

Chapter 11

C File Processing

- Storage of data in variables, arrays, and structures is temporary (RAM)-such data is lost when a program terminates.
- Files are used for permanent retention of data. Computers store files on secondary storage devices, especially disk storage devices.
- In this chapter, we explain how data files are created, updated and processed by C programs.
- We consider sequential-access files and random-access files.

Steps to Access and Process a File

Rules for accessing a file is simple.

Opening File: A file must be opened before processing/accessing. Opening a file is accomplished by using `fopen` function which is available under `<stdio.h>` library.

Processing File: After opening a file you can read or write to a file using appropriate statements. The function commonly used in file processing are `fgetc`, `fputc`, `fgets`, `fputs`, `fscanf`, `fprintf`, `fread`, `fwrite`, and `fseek`.

Closing File: A file should be closed after finishing processing the file. Closing a file can be accomplished by using `fclose` function which is available under `<stdio.h>` library. *All operating systems puts a limit on the number of the simultaneous file processing by a single program.* Some program deals with various files at the same time. Therefore, after processing a file, it is good to close the file to make sure that our program doesn't go beyond the limits put by the operating system.

Opening a File: fopen

Page 453

Prototype of `fopen` function

```
FILE *fopen(char *fileName, char *accessMode);
```

First argument is the name of the file to be processed. C language allow programmer to choose for what purpose to open the file; defined as the second argument (e.g. “r” only reading, “w” writing etc.).

`fopen` function returns a FILE structure pointer. C programs can manage a file via FILE structure pointer.

If any error occurs while opening the file, `fopen` returns a NULL pointer, otherwise it returns the FILE pointers that is used to process the file.

Other File Pointers are the `stdin`, `stdout`, and `stderr`. They provide processing between program – keyboard – screen etc.

Opening a File: fopen – Example –

Page 453

All C programs handles the file opening in the following form

```
FILE *cfPtr; //Defining a File Pointer
cfPtr = fopen( "clients.dat", "r" );
```

```
1: /* Demonstration of Opening a File */
2: #include <stdio.h>
3: int main( void )
4: {
5: FILE *cfPtr; /* Defining a file pointer */
6:
7: /* fopen opens file. Exit program if unable to create file */
8: if ( ( cfPtr = fopen( "clients.dat", "r" ) ) == NULL ) {
9: printf( "File could not be opened!\n" );
10: } /* end if */
11:
12: else {
13: printf( "File is Opened and Ready for Processing!\n" );
14: printf( "cfPtr file pointer can be used to process the file.\n" );
15: fclose( cfPtr ); /* fclose closes file */
16:     } /* End of Else */
17:
18: return 0; /* indicates successful termination */
19: } /* end main */
```

File Opening Modes

Page 457

While opening a file by using `fopen`, we can determine the mode of the file as second argument. Below table presents possible modes;*

Mode	Description
r	Open an existing file for reading.
w	Create a file for writing. If the file already exists, discard the current contents.
a	Append; open or create a file for writing at the end of the file.
r+	Open an existing file for update (reading and writing).
w+	Create a file for update. If the file already exists, discard the current contents.
a+	Append: open or create a file for update; writing is done at the end of the file.
rb	Open an existing file for reading in binary mode.
wb	Create a file for writing in binary mode. If the file already exists, discard the current contents.
ab	Append; open or create a file for writing at the end of the file in binary mode.
rb+	Open an existing file for update (reading and writing) in binary mode.
wb+	Create a file for update in binary mode. If the file already exists, discard the current contents.
ab+	Append: open or create a file for update in binary mode; writing is done at the end of the file.

**Same as previous ones.
But opens the files as
binary files.
Data in binary files are
kept in terms of bits.
If you open it, you will
see just symbols look
nonsense.**

The choice of mode should be made depends on the purpose of opening the file.

Binary files called unformatted files in some programming languages, are more efficient in processing and also takes a lot less space then formatted files. These files are machine readable not human readable.

Commonly Used File Processing Functions 1/2

`fputc (c, filePtr)`

Writes a single character to the file identified by filePtr.

`fputs (string, filePtr)`

Writes a string to the file identified by filePtr

`fgetc (filePtr)`

Reads the next character from the file identified by filePtr, or the value EOF if and end-of-file condition occurs.

`fgets (string, i, filePtr)`

Reads characters from the indicated file, until either i – 1 characters are read or a newline character is read, whichever occurs first.

`fprintf (filePtr, format, arg1, arg2, ..., argn)`

Writes the specified arguments to the file identified by filePtr, according to format.

