60-141-02 LECTURE 7: FILE PROCESSING

Edited by Dr. Mina Maleki

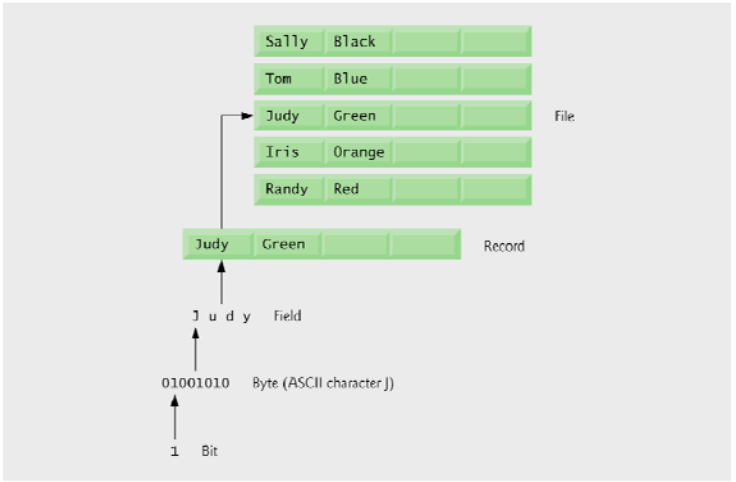
Outline

- Concept of File
- File Types
- Sequential-Access Files and Techniques
- Random-Access Files and Techniques

Data Hierarchy

- Bit – smallest data item
  - Either value of 0 or 1
- Byte – 8 bits
  - Used to store a character such as decimal digits, letters, and special symbols
- Field – group of characters conveying meaning
  - Example: your name
- Record – group of related fields
  - Represented by a struct
  - Example: In a payroll system, a record for a particular employee includes his/her identification number, name, address, etc.
- File – group of related records
  - Example: a payroll file
- Database – the group of related files

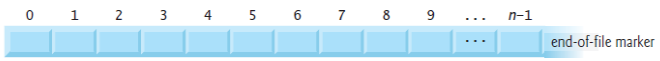
Data hierarchy



Concept of File

5

- Storage of data in variables and arrays is temporary
  - Data is lost when a program terminates.
- Files are used for permanent retention of data.
  - Files are stored on secondary storage devices (hard drive, CDs, DVDs,…)
- C views each file as a sequential stream of bytes.
- Each file ends either with an end-of-file marker or at a specific byte number recorded in a system-maintained, administrative data structure.



Concept of File …

6

- In order to communicate with a file it is necessary, first, to ***open*** a channel to the device where the file is located (or will be located, once created).
- When the program is finished with the file, it is necessary to ***close*** the channel.
- All required information concerning the file attributes is contained in a C-defined data structure called **FILE** (in <stdio.h>)
  - **FILE * filePtr ;** // pointer to struct that will hold file attributes
  - Channels may be re-opened and closed, multiple times
  - A FILE pointer may be re-assigned to different files

Concept of File …

7

- Files need to be **opened** before use.
  - Associate a "**file handler**" to each file
  - Modes: read, write, or append
- File input/output functions use the file handler (*not* the filename).
- Need to **close** the file after use.
- Basic file handling functions:
  - **`fopen()`**, **`fclose()`**, **`fscanf()`**, **`fprintf()`**.

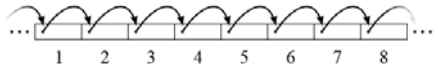
Files and Streams

8

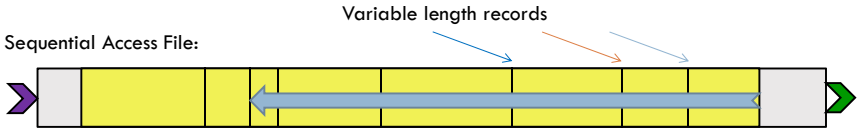
- When a file is opened, a stream is associated with it.
  - Streams provide communication channels between files and programs.
- Opening a file returns a pointer to a FILE structure that contains information used to process the file.
  - File control block (FCB) : File Name String, File Offset (Bytes), Access Mode (R,W,B,+)
- Standard streams (automatically open when program execution begins)
  - Standard input (Using file pointers stdin)
    - Enables a program to read data from the keyboard
  - Standard output (Using file pointers stdout)
    - Enables a program to print data on the screen
  - Standard error (Using file pointers stderr)

File Types

- Sequential-Access File
  - Data read in one direction starting from the beginning.

