程序设计
Programming

Lecture 3: 数据类型和控制语句

Lecture 2 回顾

- C语言中的基本元素
  - ✓标识符：变量、常量、关键字
  - ✓运算符：分类、优先级、结合性
  - ✓表达式和语句
  - ✓格式化输入输出：格式串、转换说明、转义字符
  - ✓数据类型：类型、内存长度、取值范围

C语言基本数据类型

基本数据类型及其取值范围：有符号(signed)

- 字符型 char，1字节，取值范围 -128 ~ 127
- 短整型 short，2字节，取值范围 -32,768 ~ 32,767
- 整型 int，4字节，取值范围 -2,147,483,648 ~ 2,147,483,647
- 长整型 long，8字节，取值范围很大（你的机器有可能4字节）
- 单精度浮点型 float，4字节，取值范围 -3.4*10^38 ~ 3.4*10^38，大约精确到小数点后6位
- 双精度浮点型 double，8字节，取值范围 -1.7*10^308 ~ 1.7*10^308，大约精确到小数点后15位

基本数据类型及其取值范围：有符号(signed)

- 字符型 char，1字节，取值范围 -128 ~ 127
- 短整型 short，2字节，取值范围 -32,768 ~ 32,767
- 整型 int，4字节，取值范围 -2,147,483,648 ~ 2,147,483,647
- 长整型 long，8字节，取值范围很大（你的机器有可能4字节）
- 单精度浮点型 float，4字节，取值范围 -3.4*10^38 ~ 3.4*10^38，大约精确到小数点后6位
- 双精度浮点型 double，8字节，取值范围 -1.7*10^308 ~ 1.7*10^308，大约精确到小数点后15位

signed可加可不加，即基本数据类型默认为有符号，比如signed int 等价于 int

基本数据类型及其取值范围：无符号(unsigned)

- unsigned char，1字节，取值范围 0 ~ 255
- unsigned short，2字节，取值范围 0 ~ 65,535
- unsigned int，4字节，取值范围 0 ~ 4,294,967,295
- unsigned long，8字节，取值范围很大

基本数据类型及其取值范围：无符号(unsigned)

- unsigned char，1字节，取值范围 0 ~ 255
- unsigned short，2字节，取值范围 0 ~ 65,535
- unsigned int，4字节，取值范围 0 ~ 4,294,967,295
- unsigned long，8字节，取值范围很大

1、必须加unsigned

基本数据类型及其取值范围：无符号(unsigned)

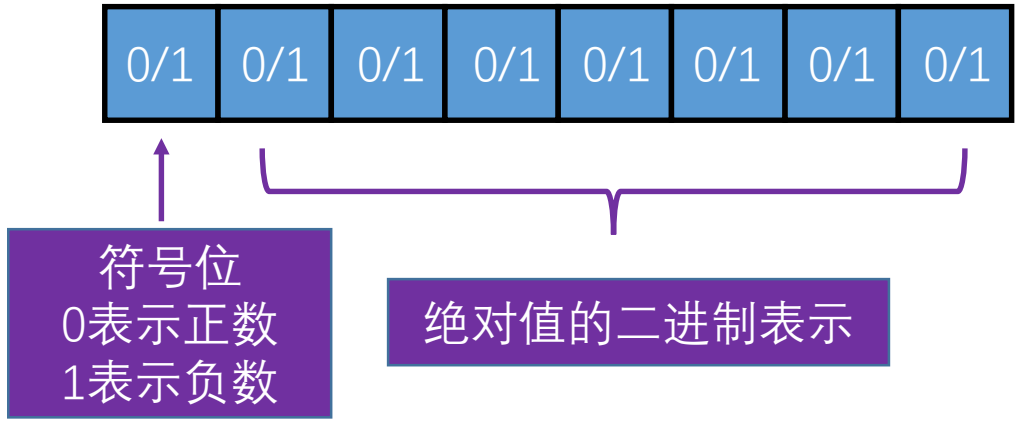
- unsigned char，1字节，取值范围 0 ~ 255
- unsigned short，2字节，取值范围 0 ~ 65,535
- unsigned int，4字节，取值范围 0 ~ 4,294,967,295
- unsigned long，8字节，取值范围很大

