Programming 2

Files with Sir Beets

Topics

Files : Text and Binary Files

- File Operations:
    - fopen() and fclose()
    - fwrite() and fread()
- fseek(), ftell(), and rewind()

Interfacing with a file

Name	Age	AP
Alex	10	67.9
Sam	22	90.5

Schema

Data Management ✓

```
#include <stdio.h>
void main(){
  ...
}
```

```
Alex     10 67.9 Sam     22 90.5
```

Programming II ✓

Data Files

A **data file** is any collection of data physically stored together by a computer under a unique file name in an external storage medium other than the computer’s main memory.

- ✓ External data storage permits a program to use the data without having to recreate them each time the program is run.
- ✓ Provides a basis for sharing data between programs, so the data output by one program can be input directly to another program

Typically, most computers require that the file name consists of no more than eight characters followed by an optional period and and extension of up to three characters. (System dependent)

Text Files: .txt .c .java .rtf

Binary Files: .mp4 .png .gif .exe .dat

Data File Operations

1. Data files are opened using **`fopen()`** for **reading**, **writing** or **appending**. This function connects a file's external name with an internal file name. After the file is opened, all subsequent accesses to the file require the internal file name.

   ```
   FILE *fopen(char *name, char *mode);
   ```

   Returned value:

   - Points to a structure that contains information about the file location of a buffer
   - current character position in the buffer
   - whether the file is being read or written
   - whether errors or end of file have occurred

2. **Read or Write**

   Functions involved: `fread(), fwrite()`

3. **Close**

   Function involed: `fclose()`

Writing to a File

Write (“w”): *data are written starting at the beginning of the file*

A file opened for writing creates a new file or erases any existing file having the same name as the opened file.

Append (“a”): *data are written starting at the end of the file*

A file opened for appending makes an existing file available for data to be added to the end of the file. If the file does not exist it is created.

```
size_t fwrite(
    const void *buffer,
    size_t size,
    size_t count,
    FILE *stream
);
```

fwrite returns the number of full items actually written, which may be less than count if an error occurs. Also, if an error occurs, the file-position indicator cannot be determined.

- *buffer*
  Pointer to data to be written.
- *size*
  Item size, in bytes.
- *count*
  number of *size* to be written
- *stream*
  Pointer to **FILE** structure.

Reading from a File

Read (“r”):

A file opened for reading makes an existing file’s data available for input.

Notes:

- Trying to read a file that does not exist is an error.
- Reading a file when you do not have permission also causes an error.
- When there is an error, fopen will return NULL.

```
size_t fread(
    void *buffer,
    size_t size,
    size_t count,
    FILE *stream

);
```

- *buffer*
  Storage location for data.
- *size*
  Item size in bytes.
- *count*
  Maximum number of size to be read
- *stream*
  Pointer to **FILE** structure.

fread() returns the number of full items actually read, which may be less than count if an error occurs or if the end of the file is encountered before reaching count.

If size or count is 0, **fread()** returns 0 and the buffer contents are unchanged.

Closing a File

```
int fclose(FILE *fp)
```

- ✓Breaks the connection between the file pointer and the external name that was established by fopen() freeing the file pointer for another file
- ✓Flushes the buffer for which functions are collecting output
- ✓Automatically called for each open file when a program terminates normally
- ✓Returns 0 for successfully closing a file
- ✓Returns EOF for error

When a file is closed, end-of-file (EOF) marker is automatically placed by the OS to the file

Fun Codes Writing to a File

```
#include <stdio.h>
#pragma pack(1)

typedef struct {
        char *id;
        char name[10];
        int age;
        float atk;
} ADV;

int main( void ) {
        FILE *fp;
        int numWritten;
        ADV a1 = {"0X1", "Alex", 10, 14.6};

        fp = fopen( "char_data.bin", "wb" );
        numWritten = fwrite( &a1, sizeof( ADV ), 1 , fp );
        fclose( fp );

}
```

Fun Codes Writing to a File

```
#include <stdio.h>
#pragma pack(1)

typedef struct {
		char *id;
		char name[10];
		int age;
		float atk;
} ADV;

int main( void ) {
		FILE *fp;
		int numWritten;
		ADV a1 = {"0X1", "Alex", 10, 14.6};

		if( (fp = fopen( "char_data.bin", "wb" ) ) != NULL){
				numWritten = fwrite( &a1, sizeof( ADV ), 1 , fp );
				fclose( fp );
		} else {
				printf("NANI!? Problem opening the file.");
		}

}
```

Fun Codes Reading from a File

```
#include <stdio.h>
#pragma pack(1)

typedef struct {
            char *id;
            char name[10];
            int age;
            float atk;
} ADV;

int main( void ) {
            FILE *fp;
            int numWritten;
            ADV a1 = {"0X1", "Alex", 10, 14.6}, a2;

            if( (fp = fopen( "char_data.bin", "rb" )) != NULL ) {
                        numWritten = fread( &a2, sizeof( ADV ), 1 , fp );
                        fclose( fp );
            } else {
                        printf( "Problem opening the file\n" );
            }

            printf("id: %s\n", a2.id);
            printf("name: %s\n", a2.name);
            printf("age: %d\n", a2.age);
            printf("atk: %.2f\n", a2.atk);

}
```

mode specifies the kind of access requested

mode	**Access**
"r"	Opens for reading. If the file does not exist or cannot be found, the fopen call fails.
"w"	Opens an empty file for writing. If the given file exists, its contents are destroyed.
"a"	Opens for writing at the end of the file (appending) without removing the end-of-file (EOF) marker before new data is written to the file. Creates the file if it does not exist.
"r+"	Opens for both reading and writing. The file must exist.
"w+"	Opens an empty file for both reading and writing. If the file exists, its contents are destroyed.
"a+"	Opens for reading and appending. The appending operation includes the removal of the EOF marker before new data is written to the file. The EOF marker is not restored after writing is completed. Creates the file if it does not exist.

