Name: Johanna (Thanh) Vo
Group team: Chris Jensen, Johanna Vo, Minseong Na, Will Bach

Program 5: Project Implementation

Overview UML diagram

This program simulates a bank that will read a string of transactions from a file (BankTransIn.txt) into an in-memory queue. In the transaction class, it will handle the functions that include opening accounts, depositing funds, transferring funds, withdrawing funds, and printing the transactional history of funds.There are two main class: BankAccount ans Banker.

BankAccount present a client's account. The constructors is for the account information(firstName, lastName, accountID) and include specific fund type mane. BankAccount class use fund class to manipulate a bank user account. Funds class is used for manipulating the different types of funds.Banker class use BSTree and Transaction class to process each new account in account list and record queue transaction. After reading transactions, we have processTransactions() in Banker Class that processes all transactions in order from the queue. Then, printOutBalance() (in Banker Class) prints out all open accounts and balances in those accounts.

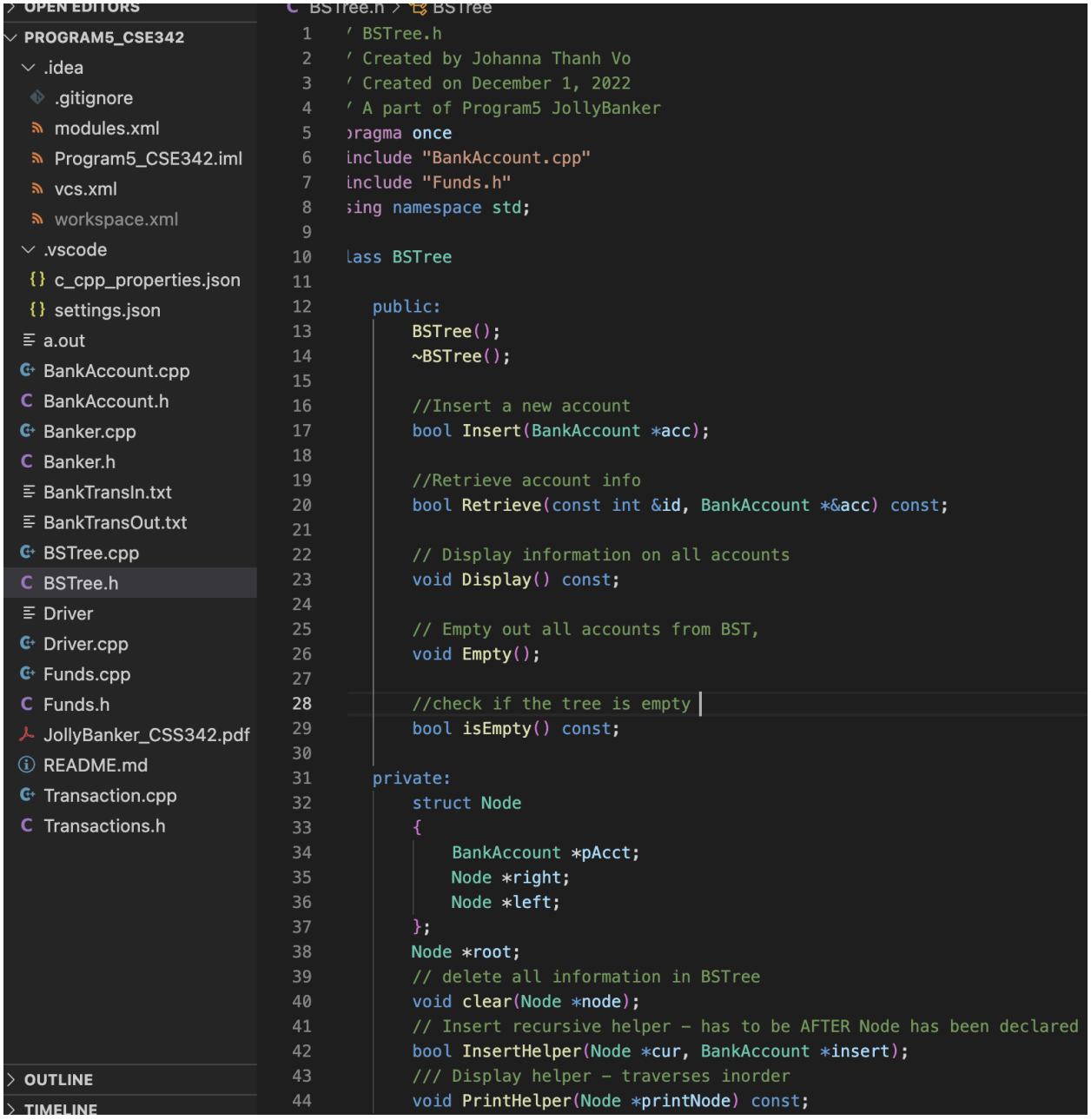
When we run Drive.cpp, All record transaction and balances account will print out in BankTransOut.txt

---

My part in the project to design and implement the BSTtree. As we know, BSTree will store account objects that are created in the Banker class. There are some implemented functions that can add new accounts, retrieve account info, display accounts, empty out all accounts from BST, and check if the tree is empty.

