C语言程序设计基础

林川

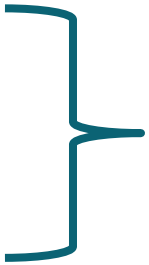
第六章 数据类型和表达式

- 数据的存储和基本数据类型
- 常量和变量
- 数据的输入和输出
- 类型转换
- 表达式

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)



后续学习

基本数据类型的存储

- 整型
- 实型
- 字符型

整型数据存储

- 整数的第一位bit用于表示整数的符号

  1 - 负数

  0 - 正数

  $-2^{15} \sim 2^{15}-1$

  1 000 0001 1000 0001

  0 000 0001 1000 0001

原码、反码、补码

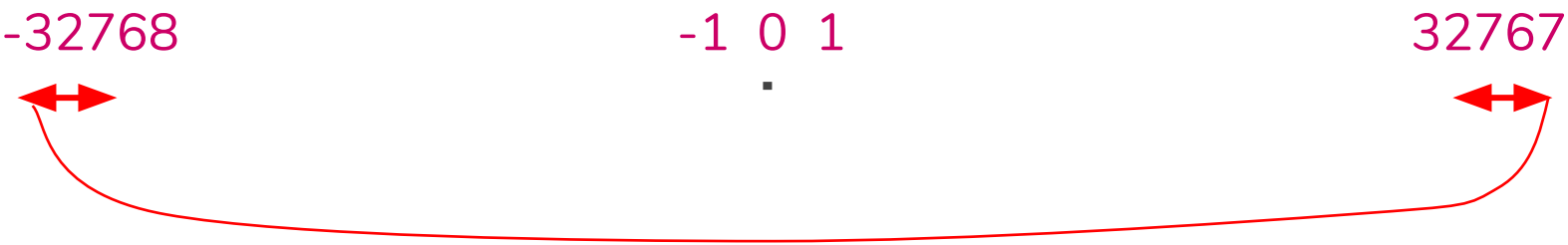
- 正数的原码、反码和补码相同
  - 1 的补码　　0 000 0000 0000 0001
  - ……
  - 32767 的补码　　0 111 1111 1111 1111

  （2^{15}-1，2个字节的存储单元能表示的最大正数）

- 负数的原码、反码和补码不同
  - -1
  - 原码　1 000 0000 0000 0001
  - 反码　1 111 1111 1111 1110　　原码取反
  - 补码　1 111 1111 1111 1111　　反码＋1

原码 反码 补码

- 32767
  - 补码　0 111 1111 1111 1111
- -32767
  - 原码　1 111 1111 1111 1111
  - 反码　1 000 0000 0000 0000　　原码取反
  - 补码　1 000 0000 0000 0001　　反码＋1
- -32768 = -32767-1
  - 补码　1 000 0000 0000 0000
  - = -2^{15}，2个字节的存储单元能表示的最小负数




```
32767    0111 1111 1111 1111
32766    0111 1111 1111 1110
32765    0111 1111 1111 1101
……
    1    0000 0000 0000 0001
    0    0000 0000 0000 0000
   -1    1111 1111 1111 1111
   -2    1111 1111 1111 1110
……
-32767   1000 0000 0000 0001
-32768   1000 0000 0000 0000
```

```
32767 + 1 = ?
 0111 1111 1111 1111
                 +1
= 1000 0000 0000 000
      (-32768)

-32768 - 1 = ???
 1000 0000 0000 0000
                 -1
= 0111 1111 1111 1111
      (32767)
```

浮点型数据存储

- 实型数据的存储

  $x = \pm m * r^e$

  m – 尾数

  r – 基数

  e – 阶码

- IEEE标准规定，常用的浮点数的格式为：

	符号位	阶码	尾数	总位数
短浮点数	1	8	23	32
浮点数	1	11	52	64
临时浮点数	1	15	64	80

字符型数据存储

- 占据一个字节
  - 存储ASCII码
  - 1 byte = 8 bits
  - 0000 0000
  - 0 ~ 255
  - -128 ~ 127

基本数据类型

- 整型

有符号整型	无符号整型	数据长度
int	unsigned [int]	32位
short [int]	unsigned short [int]	16位
long [int]	unsigned long [int]	64位

- 字符型

  char 8位

- 实型(浮点型)

  单精度浮点型 float 32位

  双精度浮点型 double 64位

基本数据类型—整型

- 扩展的整数类型：short, long, unsigned [int]

