文件

2020年7月25日 19:03

格式化的输入输出

%[flags][width][.prec][hlL]type

- printf
  - %[flags][width][.prec][hlL]type
- scanf
  - %[flag]type

Flag	含义
-	左对齐
+	在前面放 + 或 -
(space)	正数留空
0	0填充

```c
#include <stdio.h>

int main(int argc, char const *argv[])
{
    printf("%9d\n", 123);
    printf("%-9d\n", 123);

    return 0;
}
```

```
    123
123
[Finished in 0.8s]
```

```c
#include <stdio.h>

int main(int argc, char const *argv[])
{
    printf("%+9d\n", 123);
    printf("%-9d\n", -123);

    return 0;
}
```

```
    +123
-123
[Finished in 0.1s]
```

```c
#include <stdio.h>

int main(int argc, char const *argv[])
{
    printf("%09d\n", 123);
    printf("%-9d\n", -123);

    return 0;
}
```

第二个前面不可以0

```
000000123
-123
[Finished in 0.1s]
```

```c
#include <stdio.h>

int main(int argc, char const *argv[])
{
    printf("%9.2f\n", 123.0);

    return 0;
}
```

一共9位，小数点2位

```
   123.00
[Finished in 0.1s]
```

%[flags][width][.prec][hlL]type

width 或 prec	含义
number	最小字符数
*	下一个参数是字符数
.*number*	小数点后的位数
.*	下一个参数是小数点后的位数

```c
#include <stdio.h>

int main(int argc, char const *argv[])
{
    printf("%*d\n",6, 123);
    printf("%9.2f\n", 123.0);

    return 0;
}
```

6满足那个*号，占据6个字符的位置

```
   123
   123.00
[Finished in 0.1s]
```

%[flags][width][.prec][hlL]type

类型修饰	含义
hh	单个字节
h	short
l	long
ll	long long
L	long double

```c
#include <stdio.h>

int main(int argc, char const *argv[])
{
    printf("%hhd\n",12345);
    printf("%9.2f\n", 123.0);

    return 0;
}
```

12345十六进制3039

取39变成十进制——57

```
        printf("%hhd\n",12345);
                ~~~~    ^~~~~
                %d
1 warning generated.
57
  123.00
```

```c
#include <stdio.h>

int main(int argc, char const *argv[])
{
    printf("%hhd\n",(char)12345);
    printf("%9.2f\n", 123.0);

    return 0;
}
```

```
57
  123.00
[Finished in 0.1s]
```

%[flags][width][.prec][hlL]type

type	用于	type	用于
i 或 d	int	g	float
u	unsigned int	G	float
o	八进制	a 或 A	十六进制浮点
x	十六进制	c	char
X	字母大写的十六进制	s	字符串
f 或 F	float, 6	p	指针
e 或 E	指数	n	读入/写出的个数

```c
#include <stdio.h>

int main(int argc, char const *argv[])
{
    int num;
    printf("%hhd%n\n",(char)12345, &num);
    printf("%d\n", num);

    return 0;
}
```

```
57
2
[Finished in 0.1s]
```

```c
#include <stdio.h>

int main(int argc, char const *argv[])
{
    int num;
    printf("%d%n\n",12345, &num);
    printf("%d\n", num);

    return 0;
}
```

```
12345
5
[Finished in 0.1s]
```

```c
#include <stdio.h>

int main(int argc, char const *argv[])
{
    int num;
    printf("%dty%n\n",12345, &num);
    printf("%d\n", num);

    return 0;
}
```

n意思是printf做到这个地方时，输出了多少字符，填到指针所指的变量中去

```
12345ty
7
[Finished in 0.1s]
```

scanf：%[flag]type

flag	含义	flag	含义
*	跳过	l	long, double
数字	最大字符数	ll	long long
hh	char	L	long double
h	short		

```c
int main(int argc, char const *argv[])
{
    int num;
    scanf("%*d%d", &num);
    printf("%d\n", num);

    return 0;
}
```

```
MOOC:cc\ $./test
123 345
345
```

i:八进制输入——八进制输出

scanf：%[flag]type

type	用于	type	用于
d	int	s	字符串（单词）
i	整数，可能为十六进制或八进制	[...]	所允许的字符
u	unsigned int	p	指针
o	八进制		
x	十六进制		
a,e,f,g	float		
c	char		

printf和scanf的返回值

- 读入的项目数
- 输出的字符数

- 在要求严格的程序中，应该判断每次调用scanf或printf的返回值，从而了解程序运行中是否存在问题

文件输入输出

- 用>和<做重定向

```
MOOC:cc\ $./test
1234
1234
1:5
MOOC:cc\ $./test
12345
12345
1:6
MOOC:cc\ $./test > 12.out
12345
MOOC:cc\ $more 12.out
12345
1:6
MOOC:cc\ $cat > 12.in
12345
MOOC:cc\ $more 12.in
12345
MOOC:cc\ $./test < 12.in
12345
1:6
MOOC:cc\ $./test < 12.in > 12.out
MOOC:cc\ $more 12.out
12345
1:6
MOOC:cc\ $<
```

输入结束

- getchar读到了EOF
- scanf返回小于要求读的数量

FILE

- FILE* fopen(const char * restrict path, const char * restrict mode);
- int fclose(FILE *stream);
- fscanf(FILE*, ...)
- fprintf(FILE*, ...)