1、必须加unsigned

2、float和double没有无符号类型

有符号整型和字符型在内存中的表示

- char类型在内存中占据1个字节，即8个bit



- 最大：01111111 = 2^6+2^5+…+2^0 = 127
- 最小：10000000 = 2^7 = -128

原码、反码和补码

- 原码：最高位为符号位，其余位表示数值的绝对值
  - ✓13的原码为：00001101
  - ✓-13的原码为：10001101

问题在于：13 + (-13) = 10011010

原码、反码和补码

- 原码：最高位为符号位，其余位表示数值的绝对值
  - ✓13的原码为：00001101
  - ✓-13的原码为：10001101

问题在于：13 + (-13) = 10011010

- 反码：正数的反码等于原码；负数的反码符号位不变，其余位将原码表示取反
  - ✓13的反码为：00001101
  - ✓-13的反码为：11110010

13 + (-13) = 11111111

原码、反码和补码

- 原码：最高位为符号位，其余位表示数值的绝对值
  - ✓13的原码为：00001101
  - ✓-13的原码为：10001101

问题在于：13 + (-13) = 10011010

- 反码：正数的反码等于原码；负数的反码符号位不变，其余位将原码表示取反
  - ✓13的反码为：00001101
  - ✓-13的反码为：11110010

13 + (-13) = 11111111

- 补码：正数的补码等于原码，负数的补码等于反码在最低位加1
  - ✓ 13的补码为：00001101
  - ✓ -13的补码为：11110011

13 + (-13) = 00000000 = 0

原码、反码和补码

- 原码：最高位为符号位，其余位表示数值的绝对值
  - ✓13的原码为：00001101
  - ✓-13的原码为：10001101

问题在于：13 + (-13) = 10011010

- 反码：正数的反码等于原码；负数的反码符号位不变，其余位将原码表示取反
  - ✓13的反码为：00001101

13 + (-13) = 11111111

在计算机内部有符号整型和字符型都是以补码形式存储和运算！！

  - ✓ 13的补码为：00001101
  - ✓ -13的补码为：11110011

13 + (-13) = 00000000 = 0

一个特殊的补码

- char的取值范围
  - ✓正数补码 00000000 ~ 01111111，等价于 0 ~ 127
  - ✓负数补码 10000001 ~ 11111111，等价于 -127 ~ -1

- 特殊的补码：10000000 =?
  - ✓观点1：假如char用16位表示，-128的补码为11111111|10000000，截断高位字节后正好为10000000
  - ✓观点2：-127的补码为10000001，减1为10000000，-127-1=-128
  - ✓所以10000000直接被当作-128的补码，char取值范围-128 ~ 127

ASCII表，char类型的字符表示

- char类型
  - ✓以%d作为转换说明，则打印整数（ASCII值）
  - ✓以%c作为转换说明，则打印字符

- 字符型与整型
  - ✓字符‘A’ 和整数65在内存中的表示相同
  - ✓int a = ‘A’;
  - ✓char a = 65;

不建议这样写！！

ASCII表

ASCII值	控制字符	ASCII值	控制字符	ASCII值	控制字符	ASCII值	控制字符
0	NUT	32	(space)	64	@	96	、
1	SOH	33	!	65	A	97	a
2	STX	34	”	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	$	68	D	100	d
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	,	71	G	103	g
8	BS	40	(	72	H	104	h
9	HT	41	)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DCI	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	X	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	TB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[	123	{
28	FS	60	<	92	\	124	|
29	GS	61	=	93	]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	DEL

整型数据类型转换

- 有时候，不同数据类型出现在同一个表达式，或者需要强制转换数据类型

- 内存长度短的往内存长度长的类型转换
  - ✓将补码按最高位补齐

```
char a = 'A';
short s = a;
```

- ✓a: 01000001
- ✓s: 00000000 01000001

整型数据类型转换

- 内存长度短的往内存长度长的类型转换
  - ✓将补码按最高位补齐

char a = -65;

short s = a;

- ✓a: 10111111
- ✓s: 11111111 10111111

整型数据类型转换

- 内存长度长的往内存长度短的类型转换
  - ✓将高位字节直接截断

short s = 65;

char a = s;

- ✓s: 00000000 01000001
- ✓a: 01000001

整型数据类型转换

- 内存长度长的往内存长度短的类型转换
  - ✓将高位字节直接截断

short s = 513;

char a = s;

- ✓s: 00000010 00000001
- ✓a: 00000001

精度损失！！

整型数据类型转换

- 内存长度长的往内存长度短的类型转换
  - ✓将高位字节直接截断

```
short s = -255;
char a = s;
```

  - ✓s: ?
  - ✓a: ?

整型数据类型转换

- 内存长度长的往内存长度短的类型转换
  - ✓将高位字节直接截断

short s = -255;

char a = s;

