第3章

基本数据类型和表达式

基本语法单位

基本符号

- 数字 0-9
- 大小写字母 a-zA-Z
- 特殊符号 +-*/%<<=&&……

关键字

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

特定字

- 不是关键字
- 用于预处理

`define、undef、include、ifdef、endif、line……`

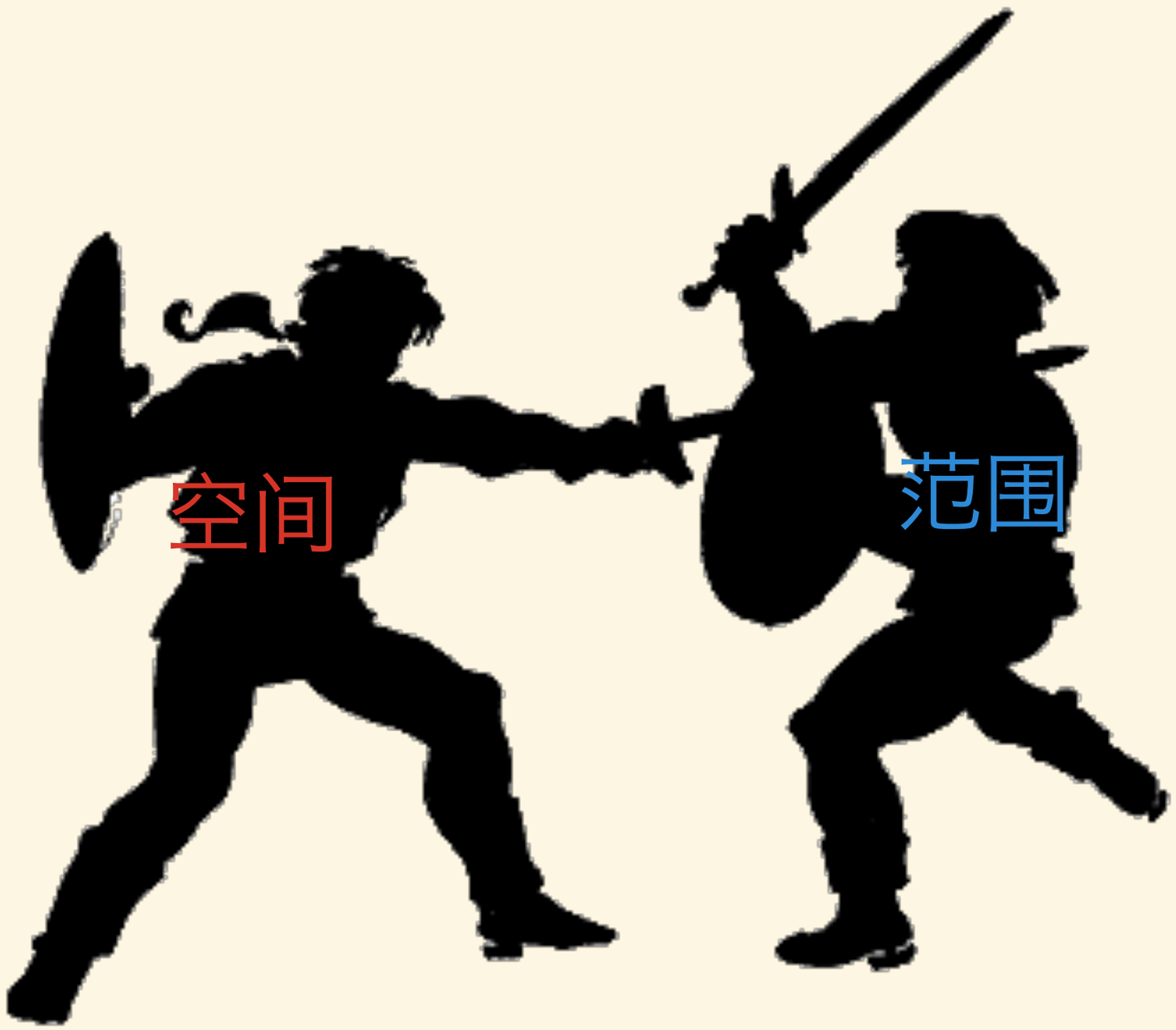
标识符

- 以英文字母或下划线开头
- 由字母、数字、下划线组成
- 大小写敏感
- 不能使用关键字
- 习惯上常量用大写字母，其余用小写

smart
5smart
bomb?
decision
key_board
key-board
FLOAT
float

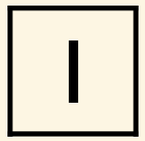
smart
5smart
bomb?
decision
key_board
key-board
FLOAT
float