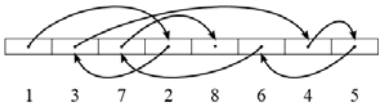


  - Usually variable length records

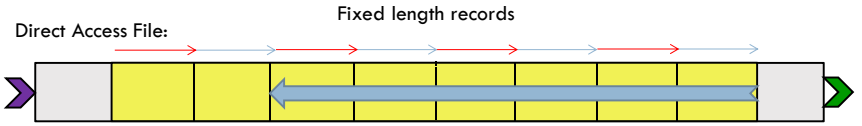


File Types …

- Random-Access File
  - Allows operations to 'seek' or move the read/write head to a particular bit position.



  - Must be fixed length records



Sequential Access File

Sequential Access File

- Often called a text file
- Records can only be accessed sequentially, one after another, from beginning to end.
- Cannot be modified without the risk of destroying other data.

Example 1 – employee list

```
Bonnel#Jacob
Carlisle#Donald
Eberg#Jack
Hou#Chang
```

Example 2 – memo

```
To all employees:

Effective January 1, 2009, the cost of dependent coverage will
increase from $35 to $38.50 per month.

Jefferson Williams
Insurance Manager
```

Example 3 – report

```
ABC Industries Sales Report

State          Sales
California     15000
Montana        10000
Wyoming         7000
               -----
Total sales:  $32000
```

Sequential Access Techniques

13

- fopen
  - opens a file in the desired mode of operation
  - Return NULL means "no file exists"
- fclose
  - closes the file and flushes buffers
- fprintf / fscanf
  - similar to printf and scanf, only use a FILE pointer as a first parameter to indicate where the data is being streamed to / from.
  - returns number of parameters outputted/ inputted, or failure of operation
- rewind
  - sets the file position to the beginning of the file of the given **stream.**

Sequential Access File Techniques

14

- WRITING (Output) - Creating a sequential access file
  - Request to create the file (open)
  - Check if the file was actually created
  - Write something to the file
  - Close the file (and save)
- READING (Input) – Reading a sequential access file
  - Request to open an existing file
  - Check if the file was actually opened
  - Check if there is something to read!
  - Read something from the file into a variable
  - Close the file

File I/O (Header)

15

- ***Step 0***: Include **`stdio.h`**.

```
#include <stdio.h>

int main()
{
  ...

  return 0;
}
```

File I/O (File Pointer)

16

- ***Step 1***: Declare a file handler (file pointer) as **`FILE *`** for each file.

```
int main()
{
  FILE  *inputfile = NULL;
  FILE  *outputfile = NULL;
  FILE  *currentfile = NULL;

  ...

  return 0;
}
```

File I/O (Open)

17

- *Step 2*: Open file using **fopen( )**.

```
int main()
{
  FILE  *inputfile = NULL;
  FILE  *outputfile = NULL;
  FILE  *currentfile = NULL;

  inputfile = fopen("Names.txt", "r");
  outputfile = fopen("marks.dat", "w");
  currentfile = fopen("logFile.txt", "a");
  ...
  return 0;
}
```

There can be many files opened at the same time, each using its own FILE structure and file pointer.

File I/O (Open)

18

- *Step 2*: Open file using **fopen( )**.

```
int main()
{
  FILE  *inputfile = NULL;
  FILE  *outputfile = NULL;
  FILE  *currentfile = NULL;

  inputfile = fopen("Names.txt", "r");
  outputfile = fopen("marks.dat", "w");
  currentfile = fopen("logFile.txt", "a");
  ...
  return 0;
}
```

Associate a file handler for every file to be used.

