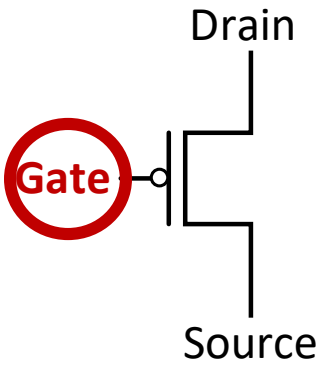
高级语言C++程序设计
Lecture 2 数据表示

李雨森
南开大学 计算机学院
2021

计算机是二进制的世界

计算机是由逻辑电路组成，逻辑电路通常只有两个状态，开关的接通与断开，分别可以用“1”和“0”表示



2.1 数制

数制

- 用一组固定的数码和一套统一的规则来表示数值的方法
- 十进制
  - 数码: 0, 1, 2, …, 9
  - 运算规则：逢十进一

十进制数**1101.11**按权展开为:

$1101.11 = 1\times10^3 + 1\times10^2 + 0\times10^1 + 1\times10^0 + 1\times10^{-1} + 1\times10^{-2}$

数制

- 二进制
  - 数码: 0, 1
  - 运算规则：逢二进一

$$\begin{array}{r} 1001 \\ +\ \ 101 \\ \hline =\ 1110 \end{array}$$

二进制数1101.11按权展开为:

$1101.11 = 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0 + 1\times2^{-1} + 1\times2^{-2}$

数制

- 二进制
  - 比特 (bit)
    - 计算机中最小的数据单位，一个二进制位可以表示0和1两种状态
  - 字节 (byte)
    - 8 bits为一个字节(byte)，记作B
    - 存储容量经常使用KB（2^{10}B）、MB（2^{20}B）、GB（2^{30}B）和TB（2^{40}B）
  - 字 (word)
    - 计算机一次能够处理的二进制位的长度
    - 64位计算机表示该计算机的字长是64个bits

数制

常用数制

数 制	十 进 制	二 进 制	八 进 制	十 六 进 制
数码	0, 1, 2, 3,…, 9	0, 1	0, 1, 2, 3,…,7	0, 1, 2, 3,…, 9, A ,B, C, D E, F
基数	10	2	8	16
位权	10^i	2^i	8^i	16^i
后缀	D (decimal)	B (binary)	Q (octal)	H (hexadecimal)

二进制数 10110：$(10110)_2$ 或 10110B

十六进制数 2D5F：$(2D5F)_{16}$ 或 2D5FH

十进制数 1234：$(1234)_{10}$ 或 1234 或 1234D

八进制数 215：$(215)_8$ 或 215Q

数制转换

非十进制转换成十进制数：按权展开求和

【例】将二进制数 $(1101.1)_2$转换成十进制数

$(1101.1)_2=1\times2^3+1\times2^2+0\times2^1+1\times2^0+1\times2^{-1}$

$=8+4+0+1+0.5$

$=13.5$

【例】将八进制数$(346)_8$转换成十进制数

解: $(346)_8=3\times8^2+4\times8^1+6\times8^0$

$=192+32+6$

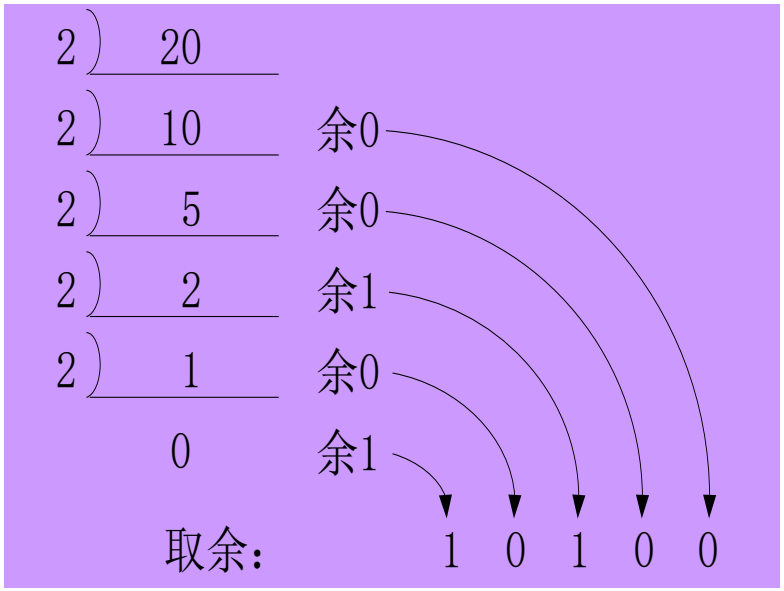
$=230$

数制转换

十进制转换成非十进制数：整数部分“除基取余”，小数部分“乘基取整”

【例】将十进制数20转换成二进制

解: 10100B



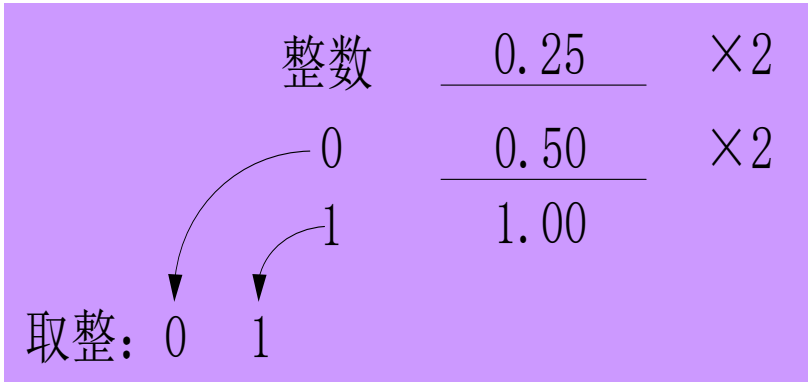
数制转换

十进制转换成非十进制数：整数部分“除基取余”，小数部分“乘基取整”

