printf

2022年3月15日 10:02

转换说明: ↗字母

格式: %m.pX %d, %f

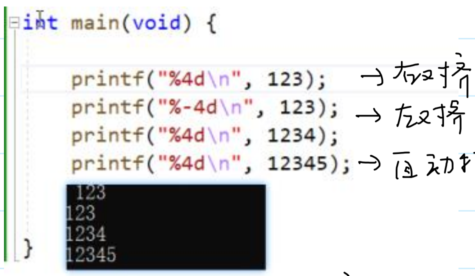
整数 整数

X: 以何种方式解释内存空间

m,p: 控制输出格式.

m: (minimum field width) 最小字段宽度 → 表明要显示的最少字符数量

%4d 123



```
int main(void) {

    printf("%4d\n", 123);
    printf("%-4d\n", 123);
    printf("%4d\n", 1234);
    printf("%4d\n", 12345);
}
```

```
 123
123
1234
12345
```

→ 右对齐

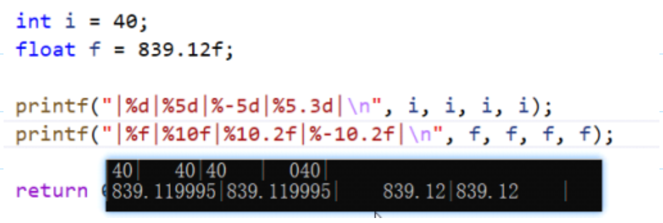
→ 左对齐

→ 自动扩展宽度

p: (presion) 精度. 精度的含义依赖转换说明符(X).

d — p代表待显示数字的最小个数(必要时会在前面补0); 如果省略p, 则默认为1.

f — p代表小数点后数字的个数(默认为6); 如果p为0, 则不显示小数点.



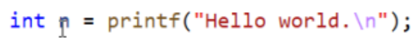
```
int i = 40;
float f = 839.12f;

printf("|%d|%5d|%-5d|%5.3d|\n", i, i, i, i);
printf("|%f|%10f|%10.2f|%-10.2f|\n", f, f, f, f);

return
```

```
40|   40|40   |  040|
839.119995|839.119995|    839.12|839.12    |
```

printf函数的返回值 → 显示字符的个数



```
int n = printf("Hello world.\n");
```

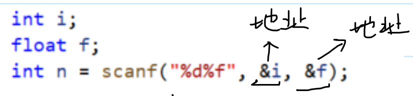
→ n=13

scanf

2022年3月15日 10:50

scanf的格式. int scanf(格式串, 表达式1, 表达式2, ...)

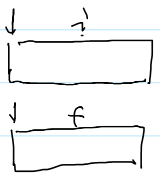
scanf本质上是一个"模式匹配函数", 试图把输入的字符与转换说明进行匹配.



```c
int i;
float f;
int n = scanf("%d%f", &i, &f);
```

从左到右依次处理格式串, 如果成功, 则继续处理后面的字符串, 如果失败, 则立即返回, 返回值表示处理转换说明成功的个数.

输入: 100 3.1415926
↓ &i ↓ &f



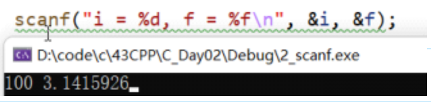
转换说明: ① 表示匹配的规则

② 表示将字符数据转换成对应的二进制数据

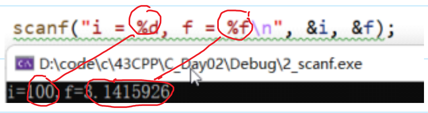
格式串: 普通字符: 其他字符 (精确匹配), 空白字符 (匹配任意多个空白字符, 包括零个)

(空格, \t, \n, \v) ⇒ 空白字符

转换说明: 和printf一致 %d, %f



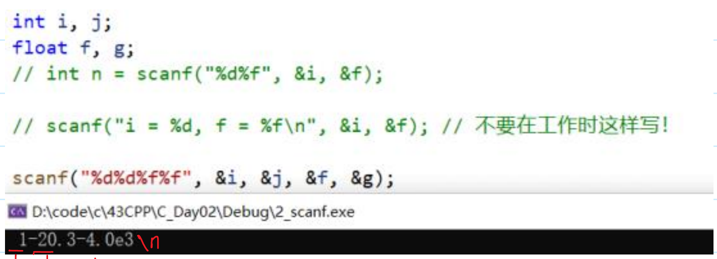
```
scanf("i = %d, f = %f\n", &i, &f);
D:\code\c\43CPP\C_Day02\Debug\2_scanf.exe
100 3.1415926_
```



```
scanf("i = %d, f = %f\n", &i, &f);
D:\code\c\43CPP\C_Day02\Debug\2_scanf.exe
i=100, f=3.1415926
```

注意事项: ① scanf匹配%d, %f (进行数值匹配时), 会匹配前面的空白字符.

