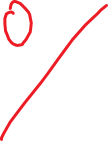
FILE IO IN C  
READ MODE TRIALS



This is a report to an experiment investigating the behavior of file pointers in the C programming language when a pointer is used to open a text and a binary file in “read” mode. A total of 10 experiments were run under different circumstances.

Specifications of trial environment:

OS: Windows 7 Ultimate 64-bit  
Processor: Intel G2010 Dual-Core CPU @2.80GHz  
Installed Memory: 6GB  
IDE: Code::Blocks v 17.12 stable  
Compiler: MinGW 32

Throughout all experiments, it should be noted that a file pointer named “fp” was declared wherever a single pointer was necessary unless stated otherwise.

Prepared by: Group 8

Team Members:

S201805068 - Showvik Biswas  
S201805080 - Md. Erfan Jahangir Chowdhury  
S201805092 - Ishika Tarin Ime  
S201805104 - Swarnali Saha  
S201805116 - Sanjida Islam Era

**Test 1: A text (.txt) file was opened in read mode and the output of its contents to the console was attempted using fgetc()**

**Result**: **SUCCESS**

**Comments:** This task was done just as a base case to demonstrate the normal functioning of the “read” mode.

Code Snippet:

FILE\* fp;

if((fp = fopen("test.txt", "r"))==NULL)

{

printf("Could not open file.");

exit(1);

}

while(!feof(fp))

{

if(feof(fp))

break;

if(ferror(fp))

exit(1);

int ch = fgetc(fp);

printf("%c", ch);

}

**Test 2: A text (.txt) file was opened in read mode and the output of its contents to the console was attempted using fgets()**

Test file content: “This is a test.”

**Result**: **SUCCESS**

**Comments:** This shows that proper use of fgets() also returns the file back without any problem.

Code Snippet:

char s[100];

fgets(s, 100, fp);

if(ferror(fp))

exit(1);

printf("%s", s);

**Test 3: A text (.txt) file was opened in read mode and the output of its contents to the console was attempted using fscanf(). However, the test content is changed to two integers.**

Test file content: “21 45”

**Result**: **SUCCESS**

**Comments:** This shows that proper use of fscanf() also returns the file back without any problem. The content was changed to ensure that the decimal format specifier works properly under these circumstances.

Code Snippet:

int a, b;

fscanf(fp, “%d %d”, &a, &b);

if(ferror(fp))

exit(1);

printf(“%d %d”, a, b);

**Test 4: A binary (.txt) file was opened in read mode and contents were tried to be read from it using fscanf()**

Two integers 20 and 30 were written to a file using the “wb” specifier in file I/O.

**Result: FAILURE**

**Output: 30 8727624**

Comments: The output did not seem to have any relevance to the input, and it was safely assumed that the output was garbage. It can hence be said that binary files cannot be read using fscanf() if the file pointer is declared in plain “read” mode. The program however returned 0. As error checking gates were present, it can be said that there was no issue in opening the file.

Code Snippet:

int a,b;

fread(&a, 4, 1, fp);  
fread(&b, 4, 1, fp);

if(ferror(fp))

exit(1);

printf(“%d %d”, a, b);

**Test 5: A test (.txt) file was opened in read mode and contents were tried to be written to it using fputs()**

This file was kept empty initially for experimental purposes.

**RESULT: FAILURE**

Process returned 1, showing the activation of the exit handler. This experiment shows fputs() does not work in read mode. However, there was a single character output to the test file, followed by a few space characters.

Output:

Code Snippet:

FILE\* fp;

if((fp = fopen("test.txt", "r"))==NULL)

{

printf("Could not open file.");

exit(1);

}

fputs("Test input", fp);

if(ferror(fp))

exit(1);

return 0;

**Test 6: A test (.txt) file was opened in read mode and contents were tried to be written to it using fprintf()**

This file was kept empty initially for experimental purposes. The words “Test Input” were attempted to be written in it.

**RESULT: FAILURE**

Process returned 1, showing the activation of the exit handler. This experiment shows fprintf() does not work in read mode. There was no output to the file, and it remained empty.

Output: NO OUTPUT

Code Snippet:

FILE\* fp;

if((fp = fopen("test.txt", "r"))==NULL)

{

printf("Could not open file.");

exit(1);

}

fputc('z', fp);

if(ferror(fp))

exit(1);

**Test 7: A test (.txt) file was opened in read mode using two file pointers and using both pointers, its contents were attempted to be read into the console.**

This file contained the sentence “The quick brown fox jumped over the lazy dog”. This time, two file pointers fp and fp2 were declared. fscanf() was used to read by fp, and fgetc() was used to read by fp2.