- ✓s: ?　　1111111100000001
- ✓a: ?　　00000001

无符号整型和字符型在内存中的表示

- 所有位数用来表示数值的绝对值

- unsigned char占据1个字节
  - ✓最小 00000000 = 0
  - ✓最大 11111111 = 255

- unsigned int占据4个字节
  - ✓最小0
  - ✓最大2^32 – 1 = 4,294,967,295

浮点型在内存中的表示

- float型占据4个字节，即32个bit

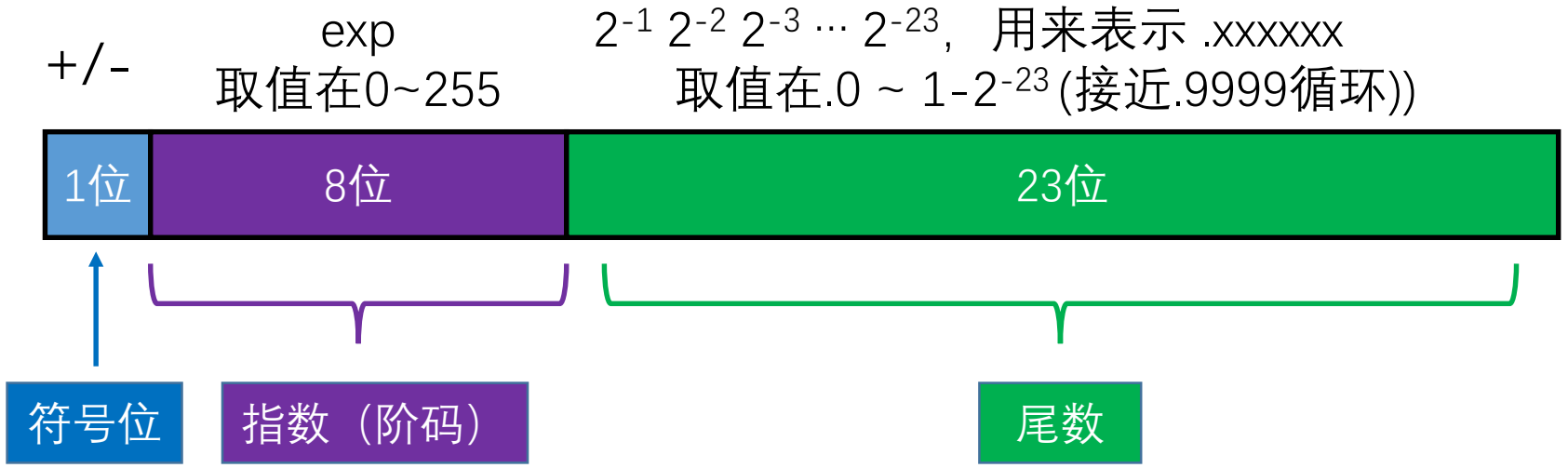
浮点型在内存中的表示

- float型占据4个字节，即32个bit

任何浮点数都可以表示为：(+/-) 1.xxxxxx × 2^X

浮点型在内存中的表示

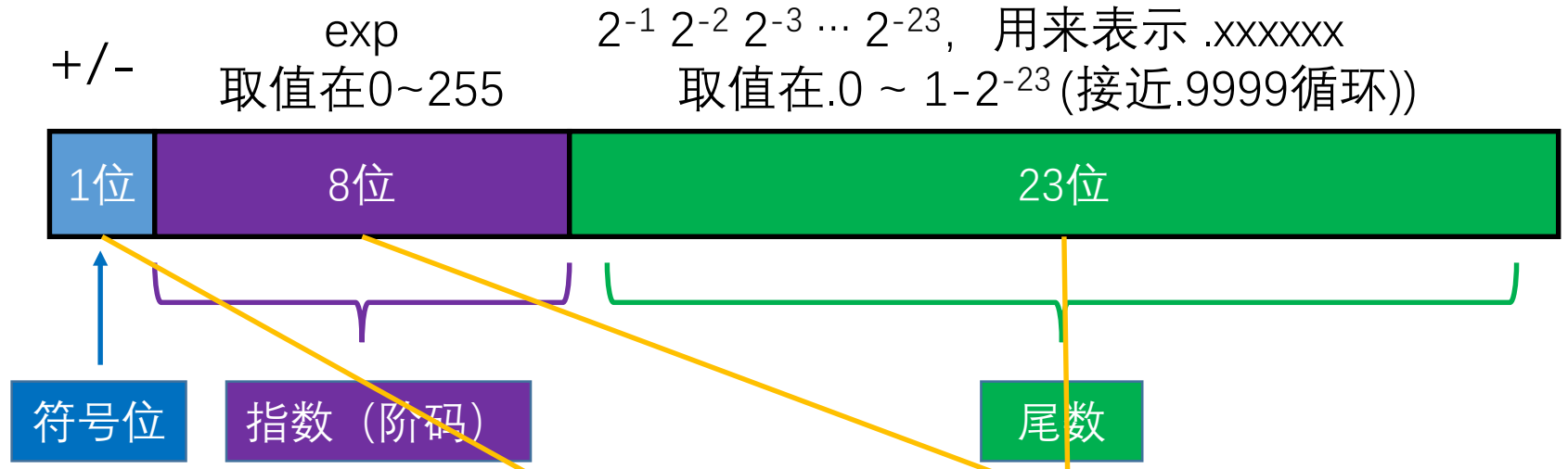
- float型占据4个字节，即32个bit



任何浮点数都可以表示为：(+/-) 1.xxxxxx × 2^X

浮点型在内存中的表示

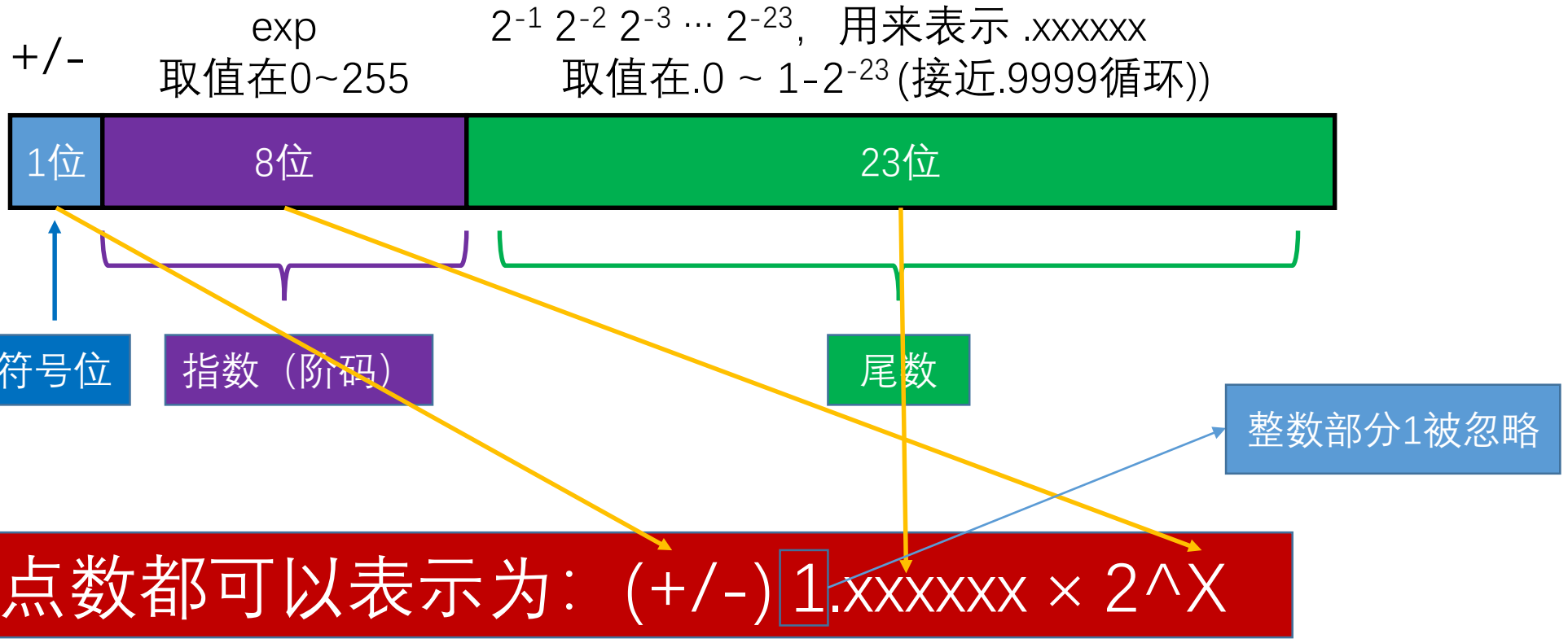
- float型占据4个字节，即32个bit



任何浮点数都可以表示为：(+/-) 1.xxxxxx × 2^X

浮点型在内存中的表示

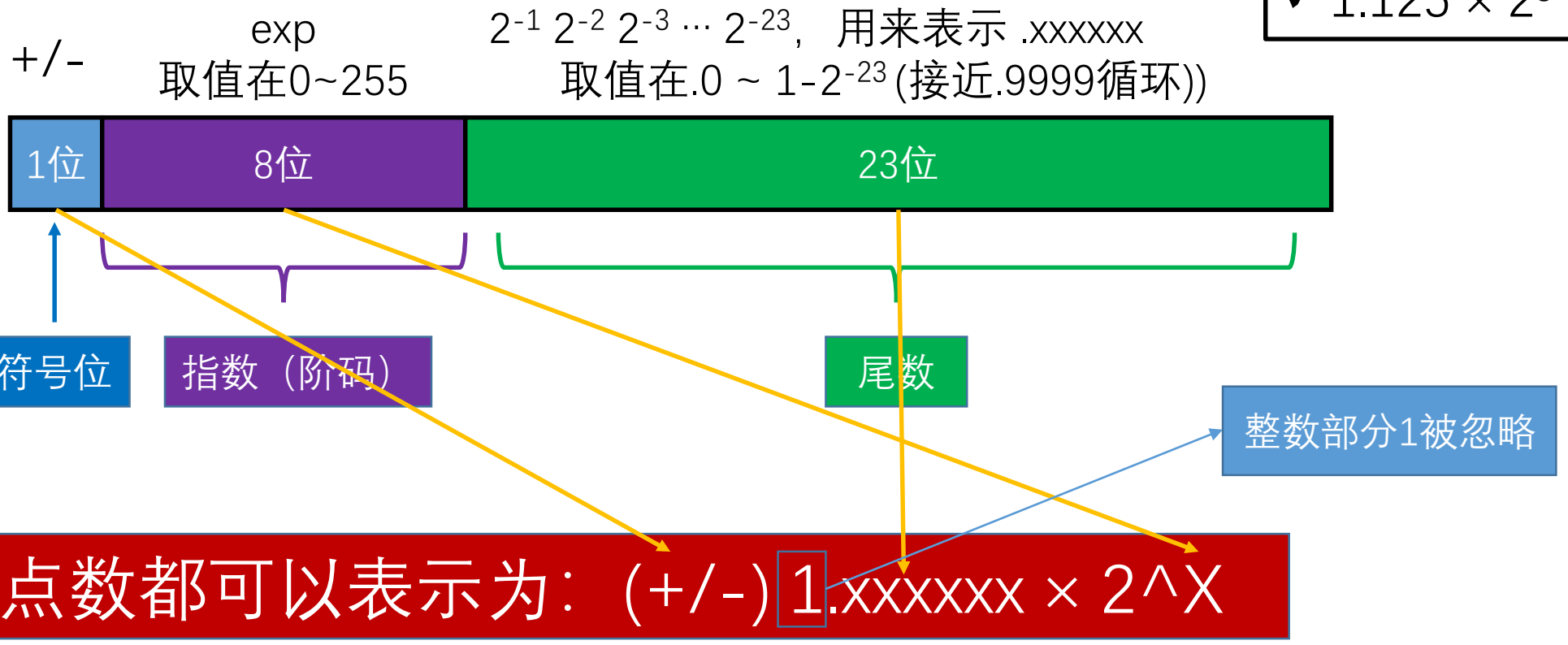
- float型占据4个字节，即32个bit



浮点型在内存中的表示

浮点数9.0:
- ✓ 9.0×2^0
- ✓ 4.5×2^1
- ✓ 2.25×2^2
- ✓ 1.125×2^3

- float型占据4个字节，即32个bit

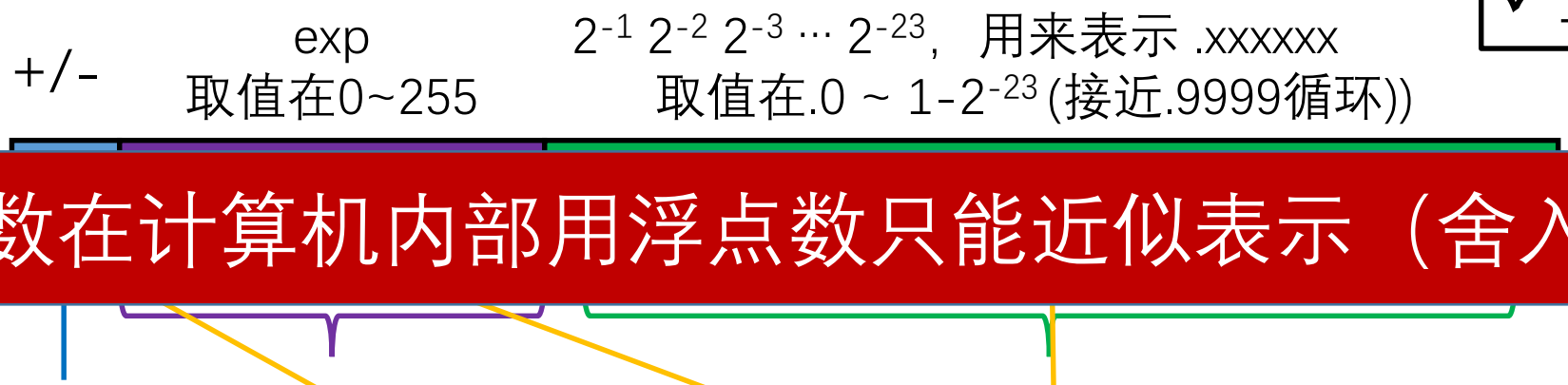


任何浮点数都可以表示为：(+/-) 1.xxxxxx × 2^X

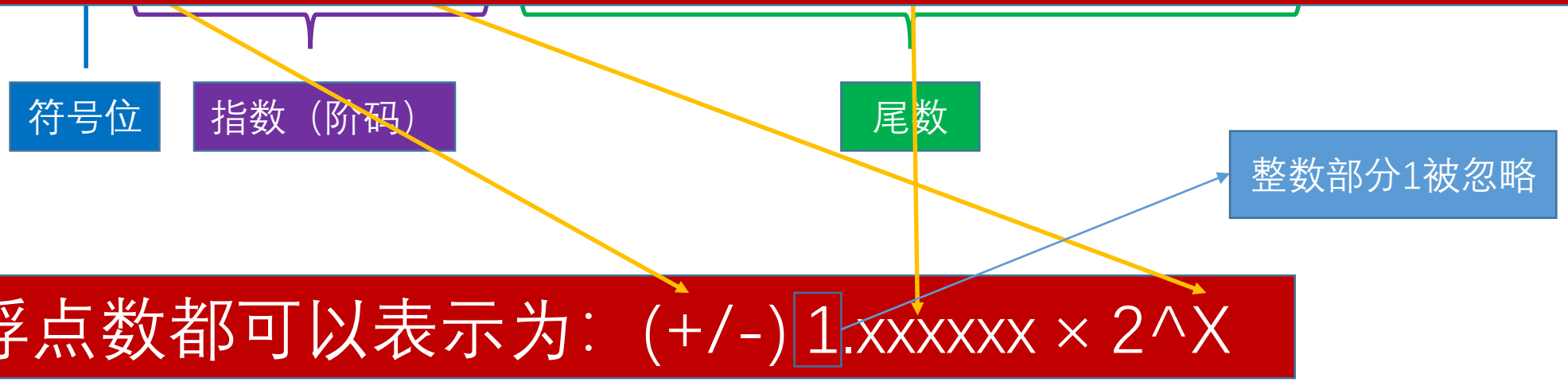
浮点型在内存中的表示

浮点数9.0:
- ✓ 9.0×2^0
- ✓ 4.5×2^1
- ✓ 2.25×2^2
- ✓ 1.125×2^3

- float型占据4个字节，即32个bit



很多实数在计算机内部用浮点数只能近似表示（舍入）！！

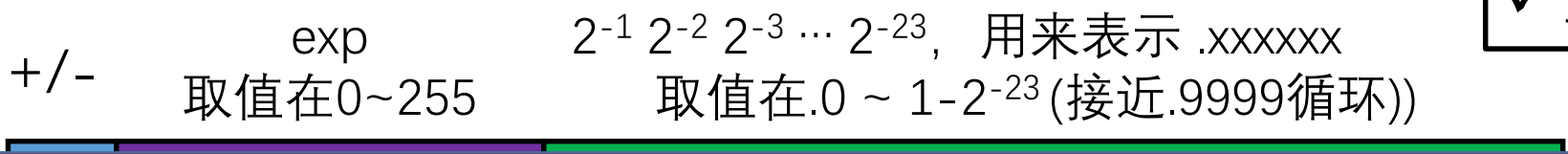


任何浮点数都可以表示为：(+/-) 1.xxxxxx × 2^X

浮点型在内存中的表示

浮点数9.0:
- ✓ 9.0×2^0
- ✓ 4.5×2^1
- ✓ 2.25×2^2
- ✓ 1.125×2^3

- float型占据4个字节，即32个bit



很多实数在计算机内部用浮点数只能近似表示（舍入）！！

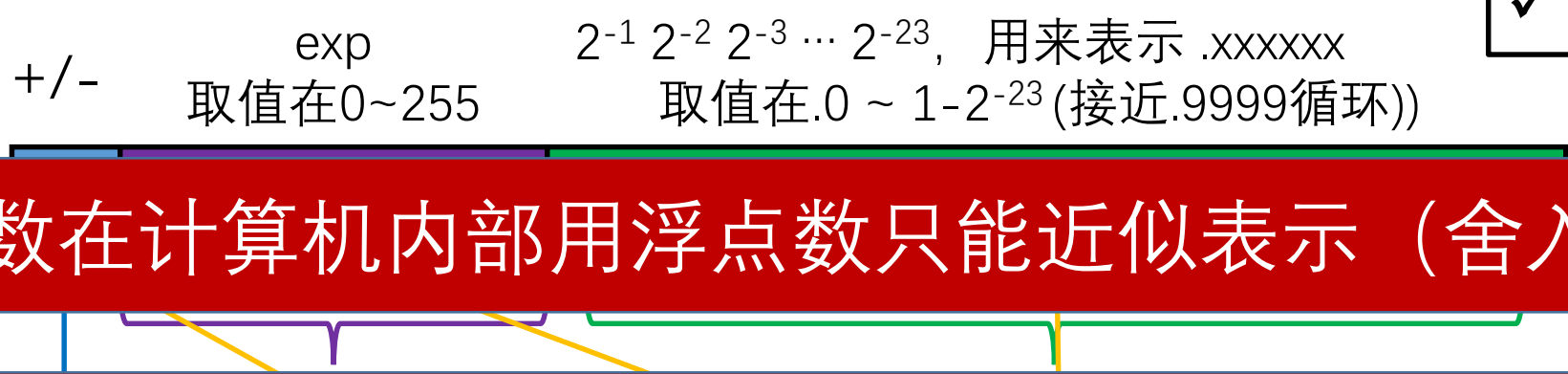
浮点运算不满足很多运算特性，如结合律、分配律等。

任何浮点数都可以表示为：(+/-) 1.xxxxxx × 2^X

浮点型在内存中的表示

浮点数9.0:
- ✓ 9.0×2^0
- ✓ 4.5×2^1
- ✓ 2.25×2^2