数据类型


空间
范围

1位

1位

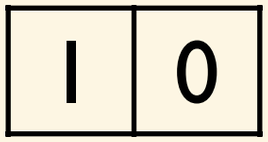


范围：0..1

2位

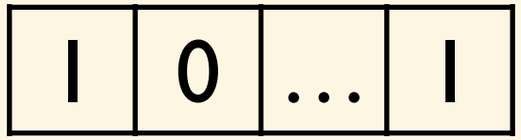
1	0

2位



范围：0..3

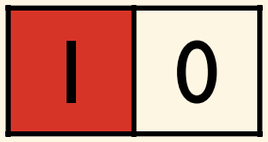
n位



范围：$0..2^n-1$

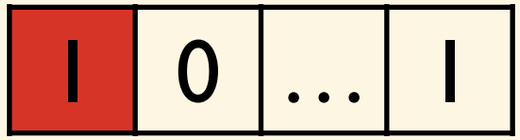
有符号2位

有符号2位



范围：-2..1

有符号n位



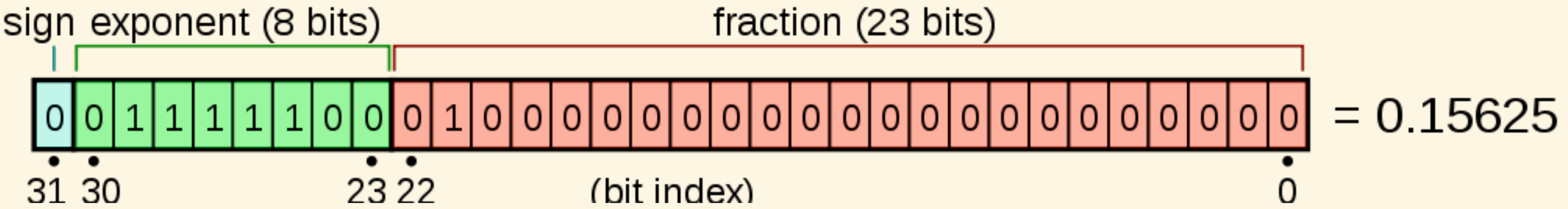
范围：$-2^{n-1}..2^{n-1}-1$

单一类型可行吗?

为什么需要多种类型?

关键字	位长(字节)	范围
char	1	-128..127（或0..255，与体系结构相关）
unsignedchar	1	0..255
signedchar	1	-128..127
int	2或4	-32768..32767或 -2147483648..2147483647
unsignedint	2或4	0..65535或 0..4294967295
signedint	2或4	-32768..32767或 -2147483648..2147483647
shortint	2	-32768..32767
unsignedshort	2	0..65535
signedshort	2	-32768..32767
longint	4或8	-2147483648..2147483647或 -9223372036854775808..9223372036854775807
unsignedlong	4或8	0..4294967295或 0..18446744073709551615
signedlong	4或8	-2147483648..2147483647或 -9223372036854775808..9223372036854775807
longlong	8	-9223372036854775808..9223372036854775807
unsignedlonglong	8	0..18446744073709551615
float	4	3.4x10−38..3.4x10+38(7sf)
double	8	1.7x10−308..1.7x10+308(15sf)
longdouble	8或以上	編譯器相關

浮点数


sign
exponent (8 bits)
fraction (23 bits)
0 0 1 1 1 1 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
= 0.15625
31 30 23 22 (bit index) 0

IEEE754

C语言中的类型

类型

- 基本型
- 构造类型
- 指针类型
- 空类型

常量

整型

- 十进制

±n 123 -1000 1.234 10-2 10/3

- 八进制

±0n 0123 -01000 0128 123 670

- 十六进制

±0xn 012c -0XFF

- 长整型

±nL 123456L 075L 0xFFFFL

实型

- [digits][.digits][E|e[+|-]digits]

```
1.57E10 .571 .57e2 -0.0025 -2.5e-3 25e-4

1.57e1.2 e3 .e3 e
```

字符型

- 单引号括起来的单个字符

‘a’ ‘9’ ‘!’

