**ICS 103: Computer Programming in C**

**Lab #8: Text Data Files**

Objective:

Learning how to use text data files for input and output.

**Why Data Files?**

When dealing with a large amount of data, it may be more convenient to read inputs and produce outputs, to and from [files](https://wiki.cs.auckland.ac.nz/enggen131/index.php/Files), rather than manually typing in inputs and printing outputs to the screen. Data files also provide data persistence.

A typical plain text file contains several lines of text that are each followed by an End-of-Line (EOL) character. The last line of a text-file may or may not be terminated by an End-of-Line character. An End-of-File (EOF) marker is placed after the final character of a text-file; this signals the end of the file.

**Using data files for input and output**

In C, the process of using data files for input/output involves the following four steps:

1. Declare pointer variables of type FILE\* to represent the files within the C program.
2. Open the files for reading/writing/appending using the **fopen** function.
3. Read/write/append from/to the files using file input/output functions.
4. Close the files after processing the data using the **fclose** function

1- Declare pointer variables of type FILE\* to represent the files within the C program.

The declaration is of the form:

**FILE \*pointerVariableName;**

where **pointerVariableName** is any valid C variable name.

**Example:**

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

**\*outfile, //pointer variable outfile for the output file**

**\*myfile; //pointer variable myfile for append file**

2- Open the files for reading/writing/appending using the **fopen** function.

The **fopen** function creates a correspondence between the pointerVariableName for the file and the file's external name. The syntax of **fopen** is:

**pointerVariableName = fopen(fileExternalName, mode);**

**Examples:**

**infile = fopen("data.txt", "r"); // open data.txt for reading**

**outfile = fopen("result.txt", "w"); // open result.txt for writing**

**myfile = fopen("news.txt", "a"); // open news.txt for appending**

**Note:**

* The prototype for **fopen** is defined in the **stdio.h** header file.
* **fopen** returns the constant **NULL** if the operation of opening the file is not successful; otherwise the starting **address** of the file is returned.
* If a file is opened for reading and it does not exist a run-time error occurs.
* If a file is opened for writing and it does not exist an empty file is created. If it exists, it is destroyed to an empty file.
* If a file is opened for appending and it does not exist it is created. If it exists, it is not destroyed.

In dealing with files, it is always a good practice to verify if the input file has been opened successfully before performing read operations. This is because reading from a file that has not been opened successfully will results in run time error, causing the program to terminate abnormally.

The following is an example of statements that handle the file not found error:

**if(infile == NULL) {**

**printf("file not found");**

**exit(1); // exit the program with error code 1**

**}**

The test can be done when attempting to open the file:

**if((infile = fopen("data.txt", "r")) == NULL){**

**printf("file not found");**

**exit(1); // exit the program with error code 1**

**}**

**Specifying a file-path in fopen**

If a data file is not in the same folder as the program accessing it, the full path of the file must be used in the **fopen** statement. There are two ways to do this:

* separate folder names by a forward slash: /
* separate folder names by two back slashes: \\

**Examples:**

**FILE \*infile1, \*infile2, \*outfile;**

**infile1 = fopen("D:/ics103/inputData.txt", "r");**

**infile2 = fopen("D:\\ics103\\data2.txt", "r");**

**outfile = fopen("E:\\MyFiles\\output.txt", "w");**

// . . .

3- Read/write from/to the files using file input/output functions.

|  |  |
| --- | --- |
| function | Description |
| fscanf(filePointerVariable, formatString, AddressList);  **Example:**  **fscanf(infile, "%lf%lf%lf", &x, &y, &z);** | Reads values from the file into variables with corresponding addresses in AddressList, using the formats in formatString. The **int** symbolic constant **EOF** is returned when end-of-file marker is detected. |

**Note:** When using fscanf to read data from a file, the programmer must know how the data is arranged in the file.

**Reading to the end of file**

The **EOF** constant that is returned by **fscanf** when the end-of-file marker is detected can be used as a sentinel in conditional loops:

**Example1: File containing ID and grade in each line with unknown number of lines.**

To read to the end of file, we read one line at a time. The program reads two values for each call to **fscanf**. Once we reach the end of file, the EOF value returned by **fscanf** stops the while loop because the condition EOF != EOF becomes false.