%s: 字符串,
%c: 字符



```
int i, j;
float f, g;
// int n = scanf("%d%f", &i, &f);

// scanf("i = %d, f = %f\n", &i, &f); // 不要在工作时这样写!

scanf("%d%d%f%f", &i, &j, &f, &g);
D:\code\c\43CPP\C_Day02\Debug\2_scanf.exe
1-20.3-4.0e3\n
```

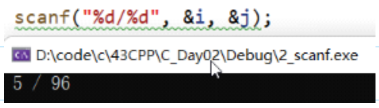
%d ← → %d

i = 1

j = -20

f = 0.3

g = -4.0e3



```
scanf("%d/%d", &i, &j);
D:\code\c\43CPP\C_Day02\Debug\2_scanf.exe
5 / 96
```

基本数据类型

2022年3月15日 11:40

基本数据类型
- 整数
- 浮点数
- 字符类型（C语言把字符类型当作整数类型）

整数类型

2022年3月15日 14:32

整数类型: 有符号整和无符号整数

short (int)
unsigned short (int)
int
unsigned (int)
long (int)
unsigned long (int)
long long (int)
unsigned long long (int)

注意事项: ① C语言没有明确规定各种整数类型的具体大小, 可能随着机器而不同.
② C语言规定各种类型的最小大小. (int类型至少占两个字节)
③ short ≤ int ≤ long ≤ long long

#1. 编码 (☆☆☆)

无符号整数 (以一个字节为例) $\underline{1001}\ \underline{0011}$ $2^7+0+0+2^4+0+0+2^1+2^0=147$

7654 3210 ←

有符号整数 (补码)   负的权重 ← $\boxed{1}001\ 0011$ $-2^7+0+0+2^4+0+0+2^1+2^0=-109$

7654 3210 -128 16   2   1

Q: 为什么有符号整数会采用补码形式呢?

加法器 (adder) ⇒ 利用加法器作减法运算 $a-b=a+(-b)$

$$\begin{array}{rrl} & 127 & 0111\ 1111 \\ - & 109 & +\ 1001\ 0011 \\ \hline & 18 & \boxed{1}0001\ 0010\ (18) \end{array}$$

$+2^7+x=\boxed{-a+2^8}$

Q: 为什么采用补码形式, 就能利用加法器作减法器.

$a+(-a)=0$ 补码: $a+(-a+2^8)=2^8$ $(N=2^8)$

数学上 ↑   ↑ 在某种程度上是等价的 $0\equiv 2^8 \pmod{2^8}$

模运算: $x=qN+r$ $(0\le r<N)$, $\underline{x \boxed{\bmod} N = r}$

同余: $x \bmod N = y \bmod N \iff x\equiv y \pmod N \iff N$ divides $(x-y)$

↓

等价类: ...-9 -6 -3 0 3 6 9 ...   (N=3)
...-8 -5 -2 1 4 7 10 ...
...-7 -4 -1 2 5 8 11

定理1: $x \equiv x' \pmod N$ 并且 $y \equiv y' \pmod N$, 那么 (替换原则) (Group Theory)

$x+y \equiv x'+y' \pmod N$ 且 $xy \equiv x'y' \pmod N$

定理2: $(x+y)+z \equiv x+(y+z) \pmod N$ (结合律原则)

$xy \equiv yx \pmod N$ (交换原则)

$x(y+z) \equiv xy+xz \pmod N$ (分配原则)

#2. 64位机器上整数类型的常见取值范围

	字节	最小值	最大值
short	2	$-2^{15}=32768$	32767
unsigned short	2	0	65535
int	4	-2147483648	2147483647
unsigned	4	0	4294967295 (约43亿)
long	8	(≥4字节)	
unsigned long	8		
long long	8	(≥8字节)	
unsigned long long	8		

整数常量

三种表示方式: 十进制: 1234, 9527 (不能以0开头)

八进制: 以0开头, 01234, 0527

十六进制: 以0x开头, 0xABC, 0xCBA, 0xA72

整数常量的类型:

十进制: int → long → long long → Error!