乘基取整：将小数部分连续乘以基数2，直到小数部分等于0为止。然后，将每次相乘所得到的数的整数部分按正序从左到右排列

【例】将十进制数0.25

转换成二进制数



如果小数部分一直不等于0呢？根据精度适当取小数位

数制转换

二进制与八进制相互转换

八进制转二进制:

将每一位八进制数直接写成相应的3位二进制数

【例】将八进制数$(425.67)_8$转换成二进制数

解: $(425.67)_8=(100\ 010\ 101.110\ 111)_2$

数制转换

二进制与八进制相互转换

二制数转八进制:

以小数点为界，向左或向右将每3位二进制数分成一组，如果不足3位，则用0补足。然后，将每一组二进制数直接写成相应的1位八进制数

【例】将二进制数$(10101111.01101)_2$转换成八进制数

解: $(10101111.01101)_2=(010\ 101\ 111.011\ 010)_2$

$=(257.32)_8$

数制转换

二进制与十六进制相互转换

十六进制转二进制:

将每一位十六进制数直接写成相应的4位二进制数

【例】将十六进制数$(2C8)_{16}$转换成二进制数。

解: $(2C8)\ 16 = (0010\ 1100\ 1000)_2$

$= (1011001000)_2$

数制转换

二进制与十六进制相互转换

二制数转十六进制:

以小数点为界，向左或向右将每4位二进制数分成一组，如果不足4位，则用0补足。然后，将每一组二进制数直接写成相应的1位十六进制数

【例】将二进制数$(1011001.11)_2$转换成十六进制数。

解: $(1011001.11)_2=(0101\ 1001.1100)_2$

$=(59.C)_{16}$

2.2 数据在计算机中的表示

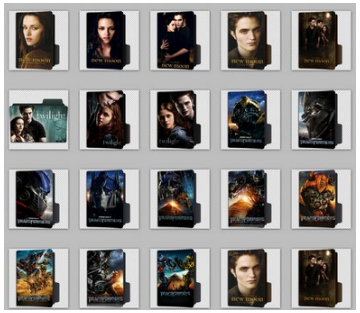
什么是数据?

数据

数值型数据：1、2、3

字符串：hello world

图像、音乐、电影...



数据在计算机中的表示：

二进制

0110 1000 1010 0000...

整数的表示

unsigned int 类型

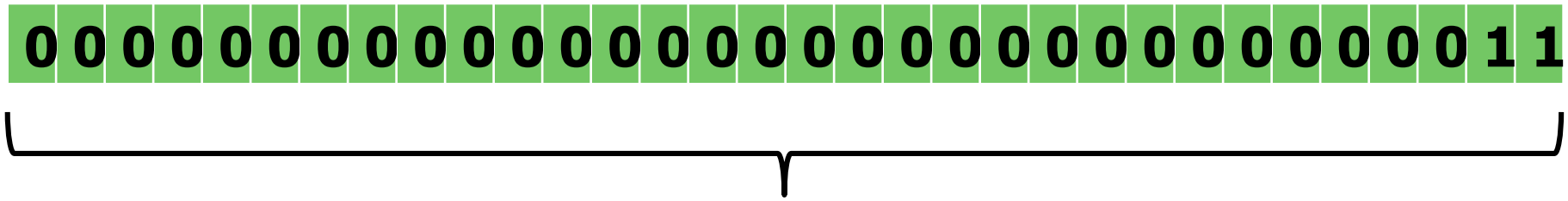
```
#include<iostream>
using namespace std;
int main(){
    unsigned int x = 3;
    cout<<x;
    return 0;
}
```

- 只能表示非负数（称为无符号整数）
- 每个unsigned int类型整数用32 bits二进制表示

整数的表示

unsigned int 类型

```
unsigned int x = 3;
```



直接用该数的二进制表示，不够32位左边补0

整数的表示

unsigned int 类型

能表示的最小数为: 0

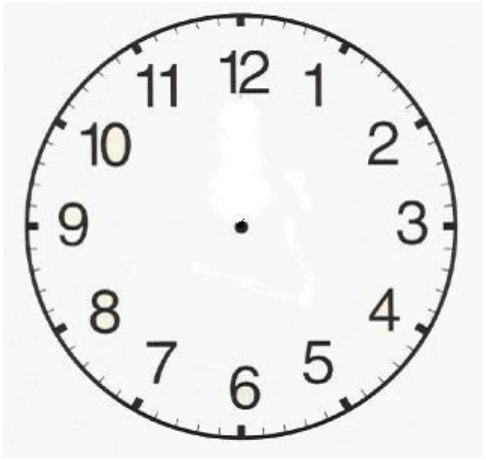
能表示的最大数为: 2^{32} -1，即4294967295

整数的表示

C++提供的无符号整数类型

类型	字节数	表示范围
unsigned int	4	$0 \sim 2^{32} - 1$
unsigned short	2	$0 \sim 2^{16} - 1$
unsigned char	1	$0 \sim 2^{8} - 1$
unsigned long int	4	$0 \sim 2^{32} - 1$

