Unit 7 File Structures

Structure:

7.1 Introduction

From the point of view of a C program, all kinds of files and devices for input and output are uniformly represented as logical data *streams*, regardless of whether the program reads or writes a character or byte at a time, or text lines, or data blocks of a given size. Streams in C can be either *text* or *binary streams*, although on some systems even this difference is nil. C leaves file management upto the execution environment, in other words, the system on which the program runs. Thus a stream is a channel by which data can flow from the execution environment to the program or from the program to its environment. Devices, such as consoles, are addressed in the same way as files.

A file represents a sequence of bytes. The fopen( ) function associates a file with a stream and initializes an object of the type FILE, which contains all

the information necessary to control the stream. Such information includes a pointer to the buffer used; a file position indicator, which specifies a position to access in the file; and flags to indicate error and end-of-file conditions.

Each of the functions that open files, namely, fopen( ), freopen( ), and tmpfile( ) returns a pointer to a FILE object for the stream associated with the file being opened. Once you have opened a file, you can call functions to transfer data and to manipulate the stream. Such functions have a pointer to a FILE object commonly called a FILE pointer as one of their arguments. The FILE pointer specifies the stream on which the operation is carried out.

Objectives:

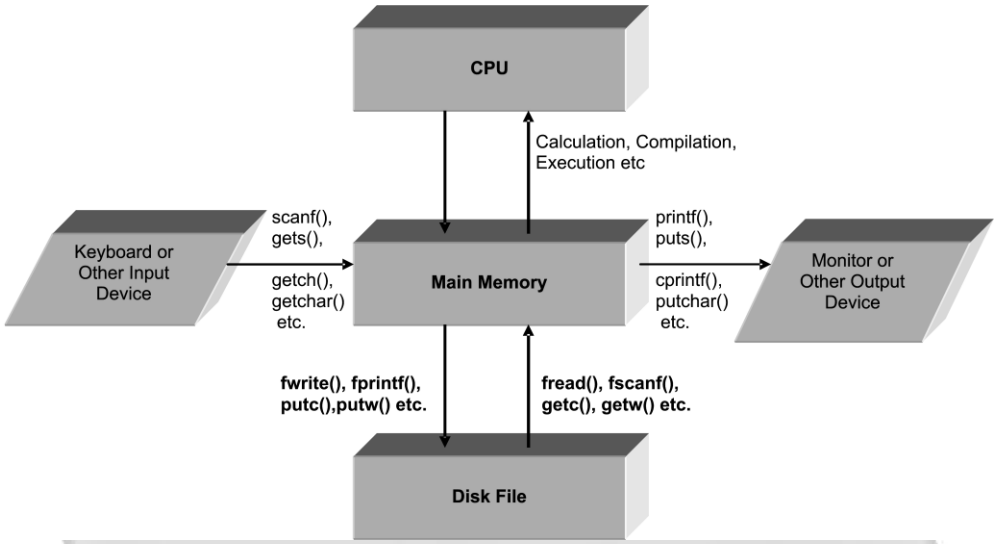
At the end of this unit, you will be able to:

- provide a overview of file.
- describe the process of File opening, Accessing Records, Closing.
- explain Sequential Random File handling methods.

7.2 Block Structure

We know that file is the storage place where we can store data in the disk permanently. Until further command no data will be deleted. There are number of functions for input and output to store data and retrieve data from the file.

The following diagram will show the detail structure of Language C file operation.



Block Structure of File Operation

Fig. 7.1

7.3 Opening, Accessing and Closing Files

To write to a new file or modify the contents of an existing file, you must first open the file. When you open a file, you must specify an access mode indicating whether you plan to read, write, or some combination of the two. The access mode specified by the second argument to fopen( ) or freopen( ) determines what input and output operations the new stream permits. The permissible values of the mode string are restricted. The first character in the mode string is always r for "read," w for "write," or a for "append," and in the simplest case, the string contains just that one character. However, the mode string may also contain one or both of the characters + and b (in either order: +b has the same effect as b+).When you have finished using a file, close it to release resources.