**Result: FAILURE**

**Comments**: The program crashed. fscanf() and fgetc() should not be used simultaneously.

**\*\*** **UNEXPECTED OBSERVATION:** When the fegtc() function was commented out, a smooth return from scanf was expected. The output however was unexpected, and unexplained, since the array taken was much larger than the output size.

Output after commentary:   
The quick brow

Code Snippet:

char s[100];

fscanf(fp, "%[^n]s", s);

while(!feof(fp))

{

if(feof(fp))

break;

if(ferror(fp))

exit(1);

int ch = fgetc(ch);

printf("%c", ch);

}

printf("\n%s", s);

**Test 8: A test (.txt) file was opened in read mode, and the binary function fread() was used to read from it.**

The test file contained the sentence “Test”

**Result: FAILURE**

**Comments:** fread() managed to get the string onto console properly, but added a few garbage characters at the end.

Output: Testn¦(

Code Snippet:

// fp and fp2 declared

char s[100];

fread(s, 4, 1, fp);

printf(“%s”, s);

**Test 9: A test (.txt) file of known character numbers was opened in read mode and the following operations were performed.**

fseek() was used to navigate to some characters away from SEEK\_SET. Then, a simple iteration of fgetc() was used to print the contents of the file to the console. After printing, fscanf() was used to read the character the “file pointer” was pointing to.

The test file contained the word “Simple”, and three letters were traversed from SEEK\_SET

**Result: FAILURE**

It was expected that the file would the console would print the last three letters of the word, but the program crashed, even with the fscanf() function commented out.

Code Snippet:

fseek(fp, 3, SEEK\_SET);

char s;

while(!feof(fp))

{

if(feof(fp))

break;

if(ferror(fp))

break;

int ch = fgetc(ch);

printf("%c", ch);

}

fscanf(fp, "%c", s);

**Test 10: A test (.txt) file was opened in read mode and the following operations were run on it**

Let’s assume we have 2 char array ch1[10] & ch2[10]

Now, if we try to assign 2 strings to them using fscanf (both the strings are less than 10) it works correctly.

But if second string is grater than 10 ch2 is assigned successfully even though it should be over flowed. Meanwhile, the first variable is assigned with the second string & it is started from 11th index.

txt file:

abcde

abcdefghijkl

code:

fscanf(fp,"%s",ch1);

fscanf(fp,"%s",ch2);

printf("%s\n",ch1)

printf("%s",ch2);

output:

kl

abcdefghijkl

Here abcde was first assigned to ch1 then abcdefghijkl was assigned to ch2 and as it is overflowed it changed ch1 to kl.

As they were declared as

char ch1[10],ch2[10];

It might me that their address are followed, so a int was assigned between them but it gives the same result.

Instead of assigning it to ch1 & ch2 , if we change the position of them but the program crashes.

Again, if the first string is greater than 10 , he program also crashes.

Here 1st string is fully assigned to ch1 but assigning to ch2 failed.

In that case, if we use fread() to assign in ch1 & fscanf() to assign in ch2 it outputs the same result mentioned above.

But i another experiment ch1 was assigned 1st 10 character along with some garbage and ch2 is assigned with the 1st string starting from 11th character

After closing the file & printing ch2 in console then if we print some string to console the string is replaced by garbage

But before closing the file we do the same thing everything works fine.

Again if we use fscanf() for ch1 and fread() for ch2 then both value is assigned with a lot of garbage.

If we use fread() to read a string & the string is less than 10 it works perfectly.

But if the length is 10 sometimes nothing is assigned to the variable, sometimes the full string along with some garbage at the end of it (total length is 13/15) is assigned to it.

If the string is greater than 10 sometimes only first 10 value along with some garbage at the end of it (total length is 13/15) is assigned to it, sometimes only the sting started from 11th index is assigned to it.

txt:

abcde

abcde

To assign them in ch1 & ch2 , if we use fscanf() for both of them ch1 is assigned fully but it also assigned some garbage in it with a new line and ch2 is fully garbage

output:

ch1: abcde

abcd`V,w1

ch2: e `

txt:

123456789

123456789

if we assign them with fread()

output:

ch1: 123456789

`V,w1

ch2: 123456789123456789

`V,w1

This means that fread() always start reading from files first position, though why the garbage value is adding is unknown

Throughout the experiment, we saw that fread() assigns some garbage value in the variable , that kind of problem isn't facing using fscanf() , though both of them creates problem if overflowed and changes another variable.

So, it is better to use fscanf() than fread() if we open the file as read mode.