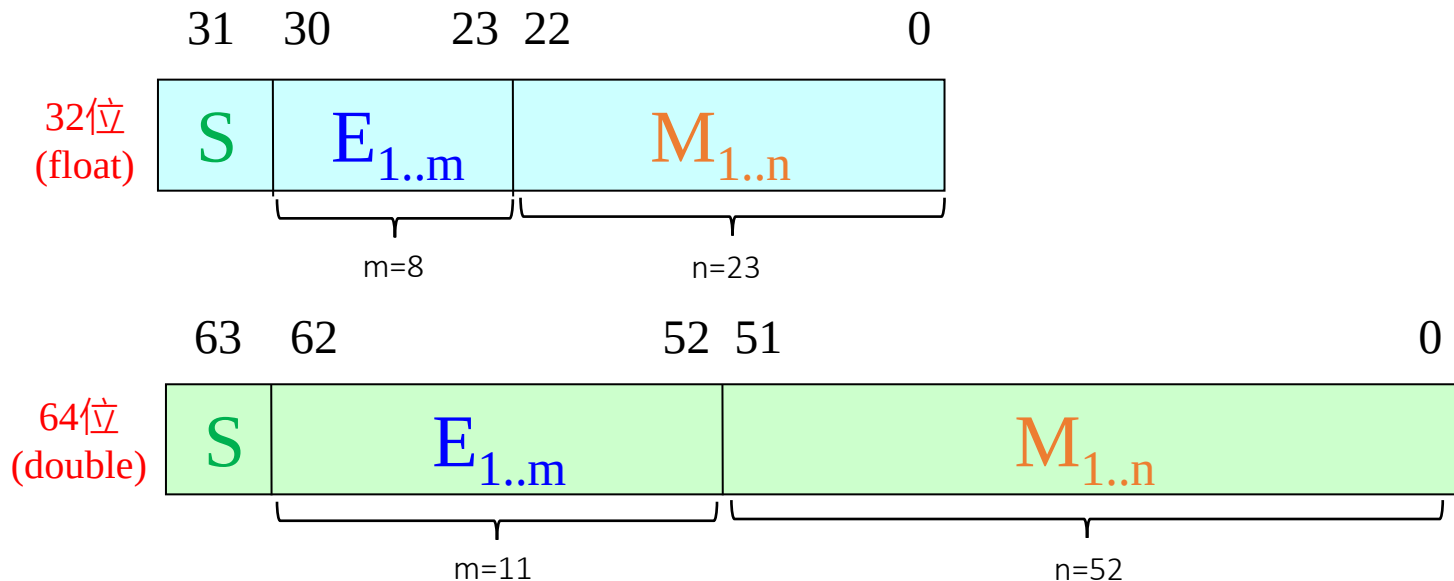
$$B2F(\vec{x}) \triangleq (-x_{\omega-1} \cdot 2^{\omega-1})^{(1-x_{\omega-1})} + \sum_{i=0}^{\omega-2} x_i \cdot 2^i \quad \text{值的范围: } [-2^{\omega-1}, 2^{\omega-1} - 1]$$

2.2.3 实数类型

■ 浮点数的标准格式IEEE754

- 尾数 (M) : 用原码表示, 规格化后使用隐藏位技术, 尾数符号在数据首位
- 阶码 (E) : 用移码表示, 偏移值为 $2^{m-1} - 1$ (float偏移量127, double偏移量1023)
- 基值 (R) : 等于2

科学计数法: $N = M \times R^E$



IEEE754的精妙之处

- 为何阶码使用移码表示：便于浮点数加减运算时对阶（比较大小）

例： $1.01 \times 2^{-1} + 1.11 \times 2^3$ $1.01 \times 2^{-1+4} + 1.11 \times 2^{3+4}$

补码： $111 < 011?$ 移码： $011 < 111$

 (-1) (3) (3) (7)

 简化比较

- 为何尾数使用原码表示：

- 补码可以连带符号位加减的好处是有“操作数宽度相同”这一条件的，浮点数尾数需要加减的时候宽度不一定相同
- 原码正负范围相同，使得对于任意可表示的 x ，其相反数 $-x$ 也可表示

IEEE754的精妙之处

- 为什么偏移量为 $2^{m-1} - 1$ ，而不是 2^{m-1}
 - 阶码全为0（即0）和阶码全为1（即 $2^m - 1$ ）这两个数保留做特殊用途
 - 去除保留的两个数，阶码范围为 $[1, 2^m - 2]$ ，中位数为 $2^{m-1} - 1$ 和 2^{m-1}
 - 偏移值取 $2^{m-1} - 1$ ，相比于 2^{m-1} ，所能表示的浮点数绝对值范围在指数层面更对称

例：对于float类型， $m=8$

取偏移值128时，阶码的真值范围 $[-127, 126]$ ，

所能表示的浮点数绝对值范围 $[5.877 \times 10^{-39}, 1.7 \times 10^{38}]$

取偏移值127时，阶码的真值范围 $[-126, 127]$ ，

所能表示的浮点数绝对值范围 $[1.175 \times 10^{-38}, 3.4 \times 10^{38}]$

相对而言，取偏移值127时，所能表示的浮点数绝对值范围在指数层面更对称

IEEE754的精妙之处

■ 隐藏位技术

- 规格化后，原码非0值浮点数的尾数数值最高位必定为1
- 在保存浮点数到内存前，通过尾数左移，强行把该位去掉
- 用同样多的位数能多存一位二进制数，有利于提高数据表示精度
- 在取回这样的浮点数到运算器执行运算时，必须先恢复该隐藏位

一个规格化的32位浮点数 x 的真值为：

$$x = (-1)^s \times (1.M) \times 2^{E-127}$$

一个规格化的64位浮点数 x 的真值为：

$$x = (-1)^s \times (1.M) \times 2^{E-1023}$$

2.2.3 实数类型

■ 例1：若浮点数 x 的二进制存储格式为 $(41360000)_H$ ，求其32位浮点数的十进制值

解： 0100,0001,0011,0110,0000,0000,0000,0000

数符：0

阶码：100,0001,0

尾数：011,0110,0000,0000,0000,0000

指数 $e = \text{阶码} - 127 = (10000010)_B - (01111111)_B = (00000011)_B = (3)_D$

包括隐藏位1的尾数： $1.M = (1.011\ 0110\ 0000\ 0000\ 0000\ 0000)_B = (1.011011)_B$

于是有 $x = (-1)^s \times 1.M \times 2^e$

$$= + (1.011011)_B \times 2^3 = (1011.011)_B = (11.375)_D$$

2.2.3 实数类型

■ 例2：将十进制数20.59375转换成32位浮点数的二进制格式存储

解：首先分别将整数和分数部分转换成二进制数：

$$(20.59375)_D = (10100.10011)_B$$

然后移动小数点，使其在第1，2位之间：

$$10100.10011 = 1.010010011 \times 2^4$$

于是得到：

$$S = 0, \quad E = e + 127 = 4 + 127 = 131 = 10000011, \quad M = 010010011$$

最后得到32位浮点数的二进制存储格式为：

$$0100,0001,1010,0100,1100,0000,0000,0000 = (41A4C000)_{16}$$

2.2.3 实数类型

■ 例3：将十进制数-0.75表示成单精度的IEEE754标准代码

解： $(-0.75)_D = (-3/4)_D = (-0.11)_B = -1.1 \times 2^{-1}$

$$= (-1)^1 \times (1 + 0.1000\ 0000\ 0000\ 0000\ 0000\ 000)_B \times 2^{-1}$$

$$= (-1)^1 \times (1 + 0.1000\ 0000\ 0000\ 0000\ 0000\ 000)_B \times 2^{126-127}$$

于是有： $S=1$, $E=(126)_D = (01111110)_B$, $M=1000 \dots 000$

1 011, 1111, 0 100, 0000, 0000, 0000, 0000, 0000

(B F 4 0 0 0 0 0)_H

思考：能否使用下面几种方式查看float型数据的内存表示？

2.2.3 实数类型

■ 如何查看float型数据在内存中的表示

①

```
float f = -0.75;  
unsigned int ff = f;  
printf("%x\n", ff);
```



②

```
float f = -0.75;  
unsigned int *ff = (unsigned int *)&f;  
printf("%x\n", *ff);
```



```
1. #include <stdio.h>  
  
2. int main(int argc, char **argv) {  
3.     float f = -0.75;  
4.     unsigned char *p;  
  
5.     p = (unsigned char*)(&f);  
6.     printf("%p 0xx\n", p, *p);  
  
7.     p++;  
8.     printf("%p 0xx\n", p, *p);  
  
9.     p++;  
10.    printf("%p 0xx\n", p, *p);  
  
11.    p++;  
12.    printf("%p 0xx\n", p, *p);  
13. }
```

```
00000000000061FE1C 0x0  
00000000000061FE1D 0x0  
00000000000061FE1E 0x40  
00000000000061FE1F 0xbf
```

说明：左侧代码查看float型数据在内存中的表示

- 使用指针p，依次指向变量f的每一个字节
- 打印指针p指向的地址以及该地址的内容（单字节）
- 浮点型数据被按照IEEE754标准进行了编码
- 可以看到在Intel 80x86平台上，使用的是小端方式

思考：如何看待“YOJ-07求三角形面积”，部分同学的输出结果是nan或者inf？

2.2.3 实数类型

■ float类型的特殊值

- 正零 : 0 00000000 00000000 00000000 00000000
- 负零 : 1 00000000 00000000 00000000 00000000
- 正无穷 : 0 11111111 00000000 00000000 00000000
- 负无穷 : 1 11111111 00000000 00000000 00000000
- 不合法数 : * 11111111 (23位小数部分不全为0)

inf: infinity
nan: not a number
ind:
indeterminate

```
1. #include <stdio.h>
2. int main(int argc, char **argv) {
3.     float a = 1.0 / 0.0;
4.     float b = 1.0 / (-0.0);
5.     float c = 0.0 / 0.0;
6.     unsigned int *aa = (unsigned int *)(&a);
7.     unsigned int *bb = (unsigned int *)(&b);
8.     unsigned int *cc = (unsigned int *)(&c);
9.     printf("%f %x\n", a, *aa);
10.    printf("%f %x\n", b, *bb);
11.    printf("%f %x\n", c, *cc);
12.}
```

Ubuntu上的输出：

```
inf 7f800000
-inf ff800000
-nan ffc00000
```

Windows上的输出：

```
1.#INF00 7f800000
-1.#INF00 ff800000
-1.#IND00 ffc00000
```

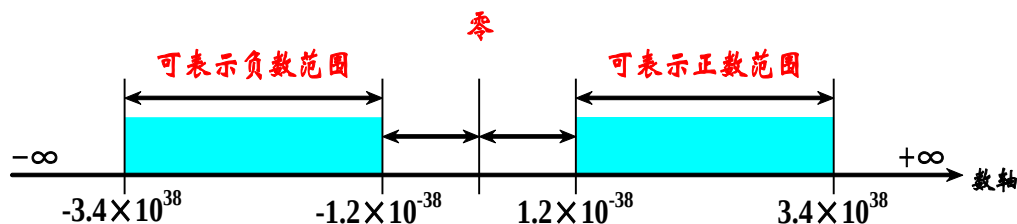
2.2.3 实数类型

float类型的范围

- 负最小值: $1\ 11111110\ 11111111\ 11111111\ 11111111 = -(1.\text{ffffe})_{\text{H}} \times 2^{127} \approx -3.4 \times 10^{38}$
- 负最大值: $1\ 00000001\ 00000000\ 00000000\ 00000000 = -(1.0)_{\text{H}} \times 2^{-126} \approx -1.2 \times 10^{-38}$
- 正最小值: $0\ 00000001\ 00000000\ 00000000\ 00000000 = +(1.0)_{\text{H}} \times 2^{-126} \approx 1.2 \times 10^{-38}$
- 正最大值: $0\ 11111110\ 11111111\ 11111111\ 11111111 = + (1.\text{ffffe})_{\text{H}} \times 2^{127} \approx 3.4 \times 10^{38}$
- 提示: 阶码全0和全1保留做特殊值, 尾数还有一个隐藏位

最大正数: $\text{Max}(+) = S_{\text{max}} \times r^{j_{\text{max}}}$ $= 0.11\dots1 \times 2^{11\dots1}$ $= (1-2^{-n}) \times 2^{2^m-1}$	最小正数: $\text{Min}(+) = S_{\text{min}} \times r^{j_{-\text{max}}}$ $= 0.10\dots0 \times 2^{-11\dots1}$ $= (2^{-1}) \times 2^{-(2^m-1)}$
最小负数: $\text{Min}(-) = -S_{\text{max}} \times r^{j_{\text{max}}}$ $= -0.11\dots1 \times 2^{11\dots1}$ $= -(1-2^{-n}) \times 2^{2^m-1}$	最大负数: $\text{Max}(-) = -S_{\text{min}} \times r^{j_{-\text{max}}}$ $= -0.10\dots0 \times 2^{-11\dots1}$ $= -(2^{-1}) \times 2^{-(2^m-1)}$

规格化形式



思考：如何看待“YOJ-07求三角形面积”的测试点3？

输入文件：605 264 501

输出文件：65153.87

选手输出：65153.88

2.2.3 实数类型

■ float类型的有效数字（数值精确的数位）

- float类型有23个二进制位用于存放尾数，另有一个隐含位固定为1（无灵活度）
- $2^{23} = 8388608 \approx 0.84 \times 10^7$
- float类型最多能有7个十进制有效数字，但能绝对保证的是6个有效数字

```
1. #include <stdio.h>
2. int main(int argc, char **argv) {
3.     float a = 0.123456789;
4.     float b = 0.987654321;
5.     float c = 123456789;
6.     float d = 987654321;
7.     float e = 123.456789;
8.     float f = 98765.4321;
9.     printf("a=%.9f\n", a);
10.    printf("b=%.9f\n", b);
11.    printf("c=%.9f\n", c);
12.    printf("d=%.9f\n", d);
13.    printf("e=%.9f\n", e);
14.    printf("f=%.9f\n", f);
15.} //%.9f: 保留小数点后9位
```

```
a=0.123456791
b=0.987654328
c=123456792.000000000
d=987654336.000000000
e=123.456787109
f=98765.429687500
```

6位有效数字：包括小数点前后总共6位10进制数位，不包括小数点。

2.2.4 字符类型

- 占用1个字节字符类型的数据，在内存中以相应的ASCII码存放
- 例：'1'的ASCII码为(49)_D，内存中存储的是ASCII码值

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

字符'1'在内存中的存储

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

整数1在内存中的存储 (short类型, 小端序)

