程序设计基础与实验

第六章: 数据类型与表达式

刘新国

浙江大学计算机学院
浙江大学 CAD&CG 国家重点实验室

December 1, 2021

内容提要

内容提要

C 语言的数据类型

基本数据类型

构造数据类型

C 语言的数据类型

基本数据类型

- 整型

构造数据类型

C 语言的数据类型

基本数据类型

- 整型
- 实型（浮点型）

构造数据类型

C 语言的数据类型

基本数据类型

- 整型
- 实型（浮点型）
- 字符型

构造数据类型

C 语言的数据类型

基本数据类型

- 整型
- 实型（浮点型）
- 字符型

构造数据类型

- 数组

C 语言的数据类型

基本数据类型

- 整型
- 实型（浮点型）
- 字符型

构造数据类型

- 数组
- 结构、联合、枚举

C 语言的数据类型

基本数据类型

- 整型
- 实型（浮点型）
- 字符型

构造数据类型

- 数组
- 结构、联合、枚举
- 指针类型

C 语言的数据类型

基本数据类型

- 整型
- 实型（浮点型）
- 字符型

构造数据类型

- 数组
- 结构、联合、枚举
- 指针类型
- 空类型（void）

整型：整数类型

在计算机中，所有的数据都是用二进制码（0，1）表示的

正整数的二进制表示【16 位表示为例】

整型：整数类型

在计算机中，所有的数据都是用二进制码（0，1）表示的

正整数的二进制表示【16 位表示为例】

右起第 n 位的权重为 2^n

整型：整数类型

在计算机中，所有的数据都是用二进制码（0，1）表示的

正整数的二进制表示【16 位表示为例】

右起第 n 位的权重为 2^n

0　0000 0000 0000 0000

整型：整数类型

在计算机中，所有的数据都是用二进制码（0，1）表示的

正整数的二进制表示【16 位表示为例】

右起第 n 位的权重为 2^n

0　0000 0000 0000 0000

1　0000 0000 0000 0001

整型：整数类型

在计算机中，所有的数据都是用二进制码（0，1）表示的

正整数的二进制表示【16 位表示为例】

右起第 n 位的权重为 2^n

0　0000 0000 0000 0000

1　0000 0000 0000 0001

2　0000 0000 0000 0010

整型：整数类型

在计算机中，所有的数据都是用二进制码（0，1）表示的

正整数的二进制表示【16 位表示为例】

右起第 n 位的权重为 2^n

0　0000 0000 0000 0000

1　0000 0000 0000 0001

2　0000 0000 0000 0010

3　0000 0000 0000 0011

整型：整数类型

在计算机中，所有的数据都是用二进制码（0，1）表示的

正整数的二进制表示【16 位表示为例】

右起第 n 位的权重为 2^n

0　0000 0000 0000 0000

1　0000 0000 0000 0001

2　0000 0000 0000 0010

3　0000 0000 0000 0011

4　0000 0000 0000 0100

整型：整数类型

在计算机中，所有的数据都是用二进制码（0，1）表示的

正整数的二进制表示【16 位表示为例】

右起第 n 位的权重为 2^n

0　0000 0000 0000 0000

1　0000 0000 0000 0001

2　0000 0000 0000 0010

3　0000 0000 0000 0011

4　0000 0000 0000 0100

5　0000 0000 0000 0101

整型：整数类型

在计算机中，所有的数据都是用二进制码（0，1）表示的

正整数的二进制表示【16 位表示为例】

右起第 n 位的权重为 2^n

0　0000 0000 0000 0000

1　0000 0000 0000 0001

2　0000 0000 0000 0010

3　0000 0000 0000 0011

4　0000 0000 0000 0100

5　0000 0000 0000 0101

6　0000 0000 0000 0110

整型：整数类型

在计算机中，所有的数据都是用二进制码（0，1）表示的

正整数的二进制表示【16 位表示为例】

右起第 n 位的权重为 2^n

0　0000 0000 0000 0000

1　0000 0000 0000 0001

2　0000 0000 0000 0010

3　0000 0000 0000 0011

4　0000 0000 0000 0100

5　0000 0000 0000 0101

6　0000 0000 0000 0110

7　0000 0000 0000 0111

整型：整数类型

在计算机中，所有的数据都是用二进制码（0，1）表示的

正整数的二进制表示【16 位表示为例】

右起第 n 位的权重为 2^n

0　0000 0000 0000 0000

1　0000 0000 0000 0001

2　0000 0000 0000 0010

3　0000 0000 0000 0011

4　0000 0000 0000 0100

5　0000 0000 0000 0101

6　0000 0000 0000 0110

7　0000 0000 0000 0111

8　0000 0000 0000 1000

整型：整数类型

在计算机中，所有的数据都是用二进制码（0，1）表示的

正整数的二进制表示【16 位表示为例】

右起第 n 位的权重为 2^n

0　0000 0000 0000 0000

1　0000 0000 0000 0001

2　0000 0000 0000 0010

3　0000 0000 0000 0011

4　0000 0000 0000 0100

5　0000 0000 0000 0101

6　0000 0000 0000 0110

7　0000 0000 0000 0111

8　0000 0000 0000 1000

... ...

整数原码表示

最高位（最左边的）的一个 bit 用作符号位

整数原码表示

最高位（最左边的）的一个 bit 用作符号位

- 0 – 正的

整数原码表示

最高位（最左边的）的一个 bit 用作符号位

- 0 – 正的
- 1 – 负的

整数原码表示

最高位（最左边的）的一个 bit 用作符号位

- 0 – 正的
- 1 – 负的

其余的 bit 用来表示数值

整数原码表示

最高位（最左边的）的一个 bit 用作符号位

- 0 – 正的
- 1 – 负的

其余的 bit 用来表示数值

0 0000 0000 0000 0000

整数原码表示

最高位（最左边的）的一个 bit 用作符号位

- 0 – 正的
- 1 – 负的

其余的 bit 用来表示数值

0　0000 0000 0000 0000

1　0000 0000 0000 0001

整数原码表示

最高位（最左边的）的一个 bit 用作符号位

- 0 – 正的
- 1 – 负的

其余的 bit 用来表示数值

0　0000 0000 0000 0000

1　0000 0000 0000 0001

2　0000 0000 0000 0010

整数原码表示

最高位（最左边的）的一个 bit 用作符号位

- 0 – 正的
- 1 – 负的

其余的 bit 用来表示数值

0	0000 0000 0000 0000
1	0000 0000 0000 0001
2	0000 0000 0000 0010
3	0000 0000 0000 0011

整数原码表示

最高位（最左边的）的一个 bit 用作符号位

- 0 – 正的
- 1 – 负的

其余的 bit 用来表示数值

0	0000 0000 0000 0000
1	0000 0000 0000 0001
2	0000 0000 0000 0010
3	0000 0000 0000 0011
4	0000 0000 0000 0100

整数原码表示

最高位（最左边的）的一个 bit 用作符号位

- 0 – 正的
- 1 – 负的

其余的 bit 用来表示数值

0	0000 0000 0000 0000
1	0000 0000 0000 0001
2	0000 0000 0000 0010
3	0000 0000 0000 0011
4	0000 0000 0000 0100
5	0000 0000 0000 0101

整数原码表示

最高位（最左边的）的一个 bit 用作符号位

- 0 – 正的
- 1 – 负的

其余的 bit 用来表示数值

0	0000 0000 0000 0000
1	0000 0000 0000 0001
2	0000 0000 0000 0010
3	0000 0000 0000 0011
4	0000 0000 0000 0100
5	0000 0000 0000 0101
...	...

整数原码表示

最高位（最左边的）的一个 bit 用作符号位

- 0 – 正的
- 1 – 负的

其余的 bit 用来表示数值

0	0000 0000 0000 0000		
1	0000 0000 0000 0001	-1	1000 0000 0000 0001
2	0000 0000 0000 0010		
3	0000 0000 0000 0011		
4	0000 0000 0000 0100		
5	0000 0000 0000 0101		
...	...		

整数原码表示

最高位（最左边的）的一个 bit 用作符号位

- 0 – 正的
- 1 – 负的

其余的 bit 用来表示数值

0	0000 0000 0000 0000		
1	0000 0000 0000 0001	-1	1000 0000 0000 0001
2	0000 0000 0000 0010	-2	1000 0000 0000 0010
3	0000 0000 0000 0011		
4	0000 0000 0000 0100		
5	0000 0000 0000 0101		
...	...		

整数原码表示

最高位（最左边的）的一个 bit 用作符号位

- 0 – 正的
- 1 – 负的

其余的 bit 用来表示数值

0	0000 0000 0000 0000		
1	0000 0000 0000 0001	-1	1000 0000 0000 0001
2	0000 0000 0000 0010	-2	1000 0000 0000 0010
3	0000 0000 0000 0011	-3	1000 0000 0000 0011
4	0000 0000 0000 0100		
5	0000 0000 0000 0101		
...	...		

整数原码表示

最高位（最左边的）的一个 bit 用作符号位

- 0 – 正的
- 1 – 负的

其余的 bit 用来表示数值

0	0000 0000 0000 0000		
1	0000 0000 0000 0001	-1	1000 0000 0000 0001
2	0000 0000 0000 0010	-2	1000 0000 0000 0010
3	0000 0000 0000 0011	-3	1000 0000 0000 0011
4	0000 0000 0000 0100	-4	1000 0000 0000 0100
5	0000 0000 0000 0101		
...	...		

整数原码表示

最高位（最左边的）的一个 bit 用作符号位

- 0 – 正的
- 1 – 负的

其余的 bit 用来表示数值

0	0000 0000 0000 0000		
1	0000 0000 0000 0001	-1	1000 0000 0000 0001
2	0000 0000 0000 0010	-2	1000 0000 0000 0010
3	0000 0000 0000 0011	-3	1000 0000 0000 0011
4	0000 0000 0000 0100	-4	1000 0000 0000 0100
5	0000 0000 0000 0101	-5	1000 0000 0000 0101
...	...		

整数原码表示

最高位（最左边的）的一个 bit 用作符号位

- 0 – 正的
- 1 – 负的

其余的 bit 用来表示数值

0	0000 0000 0000 0000		
1	0000 0000 0000 0001	-1	1000 0000 0000 0001
2	0000 0000 0000 0010	-2	1000 0000 0000 0010
3	0000 0000 0000 0011	-3	1000 0000 0000 0011
4	0000 0000 0000 0100	-4	1000 0000 0000 0100
5	0000 0000 0000 0101	-5	1000 0000 0000 0101
...	...	...	...

整数原码表示

最高位（最左边的）的一个 bit 用作符号位

- 0 – 正的
- 1 – 负的

其余的 bit 用来表示数值

0	0000 0000 0000 0000		
1	0000 0000 0000 0001	-1	1000 0000 0000 0001
2	0000 0000 0000 0010	-2	1000 0000 0000 0010
3	0000 0000 0000 0011	-3	1000 0000 0000 0011
4	0000 0000 0000 0100	-4	1000 0000 0000 0100
5	0000 0000 0000 0101	-5	1000 0000 0000 0101
...	...	...	...

整数原码表示

最高位（最左边的）的一个 bit 用作符号位

- 0 – 正的
- 1 – 负的

其余的 bit 用来表示数值

0	0000 0000 0000 0000		
1	0000 0000 0000 0001	-1	1000 0000 0000 0001
2	0000 0000 0000 0010	-2	1000 0000 0000 0010
3	0000 0000 0000 0011	-3	1000 0000 0000 0011
4	0000 0000 0000 0100	-4	1000 0000 0000 0100
5	0000 0000 0000 0101	-5	1000 0000 0000 0101
...	...	...	...

这就是整数的原码表示，但是计算机采用的是补码表示

补码

最高位的 bit 表示数的符号

补码

最高位的 bit 表示数的符号

- 0 – 正数

补码

最高位的 bit 表示数的符号

- 0 – 正数
- 1 – 负数

补码

最高位的 bit 表示数的符号

- 0 – 正数
- 1 – 负数

剩余 bit 表示数值

补码

最高位的 bit 表示数的符号

- 0 – 正数
- 1 – 负数

剩余 bit 表示数值

- 正数补码和原码是一样的

补码

最高位的 bit 表示数的符号

- 0 – 正数
- 1 – 负数

剩余 bit 表示数值

- 正数补码和原码是一样的
- 负数补码：等于原码取反（称为反码）+ 1

补码

最高位的 bit 表示数的符号

- 0 – 正数
- 1 – 负数

剩余 bit 表示数值

- 正数补码和原码是一样的
- 负数补码：等于原码取反（称为反码）+ 1
  - -1 原码：1000 0000 0000 0001

补码

最高位的 bit 表示数的符号

- 0 – 正数
- 1 – 负数

剩余 bit 表示数值

- 正数补码和原码是一样的
- 负数补码：等于原码取反（称为反码）+ 1
  - -1 原码：1000 0000 0000 0001
    -1 反码：1111 1111 1111 1110

补码

最高位的 bit 表示数的符号

- 0 – 正数
- 1 – 负数

剩余 bit 表示数值

- 正数补码和原码是一样的
- 负数补码：等于原码取反（称为反码）+ 1
  - -1 原码：1000 0000 0000 0001
    -1 反码：1111 1111 1111 1110
    -1 补码：1111 1111 1111 1111

补码

最高位的 bit 表示数的符号

- 0 – 正数
- 1 – 负数

剩余 bit 表示数值

- 正数补码和原码是一样的
- 负数补码：等于原码取反（称为反码）+ 1
  - -1 原码：1000 0000 0000 0001
    -1 反码：1111 1111 1111 1110
    -1 补码：1111 1111 1111 1111
  - -2 原码：1000 0000 0000 0010

补码

最高位的 bit 表示数的符号

- 0 – 正数
- 1 – 负数

剩余 bit 表示数值

- 正数补码和原码是一样的
- 负数补码：等于原码取反（称为反码）+ 1
  - -1 原码：1000 0000 0000 0001
    -1 反码：1111 1111 1111 1110
    -1 补码：1111 1111 1111 1111
  - -2 原码：1000 0000 0000 0010
    -2 反码：1111 1111 1111 1101

补码

最高位的 bit 表示数的符号

- 0 – 正数
- 1 – 负数

剩余 bit 表示数值

- 正数补码和原码是一样的
- 负数补码：等于原码取反（称为反码）+ 1
  - -1 原码：1000 0000 0000 0001
    -1 反码：1111 1111 1111 1110
    -1 补码：1111 1111 1111 1111
  - -2 原码：1000 0000 0000 0010
    -2 反码：1111 1111 1111 1101
    -2 补码：1111 1111 1111 1110

原码、反码、补码

正数的原码、反码和补码相同，都等于原码

负数的原码、反码和补码不同

- 负数反码的符号为1，数值位 = 原码取反

原码、反码、补码

正数的原码、反码和补码相同，都等于原码

负数的原码、反码和补码不同

- 负数反码的符号为1，数值位 = 原码取反
- 负数补码的符号为1，数值位 = 反码 + 1

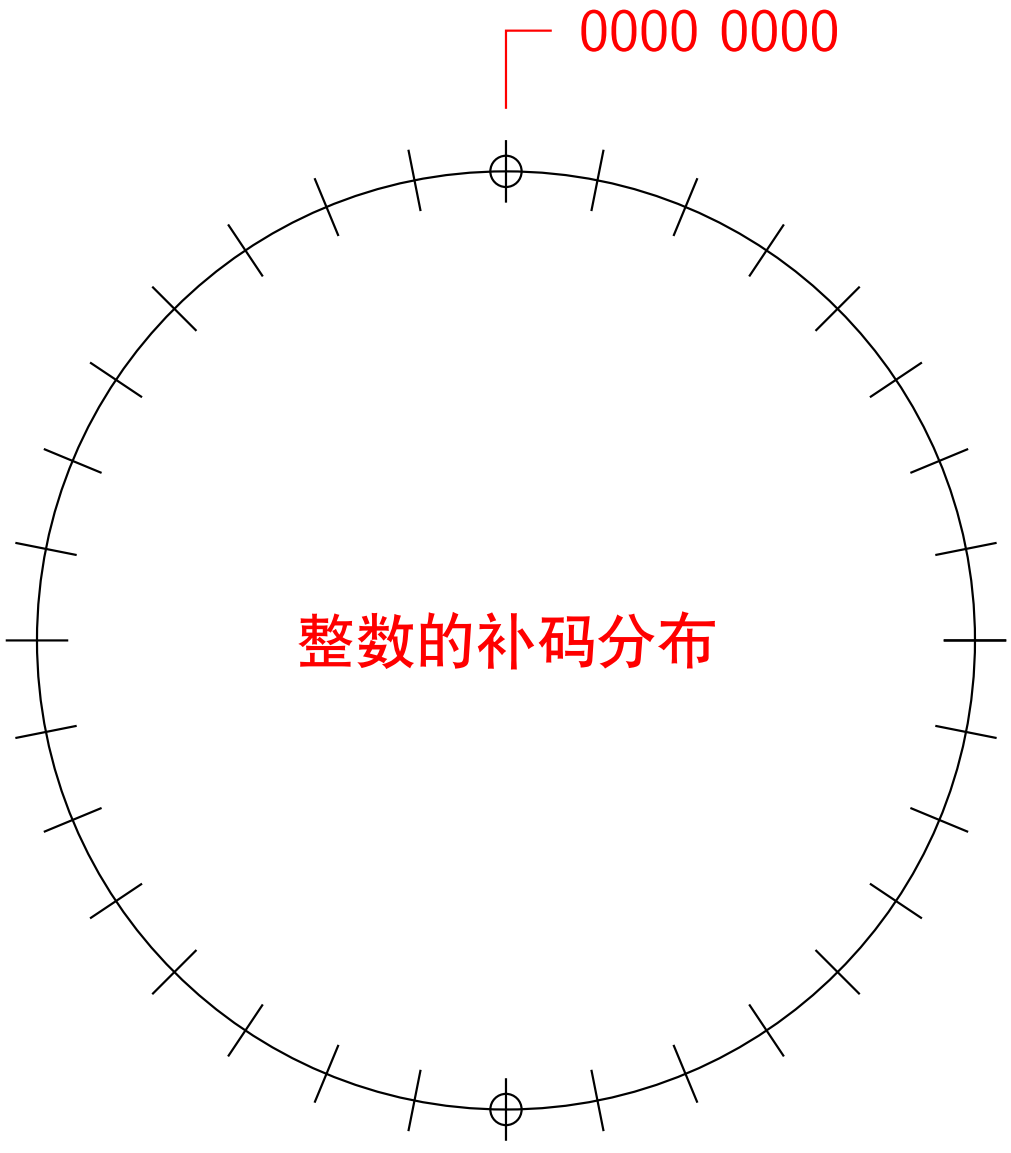
16 位整数的补码

- $2^{15}-1$ 补码: 0111 1111 1111 1111
- $2^{15}-2$ 补码: 0111 1111 1111 1110
- $2^{15}-3$ 补码: 0111 1111 1111 1101
- $2^{15}-4$ 补码: 0111 1111 1111 1100

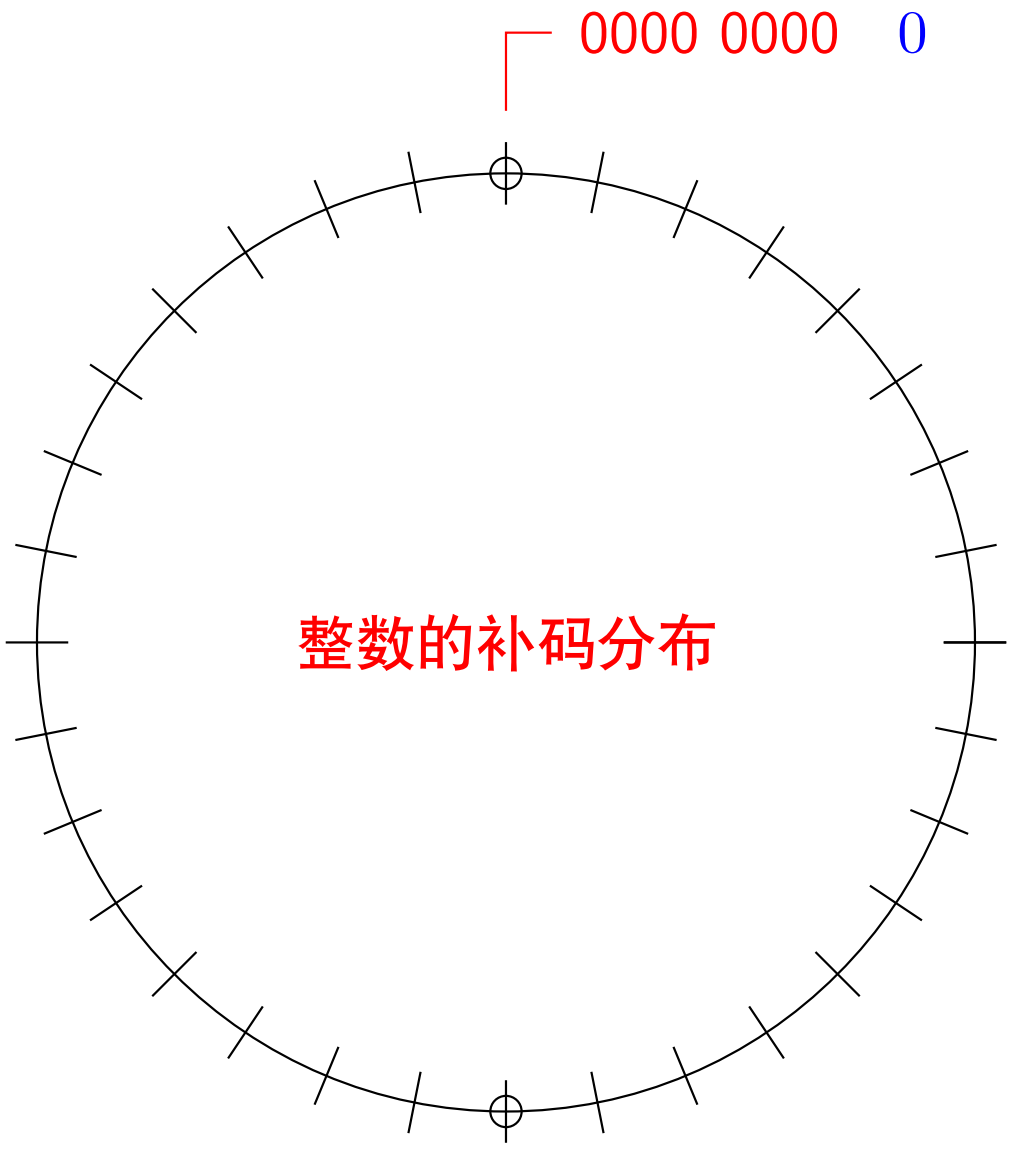
  ....
- 2 补码: 0000 0000 0000 0010
- 1 补码: 0000 0000 0000 0001
- 0 补码: 0000 0000 0000 0000
- -1 补码: 1111 1111 1111 1111
- -2 反码: 1111 1111 1111 1110

  ....
- $-2^{15}+3$ 补码: 1000 0000 0000 0011
- $-2^{15}+2$ 补码: 1000 0000 0000 0010
- $-2^{15}+1$ 补码: 1000 0000 0000 0001
- -2^{15} 补码: 1000 0000 0000 0000