The reading part of the while loop can be written as:

**int status = fscanf(infile, "%d%lf", &id, &grade);**

**while(status!= EOF){**

**// . . .**

**status = fscanf(infile, "%d%lf", &id, &grade);**

**}**

Note: status is an **int** variable.

The previous loop can be made shorter by using **fscanf** as part of the while loop condition:

**while(fscanf(infile, "%d%lf", &id, &grade)!= EOF){**

**// . . .**

**}**

**Example2: Reading one character at a time to the end of file**

**while(fscanf(infile, "%c",&ch) != EOF){**

**// . . .**

**}**

Or by using **status** variable :

**status = fscanf(infile, "%c",&ch);**

**while(status != EOF){**

**// . . .**

**status = fscanf(infile, "%c",&ch);**

**}**

**Some file output functions:**

|  |  |
| --- | --- |
| function | Description |
| fprintf(filePointerVariable, formatString, expressionList);  **Example:**  **fprintf(outfile, "Average = %.2f\n", sum / count);** | Write the values of the expressions in expressionList to the file, using the formats in the format string. |
| fprintf(filePointerVariable, string);  **Example:**  **fprintf(outfile, "Welcome to ICS 103\n");**  Alternatively, the format specifier "%s" can be used:  **fprintf(outfile, "%s", "Welcome to ICS 103\n");** | Write the string to the file. |

4- Close the files after processing the data using the **fclose** function.

The function **fclose** is used to break the link established by the **fopen** between the filePointerVariable

and the external file. The syntax of fclose is:

**fclose(filePointerVariable);**

After this function call, the filePointerVariable can be used for another file.

**Examples:**

**fclose(infile);**

**fclose(outfile);**

When you finish using a file you must always close it to release system resources. If you do not close an output file a file, then some of the data might not be written to it.

**Example 1**: The program below reads *miles* from **data.txt** and writes the corresponding *kilometers* to **result.txt**

**C program:**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #define KMS\_PER\_MILE 1.609  int main(void) {  double kms, miles;  FILE \*infile, \*outfile;  infile = fopen("data.txt","r");  if(infile == NULL){  printf("Error: Failed to open data.txt\n");  exit(1);  }  outfile = fopen("result.txt","w");  fscanf(infile, "%lf", &miles);  kms = KMS\_PER\_MILE \* miles;  fprintf(outfile, "%.2f miles equals %.2f kilometers.\n", miles, kms);  fclose(infile);  fclose(outfile);  return 0;  } |

**Example 2:** In the program below the EOF (End Of File) marker is used as a sentinel. The program reads its own text, character by character, and displays it on the screen.

Copy this program and save it as **example2.c**, then run it and you will see the whole program displayed on the screen.

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  int main (void){  FILE \*in;  in = fopen("example2.c","r");  if(in **==** NULL){  printf("Error: Failed to open example2.c\n");  exit(1);  }  char ch;  while(fscanf(in,"%c",&ch) != EOF)  printf("%c",ch);    fclose(in);  return 0;  } |

**Example 3:** The program below calculates the sum and average score of a class in a quiz; it then displays them on the screen. The quiz scores are read from an input file **scores.txt**.

#include <stdio.h>

#include <stdlib.h>

int main(void) {

FILE \*infile;

double score, sum, average;

int count, status;

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

if(infile **==** NULL){

printf("Error: Failed to open scores.txt\n");

exit(1);

}

sum = 0;

count = 0;

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

while(status!= EOF){

sum += score;

count++;

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

}

if(count **==** 0)

printf("Error: No grades read");

else{

average = sum / count;

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

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

}

fclose(infile);

return 0;

}

The contents of the input file **scores.txt** are:

|  |
| --- |
| 10.0  6.8 9.5  9.7 7.7  3.6 5.7 8.1  7.3 6.8 |

**Example 4:** The program below appends the contents of **file2.txt** to the end of **file1.txt:**

**#include <stdio.h>**

**#include <stdlib.h>**

**int main(void) {**

**FILE \*file1ptr, \*file2ptr;**

**char ch;**

**file2ptr = fopen("file2.txt", "r");**

**if(file2ptr == NULL){**

**printf("Error: Failed to open file2.txt\n");**

**exit(1);**

**}**

**file1ptr = fopen("file1.txt", "a");**

**while(fscanf(file2ptr, "%c", &ch)!= EOF)**

**fprintf(file1ptr, "%c", ch);**

**fclose(file1ptr);**

**fclose(file2ptr);**

**return 0;**

**}**

|  |  |
| --- | --- |
| file1.txt | file2.txt |
| 10.0  6.8 9.5  9.7 7.7  3.6 5.7 8.1  7.3 6.8 | 50.5 20.0  75.5  40.7 60.0 |

Assuming that file1.txt has at least one black character after its last value, the contents of file1.txt, after appending are:

10.0

6.8 9.5

9.7 7.7

3.6 5.7 8.1

7.3 6.8 50.5 20.0

75.5

40.7 60.0

Laboratory Tasks

Note: You must use EOF controlled loops in all tasks.

