第五章 标准IO

查看gcc所有宏定义：gcc -E -dM - </dev/null

随处可见的“格式化输出转换表”并不完全，如下附上几个完整的参考表：
info libc
12.12.3
http://www.gnu.org/software/libc/manual/html_node/Table-of-Output-Conversions.html
http://www.gnu.org/software/libc/manual/html_node/Integer-Conversions.html#Integer-Conversions
中文解释截图

表5-5 转换说明中的标志部分

标 志	说 明
-	在字段内左对齐输出
+	总是显示带符号转换的符号
(空格)	如果第一个字符不是符号，则在其前面加上一个空格
#	指定另一种转换形式（例如，对于十六进制格式，加0x前缀）
0	添加前导0（而非空格）进行填充

表5-6 转换说明中的长度修饰符

长度修饰符	说　明
hh	有符号或无符号的char
h	有符号或无符号的short
l	有符号或无符号的long或者宽字符
ll	有符号或无符号的long long
j	intmax_t或uintmax_t
z	size_t
t	ptrdiff_t
L	long double

表5-7 转换说明中的转换类型部分

转换类型	说　明
d、i	有符号十进制
o	无符号八进制
u	无符号十进制
x、X	无符号十六进制
f、F	double精度浮点数
e、E	指数格式的double精度浮点数
g、G	解释为f、F、e或E，取决于被转换的值
a、A	十六进制指数格式的double精度浮点数
c	字符（若带长度修饰符l，则为宽字符）
s	字符串（若带长度修饰符l，则为宽字符）
p	指向void的指针
n	将到目前为止，所写的字符数写入到指针所指向的无符号整型中
%	%字符
C	宽字符（XSI扩展，等效于lc）
S	宽字符串（XSI扩展，等效于ls）

```
struct _IO_FILE {    FILE结构体
  int _flags;       /* High-order word is _IO_MAGIC; rest is flags. */
#define _IO_file_flags _flags

  /* The following pointers correspond to the C++ streambuf protocol. */
  /* Note:  Tk uses the _IO_read_ptr and _IO_read_end fields directly. */
  char* _IO_read_ptr;   /* Current read pointer */
  char* _IO_read_end;   /* End of get area. */
  char* _IO_read_base;  /* Start of putback+get area. */
  char* _IO_write_base; /* Start of put area. */
  char* _IO_write_ptr;  /* Current put pointer. */
  char* _IO_write_end;  /* End of put area. */
  char* _IO_buf_base;   /* Start of reserve area. */
  char* _IO_buf_end;    /* End of reserve area. */
  /* The following fields are used to support backing up and undo. */
  char *_IO_save_base; /* Pointer to start of non-current get area. */
  char *_IO_backup_base;  /* Pointer to first valid character of backup area */
  char *_IO_save_end; /* Pointer to end of non-current get area. */
/usr/include/libio.h [FORMAT=unix:utf-8] [TYPE=CPP] [COL=001] [ROW=278/557(49%)]
```

```
#include <stdio.h>      // For fopen, fclose;
#include <string.h>     // For strlen;
int main(int argc, const char *argv[]) {
    FILE *fp = fopen((char *) argv[1], "a+");
    char buf[4096];
    if(strlen(argv[2]) != fwrite(argv[2], sizeof(char), strlen(argv[2]), fp)) {
        printf("fwrite error!");
    }
    printf("Write_base=%lx, Write_end=%lx, Write_buf_size=%ld\n", (unsigned long int) fp->_IO_write_base
, (unsigned long int) fp->_IO_write_end, (long int) fp->_IO_write_end - (long int) fp->_IO_write_base);
    fseek(fp, 0, SEEK_SET);
    if(1 != fread(buf, 1, 1, fp)) {
        printf("fread error!");
    }
    printf("%s\n", buf);
    printf("Read_base=%lx, Read_end=%lx, Read_buf_size=%ld\n", (unsigned long int) fp->_IO_read_base, (u
nsigned long int) fp->_IO_read_end, (long int) fp->_IO_read_end - (long int) fp->_IO_read_base);
    printf("Buf_base =%lx, Buf_end =%lx, Buf_size=%ld\n", (unsigned long int) fp->_IO_buf_base, (unsigne
d long int) fp->_IO_buf_end, (long int) fp->_IO_buf_end - (long int) fp->_IO_buf_base);
    fclose(fp);
    return 0;
}
/opt/cpp/std_IO.cpp [FORMAT=unix:utf-8] [TYPE=CPP] [COL=001] [ROW=001/19(5%)]
```

```
mail cpp # g++ std_IO.cpp -o std_IO
mail cpp # ./std_IO a_file Hello
Write_base=7f2d09a18000, Write_end=7f2d09a19000, Write_buf_size=4096
H
Read_base=7f2d09a18000, Read_end=7f2d09a18005, Read_buf_size=5
Buf_base =7f2d09a18000, Buf_end =7f2d09a19000, Buf_size=4096
```

```
mail cpp # cp /bin/bash .
mail cpp # ls -l bash
-rwxr-xr-x 1 root root 737808 12月 22 01:40 bash
mail cpp # ./std_IO bash Hello
Write_base=7f0de10d2000, Write_end=7f0de10d3000, Write_buf_size=4096
Read_base=7f0de10d2000, Read_end=7f0de10d3000, Read_buf_size=4096
Buf_base =7f0de10d2000, Buf_end =7f0de10d3000, Buf_size=4096
```

