# Protokoll Hashtabelle – Aktienprogramm

### Aufgabenstellung:

Implementieren Sie ein Programm zur Verwaltung von Aktienkursen.Um schnell nach einer bestimmten Aktie suchen zu können, sollen die Aktien in einer Hashtabelle verwaltet werden.Pro Aktie sollen Name, Wertpapierkennnummer (WKNals String)und Kürzel(auch ein beliebiger String)gespeichert werden. Weiters sind pro Aktie die Kursdaten der vergangenen 30 Tage zu verwalten mit folgenden Informationen:(Date,Open,High,Low,Close,Volume,Adj Close)

Das Programm wurde in C++ realisiert. Die von uns erzeugte Klasse Hashtable beinhaltet folgende Funktionen:

* Find\_next\_prime
* Create\_stock
* Hash\_function
* Quadratic\_probing
* Quadratic\_probing\_tag
* Quadratic\_probing\_name
* Find\_stock\_tag
* Find\_stock\_name
* Error
* To\_upper
* Add
* Del
* Import\_data
* Search
* Plot
* Save
* Load
* Quit

Mit dem Programstart wird ein Hashtable Element erzeugt, welche mit dem Standardkonstruktor Speicher für mindestens 1000 Plätze an Aktien reserviert. Find\_next\_prime erzeugt dabei die nächstgrößere Primzahl und setzt sie als Limit für die Hashtabelle und der Modulo-Funktion an.

### Hashfunktion

Folgende Funktion wurde gewählt, um die Werte gleichmäßig in die Tabelle einzufügen:

Wobei y die Primzahl ist und s der ASCII-Wert der Buchstaben ist. Es wurde weiters überlegt, ob man einen höheren Wert als 7 für die Hashfunktion verwenden soll, wir sind jedoch zum Entschluss gekommen, dass dies für eine Anzahl von 1000 Aktien ausreichend sein wird. Sollte es zum Kollisionsfall kommen, wird zu H(s) ein quadratisch wachsender Wert addiert, um eine möglichst große Verteilung der Werte in der Hashtabelle zu erzielen.

### Kollisionserkennung

Für die Kollisionserkennung wird an der Stelle, wo der Wert einzufügen ist überprüft, ob ein Wert bereits vorhanden ist. Mithilfe einer bool‘schen Member-Variable wird überprüft, ob der Wert bereits zuvor gesetzt wurde oder nicht. Sollte diese auf wahr gesetzt sein, wird überprüft, ob es sich um die gewünschte Position handelt oder nicht. Falls nicht, wird solange weiter sondiert, bis man zur gewünschten Stelle landet. Dies kann eine freie Position sein oder ein gesuchter Wert.

### Verwaltung der Kursdaten

Die Verwaltung der Kursdaten wird über eine eigene Datenstruktur umgesetzt. Dafür wurde eine eigene Klasse erzeugt, die das Importieren einer CSV Datei handelt und diese Werte in einem dynamischen Array speichert. Die maximale Anzahl an Datensätzen beträgt dabei 30. Sollte dieser Wert überschritten werden, wird an der letzten Stelle ein weiteres Element hinzugefügt und das erste Element entfernt. Die verwendete Vector-Klasse in C++ macht dies ineffizient. Beim Hinzufügen eines Elementes wird ein Speicherbereich, der um den Anteil größer ist, um das neue Element zu speichern, allokiert, anschließend alle vorhandenen Werte in den neuen Speicherbereich kopiert und dann den alten Speicherplatz freigibt.

### Löschalgorithmus

Lorem ipsum

### Aufwandsabschätzung

Hashtabellen haben den Vorteil, dass sie für die suche, einfügen und löschen eine durchschnittliche Laufzeit von O(1) haben. Dies ist möglich da in einer Hashtabelle der Suchwert mit einer Hashfunktion zu einem integer wert umgewandelt wird. Dieser wert ist nun der Speicher Ort des entsprechenden Datensatzes. Somit können daten leicht eingefügt gesucht und gelöscht werden. Im Worstcase ist die Laufzeit aber O(n). Dieser wert kommt von der Notwendigkeit einer Sondierung, wenn sich zwei Hashwerte unterschiedlicher Datensätze gleichen. Bei einem Array ist die durchschnittliche Laufzeit für suchen, einfügen und löschen O(n) und der Worstcase ist ebenfalls O(n). Dabei ist es irrelevant ob in einem Array etwas eingefügt oder gelöscht werden soll da jedenfalls der Datensatz oder ein leeres Feld gesucht werden müssen. Im fall einer verketten liste, ist es beim davon anhängig ob der Datensatz am Anfang der Liste mit O(1) eingefügt werden soll oder mit O(n) am ende der Liste. Soll der Datensatz nämlich am Ende der Liste eingefügt werden muss nämlich die ganze liste traversiert werden. Die suche hingegen ist sowohl im durchschnittlichen als auch im Worstcase O(n). Dies kommt von der Notwendigkeit im Worstcase die gesamte Liste zu durchlaufen. Aus der Laufzeit der suche resultiert daher auch die des Löschens mit O(n), weil jedes zu löschende Element erstmal gesucht werden muss.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Einfügen | | Suchen | | Löschen | |
|  | Durchschnitt | Worstcase | Durchschnitt | Worstcase | Durchschnitt | Worstcase |
| Hashtabelle | O(1) | O(n) | O(1) | O(n) | O(1) | O(n) |
| Array | O(n) | O(n) | O(n) | O(n) | O(n) | O(n) |
| Verkettete Liste | O(1) oder O(n) | O(n) | O(n) | O(n) | O(n) | O(n) |

Zusammenfassend ist im durchschnittlichen Fall die Hashtabelle immer O(1), welches nur für die verkette Liste mit dem einfügen am Kopf der liste möglich ist. Für alle anderen Fälle ist die Laufzeit O(n).

Bei zunehmenden füllgrad der Hashtabelle werden kollisionen