`fscanf (filePtr, format, arg1, arg2, ..., argn)`

Reads data items from the file identified by filePtr, according to the format.

Writing to a File – Example Program–

```
1: /* Demonstration of Writing to a File */
2: #include <stdio.h>
3: int main( void )
4: {
5:     char  gender='F'; //Female
6:     char *name="Alice", *last="Donn";
7:     int    age=39;
8:     float weight=68.7, height=165;
9:
10: FILE *fPtr; /* Defining a file pointer */
11: /* Opening a file for writing */
12: if ( ( fPtr = fopen( "trial.txt", "w" ) ) == NULL ) {
13:     printf( "File could not be opened!\n" );
14:     return 0; // Terminate the program
15: } /* end if */
16:
17: // Now Start Processing File - Start Writing
18: fprintf(fPtr,"Writing into the file indicated by fPtr\n");
19: fputc(gender,fPtr);
20: fputs(name,fPtr);
21: fprintf(fPtr,"%d",age);
22: fprintf(fPtr,"%f%h",weight,height);
23: fprintf(fPtr,"\nWriting All info Again\n");
24: fprintf(fPtr,"%c%20s%20s%10d%10.2f%10.2f",gender,name,last,age,weight,height);
25: fclose( fPtr ); /* fclose closes file */
26:
27: return 0; /* indicates successful termination */
28: } /* end main */
```

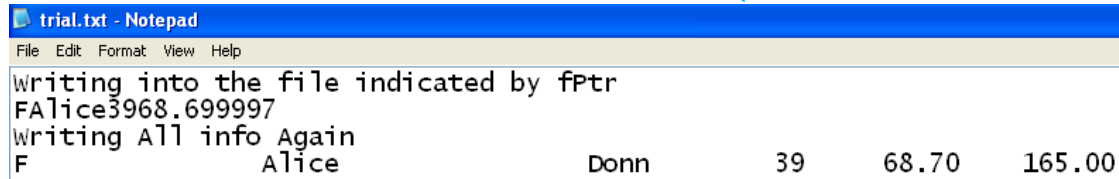
*** Content of trial.txt ***



```
trial.txt - Notepad
File Edit Format View Help
Writing into the file indicated by fPtr
F
Alice
Donn
39
68.70
165.00
Writing All info Again
F
Alice
Donn
39
68.70
165.00
```

Note: As noticed C imposes no formatting on file unless you made one.

More on a Files: EOF (end of line character)



```
trial.txt - Notepad
File Edit Format View Help
Writing into the file indicated by fPtr
FAlice3968.699997
Writing All info Again
F           Alice           Donn           39           68.70           165.00
```

As noticed, C imposes no formatting on file unless you made one. In order to access file later for reading, your program should read the data according to the way it was written. In other words, you should organize your data in the file, so you could find later when looking for something.

End-of-File Flag (EOF)

Each file has a termination character called End-of-File flag (similar to ‘\0’ in strings) and defined in `<stdio.h>`. The EOF flag located by the end of file and can’t be seen but recognized by the program.

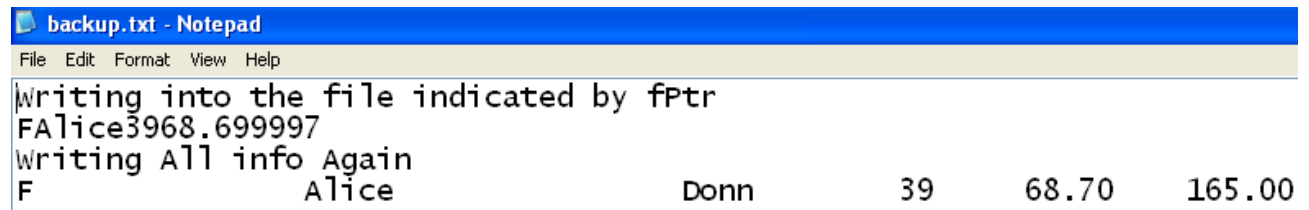
When we assign a FILE pointer to a file, pointer points the first elements and after reading values, it moves the next element. EOF flag can be used to control if we get to the end of the file otherwise, the program may go into infinite loop.

int feof (fPtr): Used to check if file pointer reached to the end of the file. Returns nonzero if fPtr reaches the end of file, otherwise returns 0.