The entire operations are described step by step:

7.3.1 File Pointer

How will we specify that we want to access a particular data file? It would theoretically be possible to mention the name of a file each time it was desired to read from or write to it. But such an approach would have a number of drawbacks. Instead, the usual approach (and the one taken in C's stdio library) is that you mention the name of the file once, at the time you *open* it. Thereafter, you use some little token--in this case, the *file pointer*--which keeps track (both for your sake and the library's) of which file you're talking about. Whenever you want to read from or write to one of the files you're working with, you identify that file by using its file pointer (that is, the file pointer you obtained when you opened the file). As we'll see, you store file pointers in variables just as you store any other data you manipulate, so it is possible to have several files open, as long as you use distinct variables to store the file pointers.

Declare a variable to store a file pointer like this:

Syntax:	FILE * <file pointer name>;
Example:	FILE *fp;

The type FILE is predefined for you by <stdio.h>. It is a data structure which holds the information and the standard I/O library needs to keep track of the file for you. For historical reasons, you declare a variable which is a pointer to this FILE type. The name of the variable can (as for any variable) be anything you choose; it is traditional to use the letters fp in the variable name (since we're talking about a file pointer). If you were reading from two files at once, you'd probably use two file pointers:

Example:	FILE *fp1, *fp2;

7.3.2 File Open Modes

There are number of several modes to open a file. The modes are given below:

Open Mode	Effect
"r"	Open text file for reading only
"w"	Truncate to 0 length, if existent, or create text file for writing only
"a"	Append; open or create text file only for writing at end of file
"r+"	Open text file for update (reading and writing)
"w+"	Truncate to 0 length, if existent, or create text file for update
"a+"	Append; open or create text file for update, writing at end of file
"rb+"	Open text file for update (reading and writing) over binary file
"wb+"	Truncate to 0 length, if existent, or create text file for update over binary file
"ab+"	Append; open or create text file for update, writing at end of file over binary file

a) Binary Mode:

In ***binary mode***, your program can access every byte in the file. When you use one of C's read functions, it "sees" the file exactly as it is on disk, character-for-character. ***Any file*** may be opened in binary mode.

b) Text Mode:

Text mode is used ***only*** for text files. When a file is opened in text mode, some characters *may* be "hidden" from your program. The characters are still in the file but your program just can't "see" them. You can think of this as a "filter" between the file and your program.

The ANSI standard for C does not specify what filtering should take place when a file is opened in text mode; it's compiler-dependent. For compilers used in the DOS/Windows environment, a file opened in text mode will normally have its **carriage-returns filtered out**.

7.3.3 File Open Functions

To open a file from the disk, we must require an opening function along with the appropriate opening mode. The standard library provides the function **fopen( )** to open a file. For special cases, the **freopen( )** and **tmpfile( )** functions also open files.

Syntax: FILE *fopen(const char **filename*, const char **mode*);

This function opens the file whose name is specified by the string *filename*. The filename may contain a directory part. The second argument, *mode*, is also a string, and specifies the access mode. The possible access modes are described in the next section. The fopen( ) function associates the file with a new stream.

Syntax: FILE *freopen(const char **filename*, const char **mode*,
FILE * restrict *stream*);

This function redirects a stream. Like fopen( ), freopen( ) opens the specified file in the specified mode. However, rather than creating a new stream, freopen( ) associates the file with the existing stream specified by the third argument. The file previously associated with that stream is closed. The most common use of freopen( ) is to redirect the standard streams, stdin, stdout, and stderr.

Syntax: FILE *tmpfile(void);

The tmpfile( ) function creates a new temporary file whose name is distinct from all other existing files, and opens the file for binary writing and reading. If the program is terminated normally, the file is automatically deleted.

7.3.4 File Read & Write Functions

There are a number of functions to read data from file and write data to the file.

a) Reading strings

The following functions allow you to read a string from a stream:

```
char *fgets( char *buf, int n, FILE *fp );

wchar_t *fgetws( wchar_t *buf, int n, FILE *fp );
```

The functions fgets( ) and fgetws( ) read up to *n* - 1 characters from the input stream referenced by *fp* into the buffer addressed by *buf*, appending a null character to terminate the string. If the functions encounter a newline character or the end of the file before they have read the maximum number of characters, then only the characters read upto that point are read into the buffer. The newline character '\n' (or, in a wide-oriented stream, L'\n') is also stored in the buffer if read.

