Reading from File

May 29, 2020

1 Algorithm

This code opens a file and reads data

2 What to focus on

1. how COBOL handles file errors
2. Conditional variables 88
3. notice how the different languages deal with opening and closing of files

3 Notes

This code follows from lessons 003 and 004. In 003 we analyze the data structures used in the code. In 004 we introduce some basic file handling. You should check those lessons before moving on. In particular, this code reads the files written in lesson 004
In this section we introduce how to handle file errors. To do so, we employ conditional variables, which are listed in the code starting with 88 in the DATA DIVISION. The first thing to notice is that data listed as 88 do not really store data. For instance, in Employee there is not a field called "EndOfFile" to be read from the file. Instead, data 88 have two functions:

1. They retain information about the status of the data. For instance, the code only set FileStatus, but not its 88 data. Once FileStatus is

set, the code goes through each 88 clause and set those values. For instance, the line:

```
88 FileNotFound      VALUE '35'.
```

stands for:

```
if FileStatus==35 then FileNotFound == TRUE
```

This is why data 88 are called **conditional** variables. They only store a true/false value if a condition is met.

2. In the previous usage of data 88, we set a value of its parent data first and then their values were automatically set. What happens if we set their value directly? This is what happen for data 88 in Employee. In this case, the code:

   ```
   88 EndOfFile  VALUE ALL '*'.
   ```

   stands for:

   ```
   if EndOfFile is set to True
   then all the values of Employee are set to "*"
   ```

As we see from these examples, data 88 is always part of a more complex data type and it has no meaning on its own. When using them, pay attention on how their value is set. Depending on whether you set them directly or indirectly it changes their behavior.

4 Translation

Handling code errors is naturally a main focus when programming. For this reason, every programming language has developed its own tools/methods/philosophy on how to deal with those.

That implies that error handling may differ a lot among programming languages. Among the languages, Java requires to catch exceptions otherwise it does not compile. In Cobol and Kotlin you may skip that.
In Cobol, after you open the file, you assign the file status to a variable. Then, you check if any of any errors occured using the data 88 approach.
In Kotlin and Java, you manage exceptions by means of the try...catch...finally structure. You can have multiple catch blocks for different errors. Note that and "end of file" error is not present by default. You check the end of file when your reader returns a specific value. The finally block is **always** executed before closing the program, whether an exception took place or not. This is why the finally block is where you close the file(s).
As a side note, Kotlin requires a special permission to set a variable to null. To do so, the symbol ? is used, otherwise the compiler refuses all null values. This behavior stems from Java programming. In fact, in Java null pointers are a common issue. Kotlin tries solving this issue by enforcing a strict control on the usage of null.