**File I/O**

Table of Contents

[Abstraction 2](#_Toc65506082)

[Working with Files 1](#_Toc65506083)

## **Abstraction**

A black box takes a certain type of input and gives a certain type of output. An example of a black box are predefined functions. It is not necessary to know the details of the abstraction layer. It simplifies things and also works as a security layer.

Software does not interact with hardware. Here, the operating system acts as a layer of abstraction.

* Program Keyboard ()

Display ()

File/Disk Operations

* Program Disk

Open Operation

Close Operation

#include <stdio.h>  
FILE \*fp  
{  
 Name  
 Location  
 Access Permissions  
 Current Position  
}

C

Name – This is the name of the file (text.txt).

Location – This is the location of the file on the hard disk. This can either be an absolute path (C:/Acer/text.txt), or a relative path (/text.txt).

Access Permissions – Read, write or execute permissions.

Current Position – This keeps track of which part of the file is currently being used (the position of the cursor). This is connected with how the stream works.

A stream can be of two types, a binary stream or a character stream. The binary stream works with binary values, while the character stream works with ASCII values.

Working with Files

FILE \*fopen(char \*fname, char \*mode);  
int fclose (FILE \*fp);  
int fgetc (FILE \*fp);  
int fputc(int ch, FILE \*fp);  
int feof(FILE \*fp); *//eof means End Of File*int ferror(FILE \*fp);

C

FILE \*fp;  
if ((fp = fopen("text.txt", "r")) == NULL)  
*//if text.txt does not exist*{  
 printf("Error");  
 return 0; *//exit(1) can also be used*}

C

The above code must always be included.

r = read character

w = write character

a = append character

rb = read binary code

rw = write binary code

ra = append binary code

Using fopen("text.txt", "w") either overwrites the current text.txt file, or creates one if it does not exist.

Using fopen("text.txt", "a") either adds new information to the current text.txt. file, or creates one if it does not exist.

FILE \*fopen(char \*frame, char \*mode);

C

This function takes as input the path of the file and mode of opening, and returns the pointer to the file. It returns NULL if unsuccessful.

int fclose (file \*fp);

C

This function takes as input the pointer to the file. It returns EOF (EOF = -1) if unsuccessful and 0 if successful.

int fgetc(FILE \*fp);

C

fgetc() reads the next character from fp and stores it in the lower order byte of the integer (an integer is of 4 bytes). It automatically moves to the next character and returns EOF when a character cannot be read.

int fputc(char ch, FILE \*fp);

C

fputc() writes the next character to fp. It behaves in a similar manner to fgetc().

Example 1:

#include <stdio.h>  
int main ()  
{  
 char str[80] = "This is a test";  
 FILE \*fp, char \*p, int i;  
 if ((fp = fopen('myfile', "w")) == **NULL**)  
 {  
 printf("Could not open");  
 exit(1);  
 }  
 p = str; *// here str works like a pointer  
 // the value of p is 'T' (the first character of str)* while (\*p)  
 {  
 if (fputc(\*p, fp) == **EOF**)  
 {  
 printf("Error writing");  
 exit(1);  
 }  
 p++;  
 }  
 fclose(fp);  
}

C

Example 2:

#include <stdio.h>  
int main ()  
{  
 char str[80] = "This is a test";  
 FILE \*fp, char \*p, int i;  
 if ((fp = fopen('myfile', "w")) == **NULL**)  
 {  
 printf("Could not open");  
 exit(1);  
 }  
 for ( ; ; )  
 {  
 i = fgetc(fp);  
 if (i == **EOF**) break;  
 puts char(i);  
 }  
 fclose(fp);  
}

C

ch = fgetc()

C

successful – returns character

unsuccessful – returns EOF

This is when a character stream is used. If a binary stream is used, any value may be returned, including -1 (the integer used for EOF). This means it is impossible to differentiate between EOF and the value -1.

Similarly, feof() will return EOF at the end of file and if there is an error, making it impossible to differentiate between the two.

feof() - int feof(FILE \*fp);

ferror() - int ferror(FILE \*fp);

Normally, when a function returns a value 0, it means no, and a non-zero value means yes. For feof(), this is the opposite. A 0 value means the end of file has not been reached while a non-zero value means the end of file has been reached. ferror() is the same.

#include<stdio.h>  
#include<stdlib.h>  
int main(int argc, char \*argv[])  
{  
 FILE \*from, \*to; char ch;  
 if (argc!=3) printf("Wrong command\n"), exit(1);  
 *// 3 = current file, from file and to file*

*// if ((from = fopen(argv[1], "rb")) == NULL)  
 // from file error* {  
 printf("Read error\n");  
 return 0;  
 }  
 if ((to = fopen(argv[2], "wb")) == **NULL**)  
 *// to file error* {  
 printf("Write error\n");  
 return 0;  
 }  
  
  
 while(!feof(from)) *// stop when end of file is reached* {  
 ch = fgetc(from);  
 if(ferror(from)) puts("Error\n"), exit(1);  
 if(!feof(from)) fputc(ch, to);  
 *// if fgetc got the EOF,*

*// don’t execute if(ferror(to)) puts("Error\n"), exit(1);* }  
 fclose(from), fclose(to);  
}

C

Teach Yourself C Chapter 09, Page - 272 (above code), 273 (comparing files)

gets() – takes only strings as input, stops taking input when enter is pressed

puts() – prints only strings and puts a newline character

scanf() – stops taking input when space/enter is pressed

– can take anything as input

printf() – prints anything

There are comparable functions for FILE data types.

fgets() – char \*fgets(char \*str, int num, FILE \*fp); *// initialization*

- char \*str is the input string

- int num is the number of characters (takes num – 1 characters)

- FILE \*fp is the file pointer

- stops taking input at NULL character or EOF

fputs() – int fputs(char \*str, FILE \*fp); *//initialization*

- char \*str is the output string

- FILE \*fp is the file pointer

- works exactly like puts(), but doesn’t automatically add a new line

#include <stdio.h>  
int main()  
{  
 FILE \*fp = fopen("text.txt", "w");  
 fputs("This is the first line. ", fp);  
 fputs("This is the second line.", fp);  
 fclose(fp);  
}

C

text.txt:

This is the first line. This is the second line.

#include <stdio.h>  
int main()  
{  
 FILE \*fp = fopen("text.txt", "r");  
 char str1[100], str2[100];  
 fgets(str1, 100, fp);  
 fgets(str2, 100, fp);  
 puts(str1);  
 puts(str2);  
 fclose(fp);  
}

C

text.txt:

This is the first line. This is the second line.

Screen:

This is the first line.

This is the second line.

One extra line is printed after each sentence since fgets() adds one newline character and puts() adds another newline character.

fscanf() – int fscanf(FILE \*fp, char \*control\_string); *//initialization*

- exactly like scanf()

- fscanf("%s %s", a, b, fp);

- FILE \*fp is the file pointer

- har \*control\_string is the part inside "". It’s the same thing.

fprintf() – int fprintf(FILE \*fp, char \*control\_string); *//initialization*

- exactly like fprintf()

- fprintf(fp, "%d %d", a, b);

All four of these functions can be used to print/take input from the screen and from the file.