CSCI204/MCS9204 Assignment 1

(Total 10 marks, Due by 11:59 pm sharp on Sunday, 27 March, 2016)

Aims

This assignment aims to establish a basic familiarity with C++ classes. The assignment introduce increasingly object-based, C++ style of solution to a problem.

General Requirements

- You should observe the common principles of OO programming when you design your classes.
- You should make proper documentation and implementation comments in your codes where they are necessary.
- Logical structures and statements are properly used for specific purposes.

Objectives

On completion of these tasks you should be able to:

- code and run C++ programs using the development environment.
- make effective use of the on-line documentation system that supports the development environment.
- code programs using C++ in a hybrid style (procedural code using instances of simple classes) and in a more object-based style.
- manipulate string data.

Task:

In these tasks, you will implement C++ program to simulate simpler ATM transactions. There is no security problems need to be considered in this subject.

Define a class **Account** in a file **Account.h** which contains the data members: customer id, BSB number, account number, first name, last name, balance, and total amount cash already has been withdrawn for today.
Define necessary constructors and member functions in the file **Account.h**. Implement constructors and member functions in a file **Account.cpp**.

Define a class **ATM** in a file **ATM.h**. It should contain data members: ATM id, total cash of $50 notes, and total cash of $20 notes in the boxes.
Define necessary constructors and member functions in the file **ATM.h**. Implement the constructors and functions in a file **ATM.cpp**.

Customer accounts' information can be loaded from a text file **accounts.txt** into a **dynamic array** (Account type, not STL) that used for all ATM transactions.

The sample text file **accounts.txt** can be downloaded from the site of this assignment. The data of the text file are as follows.

```
5
1234567,802134,11213465,Jack,Steven,2300.2
……
```

The first line is the total number of account records, then following each line stores one record of an account.

Note: The value of "total amount cash already has been withdrawn for today" can be set as zero at beginning.

ATMs' information can be obtained from a text file **ATM.txt**. At the beginning of the file, a number indicates the total number of ATMs that available. Each line of the file contains:
ATM ID, amount of $50 notes, amount of $20 notes

Implement C++ code in a file **main.cpp** to get accounts' file and ATMs' file from **the command line arguments**, create dynamic arrays to store accounts information and ATMs information, load data from the given files into the dynamic arrays of accounts and ATMs, implement C++ code in **main.cpp** to process transactions' simulations described below.

```
./ass1 accounts.txt ATM.txt
Accounts have been loaded
ATMs have been loaded
```

The program will display a menu to ask a user to choose. (Text in red indicates user's inputs)

1. Withdraw
2. Find balance
3. Add notes
4. Quit

Please choose: 1

When a customer want to withdraw cash from an ATM, the program randomly select one ATM (use pseudo random number generator to generate an ATM ID), ask the customer to input BSB number, account number, and amount of cash.

BSB: 802134
Account: 11213465
Amount: 160

The program will verify the account; check if the balance is bigger than the required cash; then the program will compute how many $50 notes and how many $20 notes should be given.

Finally the program will modify the balance of the customer's account; update the ATM cash boxes; write data of the transaction in a log file **ATM.log**; and display

You have 2×$50 notes, 3×$20 notes. Your balance is $2140.2.

Hint: Use greedy algorithm to compute number of different notes.

The log file format for above transaction should be:
ATM ID, system date, withdraw, BSB number, account number, amount of cash.
For example:
800001, 01/08/2012 13:23, withdraw, 802134, 11213465, 160.

When an input amount cannot be withdrawn, e.g. $30, the program should prompt a message, such as "The amount of cash cannot be withdrawn".

When notes in a cash box are less than $2,000, e.g. ATM ID 800001 is short of $50 notes, the program should prompt message "Please add more notes for ATM $50 note box".

When "2. Find balance" has been chosen, the program will ask a user to input BSB number and account number, then display the balance of the account.

BSB: 802134
Account: 11213465
Your balance is $2140.2

When "3. Add notes" has been selected, the program will ask the user to input

ATM ID: 800001
Note type (1-$50, 2-$20): 1
Amount: 20000

The program will update the total cash of $50 notes for ATM ID 800001, and write data in the log file ATM.log. The format of the log record is as following
ATM ID, system date and time, transaction description, amount, balance.
For example:
800001, 01/08/2012 18:00:00, Add $50 notes, 20000, 29000.

When “4. Quit” is selected, the program should save accounts data into **accounts.txt**, and save ATM data into **ATM.txt**, then quit.

Testing:

Use g++ to compile the source files by

```
g++ –o ass1 main.cpp Account.cpp ATM.cpp
```

and run the program by

```
./ass1 accounts.txt ATM.txt
```

The input testing files **accounts.txt** and **ATM.txt** can be downloaded from the assignment site.

You can use bcheck to check if there is any memory leak by

```
bcheck ./ass1 accounts.txt ATM.txt
```

Note: Your program should work on different testing data. **Do not define constant files' names inside the source code.** You should keep the program running until “4. Quit” is selected.

Submission

This assignment is due by 11.59 pm (sharp) on Sunday, 27 March, 2016.

Assignments are submitted electronically via the ***submit*** system.

For this assignment you must submit the files **Account.h, Account.cpp, ATM.h, ATM.cpp and main.cpp** via the command:

```
$ submit -u your_user_name -c CSCI204 -a 1 Account.h Account.cpp ATM.h ATM.cpp main.cpp
```

and input your password.

Make sure that you use the correct file names. The Unix system is case sensitive. You must submit all files in one ***submit*** command line.

Your program code must be in a good programming style, such as good names for variables, methods, classes, and keep indentation.

Submission via e-mail is NOT acceptable.

After submit your assignment successfully, please check your email of confirmation. **You would loss 50% ~ 100% of the marks if your program codes could not be compiled correctly.**

Late submissions do not have to be requested. Late submissions will be allowed for a few days after close of scheduled submission (up to 3 days). Late submissions attract a mark penalty; this penalty may be waived if an appropriate request for special consideration (for medical or similar problem) is made via the university SOLS system *before* the close of the late submission time. No work can be submitted after the late submission time.

A policy regarding late submissions is included in the course outline.

The assignment is an **individual assignment** and it is expected that all its tasks will be solved **individually without any cooperation** with the other students. If you have any doubts, questions, etc. please consult your lecturer or tutor during tutorial classes or office hours. Plagiarism will result in a **FAIL** grade being recorded for that assessment task.

End of specification