■ 注意：

- 字符'1'与整数1是不同的概念
 - 字符'1'代表一个形状为'1'的符号，在内存中以ASCII码形式存储，占1个字节
 - 整数1是一个数值，在内存中以二进制补码形式存储，占2/4/8个字节
 - $1 + 1 = 2$, $'1' + '1' \neq 2$
- char类型除了表示字符，还可以看作是一个特殊的整数类型
 - 1个字节的整数
 - char的范围[-128, 127], unsigned char的范围[0, 255]

2.3 常量和变量

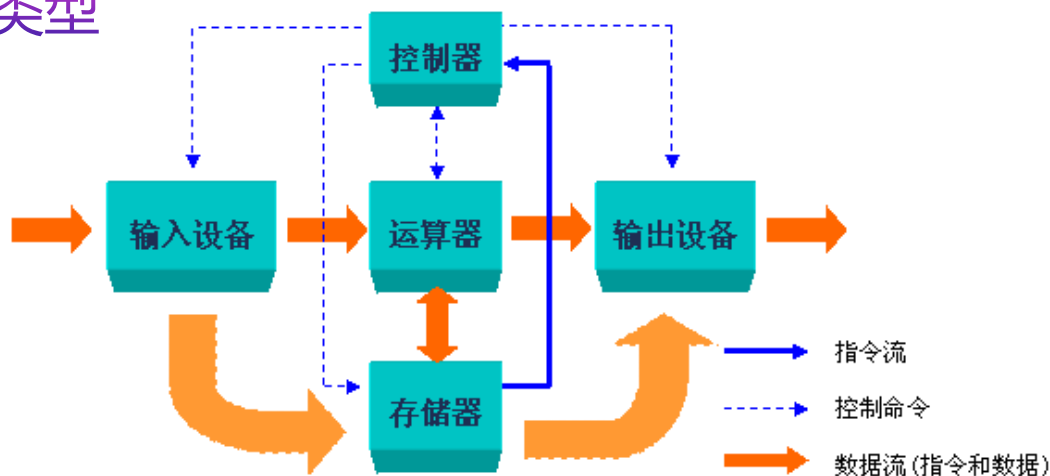
- 常量指直接给定、程序执行期间不能发生变化的数据。
- 变量的值可以在程序中改变，变量是命名的内存指定单元。
- 常量和变量均区分为不同类型

变量a

30

变量值

内存单元地址



在冯·诺依曼体系结构中，数据和代码都存放于存储器

建立起变量与变量地址的概念：提到变量就想到有一个地址与之联系
常量直接出现在代码中，占用代码存储空间，不占用数据存储空间
变量占用数据存储空间，不占用代码存储空间

2.3.1 不同类型的常量

■ 字面量常量与符号常量

■ 字面常量：从字面形式上即可识别的常量

■ 符号常量：用一个标识符来表示一个常量

```
1. #include <stdio.h>           //引用头文件
2. #define SUM_INIT 0           //定义符号常量SUM_INIT

3. int z = SUM_INIT;            /* 定义全局变量z，并进行初始化 */

4. int main(int argc, char **argv) {
5.     int x = 1;                /* 定义局部变量x，并进行初始化 */
6.     int i, n;                 /* 定义局部变量i和n，未进行初始化 */

7.     scanf("%d", &n);          //输入
8.     for (i = 1; i <= n; i++) { //L8-11行：循环语句
9.         x = x * i;
10.        z = z + x;
11.    }
12.    printf("%d\n", z);         //输出
13.    return 0;                 //函数返回值
14.}
```

2.3.1 不同类型的常量

■ 整型字面常量

- 默认：一个整型字面常量，默认是十进制表示的 int 类型（若范围允许）
- 前缀：改变当前字面常量的进制（0x/0X: 十六进制，0: 八进制）
- 后缀：改变当前字面常量的类型（U/u: unsigned, L/l: long, LL/ll: long long）

```
1. #include <stdio.h>
2. int main(int argc, char **argv) { /* sizeof(x): 获得x所占用的内存字节数 */
3.     printf("%d %d %d\n", 12, 012, 0x12);
4.     printf("%d %d %d %d\n", sizeof(0x12), sizeof(0x12U), sizeof(0x12L), sizeof(0x12ULL));
5.     printf("%d %d %d\n", sizeof(2147483647), sizeof(2147483648), sizeof(2147483648U));
6.     printf("%d %d\n", sizeof(0x80000000), sizeof(0x80000000U));
7. }
```

思考：上面代码的输出是什么？

答案（在64位Ubuntu环境下）：

12 10 18

4 4 8 8

4 8 4

4 4

2.3.1 不同类型的常量

■ 实型字面量常量

- 十进制小数形式: 123.456, 0.345, -56.79, 0.0, 12.0,
- 指数形式: $12.34e3(12.34 \times 10^3)$, $-346.87e-25(-346.87 \times 10^{-25})$,
- 默认: 一个实型字面常量, 默认是double 类型 (若范围允许)
- 后缀: 改变当前字面常量的类型 (F/f: float, L/l: long double)
- C99标准新增的表示方式: 以0x开头, 然后是16进制尾数部分, 接着是p, 后面是以2为底的阶码。例: $0xa.1fp10 = \left(10 + \frac{1}{16} + \frac{15}{256}\right) \times 2^{10} = 10364$

答案 (在64位Ubuntu环境下) :

10364.000000□10364.000000□10364.000000

4621f000□4621f000□4621f000

4□8□16

8□4□16

```
1. #include <stdio.h>
2. int main(int argc, char **argv) {
3.     float a = 10364, b = 10.364e3, c = 0xa.1fp10;
4.     printf("%f %f %f\n", a, b, c);
5.     printf("%x %x %x\n",*(unsigned int *)(&a), *(unsigned int *)(&b),*(unsigned int *)(&c));
6.     printf("%d %d %d\n", sizeof(float), sizeof(double), sizeof(long double));
7.     printf("%d %d %d\n", sizeof(12.34), sizeof(12.34F), sizeof(12.34L));
8. }
```

思考: 上面代码的输出是什么?

2.3.1 不同类型的常量

思考：字符字面常量占用的内存字节数是多少？`sizeof('a') = ?`

■ 字符字面常量

- 普通字符：用单引号括起来的字符，如：'a', 'Z', '3', '?',
- 转义字符：用"\"开头的字符序列表示特殊的字符
- 常用的特殊字符：
 - 引号：\'，\"
 - 制表符：\t，\v
 - 回车换行：\n，\r
 - 反斜杆：\\
- 使用转义字符表示
 - \o或者\oo或者\ooo：与该八进制码对应的字符（最多3位八进制数）
 - \xh或\xhh：与该十六进制码对应的字符（最多2位十六进制数）
- 例：字符'1'的多种表示方式——'1', '\061', '\x31'

2.3.1 不同类型的常量

思考：字符串字面常量占用的内存字节数是多少？
`sizeof("hello") = ?` `strlen("hello") = ?`

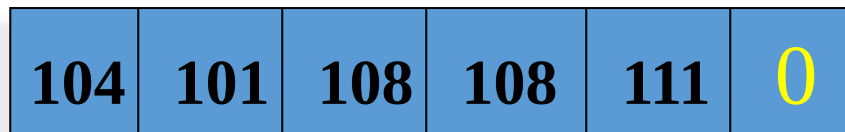
■ 字符串字面常量

- 用双引号括起来的字符序列，如：“hello”
- 字符串常量是双引号中的全部内容，但不包括双引号本身
- 字符串中字符个数称为该字符串的长度
- 字符串常量存储在机器中时，系统自动在其末尾加一个结束标志‘\0’

字符串“hello”存储为：



实际上每个字符都以ASCII码存储：



2.3.2 变量的命名规则

- 变量名可包含字母、数字和下划线，但只能以字母或下划线开头
- C++语言的关键字不能作为变量名
- 变量名是大小写敏感的
- 变量在一个函数范围内不能重名
- 练习：指出下面变量命名的正确性
 - `int money$owed;`
 - `int total_count;`
 - `int score2;`
 - `int 2ndscore;`
 - `int long;`
 - `int x, X;` 定义了几个变量？

2.3.3 变量赋值

■ 赋值表达式：<变量> = <表达式>

■ 例1: `a=1988;` // 读作将表达式的值1988赋给变量a

■ 例2: `C=sin(PI/4);` // 读作将表达式 $\pi/4$ 的正弦函数值赋给变量C

■ 变量赋值的关键要素

■ 变量必须先定义再使用

■ 在变量定义时可对其赋初值，这叫变量初始化，是一种良好的编程习惯

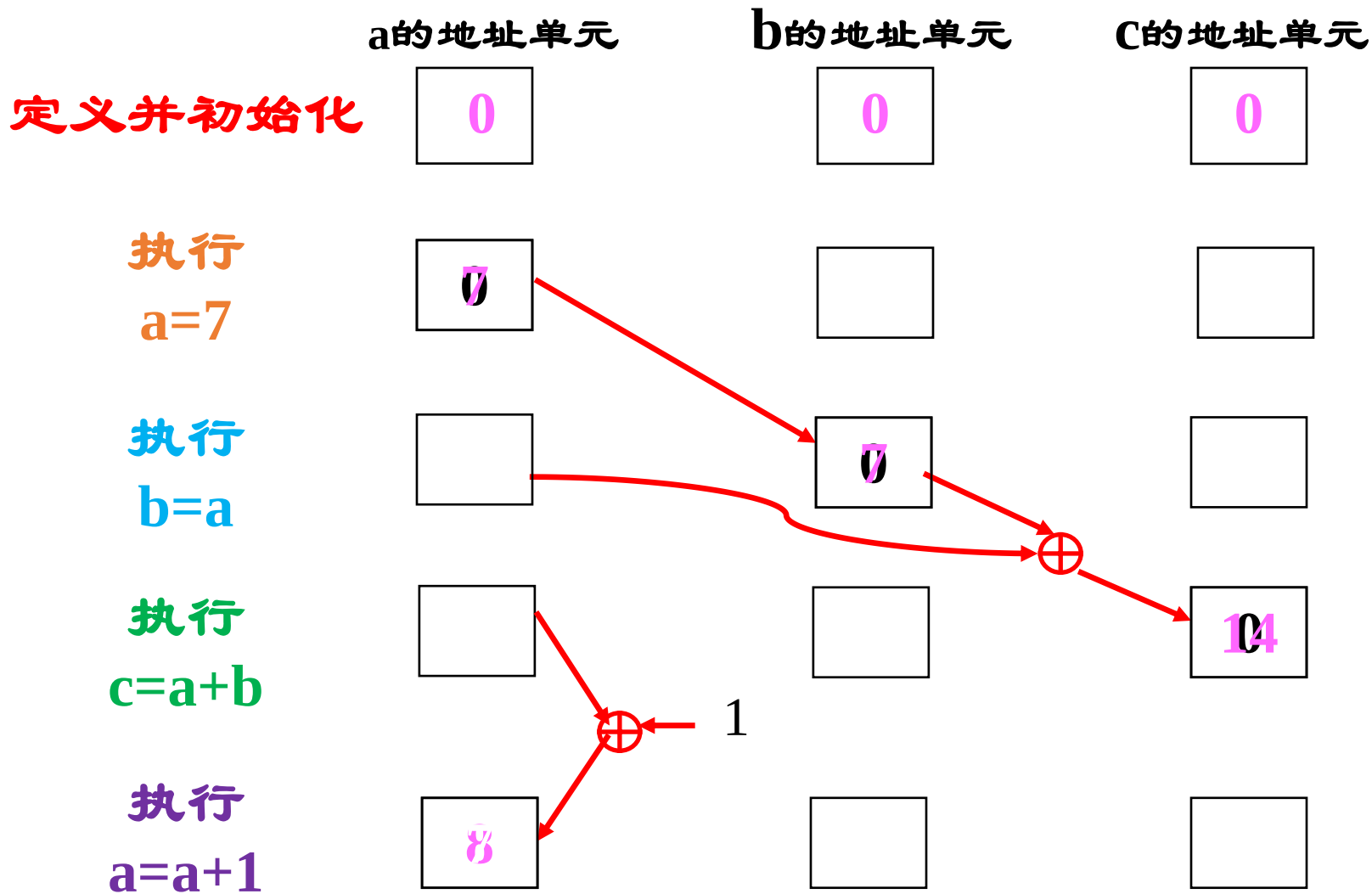
■ 对变量的赋值过程是“覆盖”写入，在变量地址单元中用新值去替换旧值

■ 读取变量的值，该变量保持不变，相当于拷贝一份出来

■ 参与表达式运算的所有变量都保持原来的值不变

2.3.3 变量赋值

```
1. int main() {  
2.     int a = 0, b = 0, c = 0;  
3.     a = 7; b = a; c = a + b; a = a + 1;  
4.     printf("%d %d %d\n", a, b, c);  
5. }
```



2.4 运算符和表达式

- 运算符：在C++语言中用来表示某种计算的符号
- 操作数：运算符操作的对象叫操作数，可以为变量（已赋值）、常量或其它有确切值的表达式
- 表达式：由运算符连接常量、变量、函数所组成的式子。每个表达式都有一个值和类型

2.4 运算符和表达式

■ 运算符涉及的操作数个数

- 单目运算符（一元运算符）：只有1个操作数。
- 双目运算符（二元运算符）：具有2个操作数。（主要类型）
- 三目运算符（三元运算符）：需要3个操作数。（唯一：条件运算符 `?:`）

■ 运算符的优先级

- 某些运算符先于其他运算符被执行
 - 例如， $x + y * 4$ ，先乘除后加减。
- 必要时可用圆括号 `()` 改变计算顺序
 - 例如，求三个数的平均值。
 - 错误的写法： $a + b + c / 3$
 - 正确的写法： $(a + b + c) / 3$

■ 运算符的结合性：

- 出现并列的优先级别相同的运算符时，由运算符的结合性决定计算的次序
- 常见的算术运算符左结合，赋值运算符右结合
 - 例： $x * y / z$ 相当于 $(x * y) / z$
 - 例： $x = y = z + 1$ 相当于 $x = (y = z + 1)$

C/C++语言运算符概览

算术运算符:	+ - * / % ++ --
赋值运算符:	= 及其扩展
求字节数 :	sizeof
强制类型转换:	(类型)
逗号运算符:	,
函数调用符运算符:	()
关系运算符:	< <= == > >= !=
逻辑运算符:	! &&
条件运算符:	?:
下标运算符:	[]
分量运算符:	. ->
位运算符 :	<< >> ~ & ^
指针运算符:	* &