Binary	Oct	Dec	Hex	Glyph
010 0000	040	32	20	
010 0001	041	33	21	!
010 0010	042	34	22	"
010 0011	043	35	23	#
010 0100	044	36	24	$
010 0101	045	37	25	%
010 0110	046	38	26	&
010 0111	047	39	27	'
010 1000	050	40	28	(
010 1001	051	41	29	)
010 1010	052	42	2A	*
010 1011	053	43	2B	+
010 1100	054	44	2C	,
010 1101	055	45	2D	-
010 1110	056	46	2E	.
010 1111	057	47	2F	/
011 0000	060	48	30	0
011 0001	061	49	31	1
011 0010	062	50	32	2
011 0011	063	51	33	3
011 0100	064	52	34	4
011 0101	065	53	35	5
011 0110	066	54	36	6
011 0111	067	55	37	7
011 1000	070	56	38	8
011 1001	071	57	39	9
011 1010	072	58	3A	:
011 1011	073	59	3B	;
011 1100	074	60	3C	<
011 1101	075	61	3D	=
011 1110	076	62	3E	>
011 1111	077	63	3F	?

Binary	Oct	Dec	Hex	Glyph
100 0000	100	64	40	@
100 0001	101	65	41	A
100 0010	102	66	42	B
100 0011	103	67	43	C
100 0100	104	68	44	D
100 0101	105	69	45	E
100 0110	106	70	46	F
100 0111	107	71	47	G
100 1000	110	72	48	H
100 1001	111	73	49	I
100 1010	112	74	4A	J
100 1011	113	75	4B	K
100 1100	114	76	4C	L
100 1101	115	77	4D	M
100 1110	116	78	4E	N
100 1111	117	79	4F	O
101 0000	120	80	50	P
101 0001	121	81	51	Q
101 0010	122	82	52	R
101 0011	123	83	53	S
101 0100	124	84	54	T
101 0101	125	85	55	U
101 0110	126	86	56	V
101 0111	127	87	57	W
101 1000	130	88	58	X
101 1001	131	89	59	Y
101 1010	132	90	5A	Z
101 1011	133	91	5B	[
101 1100	134	92	5C	\
101 1101	135	93	5D	]
101 1110	136	94	5E	^
101 1111	137	95	5F	_

Binary	Oct	Dec	Hex	Glyph
110 0000	140	96	60	`
110 0001	141	97	61	a
110 0010	142	98	62	b
110 0011	143	99	63	c
110 0100	144	100	64	d
110 0101	145	101	65	e
110 0110	146	102	66	f
110 0111	147	103	67	g
110 1000	150	104	68	h
110 1001	151	105	69	i
110 1010	152	106	6A	j
110 1011	153	107	6B	k
110 1100	154	108	6C	l
110 1101	155	109	6D	m
110 1110	156	110	6E	n
110 1111	157	111	6F	o
111 0000	160	112	70	p
111 0001	161	113	71	q
111 0010	162	114	72	r
111 0011	163	115	73	s
111 0100	164	116	74	t
111 0101	165	117	75	u
111 0110	166	118	76	v
111 0111	167	119	77	w
111 1000	170	120	78	x
111 1001	171	121	79	y
111 1010	172	122	7A	z
111 1011	173	123	7B	{
111 1100	174	124	7C	|
111 1101	175	125	7D	}
111 1110	176	126	7E	~

‘ع’ vs ع

‘9’ - 1 = ?

如何表示回车?

