

File Handling

So far you have seen that the data required to run a program is entered through the keyboard. However, data assigned in variables and arrays is only stored temporarily and after execution of the program the data is lost.

So while you are removing a bug from the program and are executing the program, you have to enter the data all over again. When the data size is large, it is convenient to store the data on a file, which can be called by the program whenever needed.

A **file** is a collection of data that resides on the hard disk or on a removable floppy disk, or on a CD, and is completely independent from your C program. It may be a source file (which stores input for your program) or it may be a target file (to store output of your program).

Text files

- When you use a file to store data for use by a program, the file usually consists of textual (alphanumeric) data and is therefore called a **text file**.
- Can be created, updated, and processed by C programs
- Are used for permanent storage of large amounts of data

A file on disk exists whether or not your program is active in memory. It has its own distinct name and distinct location. The data is usually stored in *.txt* format. An example of such a data file is *students.txt* which contains the exam grades of five students. Each line of the file contains the ID of a student followed by his grades in 3 exams.

students.txt

```
861022 65 72 56
851102 78 45 80
860501 55 75 90
841205 75 80 95
850630 40 50 48
```

Here is another example of a data file which contains the names of students and grades obtained by them.

Beta.txt

```
Jason Lee 65 72 56
Robin Faris 78 45 80
William Corman 55 75 90
Mark Weiss 75 80 95
Hoshing Chan 40 50 48
```

In order to connect your C program to a file, you need to:

- 1) Declare a file pointer variable in your program
- 2) Connect that file variable to the actual disk file with an *fopen* function call
- 3) Process the data using commands similar to *scanf* and *printf* .
- 4) Remember to break the connection when you are finished with the file by an *fclose* function call.

All communication between your program and the disk file is accomplished not by using the file's disk name, but by referencing the file pointer variable.

The only place the file's disk name is used is in the *fopen* function call.

To start with you have to declare a FILE* variable in C by writing a line like

```
FILE * infile;
```

The variable `infile` is a pointer to location of the file. You must declare a separate FILE* variable for each of the files that may be open simultaneously. For example, if your application requires you to read information from one file, process it, and then write the processed data to a second file, you will need to declare two FILE* variables. If you choose to call these variables `infile` and `outfile`, the appropriate declaration would be

```
FILE * infile;  
FILE * outfile;
```

The next step would be to associate these variables with actual files. You would also need to indicate whether you want to read data from a file or store data on a file. The *fopen* function determines the mode of communication that your program will use in communicating with the file.

A file can be opened for reading (using "*r*" mode) or opened for writing (using "*w*" mode).

Example 1:

Let us assume that a data file named *students.txt* resides on your disk drive, and that you want to open this file for reading (as an input source for your program), modify it by adding 10 points to each of the grades, and store the modified information in another file say *new.txt*.

After declaring the two file pointer variables as above, make appropriate calls to the *fopen* function as shown below:

```
infile = fopen( "students.txt", "r" );
outfile = fopen( "new.txt" , "w" );
```

Now you can start reading the data from the file into your temporary variables, in a manner similar to reading data from the keyboard. Instead of using *scanf* you now use *fscanf* as shown below to first read the ID of the student:

```
fscanf( infile, "%d", &student_ID);
```

Note the use of file scanf function *fscanf* . Also note that the logical name of the file *infile* must be included in the *fscanf* function. Every *fscanf* statement reads one token from the file (one value). Tokens are separated by spaces. You can follow this by reading the grades of the student, which are the next 3 tokens in the file.

```
fscanf( infile, "%d %d %d", &g1, &g2, &g3);
```

You could as well have written a single *fscanf* statement to read the student ID as well as the grades(4 tokens).

Modify the grades now.

```
g1 = g1 + 10;
g2 = g2 + 10;
g3 = g3 + 10;
```

If you want to print the new grades on the monitor screen

```
printf( "\n%d %d %d %d %d", student_ID, g1, g2, g3);
```

Next to store the grades into the output file *new.txt* you can use the following statement where *outfile* refers to the *new.txt* file which we have opened in the beginning.

```
fprintf( outfile, "\n%d%d%d%d", student_ID, g1, g2,
g3);
```

Of course you need to use a *for* loop to handle all the students. Finally you have to close the opened files by breaking the connection:

```
fclose(infile);
fclose(outfile);
```

Forgetting to close a file may result in corrupting the data stored on the file.