- ✓ 1.125×2^3

- float型占据4个字节，即32个bit



很多实数在计算机内部用浮点数只能近似表示（舍入）！！

浮点运算不满足很多运算特性，如结合律、分配律等。

慎用浮点型的变量和常量进行比较运算！！

浮点型在内存中的表示

- double型占据8个字节，即64个bit

整型与浮点型转换

- int转换成float

```
int a = 5;
float b = a;
printf(“%f”,  b);
```

整型与浮点型转换

- int转换成float

```
int a = 5;
float b = a;
printf("%f", b);
```

- 5.000000
  - ✓编译器会评估a的值，并转换成相应值的浮点表示，即1.25×2^2
  - ✓有可能会丢失精度！！！例如尝试将int型16777229转换为float再打印输出

整型与浮点型转换

- float转换成int

```
float a = 5.6;
int b = a;
printf("%d", b);   ?
```

将小数部分丢弃

整型与浮点型转换

- float转换成int

```
float a = 5.6;
int b = a;
printf("%d", b);   ?
```

将小数部分丢弃

- 如果超出int型取值范围?

```
float a= 10000000000.0;
int b = a;
printf("%d", b); ?
```

转换为最小的负数

类型转换 vs. 按照转换说明打印

- 类型转换：在内存中转换数值的存储形式
- 按照转换说明符打印：内存表示不变，打印成转换说明指定的形式

类型转换 vs. 按照转换说明打印

- 类型转换：在内存中转换数值的存储形式
- 按照转换说明符打印：内存表示不变，打印成转换说明指定的形式
- 例：

```
int a = 5;
float b = a;
printf("%f", b);
```

vs.

```
int a = 5;
printf("%f", a);
```

类型转换 vs. 按照转换说明打印

- 类型转换：在内存中转换数值的存储形式
- 按照转换说明符打印：内存表示不变，打印成转换说明指定的形式
- 例：

```
int a = 5;
float b = a;
```

vs.

```
int a = 5;
printf("%f", a);
```

注意printf()和scanf()中转换说明符的使用

整型数据类型转换

- 显式类型转换（强制类型转换）

```
float a = 12.3;
int b = (int) a;
```

✓注意a的数据类型不会被改变

整型数据类型转换

- 显式类型转换（强制类型转换）

  float a = 12.3;

  int b = (int) a;

  ✓注意a的数据类型不会被改变

- 隐式类型转换（自动类型转换）

  例如：float a = 12.3;

  int b = a;

  再如：

  float a = 12.3;

  int b = 13;