File I/O (Open)

19

- *Step 2*: Open file using **fopen( )**.

```
int main()
{
  FILE  *inputfile = NULL;
  FILE  *outputfile = NULL;
  FILE  *currentfile = NULL;

  inputfile = fopen("Names.txt", "r");
  outputfile = fopen("marks.dat", "w");
  currentfile = fopen("logFile.txt", "a");
  ...
  return 0;
}
```

Mode
r : read
w : write
a : append

File name

Warning: The "w" mode overwrites the file, if it exists.

File I/O (Open)

20

- *Step 2*: Open file using **fopen( )**.

Mode	Description
r	Open an existing file for reading only
w	Create a file for writing only. If the file currently exists, destroy its contents before writing to it.
a	Open an existing file or create a file for writing at the end of the file.
r+	Open an existing file for update, including both reading and writing.
w+	Create a file for update use (reading and writing). If the file already exists, destroy its current contents before writing.
a+	Append: Open or create a file for update – writing is done at the end of the file.

File I/O (Error Check)

- ***Step 3***: Check if file is opened successfully.

```
int main()
{
  FILE  *inputfile;

  inputfile = fopen("Names.txt", "r");

  if (inputfile == NULL)
  {
    printf("Unable to open input file.\n");
    return 1;
  }
  ...
  return 0;
}
```

File handler becomes NULL when an fopen() error occurs.

File I/O (Error Check)

- ***Step 3***: Check if file is opened successfully.
- What can go wrong?
  - Opening a file for writing when no disk space is available (or exceeded your quota)
  - Opening a file from a path that does not exist
  - Opening a file to which you have no proper permissions
  - Opening a file whose content is corrupted
  - Sharing violation, when someone else is writing to the file at the same time! (Exclusive lock or data overwriting may result)

File I/O (read a file)

- ***Step 4a***: Use **`fscanf()`** for input.

```
/* Assuming "Names.txt" contains a
   list of names. Read in each name, and keep count
   how many names there are in the file. */
char  name[MAXLEN];
int count = 0;

while ( fscanf(inputfile, "%s", name) == 1 )
{
  count++;
  printf("%d. %s\n", count, name);
}

printf("\nNumber of names read: %d\n", count);
```

File I/O (read)

- ***Step 4a***: Use **`fscanf()`** for input.

```
/* Assuming "Names.txt" contains a
   list of names. Read in each name, and keep count
   how many names there are in the file. */
char  name[MAXLEN];
int count = 0;

while ( fscanf(inputfile, "%s", name) == 1 )
{
  count++;
  printf("%d. %s\n",
}

printf("\nNumber of names read: %d\n", count);
```

Requires the file handler, not the file name.

File I/O (read)

- *Step 4a*: Use **`fscanf()`** for input.

```
/* Assuming "Names.txt" contains a
   list of names. Read in each name, and keep count
   how many names there are in the file. */
char  name[MAXLEN];
int count = 0;

while ( fscanf(inputfile, "%s", name) == 1 )
{
  count++;
  printf("%d. %s\n",
}

printf("\nNumber of names read: %d\n", count);
```

Other parameters: like ordinary scanf()

File I/O (read)

- *Step 4a*: Use **`fscanf()`** for input.

```
/* Assuming "Names.txt" contains a
   list of names. Read in each name, and keep count
   how many names there are in the file. */
char  name[MAXLEN];
int count = 0;

while ( fscanf(inputfile, "%s", name) == 1 )
{
  count++;
  printf("%d. %s\n",
}

printf("\nNumber of names read: %d\n", count);
```

fscanf() returns the number of input items converted and assigned successfully .