大于磁盘中一个块的大小

习题：

5-1、该题目的应该是让读者加深对setbuf和setvbuf的记忆；

```
mail ~ # grep -r _G_BUFSIZ /usr/include/*
/usr/include/_G_config.h:#define _G_BUFSIZ 8192
/usr/include/libio.h:#define _IO_BUFSIZ _G_BUFSIZ
```

```c
#include <stdio.h>      // For fopen, fclose;
#include <string.h>     // For strlen;
#include <unistd.h>
#include <sys/stat.h>

void v_setbuf(FILE *restrict fp, char *restrict buf) {
    struct stat sb;
    fstat(fp->_fileno, &sb);
    if (NULL == buf) {
        setvbuf(fp, NULL, _IONBF, 0);
    } else if (S_ISCHR(sb.st_mode)) {
        setvbuf(fp, buf, _IOLBF, BUFSIZ);
    } else {
        setvbuf(fp, buf, _IOFBF, BUFSIZ);
    }
}

int main(int argc, const char *argv[]) {
    char buf[BUFSIZ];
    FILE *fp = fopen((char *) argv[1], "a+");
    v_setbuf(fp, buf);
    if(strlen(argv[2]) != fwrite(argv[2], sizeof(char), strlen(argv[2]), fp)) {
        printf("fwrite error!");
    }
    sleep(60);
    fclose(fp);
    return 0;
}
~
/opt/cpp/v_setbuf.c [FORMAT=unix:utf-8] [TYPE=C] [COL=001] [ROW=001/28(3%)]
```

```
mail cpp # gcc -std=c99 v_setbuf.c -o v_setbuf
mail cpp # ./v_setbuf a_file some_words&
[2] 7501
mail cpp # tail a_file
mail cpp # date; sleep 60; date
一 12月 23 00:43:49 CST 2013
[2]-  Done                    ./v_setbuf a_file some_words
一 12月 23 00:44:49 CST 2013
mail cpp # tail a_file
some_wordsmail cpp #
```

整个流程表明使用了全缓冲

5-2、5.7节有说明这种情况；

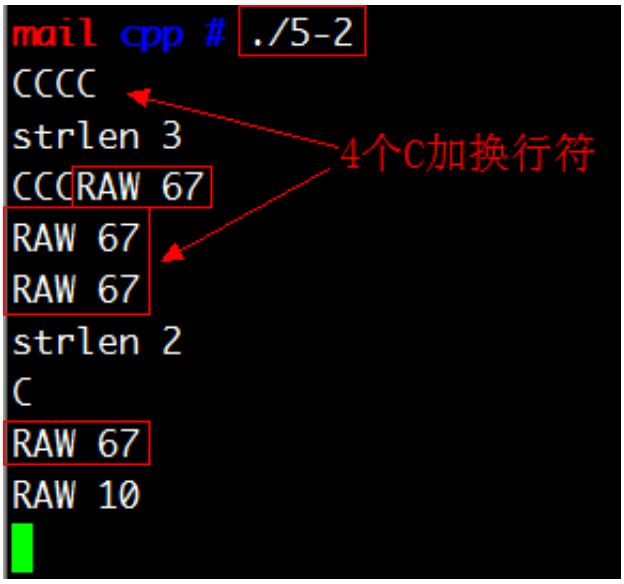
对于fgets，必须指定缓冲区的长度n。此函数一直读到下一个换行符为止，但是不超过$n-1$个字符，读入的字符被送入缓冲区。该缓冲区以null字符结尾。如若该行（包括最后一个换行符）的字符数超过$n-1$，则fgets只返回一个不完整的行，但是，缓冲区总是以null字符结尾。对fgets的下一次调用会继续读该行。

