Homework 1

Time due: 11:59 PM Friday, May 3rd

1. Imagine you are working for a small hedge fund, and your mission is to catch inside traders. You have a new pattern that you think might indicated to you that something suspicious like inside trading may be occurring: The profit made, or loss prevented, by an INIDIVIDUAL trade due to changes in the price of that stock within a 3 day window after the trade was made is greater than or equal to $500,000. You want to build an algorithm that is capable of taking the hedge fund's data as input and generating a list of occurrences of this pattern.

   For a given stock ticker, you have a pipe-delimited datasource sorted by day with two primary pieces of information: the price of said stock on each day and the trades of that stock made by each trader at the hedge fund over time. More specifically, you receive this data as a String Array where each element in the array is one of:

   - Stock Price - the value of the stock on a given day
     - These elements are simply <DAY>|<PRICE>, where DAY is the integer number of days since the fund has opened and price is integer dollar amount value of the stock on that day.
     - Price is only recorded when it has changed from the day before, and will always have an entry for day 0
   - Trade - the trade made by a specific trader for a stock on a given day
     - These elements are of the format <DAY>|<TRADER>|<TRADE_TYPE>|<AMOUNT>, where trader TRADER either bought or sold depending on the value of TRADE_TYPE and integer AMOUNT number of that stock on the specified day.
     - TRADE_TYPE will either be "BUY" or "SELL"

   Your output is another String Array that can be fed into other systems at the hedge fund and is simply an entry of the format <DAY>|<TRADER> where DAY is the day said TRADER made the trade that triggered the alert, not the day that the stock price changed. This output should be sorted by day and then trader name, and should not include duplicates.

   Example Explanation

   In this example, "1|Tom" is considered a suspicious trade because:

   - On day 0 the stock price gets set at $20 - "0|20"
   - On day 1 Tom buys 150,000 stock "1|Tom|BUY|150000"
   - On day 3, 2 days later (i.e. whithin 3 days), the stock price increases from $20 to $25 - "3|25"
     - This represents a 25 - 20 = $5 difference in stock price
     - Tom's trade on day 1 therefore earned him a profite of $5 * 150,000 = $750,000, which is over our threshold

   Also "8|Kristi" is a suspicious trader because:

   - On day 3 the stock price gets set at $25 - "3|25"
   - On day 8 Kristi sells 60,000 stock "8|Kristi|SELL|60000"
   - On day 10, 2 days later (i.e. whithin 3 days), the stock price decreases from $25 to $15 - "10|15"
     - This represents a 15 - 25 = $-10 difference in stock price
     - Kristi's trade on day 8 therefore prevented a loss of $10 * 60,000 = $600,000, which is over our threshold
   - Note that on day 11 (also within 3 days) the stock price decreases again from $15 to $5 = "11|5"
     - Compared to day 8, this represents a 5 - 25 = -$20 difference in stock price since Kristi's purchase, which amounts to preventing a loss of $1.2 million
     - However, Kristi's trade on day 8 was already flagged, so we should not add it to our results set because we do not want duplicates

Similarly, Will is not considered to have a suspicious trade:

- The price of stock on day 14 is $5 - "11|5"
- He buys 30 thousand total stock between days 14 and 16, inclusive - ["14|Will|BUY|10000", "15|Will|BUY|10000", "16|Will|BUY|10000"]
- The price of the stock on day 17 increases to $25
  - This represents a $20 difference
  - Will has profited $20 * 30,000 = $600,000 TOTAL
  - The amount of profit earned by any one of his 3 individual trades is only $20 * 10,000 = $200,000
  - $200,000 is less than the threshold amount of $500,000 for any INDIVIDUAL trade and thus is not flagged.

Here is a main file to get you started.

2. This will give you some experience using a few STL containers and iterators
   a. Implement the `removeOdds` function:

```
#include <list>
#include <vector>
#include <algorithm>
#include <iostream>
#include <cassert>
using namespace std;

  // Remove the odd integers from li.
  // It is acceptable if the order of the remaining even integers is not
  // the same as in the original list.
void removeOdds(list<int>& li)
{
}
```

```
void test1()
{
    int a[8] = { 2, 8, 5, 6, 7, 3, 4, 1 };
    list<int> x(a, a+8);  // construct x from the array
    assert(x.size() == 8 && x.front() == 2 && x.back() == 1);
    removeOdds(x);
    assert(x.size() == 4);
    vector<int> v(x.begin(), x.end());  // construct v from x
    sort(v.begin(), v.end());
    int expect[4] = { 2, 4, 6, 8 };
    for (int k = 0; k < 4; k++)
        assert(v[k] == expect[k]);
}

int main()
{
    test1();
    cout << "Passed" << endl;
}
```

   b. Implement the `removeOdds` function:

```
#include <vector>
#include <algorithm>
#include <iostream>
#include <cassert>
using namespace std;

  // Remove the odd integers from v.
  // It is acceptable if the order of the remaining even integers is not
  // the same as in the original vector.
void removeOdds(vector<int>& v)
{
}
```

```
void test2()
{
    int a[8] = { 2, 8, 5, 6, 7, 3, 4, 1 };
    vector<int> x(a, a+8);  // construct x from the array
    assert(x.size() == 8 && x.front() == 2 && x.back() == 1);
    removeOdds(x);
    assert(x.size() == 4);
    sort(x.begin(), x.end());
    int expect[4] = { 2, 4, 6, 8 };
    for (int k = 0; k < 4; k++)
        assert(x[k] == expect[k]);
}

int main()
{
    test2();
    cout << "Passed" << endl;
}
```