File I/O (read)

- *Step 4a*: Use **`fscanf()`** for input.

```
/* Assuming "Names.txt" contains a
   list of names. Read in each name, and keep count
   how many names there are in the file. */
char  name[MAXLEN];
int count = 0;

while ( fscanf(inputfile, "%s", name) == 1 )
{
  count++;
  printf("%d. %s\n",
}

printf("\nNumber of names read: %d\n", count);
```

Used to check if a read or assignment error occurred, or end of input file has been reached.

File I/O (read)

- To check for end-of-file (or any other input error):
  - check that the **number of items** converted and assigned successfully is **equal** to the **expected** number of items.

```
while ( fscanf(inpf, "%s %f", name, &mark) == 2 ){
  ...
}
```

  - Check for end of file marker (EOF)

```
fscanf(inputfile, "%s", name);
while (!foef(inputfile){
    ...
    fscanf(inputfile, "%s", name);
}
```

File I/O (read)

- *Step 4a*: Use **`fscanf()`** for input.

```
/* Assuming "Names.txt" contains a
   list of names. Read in each name, and keep count
   how many names there are in the file. */
char  name[MAXLEN];
int count = 0;

while ( fscanf(inputfile, "%s", name) == 1 )
{
  count++;
  printf("%d. %s\n", c
}

printf("\nNumber of na
```

```
fscanf(inputfile, "%s", name);
while ( !foef(inputfile){
    count++;
    printf("%d. %s\n", count, name);
    fscanf(inputfile, "%s", name);
}
```

File I/O (Create a file)

- *Step 4b*: Use **`fprintf()`** for output.

```
/* The output file "names_marks.dat" will contain
   the list of names and corresponding marks. */
  FILE  *outfile = NULL;
  outfile = fopen("names_marks.dat", "w");

  if (outfile == NULL){
    printf("Error opening output file.\n");
    return 1;
  }
…
  if ( fprintf(outfile, "%s %f\n", name, mark) <= 0 ){
      printf("Error writing to output file.\n");
      return 1;
  }
```

File I/O (Create a file)

- *Step 4b*: Use **`fprintf()`** for output.

```
/* The output file "names_marks.dat" will contain
   the list of names and corresponding marks. */
  FILE  *outfile = NULL;
  outfile = fopen("names_marks.dat", "w");

  if (outfile == NULL){
    printf("Error opening output file.\n");
    return 1;
  }
…
  if ( fprintf(outfile, "%s %f\n", name, mark) <= 0 ){
      printf("Error writing to output file.\n");
      return 1;
  }
```

File handler, not the file name.

File I/O (Create a file)

- *Step 4b*: Use **`fprintf()`** for output.

```
/* The output file "names_marks.dat" will contain
   the list of names and corresponding marks. */
  FILE  *outfile = NULL;
  outfile = fopen("names_marks.dat", "w");

  if (outfile == NULL){
    printf("Error opening output file.\n");
    return 1;
  }
…
  if ( fprintf(outfile, "%s %f\n", name, mark) <= 0 ){
      printf("Error writing to output file.\n");
      return 1;
  }
```

Other parameters: like ordinary printf()

File I/O (Create a file)

- ***Step 4b***: Use **`fprintf()`** for output.

```
/* The output file "names_marks.dat" will contain
   the list of names and corresponding marks. */
  FILE  *outfile = NULL;
  outfile = fopen("names_marks.dat", "w");

  if (outfile == NULL){
    printf("Error opening output file.\n");
    return 1;
  }
…
  if ( fprintf(outfile, "%s %f\n", name, mark) <= 0 ){
      printf("Error writing to output file.\n");
      retu
  }
```

fprintf() returns the number of characters written out successfully, or negative if an error occurs.

File I/O (Close)

- ***Step 5***: Close file using **`fclose()`**

```
int main()
{
  /*** etc ***/

  printf("\n");
  printf("Number of names read: %d\n", count);

  fclose(inputfile);
  fclose(outfile);

  return 0;
}
```

File I/O (Close)

- ***Step 5***: Close file using **`fclose()`**

```
int main()
{
  /*** etc ***/

  printf("\n");
  printf("Number of names read: %d\n", count);

  fclose(inputfile);
  fclose(outfile);

  return 0;
}
```

- *Clears input buffer.*
- *Flushes output buffer.*
- *fclose() fails when the file was not opened successfully.*

File handler not the file name.