All two functions return the value of their argument *buf*, or a null pointer if an error occurred, or if there were no more characters to be read before the end of the file.

b) Reading characters

Use the following functions to read characters from a file:

```
int fgetc( FILE * fp );

int getc( FILE *fp );

wint_t fgetwc( FILE *fp );

wint_t getwc( FILE *fp );
```

The fgetc( ) function reads a character from the input stream referenced by fp. The return value is the character read, or EOF if an error occurred. The macro getc( ) has the same effect as the function fgetc( ). The macro is commonly used because it is faster than a function call. However, if the argument *fp* is an expression with side effects, then you should use the function instead, because a macro may evaluate its argument more than once.

c) Reading mixed data types

Use the following functions to read data from a file:

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

The function fscanf() reads data from the given file *stream* in a manner exactly like scanf(). The return value of fscanf() is the number of variables that are actually assigned values, or **EOF** if no assignments could be made.

The fread( ) function reads upto *n* objects whose size is *size* from the stream referenced by *fp*, and stores them in the array addressed by *buffer*:

```
size_t fread( void *buffer, size_t size, size_t n, FILE *fp );
```

The function's return value is the number of objects transferred. A return value less than the argument *n* indicates that the end of the file was reached while reading, or that an error occurred.

d) Writing characters

The following functions allow you to write individual characters to a stream:

```
int fputc( int c, FILE *fp );
int putc( int c, FILE *fp);
wint_t fputwc( wchar_t wc, FILE *fp );
wint_t putwc( wchar_t wc, FILE *fp );
```

The function fputc( ) writes the character value of the argument *c* to the output stream referenced by *fp*. The return value is the character written, or EOF if an error occurred. The macro putc( ) has the same effect as the function fputc( ). If either of its arguments is an expression with side effects, then you should use the function instead, because a macro might evaluate its arguments more than once.

e) Writing strings

Use the following functions to write a null-terminated string to a stream:

```
int fputs( const char *s, FILE *fp );

int fputws( const wchar_t *s, FILE *fp );
```

The two functions have some features in common as well as certain differences:

fputs( ) and fputws( ) write the string *s* to the output stream referenced by *fp*. The null character that terminates the string is not written to the output stream.

All these functions return EOF (not WEOF) if an error occurred, or a non-negative value to indicate success.

f) Writing mixed data types

The fwrite( ) function sends *n* objects whose size is *size* from the array addressed by *buffer* to the output stream referenced by *fp*:

```
size_t fwrite( const void *buffer, size_t size, size_t n, FILE *fp );
```

Again, the return value is the number of objects written. A return value less than the argument *n* indicates that an error occurred.

Because the fwrite( ) functions do not deal with characters or strings as such, there are no corresponding functions for wide-oriented streams.

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

The fprintf() function sends information (the arguments) according to the specified *format* to the file indicated by *stream*. fprintf() works just like printf() as far as the format goes. The return value of fprintf() is the number of characters outputted, or a negative number if an error occurs.

7.3.5 File Close Functions

To close a file, use the fclose( ) function. The prototype of this function is:

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

The function flushes any data still pending in the buffer to the file, closes the file, and releases any memory used for the stream's input and output buffers. The fclose( ) function returns zero on success, or EOF if an error occurs.

When the program exits, all open files are closed automatically. Nonetheless, you should always close any file that you have finished processing. Otherwise, you risk losing data in the case of an abnormal program termination. Furthermore, there is a limit to the number of files that a program can open at one time; the number of allowed open files is greater than or equal to the value of the constant FOPEN_MAX.

7.4 Examples

Q1) Write a program that copy records to new records.