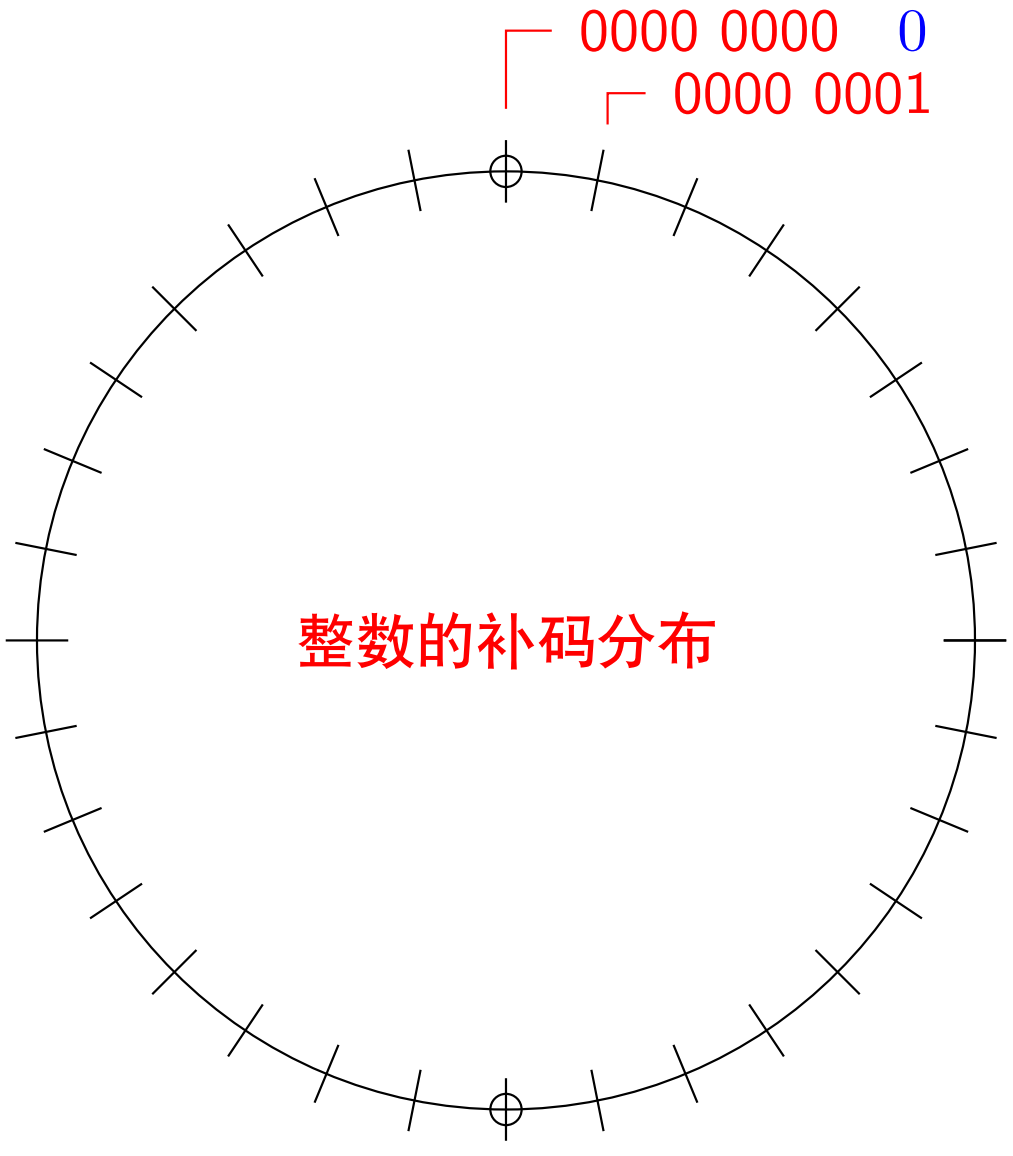
补码图解 - n 为比特总数



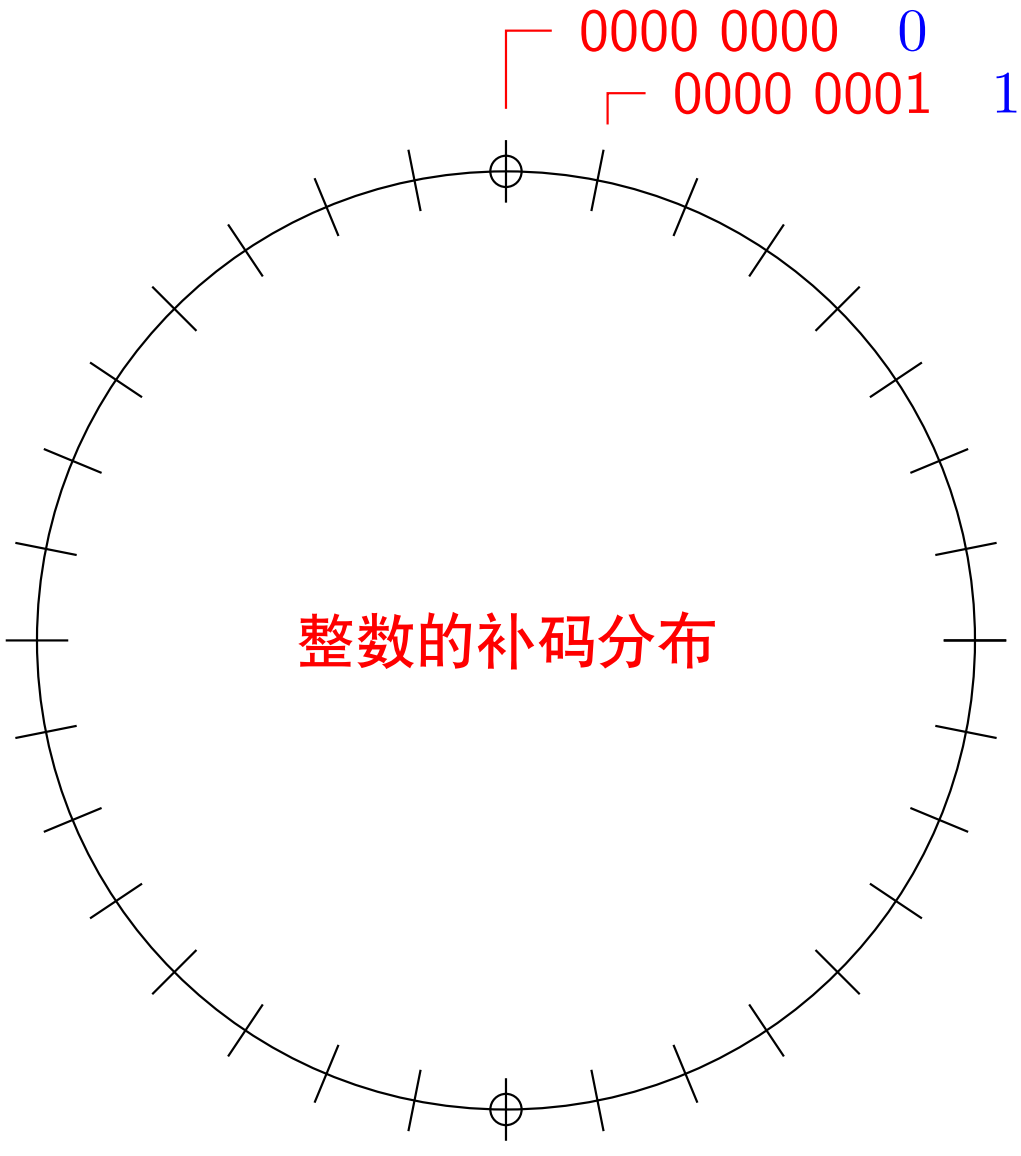
补码图解 - n 为比特总数



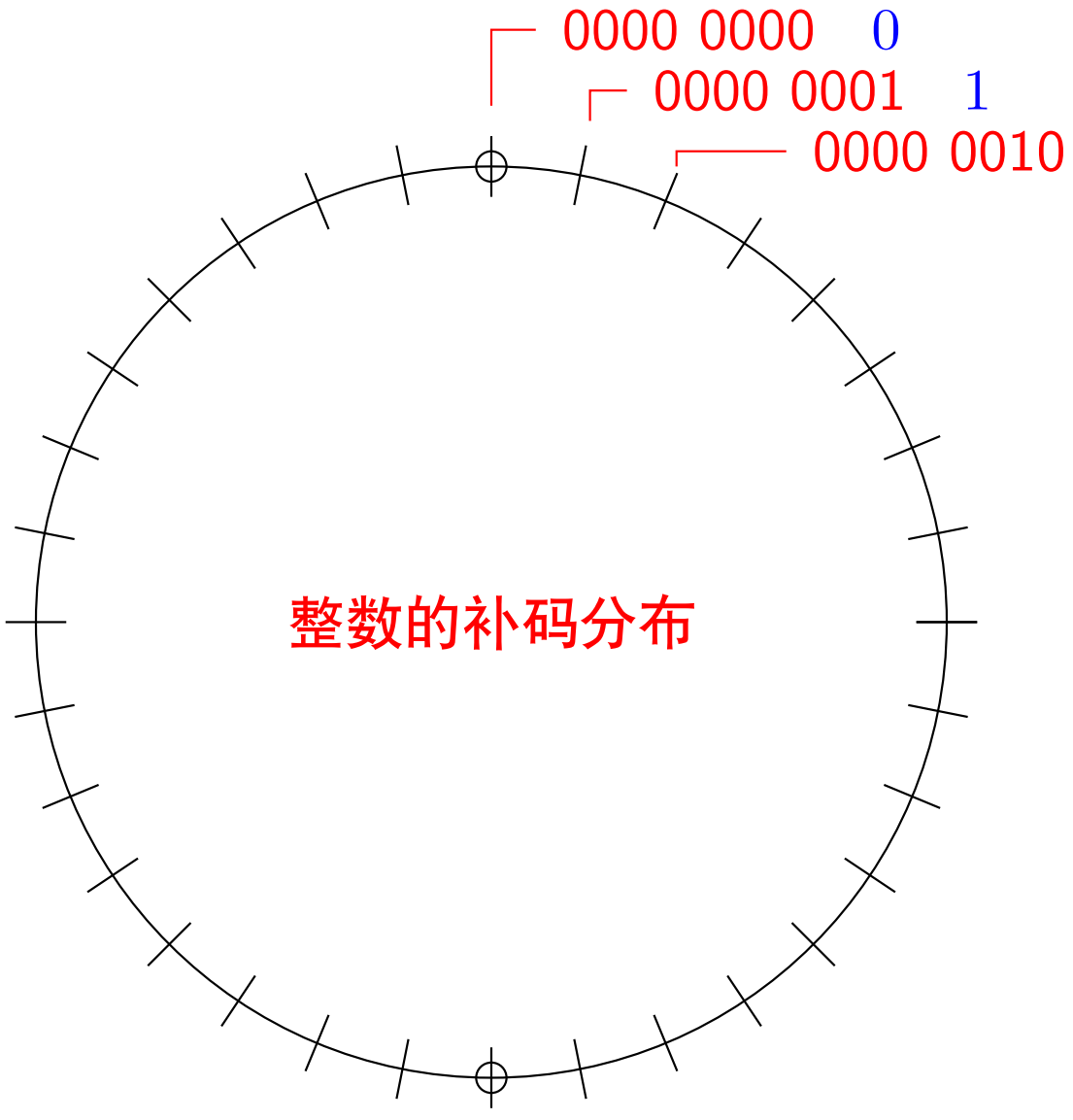
补码图解 - n 为比特总数



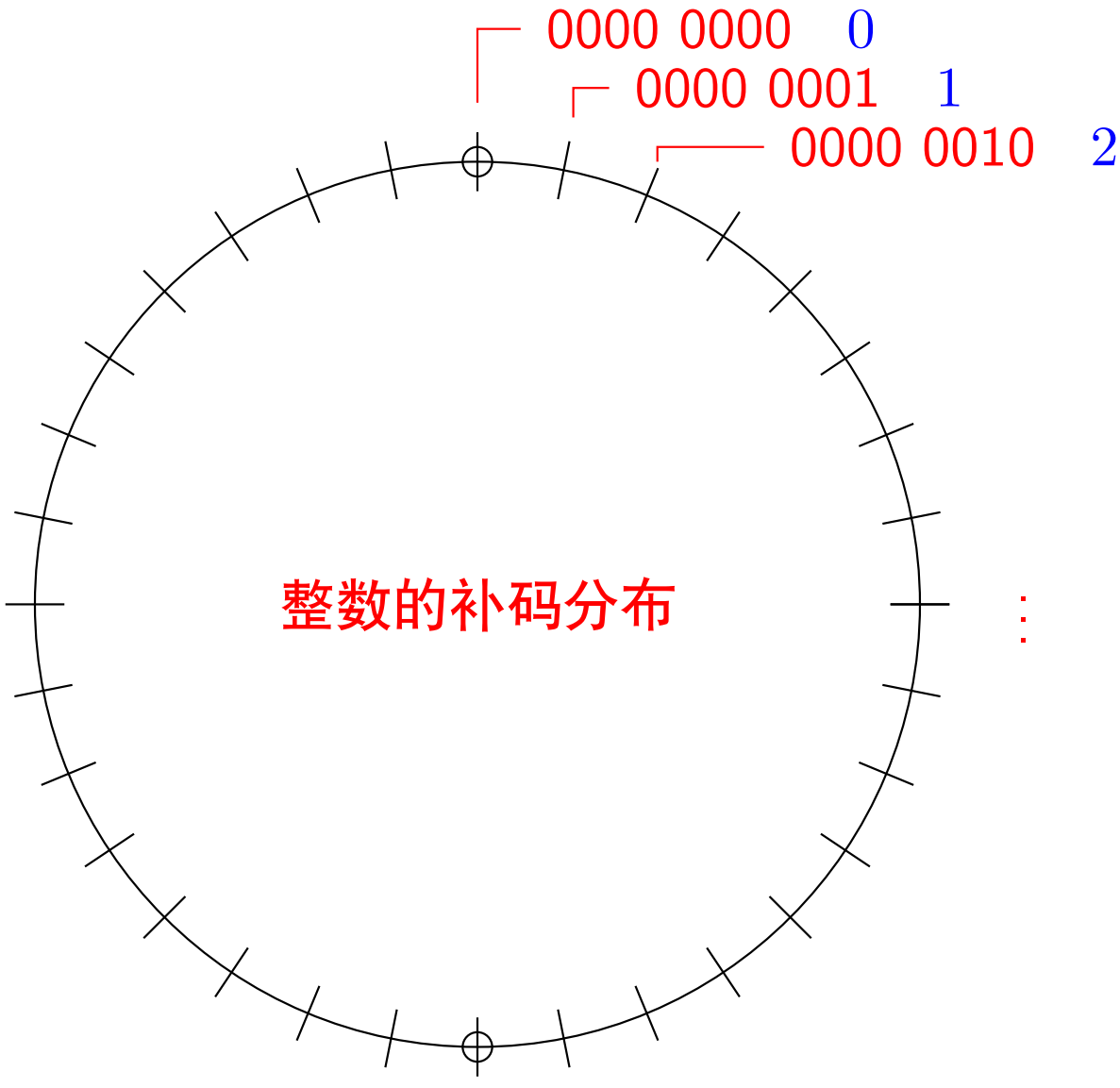
补码图解 - n 为比特总数



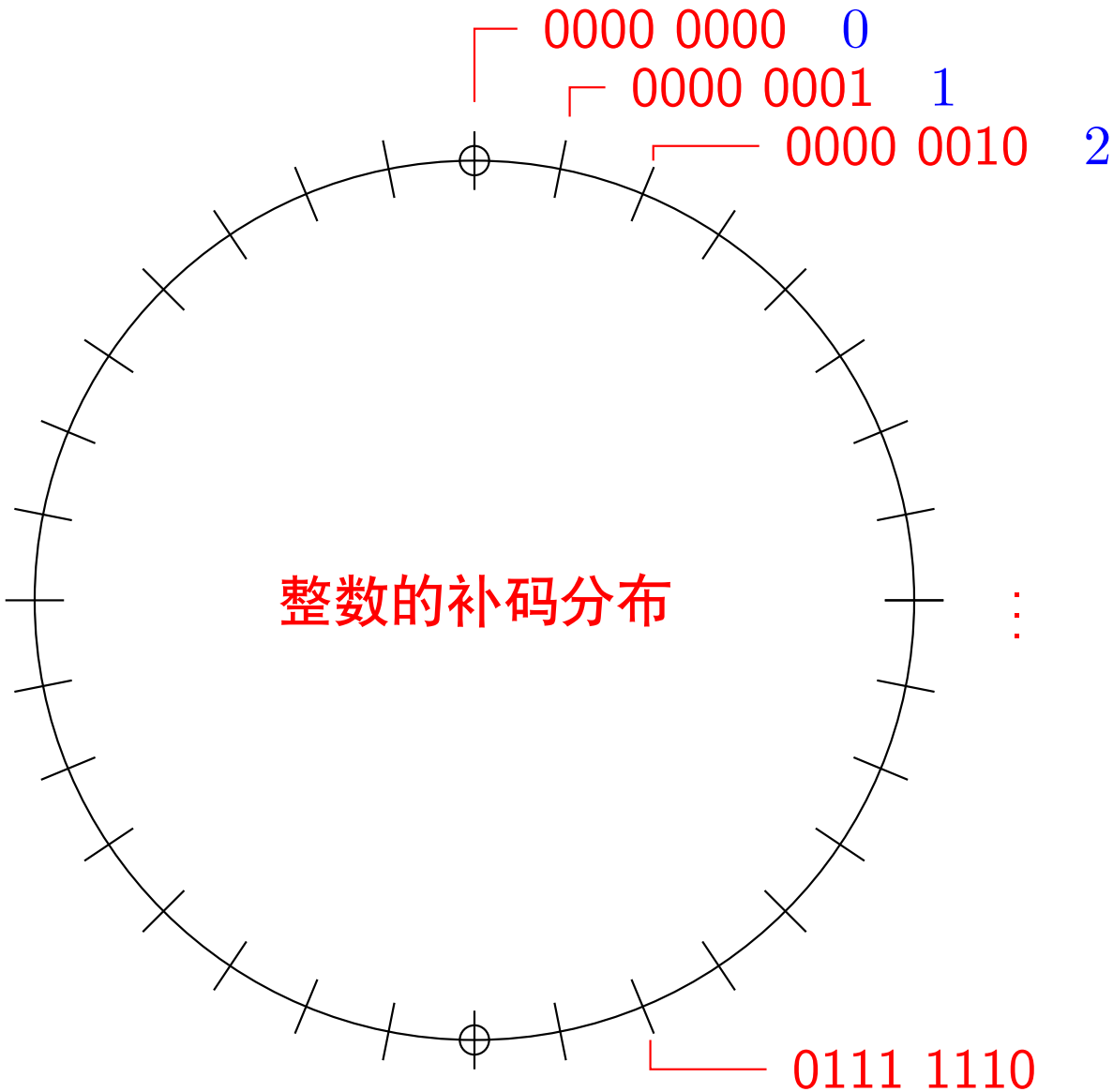
补码图解 - n 为比特总数



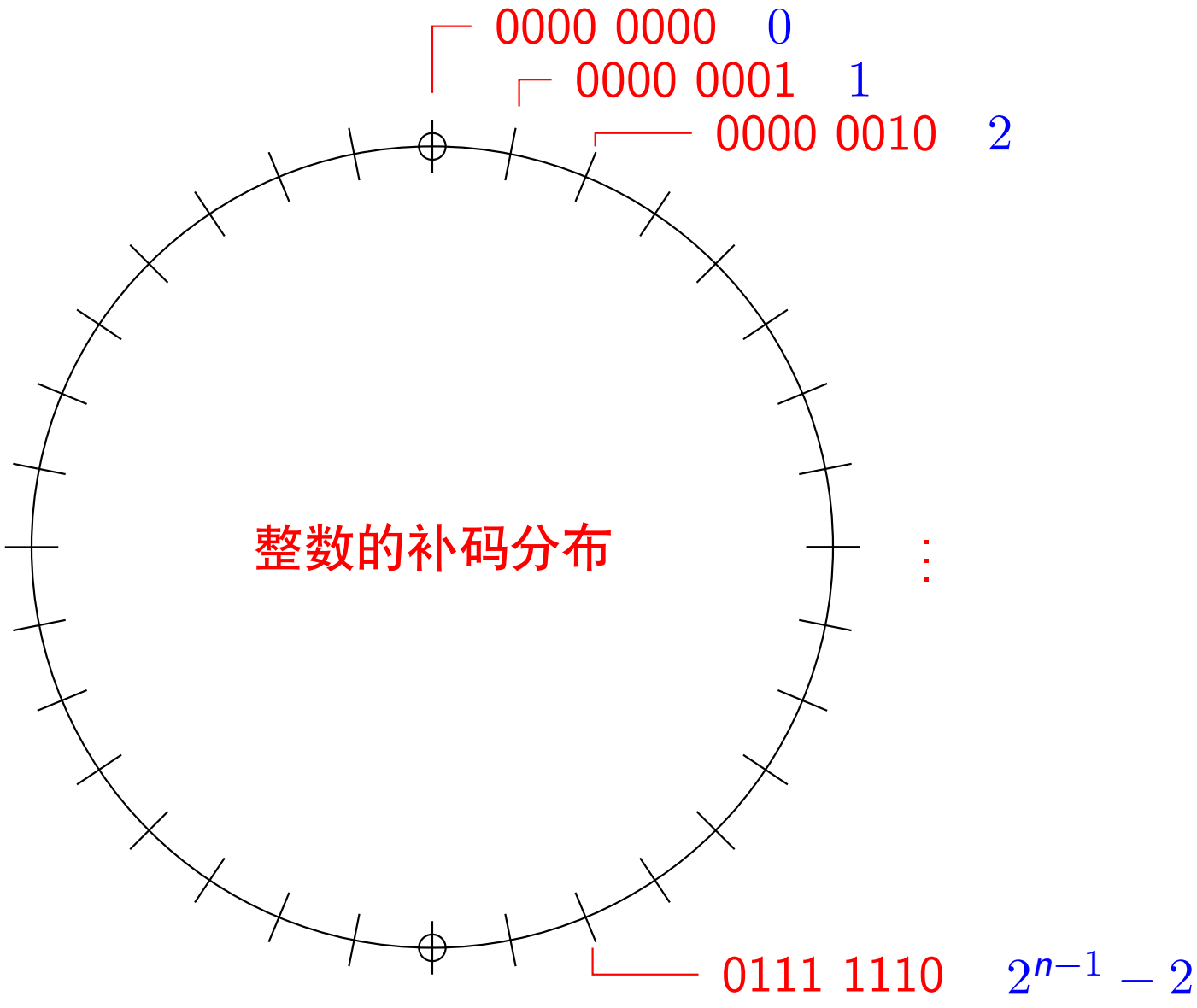
补码图解 - n 为比特总数



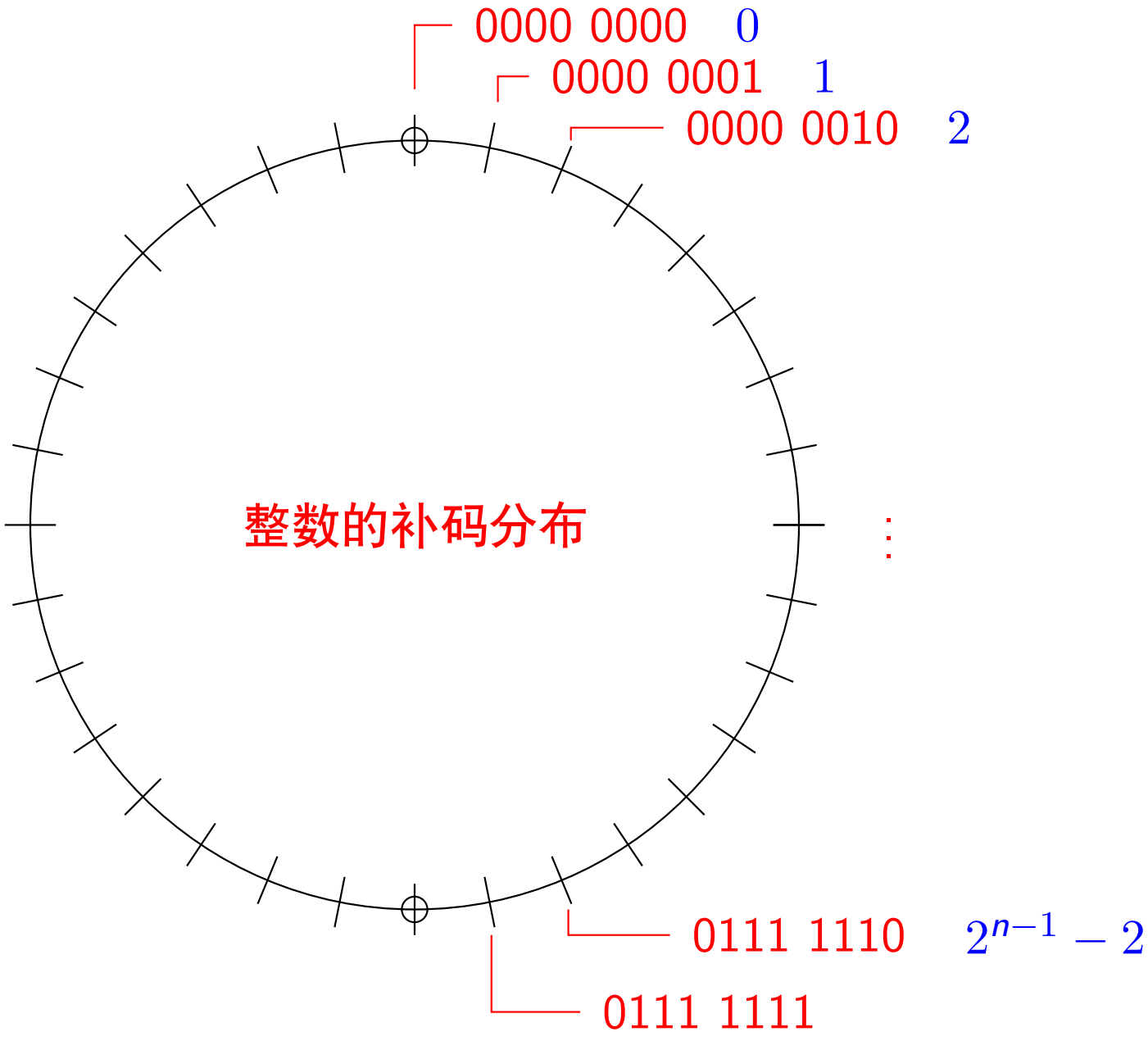
补码图解 - n 为比特总数



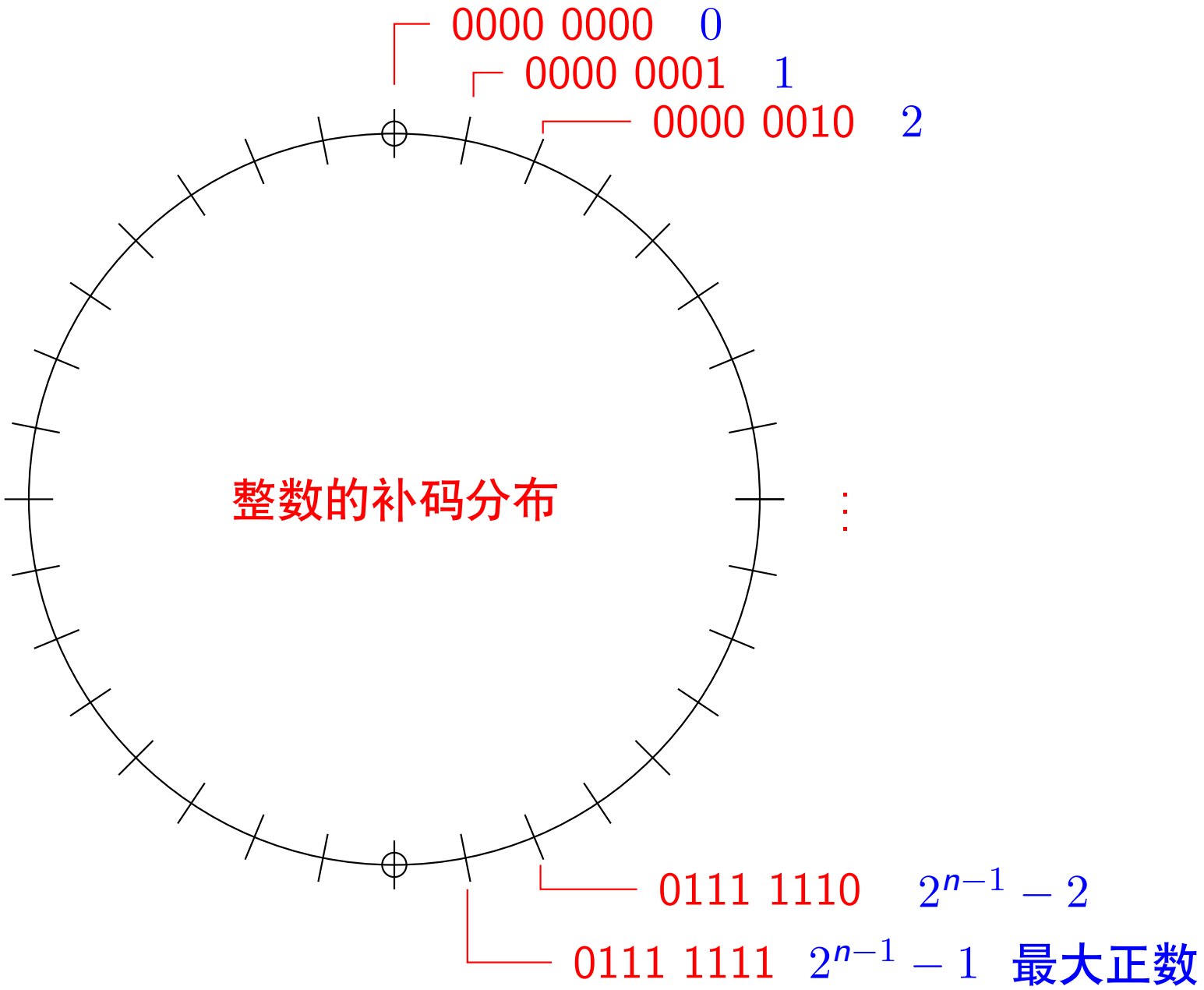
补码图解 - n 为比特总数



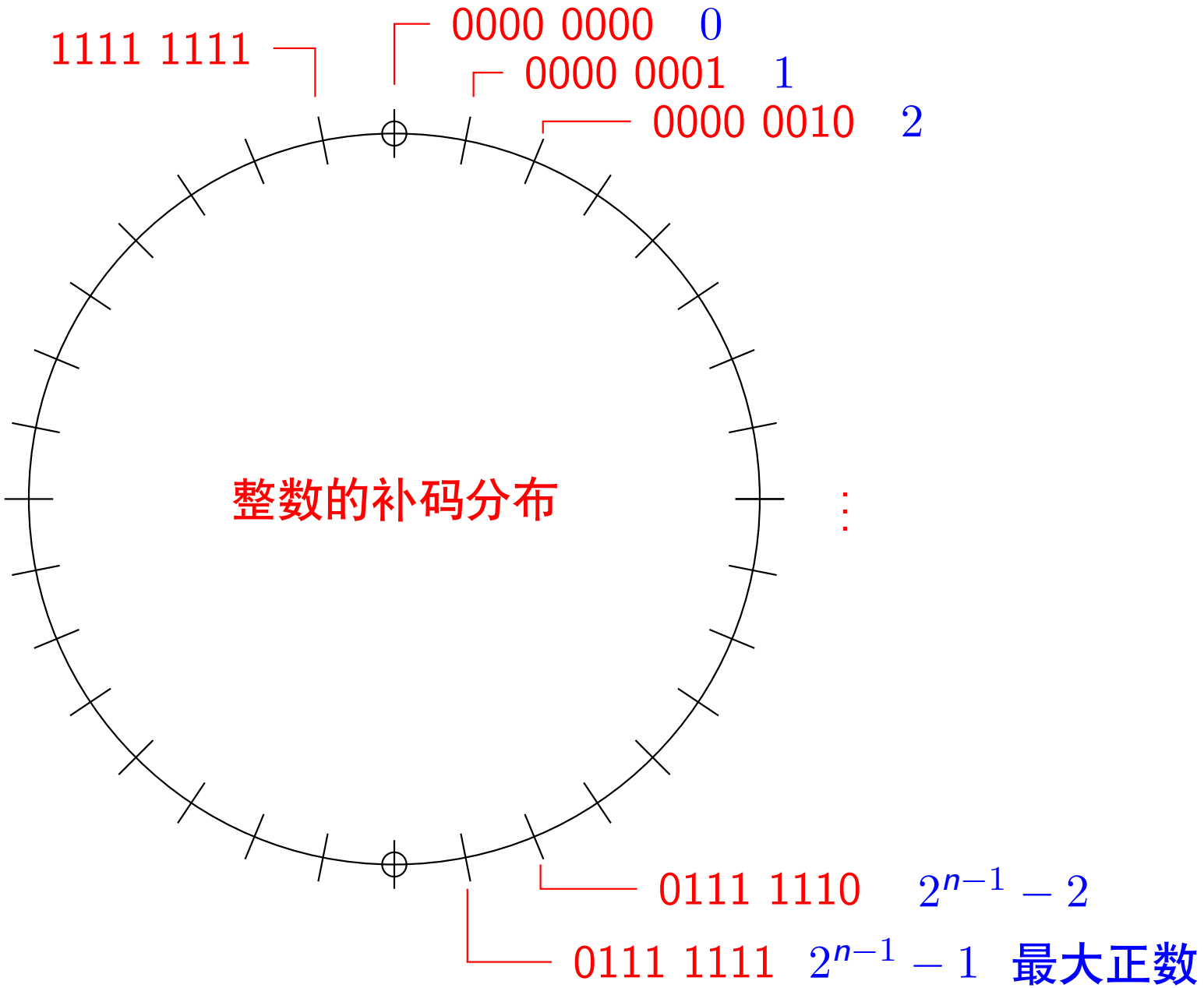
补码图解 - n 为比特总数


0000 0000 0
0000 0001 1
0000 0010 2
整数的补码分布
0111 1110 $2^{n-1}-2$
0111 1111

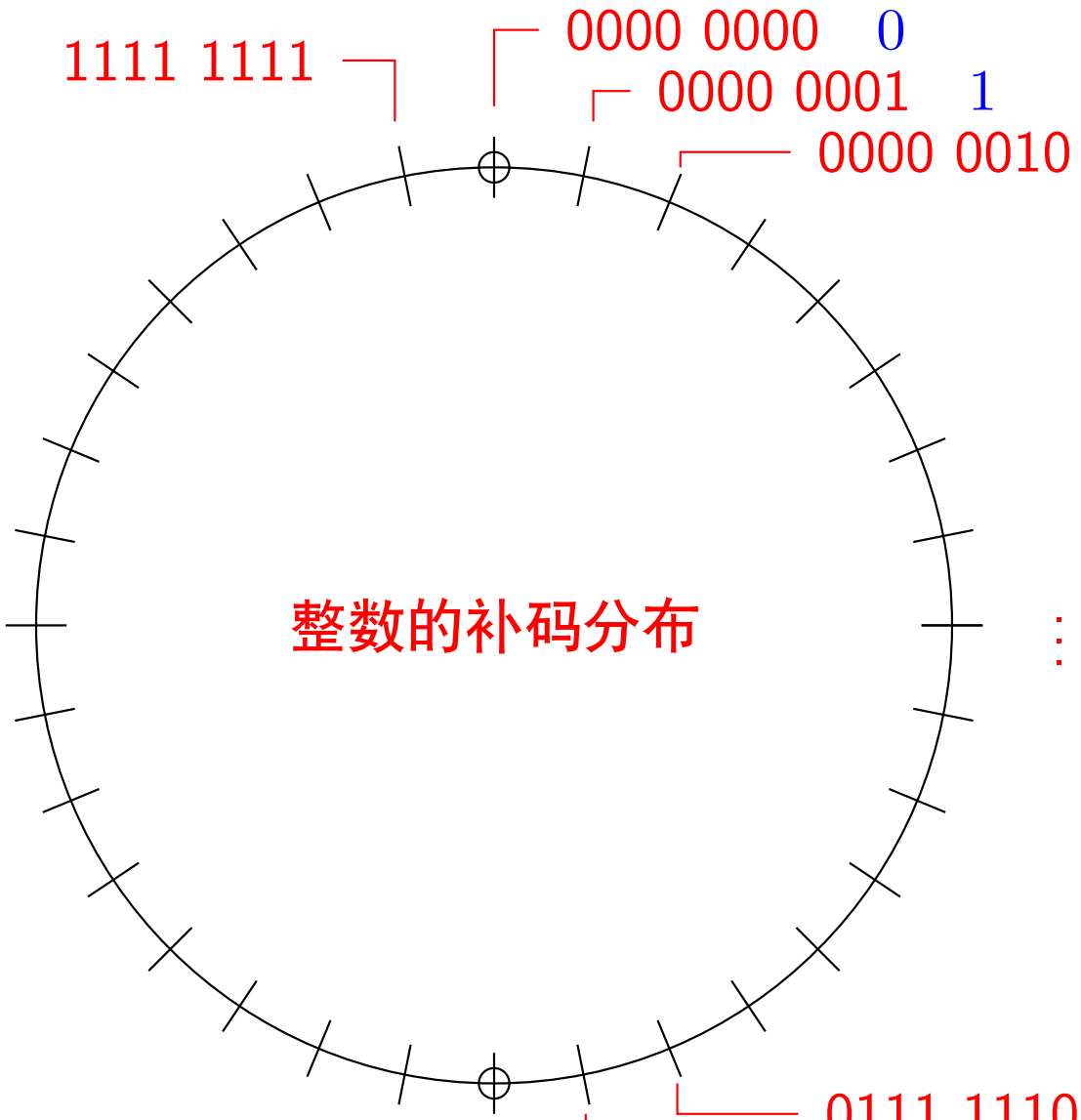
补码图解 - n 为比特总数


0000 0000 0
0000 0001 1
0000 0010 2
整数的补码分布
0111 1110 $2^{n-1}-2$
0111 1111 $2^{n-1}-1$ 最大正数

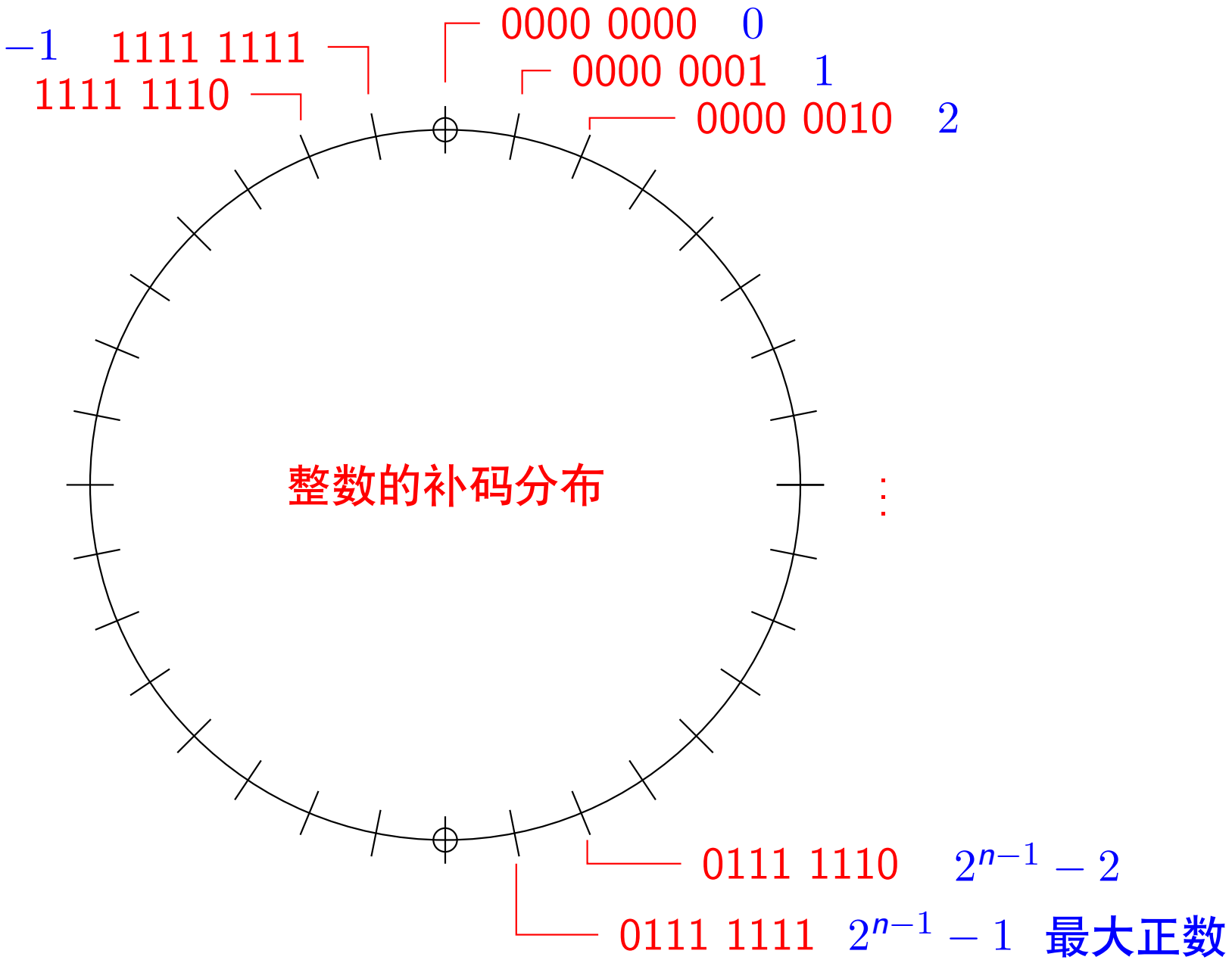
补码图解 - n 为比特总数


1111 1111
0000 0000 0
0000 0001 1
0000 0010 2
整数的补码分布
0111 1110 $2^{n-1}-2$
0111 1111 $2^{n-1}-1$ 最大正数

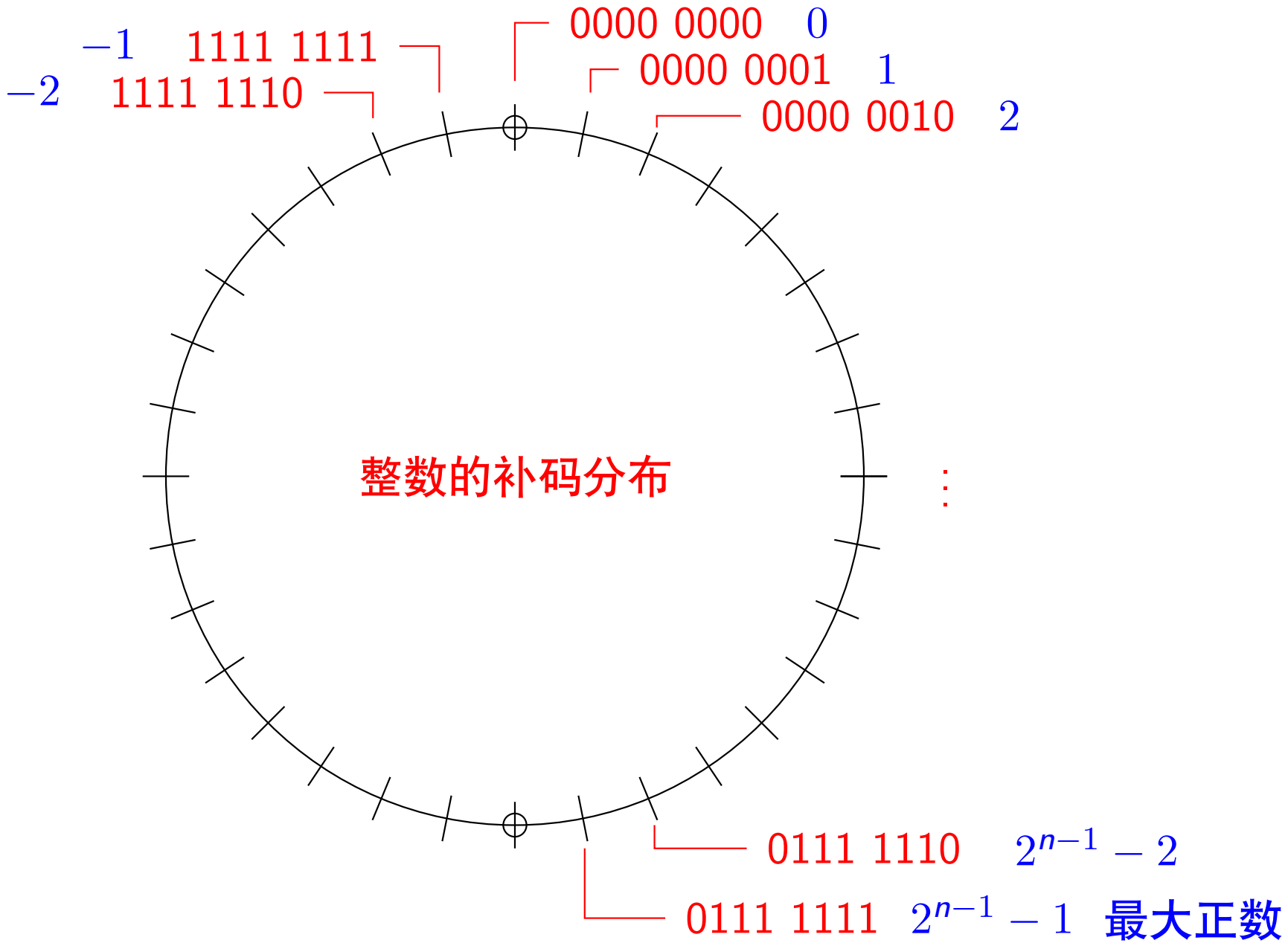
补码图解 - n 为比特总数


−1 1111 1111
0000 0000 0
0000 0001 1
0000 0010 2
整数的补码分布
0111 1110 $2^{n-1}-2$
0111 1111 $2^{n-1}-1$ 最大正数

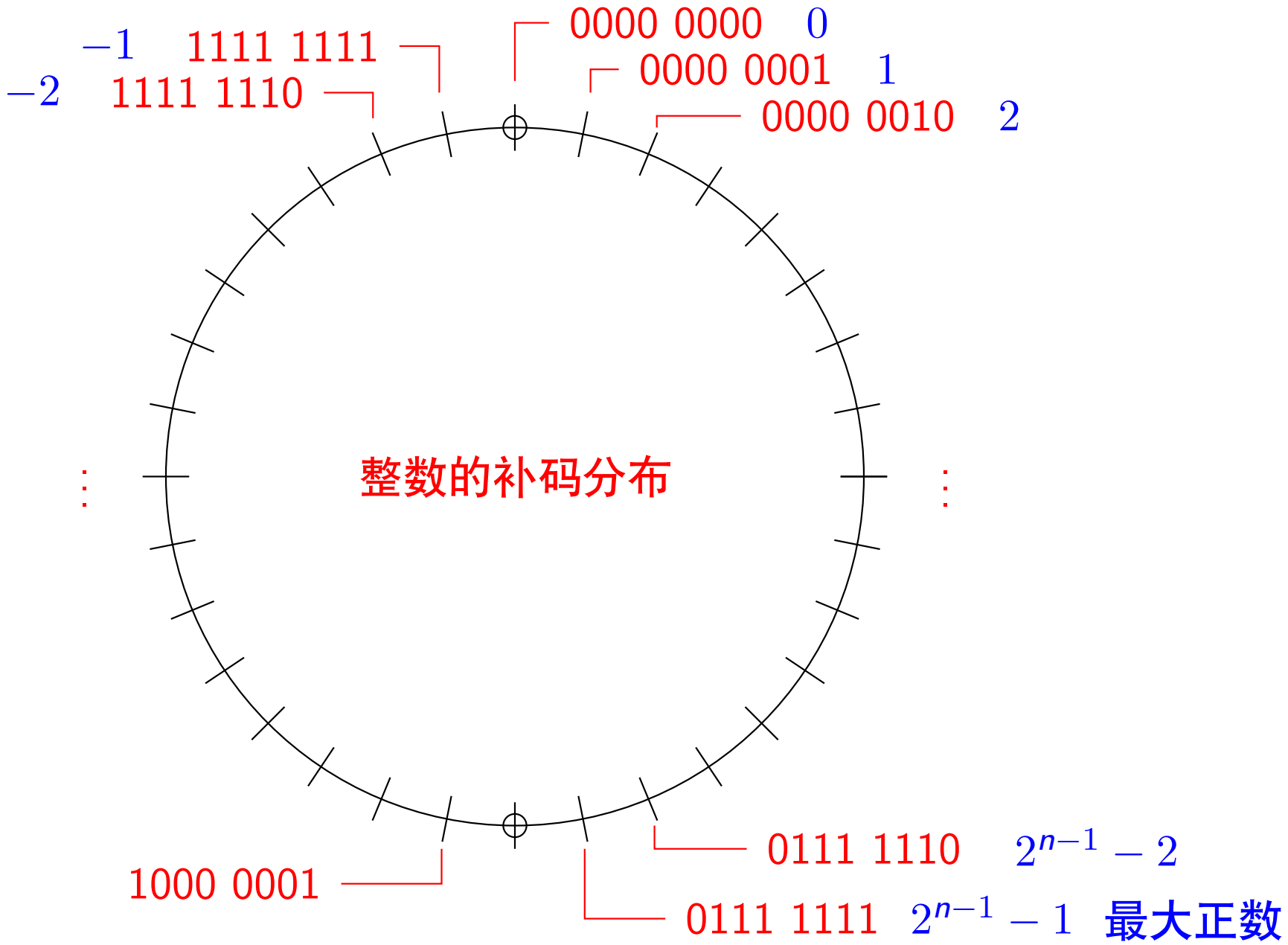
补码图解 - n 为比特总数


−1 1111 1111
1111 1110
0000 0000 0
0000 0001 1
0000 0010 2
整数的补码分布
0111 1110 $2^{n-1}-2$
0111 1111 $2^{n-1}-1$ 最大正数

