To:
Tenouk

# MODULE 9b - MORE C FILE INPUT/OUTPUT 4

---

## MODULE 19 - C++ FILE I/O

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

My Training Period: hours

**C file i/o 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:

- Some C File Management Functions.
- Other C libraries used for file I/O.

### 9.8 Redirecting The Standard Streams With freopen()

- We will discuss how to redirect the standard streams, such as stdin and stdout, to disk files. We can use freopen() function, which can associate a standard stream with a disk file.
- The prototype for the freopen() function is:

```
FILE   *freopen(const  char  *filename, const  char  *mode, FILE  *stream);
```

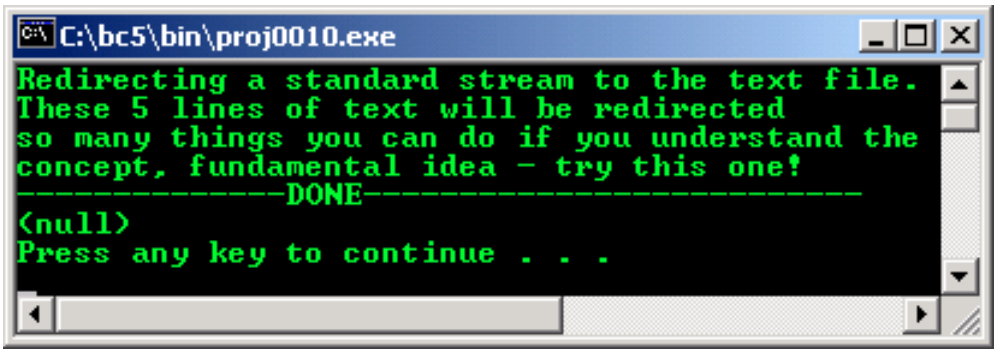
- filename is a char pointer referencing the name of a file that you want to associate with the standard stream represented by stream.
- mode is another char pointer pointing to a string that defines the way to open a file. The values that mode can have in freopen() are the same as the mode values in the fopen() function.
- The freopen() function returns a null pointer if an error occurs. Otherwise, the function returns the standard stream that has been associated with a disk file identified by filename.
- Let try a program example.

```
1.  // redirecting a standard stream
2.  #include <stdio.h>
3.  #include <stdlib.h>
4.
5.  enum  {SUCCESS, FAIL, STR_NUM = 6};
```

```
6.
7.          void  StrPrint(char  **str);
8.          int   ErrorMsg(char  *str);
9.
10.       int main(void)
11.       {
12.           // declare and define a pointer to string...
13.           char   *str[STR_NUM] = {
14.               "Redirecting a standard stream to the text file.",
15.               "These 5 lines of text will be redirected",
16.               "so many things you can do if you understand the",
17.               "concept, fundamental idea - try this one!",
18.               "--------------DONE-------------------------"};
19.
20.           char filename[] = "c:\\Temp\\testnine.txt";
21.           int  reval = SUCCESS;
22.
23.           StrPrint(str);
24.           // create file if not exist and open for writing...
25.           // if exist, discard the previous content...
26.           if(freopen(filename, "w", stdout) == NULL)
27.           {
28.               reval = ErrorMsg(filename);
29.           }
30.           else
31.           {
32.               // call StrPrint() function...
33.               StrPrint(str);
34.               // close the standard output...
35.               fclose(stdout);
36.            }
37.           return reval;
38.       }
39.
40.       // StrPrint() function definition
41.       void  StrPrint(char  **str)
42.       {
43.           int  i;
44.           for(i=0; i<STR_NUM; i++)
45.               // to standard output-screen/console...
46.               printf("%s\n", str[i]);
47.           // system("pause");
48.       }
49.
50.       // ErrorMsg() function definition
51.       int  ErrorMsg(char  *str)
52.       {
53.           printf("Problem, cannot open %s.\n", str);
54.           return  FAIL;
55.       }
```