Rewind

- There are two ways of re-reading a sequential file
  - Close the file and then re-open it
    - considered quite inefficient
  - Rewind the file to the beginning (reset the file offset value in the FCB) while leaving it open

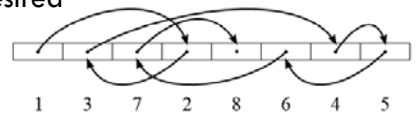
    void rewind(FILE *stream)
- Example : rewind(cfPtr) ;
  - sets the file position to the beginning of the file of the given **stream.**

Random Access File

C File processing

Random-Access File

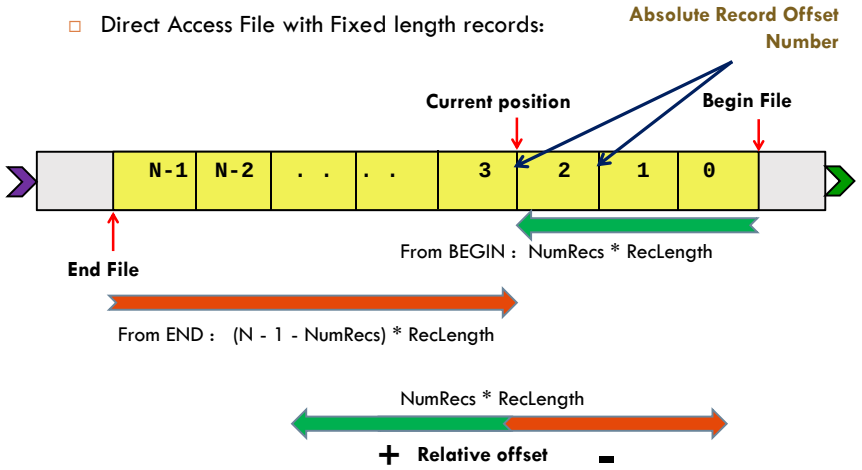
- Also called Direct Access File
- Read/Write operation can be performed directly at the position (within the file) desired



- Must be fixed length records → enable data
  - To be inserted in a file without destroying other data in the file.
  - To be updated or deleted without rewriting the entire file.
  - To be accessed directly and quickly without searching through other records.
- Appropriate for systems that require rapid access to specific data (airline reservation systems, banking systems)

Concept of Direct Access File

- Direct Access File with Fixed length records:



Random Access Techniques

- fopen : open a file- specify how its opened (read/write) and type (binary/text)
- fclose : close an opened file
- fread: read from a file
- fwrite: write to a file
- fseek: move a file pointer to somewhere in a file.
- ftell: tell you where the file pointer is located.

fopen () - Making File Connections

41

- `cfPtr1 = fopen( "MyNewFileName.dat", "wb" ) ;`
  - `open for writing`
- `fPtr2 = fopen( "MyOldFileName.dat", "rb" ) ;`
  - `open for reading`
- C supports three types of fixed length file transactions, called binary modes:
  - Read binary (rb), Write binary (wb) and Append binary (ab)
- Binary files are very similar to arrays of structures.
- Binary files have two features that distinguish them from text files:
  - You can instantly use any structure in the file.
  - You can change the contents of a structure anywhere in the file.

fopen () - Making File Connections …

42

Mode	Description (all files are binary)
rb	Open an existing file for reading only
wb	Create a file for writing only. If the file currently exists, destroy its contents before writing to it.
ab	Open an existing file or create a file for writing at the end of the file.
rb+	Open an existing file for update, including both reading and writing.
wb+	Create a file for update use. If the file already exists, destroy its current contents before writing.
ab+	Append: Open or create a file for update – writing is done at the end of the file.