打开文件的标准代码

```
FILE* fp = fopen("file","r");
if ( fp ) {
    fscanf(fp,...);
    fclose(fp);
} else {
    ...
}
```

```
int main(int argc, char const *argv[])
{
    FILE *fp = fopen("12.in", "r");
    if ( fp ) {
        int num;
        fscanf(fp, "%d", &num);
        printf("%d\n", num);
        fclose(fp);
    } else {
        printf("无法打开文件\n");
    }
    return 0;
}
```

```
MOOC:cc\ $./test
12345
MOOC:cc\ $rm 12.in
MOOC:cc\ $./test
无法打开文件
```

fopen

r	打开只读
r+	打开读写，从文件头开始
w	打开只写。如果不存在则新建，如果存在则清空
w+	打开读写。如果不存在则新建，如果存在则清空
a	打开追加。如果不存在则新建，如果存在则从文件尾开始
..x	只新建，如果文件已存在则不能打开

二进制文件

- 其实所有的文件最终都是二进制的
- 文本文件无非是用最简单的方式可以读写的文件
  - more、tail
  - cat
  - vi
- 而二进制文件是需要专门的程序来读写的文件
- 文本文件的输入输出是格式化，可能经过转码

文本 vs 二进制

- Unix喜欢用文本文件来做数据存储和程序配置
  - 交互式终端的出现使得人们喜欢用文本和计算机“talk”
  - Unix的shell提供了一些读写文本的小程序
- Windows喜欢用二进制文件
  - DOS是草根文化，并不继承和熟悉Unix文化
  - PC刚开始的时候能力有限，DOS的能力更有限，二进制更接近底层

文本 vs 二进制

- 文本的优势是方便人类读写，而且跨平台
- 文本的缺点是程序输入输出要经过格式化，开销大
- 二进制的缺点是人类读写困难，而且不跨平台
  - int的大小不一致，大小端的问题...
- 二进制的优点是程序读写快

程序为什么要文件

- 配置
  - Unix用文本，Windows用注册表
- 数据
  - 稍微有点量的数据都放数据库了
- 媒体
  - 这个只能是二进制的
- 现实是，程序通过第三方库来读写文件，很少直接读写二进制文件了

二进制读写

- size_t fread(void *restrict ptr, size_t size, size_t nitems, FILE *restrict stream);
- size_t fwrite(const void *restrict ptr, size_t size, size_t nitems, FILE *restrict stream);
- 注意FILE指针是最后一个参数
- 返回的是成功读写的字节数

为什么nitem?

- 因为二进制文件的读写一般都是通过对一个结构变量的操作来进行的
- 于是nitem就是用来说明这次读写几个结构变量！

在文件中定位

- long ftell(FILE *stream);
- int fseek(FILE *stream, long offset, int whence);
  - SEEK_SET：从头开始
  - SEEK_CUR：从当前位置开始
  - SEEK_END：从尾开始（倒过来）

可移植性

- 这样的二进制文件不具有可移植性
  - 在int为32位的机器上写成的数据文件无法直接在int为64位的机器上正确读出
- 解决方案之一是放弃使用int，而是typedef具有明确大小的类型
- 更好的方案是用文本

按位运算

·C有这些按位运算的运算符：

·& 按位的与
·| 按位的或
·~ 按位取反
·^ 按位的异或
·<< 左移
·>> 右移

按位与 &

·如果 $(x)_i == 1$ 并且 $(y)_i == 1$，那么 $(x \& y)_i = 1$

·否则的话 $(x \& y)_i = 0$

·按位与常用于两种应用：

·让某一位或某些位为0：`x & 0xFE`

·取一个数中的一段：`x & 0xFF`

按位或 |

·如果 $(x)_i == 1$ 或 $(y)_i == 1$，那么 $(x \mid y)_i = 1$

·否则的话， $(x \mid y)_i == 0$

·按位或常用于两种应用：

·使得一位或几个位为1：x | 0x01

·把两个数拼起来：0x00FF | 0xFF00

按位取反 ~

· $(\sim x)_i = 1 - (x)_i$

·把1位变0，0位变1

·想得到全部位为1的数：~0

·7的二进制是0111，x | 7使得低3位为1，而

·x & ~7，就使得低3位为0

逻辑运算vs按位运算

- 对于逻辑运算，它只看到两个值：0和1
- 可以认为逻辑运算相当于把所有非0值都变成1，然后做按位运算
  - 5 & 4 —>4而 5 && 4 —> 1 & 1 —> 1
  - 5 | 4 —> 5而 5 || 4 —> 1 | 1 —> 1
  - ~4 —> 3而 !4 —> !1 —> 0

按位异或^

·如果 $(x)_i == (y)_i$，那么 $(x \wedge y)_i = 0$

·否则的话，$(x \wedge y)_i == 1$

·如果两个位相等，那么结果为0；不相等，结果为1

·如果x和y相等，那么x ^ y的结果为0

·对一个变量用同一个值异或两次，等于什么也没做

·x ^ y ^ y —> x

左移 <<

·`i << j`

·i中所有的位向左移动j个位置，而右边填入0

·所有小于int的类型，移位以int的方式来做，结果是int

·`x <<= 1` 等价于 `x *= 2`

·`x <<= n` 等价于 `x *= 2^n`.

右移 >>

- `i >> j`
- i中所有的位向右移j位
- 所有小于int的类型，移位以int的方式来做，结果是int
- 对于unsigned的类型，左边填入0
- 对于signed的类型，左边填入原来的最高位（保持符号不变）
- `x >>= 1` 等价于 `x /= 2`
- `x >>= n` 等价于 `x /= 2^n`.

no zuo no die

- 移位的位数不要用负数，这是没有定义的行为

```
x << -2   //!!NO!!
```