CSCI3180 – Principles of Programming Languages – Spring 2022

Assignment 1 — Gringotts Wizarding Banking System in COBOL and C

Deadline: Feb 6, 2022 (Sunday) 23:59

1 Introduction

COBOL has existed for over sixty years, but remained widely and actively used across financial institutions. Some say 95 percent of Automatic Teller Machine (ATM) transactions in the world pass through COBOL programs. In this assignment, you are required to implement a simulated banking system for the Gringotts Wizarding Bank in COBOL and C.

There are **2 ATMs** in the Hogsmeade Village, numbered 711 and 713, using which wizards can access their Gringotts's account in order to make a variety of financial transactions, including deposit, withdrawal, and transfer to another wizard.

A master file is contained in the **central machine** of the Gringotts Wizarding Bank. The file contains the information of all the registered wizards of Gringotts, including each registered wizard's account number and password. Both the ATMs can access the master file *remotely*. Whenever a wizard comes and uses an ATM, the ATM will access the master file and check whether the account number and password are correct.

Transactions on the ATMs are not processed in real time in the wizarding world. Instead, the ATM transactions by the customers are accumulated and stored in a transaction file and processed only at day end. With the help of dementors, Gringotts can easily collect debts owed by wizards. Therefore, Gringotts allows wizards to have negative balance, but only wizards whose balance is non-negative in the master file can use the ATMs. At the end of a day, the central machine of Gringotts collects the two transaction files generated in these two ATMs, sorts them, merges them into one file while maintaining the ordering, updates the master file accordingly, and reports the accounts with negative balances to Gringotts.

2 Assignment Details

Only registered wizards can use the Gringotts ATMs. Gringotts has already collected information of all the registered wizards. Whenever a wizard uses an ATM, the ATM checks the user information and transaction validity, generates a data record of the transaction, and appends the record into a transaction file. At the end of a day, the central machine of Gringotts sorts the two transaction files that two ATMs produced separately, merges them into one file, updates the information of wizards, and reports the accounts with negative balances to Gringotts. You are required to implement a simulation of this banking system, including the ATMs and the central machine in both COBOL and C.

2.1 Banking Simulation System

The banking simulation can be implemented as two separate programs: the *ATMs* program which simulates the operations of the two ATMs and the *central machine* program which simulates how the transaction files collected for the ATMs are preprocessed and used to update the customer master file.

2.1.1 The ATMs Program

The ATMs program takes the master file ("master.txt") as input, interacts with bank user, and outputs two transaction files ("trans711.txt", and "trans713.txt"). The master file contains the information of all the registered wizards of Gringotts, each line for one wizard. Figure 1 shows an example of the "master.txt" file:

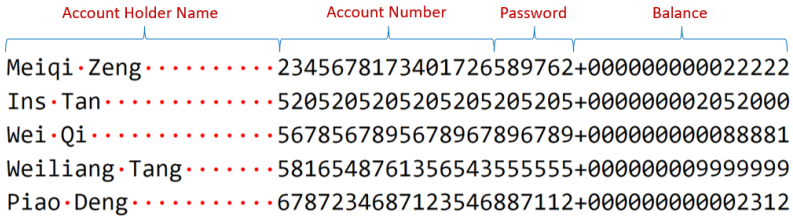
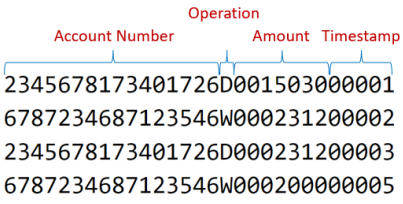


Figure 1: An example of the master file.

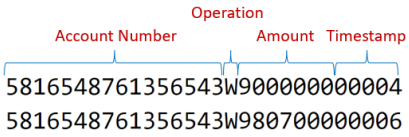
where

- *Account Holder Name*: the name of the bank account holder (20 characters).
- *Account Number*: the unique number of each account (16 digits); whenever a wizard comes to use the ATM, the ATM checks if the entered account number exists in "master.txt".
- *Password*: the password of the account (6 digits); whenever a wizard comes to use the ATM, the ATM checks if the entered password matches with that associated to the user entered account.
- *Balance*: the balance of the account (16 places), which stands for a decimal number with two decimal places. The first character represents whether the balance is positive or not. For example, the balance of the first account in Figure 1 is 222.22. Accounts with a negative balance should be barred from using the ATMs.