Moreover, some recursive functions contain clear() to help the destructor so that every node is deleted, InsertHelper() is a recursive insert version, and PrintHelper() is to display by traversing the tree from root to leaf to print out account info.

BSTree.h


OPEN EDITORS
PROGRAM5_CSE342
.idea
.gitignore
modules.xml
Program5_CSE342.iml
vcs.xml
workspace.xml
.vscode
c_cpp_properties.json
settings.json
a.out
BankAccount.cpp
BankAccount.h
Banker.cpp
Banker.h
BankTransIn.txt
BankTransOut.txt
BSTree.cpp
BSTree.h
Driver
Driver.cpp
Funds.cpp
Funds.h
JollyBanker_CSS342.pdf
README.md
Transaction.cpp
Transactions.h
OUTLINE
TIMELINE
BSTree.h > BSTree
1 ' BSTree.h
2 ' Created by Johanna Thanh Vo
3 ' Created on December 1, 2022
4 ' A part of Program5 JollyBanker
5 pragma once
6 include "BankAccount.cpp"
7 include "Funds.h"
8 sing namespace std;
9
10 lass BSTree
11
12 public:
13 BSTree();
14 ~BSTree();
15
16 //Insert a new account
17 bool Insert(BankAccount *acc);
18
19 //Retrieve account info
20 bool Retrieve(const int &id, BankAccount *&acc) const;
21
22 // Display information on all accounts
23 void Display() const;
24
25 // Empty out all accounts from BST,
26 void Empty();
27
28 //check if the tree is empty
29 bool isEmpty() const;
30
31 private:
32 struct Node
33 {
34 BankAccount *pAcct;
35 Node *right;
36 Node *left;
37 };
38 Node *root;
39 // delete all information in BSTree
40 void clear(Node *node);
41 // Insert recursive helper - has to be AFTER Node has been declared
42 bool InsertHelper(Node *cur, BankAccount *insert);
43 /// Display helper - traverses inorder
44 void PrintHelper(Node *printNode) const;

Implementation BSTree.cpp

```cpp
// BSTree.cpp
// Created by Johanna Thanh Vo
// Created on December 6, 2022
// A part of Program5 JollyBanker
#include "BSTree.h"
#include <fstream>

// constructor sets root to null
BSTree::BSTree() { root = nullptr; }

//Destructor so that every node is deleted
BSTree::~BSTree() { clear(root); }

```

Firs of all, I have include BSTree.h interface and fstream to operate on a output file (BAnkTransOut.txt)

Line 9, I have the default constructor to set root to null pointer.
Line 10, a Destructor call clear(root) function to delete every node.

```cpp
// delete all information in BSTree
// using recursively loop through the tree to delete every node
void BSTree::clear(Node *node)
{
    if (node != nullptr)
    {
        clear(node->left);
        clear(node->right);
        // delete accounts and nodes
        delete node->pAcct;
        delete node;
    }
}
```

```
14  // Insert a new account
15  bool BSTree::Insert(BankAccount *accInsert)
16  {
17      int insertAccountID = accInsert->getID();
18      //check valid ID Number
19      if (insertAccountID < 1000 || insertAccountID > 9999)
20      {
21          cerr << "ERROR: Account ID Number "<< insertAccountID
22          <<" is not Valid" << endl;
23          return false;
24      }
25
26      // Base case or if empty
27      if (root == nullptr)
28      {
29          root = new Node;
30          root->pAcct = accInsert;
31          return true;
32      }
33      else
34      {
35          Node *current = root;
36          // If the node has two or more account, run recursive helper
37          InsertHelper(current, accInsert);
38      }
39      return false;
40  }
41
```

Next, Insert(BankAcoount *accInsert) to add new accounts into tree.
Line 17, create variable insertAccountID assign accountID of new insert account
Line 19-24, Checking if insertAccountID is valid number with 4 digit. Print out error if not
Line 27-32, set a base case if empty, then set accInsert is root node
Line 34-38, we will make a key node then run recursive helper to insert the right location and follow by the rule subtree node.

