

ICS103 Programming in C

Lecture 8: Data Files

Outline

- Why data files?
- Declaring FILE pointer variables
- Opening data files for input/output
- Scanning from and printing to data files
- Closing input and output files
- Echo Prints vs. Prompts
- Handling File not found error
- EOF-controlled Loops

Why data files?

- So far, all our examples obtained their input from the keyboard and displayed their output on the Screen.
- However, in many real-life applications, the input data is so much that it will be inconvenient to expect the user to type it each time the program is run.
 - For example: A program to generate employee pay slip from employee data.
- Similarly, there are many applications where the output will be more useful if it is stored in a file rather than the screen.
 - For example: In the program that generates pay slip, how can we print the pay slips and distribute them to the employees if the output is printed on the screen?
- The good news is that C allows a programmer to direct his program to use data files, both for input and for output.

Steps For Using Data Files

- The process of using data files for input/output involves four steps as follows:
 1. Declare variables of type **FILE** to represent the files
 2. Open the files for reading/writing using the **fopen** function.
 3. Read/write from/to the files using the **fscanf** and **fprintf** functions.
 4. Close the files after processing the data using the **fclose** function.
- In what follows, we explain each of these steps.

Declaring FILE pointer variables

- The first step in using data files for input/output is to declare variables to represent the files. This is done as follows:

`FILE *infile, //pointer variable for the input file`

`*outfile; //pointer variable for the output file`

- Note that the type for declaring file variables is **FILE** in upper case.
- Also note the use of ‘*’ just before the variable identifiers.
 - This is used to indicate that the variables are pointer variables – they store memory addresses.
 - We shall learn more about pointer variables later in the course.

Opening data files for input/output

- The second step is to open the input file for reading and the output file for writing.
- Suppose we have our data in a file named, `data.txt`. Then to open it for reading the data inside we use:

```
infile = fopen("data.txt", "r");
```
- If the operation succeeds, the address of the file is stored in the variable `infile`. If it fails – for example if the file is not found, the value **NULL** is assigned to `infile`
- The `"r"` is used to indicate the purpose of opening the file – reading. `"w"` is used for opening output file as we show next.
- To open a file, `result.txt`, for the output, we write:

```
outfile = fopen("result.txt", "w");
```
- If the operation succeeds, `outfile` is assigned the address of the file. If it fails, **NULL** is assigned.
- If the file does not exist, the system will create it. If it exists it is overwritten, thus losing any data it may contain.

Scanning from and printing to data files

- The third step is to scan data from the input file and print the result into the output file.
- Suppose that the input file, `data.txt`, contains a double value representing distance in miles. Then to scan this value we use:
`miles = fscanf(infile, "%lf", &miles);`
- The `fscanf` function works in the same way as `scanf` except that it has an additional argument – the variable representing the input file.
- After computing the result, `kms = KMS_PER_MILES * miles;` we need to print the result into the output file as follows:
`fprintf(outfile, "That equals %.2f kilometers.\n", kms);`
- Again, `fprintf` function works similar to `printf` except that it has an additional argument – the variable representing the output file.
- Like `scanf` and `printf`, `fscanf` and `fprintf` are also in the `stdio` library. So we use the same `#include <stdio.h>`

Closing input and output files

- The final step in using data files is to close the files after you finish using them.
- The `fclose` function is used to close both input and output files as shown below:
 - `fclose(infile);`
 - `fclose(outfile);`
- **Warning:** It is a common error to forget to close files – this is a problem especially for output files.
 - The system may delay writing data to output files until they are closed. So if you forget to close the file you may find no data in the file even though your program actually prints the data.
- It is possible for a program to open a file for output, prints some result, close the file, and then open the same file for input.

Example 1: Miles to Kilometers conversion using data files

```
#include <stdio.h>
#define KMS_PER_MILE 1.609

int main(void) {
    double kms, miles;
    FILE *infile, *outfile;

    infile = fopen("data.txt","r");
    outfile = fopen("result.txt","w");

    fscanf(infile, "%lf", &miles);
    fprintf(outfile, "The distance in miles is %.2f.\n", miles);

    kms = KMS_PER_MILE * miles;

    fprintf(outfile, "That equals %.2f kilometers.\n", kms);
    fclose(infile);
    fclose(outfile);
    return (0);
}
```

To run this program, you need to first create a file using any text editor, such as Notepad, type a double value in the file and save it as data.txt.

Echo Prints vs. Prompts

- In the last example program, `fscanf` gets a value for miles from the data file.
- Because the program input comes from a data file, there is no need to precede this statement with a prompting message.
- Instead, we follow the call to `fscanf` with the statement

```
printf("The distance in miles is %.2f.\n",miles);
```
- This statement **echo prints** or displays the value just stored in `miles`.
- Without it, we would have no easy way of knowing what value `fscanf` obtained for miles.
- Whenever you read input from a data file, make sure to use echo print instead of a prompt.

