CS2720—Fall 2009
Assignment 1

Due: October 1, 2009 11:59 pm

- Your assignment will be marked on program correctness and readability (including comments).
- Doxygen documentation should be used.
- Write a Makefile which can be used to compile all programs in this assignment.
- Each class should be implemented in its own `.h` and `.cc` files.
- Use `const` member functions when appropriate.
- For simplicity, you may assume that your program will never be supplied invalid input.

In this problem, you will write a program that manages bank accounts.

1. Write an abstract `Customer` class which is an abstraction for a customer. The `Customer` class should have the following member functions:

   - A virtual destructor. (Aside: why do we need this?)
   - A pure virtual `void read(istream &is);` function which reads the input stream and initializes the object. **DO NOT** output any prompts in this function (and its overridden versions).
   - A pure virtual `void write(ostream &os);` function which writes the object to the output stream.

   In addition, write input and output operators (`operator>>` and `operator<<`) for the `Customer` class to perform input/output operations on the specified input/output stream.

2. Derive a `PersonalCustomer` class from the `Customer` class. It stores the extra data `name` (`string`) and `SIN` (`string`). Override the necessary functions. For the input format for `read`, the customer's information is given in one line:

```
P 123456789 John Doe
```

Each line starts with a `P`, followed by the SIN, followed by the name. Note that the name continues until the end of the line and may contain spaces.

The `write` function should write the information in the same format.

3. Derive a `BusinessCustomer` class from the `Customer` class. It stores the extra data `name` (`string`) and `registration` (`string`). Override the necessary functions. For the input format for `read`, the customer's information is given in one line:

   ```
   B 123456789X Programming R Us Ltd
   ```

   Each line starts with a `B`, followed by the registration, followed by the name.

   The `write` function should write the information in the same format.

4. Write an `Account` class which is an abstraction of bank accounts. It should have three data members: a pointer to a `Customer` object, an `accountNumber` (string), and a `balance` (integer). Provide the following member functions:

   - A default constructor: initializes the customer pointer to `NULL`.
   - A virtual destructor.
   - A virtual `void read(istream &is);` function which reads the input stream and initializes the object. **DO NOT** output any prompts in this function. The input format is:

     ```
     B 123456789X Programming R Us Ltd
     12345 999
     ```

     Each record starts with a `Customer` specification (either Personal or Business) on the first line, and the account number and the account balance on the second line.

     This function should read the first letter of the line and decide whether to allocate and read a Personal and Business customer. You may assume that the first letter is either `P` or `B`. Use the `operator>>` to read the `Customer` object.
   - A virtual `void write(ostream &os);` which writes the account information to the output stream in the format specified above.
   - An accessor `string getAccountNumber();` which returns the account number.
   - A virtual function `void deposit(int amount);` which increases the balance by `amount`.
   - A virtual function `void withdraw(int amount);` which decreases the balance by `amount`. **DO NOT** worry about checking for overdraft. The balance is allowed to be negative for this assignment.

- A virtual function `void updateMonthEnd();` which performs updates at the end of a month. It does nothing in this class.

In addition, write input and output operators (`operator>>` and `operator<<`) for the `Account` class to perform input/output operations on the specified input/output stream.

5. Derive a `SavingsAccount` class from `Account` representing a savings account. The only additional information is an `interestRate` (double) representing the monthly interest rate as a percentage.

   Provide the following member functions:

   - Override the `read()` and `write()` functions. The input format is the same as the `Account` class, except that the third line contains a floating-point number specifying the monthly interest rate.

     You may wish to write protected helper functions `readExtra()` and `writeExtra()` to read and write the extra information in `SavingsAccount`, and call them in `read()` and `write()`. This will make things easier in the other subclasses.
   - Override the `updateMonthEnd()` function to increase the balance based on the interest calculated. Truncate any fractional interest.

6. Derive a `ChequingAccount` class from `Account` representing a chequing account. The additional information are `freeWithdrawals` (integer) indicating the number of free withdrawals per month, `withdrawals` (integer) indicating the number of withdrawals in the current month, and `withdrawalFee` (integer) indicating the fee for each withdrawal over the number of free withdrawals.

   Provide the following member functions:

   - Override the `read()` and `write()` functions. The input format is the same as the `Account` class, except that the third line contains the three integers `freeWithdrawals`, `withdrawals`, and `withdrawalFee` separated by spaces.
   - Override the `updateMonthEnd()` function to reset `withdrawals`.
   - Override the `withdraw()` function to keep track of the number of withdrawals, and to decrease the balance by the additional `withdrawalFee` when the number of withdrawals (including the one being processed) exceeds the number of free withdrawals.

7. Derive a `ChequingSavingsAccount` class fromk `ChequingAccount` and `SavingsAccount`.

   Provide the following member function:

   - Override the `read()` and `write()` functions. The input format is the same as the `Account` class, followed by a line containing the extra information for the `SavingsAccount`, followed by a line containing the extra information for the `ChequingAccount`. In other words:

```
Line 1: customer information
Line 2: account number and balance
Line 3: interest rate
Line 4: freeWithdrawals withdrawals withdrawalFee
```

- Override all other functions if necessary to combine the functionality of both `ChequingAccount` and `SavingsAccount`.

8. Write a main program (`bank`) which first asks the user to enter a file name containing the account information. The program then reads the account information into a vector of `Account` pointers.

   The format of the account file is as follows. Each account starts with a line containing `S` if it is a savings account, `C` if it is a chequing account, or `CS` if it is a chequing savings account. This is followed by the corresponding account information in the format expected by the `read()` function of the appropriate type of account.

   The program repeatedly presents the user with a menu and perform the appropriate actions:

   - List (L)—prints the information of each account to the screen using `operator<<` for `Account`.
   - Withdraw (W)—asks the user for an account number and an amount, and withdraw the specified amount from the appropriate account. Use linear search to find the correct account. You may assume that the user enters a correct account number and there is exactly one account with the given number.
   - Deposit (D)—asks the user for an account number and an amount, and deposit the specified amount into the appropriate account. Use linear search to find the correct account. You may assume that the user enters a correct account number and there is exactly one account with the given number.
   - Update (U)—performs end-of-month update on all accounts.
   - Quit (Q)—quits the program.