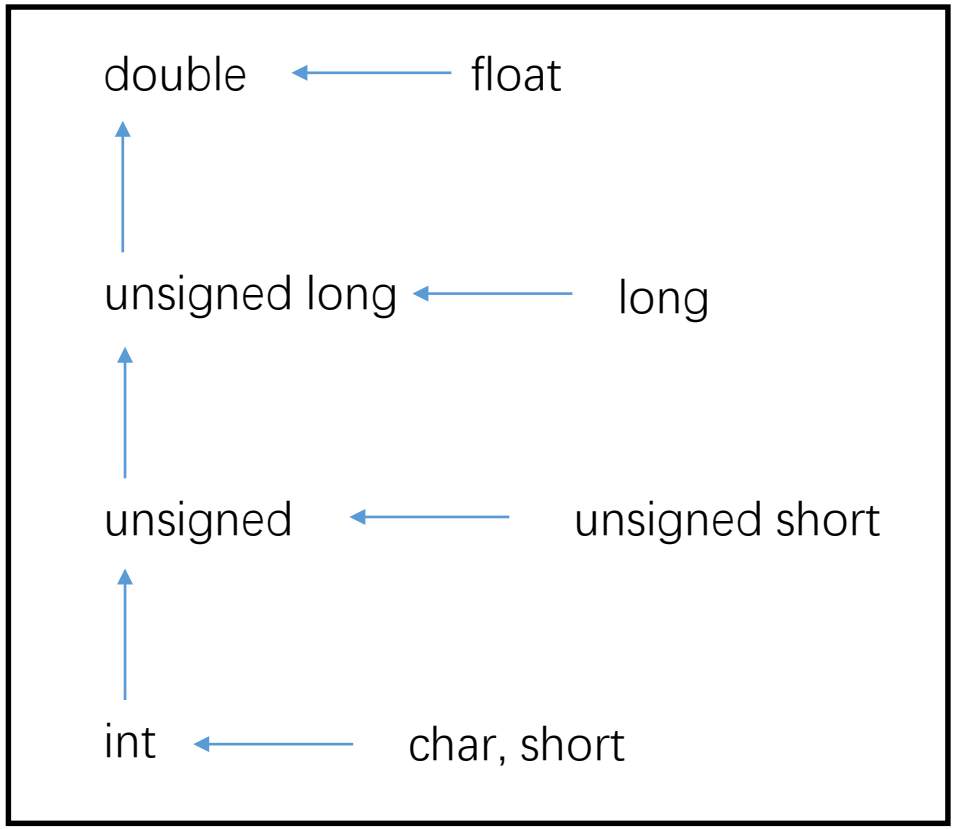
  float c = a + b;

  （将a和b隐式转成double，再相加）

整型数据类型转换

- 显式类型转换（强制类型转换）
  float a = 12.3;
  int b = (int) a;
  ✓注意a的数据类型不会被改变

- 隐式类型转换（自动类型转换）
  例如：float a = 12.3;
  int b = a;
  再如：
  float a = 12.3;
  int b = 13;
  float c = a + b;
  （将a和b隐式转成double，再相加）



非赋值隐式类型转换规则，e.g., char + long - double

C语言控制结构

三种基本控制结构

- 顺序结构
  - ✓在主函数中自上而下，依次执行
- 分支结构
  - ✓先做判断再做执行选择，根据条件语句的结果（0或非0）来选择分支
  - ✓任何有返回值的表达式或语句都可以作为条件语句（一般建议使用意义较为明确的表达式，如关系表达式和逻辑表达式）
- 循环结构
  - ✓反复执行某个功能
  - ✓根据循环条件的结果，判断继续循环还是退出循环
  - ✓循环条件中有浮点运算时要格外注意！！（一般避免使用浮点型进行循环迭代）

分支结构：if-else

- 二分支结构

```
if (表达式)
        语句1;
else
        语句2;

或者

if (表达式)
        语句1;
```

- 表达式如果成立（即值为非0），执行语句1，否则执行语句2
- 如果没有else部分，表达式不成立则什么都不执行

- 多分支结构

```
if (表达式1)
        语句1;
else if (表达式2)
        语句2;
else if (表达式3)
        语句3;
else
        语句4;
```

- 表达式1如果成立，执行语句1
- 否则，如果表达式2成立，执行语句2
- 否则，如果表达式3成立，执行语句3
- 否则，执行语句4

- 嵌套多分支结构

```
if (表达式1)
        if (表达式2)
                语句1;
        else
                语句2;
else
        if (表达式3)
                语句3;
        else
                语句4;
```

- 表达式1如果成立，进入上半部分分支结构
- 否则，则进入下半部分分支结构

二分支if-else示例

```c
#include <stdio.h>
int main()
{
    int a = 6;
    if (a < 5)
        printf("a < 5\n");
    else
        printf("a >= 5\n");

    return 0;
}
```

多分支if-else示例

```
1  #include<stdio.h>
2  int main()
3  {
4      int a = 6;
5      if(a < 5)
6          printf("a < 5\n");
7      else if(a < 10) //处理a >= 5 && a < 10的情况
8          printf("a >= 5 && a < 10\n");
9      else if(a < 15) //处理 a >= 10 && a < 15的情况
10         printf("a >= 10 && a < 15\n");
11     else              //处理a >= 15的情况
12         printf("a >= 15\n");
13
14     return 0;
15 }
```

嵌套if-else示例

```c
#include<stdio.h>
int main()
{
    int a = 6;
    if(a < 10)
    {
        if (a < 5)
            printf("a < 5\n");
        else
            printf("a >= 5 && a < 10\n");
    }
    else  //处理a>=10的情况
    {
        if (a < 15)
            printf("a >= 10 && a < 15\n");
        else
            printf("a >= 15\n");
    }
    return 0;
}
```

分支结构：switch

```
switch (条件表达式)
{
        case 常量表达式1:
                语句段1;
                break;
        case 常量表达式2:
                语句段2;
                break;
              ……
        case 常量表达式n:
                语句段n;
                break;
        default:
                语句段n+1;
                break;
}
```