函数fputs将一个以null符终止的字符串写到指定的流，尾端的终止符null不写出。注意，这并不一定是每次输出一行，因为它并不要求在null符之前一定是换行符。通常，在null符之前是一个换行符，但并不要求总是如此。

```c
#include <stdio.h>
#include <string.h>
#define MAXLINE 4

int main(int argc, const char *argv[]) {
    char buf[MAXLINE];
    int i;
    while(fgets(buf, MAXLINE, stdin) != NULL) {
        printf("strlen %lu\n", strlen(buf));
        if (fputs(buf, stdout) == EOF) {
            printf("fputs error!");
        }
        for (i = 0; i < strlen(buf); i++) {
            printf("RAW %d\n", buf[i]);
        }
    }
    return 0;
}
```

```
mail cpp # ./5-2
CCCC
strlen 3
CCCRAW 67
RAW 67
RAW 67
strlen 2
C
RAW 67
RAW 10
```



5-3、意味着什么也没输出；

```c
#include <stdio.h>

int main(int argc, const char *argv[]) {
    printf("%d\n", printf(""));
    return 0;
}
```

```
mail cpp # g++ 5-3.cpp -o 5-3
mail cpp # ./5-3
0
```

5-4、这个藏的比较深，有个AIX可以试下；

这三个函数在返回下一个字符时，会将其unsigned char类型转换为int类型。说明为不带符号的理由是，如果最高位为1也不会使返回值为负。要求整型返回值的理由是，这样就可以返回所有可能的字符值再加上一个已出错或已到达文件尾端的指示值。在<stdio.h>中的常量EOF被要求是一个负值，其值经常是-1。这就意味着不能将这三个函数的返回值存放在一个字符变量中，以后还要将这些函数的返回值与常量EOF相比较。

5-5、该题让人一头雾水，看后面答案更是让人哑口无言；

```c
int
__path_search (char *tmpl, size_t tmpl_len, const char *dir, const char *pfx,
               int try_tmpdir)
{
  const char *d;
  size_t dlen, plen;

  if (!pfx || !pfx[0])
    {
      pfx = "file";
      plen = 4;
    }
  else
    {
      plen = strlen (pfx);
      if (plen > 5)
    plen = 5;
    }

  if (try_tmpdir)
    {
      d = __secure_getenv ("TMPDIR");
      if (d != NULL && direxists (d))
~/glibc-2.15/sysdeps/posix/tempname.c [FORMAT=unix:utf-8] [TYPE=C] [COL=001] [
```

5-6、FILE结构体的_fileno成员既该对象的文件描述符；

```c
#include <stdio.h>      // For fopen, fclose;
#include <unistd.h>     // For fsync;
#include <string.h>     // For strlen;
int main(int argc, const char *argv[]) {
    FILE *fp = fopen((char *) argv[1], "a+");
    char buf[4096];
    if(strlen(argv[2]) != fwrite(argv[2], sizeof(char), strlen(argv[2]), fp)) {
        printf("fwrite error!");
    }
    fsync(fp->_fileno);
    fclose(fp);
    return 0;
}
```

```
mail cpp # g++ 5-6.cpp -o 5-6
mail cpp # ./5-6 a_file XXXXXX
mail cpp # cat a_file
some_wordsXXXXXXmail cpp #
```

5-7、注意fgets；

对于行缓冲有两个限制。第一，因为标准I/O库用来收集每一行的缓冲区的长度是固定的，所以只要填满了缓冲区，那么即使还没有写一个换行符，也进行I/O操作。第二，任何时候只要通过标准I/O库要求从(a)一个不带缓冲的流，或者(b)一个行缓冲的流（它要求从内核得到数据）得到输入数据，那么就会造成冲洗所有行缓冲输出流。在(b)中带了一个在括号中的说明，其理由是，所需的数据可能已在该缓冲区中，它并不要求在需要数据时才从内核读数据。很明显，从不带缓冲的一个流中进行输入（(a)项）要求当时从内核得到数据。