八进制、十六进制: int → unsigned → long → unsigned long → long long → unsigned long long → Error!

可以在整数常量后面添加后缀指定整数常量的类型

U(u) (unsigned): 43U

L(l) (long): 43L

LL(ll) (long long): 43LL

混合使用: 43LLU, 43ULL

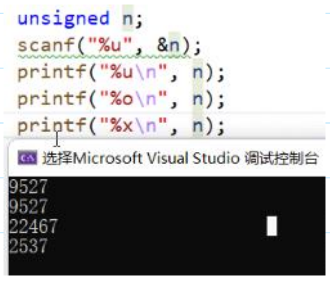
#4. 读写整数

%u: 无符号十进制整数

%o: 无符号八进制整数

%x: 无符号十六进制整数

%d: 有符号十进制整数

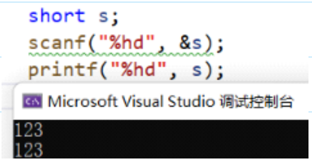


```
unsigned n;
scanf("%u", &n);
printf("%u\n", n);
printf("%o\n", n);
printf("%x\n", n);
```

选择Microsoft Visual Studio 调试控制台

```
9527
9527
22467
2537
```

读写短整数，在 u,o,x,d 前面添加 h. (short)



```
short s;
scanf("%hd", &s);
printf("%hd", s);
```

Microsoft Visual Studio 调试控制台

```
123
123
```

读写长整数，在 u,o,x,d 前面添加 l.

读写长长整整，在 u,o,x,d 前面添加 ll

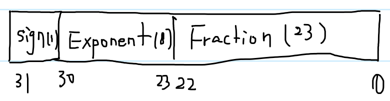
浮点数类型

2022年3月15日 15:55

浮点数 { float (4字节) / double (8字节) / long double (用于高精度计算中，一般用不到)

#1. 编码 (IEEE 754 标准) (☆☆☆)

float



sign: 1表示负，0表示正 $(-1)^S$

Exponent: 2^{E-127} (移码)，$-127\sim128$，表示指数的范围：$-126\sim127$. 其中 -127 (全为0) 和 128 (全为1) 有特殊用途.

$\quad\quad \hookrightarrow 2^7-1$

Fraction: 小数部分

三个特殊值.

Exponent: 0000 0000，Fraction: 00…0，表示 ±0

Exponent: 1111 1111，Fraction: 00…0，表示 ±∞

Exponent: 1111 1111，Fraction: 不全为0，表示 NaN (Not a Number).

规约数：

指数范围：E: [00000001, 11111110]

$(-1)^S \times 1.F \times 2^{E-127}$

非规约数：

Exponent: 0000 0000 Fraction: 不全为0

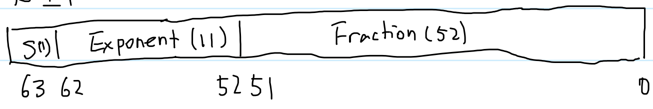
表示十分接近于零的浮点数，非规约数的绝对值都小于规约数

$(-1)^S \times 0.F \times 2^{-126}$ → 不是 −127 !!!

最大的数： $\underbrace{0}_{1}\ \underbrace{11111110}_{8}\ \underbrace{1111\cdots1}_{23}$ $\quad (-1)^0 \times 1.\underbrace{111\ldots1}_{23} \times 2^{127} \approx 2^{128} \approx 3.4\times10^{38}$

最接近于0的正数： 0 00000000 0000000…01 $\quad (-1)^0 \times 0.\underbrace{0\ldots0}_{22}1 \times 2^{-126} = 2^{-23}\times2^{-126} = 2^{-149} \approx 1.4\times10^{-45}$

double 类型：



S: 0表示正，1表示负

Exponent: E − 1023，$-1023\sim1024$，指数范围：$-1022\sim1023$ 其中 −1023 (全为0) 和 1024 (全为1) 有特殊用途.

Fraction: 小数

作业：计算 double 类型能表示的最大正值和正小值.

#2. 浮点常量.

