CSCI3180 Assignment 1 Report

Nikunj Taneja (SID: 1155123371)

5 Feb 2022

I. COBOL v/s C for Implementing a Simulated Banking System (SBS):

For this question, I have divided the implementation of SBS into 3 main parts:

a. Handling user input, which basically encompasses the user interaction part of the system, so mainly atms.c and atms.cob
b. Reading and writing to files (or simply, file I/O), which mainly deals with reading and writing records to the respective files
c. Business logic, which simply includes everything else — from using simple arithmetic to calculate the updated balance, to sorting the records using a set of keys.

For part a., I think COBOL and C are mostly the same since the input validation wasn't too hard to implement in COBOL because the learning curve isn't too steep (strictly speaking for syntax here). The code in C does look cleaner, though (see Figure 1a and 1b).

```
426             do {
427                 atm = prompt_user(ATM_PROMPT);
428             } while (validate_input(atm, ATM_PROMPT) != SUCCESS);
429
430             do {
431                 acc = prompt_user(ACCOUNT_PROMPT);
432                 pwd = prompt_user(PASSWORD_PROMPT);
433                 auth_err = authenticate_user(acc, pwd);
434             } while (auth_err == INCORRECT_ACC_PWD);
435
436             if (auth_err == NEG_BALANCE)
437                 continue;
438
439             do {
440                 service = prompt_user(SERVICE_PROMPT);
441             } while (validate_input(service, SERVICE_PROMPT) != SUCCESS);
442
443             // handle the requested service
444             service_code = get_service_code(service);
445             timestamp = handle_service(service_code, atm, acc, timestamp);
446
447             // ask if user wants to continue
448             do {
449                 cont = prompt_user(CONTINUE_PROMPT);
450             } while (validate_input(cont, CONTINUE_PROMPT) == INVALID_INPUT);
451
```

Figure 1a. User Interaction Code in C

```
53          MAIN-PARAGRAPH.
54              OPEN OUTPUT TRANS711-FILE.
55              OPEN OUTPUT TRANS713-FILE.
56              DISPLAY "##########################################".
57              DISPLAY "##         Gringotts Wizarding Bank         ##".
58              DISPLAY "##                 Welcome                  ##".
59              DISPLAY "##########################################".
60              GO TO ATM-PROMPT.
61
62          ATM-PROMPT.
63              DISPLAY "=> PLEASE CHOOSE THE ATM".
64              DISPLAY "=> PRESS 1 FOR ATM 711".
65              DISPLAY "=> PRESS 2 FOR ATM 713".
66
67              ACCEPT ATM-INPUT FROM SYSIN
68
69              IF ATM-INPUT NOT = 1 AND ATM-INPUT NOT = 2 THEN
70              DISPLAY "=> INVALID INPUT"
71              GO TO ATM-PROMPT
72              END-IF.
73
74              GO TO ACC-PWD-PROMPT.
75
76          ACC-PWD-PROMPT.
77              DISPLAY "=> ACCOUNT"
78              ACCEPT ACC-INPUT FROM SYSIN
79              DISPLAY "=> PASSWORD"
80              ACCEPT PWD-INPUT FROM SYSIN
81              OPEN INPUT MASTER-FILE.
82              GO TO USER-AUTH.
83
84          USER-AUTH.
85              READ MASTER-FILE AT END
86              DISPLAY "=> INCORRECT ACCOUNT/PASSWORD"
87              CLOSE MASTER-FILE
88              GO TO ACC-PWD-PROMPT.
89
90              IF ACC = ACC-INPUT AND PWD = PWD-INPUT THEN
91              CLOSE MASTER-FILE
```

Figure 1b. User Interaction Code in COBOL

By reading just 10-15 lines of code in C, one can understand the full user flow in C. To do the same in COBOL, one would have to essentially read the entire program (fun fact: Figure 1b isn't even the entire user interaction code in COBOL). If the system that needs to be implemented is much bigger than SBS, then C would be much simpler since it's much easier to abstract away complex logic using "functions" or "procedure" calls. Of course, this helps the person reading the code much more than the programmer themselves.

