CS 1410 Introduction to Computer Science – CS2
Assignment #4: File I/O
Total Points: 50 points

Problem description

Assume that you are working for a local bank to develop a new banking system. The current system stores bank accounts in a text file, commonly called *"comma separated values"* (CSV). The first line in this file contains the field names and remaining ones have detailed information of the stored accounts, each line for an account. All data values are separated by commas. This is an example snippet of this file.

```
Account,Name,Balance
333,Kate Wilson,1512.34
101,Adam Smith,100.23
212,Mary Lee,-10.56
```

It is easy to see that this is not an efficient way to store and manage data. For example, when a customer deposits a check, the current system needs to search for his account and update the new balance. Because this is a text file, to search for an account, the system needs to read all the file content until it finds the corresponding account. Then, it needs to rewrite the whole file so the line for that account can be updated with the new balance. (There is no other way to update a line in a text file).

Thus, in the new system, you decide to store bank accounts in a binary file because you could read and update a specified part of a binary file at the same time without touching others. In addition, you can also maintain an in-memory index which can help the search for the accounts stored in the file faster. You call this file format BNK and design it with the following structure.

1. Header
The first part of a BNK file is the header. It is 32-byte long and stores the following information:

- The first 4 bytes is the char sequence BANK, which is used as a signature for BNK files. That means, if a file does not contain 4 characters BANK at the beginning, it is not a BNK file.

- The next item is the total number of accounts, stored as a 4-byte integer.

- The remaining 24 bytes are space reversed for future usages.

This header can be declared in C++ as a struct, for example:

```
struct BNKHeader {
      char signature[4]; // {'B','A','N','K'};
      int numberOfAccounts;
      char reserved[24];
};
```

2. Account Data

If N is the total number of accounts, there will be N account records stored consecutively after the header. Each record stores the account number (a 4-byte integer), the holder name (at most 20 characters including the ending NULL), the balance (a double value of 8 bytes), and a reserved space of 96 bytes. That means, the size of an account record is 128 bytes. We can declare such a record as a struct in C++ as the following:

```
struct BNKAccount {
      int number;
      char name[20];
      double balance;
      char reserved[96];
};
```

3. Index Data

The last part of a BNK file contains N consecutive index records. Each record contains the account number and the position of the corresponding account record in the BNK file. For those index records, the account numbers are sorted increasingly, so we can use binary search later on it. For example, for the account data given above in the CSV file, the record of account 333 is stored as position 32 (right after the 32-byte header), that of account 101 is stored at position 32+128=160, and that of account 212 is stored at position 160+128=288. The index data will contain the following records: {101, 160}, {212, 288}, and {333, 32}. With such file positions, we can access an account easily. For example, after loading the index into memory, if we want to know the record of account 212, we will search for account number 212 in the index and find its position is 288 in the BNK file.

Index records can be declared using the following struct in C++

```
struct BNKIndex {
      int accountNumber;
      long filePosition;
};
```

Attention: To ensure the portibility of your code, you should always use function **sizeof** to compute the data size of those structs. For example, to read to the memory an array of N BNKIndex structs from a binary file, you could use this statement:

```
file.read((char*) index, N * sizeof(BNKIndex));
```

Programing tasks

Task 1 (25 points). You first need to write a program to convert a CSV file to a BNK file. This program can work following this procedure:

[1 point] Open the CSV file (as text, for input) and BNK file (as binary, for output).

[5 points] Read quickly through the CSV file (e.g. just counting the number of lines without parsing each line) to determine the total number of accounts (N).

[2 points] Put N into a BNKHeader struct and write it to the BNK file.

[2 points] Allocate a (dynamic) array of N BNKIndex records for the in-memory index data.

[10 points] Re-read the CSV file line by line, parse each line into a BNKAccount record, and write it to the BNK file. Before writing, you also add to the in-memory index array the information for this BNKAccount record (e.g. its account number and its corresponding file position, which is provided by function **tellp**).

[5 points] After parsing the CSV file, you sort the index array by account numbers, and write it to BNK file.

Task 2 (25 points). Now, you write a program to search and update accounts in a BNK file. This program can work following this procedure:

[2 points] Open the BNK file and check whether it contains the signature BANK.

[5 points] Load the index into memory. Hint: If this file has N accounts, the index data will be stored at position `sizeof(BNKHeader) + N * sizeof(BNKAccount)`.

[18 points] Print a menu for the user with the following functions: display an account [5 points], deposit to an account [5 points], and withdraw from an account [5 points]. For each function, the user will provide the account number and your program uses the index to locate the corresponding account record in the BNK file (e.g. using function **seekg**) and display or update it. You should implement a binary search function on the index [3 points] to find the file position for a given account number.

You will need to submit your solution as two .cpp files, each implementing a program. Those files should be named convert.cpp and manage.cpp, respectively. You should also prepare and test your program with a CSV file with at least 10 accounts. You can use Excel to create this file or generate it automatically. The .cpp files and the test CSV file should be compressed and submitted as a file named as Firstname_Lastname_HW4.zip