In this project, you will build a program that processes bank transactions. Your program will maintain information about banks and user accounts. Your program will permit users to open accounts, make deposits/withdrawals, and close accounts. Because you will not be told in advance how many banks or users there will be, nor the size of text information associated with the banks and users, you must employ dynamic data structures to implement your project.

1 Input

Your program will receive transactions from standard input, and perform processing in response to each transaction. We have provided a program, called xact_driver, that creates the transactions. This program is posted on the course website which can be found in the following directory on Glue:

/dept/enee/public_html/class/enee150.S2018/assignments/pr3

You can copy the program to your home directory if you “cd” to the above directory and type cp xact_driver ~. The xact_driver program takes a single command line argument, which specifies a test case number, and then writes to standard output a sequence of transactions. Because the transactions are written to standard output, you can redirect them to the standard input of your program by using the | operator on the UNIX command line. For example, to test your program on test case #1 (assuming your program is called “pr3”), you can type:

xact_driver 1 | pr3

The above command is how we will grade your project. However, to enable you to develop your project on non-Glue platforms, we have also provided the test cases as files on the course website. You can download these test case files from the course website (just follow the “Project 3 Files” hyperlink), or by copying them from the same directory given above. You can redirect each one of these test cases into the standard input of your program using the < operator on the UNIX command line. For example, your program can run on test case #1 by typing:

pr3 < test1.in
It is important to emphasize that your program must receive the transactions from standard input. You are not allowed to access the transactions in the test case files by performing file I/O from your program. Regardless of where you develop your program, we recommend that you eventually copy your code to the Glue system and try all test cases by redirecting output from the xact driver program directly into the standard input of your program.

2 Transactions and Data Fields

Each transaction that the xact driver program sends to your program is a sequence of ASCII characters. In total, there are 8 different types of transactions, each identified by a transaction ID numbered 1–8:

<table>
<thead>
<tr>
<th>ID</th>
<th>Transaction</th>
<th>ID</th>
<th>Transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Terminate</td>
<td>5</td>
<td>Print Account</td>
</tr>
<tr>
<td>2</td>
<td>New Bank</td>
<td>6</td>
<td>Deposit</td>
</tr>
<tr>
<td>3</td>
<td>Print Bank</td>
<td>7</td>
<td>Withdrawal</td>
</tr>
<tr>
<td>4</td>
<td>New Account</td>
<td>8</td>
<td>Close Account</td>
</tr>
</tbody>
</table>

Each transaction will begin with a transaction ID that specifies the type of transaction being sent. The transaction ID will be immediately followed by a new-line character. After the transaction ID is sent, usually one or more data fields will also be sent depending on the transaction. Section 3 will describe the number and type of data fields sent for each transaction listed above.

A single data field within a transaction consists of a sequence of characters terminated by a new-line character. All the data fields are sent back-to-back—i.e., the first character of a data field is sent immediately after the new-line character terminating the previous data field (or transaction ID if it’s the first data field in the transaction). When the last data field and new-line character are sent, the transaction is complete, and the transaction ID for the next transaction will be sent.

Your program must parse the transactions and their associated data fields as they are sent by the xact driver program. For data fields containing a single number, you can simply use scanf (with the appropriate formatting placeholder) to convert the data field into its corresponding numeric value. For data fields containing text information, you will not be told beforehand how many characters will be in the data field. Instead, your program must be able to handle an unbounded size for all text data fields. To do this, your program should read text data fields into character arrays, and dynamically increase the size of the character array if its original size is insufficient to store the entire data field.
3 Transaction Format

3.1 Terminate

The “terminate” transaction is specified with ID = 1. When your program sees a terminate transaction, it should exit. The terminate transaction is always the last transaction sent by the \texttt{xact\_driver} program. This transaction has no data fields; its format is:

$$1\backslash n$$

3.2 New Bank

The “new bank” transaction is specified with ID = 2, and provides the information for a bank. The bank information (consisting of the bank’s name and address) is sent in 5 data fields after the transaction ID. The format for the new bank transaction is:

$$2\backslash n$$
$$<\text{Bank name}>\backslash n$$
$$<\text{Street address}>\backslash n$$
$$<\text{City}>\backslash n$$
$$<\text{State}>\backslash n$$
$$<\text{Zip Code}>\backslash n$$

The first 4 data fields are variable-sized text; the last data field represents an integer value. There is no upper bound on the number of new bank transactions sent by the \texttt{xact\_driver} program. \textit{You are required to store the information for all the banks in a linked list}, where each link node in the list stores the information for a single bank.

3.3 Print Bank

The “print bank” transaction is specified with ID = 3, and requests your program to print the information associated with a bank. The bank to print is specified by a single data field. The format for the print bank transaction is:

$$3\backslash n$$
$$<\text{Bank name}>\backslash n$$

When printing the requested bank information, first print a line with three asterisks (“***”). Next, print “Bank:” followed by the name of the bank, “Address:” followed by the bank’s street address, comma, city, comma, state, and zip code (all on a single line). Finally, print “Holdings:” followed by the amount of money held in the bank. (More on how to compute the bank’s holdings later).
Test case #1 in \texttt{xact\_driver} sends a single new bank transaction, then a single print bank transaction, then a terminate transaction. Here’s the correct output for this test case (it’s provided in \texttt{test1.out}; your output should match it exactly):

\texttt{z: xact\_driver 1 | pr3}

\texttt{***}

\texttt{Bank: SunTrust}
\texttt{Address: 53 Bernal Avenue, Nashville-Davidson, Tennessee 10302}
\texttt{Holdings: $0.00}

### 3.4 New Account

The “new account” transaction is specified with ID = 4, and provides the information for a new bank account. The account information (consisting of the account owner’s last name, first name, and address, as well as the bank and account number associated with the account) is sent in 8 data fields after the transaction ID. The format for the new account transaction is:

\begin{verbatim}
4\n<Last name>\n<First name>\n<Street address>\n<City>\n<State>\n<Zip Code>\n<Bank name>\n<Account number>
\end{verbatim}

The first 5 data fields and the 7th data field are variable-sized text; the 6th and 8th data fields represent integer values. In addition to the above information, you must also maintain a balance for each account, which is a floating point value. Initially, each account should have a balance of $0.

