Files

What is a file?

- In C programming, file is a place on your physical disk where information is stored.

Why files are needed?

- When a program is terminated, the entire data is lost. Storing in a file will preserve your data even if the program terminates.
- If you have to enter a large number of data, it will take a lot of time to enter them all.
  However, if you have a file containing all the data, you can easily access the contents of the file using few commands in C.
- You can easily move your data from one computer to another without any changes.

Why files are needed?

- Storage of data in variables and arrays is temporary—such data is lost when a program terminates.
- Files are used for *permanent* retention of data.
- Computers store files on secondary storage devices, such as hard drives, CDs, DVDs and flash drives.

Files and Streams

- C views each file simply as a sequential stream of bytes.
- Each file ends end-of-file marker.
- When a file is opened, a stream is associated with it.
- Three files and their associated streams are automatically opened when program execution begins—the standard input, the standard output and the standard error.
- Streams provide communication channels between files and programs.

Files and Streams

- For example, the standard input stream enables a program to read data from the keyboard, and the standard output stream enables a program to print data on the screen.
- Opening a file returns a pointer to a FILE structure (defined in <stdio.h>) that contains information used to process the file.
- Each array element contains a file control block (FCB) that the operating system uses to administer a particular file.
- The standard input, standard output and standard error are manipulated using file pointers stdin, stdout and stderr.

Files and Streams

- The standard library provides many functions for reading data from files and for writing data to files.
- Function `fgetc`, like `getchar`, reads one character from a file.
- Function `fgetc` receives as an argument a `FILE` pointer for the file from which a character will be read.
- The call `fgetc( stdin )` reads one character from `stdin`—the standard input.
- This call is equivalent to the call `getchar()`.
- Function `fputc`, like `putchar`, writes one character to a file.
- Function `fputc` receives as arguments a character to be written and a pointer for the file to which the character will be written.

Files and Streams

- The function call `fputc( 'a', stdout )` writes the character `'a'` to `stdout`—the standard output.
- This call is equivalent to `putchar( 'a' )`.
- Several other functions used to read data from standard input and write data to standard output have similarly named file-processing functions.
- The `fgets` and `fputs` functions, for example, can be used to *read a line from a file* and *write a line to a file*, respectively.

Types of Files

- When dealing with files, there are two types of files you should know about:
- Text files
- Binary files

Text files

- Text files are the normal .txt files that you can easily create using Notepad or any simple text editors.
- When you open those files, you'll see all the contents within the file as plain text. You can easily edit or delete the contents.
- They take minimum effort to maintain, are easily readable, and provide least security.

Binary files

- Binary files are mostly the .bin files in your computer.
- Instead of storing data in plain text, they store it in the binary form (0's and 1's).
- They can hold higher amount of data, are not readable easily and provides a better security than text files.

File Operations

- In C, you can perform four major operations on the file, either text or binary:
- Creating a new file
- Opening an existing file
- Closing a file
- Reading from and writing information to a file

Working with files

- When working with files, you need to declare a pointer of type file.
- This declaration is needed for communication between the file and program.
- FILE *fptr;

Working with files

- We initialize the pointer **fp** to **NULL** as a precaution against premature dereferencing.
- If our program accesses data at **fp** before the connection to the file is open, our program may generate a segmentation fault.

Opening a file - for creation and edit

- Opening a file is performed using the library function in the **"stdio.h"** header file: fopen().
- The syntax for opening a file in standard I/O is:
- FILE *fopen(const char filename[], const char mode[]);

Opening a file

- This prototype tells you that fopen() returns a pointer to type FILE, which is a structure declared in stdio.h.
- However, for each file that you want to open, you must declare a pointer to type FILE.
- When you call fopen(), that function creates an instance of the FILE structure and returns a pointer to that structure.
- You use this pointer in all subsequent operations on the file.
- If fopen() fails, it returns NULL. Such a failure can be caused, for example, by a hardware error.

Opening a file

- The argument filename is the name of the file to be opened.
- The argument mode specifies the mode in which to open the file. In this context, mode controls whether the file is binary or text and whether it is for reading, writing, or both.

