Introduction to Scientific and Engineering Computation (BIL 104E)

Lecture 13

Reading and Writing with Files
&
Special File Functions

Reading and Writing with Files

- **Files and streams**
- **Opening a file with fopen()**
- **Closing a file with fclose()**
- **The fgetc() and fputc() functions**
- **The fgets() and fputs() functions**
- **The fread() and fwrite() functions**
- **The feof() function**

Reading and Writing with Files : What Is a File?

A file represents a concrete device with which you want to exchange information.

Working with a file, you have to follow necessary steps:

Step 1) You have to open the file.

Step 2) You can start to exchange information with the file opened.

Step 3) You need to close the opened file after you finish to work with it.

Reading and Writing with Files: What Is a Stream?

The data flow you transfer from your program to a file, or vice versa, is called a stream, which is a series of bytes.

There are two formats of streams.
The **text stream** consists of a sequence of characters. each line of characters in a text stream may be terminated by a newline character (‘\n’).

The **binary stream** is a series of bytes, for example, the content of an executable program file.
Binary streams are primarily used for nontextual data, where the exact contents of the data are maintained without regard to appearance.

Reading and Writing with Files: Buffered I/O?

The **buffer** is a memory area which is used to store temporarily output or input data.

With the help of buffers, the operating system can improve efficiency by reducing the number of accesses **to I/O** devices.

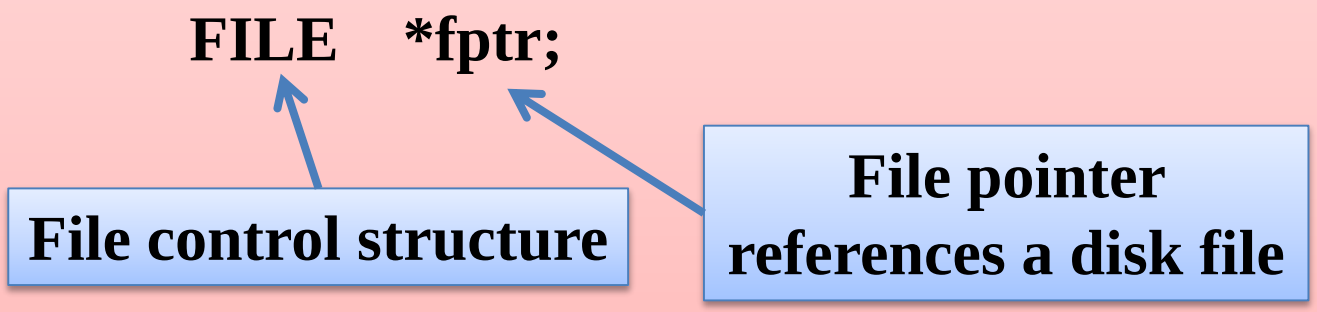
Access to **I/O** device is generally much slower than direct access to memory.

By default, all **I/O streams** are buffered. Buffered **I/O** is also called high-level **I/O**, whereas unbuffered **I/O** (directly to the device) is called low-level I/O.

Reading and Writing with Files: Pointers of FILE

The **FILE structure** is the file control structure defined in the header file **stdio.h**.

For instance:



A **file pointer** is used by a stream to conduct the operation of the **I/O** functions.

One member of the **FILE structure** is **file position indicator**, which points to the position in a file to be read from or written to.

Reading and Writing with Files: Opening a File

The C I/O function **fopen()** opens a file and associates a stream with the opened file.

The syntax for the **fopen()** function is

```
#include <stdio.h>
    FILE *fopen(const char *filename, const char *mode);
```

filename is a char pointer that references a string containing a filename. The filename is given to the file that is about to be opened by the **fopen()** function.

mode points to another string that specifies the way to open the file.

The **fopen()** function returns a pointer of type FILE.

If an error occurs during the procedure to open a file, the **fopen()** function returns a null pointer.

Reading and Writing with Files: Opening a File

The mode parameter is made by a combination of the characters r (read), w (write), b (binary), a (append), and + (update).

Does the file already exists?	Append Mode (a)	Writing Mode (w)	Reading Mode (r)
YES	New data will be added to the end of existing file.	Existing file will be discarded and new file will be created	Data will be read from existing file.
No	New data will be added to new file created.	New file will be created	Fail message

Using the + character makes both reading and writing possible.