When looking for a place to insert a new account, we will traverse the tree from root-to-leaf, making comparison to accountID was stored in tree and deciding based on the comparison to continue searching in the left or right subtrees.

Line 128-143, examine the current node accountID and recursively insert the new accountID to the right subtree if its accountID is greater.

```
125 bool BSTree::InsertHelper(Node *current, BankAccount *insert)
126 {
127     // If newAccount < current node then start going down left side
128     if (insert->getID() < current->pAcct->getID())
129     {
130         if (current->left == NULL)
131         {
132             Node *insInTree = new Node();
133             insInTree->pAcct = insert;
134             insInTree->left = NULL;
135             insInTree->right = NULL;
136             current->left = insInTree;
137             return true;
138         }
139         else
140         {
141             return InsertHelper(current->left, insert);
142         }
143     }
144     // Else if newAccount > node then start going down right side
145     else if (insert->getID() > current->pAcct->getID())
146     {
147         if (current->right == nullptr)
148         {
149             Node *insInTree = new Node();
150             insInTree->pAcct = insert;
151             insInTree->left = nullptr;
152             insInTree->right = nullptr;
153             current->right = insInTree;
154             return true;
155         }
156         else
157         {
158             return InsertHelper(current->right, insert);
159         }
160     }
161     else // Displays error if account is already in the BST
162     {
163         ofstream outfile;
164         outfile.open("BankTransOut.txt", std::ios_base::app);
165         outfile << "ERROR: Account " << insert->getID() << " is already open.
166         outfile.close();
167         return false;
168     }
169 }
```

Line 145-160, examine the current node accountID and recursively insert the new accountID to the right subtree if its accountID is greater.
Line 161-168 Displays an error into the output file (BankTransOut.txt) if the account is already in the BST

```
41
42  bool BSTree::Retrieve(const int &id, BankAccount *&acc) const
43  {
44      Node *current = root;
45      bool search = false;
46
47      while (!search)
48      {
49          if (current != nullptr && id == current->pAcct->getID())
50          {
51              search = true;
52              acc = current->pAcct;
53              return search;
54          }
55          else if (current != nullptr && id > current->pAcct->getID())
56          {
57              current = current->right;
58          }
59          else if (current != nullptr && id < current->pAcct->getID())
60          {
61              current = current->left;
62          }
63          else
64          {
65              search = true;
66          }
67      }
68      ofstream outfile;
69      outfile.open("BankTransOut.txt", std::ios_base::app);
70      outfile << "ERROR: Account " << id << " not Found. Transaction refused" <<
71      outfile.close();
72      return false;
73  }
```

Next, Retrieve(const int &id, BankAccount *&acc) will be true if it found the account in BST by using while loop throughout (line 47-67)searching matching accountID and key current node; and false will be display the error in to output file (BankTransOut.txt).

Next, line 76-80, Empty() function to empty out all accounts from BST by deleting the root and make null pointer

Clearly, in lines 82-92, check if the tree is empty when both side subtrees equal null pointer.

```
76  void BSTree::Empty()
77  {
78      delete root;
79      root = nullptr;
80  }
81
82  bool BSTree::isEmpty() const
83  {
84      if (root->left == nullptr && root->right == nullptr)
85      {
86          return true;
87      }
88      else
89      {
90          return false;
91      }
92  }
93
94  void BSTree::Display() const
95  {
96      if (root == nullptr)
97      {
98          cerr << "ERROR: ACCOUNT LIST IS EMPTY" << endl;
99      }
100     PrintHelper(root);
101 }
```

Line 94-101, Display() function to print information on all account objects. Print cerr to terminal if account list empty when root is null. Else, run recursive helper to print in order traversal on the tree

Line 104-122, when the current node is not NULL, open the output file tp print all account information with balances of ten types of funds. This recursive stop until current equal NULL

```
102
103  void BSTree::PrintHelper(Node *current) const{
104      if (current != NULL) {
105          ofstream outfile;
106          outfile.open("BankTransOut.txt", std::ios_base::app);
107          outfile << current->pAcct->getLastName() << " ";
108          outfile << current->pAcct->getFirstName();
109          outfile << " Account ID: ";
110          outfile << current->pAcct->getID() << endl;
111
112          for (int i = 0; i < 10; i++) {
113              outfile << "    " << current->pAcct->getSubAccName(i)
114                      << ": $" << current->pAcct->getSubAccBalance(i)
115                      << endl;
116          }
117          outfile << endl;
118          PrintHelper(current->right);
119          PrintHelper(current->left);
120          outfile << endl;
121          outfile.close();
122      }
123  }
```