思考：(-7)/(unsigned)(4)的结果？

2.4.1 算术运算符

■ 常见的算术运算符

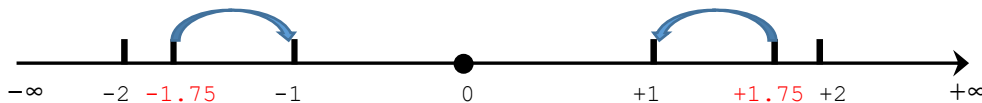
运算符名称	算术运算符	代数表达式	C++语言表达式	适用的数据类型
正号	+	+ a	+ a	整数、字符、浮点数
负号	-	- b	- b	整数、字符、浮点数
加	+	f + 7	f + 7	整数、字符、浮点数
减	-	p - c	p - c	整数、字符、浮点数
乘	*	b m	b * m	整数、字符、浮点数
除	/	x / y	x / y	整数、字符、浮点数
取模	%	r mod s	r % s	整数、字符

■ 正负号是单目运算符，其余是双目运算符

■ 优先级：

- 第1优先级：正号、负号
- 第2优先级：乘、除、取模
- 第3优先级：加、减

例： $7 \div 4 = 1.75$ $(-7) \div 4 = -1.75$ $7 \div (-4) = -1.75$ $(-7) \div (-4) = 1.75$



■ 结合性：左结合

$$\begin{array}{llll} 7/4 = 1 & (-7)/4 = -1 & 7/(-4) = -1 & (-7)/(-4) = 1 \\ 7\%4 = 3 & (-7)\%4 = -3 & 7\%(-4) = 3 & (-7)\%(-4) = -3 \end{array}$$

■ 除号运算符的使用问题：

- 两个操作数全为整型数（包括char、short、int、long、long long）时，为整除运算
- 当有任一操作数为实型数时，为普通除法运算
- C99标准规定整数除法采用“趋零截尾”，向0方向取最接近精确值的整数，换言之就是舍去小数部分

■ 取模运算符的使用问题：

- 操作数只能整型数据（包括char、short、int、long、long long）
- C++语言中的取模运算： $x \% y = x - x/y*y$

2.4.1 算术运算符

■ 自增和自减运算符

- 自增运算符：`++`，将操作数的值增1
- 自减运算符：`--`，将操作数的值减1
- 操作数必须是整型和字符型变量
- 单目运算符
- 优先级高于常见的算术运算符
- 结合性：右结合
- 使用时的注意事项：
 - `++` (`--`) 在前：先加（减）后用
 - `++` (`--`) 在后：先用后加（减）

2.4.1 算术运算符

■ 练习：下面代码的输出是什么？

```
1. #include <stdio.h>
2. int main() {
3.     int i=6, a, b;

4.     printf("%d ", ++i);
5.     printf("%d ", i++);

6.     a=--i; printf("%d ", a);
7.     b=i--; printf("%d\n", b);

8.     printf("%d ", -i++);
9.     printf("%d ", --++i);
10.    printf("%d\n", i);

11.    i = 8;
12.    printf("%d ", ++i + i++);
13.    printf("%d\n", i);

14.    i = 8;
15.    printf("%d %d\n", i, i++ + i++);
16. }
```

答案（GCC编译器）：

7 7 7 7

-6 -8 -8

19 10

10 17

答案（Visual Studio 2019编译器）：

7 7 7 7

-6 -8 -8

18 10

10 16

注意：C标准规定了自增/自减运算符的增/减操作在一个完整的表达式计算后完成，但没有规定在哪个子表达式计算后进行。不同的编译器有不同的执行结果。

原则：

- 当一个变量多次出现在一个表达式里时，不要对其进行自增/自减操作
- 若一个变量出现在同一个函数调用的多个参数中，不要对其进行自增/自减操作

2.4.2 赋值运算符

■ 赋值运算符

- 简单赋值运算符：=
- 复合赋值运算符：+=, -=, *=, /=, %=
- 优先级：低于算术运算符
- 结合性：右结合

■ 赋值表达式

<变量> <赋值运算符> <表达式>

d = 23

- 作用：将表达式的值赋给变量
- 左值和右值：
 - 左值是可寻址的变量，可以出现在等号左边或者右边
 - 右值是不可寻址的常量，或在表达式求值过程中创建的临时量，只能出现在等号右边
 - 例：x = y + 1;
 - x做左值，代表的是存储变量x的内存单元（地址）
 - y做右值，代表的是存储变量y的内存单元的值
- 赋值表达式的值就是被赋值的变量的值

2.4.2 赋值运算符

■ 简单赋值运算符：=

举例

```
c=a+b
```

```
a=b=c=d=10
```

```
x=(a=5)+(b=8)
```



```
c=(a+b)
```

```
a=(b=(c=(d=10)))
```

```
a=5, b=8, x=a+b
```

■ 复合赋值运算符：+=, -=, *=, /=, %=

■ 简化了赋值表达式

<variable> <operator> = <expression>

- 由下面的表达式简化而来

<variable> = <variable> <operator> <expression>

举例

```
a+=5
```

```
x*=y+7
```

```
x+=x-=x*=x
```



```
a=a+5
```

```
x=x*(y+7)
```

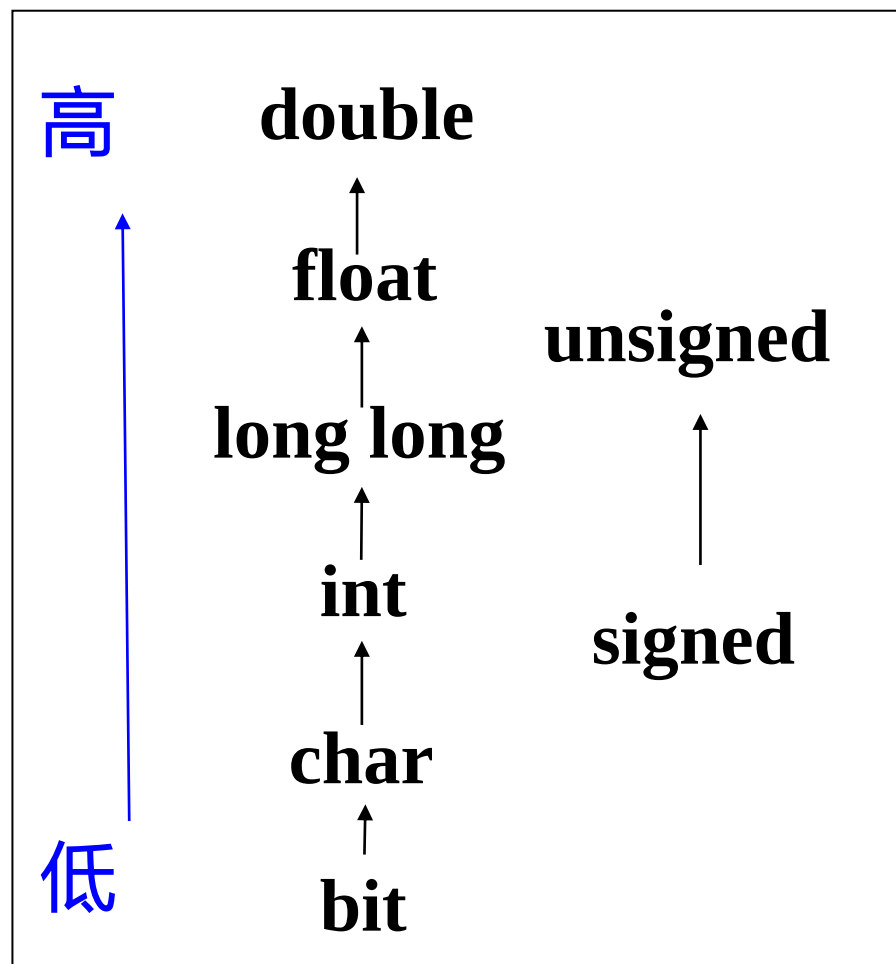
```
x=x+(x=x-(x=x*x))
```

2.4.3 类型转换

- 隐式类型转换：处理不同类型的数据时，计算机自动进行转换
 - 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
- 显式类型转换：程序员在代码中使用强制类型转换运算符
 - 强制类型转换运算：**(类型标识符) 表达式**
 - 强制类型转换是一个单目运算符，各种数据类型都可用于显式转换运行符
 - 例：

```
(char) ( 3 - 3.14159 * x )  
k=(int) ((int)x + (float)i + j)  
(float) (x = 99)  
(double) (5 % 3)
```
 - 注意：
 - 表达式应该用括号括起来：(int) x + y只将x转换成整型，然后与y相加
 - 对一个变量进行显式转换后，得到一个所需类型的中间变量，原来变量类型不变

数据类型的级别

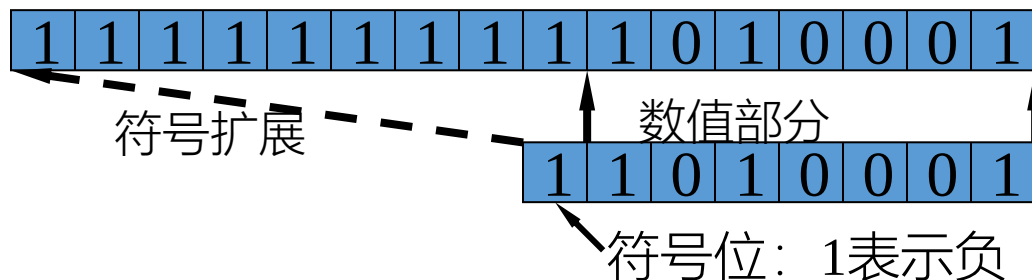


- 由低级别的数据类型转换为高级别的数据类型，称为**类型提升**（扩展）
- 由高级别的数据类型转换为低级别的数据类型，称为**类型下降**（截断）
- 同一长度的数据带符号与不带符号的属于相同级别，二者之间可转换

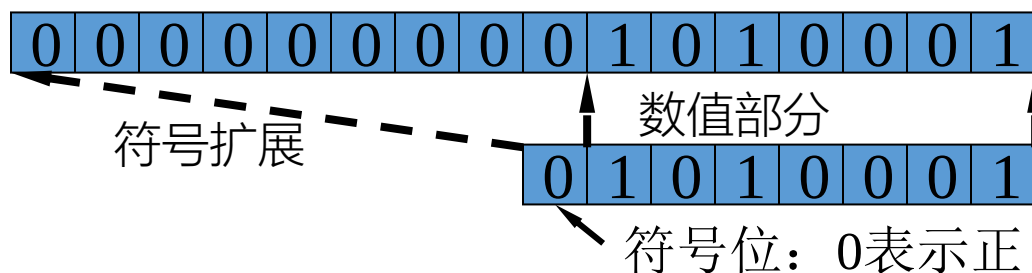
整数的扩展：符号扩展与零扩展

- 将signed型的整型数提升为较长signed型时，增加的位的状态与原来较短的数据中的符号位相同，称为**符号扩展**
- 将unsigned型扩展为较长的整型数据时，增加的位全部置0，称为**零扩展**
- 符号扩展和零扩展的目的都是为了**保证数值不变**

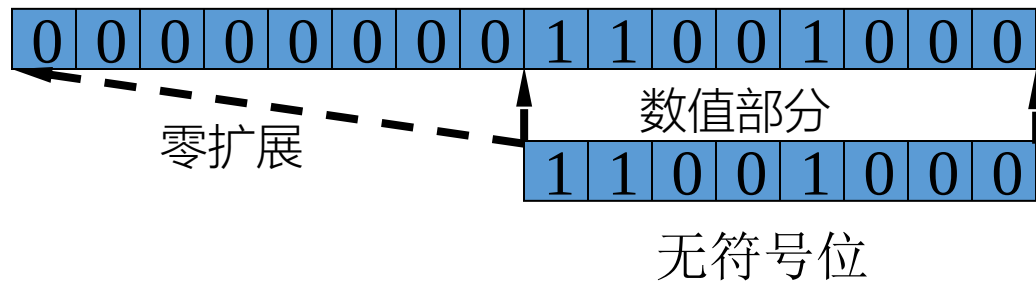
整数的扩展：符号扩展与零扩展



signed型



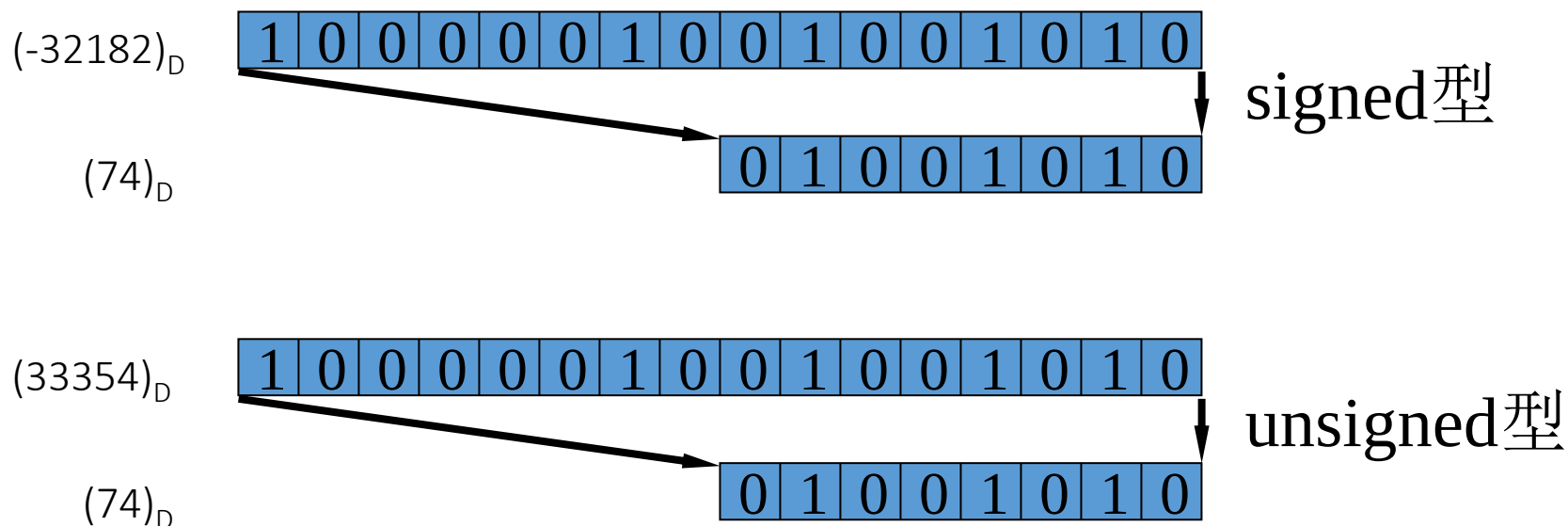
signed型



unsigned型

整数的截断

- 当较长的整数转换为较短的整数时，要将高位截去只将低位字节送过去，这会产生很大的误差



无符号整数与有符号整数的转换

- 一个无符号整型的数值范围绝对值比有符号整型大一倍。如 short 型变量占 2 个字节，其取值范围为 $[-32768, 32767]$ ，而 unsigned short 型变量数的范围为 $[0, 65535]$
- 由 signed 型转换为同一长度的 unsigned 型时，原来的符号位不再作为符号位，而成为数据的一部分
 - 正数的符号位是“0”，转换为 unsigned 时，数值不变
 - 负数的符号位是“1”，转换为 unsigned 时，数值发生变化
- 由 unsigned 型转换为同一长度的 signed 型时，原数据的最高位被作为符号位使用