Modes of opening a file

mode	Meaning
• r	Opens the file for reading. If the file doesn't exist, fopen() returns NULL.
• w	Opens the file for writing. If a file of the specified name doesn't exist, it is created. If a file of the specified name does exist, it is deleted without warning, and a new, empty file is created.
• a	Opens the file for appending. If a file of the specified name doesn't exist, it is created. If the file does exist, new data is appended to the end of the file.

Contd….

Mode	Meaning
• r+	Opens the file for reading and writing. If a file of the specified name doesn't exist, it is created. If the file does exist, new data is added to the beginning of the file, overwriting existing data.
• w+	Opens the file for reading and writing. If a file of the specified name doesn't exist, it is created. If the file does exist, it is overwritten.
• a+	Opens a file for reading and appending. If a file of the specified name doesn't exist, it is created. If the file does exist, new data is appended to the end of the file.

Common Programming Errors

- Opening a file for reading or writing without having been granted the appropriate access rights to the file is an error.
- Opening a file for writing when no space is available is a runtime error.

Contd…

- The default file mode is text.
- To open a file in binary mode, you append a b to the mode argument.
- Thus, a mode argument of a would open a text-mode file for appending, whereas ab would open a binary-mode file for appending.

Creating a file

```
1  // Fig. 11.2: fig11_02.c
2  // Creating a sequential file
3  #include <stdio.h>
4
5  int main( void )
6  {
7     unsigned int account; // account number
8     char name[ 30 ]; // account name
9     double balance; // account balance
10
11    FILE *cfPtr; // cfPtr = clients.dat file pointer
12
13    // fopen opens file. Exit program if unable to create file
14    if ( ( cfPtr = fopen( "clients.dat", "w" ) ) == NULL ) {
15       puts( "File could not be opened" );
16    } // end if
17    else {
18       puts( "Enter the account, name, and balance." );
19       puts( "Enter EOF to end input." );
20       printf( "%s", "? " );
21       scanf( "%d%29s%lf", &account, name, &balance );
22
```

Creating a sequential file. (Part 1 of 2.)

Contd…

```
23          // write account, name and balance into file with fprintf
24          while ( !feof( stdin ) ) {
25             fprintf( cfPtr, "%d %s %.2f\n", account, name, balance );
26             printf( "%s", "? " );
27             scanf( "%d%29s%lf", &account, name, &balance );
28          } // end while
29
30          fclose( cfPtr ); // fclose closes file
31       } // end else
32    } // end main
```

```
Enter the account, name, and balance.
Enter EOF to end input.
? 100 Jones 24.98
? 200 Doe 345.67
? 300 White 0.00
? 400 Stone -42.16
? 500 Rich 224.62
? ^Z
```

Creating a sequential file. (Part 2 of 2.)

Contd…

- Now let’s examine this program.
- Line 11 states that cfptr is a *pointer to a FILE structure*.
- A C program administers each file with a separate FILE structure.
- Line 14 names the file—"clients.dat"—to be used by the program and establishes a “line of communication” with the file.
- The file pointer cfPtr is assigned *a pointer to the FILE structure* for the file opened with fopen.
- Function fopen takes two arguments: a filename (which can include path information leading to the file’s location) and a file open mode.

Contd…

- The file open mode `"w"` indicates that the file is to be opened for writing.
- If a file *does not* exist and it's opened for writing, `fopen` creates the file.

Contd…

- If an existing file is opened for writing, the contents of the file are *discarded without warning*.
- In the program, the `if` statement is used to determine whether the file pointer `cfPtr` is `NULL` (i.e., the file is not opened).
- If it's `NULL`, the program prints an error message and terminates.
- Otherwise, the program processes the input and writes it to the file.

Contd…

- The program prompts the user to enter the fields for each record or to enter *end-of-file* when data entry is complete.
- Line 24 uses function feof to determine whether the end-of-file indicator is set for the file to which stdin refers.
- The *end-of-file* indicator informs the program that there's no more data to be processed.
- In Fig. 2, the *end-of-file indicator* is set for the standard input when the user enters the end-of-file key combination.
- The argument to function feof is a pointer to the file being tested for the end-of-file indicator (stdin in this case).