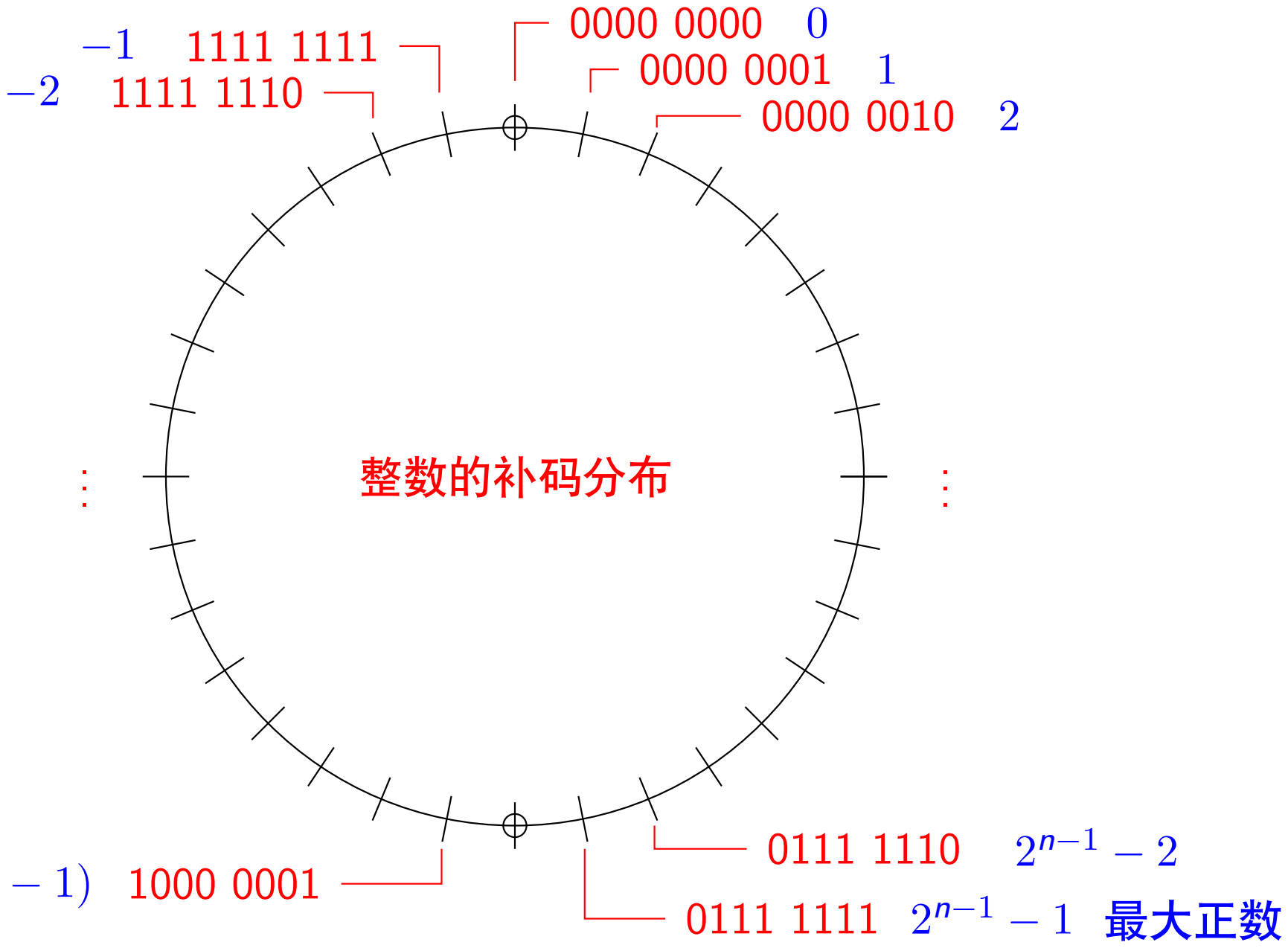
补码图解 - n 为比特总数


0000 0000 0
0000 0001 1
0000 0010 2
−1 1111 1111
−2 1111 1110
整数的补码分布
0111 1110 $2^{n-1}-2$
0111 1111 $2^{n-1}-1$ 最大正数

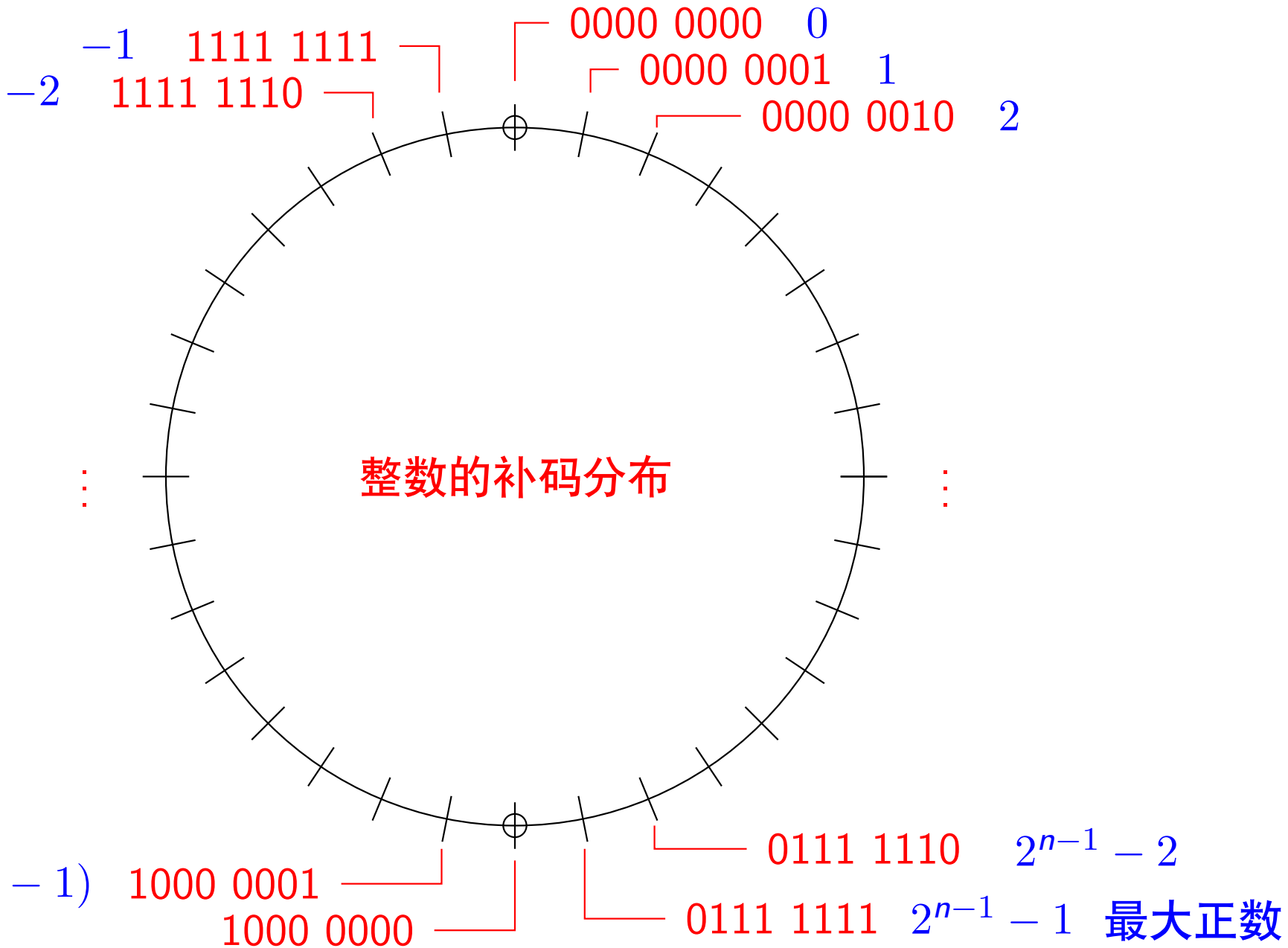
补码图解 - n 为比特总数


0000 0000 0
−1 1111 1111
0000 0001 1
−2 1111 1110
0000 0010 2
整数的补码分布
0111 1110 $2^{n-1}-2$
1000 0001
0111 1111 $2^{n-1}-1$ 最大正数

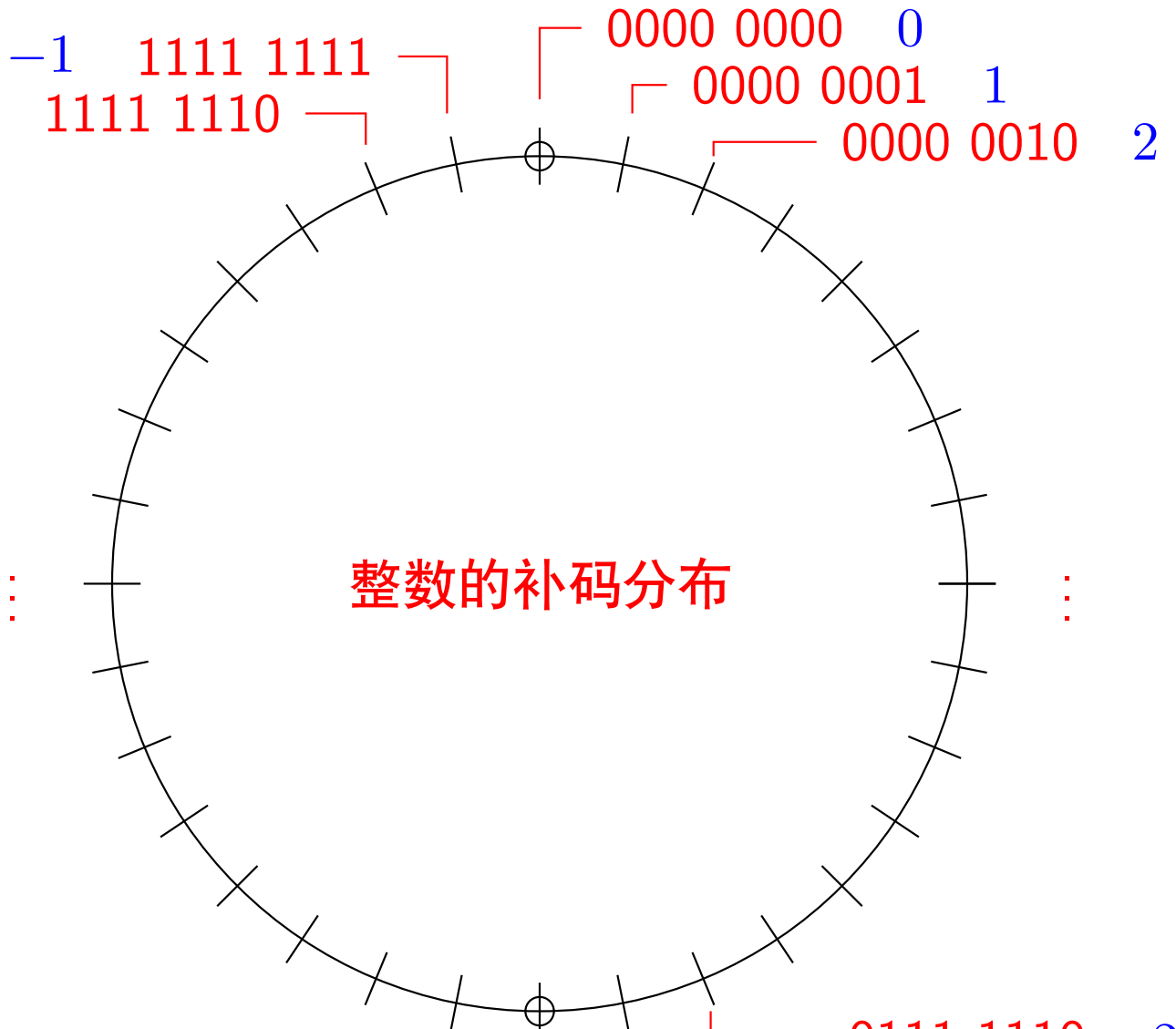
补码图解 - n 为比特总数


0000 0000 0
0000 0001 1
0000 0010 2
−1 1111 1111
−2 1111 1110
整数的补码分布
0111 1110 $2^{n-1}-2$
0111 1111 $2^{n-1}-1$ 最大正数
$-(2^{n-1}-1)$ 1000 0001

补码图解 - n 为比特总数


0000 0000 0
0000 0001 1
0000 0010 2
−1 1111 1111
−2 1111 1110
整数的补码分布
0111 1110 $2^{n-1}-2$
0111 1111 $2^{n-1}-1$ 最大正数
1000 0000
1000 0001 $-(2^{n-1}-1)$

补码图解 - n 为比特总数


0000 0000 0
0000 0001 1
0000 0010 2
−1 1111 1111
−2 1111 1110
整数的补码分布
0111 1110 $2^{n-1}-2$
0111 1111 $2^{n-1}-1$ 最大正数
$-(2^{n-1}-1)$ 1000 0001
-2^{n-1} 1000 0000

整数类型的表示范围

用 N 个 bit 表示整数

- 最大的正整数是：$2^{N-1}-1$

整数类型的表示范围

用 N 个 bit 表示整数

- 最大的正整数是：$2^{N-1}-1$
- 最小的负整数是：-2^{N-1}

整数类型的表示范围

用 N 个 bit 表示整数

- 最大的正整数是：$2^{N-1}-1$
- 最小的负整数是：-2^{N-1}
- 被表示的整数总数为：正整数 $+0+$ 负整数

$$2^{N-1}-1 \quad + \quad 1 \quad + \quad 2^{N-1} = 2^N$$

整数类型的表示范围

用 N 个 bit 表示整数

- 最大的正整数是：$2^{N-1}-1$
- 最小的负整数是：-2^{N-1}
- 被表示的整数总数为：正整数 $+0+$ 负整数

$$2^{N-1}-1 \quad + \quad 1 \quad + \quad 2^{N-1} = 2^N$$

没有浪费任何 bit，因为 N 个 bit 最多只能表示 2^N 个不同的状态

无符号整数

无符号整数：没有符号位的整数

- 没有符号位，全部用来存储正整数的原码

无符号整数：没有符号位的整数

- 没有符号位，全部用来存储正整数的原码
  - 正整数的原码和补码是一样的

无符号整数

无符号整数：没有符号位的整数

- 没有符号位，全部用来存储正整数的原码
  - 正整数的原码和补码是一样的
- N 个 bit 的无符号整数的表示范围：$[0,\ 2^N - 1]$

无符号整数

无符号整数：没有符号位的整数

- 没有符号位，全部用来存储正整数的原码
  - 正整数的原码和补码是一样的
- N 个 bit 的无符号整数的表示范围：$[0,\ 2^N - 1]$

  一共表示了 2^N 个整数

二进制的小数

将 6.625 转化成二进制

- 首先将整数部分转化成二进制：$6 = 110$ （除以 2 取余数）

二进制的小数

将 6.625 转化成二进制

- 首先将整数部分转化成二进制：$6 = 110$ （除以 2 取余数）
- 然后将小数部分 0.625 转化为二进制：（乘以 2 取整数）

二进制的小数

将 6.625 转化成二进制

- 首先将整数部分转化成二进制：$6 = 110$ （除以 2 取余数）
- 然后将小数部分 0.625 转化为二进制：（乘以 2 取整数）
  - $0.625 * 2 = 1.25$, 将整数部分 1 添加到小数后面，得到 110.1

二进制的小数

将 6.625 转化成二进制

- 首先将整数部分转化成二进制：$6 = 110$ （除以 2 取余数）
- 然后将小数部分 0.625 转化为二进制：（乘以 2 取整数）
  - $0.625 * 2 = 1.25$, 将整数部分 1 添加到小数后面，得到 110.1
  - $0.25 * 2 = 0.5$, 将整数部分 0 添加到小数后面，得到 110.10

二进制的小数

将 6.625 转化成二进制

- 首先将整数部分转化成二进制：$6 = 110$ （除以 2 取余数）
- 然后将小数部分 0.625 转化为二进制：（乘以 2 取整数）
  - $0.625 * 2 = 1.25$, 将整数部分 1 添加到小数后面，得到 110.1
  - $0.25 * 2 = 0.5$, 将整数部分 0 添加到小数后面，得到 110.10
  - $0.5 * 2 = 1.0$, 将整数部分 1 添加到小数后面，得到 110.101

二进制的小数

将 6.625 转化成二进制

- 首先将整数部分转化成二进制：$6 = 110$ （除以 2 取余数）
- 然后将小数部分 0.625 转化为二进制：（乘以 2 取整数）
  - $0.625 * 2 = 1.25$, 将整数部分 1 添加到小数后面，得到 110.1
  - $0.25 * 2 = 0.5$, 将整数部分 0 添加到小数后面，得到 110.10
  - $0.5 * 2 = 1.0$, 将整数部分 1 添加到小数后面，得到 110.101
  - 小数部分为 0，结束。

二进制的小数

将 6.625 转化成二进制

- 首先将整数部分转化成二进制：$6 = 110$ （除以 2 取余数）
- 然后将小数部分 0.625 转化为二进制：（乘以 2 取整数）
  - $0.625 * 2 = 1.25$, 将整数部分 1 添加到小数后面，得到 110.1
  - $0.25 * 2 = 0.5$, 将整数部分 0 添加到小数后面，得到 110.10
  - $0.5 * 2 = 1.0$, 将整数部分 1 添加到小数后面，得到 110.101
  - 小数部分为 0，结束。

二进制的小数

将 6.625 转化成二进制

- 首先将整数部分转化成二进制：$6 = 110$ （除以 2 取余数）
- 然后将小数部分 0.625 转化为二进制：（乘以 2 取整数）
  - $0.625 * 2 = 1.25$, 将整数部分 1 添加到小数后面，得到 110.1
  - $0.25 * 2 = 0.5$, 将整数部分 0 添加到小数后面，得到 110.10
  - $0.5 * 2 = 1.0$, 将整数部分 1 添加到小数后面，得到 110.101
  - 小数部分为 0，结束。

将 0.7 转化成二进制

- $0.7 * 2 = 1.4$, 将整数部分 1 添加到小数后面，得到 0.1

二进制的小数

将 6.625 转化成二进制

- 首先将整数部分转化成二进制：$6 = 110$ （除以 2 取余数）
- 然后将小数部分 0.625 转化为二进制：（乘以 2 取整数）
  - $0.625 * 2 = 1.25$, 将整数部分 1 添加到小数后面，得到 110.1
  - $0.25 * 2 = 0.5$, 将整数部分 0 添加到小数后面，得到 110.10
  - $0.5 * 2 = 1.0$, 将整数部分 1 添加到小数后面，得到 110.101
  - 小数部分为 0，结束。

将 0.7 转化成二进制

- $0.7 * 2 = 1.4$, 将整数部分 1 添加到小数后面，得到 0.1
- $0.4 * 2 = 0.8$, 将整数部分 0 添加到小数后面，得到 0.10

二进制的小数

将 6.625 转化成二进制

- 首先将整数部分转化成二进制：$6 = 110$ （除以 2 取余数）
- 然后将小数部分 0.625 转化为二进制：（乘以 2 取整数）
  - $0.625 * 2 = 1.25$, 将整数部分 1 添加到小数后面，得到 110.1
  - $0.25 * 2 = 0.5$, 将整数部分 0 添加到小数后面，得到 110.10
  - $0.5 * 2 = 1.0$, 将整数部分 1 添加到小数后面，得到 110.101
  - 小数部分为 0，结束。

将 0.7 转化成二进制

- $0.7 * 2 = 1.4$, 将整数部分 1 添加到小数后面，得到 0.1
- $0.4 * 2 = 0.8$, 将整数部分 0 添加到小数后面，得到 0.10
- $0.8 * 2 = 1.6$, 将整数部分 1 添加到小数后面，得到 0.101

二进制的小数

将 6.625 转化成二进制

- 首先将整数部分转化成二进制：$6 = 110$ （除以 2 取余数）
- 然后将小数部分 0.625 转化为二进制：（乘以 2 取整数）
  - $0.625 * 2 = 1.25$, 将整数部分 1 添加到小数后面，得到 110.1
  - $0.25 * 2 = 0.5$, 将整数部分 0 添加到小数后面，得到 110.10
  - $0.5 * 2 = 1.0$, 将整数部分 1 添加到小数后面，得到 110.101
  - 小数部分为 0，结束。

将 0.7 转化成二进制

- $0.7 * 2 = 1.4$, 将整数部分 1 添加到小数后面，得到 0.1
- $0.4 * 2 = 0.8$, 将整数部分 0 添加到小数后面，得到 0.10
- $0.8 * 2 = 1.6$, 将整数部分 1 添加到小数后面，得到 0.101
- $0.6 * 2 = 1.2$, 将整数部分 1 添加到小数后面，得到 0.1011

二进制的小数

将 6.625 转化成二进制

- 首先将整数部分转化成二进制：$6 = 110$ （除以 2 取余数）
- 然后将小数部分 0.625 转化为二进制：（乘以 2 取整数）
  - $0.625 * 2 = 1.25$, 将整数部分 1 添加到小数后面，得到 110.1
  - $0.25 * 2 = 0.5$, 将整数部分 0 添加到小数后面，得到 110.10
  - $0.5 * 2 = 1.0$, 将整数部分 1 添加到小数后面，得到 110.101
  - 小数部分为 0，结束。

将 0.7 转化成二进制

- $0.7 * 2 = 1.4$, 将整数部分 1 添加到小数后面，得到 0.1
- $0.4 * 2 = 0.8$, 将整数部分 0 添加到小数后面，得到 0.10
- $0.8 * 2 = 1.6$, 将整数部分 1 添加到小数后面，得到 0.101
- $0.6 * 2 = 1.2$, 将整数部分 1 添加到小数后面，得到 0.1011
- $0.2 * 2 = 0.4$, 将整数部分 0 添加到小数后面，得到 0.10110

二进制的小数

将 6.625 转化成二进制

- 首先将整数部分转化成二进制：$6 = 110$ （除以 2 取余数）
- 然后将小数部分 0.625 转化为二进制：（乘以 2 取整数）
  - $0.625 * 2 = 1.25$, 将整数部分 1 添加到小数后面，得到 110.1
  - $0.25 * 2 = 0.5$, 将整数部分 0 添加到小数后面，得到 110.10
  - $0.5 * 2 = 1.0$, 将整数部分 1 添加到小数后面，得到 110.101
  - 小数部分为 0，结束。

将 0.7 转化成二进制

- $0.7 * 2 = 1.4$, 将整数部分 1 添加到小数后面，得到 0.1
- $0.4 * 2 = 0.8$, 将整数部分 0 添加到小数后面，得到 0.10
- $0.8 * 2 = 1.6$, 将整数部分 1 添加到小数后面，得到 0.101
- $0.6 * 2 = 1.2$, 将整数部分 1 添加到小数后面，得到 0.1011
- $0.2 * 2 = 0.4$, 将整数部分 0 添加到小数后面，得到 0.10110
- 一直进行下去，直到小数为 0，或者达到指定的位数。

实数类型的表示

S	E	M
符号	阶码	尾数
1	8/11	23/52

实数类型的表示

S	E	M

符号	阶码	尾数
1	8/11	23/52

IEEE 浮点数表示：$S \times 2^E \times 1.M$ 或者 $S \times 2^E \times 0.M$

- 符号：1 个 bit，0–正数，1–负数

实数类型的表示

S	E	M
符号	阶码	尾数
1	8/11	23/52

IEEE 浮点数表示：$S \times 2^E \times 1.M$ 或者 $S \times 2^E \times 0.M$

- 符号：1 个 bit，0–正数，1–负数
- 阶码：浮点数的指数，采用移码表示法（无符号的指数）

实数类型的表示

S	E	M
符号	阶码	尾数
1	8/11	23/52

IEEE 浮点数表示：$S \times 2^E \times 1.M$ 或者 $S \times 2^E \times 0.M$

- 符号：1 个 bit，0–正数，1–负数
- 阶码：浮点数的指数，采用移码表示法（无符号的指数）
  - float 浮点数：8 bits，指数 = 阶码 - 127

实数类型的表示

S	E	M
符号	阶码	尾数
1	8/11	23/52

IEEE 浮点数表示：$S \times 2^E \times 1.M$ 或者 $S \times 2^E \times 0.M$

- 符号：1 个 bit，0–正数，1–负数
- 阶码：浮点数的指数，采用移码表示法（无符号的指数）
  - float 浮点数：8 bits，指数 = 阶码 - 127
    $E_{min} = -127, E_{max} = 128$

实数类型的表示

S	E	M
符号	阶码	尾数
1	8/11	23/52

IEEE 浮点数表示：$S \times 2^E \times 1.M$ 或者 $S \times 2^E \times 0.M$

- 符号：1 个 bit，0–正数，1–负数
- 阶码：浮点数的指数，采用移码表示法（无符号的指数）
  - float 浮点数：8 bits，指数 = 阶码 - 127
    $E_{min} = -127, E_{max} = 128$
  - double 浮点数：11 bits，指数 = 阶码 - 1023

实数类型的表示

S	E	M
符号	阶码	尾数
1	8/11	23/52

IEEE 浮点数表示：$S \times 2^E \times 1.M$ 或者 $S \times 2^E \times 0.M$

- 符号：1 个 bit，0–正数，1–负数
- 阶码：浮点数的指数，采用移码表示法（无符号的指数）
  - float 浮点数：8 bits，指数 = 阶码 - 127
    $E_{min} = -127, E_{max} = 128$
  - double 浮点数：11 bits，指数 = 阶码 - 1023
    $E_{min} = -1023, E_{max} = 1024$

实数类型的表示

S	E	M
符号	阶码	尾数
1	8/11	23/52

IEEE 浮点数表示：$S \times 2^E \times 1.M$ 或者 $S \times 2^E \times 0.M$

- 符号：1 个 bit，0–正数，1–负数
- 阶码：浮点数的指数，采用移码表示法（无符号的指数）
  - float 浮点数：8 bits，指数 = 阶码 - 127
    $E_{min} = -127, E_{max} = 128$
  - double 浮点数：11 bits，指数 = 阶码 - 1023
    $E_{min} = -1023, E_{max} = 1024$
- 尾数：二进制的小数

实数类型的表示

S	E	M
符号	阶码	尾数
1	8/11	23/52

IEEE 浮点数表示：$S \times 2^E \times 1.M$ 或者 $S \times 2^E \times 0.M$

- 符号：1 个 bit，0–正数，1–负数
- 阶码：浮点数的指数，采用移码表示法（无符号的指数）
  - float 浮点数：8 bits，指数 = 阶码 - 127
    $E_{min} = -127, E_{max} = 128$
  - double 浮点数：11 bits，指数 = 阶码 - 1023
    $E_{min} = -1023, E_{max} = 1024$
- 尾数：二进制的小数
  - 尾数 = 1.M，当指数段不全为 0 （格式化值）

实数类型的表示

S	E	M
符号	阶码	尾数
1	8/11	23/52

IEEE 浮点数表示：$S \times 2^E \times 1.M$ 或者 $S \times 2^E \times 0.M$

- 符号：1 个 bit，0–正数，1–负数
- 阶码：浮点数的指数，采用移码表示法（无符号的指数）
  - float 浮点数：8 bits，指数 = 阶码 - 127
    $E_{min} = -127, E_{max} = 128$
  - double 浮点数：11 bits，指数 = 阶码 - 1023
    $E_{min} = -1023, E_{max} = 1024$
- 尾数：二进制的小数
  - 尾数 = 1.M，当指数段不全为 0 （格式化值）
  - 尾数 = 0.M，当指数段全为 0 （非格式化值）

特殊的浮点数

指数段	尾数段	数值	注解
有 0 有 1	–	$1.M \times 2^E$	一般情况
全 0	全 0	0	符号位除外，所有 bit 为 0
全 0	不全为 0	$0.M \times 2^{E_{min}}$	非常接近 0 的小数
全 1	全 0	∞	无穷大
全 1	不全为 0	NaN	(Not any Number) 非数值

详细可参考 IEEE_754 浮点数标准

- float 具有 4 个字节（32 个 bit）

特殊的浮点数

指数段	尾数段	数值	注解
有 0 有 1	–	$1.M \times 2^E$	一般情况
全 0	全 0	0	符号位除外，所有 bit 为 0
全 0	不全为 0	$0.M \times 2^{E_{min}}$	非常接近 0 的小数
全 1	全 0	∞	无穷大
全 1	不全为 0	NaN	(Not any Number) 非数值

详细可参考 IEEE_754 浮点数标准

- float 具有 4 个字节（32 个 bit）
- double 具有 8 个字节（64 个 bit）

特殊的浮点数

指数段	尾数段	数值	注解
有 0 有 1	–	$1.M \times 2^E$	一般情况
全 0	全 0	0	符号位除外，所有 bit 为 0
全 0	不全为 0	$0.M \times 2^{E_{min}}$	非常接近 0 的小数
全 1	全 0	∞	无穷大
全 1	不全为 0	NaN	(Not any Number) 非数值

详细可参考 IEEE_754 浮点数标准

- float 具有 4 个字节（32 个 bit）
- double 具有 8 个字节（64 个 bit）
- double 具有更高的精度和更大的表示范围

字符的表示

每个字符用一个字节（8 个 bit）存储 ASCII 码值

字符的表示

每个字符用一个字节（8 个 bit）存储 ASCII 码值

- 等同于 8 bit 长的无符号整数

字符的表示

码值	字符	码值	字符	码值	字符	码值	字符
0 (0x0)	[null]	32 (0x20)	[space]	64 (0x40)	@	96 (0x60)	‘
1 (0x1)	[start heading]	33 (0x21)	!	65 (0x41)	A	97 (0x61)	a
2 (0x2)	[start text]	34 (0x22)	”	66 (0x42)	B	98 (0x62)	b
3 (0x3)	[end text]	35 (0x23)	#	67 (0x43)	C	99 (0x63)	c
4 (0x4)	[end trans.]	36 (0x24)	$	68 (0x44)	D	100 (0x64)	d
5 (0x5)	[enquiry]	37 (0x25)	%	69 (0x45)	E	101 (0x65)	e
6 (0x6)	[ack.]	38 (0x26)	&	70 (0x46)	F	102 (0x66)	f
7 (0x7)	[bell]	39 (0x27)	’	71 (0x47)	G	103 (0x67)	g
8 (0x8)	[backspace]	40 (0x28)	(	72 (0x48)	H	104 (0x68)	h
9 (0x9)	[horiz. tab]	41 (0x29)	)	73 (0x49)	I	105 (0x69)	i
10 (0xA)	[new line]	42 (0x2A)	*	74 (0x4A)	J	106 (0x6A)	j
11 (0xB)	[vert. tab]	43 (0x2B)	+	75 (0x4B)	K	107 (0x6B)	k
12 (0xC)	[new page]	44 (0x2C)	,	76 (0x4C)	L	108 (0x6C)	l
13 (0xD)	[carriage ret.]	45 (0x2D)	-	77 (0x4D)	M	109 (0x6D)	m
14 (0xE)	[shift out]	46 (0x2E)	.	78 (0x4E)	N	110 (0x6E)	n
15 (0xF)	[shift in]	47 (0x2F)	/	79 (0x4F)	O	111 (0x6F)	o
16 (0x10)	[DL escape]	48 (0x30)	0	80 (0x50)	P	112 (0x70)	p
17 (0x11)	[dev. cont. 1]	49 (0x31)	1	81 (0x51)	Q	113 (0x71)	q
18 (0x12)	[dev. cont. 2]	50 (0x32)	2	82 (0x52)	R	114 (0x72)	r
19 (0x13)	[dev. cont. 3]	51 (0x33)	3	83 (0x53)	S	115 (0x73)	s
20 (0x14)	[dev. cont. 4]	52 (0x34)	4	84 (0x54)	T	116 (0x74)	t
21 (0x15)	[neg. ack.]	53 (0x35)	5	85 (0x55)	U	117 (0x75)	u
22 (0x16)	[synch. idle]	54 (0x36)	6	86 (0x56)	V	118 (0x76)	v
23 (0x17)	[end trans. block]	55 (0x37)	7	87 (0x57)	W	119 (0x77)	w
24 (0x18)	[cancel]	56 (0x38)	8	88 (0x58)	X	120 (0x78)	x
25 (0x19)	[end medium]	57 (0x39)	9	89 (0x59)	Y	121 (0x79)	y
26 (0x1A)	[substitute]	58 (0x3A)	:	90 (0x5A)	Z	122 (0x7A)	z
27 (0x1B)	[escape]	59 (0x3B)	;	91 (0x5B)	[	123 (0x7B)	{
28 (0x1C)	[file sep.]	60 (0x3C)	<	92 (0x5C)	\	124 (0x7C)	|
29 (0x1D)	[group sep.]	61 (0x3D)	=	93 (0x5D)	]	125 (0x7D)	}
30 (0x1E)	[record sep.]	62 (0x3E)	>	94 (0x5E)	^	126 (0x7E)	∼
31 (0x1F)	[unit sep.]	63 (0x3F)	?	95 (0x5F)	_	127 (0x7F)	[delete]

基本数据类型总览

类型名称	类型名	长度	取值范围

基本数据类型总览

类型名称	类型名	长度	取值范围
整型	int	32 位	$-2^{31} \sim 2^{31}-1$

基本数据类型总览

类型名称	类型名	长度	取值范围
整型	int	32 位	$-2^{31} \sim 2^{31}-1$
短整型	short [int]	16 位	$-2^{15} \sim 2^{15}-1$

基本数据类型总览

类型名称	类型名	长度	取值范围
整型	int	32 位	$-2^{31} \sim 2^{31}-1$
短整型	short [int]	16 位	$-2^{15} \sim 2^{15}-1$
长整型	long [int]	32 位	$-2^{31} \sim 2^{31}-1$

基本数据类型总览

类型名称	类型名	长度	取值范围
整型	int	32 位	$-2^{31} \sim 2^{31}-1$
短整型	short [int]	16 位	$-2^{15} \sim 2^{15}-1$
长整型	long [int]	32 位	$-2^{31} \sim 2^{31}-1$
无符号整型	unsigned [int]	32 位	$0 \sim 2^{32}-1$

基本数据类型总览

类型名称	类型名	长度	取值范围
整型	int	32 位	$-2^{31} \sim 2^{31}-1$
短整型	short [int]	16 位	$-2^{15} \sim 2^{15}-1$
长整型	long [int]	32 位	$-2^{31} \sim 2^{31}-1$
无符号整型	unsigned [int]	32 位	$0 \sim 2^{32}-1$
无符号短整型	unsigned short [int]	16 位	$0 \sim 2^{16}-1$