无符号整数：19

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

有符号整数的补码形式：19

无符号整数：147

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

有符号整数的补码形式：-109

整数的转换

```
1. #include <stdio.h>
2. int main() {
3.     char a = 'a';    //ASCII: 0x61 (97)
4.     char b = '0';    //ASCII: 0x30 (48)
5.     printf("%d %d %d %d\n", a+b, sizeof(a), sizeof(b), sizeof(a+b));
6.     printf("%d %d\n", (char)(a+b), sizeof((char)(a+b)));
7.
8.     short c = 32767;    //MAX_SHORT_INT
9.     short d = 10;
10.    printf("%d %d %d %d %d\n", c+d, (short)(c+d), sizeof(c), sizeof(d), sizeof(c+d));
11.    printf("%d %d\n", (short)(c+d), sizeof((short)(c+d)));
12.
12.    short e = -32182;
13.    char f = (char)e;
14.    unsigned short g = 33354;
15.    unsigned char h = (unsigned char)g;
16.    printf("%d %d %d %d\n", e, f, g, h);
17.
17.    char i = -109;
18.    unsigned char j = (unsigned char)i;
19.    unsigned char k = 147;
20.    char l = (char)k;
21.    printf("%d %d %d %d\n", i, j, k, l);
22.}
```

```
145 1 1 4
-111 1
32777 -32759 2 2 4
-32759 2
-32182 74 33354 74
-109 147 147 -109
```

浮点数的转换

■ float → double

- 阶码：float数值阶码的机器数-127 → 阶码的真值，
阶码的真值+1023 → double数值阶码的机器数
- 尾数：double数值的尾数前23位取至float数值的尾数，后29位补0

```
1. #include <stdio.h>
2. int main() {
3.     float f = 123.456789;
4.     double d = f;
5.     unsigned int *pf = (unsigned int *)(&f);
6.     unsigned long long *pd = (unsigned long long *)(&d);
7.     printf("float: %.9lf %x\n", f, *pf);
8.     printf("double: %.9lf %llx\n", d, *pd);
9. }
```

```
float: 123.456787109 42f6e9e0
double: 123.456787109 405edd3c00000000
```

$(42f6e9e0)_H = (0100,0010,1111,0110,1110,1001,1110,0000)_B$

$(405edd3c00000000)_H = (0100,0000,0101,1110,1101,1101,0011,1100,0000,0000,0000,0000,0000,0000,0000)_B$

浮点数的转换

■ double → float

- 如果double数值超出float的有效数字，则截取尾数的前23位
- 如果double数值超出float的表示范围，则转换后为±INF

```
1. #include <stdio.h>
2. int main() {
3.     double d1 = 0x1.ffffep128, d2 = -987654.321;
4.     float f1 = d1, f2 = d2;
5.     unsigned long long *pd1 = (unsigned long long *)(&d1);
6.     unsigned long long *pd2 = (unsigned long long *)(&d2);
7.     unsigned int *pf1 = (unsigned int *)(&f1);
8.     unsigned int *pf2 = (unsigned int *)(&f2);
9.     printf("double: %.9lf %llx\n", d1, *pd1);
10.    printf("float: %.9lf %x\n", f1, *pf1);
11.    printf("double: %.9lf %llx\n", d2, *pd2);
12.    printf("float: %.9lf %x\n", f2, *pf2);
13. }
```

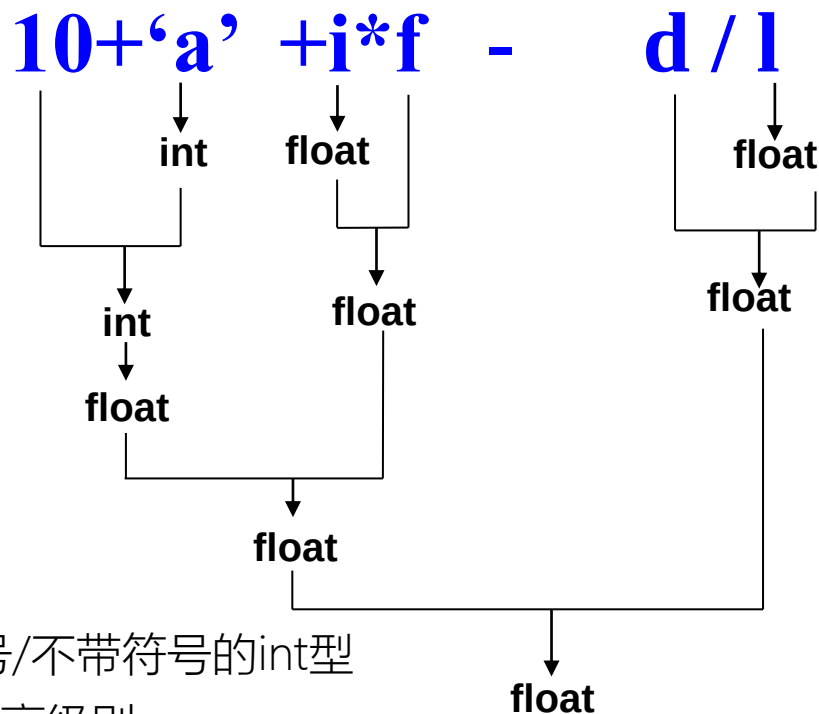
double: 6805640848047696100000000000000000000000.00000000 47fffffe00000000
float: 1.#INF00000 7f800000
double: -987654.321000000 c12e240ca45a1cac
float: -987654.312500000 c9712065

$(c12e240ca45a1cac)_H = (1100,0001,0010,1110,0010,0100,0000,1100,1010,0100,0101,1010,0010,0000,0000,0000)_B$

$(c9712065)_H = (1100,1001,0111,0001,0010,0000,0110,0101)_B$

运算转换

例 `int i;`
`float d,f;`
`long l;`



■ 转换目的:

- 将短的数扩展成机器处理的长度
- 使运算符两端具有共同的类型

■ 转换准则:

- `char`或`short`全部自动转换为相应的带符号/不带符号的`int`型
- 同样整型（带符号或无符号），低级别转高级别
- 若无符号操作数级别高于或等于另一操作数，则另一带符号操作数转为无符号
- 如果带符号操作数能容纳无符号操作数的所有可能值，则无符号转为带符号

```
1. #include <stdio.h>
2. #include <math.h>
3. int main(){
4.     unsigned int a = pow(2,32) - 2;
5.     int b=-1;
6.     long long c=-1;
7.     if (a>b) printf("a is greater than b\n"); else if (a<b) printf("a is less than b\n");
8.     if (a>c) printf("a is greater than c\n"); else if (a<c) printf("a is less than c\n");
9.     return 0;
10. }
```

a is less than b
a is greater than c

赋值转换

- 通过赋值使“=”右边表达式的类型自动转换为左边变量的类型
- 赋值转换具有强制性，可能是类型提升，也可能是类型下降

① **实型 ➡ 整型**：舍弃小数，超出整型表示范围，则取整型的最大值。

```
例： int i = 375.986;           i=375  
      int j = 2147483648.123;    j=2147483647  
      printf("i=%d", i);  
      printf("j=%d", j);
```

② **整型 ➡ 实型**：数值不变，但以浮点形式存到变量中。

```
例： float f;  
      f=36;                     f=36.000000
```

赋值转换

③字符型 ➡ 整型：将字符ASCII码值放到整型变量低八位，高八位为0。

④整型 ➡ 字符型：只把低八位赋予字符变量。

⑤signed ➡ unsigned：原样照赋，只是最高位的解读不同。

```
1. #include <stdio.h>
2. int main() {
3.     int a,b=322,i;
4.     float x,y=8.88;
5.     char c1='k',c2;
6.     a=y; x=b; i=c1; c2=b;
7.     printf("%d,%f,%d,%c",a,x,i,c2);
8. }
```

8,322.000000,107,B

```
1. #include <stdio.h>
2. int main() {
3.     unsigned int a;
4.     int b = -1;
5.     a = b;
6.     printf("%u",a);
7. }
```

4294967295

练习

例: `float f;`
`f=6/7;`
问: `f=?`

`f=0.8571` ✕

`f=0.0` ✓

改为: `f=6.0/7` 或 `f=6/7.0` 或 `f=(float)6/7`

2.4.4 逗号运算符

- 逗号运算符 “,” 将两个及其以上的式子联接起来
- 从左往右逐个计算表达式，整个表达式的值为最后一个表达式的值
- 逗号运算符的优先级别在所有运算符中最低

例1：逗号表达式(3+5, 6+8)。求解过程分解如下：

- ①. 先求解表达式3+5，得到值8；
- ②. 再求解表达式6+8，得到值14；
- ③. 整个表达式的值为14。

例2：求解表达式a=(a=3*5,a*4)。求解过程分解如下：

- ①. tmp1=(a=3*5) tmp1: 15, a: 15
- ②. tmp2=a*4 tmp2: 60, a: 15
- ③. a=(tmp1, tmp2) a: 60, tmp1: 15, tmp2: 60

思考：表达式(a=a=3*5,a*4)的值是多少，表达式计算完成后a的值是多少？

2.5 数学函数

■ 标准库函数：C++ 编译系统为方便用户使用而提供的公共函数

- 输入/输出函数
- 数学函数
- 字符串函数
-

■ 标准库参考手册

C++ 标准库 `<cmath>`

C++ 标准库提供了丰富的功能，其中 `<cmath>` 是一个包含数学函数的头文件，它提供了许多基本的数学运算和常数。

`<cmath>` 是 C++ 标准库中的一个头文件，它定义了一组数学函数，这些函数可以执行基本的数学运算，如幂运算、三角函数、对数、绝对值等。

要使用 `<cmath>` 中的函数，你需要在你的 C++ 程序中包含这个头文件：

```
#include <cmath>
```

序号	函数 & 描述
1	<code>double acos(double x)</code> 返回以弧度表示的 x 的反余弦。

学习更多： <https://www.runoob.com/cplusplus/cpp-libs-cmath.html>

2.5 数学函数

- 数学函数：

- 求绝对值函数
- 指数和对数函数
- 三角函数
- 取整函数
-

注：需引用头文件`#include <cmath>`，编译时链接相应的库

2.5.1 求绝对值函数

■ abs函数

- 函数原型: `int abs(int i);`
- 功能: 返回整数的绝对值。

■ fabs函数

- 函数原型: `double fabs(double x);`
- 功能: 返回浮点数的绝对值。

2.5.2 指数和对数函数

■ sqrt函数

- 函数原型: `double sqrt(double x);`
- 功能: 计算平方根, 返回x的平方根, x应大于等于0

■ exp 函数

- 函数原型: `double exp(double x);`
- 功能: 返回指数函数 e^x 的值。

■ pow 函数

- 函数原型: `double pow(double x, double y);`
- 功能: 返回指数函数(x的y次方)的值。

若x为负数且y为小数, 或者x为0且y小于等于0, 将出现结果错误。

■ log 函数

- 函数原型: `double log(double x);`
- 功 能: 返回自然对数函数 $\ln(x)$ (即 $\log_e x$) 的值。

■ log10函数

- 函数原型: `double log10(double x);`
- 功 能: 返回以10为底的对数函数 (即 $\log_{10} x$) 的值。

2.5.3 三角函数

■ sin函数

- 函数原型: `double sin(double x);`
- 功能: 正弦函数, 返回 x 的正弦 (即 $\sin(x)$) 的值, x 的单位为弧度。

■ asin函数

- 函数原型: `double asin(double x);`
- 功能: 反正弦函数, 返回 x 的反正弦 (即 $\sin^{-1}(x)$) 的值, x 应在-1到1范围内。

■ cos函数

- 函数原型: `double cos(double x);`
- 功能: 余弦函数, 返回 x 的余弦 (即 $\cos(x)$) 的值, x 的单位为弧度。

■ acos函数

- 函数原型: `double acos(double x);`
- 功能: 反余弦函数, 返回 x 的反余弦 (即 $\cos^{-1}(x)$) 的值, x 应在-1到1范围内。

■ tan函数

- 函数原型: `double tan(double x);`
- 功能: 正切函数, 返回 x 的正切 (即 $\tan(x)$) 的值, x 为弧度。

■ atan函数

- 函数原型: `double atan(double x);`
- 功能: 反正切函数, 返回 x 的反正切 (即 $\tan^{-1}(x)$) 的值。

2.5.4 取整函数

■ ceil函数

- 函数原型: `double ceil(double x);`
- 功能: 向上舍入, 返回不小于x的最小整数。

■ floor函数

- 函数原型: `double floor(double x);`
- 功能: 向下舍入, 返回不大于x的最大整数。

■ round函数

- 函数原型: `double round(double x);`
- 功能: 四舍五入, 返回x的四舍五入整数值。

练习：编程实现下列计算

1、 计算 $y = 1 + \frac{1}{1 + \frac{1}{1 + \frac{1}{5}}}$

2、 计算 $\sqrt{3^2 + 4^2}$

3、 计算 $\sqrt{\frac{1 - \cos(\pi / 3)}{2}}$

4、 计算 $y = 2 \sin^2(\pi / 4) + \sin(\pi / 4) \square \cos(\pi / 4) - \cos^2(\pi / 4)$

5、 计算 $y = \frac{2\sqrt{5}(\sqrt{6} + \sqrt{3})}{6 + 3}$

6、 计算 $y = \frac{\ln 5(\ln 3) - \ln 2}{\sin(\pi / 3)}$

2.6 语句

- 语句是构造程序的基本成分，程序是一系列语句的集合
- C/C++语言中，用一个分号标识语句结束
- 语句分类
 - 表达式语句
 - 声明/定义语句
 - 控制语句
 - 复合语句