模的概念

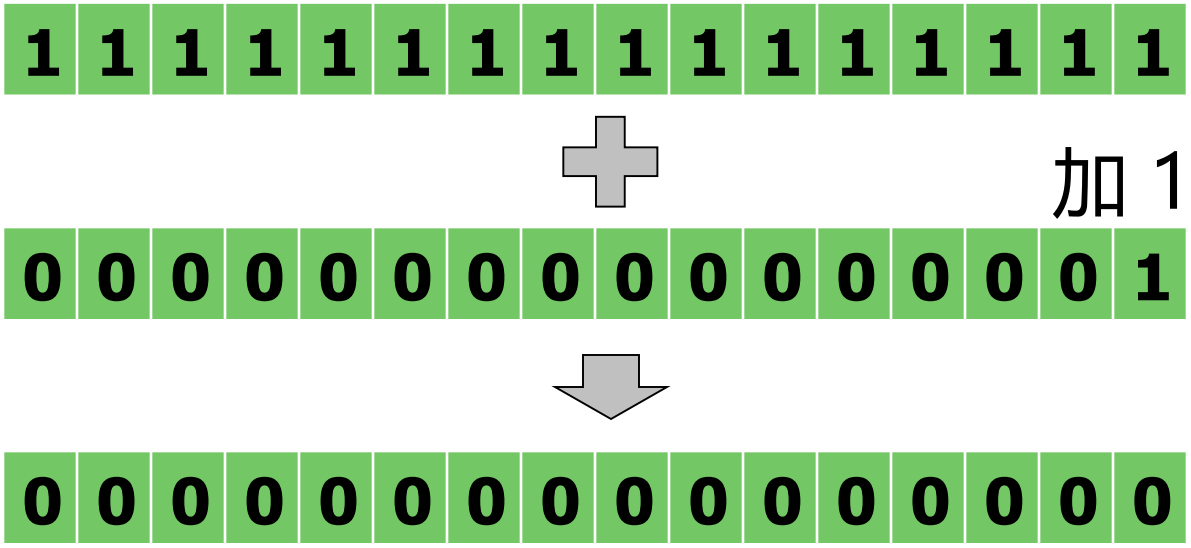


3点钟加上12小时，
结果是几点钟?

在一个计量系统中，计量范围最大值 + 1称为模，
在有模的计量系统中，任意值加上模等于其本身

二进制的模

N位二进制能表示的最大值是 2^N-1



结果变为N+1位，值为2^N，其中低N位重新变为0

在N位二进制系统中，2^N 称为该系统的模，任意数加上2^N都等于其本身（只保留低N位）

二进制的模

```
#include<iostream>
using namespace std;

int main(void){
    unsigned int x = 4294967295;//2^32-1
    cout<<x+1;
    return 0;
}
```

根据模的概念，该程序的输出结果应该是?

整数的表示

int 类型

```
#include<iostream>
using namespace std;
int main(){
    int x = 3;
    cout<<x;
    return 0;
}
```

- 即能表示正数，又能表示负数（称为有符号整数）
- 每个int类型整数占32 bits
- 最高位称为符号位：

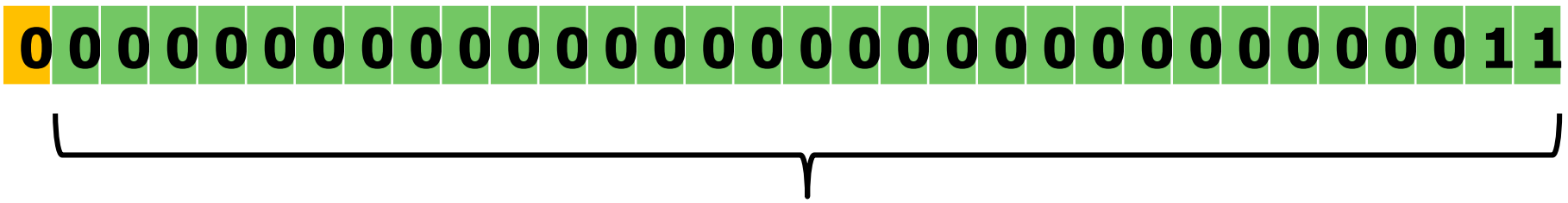
  0表示正数，1表示负数

整数的表示

int 类型的正数 （原码表示法）

```
int x = 3;
```

符号位为0



其余31位为该数的二进制表示

整数的表示

int 类型的正数 （原码表示法）

能表示的最大正数为: $2^{31}-1$，即2147483647

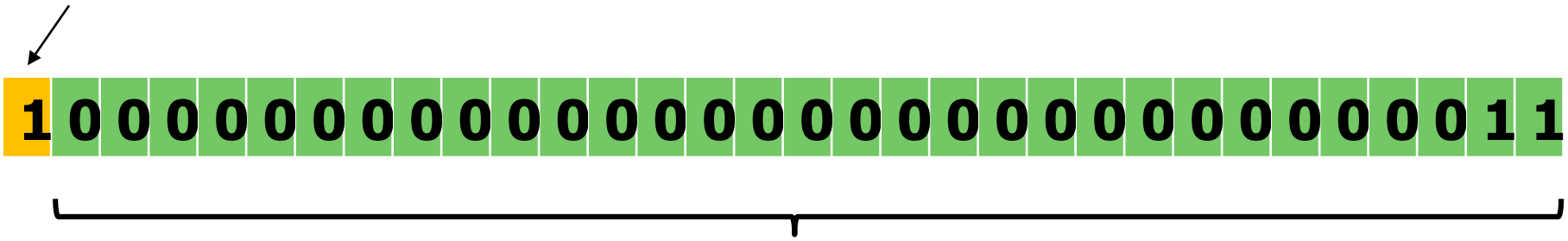
`int x = 2147483648;` 会发生什么？

整数的表示

int 类型的负数

```
int x = -3;
```

符号位为1



假设：其余31位用+3的二进制表示

整数的表示

int 类型的负数

-3的二进制表示

+3的二进制表示

相加得到-6的二进制表示

算数运算无法直接进行!