**55 lines: Output:**

---




- Notice that the last line in the output is NULL, why? Because NULL is appended at the end of the string. We enumerate STR_NUM = 6, but there are only 5 lines of text, if you don't want to see the NULL, change STR_NUM = 5.
- The purpose of this program is to save a paragraph, consist of five lines of text, into a text file, **testnine.txt**. We call the printf() function instead of the fprintf() function or other disk I/O functions after we redirect the default stream, stdout, of the printf() function to point to the text file.
- The function that actually does the writing is called StrPrint(), which invoke the C function printf() to send out formatted character strings to the output stream.
- In main() function, we call the StrPrint() function in line 33 before we redirect stdout to the **testnine.txt** file. The paragraph is printed on the screen because the printf() function automatically sends out the paragraph to stdout that directs to the screen by default.
- Then in line 26, we redirect stdout to the **testnine.txt** text file by calling the freopen() function. The "w" is used as the mode that indicates to open the text file for writing.
- If freopen() is successful, we then call the StrPrint() function in line 33. However, this time, the StrPrint() function writes the paragraph into the opened text file, **testnine.txt**. The reason is that stdout is now associated with the text file, not the screen.
- There is a set of low-level I/O functions, such as open(), create(), close(), read(), write(), lseek() and tell() that you may still see them in some platform-dependent C programs.

## 9.9 File Management Functions

- It refers to dealing with existing files, not reading or writing to them, but renaming, deleting and copying them. Normally the file management functions are provided in the standard library function. Again, do not reinvent the wheels : o).

### 9.9.1 Deleting A File

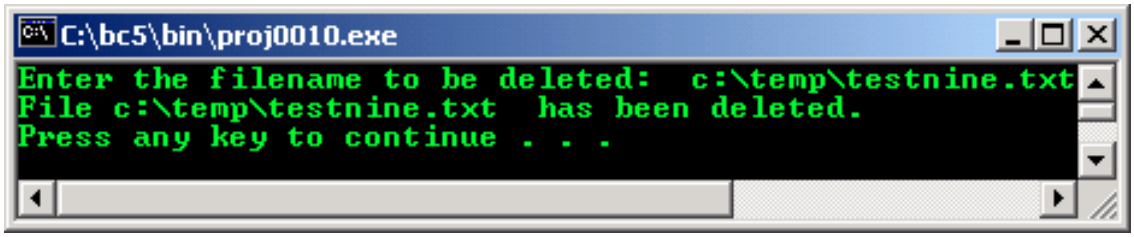
- We use function remove() to delete a file. Its prototype is in stdio.h file and the prototype is as follows:

```
int    remove(const    char    *filename);
```

- The variable filename is a pointer to the name of the file to be deleted. The specified file must not be opened. If the file exists, it is deleted (just as if the del in DOS and rm command in UNIX), and remove() return 0.
- If the file doesn't exist, if it's read only, if you don't have sufficient access rights or permission, or if some other error occurs, remove() return –1. Be careful if you remove a file, it is gone forever.
- Let try a program example.

```
1.      // demonstrates the remove() function
2.      #include <stdio.h>
3.      #include <stdlib.h>
4.
5.      void main()
6.      {
7.          // declare an array to store file name...
8.          char  filename[80];
9.
10.        printf("Enter the filename to be deleted:  ");
11.        gets(filename);
12.
13.        // check any error...
14.        if(remove(filename) == 0)
15.          printf("File %s  has been deleted.\n", filename);
16.        else
17.          fprintf(stderr, "Error deleting file %s.\n", filename);
18.        // system("pause");
19.    }
```

**19 lines: Output:**




- This program prompts the user on line 10 for the file name to be deleted. Line 14 then calls remove() to delete the entered file. If the return value is 0, the file was removed, and a message is displayed stating this fact. If the return value is not zero, an error occurred, and the file was not removed.

### 9.9.2 Renaming A File

- The rename() function changes the name of an existing disk file. The function prototype, in stdio.h, is as follows:

```
int    rename(const    char    *oldname, const    char    *newname);
```

- Both names must refer to the same disk drive; you can't rename a file to a different disk drive means if the old name is in drive **C:\test.txt**, you can't rename it to **D:\testnew.txt**. The function rename() returns 0 on success, or –1 if an error occurs. Errors can be caused by the following conditions (among others):

