HW #4 (Parameterized stack)

Part A

- Write C++ code for the Dynamic Stack Abstract Data Type. Your member functions should be in compliance with the following definitions which you should put in the **Stack.h** file.

// ========== **Stack.h** ==========

typedef int DataType; // type of data items in the stack

const int MaxStackSize = 2; // initial value of _stackSize

HW #4 (2)

```
class Stack {
private:
  DataType *_stackList;      // array of DataTypes
  int _stackSize;            // the size of _stackList
  int _top;                  // the index of the top element

  // push item on top of the stack and extend the
  // stack if necessary
  void PushExtend(const DataType& item);
```

HW #4 (3)

```
public:
  Stack(void);          // constuct the stack

  // push item on top of the stack
  void Push(const DataType& item);

  DataType Pop(void);       // pop the stack
  void ClearStack(void);    // reset the value of _top to -1

  // return item on top of the stack
  DataType Top(void) const;
```

HW #4 (4)

```
  // check if the stack is empty
  bool IsStackEmpty(void) const;
  bool IsStackFull(void) const;       // check if the stack is full
};
// ================ Stack.h ends ================
```

- A static stack was given a fixed size (let us agree to call it **MaxStackSize**). When the number of items in the stack was equal to **MaxStackSize**, and a new item was added, the program would issue the stack-overflow error message and exit.

HW #4 (5)

- Unlike the static Stack, **your Stack** should extend itself when a new element is added and its size is equal to **MaxStackSize**.

- In your definition of Push, you should first check if the stack is full. If the stack is full, you should call PushExtend. PushExtend should allocate a new array of items, twice the size of the current array, copy the existing items into the new array, push the new item on top of the new array, delete _**stackList**, and set _**stackList** to the new array.

HW #4 (6)

- Write your C++ code for the private and public member functions of **Stack.h** in **Stack.cpp**. Remember to include **iostream.h**, **stdlib.h**, and **Stack.h** at the beginning of **Stack.cpp**. Test your implementation with the following program which you should save in **Driver.cpp**:

// ================ **Driver.cpp** ====================

```
#include <iostream.h>
#include "Stack.h"
```

HW #4 (7)

```
int main(int argc, char **argv) {
  Stack s; int i;

  for(i = 0; i < 10; i++) s.Push(i);
  for(i = 0; i < 10; i++) cout << s.Pop() << endl;
  if ( s.IsStackEmpty() ) cout << "Stack is empty" << endl;

  return 0;
}
// ================ Driver.cpp ends ================
```

HW #4 (8)

Part B

- The Stack that you implemented in Part A can handle only **integers**. Parameterize your implementation with templates so that your Stack (called **Stack2**) can handle other data types. Test your implementation with the following program, which you should save in **Driver2.cpp**:

// ================ **Driver2.cpp** ==================

```
#include <iostream.h>
#include <stdlib.h>
```

HW #4 (9)

```
int main(int argc, char **argv) {
  Stack<int> s; Stack<double> s2;
  int i; double j;

  for(i = 0; i < 10; i++) s.Push(i);
  for(i = 0; i < 10; i++) cout << s.Pop() << endl;
  if ( s.IsStackEmpty() ) cout << "Stack is empty" << endl;
```

HW #4 (10)

```
  for(i = 0; i < 10; i++) { j = i * 1.1; s2.Push(j); }
  for(i = 0; i < 10; i++) cout << s2.Pop() << endl;
  if ( s2.IsStackEmpty() ) cout << "Stack is empty" << endl;

  return 0;
}

// ================ Driver2.cpp ends ================
```

HW #4 (11)

Part C

- You will write a main program and several classes to create and print a small collection of bank accounts. You will also apply deposit and withdrawal transactions to those bank accounts.
- Input for this program consists of two files.
- The first file, named **accounts.txt**, contains a single **bank** object written out in ASCII character format, which means that you can use the >> operator to read the fields of these records.. This object will be read by the **read_accounts()** member function of the **bank** class (see the member function description below for additional details).

HW #4 (12)

- The second file, named **transactions.txt**, once again contains a series of transaction records in ASCII character format, which means that you can also use the >> operator to read the fields of these records. A typical transaction is shown below. The first field on the transaction record is the date of the transaction, followed by an account number, then the transaction type ('D' for deposit or 'W' for withdrawal), and finally a transaction amount.