整数的表示

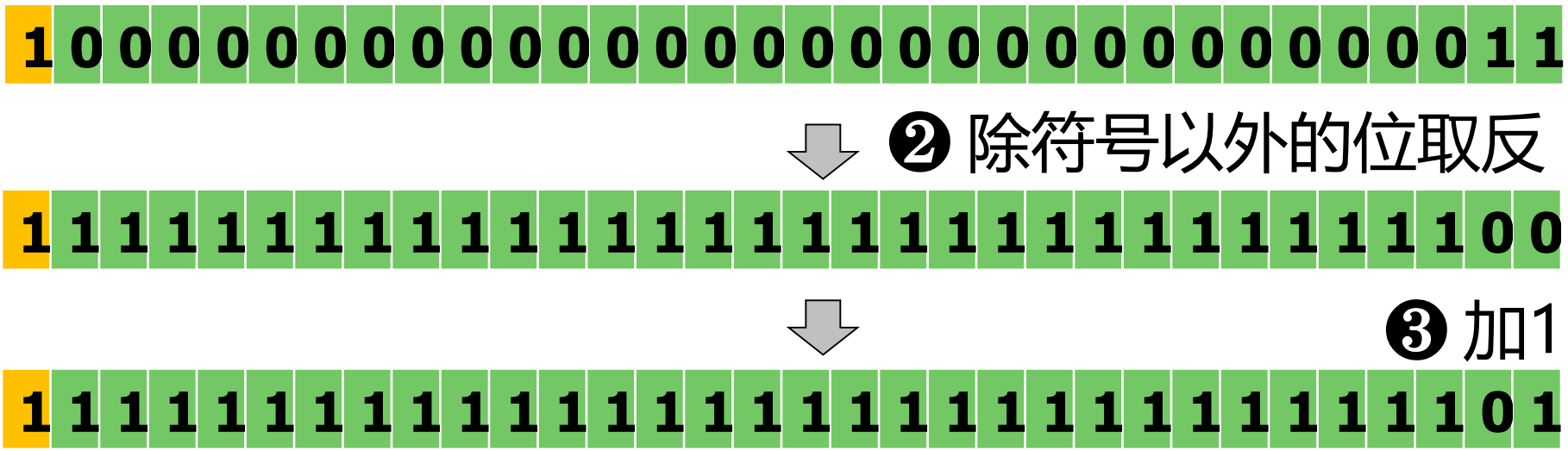
int 类型的负数 (补码表示法)

```
int x = -3;
```

分三步

符号位为1

❶ 其余31位用|-3|的二进制表示



最终得到-3的补码表示

整数的表示

int 类型的负数 （补码表示法）

-3的二进制表示（补码）

+3的二进制表示（原码）

相加得到0的原码

算数运算可以直接进行！

整数的表示

补码的性质

$[x]_{补}+[-x]_{补}=$ 模，两个相反数的补码之和等于模（正数的补码即原码）

$[x+y]_{补}=[x]_{补}+[y]_{补}$，即两数之和的补码等于各自补码的和

$[x-y]_{补}=[x]_{补}+[-y]_{补}$，即两数之差的补码等于被减数的补码与减数相反数的补码之和

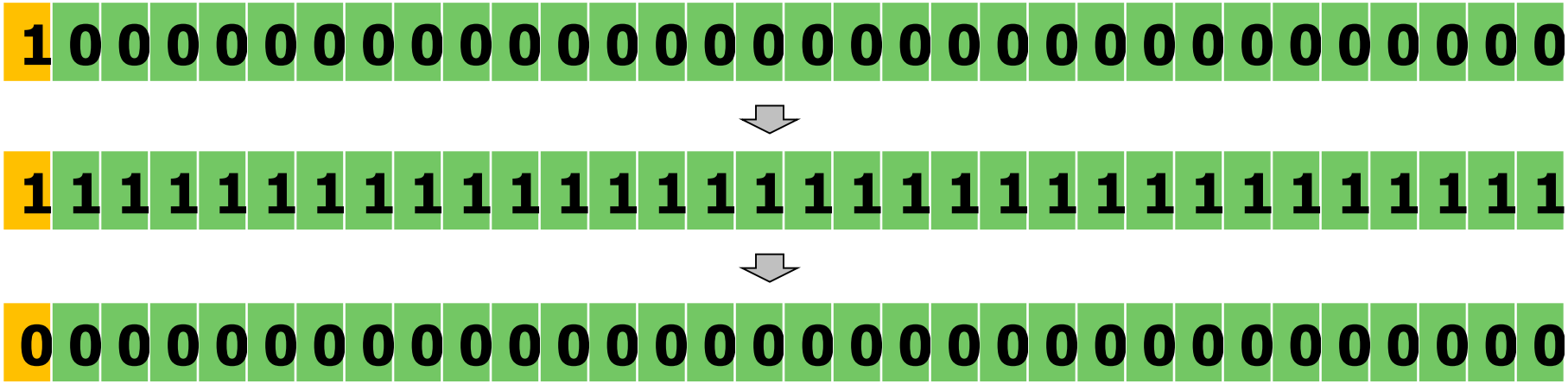
$[[x]_{补}]_{补}=[x]_{原}$，即按求补的方法，对$[x]_{补}$再求补一次，结果等于$[x]_{原}$