基本数据类型总览

类型名称	类型名	长度	取值范围
整型	int	32 位	$-2^{31} \sim 2^{31}-1$
短整型	short [int]	16 位	$-2^{15} \sim 2^{15}-1$
长整型	long [int]	32 位	$-2^{31} \sim 2^{31}-1$
无符号整型	unsigned [int]	32 位	$0 \sim 2^{32}-1$
无符号短整型	unsigned short [int]	16 位	$0 \sim 2^{16}-1$
无符号长整型	unsigned long [int]	32 位	$0 \sim 2^{32}-1$

基本数据类型总览

类型名称	类型名	长度	取值范围
整型	int	32 位	$-2^{31} \sim 2^{31}-1$
短整型	short [int]	16 位	$-2^{15} \sim 2^{15}-1$
长整型	long [int]	32 位	$-2^{31} \sim 2^{31}-1$
无符号整型	unsigned [int]	32 位	$0 \sim 2^{32}-1$
无符号短整型	unsigned short [int]	16 位	$0 \sim 2^{16}-1$
无符号长整型	unsigned long [int]	32 位	$0 \sim 2^{32}-1$
字符型	char	8 位	$0 \sim 255$

基本数据类型总览

类型名称	类型名	长度	取值范围
整型	int	32 位	$-2^{31} \sim 2^{31}-1$
短整型	short [int]	16 位	$-2^{15} \sim 2^{15}-1$
长整型	long [int]	32 位	$-2^{31} \sim 2^{31}-1$
无符号整型	unsigned [int]	32 位	$0 \sim 2^{32}-1$
无符号短整型	unsigned short [int]	16 位	$0 \sim 2^{16}-1$
无符号长整型	unsigned long [int]	32 位	$0 \sim 2^{32}-1$
字符型	char	8 位	$0 \sim 255$
单精度浮点数	float	32 位	约 $\pm[10^{-38}, 10^{38}]$

基本数据类型总览

类型名称	类型名	长度	取值范围
整型	int	32 位	$-2^{31} \sim 2^{31}-1$
短整型	short [int]	16 位	$-2^{15} \sim 2^{15}-1$
长整型	long [int]	32 位	$-2^{31} \sim 2^{31}-1$
无符号整型	unsigned [int]	32 位	$0 \sim 2^{32}-1$
无符号短整型	unsigned short [int]	16 位	$0 \sim 2^{16}-1$
无符号长整型	unsigned long [int]	32 位	$0 \sim 2^{32}-1$
字符型	char	8 位	$0 \sim 255$
单精度浮点数	float	32 位	约 $\pm[10^{-38}, 10^{38}]$
双精度浮点数	double	64 位	约 $\pm[10^{-308}, 10^{308}]$

基本数据类型的常量表示

- 整数常量
- 浮点数常量
- 字符常量

整数常量表示

- 计算机系统内部采用二进制表示

整数常量表示

- 计算机系统内部采用二进制表示
- 但是 C 语言语法不支持用 2 进制表示常量

整数常量表示

- 计算机系统内部采用二进制表示
- 但是 C 语言语法不支持用 2 进制表示常量

C 语言语法支持的常量表示

整数常量表示

- 计算机系统内部采用二进制表示
- 但是 C 语言语法不支持用 2 进制表示常量

C 语言语法支持的常量表示

- 10 进制常量表示

整数常量表示

- 计算机系统内部采用二进制表示
- 但是 C 语言语法不支持用 2 进制表示常量

C 语言语法支持的常量表示

- 10 进制常量表示
  - 由 $+, -$ 符号，$0 \sim 9$ 组成

整数常量表示

- 计算机系统内部采用二进制表示
- 但是 C 语言语法不支持用 2 进制表示常量

C 语言语法支持的常量表示

- 10 进制常量表示
  - 由 $+, -$ 符号，$0 \sim 9$ 组成
  - 不能以 0 开头

整数常量表示

- 计算机系统内部采用二进制表示
- 但是 C 语言语法不支持用 2 进制表示常量

C 语言语法支持的常量表示

- 10 进制常量表示
  - 由 $+, -$ 符号，$0 \sim 9$ 组成
  - 不能以 0 开头
- 8 进制常量表示：以 0 开头，数字 $0 \sim 7$ 组成的常量

整数常量表示

- 计算机系统内部采用二进制表示
- 但是 C 语言语法不支持用 2 进制表示常量

C 语言语法支持的常量表示

- 10 进制常量表示
  - 由 $+, -$ 符号，$0 \sim 9$ 组成
  - 不能以 0 开头
- 8 进制常量表示：以 0 开头，数字 $0 \sim 7$ 组成的常量
- 16 进制常量表示：以 0x 或 0X 开头，数字 $0 \sim 9$、字母 $a \sim f$ 或 $A \sim F$ 组成的常量

8 进制整数表示

由 $+, -$ 符号，$0, 1, 2, \cdots, 7$ 组成

- 必须以 0 开始（和 10 进制表示区分开来了）

8 进制整数表示

由 $+, -$ 符号，$0, 1, 2, \cdots, 7$ 组成

- 必须以 0 开始（和 10 进制表示区分开来了）
- 右起第 k 位的权重是 8^k

$$a_n a_{n-1} a_{n-2} \ldots a_1 a_0 = \sum_{k=0}^{n} a_k 8^k$$

8 进制整数表示

由 $+, -$ 符号，$0, 1, 2, \cdots, 7$ 组成

- 必须以 0 开始（和 10 进制表示区分开来了）
- 右起第 k 位的权重是 8^k

$$a_n a_{n-1} a_{n-2} \ldots a_1 a_0 = \sum_{k=0}^{n} a_k 8^k$$

- C 语言数数的时候都是从 0 开始

8 进制整数表示

由 $+, -$ 符号，$0, 1, 2, \cdots, 7$ 组成

- 必须以 0 开始（和 10 进制表示区分开来了）
- 右起第 k 位的权重是 8^k

$$a_n a_{n-1} a_{n-2} \ldots a_1 a_0 = \sum_{k=0}^{n} a_k 8^k$$

  - C 语言数数的时候都是从 0 开始

几个例子

8 进制整数表示

由 $+, -$ 符号，$0, 1, 2, \cdots, 7$ 组成

- 必须以 0 开始（和 10 进制表示区分开来了）
- 右起第 k 位的权重是 8^k

$$a_n a_{n-1} a_{n-2} \ldots a_1 a_0 = \sum_{k=0}^{n} a_k 8^k$$

  - C 语言数数的时候都是从 0 开始

几个例子

- 010, 10 等于几？

8 进制整数表示

由 $+, -$ 符号，$0, 1, 2, \cdots, 7$ 组成

- 必须以 0 开始（和 10 进制表示区分开来了）
- 右起第 k 位的权重是 8^k

$$a_n a_{n-1} a_{n-2} \ldots a_1 a_0 = \sum_{k=0}^{n} a_k 8^k$$

  - C 语言数数的时候都是从 0 开始

几个例子

- 010, 10 等于几？
- -023 与 23 等于几？

16 进制整数表示

由 $+, -$ 符号，$0, 1, 2, \cdots, 9, A, B, C, D, E, F$ 组成

- 必须以 0X 开始（以便与 8 进制，10 进制表示区分开来了）

16 进制整数表示

由 $+, -$ 符号，$0, 1, 2, \cdots, 9, A, B, C, D, E, F$ 组成

- 必须以 0X 开始（以便与 8 进制，10 进制表示区分开来了）
- 右起第 k 位的权重是 16^k

$$a_n a_{n-1} a_{n-2} \ldots a_1 a_0 = \sum_{k=0}^{n} a_k 16^k$$

16 进制整数表示

由 $+, -$ 符号，$0, 1, 2, \cdots, 9, A, B, C, D, E, F$ 组成

- 必须以 0X 开始（以便与 8 进制，10 进制表示区分开来了）
- 右起第 k 位的权重是 16^k

$$a_n a_{n-1} a_{n-2} \ldots a_1 a_0 = \sum_{k=0}^{n} a_k 16^k$$

- 字母也可以小写：x, a, b, c, d, e, f

16 进制整数表示

由 $+, -$ 符号，$0, 1, 2, \cdots, 9, A, B, C, D, E, F$ 组成

- 必须以 0X 开始（以便与 8 进制，10 进制表示区分开来了）
- 右起第 k 位的权重是 16^k

$$a_n a_{n-1} a_{n-2} \ldots a_1 a_0 = \sum_{k=0}^{n} a_k 16^k$$

- 字母也可以小写：x, a, b, c, d, e, f

几个例子

16 进制整数表示

由 $+, -$ 符号，$0, 1, 2, \cdots, 9, A, B, C, D, E, F$ 组成

- 必须以 0X 开始（以便与 8 进制，10 进制表示区分开来了）
- 右起第 k 位的权重是 16^k

$$a_n a_{n-1} a_{n-2} \ldots a_1 a_0 = \sum_{k=0}^{n} a_k 16^k$$

- 字母也可以小写：x, a, b, c, d, e, f

几个例子

- 016, 16, 0x16 等于几？

16 进制整数表示

由 $+, -$ 符号，$0, 1, 2, \cdots, 9, A, B, C, D, E, F$ 组成

- 必须以 0X 开始（以便与 8 进制，10 进制表示区分开来了）
- 右起第 k 位的权重是 16^k

$$a_n a_{n-1} a_{n-2} \ldots a_1 a_0 = \sum_{k=0}^{n} a_k 16^k$$

- 字母也可以小写：x, a, b, c, d, e, f

几个例子

- 016, 16, 0x16 等于几？
- 013, -16, -0x12 等于几？

16 进制整数表示

由 $+, -$ 符号，$0, 1, 2, \cdots, 9, A, B, C, D, E, F$ 组成

- 必须以 0X 开始（以便与 8 进制，10 进制表示区分开来了）
- 右起第 k 位的权重是 16^k

$$a_n a_{n-1} a_{n-2} \ldots a_1 a_0 = \sum_{k=0}^{n} a_k 16^k$$

- 字母也可以小写：x, a, b, c, d, e, f

几个例子

- 016, 16, 0x16 等于几？
- 013, -16, -0x12 等于几？
- 0XFFFF 是什么？

扩展的整数类型

采用后缀的方式表示扩展的整数类型

无符号	U	u	unsigned
长整型	L	l	long
无符号长整型	UL/LU	ul/lu	unsigned long

举几个例子

扩展的整数类型

采用后缀的方式表示扩展的整数类型

无符号	U	u	unsigned
长整型	L	l	long
无符号长整型	UL/LU	ul/lu	unsigned long

举几个例子

- $-12L$、$-123456789L$、$12u$、$034U$

扩展的整数类型

采用后缀的方式表示扩展的整数类型

无符号	U	u	unsigned
长整型	L	l	long
无符号长整型	UL/LU	ul/lu	unsigned long

举几个例子

- $-12L$、$-123456789L$、$12u$、$034U$
- $12LU$、$0X567LU$

扩展的整数类型

采用后缀的方式表示扩展的整数类型

无符号	U	u	unsigned
长整型	L	l	long
无符号长整型	UL/LU	ul/lu	unsigned long

举几个例子

- $-12L$、$-123456789L$、$12u$、$034U$
- $12LU$、$0X567LU$
- 如果定义：short n = 123456789; 那么 n=?

扩展的整数类型

采用后缀的方式表示扩展的整数类型

无符号	U	u	unsigned
长整型	L	l	long
无符号长整型	UL/LU	ul/lu	unsigned long

举几个例子

- $-12L$、$-123456789L$、$12u$、$034U$
- $12LU$、$0X567LU$
- 如果定义：short n = 123456789; 那么 n=?
  - n = -13035, why?

整数溢出的处理

- 123456789 = 0x75bcd15

整数溢出的处理

- 123456789 = 0x75bcd15
- 超出 short 部分被截断，留下低位的 2 个字节 (short 是 2 个字节)

整数溢出的处理

- 123456789 = 0x75bcd15
- 超出 short 部分被截断，留下低位的 2 个字节 (short 是 2 个字节)
- 留下来的内容是 cd15 = 1100 1101 0001 0101，即存储在 n 中的内容

整数溢出的处理

- 123456789 = 0x75bcd15
- 超出 short 部分被截断，留下低位的 2 个字节 (short 是 2 个字节)
- 留下来的内容是 cd15 = 1100 1101 0001 0101，即存储在 n 中的内容

那么：n= ？

整数溢出的处理

- 123456789 = 0x75bcd15
- 超出 short 部分被截断，留下低位的 2 个字节 (short 是 2 个字节)
- 留下来的内容是 cd15 = 1100 1101 0001 0101，即存储在 n 中的内容

那么：n= ?

- 第一个 bit 值为 1，是符号位，所以 n 的值是负的

整数溢出的处理

- 123456789 = 0x75bcd15
- 超出 short 部分被截断，留下低位的 2 个字节 (short 是 2 个字节)
- 留下来的内容是 cd15 = 1100 1101 0001 0101，即存储在 n 中的内容

那么：n= ?

- 第一个 bit 值为 1，是符号位，所以 n 的值是负的
- 数值部分的反码为：100 1101 0001 0100,

整数溢出的处理

- 123456789 = 0x75bcd15
- 超出 short 部分被截断，留下低位的 2 个字节 (short 是 2 个字节)
- 留下来的内容是 cd15 = 1100 1101 0001 0101，即存储在 n 中的内容

那么：n= ?

- 第一个 bit 值为 1，是符号位，所以 n 的值是负的
- 数值部分的反码为：100 1101 0001 0100,
- 原码为：011 0010 1110 1011

整数溢出的处理

- 123456789 = 0x75bcd15
- 超出 short 部分被截断，留下低位的 2 个字节 (short 是 2 个字节)
- 留下来的内容是 cd15 = 1100 1101 0001 0101，即存储在 n 中的内容

那么：n= ?

- 第一个 bit 值为 1，是符号位，所以 n 的值是负的
- 数值部分的反码为：100 1101 0001 0100,
- 原码为：011 0010 1110 1011
  - 写成 16 进制：32eb

整数溢出的处理

- 123456789 = 0x75bcd15
- 超出 short 部分被截断，留下低位的 2 个字节 (short 是 2 个字节)
- 留下来的内容是 cd15 = 1100 1101 0001 0101，即存储在 n 中的内容

那么：n= ?

- 第一个 bit 值为 1，是符号位，所以 n 的值是负的
- 数值部分的反码为：100 1101 0001 0100,
- 原码为：011 0010 1110 1011
  - 写成 16 进制：32eb
  - 写成 10 进制：$= 3 \times 16^3 + 2 \times 16^2 + 14 \times 16 + 11 = 13035$

整数溢出的处理

- 123456789 = 0x75bcd15
- 超出 short 部分被截断，留下低位的 2 个字节 (short 是 2 个字节)
- 留下来的内容是 cd15 = 1100 1101 0001 0101，即存储在 n 中的内容

那么：n= ?

- 第一个 bit 值为 1，是符号位，所以 n 的值是负的
- 数值部分的反码为：100 1101 0001 0100,
- 原码为：011 0010 1110 1011
  - 写成 16 进制：32eb
  - 写成 10 进制：$= 3 \times 16^3 + 2 \times 16^2 + 14 \times 16 + 11 = 13035$
- $n = -13035$

整数溢出的处理

直接舍去溢出部分的内容

整数溢出的处理

直接舍去溢出部分的内容

具体步骤：

- 将数值写下来，并转成 16 进制表示 (每个 16 进制数占 4 个 bit，半个字节)

整数溢出的处理

直接舍去溢出部分的内容

具体步骤：

- 将数值写下来，并转成 16 进制表示 (每个 16 进制数占 4 个 bit，半个字节)
- 根据目标类型的所占字节数，舍去高位超出部分的字节

整数溢出的处理

直接舍去溢出部分的内容

具体步骤：

- 将数值写下来，并转成 16 进制表示 (每个 16 进制数占 4 个 bit，半个字节)
- 根据目标类型的所占字节数，舍去高位超出部分的字节
- 保留的内容即为最后的结果。(最后如果需要，转成 10 进制)

整数运算溢出的处理

- 整数的运算是在补码上进行的。

整数运算溢出的处理

- 整数的运算是在补码上进行的。
- 如果运算结果溢出了，那么舍去高位部分（方法同上）

字符类型常量

- 数字字符：'0' '9'

字符类型常量

- 数字字符：'0' '9'
- 小写字母：'a' 'z'

字符类型常量

- 数字字符：'0' '9'
- 小写字母：'a' 'z'
- 大写字母：'A' 'Z'

字符类型常量

- 数字字符：'0' '9'
- 小写字母：'a' 'z'
- 大写字母：'A' 'Z'
- 其他可见字符：'+', '-', ...

字符类型常量

- 数字字符：'0' '9'
- 小写字母：'a' 'z'
- 大写字母：'A' 'Z'
- 其他可见字符：'+', '-', ...
- 见 ASCII 码表

字符类型常量

转义字符 \

字符	含义
\n	换行符
\t	横向跳格符，或制表符
\\	表示反斜杠字符本身 \
\"	双引号字符本身"
\'	单引号字符本身'
\ddd	1-3 位 8 进制数作为 ASCII 码所代表的字符 例如
\xhh	1-2 位 16 进制数作为 ASCII 码所代表的字符

字符类型常量

转义字符 \

字符	含义
\n	换行符
\t	横向跳格符，或制表符
\\	表示反斜杠字符本身 \
\"	双引号字符本身"
\'	单引号字符本身'
\ddd	1-3 位 8 进制数作为 ASCII 码所代表的字符 例如
\xhh	1-2 位 16 进制数作为 ASCII 码所代表的字符

不要和 printf 函数里面% 混起来了

- % 是 printf/scanf 等输入输出函数约定的格式控制字符，是他们特有的

字符类型常量

转义字符 \

字符	含义
\n	换行符
\t	横向跳格符，或制表符
\\	表示反斜杠字符本身 \
\"	双引号字符本身"
\'	单引号字符本身'
\ddd	1-3 位 8 进制数作为 ASCII 码所代表的字符 例如
\xhh	1-2 位 16 进制数作为 ASCII 码所代表的字符

不要和 printf 函数里面% 混起来了

- % 是 printf/scanf 等输入输出函数约定的格式控制字符，是他们特有的
- 在 printf/scanf 等输入输出函数中用%% 表示一个%，与这里用 \\ 表示一个 \ 方法雷同。

实数类型常量

float

- float 只具有 7~8 个有效数字

double

实数类型常量

float

- float 只具有 7∼8 个有效数字
- float 的表示范围约：$\pm(10^{-38} \sim 10^{38})$

double

实数类型常量

float

- float 只具有 7~8 个有效数字
- float 的表示范围约：$\pm(10^{-38} \sim 10^{38})$

double

- double 具有 15~16 个有效数字

实数类型常量

float

- float 只具有 7~8 个有效数字
- float 的表示范围约：$\pm(10^{-38} \sim 10^{38})$

double

- double 具有 15~16 个有效数字
- double 的表示范围约：$\pm(10^{-308} \sim 10^{308})$

数值精度和取值范围

数值精度和取值范围是两个不同的概念

- float x = 1234567.89;

数值精度和取值范围

数值精度和取值范围是两个不同的概念

- float x = 1234567.89;
  - 虽在取值范围内，但无法精确表达。

数值精度和取值范围

数值精度和取值范围是两个不同的概念

- float x = 1234567.89;
  - 虽在取值范围内，但无法精确表达。
  - 因为 float 只有 7 个有效数字

数值精度和取值范围

数值精度和取值范围是两个不同的概念

- float x = 1234567.89;
  - 虽在取值范围内，但无法精确表达。
  - 因为 float 只有 7 个有效数字
- float y = 1.2e55;

数值精度和取值范围

数值精度和取值范围是两个不同的概念

- float x = 1234567.89;
  - 虽在取值范围内，但无法精确表达。
  - 因为 float 只有 7 个有效数字
- float y = 1.2e55;
  - y 的精度要求不高，但超出取值范围。

数值精度和取值范围

数值精度和取值范围是两个不同的概念

- float x = 1234567.89;
  - 虽在取值范围内，但无法精确表达。
  - 因为 float 只有 7 个有效数字
- float y = 1.2e55;
  - y 的精度要求不高，但超出取值范围。
  - 因为 float 最大数约为 10^{38}

数值精度和取值范围

数值精度和取值范围是两个不同的概念

- float x = 1234567.89;
  - 虽在取值范围内，但无法精确表达。
  - 因为 float 只有 7 个有效数字
- float y = 1.2e55;
  - y 的精度要求不高，但超出取值范围。
  - 因为 float 最大数约为 10^{38}
- **并非所有实数都能在计算机中精确表示**

数值精度和取值范围

数值精度和取值范围是两个不同的概念

- float x = 1234567.89;
  - 虽在取值范围内，但无法精确表达。
  - 因为 float 只有 7 个有效数字
- float y = 1.2e55;
  - y 的精度要求不高，但超出取值范围。
  - 因为 float 最大数约为 10^{38}
- 并非所有实数都能在计算机中精确表示
  - 最多表示 2^{32} 个 float

数值精度和取值范围

数值精度和取值范围是两个不同的概念

- float x = 1234567.89;
  - 虽在取值范围内，但无法精确表达。
  - 因为 float 只有 7 个有效数字
- float y = 1.2e55;
  - y 的精度要求不高，但超出取值范围。
  - 因为 float 最大数约为 10^{38}
- 并非所有实数都能在计算机中精确表示
  - 最多表示 2^{32} 个 float
  - 最多表示 2^{64} 个 double

实数常量的表示

- 普通表示：符号 + 整数部分 + 小数点 + 小数部分

实数常量的表示

- 普通表示：符号 + 整数部分 + 小数点 + 小数部分
  - 例如：-12345.678

实数常量的表示

- 普通表示：符号 + 整数部分 + 小数点 + 小数部分
  - 例如：-12345.678
- 科学计数法表示：符号 + 尾数 e/E 指数

实数常量的表示

- 普通表示：符号 + 整数部分 + 小数点 + 小数部分
  - 例如：-12345.678
- 科学计数法表示：符号 + 尾数 e/E 指数
  - 例如：$-1.2345678E5$

实数常量的表示

- 普通表示：符号 + 整数部分 + 小数点 + 小数部分
  - 例如：-12345.678
- 科学计数法表示：符号 + 尾数 e/E 指数
  - 例如：$-1.2345678E5$
  - 一般表示数值很大，或者很小的数。

实数常量的表示

- 普通表示：符号 + 整数部分 + 小数点 + 小数部分
  - 例如：-12345.678
- 科学计数法表示：符号 + 尾数 e/E 指数
  - 例如：$-1.2345678E5$
  - 一般表示数值很大，或者很小的数。
    - 例如普朗克常量 6.026×10^{-27} 可表示为：6.026E-27，或者 60.26E-28

实数常量的表示

- 普通表示：符号 + 整数部分 + 小数点 + 小数部分
  - 例如：-12345.678
- 科学计数法表示：符号 + 尾数 e/E 指数
  - 例如：$-1.2345678E5$
  - 一般表示数值很大，或者很小的数。
    - 例如普朗克常量 6.026×10^{-27} 可表示为：6.026E-27，或者 60.26E-28
- 实型常量的类型都是 double

实数常量的表示

- 普通表示：符号 + 整数部分 + 小数点 + 小数部分
  - 例如：-12345.678
- 科学计数法表示：符号 + 尾数 e/E 指数
  - 例如：$-1.2345678E5$
  - 一般表示数值很大，或者很小的数。
    - 例如普朗克常量 6.026×10^{-27} 可表示为：6.026E-27，或者 60.26E-28
- 实型常量的类型都是 double
- 如需表示 float 类型常量，可用 f 作为后缀。

实数常量的表示

- 普通表示：符号 + 整数部分 + 小数点 + 小数部分
  - 例如：-12345.678
- 科学计数法表示：符号 + 尾数 e/E 指数
  - 例如：$-1.2345678E5$
  - 一般表示数值很大，或者很小的数。
    - 例如普朗克常量 6.026×10^{-27} 可表示为：6.026E-27，或者 60.26E-28
- 实型常量的类型都是 double
- 如需表示 float 类型常量，可用 f 作为后缀。
  - 例如：3.14f, 5f

内容提要

整数的输入/输出

在 C 语言中，输入输出是通过函数完成的

`printf`, `scanf`, `getchar`, `putchar`, 等等

整数的输入/输出

在 C 语言中，输入输出是通过函数完成的

`printf`， `scanf`， `getchar`， `putchar`， 等等

printf 和 scanf 函数格式控制符

整数的输入/输出

在 C 语言中，输入输出是通过函数完成的

`printf`， `scanf`， `getchar`， `putchar`， 等等

printf 和 scanf 函数格式控制符

- 扩展的整数格式

	十进制	八进制	十六进制
int	%d	%o	%x
unsigned	%u	%o	%x
long	%ld	%lo	%lx
unsigned long	%lu	%lo	%lx

整数的输入/输出

在 C 语言中，输入输出是通过函数完成的

`printf`， `scanf`， `getchar`， `putchar`， 等等

printf 和 scanf 函数格式控制符

- 扩展的整数格式

	十进制	八进制	十六进制
int	%d	%o	%x
unsigned	%u	%o	%x
long	%ld	%lo	%lx
unsigned long	%lu	%lo	%lx

- %d - 十进制输入输出一个整数

整数的输入/输出

在 C 语言中，输入输出是通过函数完成的

`printf`， `scanf`， `getchar`， `putchar`， 等等

printf 和 scanf 函数格式控制符

- 扩展的整数格式

	十进制	八进制	十六进制
int	%d	%o	%x
unsigned	%u	%o	%x
long	%ld	%lo	%lx
unsigned long	%lu	%lo	%lx

- %d - 十进制输入输出一个整数
- %u - 十进制输入输出一个无符号整数

整数的输入/输出

在 C 语言中，输入输出是通过函数完成的

`printf`， `scanf`， `getchar`， `putchar`， 等等

printf 和 scanf 函数格式控制符

- 扩展的整数格式

	十进制	八进制	十六进制
int	%d	%o	%x
unsigned	%u	%o	%x
long	%ld	%lo	%lx
unsigned long	%lu	%lo	%lx

- %d - 十进制输入输出一个整数
- %u - 十进制输入输出一个无符号整数
- %o - 八进制输入输出一个整数

整数的输入/输出

在 C 语言中，输入输出是通过函数完成的

`printf`， `scanf`， `getchar`， `putchar`， 等等

printf 和 scanf 函数格式控制符

- 扩展的整数格式

	十进制	八进制	十六进制
int	%d	%o	%x
unsigned	%u	%o	%x
long	%ld	%lo	%lx
unsigned long	%lu	%lo	%lx

- %d - 十进制输入输出一个整数
- %u - 十进制输入输出一个无符号整数
- %o - 八进制输入输出一个整数
- %x - 十六进制输入输出一个整数

整数的输入/输出

以下语句的输出是什么？

```
printf("%d, %o, %x\n", 10, 10, 10);
printf("%d, %d, %d\n", 10, 010, 0x10);
printf("%d, %x\n", 010, 012);
```

整数的输入/输出

以下语句的输出是什么？

```
printf("%d, %o, %x\n", 10, 10, 10);
printf("%d, %d, %d\n", 10, 010, 0x10);
printf("%d, %x\n", 010, 012);
```

输出结果是：

整数的输入/输出

以下语句的输出是什么？

```
printf("%d, %o, %x\n", 10, 10, 10);
printf("%d, %d, %d\n", 10, 010, 0x10);
printf("%d, %x\n", 010, 012);
```

输出结果是：

10, 12, a

整数的输入/输出

以下语句的输出是什么？

```
printf("%d, %o, %x\n", 10, 10, 10);
printf("%d, %d, %d\n", 10, 010, 0x10);
printf("%d, %x\n", 010, 012);
```

输出结果是：

10, 12, a

10, 8, 16

整数的输入/输出

以下语句的输出是什么？

```
printf("%d, %o, %x\n", 10, 10, 10);
printf("%d, %d, %d\n", 10, 010, 0x10);
printf("%d, %x\n", 010, 012);
```

输出结果是：

10, 12, a

10, 8, 16

8, a

整数的输入/输出

以下语句的输出是什么？

```
printf("%d, %o, %x\n", 10, 10, 10);
printf("%d, %d, %d\n", 10, 010, 0x10);
printf("%d, %x\n", 010, 012);
```

输出结果是：

10, 12, a

10, 8, 16

8, a

注意：用%o 和%x 以及%u 只能输出无符号整数

整数的输入/输出

以下语句的输出是什么？

```
printf("%d, %o, %x\n", 10, 10, 10);
printf("%d, %d, %d\n", 10, 010, 0x10);
printf("%d, %x\n", 010, 012);
```

输出结果是：

10, 12, a

10, 8, 16

8, a

注意：用%o 和%x 以及%u 只能输出无符号整数

- printf("%x, %o", -1, -1);

整数的输入/输出

以下语句的输出是什么？

```
printf("%d, %o, %x\n", 10, 10, 10);
printf("%d, %d, %d\n", 10, 010, 0x10);
printf("%d, %x\n", 010, 012);
```

输出结果是：

10, 12, a

10, 8, 16

8, a

注意：用%o 和%x 以及%u 只能输出无符号整数

- printf("%x, %o", -1, -1);
- **输出结果为：** ffffffff, 37777777777

整数的输入/输出

以下语句的输出是什么？

```
printf("%d, %o, %x\n", 10, 10, 10);
printf("%d, %d, %d\n", 10, 010, 0x10);
printf("%d, %x\n", 010, 012);
```

输出结果是：

10, 12, a

10, 8, 16

8, a

注意：用%o 和%x 以及%u 只能输出无符号整数

- printf("%x, %o", -1, -1);
- 输出结果为：ffffffff, 37777777777
- -1 的补码为 32 个 1，采用%o 输出的时候，被解读为无符号的整数。按 16 进制为：ffffffff，按 8 进制为：37777777777

实数 float 和 double 的输入/输出

函数	数据类型	格式	含义
printf	float	%f	以小数形式输出浮点数 (保留 6 位小数)
	float	%e	以指数形式输出浮点数 (小数点前仅有 1 位非零数)
scanf	float	%f	以小数形式或者指数形式输入浮点数
		%e	
	double	%lf	以小数形式或者指数形式输入浮点数
		%le	

输入 double 的时候，一定要用%lf 和%le

printf 函数格式化输出的位数/宽度控制

输出整数时，可以增加宽度控制

- %md　指定输出的宽度为 m (即占据 m 个字符位置，包括符号位)

printf 函数格式化输出的位数/宽度控制

输出整数时，可以增加宽度控制

- %md　指定输出的宽度为 m (即占据 m 个字符位置，包括符号位)
  - 如果数据的实际宽度小于 m，那么左边补空格

printf 函数格式化输出的位数/宽度控制

输出整数时，可以增加宽度控制

- %md　指定输出的宽度为 m (即占据 m 个字符位置，包括符号位)
  - 如果数据的实际宽度小于 m，那么左边补空格
  - 如果实际宽度大于 m，那么按实际宽度输出

printf 函数格式化输出的位数/宽度控制

输出整数时，可以增加宽度控制

- %md　指定输出的宽度为 m (即占据 m 个字符位置，包括符号位)
  - 如果数据的实际宽度小于 m，那么左边补空格
  - 如果实际宽度大于 m，那么按实际宽度输出
- 如果输入：17 17，下面代码段的输出是什么？

printf 函数格式化输出的位数/宽度控制

输出整数时，可以增加宽度控制

- %md　指定输出的宽度为 m (即占据 m 个字符位置，包括符号位)
  - 如果数据的实际宽度小于 m，那么左边补空格
  - 如果实际宽度大于 m，那么按实际宽度输出
- 如果输入：17 17，下面代码段的输出是什么？

printf 函数格式化输出的位数/宽度控制

输出整数时，可以增加宽度控制

- %md　指定输出的宽度为 m (即占据 m 个字符位置，包括符号位)
  - 如果数据的实际宽度小于 m，那么左边补空格
  - 如果实际宽度大于 m，那么按实际宽度输出
- 如果输入：17 17，下面代码段的输出是什么？

```
printf("input a, b:");
scanf("%o%d", &a, &b);
printf("%d%5d\n", a, b);
```

输出是：15␣␣␣17

printf 函数格式化输出的位数/宽度控制

输出整数时，可以增加宽度控制

- %md　指定输出的宽度为 m (即占据 m 个字符位置，包括符号位)
  - 如果数据的实际宽度小于 m，那么左边补空格
  - 如果实际宽度大于 m，那么按实际宽度输出
- 如果输入：17 17，下面代码段的输出是什么？

```
printf("input a, b:");
scanf("%o%d", &a, &b);
printf("%d%5d\n", a, b);
```

输出是：15␣␣␣17

17 的左边补了三个空格

printf 函数格式化输出的位数/宽度控制

输出整数时，可以增加宽度控制

- %md　指定输出的宽度为 m (即占据 m 个字符位置，包括符号位)
  - 如果数据的实际宽度小于 m，那么左边补空格
  - 如果实际宽度大于 m，那么按实际宽度输出
- 如果输入：17 17，下面代码段的输出是什么？

```
printf("input a, b:");
scanf("%o%d", &a, &b);
printf("%d%5d\n", a, b);
```

输出是：15␣␣␣17

17 的左边补了三个空格

- 整数输出的宽度控制还有：%mo、%mx、%mu 等等。例如

printf 函数格式化输出的位数/宽度控制

输出整数时，可以增加宽度控制

- %md　指定输出的宽度为 m (即占据 m 个字符位置，包括符号位)
  - 如果数据的实际宽度小于 m，那么左边补空格
  - 如果实际宽度大于 m，那么按实际宽度输出
- 如果输入：17 17，下面代码段的输出是什么？

```
printf("input a, b:");
scanf("%o%d", &a, &b);
printf("%d%5d\n", a, b);
```

输出是：15␣␣␣17

17 的左边补了三个空格

- 整数输出的宽度控制还有：%mo、%mx、%mu 等等。例如
  - printf("%5o%5x%6u\n", a, b, c);

printf 函数格式化输出的位数/宽度控制

输出整数时，可以增加宽度控制

- %md　指定输出的宽度为 m (即占据 m 个字符位置，包括符号位)
  - 如果数据的实际宽度小于 m，那么左边补空格
  - 如果实际宽度大于 m，那么按实际宽度输出
- 如果输入：17 17，下面代码段的输出是什么？

```
printf("input a, b:");
scanf("%o%d", &a, &b);
printf("%d%5d\n", a, b);
```

输出是：15␣␣␣17

17 的左边补了三个空格

- 整数输出的宽度控制还有：%mo、%mx、%mu 等等。例如
  - printf("%5o%5x%6u\n", a, b, c);
  - 注意前文提到 o,x,u 只能输出正整数，以及如何处理负数的

printf 函数格式化输出的位数/宽度控制

输出浮点数时，也可以增加宽度控制

- %m.nf　指定输出的宽度为 m (即包括小数点和符号位)，其中小数点后面的宽度为 n

printf 函数格式化输出的位数/宽度控制

输出浮点数时，也可以增加宽度控制

- %m.nf　指定输出的宽度为 m (即包括小数点和符号位)，其中小数点后面的宽度为 n
  - 如果数据的实际宽度小于 m，那么左边补空格

printf 函数格式化输出的位数/宽度控制

输出浮点数时，也可以增加宽度控制

- %m.nf　指定输出的宽度为 m (即包括小数点和符号位)，其中小数点后面的宽度为 n
  - 如果数据的实际宽度小于 m，那么左边补空格
  - 如果实际宽度大于 m，那么按实际宽度输出

printf 函数格式化输出的位数/宽度控制

输出浮点数时，也可以增加宽度控制

- %m.nf　指定输出的宽度为 m (即包括小数点和符号位)，其中小数点后面的宽度为 n
  - 如果数据的实际宽度小于 m，那么左边补空格
  - 如果实际宽度大于 m，那么按实际宽度输出
- 下面代码段的输出是什么？

printf 函数格式化输出的位数/宽度控制

输出浮点数时，也可以增加宽度控制

- %m.nf　指定输出的宽度为 m (即包括小数点和符号位)，其中小数点后面的宽度为 n
  - 如果数据的实际宽度小于 m，那么左边补空格
  - 如果实际宽度大于 m，那么按实际宽度输出
- 下面代码段的输出是什么？

printf 函数格式化输出的位数/宽度控制

输出浮点数时，也可以增加宽度控制

- %m.nf　指定输出的宽度为 m (即包括小数点和符号位)，其中小数点后面的宽度为 n
  - 如果数据的实际宽度小于 m，那么左边补空格
  - 如果实际宽度大于 m，那么按实际宽度输出
- 下面代码段的输出是什么？

```
double d = 3.1415926;
printf("%f,%e\n", d, d);
printf("%5.3f,%5.2f,%.2f\n", d, d, d);
```

printf 函数格式化输出的位数/宽度控制

输出浮点数时，也可以增加宽度控制

- %m.nf　指定输出的宽度为 m (即包括小数点和符号位)，其中小数点后面的宽度为 n
  - 如果数据的实际宽度小于 m，那么左边补空格
  - 如果实际宽度大于 m，那么按实际宽度输出
- 下面代码段的输出是什么？

```
double d = 3.1415926;
printf("%f,%e\n", d, d);
printf("%5.3f,%5.2f,%.2f\n", d, d, d);
```