2.6 语句

■ first.c涉及各种不同类型的语句

```
1. #include <stdio.h>
2. #define SUM_INIT 0

3. int z = SUM_INIT;                //声明定义语句

4. int main(int argc, char **argv) {
5.     int x = 1;                    //声明定义语句
6.     int i, n;                     //声明定义语句

7.     scanf("%d", &n);              //表达式语句（函数调用）
8.     for (i = 1; i <= n; i++)       //控制语句
9.     {
9.         x = x * i;                //表达式语句
10.        z = z + x;                 //表达式语句
11.    }                               } //复合语句
12.    printf("%d", z);               //表达式语句（函数调用）
13.    return 0;                      //控制语句
14.}
```

2.6 语句

■ first.cpp涉及各种不同类型的语句

```
1. #include <iostream>
2. #define SUM_INIT 0

3. int z = SUM_INIT;                //声明定义语句

4. int main(int argc, char **argv) {
5.     int x = 1;                    //声明定义语句
6.     int i, n;                    //声明定义语句

7.     std::cin >> n;                //表达式语句（函数调用）
8.     for (i = 1; i <= n; i++)        //控制语句
9.     {
9.         x = x * i;                //表达式语句
10.        z = z + x;                //表达式语句
11.    }                               } //复合语句
12.    std::cout << z;                //表达式语句（函数调用）
13.    return 0;                      //控制语句
14.}
```

2.6.1 表达式语句

- 表达式语句由一个表达式加一个分号构成，任何表达式都可以加上分号而成为语句
- 最典型的表达式语句是由赋值表达式构成的赋值表达式语句
 - 赋值表达式： $a = 3$
 - 赋值表达式语句： $a = 3;$
- 副作用：对程序变量或状态的修改
 - 无副作用的表达式语句： $i; x + y;$
 - 有副作用的表达式语句： $i++; z = x + y;$

2.6.2 声明/定义定义语句

- 声明/定义语句用于声明/定义函数或变量，可以全局的或局部的
- 函数声明语句：描述一个函数接口，包括函数名字、形参个数及类型、返回值类型

- 通知编译器，以便在调用该函数时进行检查
- 告诉程序设计人员，如何调用该函数

```
int scanf(const char *format, ...);  
int printf ( const char * format, ... );
```

} //在stdio.h中声明

- 变量定义语句：用于为变量分配存储空间，还可为变量指定初始值
 - `int x = 1;`
 - `int i, n;`

2.6.3 控制语句

■ 选择语句：

- `if () ... else ...`：两个分支
- `switch ()`：多个分支

■ 循环语句：

- `for () ...`
- `while () ...`
- `do ... while ()`

■ 改变执行流程语句：

- 结束本次循环语句：`continue`
- 终止执行switch或循环语句：`break`
- 转向语句：`goto`（慎用）
- 从函数返回语句：`return`

2.6.4 复合语句

- 用{}把一些语句和声明括起来成为复合语句（又称为语句块）
- 可以在复合语句中定义局部变量
- 例：

```
{  
    float pi=3.14159, r=2.5, area;           //定义局部变量  
    area = pi * r * r;  
    printf("area: %f", area);  
}
```

2.7 数据的输入输出

■ 输入输出是以计算机主机为主体而言的，C语言本身不提供输入输出语句

- 输入和输出操作是由C标准函数库中的函数来实现的
- 优点：简化编译系统简化，增强通用性和可移植性

■ 要在程序文件的开头用预处理指令#include把有关头文件放在本程序中

#include<stdio.h>

#include指令说明

三种形式：

#include "c:\cpp\include\myfile.h" ①

#include "myfile.h" ② ③

#include <myfile.h> ③

- ① 按指定路径查找文件
 - ② 源程序文件所在目录
 - ③ C编译系统指定的include目录
-

2.7.1 用printf函数输出数据

■ printf用来向终端（或系统隐含指定的输出设备）输出若干个任意类型的数据

printf（格式控制，输出表列）

(1) “**格式控制**”是用双引号括起来的一个字符串，称为格式控制字符串，简称格式字符串。包括：

- ① **格式声明**。格式声明由“%”和格式字符组成。作用是将输出的数据转换为指定的格式后输出。
- ② **普通字符**。普通字符即需要在输出时原样输出的字符。

(2) **输出表列**是程序需要输出的一些数据，可以是常量、变量或表达式。

```
printf("i=%d,c=%c\n", i, c)
```

普通字符

格式声明

i, c

格式控制

输出列表

printf的 格式声明

```
1. #include <stdio.h>
2. int main(int argc, char **argv) {
3.     char c = 'A';
4.     float f = 1234.567890;
5.     char *str = "Hello\0World";
6.     printf("%c %d %o %x\n", c, c, c, c);
7.     printf("%f %e %g\n", f, f, f);
8.     printf("%10d\n%-10d\n%010d\n", c, c, c);
9.     printf("%10.4f\n%10.8f\n%4.6f\n", f, f, f);
10.    printf("%s\n", str);
11. }
```

```
A 65 101 41
1234.567871 1.234568e+003 1234.57
65
65
0000000065
1234.5679
1234.56787109
1234.567871
Hello
```

% 附加字符 格式字符

- (1) printf函数输出时，务必注意输出对象的类型应与上述格式说明匹配，否则将会出现错误。
- (2) 除了X,E,G外，其他格式字符必须用小写字母，如%d不能写成%D。
- (3) 可以在printf函数中的格式控制字符串内包含转义字符，如\n,\t,\b,\r,\f和\377等。
- (4) 一个格式声明以“%”开头，以格式字符之一为结束，中间可以插入附加格式字符（也称修饰符）。
- (5) 如果想输出字符“%”，应该在“格式控制字符串”中用连续两个“%”表示，如：
printf("%f%%\n",1.0/3);

格式字符	说 明
d,i	以带符号十进制形式输出整数（正数不输出符号）
o	以八进制无符号形式输出整数（不输出前导符0）
x,X	以十六进制无符号形式输出整数（不输出前导符0x），用x则输出十六进制数的a~f时以小写形式输出，用X时，则以大写字母输出
u	以无符号十进制形式输出整数
c	以字符形式输出，只输出一个字符
s	输出字符串
f	以小数形式输出单、双精度数，隐含输出6位小数
e,E	以指数形式输出实数，用e时指数以“e”表示(如1.2e+02)，用E时指数以“E”表示(如1.2E+02)
g,G	选用%f或%e格式中输出宽度较短的一种格式，不输出无意义的0。用G时，若以指数形式输出，则指数以大写表示

附加字符	说 明
l	长整型整数，可加在格式符 d、o、x、u 前面)
m (代表一个正整数)	数据最小宽度
n (代表一个正整数)	对实数，表示输出 n 位小数；对字符串，表示截取的字符个数
-	输出的数字或字符在域内向左靠

2.7.2 用scanf函数输入数据

■ scanf用来从键盘读入数据

scanf（格式控制，地址表列）

(1) “**格式控制**”是用双引号括起来的一个字符串，含义同printf函数。包括：

① **格式声明**。以%开始，以一个格式字符结束，中间可以插入附加的字符。

② **普通字符**。

(2) **地址表列**是由若干个地址组成的表列，可以是变量的地址，或字符串的首地址。

```
scanf("a=%f,b=%f,c=%f", &a, &b, &c );
```

普通字符 格式声明

格式控制

地址列表

scanf的 格式声明

```
1. #include <stdio.h>
2. int main(int argc, char **argv) {
3.     char a, b, c, d, e[100], f;
4.     int g, h, i;
5.     scanf("%c%d%o%x", &a, &b, &c, &d);
6.     printf("%c%c%c%c\n", a, b, c, d);
7.     scanf("%3d%5d*%c%d", &g, &h, &i);
8.     printf("%d %d %d\n", g, h, i);
9.     scanf("%s", e);
10.    printf("%s", e);
11. }
```

输入：

A 65 101 41
12345678,90
Hello World

输出：

AAAA
123 45678 90
Hello

% 附加字符 格式字符

(1) scanf函数中的格式控制后面应当是**变量地址**，而不是变量名。

应与上述格式说明匹配，否则将会出现错误。

(2)如果在格式控制字符串中除了格式声明以外还有其他字符，则在输入数据时在对应的位置上应输入与这些字符相同的字符。

(3)在用“%c”格式声明输入字符时，空格字符和“转义字符”中的字符都作为有效字符输入。

(4) 在输入数值数据时，如输入空格、回车、Tab键或遇非法字符(不属于数值的字符)，认为该数据结束。

格式字符	说 明
d,i	输入 有符号十进制 整数
u	输入 无符号十进制 整数
o	输入 无符号八进制 整数
x,X	输入 无符号十六进制 整数(大小写作用相同)
c	输入单个字符
s	输入字符串，将字符串送到一个字符数组中，在输入时以非空白字符开始，以 第一个分隔符 (\n、\t、空格) 结束
f	输入实数，可以用小数形式或指数形式输入
e,E,g,G	与f作用相同，e与f、g可以互相替换(大小写作用相同)

附加字符	说 明
l	输入长整型数据（可用%ld，%lo，%lx，%lu）以及 double型数据 (用%lf或%le)
h	输入短整型数据（可用%hd，%ho，%hx）
域宽	指定输入数据所占宽度（列数），域宽应为正整数，注意： 输入浮点数时不能规定精度
*	本输入项在读入后不赋给相应的变量

思考：输出为什么是Hapy，而不是Hap？

2.7.3 字符输入输出函数

■ putchar函数：从计算机向显示器输出一个字符

- putchar(c) 等价于 printf("%c", c)
- 用putchar函数既可以输出可显示字符，也可以输出控制字符和转义字符
- putchar(c)中的c可以是字符常量、整型常量、字符变量或整型变量(其值在字符的ASCII代码范围内)

```
1. #include <stdio.h>

2. int main(int argc, char **argv) {
3.     char H = 'H', a = '\x61', y = '\171';
4.     int p = 0x70;
5.     putchar(H);
6.     putchar(a);
7.     putchar(p);
8.     putchar(p);
9.     putchar(8);
10.    putchar(y);
11.    putchar(8);
12.}
```

ASCII码0x8是控制字符Backspace（回退删除）

Hapy

2.7.3 字符输入输出函数

■ getchar函数：从键盘读入一个字符

- `c=getchar()` 等价于 `scanf("%c", &c)`
- `getchar`函数没有参数，函数的值就是从输入设备得到的字符
- `getchar`函数可以从输入设备获得一个**可显示字符**或者**控制字符**
- 用`getchar`函数得到的字符可以赋给一个字符变量或整型变量，也可作为表达式的一部分

```
1. #include <stdio.h>

2. int main(int argc, char **argv) {
3.     char a, b;
4.     a = getchar();
5.     b = getchar();
6.     putchar(a);
7.     putchar(b);
8.     putchar(getchar());
9. }
```

输入：BOY

输出：BOY

2.7.4 字符串输入输出函数

- **puts函数**：将一个字符串(以'\0'结束的字符序列)输出到终端
 - 用puts函数输出的字符串中可以包含转义字符。
 - 用puts输出时将字符串结束标志'\0'转换成'\n'，即输出完字符串后换行
 - puts(str)相当于printf("%s\n", str)
- **gets函数**：从终端输入一个字符串到字符数组
 - 该函数值是字符数组的起始地址
 - gets(str)不完全相当于scanf("%s\n", str)
 - 若从键盘输入My□Computer↵，将输入的字符串"My□Computer"送给字符数组str（11个字符外加1个'\0'），回车被消耗掉，输入缓冲区当前位置指向下一个字符

```
1. #include <stdio.h>
2. int main(int argc, char **argv) {
3.     char str1[]={ "Info\nTuring" };
4.     char str2[100];
5.     gets(str2);
6.     puts(str1);
7.     puts(str2);
8. }
```

输入：RUC

输出：
Info
Turing
RUC

2.7.5 格式化输入背后的原理

- 输入缓冲区：存储用户键盘输入的字符数据
 - 当用户敲击Enter键，数据进入输入缓冲区
 - 若输入缓冲区中的数据不够满足读入操作的需求，则继续等待用户输入
 - 输入缓冲区中的残存数据会影响下一次数据读入操作
- getchar：获得输入缓冲区的当前字符
- gets：从输入缓冲区读入字符直至遇到换行符

终端：100□□a↵

输入缓冲区：

\x31	\x30	\x30	\x20	\x20	\x61	\x0A
------	------	------	------	------	------	------

- 思考：如何看待“YOJ-09按要求输入输出”？
- 输入两个字母，每个字母前面加一个空格，中间加一个逗号，需要什么格式声明？
 - 是否有更简单的实现方式？

2.7.5 格式化输入背后的原理

■ scanf: 按照格式声明指定的格式从输入缓冲区中读取数据

■ 依次匹配格式声明和输入缓冲区

- %c: 获得输入缓冲区的当前字符，即便是一个分隔符
- %s/%d/%f: 跳过前导分隔符，读取输入缓冲区内容，直至遇到第一个分隔符结束，不消耗后导分隔符

■ 将输入缓冲区的字符数据转换成格式化字符串指定类型的数据

■ 若遇到格式声明中的分隔符（\n、\t、空格），则消耗缓冲区当前位置往后所有分隔符

■ 当输入缓冲区中的数据不满足格式声明要求时，立即终止本次数据读取

终端: 100 □ □ a ↵

输入缓冲区:

\x31\x30\x30\x20\x20\x61\x0A

格式声明: "%d □ %c"

a □ □ 100 ↵

\x61\x20\x20\x31\x30\x30\x0A

"%c □ %d"

2.7.5 格式化输入背后的原理

■ 以下程序在给定的输入下，输出什么？

```
int x; char y;  
scanf("%d%c", &x, &y);  
printf("%d %c", x, y);
```

input: 1000.1↵

output: 1000 .

```
char x, y, z;  
scanf("%c%c%c", &x, &y, &z);  
printf("%c%c%c", x, y, z);
```

input: a b c↵

output: a b c

```
char x[100]; char y;  
scanf("%s%c", &x, &y);  
printf("%s %d", x, y);
```

input: abcd↵

output: abcd10

```
unsigned int x, y;  
scanf("%d%d", &x, &y);  
printf("%d %d", x, y);
```

input: 12345678987654321↵
12↵

output:
1653732529 12

```
char x[100]; int y = 0;  
scanf("%s,%d", &x, &y);  
printf("%s %d", x, y);
```

input: 1234,1↵

output: 1234,1 0

input: 1234 ,1↵
output: 1234 0

修正: 
`scanf("%s %d", &x, &y);`

```
int x=0, y=0, z=0;  
scanf("%d,%d", &x, &y);  
scanf("%d", &z);  
printf("%d %d %d\n", x, y, z);
```

input: 1,2 3↵

output: 1 2 3

input: 1 2 3↵

output: 1 0 2

```
char x[100], y;  
gets(x);  
y = getchar();  
printf("%s %c", x, y);
```

input: abc d↵

e↵

output: abc d e

note:

(12345678987654321)₁₀

<==>10进制转2进制

(001010111101110001010100
011000101001000111110100
10110001)₂

=|=>截断

(011000101001000111110100
10110001)₂

<==>2进制转10进制

(1653732529)₁₀

7/20 单选题 (分值5.0分, 难度: 易)

已有如下定义和输入语句, 若要求a1, a2, c1, c2的值分别为10、20、A和B, 当从第一列开始输入数据时, 正确的数据输入方式是: (注: ■代表空格)

```
int a1,a2;char c1,c2;  
scanf("%d%c%d%c",&a1,&c1,&a2,&c2);
```

- ☐ A.10A■20B<回车>
- ☒ B.10■A■20■B<回车>
- ☐ C.10■A20B■<回车>
- ☐ D.10A20■B<回车>

作答保存成功

```
1  #include <stdio.h>  
2  int main()  
3  {  
4      int a1, a2;  
5      char c1, c2;  
6      scanf("%d%c%d%c", &a1, &c1, &a2, &c2);  
7      printf("%d %d %c %c", a1, a2, c1, c2);  
8      return 0;  
9  }
```

如果输入如下, 则输出会是什么

Input:10A□20B

Output:?

