To:
Tenouk

# MODULE 9a - MORE C FILE INPUT/OUTPUT 2

---

MODULE 19 - C++ FILE I/O

*create this, delete that, write this, read that, close this, open that*

My Training Period: hours

**C file input/output abilities:**

This is a continuation from previous Module. For C++ and MFC (Windows GUI programming) it is called Serialization and the topics are in Single Document Interface (SDI) and Multiple Document Interface (MDI). The C++ standard input/output file is discussed in C++ file input/output. The source code for this Module is: C file input/output program source codes. Trainee must be able to understand and use:

- A sequential access file – Read and Write related functions.
- Characters, lines and blocks disk file reading and writing related functions.
- A Random access files – Read and Write related functions.
- Some C File Management Functions.
- Other C libraries used for file I/O.

### 9.6.2 Reading And Writing Disk File

- The previous program example does not do anything with the **tkk1103.txt** text file, except open and close it. Some text has been saved in tkk1103.txt, so how can you read them from the file?
- In C you can perform I/O operations in the following ways:

1. Read or write **one character at a time**.
2. Read or write **one line of text** (that is, one line of characters) at a time.
3. Read or write **one block of characters** at a time.

#### 9.6.2.1 One Character At A Time

- Among the C I/O functions, there is a pair of functions, fgetc()/fgetc_s() and fputc()/fputc_s(), that can be used to read from or write to a disk file one character at a time. The _s version is a secure version.
- The prototype for the fgetc() function is:

```
int    fgetc(FILE  *stream);
```

- The stream is the file pointer that is associated with a stream. The fgetc() function fetches the next character from the stream specified by stream. The function then returns the value of an int that is converted from the character.
- The prototype for the fputc() function is:

```
int    fputc(int c, FILE *stream);
```

- c is an int value that represents a character. In fact, the int value is converted to an unsigned char before being output. stream is the file pointer that is associated with a stream. The fputc() function returns the character written if the function is successful, otherwise, it returns EOF. After a character is written, the fputc() function advances the associated file pointer.
- Let explore the program example. Before that, you have to create two text files named, **testone.txt** and **testtwo.txt** then save it in the same folder where the your main() program is or provide the full path strings if the files is in another folder. Then for file **testtwo.txt**, write the following texts and save it.

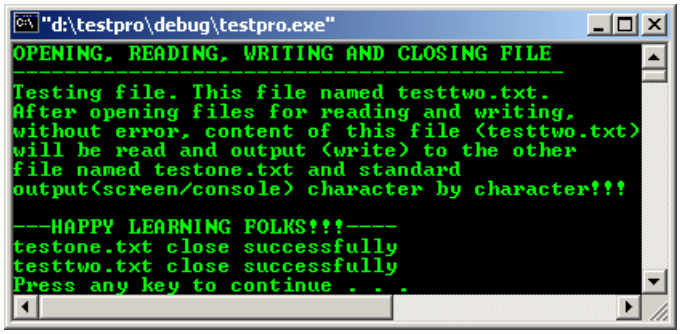
```
OPENING, READING, WRITING AND CLOSING FILE
------------------------------------------
Testing file. This file named testtwo.txt.
After opening files for reading and writing,
without error, content of this file (testtwo.txt)
will be read and output (write) to the other
file named testone.txt and standard
output(screen/console) character by character!!!

---HAPPY LEARNING FOLKS!!!----
```

Content of file testtwo.txt

- Then, if you run the program with no error, modify the content of the **testtwo.txt**, recompile and rerun the program. The displaying texts and the content of **testone.txt** also will change.
- Next check also the content of **testone.txt** file, the content should be same as **testtwo.txt** file and the texts displayed on your screen.

```
1.  // reading and writing one character at a time
2.  #include <stdio.h>
3.  #include <stdlib.h>
4.
5.  // enumerated data type, SUCCESS = 0, FAIL = 1
6.  enum  {SUCCESS, FAIL};
7.
8.  // prototype function for reading from and writing...
9.  void CharReadWrite(FILE *fin, FILE *fout);
10.
11. int main()
12. {
13.   // declare two file pointers...
14.   FILE *fptr1, *fptr2;
15.   // define the two files name...
16.   char filename1[] = "testone.txt";
17.   char filename2[] = "testtwo.txt";
18.   int reval = SUCCESS;
19.
20.   // test the opening filename1 for writing....
21.   // if fails...
22.   if ((fptr1 = fopen(filename1, "w")) == NULL)
23.   {
24.       printf("Problem, cannot open %s.\n", filename1);
25.       reval = FAIL;
26.   }
27.   // if opening filename1 for writing is successful,
28.   // test for opening for reading filename2, if fails...
29.   else if ((fptr2 = fopen(filename2, "r")) == NULL)
30.   {
31.       printf("Problem, cannot open %s.\n", filename2);
32.
33.       reval = FAIL;
34.   }
35.   // if successful opening for reading from filename2
36.   // and writing to filename1...
37.   else
38.   {
39.       // function call for reading and writing...
40.       CharReadWrite(fptr2, fptr1);
41.       // close both files...
42.       if(fclose(fptr1)==0)
43.        printf("%s closed successfully\n", filename1);
44.       if(fclose(fptr2)==0)
45.        printf("%s closed successfully\n", filename2);
46.   }
47. // for Borland if compiled using its IDE...
48. // system("pause");
49. return reval;
50. }
51.
52. // read write function definition
53. void  CharReadWrite(FILE *fin, FILE *fout)
54. {
55.   int  c;
56.   // if the end of file is reached, do...
57.   while ((c = fgetc(fin)) != EOF)
58.   {
59.       // write to a file...
60.       fputc(c, fout);
61.       // display on the screen...
62.       putchar(c);
63.   }
64.   printf("\n");
65. }
```