There is no upper bound on the number of new account transactions sent by the \texttt{xact\_driver} program. \textit{You are required to store the information for all the accounts in a linked list}, where each link node in the list stores the information for a single account.

### 3.5 Print Account

The “print account” transaction is specified with ID = 5, and requests your program to print the information associated with an account. The account to print is specified as an account number in a single data field. The format for the print account transaction is:

\begin{verbatim}
5\n<Account number>\n\end{verbatim}
When printing the requested account information, first print a line with three asterisks ("***"). Next, print “Name:” followed by the account owner’s last name, comma, and first name on a single line. Then, print “Bank:” followed by the name of the bank for the account. Next, print “Account Number:” followed by the account number. Then, print “Address:” followed by the account owner’s street address, comma, city, comma, state, and zip code (all on a single line). Finally, print “Balance:” followed by the account’s balance. When printing the balance, only print 2 places after the decimal point.

Test case #2 in xact_driver sends a new bank transaction, then a new account transaction, then a print account transaction, and then a terminate transaction. Here’s the correct output for this test case (it’s provided in test2.out; your output should match it exactly):

```
z: xact_driver 2 | pr3
***
Name: Dawson, Tano
Bank: SunTrust
Account Number: 263339
Address: 230 Powell Street, Newark, New Jersey 32350
Balance: $0.00
```

### 3.6 Deposit

The “deposit” transaction is specified with ID = 6, and performs a deposit into an existing account. The account number to receive the deposit is specified as a data field with a single integer value, and the amount of the deposit is specified as a data field with a single floating point value. The format for the deposit transaction is:

```
6\n<Account number>\n<Deposit amount>\n```

When your program receives a deposit transaction, you should find the specified account in your accounts linked list, and increment the account’s balance by the deposit amount. You may assume that the account number specified by a deposit transaction is always an account that was created by a previous new account transaction.

### 3.7 Withdrawal

The “withdrawal” transaction is specified with ID = 7, and performs a withdrawal from an account. The format for the withdrawal transaction is similar to the deposit transaction:

```
7\n<Account number>\n<Withdrawal amount>\n```
When your program receives a withdrawal transaction, you should find the specified account in your accounts linked list, and decrement the account’s balance by the withdrawal amount. You may assume that the account number specified by a withdrawal transaction is always an account that was created by a previous new account transaction. You may also assume that the withdrawal amount will never exceed the account balance.

Test case #3 in xact_driver sends a new bank transaction, then a new account transaction, then a deposit transaction followed by a withdrawal transaction, then a print account transaction followed by a print bank transaction, and then a terminate transaction. Here’s the correct output for this test case (it’s provided in test3.out; your output should match it exactly):

```
z: xact_driver 3 | pr3
***
Name: Dawson, Tano
Bank: SunTrust
Account Number: 263339
Address: 230 Powell Street, Newark, New Jersey 32350
Balance: $950.67
***
Bank: SunTrust
Address: 53 Bernal Avenue, Nashville-Davidson, Tennessee 10302
Holdings: $950.67
```

Notice the holdings value of the SunTrust bank is $950.67. This is because in test case #3, the SunTrust bank contains a single account with a balance of $950.67. In general, the holdings value of a bank is the sum of the balances of all accounts contained within that bank. Your program should always print the bank’s holdings value when it receives a print bank transaction.

### 3.8 Close Account

The “close account” transaction is specified with ID = 8, and closes an account. The account number of the account to close is specified as a single integer value in a data field. The format for the close account transaction is:

```
8
<Account number>
```

When your program receives a close account transaction, you should find the specified account in your accounts linked list, and delete the corresponding link node from the linked list. Be sure to free the link node as well as any malloc’d arrays associated with the link node to prevent a memory leak. You may assume that the account number specified by a close account transaction is always an account that was created by a previous new
account transaction. Once closed, the account’s balance does not contribute to its bank’s holdings value when responding to future print bank transactions.

Test case #4 in xact_driver sends a new bank transaction, then 3 new account transactions, then 2 deposit and 2 withdrawal transactions, then a close account transaction, then a print account transaction followed by a print bank transaction, and then a terminate transaction. Here’s the correct output for this test case (it’s provided in test4.out; your output should match it exactly):

```
z: xact_driver 4 | pr3
***
Name: George, Brylea
Bank: SunTrust
Account Number: 583926
Address: 479 Townsend Street, Durham, North Carolina 80090
Balance: $860.67
***
Bank: SunTrust
Address: 53 Bernal Avenue, Nashville-Davidson, Tennessee 10302
Holdings: $1786.18
```

4 Other Test Cases

In addition to the 4 test cases described above, we have also provided 3 other test cases. To generate them, simply run xact_driver 5, xact_driver 6, and xact_driver 7 (or download test5.in, test6.in, and test7.in from the course website). These test cases have increasing numbers of transactions, with different mixes of new bank, new account, deposit, withdrawal, and close account transactions. As in the past, all of the test cases provided with this assignment will be used to grade your project, along with some unpublished test cases.