To:
Tenouk

MODULE 9a_1 - MORE C FILE INPUT/OUTPUT 3

MODULE 19 - C++ FILE I/O

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

My Training Period: xx hours

9.7 Random Access To Disk Files

- Before this you have learned how to read or write data sequentially. In many cases, however, you may need to access particular data somewhere in the middle of a disk file.
- Random access is another way to read and write data to disk file. Specific file elements can be accessed in random order.
- There are two C I/O functions, fseek() and ftell(), that are designed to deal with random access.
- You can use the fseek() function to move the file position indicator to the spot you want to access in a file. The prototype for the fseek() function is:

```
int    fseek(FILE    *stream, long    offset, int    whence);
```

- stream is the file pointer with an opened file. offset indicates the number of bytes from a fixed position, specified by whence, that can have one of the following integral values represented by SEEK_SET, SEEK_CUR and SEEK_END.
- If it is successful, the fseek() function return 0, otherwise the function returns a nonzero value.
- whence provides the offset bytes from the file location. whence must be one of the values 0, 1, or 2 which represent three symbolic constants (defined in stdio.h) as follows:

Constants	whence	File location
SEEK_SET	0	File beginning
SEEK_CUR	1	Current file pointer position
SEEK_END	2	End of file

Table 9.10: offset bytes

- If SEEK_SET is chosen as the third argument to the fseek() function, the offset is counted from the beginning of the file and the value of the offset is greater than or equal to zero.
- If however, SEEK_END is picked up, then the offset starts from the end of the file, the value of the offset should be negative.
- When SEEK_CUR is passed to the fseek() function, the offset is calculated from the current value of the file position indicator.
- You can obtain the value of the current position indicator by calling the ftell() function. The prototype for the ftell() function is,

```
long    ftell(FILE    *stream);
```

- stream is the file pointer associated with an opened file. The ftell() function returns the current value of the file position indicator.
- The value returned by the ftell() function represents the number of bytes from the beginning of the file to the current position pointed to by the file position indictor.
- If the ftell() function fails, it returns –1L (that is, a long value of minus 1). Let explore the program example. Create and make sure text file named **tesseven.txt** is located in the **C:\Temp** folder before you can execute the program. The contents of the **tesseven.txt** is,

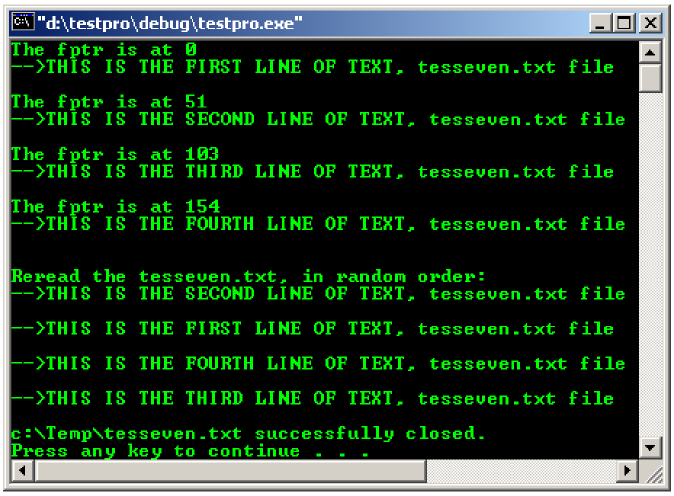
```
THIS IS THE FIRST LINE OF TEXT, tesseven.txt file
THIS IS THE SECOND LINE OF TEXT, tesseven.txt file
THIS IS THE THIRD LINE OF TEXT, tesseven.txt file
THIS IS THE FOURTH LINE OF TEXT, tesseven.txt file
```