When a wizard uses an ATM, the ATM generates a record of the transaction and stores the record in a transaction file. Each ATM maintains its own transaction file. ATM711 maintains "trans711.txt" while ATM713 maintains "trans713.txt". Figure 2 shows examples of the "trans711.txt" and "trans713.txt" files:



(a) An example of the "trans711.txt" file.



(b) An example of the "trans713.txt" file.

Figure 2: Examples of the transaction files.

where

- *Account Number*: the account number of the wizard making the transaction (16 digits).
- *Operation*: the transaction (1 character): "D" (deposit) and "W" (withdrawal). Note that there are no "T" transactions. A transfer transaction will be written as two transaction records: first for the withdrawal from the sender, and second for the deposit to the receiver. For example, the second and the third lines in Figure 2(a) show the records of a transfer transaction: account 6787234687 transfers 23.12 to account 2345678173401726.

- *Amount*: the amount of money that the wizard wants to deposit/withdraw (7 places, which stands for a decimal number with two decimal places).
- *Timestamp*: the order number of each transaction, which is automatically generated by the program (5 digits). The timestamp starts from 00000. Whenever a new valid transaction takes place, the timestamp is increased by 1. We assume there are at most 100000 transactions altogether in the two ATMs in a day.

When a wizard comes and uses an ATM, the followings will happen:

1. The program asks the wizard to choose the ATM machine to use (as shown in Figure 3). If the wizard does not enter "1" or "2", the program should output "=> INVALID INPUT", until the wizard enters "1" or "2". Then the ATM asks the wizard to enter the account number and password. The ATM checks whether the account is registered and the password is correct by searching "master.txt". If that is the case, the ATM should output "=> INCORRECT ACCOUNT/PASSWORD", until the wizard enters a registered account and correct password. The ATM checks whether the balance of this account is non-negative. If the balance of the account is negative, the ATM should output "=> NEGATIVE REMAINS TRANSACTION ABORT", and this transaction is ended. Accounts with a negative balance are barred from using the ATMs.
2. The wizard enters "D" (deposit), "T" (transfer), or "W" (withdrawal) to select the transaction type (as shown in Figure 3). If the wizard enters other characters, the ATM should output "=> INVALID INPUT" and ask the wizard to enter again, until the wizard enters "D", "T", or "W".
   (a) If the wizard enters "D" (deposit), the ATM should output"=> AMOUNT" to ask the wizard to enter the amount of money to deposit. If the amount is smaller than 0, the ATM should output "=> INVALID INPUT" and ask the wizard to enter the amount again, until the wizard enters an amount which is non-negative. After that, the ATM will generate a transaction record, store it in its own transaction file, and end the transaction.
   (b) If the wizard enters "T" (transfer), the ATM should output "=> TARGET ACCOUNT" to ask the wizard to enter the receiver's account. The ATM will check whether the receiver's account is registered and whether the receiver's account is the sender's account. If that is the case, the ATM should output "=> TARGET ACCOUNT DOES NOT EXIST" or "=> YOU CANNOT TRANSFER TO YOURSELF" and ask the wizard to enter another account, until the entered account is registered and not the sender's own account. Then the ATM should output"=> AMOUNT" to ask the wizard to enter the amount of money to transfer, and check whether this amount is larger than the wizard's balance by searching "master.txt". If the amount is larger than the wizard's balance in "master.txt" or negative, the ATM should output "=> INSUFFICIENT BALANCE" or "=> INVALID INPUT" and ask the wizard to enter the amount again, until the wizard enters an amount smaller than or equal to the wizard's balance but non-negative. After that, the ATM will generate two transaction records: the first one for the withdrawal from the sender, and the second one for the deposit to the receiver. The transactions are then stored in the ATM's transaction file, and the transaction is ended.
   (c) If the wizard enters "W" (withdrawal), the ATM should output"=> AMOUNT" to ask the wizard to enter the amount of withdrawal money, and check whether this amount is larger than the wizard's balance by searching "master.txt". If the amount is larger than the wizard's balance in "master.txt" or negative, the ATM should output "=> INSUFFICIENT BALANCE" or "=> INVALID INPUT" respectively and ask the wizard to enter the amount again, until the wizard enters an amount smaller than or equal to his balance but non-negative. Notice that you can safely assume the input for amount is numerical with at most 2 decimal points. After that, the ATM will generate a transaction record, store it in its own transaction file, and end the transaction.

3. When a transaction ends, the ATM outputs "=> CONTINUE?" to ask the wizard whether to continue the transaction or not. If the wizard enters "Y", return to step 1. Otherwise, exit the ATMs program.