[File Name: u7q1.c]

```
// Copy records to a new file, filtering out those with the key value 0.  // -----
-----------------------------------------------------
  #include <stdio.h>
  #include <stdlib.h>
```

```
#define ARRAY_LEN 100 // Max. number of records in the buffer.
                              // A structure type for the records:
typedef struct {
            long key;
            char name[32];
            /* ... other fields in the record ... */
               } Record_t;

   char inFile[ ]  = "records.dat",          // Filenames.
   outFile[ ] = "packed.dat";

   // Terminate the program with an error message:

void error_exit( int status, const char *error_msg )
{
  fputs( error_msg, stderr );
  exit( status );
}

int main( )
{
  FILE *fpIn, *fpOut;
  Record_t record, *pArray;
  unsigned int i;

  if (( fpIn = fopen( inFile, "rb" )) == NULL )       // Open to read.
    error_exit( 1, "Error on opening input file." );

  else if (( fpOut = fopen( outFile, "wb" )) == NULL )
                                                      // Open to write.
    error_exit( 2, "Error on opening output file." );
```

```
  else if (( pArray = malloc( ARRAY_LEN )) == NULL )
                                                    // Create the
    error_exit( 3, "Insufficient memory." );        // buffer.

  i = 0;                          // Read one record at a time:
  while ( fread( &record, sizeof(Record_t), 1, fpIn ) == 1 )
  {
    if ( record.key != 0L )         // If not marked as deleted ...
    {                             // ... then copy the record:
      pArray[i++] = record;
      if ( i == ARRAY_LEN )               // Buffer full?
      {                                   // Yes: write to file.
        if ( fwrite( pArray, sizeof(Record_t), i, fpOut) < i )
          break;
        i = 0;
      }
    }
  }
  if ( i > 0 )                    // Write the remaining records.
    fwrite( pArray, sizeof(Record_t), i, fpOut );

  if ( ferror(fpOut) )                        // Handle errors.
    error_exit( 4, "Error on writing to output file." );
  else if ( ferror(fpIn) )
    error_exit( 5, "Error on reading input file." );

  return 0;
}
```

Q2) Write a C program that will perform all such operations like append, list, modify, insert and delete records over file.

[File Name: u7q2.c]