Contd…

- The function returns a nonzero (true) value when the end-of-file indicator has been set; otherwise, the function returns zero.
- The `while` statement that includes the `feof` call in this program continues executing while the end-of-file indicator is not set.
- Line 25 writes data to the file `clients.dat`.
- The data may be retrieved later by a program designed to read the file.

Contd…

- Function `fprintf` is equivalent to `printf` except that `fprintf` also receives as an argument a file pointer for the file to which the data will be written.

Contd…

- After the user enters end-of-file, the program closes the clients.dat file with fclose and terminates.
- Function fclose also receives the file pointer (rather than the filename) as an argument.
- *If function fclose is not called explicitly, the operating system normally will close the file when program execution terminates.*
- This is an example of operating system “housekeeping.”

Writing and Reading File Data

- A program that uses a disk file can write data to a file, read data from a file, or a combination of the two.

Formatted File Input and Output

- Formatted file input/output deals with text and numeric data that is formatted in a specific way.
- It is directly analogous to formatted keyboard input and screen output done with the printf() and scanf() functions.

Contd…

- Formatted file output is done with the library function fprintf().
- The prototype of fprintf() is in the header file stdio.h, and it reads as follows:
- The prototype for this library function is

```
int fprintf(FILE *, const char [], ...);
```

Contd…

- The first parameter receives the address of the **FILE** object.
- The second parameter receives the address of the string literal that specifies the format.

Example

- /* Demonstrates the fprintf() function. */

```
#include<stdlib.h>
#include<stdio.h>
int main( void )
{
        FILE *fp;
        float data[5];
        int count;
        char filename[20];
        puts(“Enter 5 floating-point numerical values.”);
}
```

Example

```
for (count = 0; count < 5; count++)
        scanf("%f", &data[count]);
/* Get the filename and open the file. */
puts("Enter a name for the file.");
gets(filename);
if ( (fp = fopen(filename, "w")) == NULL)
{
        fprintf(stderr, "Error opening file %s.",     filename);

        exit(1);
}
```

Example

```
/* Write the numerical data to the file and to stdout. */
for (count = 0; count < 5; count++)
        {
                fprintf(fp, “\ndata[%d] = %f”, count,
data[count]);
        }
        fclose(fp);
        printf(“\n”);
        return 0;
}
```

Example

- Enter 5 floating-point numerical values.
  3.14159
  9.99
  1.50
  3.
  1000.0001

Enter a name for the file.
numbers.txt
data[0] = 3.141590
data[1] = 9.990000
data[2] = 1.500000
data[3] = 3.000000
data[4] = 1000.000122

Contd…

- You might wonder why the program displays 1000.000122 when the value entered was 1000.0001.
- This isn’t an error in the program.
- It’s a normal consequence of the way C stores numbers internally.
- Some floating-point values can’t be stored exactly, so minor inaccuracies such as this one sometimes result.

Formatted File Input

- For formatted file input, use the fscanf() library function, which is used like scanf().

- The prototype for fscanf() is

```
int fscanf(FILE *fp, const char [], ...);
```

Contd…

- The first parameter receives the address of the **FILE** object.
- The second parameter receives the address of the string literal that specifies the format.
- This literal contains the conversion specifiers to be used in translating the file data to data stored in memory.
- The function fscanf() works exactly the same as scanf(), except that characters are taken from the specified stream rather than from stdin

Example

- /* Reading formatted file data with fscanf(). */

```
#include<stdlib.h>
#include<stdio.h>
int main( void )
{
        float f1, f2, f3, f4, f5;
        FILE *fp;
        if ( (fp = fopen("INPUT.TXT", "r")) == NULL)
        {
                fprintf(stderr, "Error opening file.\n");
                exit(1);
        }
        fscanf(fp, "%f %f %f %f %f", &f1, &f2, &f3, &f4, &f5);
        printf("The values are %f, %f, %f, %f, and %f\n.", 18: f1, f2, f3, f4, f5);
        fclose(fp);
        return 0; }
```

Unformatted Input and Output

- It is also possible to read (or write) a single character at a time--this can be useful if you wish to perform character-by-character input (for instance, if you need to keep track of every piece of punctuation in a file it would make more sense to read in a single character than to read in a string at a time.)
- The fgetc function, which takes a file pointer, and returns an int, will let you read a single character from a file.
- The fgetc() Function: reads a single character from an open file.
- The prototype is int fgetc(FILE **fp*);

Fgetc()

- When called, fgetc() reads characters from fp into memory.
- Characters are read until a newline is encountered or until n-1 characters have been read, whichever occurs first.