**65 lines: Output:**




- This program read one character from a file, writes the character to another file, and then display the character to screen, a standard output.

#### 9.6.2.2 One Line At A Time

- Besides reading or writing one character at a time, you can also read or write one character line at time. There is a pair of C I/O functions, fgets()/fgets_s() and fputs()/fputs_s(), that allows you to do so. The fgets_s() and fputs_s() are the secure versions.
- The prototype for the fgets() function is:

```
char    *fgets(char    *s, int    n, FILE    *stream);
```

- s, references a character array that is used to store characters read from the opened file pointed to by the file pointer stream. n specifies the maximum number of array elements. If it is successful, the fgets() function returns the char pointers s. If EOF is encountered, the fgets() function returns a null pointer and leaves the array untouched. If an error occurs, the function returns a null pointer, and the contents of the array are unknown.
- The fgets() function can read up to n-1 characters, and can append a null character after the last character fetched, until a newline or an EOF is encountered.
- If a newline is encountered during the reading, the fgets() function includes the newline in the array. This is different from what the gets() function does. The gets() function just replaces the newline character with a null character.
- The prototype for the fputs() function is:

```
int    fputs(const    char    *s, FILE    *stream);
```

- s points to the array that contains the characters to be written to a file associated with the file pointer stream. The const modifier indicates that the content of the array pointed to by s **cannot be changed**. If it fails, the fputs() function returns a nonzero value, otherwise, it returns zero.
- The character array must include a null character at the end as the terminator to the fputs() function. Also, unlike the puts() function, the fputs() function does not insert a newline character to the string written to a file.
- Let try a program example. First of all, create two text file named **testhree.txt** and **testfour.txt** and put it under folder C:\. File **testfour.txt** should contain the following texts:

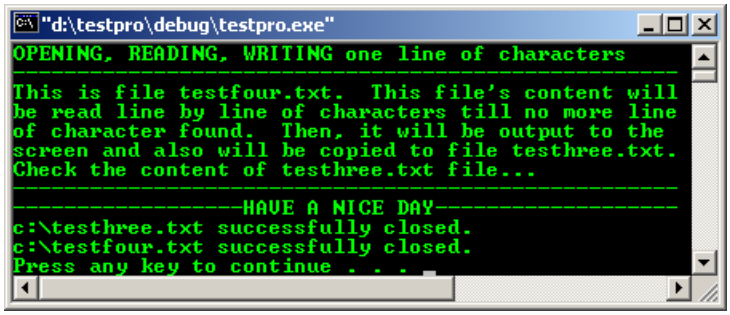
```
OPENING, READING, WRITING one line of characters
-------------------------------------------------------
This is file testfour.txt.  This file's content will
be read line by line of characters till no more line
of character found.  Then, it will be output to the
screen and also will be copied to file testhree.txt.
Check the content of testhree.txt file...
-------------------------------------------------------
------------------HAVE A NICE DAY--------------------
```