The content of tesseven.txt file

```
// random access to a file
#include <stdio.h>
#include <stdlib.h>

enum  {SUCCESS, FAIL, MAX_LEN = 120};

// function prototypes, seek the file position indicator
void  PtrSeek(FILE  *fptr);
// function prototype, tell the file position indicator...
long  PtrTell(FILE  *fptr);
// function prototype read and writes...
void  DataRead(FILE  *fptr);
int   ErrorMsg(char  *str);

int main(void)
{
     FILE  *fptr;
     char filename[] = "c:\\Temp\\tesseven.txt";
```

```
    int  reval  = SUCCESS;

    // if there is some error opening file for reading...
    if((fptr = fopen(filename, "r")) == NULL)
    {
        reval = ErrorMsg(filename);
    }
    // if opening is successful..
    else
    {
        // PtrSeek() function call...
        PtrSeek(fptr);
        //close the file stream...
        if(fclose(fptr)==0)
            printf("%s successfully closed.\n", filename);
    }
    // for Borland...
    // system("pause");
    return  reval;
}

// PtrSeek() function definition
void  PtrSeek(FILE  *fptr)
{
    long   offset1, offset2, offset3, offset4;

    offset1 = PtrTell(fptr);
    DataRead(fptr);
    offset2 = PtrTell(fptr);
    DataRead(fptr);
    offset3 = PtrTell(fptr);
    DataRead(fptr);
    offset4 = PtrTell(fptr);
    DataRead(fptr);

    printf("\nReread the tesseven.txt, in random order:\n");
    // re-read the 2nd line of the tesseven.txt
    fseek(fptr, offset2, SEEK_SET);
    DataRead(fptr);
    // re-read the 1st line of the tesseven.txt
    fseek(fptr, offset1, SEEK_SET);
    DataRead(fptr);
    // re-read the 4th line of the tesseven.txt
    fseek(fptr, offset4, SEEK_SET);
    DataRead(fptr);
    // re-read the 3rd line of the tesseven.txt
    fseek(fptr, offset3, SEEK_SET);
    DataRead(fptr);
}

// PtrTell() function definition
long   PtrTell(FILE  *fptr)
{
    long  reval;
    // tell the fptr position...
    reval = ftell(fptr);
    printf("The fptr is at %ld\n", reval);
    return  reval;
}

// DataRead() function definition
void  DataRead(FILE  *fptr)
{
    char  buff[MAX_LEN];
    // reading line of text at the fptr position...
    fgets(buff, MAX_LEN, fptr);
    // and display the text...
    printf("-->%s\n", buff);
}

// error message function definition
int  ErrorMsg(char *str)
{
    // display this error message...
    printf("Problem, cannot open %s.\n", str);
    return FAIL;
}
```

95 lines: Output:



```
The fptr is at 0
-->THIS IS THE FIRST LINE OF TEXT, tesseven.txt file

The fptr is at 51
-->THIS IS THE SECOND LINE OF TEXT, tesseven.txt file

The fptr is at 103
-->THIS IS THE THIRD LINE OF TEXT, tesseven.txt file

The fptr is at 154
-->THIS IS THE FOURTH LINE OF TEXT, tesseven.txt file


Reread the tesseven.txt, in random order:
-->THIS IS THE SECOND LINE OF TEXT, tesseven.txt file

-->THIS IS THE FIRST LINE OF TEXT, tesseven.txt file

-->THIS IS THE FOURTH LINE OF TEXT, tesseven.txt file

-->THIS IS THE THIRD LINE OF TEXT, tesseven.txt file

c:\Temp\tesseven.txt successfully closed.
Press any key to continue . . .
```

- We try to open the **tesseven.txt** file for reading by calling the fopen() function. If successful, we invoke the PtrSeek() function with the fptr file pointer as the argument in line 30.

```
PtrSeek(fptr);
```

- The definition of our first function PtrSeek() is shown in lines 41-67. The statement in line 45 obtains the original value of the fptr file pointer by calling another function, PtrTell(), which is defined in lines 70–77.
- The PtrTell() function can find and print out the value of the file position indicator with the help of the ftell() function.
- The third function, DataRead() is called to read one line of characters from the opened file and print out the line of characters on the screen. Line 47 gets the new value of the fptr file position indicator right after the reading and assigns the value to another long variable, offset2.
- Then the DataRead() function in line 48 reads the second line of characters from the opened file. Line 49 obtains the value of the file position indicator that points to the first byte of the third line and assigns the value to the third long variable offset3 and so on for the fourth line of text.
- Line 50 calls the DataRead() function to read the third line and print it out on the screen.
- From the first portion of the output, you can see the four different values of the file position indicator at four different positions, and the four lines of texts. The four values of the file position indicator are saved by offset1, offset2, offset3 and offset4 respectively.
- Then, we read the lines of text randomly, one line at a time. Firstly read the second line, then the first line, fourth and finally the third one.
- C function, called rewind(), can be used to rewind the file position indicator. The prototype for the rewind() function is:

```
void    rewind(FILE    *stream);
```

- Here, stream is the file pointer associated with an opened file. No value is returned by rewind() function. In fact the following statement of rewind() function:

```
rewind(fptr);
```