浮点数常量有多种表示方法，要么包含小数点，要么包含字母E(e)

57.0　　57.　　5.70e1，　.57e2　　570e-1

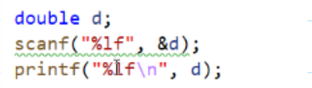
以10为底

浮点数常量默认是double，如果需要表示float，应该在浮点数常量后面添加字母F(f)

读写浮点数

%f：float

%lf：double　(注意l不能大写)

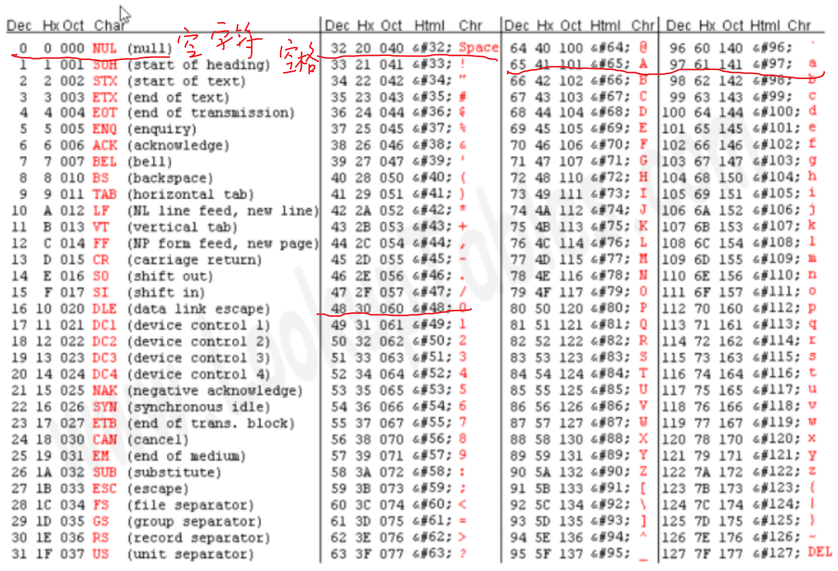


```
double d;
scanf("%lf", &d);
printf("%lf\n", d);
```

字符数据类型

2022年3月15日　　17:02

char类型大小为1个字节，并且采用ASCII编码表示，ASCII编码用7位表示128个字符（最高位都为0）



Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null) 空字符	32	20	040	&#32;	Space 空格	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1 (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2 (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3 (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4 (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

'\0': 空字符　　' ' = 32　　'0' = 48　　'A' = 65　　'a' = 97

#1　C语言把字符类型当作小的整数类型来使用，因此可以对字符执行算术运算和比较运算.

```c
int i = 'a';
char ch = 'A';
ch = ch + 1;
ch++;

if (ch >= 'A' && ch <= 'Z') {
    ch = ch + 'a' - 'A';
}
```

#2　不能直接输入字符 ⟶ 转义序列

字符转义序列:

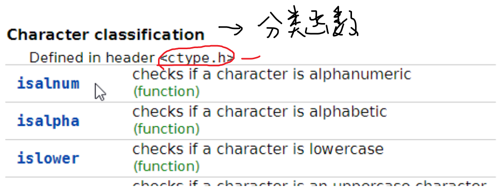
\a (alert, bell)	\v (vertical tab)
\b (backspace)	\\ (back slash)
\f (form feed)	\? (question mark)
\n (newline)	\' (single quote)
\r (carriage return)	\" (double quote)
\t (horizontal tab)	

数字转义序列

八进制表示形式：以\开头，后面接最多3个八进制数字　　\0, \101 'A'

十六进制表示形式：以\x开头，后面接十六进制数字。　　\x0, \x41

#3 字符处理函数.



Character classification ⟶ 分类函数

Defined in header <ctype.h>

isalnum	checks if a character is alphanumeric (function)
isalpha	checks if a character is alphabetic (function)
islower	checks if a character is lowercase (function)
	checks if a character is an uppercase character

Character classification → 分类函数

Defined in header <ctype.h>

isalnum	checks if a character is alphanumeric (function)
isalpha	checks if a character is alphabetic (function)
islower	checks if a character is lowercase (function)
isupper	checks if a character is an uppercase character (function)
isdigit	checks if a character is a digit (function)
isxdigit	checks if a character is a hexadecimal character (function)
iscntrl	checks if a character is a control character (function)
isgraph	checks if a character is a graphical character (function)
isspace	checks if a character is a space character (function)
isblank (C99)	checks if a character is a blank character (function)
isprint	checks if a character is a printing character (function)
ispunct	checks if a character is a punctuation character (function)