转义符

转义字符	意义	ASCII码值（十进制）
\a	响铃(BEL)	007
\b	退格(BS)	008
\f	换页(FF)	012
\n	换行(LF)	010
\r	回车(CR)	013
\t	水平制表(HT)	009
\v	垂直制表(VT)	011
\\	反斜杠	092
\?	问号字符	063
\'	单引号字符	039
\"	双引号字符	034
\0	空字符(NULL)	000
\ddd	任意字符	三位八进制
\xhh	任意字符	二位十六进制

字符串

- 双引号括起来的一串字符

```
“hello\n”
```

h	e	l	l	o	\n	\0

符号常量

```
#define <符号常量名> <常量>

#define PI  3.1415926

#define TRUE 1

#define FALSE 0

#define STAR ‘*’
```

变量

声明

```
type variable_list;

int i, j, k;
short int si;

unsigned int ui;

double balance, profit, loss;
```

整型

```c
#include <stdio.h>

int main()
{
    int a,b,c,d;
    unsigned u;

    a=22;
    b=-11;
    u=5;
    c=a+u;
    d=b+u;
    printf("a+u=%d, b+u=%d\n",c,d);

    return 0;
}
```

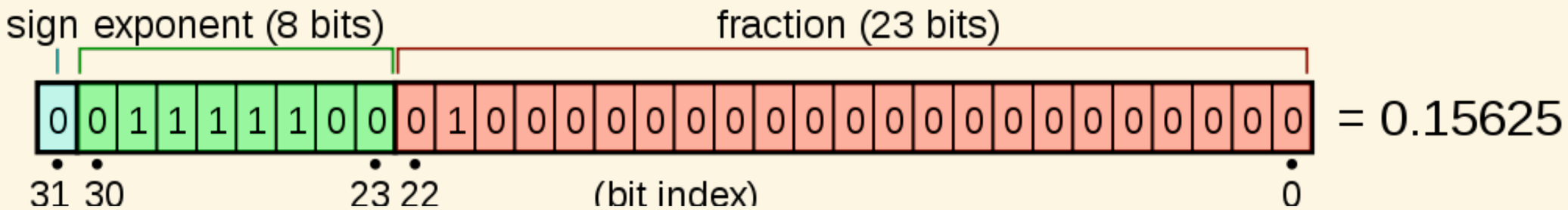
整型

```c
#include <stdio.h>

int main()
{
    int a,b,c,d;
    unsigned u;

    a=22;
    b=-11;
    u=5;
    c=a+u;
    d=b+u;
    printf("a+u=%d, b+u=%d\n",c,d);

    return 0;
}
```

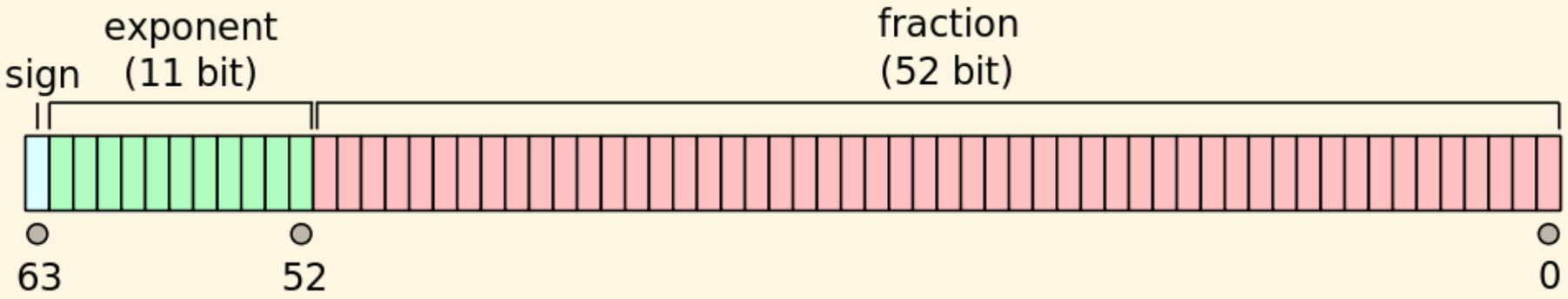
```
a+u=27, b+u=-6
```

实型

```
float x, y;
```




```
double z;
```



字符型

- 只能放字符，不能放字符串

```
char c1, c2;


c1 = 'a';

c2 = 'b';
```

初始化

```
char c = 'A', ky = 'K';

int j, i = 1;

float sum = 3.56;

int a = b = c = 3;

int a = 3, b = 3, c = 3;

int x;

x = 10;
```