Here is the complete program to handle the file *students.txt*

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

    FILE * infile;
    FILE * outfile;
    int student_ID,j,g1,g2,g3;
    infile = fopen( "students.txt", "r" );
    outfile = fopen( "new.txt" , "w");

    for (j =1; j <= 5; j++)
    {
        fscanf( infile, "%d", &student_ID);
        fscanf( infile," %d %d %d ", &g1, &g2, &g3);
        g1 = g1 + 10;
        g2 = g2 + 10;
        g3 = g3 + 10;
        printf("%d %d %d %d\n", student_ID, g1,
g2,g3);
        fprintf(outfile,"%d %d %d %d \n", student_ID,
g1,g2,g3);
    }
    fclose(infile); // Close both files.
    fclose(outfile);

    return 0;
}
```

students.txt

```
861022 65 72 56
851102 78 45 80
860501 55 75 90
841205 75 80 95
850630 40 50 48
```

new.txt

```
861022 75 82 66
851102 88 55 90
860501 65 85 100
841205 85 90 105
850630 50 60 58
```

Example 2:

In the next example, the input file stores the maximum temperature data for 10 days, and we need to find the average maximum temperature.

max_temp.txt

```
82 78 81 84 86 83 82 80 85 86
```

```
#include <stdio.h>

int main () {

    FILE * infile;
    int temp,j,sum=0;
    double average;
    infile = fopen( "max_temp.txt", "r" );

    for (j =1; j <= 10; j++)
    {
        fscanf( infile, "%d", &temp);
        sum = sum + temp;
    }
    average = sum/10.0;

    printf( "\naverage max temp = %.2f\n", average);
    fclose(infile);
    return 0;
}
average max temp = 82.70
```

Now suppose the data was stored in the file as

max_temp.txt

```
82      78 81      84 86 83 82 80 85 86
```

Let us see what difference it makes to the output.

Here is the output:

```
average max temp = 82.70
```

Thus it makes no difference. Now let the data be stored in the file as

max_temp.txt

```
82 78
81 84
86 83
82 80
85 86
```

Again we find that the file is read in the same way as before, and gives the following output.

average max temp = 82.70

Example 3:

We can process a file even if we do not know the number of days for which the data is stored on the file, provided we make sure that the last value stored is a negative number (assuming that each data value is a positive number).

```
FILE *infile;
int temp = 0, sum = 0, count= 0;

infile = fopen("max_temp.txt", "r");

while (1) {
    fscanf(infile, "%d", &temp);
    if(temp<0)
        break;
    sum += temp;
    count++;
}
average = sum*1.0/count;

printf( "\naverage max temp = %.2f\n", average);
fclose(infile);
return 0;
```

max_temp.txt

```
82 78 81 84 86 83 82 80 85 86 -1
```

average max temp = 82.70

Example 4:

You can also process actual text files containing letters. Here is a program which looks at a text message and converts it into a secret message text, so that the reader cannot guess the true contents of the file.

The secret coding employed here is to replace the first letter of the alphabet by the last letter of the alphabet (in capitals), the second letter by the last but one letter, for example the code replaces a by Z, b by Y, c by X and so on.

It makes use of file functions *fgetc* and *fputc* to handle the characters in a file.

```
#include <stdio.h>

int main() {

    FILE *infile, *outfile;
    int c, codechar;

    // Open both the input and output files.
    infile = fopen("fruit.txt", "r");
    outfile = fopen("fruitcode.txt", "w");

    // Continue reading in characters till the
    //end of the input file.
    while ((c = fgetc(infile)) != EOF) {
        if (isspace(c))
            fputc(c, outfile);

        // Only process alphabetic characters.
        if (isalpha(c)) {

            // Write out the encoded character to
            // the output file.
            codechar = ('Z' - toupper(c)) + 'A';
            fputc(codechar, outfile);

        }
    }
    fclose(infile); // Close both files.
    fclose(outfile);
}
```

Here is the input data file:

fruit.txt

```
mango peach orange apple
```

Here is the output produced by the program.

frucodet.txt

```
NZMTL KVZXS LIZMTV ZKKOV
```