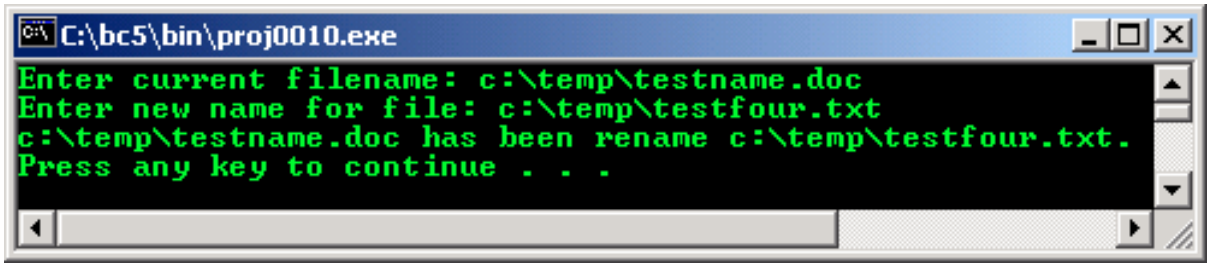
    1. The file oldname does not exist.
    2. A file with the name newname already exists.
    3. You try to rename to another disk.

- Let take a look at the following program example.

```
1.      // using rename() to change a filename
2.      #include <stdio.h>
3.      #include <stdlib.h>
4.
5.      void main()
6.      {
7.          char  oldname[80], newname[80];
8.
9.          printf("Enter current filename: ");
10.         gets(oldname);
11.         printf("Enter new name for file: ");
12.         gets(newname);
13.
14.         if(rename(oldname, newname) == 0)
15.         {
16.             printf("%s has been rename %s.\n", oldname, newname);
17.         }
18.         else
19.         {
20.             fprintf(stderr, "An error has occurred renaming %s.\n", oldname);
21.         }
22.         // system("pause");
23.     }
```

**23 lines: Output:**

---




- This program example, with only 23 lines of code, replaces an operating system rename command, and it's a much friendlier function. Line 9 prompts for the name of the file to be renamed. Line 11 prompts for the new filename.

- The if statement checks to ensure that the renaming of the file was carried out correctly. If so, line 16 prints an affirmative message, otherwise, line 20 prints a message stating that there was an error.

### 9.7.2 Copying A File

- Copying a file performs an exact duplicate with a different name (or with the same name but in a different drive or directory). There are no library functions; you have to write your own.
- The steps are:

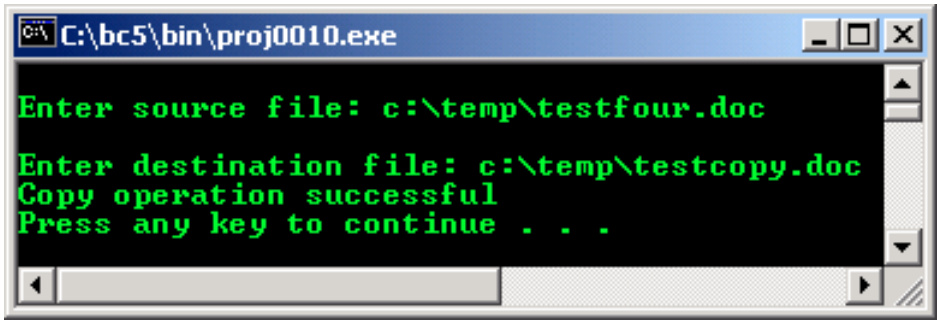
1. Open the source file for reading in binary mode, using binary mode ensures that the function can copy all sorts of content, not just texts.
2. Open the destination file for writing in binary mode.
3. Read a character from the source file. When a file is first opened, the pointer is at the start of the file, so there is no need to position the file pointer explicitly.
4. If the function feof() indicates that you're reached the end of the source file, you're done and can close both files and return to the calling program.
5. If you haven't reached end-of-file, write the character to the destination file, and then loop back to step 3.