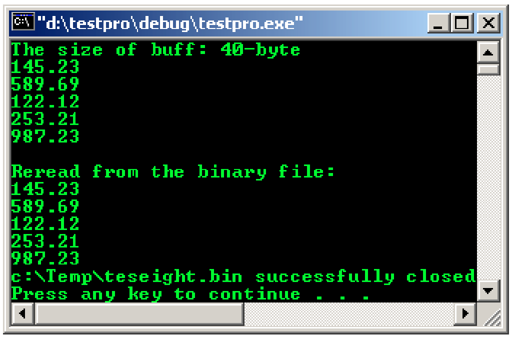
- Is equivalent to this:

```
(void) fseek(fptr, 0L, SEEK_SET);
```

- The void data type is cast to the fseek() function because the rewind() function does not return a value. Study the following program example.
- This program also contains example of reading and writing binary data. We create and open the **teseight.bin** file for writing.

```
1.     // reading, writing, rewind and binary data
2.     #include <stdio.h>
3.     #include <stdlib.h>
4.
5.     enum  {SUCCESS, FAIL, MAX_NUM = 5};
6.
7.     // function prototypes...
8.     void  DataWrite(FILE  *fout);
9.     void  DataRead(FILE  *fin);
10.    int   ErrorMsg(char  *str);
11.
12.    int main(void)
13.    {
14.        FILE  *fptr;
15.        // binary type files...
16.        char  filename[] = "c:\\Temp\\teseight.bin";
17.        int  reval = SUCCESS;
18.
19.         // test for creating, opening binary file for writing...
20.         if((fptr = fopen(filename, "wb+")) == NULL)
21.         {
22.             reval   = ErrorMsg(filename);
23.         }
24.         else
25.         {
26.             // write data into file teseight.bin
27.             DataWrite(fptr);
28.             // reset the file position indicator...
29.             rewind(fptr);
30.             // read data...
31.             DataRead(fptr);
32.             // close the file stream...
33.             if(fclose(fptr)==0)
34.              printf("%s successfully closed\n", filename);
35.         }
36.         // for Borland
37.         // system("pause");
38.         return  reval;
39.     }
40.
41.     // DataWrite() function definition
42.     void  DataWrite(FILE  *fout)
43.     {
44.         int   i;
45.         double   buff[MAX_NUM] = { 145.23, 589.69, 122.12, 253.21, 987.234};
46.
47.         printf("The size of buff: %d-byte\n", sizeof(buff));
48.         for(i=0; i<MAX_NUM; i++)
49.         {
50.             printf("%5.2f\n", buff[i]);
51.             fwrite(&buff[i], sizeof(double), 1, fout);
52.         }
53.     }
54.
55.     // DataRead() function definition
56.     void  DataRead(FILE  *fin)
57.     {
58.         int   i;
59.         double  x;
60.
61.         printf("\nReread from the binary file:\n");
62.         for(i=0; i<MAX_NUM; i++)
63.         {
64.             fread(&x, sizeof(double), (size_t)1, fin);
65.             printf("%5.2f\n", x);
66.         }
67.     }
68.
69.     // ErrorMsg() function definition
70.     int  ErrorMsg(char  *str)
71.     {
72.         printf("Cannot open %s.\n", str);
73.         return  FAIL;
74.     }
```

74 lines, Output:



```
The size of buff: 40-byte
145.23
589.69
122.12
253.21
987.23

Reread from the binary file:
145.23
589.69
122.12
253.21
987.23
c:\Temp\teseight.bin successfully closed
Press any key to continue . . .
```

- This program writes five values of the double data type into a binary file named **teseight.bin** and then rewind the file position indicator and re-read the five double values from the binary file.
- The two functions, DataWrite() and DataRead(), that perform the writing and reading, declared in lines 8 and 9. The enum names, SUCCESS, FAIL, and MAX_NUM, are defined in line 5 with values 0, 1, and 5 respectively.
- The statement in line 20, tries to create and open a binary file called **teseight.bin** for both reading and writing.
- If the fopen() function is successful, the DataWrite() function is called in line 27 to write four double data items, into the opened binary file, according to the definition of the DataWrite () function.
- The fwrite() function in line 51 does the writing. Right after the execution of the DataWrite () function, the file position indicator is reset to the beginning of the binary file by calling the rewind() function in line 29 because we want to re read all five double data items written to the file.

- The fread() function is used to perform the reading operation. The output from running the program shows the five double data items before the writing and after the reading as well.
- As you learned, two C library functions scanf()/scanf_s() and printf()/printf_s() can be used to read or write formatted data through the standard I/O (that is, stdin and stdout). For C disk file I/O functions, there are two equivalent functions; fscanf()/fscanf_s() and fprintf()/fprintf_s() functions allow the programmer to specify I/O streams.