Content of testfour.txt file

```
1.  // reading and writing one line at a time
2.  #include <stdio.h>
3.  #include <stdlib.h>
4.
5.  enum  {SUCCESS, FAIL, MAX_LEN = 100};
6.
7.  // a function prototype for read and writes by line...
8.  void  LineReadWrite(FILE *fin, FILE *fout);
9.
10. int main(void)
11. {
12.     FILE *fptr1, *fptr2;
13.     // file testhree.txt is located at the root, c:
14.     // you can put this file at any location provided
15.     // you provide the full path, same for testfour.txt
16.     char filename1[] = "c:\\testhree.txt";
```

```
17.         char  filename2[] = "c:\\testfour.txt";
18.         char  reval = SUCCESS;
19.
20.         // test opening testthree.txt file for writing, if fail...
21.         if((fptr1 = fopen(filename1,"w")) == NULL)
22.         {
23.             printf("Problem, cannot open %s for writing.\n", filename1);
24.             reval = FAIL;
25.         }
26.
27.         // test opening testfour.txt file for reading, if fail...
28.         else if((fptr2=fopen(filename2, "r"))==NULL)
29.         {
30.             printf("Problem, cannot open %s for reading.\n", filename2);
31.             reval = FAIL;
32.         }
33.
34.         // if opening for writing and reading successful, do...
35.         else
36.         {
37.             // function call for read and write, line by line...
38.             LineReadWrite(fptr2, fptr1);
39.             // close both files stream...
40.             if(fclose(fptr1)==0)
41.                 printf("%s successfully closed.\n", filename1);
42.             if(fclose(fptr2)==0)
43.                 printf("%s successfully closed.\n", filename2);
44.           }
45.         // for Borland screenshot
46.         // system("pause");
47.         return reval;
48.     }
49.
50.     // function definition for line read, write...
51.     void LineReadWrite(FILE *fin, FILE *fout)
52.     {
53.         // local variable...
54.         char buff[MAX_LEN];
55.         while(fgets(buff, MAX_LEN, fin) !=NULL)
56.         {
57.           // write to file...
58.           fputs(buff, fout);
59.           // write to screen...
60.           printf("%s", buff);
61.         }
62.     }
```

**62 lines: Output:**




- In this program example, the text files are located in **C:** drive. The fgets() function is called repeatedly in a while loop to read one line of characters at a time from the **testfour.txt** file, until it reaches the end of the text file.
- In line 54, the array name buff and the maximum number of the array elements MAX_LEN are passed to the fgets() function, along with the file pointer fin that is associated with the opened testfour.txt file.
- Meanwhile, each line read by the fgets() function is written to another opened text file called testhree.txt that is associated with the file pointer fout. This is done by invoking the fputs () function in line 58.
- The statement in line 60 prints the contents of each string on the screen so that you see the contents of the **testfour.txt** file. You also can view the **testhree.txt** file content in a text editor to make sure that the contents of the **testfour.txt** file have been written to the **testhree.txt** file.

#### 9.6.2.3 One Block At A Time

- You can also read or write a block of data at a time. There are two C I/O functions, fread()

and fwrite(), that can be used to perform block I/O operations.

- The prototype for the fread() function is:

```
size_t  fread(void  *ptr, size_t  size, size_t  n, FILE  *stream);
```

- The ptr is a pointer to an array in which the data is stored. size indicates the size of each array element. n specifies the number of elements to be read. stream is a file pointer that is associated with the opened file for reading.
- size_t is an integral type defined in the header file stdio.h. The fread() function returns the number of elements actually read.
- The number of elements read by the fread() function should be equal to the value specified by the third argument to the function, unless an error occurs or an EOF is encountered.
- The fread() function returns the number of elements that are actually read, if an error occurs or an EOF is encountered.
- The prototype for the fwrite() function is:

```
size_t  fwrite(const  void  *ptr, size_t  size, size_t  n, FILE  *stream);
```

- ptr references the array that contains the data to be written to an opened file pointed to by the file pointer stream. size indicates the size of each element in the array. n specifies the number of elements to be written.
- The fwrite() function returns the number of elements actually written.
- If there is no error occurring, the number returned by fwrite() should be the same as the third argument in the function. The return value may be less than the specified value if an error occurs.
- That is the programmer's responsibility to ensure that the array is large enough to hold data for either the fread() function or the fwrite() function.
- In C, a function called feof() can be used to determine when the end of a file is encountered. This function is more useful when you are reading a binary file because the values of some bytes may be equal to the value EOF.
- If you determine the end of a binary file by checking the value returned by fread(), you may end up at the wrong position.
- Using the feof() function helps you to avoid mistakes in determining the end of a file. The prototype for the feof() function is:

```
int  feof(FILE  *stream);
```

- Here, stream is the file pointer that is associated with an opened file. The feof() function returns 0 if the end of the file has not been reached, otherwise, it returns a nonzero integer.
- Let take a look at the program example. Create two files named it **testfive.txt** and **testsix.txt** in **C:\Temp** folder or other folder that you choose provided that you provide the full path strings in the program. Write the following texts into **testsix.txt** file and save it.