$3.141593, 3.14159e+00$

printf 函数格式化输出的位数/宽度控制

输出浮点数时，也可以增加宽度控制

- %m.nf　指定输出的宽度为 m (即包括小数点和符号位)，其中小数点后面的宽度为 n
  - 如果数据的实际宽度小于 m，那么左边补空格
  - 如果实际宽度大于 m，那么按实际宽度输出
- 下面代码段的输出是什么？

```
double d = 3.1415926;
printf("%f,%e\n", d, d);
printf("%5.3f,%5.2f,%.2f\n", d, d, d);
```

$3.141593, 3.14159e+00$

$3.142, \sqcup 3.14, 3.14$

printf 函数格式化输出的位数/宽度控制

输出浮点数时，也可以增加宽度控制

- %m.nf　指定输出的宽度为 m (即包括小数点和符号位)，其中小数点后面的宽度为 n
  - 如果数据的实际宽度小于 m，那么左边补空格
  - 如果实际宽度大于 m，那么按实际宽度输出
- 下面代码段的输出是什么？

```
double d = 3.1415926;
printf("%f,%e\n", d, d);
printf("%5.3f,%5.2f,%.2f\n", d, d, d);
```

$3.141593, 3.14159e+00$

$3.142, \sqcup 3.14, 3.14$

- 在%.2f 中没有指定总的宽度 m，那么输出宽度为实际宽度。

字符类型的输出

- 调用 printf/scanf 函数，用%c 格式控制

字符类型的输出

- 调用 printf/scanf 函数，用%c 格式控制
- 调用 getchar/putchar 函数，输入和输出一个字符

字符类型的输出

- 调用 printf/scanf 函数，用%c 格式控制
- 调用 getchar/putchar 函数，输入和输出一个字符
- 假设输入：A bc< 回车 >，那么以下代码的输出是什么？

字符类型的输出

- 调用 printf/scanf 函数，用%c 格式控制
- 调用 getchar/putchar 函数，输入和输出一个字符
- 假设输入：A bc< 回车 >，那么以下代码的输出是什么？

字符类型的输出

- 调用 printf/scanf 函数，用%c 格式控制
- 调用 getchar/putchar 函数，输入和输出一个字符
- 假设输入：A bc< 回车 >，那么以下代码的输出是什么？

```
scanf("%c%c%c", &a, &b, &c);
printf("%c%c%c%c", a, '#', b, '#', c);
```

字符类型的输出

- 调用 printf/scanf 函数，用%c 格式控制
- 调用 getchar/putchar 函数，输入和输出一个字符
- 假设输入：A bc< 回车 >，那么以下代码的输出是什么？

```
scanf("%c%c%c", &a, &b, &c);
printf("%c%c%c%c", a, '#', b, '#', c);
```

输出是：A#␣#b

字符类型的输出

- 调用 printf/scanf 函数，用%c 格式控制
- 调用 getchar/putchar 函数，输入和输出一个字符
- 假设输入：A bc< 回车 >，那么以下代码的输出是什么？

```
scanf("%c%c%c", &a, &b, &c);
printf("%c%c%c%c", a, '#', b, '#', c);
```

输出是：A#␣#b

字符输入输出需要注意：

- scanf 和 getchar 会将空格，回车当普通字符读入

字符类型的输出

- 调用 printf/scanf 函数，用%c 格式控制
- 调用 getchar/putchar 函数，输入和输出一个字符
- 假设输入：A bc< 回车 >，那么以下代码的输出是什么？

```
scanf("%c%c%c", &a, &b, &c);
printf("%c%c%c%c", a, '#', b, '#', c);
```

输出是：A#␣#b

字符输入输出需要注意：

- scanf 和 getchar 会将空格，回车当普通字符读入
- printf 和 putchar 会将空格，回车当普通字符输出，输出回车符就是换行

字符类型的输出

- 调用 printf/scanf 函数，用%c 格式控制
- 调用 getchar/putchar 函数，输入和输出一个字符
- 假设输入：A bc< 回车 >，那么以下代码的输出是什么？

```
scanf("%c%c%c", &a, &b, &c);
printf("%c%c%c%c", a, '#', b, '#', c);
```

输出是：A#␣#b

字符输入输出需要注意：

- scanf 和 getchar 会将空格，回车当普通字符读入
- printf 和 putchar 会将空格，回车当普通字符输出，输出回车符就是换行
- 输出字符时，两侧没有单引号

字符类型的输出，例程 6-1

```c
#include <stdio.h>
int main(void)
{
    char ch, upper;
    while( scanf("%c",&ch)==1 && ch!='\n' ) {
        if( ch >= 'a' && ch <= 'z' ) {
            upper= ch - 'a' + 'A';
            printf("%c --> %c  --> %d\n", ch,
                upper, upper%10);
        }
    }
    return 0;
}
```

字符类型的输出，例程 6-1

```
#include <stdio.h>
int main(void)
{
    char ch, upper;
    while( scanf("%c",&ch)==1 && ch!='\n' ) {
        if( ch >= 'a' && ch <= 'z' ) {
            upper= ch - 'a' + 'A';
            printf("%c --> %c  --> %d\n", ch,
                upper, upper%10);
        }
    }
    return 0;
}
```

- scanf 返回值：成功读取的数据个数

字符类型的输出，例程 6-1

```c
#include <stdio.h>
int main(void)
{
    char ch, upper;
    while( scanf("%c",&ch)==1 && ch!='\n' ) {
        if( ch >= 'a' && ch <= 'z' ) {
            upper= ch - 'a' + 'A';
            printf("%c --> %c  --> %d\n", ch,
                upper, upper%10);
        }
    }
    return 0;
}
```

- scanf 返回值：成功读取的数据个数
- 字母小写 $\Longrightarrow$ 大写：c - 'a' + 'A'

字符类型的输出，例程 6-1

```c
#include <stdio.h>
int main(void)
{
    char ch, upper;
    while( scanf("%c",&ch)==1 && ch!='\n' ) {
        if( ch >= 'a' && ch <= 'z' ) {
            upper= ch - 'a' + 'A';
            printf("%c --> %c  --> %d\n", ch,
                upper, upper%10);
        }
    }
    return 0;
}
```

- scanf 返回值：成功读取的数据个数
- 字母小写 $\Longrightarrow$ 大写：c - 'a' + 'A'
- 字母大写 $\Longrightarrow$ 小写：c - 'A' + 'a'

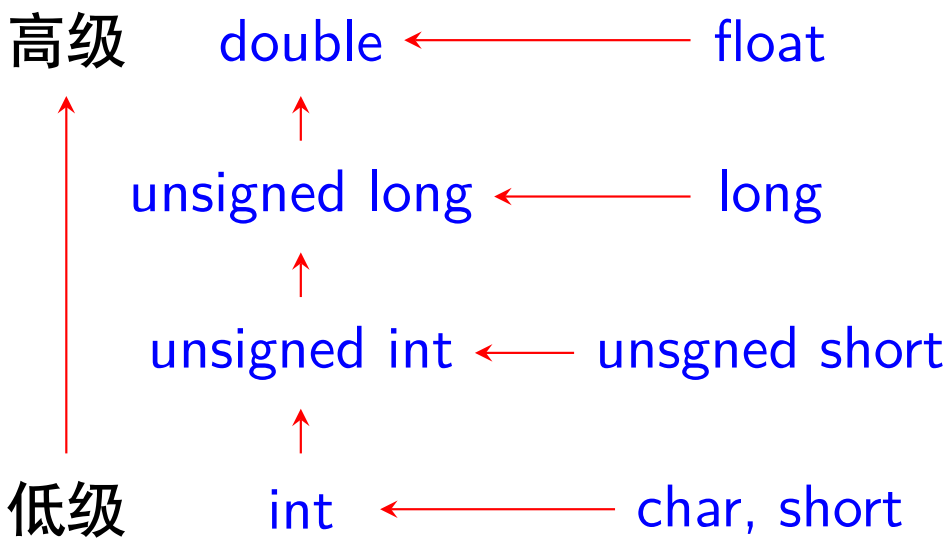
内容提要

数据类型的转换

自动类型转换、强制类型转换

自动类型转换、强制类型转换

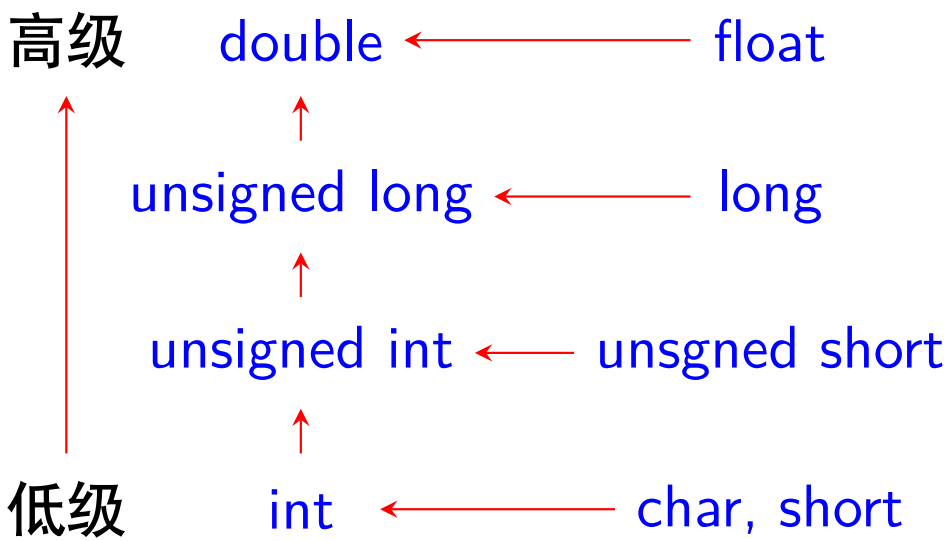
自动类型转换



自动类型转换、强制类型转换

自动类型转换

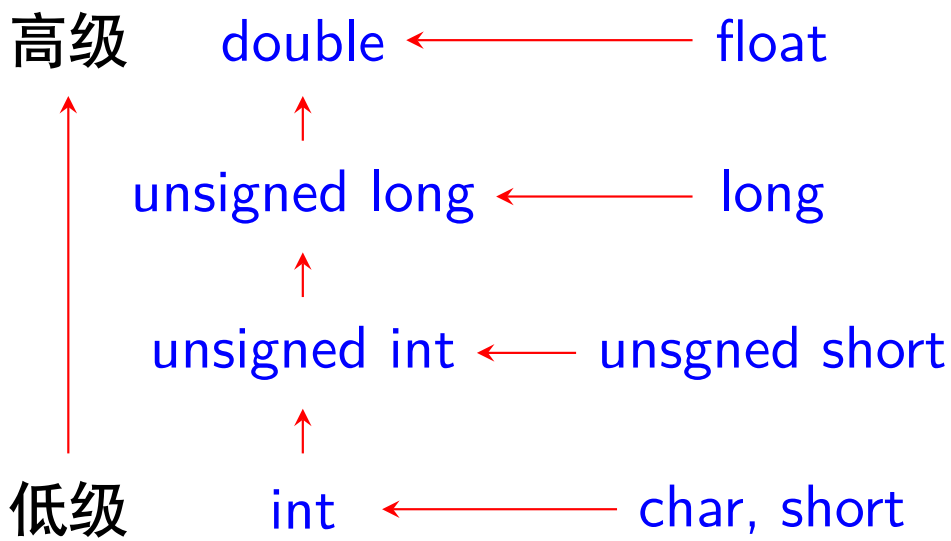
- 水平方向的转换



自动类型转换、强制类型转换

自动类型转换

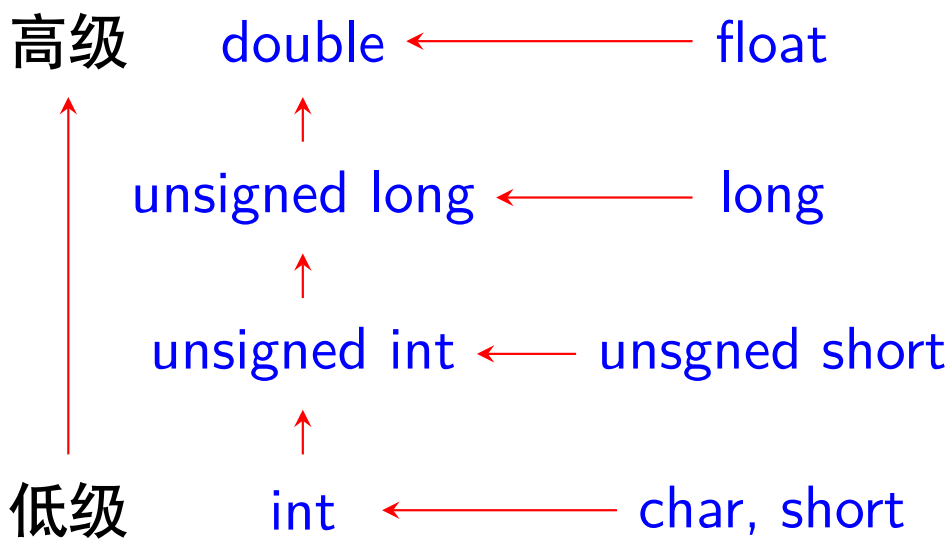
- 水平方向的转换
  - char 和 short 自动转换为 int



自动类型转换、强制类型转换

自动类型转换

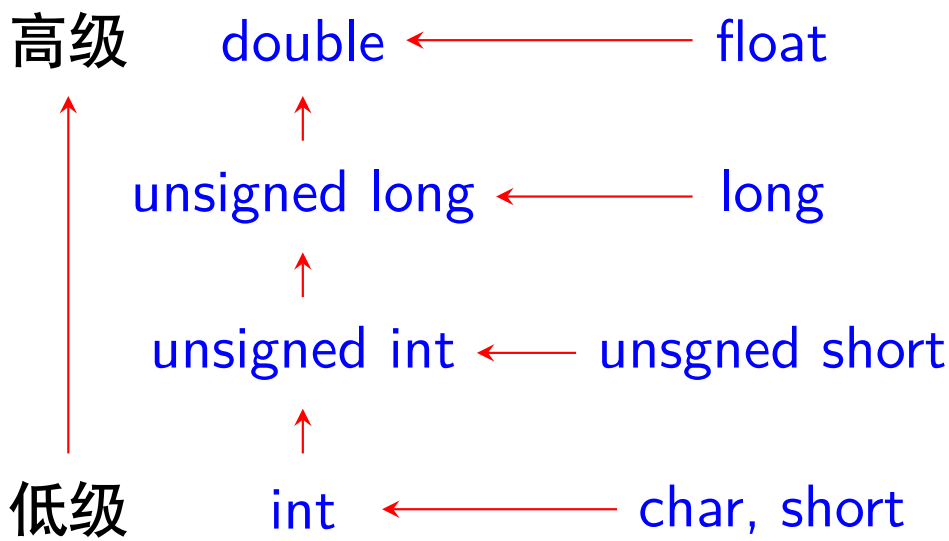
- 水平方向的转换
  - char 和 short 自动转换为 int
  - unsigned short 自动转换为 unsigned int



自动类型转换、强制类型转换

自动类型转换

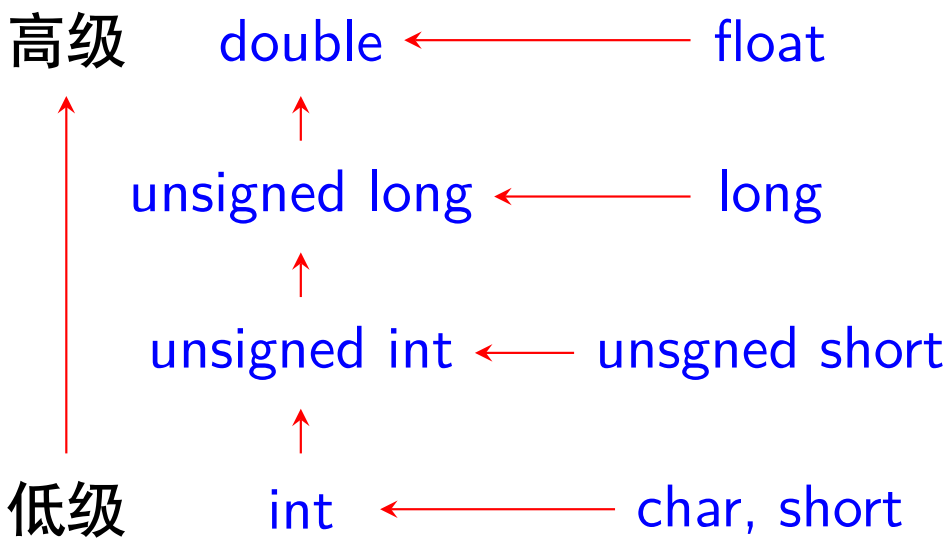
- 水平方向的转换
  - char 和 short 自动转换为 int
  - unsigned short 自动转换为 unsigned int
  - long 自动转换为 unsigned long



自动类型转换、强制类型转换

自动类型转换

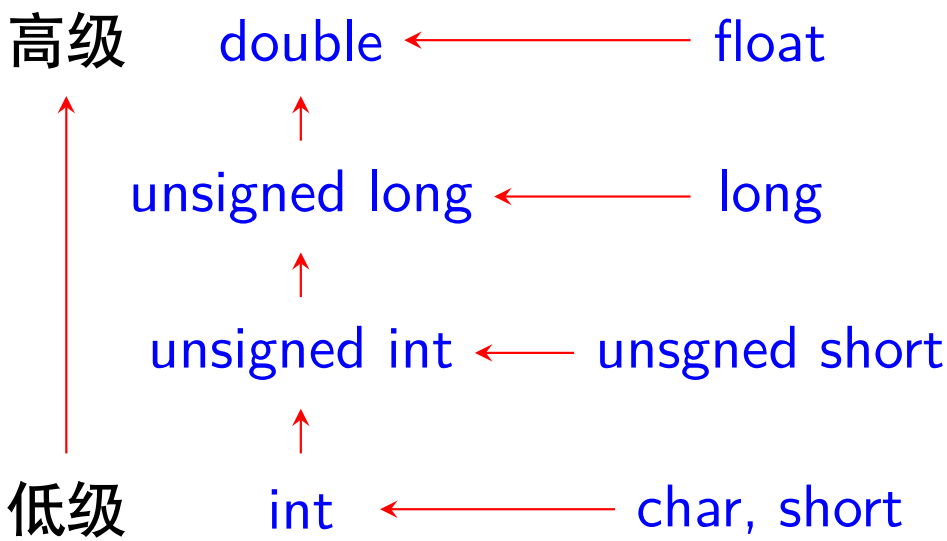
- 水平方向的转换
  - char 和 short 自动转换为 int
  - unsigned short 自动转换为 unsigned int
  - long 自动转换为 unsigned long
  - float 自动转换为 double



自动类型转换、强制类型转换

自动类型转换

- 水平方向的转换
  - char 和 short 自动转换为 int
  - unsigned short 自动转换为 unsigned int
  - long 自动转换为 unsigned long
  - float 自动转换为 double
- 垂直方向转换

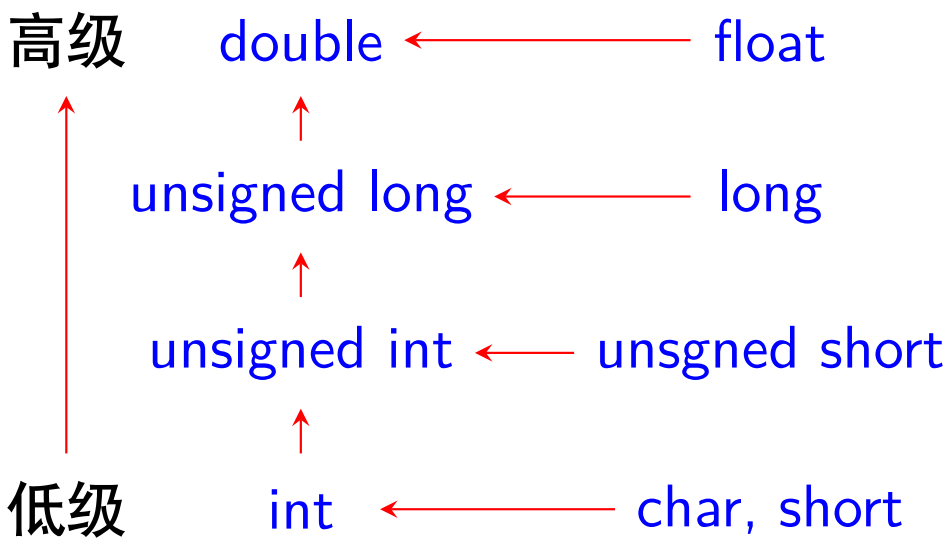


数据类型的转换

自动类型转换、强制类型转换

自动类型转换

- 水平方向的转换
  - char 和 short 自动转换为 int
  - unsigned short 自动转换为 unsigned int
  - long 自动转换为 unsigned long
  - float 自动转换为 double
- 垂直方向转换
  - 水平方向自动转换之后，如果数据类型仍然不相同，那么进行垂直方向的转换

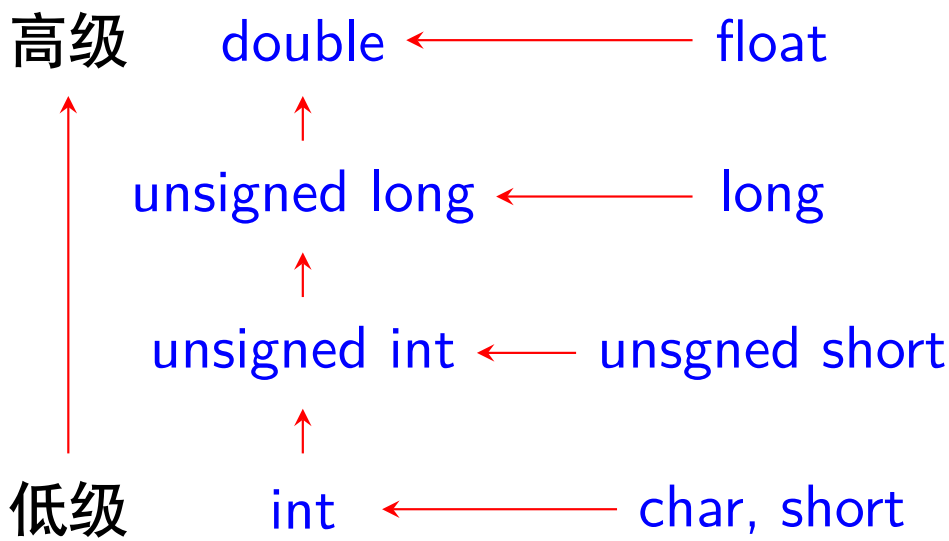


数据类型的转换

自动类型转换、强制类型转换

自动类型转换

- 水平方向的转换
  - char 和 short 自动转换为 int
  - unsigned short 自动转换为 unsigned int
  - long 自动转换为 unsigned long
  - float 自动转换为 double
- 垂直方向转换
  - 水平方向自动转换之后，如果数据类型仍然不相同，那么进行垂直方向的转换
- 自动转换的原则：

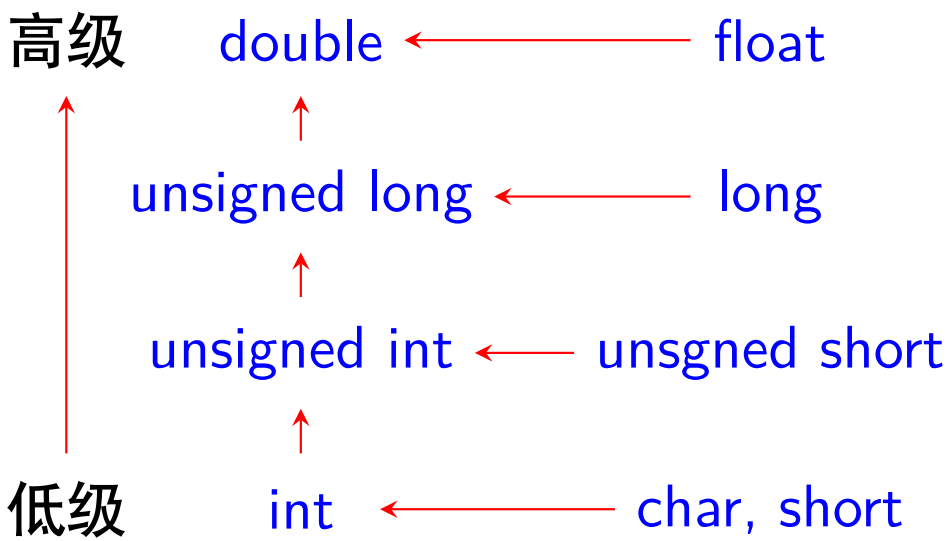


数据类型的转换

自动类型转换、强制类型转换

自动类型转换

- 水平方向的转换
  - char 和 short 自动转换为 int
  - unsigned short 自动转换为 unsigned int
  - long 自动转换为 unsigned long
  - float 自动转换为 double
- 垂直方向转换
  - 水平方向自动转换之后，如果数据类型仍然不相同，那么进行垂直方向的转换
- 自动转换的原则：
  - 数据字节从短到长

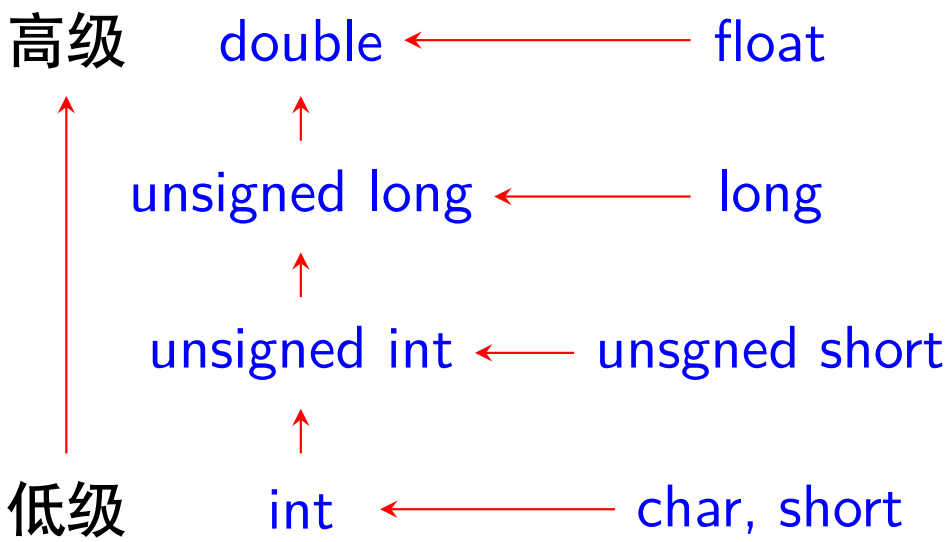


数据类型的转换

自动类型转换、强制类型转换

自动类型转换

- 水平方向的转换
  - char 和 short 自动转换为 int
  - unsigned short 自动转换为 unsigned int
  - long 自动转换为 unsigned long
  - float 自动转换为 double
- 垂直方向转换
  - 水平方向自动转换之后，如果数据类型仍然不相同，那么进行垂直方向的转换
- 自动转换的原则：
  - 数据字节从短到长
  - 表示范围从小到大



强制类型转换

- 使用强制类型转换运算符，可以将一个表达式的值转换为给定的类型

(类型名) 表达式;

强制类型转换

- 使用强制类型转换运算符，可以将一个表达式的值转换为给定的类型

(类型名) 表达式;

- 强制类型转换运算符：(类型名)

强制类型转换

- 使用强制类型转换运算符，可以将一个表达式的值转换为给定的类型

(类型名) 表达式;

- 强制类型转换运算符：(类型名)
- 是一种单目运算符，高于任何双目运算符的优先级

强制类型转换

- 使用强制类型转换运算符，可以将一个表达式的值转换为给定的类型

  (类型名) 表达式;

- 强制类型转换运算符：(类型名)
- 是一种单目运算符，高于任何双目运算符的优先级
- 无论是何种类型转换，都是为了本次运算对数据进行临时转换，并没有改变原来的数据

强制类型转换

思考：以下代码片段的输出是什么？

```
float x = 1.5;
printf("%f", (int)x + 1.8);
printf("%f", (int)(x+1.8));
printf("%d", (int)x);
printf("%d", x);
```

- printf 函数不会根据% 号后面的格式，进行类型转换

强制类型转换

思考：以下代码片段的输出是什么？

```
float x = 1.5;
printf("%f", (int)x + 1.8);
printf("%f", (int)(x+1.8));
printf("%d", (int)x);
printf("%d", x);
```

- printf 函数不会根据% 号后面的格式，进行类型转换
  - 因为加入类型转换代码 (进行类型转换)是编译器的工作

强制类型转换

思考：以下代码片段的输出是什么？

```
float x = 1.5;
printf("%f", (int)x + 1.8);
printf("%f", (int)(x+1.8));
printf("%d", (int)x);
printf("%d", x);
```

- printf 函数不会根据% 号后面的格式，进行类型转换
  - 因为加入类型转换代码 (进行类型转换)是编译器的工作
- 调用 printf 函数时，参数值被原封不动地复制给 printf 函数

强制类型转换

思考：以下代码片段的输出是什么？

```
float x = 1.5;
printf("%f", (int)x + 1.8);
printf("%f", (int)(x+1.8));
printf("%d", (int)x);
printf("%d", x);
```

- printf 函数不会根据% 号后面的格式，进行类型转换
  - 因为加入类型转换代码 (进行类型转换)是编译器的工作
- 调用 printf 函数时，参数值被原封不动地复制给 printf 函数
- 然后在 printf 函数的代码中，按照% 号后面的格式对参数的 bit 值进行解释

强制类型转换

思考：以下代码片段的输出是什么？

```
float x = 1.5;
printf("%f", (int)x + 1.8);
printf("%f", (int)(x+1.8));
printf("%d", (int)x);
printf("%d", x);
```

- printf 函数不会根据% 号后面的格式，进行类型转换
  - 因为加入类型转换代码 (进行类型转换)是编译器的工作
- 调用 printf 函数时，参数值被原封不动地复制给 printf 函数
- 然后在 printf 函数的代码中，按照% 号后面的格式对参数的 bit 值进行解释
- 若% 格式与参数类型一致，那么可以保证输出结果正确

强制类型转换

思考：以下代码片段的输出是什么？

```
float x = 1.5;
printf("%f", (int)x + 1.8);
printf("%f", (int)(x+1.8));
printf("%d", (int)x);
printf("%d", x);
```

- printf 函数不会根据% 号后面的格式，进行类型转换
  - 因为加入类型转换代码 (进行类型转换)是编译器的工作
- 调用 printf 函数时，参数值被原封不动地复制给 printf 函数
- 然后在 printf 函数的代码中，按照% 号后面的格式对参数的 bit 值进行解释
- 若% 格式与参数类型一致，那么可以保证输出结果正确
- 否则，不能保证输出结果正确

内容提要

表达式

- 算术表达式
- 赋值表达式
- 关系表达式
- 逻辑表达式
- 条件表达式
- 逗号表达式
- 按位运算
- 其他运算

注意：表达式都是有值

算术表达式

- **双目运算符**：$+$ $-$ $*$ $/$ $\%$

算术表达式

- 双目运算符：$+ \quad - \quad * \quad / \quad \%$
- 单目运算符：$- \quad + \quad ++ \quad --$

算术表达式

- 双目运算符：$+ \quad - \quad * \quad / \quad \%$
- 单目运算符：$- \quad + \quad ++ \quad --$

自增运算符：$++$

算术表达式

- 双目运算符：$+ \quad - \quad * \quad / \quad \%$
- 单目运算符：$- \quad + \quad ++ \quad --$

自增运算符：$++$

- $x++ \Longleftrightarrow$ 取 x 的值作为表达式的值，然后执行 $x = x + 1$

算术表达式

- 双目运算符：$+ \quad - \quad * \quad / \quad \%$
- 单目运算符：$- \quad + \quad ++ \quad --$

自增运算符：$++$

- $x++ \Longleftrightarrow$ 取 x 的值作为表达式的值，然后执行 $x = x + 1$
- $++x \Longleftrightarrow$ 执行 $x = x + 1$，然后取 x 的值作为表达式的值

算术表达式

- 双目运算符：$+ \quad - \quad * \quad / \quad \%$
- 单目运算符：$- \quad + \quad ++ \quad --$

自增运算符：$++$

- $x++ \Longleftrightarrow$ 取 x 的值作为表达式的值，然后执行 $x = x + 1$
- $++x \Longleftrightarrow$ 执行 $x = x + 1$，然后取 x 的值作为表达式的值

自减运算符：$--$

算术表达式

- 双目运算符：$+ \quad - \quad * \quad / \quad \%$
- 单目运算符：$- \quad + \quad ++ \quad --$

自增运算符：$++$

- $x++ \Longleftrightarrow$ 取 x 的值作为表达式的值，然后执行 $x = x + 1$
- $++x \Longleftrightarrow$ 执行 $x = x + 1$，然后取 x 的值作为表达式的值

自减运算符：$--$

- $x-- \Longleftrightarrow$ 取 x 的值作为表达式的值，然后执行 $x = x - 1$

算术表达式

- 双目运算符：$+ \quad - \quad * \quad / \quad \%$
- 单目运算符：$- \quad + \quad ++ \quad --$

自增运算符：$++$

- $x++ \Longleftrightarrow$ 取 x 的值作为表达式的值，然后执行 $x = x + 1$
- $++x \Longleftrightarrow$ 执行 $x = x + 1$，然后取 x 的值作为表达式的值

自减运算符：$--$

- $x-- \Longleftrightarrow$ 取 x 的值作为表达式的值，然后执行 $x = x - 1$
- $--x \Longleftrightarrow$ 执行 $x = x - 1$，然后取 x 的值作为表达式的值

算术表达式

- 双目运算符：$+ \quad - \quad * \quad / \quad \%$
- 单目运算符：$- \quad + \quad ++ \quad --$

自增运算符：$++$

- $x++ \Longleftrightarrow$ 取 x 的值作为表达式的值，然后执行 $x = x + 1$
- $++x \Longleftrightarrow$ 执行 $x = x + 1$，然后取 x 的值作为表达式的值

自减运算符：$--$

- $x-- \Longleftrightarrow$ 取 x 的值作为表达式的值，然后执行 $x = x - 1$
- $--x \Longleftrightarrow$ 执行 $x = x - 1$，然后取 x 的值作为表达式的值

算术表达式

- 双目运算符：$+ \quad - \quad * \quad / \quad \%$
- 单目运算符：$- \quad + \quad ++ \quad --$

自增运算符：$++$

- $x++ \Longleftrightarrow$ 取 x 的值作为表达式的值，然后执行 $x = x + 1$
- $++x \Longleftrightarrow$ 执行 $x = x + 1$，然后取 x 的值作为表达式的值

自减运算符：$--$

- $x-- \Longleftrightarrow$ 取 x 的值作为表达式的值，然后执行 $x = x - 1$
- $--x \Longleftrightarrow$ 执行 $x = x - 1$，然后取 x 的值作为表达式的值