Reading and Writing with Files: Opening a File

The following list shows the possible ways to open a file by various strings of modes:

- **“r”** opens an existing text file for reading.
- **“w”** creates a text file for writing.
- **“a”** opens an existing text file for appending.
- **“r+”** opens an existing text file for reading or writing.
- **“w+”** creates a text file for reading or writing.
- **“a+”** opens or create a text file for appending.
- **“rb”** opens an existing binary file for reading.
- **“wb”** creates a binary file for writing.
- **“ab”** opens an existing binary file for appending.
- **“r+b”** opens an existing binary file for reading or writing.
- **“w+b”** creates a binary file for reading or writing.
- **“a+b”** opens or creates a binary file for appending.

Reading and Writing with Files: Opening a File

The following statements try to open a file called test.txt:

```
FILE *fptr;

if (( fptr = fopen("test.txt", "r"))  ==  NULL){
        printf("Cannot open test.txt file.\n");
        exit(1);
}
```

"r" is used to indicate that a text file is about to be opened for reading only.

If an error occurs when the **fopen()** function tries to open the file, the function returns **a null pointer**.
Then the program is aborted by calling the **exit()** function with a nonzero value.

Reading and Writing with Files: Closing a File

After a disk file is read, written, or appended with some new data, you have to disassociate the file from a specified stream by calling the **fclose()** function.

The syntax for the **fclose()** function is

```
#include <stdio.h>
    int fclose(FILE *stream);
```

stream is a file pointer that is associated with a stream to the opened file.

If **fclose()** closes a file successfully, it returns **0**. Otherwise, the function returns **EOF**.

Normally, the **fclose()** function fails only when the disk is removed before the function is called or there is no more space left on the disk.

Reading and Writing with Files: Closing a File

```
#include <stdio.h>
enum {SUCCESS, FAIL};
main(void)
{
   FILE *fptr;
   char filename[]= "haiku.txt";
   int reval = SUCCESS;

   if ((fptr = fopen(filename, "r")) == NULL){
     printf("Cannot open %s.\n", filename);
     reval = FAIL;
   }
   else {
       printf("The value of fptr: 0x%p\n", fptr);
       printf("Ready to close the file.");
       fclose(fptr);
   }
   getchar();

   return reval;
}
```

```
The value of fptr: 0x760609B8
Ready to close the file.
```

Reading and Writing Disk Files:One Character at a Time

Among the C **I/O** functions, there is a pair of functions, **fgetc()** and **fputc()**, that can be used to read from or write to a disk file one character at a time.

The syntax for the fgetc() function is

```
#include <stdio.h>
        int fgetc(FILE *stream);
```

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.

EOF is returned if **fgetc()** encounters the end of the file, or if there is an error.

Reading and Writing Disk Files:One Character at a Time

The syntax for the fputc() function is

#include <stdio.h>

int fputc(int c , FILE *stream);

c is an int value that represents a character.

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.

Reading and Writing Disk Files:One Character at a Time

```
#include <stdio.h>
enum {SUCCESS, FAIL};
void CharReadWrite(FILE *fin, FILE *fout);
main(void)
   FILE *fptr1, *fptr2;
   char filename1[]= "outhaiku.txt";
   char filename2[]= "haiku.txt";
   int reval = SUCCESS;
   if ((fptr1 = fopen(filename1, "w")) == NULL){
     printf("Cannot open %s.\n", filename1);
     reval = FAIL;
   }
   else
       if ((fptr2 = fopen(filename2, "r")) == NULL){
          printf("Cannot open %s.\n", filename2);
          reval = FAIL;
       }
       else {
           CharReadWrite(fptr2, fptr1);
           fclose(fptr1);
           fclose(fptr2);
       }
   getchar();
   return reval;
}
```

```
/* function definition */
void CharReadWrite(FILE *fin, FILE
*fout)
{
   int c;

   while ((c=fgetc(fin)) != EOF){
        fputc(c, fout); /* write to a file */
        putchar(c); /* put the character to the
screen */
   }
}
```

Leading me along
my shadow goes back home
from looking at the moon.
--- Sodo
(1641-1716)

A storm wind blows
out from among the grasses
the full moon grows.
--- Chora
(1729-1781)

Reading and Writing Disk Files: One Line at a Time

There is a pair of C **I/O** functions, **fgets()** and **fputs()**, that allows you can also read or write one character line at time.