  06/19 1111111111 D 430.00

- You will need to declare variables to hold the data read for each of these fields. To read transaction records until end of file is reached, use a loop like the following:

HW #4 (13)

```
while (trans_file >> date) {
    // Read remaining data of the transaction record.
    trans_file >> account_number;
    trans_file >> type;
    trans_file >> amount;
    // Process this transaction.
    . . .
}
```

where **trans_file** is the name of the **ifstream** variable opened for the transaction file.

HW #4 (14)

- You will write five files for this assignment:

1) account.h

- This header file will contain the class definition for a class called **account**. The **account** class represents information about a person's bank account. The header file should include an appropriate set of header guards to prevent it from being included more than once in the same source file.

Data Members

- The **account** class should have the following private data members:

HW #4 (15)

- An account number (a **char** array with room for 10 characters PLUS the null character, i.e., 11 elements total).
- A customer name (a **char** array with room for 20 characters PLUS the null character).
- A current account balance (a **double** variable).
- Note: Make that sure you code your data members in THE EXACT ORDER LISTED ABOVE and with THE EXACT SAME DATA TYPES. If you use **float** instead of **double** or only make the name array 20 characters long instead of 21, your program will not work correctly.

HW #4 (16)

Member Functions

- The **account** class definition should contain public prototypes for all of the member functions in the **account.cpp** source code file described below.
- This source code file will contain the member function definitions for the **account** class. The required member functions are described below:

2) account.cpp

- This source code file will contain the member function definitions for the **account** class. The required member functions are described below:

HW #4 (17)

Default constructor

- The default constructor should set the account number and customer name data members to the string literal "None". The account balance data member should be set to 0.

- **get_account_number()**: This member function has no parameters. It should return the account number.
- **get_balance()**: This member function has no parameters. It should return the current account balance.

HW #4 (18)

- **process_deposit()**: This member function should take a **double** deposit amount and add it to the balance for the bank account. It returns nothing.

- **process_withdrawal()**: This member function should take a **double** withdrawal amount. If the bank account's balance is less than the withdrawal amount, the member function should just return **false**. Otherwise, subtract the withdrawal amount from the balance of the bank account and return **true**.

HW #4 (19)

- **print()**: This member function has no parameters and returns nothing. It should print the values of the data members for the account in a format similar to the following:

  Account Number: 0003097439

  Name: John Smith

  Balance: $5234.38

HW #4 (20)

3) bank.h

- This header file will contain the class definition for a class called **bank**. The **bank** class represents information about a collection of bank accounts. The header file should include an appropriate set of header guards to prevent it from being included more than once in the same source file.

HW #4 (21)

Data Members

- The **bank** class should have the following three private data members:
- A bank name (a **char** array with room for 30 characters PLUS the null character).
- An array of 20 **account** objects.
- An **integer** that specifies the number of array elements that are filled with valid data.
- Note: Once again, make sure that you code your data members in the exact order listed above and with the exact same data types.

HW #4 (22)

Member Functions

- The **bank** class definition should contain public prototypes for all of the member functions in the **bank.cpp** source code file described below.

4) bank.cpp

- This source code file will contain the member function definitions for the **bank** class. The required member functions are described below:

HW #4 (23)

Default constructor

- The default constructor should set the bank name data member to the string literal "None". The number of accounts data member should be set to 0. No initialization is necessary for the array of **account** objects, since the **account** default constructor will automatically be called for every object in the array.
- **read_accounts()**: This member function takes one parameter, a string that contains the name of a file. This string parameter can be a C/C++ string. The function returns nothing.

HW #4 (24)

- This constructor should do the following:
- Declare and open an input file stream variable for the file name string passed in as a parameter.
- Check to make sure the file was opened successfully. If not, print an error message and exit the program.
- Read the database file into your **bank** object.
- Close the file stream variable.

HW #4 (25)

- Sort the account objects in the array in **ascending** order by account number using a sorting algorithm of your choice. Note that the account numbers are C/C++ strings, which means that you will **not** be able to compare them using the standard relational operators. The account number is also private data of the **account** class, so code in the **bank** will need to call **get_account_number()** for an **account** object rather than accessing the object's account number directly.

HW #4 (26)

- Note that the code described above will read data into **all** of the **account** data members. That includes the bank name, the array of 20 **account** objects, and the number of array elements filled with valid data. No further initialization of the data members will be needed.