整数的表示

int 类型　　0 的表示

（1）将0看成正数（原码表示）

（2）将0看成负数（补码表示）



0 的原码表示和补码表示相同！

整数的表示

int 类型

-2^{31} 的补码表示：

按理说，需要33位才能表示

1 1000000000000000000000000000000

取反

1 0111111111111111111111111111111

加1

1 1000000000000000000000000000000

舍弃最高位，剩余的32位即为-2^{31}的补码表示

整数的表示

int 类型总结

- 每个int类型整数用32位二进制表示
- 正数用原码表示（正数的补码即是原码）
- 负数用补码表示
- 表示的范围： -2^{31} ~ $2^{31}-1$

整数的表示

C++提供的有符号整数类型

类型	字节数	表示范围
int	4	$-2^{31} \sim 2^{31}-1$
short	2	$-2^{15} \sim 2^{15}-1$
long	4	$-2^{31} \sim 2^{31}-1$
long long	8	$-2^{63} \sim 2^{63}-1$
char	1	$-2^{7} \sim 2^{7}-1$

浮点数的表示

计算机表示浮点数（小数）的困境

- 二进制数能够表示的状态有限
  - N位二进制数最多可表示2^N种状态
- 小数是连续的
  - 0-1之间就有无穷多个小数

解决方案：近似和压缩

浮点数的表示

计算表示浮点数的核心思想：

任意一个实数都可以近似表示成：**$(\pm)(1.a)x(2^b)$**，a称作尾数，b称作阶码

-0.75(10进制) = -0.11(2进制) = $-1.1x2^{-1}$

0.75的符号为-，尾数a为1，阶码b为-1

我们只要用二进制保存符号(±)、a的值和b的值，就可以反推出原实数！（这就是压缩的思想）

浮点数的表示

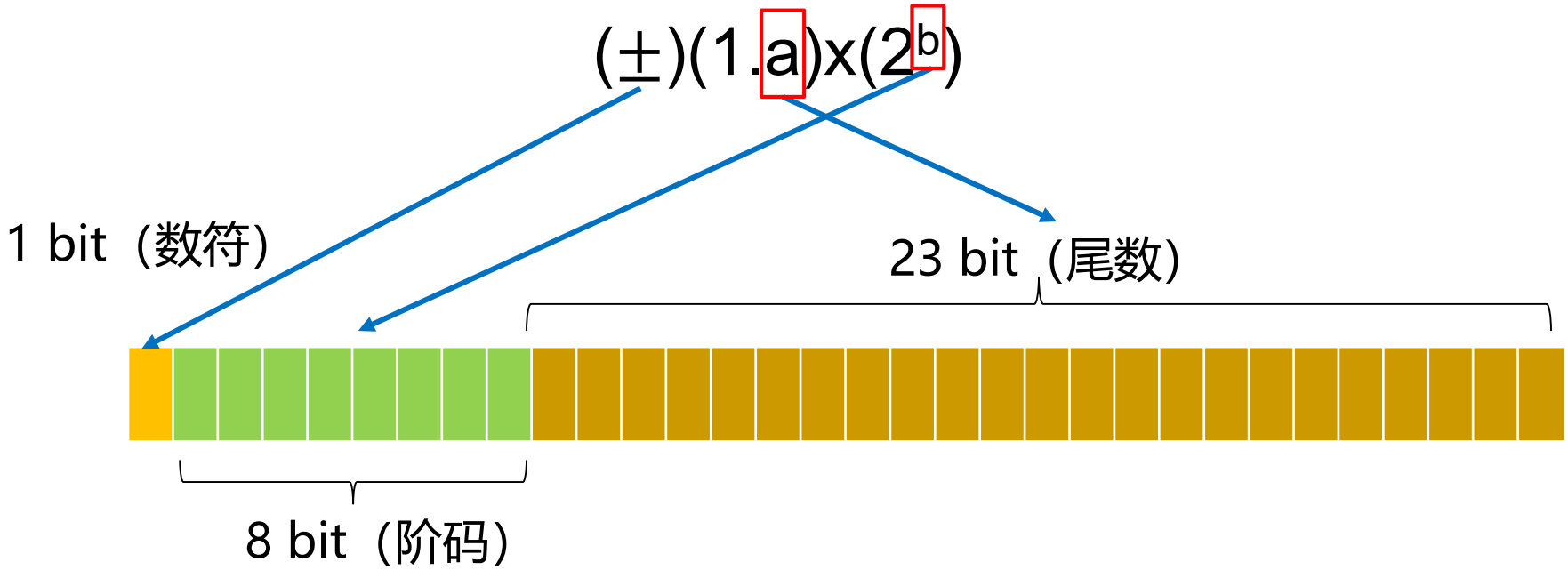
float 类型（单精度浮点数）

```
#include<iostream>
using namespace std;
int main(){
    float x = 0.35;
    cout<<x;
    return 0;
}
```