The syntax for the **fgets()** function is

```
#include <stdio.h>
        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 pointer **s**.
If **EOF** is encountered, the **fgets()** function returns a null pointer.

If an error occurs, the function returns a null pointer

Reading and Writing Disk Files:One Line at a Time

The syntax for the **fputs()** function is

```
#include <stdio.h>
        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 by the **fputs()** function.

If it fails, the **fputs()** function returns a nonzero value; otherwise, it returns zero.

Reading and Writing Disk Files:One Line at a Time

```
#include <stdio.h>
enum {SUCCESS, FAIL, MAX_LEN = 81};
void LineReadWrite(FILE *fin, FILE *fout);
main(void)
{   FILE *fptr1, *fptr2;
    char filename1[]= "outhaiku.txt";
    char filename2[]= "haiku.txt";
    int reval = SUCCESS;
    if ((fptr1 = fopen(filename1, "w")) == NULL){
      printf("Cannot open %s for writing.\n", filename1);
       reval = FAIL;
    }
    else
        if ((fptr2 = fopen(filename2, "r")) == NULL){
           printf("Cannot open %s for reading.\n", filename2);
           reval = FAIL;
        }
        else {
            LineReadWrite(fptr2, fptr1);
            fclose(fptr1);
            fclose(fptr2);
        }
     getchar();
     return reval;
}
```

```
/* function definition */
void    LineReadWrite(FILE    *fin,    FILE
*fout)
{
    char buff[MAX_LEN];

  while (fgets(buff, MAX_LEN, fin) != NULL){
          fputs(buff, fout);
          printf("%s", buff);
    }
}
```

Leading me along
my shadow goes back home
from looking at the moon.
--- Sodo
(1641-1716)

A storm wind blows
out from among the grasses
the full moon grows.
--- Chora
(1729-1781)

Reading and Writing Disk Files:One Block at a Time

In C, there are two I/O functions, **fread()** and **fwrite()**, that can be used to perform block I/O operations.

The syntax for the **fread()** function is

```
#include <stdio.h>
    size_t fread(void *ptr, size_t size, size_t n, FILE *stream);
```

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 read. stream is a file pointer that is associated with the opened file for reading.

The **fread()** function returns the number of elements actually read.

Reading and Writing Disk Files:One Block at a Time

The syntax for the **fwrite()** function is

#include <stdio.h>

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.

Reading and Writing Disk Files: feof() function

A function called feof() can be used to determine when the end of a file is encountered.

This function is more useful when you're reading a binary file because the values of some bytes may be equal to the value of **EOF**.

The syntax for the **feof()** function is

```
#include <stdio.h>
        int feof(FILE *stream);
```

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.

Reading and Writing Disk Files: feof() function

```
#include <stdio.h>
enum {SUCCESS, FAIL, MAX_LEN = 80};
void BlockReadWrite(FILE *fin, FILE *fout);
int ErrorMsg(char *str);
main(void)
{
   FILE *fptr1, *fptr2;
   char filename1[]= "outhaiku.txt";
   char filename2[]= "haiku.txt";
   int reval = SUCCESS;
   if ((fptr1 = fopen(filename1, "w")) == NULL){
     reval = ErrorMsg(filename1);
   }
   else
       if ((fptr2 = fopen(filename2, "r")) == NULL){
          reval = ErrorMsg(filename2);
       }
       else {
           BlockReadWrite(fptr2, fptr1);
           fclose(fptr1);
           fclose(fptr2);
       }
   getchar();
   return reval;
}
```

Reading and Writing Disk Files: feof() function

```
/* function definition */
void BlockReadWrite(FILE *fin, FILE *fout)
{
   int num;
   char buff[MAX_LEN + 1];

   while (!feof(fin)){
        num = fread(buff, sizeof(char), MAX_LEN, fin);
        buff[num * sizeof(char)] = '\0'; /* append a null character */
        printf("%s", buff);
        fwrite(buff, sizeof(char), num, fout);
   }
}
/* function definition */
int ErrorMsg(char *str)
{
   printf("Cannot open %s.\n", str);
   return FAIL;

}
```

Reading and Writing Disk Files: feof() function

Computer Screen

Leading me along
my shadow goes back home
from looking at the moon.
--- Sodo
(1641-1716)

A storm wind blows
out from among the grasses
the full moon grows.
--- Chora
(1729-1781)

Special File Functions: fseek() and ftell()

One of the members in the **FILE structure** is called the **file position indicator**. The file position indicator has to point to the desired position in a file before data can be read from or written to there.