Backing up a File – Example Program –

```
1: /* Demonstration of Copying a File to Another File*/
2: // First File is processed char by char and copied to second file.
3: #include <stdio.h>
4: //Filecopy copies the content of first file to second file
5: void filecopy ( FILE *ifP , FILE *ofP);
6:
7: int main( void )
8: {
9: FILE *fP1 , *fP2; /* Defining two file pointers */
10: /* Opening the files to be processed*/
11: if ( ( fP1 = fopen( "trial.txt", "r" ) ) == NULL ) {
12: printf( "Input File could not be opened!\n" );
13: return 0; //Terminating Program
14: } /* end if */
15: // File is Opened and Ready to process via fP1 Pointer
16: // Now Open backup.txt file and associate with fP2 pointer
17: if ( ( fP2 = fopen( "backup.txt", "w" ) ) == NULL ) {
18: printf( "Backup File could not be opened!\n" );
19: return 0; //Terminating Program
20: } /* end if */
21:
22: filecopy ( fP1 , fP2 ); //Calling function with two pointers
23: fclose( fP1 ); fclose( fP2 ); /* Closing the files */
24: return 0; } /* end main */
25:
26: /* filecopy: copy file ifP to file ofP */
27: void filecopy ( FILE *ifP , FILE *ofP)
28: {
29: char c;
30: while ((c = getc(ifP)) != EOF) // Read Character from ifP
31: putc(c, ofP); // Write Character to ofP
32: } // End of Function
```

*** Content of backup.txt after copying***



```
backup.txt - Notepad
File Edit Format View Help
Writing into the file indicated by fPtr
FAllice3968.699997
Writing All info Again
F                Alice                Donn                39                68.70                165.00
```

Storing Employee Records in a File – Example Program –

Write a C program that stores the Employee records including *Identification Number, Name, Last Name, Age, and Salary* information in a text file named **employee.txt**. *

ID. Num.	Name	Last Name	Age	Salary
1	Evonne	Koen	33	1500.5
2	Jana	Loehr	36	3300
3	Krystyna	Dunford	29	1200.75
4	Maris	Pinney	39	2900
5	Roscoe	Cutsforth	27	3250
6	Chester	Patrick	34	2225.25
7	Detra	Tagle	32	2750.5
8	Deadra	Palmieri	35	2600.25
9	Sharen	Mcelfresh	36	2425.25
10	Andre	Rodman	30	3750.5

```
struct records {  
    int    id;  
    char   *name;  
    Char   *last;  
    int    age;  
    doubl e float;  
};
```

In order to save time, instead of reading the employee records from keyboard, you may define these records in a structure of array of type records in your program.

Storing Employee Records in a File – Example Program –

Below program defines the records in a structure then writing each record to file as one record per line.*

```
1: /* Demonstration of Writing to a File */
2: #include <stdio.h>
3: struct records { //Defining the Structure
4: int id; char *name; char *last; int age; float salary; };
5: int main( void )
6: {
7: struct records employee[10]={ {1,"Evonne","Koen",33,1500.5},
8: {2,"Jana","Loehr",36,3300}, {3,"Krystyna","Dunford",29,1200.75},
9: {4,"Maris","Pinney",39,2900}, {5,"Roscoe","Cutsforth",27,3250},
10: {6,"Chester","Patrick",34,2225.25}, {7,"Detra","Tagle",32,2750.5},
11: {8,"Deadra","Palmieri",35,2600.25}, {9,"Sharen","Mcelfresh",36,2425.25},
12: {10,"Andre","Rodman",30,3750.5}}; //Employee structure array initialized
13: int i=0; // Counter initialized to 0
14: FILE *fPtr; /* Defining a file pointer */
15: if ( ( fPtr = fopen( "employee.txt", "w" ) ) == NULL ) {
16: printf( "File could not be opened!\n" );
17: return 0; //Terminate the Program
18: } // End of if statement block
19: //File is Opened and Ready to process via cfPtr Pointer
20: while ( i < 10 ){
21: fprintf(fPtr,"%3d%15s%15s%3d%10.2f\n",employee[i].id,employee[i].name,
22: employee[i].last,employee[i].age,employee[i].salary);
23: i++;
24: } // End of While Repetition
25: fclose( fPtr ); /* fclose closes file */
26:
27: return 0; /* indicates successful termination */
28: } /* end main */
```

* Content of ...*



1	Evonne	Koen	33	1500.50
2	Jana	Loehr	36	3300.00
3	Krystyna	Dunford	29	1200.75
4	Maris	Pinney	39	2900.00
5	Roscoe	Cutsforth	27	3250.00
6	Chester	Patrick	34	2225.25
7	Detra	Tagle	32	2750.50
8	Deadra	Palmieri	35	2600.25
9	Sharen	Mcelfresh	36	2425.25
10	Andre	Rodman	30	3750.50