- Let try a program example.

```
1.  // copying a file
2.  #include <stdio.h>
3.  #include <stdlib.h>
4.
5.  int  file_copy(char   *oldname, char   *newname);
6.
7.  void  main()
8.  {
9.      char  source[80], destination[80];
10.
11.     // get the source and destination names
12.     printf("\nEnter source file: ");
13.     gets(source);
14.     printf("\nEnter destination file: ");
15.     gets(destination);
16.
17.     if(file_copy(source, destination) == 0)
18.         puts("Copy operation successful");
19.     else
20.         fprintf(stderr, "Error during copy operation");
21.     // system("pause");
22. }
23.
24. int  file_copy(char *oldname, char  *newname)
25. {
```

```
26.         FILE  *fold, *fnew;
27.         int  c;
28.
29.         // open the source file for reading in binary mode
30.         if((fold = fopen(oldname, "rb")) == NULL)
31.            return  -1;
32.         // open the destination file for writing in binary mode
33.         if((fnew = fopen(newname, "wb" )) == NULL)
34.         {
35.            fclose(fold);
36.            return  -1;
37.         }
38.
39.         // read one byte at a time from the source, if end of file
40.         // has not been reached, write the byte to the destination
41.         while(1)
42.         {
43.            c  =  fgetc(fold);
44.
45.            if(!feof(fold))
46.               fputc(c, fnew);
47.            else
48.               break;
49.         }
50.         fclose(fnew);
51.         fclose(fold);
52.         return  0;
53.      }
```

**53 lines: Output:**




- The file_copy() function let you copy anything from a small text file to a huge program file. But for this program, if the destination file already exists, the function overwrites it without asking.
- Lines 24 through 37 create a copy function. Line 30 open the source file, pointed by fold pointer, in binary read mode.
- Line 33 open the destination file, pointed by fnew pointer, in binary write mode. Line 35 closes the source file if there is an error opening the destination file. The while loop does the actual copying of the file. Line 43 gets a character from the source file, pointed by fold pointer assign to the variable c.
- Line 45 tests to see whether the end-of-line marker was read. If the end of the file has been reached, a break statement is executed in order to get out of the

while loop in line 48.

- If the end of the file has not been reached, the character is written to the destination file, pointed by fnew pointer in line 46.
- For your information the C++ file I/O is discussed in Module 19.
- The following is a previous C program example, read and write files under the current working directory using gcc. Create 2 files named **testhree.txt** and **testfour.txt** under the current working directory and save some texts in the **testfour.txt**.

```
/***************readline.c************
/* reading and writing one line at a time*/
#include <stdio.h>
#include <stdlib.h>

enum  {SUCCESS, FAIL, MAX_LEN = 100};

/* a function prototype for read and writes by line... */
void LineReadWrite(FILE *fin, FILE *fout);

int main(void)
{
    FILE *fptr1, *fptr2;
    /* file testhree.txt is located at current directory.
      you can put this file at any location provided
      you provide the full path, same for testfour.txt */

    char filename1[ ] = "testhree.txt";
    char filename2[ ] = "testfour.txt";
    char reval = SUCCESS;

    /* test opening testhree.txt file for writing, if fail... */
    if((fptr1 = fopen(filename1,"w")) == NULL)
    {
        printf("Problem, cannot open %s for writing.\n", filename1);
        reval = FAIL;
    }

    /* test opening testfour.txt file for reading, if fail... */
    else if((fptr2=fopen(filename2, "r"))==NULL)
    {
        printf("Problem, cannot open %s for reading.\n", filename2);
        reval = FAIL;
    }

    /* if opening fro writing and reading successful, do... */
    else
    {
        /* function call for read and write, line by line... */
        LineReadWrite(fptr2, fptr1);
        /* close both files stream... */
```

```
        if(fclose(fptr1)==0)
            printf("%s successfully closed.\n", filename1);
        if(fclose(fptr2)==0)
            printf("%s successfully closed.\n", filename2);
    }
    return reval;
  }

    /* function definition for line read, write. */
    void LineReadWrite(FILE *fin, FILE *fout)
    {
        /* local variable... */
        char buff[MAX_LEN];
        while(fgets(buff, MAX_LEN, fin) !=NULL)
        {
              /* write to file... */
              fputs(buff, fout);
              /* write to screen... */
              printf("%s", buff);
        }
}
```

```
[bodo@bakawali ~]$ gcc readline.c -o readline
[bodo@bakawali ~]$ ./readline

------------------LINUX LOR!-----------------------
------------FEDORA 3, gcc x.x.x--------------------
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-------------------

testhree.txt successfully closed.
testfour.txt successfully closed.
```

C & C++ programming tutorials

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

1. 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).
2. Check the best selling C / C++ books at Amazon.com.
3. The source code for this Module is: C file input/output program source codes.
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.

**|< C File I/O 3 | Main | C File I/O 5 >| Site Index | Download |**

**C File Input/Output: Part 1 | Part 2 | Part 3 | Part 4 | Part 5**