1. The conveniences and difficulties in implementation

Conveniences in COBOL over C:

a) Auto-formatting

In this assignment, both input and output file's balance/amount need to autofill 0's at the front of number (example, 54.00 become 000000000005400), C program will cancel those leading 0's automatically, which means that we need a for-loop to output those leading 0's, but COBOL will automatically output it in write process. COBOL takes a small convenience at that point.

b) Large number calculation

We may need to handle large number when doing transaction. In COBOL programming, we can often . However, int type in C programming cannot handle signed 15-digit number, we need to declare long long int to handle it. However, if the value of the variable is larger than 10^{15}, COBOL will much more convenient as in C, one long long int variable cannot handle it anymore, but 9(16) in COBOL can. Note that double formal in C may cause some small errors in accuracy.

c) Processing file data

In COBOL, after we declare the file variable, different data will be stored in each category (name, account number, balance...) automatically. But C need to require one more function to split the string and put it into different category, COBOL takes a small convenience at that point.

```c
Account readline(char mystring[], Account my)
{
    strncpy(my.name, &mystring[0], 20);
    my.name[20] = '\0';
    strncpy(my.number, &mystring[20], 16);
    my.number[16] = '\0';
    strncpy(my.password, &mystring[36], 6);
    my.password[6] = '\0';
    strncpy(my.balance, &mystring[42], 16);
    my.balance[16] = '\0';
    return my;
}
```

C program

```
COPY713-PARAGRAPH.
    READ OUTPUTTWO INTO MYOPERATION
       AT END GO TO FINALSORT-PARAGRAPH
       NOT AT END GO TO COPY713BYLINE-PARAGRAPH
    END-READ.
```

COBOL

Conveniences in C over COBOL:

a) Paragraph/Function

Because only GO TO function is allowed for COBOL, which means that many paragraphs are declared when we want to simulate the while-loop. However, sometimes different paragraphs may be used for same or similar purpose. Which makes it difficult to read. In C, we can declare one function to one purpose, which increases readability.

```
        CLOSE INPUTMERGED.
        GO TO COPY711-PARAGRAPH.
MYMERGE713-PARAGRAPH.
        CLOSE OUTPUTONE.
        OPEN INPUT OUTPUTTWO.
        GO TO COPY713-PARAGRAPH.
COPY713-PARAGRAPH.
        READ OUTPUTTWO INTO MYOPERATION
           AT END GO TO FINALSORT-PARAGRAPH
           NOT AT END GO TO COPY713BYLINE-PARAGRAPH
        END-READ.
COPY713BYLINE-PARAGRAPH.
        OPEN EXTEND INPUTMERGED.
          MOVE MNUMBE TO IMNUMBE.
          MOVE MACTION TO IMACTION.
          MOVE MAMOUNT TO IMAMOUNT.
          MOVE MTIME TO IMTIME.
          WRITE MYINPUTMERGED
          END-WRITE.
        CLOSE INPUTMERGED.
        GO TO COPY713-PARAGRAPH.
```

COBOL. So many similar paragraphs!

b) File pointer/variable declaration

Each file variable is linked to one *.txt* file. Therefore when processing the *central.c* file, we need to declare 8 file variable with similar structure. Therefore, many variable with similar name will be created, we may encountered some situation that we cannot handle such many variable. A significant example is . In C, file pointer is needed when start opening file, and that pointer can also be used for more than one *.txt* file, we can also define structure to remember a record (Transaction, Master), which means that we can only define the number of file/variable we need to use at the same time, which much saving the memory, and make it easier to write.

```
void update_func()
{
    FILE *fpsort;
    FILE *fpmaster;
    FILE *fpmaster_updated;
    FILE *fpneg;
    char scanline[256];
    Transaction curr_transaction;
    Master curr_master;
    fpsort = fopen("transac_sorted.txt", "r");
    fpmaster = fopen("master.txt", "r");
    fpmaster_updated = fopen("master_updated.txt", "w");
    fpneg = fopen("negReport.txt", "w");
    fgets(scanline, 99, fpmaster);
    curr_master = master_readline(scanline);
```

C, file pointer + fopen/fclose is er

```
FILE SECTION.
FD INPUT-FILE.
01 ACCOUNT.
   02 NAME PIC A(20).
   02 NUMBE PIC 9(16).
   02 PASSWORD PIC 9(6).
   02 BALANCE PIC S9(13)V9(2) SIGN LEADING SEPARATE.
FD INPUTONE.
01 MYINPUTONE.
   02 IONUMBE PIC 9(16).
   02 IOACTION PIC A(1).
   02 IOAMOUNT PIC 9(5)V9(2).
   02 IOTIME PIC 9(5).
FD INPUTTWO.
01 MYINPUTTWO.
   02 ITNUMBE PIC 9(16).
   02 ITACTION PIC A(1).
   02 ITAMOUNT PIC 9(5)V9(2).
   02 ITTIME PIC 9(5).
FD OUTPUTONE.
01 MYOUTPUTONE.
   02 OONUMBE PIC 9(16).
   02 OOACTION PIC A(1).
   02 OOAMOUNT PIC 9(5)V9(2).
   02 OOTIME PIC 9(5).
FD OUTPUTTWO.
01 MYOUTPUTTWO.
   02 OTNUMBE PIC 9(16).
   02 OTACTION PIC A(1).
   02 OTAMOUNT PIC 9(5)V9(2).
   02 OTTIME PIC 9(5).
```

COBOL, HUGE AMOUNT OF SIMILAR VARIABLES

2. Comparison between Python and COBOL

a) Variable declaration: Python can declare it

b) Function/Procedure: Function declared in python will not be ran automatically , but paragraph in COBOL

c) Programming paradigms: Both are procedural, imperative, object-oriented, but COBOL is business oriented, Python is more widely used, more on statistic/machine learning

d) Data-types: Python have many different data types, like boolean, integer, list, array, string, tuple... etc we can base on it to declare some new data types easily like trees. But as COBOL is business oriented, so it can numerical data and string only.

3. Conclusion: Whether COBOL is suitable for writing this type of assignment.

Programming difficulty: Even global variable make it to be easier to be implemented. The major difficulty comes when similar variable have to declared many times for different file. The GO TO and paragraphs

also lowers the readability and writability, which makes it takes longer time for debugging.
Efficiency: It is quite complicated on variable and paragraph part, the efficiency lowered because of more memory are used. But for numerical data computation, no matter how large the data is, it is still efficient and reliable.