Reading & Processing Files in C

→ As scientists, one of the most important things we do with computers is to read, write, and process data files!

→ I don't love how Zybooks handles this topic (Chapter 3).

Example: We have a data file, which contains lines of data. Each line is of the following format:

LastName [tab] FirstName [tab] XX [tab] YY [tab] ZZ

where XX, YY, ZZ are test scores. The fields are separated by tab characters. There is a linefeed character (\n) at the end of each line.

C is a mid-level language! We need to know, typically, the exact format of the file, in order to work with it!!

StudentInfo.tsv

In general, most data files will have lines of information, with fields separated by some kind of character →
tabs, commas, colons, spaces, ...
↑ bad! Hard to work with!!!

Theory:

1. Read one line at a time
2. Process/Parse the line
3. Store information in arrays.

Assumptions:

(i) max. line length = 80 characters
(ii) all lines are well formed
(iii) we know how many lines there are, maximum.

We are now going to introduce a LOT

of new concepts, all of which are necessary to solve the problem of reading this data file (and processing it).

Up to now, we have seen two functions (in stdio.h) for input and output: printf, scanf.

These functions, by definition, print to the "screen", and read from the "keyboard". In most languages, these are known as "standard output", and "standard input".

In C: stdout, stdin

We will introduce two new functions,

fprintf( )

fscanf( )

These functions can take input from, or write output to, lots of places!

(stdout, stdin, files, sockets, peripheral devices, ethernet, USB, ...)

Ex:

```
printf("Hello World!");
fprintf(stdout, "Hello World!");
```

extra argument! — the rest is the same.

```
scanf("%d", &x);
fscanf(stdin, "%d", &x);
```

extra argument! — the rest is the same

Big Question! How to I read/write to a file???

There is a new variable type, defined in stdlib.h, called

FILE*

We will learn a lot more about this but for now, just remember DOINTER TO

later, ... that this represents a ... A FILE. (It's actually an address of a place on the disk where the file starts, btw).

Ex:

```
int i = 0;
double x = 0.0;
FILE* inFile = NULL;
```

a NULL pointer is the equivalent of initializing to "zero"!!

RULE: Always initialize every variable, lest ye face the wrath of hell of the gods of computing!!!

So, suppose we have a file, on the disk, called StudentInfo.tsv. How do we figure out the value of the FILE* pointer for this particular file???

Answer: There is a function in stdlib.h called

fopen( )

that returns a FILE* pointer!!

Ex.

```
FILE* inFile = NULL;
inFile = fopen("StudentInfo.csv", "r");
```

"r" = open for READING!

Now, we can read from this file, using fscanf!

```
fscanf(inFile, ..... );
```

Next Question: how do I ... the user to specify the filename?

① `char filename[80];`
`printf("Enter the file name: \n");`

② `scanf("%[\n]%*c", filename);`

① Create a "string", which is an array of 80 chars. We don't now how long the file name will be!

StudentInfo.tsv \n
0 1 ... 14 15

② `%[\n]%*c` ????

`%[\n]` — keep reading until we reach a "\n" newline character

`%*c` — read one char at a time, discard the \n !!

So, now we have opened a file for reading! Phew!! That was a lot.

```
FILE* inFile = NULL;
char filename[80];
printf("Enter the filename: \n");
scanf("%[\n]%*c", filename);
inFile = fopen(filename, "r");
```

OK, so now we have to read in a line of data, and parse it.

① `char line[80];` — actually reads one field at a time!!

② `while(EOF != fscanf(inFile, "%s", line)){`

```
    ...
    process line
    ...
```

}

① Create a char array (i.e. a string) to hold the entire line.

② Loop through the lines (or fields!) in the file until the End-Of-File (EOF) is reached.

(Note: fscanf( ) returns an EOF error code if unsuccessful, otherwise it returns nothing)

Final Step: Processing the Line!!
(one field at a time!)

① create some variables to hold the line number (j), and the column number (i). We need these to know the array index for our data arrays, and which data array to use.

```
int i = 0;  int j = 0
```

We will need two new functions from string.h →

① strncpy( )

② atoi( )

① → strncpy( ) copies a string input into a char[ ] array.

② → atoi( ) copies a string input into an integer.

```
const unsigned int nMax = 1000;
Char lastName[nMax][80],
     firstName[nMax][80];
int  score1[nMax], score2[nMax],
     score3[nMax];
     letterGrade[nMax];
```

```
char
double scoreAvg[nMax];
double test1Avg, test2Avg, test3Avg;
int i=0, j=0;
while ( ..... ) {
    if (i==0) {
        strncpy (lastName[j], line, 80);
    }
    if (i==1) {
        strncpy (firstName[j], line, 80);
    }
    if (i=2) {
        score[j] = atoi (line);
    }
    .
    .
    .
    i = i+1;
    if (i%5 == 0) {
        i = 0;
        j = j+1;
    }
}
```

reset i, increment j, every 5 fields!

All that is left to be done, now, is to process the arrays, and output the results!!

For this, we want to loop through the arrays that have been filled.

define type!! starting value ending condition increment

```
for (int idx=0; idx < j; i++) {
    .
    .
    .
    do stuff
    .
    .
    .
}
```

Steps:

① keep track of the sum of each test score, so that we can calculate a

class average.

② Calculate the average score for each student.

③ Calculate the final grade.

Step ③ → We need to use an if...else construct. Same as Python, but with different syntax:

```
if ( scoreAvg[idx] >= 90.0 ) {
    letterGrade[idx] = 'A';
} else {
    if (scoreAvg[idx] >= 80.0) {
        letterGrade[idx] = 'B';
    } else {
        :
    }
}
```

In general:

```
if ( condition ) {
    → do something
} else {
    → do something else
}
```

Final Step:

```
fclose (inFile);
fclose (outFile);
```

N.B. always close your open files!