- 每个float类型数占32 bits
- 1 bit用来表示符号，8 bits用来表示阶码，23 bits用来表示尾数

浮点数的表示

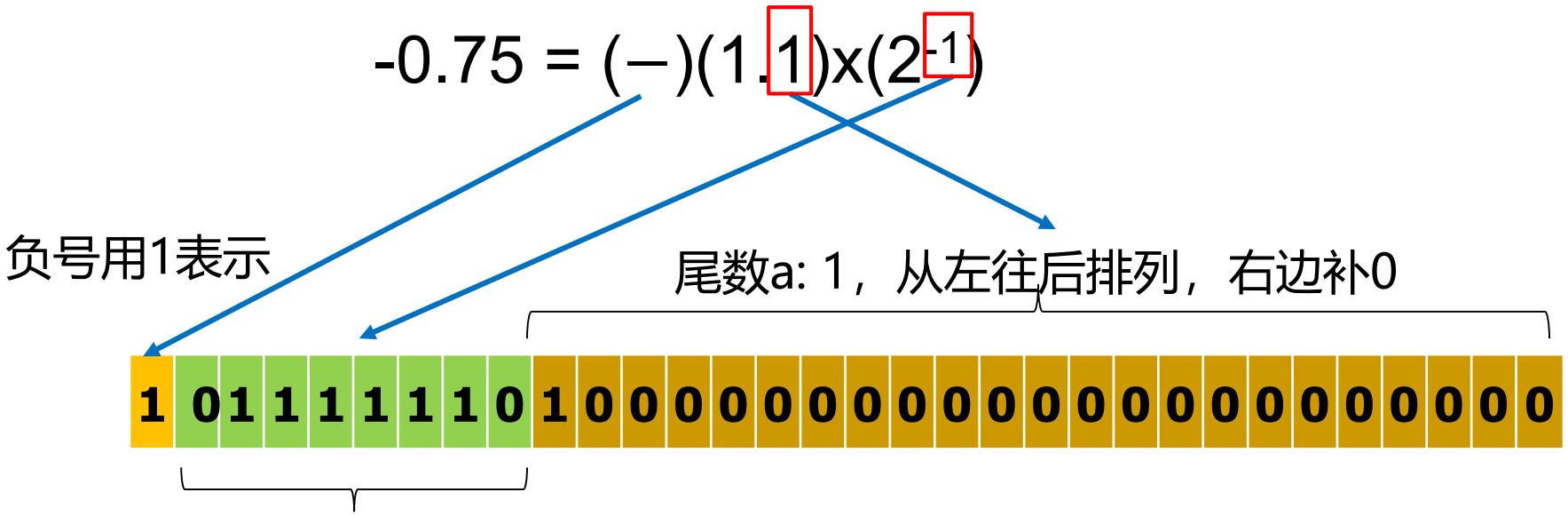
float 类型（单精度浮点数）



IEEE 754 格式（32 bit）

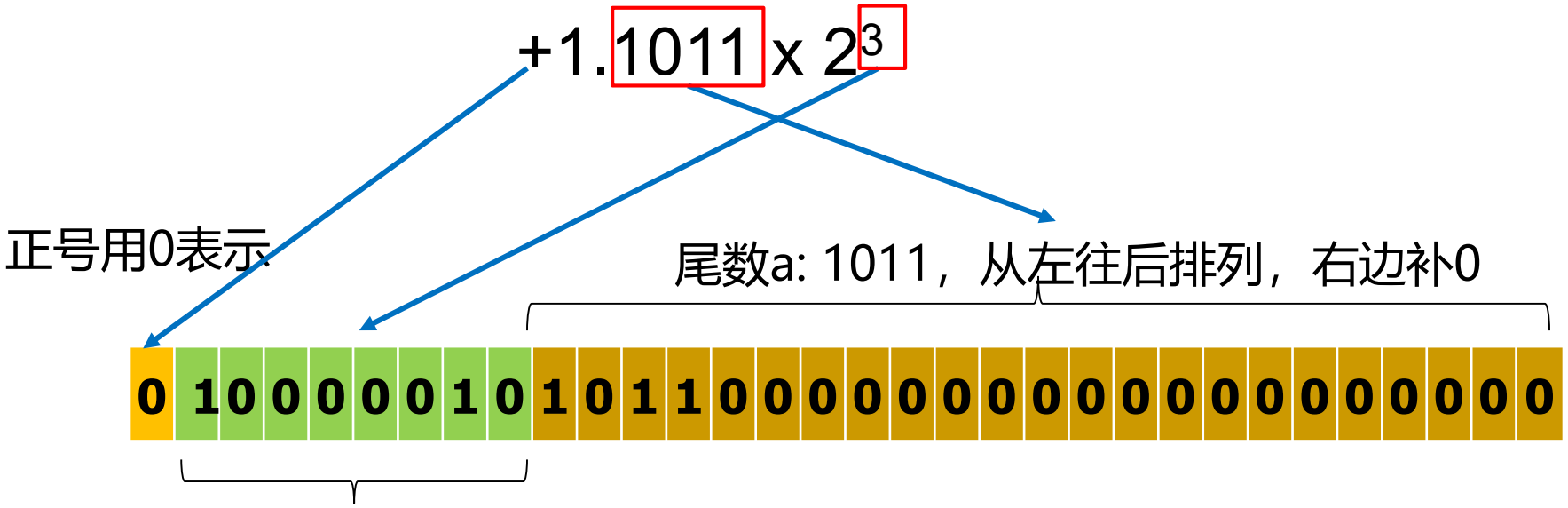
浮点数的表示

float 类型（单精度浮点数）



浮点数的表示

float 类型（单精度浮点数）



浮点数的表示

float 类型（单精度浮点数）

能表示的最大值：$(2-2^{-23})\times 2^{127} \approx 3.4\times 10^{38}$

(127 + 127) = 254

浮点数的表示

float 类型（单精度浮点数）

能表示的最小值：$-(2-2^{-23})\times 2^{127} \approx -3.4\times 10^{38}$

(127 + 127) = 254

浮点数的表示

float 类型（单精度浮点数）

特殊值：

- 阶码的8 bit 全0，表示0
- 阶码的8 bit 全1，表示无穷大

有效数字：

尾数部分表示小数点后的值，23 bits能表示的最大数是$2^{23}-1=8388607$，所以float类型能表示的最高精度是小数点后7位，即7位有效数字

浮点数的表示

C++提供的浮点数类型

类型	字节数	尾数	阶码	表示范围
float	4	23	8	-3.4E-38~3.4E+38
double	8	52	11	1.7E-308~1.7E+308

字符的表示

char 类型

```
#include<iostream>
using namespace std;
int main(){
    char c = 'A';
    char d = '0';
    char e = ';';
    return 0;
}
```

- 字符包括英文字母、数字、标点符号等
- 字符个数少于256个，可以直接用二进制数来表示

字符的表示

char 类型

ASCII码（国际标准编码）

* 每个字符用一个8 bit的二进制数表示，例如，字符G用01000111表示

* 8bit的最高位通常为0，只使用00000000～01111111为常用的128个字符编码

* 字符编码0～31以及127是不可打印字符，字符编码32是空格

* 通常用十进制或十六进制的形式来书写ASCII码，例如，字符G的ASCII码十进制写法是71

字符的表示

char 类型

ASCII编码表（国际标准编码）

8 bit中的高4位

8 bit中的低4位

L \ H	0000	0001	0010	0011	0100	0101	0110	0111
0000	NUL	DLE	SP	0	@	P	‘	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	“	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	,	7	G	W	g	w
1000	BS	CAN	)	8	H	X	h	x
1001	HT	EM	(	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	’	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

字符的表示

char 类型

```
#include<iostream>
using namespace std;
int main(){
    char c = 'A';
    cout<<c;
    c = c + 1;
    cout<<c;
    return 0;
}
```

程序输出：A B

char类型变量可以看作8bit表示的正整数（其值就是对应的ASCII码），可以进行加减乘除操作

c加1之后变为66，对应字符B的ASCII码

字符的表示

char 类型

```
#include<iostream>