c. Implement the `removeBad` function:

```
#include <list>
#include <vector>
#include <algorithm>
#include <iostream>
#include <cassert>
using namespace std;

vector<int> destroyedOnes3;

class Movie3
{
  public:
    Movie3(int r) : m_rating(r) {}
    ~Movie3() { destroyedOnes3.push_back(m_rating); }
    int rating() const { return m_rating; }
  private:
    int m_rating;
};

  // Remove the movies in li with a rating below 50 and destroy them.
  // It is acceptable if the order of the remaining movies is not
  // the same as in the original list.
void removeBad(list<Movie3*>& li)
{
}

void test3()
{
    int a[8] = { 85, 80, 30, 70, 20, 15, 90, 10 };
    list<Movie3*> x;
    for (int k = 0; k < 8; k++)
        x.push_back(new Movie3(a[k]));
    assert(x.size() == 8 && x.front()->rating() == 85 && x.back()->rating() == 10);
    removeBad(x);
    assert(x.size() == 4 && destroyedOnes3.size() == 4);
    vector<int> v;
    for (list<Movie3*>::iterator p = x.begin(); p != x.end(); p++)
    {
        Movie3* mp = *p;
        v.push_back(mp->rating());
    }
    sort(v.begin(), v.end());
    int expect[4] = { 70, 80, 85, 90 };
    for (int k = 0; k < 4; k++)
        assert(v[k] == expect[k]);
```

```
    sort(destroyedOnes3.begin(), destroyedOnes3.end());
    int expectGone[4] = { 10, 15, 20, 30 };
    for (int k = 0; k < 4; k++)
        assert(destroyedOnes3[k] == expectGone[k]);
}

int main()
{
    test3();
    cout << "Passed" << endl;
}
```

d. Implement the `removeBad` function:

```
#include <vector>
#include <algorithm>
#include <iostream>
#include <cassert>
using namespace std;

vector<int> destroyedOnes4;

class Movie
{
  public:
    Movie(int r) : m_rating(r) {}
    ~Movie() { destroyedOnes4.push_back(m_rating); }
    int rating() const { return m_rating; }
  private:
    int m_rating;
};

  // Remove the movies in v with a rating below 50 and destroy them.
  // It is acceptable if the order of the remaining movies is not
  // the same as in the original vector.
void removeBad(vector<Movie*>& v)
{
}

void test4()
{
    int a[8] = { 85, 80, 30, 70, 20, 15, 90, 10 };
    vector<Movie*> x;
    for (int k = 0; k < 8; k++)
        x.push_back(new Movie(a[k]));
    assert(x.size() == 8 && x.front()->rating() == 85 && x.back()->rating() == 10);
    removeBad(x);
    assert(x.size() == 4 && destroyedOnes4.size() == 4);
    vector<int> v;
    for (int k = 0; k < 4; k++)
        v.push_back(x[k]->rating());
    sort(v.begin(), v.end());
    int expect[4] = { 70, 80, 85, 90 };
    for (int k = 0; k < 4; k++)
        assert(v[k] == expect[k]);
    sort(destroyedOnes4.begin(), destroyedOnes4.end());
    int expectGone[4] = { 10, 15, 20, 30 };
    for (int k = 0; k < 4; k++)
        assert(destroyedOnes4[k] == expectGone[k]);
}

int main()
{
    test4();
    cout << "Passed" << endl;
}
```

3. Some word games, like Scrabble, require rearranging a combination of letters to make a word. This type of arrangement is generally referred to as an anagram, it's known as a permutation in mathematics.Write a C++ program that searches for ``anagrams'' in a dictionary. An anagram is a word obtained by scrambling the letters of some string. For example, the word ``pot'' is an anagram of the string `"otp." A sample run of the program is given below. Your output does not have to be formatted exactly the same as that shown in the sample, but should be in a similar style. You can use words.txt as your dictionary file. For this solution you **must not use recursion**. Instead look up and use the **`std::next_permutation`** function in the algorithm library.

Sample Runs

Here are two examples of how the program might work:

```
Please enter a string for an anagram: opt
Matching word opt
Matching word pot
Matching word top

Please enter a string for an anagram: blah
No matches found
```

Requirements

You must write these two functions with the exact same function signature (include case):

int loadDictionary(istream &dictfile, vector<string>& dict);

Places each string in `dictfile` into the vector `dict`. Returns the number of words loaded into `dict`.

int permute(string word, vector<string>& dict, vector<string>& results);

Places all the permutations of `word`, which are found in `dict` into `results`. Returns the number of matched words found.

For words with double letters you may find that *different* permutations match the same word in the dictionary. For example, if you find all the permutations of the string *kloo* using the algorithm we've discussed you may find that the word *look* is found twice. The *o*'s in *kloo* take turns in front. Your program should ensure that matches are unique, in other words, the `results` array returned from the `permute` function should have no dupicates.

Turn it in

What you will turn in for this assignment is a zip file containing these files:

1. A zip file which contains files named:
   a. **trader.cpp** (question 1)
   b. **containers.cpp** (question 2, this should have all the code excluding the main functions)
   c. **permutations.cpp** (question 3)