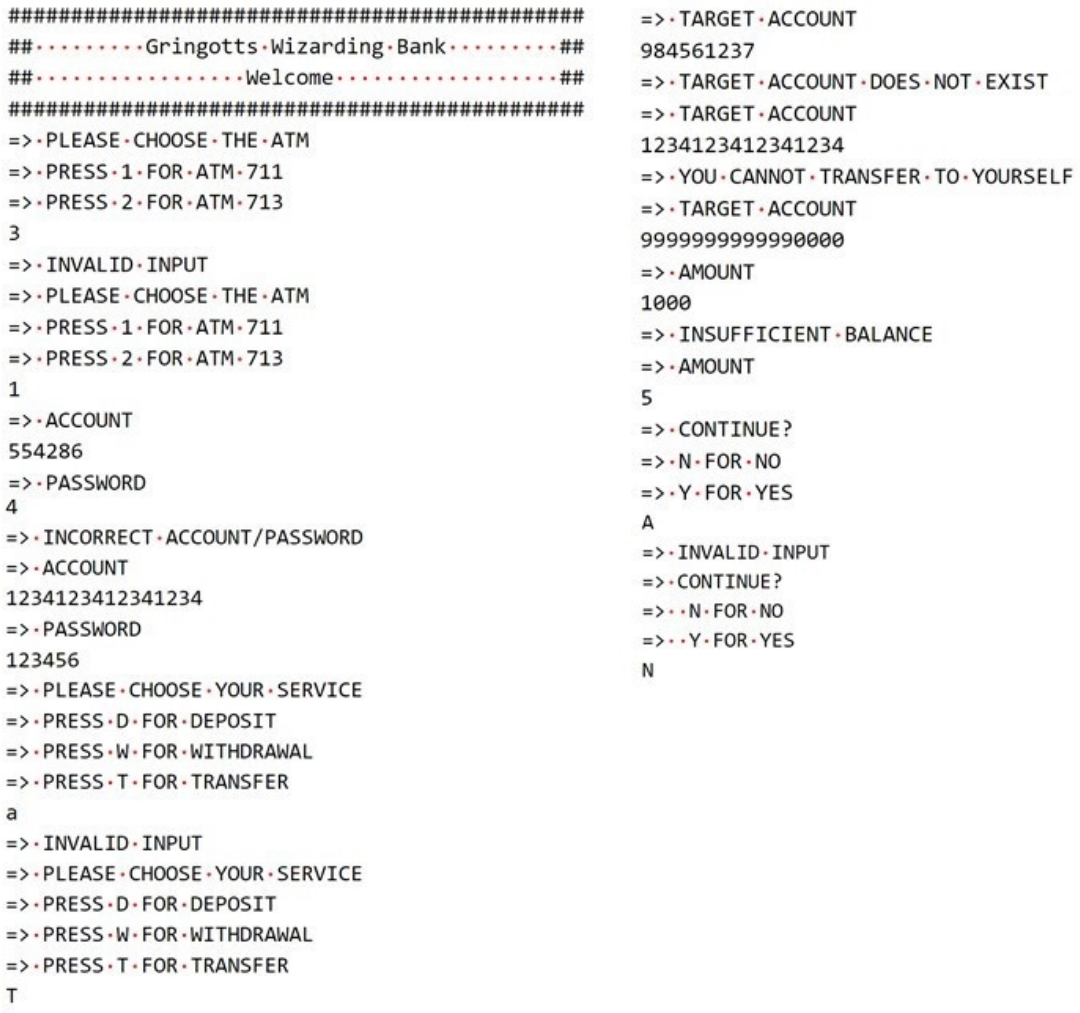


Figure 3: Screen shots of the ATMs program.

As we mentioned before, Gringotts allows wizards to have negative balance. At the end of each day, the central machine will sort out a list of accounts with negative balances and output a report (refer to Section 2.1.2).

2.1.2 The Central Machine Program

The central machine program takes 3 files as input: "master.txt", "trans711.txt" and "trans713.txt", and outputs 5 files: "transSorted711.txt", "transSorted713.txt", "transSorted.txt", "updatedMaster.txt", and "negReport.txt".

Figures 1 and 2 show examples of "master.txt" and "trans711.txt". For detailed description, please refer to Section 2.1.1. The format of "transSorted711.txt", "transSorted713.txt", and "transSorted.txt" are the same as the transaction file (refer to Section 2.1.1), and the format of "updatedMaster.txt" is the same as "master.txt" (refer to Section 2.1.1). Figures 4 and 5 show examples of the "transSorted.txt" file and the "updatedMaster.txt" file.

