Protokoll Übung 1 – Hashtabelle

Beschreibung der Datenstruktur und Hashfunktion:

Das Programm basiert grundsätzlich auf der Klasse StockManager. Diese ist dafür verantwortlich die Aktien (Stocks) zu verwalten und den Zugriff auf die Aktiendaten zu gewährleisten.

Diese Klasse besteht aus :

- einer Konstanten MAX_STOCKS, die die Anzahl der Aktien limitiert (private)
- dem int numStock, welches angibt wie viele Aktien aktuell im Stockmanager gespeichert werden (private)
- zwei Arrays den Hashtabellen für Name und Symbol, um die Suche zu erleichtern (private)
- sowie den 9 Funktionen: customHash, addStock, deleteStock, importStockData, searchStock, plotStock, saveToFile, loadFromFile (public)

```
class StockManager{
public:
    //Hashfunktion
    int customHash(const string& word);
    void addStock(const string& name, const string& wkn, const string&
symbol);
    void deleteStock();
    void importStockData();
    void searchStock();
    void plotStock();
    void saveToFile();
    void loadFromFile();
private:
    //Es d�rfen Maximal 1000 Stocks in der Hashtabelle sein
    static const int MAX_STOCKS = 1000;
    //Hashtabelle f�r symbol
    Stock stocks[MAX_STOCKS];
    //zweite Hashtabelle f�r suche nach namen
    Stock stocksbyname[MAX_STOCKS];
    int numStocks = 0;
};
```

Außerdem gibt es zwei Structs um die Struktur der Aktien besser darstellen und abspeichern zu können:

- Stocks, die einzelne Aktie, wo Name, WKN und Symbol gespeichert werden, sowie die Historie in Form eines Vektors aus StockData
- StockData, wo werden die Details der historischen Daten abgespeichert

```
struct StockData {
    string date;
    double open;
    double high;
    double low;
    double close;
    double volume;
    double adjClose;
};

//Struktur f�r Stock Files
struct Stock {
    string name;
    string wkn;
    string symbol;
    //Speicherung der Daten
    //Vektor l�sst sich besser erweitern als array
    vector<StockData> history;
};
```

In der Main wird der StockManager erstellt und dann nach User eingaben die einzelnen Funktionen des Mangers aufgerufen.

Die Hashfunktion wird von den Funktionen dann selbst aufgerufen, um einen Index berechnen zu lassen. Dabei wird in den Funktionen selbst auch darauf geachtet, wie sich der Index bei einer Kollision verschieben kann.

Die Funktion selbst funktioniert folgendermaßen: Aus einem mitgegebenen String wird durch die Hashfunktion ein Index zwischen 0 und 999 generiert. Dabei werden die ASCII-Werte der Zeichen summiert und jeweils mit ihrer Position multipliziert um eine bessere Verteilung zu ermöglichen (selbe Buchstaben, aber andere Reihenfolge != selber Index). Am Ende wird der resultierende Wert noch durch die Maximale Anzahl an Stocks moduliert, um sicherzustellen, dass der Index im gültigen Bereich liegt.

```
int total = 0;
        int round = 1;
        for (char c : word) {
            // Das ASCII value von den stellen addieren
            // jede stelle mit ihrer position mulitplizieren, um Reihenfolge
wichtig zu machen
            total += static_cast<int>(c)*round;
            round ++;
        }
        // Modulo der Maximalen Gr��e, damit alle Werte im Bereich zwischen 0
und 999 liegen
        int hashValue = (total % MAX_STOCKS);
        return hashValue;
```

Aufwandsabschätzung:

- **customHash**: Die Laufzeit dieser Funktion beträgt **O(n)**, wobei n die Länge des Eingabestrings ist. Da die Funktion jedes Zeichen im String einmal durchläuft, ist die Laufzeit linear zur Länge des Strings.
- **addStock**: Die Laufzeit beträgt im schlimmsten Fall **O(m)**, wobei m die maximale Anzahl von Aktien (MAX_STOCKS) ist. Dies liegt daran, dass die Funktion im schlimmsten Fall alle Plätze in der Hashtabelle durchlaufen muss, um einen freien Platz zu finden.
- **deleteStock**: Die Laufzeit beträgt im schlimmsten Fall ebenfalls **O(m)**, da sie im schlimmsten Fall alle Plätze in der Hashtabelle durchläuft, um die zu löschende Aktie zu finden.
- **importStockData**: Die Laufzeit beträgt **O(m + n)**, wobei m die maximale Anzahl von Aktien (MAX_STOCKS) und n die Anzahl der Zeilen in der Datei ist. Dies liegt daran, dass die Funktion im schlimmsten Fall alle Plätze in der Hashtabelle durchlaufen muss, um die zu importierende Aktie zu finden (O(m)), und dann jede Zeile in der Datei einmal durchläuft, um die Daten zu importieren (O(n)).
- **searchStock**: Im schlimmsten Fall beträgt die Laufzeit **O(2m)**, da die Funktion zuerst in der Hashtabelle nach dem Namen und dann nach dem Symbol sucht. Jede Suche könnte im schlimmsten Fall alle Plätze in der Hashtabelle durchlaufen müssen, um die gesuchte Aktie zu finden.
- **plotStock**: Die Laufzeit beträgt **O(m + n),** wobei m die maximale Anzahl von Aktien (MAX_STOCKS) und n die Anzahl der Datenpunkte in der Historie der Aktie ist. Das liegt daran, dass die Funktion im schlimmsten Fall alle Plätze in der Hashtabelle durchlaufen muss, um die zu plottende Aktie zu finden (O(m)), und dann jeden Datenpunkt in der Historie der Aktie einmal durchläuft, um den Plot zu erstellen (O(n)).
- **saveToFile**: Die Laufzeit beträgt **O(m * n)**, wobei m die maximale Anzahl von Aktien (MAX_STOCKS) und n die durchschnittliche Anzahl von Datenpunkten in der Historie jeder Aktie ist. Die Funktion durchläuft jeden Platz in der Hashtabelle (O(m)) und für jede nicht leere oder gelöschte Aktie durchläuft sie jeden Datenpunkt in ihrer Historie, um die Daten in die Datei zu schreiben (O(n)).
- **loadFromFile**: Die Laufzeit beträgt ebenfalls **O(m * n)**, wobei m die maximale Anzahl von Aktien (MAX_STOCKS) und n die Anzahl der Zeilen in der Datei ist. Die Funktion durchläuft jede Zeile in der Datei einmal (O(n)) und für jede Zeile, die eine Aktie repräsentiert, durchläuft sie im schlimmsten Fall alle Plätze in der Hashtabelle, um einen freien Platz für die Aktie zu finden (O(m)).