Reading Employee Records from a File – Example Program –

Let's now retrieve the data stored in employee.txt in the previous example.*

```
1: /* Demonstration of Reading Employee Rec. from a File */
2: #include <stdio.h>
3: struct records { //Defining the Structure
4: int id; char name[30]; char last[30]; int age; float salary; };
5: int main( void )
6: {
7: struct records employee;
8: int num=0; // Counter and num: Number of records
9: FILE *fPtr; /* Defining a file pointer */
10:
11: if ( ( fPtr = fopen( "employee.txt", "r" ) ) == NULL ) {
12: printf( "File could not be opened!\n" );
13: printf( "Press Any Key to Terminate the Program!\n" );
14: getch();
15: return 0; // Terminating Program
16: } /* end if */
17: // Read From the File Until EOF found!
18: do{
19: fscanf(fPtr,"%d%10s%10s%d%f",&employee.id,employee.name,
20: employee.last,&employee.age,&employee.salary);
21: printf("%3d%15s%15s%3d%10.2f\n",employee.id,employee.name,
22: employee.last,employee.age,employee.salary);
23: num++; // One Record Read - Increase the counter by +1
24: } while ( !feof(fPtr) ); // End of While Repetition
25: // File processing Finished now close the file
26: fclose( fPtr ); /* fclose closes file */
27: printf("Number of Records: %d",num);
28: getch();
29: return 0; /* indicates successful termination */
30: } /* end main */
```

* Content of employee.txt*



1	Evonne	Koen	33	1500.50
2	Jana	Loehr	36	3300.00
3	Krystyna	Dunford	29	1200.75
4	Maris	Pinney	39	2900.00
5	Roscoe	Cutsforth	27	3250.00
6	Chester	Patrick	34	2225.25
7	Detra	Tagle	32	2750.50
8	Deadra	Palmieri	35	2600.25
9	Sharen	Mcelfresh	36	2425.25
10	Andre	Rodman	30	3750.50

* Output of the Program*



1	Evonne	Koen	33	1500.50
2	Jana	Loehr	36	3300.00
3	Krystyna	Dunford	29	1200.75
4	Maris	Pinney	39	2900.00
5	Roscoe	Cutsforth	27	3250.00
6	Chester	Patrick	34	2225.25
7	Detra	Tagle	32	2750.50
8	Deadra	Palmieri	35	2600.25
9	Sharen	Mcelfresh	36	2425.25
10	Andre	Rodman	30	3750.50
Number of Records: 10				

Commonly Used File Processing Functions 2/2

Commonly used other file processing functions are the followings;

`int feof (filePtr)`

Returns nonzero if the identified file has reached the end of the file and returns zero otherwise. feof function is used to check if the pointer reaches to the end of the file.

`void rewind (filePtr)`

Resets the filePtr pointer back to the beginning of the file.

`remove (char *fileName)`

Removes the specified file. A nonzero value is returned on failure.

`rename (char *fileName1, char *fileName2)`

Renames the file fileName1 to fileName2, returning a nonzero result on failure.

`fread`, `fwrite`, and `fseek` functions are also commonly used in processing of Random-Access Files.

Processing Employee Records – Comprehensive Example –

Let's now consider a more comprehensive example on file processing. In this example, we will again use Employee Records stored in `employee.txt`. Our program will allow user to search for employee records according to following menu

- 1) Print Number of Records
 - 2) Search by Employee ID
 - 3) Search by Employee Name
 - 4) Add New Employee Record
 - 5) Delete a Record for given Employee ID
- Exit other than chosen 1-5

Note: Opening the file and control of menu will be done by main program by using “fopen” and “switch case” respectively.

`rewind` function will be used to move the file pointer to beginning of the file for subsequent choices.

