BANK CLASS

```
#include <queue>
#include <fstream>
#include "bstree.h"
#include "transaction.h"

using namespace std;

class Bank {
public:
    Bank();
    ~Bank();

    bool ReadTransactions(const string &filename);
    void ProcessTransactions();
    void DisplayTransactions();

private:
    queue<Transaction> transactions_;
    BSTree accounts_;
};
```

BSTREE CLASS

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

using namespace std;

class BSTree {
public:
    BSTree();
    BSTree(const BSTree &tree);
    ~BSTree();

    Node* root();

    bool Insert(Account account);
    bool Retrieve(const int &account_id, Account*
&account);
    bool Delete(const int &account_id, Account*
&account);

    BSTree& operator=(const BSTree &tree);

    void Display();
    int Size();

private:
    struct Node {
        Account* account_;
        Node* left_;
        Node* right_;
    };
    Node* root_;
};
```

ACCOUNT CLASS

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

using namespace std;

class Account {
public:
    Account(int id, const string &first_name, const
string &last_name);

    int id() const;
    string first_name() const;
    string last_name() const;
    int funds_balance(int fund_id) const;
    vector<int> funds_balance() const;\
    string fund_name(int fund_id) const;
    vector<string> fund_names() const;
    vector<Transaction> transactions() const;

    bool Deposit(int fund_id, int amount);
    bool Withdraw(int fund_id, int amount);
    bool Transfer(int source_fund_id, int dest_fund_id,
int amount);
    bool TransferAccounts(int source_fund_id, Account*
&dest_account, int dest_fund_id, int amount);

    void AddTransaction(const Transaction &transaction);
    void DisplayTransactions();
    void DisplayFundTransactions(int fund_id);


private:
    int id_ = -1;
    string first_name_ = "first";
    string last_name_ = "last";
    vector<int> funds_balance_ = {0, 0, 0, 0, 0, 0, 0, 0,
0, 0};
    vector<string> fund_names_ = {
        "Money Market",
        "Prime Money Market",
        "Long-Term Bond",
        "Short-Term Bond",
        "S+P Index Fund",
        "Value Fund",
        "Growth Equity Fund",
        "Growth Index Fund",
        "Crypto ETF",
        "Precious Metals ETF"
    };
    vector<Transaction> transactions_;
};
```

TRANSACTION CLASS

```
#include <iostream>
#include <string>

using namespace std;

class Transaction {
public:
    // Open
    Transaction(char type, int account_id, int fund,
string last_name, string first_name);
    // Deposit or Withdraw
    Transaction(char type, int account_id, int fund, int
amount);
    // Transfer
    Transaction(char type, int source_id, int
source_fund, int dest_id, int dest_fund, int amount);
    // Display History for Client
    Transaction(char type, int account_id);
    // Display History for Fund
    Transaction(char type, int account_id, int fund);
    // Destructor
    ~Transaction();

    // Getters
    char type() const;
    int account_id() const;
    int fund_id() const;
    int amount() const;
    int source_fund() const;
    int dest_fund() const;
    string last_name() const;
    string first_name() const;

    // Output Stream Overload
    friend ostream& operator<<(ostream &os, const
Transaction &transaction);

private:
    char type_ = " ";
    int account_id_ = -1;
    int fund_id_ = -1;
    int amount_ = -1;
    int source_fund_ = -1;
    int dest_fund_ = -1;

    string last_name_ = "last";
    string first_name_ = "first";
};
```

As given by the 4 classes given above, they work together to take in file input and output first the bank class will have a ReadTransactions which will read into the file and create a bunch of Transactions Objects and puts it into a queue. ProcessTransactions will then see the Transaction Objects values and navigate to the corresponding account, if an account exists, or creates a new account if it tells it to create it. These accounts are created and then put into a BSTree Data Structure which will organize the accounts based on account_id. Each account has different methods based on what Transactions needs to run, such as Deposit, Withdraw, and Transfer. But Transfer has 2 methods, one for 2 separate accounts and one for a single account between funds. If money is being withdrawn or transferred for either Money Market fund, and the balance seems to be less than the given amount, it will check the other Money Market fund, to see if it can withdraw some from there and add it to the destination account. Display Transactions works by going into each account, from least to greatest id number, and displaying each value from the first Transaction to last Transaction in an account. These .h files are by no means completed, but are close enough to show my progress on my design for Program 5 so far.