$++$ 和 $--$ 要改变操作数的值，只能作用在变量上面，或者相当于变量的对象上（以后学习到的数组元素，或指针指向的元素）。

$++3$ 或者 $(i + x)--$ 都是非法的

运算符的优先级和结合性

优先级

运算符的优先级和结合性

优先级

- () 优先级最高，可以改变优先级

运算符的优先级和结合性

优先级

- () 优先级最高，可以改变优先级
- 单目运算 > 双目 > 多目

运算符的优先级和结合性

优先级

- () 优先级最高，可以改变优先级
- 单目运算 > 双目 > 多目
  - ！（逻辑非）、－（取反）、++、−−

运算符的优先级和结合性

优先级

- () 优先级最高，可以改变优先级
- 单目运算 > 双目 > 多目
  - ！(逻辑非)、− (取反)、++、−−
- 算术运算 > 关系运算 > 逻辑运算 > 赋值运算

运算符的优先级和结合性

优先级

- () 优先级最高，可以改变优先级
- 单目运算 $>$ 双目 $>$ 多目
  - !（逻辑非）、$-$（取反）、$++$、$--$
- 算术运算 $>$ 关系运算 $>$ 逻辑运算 $>$ 赋值运算
  - 算术乘除：$*$，$/$，$\%$

运算符的优先级和结合性

优先级

- () 优先级最高，可以改变优先级
- 单目运算 > 双目 > 多目
  - !（逻辑非）、−（取反）、++、−−
- 算术运算 > 关系运算 > 逻辑运算 > 赋值运算
  - 算术乘除：∗，/，%
  - 算术加减：+，−

运算符的优先级和结合性

优先级

- () 优先级最高，可以改变优先级
- 单目运算 > 双目 > 多目
  - !（逻辑非）、−（取反）、++、−−
- 算术运算 > 关系运算 > 逻辑运算 > 赋值运算
  - 算术乘除：*，/，%
  - 算术加减：+，−
  - 关系：< > <= >=

运算符的优先级和结合性

优先级

- () 优先级最高，可以改变优先级
- 单目运算 > 双目 > 多目
  - !（逻辑非）、−（取反）、++、−−
- 算术运算 > 关系运算 > 逻辑运算 > 赋值运算
  - 算术乘除：*，/，%
  - 算术加减：+，−
  - 关系：< > <= >=
  - 关系：== !=（低于大小判断）

运算符的优先级和结合性

优先级

- () 优先级最高，可以改变优先级
- 单目运算 > 双目 > 多目
  - !（逻辑非）、−（取反）、++、−−
- 算术运算 > 关系运算 > 逻辑运算 > 赋值运算
  - 算术乘除：*，/，%
  - 算术加减：+，−
  - 关系：< > <= >=
  - 关系：== !=（低于大小判断）
  - 逻辑与：&&

运算符的优先级和结合性

优先级

- () 优先级最高，可以改变优先级
- 单目运算 > 双目 > 多目
  - !（逻辑非）、−（取反）、++、−−
- 算术运算 > 关系运算 > 逻辑运算 > 赋值运算
  - 算术乘除：*，/，%
  - 算术加减：+，−
  - 关系：< > <= >=
  - 关系：== !=（低于大小判断）
  - 逻辑与：&&
  - 逻辑或：||

运算符的优先级和结合性

优先级

- () 优先级最高，可以改变优先级
- 单目运算 > 双目 > 多目
  - !（逻辑非）、−（取反）、++、−−
- 算术运算 > 关系运算 > 逻辑运算 > 赋值运算
  - 算术乘除：*, /, %
  - 算术加减：+, −
  - 关系：< > <= >=
  - 关系：== !=（低于大小判断）
  - 逻辑与：&&
  - 逻辑或：||
- 条件表达式运算符：?:

运算符的优先级和结合性

优先级

- () 优先级最高，可以改变优先级
- 单目运算 > 双目 > 多目
  - !（逻辑非）、−（取反）、++、−−
- 算术运算 > 关系运算 > 逻辑运算 > 赋值运算
  - 算术乘除：*, /, %
  - 算术加减：+, −
  - 关系：< > <= >=
  - 关系：== !=（低于大小判断）
  - 逻辑与：&&
  - 逻辑或：||
- 条件表达式运算符：?:
  - < 条件表达式 > ? < 表达式 1> : < 表达式 2>

运算符的优先级和结合性

优先级

- () 优先级最高，可以改变优先级
- 单目运算 > 双目 > 多目
  - !（逻辑非）、−（取反）、++、−−
- 算术运算 > 关系运算 > 逻辑运算 > 赋值运算
  - 算术乘除：*, /, %
  - 算术加减：+, −
  - 关系：< > <= >=
  - 关系：== !=（低于大小判断）
  - 逻辑与：&&
  - 逻辑或：||
- 条件表达式运算符：?:
  - < 条件表达式 > ? < 表达式 1> : < 表达式 2>
- 赋值运算符：=

运算符的优先级和结合性

优先级

- () 优先级最高，可以改变优先级
- 单目运算 > 双目 > 多目
  - !（逻辑非）、−（取反）、++、−−
- 算术运算 > 关系运算 > 逻辑运算 > 赋值运算
  - 算术乘除：*, /, %
  - 算术加减：+, −
  - 关系：< > <= >=
  - 关系：== !=（低于大小判断）
  - 逻辑与：&&
  - 逻辑或：||
- 条件表达式运算符：?:
  - < 条件表达式 > ? < 表达式 1> : < 表达式 2>
- 赋值运算符：=
  - 以及各种复合赋值运算符

运算符的优先级和结合性

结合性：同级的多个运算符，按照结合性定义运算顺序

运算符的优先级和结合性

结合性：同级的多个运算符，按照结合性定义运算顺序

- 从右 $\Rightarrow$ 左结合：单目运算符，赋值运算符

运算符的优先级和结合性

结合性：同级的多个运算符，按照结合性定义运算顺序

- 从右 $\Rightarrow$ 左结合：单目运算符，赋值运算符
- 从左 $\Rightarrow$ 右结合：其余的。这部分占大多数

运算符的优先级和结合性

结合性：同级的多个运算符，按照结合性定义运算顺序

- 从右 $\Rightarrow$ 左结合：单目运算符，赋值运算符
- 从左 $\Rightarrow$ 右结合：其余的。这部分占大多数

举几个例子

运算符的优先级和结合性

结合性：同级的多个运算符，按照结合性定义运算顺序

- 从右 $\Rightarrow$ 左结合：单目运算符，赋值运算符
- 从左 $\Rightarrow$ 右结合：其余的。这部分占大多数

举几个例子

- $-5+3\%2 \Longleftrightarrow (-5)+(3\%2)$

运算符的优先级和结合性

结合性：同级的多个运算符，按照结合性定义运算顺序

- 从右 $\Rightarrow$ 左结合：单目运算符，赋值运算符
- 从左 $\Rightarrow$ 右结合：其余的。这部分占大多数

举几个例子

- $-5+3\%2 \Longleftrightarrow (-5)+(3\%2)$
- $3*5\%3 \Longleftrightarrow (3*5)\%3$

运算符的优先级和结合性

结合性：同级的多个运算符，按照结合性定义运算顺序

- 从右 $\Rightarrow$ 左结合：单目运算符，赋值运算符
- 从左 $\Rightarrow$ 右结合：其余的。这部分占大多数

举几个例子

- $-5+3\%2 \Longleftrightarrow (-5)+(3\%2)$
- $3*5\%3 \Longleftrightarrow (3*5)\%3$
- $-x++ \Longleftrightarrow -(x++)$

运算符的优先级和结合性

结合性：同级的多个运算符，按照结合性定义运算顺序

- 从右 $\Rightarrow$ 左结合：单目运算符，赋值运算符
- 从左 $\Rightarrow$ 右结合：其余的。这部分占大多数

举几个例子

- $-5+3\%2 \Longleftrightarrow (-5)+(3\%2)$
- $3*5\%3 \Longleftrightarrow (3*5)\%3$
- $-x++ \Longleftrightarrow -(x++)$
  - 单目运算符，从右开始结合

运算符的优先级和结合性

结合性：同级的多个运算符，按照结合性定义运算顺序

- 从右 $\Rightarrow$ 左结合：单目运算符，赋值运算符
- 从左 $\Rightarrow$ 右结合：其余的。这部分占大多数

举几个例子

- $-5+3\%2 \Longleftrightarrow (-5)+(3\%2)$
- $3*5\%3 \Longleftrightarrow (3*5)\%3$
- $-x++ \Longleftrightarrow -(x++)$
  - 单目运算符，从右开始结合
  - 假设 x=6，那么 $-x++$ 的值等于 ?，之后 x 的值等于几 ?

运算符的优先级和结合性

结合性：同级的多个运算符，按照结合性定义运算顺序

- 从右 $\Rightarrow$ 左结合：单目运算符，赋值运算符
- 从左 $\Rightarrow$ 右结合：其余的。这部分占大多数

举几个例子

- $-5+3\%2 \iff (-5)+(3\%2)$
- $3*5\%3 \iff (3*5)\%3$
- $-x++ \iff -(x++)$
  - 单目运算符，从右开始结合
  - 假设 x=6，那么 $-x++$ 的值等于 ?，之后 x 的值等于几？
  - -6，7

赋值表达式

变量 = 表达式

- 首先计算右侧表达式的值

赋值表达式

变量 = 表达式

- 首先计算右侧表达式的值
- 然后将这个值赋予左侧的变量

赋值表达式

变量 = 表达式

- 首先计算右侧表达式的值
- 然后将这个值赋予左侧的变量
  - 如果类型不匹配，那么在赋值之前先将这个值转换为变量的类型

赋值表达式

变量 = 表达式

- 首先计算右侧表达式的值
- 然后将这个值赋予左侧的变量
  - 如果类型不匹配，那么在赋值之前先将这个值转换为变量的类型
- 最后将被赋予的值作为本赋值表达式的值

赋值表达式

变量 = 表达式

- 首先计算右侧表达式的值
- 然后将这个值赋予左侧的变量
  - 如果类型不匹配，那么在赋值之前先将这个值转换为变量的类型
- 最后将被赋予的值作为本赋值表达式的值

举几个例子

赋值表达式

变量 = 表达式

- 首先计算右侧表达式的值
- 然后将这个值赋予左侧的变量
  - 如果类型不匹配，那么在赋值之前先将这个值转换为变量的类型
- 最后将被赋予的值作为本赋值表达式的值

举几个例子

- `int n = 3.14 * 2`

赋值表达式

变量 = 表达式

- 首先计算右侧表达式的值
- 然后将这个值赋予左侧的变量
  - 如果类型不匹配，那么在赋值之前先将这个值转换为变量的类型
- 最后将被赋予的值作为本赋值表达式的值

举几个例子

- int n = 3.14 * 2
- float x = 10/4;

复合赋值表达式

变量　复合赋值运算符　表达式

- x += **表达式** $\Longleftrightarrow$ x = x + **表达式**

复合赋值表达式

变量　复合赋值运算符　表达式

- x += 表达式 $\Longleftrightarrow$ x = x + 表达式
- x -= 表达式 $\Longleftrightarrow$ x = x - 表达式

复合赋值表达式

变量　复合赋值运算符　表达式

- x += 表达式 $\Longleftrightarrow$ x = x + 表达式
- x -= 表达式 $\Longleftrightarrow$ x = x - 表达式
- x *= 表达式 $\Longleftrightarrow$ x = x * 表达式

复合赋值表达式

变量 复合赋值运算符 表达式

- x += 表达式 $\Longleftrightarrow$ x = x + 表达式
- x -= 表达式 $\Longleftrightarrow$ x = x - 表达式
- x *= 表达式 $\Longleftrightarrow$ x = x * 表达式
- x /= 表达式 $\Longleftrightarrow$ x = x / 表达式

复合赋值表达式

变量　复合赋值运算符　表达式

- x += 表达式 $\Longleftrightarrow$ x = x + 表达式
- x -= 表达式 $\Longleftrightarrow$ x = x - 表达式
- x *= 表达式 $\Longleftrightarrow$ x = x * 表达式
- x /= 表达式 $\Longleftrightarrow$ x = x / 表达式
- x %= 表达式 $\Longleftrightarrow$ x = x % 表达式

复合赋值表达式

变量　复合赋值运算符　表达式

- x += 表达式 $\Longleftrightarrow$ x = x + 表达式
- x -= 表达式 $\Longleftrightarrow$ x = x - 表达式
- x *= 表达式 $\Longleftrightarrow$ x = x * 表达式
- x /= 表达式 $\Longleftrightarrow$ x = x / 表达式
- x %= 表达式 $\Longleftrightarrow$ x = x % 表达式

举几个例子

复合赋值表达式

变量　复合赋值运算符　表达式

- x += 表达式 $\Longleftrightarrow$ x = x + 表达式
- x -= 表达式 $\Longleftrightarrow$ x = x - 表达式
- x *= 表达式 $\Longleftrightarrow$ x = x * 表达式
- x /= 表达式 $\Longleftrightarrow$ x = x / 表达式
- x %= 表达式 $\Longleftrightarrow$ x = x % 表达式

举几个例子

- x += 5;

复合赋值表达式

变量 复合赋值运算符 表达式

- x += 表达式 $\Longleftrightarrow$ x = x + 表达式
- x -= 表达式 $\Longleftrightarrow$ x = x - 表达式
- x *= 表达式 $\Longleftrightarrow$ x = x * 表达式
- x /= 表达式 $\Longleftrightarrow$ x = x / 表达式
- x %= 表达式 $\Longleftrightarrow$ x = x % 表达式

举几个例子

- x += 5;
- x += x * 2 + y;

复合赋值表达式

变量　复合赋值运算符　表达式

- x += 表达式 $\Longleftrightarrow$ x = x + 表达式
- x -= 表达式 $\Longleftrightarrow$ x = x - 表达式
- x *= 表达式 $\Longleftrightarrow$ x = x * 表达式
- x /= 表达式 $\Longleftrightarrow$ x = x / 表达式
- x %= 表达式 $\Longleftrightarrow$ x = x % 表达式

举几个例子

- x += 5;
- x += x * 2 + y;
- x *= x;

复合赋值表达式

变量 复合赋值运算符 表达式

- x += 表达式 $\Longleftrightarrow$ x = x + 表达式
- x -= 表达式 $\Longleftrightarrow$ x = x - 表达式
- x *= 表达式 $\Longleftrightarrow$ x = x * 表达式
- x /= 表达式 $\Longleftrightarrow$ x = x / 表达式
- x %= 表达式 $\Longleftrightarrow$ x = x % 表达式

举几个例子

- x += 5;
- x += x * 2 + y;
- x *= x;
- y *= x + 2;

复合赋值表达式

变量　复合赋值运算符　表达式

- x += 表达式 $\Longleftrightarrow$ x = x + 表达式
- x -= 表达式 $\Longleftrightarrow$ x = x - 表达式
- x *= 表达式 $\Longleftrightarrow$ x = x * 表达式
- x /= 表达式 $\Longleftrightarrow$ x = x / 表达式
- x %= 表达式 $\Longleftrightarrow$ x = x % 表达式

举几个例子

- x += 5;
- x += x * 2 + y;
- x *= x;
- y *= x + 2;

复合赋值表达式

变量 复合赋值运算符 表达式

- x += 表达式 $\Longleftrightarrow$ x = x + 表达式
- x -= 表达式 $\Longleftrightarrow$ x = x - 表达式
- x *= 表达式 $\Longleftrightarrow$ x = x * 表达式
- x /= 表达式 $\Longleftrightarrow$ x = x / 表达式
- x %= 表达式 $\Longleftrightarrow$ x = x % 表达式

举几个例子

- x += 5;
- x += x * 2 + y;
- x *= x;
- y *= x + 2;

注意：首先计算右侧表达式的值，然后执行复合赋值

赋值表达式优先级和结合性

- 优先级非常低，几乎是最低的，倒数老二

赋值表达式优先级和结合性

- 优先级非常低，几乎是最低的，倒数老二
- 从右开始结合

赋值表达式优先级和结合性

- 优先级非常低，几乎是最低的，倒数老二
- 从右开始结合

再举几个例子

赋值表达式优先级和结合性

- 优先级非常低，几乎是最低的，倒数老二
- 从右开始结合

再举几个例子

- $x = y = z = 3; \Longleftrightarrow x = (y = (z = 3));$

赋值表达式优先级和结合性

- 优先级非常低，几乎是最低的，倒数老二
- 从右开始结合

再举几个例子

- $x = y = z = 3; \iff x = (y = (z = 3));$
- 假设 $y = 2;$ 那么：
  $x = y \mathrel{+}= z = 3; \iff x = (y \mathrel{+}= (z = 3));$

赋值表达式优先级和结合性

- 优先级非常低，几乎是最低的，倒数老二
- 从右开始结合

再举几个例子

- $x = y = z = 3; \iff x = (y = (z = 3));$
- 假设 $y = 2;$ 那么：
  $x = y \mathrel{+}= z = 3; \iff x = (y \mathrel{+}= (z = 3));$
- $x = y + z = 5; \iff ???;$

赋值表达式优先级和结合性

- 优先级非常低，几乎是最低的，倒数老二
- 从右开始结合

再举几个例子

- $x = y = z = 3; \iff x = (y = (z = 3));$
- 假设 $y = 2;$ 那么：
  $x = y \mathrel{+}= z = 3; \iff x = (y \mathrel{+}= (z = 3));$
- $x = y + z = 5; \iff ???;$
  - $x = y + z = 5; \iff x = ((y+z) = 5);$

赋值表达式优先级和结合性

- 优先级非常低，几乎是最低的，倒数老二
- 从右开始结合

再举几个例子

- $x = y = z = 3; \iff x = (y = (z = 3));$
- 假设 $y = 2;$ 那么：
  $x = y \mathrel{+}= z = 3; \iff x = (y \mathrel{+}= (z = 3));$
- $x = y + z = 5; \iff ???;$
  - $x = y + z = 5; \iff x = ((y+z) = 5);$
  - 不能执行 $y+z = 5$，因为 $y+z$ 不是一个可赋值的变量

关系表达式

通过关系运算符连接起来的算式：

< > <= >= == !=

关系表达式

通过关系运算符连接起来的算式：

`<`　`>`　`<=`　`>=`　`==`　`!=`

- 优先级仅次于 $+-$。（记住：比较大小要使用减法的）

关系表达式

通过关系运算符连接起来的算式：

`<`　`>`　`<=`　`>=`　`==`　`!=`

- 优先级仅次于 $+-$。（记住：比较大小要使用减法的）

举几个例子

关系表达式

通过关系运算符连接起来的算式：

$$< \quad > \quad <= \quad >= \quad == \quad !=$$

- 优先级仅次于 $+-$。（记住：比较大小要使用减法的）

举几个例子

- a > b == c

关系表达式

通过关系运算符连接起来的算式：

$$< \quad > \quad <= \quad >= \quad == \quad !=$$

- 优先级仅次于 $+-$。（记住：比较大小要使用减法的）

举几个例子

- a $>$ b $==$ c
- d $=$ a $>$ b

关系表达式

通过关系运算符连接起来的算式：

$$< \quad > \quad <= \quad >= \quad == \quad !=$$

- 优先级仅次于 $+-$。（记住：比较大小要使用减法的）

举几个例子

- a $>$ b $==$ c
- d $=$ a $>$ b
- ch $>$ 'a' $+$ 1

关系表达式

通过关系运算符连接起来的算式：

$$< \quad > \quad <= \quad >= \quad == \quad !=$$

- 优先级仅次于 $+-$。（记住：比较大小要使用减法的）

举几个例子

- a $>$ b $==$ c
- d $=$ a $>$ b
- ch $>$ 'a' $+$ 1
- d $=$ a $+$ b $>$ c

关系表达式

通过关系运算符连接起来的算式：

$$< \quad > \quad <= \quad >= \quad == \quad !=$$

- 优先级仅次于 $+-$。（记住：比较大小要使用减法的）

举几个例子

- a > b == c
- d = a > b
- ch > 'a' + 1
- d = a + b > c
- 3 <= x <= 5

关系表达式

通过关系运算符连接起来的算式：

$$< \quad > \quad <= \quad >= \quad == \quad !=$$

- 优先级仅次于 $+-$。（记住：比较大小要使用减法的）

举几个例子

- a > b == c
- d = a > b
- ch > 'a' + 1
- d = a + b > c
- 3 <= x <= 5
- b - 1 == a != c

关系表达式

通过关系运算符连接起来的算式：

$$< \quad > \quad <= \quad >= \quad == \quad !=$$

- 优先级仅次于 $+-$。（记住：比较大小要使用减法的）

举几个例子

- a > b == c $\iff$ (a > b) == c
- d = a > b
- ch > 'a' + 1
- d = a + b > c
- 3 <= x <= 5
- b - 1 == a != c

关系表达式

通过关系运算符连接起来的算式：

$$< \quad > \quad <= \quad >= \quad == \quad !=$$

- 优先级仅次于 $+-$。（记住：比较大小要使用减法的）

举几个例子

- a $>$ b $==$ c $\iff$ (a $>$ b) $==$ c
- d $=$ a $>$ b $\iff$ d $=$ (a $>$ b)
- ch $>$ 'a' $+$ 1
- d $=$ a $+$ b $>$ c
- 3 $<=$ x $<=$ 5
- b - 1 $==$ a $!=$ c

关系表达式

通过关系运算符连接起来的算式：

$$< \quad > \quad <= \quad >= \quad == \quad !=$$

- 优先级仅次于 $+-$。（记住：比较大小要使用减法的）

举几个例子

- a $>$ b $==$ c $\iff$ (a $>$ b) $==$ c
- d $=$ a $>$ b $\iff$ d $=$ (a $>$ b)
- ch $>$ 'a' $+$ 1 $\iff$ ch $>$ ('a' $+$ 1)
- d $=$ a $+$ b $>$ c
- 3 $<=$ x $<=$ 5
- b - 1 $==$ a $!=$ c

关系表达式

通过关系运算符连接起来的算式：

$$< \quad > \quad <= \quad >= \quad == \quad !=$$

- 优先级仅次于 $+-$。（记住：比较大小要使用减法的）

举几个例子

- a $>$ b $==$ c $\iff$ (a $>$ b) $==$ c
- d $=$ a $>$ b $\iff$ d $=$ (a $>$ b)
- ch $>$ 'a' $+$ 1 $\iff$ ch $>$ ('a' $+$ 1)
- d $=$ a $+$ b $>$ c $\iff$ d $=$ ((a $+$ b) $>$ c)
- 3 $<=$ x $<=$ 5
- b - 1 $==$ a $!=$ c

关系表达式

通过关系运算符连接起来的算式：

$$< \quad > \quad <= \quad >= \quad == \quad !=$$

- 优先级仅次于 $+-$。（记住：比较大小要使用减法的）

举几个例子

- a > b == c $\iff$ (a > b) == c
- d = a > b $\iff$ d = (a > b)
- ch > 'a' + 1 $\iff$ ch > ('a' + 1)
- d = a + b > c $\iff$ d = ((a + b) > c)
- 3 <= x <= 5 $\iff$ (3 <= x) <= 5
- b - 1 == a != c

关系表达式

通过关系运算符连接起来的算式：

$$< \quad > \quad <= \quad >= \quad == \quad !=$$

- 优先级仅次于 $+-$。（记住：比较大小要使用减法的）

举几个例子

- a > b == c $\iff$ (a > b) == c
- d = a > b $\iff$ d = (a > b)
- ch > 'a' + 1 $\iff$ ch > ('a' + 1)
- d = a + b > c $\iff$ d = ((a + b) > c)
- 3 <= x <= 5 $\iff$ (3 <= x) <= 5
- b - 1 == a != c $\iff$ ((b - 1) == a) != c

关系表达式

假设：

```
char ch='w'; int x=10, a=2, b=3, c=1, d;
```

那么下面表达式的值是多少？

- a > b == c

关系表达式

假设：

```
char ch='w'; int x=10, a=2, b=3, c=1, d;
```

那么下面表达式的值是多少？

- a > b == c　　值为 0

关系表达式

假设：

```
char ch='w'; int x=10, a=2, b=3, c=1, d;
```

那么下面表达式的值是多少？

- a > b == c　　值为 0
- d = a < b

关系表达式

假设：

```
char ch='w'; int x=10, a=2, b=3, c=1, d;
```

那么下面表达式的值是多少？

- a > b == c　　值为 0
- d = a < b　　值为 1

关系表达式

假设：

```
char ch='w'; int x=10, a=2, b=3, c=1, d;
```

那么下面表达式的值是多少？

- a > b == c　　值为 0
- d = a < b　　值为 1
- ch > 'a' + 1

关系表达式

假设：

```
char ch='w'; int x=10, a=2, b=3, c=1, d;
```

那么下面表达式的值是多少？

- a > b == c　　值为 0
- d = a < b　　值为 1
- ch > 'a' + 1　　值为 1

关系表达式

假设：

```
char ch='w'; int x=10, a=2, b=3, c=1, d;
```

那么下面表达式的值是多少？

- a > b == c　　值为 0
- d = a < b　　值为 1
- ch > 'a' + 1　　值为 1
- d = a + b > c

关系表达式

假设：

```
char ch='w'; int x=10, a=2, b=3, c=1, d;
```

那么下面表达式的值是多少？

- a > b == c　　值为 0
- d = a < b　　值为 1
- ch > 'a' + 1　　值为 1
- d = a + b > c　　值为 1

关系表达式

假设：

```
char ch='w'; int x=10, a=2, b=3, c=1, d;
```

那么下面表达式的值是多少？

- a > b == c　　值为 0
- d = a < b　　值为 1
- ch > 'a' + 1　　值为 1
- d = a + b > c　　值为 1
- b - 1 == a != c

关系表达式

假设：

```
char ch='w'; int x=10, a=2, b=3, c=1, d;
```

那么下面表达式的值是多少？

- a > b == c　　值为 0
- d = a < b　　值为 1
- ch > 'a' + 1　　值为 1
- d = a + b > c　　值为 1
- b - 1 == a != c　　值为 0

关系表达式

假设：

```
char ch='w'; int x=10, a=2, b=3, c=1, d;
```

那么下面表达式的值是多少？

- a > b == c　　值为 0
- d = a < b　　值为 1
- ch > 'a' + 1　　值为 1
- d = a + b > c　　值为 1
- b - 1 == a != c　　值为 0
- 3 <= x <= 5

关系表达式

假设：

```
char ch='w'; int x=10, a=2, b=3, c=1, d;
```

那么下面表达式的值是多少？

- a > b == c　　值为 0
- d = a < b　　值为 1
- ch > 'a' + 1　　值为 1
- d = a + b > c　　值为 1
- b - 1 == a != c　　值为 0
- 3 <= x <= 5　　值为 1

逻辑运算

逻辑运算的对象为逻辑值，逻辑运算的结果：真、假

逻辑运算

逻辑运算的对象为逻辑值，逻辑运算的结果：真、假

- 真：1
- 假：0

逻辑运算

逻辑运算的对象为逻辑值，逻辑运算的结果：真、假

- 真：1
- 假：0

逻辑与：　&&

a && b 为真 $\Longleftrightarrow$ a 和 b 都为真

逻辑运算

逻辑运算的对象为逻辑值，逻辑运算的结果：真、假

- 真：1
- 假：0

逻辑与：　&&

a && b 为真 $\Longleftrightarrow$ a 和 b 都为真

逻辑或：　||

a || b 为真 $\Longleftrightarrow$ a 和 b 至少一个为真（a,b 不全为假）

逻辑运算

逻辑运算的对象为逻辑值，逻辑运算的结果：真、假

- 真：1
- 假：0

逻辑与：　&&

a && b 为真 $\Longleftrightarrow$ a 和 b 都为真

逻辑或：　||

a || b 为真 $\Longleftrightarrow$ a 和 b 至少一个为真（a,b 不全为假）

逻辑非：　!（惊叹号）

!a 为真 $\Longleftrightarrow$ a 为假
!a 为假 $\Longleftrightarrow$ a 为真

逻辑运算应用

判断字符 ch 是否为数字字符

```
ch>='0' && ch<='9'
```

```
if( ch>='0' && ch <='9' )
    printf("It is a digital\n");
else
    printf("It is NOT a digital\n");
```

逻辑运算应用

判断字符 ch 是否为小写字母

```
ch>='a'  &&  ch<='z'
```

逻辑运算应用

判断字符 ch 是否为小写字母

```
ch>='a' && ch<='z'
```

判断字符 ch 是否为大写字母

```
ch>='A' && ch<='Z'
```

逻辑运算应用

判断字符 ch 是否为小写字母

```
ch>='a' && ch<='z'
```

判断字符 ch 是否为大写字母

```
ch>='A' && ch<='Z'
```

判断字符 ch 是否为写字母

```
(ch>='a' && ch<='z') || (ch>='A' && ch<='Z')
```

逻辑运算符的优先级和结合性

优先级

- ! （逻辑非）

逻辑运算符的优先级和结合性

优先级

- ! （逻辑非）
- 算术运算符

逻辑运算符的优先级和结合性

优先级

- ! （逻辑非）
- 算术运算符
- 关系运算符

逻辑运算符的优先级和结合性

优先级

- ! （逻辑非）
- 算术运算符
- 关系运算符
- && （逻辑与）

逻辑运算符的优先级和结合性

优先级

- ! （逻辑非）
- 算术运算符
- 关系运算符
- && （逻辑与）
- || （逻辑或）

逻辑运算符的优先级和结合性

优先级

- ! （逻辑非）
- 算术运算符
- 关系运算符
- && （逻辑与）
- || （逻辑或）
- 赋值运算符

逻辑运算符的优先级和结合性

优先级

- ! （逻辑非）
- 算术运算符
- 关系运算符
- && （逻辑与）
- || （逻辑或）
- 赋值运算符

结合性

从左开始结合

逻辑运算表达式的运算

用逻辑运算符将关系表达式或逻辑量连接起来的式子

假设定义了如下变量：

```
char ch = 'w';
int a = 2, b = 0, c = 0;
float x = 3.0;
```

那么下面的是逻辑表达式吗？值为多少？

- a && b

逻辑运算表达式的运算

用逻辑运算符将关系表达式或逻辑量连接起来的式子

假设定义了如下变量：

```
char ch = 'w';
int a = 2, b = 0, c = 0;
float x = 3.0;
```

那么下面的是逻辑表达式吗？值为多少？

- a && b　　0

逻辑运算表达式的运算

用逻辑运算符将关系表达式或逻辑量连接起来的式子

假设定义了如下变量：

```
char ch = 'w';
int a = 2, b = 0, c = 0;
float x = 3.0;
```

那么下面的是逻辑表达式吗？值为多少？

- a && b　　0
- a || b && c

逻辑运算表达式的运算

用逻辑运算符将关系表达式或逻辑量连接起来的式子

假设定义了如下变量：

```
char ch = 'w';
int a = 2, b = 0, c = 0;
float x = 3.0;
```

那么下面的是逻辑表达式吗？值为多少？

- a && b　　0
- a || b && c　　1

逻辑运算表达式的运算

用逻辑运算符将关系表达式或逻辑量连接起来的式子

假设定义了如下变量：

```
char ch = 'w';
int a = 2, b = 0, c = 0;
float x = 3.0;
```

那么下面的是逻辑表达式吗？值为多少？

- a && b　　0
- a || b && c　　1
- !a && b

逻辑运算表达式的运算

用逻辑运算符将关系表达式或逻辑量连接起来的式子

假设定义了如下变量：

```
char ch = 'w';
int a = 2, b = 0, c = 0;
float x = 3.0;
```

那么下面的是逻辑表达式吗？值为多少？

- a && b　　0
- a || b && c　　1
- !a && b　　0

逻辑运算表达式的运算

用逻辑运算符将关系表达式或逻辑量连接起来的式子

假设定义了如下变量：

```
char ch = 'w';
int a = 2, b = 0, c = 0;
float x = 3.0;
```

那么下面的是逻辑表达式吗？值为多少？

- a && b　　0
- a || b && c　　1
- !a && b　　0
- a || 3+10 && 2

逻辑运算表达式的运算

用逻辑运算符将关系表达式或逻辑量连接起来的式子

假设定义了如下变量：

```
char ch = 'w';
int a = 2, b = 0, c = 0;
float x = 3.0;
```

那么下面的是逻辑表达式吗？值为多少？

- a && b　　0
- a || b && c　　1
- !a && b　　0
- a || 3+10 && 2　　1

逻辑运算表达式的运算

用逻辑运算符将关系表达式或逻辑量连接起来的式子

假设定义了如下变量：

```
char ch = 'w';
int a = 2, b = 0, c = 0;
float x = 3.0;
```

那么下面的是逻辑表达式吗？值为多少？

- a && b　　0
- a || b && c　　1
- !a && b　　0
- a || 3+10 && 2　　1
- !(x == 2)

逻辑运算表达式的运算

用逻辑运算符将关系表达式或逻辑量连接起来的式子

假设定义了如下变量：

```
char ch = 'w';
int a = 2, b = 0, c = 0;
float x = 3.0;
```

那么下面的是逻辑表达式吗？值为多少？

- a && b　　0
- a || b && c　　1
- !a && b　　0
- a || 3+10 && 2　　1
- !(x == 2)　　1

逻辑运算表达式的运算

用逻辑运算符将关系表达式或逻辑量连接起来的式子

假设定义了如下变量：

```
char ch = 'w';
int a = 2, b = 0, c = 0;
float x = 3.0;
```

那么下面的是逻辑表达式吗？值为多少？

- a && b　　0
- a || b && c　　1
- !a && b　　0
- a || 3+10 && 2　　1
- !(x == 2)　　1
- !x == 2

逻辑运算表达式的运算

用逻辑运算符将关系表达式或逻辑量连接起来的式子

假设定义了如下变量：

```
char ch = 'w';
int a = 2, b = 0, c = 0;
float x = 3.0;
```

那么下面的是逻辑表达式吗？值为多少？

- a && b　　0
- a || b && c　　1
- !a && b　　0
- a || 3+10 && 2　　1
- !(x == 2)　　1
- !x == 2　　0

逻辑运算表达式的运算

用逻辑运算符将关系表达式或逻辑量连接起来的式子

假设定义了如下变量：

```
char ch = 'w';
int a = 2, b = 0, c = 0;
float x = 3.0;
```

那么下面的是逻辑表达式吗？值为多少？

- a && b　　0
- a || b && c　　1
- !a && b　　0
- a || 3+10 && 2　　1
- !(x == 2)　　1
- !x == 2　　0
- ch || b

逻辑运算表达式的运算

用逻辑运算符将关系表达式或逻辑量连接起来的式子

假设定义了如下变量：

```
char ch = 'w';
int a = 2, b = 0, c = 0;
float x = 3.0;
```

那么下面的是逻辑表达式吗？值为多少？

- a && b　　0
- a || b && c　　1
- !a && b　　0
- a || 3+10 && 2　　1
- !(x == 2)　　1
- !x == 2　　0
- ch || b　　1

逻辑运算表达式的计算过程

```c
int fa()
{
    printf("A");
    return 1;
}
```

```c
int fb()
{
    printf("B");
    return 1;
}
```

假设定义了上述两个函数，思考：下列程序的输出是什么？

逻辑运算表达式的计算过程

```c
int fa()
{
    printf("A");
    return 1;
}
```

```c
int fb()
{
    printf("B");
    return 1;
}
```

假设定义了上述两个函数，思考：下列程序的输出是什么？

```c
if(  fa() && fb() ) printf("C\n");
```

逻辑运算表达式的计算过程

```
int fa()
{
    printf("A");
    return 1;
}
```

```
int fb()
{
    printf("B");
    return 1;
}
```

假设定义了上述两个函数，思考：下列程序的输出是什么？

```
if(  fa() && fb() ) printf("C\n");
```

ABC

逻辑运算表达式的计算过程

```c
int fa()
{
    printf("A");
    return 1;
}
```

```c
int fb()
{
    printf("B");
    return 1;
}
```

假设定义了上述两个函数，思考：下列程序的输出是什么？

```c
if(   fa() && fb() ) printf("C\n");
```

ABC

```c
if( ! fa() && fb() ) printf("C\n");
```

逻辑运算表达式的计算过程

```c
int fa()
{
    printf("A");
    return 1;
}
```

```c
int fb()
{
    printf("B");
    return 1;
}
```