- The prototype for the fscan() function is:

```
int   fscanf(FILE   *stream, const   char   *format,...);
```

- stream is the file pointer associated with an opened file. format, which usage is similar to the scanf() function, is a char pointer pointing to a string that contains the format specifiers. If successful, the fscan() function returns the number of data items read. Otherwise, the function returns EOF.
- The prototype for the fprintf() function is:

```
int   fprintf(FILE   *stream, const   char   *format, …);
```

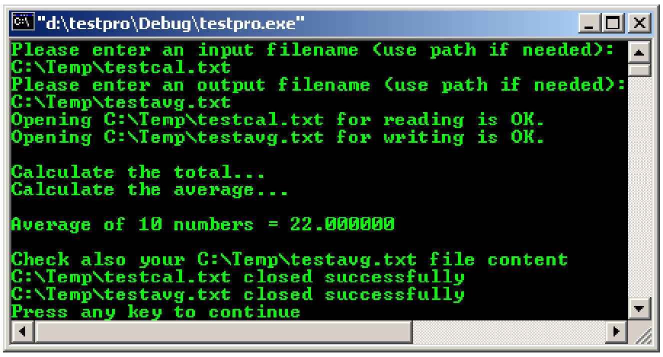
- Here, stream is the file pointer associated with an opened file. format, is similar as in the printf() function, is a char pointer pointing to a string that contains the format specifiers.
- If successful, the fprintf() function returns the number of formatted expressions. Otherwise, the function returns a negative value.
- Let try a program example. Firstly create **testcal.txt** file with the following data and save it.

```
23 12 33 10 4 6 44 31 7 50
```

- Then create another text file named **testavg.txt** for writing the average value computed from data read from **testcal.txt** file. Then compile and run the following program.

```
/* C Program to calculate the average of a list of numbers. */
/* calculate the total from one file, output the average into another file */
#include <stdio.h>
/* for exit() */
#include <stdlib.h>

int main(void)
{
    int value, total = 0, count = 0;
    /* fileptrIn and fileptrOut are variables of type (FILE *) */
    FILE * fileptrIn, * fileptrOut;
    char filenameIn[100], filenameOut[100];
    printf("Please enter an input filename (use path if needed):\n");
    scanf("%s", filenameIn);
    printf("Please enter an output filename (use path if needed):\n");
    scanf("%s", filenameOut);
    /* open files for reading, "r" and writing, "w" */
    if((fileptrIn = fopen(filenameIn, "r")) == NULL)
    {
        printf("Error opening %s for reading.\n", filenameIn);
        exit (1);
    }
    else
        printf("Opening %s for reading is OK.\n", filenameIn);
    if((fileptrOut = fopen(filenameOut, "w")) == NULL)
    {
        printf("Error opening %s for writing.\n", filenameOut);
        exit (1);
    }
    else
        printf("Opening %s for writing is OK.\n", filenameOut);
    /* fscanf */
    printf("\nCalculate the total...\n");
    while(EOF != fscanf(fileptrIn, "%i", &value))
    {
        total += value;
        ++count;
    }/* end of while loop */
    /* write the average value to the file. */
    /* fprintf */
    printf("Calculate the average...\n\n");
    fprintf(fileptrOut, "Average of %i numbers = %f \n", count, total/(double)count);
    printf("Average of %i numbers = %f \n\n", count, total/(double)count);
    printf("Check also your %s file content\n", filenameOut);
    if(fclose(fileptrIn) == 0)
        printf("%s closed successfully\n", filenameIn);
    if(fclose(fileptrOut) == 0)
        printf("%s closed successfully\n", filenameOut);
    return 0;
}
```

Output:



```
Please enter an input filename (use path if needed):
C:\Temp\testcal.txt
Please enter an output filename (use path if needed):
C:\Temp\testavg.txt
Opening C:\Temp\testcal.txt for reading is OK.
Opening C:\Temp\testavg.txt for writing is OK.

Calculate the total...
Calculate the average...

Average of 10 numbers = 22.000000

Check also your C:\Temp\testavg.txt file content
C:\Temp\testcal.txt closed successfully
C:\Temp\testavg.txt closed successfully
Press any key to continue
```

C & C++ programming tutorials

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.

|< C File I/O 2 | Main | C File I/O 4 >| Site Index | Download |

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