using namespace std;
int main(){
    char c = 65;
    cout<<c;
    c = c + 1;
    cout<<c;
    return 0;
}
```

程序输出：A B

char类型变量可以直接赋值为整数，对应某个字符的ASCII码

字符串的表示

string 类型

```
#include<iostream>
using namespace std;
int main(){
    string s = "ABC";
    cout<<s;
    return 0;
}
```

- 字符串本质上就是一个字符序列，其中每个字符用其对应的ASCII码表示

"ABC" 的二进制表示

	'A'	'B'	'C'
ASCII码 二进制	01000001	01000010	01000011
ASCII码 十进制	65	66	67
ASCII码 十六进制	41 H	42 H	43 H

逻辑值的表示

逻辑型数据只有“逻辑真”和“逻辑假”两个值，用0来表示“逻辑假”，非0表示“逻辑真”

bool 类型

```
#include<iostream>
using namespace std;
int main(){
    bool f = true;
    bool g = false;
    cout<<f<<g;
    return 0;
}
```

程序输出：1 0

- C++提供bool 类型的变量专门表示逻辑值
- 每个bool类型变量用8 bit二进制表示
- bool 类型变量可用true和false进行赋值，值分别是1和0（真和假）

逻辑值的表示

逻辑型数据只有“逻辑真”和“逻辑假”两个值，用0来表示“逻辑假”，非0表示“逻辑真”

bool 类型

```
#include<iostream>
using namespace std;
int main(){
    bool f = 12;
    bool g = -90;
    cout<<f<<g;
    return 0;
}
```

- bool类型的变量也可以赋值为其他整数，非0表示真，0表示假

程序输出：1 1

C++支持的数据类型

基本数据类型	表示形式	C++中相应的类型标识符	占用空间(字节)
逻辑型	逻辑型	bool	1
字符型	字符型	char	1
整数类型	有符号整型	short	2
		int	4
		long	4
		long long	8
	无符号整型	unsigned short	2
		unsigned int	4
浮点数类型	单精度浮点型	float	4
	双精度浮点型	double	8

2.3 数据类型转换

数据类型转换

默认的类型转换

混合类型的算术表达式中，都转换成最宽的数据类型

```
#include<iostream>
using namespace std;
int main(){
    int a = 3;
    double b = 3.14;
    cout<<a+b; //默认转换成 double
    return 0;
}
```

数据类型转换

默认的类型转换

混合类型的算术表达式中，都转换成最宽的数据类型

```
#include<iostream>
using namespace std;
int main(){
    char c = 'A';
    cout<<c; //输出字符A
    cout<<c+1; //输出66，即字符B的ASCII值
    return 0;
}
```

c是字符类型，所以输出字符A，c+1是int类型，所以输出66

数据类型转换

默认的类型转换

用一种类型的表达式赋值给另一种类型的对象，转换成目标对象的数据类型

```
#include<iostream>
using namespace std;
int main(){
    int a = 3;
    double b = 3.14;
    a = b;  //取b的整数部分赋值给a
    b = a;  //将a转换为3.0赋值给b
    return 0;
}
```

