C Programming

File Handling

王晓林

June 28, 2011

✉ wx672ster@gmail.com
☎ 13577067397

References

 C Primer Plus 5th Edition, November 23, 2004, Sams

What Is a File?

- In UNIX, everything is a file.
- In UNIX, everything is a stream of bytes.

Stream

- is a continuous sequence of bytes.
- is a portable way of reading and writing data.
- is a common, logical interface to a *file* (disk file, directory, the screen, the keyboard, a sound card, a NIC, etc.)

Text View, Binary View

- text output: printable characters + CR/LF/Tab
- binary output: not human readable

Text vs. Binary

```
|--------+--------+--------+--------|
|    5   |   6    |   7    |   8    | <- string
|--------+--------+--------+--------|
|00110101|00110110|00110111|00111000| <- ascii
|--------+--------+--------+--------|


|--------+--------|
|      5678       | <- decimal number
|--------+--------|
|00010110|00101110| <- binary number
|--------+--------|
```

Functions

As a programmer, you will have to write programs that

- create files
- write into files
- read from files

Functions to use:

fopen()	getc()	putc()	fclose()
fprintf()	fscanf()	fgets()	fputs()
rewind()	fseek()	ftell()	fflush()
fgetpos()	fsetpos()	feof()	ferror()
ungetc()	setvbuf()	fread()	fwrite()

FILE --- An Internal C Data Structure

- We just need to declare a variable or pointer of *FILE* type in our programs.
- We do not need to know any more specifics about *FILE* definition.
- We must open a *stream* before doing any I/O,
- then access it
- and then close it.

Example

```c
/* count.c -- using standard I/O */
#include <stdio.h>

#include <stdlib.h> // ANSI C exit() prototype
int main(int argc, char *argv[])
{
    int ch;          // place to store each character as read
    FILE *fp;        // "file pointer"
    long count = 0;

    if (argc != 2)
    {
        printf("Usage: %s filename\n", argv[0]);
        exit(1);
    }
    if ((fp = fopen(argv[1], "r")) == NULL)
    {
        printf("Can't open %s\n", argv[1]);
        exit(1);
    }
    while ((ch = getc(fp)) != EOF)
    {
       putc(ch,stdout);  // same as putchar(ch);
       count++;
    }
    fclose(fp);
    printf("File %s has %ld characters\n", argv[1], count);

    return 0;
}
```

In the program ...

1. checks the value of *argc*
2. Using *argv*[0] instead of the program name explicitly
3. *exit*()
4. *fopen*() and *fclose*()
5. *getc*() and *putc*()
6. *EOF*
7. *stdin*, *stdout*, and *stderr*

stdin, stdout, stderr

Deferent names

- Standard files
- Standard file pointers
- Predefined file descriptors
- Predefined streams

- stdin
stdout are automatically open to all C programs.
stderr
- There is no need to use fopen on them.

fopen()

- returns a file pointer
- returns NULL if failed to open a file

```
1 /* declare a stream and prototype fopen */
2 FILE *stream, *fopen();
3
4 stream = fopen("myfile.dat","r");
```

```
1 /* it is good practice
2    to to check file is opened correctly */
3 if ((stream = fopen("myfile.dat", "r"))
4               == NULL)
5 {
6   printf("Can't open %s", "myfile.dat");
7   exit(1);
8 }
```

Mode Strings for fopen()

"r"	Open a text file for reading
"w"	Open a text file for writing
"a"	appending to the end of a file
"r+"	Open a text file for update
"w+"	Open a text file for update
"a+"	Open a text file for update
"rb", "rb+", "r+b"	binary read
"wb", "wb+", "w+b"	binary write
"ab", "ab+", "a+b"	binary append

getc() and putc()

```
1 ch = getc(fp);     // read from fp
2 ch = getc(stdin);  // read from keyboard
3 ch = getchar();    // read from keyboard
4 putc(ch,fpout);    // write to fpout
5 putc(ch,stdout);   // write to screen
6 putchar(ch);       // write to screen
```

EOF

Good design to avoid problems attempting to read an empty file

```
1 int ch;
2 FILE *fp;
3
4 fp = fopen("wacky.txt", "r");
5
6 while (( ch = getc(fp)) != EOF)
7 {
8   putchar(ch);   // process input
9 }
```

EOF

Bad design

```
1  int ch;
2  FILE *fp;
3
4  fp = fopen("wacky.txt", "r");
5
6  while (ch != EOF)   // ch undetermined
7  {
8    ch = getc(fp);    // get input
9    putchar(ch);      // process input
10 }
```

Two problems:

1. the first time ch is compared with EOF, it has not yet been assigned a value.
2. if getc() does return EOF, the loop tries to process EOF as if it were a valid character.

fclose()

```
1 if (fclose(fp) != 0)
2   printf("Error in closing file %s\n", argv[1]);
```