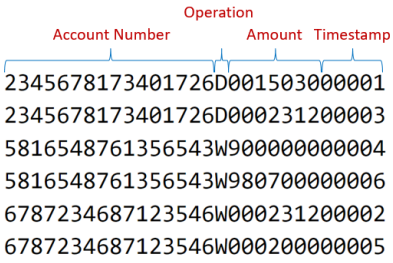


Figure 4: An example of the "transSorted.txt" file.

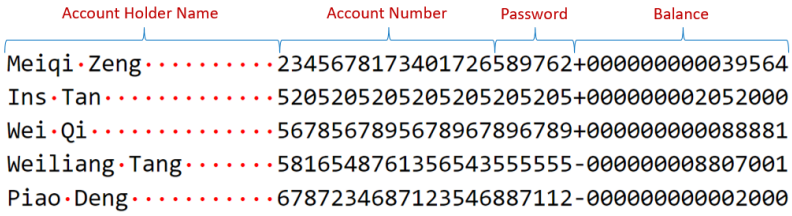


Figure 5: An example of the "updatedMaster.txt" file.

Figure 6 shows an example of the "negReport.txt" file:

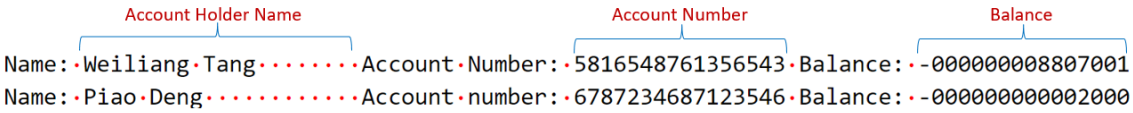


Figure 6: An example of the "negReport.txt" file.

where

- *Account Holder Name*: the name of the account user (20 characters).
- *Account Number*: the unique number of each account (16 digits).
- *Balance*: the balance of the account (16 places), which stands for a decimal number with two decimal places. The first place represents whether the balance is positive or not. All balances in "negReport.txt" are negative.

Note that in each line, there is "Name: " before the account holder name, "Account Number: " before the account number, and "Balance: " before the balance. In addition, the lines are sorted in increasing order of the account numbers.

At the end of a day, the central machine starts to work. The working flow of the central machine can be divided into three steps:

1. **Sort.** The central machine sorts the two transaction files ("trans711.txt" and "trans713.txt") by two keys from smallest to largest. The primary key is the account number, and the secondary key is the timestamp. The output of Step 1 is 2 files: "transSorted711.txt" and "transSorted713.txt".
2. **Merge.** After obtaining "trans711.txt" and "trans713.txt", the central machine merges them into one file ("transSorted.txt") by maintaining the sorted order again using the account numbers and the timestamps as the primary and secondary keys respectively. The order is from smallest to largest. The output of Step 2 is: "transSorted.txt".

3. **Update.** The central machine uses "transSorted.txt" to update information of the accounts in "master.txt" when necessary, and outputs a new file ("updatedMaster.txt"). If the balance of an account becomes negative, record it also in "negReport.txt". The output of Step 3 is 2 files: "updatedMaster.txt" and "negReport.txt".

Note: we will provide the "sort" function code in C for you, and you can use the "SORT" statement in COBOL.

2.2 General Specification

You are required to write two versions, one in COBOL and the other one in C, for the ATMs and the Central Machine program respectively. You should name your COBOL sources as **atms.cob** and **central.cob**, and your C sources as **atms.c** and **central.c**.

1. **Input and Output Specification**
   The ATMs program should read one input file:

   (1) **master.txt**, which contains the information of all the registered wizards of Gringotts.

   The detailed specification of input format is given in Section 2.1.1.

   The ATMs program should output two transaction files:

   (1) **trans711.txt**, which contains the transaction records of ATM 711.
   (2) **trans713.txt**, which contains the transaction records of ATM 713.

   The detailed specification of output format is given in Section 2.1.1.

   The central machine program should read three input files:

   (1) **master.txt**, which contains the information of all the registered wizards of Gringotts.
   (2) **trans711.txt**, which contains the transaction records of ATM 711.
   (3) **trans713.txt**, which contains the transaction records of ATM 713.

   The detailed specification of input format is given in Section 2.1.2.