假设定义了上述两个函数，思考：下列程序的输出是什么？

```c
if(   fa() && fb() ) printf("C\n");
```

ABC

```c
if( ! fa() && fb() ) printf("C\n");
```

A

逻辑运算表达式的计算过程

```c
int fa()
{
    printf("A");
    return 1;
}
```

```c
int fb()
{
    printf("B");
    return 1;
}
```

假设定义了上述两个函数，思考：下列程序的输出是什么？

```c
if(   fa() && fb() ) printf("C\n");
```

ABC

```c
if( ! fa() && fb() ) printf("C\n");
```

A

```c
if(   fa() || fb() ) printf("C\n");
```

逻辑运算表达式的计算过程

```c
int fa()
{
    printf("A");
    return 1;
}
```

```c
int fb()
{
    printf("B");
    return 1;
}
```

假设定义了上述两个函数，思考：下列程序的输出是什么？

```c
if(   fa() && fb() ) printf("C\n");
```

ABC

```c
if( ! fa() && fb() ) printf("C\n");
```

A

```c
if(   fa() || fb() ) printf("C\n");
```

AC

逻辑运算表达式的计算过程

```c
int fa()
{
    printf("A");
    return 1;
}
```

```c
int fb()
{
    printf("B");
    return 1;
}
```

假设定义了上述两个函数，思考：下列程序的输出是什么？

```c
if(   fa() && fb() ) printf("C\n");
if( ! fa() && fb() ) printf("C\n");
if(   fa() || fb() ) printf("C\n");
if( ! fa() || fb() ) printf("C\n");
```

ABC

A

AC

逻辑运算表达式的计算过程

```c
int fa()
{
    printf("A");
    return 1;
}
```

```c
int fb()
{
    printf("B");
    return 1;
}
```

假设定义了上述两个函数，思考：下列程序的输出是什么？

```c
if(   fa() && fb() ) printf("C\n");
```
ABC

```c
if( ! fa() && fb() ) printf("C\n");
```
A

```c
if(   fa() || fb() ) printf("C\n");
```
AC

```c
if( ! fa() || fb() ) printf("C\n");
```
ABC

逻辑运算表达式的计算过程

从左至右开始计算，当可以提前确定表达式值的时候，
停止后面的计算

表达式：exp1 && exp2 的计算过程

逻辑运算表达式的计算过程

从左至右开始计算，当可以提前确定表达式值的时候，
停止后面的计算

表达式：exp1 && exp2 的计算过程

- 先计算 exp1 的值

逻辑运算表达式的计算过程

从左至右开始计算，当可以提前确定表达式值的时候，
停止后面的计算

表达式：exp1 && exp2 的计算过程

- 先计算 exp1 的值
- 若 exp1 的值为 0，则 exp1 && exp2 的值为 0，《结束》

逻辑运算表达式的计算过程

从左至右开始计算，当可以提前确定表达式值的时候，
停止后面的计算

表达式：exp1 && exp2 的计算过程

- 先计算 exp1 的值
- 若 exp1 的值为 0，则 exp1 && exp2 的值为 0，《结束》
- 否则，计算 exp2 的值，作为 exp1 && exp2 的值

逻辑运算表达式的计算过程

从左至右开始计算，当可以提前确定表达式值的时候，
停止后面的计算

表达式：exp1 && exp2 的计算过程

- 先计算 exp1 的值
- 若 exp1 的值为 0，则 exp1 && exp2 的值为 0，《结束》
- 否则，计算 exp2 的值，作为 exp1 && exp2 的值

表达式：exp1 || exp2 的计算过程

逻辑运算表达式的计算过程

从左至右开始计算，当可以提前确定表达式值的时候，
停止后面的计算

表达式：exp1 && exp2 的计算过程

- 先计算 exp1 的值
- 若 exp1 的值为 0，则 exp1 && exp2 的值为 0，《结束》
- 否则，计算 exp2 的值，作为 exp1 && exp2 的值

表达式：exp1 || exp2 的计算过程

- 先计算 exp1 的值

逻辑运算表达式的计算过程

从左至右开始计算，当可以提前确定表达式值的时候，
停止后面的计算

表达式：exp1 && exp2 的计算过程

- 先计算 exp1 的值
- 若 exp1 的值为 0，则 exp1 && exp2 的值为 0，《结束》
- 否则，计算 exp2 的值，作为 exp1 && exp2 的值

表达式：exp1 || exp2 的计算过程

- 先计算 exp1 的值
- 若 exp1 的值为 1，则表达式的值为 1，《结束》

逻辑运算表达式的计算过程

从左至右开始计算，当可以提前确定表达式值的时候，
停止后面的计算

表达式：exp1 && exp2 的计算过程

- 先计算 exp1 的值
- 若 exp1 的值为 0，则 exp1 && exp2 的值为 0，《结束》
- 否则，计算 exp2 的值，作为 exp1 && exp2 的值

表达式：exp1 || exp2 的计算过程

- 先计算 exp1 的值
- 若 exp1 的值为 1，则表达式的值为 1，《结束》
- 否则，计算 exp2 的值，作为 exp1 || exp2 的值

写出符合要求的逻辑表达式

- x 为零

写出符合要求的逻辑表达式

- x 为零
  - x == 0

写出符合要求的逻辑表达式

- x 为零
  - x == 0
  - !x

写出符合要求的逻辑表达式

- x 为零
  - x == 0
  - !x
- x 不为零

写出符合要求的逻辑表达式

- x 为零
  - x == 0
  - !x
- x 不为零
  - x != 0

写出符合要求的逻辑表达式

- x 为零
  - x == 0
  - !x
- x 不为零
  - x != 0
  - x

写出符合要求的逻辑表达式

- x 为零
  - x == 0
  - !x
- x 不为零
  - x != 0
  - x
- x 和 y 不同时为零

写出符合要求的逻辑表达式

- x 为零
  - x == 0
  - !x
- x 不为零
  - x != 0
  - x
- x 和 y 不同时为零
  - !(x == 0 && y==0)

写出符合要求的逻辑表达式

- x 为零
  - x == 0
  - !x
- x 不为零
  - x != 0
  - x
- x 和 y 不同时为零
  - !(x == 0 && y==0)
  - x != 0 || y!=0

写出符合要求的逻辑表达式

- x 为零
  - x == 0
  - !x
- x 不为零
  - x != 0
  - x
- x 和 y 不同时为零
  - !(x == 0 && y==0)
  - x != 0 || y!=0
  - x || y

条件表达式和三目运算符？：

exp1 ? exp2 : exp3

条件表达式和三目运算符？：

exp1 ? exp2 : exp3

条件表达式的计算

条件表达式和三目运算符？：

exp1 ? exp2 : exp3

条件表达式的计算

if(expr1为真)
    条件表达式的值为expr2的值
else
    条件表达式的值为expr3的值

条件表达式和三目运算符？：

exp1 ? exp2 : exp3

条件表达式的计算

```
if( expr1为真 )
    条件表达式的值为expr2的值
else
    条件表达式的值为expr3的值
```

例如：

```
y = a > b ? a : b;   /*将a,b的最大值赋值给y*/
```

条件表达式和三目运算符？：

exp1 ? exp2 : exp3

条件表达式的计算

```
if( expr1为真 )
    条件表达式的值为expr2的值
else
    条件表达式的值为expr3的值
```

例如：

```
y = a > b ? a : b;   /*将a,b的最大值赋值给y*/

y = a < b ? a : b;   /*将a,b的最小值赋值给y*/
```

条件表达式和三目运算符？：

exp1 ? exp2 : exp3

条件表达式的计算

```
if( expr1为真 )
    条件表达式的值为expr2的值
else
    条件表达式的值为expr3的值
```

例如：

```
y = a > b ? a : b;   /*将a,b的最大值赋值给y*/

y = a < b ? a : b;   /*将a,b的最小值赋值给y*/

/* 将a、b、c中的最大值赋值给y */
y = a > b ? (a>c?a:c) : (b>c?b:c); /*嵌套*/
```

逗号表达式

expr1, expr2, ……, exprN

用逗号, 分隔开来的若干个表达式的排列

- **依次计算** expr 1 , expr 2 ,……, exprN

逗号表达式

expr1, expr2, ⋯⋯, exprN

用逗号, 分隔开来的若干个表达式的排列

- 依次计算 expr 1 , expr 2 ,⋯⋯, exprN
- 并将最后一个表达式 exprN 的值作为逗号表达式的值.

逗号表达式

expr1, expr2, ……, exprN

用逗号, 分隔开来的若干个表达式的排列

- 依次计算 expr 1 , expr 2 ,…… , exprN
- 并将最后一个表达式 exprN 的值作为逗号表达式的值.
- 逗号运算符的优先级最低

逗号表达式

expr1, expr2, ……, exprN

用逗号, 分隔开来的若干个表达式的排列

- 依次计算 expr 1 , expr 2 ,…… , exprN
- 并将最后一个表达式 exprN 的值作为逗号表达式的值.
- 逗号运算符的优先级最低

逗号表达式

expr1, expr2, ……, exprN

用逗号, 分隔开来的若干个表达式的排列

- 依次计算 expr 1 , expr 2 ,…… , exprN
- 并将最后一个表达式 exprN 的值作为逗号表达式的值.
- 逗号运算符的优先级最低

例如:

```
int a, b, c;
a=2, b=3, c=a+b;
```

逗号表达式

过去我们写下面的循环：

```
sum = 0;
k = 0;
for( j = 0;  j<100; j++) {
     sum += j * k;
     k += 2;
}
```

采用逗号表达式，现在可以这样写：

```
for( sum = 0, j = 0, k = 0;  j<100; j++, k+=2)
     sum += j * k;
```

内容提要

位运算

位运算是定义在整数类型数据上的一种运算，包括：

位逻辑运算 – 在数据的每一位 bit 上独立进行运算

位运算

位运算是定义在整数类型数据上的一种运算，包括：

位逻辑运算 – 在数据的每一位 bit 上独立进行运算

- 按位反 $\sim$

位运算

位运算是定义在整数类型数据上的一种运算，包括：

位逻辑运算 – 在数据的每一位 bit 上独立进行运算

- 按位反 $\sim$
- 按位与 $\&$

位运算

位运算是定义在整数类型数据上的一种运算，包括：

位逻辑运算 – 在数据的每一位 bit 上独立进行运算

- 按位反 $\sim$
- 按位与 &
- 按位或 |

位运算

位运算是定义在整数类型数据上的一种运算，包括：

位逻辑运算 – 在数据的每一位 bit 上独立进行运算

- 按位反 ~
- 按位与 &
- 按位或 |
- 按位异或 ^

位运算

位运算是定义在整数类型数据上的一种运算，包括：

位逻辑运算 – 在数据的每一位 bit 上独立进行运算

- 按位反 $\sim$
- 按位与 &
- 按位或 |
- 按位异或 ^

移位运算 – 在数据的所有的 bit 位依次进行移位

位运算

位运算是定义在整数类型数据上的一种运算，包括：

位逻辑运算 – 在数据的每一位 bit 上独立进行运算

- 按位反 $\sim$
- 按位与 &
- 按位或 |
- 按位异或 ^

移位运算 – 在数据的所有的 bit 位依次进行移位

- 左移位 $<<$

位运算

位运算是定义在整数类型数据上的一种运算，包括：

位逻辑运算 – 在数据的每一位 bit 上独立进行运算

- 按位反 $\sim$
- 按位与 &
- 按位或 |
- 按位异或 ^

移位运算 – 在数据的所有的 bit 位依次进行移位位

- 左移位 $<<$
- 右移位 $>>$

位逻辑运算

考虑两个 8 位整数 (16 位和 32 位整数也一样):

$$\mathsf{a} = 0\ 0\ 0\ 0\ 1\ 0\ 1\ 1$$
$$\mathsf{b} = 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1$$

- 按位反: $\sim \mathsf{a} = 1\ 1\ 1\ 1\ 0\ 1\ 0\ 0$

位逻辑运算

考虑两个 8 位整数 (16 位和 32 位整数也一样):

$$\mathtt{a} = 0\ 0\ 0\ 0\ 1\ 0\ 1\ 1$$
$$\mathtt{b} = 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1$$

- 按位反: $\sim \mathtt{a} = 1\ 1\ 1\ 1\ 0\ 1\ 0\ 0$
- 按位与: $\mathtt{a}\ \&\ \mathtt{b} = 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1$

位逻辑运算

考虑两个 8 位整数 (16 位和 32 位整数也一样):

$$\begin{aligned} a &= 0\ 0\ 0\ 0\ 1\ 0\ 1\ 1 \\ b &= 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1 \end{aligned}$$

- 按位反: $\sim a = 1\ 1\ 1\ 1\ 0\ 1\ 0\ 0$
- 按位与: $a\ \&\ b = 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1$
- 按位或: $a\ |\ b = 0\ 1\ 1\ 0\ 1\ 1\ 1\ 1$

位逻辑运算

考虑两个 8 位整数 (16 位和 32 位整数也一样):

$$\begin{aligned} a &= 0\ 0\ 0\ 0\ 1\ 0\ 1\ 1 \\ b &= 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1 \end{aligned}$$

- 按位反: $\sim a = 1\ 1\ 1\ 1\ 0\ 1\ 0\ 0$
- 按位与: $a\ \&\ b = 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1$
- 按位或: $a\ |\ b = 0\ 1\ 1\ 0\ 1\ 1\ 1\ 1$
- 按位异或: $a\ \hat{}\ b = 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0$

位逻辑运算

考虑两个 8 位整数 (16 位和 32 位整数也一样):

$$\begin{aligned} a &= 0\ 0\ 0\ 0\ 1\ 0\ 1\ 1 \\ b &= 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1 \end{aligned}$$

- 按位反: $\sim a = 1\ 1\ 1\ 1\ 0\ 1\ 0\ 0$
- 按位与: $a\ \&\ b = 0\ 0\ 0\ 0\ 1\ 0\ 0\ 1$
- 按位或: $a\ |\ b = 0\ 1\ 1\ 0\ 1\ 1\ 1\ 1$
- 按位异或: $a\ \hat{}\ b = 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0$
  - 互异为真

异或运算（^）

^ 可以理解为抛弃进位的加法。例如:

$1 \oplus 1 = 0;$

$0 \oplus 0 = 0;$

$1 \oplus 0 = 1;$

$0 \oplus 1 = 1;$

$a \oplus b =$ 末位 $(a + b)$

异或运算（^）

^ 可以理解为抛弃进位的加法。例如:

$1 \oplus 1 = 0$;

$0 \oplus 0 = 0$;

$1 \oplus 0 = 1$;

$0 \oplus 1 = 1$;

$a \oplus b =$ 末位 $(a + b)$

异或运算 ^ 具有可结合性，证明：

- $a \hat{} b \hat{} c =$ 末位 $(a + b) \hat{} c$

异或运算（^）

^ 可以理解为抛弃进位的加法。例如:

$1 \oplus 1 = 0$;

$0 \oplus 0 = 0$;

$1 \oplus 0 = 1$;

$0 \oplus 1 = 1$;

$a \oplus b =$ 末位 $(a + b)$

异或运算 ^ 具有可结合性，证明：

- $a \hat{} b \hat{} c =$ 末位 $(a + b) \hat{} c =$ 末位 $(a + b + c)$

异或运算（^）

^ 可以理解为抛弃进位的加法。例如:

$1 \oplus 1 = 0$;

$0 \oplus 0 = 0$;

$1 \oplus 0 = 1$;

$0 \oplus 1 = 1$;

$a \oplus b =$ 末位 $(a + b)$

异或运算 ^ 具有可结合性，证明：

- a ^ b ^ c = 末位 (a + b) ^ c = 末位 (a + b + c)
- a ^ (b ^ c) = a ^ 末位 (b + c)

异或运算（^）

^ 可以理解为抛弃进位的加法。例如:

$1 \oplus 1 = 0$;

$0 \oplus 0 = 0$;

$1 \oplus 0 = 1$;

$0 \oplus 1 = 1$;

$a \oplus b =$ 末位 $(a + b)$

异或运算 ^ 具有可结合性，证明：

- a ^ b ^ c = 末位 (a + b) ^ c = 末位 (a + b + c)
- a ^ (b ^ c) = a ^ 末位 (b + c) = 末位 (a + b + c)

异或运算（^）

^ 可以理解为抛弃进位的加法。例如:

$1 \oplus 1 = 0$;

$0 \oplus 0 = 0$;

$1 \oplus 0 = 1$;

$0 \oplus 1 = 1$;

$a \oplus b =$ 末位 $(a + b)$

异或运算 ^ 具有可结合性，证明：

- a ^ b ^ c = 末位 (a + b) ^ c = 末位 (a + b + c)
- a ^ (b ^ c) = a ^ 末位 (b + c) = 末位 (a + b + c)

  因此 a ^ b ^ c = a ^ (b ^ c)

异或运算（^）

^ 可以理解为抛弃进位的加法。例如:

$1 \oplus 1 = 0$;

$0 \oplus 0 = 0$;

$1 \oplus 0 = 1$;

$0 \oplus 1 = 1$;

$a \oplus b =$ 末位 $(a + b)$

异或运算 ^ 具有可结合性，证明：

- a ^ b ^ c = 末位 (a + b) ^ c = 末位 (a + b + c)
- a ^ (b ^ c) = a ^ 末位 (b + c) = 末位 (a + b + c)

  因此 a ^ b ^ c = a ^ (b ^ c)

异或运算（^）

^ 可以理解为抛弃进位的加法。例如:

$1 \oplus 1 = 0$;

$0 \oplus 0 = 0$;

$1 \oplus 0 = 1$;

$0 \oplus 1 = 1$;

$a \oplus b =$ 末位 $(a + b)$

异或运算 ^ 具有可结合性，证明:

- a ^ b ^ c = 末位 (a + b) ^ c = 末位 (a + b + c)
- a ^ (b ^ c) = a ^ 末位 (b + c) = 末位 (a + b + c)

  因此 a ^ b ^ c = a ^ (b ^ c)

异或运算 ^ 具有可交换性:

a ^ b = b ^ a

位运算的性质

- 按位与、或、异或都是可交换的

位运算的性质

- 按位与、或、异或都是可交换的
  - a & b = b & a

位运算的性质

- 按位与、或、异或都是可交换的
  - a & b = b & a
- 按位与、或、异或都是可结合的

位运算的性质

- 按位与、或、异或都是可交换的
  - a & b = b & a
- 按位与、或、异或都是可结合的
  - a & b & c = a & (b & c)

位运算的性质

- 按位与、或、异或都是可交换的
  - a & b = b & a
- 按位与、或、异或都是可结合的
  - a & b & c = a & (b & c)
- ~0 = 0xFFFFFFFF

位运算的性质

- 按位与、或、异或都是可交换的
  - a & b = b & a
- 按位与、或、异或都是可结合的
  - a & b & c = a & (b & c)
- ~ 0 = 0xFFFFFFFF
- a & 0 = 0;

位运算的性质

- 按位与、或、异或都是可交换的
  - a & b = b & a
- 按位与、或、异或都是可结合的
  - a & b & c = a & (b & c)
- ~ 0 = 0xFFFFFFFF
- a & 0 = 0;　　a & ~ 0 = a;

位运算的性质

- 按位与、或、异或都是可交换的
  - a & b = b & a
- 按位与、或、异或都是可结合的
  - a & b & c = a & (b & c)
- ~ 0 = 0xFFFFFFFF
- a & 0 = 0; a & ~ 0 = a;
- a | 0 = a;

位运算的性质

- 按位与、或、异或都是可交换的
  - a & b = b & a
- 按位与、或、异或都是可结合的
  - a & b & c = a & (b & c)
- ~ 0 = 0xFFFFFFFF
- a & 0 = 0;　　a & ~ 0 = a;
- a | 0 = a;　　a | ~ 0 = ~ 0;

位运算的性质

- 按位与、或、异或都是可交换的
  - a & b = b & a
- 按位与、或、异或都是可结合的
  - a & b & c = a & (b & c)
- ~ 0 = 0xFFFFFFFF
- a & 0 = 0; a & ~ 0 = a;
- a | 0 = a; a | ~ 0 = ~ 0;
- a ^ 0 = a;

位运算的性质

- 按位与、或、异或都是可交换的
  - a & b = b & a
- 按位与、或、异或都是可结合的
  - a & b & c = a & (b & c)
- ∼0 = 0xFFFFFFFF
- a & 0 = 0;　　a & ∼0 = a;
- a | 0 = a;　　a | ∼0 = ∼0;
- a ^ 0 = a;　　a ^ ∼0 = ∼a;

位运算的性质

- 按位与、或、异或都是可交换的
  - a & b = b & a
- 按位与、或、异或都是可结合的
  - a & b & c = a & (b & c)
- ∼0 = 0xFFFFFFFF
- a & 0 = 0;　　a & ∼0 = a;
- a | 0 = a;　　a | ∼0 = ∼0;
- a ^ 0 = a;　　a ^ ∼0 = ∼a;
- a & a = a;

位运算的性质

- 按位与、或、异或都是可交换的
  - a & b = b & a
- 按位与、或、异或都是可结合的
  - a & b & c = a & (b & c)
- ∼0 = 0xFFFFFFFF
- a & 0 = 0;　　a & ∼0 = a;
- a | 0 = a;　　a | ∼0 = ∼0;
- a ^ 0 = a;　　a ^ ∼0 = ∼a;
- a & a = a;　　a & ∼a = 0;

位运算的性质

- 按位与、或、异或都是可交换的
  - a & b = b & a
- 按位与、或、异或都是可结合的
  - a & b & c = a & (b & c)
- ∼0 = 0xFFFFFFFF
- a & 0 = 0;　　a & ∼0 = a;
- a | 0 = a;　　a | ∼0 = ∼0;
- a ^ 0 = a;　　a ^ ∼0 = ∼a;
- a & a = a;　　a & ∼a = 0;
- a | a = a;

位运算的性质

- 按位与、或、异或都是可交换的
  - a & b = b & a
- 按位与、或、异或都是可结合的
  - a & b & c = a & (b & c)
- ∼0 = 0xFFFFFFFF
- a & 0 = 0;　　a & ∼0 = a;
- a | 0 = a;　　a | ∼0 = ∼0;
- a ^ 0 = a;　　a ^ ∼0 = ∼a;
- a & a = a;　　a & ∼a = 0;
- a | a = a;　　a | ∼a = ∼0;

位运算的性质

- 按位与、或、异或都是可交换的
  - a & b = b & a
- 按位与、或、异或都是可结合的
  - a & b & c = a & (b & c)
- ∼0 = 0xFFFFFFFF
- a & 0 = 0;　　a & ∼0 = a;
- a | 0 = a;　　a | ∼0 = ∼0;
- a ^ 0 = a;　　a ^ ∼0 = ∼a;
- a & a = a;　　a & ∼a = 0;
- a | a = a;　　a | ∼a = ∼0;
- a ^ a = 0;

位运算的性质

- 按位与、或、异或都是可交换的
  - a & b = b & a
- 按位与、或、异或都是可结合的
  - a & b & c = a & (b & c)
- ~0 = 0xFFFFFFFF
- a & 0 = 0;　　a & ~0 = a;
- a | 0 = a;　　a | ~0 = ~0;
- a ^ 0 = a;　　a ^ ~0 = ~a;
- a & a = a;　　a & ~a = 0;
- a | a = a;　　a | ~a = ~0;
- a ^ a = 0;　　a ^ ~a = ~0;

位运算的性质

- 按位与、或、异或都是可交换的
  - a & b = b & a
- 按位与、或、异或都是可结合的
  - a & b & c = a & (b & c)
- ∼0 = 0xFFFFFFFF
- a & 0 = 0;　　a & ∼0 = a;
- a | 0 = a;　　a | ∼0 = ∼0;
- a ^ 0 = a;　　a ^ ∼0 = ∼a;
- a & a = a;　　a & ∼a = 0;
- a | a = a;　　a | ∼a = ∼0;
- a ^ a = 0;　　a ^ ∼a = ∼0;
- a ^ b ^ b = a;

移位运算

考虑两个 8 位整数 (16 位和 32 位整数也一样):

$$\mathtt{a} = 0\ 0\ 0\ 0\ 1\ 0\ 1\ 1$$
$$\mathtt{b} = 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1$$

- 左移位　　$\mathtt{a} << 1 = 0\ 0\ 0\ 1\ 0\ 1\ 1\ 0$

移位运算

考虑两个 8 位整数 (16 位和 32 位整数也一样):

$$\mathtt{a} = 0\ 0\ 0\ 0\ 1\ 0\ 1\ 1$$
$$\mathtt{b} = 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1$$

- 左移位　　$\mathtt{a} << 1 = 0\ 0\ 0\ 1\ 0\ 1\ 1\ 0$
  - 右侧补 0

移位运算

考虑两个 8 位整数 (16 位和 32 位整数也一样):

$$\mathtt{a} = 0\ 0\ 0\ 0\ 1\ 0\ 1\ 1$$
$$\mathtt{b} = 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1$$

- 左移位 $\mathtt{a} << 1 = 0\ 0\ 0\ 1\ 0\ 1\ 1\ 0$
  - 右侧补 0
- 右移位 $\mathtt{a} >> 1 = 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1$
  $\mathtt{b} >> 2 = 0\ 0\ 0\ 1\ 1\ 0\ 1\ 1$

移位运算

考虑两个 8 位整数 (16 位和 32 位整数也一样):

$$\mathtt{a} = 0\ 0\ 0\ 0\ 1\ 0\ 1\ 1$$
$$\mathtt{b} = 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1$$

- 左移位 $\mathtt{a} << 1 = 0\ 0\ 0\ 1\ 0\ 1\ 1\ 0$
  - 右侧补 0
- 右移位 $\mathtt{a} >> 1 = 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1$
  $\mathtt{b} >> 2 = 0\ 0\ 0\ 1\ 1\ 0\ 1\ 1$
  - 对于无符号的数，左侧补 0

移位运算

考虑两个 8 位整数 (16 位和 32 位整数也一样):

$$\mathtt{a} = 0\ 0\ 0\ 0\ 1\ 0\ 1\ 1$$
$$\mathtt{b} = 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1$$

- 左移位 $\mathtt{a} << 1 = 0\ 0\ 0\ 1\ 0\ 1\ 1\ 0$
  - 右侧补 0
- 右移位 $\mathtt{a} >> 1 = 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1$
  $\mathtt{b} >> 2 = 0\ 0\ 0\ 1\ 1\ 0\ 1\ 1$
  - 对于无符号的数，左侧补 0
  - 对于有符号的数

移位运算

考虑两个 8 位整数 (16 位和 32 位整数也一样):

$$a = 0\ 0\ 0\ 0\ 1\ 0\ 1\ 1$$
$$b = 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1$$

- 左移位　　$a << 1 = 0\ 0\ 0\ 1\ 0\ 1\ 1\ 0$
  - 右侧补 0
- 右移位　　$a >> 1 = 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1$
  $b >> 2 = 0\ 0\ 0\ 1\ 1\ 0\ 1\ 1$
  - 对于无符号的数，左侧补 0
  - 对于有符号的数
    - 左侧既可以填充 0，

移位运算

考虑两个 8 位整数 (16 位和 32 位整数也一样):

$$\mathtt{a} = 0\,0\,0\,0\,1\,0\,1\,1$$
$$\mathtt{b} = 0\,1\,1\,0\,1\,1\,0\,1$$

- 左移位　　$\mathtt{a} << 1 = 0\,0\,0\,1\,0\,1\,1\,0$
  - 右侧补 0
- 右移位　　$\mathtt{a} >> 1 = 0\,0\,0\,0\,0\,1\,0\,1$
  $\mathtt{b} >> 2 = 0\,0\,0\,1\,1\,0\,1\,1$
  - 对于无符号的数，左侧补 0
  - 对于有符号的数
    - 左侧既可以填充 0，
    - 也可以复制符号位上的值。

移位运算

考虑两个 8 位整数 (16 位和 32 位整数也一样):

$$a = 0\ 0\ 0\ 0\ 1\ 0\ 1\ 1$$
$$b = 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1$$

- 左移位　　$a << 1 = 0\ 0\ 0\ 1\ 0\ 1\ 1\ 0$
  - 右侧补 0
- 右移位　　$a >> 1 = 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1$
  $b >> 2 = 0\ 0\ 0\ 1\ 1\ 0\ 1\ 1$
  - 对于无符号的数，左侧补 0
  - 对于有符号的数
    - 左侧既可以填充 0，
    - 也可以复制符号位上的值。
    - C 语言没有规定，取决于编译器的实现。(大部分编译器复制符号位上的值)

移位运算

考虑两个 8 位整数 (16 位和 32 位整数也一样):

$$a = 0\ 0\ 0\ 0\ 1\ 0\ 1\ 1$$
$$b = 0\ 1\ 1\ 0\ 1\ 1\ 0\ 1$$

- 左移位　　$a << 1 = 0\ 0\ 0\ 1\ 0\ 1\ 1\ 0$
  - 右侧补 0
- 右移位　　$a >> 1 = 0\ 0\ 0\ 0\ 0\ 1\ 0\ 1$
  $b >> 2 = 0\ 0\ 0\ 1\ 1\ 0\ 1\ 1$
  - 对于无符号的数，左侧补 0
  - 对于有符号的数
    - 左侧既可以填充 0，
    - 也可以复制符号位上的值。
    - C 语言没有规定，取决于编译器的实现。(大部分编译器复制符号位上的值)
    - 应该避免对有符号数进行右移运算

移位运算的性质

对于无符号的整数，移位与 2 的关系

移位运算的性质

对于无符号的整数，移位与 2 的关系

- 左移 1 位 $\Longleftrightarrow \times 2$

移位运算的性质

对于无符号的整数，移位与 2 的关系

- 左移 1 位 $\Longleftrightarrow \times 2$
- 右移 1 位 $\Longleftrightarrow \div 2$ （在不发生溢出的时候）

位运算与赋值运算复合

- `&=` 例如：a &= b $\Longleftrightarrow$ a = a & b
- `|=` 例如：a |= b $\Longleftrightarrow$ a = a | b
- `^=` 例如：a ^= b $\Longleftrightarrow$ a = a ^ b
- `>>=` 例如：a >>= n $\Longleftrightarrow$ a = a >> n
- `<<=` 例如：a <<= n $\Longleftrightarrow$ a = a << n

位运算与赋值运算复合

- &= 例如：a &= b $\Longleftrightarrow$ a = a & b
- |= 例如：a |= b $\Longleftrightarrow$ a = a | b
- ^= 例如：a ^= b $\Longleftrightarrow$ a = a ^ b
- >>= 例如：a >>= n $\Longleftrightarrow$ a = a >> n
- <<= 例如：a <<= n $\Longleftrightarrow$ a = a << n

这些复合赋值运算符和其它的赋值运算符具有相同的优先级和结合性

异或运算应用 – 无临时变量的交换

```
int a = 1, b = 2;
```

标准的交换步骤

```
int temp;

temp = a;
a = b;
b = temp;
```

异或运算应用 – 无临时变量的交换

```
int a = 1, b = 2;
```

标准的交换步骤

```
int temp;

temp = a;
a = b;
b = temp;
```

需要定义和使用一个临时变量 temp

异或运算应用 – 无临时变量的交换

```
int a = 1, b = 2;
```

标准的交换步骤

```
int temp;

temp = a;
a = b;
b = temp;
```

需要定义和使用一个临时变量 temp

使用异或运算进行交换

```
a = a ^ b;
b = b ^ a;
a = a ^ b
```

异或运算应用 – 无临时变量的交换

```
int a = 1, b = 2;
```

标准的交换步骤

```
int temp;

temp = a;
a = b;
b = temp;
```

需要定义和使用一个临时变量 temp

使用异或运算进行交换

```
a = a ^ b;
b = b ^ a;
a = a ^ b
```

或

```
a ^= b; b ^= a; a ^= b;
```

异或运算应用 – 无临时变量的交换

```
int a = 1, b = 2;
```

标准的交换步骤

```
int temp;

temp = a;
a = b;
b = temp;
```

需要定义和使用一个临时变量 temp

使用异或运算进行交换

```
a = a ^ b;
b = b ^ a;
a = a ^ b
```

或

```
a ^= b; b ^= a; a ^= b;
```

或

```
a ^= b ^= a ^= b
```

异或运算应用 – 无临时变量的交换

```
int a = 1, b = 2;
```

标准的交换步骤

```
int temp;

temp = a;
a = b;
b = temp;
```

需要定义和使用一个临时变量 temp

使用异或运算进行交换

```
a = a ^ b;
b = b ^ a;
a = a ^ b
```

```
a ^ b ^ b = a
a ^ b ^ a = b
```

或

```
a ^= b; b ^= a; a ^= b;
```

或

```
a ^= b ^= a ^= b
```

异或运算应用 – 确定重复数

假设有 N+1 个变量，存放了整数 $1, 2, 3, \ldots, N$，其中有且只有一个整数重复了 2 次。如何快速确定哪一个整数重复了？

可行的方法：

- 逐对比较，检查没有相同的。运算量：$N \times (N-1)/2$

异或运算应用 – 确定重复数

假设有 N+1 个变量，存放了整数 $1, 2, 3, \ldots, N$，其中有且只有一个整数重复了 2 次。如何快速确定哪一个整数重复了？

可行的方法：

- 逐对比较，检查没有相同的。运算量：$N \times (N-1)/2$
- 计算所有的变量的和，与 1+2+...+N 比较，得到重复的数。运算量 N+1 次加减法

异或运算应用 – 确定重复数

假设有 N+1 个变量，存放了整数 $1, 2, 3, \ldots, N$，其中有且只有一个整数重复了 2 次。如何快速确定哪一个整数重复了？

可行的方法：

- 逐对比较，检查没有相同的。运算量： $N \times (N-1)/2$
- 计算所有的变量的和，与 1+2+...+N 比较，得到重复的数。运算量 N+1 次加减法
  - 加法求和可能会溢出（当 N 非常大的时候）

异或运算应用 – 确定重复数

假设有 N+1 个变量，存放了整数 $1, 2, 3, \ldots, N$，其中有且只有一个整数重复了 2 次。如何快速确定哪一个整数重复了？

可行的方法：

- 逐对比较，检查没有相同的。运算量：$N \times (N-1)/2$
- 计算所有的变量的和，与 1+2+...+N 比较，得到重复的数。运算量 N+1 次加减法
  - 加法求和可能会溢出（当 N 非常大的时候）
- 把加法换作：异或

异或运算应用 – 确定重复数

假设有 N+1 个变量，存放了整数 $1, 2, 3, \ldots, N$，其中有且只有一个整数重复了 2 次。如何快速确定哪一个整数重复了？

可行的方法：

- 逐对比较，检查没有相同的。运算量：$N \times (N-1)/2$
- 计算所有的变量的和，与 1+2+...+N 比较，得到重复的数。运算量 N+1 次加减法
  - 加法求和可能会溢出（当 N 非常大的时候）
- 把加法换作：异或

异或运算应用 – 确定重复数

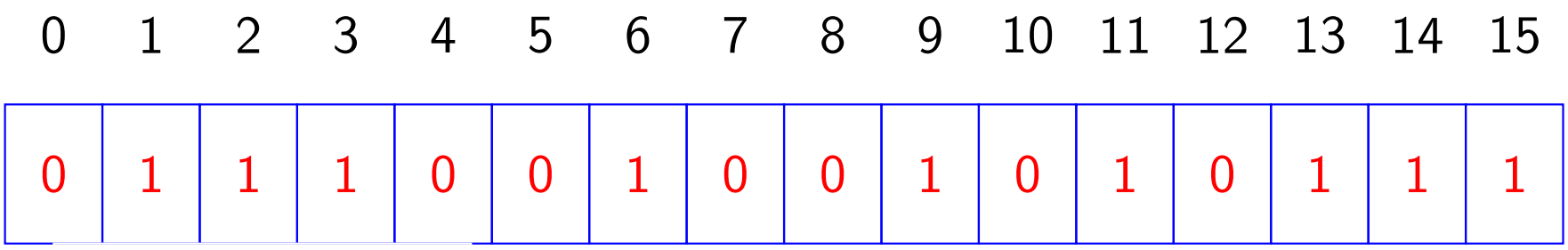
假设有 N+1 个变量，存放了整数 $1, 2, 3, \ldots, N$，其中有且只有一个整数重复了 2 次。如何快速确定哪一个整数重复了？

