HW2: Bank Accounts - using Structures/Classes:

You have been hired as a programmer by a major bank. Your first project is a small banking transaction system. Each account consists of a number and a balance. The user of the program (the teller) can create a new account, as well as perform deposits, withdrawals, balance inquiries, close accounts, etc..

Initially, the account information of existing customers is to be read into an array of BankAccount structures (or simple classes). **The data members of the structure will include; first name, last name, social security number, account number, account type (Checking, Savings, or CD), and account balance**. The bank can handle up to **MAX_NUM** accounts. Use the following function to read in the data values:

```
int read_accts(BankAccount account[], int max_accts);
```

This function fills up the array (up to **max_accts**) and returns the actual number of accounts read in (referred to as **num_accts**).

After initialization, **print the initial database of accounts**. Use function print_accts() described below.

The program then allows the user to select from the following menu of transactions:

Select one of the following:
- W - Withdrawal
- D - Deposit
- N - New account
- B - Balance
- I - Account Info
- X - Delete Account
- Q - Quit

Use the following function to produce the menu:

```
void menu();
```

This function only displays the menu. The **main program** then prompts the user for a selection. You should verify that the user has typed in a valid selection (otherwise print out an error message and repeat the prompt).

Once the user has entered a selection, one of the following functions should be called to perform the specific transaction. **At the end, before the user quits, the program prints the contents of the database**.

```
int findacct(const BankAccount account[], int num_accts, int requested_account);
```

This function **returns the index of requested_account in the array account** if the account exists, and -1 if it doesn't. It is called by all the remaining functions.

```
void withdrawal(BankAccount account[], int num_accts);
```

This function prompts the user for the account number. If the account does not exist, it prints an error message. Otherwise, it asks the user for the amount of the withdrawal. If the account does not contain sufficient funds, an it prints an error message and does not perform the transaction.

```
void deposit(BankAccount account[], int num_accts);
```

This function prompts the user for the account number. If the account does not exist, it prints an error message. Otherwise, it asks the user for the amount of the deposit.

```
int new_acct(BankAccount account[], int num_accts);
```

This function prompts the user for a new account number. If the account already exists, it prints an error message. Otherwise, it adds the account to the database. **The program then prompts the user to enter the new depositor's first name, last name, social security number, the account type (Checking, Savings, or CD), and the initial opening deposit.**. The function returns the new number of accounts in the database.

```
int delete_acct(BankAccount account[], int num_accts);
```

This function prompts the user for an account number. If the account does not exist, or if the account exists but has a non-zero balance, it prints an error message. Otherwise, it closes and deletes the account. It returns the new number of accounts.

```
void balance(const BankAccount account[], int num_accts);
```

This function prompts the user for an account number. If the account does not exist, it prints an error message. Otherwise, it prints the account balance.

```
void account_info(const BankAccount account[], int num_accts);
```

This function prompts the user for a **social security number**. If the account does not exist, it prints an error message. Otherwise, it prints the complete account information for the account requested .

```
void print_accts(const BankAccount account[], int num_accts);
```

This function prints a table of the complete account information for every active account.

Make sure to use enough test cases so as to completely test program functionality.

EXTRA CREDIT #1: Use nested structures/classes:
1. A BankAccount consists of a Depositor, an account number, an account type, and a balance.
2. A Depositor has a Name and a social security number.
3. A Name consists of first and last names.

EXTRA CREDIT #2:
Use a constructor to initialize the data members of a new account (including the initial accounts of the database).
Hint: a constructor is a function that can be called.

Notes:
1. All output must be file directed
2. Only output must go to the file - not interactive prompts.
3. No global variables are allowed
4. The program and all functions must be properly commented.