Example

```
#include <stdio.h>
int main ()
{
        FILE *fp;
        int c;
        int n = 0;
        fp = fopen("file.txt","r");
        if(fp == NULL)
        {
                perror("Error in opening file");
                return(-1);
        }
```

Example

```
do
{
        c = fgetc(fp);
        if( feof(fp) )
        {
                break ;
        }
        printf("%c", c);
}while(1);
fclose(fp);
return(0);
}
```

Fputc()

- The fputc function allows you to write a character at a time--you might find this useful if you wanted to copy a file character by character. It looks like this:
- int fputc(int c, FILE *fp);

Fputc()

- Note that the first argument should be in the range of an unsigned char so that it is a valid character.
- The second argument is the file to write to.
- On success, fputc will return the value c, and on failure, it will return EOF.

Rewind function

- To retrieve data sequentially from a file, a program normally starts reading from the beginning of the file and reads all data consecutively until the desired data is found.
- It may be desirable to process the data sequentially in a file several times (from the beginning of the file) during the execution of a program.

Rewind function

- The statement
  - `rewind( cfPtr );`

  causes a program's file position pointer—which indicates the number of the next byte in the file to be read or written—to be repositioned to the *beginning* of the file (i.e., byte 0) pointed to by `cfPtr`.
- Rather it's an integer value that specifies the byte in the file at which the next read or write is to occur.
- This is sometimes referred to as the file offset.
- The file position pointer is a member of the `FILE` structure associated with each file.

Rewind function

- The C library function **void rewind(FILE *fp)** sets the file position to the beginning of the file of the given **stream**.
- In other words, to jump to the beginning of a file, instead of disconnecting and re-connecting it, we simply rewind the file.

Example

```
#include <stdio.h>
int main()
{
        char str[] = “Let us learn C";
        FILE *fp; int ch; /* First let's write some content in the file */
        fp = fopen( "file.txt" , "w" );
        fwrite(str , 1 , sizeof(str) , fp );
        fclose(fp);
        fp = fopen( "file.txt" , "r" );
       while(1)
        {
                ch = fgetc(fp);
                if( feof(fp) )
                {
                        break ;
                }
```

Example

```
printf("%c", ch);
}
rewind(fp);
printf("\n");
while(1)
{
        ch = fgetc(fp);
        if( feof(fp) )
        { break ;
        }
        printf("%c", ch);
        }
        fclose(fp);
        return(0);
}
```

Record

- A **record-oriented file system** is a file system where data is stored as collections of records.
- This is in contrast to a byte-oriented file system, where the data is treated as an unformatted stream of bytes.
- There are several different possible record formats; the details vary depending on the particular system.
- In general the formats can be fixed-length or variable length.

Record

- The record is a sequence of characters that ends with a record delimiter. The typical record delimiter is the newline character (**\n**).

Example

```
// Number of Records
// records.c
#include <stdio.h>
int main(void)
{
        FILE *fp = NULL;
        int c, nrecs;
        fp = fopen("produce.txt", "r");
        if (fp != NULL) {
        nrecs = 0;
        do
        {
                c = fgetc(fp);
               if (c != EOF)
```

Example

```
{
        if ((char)c == '\n')
                nrecs++;
                }
        } while (feof(fp) == 0);
printf("%d records on file\n",  nrecs);
fclose(fp);
        }
return 0;
}
```

Contd…

- Since this program determines the number of records in the file by counting the newline characters, to report the correct number of records, the last record in the file must end with a newline character.
- If the last record does not end with a newline character, the count will be off by one.

Fields

- A single piece of information about an object.
- If the object were an Employee, a field would be Firstname, Lastname, or City or State.
- A field holds one element of information within a single record.
- We separate adjacent fields within a record by a field delimiter.

Table

A table is a set of records in which each record contains the same number of fields.