Comprehensive Example Program (Main – Part 1/7) **

```
1:  /* Program to Process Employee Rec. from a File + Add-Delete*/
2:  #include <stdio.h>
3:  #include <string.h>
4:  struct records { //Defining the Structure
5:  int id;
6:  char name[50];
7:  char last[50];
8:  int age;
9:  float salary; };
10: // Prototypes for the Functions
11: void num_of_record( FILE *fp );
12: void search_id(FILE *fp , int id);
13: void search_name(FILE *fp , char *name);
14: void delete_id(FILE *fp , int id);
15: void add_record(FILE *fp);
16: void delete_record(FILE *fp , int id);
17:
18: int main( void )
19: {
20: int opt;      // Option from the menu
21: int id;       // id number of user in interest
22: char *name;   // name of the user in interest
23: FILE *fPtr;  /* Defining a file pointer */
24: // Opening file with Read and Update Mode
25: if ( ( fPtr = fopen( "employee.txt", "r+" ) ) == NULL ) {
26: printf( "File could not be opened!\n" );
27: return 0 ; // Terminating the program
28: } /* end if */
```

Prototypes of the Functions

Opening File

Comprehensive Example Program (Main – Part 2/7) **

```
29: while( 1 ){//Note Program will be terminated in switch with retu
30: rewind(fPtr); //Rewind the pointer back to the beginning
31: system ("cls");//Using system command cls to clean the screen
32: printf("\n\nPlease Choose from one of the following Options:\n")
33: printf ("1.Display Number of Records\n");
34: printf ("2.Search Record by Employee ID\n");
35: printf ("3.Search Record by Employee Name\n");
36: printf ("4.Add New Employee Record\n");
37: printf ("5.Delete Record of Employee for given Employee ID\n");
38: printf ("__Exit (int other than 1-5)\n\n");
39: printf ("Enter Your Choice:");
40: scanf("%d",&opt); // Reeking Option by User
41: switch ( opt ){
42: case 1:
43: num_of_record( fPtr );
44: break;
45: case 2:
46:     printf("Enter Employee ID:");
47:     scanf("%d",&id);
48:     search_id( fPtr , id );
49: break;
50: case 3:
51:     printf("Enter Employee Name:");
52:     scanf("%s",name);
53:     search_name( fPtr , name );
54: break;
55: case 4:
56: add_record ( fPtr );
57: fclose (fPtr);
58: fPtr = fopen( "employee.txt", "r+" );
59: break;
60: case 5:
61:     printf("Enter the Employee ID to delete:");
62:     scanf("%d",&id);
63: delete_record ( fPtr , id );
64: fPtr = fopen( "employee.txt", "r+" );
65: break;
66: default:
67:     fclose( fPtr ); /* fclose closes file before exiting */
68:     return 0; // Terminating Program
69: break;
70: } // End of Switch Case
71: } // End of Do Repetition
72: return 0; /* indicates successful termination */
73: } /* end main */
```

Program Menu

**Switch Case Statement
Distributing Task over Functions**

Number of Records (num_of_record function – Part 3/7)

```
74:
75: void num_of_record( FILE *fp ){
76:     int num=0;
77:     struct records employee;
78:     do{
79:         fscanf(fp, "%d%s%d%f\n", &employee.id, employee.name,
80:             employee.last, &employee.age, &employee.salary);
81:         num++; // One Record Read - Increase the counter by +1
82:         printf("%3d%15s%15s%3d%10.2f\n", employee.id, employee.name,
83:             employee.last, employee.age, employee.salary);
84:     }while ( !feof(fp) ); // End of While Repetition
85:     printf("Number of Records is %d.", num);
86:     printf("\nPress Any Key to Continue!");
87:     getch();
88: } // End of Function
89:
```

num_of_record function:

- Function receives the pointer to the file.
- It reads the data as element of structures and printing them to the screen.
- It also keep tracks of the number of data to find the total number of data.

Searching Records by ID (search_id function – Part 4/7)

```
90: void search_id( FILE *fp , int id ){
91: struct records employee;
92: do{
93: fscanf(fp, "%d%s%d%f\n", &employee.id, employee.name,
94: employee.last, &employee.age, &employee.salary);
95: if ( id == employee.id){ // Record found print info
96: printf("%-15s: %-3d\n%-15s: %-15s\n%-15s: %-15s\n%-15s: %-3d\n%-15s: %-.2f\n",
97: "EMPLOYEE ID", employee.id, "EMPLOYEE NAME", employee.name, "EMP. LAST NAME", employee.last,
98: "EMPLOYEE AGE", employee.age, "EMPLOYEE SALARY", employee.salary);
99: printf("\nPress Any Key to Continue!");
100: getch();
101: return ; //Terminate the Function
102: } // End of if Statement
103: }while ( !feof(fp) ); // End of While Repetition
104: printf("\nEmployee with ID Number %d is not found!", id);
105: printf("\nPress Any Key to Continue!");
106: getch();
107: } // End of Function
```

search_id function:

- Function receives the pointer to the file and the id number.
- It reads the records as a member of structure from the file.
- If the record found, it terminates the program, because id is unique for an employee.
- If the employee doesn't find then it prompts it is not found.