可行的方法：

- 逐对比较，检查没有相同的。运算量：$N \times (N-1)/2$
- 计算所有的变量的和，与 1+2+...+N 比较，得到重复的数。运算量 N+1 次加减法
  - 加法求和可能会溢出（当 N 非常大的时候）
- 把加法换作：异或

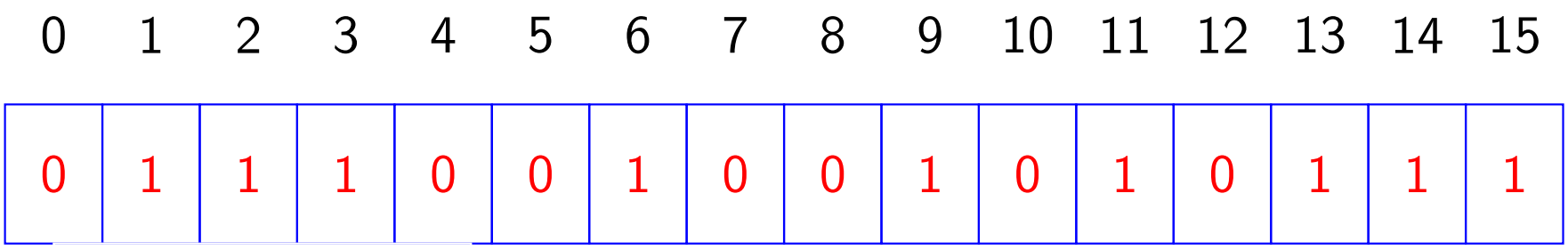
```
a0^a1^a2^...^aN ^ 1^2^...^N
```

位运算应用 – 输出整数的 bit 值



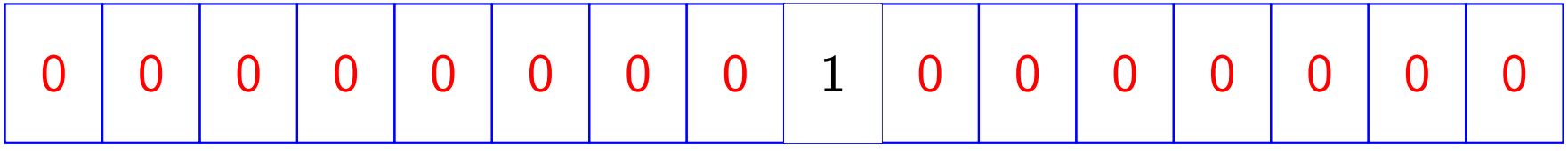
如何获取第 k 位的值？

位运算应用 – 输出整数的 bit 值



如何获取第 k 位的值？

产生一个掩码：$1 << k$，（只有第 k 位为 1）



位运算应用 – 输出整数的 bit 值

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	1	1	1	0	0	1	0	0	1	0	1	0	1	1	1

如何获取第 k 位的值？

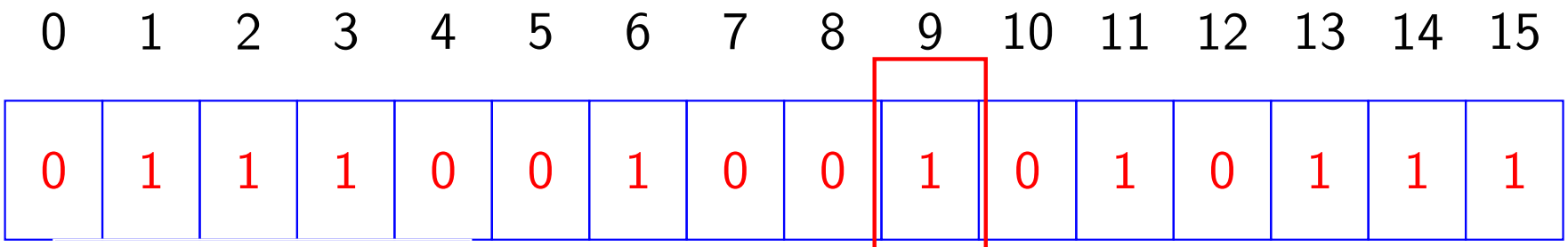
产生一个掩码：$1 << k$，（只有第 k 位为 1）

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

执行按位与运算：$n \,\&\, (1 << k)$

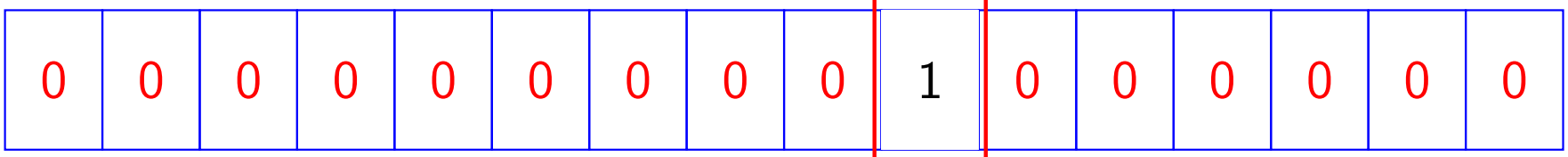
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

位运算应用 – 输出整数的 bit 值

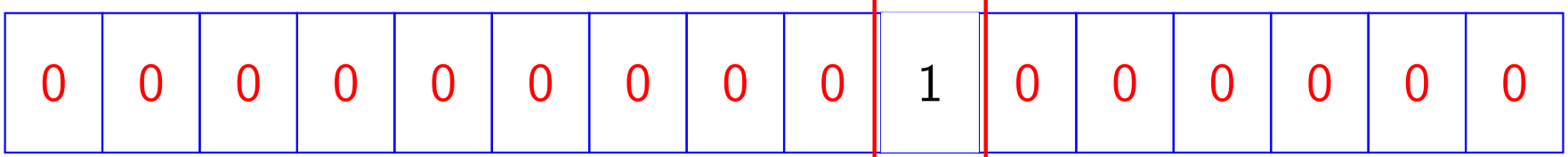


如何获取第 k 位的值？

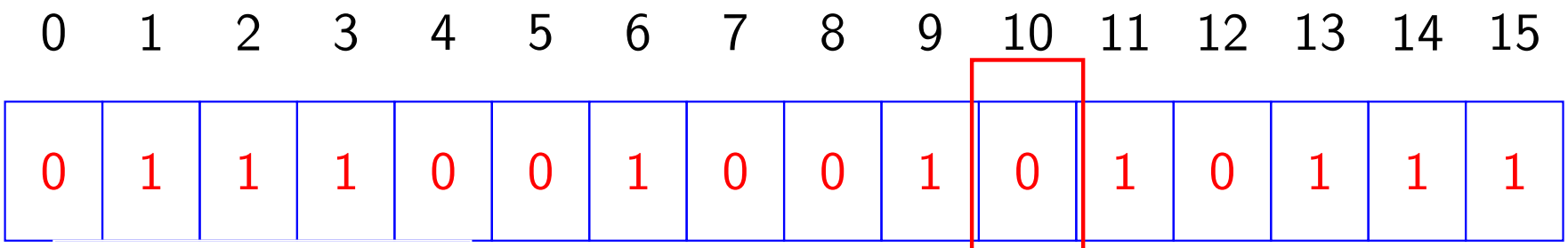
产生一个掩码：$1 << k$，（只有第 k 位为 1）



执行按位与运算：$n \,\&\, (1 << k)$

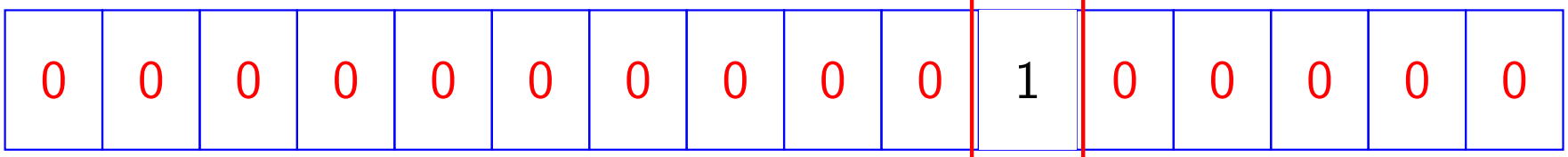


位运算应用 – 输出整数的 bit 值

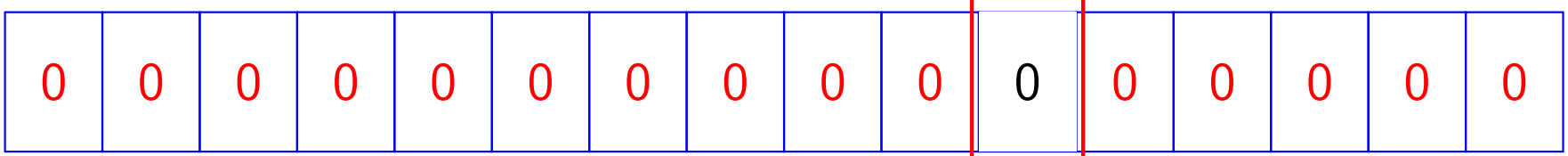


如何获取第 k 位的值？

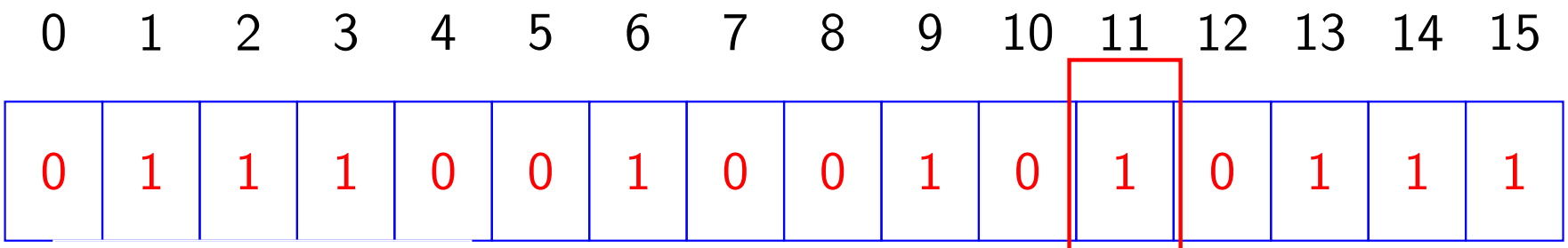
产生一个掩码：$1 << k$，（只有第 k 位为 1）



执行按位与运算：$n \,\&\, (1 << k)$

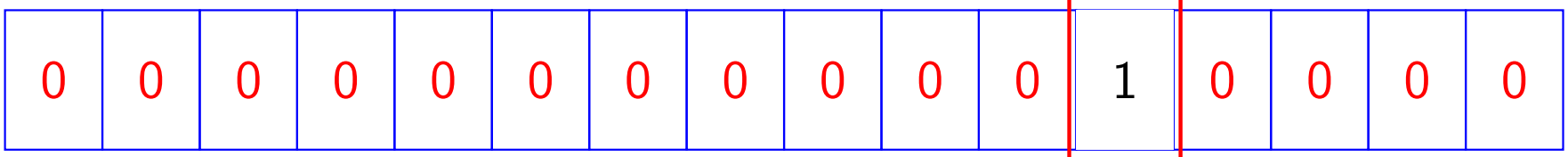


位运算应用 – 输出整数的 bit 值

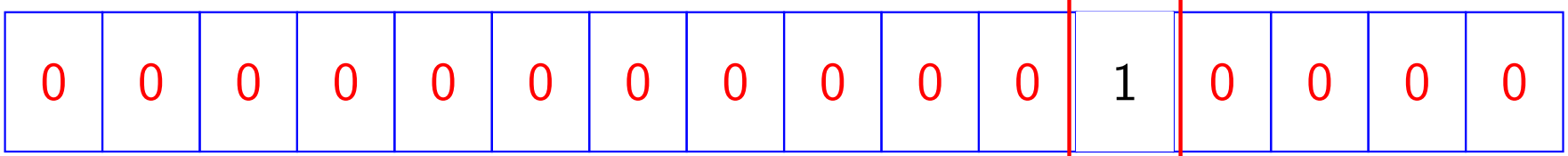


如何获取第 k 位的值？

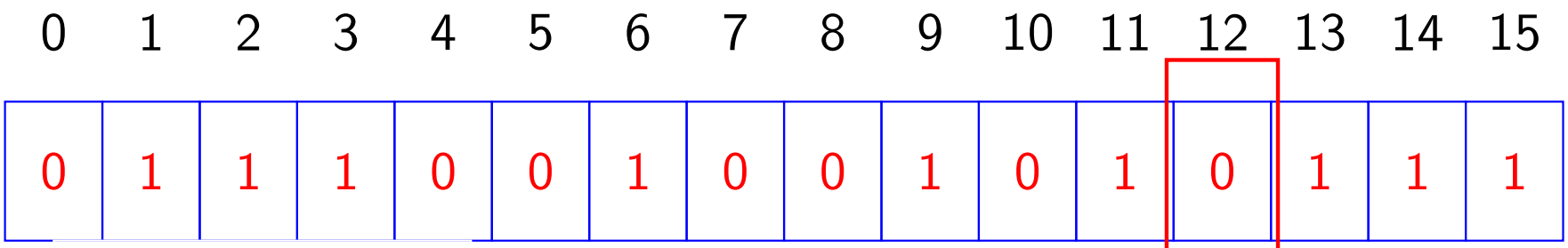
产生一个掩码：$1 << k$，（只有第 k 位为 1）



执行按位与运算：$n \,\&\, (1 << k)$

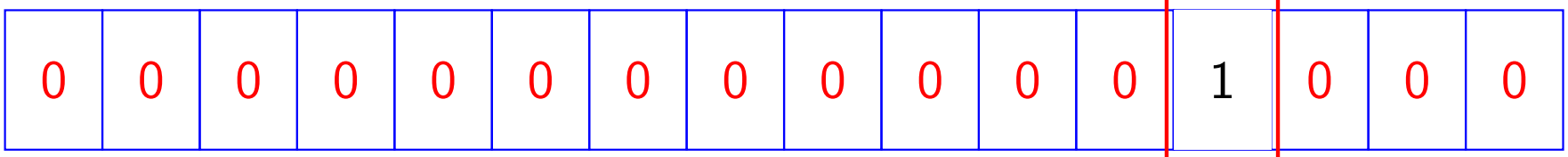


位运算应用 – 输出整数的 bit 值

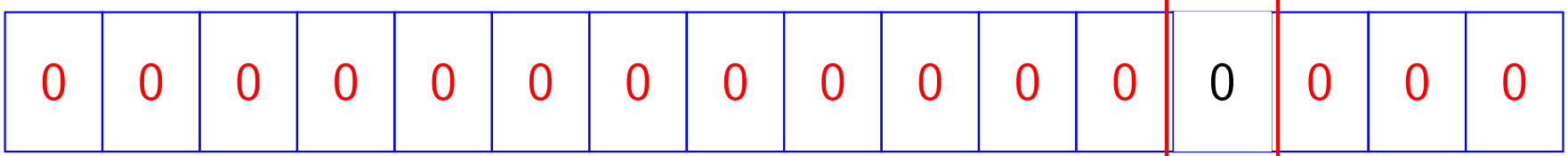


如何获取第 k 位的值？

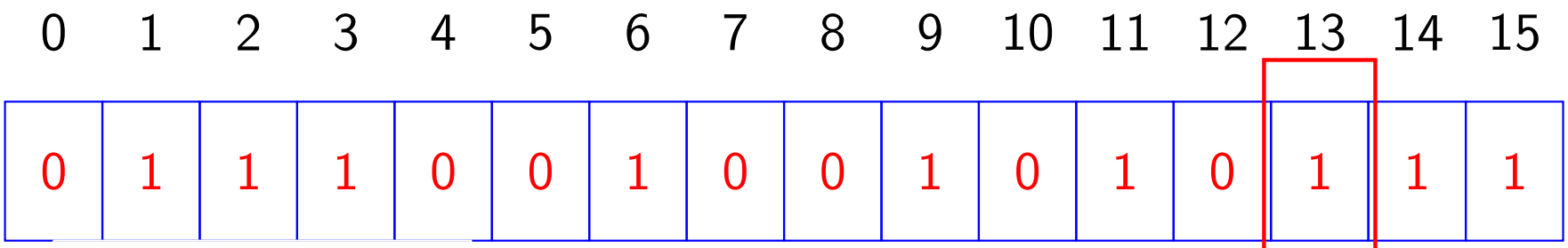
产生一个掩码：$1 << k$，（只有第 k 位为 1）



执行按位与运算：$n \,\&\, (1 << k)$

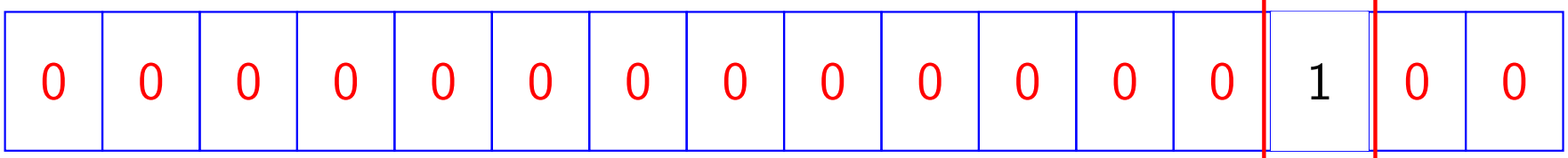


位运算应用 – 输出整数的 bit 值

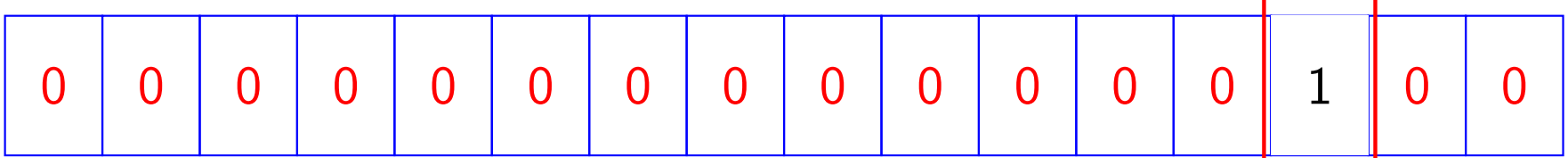


如何获取第 k 位的值？

产生一个掩码：$1 << k$，（只有第 k 位为 1）



执行按位与运算：$n \,\&\, (1 << k)$



位运算应用 – 输出整数的 bit 值

```
void print_bits ( int n, int high, int low )
{
    int k;
    for( k = high; k>=low; k-- ) {
        int mask = 1<<k; //生成掩码
        if( mask & n )    //掩码对应的bit值
            putchar('1');
        else
            putchar('0');
    }
}
```

- 应用移位运算，为第 k 位比特位制造一个掩码:
  $mask = 1 << k$
- 将掩码 mask 和整数进行逻辑与，获得 bit 值: mask & n
  mask & n 非零，代表该比特为 1

位运算应用 – 输出整数的 bit 值

```
int main()
{
    int n; float x;
    printf("输入一个整数和实数:");
    scanf("%d %f", &n, &x);
    printf("整数%9d = ", n); print_bits(n, 31, 0);
                putchar('\n');
    printf("实数%9f = ", x); print_bits(*(int*)&x,
        31, 0); putchar('\n');
    return 0;
}
```

位运算应用 – 输出整数的 bit 值

```
int main()
{
    int n; float x;
    printf("输入一个整数和实数:");
    scanf("%d %f", &n, &x);
    printf("整数%9d = ", n); print_bits(n, 31, 0);
                  putchar('\n');
    printf("实数%9f = ", x); print_bits(*(int*)&x,
        31, 0); putchar('\n');
    return 0;
}
```

```
输入一个整数和实数:234 13.567
整数       234 = 00000000000000000000000011101010
实数 13.567000 = 01000001010110010001001101111
```

位运算应用 – 输出整数的 bit 值

```
int main()
{
    int n; float x;
    printf("输入一个整数和实数:");
    scanf("%d %f", &n, &x);
    printf("整数%9d = ", n); print_bits(n, 31, 0);
                putchar('\n');
    printf("实数%9f = ", x); print_bits(*(int*)&x,
        31, 0); putchar('\n');
    return 0;
}
```

```
输入一个整数和实数:234 13.567
整数       234 = 00000000000000000000000011101010
实数 13.567000 = 01000001010110010001001001101111
```

这里利用了指针和类型转换: &x 获取了float x 的指针，然后将其转换为整数指针 (int*)，在通过整数指针，将它的 bits 转化为整数，保存到变量 n 中。只有整数才可以进行位运算

内容提要

位运算应用 – 查看浮点数 float 的符号位，阶码，尾数

符号	阶码	尾数
1 bit	8 bit	23 bit

位运算应用 – 查看浮点数 float 的符号位，阶码，尾数

符号	阶码	尾数
1 bit	8 bit	23 bit

获取符号

```
int get_float_sign ( float x )
{
    int n = *(int*)&x; //复制到int变量中
    int mask = 1<<31;  //生成符号位的掩码
    return mask & n ? -1 : 1;
}
```

位运算应用 – 查看浮点数 float 的符号位，阶码，尾数

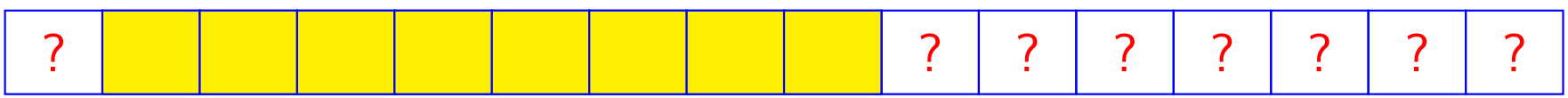
获取阶码

```
int get_float_exp ( float x )
{
    int n = *(int*)&x; //复制到int变量中
    n = (n >> 23) & 0xFF;
    n = n - 127;
    return n;
}
```

位运算应用 – 查看浮点数 float 的符号位，阶码，尾数

获取阶码

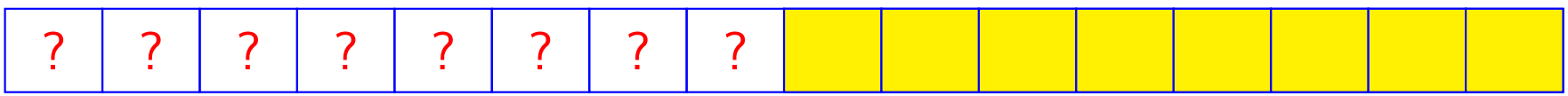
```
int get_float_exp ( float x )
{
    int n = *(int*)&x; //复制到int变量中
    n = (n >> 23) & 0xFF;
    n = n - 127;
    return n;
}
```



位运算应用 – 查看浮点数 float 的符号位，阶码，尾数

获取阶码

```
int get_float_exp ( float x )
{
    int n = *(int*)&x; //复制到int变量中
    n = (n >> 23) & 0xFF;
    n = n - 127;
    return n;
}
```

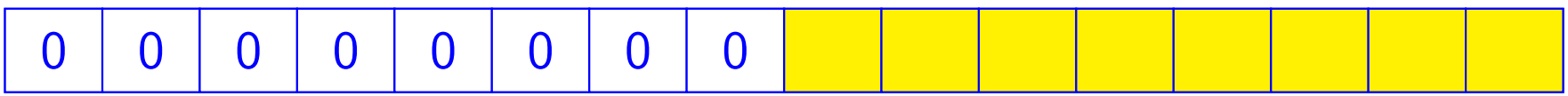


- 向右移位（23 位）

位运算应用 – 查看浮点数 float 的符号位，阶码，尾数

获取阶码

```
int get_float_exp ( float x )
{
    int n = *(int*)&x; //复制到int变量中
    n = (n >> 23) & 0xFF;
    n = n - 127;
    return n;
}
```



- 向右移位（23 位）
- 保留低 8 位，其余清零

位运算应用 – 查看浮点数 float 的符号位，阶码，尾数

获取尾数

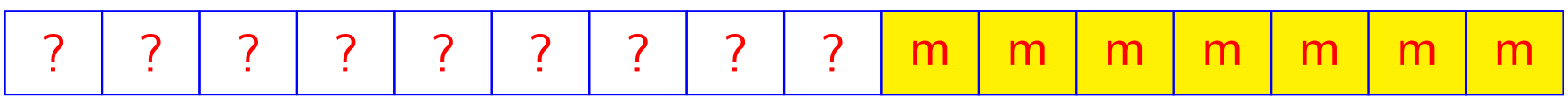
```
float get_float_mantissa ( float x )
{
    int n = *(int*)&x; //复制到int变量中
    int e = n & (0XFF<<23); // 阶码段
    n &= ~(0X1FF<<23); // 符号位和阶码段清零
    if( e ) {//不全为0
        n |= 0x7F<<23; // 阶码段置为0X7F (0x7F
            =01111111, 代表指数为0)
    }
    return *(float*)&n;
}
```

?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

位运算应用 – 查看浮点数 float 的符号位，阶码，尾数

获取尾数

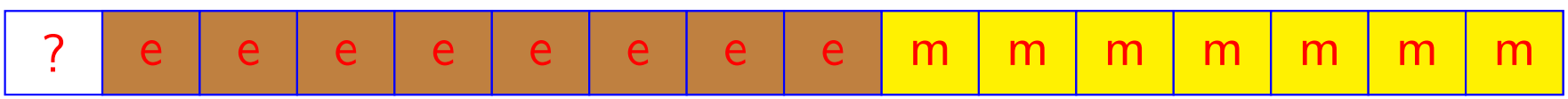
```
float get_float_mantissa ( float x )
{
    int n = *(int*)&x; //复制到int变量中
    int e = n & (0XFF<<23); // 阶码段
    n &= ~(0X1FF<<23); // 符号位和阶码段清零
    if( e ) {//不全为0
        n |= 0x7F<<23; // 阶码段置为0X7F (0x7F
            =01111111, 代表指数为0)
    }
    return *(float*)&n;
}
```



位运算应用 – 查看浮点数 float 的符号位，阶码，尾数

获取尾数

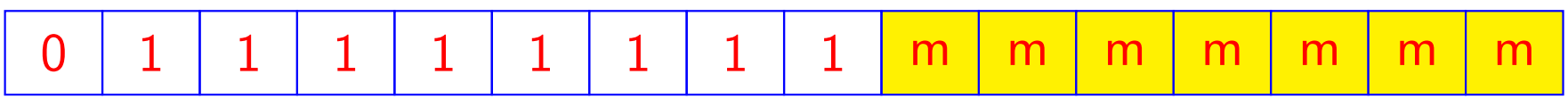
```
float get_float_mantissa ( float x )
{
    int n = *(int*)&x; //复制到int变量中
    int e = n & (0XFF<<23); // 阶码段
    n &= ~(0X1FF<<23); // 符号位和阶码段清零
    if( e ) {//不全为0
        n |= 0x7F<<23; // 阶码段置为0X7F (0x7F
            =01111111, 代表指数为0)
    }
    return *(float*)&n;
}
```



位运算应用 – 查看浮点数 float 的符号位，阶码，尾数

获取尾数

```
float get_float_mantissa ( float x )
{
    int n = *(int*)&x; //复制到int变量中
    int e = n & (0XFF<<23); // 阶码段
    n &= ~(0X1FF<<23); // 符号位和阶码段清零
    if( e ) {//不全为0
        n |= 0x7F<<23; // 阶码段置为0X7F (0x7F
            =01111111, 代表指数为0)
    }
    return *(float*)&n;
}
```



位运算应用 – 查看浮点数 float 的符号位，阶码，尾数

查看浮点数

```
void print_float(float x)
{
    print_bits( *(int*)&x, 31, 31 );
    printf("<%c> ", get_float_sign(x)>0?'+':'-');
    print_bits(*(int*)&x, 30, 23 ); print_space(1);
    printf("<%-4d> ", get_float_exp(x));
    print_bits(*(int*)&x, 22, 0 );
    printf(" <%f>\n", get_float_mantissa(x));
}
```

```
输入一个浮点数:12.3456
符号 阶码段    <指数> 尾数段                          <尾数>
0<+> 10000010 <3    > 10001011000011110010100 <1.543200>

输入一个浮点数:1.5432
符号 阶码段    <指数> 尾数段                          <尾数>
0<+> 01111111 <0    > 10001011000011110010100 <1.543200>
```

内容提要

长度运算符 sizeof

一个计算常量、变量、以及数据类型的长度的运算符。长度以字节为单位

- 单目运算符

长度运算符 sizeof

一个计算常量、变量、以及数据类型的长度的运算符。长度以字节为单位

- 单目运算符
- 用 sizeof 表示

长度运算符 sizeof

一个计算常量、变量、以及数据类型的长度的运算符。长度以字节为单位

- 单目运算符
- 用 sizeof 表示
- 运算对象可以是

长度运算符 sizeof

一个计算常量、变量、以及数据类型的长度的运算符。长度以字节为单位

- 单目运算符
- 用 sizeof 表示
- 运算对象可以是
  - 常量，例如 sizeof(5)

长度运算符 sizeof

一个计算常量、变量、以及数据类型的长度的运算符。长度以字节为单位

- 单目运算符
- 用 sizeof 表示
- 运算对象可以是
  - 常量，例如 sizeof(5)
  - 变量，例如 sizeof(x)

长度运算符 sizeof

一个计算常量、变量、以及数据类型的长度的运算符。长度以字节为单位

- 单目运算符
- 用 sizeof 表示
- 运算对象可以是
  - 常量，例如 sizeof(5)
  - 变量，例如 sizeof(x)
  - 数据类型名，例如 sizeof(float)

长度运算符 sizeof

一个计算常量、变量、以及数据类型的长度的运算符。长度以字节为单位

- 单目运算符
- 用 sizeof 表示
- 运算对象可以是
  - 常量，例如 sizeof(5)
  - 变量，例如 sizeof(x)
  - 数据类型名，例如 sizeof(float)

假设定义了：int a; 那么：

长度运算符 sizeof

一个计算常量、变量、以及数据类型的长度的运算符。长度以字节为单位

- 单目运算符
- 用 sizeof 表示
- 运算对象可以是
  - 常量，例如 sizeof(5)
  - 变量，例如 sizeof(x)
  - 数据类型名，例如 sizeof(float)

假设定义了：int a; 那么：

- 表达式 sizeof(a) 的值为变量 a 的长度，等于 4

长度运算符 sizeof

一个计算常量、变量、以及数据类型的长度的运算符。长度以字节为单位

- 单目运算符
- 用 sizeof 表示
- 运算对象可以是
  - 常量，例如 sizeof(5)
  - 变量，例如 sizeof(x)
  - 数据类型名，例如 sizeof(float)

假设定义了：int a; 那么：

- 表达式 sizeof(a) 的值为变量 a 的长度，等于 4
- 表达式 sizeof(int) 表示整型 int 的长度，等于 4

长度运算符 sizeof

一个计算常量、变量、以及数据类型的长度的运算符。长度以字节为单位

- 单目运算符
- 用 sizeof 表示
- 运算对象可以是
  - 常量，例如 sizeof(5)
  - 变量，例如 sizeof(x)
  - 数据类型名，例如 sizeof(float)

假设定义了：int a; 那么：

- 表达式 sizeof(a) 的值为变量 a 的长度，等于 4
- 表达式 sizeof(int) 表示整型 int 的长度，等于 4
- 表达式 sizeof(double) 表示 doulbe 类型数据的长度，等于 8

运算符优先级

优先级	名称	运算符	功能和特点
1. 初等类	括号	()	可改变优先级顺序
	下标	[]	数组下标
	指针	$->$	结构指针引用成员
	指针	.	结构体引用成员

运算符优先级

优先级	名称	运算符	功能和特点
1. 初等类	括号	()	可改变优先级顺序
	下标	[]	数组下标
	指针	$->$	结构指针引用成员
	指针	.	结构体引用成员
2. 单目类 右 $\rightarrow$ 左结合	逻辑非	!	逻辑取反
	按位反	$\sim$	按位取反
	正号	$+$	指定符号为正
	负号	$-$	取相反值
	类型转换	(类型名)	强制转换类型
	地址	&	去变量地址
	自增	$++$	自增
	自减	$--$	自减
	长度	sizeof	取长度/字节数

运算符优先级

优先级	名称	运算符	功能和特点
3. 算术乘除	乘号	*	算数运算
	除号	/	
	余数	%	
4. 算术加减	加号	+	
	减号	−	
5. 移位类	左移	<<	向左移位
	右移	>>	向右移位
6. 关系比较	大于	>	结果为逻辑值
	小于	<	
	不小于	>=	
	不大于	<=	
7. 相等比较	等于	==	
	不等	!=	

运算符优先级

优先级	名称	运算符	功能和特点
8	按位与	&	位逻辑运算
9	按位异或	^	
10	按位或	|	
11	逻辑与	&&	逻辑运算
12	逻辑或	||	
13	条件运算	? :	右 → 左结合
14	赋值	= += -= *= /= %= &= ^= |= >>= <<=	右 → 左结合
15	逗号运算符	,	构造逗号表达式

字符处理程序解析

读入一行字符，统计以空格分隔的单词个数。

```
int cnt = 0, word = 0;
char ch;
printf("输入一行字符: ");
while( (ch = getchar()) != '\n' ) {
    if( ch==' ' ) //单词分隔符
        word = 0; //清除单词标志
    else if( word==0 ) {//开始新的单词
        word = 1; //设置单词标志
        cnt++;    //增加单词个数
    }
}
printf("单词个数=%d\n", cnt);
```

字符处理程序解析

读入一行字符，统计以空格分隔的单词个数。

```
int cnt = 0, word = 0;
char ch;
printf("输入一行字符: ");
while( (ch = getchar()) != '\n' ) {
    if( ch==' ' ) //单词分隔符
        word = 0; //清除单词标志
    else if( word==0 ) {//开始新的单词
        word = 1; //设置单词标志
        cnt++;    //增加单词个数
    }
}
printf("单词个数=%d\n", cnt);
```

字符处理程序解析

读入一行字符，统计以空格分隔的单词个数。

```
int cnt = 0, word = 0;
char ch;
printf("输入一行字符: ");
while( (ch = getchar()) != '\n' ) {
    if( ch==' ' ) //单词分隔符
        word = 0; //清除单词标志
    else if( word==0 ) {//开始新的单词
        word = 1; //设置单词标志
        cnt++;    //增加单词个数
    }
}
printf("单词个数=%d\n", cnt);
```

字符处理程序解析

```
while( (ch = getchar()) != '\n' )
```

能否替换为下面的代码？

```
while( ch = getchar() != '\n' )
```

为什么？

总结

- 基本的数据类型

总结

- 基本的数据类型
- 补码表示

总结

- 基本的数据类型
- 补码表示
- 扩展的整数类型

总结

- 基本的数据类型
- 补码表示
- 扩展的整数类型
- 浮点数表示

总结

- 基本的数据类型
- 补码表示
- 扩展的整数类型
- 浮点数表示
- 数据的输入输出格式控制符

总结

- 基本的数据类型
- 补码表示
- 扩展的整数类型
- 浮点数表示
- 数据的输入输出格式控制符
- 表达式

总结

- 基本的数据类型
- 补码表示
- 扩展的整数类型
- 浮点数表示
- 数据的输入输出格式控制符
- 表达式
- 位运算、位逻辑、移位运算

总结

- 基本的数据类型
- 补码表示
- 扩展的整数类型
- 浮点数表示
- 数据的输入输出格式控制符
- 表达式
- 位运算、位逻辑、移位运算
- 应用位运算，解析浮点数

总结

- 基本的数据类型
- 补码表示
- 扩展的整数类型
- 浮点数表示
- 数据的输入输出格式控制符
- 表达式
- 位运算、位逻辑、移位运算
- 应用位运算，解析浮点数
- 运算符和优先级

总结

- 基本的数据类型
- 补码表示
- 扩展的整数类型
- 浮点数表示
- 数据的输入输出格式控制符
- 表达式
- 位运算、位逻辑、移位运算
- 应用位运算，解析浮点数
- 运算符和优先级
- 逗号表达式、条件表达式

今天到此为止