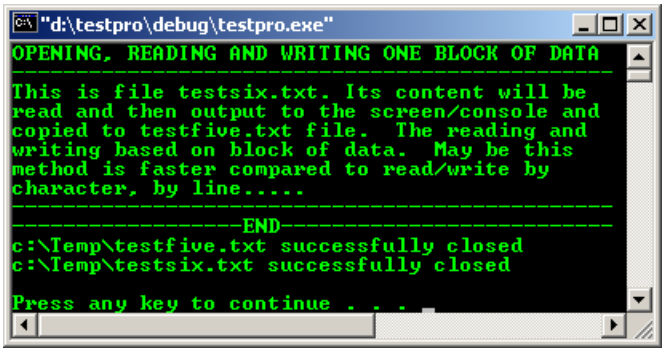
```
OPENING, READING AND WRITING ONE BLOCK OF DATA
-----------------------------------------------
This is file testsix.txt. Its content will be
read and then output to the screen/console and
copied to testfive.txt file.  The reading and
writing based on block of data.  May be this
method is faster compared to read/write by
character, by line.....
-----------------------------------------------
-----------------END-------------------------
```

Content of testsix.txt file

```
1.  // reading and writing one block at a time
2.  #include <stdio.h>
3.  #include <stdlib.h>
4.
5.  // declare enum data type, you will
6.  // learn this in other module...
7.  enum  {SUCCESS, FAIL, MAX_LEN = 80};
8.
9.  // function prototype for block reading and writing
10. void  BlockReadWrite(FILE  *fin, FILE  *fout);
11. // function prototype for error messages...
12. int   ErrorMsg(char  *str);
13.
14. int main(void)
15. {
16.     FILE  *fptr1, *fptr2;
17.     // define the filenames...
18.     // the files location is at c:\Temp
19.     char  filename1[] = "c:\\Temp\\testfive.txt";
```

```
20.         char  filename2[] = "c:\\Temp\\testsix.txt";
21.         int  reval = SUCCESS;
22.
23.         // test opening testfive.txt file for writing, if fail...
24.         if((fptr1 = fopen(filename1, "w")) == NULL)
25.         {
26.             reval = ErrorMsg(filename1);
27.         }
28.
29.         // test opening testsix.txt file for reading, if fail...
30.         else if ((fptr2 = fopen(filename2, "r")) == NULL)
31.         {
32.             reval = ErrorMsg(filename2);
33.         }
34.         // if opening files for writing and reading is successful, do...
35.         else
36.         {
37.             // call function for reading and writing
38.             BlockReadWrite(fptr2, fptr1);
39.             // close both files streams...
40.             if(fclose(fptr1)==0)
41.              printf("%s successfully closed\n", filename1);
42.             if(fclose(fptr2)==0)
43.              printf("%s successfully closed\n", filename2);
44.         }
45.         printf("\n");
46.         // for Borland...
47.         // system("pause");
48.         return  reval;
49.     }
50.
51.     // function definition for block read, write
52.     void   BlockReadWrite(FILE  *fin, FILE  *fout)
53.     {
54.         int   num;
55.         char  buff[MAX_LEN + 1];
56.         // while not end of file for input file, do...
57.         while(!feof(fin))
58.         {
59.             // reading...
60.             num = fread(buff, sizeof(char), MAX_LEN, fin);
61.             //append a null character
62.             buff[num * sizeof(char)] = '\0';
63.             printf("%s", buff);
64.             // writing...
65.             fwrite(buff, sizeof(char), num, fout);
66.         }
67.     }
68.
69.     // function definition for error message
70.     int  ErrorMsg(char  *str)
71.     {
72.         // display the error message...
73.         printf("Problem, cannot open %s.\n", str);
74.         return    FAIL;
75.     }
```

**75 lines: Output:**




- Note the use of fread() and fwrite() functions in the program. This program shows you how to invoke the fread() and fwrite() to perform block I/O operations.
- The testsix.txt file is read by the fread() function, and the fwrite() function used to write the contents read from testsix.txt to another file called testfive.txt.

**Further C file i/o reading and digging:**

1. The source code for this Module is: C file input/output program source codes.
2. For C++ and MFC (Windows GUI programming) it is called Serialization and the topics are in Single Document Interface (SDI) and Multiple Document Interface (MDI).
3. Check the best selling C / C++ books at Amazon.com.
4. Wide character/Unicode is discussed Character Sets, Unicode & Locale and the implementation using Microsoft C is discussed Windows Users & Groups C programming.
5. Implementation specific information for Microsoft can be found Microsoft C Run-Time Tutorials and More Win32 Windows C Run-Time programming Tutorials.