有符号整型	无符号整型	数据长度
int	unsigned [int]	16或32位
short [int]	unsigned short [int]	16位
long [int]	unsigned long [int]	32位

short（有符号）

1 000 0000 0000 0000 -32768(-2^{15})

0 111 1111 1111 1111 32767($2^{15}-1$)

unsigned short(无符号)

0000 0000 0000 0000 0

1111 1111 1111 1111 65535($2^{16}-1$)

2^0 + 2^1 + 2^2 + … + 2^15 = 2^16 - 1

整数类型的取值范围

- int 32位 $[-2^{31}, 2^{31}-1]$
- short [int] 16位 $[-2^{15}, 2^{15}-1]$
- long [int] 32位 $[-2^{31}, 2^{31}-1]$

- unsigned [int] 32位 $[0, 2^{32}-1]$
- unsigned short [int] 16位 $[0, 2^{16}-1]$
- unsigned long [int] 32位 $[0, 2^{32}-1]$

基本数据类型—字符型

- 小写字母：'a' 'b' 'c' … 'z'
- 大写字母：'A' 'B' 'C' … 'Z'
- 数字：'0' '1' '2' … '9'
- 括号、标点符号、运算符

  ( ) { } , . ' " ! # @

  + - * / % > < =

  等等

基本数据类型—字符型

- 转义字符
  - 换行符 \n
  - 制表符 \t
  - 反斜杠 \\
  - 双引号 \"
  - 单引号 \'
  - \ddd 1-3位八进制码代表的字符
  - \xhh 1-2位十六进制码代表的字符
- ASCII码表

%就是%，不是\%。但是
在scanf和printf函数的控制字符串中，
%具有特殊作用
(将其后的字符解释为格式字符)，
所以用%%表示字符%本身

基本数据类型—字符型

- 字符具有数值特征(值为ASCII码的整数)

  'A' 65 0100 0001

- 适用算术运算、关系运算

- 整型变量和字符变量的定义和赋值可以互换【 ASCII码范围】

```
char  c;
c = 'A'; 或 c = 65;
c+1 就是字符'B'
```

基本数据类型－实型

- 实型(浮点型)数据
- 单精度浮点型 float
- 双精度浮点型 double

	存储	数据精度（有效数字）	取值范围
float	4字节	7/8位	$\pm(10^{-38} - 10^{38})$
double	8字节	16位	$\pm(10^{-308} - 10^{-308})$

数据精度和取值范围

- 数据精度 与 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。
  - float y = 1.2e55;
    - y 的精度要求不高，但超出取 值范围。
- 并非所有实数都能在计算机中精确表示

实数的常量表示

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

- 科学计数法表示
  -1.2345678E5

- 实型常量的类型都是double
- 用f作为后缀，表示浮点数常量
  3.14f

数据的输入输出

printf (格式控制字符串, 输出参数1, … , 输出参数n);
scanf (格式控制字符串, 输入参数1, … , 输入参数n);

格式控制字符串
"%d%f%c"
"k = %d, x = %f, h = %c"

- 格式控制说明符 %
  - 字符char：%c
  - 实数float：%f
  - 实数doulbe：%lf
  - 整数int：　%d

整型数据的输入输出

- 扩展整数的格式控制符

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

【示例】整型数据输出格式

```
# include <stdio.h>
void main(void)
{
    printf("%d, %o, %x\n", 10, 10, 10);
    printf("%d, %d, %d\n", 10, 010, 0x10);
    printf("%d, %x\n", 012, 012);
}
```

运行结果是什么？

10, 12, a

10, 8, 16

10, a

输出格式的宽度控制

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

如果输入17 17

那么运行结果是什么？

15　17

- 宽度控制 %md 表示：数据输出的宽度为m（包括符号位）。
  - 若实际宽度不足m个，左边补充空格。
  - 若大于m，则按照实际宽度输出。

实型数据的输入和输出

- float：%f 或 %e
  - 以小数或指数形式 输入一个单精度浮点数
- double： %lf或%le
  - 以小数或指数形式 输入一个双精度浮点数
- 输出 printf()
  - float 和double使用相同的格式控制 说明
  - %f：以小数形式 输出浮点数，保留 6位小数
  - %e：以指数形式 输出

实型数据输出示例

```
double d = 3.1415926;

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

一共5位
小数3位
小数点1位