Character manipulation → 操作

tolower	converts a character to lowercase (function)
toupper	converts a character to uppercase (function)

ASCII values			characters	iscntrl ~~iswcntrl~~	isprint ~~iswprint~~	isspace ~~iswspace~~	isblank ~~iswblank~~	isgraph ~~iswgraph~~	ispunct ~~iswpunct~~	isalnum ~~iswalnum~~	isalpha ~~iswalpha~~	isupper ~~iswupper~~	islower ~~iswlower~~	isdigit ~~iswdigit~~	isxdigit ~~iswxdigit~~
decimal	hexadecimal	octal													
0–8	\x0–\x8	\0–\10	control codes (NUL, etc.)	≠0	0	0	0	0	0	0	0	0	0	0	0
9	\x9	\11	tab (\t)	≠0	0	≠0	≠0	0	0	0	0	0	0	0	0
10–13	\xA–\xD	\12–\15	whitespaces (\n, \v, \f, \r)	≠0	0	≠0	0	0	0	0	0	0	0	0	0
14–31	\xE–\x1F	\16–\37	control codes	≠0	0	0	0	0	0	0	0	0	0	0	0
32	\x20	\40	space	0	≠0	≠0	≠0	0	0	0	0	0	0	0	0
33–47	\x21–\x2F	\41–\57	!"#$%&'()*+,-./	0	≠0	0	0	≠0	≠0	0	0	0	0	0	0
48–57	\x30–\x39	\60–\71	0123456789	0	≠0	0	0	≠0	0	≠0	0	0	0	≠0	≠0
58–64	\x3A–\x40	\72–\100	:;<=>?@	0	≠0	0	0	≠0	≠0	0	0	0	0	0	0
65–70	\x41–\x46	\101–\106	ABCDEF	0	≠0	0	0	≠0	0	≠0	≠0	≠0	0	0	≠0
71–90	\x47–\x5A	\107–\132	GHIJKLMNOP QRSTUVWXYZ	0	≠0	0	0	≠0	0	≠0	≠0	≠0	0	0	0
91–96	\x5B–\x60	\133–\140	[\]^_`	0	≠0	0	0	≠0	≠0	0	0	0	0	0	0
97–102	\x61–\x66	\141–\146	abcdef	0	≠0	0	0	≠0	0	≠0	≠0	0	≠0	0	≠0
103–122	\x67–\x7A	\147–\172	ghijklmnop qrstuvwxyz	0	≠0	0	0	≠0	0	≠0	≠0	0	≠0	0	0
123–126	\x7B–\x7E	\172–\176	{|}~	0	≠0	0	0	≠0	≠0	0	0	0	0	0	0
127	\x7F	\177	backspace character (DEL)	≠0	0	0	0	0	0	0	0	0	0	0	0

#4 读写字符.

1) scanf / printf 配合 %c 来读写字符.

注意事项: %c 不会忽略前面的空白字符.

Q: 如何忽略字符前面的空白字符?

" %c"

2) getchar / putchar

getchar

Defined in header <stdio.h>

```
int getchar(void);
```

Reads the next character from stdin.

getchar() 和 putchar(ch) 的效率是比 scanf / printf.

如果只是读写字符数据, 建议使用 getchar() 和 putchar().

putchar

Defined in header <stdio.h>

```
int putchar( int ch );
```

Writes a character ch to stdout.

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

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

惯用法（成语）:

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

$\longrightarrow$ 读取这行剩余的字符.

布尔类型

2022年3月15日 17:49

C99中定义了布尔类型，在<stdbool.h>头文件中.

```c
bool flag1, flag2;
flag1 = true;
flag2 = false;
```

bool类型，本质上是无符号整数类型

true = 1

false = 0

```c
printf("%u", true);
printf("%u", false);
```

→宏　#define true 1

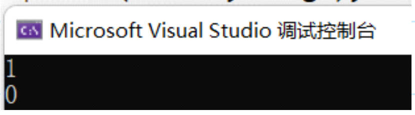
→宏　#define false 0

注意事项：1给布尔类型变量赋值，非零会得到true，零会得到false.

```c
bool flag1, flag2;
flag1 = 5;
flag2 = 0;

printf("%u\n", flag1);
printf("%u\n", flag2);
```

→任何非零的都会转换为1



```
1
0
```