   The central machine program should output five files:

   (1) **transSorted711.txt**, which contains the sorted transaction records of ATM 711.
   (2) **transSorted713.txt**, which contains the sorted transaction records of ATM 713.
   (3) **transSorted.txt**, which contains the merged transaction records of ATM 711 and ATM 713.
   (4) **updatedMaster.txt**, which contains the updated information of all the registered wizards of Gringotts.
   (5) **negReport.txt**, which contains the information of those whose deposit amount is negative.

   The detailed specification of output format is given in Section 2.1.2.

2. **Restrictions on using COBOL and C**

   (a) For COBOL, in order to force you to program as in the old days, ONLY 2 keywords are allowed in selection and loop statements: "IF" and "GOTO". You are not allowed to use modern control constructs, such as if-then-else or while loop. Using any other keywords will receive marks deduction. But for C, you can use whatever you want.

(b) For COBOL and C, you cannot use arrays or other data structure to store the records of "master.txt". Whenever you need to search the records of "master.txt", read the file directly.

(c) For COBOL, you cannot use "merge" statement directly. Please write a merge function to merge two files by yourself.

3. **Error Handling**
   The programs should also handle possible errors gracefully by printing meaningful error messages to the standard output. For example, your program should be able to check whether the input file exists or not. If not, display a warning message "non-existing file!". However, you **CAN** assume that the input files are free of format on content errors.

4. **Good Programming Style**
   A good programming style not only improves your grade but also helps you a lot in debugging. Poor programming style will receive marks deduction. Construct your program with good readability and modularity. Provide sufficient documentation by commenting your codes properly but never redundantly. Divide up your programs into subroutines instead of clogging the main program. The main section of your program should only handle the basic file manipulation such as file opening and closing, and subprogram calling. The main purpose of programming is not just to make the program right but also make it good.

5. **Other Notes**
   You are **NOT** allowed to implement your program in another language (e.g. Java/Python) and then initiate system calls or external library calls in COBOL and C. Your source codes will be compiled and PERUSED, and the object code tested!

   Do not implement your programs in multiple source files. Although COBOL and C do allow you to build a project with subroutines scattered among multiple source files, you should only submit one source file for each language.

   NO PLAGIARISM!!!! You are free to design your own algorithm and code your own implementation, but you should not "borrow" codes from your classmates. If you use an algorithm or code snippet that is publicly available or use codes from your classmates or friends, be sure to DECLARE it in the comments of your program. Failure to comply will be considered as plagiarism.

   A crash introduction to COBOL will be given in the upcoming tutorials. Please DO attend the tutorials to get a brief idea on COBOL, and then learn the language by yourselves. We assume that you are proficient in C. For a more in-depth study, we encourage students to search relevant resources on the Internet (just Google it!).

2.3 Input File Format Specification

As far as the outline banking system is concerned, there is only one input file, "master.txt", which is in plain ASCII text. The lines should be sorted from smallest to largest by the account number. Each line is ended with the characters "\r\n" on windows machine, including the last line. You can write your programs on whatever OS you like, but please verify that they can be compiled and run correctly on the virtual machine we provided because we will grade your assignment there. You should strictly follow the format as stated in the following.

- Each line of "master.txt" contains 4 fields of fixed length for an account. For the meaning of each field, please refer to Section 2.1.1.

  1. *Account Holder Name*: a string of 20 characters (spaces are padded at the end in case the card-holder name is less than 20 characters).
  2. *Account Number*: a 16-digit number (the unique number for each account).
  3. *Password*: a 6-digit number.

4. *Balance*: a 16-place decimal number with two decimal places, the first place is "+" or "-", denoting whether the balance is positive or not (having leading zeros if it is less than 15 places after the symbol).

You may assume the input file strictly follow the format specified in Section 2.1.1.

2.4 Output Files Format Specification

As far as the outline banking system is concerned, there are 7 output files. The ATMs program outputs 2 files ("trans711.txt", and "trans713.txt"), and the Central Machine program outputs 5 files ("transSorted711.txt", "transSorted713.txt", "transSorted.txt", "updatedMaster.txt", and "negReport.txt").

The format of "trans711.txt", "trans713.txt", "transSorted711.txt", "transSorted713.txt", and "transSorted.txt" are the same. These 5 files should strictly follow the format as stated in the following:

- Each line contains 4 fields of fixed lengths of a transaction. In "transSorted711.txt" and "transSorted713.txt", the lines should be sorted from smallest to largest by the account number first, and then sorted from smallest to largest by the timestamp in case of ties. The "transSorted.txt" file should be ordered the same. For the meaning of each field, you can refer to the Section 2.1.1.
  1. *Account Number*: a 16-digit number.
  2. *Operation*: 1 character, "D" and "W".
  3. *Amount*: a 7-place decimal number with two decimal places (having leading zeros if it is less than 7 places).
  4. *Timestamp*: a 5-digit number (having leading zeros if it has less than 5 digits).