```
3.141593,3.14159e+00
3.142, 3.14,3.14
```

实型数据输入输出示例

```
/*假定float的精度为7位，double的精度为16位*/
# include <stdio.h>
int main(void)
{
    float f;
    double d;

    printf("input f, d:");
    scanf("%f%lf", &f, &d);
    printf("f = %f\n d = %f \n", f, d);
    d = 1234567890123.12;
    printf("d = %f \n", d);
    return 0;
}
```

```
input f, d:
1234567890123.123456 1234567890123.123456
f = 1234567954432.000000
d = 1234567890123.123540
d = 1234567890123.120120
```

字符型数据输入输出

- scanf() 和 printf()

  %c

```
char ch;
scanf("%c", &ch);
printf("%c", ch);
```

- getchar() 和 putchar()

```
char ch;
ch = getchar( );
putchar(ch);
```

输入输出一个字符

输入输出字符示例

```
# include <stdio.h>

int main(void)
{
    char ch1, ch2, ch3;

    scanf("%c%c%c", &ch1, &ch2, &ch3);
    printf("%c%c%c%c%c", ch1, '#', ch2, '#', ch3);

    return 0;
}
```

AbC

A#b#C

A bC

A# #b

输出字符型数据

```
/* 字符b的ASCII码98 */
# include <stdio.h>
int main(void)
{  char ch = 'b';

    printf("%c, %d\n", 'b', 'b');
    printf("%c, %d\n", 98, 98);
    printf("%c, %d\n", 97, 'b'-1);
    printf("%c, %d\n", ch - 'a' + 'A',
            ch - 'a' + 'A');
    return 0;
}
```

```
b, 98
b, 98
a, 97
B, 66
```

字符运算

- 大小写英文字母转换

  'B' - 'b' == 'A' - 'a'

  .....

  'Z' - 'z' == 'A' - 'a'

  大写字母 == 小写字母+ 'A' - 'a'

  小写字母 = =大写字母+ 'a' - 'A'

◆ 数字字符和数字转换

9 - 0 = '9' - '0'

8 - 0 = '8' - '0 '

……

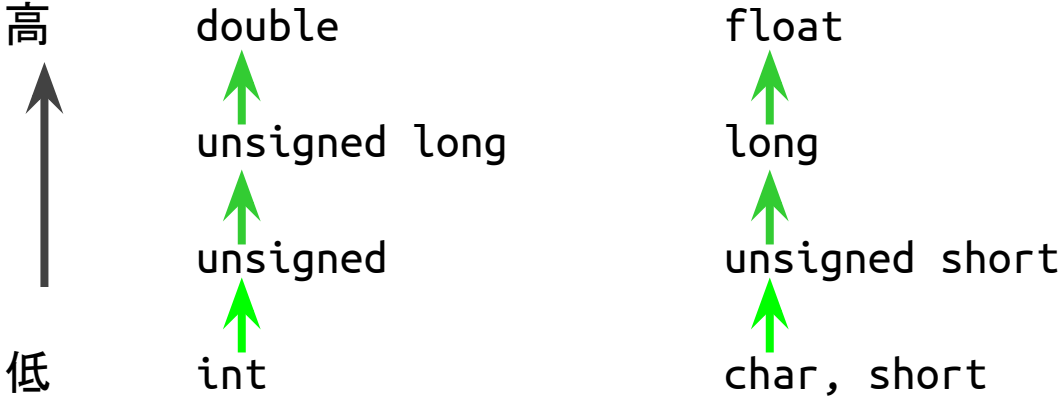
1 - 0 = '1' - '0'

数字字符	=	数字	+	'0'
数字	=	数字字符	-	'0'

4. 类型转换

- 不同类型数据的混合运算，先转换为同一类型，再运算。
- 自动类型转换
  - 非赋值运算的类型转换
  - 赋值运算的类型转换
- 强制类型转换

自动类型转换(非赋值运算)



- 水平方向:自动
- 垂直方向:低 → 高
- 短->长
- 带符号->无符号

自动类型转换(非赋值运算)

'A' + 12 – 10.05

65

77

66.95

自动类型转换(赋值运算)

变量 = 表达式

- 计算赋值运算符右侧表达式的值
- 将赋值运算符右侧表达式的值赋给左侧的变量

将赋值运算符右侧表达式的类型
自动转换
赋值号左侧变量的类型

【例子】自动转换

long a = -1;

unsigned long b, c;

b = a;　　问 b = ?

c = a+1;　　问 c = ?

c = c + a;　问 c = ?

自动类型转换(赋值运算)