7/20 单选题 (分值5.0分, 难度: 易)

已有如下定义和输入语句, 若要求a1, a2, c1, c2的值分别为10、20、A和B, 当从第一列开始输入数据时, 正确的数据输入方式是: (注:  代表空格)

```
int a1,a2;char c1,c2;  
scanf("%d%c%d%c",&a1,&c1,&a2,&c2);
```

- ☐ A.10A  20B<回车>
- ☒ B.10  A  20  B<回车>
- ☐ C.10  A20B  <回车>
- ☐ D.10A20  B<回车>

作答保存成功

%d

输入缓冲区:

1	0	A		2	0	B	
---	---	---	---	---	---	---	---

```
1  #include <stdio.h>  
2  int main()  
3  {  
4      int a1, a2;  
5      char c1, c2;  
6      scanf("%d%c%d%c", &a1, &c1, &a2, &c2);  
7      printf("%d %d %c %c", a1, a2, c1, c2);  
8      return 0;  
9  }
```

如果输入如下, 则输出会是什么

Input:10A  20B

Output:?

7/20 单选题 (分值5.0分, 难度: 易)

已有如下定义和输入语句, 若要求a1, a2, c1, c2的值分别为10、20、A和B, 当从第一列开始输入数据时, 正确的数据输入方式是: (注: ■代表空格)

```
int a1,a2;char c1,c2;  
scanf("%d%c%d%c",&a1,&c1,&a2,&c2);
```

- ☐ A.10A■20B<回车>
- ☒ B.10■A■20■B<回车>
- ☐ C.10■A20B■<回车>
- ☐ D.10A20■B<回车>

作答保存成功

输入缓冲区:

%c

A	■	2	0	B	↵
---	---	---	---	---	---

```
1 #include <stdio.h>  
2 int main()  
3 {  
4     int a1, a2;  
5     char c1, c2;  
6     scanf("%d%c%d%c", &a1, &c1, &a2, &c2);  
7     printf("%d %d %c %c", a1, a2, c1, c2);  
8     return 0;  
9 }
```

如果输入如下, 则输出会是什么

Input:10A■20B

Output:?

7/20 单选题 (分值5.0分, 难度: 易)

已有如下定义和输入语句, 若要求a1, a2, c1, c2的值分别为10、20、A和B, 当从第一列开始输入数据时, 正确的数据输入方式是: (注:  代表空格)

```
int a1,a2;char c1,c2;  
scanf("%d%c%d%c",&a1,&c1,&a2,&c2);
```

- ☐ A.10A  20B<回车>
- ☒ B.10  A  20  B<回车>
- ☐ C.10  A20B  <回车>
- ☐ D.10A20  B<回车>

作答保存成功

输入缓冲区:

%d

	2	0	B	↵
---	---	---	---	---

```
1  #include <stdio.h>  
2  int main()  
3  {  
4      int a1, a2;  
5      char c1, c2;  
6      scanf("%d%c%d%c", &a1, &c1, &a2, &c2);  
7      printf("%d %d %c %c", a1, a2, c1, c2);  
8      return 0;  
9  }
```

如果输入如下, 则输出会是什么

Input:10A  20B

Output:?

7/20 单选题 (分值5.0分, 难度: 易)

已有如下定义和输入语句, 若要求a1, a2, c1, c2的值分别为10、20、A和B, 当从第一列开始输入数据时, 正确的数据输入方式是: (注: ■代表空格)

```
int a1,a2;char c1,c2;  
scanf("%d%c%d%c",&a1,&c1,&a2,&c2);
```

- ☐ A.10A ■ 20B<回车>
- ☒ B.10 ■ A ■ 20 ■ B<回车>
- ☐ C.10 ■ A20B ■ <回车>
- ☐ D.10A20 ■ B<回车>

作答保存成功

输入缓冲区:

%d

2	0	B	↵
---	---	---	---

```
1 #include <stdio.h>  
2 int main()  
3 {  
4     int a1, a2;  
5     char c1, c2;  
6     scanf("%d%c%d%c", &a1, &c1, &a2, &c2);  
7     printf("%d %d %c %c", a1, a2, c1, c2);  
8     return 0;  
9 }
```

如果输入如下, 则输出会是什么

Input:10A□20B

Output:?

7/20 单选题 (分值5.0分, 难度: 易)

已有如下定义和输入语句, 若要求a1, a2, c1, c2的值分别为10、20、A和B, 当从第一列开始输入数据时, 正确的数据输入方式是: (注: ■代表空格)

```
int a1,a2;char c1,c2;  
scanf("%d%c%d%c",&a1,&c1,&a2,&c2);
```

- ☐ A.10A■20B<回车>
- ☒ B.10■A■20■B<回车>
- ☐ C.10■A20B■<回车>
- ☐ D.10A20■B<回车>

作答保存成功

输入缓冲区:

%c
B

```
1 #include <stdio.h>  
2 int main()  
3 {  
4     int a1, a2;  
5     char c1, c2;  
6     scanf("%d%c%d%c", &a1, &c1, &a2, &c2);  
7     printf("%d %d %c %c", a1, a2, c1, c2);  
8     return 0;  
9 }
```

如果输入如下, 则输出会是什么

Input:10A□20B

Output:?

7/20 单选题 (分值5.0分, 难度: 易)

已有如下定义和输入语句, 若要求a1, a2, c1, c2的值分别为10、20、A和B, 当从第一列开始输入数据时, 正确的数据输入方式是: (注: ■代表空格)

```
int a1,a2;char c1,c2;  
scanf("%d%c%d%c",&a1,&c1,&a2,&c2);
```

- ☐ A.10A ■ 20B<回车>
- ☒ B.10 ■ A ■ 20 ■ B<回车>
- ☐ C.10 ■ A20B ■ <回车>
- ☐ D.10A20 ■ B<回车>

作答保存成功

输入缓冲区:

%c
B

```
1  #include <stdio.h>  
2  int main()  
3  {  
4      int a1, a2;  
5      char c1, c2;  
6      scanf("%d%c%d%c", &a1, &c1, &a2, &c2);  
7      printf("%d %d %c %c", a1, a2, c1, c2);  
8      return 0;  
9  }
```

如果输入如下, 则输出会是什么

Input:10A□20B

Output: 10□20□A□B

7/20 单选题 (分值5.0分, 难度: 易)

已有如下定义和输入语句, 若要求a1, a2, c1, c2的值分别为10、20、A和B, 当从第一列开始输入数据时, 正确的数据输入方式是: (注: ■代表空格)

```
int a1,a2;char c1,c2;  
scanf("%d%c%d%c",&a1,&c1,&a2,&c2);
```

- ☐ A.10A■20B<回车>
- ☒ B.10■A■20■B<回车>
- ☐ C.10■A20B■<回车>
- ☐ D.10A20■B<回车>

作答保存成功

%d

输入缓冲区:

1	0	■	A	■	2	0	■	B	↵
---	---	---	---	---	---	---	---	---	---

```
1 #include <stdio.h>  
2 int main()  
3 {  
4     int a1, a2;  
5     char c1, c2;  
6     scanf("%d%c%d%c", &a1, &c1, &a2, &c2);  
7     printf("%d %d %c %c", a1, a2, c1, c2);  
8     return 0;  
9 }
```

如果输入如下, 则输出会是什么

Input:10□A□20□B

Output:?

7/20 单选题 (分值5.0分, 难度: 易)

已有如下定义和输入语句, 若要求a1, a2, c1, c2的值分别为10、20、A和B, 当从第一列开始输入数据时, 正确的数据输入方式是: (注: ■代表空格)

```
int a1,a2;char c1,c2;  
scanf("%d%c%d%c",&a1,&c1,&a2,&c2);
```

- ☐ A.10A ■ 20B<回车>
- ☒ B.10 ■ A ■ 20 ■ B<回车>
- ☐ C.10 ■ A20B ■ <回车>
- ☐ D.10A20 ■ B<回车>

作答保存成功

输入缓冲区:

%c

□	A	□	2	0	□	B	↵
---	---	---	---	---	---	---	---

```
1  #include <stdio.h>  
2  int main()  
3  {  
4      int a1, a2;  
5      char c1, c2;  
6      scanf("%d%c%d%c", &a1, &c1, &a2, &c2);  
7      printf("%d %d %c %c", a1, a2, c1, c2);  
8      return 0;  
9  }
```

如果输入如下, 则输出会是什么

Input:10□A□20□B

Output:?

7/20 单选题 (分值5.0分, 难度: 易)

已有如下定义和输入语句, 若要求a1, a2, c1, c2的值分别为10、20、A和B, 当从第一列开始输入数据时, 正确的数据输入方式是: (注: ■代表空格)

```
int a1,a2;char c1,c2;  
scanf("%d%c%d%c",&a1,&c1,&a2,&c2);
```

- ☐ A.10A■20B<回车>
- ☒ B.10■A■20■B<回车>
- ☐ C.10■A20B■<回车>
- ☐ D.10A20■B<回车>

作答保存成功

输入缓冲区:

%d 停

A	■	2	0	■	B	↵
---	---	---	---	---	---	---

```
1  #include <stdio.h>  
2  int main()  
3  {  
4      int a1, a2;  
5      char c1, c2;  
6      scanf("%d%c%d%c", &a1, &c1, &a2, &c2);  
7      printf("%d %d %c %c", a1, a2, c1, c2);  
8      return 0;  
9  }
```

如果输入如下, 则输出会是什么

Input:10□A□20□B

Output:?

7/20 单选题 (分值5.0分, 难度: 易)

已有如下定义和输入语句, 若要求a1, a2, c1, c2的值分别为10、20、A和B, 当从第一列开始输入数据时, 正确的数据输入方式是: (注: ■代表空格)

```
int a1,a2;char c1,c2;
scanf("%d%c%d%c",&a1,&c1,&a2,&c2);
```

- ☐ A.10A ■ 20B<回车>
- ☒ B.10 ■ A ■ 20 ■ B<回车>
- ☐ C.10 ■ A20B ■ <回车>
- ☐ D.10A20 ■ B<回车>

作答保存成功

输入缓冲区:

%d 停

A		2	0		B	↵
---	--	---	---	--	---	---

```
1 #include <stdio.h>
2 int main()
3 {
4     int a1, a2;
5     char c1, c2;
6     scanf("%d%c%d%c", &a1, &c1, &a2, &c2);
7     printf("%d %d %c %c", a1, a2, c1, c2);
8     return 0;
9 }
```

如果输入如下, 则输出会是什么

Input:10□A□20□B

Output:10 □32767□□□

变量a2未经过初始化, 输出的32767是一个“脏”值, 换一个编译器或者机器可能输出的值就不一样, 输出的a2是什么值也是未定义的行为。

以下程序的输出为：

```
#include <stdio.h>
int main() {
    float a = 3490593.0;
    printf("%d\n",a);
    return 0;
}
```

- ☐ A.3490593
- ☐ B.1247087748
- ☒ C.-2147483648
- ☐ D.不确定

float => int :

```
1  #include <stdio.h>
2  int main(){
3      float a = 3490593.0;
4      float b = 3490593.0;
5      float d = 3490593.0;
6      printf("%d\n",a);
7      printf("%d\n",b);
8      printf("%d\n",d);
9      return 0;
10 }
```

```
-7000
1431745200
1431745200
```

int => float:

```
1  #include <stdio.h>
2  int main(){
3      int a = 42, b = 123, c = 42;
4      printf("%f\n", a);
5      printf("%f\n", b);
6      printf("%f\n", c);
7      return 0;
8  }
```

```
0.000000
0.000000
0.000000
```

如果 printf 函数的格式字符串和参数类型不匹配，会发生**未定义行为**（undefined behavior）。这意味着程序可能会以不可预测的方式运行，可能会导致程序崩溃、数据损坏、安全漏洞或其他问题

double <=> float:

```
1  #include <stdio.h>
2  int main(){
3      float a = 2E10;
4      double b = 2E50;
5      printf("float a = %f\n", a);
6      printf("double b = %lf\n", b);
7      printf("a float -> double = %lf\n",a);
8      printf("b double -> float = %f\n",b);
9
10     a = 1.12345678901234567890123;
11     b = 1.123456789012345678901234567890123456789012345;
12     printf("float a = %.30f\n", a);
13     printf("double b = %.30lf\n", b);
14     printf("a float -> double = %.30lf\n",a);
15     printf("b double -> float = %.30f\n",b);
16
17     return 0;
18 }
```

```
float a = 20000000000.000000
double b = 2000000000000000015259539682183774006589929941893120.000000
a float -> double = 20000000000.000000
b double -> float = 2000000000000000015259539682183774006589929941893120.000000
float a = 1.123456835746765136718750000000
double b = 1.123456789012345691247674039914
a float -> double = 1.123456835746765136718750000000
b double -> float = 1.123456789012345691247674039914
```

虽然在gcc中，double和float不管用%lf还是用%f输出，其结果都是一样的。但在别的编译器上结果可能不一样，严重的可能导致漏洞和错误。所以，在使用printf时还是要注意**格式化字符和所传参数对应**。

6. 单选题 (5.0分)

若x,y均定义为int型, z定义为double型, 以下不合法的scanf函数调用语句是:

- ☐ A. scanf("%d%lx,%le",&x,&y,&z);
- ☐ B. scanf("%2d * %d%lf",&x,&y,&z);
- ☐ C. scanf("%x%*d%o",&x,&y);
- ☒ D. scanf("%x%o%6.2f",&x,&y,&z);

