In this project we manage the account with banking and stocking options. In this document, we suggest a few different ways to do this. There is very few requirements for the program structures. I will attempt to clarify the requirements of this project.

Main Requirements
Using input/output file stream class, information regarding all transactions will be saved when running the program in two files: bank_transaction_history.txt and stock_transaction_history.txt

Using inheritance - we will use inheritance to make base class and derived classes.

Randomize the stock information files - for every function that requires the *current* price of the stock, you have to read from the one of the 4 files provided. You will select the file at random.

Keep a vector in the `main.cpp` file for storing all the current stocks that you own with the amount of shares.

Suggestions
In this section, I will suggest a way for the program structures

1. Linking the `cashBalance` by private variable
   You can have the private variable `cashBalance` at the base class `account`. In this structure, you can derive it into two further substructure. The first one is to have the `bankAccount` class and `stockAccount` class as the derived class of `account` class. When working in this structure, you have to update the `cashBalance` of all objects in order to have them matching each other.

Streaming Text Files
All files are now in UTF-8 format. You can read the string from the file and convert it to double.

1. **Get random text file -** we can make the name of the text file to be read as the following

```
srand(time(NULL));
int num = rand()%4 + 1;
string name = "stock" + to_string(num) + ".txt";
ifstream file;
file.open(name);
// more code here
file.close();
```

2. **Instream / outstream text file** - we can use the fstream class for this purpose. For example, the following code snippets reading from the text file `example1.txt` and saving a string to `example2.txt`

Reading file line by line

```
ifstream file("example1.txt");
string  line;
while (getline(file, line))
     cout  << line << endl;
file.close();
```

Saving line to file

```
string line = "this is a test line\n";
ofstream myfile;
myfile.open("example2.txt", ios_base::app);
myfile << line;
myfile.close();
```

3. **Convert string to double -** for example, convert the string from `cashBalance.txt` to double variable

```
ifstream file("cashBalance.txt");
string line;
file >> line;
file.close();

double balance = stod(line);
```

4. **Parsing strings -** when reading the string from the stock file, we would need to parse out the smaller strings that contain the information that we need. There are a few things we need to pay attention to. Each of the smaller strings (symbol, company name, price) in the stock file are separated by the tab (`"\t"`). We can use this as the delimiter. Each line we read from the stock file is ended with the `newline` character (`"\n"`).

Parsing substring

```
string line = "VNET   21Vianet Group Inc. ADR    5.55";
string delim = "\t".
vector <string> tokens;
for (int i = 0; i < 3; i++) // making 3 tokens
     if (i == 2) { // handling newline character
          line  = line.substr(line.rfind(delim)+1,
          string::npos);
          line .resize(line.length()-1); // remove newline

          tokens.push_back(line);
     }

     tokens.push_back(line.substr(0, line.find(delim)));
     line = line.substr(line.find(delim)+1, string:npos);
}
```

Quick explaination: we use the tab (`"\t"`) as the delimiter. We search the string until we see the first delimiter. We then make substring from position 0 to the position of the first delimiter. Remember the index starts from zero but position is not. Therefore from 0 to the position of the first delimiter will contain all of the first token, for example, stock symbol. We the save this token into the vector of strings. The same operation can be done for the second token.

For the last token, the difference in the format is that it preceded by multiple tab (`"\t"`) and ends with the newline and NULL character. We use the `rfind()` to find the last

occurrence of the tab character (`"\t"`). The substring is made from the last occurrence of tab (`"\t"`) plus 1, to the end of the string (`string::npos`). This includes the newline and NULL characters. We get rid off of these by `resize` the string. Now we can append more token after the last token if we need to.

Remember to put the newline character (`"\n"`) at the end of the string when saving it to the file. Otherwise all the line will be continuously written at the end of the previous line.

The remaining of the project is very similar to project 1. I think you can finish it without a hitch from here on.

Good luck