- 执行与条件表达式值相等的常量表达式所对应的语句，执行完毕break跳出switch结构
- 若所有常量表达式都不等于条件表达式值，则执行default所对应的语句

- 不能出现相等的常量表达式
- default分支如果省略，则可能会出现什么都不执行的情况

- 思考：假如没有break?

switch分支结构示例

```c
#include <stdio.h>
int main()
{
    int a = 6;
    switch (a)
    {
        case 2:
            printf("a == 2\n");
            break;
        case 4:
            printf("a == 4\n");
            break;
        case 6:
            printf("a == 6\n");
            break;
        default:
            printf("*a == %d*\n", a);
            break;
    }

    return 0;
}
```

switch分支结构示例

```c
#include <stdio.h>
int main()
{
    int a = 6;
    switch (a)
    {
        case 2:
            printf("a == 2\n");
            break;
        case 4:
            printf("a == 4\n");
            break;
        case 6:
            printf("a == 6\n");
            break;
        default:
            printf("*a == %d*\n", a);
            break;
    }

    return 0;
}
```

```c
#include <stdio.h>
int main()
{
    int a = 6;
    switch (a)
    {
        case '2':
            printf("a == 2\n");
            break;
        case '4':
            printf("a == 4\n");
            break;
        case '6':
            printf("a == 6\n");
            break;
        default:
            printf("*a == %d*\n", a);
            break;
    }

    return 0;
}
```

循环结构：for循环

- for循环

```
for (表达式1; 表达式2; 表达式3)
    循环体语句
```

- 表达式1：初始化循环变量，只执行一次
- 表达式2：循环条件表达式，判断是否继续循环（循环过程中重复执行）
- 表达式3：循环变量迭代方式（循环变量步长，循环过程中重复执行）
- 循环体语句：如果有多条语句，需加大括号把所有语句包含！！

for循环示例

```c
#include<stdio.h>
int main()
{
    for(int i = 0; i < 3; i++)
    {
        printf("%d\n", i);
    }
    return 0;
}
```

一层循环

for循环示例

```c
#include<stdio.h>
int main()
{
    for(int i = 0; i < 3; i++)
    {
        printf("%d\n", i);
    }
    return 0;
}
```

一层循环

```c
#include<stdio.h>
int main()
{
    for(int i = 0; i < 3; i++)
    {
        for(int j = 0; j < 3; j++)
        {
            printf("%d, %d\n", i, j);
        }
    }
    return 0;
}
```

两层循环（嵌套循环：先执行内层循环，后执行外层外层）

循环结构：while循环

- while循环

```
while (表达式)
    循环体语句
```

- 表达式：循环条件表达式，判断是否继续循环，循环中重复执行
  - ✓在while语句之前进行表达式初始化！！(如int i = 0)
  - ✓循环迭代方式（循环变量步长）在循环体语句内编写（如i++）
- 如果循环体语句内没有编写循环迭代方式，可能造成死循环（无限循环）！！

while循环示例

```c
#include<stdio.h>
int main()
{
    int i = 0; //在循环外初始化循环变量
    while (i < 3)
    {
        printf("%d\n", i);
        i++;   //循环变量迭代
    }
    return 0;
}
```

一层循环

while循环示例

```c
#include<stdio.h>
int main()
{
    int i = 0; //在循环外初始化循环变量
    while (i < 3)
    {
        printf("%d\n", i);
        i++;   //循环变量迭代
    }
    return 0;
}
```

一层循环

```c
#include<stdio.h>
int main()
{
    int i = 0; //外层循环变量初始化
    while (i < 3)
    {
        int j = 0; //内层循环变量初始化
        while (j < 3)
        {
            printf("%d, %d\n", i, j);
            j++;   //内层循环变量迭代
        }
        i++;   //外层循环变量迭代
    }
    return 0;
}
```

两层循环（嵌套循环）

循环结构：do-while

- do-while循环

```
do
{
    循环体语句
} while (表达式);
```

- 先执行循环体，然后判断循环条件
  - ✓至少会执行一次循环体
- 其他特性与while循环类似

break和continue

- break
  - 强制提前终止break所在的最内层循环结构
  - 一般与if条件语句一起使用，即满足某条件，提前终止最内层循环

- continue
  - 跳过continue所在的此次最内层循环结构后面的语句，执行下一次循环
  - for循环中，continue不会跳过表达式3（即循环变量迭代语句）
  - while和do-while循环中？

break和continue示例

```
#include<stdio.h>
int main()
{
    for(int i = 0; i < 3; i++)
    {
        for(int j = 0; j < 3; j++)
        {
            if(i == j)
                break;
            printf("%d, %d\n", i, j);
        }
    }
    return 0;
}
```

break和continue示例

```c
#include<stdio.h>
int main()
{
    for(int i = 0; i < 3; i++)
    {
        for(int j = 0; j < 3; j++)
        {
            if(i == j)
                break;
            printf("%d, %d\n", i, j);
        }
    }
    return 0;
}
```

```c
#include<stdio.h>
int main()
{
    for(int i = 0; i < 3; i++)
    {
        for(int j = 0; j < 3; j++)
        {
            if(i == j)
                continue;
            printf("%d, %d\n", i, j);
        }
    }
    return 0;
}
```

小结

- C语言的基本数据类型
  - ✓有符号和无符号
  - ✓各种基本类型在内存中的表示
  - ✓基本数据类型转换方式

- C语言控制结构
  - ✓顺序结构
  - ✓分支结构
  - ✓循环结构

L04

- 函数