HEX	3 FFFF FFFF
DEC	17,179,869,183
OCT	177 777 777 777
BIN	0011 1111 1111 1111 1111 1111 1111 1111

A选项是合法的, 编译器为我们进行了long到int的隐式转换。但是long到int可能存在截断问题, 因而存在漏洞。

```
1  #include <stdio.h>
2  int main(){
3      int x, y;
4      double z;
5      scanf("%d%lx,%le",&x,&y,&z);
6      printf("%d %d %lf\n", x, y, z);
7  }
```

样例1:

当输入的16进制数为12时, y输出的结果为18, 是正确的。

```
12 12,2e5
12 18 200000.000000
```

样例2:

当输入的16进制数是3fffffff时, 因为存在long到int的转换, 会发生截断, 导致输入到y中的数值为ffffff, 对应的数值为-1

```
12 3fffffff,2e5
12 -1 200000.000000
```


6. 单选题 (5.0分)

若x,y均定义为int型, z定义为double型, 以下不合法的scanf函数调用语句是:

- ☐ A. scanf("%d|x,%le",&x,&y,&z);
- ☐ B. scanf("%2d * %d%lf",&x,&y,&z);
- ☐ C. scanf("%x%d*o",&x,&y);
- ☒ D. scanf("%x%o%6.2f",&x,&y,&z);

HEX 12
DEC 18

```
1 #include <stdio.h>
2 int main(){
3     int x, y;
4     double z;
5     scanf("%x%o%6.2f",&x,&y,&z);
6     printf("%d %d %lf\n", x, y, z);
7 }
```

样例1:

```
12 12 1234.1234
18 10 0.000000
```

在scanf中输入浮点数时指定精度会导致一些未定义的行为。所以D选项是不合法的。

附加字符	说 明
l	输入长整型数据 (可用%d, %lo, %lx, %lu) 以及double型数据 (用%lf或%le)
h	输入短整型数据 (可用%hd, %ho, %hx)
域宽	指定输入数据所占宽度 (列数), 域宽应为正整数, 注意: 输入浮点数时不能规定精度
*	本输入项在读入后不赋给相应的变量

2.8 数据类型与安全

- 问题：人用纸笔进行计算和用计算机程序进行计算相比有什么“优势”？



VS.



- 事实上，程序中存储数据的变量的默认空间大小是**有限的**！但是，只要纸足够大，人能够进行**任意大小**的数据计算
- 计算机程序中的变量通常被赋予了一个固定大小的存储空间，若在计算过程中，如果变量的**存储空间不够用时**，可能会出现一个**非期望的值**
- 非期望的值可能会导致程序执行结果错误，甚至引发**安全问题**

更多内容：《程序设计安全》（信息安全专业选修课）

2.8.1 整数安全

四千二百万“余额”交通卡惊现上海地铁 运营方：数据损坏

2013-03-16 13:30

来源：新民网 记者：李欣、沈文林 新民网编辑：戴慧菁

参与评论(0) 打印 字号：T|T

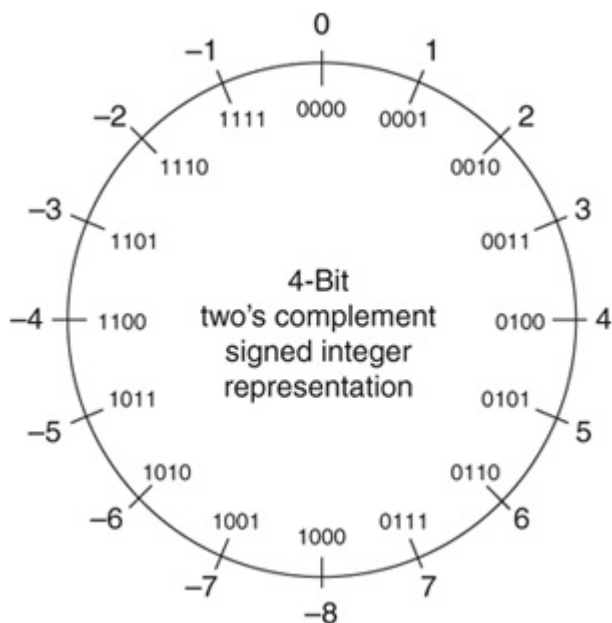


新闻背后的技术原理是什么？真的是数据损坏吗？

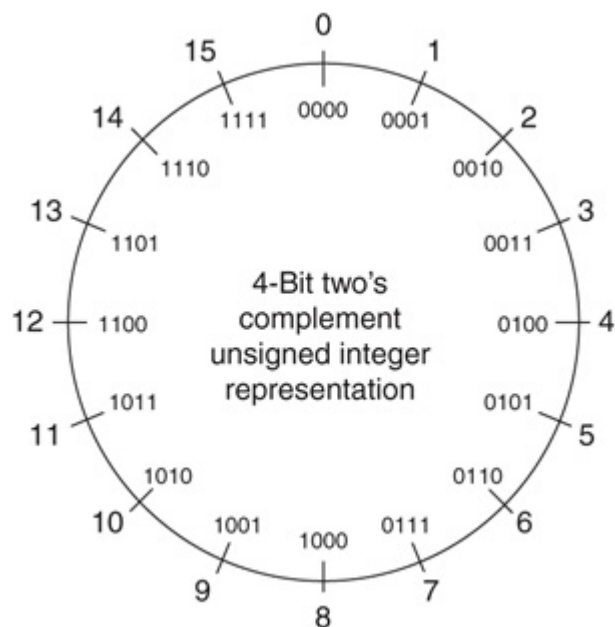
2.8.1 整数安全

■ 几乎所有现代计算机都采用了补码表示法表示整数，C提供了丰富的整数类型（有无符号、占用不同内存字节）

- n 位带符号整型的取值范围是 -2^{n-1} 到 $2^{n-1}-1$
- n 位无符号整型的取值范围是0到 2^n-1



带符号整数表示（4位、补码） **-8 ~ 7**



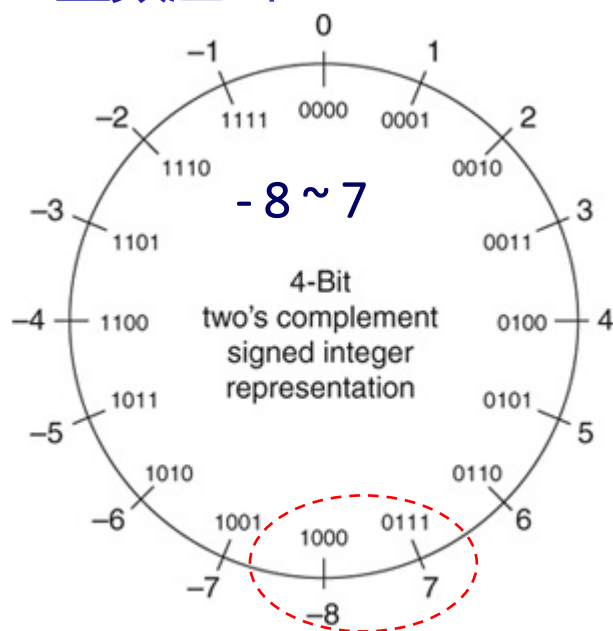
无符号整数表示（4位、补码） **0 ~ 15**

2.8.1 整数安全

- 计算机程序不可避免地会同时处理各种不同类型的整型数据，
- 在涉及混合类型的数据计算时，需要统一操作数类型，这涉及各种整型数据间的类型转换
- 整数相关的错误与整型的长度与是否带符号密切相关
- 整数错误情形
 - 整数溢出
 - 符号曲解
 - 截断错误

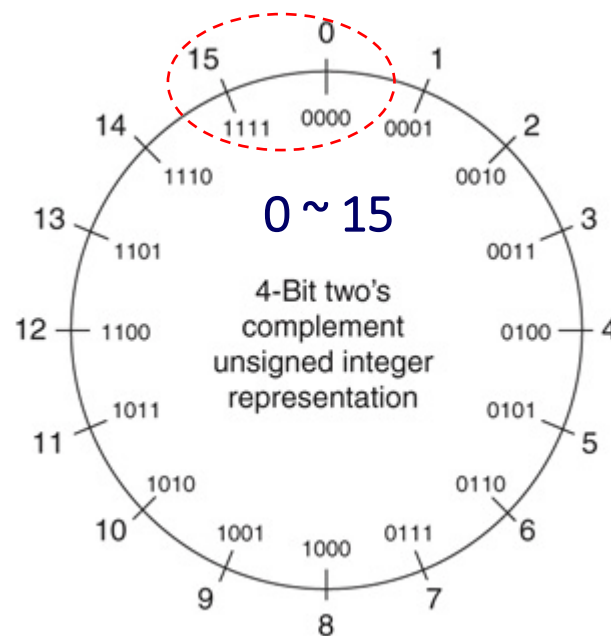
整数溢出

- 当一个整数被增加超过其最大值或被减小小于其最小值时，即发生整数溢出



带符号整数表示（4位、补码）

$1000 - 1 = 0111$ $-8 - 1 = -9$
 $0111 + 1 = 1000$ $7 + 1 = -8$



无符号整数表示（4位、补码/原码）

$1111 + 1 = 10000$ $15 + 1 = 0$
 $0000 - 1 = 1111$ $0 - 1 = 15$

无论怎么算，结果都被限定在当前整型的范围内！

整数溢出

- 超出存储空间表示范围后，溢出往往导致回绕

```
1. int i;  
2. unsigned int j;  
  
3. i = INT_MAX; // 2,147,483,647  
4. i++;  
5. printf("i = %d\n", i); /* i = -2,147,483,648 */  
  
6. j = UINT_MAX; // 4,294,967,295;  
7. j++;  
8. printf("j = %u\n", j); /* j = 0 */  
  
9. i = INT_MIN; // -2,147,483,648;  
10. i--;  
11. printf("i = %d\n", i); /* i = 2,147,483,647 */  
  
12. j = 0;  
13. j--;  
14. printf("j = %u\n", j); /* j = 4,294,967,295 */
```

最大的带符号整形加1
后变成了最小的

最小的无符号整形减1
后变成了最大的

该循环永远不会终止: `for (unsigned i = n; --i >= 0; )`

符号曲解

- 带符号数和无符号数相互转换时可能会发生符号错误

- 符号位失去了原有的作用，被作为值的一部分

例：char $\rightarrow$ unsigned char （带符号 $\rightarrow$ 无符号）

1111 1111 (-1) $\rightarrow$ 1111 1111 (127)

（符号位被当作数值位）

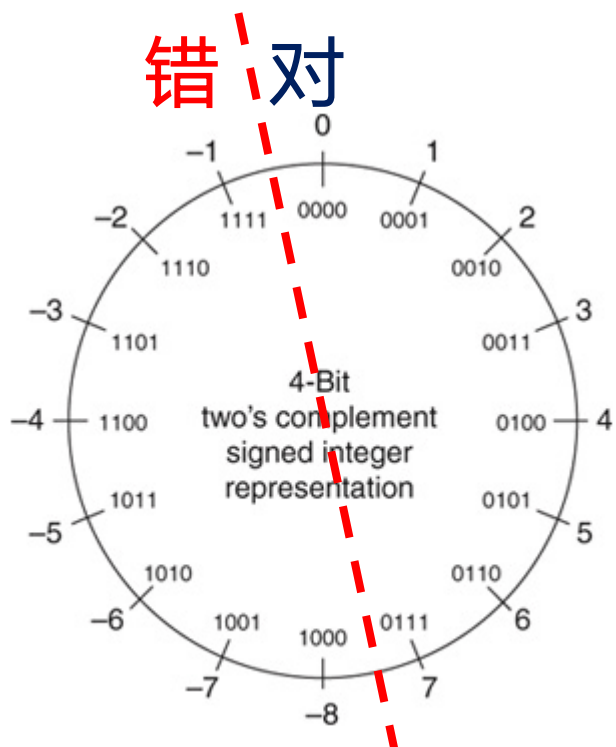
- 数据位被当作了符号位

例：unsigned char $\rightarrow$ char （无符号 $\rightarrow$ 带符号）

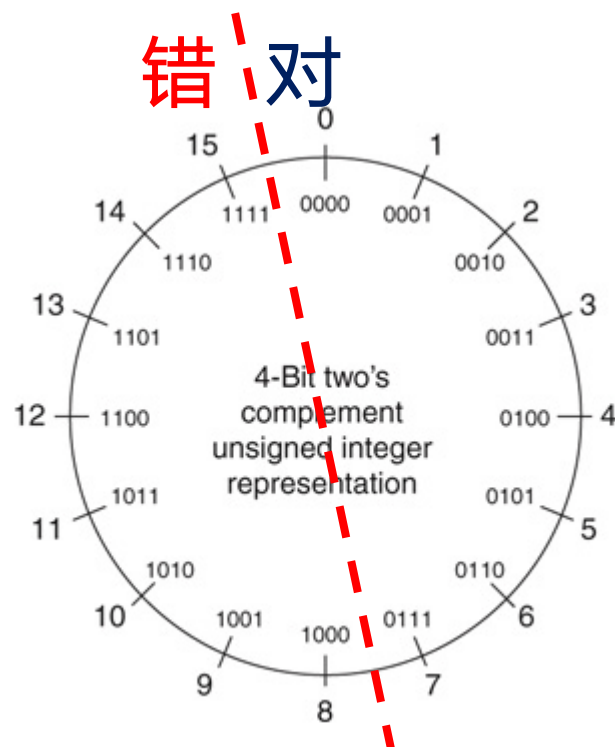
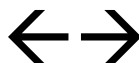
1000 0001 (129) $\rightarrow$ 1000 0001 (-127)

（最高数值位被当作符号位）

符号曲解



带符号整数表示（4位、补码）



无符号整数表示（4位、补码/原码）

同样大小无符号/带符号整型转换时，一半的情况可能会出错

符号曲解

- 常见错误情形：当一个负值的带符号整型（负值最高位为1）转换为无符号整型或被当作无符号整型处理时，将被作为一个非常大的数

```
1. unsigned int ulong = ULONG_MAX;
2. char c = -1;

3. if (c == ulong) {
4.     printf("Why is -1 = 4,294,967,295???\n");
5. }
```

相等比较也是一种计算，这2个操作数将转换到unsigned int，-1就被当作了最大的无符号整数（符号位扩展）

1111 1111 → 1111 1111 1111 1111 1111 1111 1111 1111

截断错误

- 当将一个较大的整型转换到较小的整型（如赋值），并且该数的原值超过较小类型的表示范围，则会出现截断错误
- 一般情况下，原值的低位被保留而高位被丢弃

```
1. unsigned int a = 0x110;  
2. unsigned int b = 0x230;  
3. unsigned char c = a + b;  
4. printf("%x %x\n", a + b, c);
```

340 40

回顾：美链币智能合约漏洞

```
1. function batchTransfer(uint256 _value, uint256 _receivers[]) {
2.     uint cnt = _receivers.length;
3.     uint256 amount = uint256(cnt) * _value;
4.     require(cnt > 0 && cnt <= 20);
5.     require(_value > 0 && balances[msg.sender] >= amount);
6.     balances[msg.sender] = balances[msg.sender].sub(amount);
7.     for (uint i = 0; i < cnt; i++) {
8.         balances[_receivers[i]] = balances[_receivers[i]].add(_value);
9.     }
10.     .....
11.     return true;
12. }
```

整数溢出导致回绕，amount变成很小的无符号数

无效的检查，无法保证用户余额大于交易金额

四千二百万“余额”交通卡惊现上海地铁 运营方：数据损坏

2013-03-16 13:30

来源：新民网 记者：李欣、沈文林 新民网编辑：戴慧青

参与评论(0) 打印 字号：T|T



输出示例:

```
printf ("-20=%lu\n", -20);
```

新闻背后的技术原理是什么?