For part b., COBOL beats C hands down. This is because for a system like SBS, COBOL provides for a much simpler API (see Figure 2a) to read and write from files, while C allows for much more flexibility. For instance, if there was a JSON-like format in which records were stored in *trans711.txt* and *trans713.txt*, building a parser using C would be much simpler than in COBOL. But since the SBS records are very standard and easily parsable, COBOL's out-of-the-box simple API beats C's apparent flexibility (e.g. it handles leading 0's in balance and whitespace in names).

```
121         DEPOSIT-HANDLER.
122             DISPLAY "=> AMOUNT"
123             ACCEPT AMT-INPUT FROM SYSIN
124             COMPUTE AMT-INTEGER = AMT-INPUT * 100.00.
125             IF AMT-INPUT < 0 THEN
126             DISPLAY "INCORRECT AMOUNT"
127             GO TO DEPOSIT-HANDLER
128             END-IF.
129
130             IF ATM-INPUT = 1 THEN
131             SET ACC-711 TO ACC-INPUT
132             MOVE "D" TO OP-711
133             SET AMT-711 TO AMT-INTEGER
134             SET TS-711 TO TXN-TIMESTAMP
135             WRITE TRANS711-RECORD
136             COMPUTE TXN-TIMESTAMP = TXN-TIMESTAMP + 1
137             END-IF.
138
139             IF ATM-INPUT = 2 THEN
140             SET ACC-713 TO ACC-INPUT
141             MOVE "D" TO OP-713
142             SET AMT-713 TO AMT-INTEGER
143             SET TS-713 TO TXN-TIMESTAMP
144             WRITE TRANS713-RECORD
145             COMPUTE TXN-TIMESTAMP = TXN-TIMESTAMP + 1
146             END-IF.
147
148             GO TO CONTINUE-PROMPT.
```

Figure 2. File I/O API in COBOL

The only con in COBOL is the code repetition. What if there were 20 ATMs, each with its own transaction file? Thus, COBOL doesn't provide much scalability or extensibility (at least not with the code I wrote).

For part c., I think C is marginally better than COBOL since I didn't have to think too much about the control flow of the program due to the support for conditional statements (if-then-else block and switch-case), looping (for-loops and while-loops) and

function/procedure calls). In COBOL, I had to be quite careful since a lot of times, there was no else block but only if-then. The use of *GO TO* in COBOL created a lot of spaghetti code which was quite hard to debug especially in central.cob since my master updating logic runs in *O(n + m)* time (instead of *O(n * m)* time), where *n* is the number of records in *master.txt* and *m* is the number of records in *transSorted.txt*, which was fairly more complex to implement. Having said that, it's hard to ignore how easy it is to sort the transactions in COBOL. I'm sure that if we had been allowed to use MERGE, I would really seriously consider COBOL as my language of choice for writing applications like SBS. What essentially puts me on the fence is the lack of support for function/procedure calls and the complex debugging that comes with the spaghetti code.

In conclusion, if I didn't know either C or COBOL and I had to pick one language to implement the SBS all over again, I would probably pick C six times out of ten (given the constraints that we had for this assignment). This is because the C code is easier to debug, maintain and scale although the development time in C would probably be more due to the lack of a simple file I/O API and out-of-the-box sorting. And of course, also because of my personal distaste for *GO TO* and spaghetti code.

II. Has COBOL stood the test of time? COBOL v/s Python:

Well, the short answer to the above question is clearly not. However, I don't think the short answer would fetch me many marks so the following is a comparison of COBOL with my favorite language, Python. Python puts most of my qualms regarding C, in the answer to the previous question, to rest by providing an easy-to-learn simple and rich syntax along with a ton of features. With the support for OOP and simple file I/O, if I had to implement SBS all over again in C, COBOL or Python (and again, to make it fair, I didn't know any of the three languages), I would definitely pick Python ten times out of ten. Why? Well, it is very easy to minimize code repetition in Python and write robust, scalable and most importantly, correct code in just the first go. The stack trace and helpful error messages during runtime errors provide for easy debugging as well. That is very hard to compete against if your weapon is COBOL. Of course, Python code wouldn't be as efficient but then it's always possible to optimize it using various open source libraries. Admittedly, that would also increase the size of the application, but let's ignore that since disk memory isn't really a big issue in the 21st century and we're strictly comparing the two languages. Python also provides support for essentially infinitely-sized numbers that adds to its scalability factor while COBOL's integers are limited to a maximum of 31 digits, as per my knowledge from the IBM documentation. Well, if Python is so rich, then why hasn't the financial world switched to Python for their transaction systems instead of

running COBOL mainframes? One simple rule: don't fix what's not broken. :)

III. Is COBOL suitable for writing applications like SBS?

This isn't a hard question. Of course, yes. Why? All the reasons I mentioned in answer to the first question. Easy file I/O and out-of-the-box support for sorting (that too on multiple keys!) combined with the not-so-steep learning curve, even though the language is decades old, makes it a suitable language for such applications. Arguably, my development time in COBOL and C was probably the same, just that learning COBOL on the fly made me prone to a lot of bugs and debugging was just the worst. But if I were equally proficient in COBOl and C (and the constraints were removed so that I didn't have to use GO TO), I think I would take less time to write the SBS in COBOL than in C. For these reasons, I believe COBOL is a suitable language for writing applications like SBS.