Solution:

```
#include<stdio.h>
#include<string.h>
typedef struct
{
char name[40];
int roll;
}rec;

FILE *p,*q;     /* File Pointer Declaration */

void main()
{
char ch;
void insert(),delete1(),append(),list(),modify();  // Function Prototype
do{
  clrscr();
  printf("\n\n\n\t\t FILE OPERATION\n");
  printf("\t\t1.  Append\n\t\t2.  List\n\t\t3.  Modify\n\t\t4.  Delete\n\t\t5.
Insart\n\t\t6. Exit\n");
  printf("\n Your Choice please.....");
  fflush(stdin);
  ch=getchar();
  switch(ch)
  {
```

```
 case '1':     // Append Records
   append();
   break;
 case '2':   //  List Records
   list();
   getch();
   break;
 case '3': // Modify Records
   modify();
   break;
 case '4':   // Delete Records
   delete1();
   break;
 case '5':   // Insert Records
   insert();
   break;
 case '6':  // Exit From Program
   exit(0);
   break;
 default :
   printf("Invalid Choice ");    getch();
  }
 }while(1);
}
/************************ end of main() ***********/
 void append()   // Function for append records
 {
  rec stu;
  p=fopen("data.dbf","ab");
```

```
 if (p=='\0')
  { printf("unable to open file"); getch(); }
 do{
 printf("Enter name [ x ] to exit ");
 fflush(stdin); gets(stu.name);
 if(strcmpi(stu.name,"x")==0)  break;
 printf("Enter Roll..... ");
 scanf("%d",&stu.roll);
 fwrite(&stu,sizeof(stu),1,p);
 }while(1);
 fclose(p);
}
//******************************** list () **********************
 void list()
 {
 rec stu;
 p=fopen("data.dbf","rb");
 clrscr();
 printf("ROLL        NAME \n");
 printf("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n");
 while(fread(&stu,sizeof(stu),1,p)==1)
 printf(" %-10d %-30s\n",stu.roll,stu.name);
 printf("\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n");
 fclose(p);
 }
//***************************** modify() ******************************
 void modify()
 {
 rec stu;
```

```
 int rl,flag=0;
 clrscr();
 printf("Enter roll for Modification");
 scanf("%d",&rl);
 p=fopen("data.dbf","rb+");
 while(fread(&stu,sizeof(stu),1,p)==1)
 {
  if(stu.roll==rl)
  {
   printf("*** Old Values *** \n");
   printf(" NAME  : %s\n",stu.name);
   printf(" ROLL  : %d\n",stu.roll);
   printf("****************** \n");
   printf(" New Name : ");fflush(stdin); gets(stu.name);
   printf(" New Roll : ");scanf("%d",&stu.roll);
   fseek(p,-(long)sizeof(stu),1);
   fwrite(&stu,sizeof(stu),1,p);
   flag=1;
   }
 }
  if(flag==0)
  { printf("No such record is found"); getch(); }
 fclose(p);
 }
/***************************** insert() *****************************/
 void insert()
 {
 rec stu,temp;
 int rl,flag=0;
```

```
 clrscr();
 printf("Enter The roll after which the record will be inserted.. ");
 scanf("%d",&rl);
 p=fopen("data.dbf","rb+");
 q=fopen("temp","w");
 while(fread(&stu,sizeof(stu),1,p)==1)
 {
    fwrite(&stu,sizeof(stu),1,q);
    if(rl==stu.roll)
    {
        printf("Enter name : ");fflush(stdin); gets(temp.name);
        printf("Enter roll : ");scanf("%d",&temp.roll);
        fwrite(&temp,sizeof(temp),1,q);
        flag=1;
    }
 }
 if(flag==0)
  {  printf("The roll is not Matched"); getch();}
 fclose(p); fclose(q);
 remove("data.dbf");
 rename("temp","data.dbf");
}
//************************************  delete1() **********************
 void delete1()
 {
 rec stu,temp1;
 int rol,flag=0;
 printf("Enter roll No. to be deleted "); fflush(stdin); scanf("%d",&rol);
 p=fopen("data.dbf","rb+");
```

```
 q=fopen("temp","w");
 while(fread(&stu,sizeof(stu),1,p)==1)
  {
    if(stu.roll!=rol)   fwrite(&stu,sizeof(stu),1,q);
    if(stu.roll==rol)   flag=1;
  }

  if(flag==0)
          printf("No Match Found");
  else
          printf("Record Successfully Deleted");
 fclose(p); fclose(q);
 remove("data.dbf");
 rename("temp","data.dbf");
 getch();
 }
//---------------------- end of program ----------------------------------------
```

7.5 Summary

A file represents a sequence of bytes. Through the program, we can store data into the disk. By using pointers, we can do the various file operations. Each of the functions that open files, namely fopen( ), freopen( ), and tmpfile(), returns a pointer to a FILE object for the stream associated with the file being opened. Once you have opened a file, you can call functions to transfer data and to manipulate the stream. This unit covers all file operations.

Self Assessment Questions

1. In Language C, any number of files can be opened at a time. (True / False)
2. “w+” and “wb+” are same about it operation. (True / False)
3. FILE is a function. (True / False)
4. fopen() is a function through which we can open a disk file. (True / False)
5. putc() and puts() are the same in nature. (True / False)

7.6 Terminal Questions

1. What is data file?
2. How many modes are used to open a file?
3. Write the syntax of fread() function.
4. What is FILE?
5. If we want to open multiple files at a time then how many file pointers are needed?
6. Write a program to accept name and store into file in short form as Sikkim Manipal University → SMU.
7. Write a program that checks how many characters, words, lines are present in the text file.
8. Write a program that reads numbers from a file and find the greatest and least among the numbers.

7.7 Answer to Self Assessment and Terminal Questions

Self Assessment Questions

1. False
2. True
3. False
4. True
5. False

Terminal Questions

1. 7.2
2. 7.3.2
3. 7.3.4 - C
4. 7.3.1
5. 7.3.1