*Add **b** to make it suitable for binary files*

mode	**Access**
"rb"	Opens for reading. If the file does not exist or cannot be found, the fopen call fails.
"wb"	Opens an empty file for writing. If the given file exists, its contents are destroyed.
"ab"	Opens for writing at the end of the file (appending) without removing the end-of-file (EOF) marker before new data is written to the file. Creates the file if it does not exist.
"r+b"	Opens for both reading and writing. The file must exist.
"w+b"	Opens an empty file for both reading and writing. If the file exists, its contents are destroyed.
"a+b"	Opens for reading and appending. The appending operation includes the removal of the EOF marker before new data is written to the file. The EOF marker is not restored after writing is completed. Creates the file if it does not exist.

Random File Access

rewind()   fseek()   ftell()

File Organization

File organization refers to the way data are stored in a file. Files are said to have sequential organization. Reading the file is also in a sequential manner.

File access refers to retrieving data from a file.

rewind()

`rewind()` - resets the **current position** to the **start** of the **file**

General form: `rewind( fp )`

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

`rewind()` is done automatically when a file is opened in read mode

fseek()

`fseek()` - allows the programmer to move to any position in the file. Each character in a data file is located by its location in the file. **Returns `0`** if successful, non-zero otherwise.

Position vs offset

General form: `fseek( fp, offset, origin )`

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

`origin =  { 0, 1, 2 }`
SEEK_SET - offset is relative to the start of the file
SEEK_CUR - offset is relative to the current position in the file
SEEK_END - offset is relative to the end of the file

Offset
`>0` (positive) move forward
`<0` (negative) move backward

ftell()

`ftell()` - returns the offset value of the next character that will be read or written

General form: `ftell( fp )`

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

Returns a long integer

Fun Codes

```
FILE *fp;
int numWritten;
ADV a1[] = {
   "0X1", "Alex", 10, 14.6,
   "F22", "Samu", 35, 1000,
   "ADC", "Neneve", 19, 200
}, a2;

if( (fp = fopen( "char_data.bin", "wb" )) != NULL ) {
            numWritten = fwrite( &a1[0], sizeof( ADV ), 1 , fp );
            numWritten = fwrite( &a1[1], sizeof( ADV ), 1 , fp );
            numWritten = fwrite( &a1[2], sizeof( ADV ), 1 , fp );
            fclose( fp );
} else {
            printf( "Problem opening the file\n" );
}

if( (fp = fopen( "char_data.bin", "rb" )) != NULL ) {
            numWritten = fread( &a2, sizeof( ADV ), 1 , fp );
            printADV(a2);
            numWritten = fread( &a2, sizeof( ADV ), 1 , fp );
            printADV(a2);
            fseek(fp, sizeof(ADV)*-1L, SEEK_END);
            numWritten = fread( &a2, sizeof( ADV ), 1 , fp );
            printADV(a2);
            fclose( fp );
} else {
            printf( "Problem opening the file\n" );
}
```

Fun Reads

When a call is made to fopen

1. Computer checks whether the file currently exists on the system
2. If the file having the indicated file name exists, file is opened.
3. If the file does not exist, and opened as “w” or “a”, a blank file is opened having the indicated file name is created.
4. If a file opened for reading does not exist, fopen returns NULL.
5. NULL can be used to test that an existing file has been opened.

Fun Reads

When a call is made to `fread`

- The **fread** function reads up to *count* items of *size* bytes from the input *stream* and stores them in *buffer*.
- The file pointer associated with *stream* (if there is one) is increased by the number of bytes actually read. If the given stream is opened in text mode, Windows-style newlines are converted into Unix-style newlines. That is, carriage return-line feed (CRLF) pairs are replaced by single line feed (LF) characters. The replacement has no effect on the file pointer or the return value. The file-pointer position is indeterminate if an error occurs. The value of a partially read item cannot be determined.

Fun Reads

When a call is made to `fwrite`

- The **fwrite** function writes up to *count* items, of *size* length each, from *buffer* to the output *stream*. The file pointer associated with *stream* (if there is one) is incremented by the number of bytes actually written.
- If *stream* is opened in text mode, each line feed is replaced with a carriage return-line feed pair. The replacement has no effect on the return value.

Fun Reads

Common Programming Errors

1. External file name used over the internal file name

   Only fopen uses the external file name, the others use the internal file name

2. Forgetting to include a file name when accessing data files
3. When using EOF sentinel to detect the end of the file, any variable used to accept the EOF must be declared as an integer variable and not a character variable.

   EOF is an integer value without a character representation

4. What is the value of EOF?