```
double x;
x = 1;
```

x = 1.0

```
short a = 1000;
char b = 'A';
long c;
c = a + b;
```

c = 1065

```
int ai;
ai = 2.56;
```

ai = 2

```
short bi;
bi = 0x12345678L
```

bi = 0x5678

强制类型转换

强制类型转换运算符

(类型名) 表达式

(double)3

(int)3.8

(double)(5/2)

(double)5/2

5. 表达式

- 表达式：由运算符和运算对象（操作数）组成的有意义的运算式子，它的值和类型由参加运算的运算符和运算对象决定。
  - 运算符：具有运算功能的符号
  - 运算对象：常量、变量和函数等表达式
- 算术表达式、赋值表达式、关系表达式、逻辑表达式、条件表达式和逗号表达式等

- 表达式可以嵌套

算术表达式－算术运算符

- 单目　+　-　++　--
- 双目　+　-　*　/　%

注意

- / 整数除整数，得整数

  1/4 = 0，10/3 = 3
- % 模(求余)：针对整型数据

  5%6 = 5，9%4 = 1，100%4 = 0
- + 和 –
  - 单目运算符，+10 和 –10
  - 双目运算符，x+10 和 y –10
- 双目运算符两侧操作数的类型要相同，否则，自动类型转换后，再运算。

自增运算符++和自减运算符--

int n;

n++ ++n n-- --n（只适合变量运算）

- 使变量的值增1或减1

  ++n n++ n = n + 1

  --n n-- n = n - 1

- 取变量的值作为表达式的值

  ++n：n = n + 1；取n值作为表达式 ++n 的值

  n++：取n值作为表达式 n++ 的值；n = n + 1

自增运算和自减运算

```
int n, m;

n=2;
m=++n;
```

等价于n=n+1
m=n

结果：n为3，m 为3

```
n=2;
m=n++;
```

等价于m=n
m=n+1

结果：n为3，m 为3

算术运算符的优先级和结合性

从右向左	单目 + - ++ --	高
从左向右	双目 * / %	
从左向右	双目 + -	低

-5 + 3%2 = (-5) + (3%2) = -4

3 * 5 % 3 = (3*5) % 3 = 0

-i++ -(i++)

写出C表达式

数学式	C算术表达式
$s(s-a)(s-b)(s-c)$	`s*(s-a)*(s-b)*(s-c)`
$(x+2)e^{2x}$	`(x+2)*exp(2*x)`
$\frac{-b+\sqrt{b^2-4ac}}{2a}$	`(-b+sqrt(b*b-4*a*c))/(2*a)`

赋值表达式

- 赋值运算符 =

  x = 3*4

  优先级较低，结合性从右向左

  x = y = 3

  x = (y = 3)

赋值表达式

变量 = 表达式

- 计算赋值运算符右侧表达式的值
- 将赋值运算符右侧表达式的值赋给左侧的变量
  - **右侧表达式的类型自动转换成左侧变量的类型**
- 将赋值运算符左侧的变量的值作为表达式的值

```
int n;
double x, y;
n = 3.14 * 2;
x = 10 / 4;
x = (y = 3);
```

复合赋值运算符

- 赋值运算符
  - 简单赋值运算符 =
  - 复合赋值运算符
    - 复合算术赋值运算符 += -= *= /= %=
    - 复合位赋值运算符
- 赋值表达式

  变量 赋值运算符 表达式

  x += exp ⟹ x = x + exp

  x *= y – 3 ⟹ x = x * (y – 3)
- 从右到左结合

  y = 3;

  x = y += 5;

关系表达式－关系运算符

- 比较两个操作数，比较的结果：真 假

  x < y　x <= y　x == y

  x > y　x >= y　x != y

- 优先级
  - 算术运算符
  - < <= > >=
  - == !=
  - 赋值运算符
- 左结合

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

```
(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

- 用**关系运算符**将2个**表达式**连接起来的式子
  - 0　a > b == c
  - 0　d = a > b
  - 1　ch > 'a' + 1
  - 1　d = a + b > c
  - 0　b - 1 == a != c
  - 1　3 <= x <= 5

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

- 关系运算的结果
  - 真　1
  - 假　0

逻辑运算

运算对象为逻辑值

逻辑运算结果：1(真)　0 (假)