- **process_transactions()**: This member function takes one parameter, a string that contains the name of a file of transaction data. This string parameter can be a C/C++ string. The function returns nothing.

HW #4 (27)

- The member function should open the specified transaction file for input. Make sure to test that the file was opened successfully; if it was not, print an error message and exit the program.

- Before reading any transactions, the function should print a report header and column headers. The function should then read transaction data from the file until end of file is reached.

HW #4 (28)

- Once all of the fields for a given transaction have been read, perform a **binary search** of the accounts array for the account number given in the transaction. If the account number from the transaction record is present in the accounts array, then the transaction may be processed. For a deposit, simply call the **process_deposit()** member function for the object that contains the matching account number, passing it the transaction amount. For a withdrawal, call the **process_withdrawal()** member function for the object that contains the matching account number, passing it the transaction amount.

HW #4 (29)

- For each transaction record processed, print a line in a transaction report with the data from the record and the updated balance for that account. If the transaction account number was not found in the account array or if a charge exceeded the account's credit limit (i.e., if the **process_withdrawal()** member function returned **false**), print an appropriate error message instead of the account balance.

- After all transactions have been processed, close the transaction file.

HW #4 (30)

- **print()**: This member function takes no parameters and returns nothing.
- This member function should first print a descriptive header line that includes the bank name (e.g., "Account Listing for First National Bank"). It should then loop through the array of **account** objects and print each of the elements that contains account data (i.e., element 0 up to but not including element number of accounts), with a blank line between each account.

HW #4 (31)

- Here we see some of the power of object-oriented programming. Since the **account** class has a **print()** member function, you can just call that function for each element of the array to print all the data members of each **account** object.
- You are welcome to write additional private member functions for the **bank** class as you see fit. For example, you may want to put your sorting algorithm code in its own member function and call it from **read_accounts()** or place the binary search code in its own member function and call it from **process_transactions()**.

HW #4 (32)

5) main.cpp

- This file will contain the program's **main()** function. The logic for this function is quite short:
- Declare a **bank** object.
- Use the **bank** object to call the member function **read_accounts()**, passing the file name "**accounts.txt**" as an argument to the function.
- Call the **print()** member function for the **bank** object.
- Call the **process_transactions()** member function for the **bank** object. Pass the file name "**transactions.txt**" as an argument to the function.
- Call the **print()** member function for the **bank** object.

Sample Output

Account Listing for First National Bank

Account Number: 1132264809
Name: Joanna Madsen
Balance: $2805.65

Account Number: 5540853032
Name: Trey Donner
Balance: $4850.75

Account Number: 5745648360
Name: Ronald Jones
Balance: $1340.53

Sample Output (2)

Account Number: 5745734564
Name: Karin Hunt
Balance: $4476.00

Account Number: 6379094723
Name: Blake Reynolds
Balance: $2703.62

Account Number: 7307830409
Name: Jon Mitchell
Balance: $207.45

Account Number: 7415949234
Name: Susan Garcia
Balance: $3738.64

Sample Output (3)

Account Number: 9858542030
Name: Keiko Tanaka
Balance: $11343.82

Transaction Report

Date	Account	Type	Amount	New Balance
08/19	1130034922	D	5500.00	*** Invalid account number ***
08/19	5540853032	W	430.00	4420.75
08/20	7415949234	D	3620.45	7359.09
08/20	9858542030	W	130.00	11213.82
08/20	1132264809	W	3275.23	*** Insufficient funds ***
08/20	6379094723	W	250.00	2453.62

Sample Output (4)

Account Listing for First National Bank

Account Number: 1132264809
Name: Joanna Madsen
Balance: $2805.65

Account Number: 5540853032
Name: Trey Donner
Balance: $4420.75

Account Number: 5745648360
Name: Ronald Jones
Balance: $1340.53

Account Number: 5745734564
Name: Karin Hunt
Balance: $4476.00

Sample Output (5)

Account Number: 6379094723
Name: Blake Reynolds
Balance: $2453.62

Account Number: 7307830409
Name: Jon Mitchell
Balance: $207.45

Account Number: 7415949234
Name: Susan Garcia
Balance: $7359.09

Account Number: 9858542030
Name: Keiko Tanaka
Balance: $11213.82