fread() and fwrite()

43

- The **fread** and **fwrite** function takes four parameters:
  - A memory address
  - Number of bytes to read/write per block
  - Number of blocks to read/write
  - A file handler
- Reading from a direct access file
  - `fread(&my_record, sizeof(struct rec), 1, cfPtr);`
  - Read x bytes (size of `rec`) from the file `cfPtr` into memory address `&my_record`
- Writing to a direct access file
  - `fwrite(&my_record,sizeof(struct rec), NumRecs, cfPtr);`

Example 1

44

```
#include<stdio.h>
struct rec {int x,y,z;};
int main()   {
   int counter;
   struct rec my_record;
   FILE *ptr_myfile;
   ptr_myfile = fopen("test.bin","wb");
   if (!ptr_myfile) {
      printf("Unable to open file!"); return 1;
   }
   for (counter=1; counter <= 10; counter++) {
      my_record.x= counter;
      fwrite(&my_record,sizeof(struct rec),1,ptr_myfile);
   }
   fclose(ptr_myfile);
   return 0;
}
```

Example 1

45

```
#include<stdio.h>
struct rec   {int x,y,z;};
int main()   {
   int counter;
   struct rec my_record;
   FILE *ptr_myfile;
   ptr_myfile=fopen("test.bin","wb");
   if (!ptr_myfile) {
      printf("Unable to open file!"); return 1;
   }
   for ( counter=1; counter <= 10
      my_record.x= counter;
      fwrite(&my_record,sizeof(struct rec),1,ptr_myfile);
   }
   fclose(ptr_myfile);
   return 0;
}
```

File handler

File Name

Mode

File handler becomes NULL when an fopen() error occurs.

Example 1

46

```
#include<stdio.h>
struct rec   {int x,y,z;};
int main()   {
   int counter;
   struct rec my_record;
   FILE *ptr_myfile;
   ptr_myfile=fopen("test.bin","wb");
   if (!ptr_myfile) {
      printf("Unable to open file!"); return 1;
   }
   for ( counter=1; counter <= 10; counter++) {
      my_record.x= counter;
      fwrite(&my_record,sizeof(struct rec),1,ptr_myfile);
   }
   fclose(ptr myfile);
   return 0;
}
```

Example 1

47

```
#include<stdio.h>
struct rec   {int x,y,z;};
int main()   {
   int counter;
   struct rec my_record;
   FILE *ptr_myfile;
   ptr_myfile=fopen("test.bin","wb");
   if (!ptr_myfile) {
      printf("Unable to open file!"); return 1;
   }
   for ( counter=1; counter <= 10; counter++) {
      my_record.x= counter;
      fwrite(&my_record,sizeof(struct rec),1,ptr_myfile);
   }
   fclose(ptr_myfile);
   return 0;
}
```

- *Clears input buffer.*
- *Flushes output buffer.*
- *fclose() fails when the file was not opened successfully.*

Example 2

48

```
#include<stdio.h>
struct rec   {int x,y,z;};
int main()   {
   int counter;
   struct rec my_record;
   FILE *ptr_myfile;
   ptr_myfile=fopen("test.bin",“rb");
   if (!ptr_myfile) {
      printf("Unable to open file!"); return 1;
   }
   for ( counter=1; counter <= 10; counter++) {
      fread(&my_record,sizeof(struct rec),1,ptr_myfile);
      printf("%d\n",my_record.x);
   }
   fclose(ptr_myfile);
   return 0;
}
```

Example 1

49

```
#include<stdio.h>
struct rec   {int x,y,z;};
int main()   {
   int counter;
   struct rec my_record;
   FILE *ptr_myfile;
   ptr_myfile=fopen("test.bin","rb");
   if (!ptr_myfile) {
      printf("Unable to open file!"); return 1;
   }
   for ( counter=1; counter <= 10; counter++) {
      fread(&my_record,sizeof(struct rec),1,ptr_myfile);
      printf("%d\n",my_record.x);
   }
   fclose(ptr_myfile);
   return 0;
}
```

fseek()

50

- Seeking a record in a direct access file :

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

  - offset is the number of bytes to move the position indicator
  - origin says where to move from
- Three options/constants are defined for origin
  - SEEK_SET - move the indicator offset bytes from the beginning
  - SEEK_CUR - move the indicator offset bytes from its current position
  - SEEK_END - move the indicator offset bytes from the end
- The new position, measured in **characters** from the beginning of the file, is obtained by adding offset to the reference position specified by origin.