&& 逻辑与：a && b 为真 <=> a和b都为真

a && b 为假 <=> a和b不全为真

（至少一个为假）

|| 逻辑或：a || b 为真 <=> a和b不全为假

（至少一个为真）

! 逻辑非：! a 为真　<=> a为假

! A 为假　<=> a为真

a与b的逻辑运算

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

逻辑运算运用

- 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");
```

逻辑运算运用(续)

- 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
  - !
  - 算术运算符
  - 关系运算符
  - &&
  - ||
  - 赋值运算符
- 左结合

```
a || b && c                a || (b && c)
!a && b                    (!a) && b
x >= 3 && x <= 5           (x >= 3) && (x <= 5)
!x == 2                    (!x) == 2
a || 3 + 10 && 2           a || ((3 + 10) && 2)
```

逻辑表达式

用**逻辑运算符**将**关系表达式**或**逻辑量**连接起来的式子

哪些是逻辑表达式?

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

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

exp1 && exp2

　　先算exp1，若其值为0，

　　则 STOP

exp1 || exp2

　　先算exp1，若其值为1，

　　则STOP

写出满足要求的逻辑表达式

- x 为零
  - 关系表达式 x == 0
  - 逻辑表达式 !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

x取0　　x==0　真

x取非0　　x==0　假

x取0　　!x　真

x取非0　!x　假

条件表达式

exp1 ? exp2 : exp3

if (exp1)
　值为exp2
else
　值为exp3

```
if ( a>b )
    z = a;
else
    z = b;
```

```
z = (a>b) ? a : b;
```

$$f(x)=\begin{cases} x+2 & x>0 \\ x^2 & x\le 0 \end{cases}$$

```
if ( x>0 )
    y=x+2;
else
    y=x*x;
```

```
y = (x>0) ? x+2 : x*x;
```

逗号表达式

表达式1, 表达式2，……, 表达式n

依次计算表达式1，表达式2,……，表达式n，并将最后一个表达式的值作为逗号表达式的值.

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

逗号运算符的优先级最低，左结合

可以不用括号：a=2, b=3, c=a+b

逗号表达式的用途

```
sum = 0;
for(i = 0; i <= 100; i++)
    sum = sum + i;
```

```
for(i = 0, sum = 0; i <= 100; i++)
    sum = sum + i;
```

位(bit)运算

- 位逻辑运算
  - ~ 按位取反　　单目　右结合
  - & 按位与
  - ^ 按位异或:相同取0，不同取1
  - | 按位或
- 移位运算
  - << 对操作数左移给出的位数
  - >> 对操作数右移给出的位数
- 复合位赋值运算

位逻辑运算

~ 按位取反
& 按位与
^ 按位异或：相同取0，不同取1
| 按位或

```
char x=0      00010000
char y=3      00010011
x & y         00010000
x | y         00010011
x ^ y         00000011

1010 ^ 0101 =1111
```

注意区分：

& 和 &&

| 和 ||

位移位运算

<< 对操作数左移给出的位数

>> 对操作数右移给出的位数

x<<3 将x向左移3位，空出的位用零填补

```
00111010 << 3
11010000
```

x>>3 将x向右移3位

```
00111010 >> 3
00000111
```

复合位赋值运算符

&=

|=

^=

>>=

<<=

a &= b　相当于　a = a & b

a <<= 2　相当于　a = a << 2

其他运算

- 长度运算符 sizeof

单目运算符，计算变量或数据类型的字节长度

int a;

sizeof(a)

求整型变量 a 的长度，值为4(字节/byte）

sizeof(int)

求整型的长度，值为4

sizeof(double)

求双精度浮点型的长度，值为8

运算符的优先级和结合性

- ( )
- ! - + ++ -- (类型名）sizeof
- * / %
- + -
- < <= > >=
- == !=
- &&
- ||
- ? :
- = += -= *= /= %=
- ,

程序解析—大小写字母转换

```
while( (ch = getchar()) != '\n' )
{
   if(ch >= 'A' && ch <= 'Z')
      ch = ch + 'a' - ‘A’;
   else if((ch >= 'a' && ch <= 'z' )
      ch = ch - 'a' + 'A';
   putchar(ch);
   ch = getchar();
}
```

可以把

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

改为

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

吗？

本章内容总结

- 数据的存储和基本数据类型
- 数据的输入和输出
- 类型转换
- 表达式