**Task 1:**

Each line of a text-file **grades.txt** contains the ID of a student and his grades in two exams. Write a C program that reads **grades.txt** and writes to an output file **output.txt** the ID and the worst exam grade and the average grade for each student. The worst exam grade and the average of each student must be computed in a **single** function that receives two exam grades and returns two values. This function must not contain fprintf or fscanf calls.

**Note:** Your program must be general, it must work for any number of lines in the input file.

Sample input file **grades.txt**:

|  |
| --- |
| 201100010 95.0 80.0  201100020 45.0 50.0  201100030 97.0 90.0  201100040 60.0 80.0  201100050 62.0 60.0 |

Sample output file **output.txt**:

|  |
| --- |
| ID WORST EXAM SCORE EXAM AVERAGE  201100010 80.0 87.5  201100020 45.0 47.5  201100030 90.0 93.5  201100040 60.0 70.0  201100050 60.0 61.0 |

**Task 2:**

The grade distribution of a certain course is according to the following table:

|  |  |
| --- | --- |
| **Evaluation Method** | **Weight** |
| Lab | 20% |
| Coursework | 15% |
| Midterm Exam | 30% |
| Final Exam | 35% |

Each line in a text-file **scores.txt** file contains five values: ID#, lab grade out of 100, coursework grade out of 100, midterm exam grade out of 100, and the final exam grade out of 100. If the letter grade a student gets is according to:

|  |  |
| --- | --- |
| Grade range | Letter Grade |
| [85 , 100] | A |
| [70, 85) | B |
| [60, 70) | C |
| [40, 60) | D |
| [0, 40) | F |

write a C program that reads **scores.txt** and writes to each line of a text-file **results.txt** the ID of a student, his total score out of 100 and his letter grade. Finding the total score must be implemented in a function that receives lab grade out of 100, coursework grade out of 100, midterm exam grade out of 100, and the final exam grade out of 100. Finding the letter grade must be implemented by writing a function that receives the total score as a value of type **double** and returns the letter grade as **char**. These two functions must not contain fprintf or fscanf calls.

**Note:** Your program must be general, it must work for any number of lines in the input file.

Sample input file **scores.txt**:

|  |
| --- |
| 201100010 95.0 80.0 75.0 80.0  201100020 45.0 50.0 30.0 50.0  201100030 97.0 90.0 90.0 95.0  201100040 60.0 80.0 75.0 75.0  201100050 62.0 60.0 70.0 60.0 |

Sample output file **results.txt**:

|  |
| --- |
| ID SCORE GRADE  201100010 81.5 B  201100020 43.0 D  201100030 93.1 A  201100040 72.8 B  201100050 63.4 C |

**Task 3:**

Create a text file **data.txt** using the data shown below and then write a C program that reads the file **data.txt** character by character. It then displays the number of lines, number of digits, lowercase letters, uppercase letters, and other characters in an output file **summary.txt** as shown below.

Note: Character digits are represented internally as an interval with increasing characters from ‘0’ to ‘9’. Thus, any character digit belongs to the interval [‘0’,’9’]. The same applies for letters i.e. a lowercase letter belongs to the interval [‘a’,’z’], and an uppercase letter belongs to the interval [‘A’,’Z’].

**Hint:** Each line in a text-file, except the last line, is terminated by the new line character **'\n'**.

|  |  |
| --- | --- |
| Input file: **data.txt** | Output file: **summary.txt** |
| kadFath%^&5453  as\*(){}765  129(\*&aBgKM+  234 | Number of lines=4  Number of lowercase letters = 10  Number of uppercase letters = 4  Number of digits = 13  Number of other characters = 12 |

**Note:**

* The new line character **'\n'** should not be counted in other characters.
* When you create the input file make sure it does not have an extra blank line at the end.