Example 1

51

```
/* fseek example */
#include <stdio.h>

int main ()
{
  FILE * pFile;
  pFile = fopen ("example.txt" , "wb");
  fputs ("This is an apple." , pFile);
  fseek (pFile , 9 , SEEK_SET );
  fputs (" sam" , pFile);
  fclose (pFile);
  return 0;
}
```

This is a sample

Example 2

52

Reads file records backward

```
#include<stdio.h>
struct rec   {int x,y,z;};
int main()   {
   int counter;
   struct rec my_record;
   FILE *ptr_myfile;
   ptr_myfile = fopen("test.bin","rb");
   if (!ptr_myfile) {
      printf("Unable to open file!"); return 1;
   }
   for (counter=9; counter >= 0; counter--) {
      fseek(ptr_myfile,sizeof(struct rec)*counter
            ,SEEK_SET);
      fread(&my_record,sizeof(struct rec),1,ptr_myfile);
      printf("%d\n",my_record.x);
   }
   fclose(ptr_myfile);
   return 0;}
```

Example 3

Override file records based on their ID (line)

```
#include <stdio.h>
struct rec_t {
    int ID ;          // Assume 1 <= ID <= 100
    char Name[50] ;
    double Score ;
}
int main( ) {
    FILE * cfPtr ;
    struct rec_t Rec ;

    cfPtr = fopen( "Score.dat", "wb" ) ;
    while(scanf("%d", &Rec.ID) != EOF ) {
        scanf("%s%lf", Rec.Name, &Rec.Score) ;

        fseek(cfPtr,(Rec.ID - 1)*sizeof(struct rec_t ),
                                                SEEK_SET);
        fwrite(&Rec, sizeof(struct rec_t ),1,cfPtr ) ;
    }
    fclose(cfPtr);
    return 0 ;
}
```

ftell ()

- Get current position in stream

  long int ftell (FILE * stream);

  - Returns number of bytes from the beginning of the file.
  - The first byte of the file is byte 0.
  - If an error occurs, ftell () returns -1.

Displays size of myfile in bytes

```
#include <stdio.h>
int main () {
    FILE * pFile;
    long size;
    pFile = fopen("myfile.txt","rb");
    fseek (pFile, 0, SEEK_END);
    size=ftell (pFile);
    fclose (pFile);
    printf ("Size of myfile.txt: %ld bytes.\n",size);
    return 0;
}
```

Sequential vs. Random Access File

Sequential Access File

- Opens the file and positions the read/write head at the beginning of the file (except append where it starts writing at the end of the file).
- we may need to move through multiple records before we finally arrive at the file position desired
- To read a given record, all previous records have to be read in order for the read head to be moved to the record required.

Random Access File

- Can access a record directly.
- Highly structured and well organized.
- Operations such as fwrite, fread and fseek are used.
- We need to keep track of specific byte positions, such as calculate where a given record is located in the file (calculate the byte offset starting at position 0)
- Data can be added easily to a random-access file without destroying other data in the file.

Lecture 7: Summary

- File Types
- Sequential Access Files and Techniques
- Random (Direct) Access Files and Techniques
- Reading
  - Chapter 11: File Processing
    - Moving beyond RAM to include data on persistent storage in the file system.
- Assignment
  - Deadline of the fourth assignment is **March 25**.
  - Deadline of the fifth assignment is **March 29**.