ÜBUNG 1 – HASHTABELLE

Ausgewählte Programmiersprache(n): C++, Python (Piton:D)

❖ Python:

Plot.py:

Hier werden Daten aus einer CSV-Datei gelesen und in ein Pandas-Dataframe geladen. Das Datum wird in ein Datetime-Objekt konvertiert und der durchschnittliche Aktienpreis pro Tag wird berechnet. Die letzten 30 Tage der Daten werden ausgewählt und in einem Diagramm dargestellt. Schließlich wird die CSV-Datei gelöscht.

❖ Main

main.cpp → Die Klasse App wird inkludiert & aufgerufen

```cpp
#include "App.h"

int main() {
    App app;
    app.run();
    return 0;
}
```

❖ Die Klassen

app.h →Die App-Klasse:

```cpp
class App {
public:
    void run();
private:
    StockHashTable stockHashTable;
};
```

➔ Es wurde eine Methode mit dem Namen "run()" und eine Instanzvariable mit dem Namen "stockHashtable" deklariert.

➔ In der „App.cpp“ wird ein Menu ausgegeben, das je nach Auswahl des Benutzers zu 8 verschiedene Fällen führt. In einigen Fällen werden verschiedene Methoden & Klassen aufgerufen. Dabei werden auch Meldung wie zB: „Daten wurden hinzugefügt/gelöscht/importiert“ angezeigt.

➔ Die verfügbaren Auswahlmöglichkeiten sind: 1) Add, 2) Remove, 3) Import, 4) Search, 5)Plot, 6) Save, 7) Load, 8) Quit.

UI.h → Die UI-Klasse:

```
#include "stock.h"

class UI {
public:
    static void printMenu();
    static int scanChoice();
    static std::shared_ptr<Stock> scanStock();
    static std::string scanFilePath();
    static std::pair<std::string,bool> scanKeyword();
};
```

In der UI-Klasse wurden fünf Methoden deklariert & definiert. Wie der Name bereits andeutet, ist die UI Klasse dafür zuständig, Texte oder Meldungen auszugeben sowie Benutzereingaben zu lesen. Hierzu gehört beispielsweise das Anzeigen von Menü, das Einlesen von Benutzereingaben. Alle Interaktionen zwischen Benutzer und Programm werden hier implementiert.

stock.h→ Die Stock-Klasse:

Hier wurden drei Klassen deklariert: "PriceData", "Stock" und "StockEntry".

Die "PriceData"-Klasse dient zur Verwaltung von Preisdaten und enthält Methoden zum Setzen und Abrufen von Preisdaten sowie zum Ausgeben und Lesen von Preisdaten.

Die "Stock"-Klasse dient zur Verwaltung von Aktiendaten und enthält Methoden zum Setzen und Abrufen von Aktiendaten sowie zum Hinzufügen und Importieren von Preisdaten.

Die "StockEntry"-Klasse dient als Wrapper-Klasse für die "Stock"-Klasse und enthält Methoden zum Abrufen und Setzen von Aktiendaten sowie zum Zurücksetzen von gelöschten Daten.

StockHashTable.h → Die StockHashTable-Klasse:

Hier wurde eine Klasse namens „StockHashTable“ definiert, dient als Wrapper für zwei separate Hash-Tabellen. Jede Tabelle enthält Einträge von Aktienobjekten, die jeweils nach Namen oder Abkürzung sortiert sind.

Die Größe der Hash-Tabellen wird als Konstante definiert und ist standardmäßig auf 10 gesetzt. Die StockHashTable-Klasse bietet Methoden zum Hinzufügen, Suchen und Entfernen von Aktieneinträgen, sowie zum Laden und Speichern von Einträgen aus/in eine Datei.

Um Einträge in den Hash-Tabellen zu speichern, verwendet die Klasse die StockEntry-Klasse als Wrapper für Aktienobjekte. Die Klasse StockHashTable enthält auch einige private Hilfsmethoden zur Hashing-Funktion, Suche und Löschung aller Einträge.

❖ Die Hash-Methode

```
int StockHashTable::hash(const std::string &key) const {
    uint32_t hash = 0;
    for (size_t i = 1; i <= key.length(); ++i) {
        hash += key[i] * i;
        hash += (hash << 10);
        hash ^= (hash >> 6);
    }
    hash += (hash << 3);
    hash ^= (hash >> 11);
    hash += (hash << 15);
    return hash % DEFAULT_SIZE; //default_size=10
}
```

Dies ist eine Hashfunktion, die eine Eingabezeichenfolge in einen Hashwert umwandelt, der dann als Index in einer Hashtabelle verwendet werden kann, um den Wert mit der angegebenen Zeichenfolge als Schlüssel aufzurufen.
Die Hashfunktion wird wie folgt implementiert:

- Zuerst wird hash auf 0 gesetzt.
- Dann wird über jede Stelle im Schlüssel durch eine for-Schleife iteriert. Die Schleife beginnt bei 1, da die Verwendung von 0 zu einem Hashwert von 0 führen würde.
- Für jeden Buchstaben im Schlüssel wird der Hashwert aktualisiert. Dabei wird der ASCII-Code des Buchstabens mit seiner Position im Schlüssel multipliziert und mit dem aktuellen Hashwert addiert.
- Der Hashwert wird dann nach links um 10 Bit verschoben und mit dem aktuellen Wert XOR-verknüpft.
- Der Hashwert wird dann nach rechts um 6 Bit verschoben und erneut XOR-verknüpft.

  Diese Bit-Shift-Operationen werden in der Funktion verwendet, um die Streuung der Werte in der Hash-Tabelle zu erhöhen und eine gleichmäßige Verteilung der Einträge in der Tabelle zu erreichen.

- Dies wird dann noch zweimal wiederholt, wobei die Anzahl der verschobenen Bits pro Schleifendurchlauf variiert.
- Schließlich wird der Hashwert mod der Standardgröße berechnet, um einen Index innerhalb der Hashtabelle zu erhalten.

Insgesamt versucht diese Hashfunktion, eine Zeichenfolge in eine numerische Darstellung umzuwandeln, die als Hashwert verwendet werden kann, auch wenn die ursprüngliche Zeichenfolge nicht eindeutig ist. Die Funktion nutzt verschiedene Techniken wie Bitverschiebungen und XOR-Verknüpfungen, um eine gleichmäßige Verteilung der Hashwerte in der Hashtabelle sicherzustellen und somit eine effektive Kollisionsbehandlung zu ermöglichen.

❖ Aufwandschätzung

Die Methoden add()/search()/ remove() haben eine Laufzeit von O(1). Die Hash-Funktion selbst hat eine Laufzeit von O(n). Die Verwendung von Bit-Operationen wie <</>> hilft jedoch, die Laufzeit zu minimieren, da sie schneller sind als die Multiplikationen oder Divisionen.

Insgesamt hat unser Code eine erwartete Laufzeit von O(1), für die wichtigsten Operationen, was ihn zu einer effizienten Implementierung einer Hash-Tabelle macht.