Open two files simultaneously

```
1  // reducto.c -- reduces your files by two-thirds!
2  #include <stdio.h>
3  #include <stdlib.h>     // for exit()
4  #include <string.h>     // for strcpy(), strcat()
5  #define LEN 40
6
7  int main(int argc, char *argv[])
8  {
9      FILE  *in, *out;    // declare two FILE pointers
10     int ch;
11     char name[LEN];     // storage for output filename
12     int count = 0;
13
14 // check for command-line arguments
15     if (argc < 2)
16     {
17         fprintf(stderr, "Usage: %s filename\n", argv[0]);
18         exit(1);
19     }
20 // set up input
21     if ((in = fopen(argv[1], "r")) == NULL)
22     {
23         fprintf(stderr, "I couldn't open the file \"%s\"\n", argv[1]);
24         exit(2);
25     }
26 // set up output
27     strncpy(name,argv[1], LEN - 5); // copy filename
28     name[LEN - 5] = '\0';
29     strcat(name,".red");            // append .red
30     if ((out = fopen(name, "w")) == NULL)
31     {                          // open file for writing
32         fprintf(stderr,"Can't create output file.\n");
33         exit(3);
34     }
35 // copy data
36     while ((ch = getc(in)) != EOF)
37         if (count++ % 3 == 0)
38             putc(ch, out);  // print every 3rd char
39 // clean up
40     if (fclose(in) != 0 || fclose(out) != 0)
41         fprintf(stderr,"Error in closing files\n");
42
43     return 0;
44 }
```

fprintf(), fscanf()

- fprintf(fp, "Syntax error on line %d", lineno);
- fprintf(stderr, "Syntax error on line %d", lineno);
- printf("Syntax error on line %d", lineno);
- fscanf(fp, "%s", string);
- fscanf(stdin, "%s", string);
- scanf("%s", string);

uses fprintf(), fscanf(), and rewind()

```c
/* addaword.c -- uses fprintf(), fscanf(), and rewind() */
#include <stdio.h>
#include <stdlib.h>
#define MAX 40

int main(void)
{
    FILE *fp;
    char words[MAX];

    if ((fp = fopen("wordy", "a+")) == NULL)
    {
        fprintf(stdout,"Can't open \"words\" file.\n");
        exit(1);
    }

    puts("Enter words to add to the file; press the Enter");
    puts("key at the beginning of a line to terminate.");
    while (gets(words) != NULL && words[0] != '\0')
        fprintf(fp, "%s ", words);

    puts("File contents:");
    rewind(fp);             /* go back to beginning of file */
    while (fscanf(fp,"%s",words) == 1)
        puts(words);

    if (fclose(fp) != 0)
        fprintf(stderr,"Error closing file\n");

    return 0;
}
```

fgets(), fputs()

- fgets(buf, MAX, fp);
  - buf is the name of a char array
  - MAX is the maximum size of the string
  - fp is the pointer-to-FILE

  fgets() returns the value NULL when it encounters EOF
- fgets(buf, MAX, stdin);
- gets(buf);

- fputs(buf, fp);
  - buf is the string address
  - fp identifies the target file
- fputs(buf, stdout);
- puts(buf);

using fgets() and fputs()

```
1  #include <stdio.h>
2  #define MAXLINE 20
3  int main(void)
4  {
5    char line[MAXLINE];
6
7    while (fgets(line, MAXLINE, stdin) != NULL &&
8              line[0] != '\n')
9           fputs(line, stdout);
10   return 0;
11 }
```

Jump To a Certain Position In a File

fseek(), ftell()

```c
long int pos;
pos = ftell(fp); // record current position
// after some operation, you can ...
fseek(fp, pos, SEEK_SET); // go back to pos
```

fp: file pointer

pos: offset, long int

SEEK_...: identifies the starting point.

- SEEK_SET: Beginning of file
- SEEK_CUR: Current position
- SEEK_END: End of file

Example

```
1  // go to the beginning of the file
2  fseek(fp, 0L, SEEK_SET);
3  // go 10 bytes into the file
4  fseek(fp, 10L, SEEK_SET);
5  // advance 2 bytes from the current position
6  fseek(fp, 2L, SEEK_CUR);
7  // go to the end of the file
8  fseek(fp, 0L, SEEK_END);
9  // back up 10 bytes from the end of the file
10 fseek(fp, -10L, SEEK_END);
```

L: long integer

positive: move forward

negative: move backword

zero: stay put

reverse.c -- displays a file in reverse order

```
 1 #include <stdio.h>
 2 #include <stdlib.h>
 3 #define CNTL_Z '\032'    /* eof marker in DOS text files */
 4 #define SLEN 50
 5 int main(void)
 6 {
 7     char file[SLEN];
 8     char ch;
 9     FILE *fp;
10     long count, last;
11
12     puts("Enter the name of the file to be processed:");
13     gets(file);
14     if ((fp = fopen(file,"rb")) == NULL)
15     {                        /* read-only and binary modes */
16         printf("reverse can't open %s\n", file);
17         exit(1);
18     }
19
20     fseek(fp, 0L, SEEK_END);          /* go to end of file */
21     last = ftell(fp);
22
23     for (count = last- 1; count >= 0; count--)
24     {
25         fseek(fp, count, SEEK_SET); /* go backward        */
26         ch = getc(fp);
27     /* for DOS, works with UNIX */
28         if (ch != CNTL_Z && ch != '\r')
29             putchar(ch);
30     }
31   putchar('\n');
32   fclose(fp);
33
34   return 0;
35 }
```

fread(), fwrite()

fprintf() vs. fwrite()

```
|--------+--------|
|00010110|00101110| <- int num = 5678;
|--------+--------|

|--------+--------+--------+--------|
|   '5'  |  '6'   |  '7'   |  '8'   | <- characters
|--------+--------+--------+--------|
|00110101|00110110|00110111|00111000| <- fprintf(fp, "%d", num);
|--------+--------+--------+--------|

|--------+--------|
|      5678       | <- decimal number
|--------+--------|
|00010110|00101110| <- fwrite(&num, sizeof(int), 1, fp);
|--------+--------|
```