表达式和运算符

表达式

- 由操作数和运算符组成
- 操作数包括常量、变量、函数和表达式
- 表达式具有类型和值

运算符

- 优先级
- 结合律

算术运算符

说明

运算符	作用
–	减法，也是一元减法
+	加法
*	乘法
/	除法
%	模运算
--	自减（减1）
++	自增（增1）

优先级

1.++、--

2.-（取反）

3.*、/、%

4.+、-

类型和值

- /
- % 只适用于整数
- ++x 与 x++
- 优先级

赋值运算符

简单赋值

- <变量标识符> = <表达式>
- 值和类型

```
x = 10;

a = b = c = d = 0; ⇔ a = (b = (c = (d = 0)));

float f = 23;

int i = 3.56;
```

复合赋值

- 相当于两个运算符的结合
- 更简洁、效率更高

```
a += b; ⇔ a = a + b;

x *= y + 10 - z; ⇔ x = x * (y + 10 - z);

a = (b = 5);

a = b = c = 5;

a = 5 + (c = 6);

a = (b = 4) + (c = 6);

a = (b = 10) / (c = 2);
```

关系运算符

真? 假?

- 判断真假时

真 ⇔ 非0

假 ⇔ 0

- 返回值

真 ⇔ 1

假 ⇔ 0

说明

优先级	运算符	意义	例	结果
6	<	小于	A’<’B’	真
	<=	小于等于	12.5<=10	假
	>	大于	A’>’B’	假
	>=	大于等于	A’+2>=’B’	真
7	==	等于	A’= =’B’	假
	!=	不等	A’!=’B’	真

逻辑运算符

真值表

优先级	运算符	意义	例	结果
2	!	逻辑非	！ 7	0
11	&&	逻辑与	A’&&’B’	1
12	||	逻辑成	3||4	1

a	b	!a	!b	a&&b	a||b
0	0	1	1	0	0
0	1	1	0	0	1
1	0	0	1	0	1
1	1	0	0	1	1

```
5 > 3 && 2 || 8 < 4 - !0
```

闰年的逻辑表达式

```
'a' && 'd'
```

```
int a = 1, b = 2, c = 5, d = 4;
a > b && c = d * 3 && a + b;
```


```
int a = 4, b = 5;
a && b;
a || b;
!a && b;
```

位运算符

运算符	含义	描述
&	按位与	如果两个相应的二进制位都为1，则该位的结果值为1，否则为0
|	按位或	两个相应的二进制位中只要有一个为1，该位的结果值为1
^	按位异或	若参加运算的两个二进制位值相同则为0，否则为1
~	取反	~是一元运算符，用来对一个二进制数按位取反，即将0变1，将1变0
<<	左移	用来将一个数的各二进制位全部左移N位，右补0
>>	右移	将一个数的各二进制位右移N位，移到右端的低位被舍弃，对于无符号数，高位补0

```
3 & 5;
3 && 5;
3 | 5;
3 || 5;
3 << 2;
5 >> 1;
```

逗号运算符

<表达式1>, <表达式2>, <表达式3>,…, <表达式n>

```
3+5, 6+8
a = 3 * 5, a * 4

x = (a = 3, 6 * 3)
x = a = 3, 6 * 3
```

条件运算符

<表达式1> ? <表达式2> : <表达式3>

```
m = (a > b) ? a : b
```

```
if (a > b)
    m = a
else
    m = b
```

类型转换

1. 若一个操作数类型为long double，则其他操作数都转化为long double；

2. 否则，若一个操作数的类型为double，则其他操作数都转化为double；

3. 否则，若一个操作数的类型为float，则表达式中其他操作数都转化为float类型；

4. 否则，若一个操作数的类型为unsigned long int，则表达式中其他操作数都转化为unsigned long int类型；

5. 否则，若一个操作数的类型为long int，其他操作数中有unsigned int， 如果long int能容纳unsigned int所有的值，那么unsigned int转化为long int，如果long int不能容纳unsigned int的值，那么所有的类型都转化为long int；

6. 否则，若一个操作数的类型为long int，则其他操作数都转化为long int类型；

7. 否则，若一个操作数为unsigned int，则其他操作数都转化为unsigned int。