数据类型转换

强制类型转换

用户在程序中标明如何转换

```
#include<iostream>
using namespace std;
int main(){
    int a = 3, b = 2;
    cout<<a/b; //整数相除，商取下整
    cout<<(float)a/b; //先将a转换成float类型，再除b
    cout<<(float)(a/b); //先算(a/b)，然后将结果转换为
                        float类型
    return 0;
}
```

小括号的优先级最高

程序输出：1 1.5 1.0

2.4 常量与变量

常量

- 程序运行过程中不能改变值的数据
  - 字面值常量：数值直接写在程序里面

```
#include<iostream>
using namespace std;
int main(){
    cout<<100; //十进制数
    cout<<0x64; //十六进制数
    cout<<0144; //八进制数
    cout<<100L; //long int
    cout<<100u; //unsigned int
    cout<<100.01; //double类型
    cout<<100.01f; //float类型
    return 0;
}
```

常量

- 程序运行过程中不能改变值的数据
  - 字面值常量：数值直接写在程序里面

```
#include<iostream>
using namespace std;
int main(){
    cout<<'a'; //普通字符
    cout<<'0'; //数字字符
    cout<<"hello"; //字符串
    cout<<"a"; //长度为1的字符串
    cout<<true; //逻辑常量
    cout<<1.23e5; //1.25x10^5
    return 0;
}
```

常量

- 程序运行过程中不能改变值的数据
  - 字面值常量：数值直接写在程序里面

```
#include<iostream>
using namespace std;
int main(){
    cout<<'\n'; //换行
    cout<<'\r'; //回车
    cout<<'\''; //单引号
    cout<<'\\'; //反斜杠
    cout<<"abc\"d";
    return 0;
}
```

这些字符不可见，无法直接输入，用转义字符表示：即反斜杠 \ 加一个字母

常量

- 程序运行过程中不能改变值的数据
  - 字面值常量：数值直接写在程序里面

```
#include<iostream>
using namespace std;
int main(){
    cout<<'\n'; //换行
    cout<<'\r'; //回车
    cout<<'\''; //单引号
    cout<<'\\'; //反斜杠
    cout<<"abc\"d";
    return 0;
}
```

程序输出：
'\abc"d

单引号、反斜杠、双引号这些字符有特殊的含义，为了避免混淆，也用转义字符表示：即反斜杠 \ 加原字符

常量

- 程序运行过程中不能改变值的数据
  - 常变量：变量前面加 const

```
#include<iostream>
using namespace std;
int main(){
    const double pi = 3.14159;
    double r = 2.0;
    pi = 4.15; //编译错误，常量不能修改!
    double area = pi*r*r;
    cout<<area;
    return 0;
}
```

用pi代替3.14159，让程序看起来更简洁，const可以防止别人误改

常量

- 程序运行过程中不能改变值的数据
  - 宏定义

```
#include<iostream>
using namespace std;

#define PI 3.14159

int main() {

    double r = 2.0;
    cout<<PI*r*r<<endl;

    return 0;
}
```

宏定义：
将PI定义为3.14159，程序所有出现PI的地方，都会被替换成3.14159

PI在程序编译阶段会被替换为3.14159

常量

- 程序运行过程中不能改变值的数据
  - 宏定义

```
#include<iostream>
using namespace std;

#define X 100+50

int main() {

    cout<<X<<endl;
    cout<<X*X<<endl;

    return 0;
}
```

宏定义：
将X定义为100+50，
程序中所有出现X的地方都将被换成100+50

宏定义的替换是符号替换，即将100+50看成一串符号，并不是看成加法，所以不会替换为150

替换为100+50*100+50，注意没有括号！

变量

- 程序运行过程中其值能够改变的数据
- 根据变量的数据类型，在内存中分配相应大小的空间

```
#include<iostream>
using namespace std;

int main() {
    int x = 4;
    cout<<x;
    x = 6;
    cout<<x;
    return 0;
}
```

x的值会发生变化：初始值为4，中途被重新赋值为6

变量

- 程序运行过程中其值能够改变的数据
- 根据变量的数据类型，在内存中分配相应大小的空间

- 变量必须先定义，后使用
- 定义的时候可以初始化
- 变量存储在内存中，空间大小由其数据类型决定

```
#include<iostream>
using namespace std;
int main() {
    int x, y;
    cin>>x>>y;
    double z = 1.0;
    z = z + x + y;
    cout<<z;
    return 0;
}
```

变量的作用域

- 作用域：变量有效的范围

```
#include<iostream>
using namespace std;
int a = 10;
int main() {
    int x = 4;
    cout<<x;
    {
      int y = 6;
      x + y; //正确
      y + a; //正确
    }
    x + a; //正确
    x + y; //错误，y不可见
    return 0;
}
```

全局变量，整个程序文件可见

局部变量，仅main函数可见

局部变量，仅花括号内可见

可简单理解为：
包含变量的最里层的一对花括号，就是变量的作用域

变量的生命期

- 生命期：变量从创建到消失的时间范围

```
#include<iostream>
using namespace std;
int a = 10;
int main() {
    int x = 4;
    cout<<x;
    {
      int y = 6;
      x + y; //正确
      y + a; //正确
    }
    x + a; //正确
    x + y; //错误，y在此处不可见
    return 0;
}
```

全局变量，整个程序执行周期都在

局部变量x，从创建到main函数结束

局部变量y，从创建到花括号结束

END