Searching Records by Name (search_name function – Part 5/7)

```
108:
109: void search_name( FILE *fp , char name[30] ){
110: struct records employee;
111: int found = 0;
112: do{
113: fscanf(fp, "%d%s%s%d%f\n", &employee.id, employee.name,
114: employee.last, &employee.age, &employee.salary);
115: if ( !strcmp (name, employee.name) ){ // Record found print info
116: printf("\n%-15s: %-3d\n%-15s: %-15s\n%-15s: %-3d\n%-15s: %-.2f\n",
117: "EMPLOYEE ID", employee.id, "EMPLOYEE NAME", employee.name, "EMP. LAST NAME", employee.last,
118: "EMPLOYEE AGE", employee.age, "EMPLOYEE SALARY", employee.salary);
119: found = 1 ; // Found tracker is now 1
120: } // End of if Statement
121: }while ( !feof(fp) ); // End of While Repetition
122: if ( found == 0 ) printf("Employee with Name %s is not found!", name);
123: printf("\nPress Any Key to Continue!");
124: getch();
125: } // End of Function
```

search_name function:

- Function receives the pointer to the file and the name to be searched.
- It reads the records as members of structure and printing the record with the given name to the screen (multiple records may have the same name).
- Using a found variable to keep track down if the record is not found.
- If the employee doesn't find then it prompts it is not found.

Adding New Record (add_record function – Part 6/7)

```
126:
127: void add_record(FILE *fp) {
128:     struct records temp;
129:     while ( getc(fp) != EOF ); // First Move to the end of the file
130:     printf("Enter New Record ID:");
131:     scanf("%d",&temp.id);
132:     printf("Enter New Record Name:");
133:     scanf("%s",temp.name);
134:     printf("Enter New Record Last Name:");
135:     scanf("%s",temp.last);
136:     printf("Enter New Record Age:");
137:     scanf("%d",&temp.age);
138:     printf("Enter New Record Salary:");
139:     scanf("%f",&temp.salary);
140:     fprintf(fp, "%3d%15s%15s%3d%10.2f\n",temp.id,temp.name,
141:     temp.last,temp.age,temp.salary);
142:     printf("\nPress Any Key to Continue!");
143:     getch();
144: } // End of Function
145:
```

add_record function:

Function receives the pointer to the file.

Moves the file pointer to EOF of file (end of file).

It writes new given records to the EOF.

Delete Record for a Given ID (delete_record function – Part 7/7)

```
146: void delete_record(FILE *fp , int id){
147: FILE *ftemp; //Create a new temporary file pointer
148: if ( ( ftemp = fopen( "temp.temp", "w" ) ) == NULL ) {
149: printf( "Temp File Can not be Created!\n" );
150: return ; // Terminating the Function
151: } /* end if */
152: struct records employee;
153: do{
154: fscanf(fp, "%d%s%s%d\n", &employee.id, employee.name,
155: employee.last, &employee.age, &employee.salary);
156: if (id != employee.id){ // Skip Printing id
157: fprintf(ftemp, "%3d%15s%15s%3d%10.2f\n", employee.id, employee.name,
158: employee.last, employee.age, employee.salary);}
159: } while ( !feof(fp) ); // End of While Repetition
160: // Before Moving File - First Clear Up the Pointers
161: fclose(fp);
162: fclose(ftemp);
163: remove ( "employee.txt" );
164: rename ( "temp.temp", "employee.txt");
165: printf("Record Deleted! Press Any Key to Continue");
166: getch();
167: } // End of Function
```

delete_record function:

- Function receives the pointer to the file and id to be deleted.
- It reads the data and writes the data to a **temporary file** by skipping the given ID.
- After processing whole file, it moves the **temporary file** to **employee.txt**

Sequential Files:

Pages 453-461

- The examples presented in the previous slides demonstrated how to create a file for sequential access and how to access data/records from the sequential files.
- In sequential file processing, in order to reach to a record, we need to start from beginning then read all records till reaching that record.

11.6 Random-Access Files:

Page 462

- Another type of file processing is Random-Access File processing. In this type of data processing, instead of reading all previous records to reach to a specific record, one may directly skip to the record of interest (by using fseek function).
- Random-Access file data storage is important for large scale of data processing like Airline Ticketing systems or Banking Account information systems where the users require instant/fast access to the record.