Echo Prints vs. Prompts ...

```
#include <stdio.h>
#define KMS_PER_MILE 1.609

int main(void) {
    double kms, miles;
    FILE *infile, *outfile;
    infile = fopen("data.txt","r");
    outfile = fopen("result.txt","w");

    //Scan and echo the distance in miles
    fscanf(infile, "%lf", &miles);
    fprintf(outfile, "The distance in miles is %.2f.\n", miles);

    kms = KMS_PER_MILE * miles;

    fprintf(outfile, "That equals %.2f kilometers.\n", kms);
    fclose(infile);
    fclose(outfile);
    return (0);
}
```

Handling File not found error

- A common error in using data files is forgetting to create the input file before running the program.
- Of course this will make the program generate a run-time error.
- Recall that the **fopen** function assigns **NULL** to the file pointer variable if the open operation fails.
- A common practice is to check the value of the file variable immediately after the **fopen** statement and stop the program right there if the variable has a **NULL** value.
- You can stop a program at any point by calling the **exit** function.

```
if (infile==NULL) { // to check if input file is opened properly or not  
    printf("Sorry, input file not found");  
    exit(1); // terminates the program  
}
```

- Note: **exit** should be called with an argument of 1. This tells the operating system that the program stops due to an error. 0 (used with **return** (0)) indicates success

Handling File not found error ...

```
#include <stdio.h>
#define KMS_PER_MILE 1.609

int main(void) {
    double kms, miles;
    FILE *infile, *outfile;
    infile = fopen("data.txt","r");
    if (infile==NULL) { // to check if input file is opened properly or not
        printf("Sorry, input file not found");
        exit(1); // terminates the program
    }

    outfile = fopen("result.txt","w");
    fscanf(infile, "%lf", &miles);
    fprintf(outfile, "The distance in miles is %.2f.\n", miles);

    kms = KMS_PER_MILE * miles;

    fprintf(outfile, "That equals %.2f kilometers.\n", kms);
    fclose(infile);
    fclose(outfile);
    return (0);
}
```

EOF-controlled Loops

- The last example reads a single value from the input file – this can easily be provided by the user.
- A more common application of data files is where the input data is large – for example, finding class average from grades of students in a quiz.
- The grades are normally stored in an input file and the program needs to read them one at a time in a loop, until all of them are read and added to a sum variable.
- The question here is, how many times should the program scan for the values?
- We may use counting loop if we know how many students are in the class, but this will require changing the program to work for a different class size.
- The good news is, **fscanf** returns a special value, **EOF**, when it encounters end of file – no more data values to read.
- We can take advantage of this by using it as a condition for terminating our loops – reads as long as we have not reached end of file. Such loops are commonly called EOF-controlled Loops.¹⁴

Example: EOF-controlled Loops

/*finds the sum and average score of a class in a quiz.

The scores are read from an input file, scores.txt */

```
#include <stdio.h>
```

```
int main (void) {
```

```
    FILE *infile;
```

```
    double score, sum=0, average;
```

```
    int count=0, input_status;
```

```
    infile = fopen("scores.txt", "r");
```

```
    input_status = fscanf(infile, "%lf", &score);
```

```
    while (input_status != EOF)
```

```
    {
```

```
        printf("%f\n ", score);
```

```
        sum += score;
```

```
        count++;
```

```
        input_status = fscanf(infile, "%lf", &score);
```

```
    }
```

```
    average = sum / count;
```

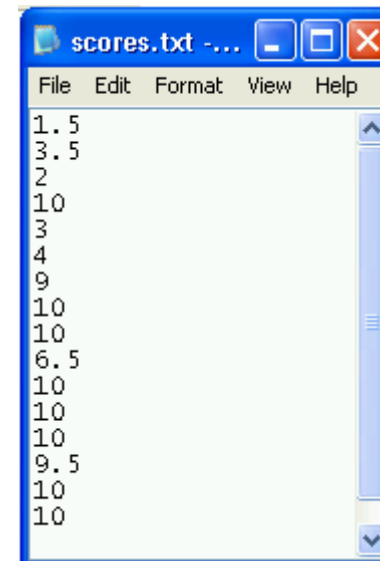
```
    printf("\nSum of the scores is %f\n", sum);
```

```
    printf("Average score is %.2f\n", average);
```

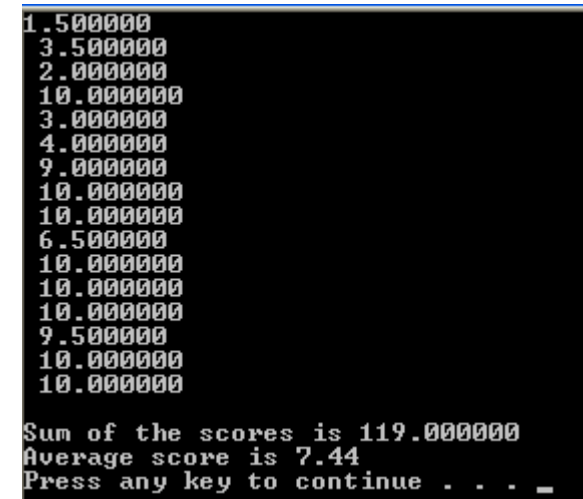
```
    fclose(infile);
```

```
    system("pause");
```

```
    return 0;
```



```
scores.txt
File Edit Format View Help
1.5
3.5
2
10
3
4
9
10
10
6.5
10
10
10
9.5
10
10
```



```
1.500000
3.500000
2.000000
10.000000
3.000000
4.000000
9.000000
10.000000
10.000000
6.500000
10.000000
10.000000
10.000000
9.500000
10.000000
10.000000
Sum of the scores is 119.000000
Average score is 7.44
Press any key to continue . . . _
```