You can use the **fseek()** function to move the file position indicator to the spot you want to access in a file.

Special File Functions: fseek() and ftell()

The syntax for the **fseek()** function is

#include <stdio.h>

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

stream is the file pointer associated 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 = 0**, **SEEK_CUR = 1**, and **SEEK_END =** 2.

SEEK_SET, **SEEK_CUR** and **SEEK_END** determines the file indicator pozition according to beginning of the file, current position of the file position indicator and end of the file ,respectively.

If it is successful, the **fseek()** function returns **0**; otherwise, the function returns a nonzero value.

Special File Functions: fseek() and ftell()

You can obtain the current value of the file position indicator by calling the **ftell()** function.

The syntax for the **ftell()** function is

```
#include <stdio.h>
        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 indicator.

Special File Functions: rewind()

The rewind() function is used to reset the file position indicator and put it at the beginning of a file.

The syntax for the **rewind()** function is

```
#include <stdio.h>
        void rewind(FILE *stream);
```

stream is the file pointer associated with an opened file. No value is returned by the rewind() function.

Special File Functions: fscanf() and fprintf()

The **fscanf()** and **fprintf()** functions allow the programmer to specify **I/O stre**ams other than **stdin** and **stdout**.

The syntax for the **fscanf()** function is

```
#include <stdio.h>
        int fscanf(FILE *stream, const char *format, …);
```

stream is the file pointer associated with an opened file.

format, whose usage is the same as in the **scanf()** function, is a char pointer pointing to a string that contains the format specifiers.

If successful, the **fscanf()** function returns the number of data items read. Otherwise, the function returns **EOF**.

Special File Functions: fscanf() and fprintf()

The syntax for the fprintf() function is

```
#include <stdio.h>
    int fprintf(FILE *stream, const char *format, ...);
```

stream is the file pointer associated with an opened file.

format, whose usage is the same 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.

Special File Functions: fscanf() and fprintf()

```
#include <stdio.h>
enum {SUCCESS, FAIL,
     MAX_NUM = 3,
     STR_LEN = 23};
void DataWrite(FILE *fout);
void DataRead(FILE *fin);
int ErrorMsg(char *str);
main(void)
{
FILE *fptr;
    char filename[]= "strnum.mix";
    int reval = SUCCESS;

    if ((fptr = fopen(filename, "w+")) == NULL){
       reval = ErrorMsg(filename);
    }
    else {
         DataWrite(fptr);
         rewind(fptr);
         DataRead(fptr);
         fclose(fptr);
         }
    getchar();
    return reval;
}
```

Special File Functions: fscanf() and fprintf()

```
/* function definition */
void DataWrite(FILE *fout)
{
   int i;
   char cities[MAX_NUM][STR_LEN] = {
                        "St.Louis->Houston:",
                        "Houston->Dallas:",
                        "Dallas->Philadelphia:"};
   int miles[MAX_NUM] = {
                    845,
                    243,
                    1459};

   printf("The data written:\n");
   for (i=0; i<MAX_NUM; i++){
      printf("%-23s %d miles\n", cities[i], miles[i]);
      fprintf(fout, "%s %d", cities[i], miles[i]);
   }
}
```

Special File Functions: fscanf() and fprintf()

```
/* function definition */
void DataRead(FILE *fin)
{
   int i;
   int miles;
   char cities[STR_LEN];

   printf("\nThe data read:\n");
   for (i=0; i<MAX_NUM; i++){
      fscanf(fin, "%s%d", cities, &miles);
      printf("%-23s %d miles\n", cities, miles);
   }
}
/* function definition */
int ErrorMsg(char *str)
{
   printf("Cannot open %s.\n", str);
   return FAIL;
}
```

Special File Functions: fscanf() and fprintf()

Computer Screen

```
The data written:
St.Louis->Houston:      845 miles
Houston->Dallas:        243 miles
Dallas->Philadelphia:   1459 miles

The data read:
St.Louis->Houston:      845 miles
Houston->Dallas:        243 miles
Dallas->Philadelphia:   1459 miles
```