```
CSCI3180 Principles of Programming Languages

--- Declaration ---

I declare that the assignment here submitted is original except for source
material explicitly acknowledged. I also acknowledge that I am aware of
University policy and regulations on honesty in academic work, and of the
disciplinary guidelines and procedures applicable to breaches of such policy
and regulations, as contained in the website
    http://www.cuhk.edu.hk/policy/academichonesty/

Assignment 1
Name : YU Si Hong
Student ID : 1155141630
Email Addr : 1155141630@link.cuhk.edu.hk
```

Report of assignment 1

Comparation of conveniences and difficulties with C

Generally speaking, COBOL has a strict rule and a pitifully number of statements to implement a specific function, whereas C is more flexible. You may be thankful at first that you don't need to remember many statements to write a COBOL programme. However, when scale goes large, the COBOL programme will become horrible, for its GO TO statements makes it difficult to read, its file declaration makes it repetitive and redundant. Dividing the programme into blocks just like C does makes it plain and simple.

Read/Write file in certain format

In COBOL, each single file must has a declaration of its form and data record before opening it. Work will become tedious and repetitive when you want to open multiple files in one programme.

- Some declarations in atms.cob
    - Declare file form:

```
    FILE-CONTROL.
        SELECT MASTER ASSIGN TO 'master.txt'
            ORGANIZATION IS LINE SEQUENTIAL
            FILE STATUS MS.
```

    - Declare data record:

```
    FD MASTER.
    01 MRECORD.
        02 MNAME PIC A(20).
        02 MACC PIC 9(16).
        02 MPSWD PIC 9(6).
        02 MBALANCE PIC S9(13)V9(2) SIGN LEADING SEPARATE.
```

In C, file can be open without any preparation and there are many ways to accomplish read and write. It is more flexible but everything needs to be handled with more care.

- A whole picture of opening and closing file in central.c

```
FILE *f = fopen(fil, "r");
if (f == NULL)
{
    printf("=> ERROR IN OPENING %s FILE\n", fil);
    return 1;
}
fclose(f);
```

- 3 ways to get record from file in central.c
  - (a) fgets

```
while (fgets(bf, 32, fi1))
{
    fputs(bf, fi3);
}
```

  - (b) feof

```
// while (1)
// {
//     if (feof(fi1))
//     {
//         break;
//     }
//     fgets(bf, 31, fi1);
//     fputs(bf, fi3);
// }
```

  - (c) fscanf

```
while (1)
{
```

```
      if (EOF == fscanf(fi2, "%29s", bf))
          break;
      fgetc(fi2);
      fputs(bf, fi3);
      fprintf(fi3, "\n");
  }
```

Simulate loop

In COBOL, loop is simulated by using GO TO to jump to the beginning of tha loop when you are at the end of the loop and using GO TO to jump out of the loop if you want to leave. When there are many loops, it becomes really difficult to read.

- A loop to read trans711.txt in central.cob

```
    READ711.
        READ TRANS711
        NOT AT END
            MOVE T1ACC TO TP2ACC
            MOVE T1AMOUNT TO TP2AMOUNT
            MOVE T1OPERATION TO TP2OPERATION
            MOVE T1TIME TO TP2TIME
            WRITE TP2RECORD
            GO TO READ711
        AT END
            CLOSE TRANS711
            OPEN INPUT TRANS713
            GO TO READ713
        END-READ.
```

In C, loop can be simulated by while() or for(), which makes code more plain and clean.

- A very short loop to read trans711.txt in central.c

```
while (fgets(bf, 32, fi1))
{
    fputs(bf, fi3);
}
```

Procedure/Function call

In COBOL, the whole procedure is an indivisible entity, and the only thing allowed is using GO TO to jump to somewhere continuously.

- Implement of withdrawal in atms.cob

```
OPEW.
       DISPLAY "=> AMOUNT".
       ACCEPT AMOUNT.
       IF AMOUNT IS NEGATIVE THEN
           DISPLAY "=> INVALID INPUT"
           GO TO OPEW
           END-IF.
       IF AMOUNT > MBALANCE THEN
           DISPLAY "=> INSUFFICIENT BALANCE"
           GO TO OPEW
           END-IF.
       DISPLAY "=> WITHDRAW ", AMOUNT, " TO ", ACC1.
       IF ATM = 1 THEN
           MOVE ACC1 TO T1ACC
           MOVE 'W' TO T1OPERATION
           MOVE AMOUNT TO T1AMOUNT
           MOVE STAMP TO T1TIME
           WRITE T1RECORD
           END-IF.
       IF ATM = 2 THEN
           MOVE ACC1 TO T3ACC
           MOVE 'W' TO T3OPERATION
           MOVE AMOUNT TO T3AMOUNT
           MOVE STAMP TO T3TIME
           WRITE T3RECORD
           END-IF.
       ADD 1 TO STAMP.
       GO TO CONTI.
```

In C, the procedure can be divided into any number of blocks.

- Implement of withdrawal in atms.c
  - Block 1

```
if (opt == 'W')
{
    char *balance = mbalance;
    double amount = withdrawal(balance);
    writeAtm(acc1, &atm, &opt, &amount);
}
```

  - Block 2

```
double withdrawal(char *balance)
{
    double amount = 0;
    while (1)
    {
```

```
        ...
        return amount;
    }
}
```

- Block 3

```
void writeAtm(char *acc, char *atm, char *opt, double *amount)
{
    // (a) get timestamp string
    ...

    // (b) get amount string
    ...

    // (c) prepare record
    ...

    // (d) write to ATM
    ...

    stamp++;
    return;
}
```

Comparation with python

COBOL is an old language, while python is new, and there are many differences between them.

Variable declaration

In COBOL, all variable is global variable and must be written in a specific area so call data division; in python, you can define the scope of a variable as you want.

- Define variables in atms.cob

```
WORKING-STORAGE SECTION.
   01 ATM PIC 9.
   01 OPE PIC A.
   01 OPE2 PIC A.
   01 ACC1 PIC 9(16).
   01 ACC2 PIC 9(16).
   01 AMOUNT PIC S9(5)V9(2).
   01 PSWD PIC 9(6).
   01 STAMP PIC 9(5) VALUE 0.
   01 TBALANCE PIC S9(13)V9(2) SIGN LEADING SEPARATE.
   77 MS PIC X(02) VALUE SPACES.
```

Paradigm

COBOL is a typical language with only non-structured programming; python supports many programming paradigms except non-structured programming.

Data type

In COBOL, all data type should be defined by yourself; in python, data type will be automatically matched for you.

Parameter parsing

COBOL looks similar to English syntax; python is more abstract.

Suitable ?

- I don't think so.
  COBOL was likely made for this job at that time. However, in today's world it can still accomplish the job but definitely not the best choice. Because of the GO TO based paradigm, programming difficulty increases with scale. And the efficiency of programming is always at a low level.