The format of "updatedMaster.txt" is the same as "master.txt", please refer to the Section 2.3.

For "negReport.txt", you should strictly follow the format as stated in the following:

- Each line of "negReport.txt" contains 3 fields of fixed length for an account. For the meaning of each field, please refer to Section 2.1.2.
  1. *Account Holder Name*: a string of 20 characters (spaces are padded at the end in case the account holder name is less than 20 characters).
  2. *Account Number*: a 16-digit number (the unique number for each account).
  3. *Balance*: a 16-place decimal number with two decimal places, the first place is "+" or "-", representing whether the balance is positive or not (having leading zeros if it has less than 15 places after the symbol).

You should strictly follow all the output files strictly follow the format specified in Section 2.1.2.

3 Report

You should give a simple report to comment on whether COBOL and C are more appropriate in implementing the ATMs program and the Central Machine program within one PDF file. The following provides guidance to your analysis.

1. Compare the conveniences and difficulties in implementing the Simulated Banking System in COBOL and C. You can divide the implementation into specific tasks such as "reading file in certain format", "simulating loops", "procedure/function call" and so on. Give code segments in your programs to support your explanation.
2. Compare COBOL with modern programming languages (e.g. Java/Python/...) from different aspects (e.g. variable declarations, paradigm, data type, parameter parsing, ...). You are free to pick your favorite modern programming language.
3. Do you think COBOL is suitable for writing applications like in this assignment, especially when some of the input needs to be passed several times? Explain in terms of, say, the aspect like programming difficulty, efficiency of you program, etc.

4 Submission Guidelines

Please read the guidelines CAREFULLY. If you fail to meet the deadline because of submission problem on your side, marks will still be deducted.

The late submission policy is as follows:

- 1 day late: -20 marks

- 2 days late: -40 marks

- 3 days late: -100 marks

So please start your work early!

1. In the following, **SUPPOSE**

   your name is *Chan Tai Man*,
   your student ID is *1155234567*,
   your username is *tmchan*, and
   your email address is *tmchan@cse.cuhk.edu.hk*.

2. In your source files, insert the following header. REMEMBER to insert the header according to the comment rule of COBOL and C.

```
/*
 * 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 : Chan Tai Man
 * Student ID : 1155234567
 * Email Addr : tmchan@cse.cuhk.edu.hk
 */
```

   The sample file header is available at

`http://course.cse.cuhk.edu.hk/~csci3180/resource/header.txt`

3. Make sure you compile and run the COBOL program without any problem with OpenCOBOL 1.1.0 and gcc 2.95.2 on Windows computers in SHB924/904. We will grade your works based on those machines.

4. The report should be submitted to VeriGuide, which will generate a submission receipt. The report and receipt should be submitted together with your COBOL and C codes in the same ZIP archive.

5. The COBOL source should have the filename "atms.cob" and "central.cob". The C source should have the filename "atms.c" and "central.c". The report should have the filename "report.pdf". The VeriGuide receipt of report should have the filename "receipt.pdf". All file naming should be followed strictly and without the quotes.

6. `Tar` your source files to `username.tar` by

   ```
   tar cvf tmchan.tar atms.cob central.cob atms.c central.c report.pdf receipt.pdf
   ```

7. `Gzip` the `tarred` file to `username.tar.gz` by

   ```
   gzip tmchan.tar
   ```

8. `Uuencode` the `gzipped` file and send it to the course account with the email title "HW1 *studentID yourName*" by

   ```
   uuencode tmchan.tar.gz tmchan.tar.gz \
   | mailx -s "HW1 1155234567 Chan Tai Man" csci3180@cse.cuhk.edu.hk
   ```

9. Please submit your assignment using your Unix accounts.

10. An acknowledgement email will be sent to you if your assignment is received. **DO NOT** delete or modify the acknowledgement email. You should contact your TAs for help if you do not receive the acknowledgement email within 5 minutes after your submission. **DO NOT** re-submit just because you do not receive the acknowledgement email.

11. You can check your submission status at

    `http://course.cse.cuhk.edu.hk/~csci3180/submit/hw1.html`.

12. You can re-submit your assignment, but we will only grade the latest submission.

13. Enjoy your work :>