-20被诠释为一个正整数☺

-20 (带符号) =

1111 1111 1111 1111 1111 1111
1110 1100

= 4294967276 (无符号)

交通卡系统使用整数来存储余额
(单位为分)，某张地铁卡欠费
2角 (余额为-20)

交通卡系统在显示余额时，按照无
符号整数形式输出 (例如: %lu) ,
导致显示四千多万余额 (隐式转换)

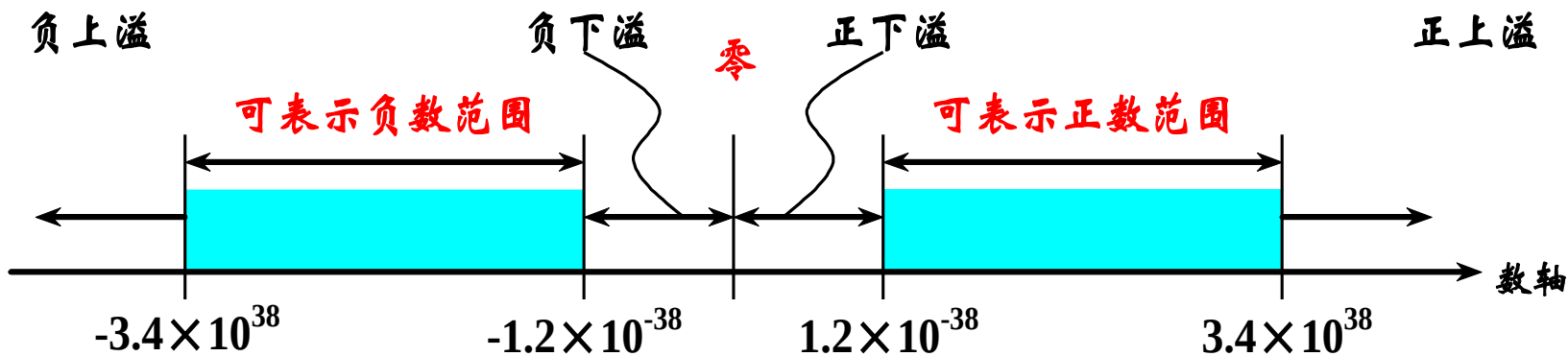
交通卡系统应该能考虑到负余额情
况，使用带符号数来存储处理余额

2.8.1 整数安全

- 缓解方法：在编码时应该注意检测操作数的大小。此外，还应该选择合适的整数类型，以降低整数错误的风险，3个常见的策略：
 - 避免不必要的带符号整型（C++语言默认带符号）
 - 避免不必要的混合类型计算（例如：比较）
 - 不用过分考虑内存消耗（例如：用int而非char型表示气温）

2.8.2 浮点数安全

■ 溢出：数值超出可表示的范围



```
1. int main(void) {
2.     float positive_toobig = 3.4e38 * 10;
3.     float positive_toosmall = 1.175494e-38 / 10;
4.     float positive_extrasmall = 1.175494e-38 / 1000000000;

5.     float negative_toobig = -3.4e38 * 10;
6.     float negative_toosmall = -1.175494e-38 / 10;
7.     float negative_extrasmall = -1.175494e-38 / 1000000000;

8.     printf("%e %e\n", positive_toobig, negative_toobig);
9.     printf("%e %e\n", positive_toosmall, negative_toosmall);
10.    printf("%e %e\n", positive_extrasmall, negative_extrasmall);
11. }
```

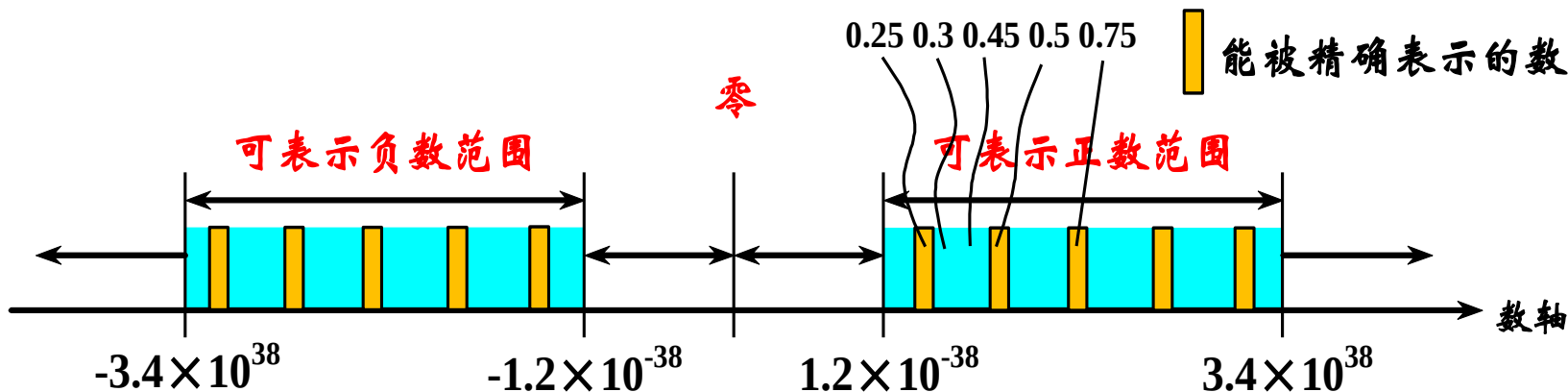
```
inf -inf
1.175495e-39 -1.175495e-39
0.000000e+00 -0.000000e+00
```

思考：下列代码的输出结果是什么？为啥？

```
double uv = 0.3;
if (uv == 0.3) printf("uv equals to 0.3\n");
```

2.8.2 浮点数安全

精度丢失：数值无法被精确表示



```
1. int main(void)
2. {
3.     float x = 0.3, y = 0.45;
4.     float a = 0.25, b = 0.5;
5.     float z = 0.75;
6.     if (x == 0.3) printf("x equals to 0.3\n");
7.     if (y == 0.45) printf("y equals to 0.45\n");
8.     if (a == 0.25) printf("a equals to 0.25\n");
9.     if (b == 0.5) printf("b equals to 0.5\n");
10.    if (z == 0.75) printf("z equals 0.75\n");
11.    if (x+y == z) printf("x+y equals to z\n");
12.    if (a+b == z) printf("a+b equals to z\n");
13.}
```

```
a equals to 0.25
b equals to 0.5
z equals 0.75
x+y equals to z
a+b equals to z
```


在线工具集合

- ASCII码查询

<https://www.mokuge.com/tool/asciito16/>

- 汉字国标码查询

<https://www.qqxiuzi.cn/bianma/guobiaoma.php>

- 原码/反码/补码计算器：

<http://www.atoolbox.net/Tool.php?Id=952>

- 进制转换

<http://www.speedfly.cn/tools/hexconvert/>

- C/C++标准库参考手册

<https://www.runoob.com/cprogramming/c-standard-library.html>

<https://www.runoob.com/cplusplus/cpp-standard-library.html>

课堂练习

🏠 问题描述

✈ 代码提交

🔍 查看结果 3

🔗 参考代码 ...

时间限制：1.0s; 内存限制：256.0MB

📖 算术表达式

有如图所示的变量定义及14个表达式。试按你对表达式的理解，在题目给定的变量定义条件下，心（笔）算相应的值，然后将答案替换到代码中去。比如，如果第1个表达式的值为5，则 the answer of Q1 替换为5；如果为浮点数5，则写为5.0。注意不要有多余的空格。此程序的作用是输入题号，输出相应的答案。

一、计算题

设有变量定义 `char a='a'; int i=3, j=5, b; float x=2.5;`
`double y=5.0, z;`，计算下面表达式的值。

- | | |
|------------------------------------|--|
| 1. <code>a+i-j+x/y</code> | 2. <code>(x+y)+i++</code> |
| 3. <code>y+=i-=j*==++x;</code> | 4. <code>b=a+=j%i</code> |
| 5. <code>a=a+i, a+j</code> | 6. <code>a=(a+ i, a+j)</code> |
| 7. <code>y=(x=2, x+1, x+2)</code> | 8. <code>i -=j*=x+y</code> |
| 9. <code>(i++)*(--j)</code> | 10. <code>z= (i++)*(j++)</code> |
| 11. <code>(int)x / (int)y+y</code> | 12. <code>(float)i/(++j)</code> |
| 13. <code>-j%i+j</code> | 14. <code>(int)y% i %(int)(x+y)</code> |

课堂练习

```
void question3()
```

```
{
```

```
    char a = 'a';
```

```
    int i = 3, j = 5, b;
```

```
    float x = 2.5;
```

```
    double y = 5.0, z;
```

```
    printf("%d %lf\n", sizeof((y+=i-=j*=++x)), (y+=i-=j*=++x));
```

```
}
```

```
void question5()
```

```
{
```

```
    char a = 'a';
```

```
    int i = 3, j = 5, b;
```

```
    float x = 2.5;
```

```
    double y = 5.0, z;
```

```
    printf("%d %d\n", sizeof((a=a+i,a+j)), (a=a+i,a+j));
```

```
}
```

```
void question7()
```

```
{
```

```
    char a = 'a';
```

```
    int i = 3, j = 5, b;
```

```
    float x = 2.5;
```

```
    double y = 5.0, z;
```

```
    printf("%d %lf\n", sizeof(y = (x=2,x+1, x+2)), (y = (x=2,x+1, x+2)));
```

```
}
```

```
int main()
```

```
{
```

```
    question3();
```

```
    question5();
```

```
    question7();
```

```
}
```

课堂练习

```
char a = 'a';
int i = 3, j = 5, b;
float x = 2.5;
double y = 5.0, z;
```

3. $y += i - j * ++x$
- ①. 一目运算符++ 优先做高于二目运算符.
故先算 $++x$ 值为 3.5
 - ②. 赋值操作右结合.
故先算 $*$ = 内算 $-$ = 最后算 $+=$
 $j * 3.5$ 值为 17.5 赋值转换为右值
 $i - 17.5$ 值为 -14.5 共转 int
 $y += (-14.5)$ 值为 -9.0 赋值转换为
 右值类型 double

5. $a = a + i * a + j$
- ①. 逗号表达式从右往左计算.
最后结果为最后一个逗号表达式右值的值.
 - ②. $a = a + i$ 值为 $17 + 3 = 20$. 类型为 int
 $a + j = 20 + 5 = 25$. 类型为 int. char 提升
 - ③. 最后值为 25

7. $y = (x = 2, x + 1, x + 2)$
- ①. 先计算多少右值
 $\begin{array}{ccc} x = 2 & x + 1 & x + 2 \\ \hline 2.0 & 3.0 & 4.0 \end{array}$
 逗号表达式值为 4.0
 - ②. $y = 4.0$ 的值为 4.0

关于yoj#8 计算学分绩点 的一点点小疑问

■ 我在写 yoj#8 计算学分绩点 时试图不使用机械重复的编码方式编写程序，并简化代码行数（如下）（也不知道这个方法是不是就更简单或高级？）但不幸地发现似乎只比纯粹一行行列出 成绩比较—判断—赋值学分—输出 少了四行代码.....且占用的内存更多。

■ 请问这是否意味着我使用的“简化”代码并不如“机械编写”的代码适用？有没有其他更好一点的方法呢？（我这个暑假才开始接触编程，感觉很多知识都没有了解过.....）

“简化”代码及评测结果如下:

#8 计算学分绩点

Accepted

100

91 ms

632 KB

cpp / 630 B

```
#include <iostream>
using namespace std;
int main()
{
    int grade, n = 0;
    double credit;
    cin >> grade;
    if (grade >= 90 && grade <= 100)
        credit = 4.0;
    if (grade < 60)
        credit = 0;
    if (grade <= 89 && grade >= 60)
    {
        while (grade > 65)
        {
            grade -= 10;
            n++;
        }
        if (grade >= 63 && grade <= 65)
            credit = n + 1.3;
        if (grade >= 60 && grade <= 62)
            credit = n + 1.0;
        if (grade >= 56 && grade <= 59)
            credit = n + 0.7;
    }
    if (credit == 0)
        cout << credit;
    else
    {
        cout.setf(ios::fixed);
        cout.setf(ios::showpoint);
        cout.precision(1);
        cout << credit;
    }
    return 0;
}
```

重复性编写的代码及评测结果如下:

#8 计算学分绩点

Accepted

100

85 ms

504 KB

cpp / 838 B

吴

```
#include <iostream>
using namespace std;
int main()
{
    int grade;
    double credit;
    cin >> grade;
    if (grade >= 90 && grade <= 100)
        credit = 4.0;
    if (grade >= 86 && grade <= 89)
        credit = 3.7;
    if (grade >= 83 && grade <= 85)
        credit = 3.3;
    if (grade >= 80 && grade <= 82)
        credit = 3.0;
    if (grade >= 76 && grade <= 79)
        credit = 2.7;
    if (grade >= 73 && grade <= 75)
        credit = 2.3;
    if (grade >= 70 && grade <= 72)
        credit = 2.0;
    if (grade >= 66 && grade <= 69)
        credit = 1.7;
    if (grade >= 63 && grade <= 65)
        credit = 1.3;
    if (grade >= 60 && grade <= 62)
        credit = 1.0;
    if (grade < 60)
        credit = 0;
    if (credit == 0)
        cout << credit;
    else
    {
        cout.setf(ios::fixed);
        cout.setf(ios::showpoint);
        cout.precision(1);
        cout << credit;
    }
    return 0;
}
```