*Protokoll zur Beschreibung der Datenstruktur*

**Hashfunktion:**

Wir haben uns für die Hashfunktion aus Beispiel 1.9. vom Übungsblatt entschieden: 

Als erstes berechnen wir den Hashwert und lassen diesen dann % durch die size unseres Hashtables rechnen. Der ausgegebene Wert dient uns als Index für unseren Hashtable.

Der Hashtable selbst besteht aus einem Array von Pointern, welche auf verkettete Listen (oder NULL bei keinen Einträgen) verweisen. Somit haben wir durch die verketteten Listen auch gleich die Kollisionsbehandlung abgedeckt und können so Kollisionen vermeiden.

**Kollisionsbehandlung:**

Wird bei einem gleichen Index ein Element im Hashtable hinzugefügt, dann wird an diesem Index ein neues Listenelement hinten an die Liste angehängt. Somit entstehen keine Kollisionen und Daten können trotz gleichen Indexes abgespeichert werden.

**Verwaltung der Kursdaten:**

Um unsere Kursdaten zu verwalten haben wir eine ADD, DELETE und SEARCH function eingebaut. Mit diesen Funktionen ist es uns möglich neue Elemente zu erzeugen, gewünschte Elemente zu löschen und nach bestimmten Elementen in unserem Hashtable zu suchen.

Die ADD function nimmt vom user einen Namen, ein Kürzel und eine WKN entgegen. Sie gibt dann einen stock zurück der diese Werte beinhaltet und gibt diesen stock dann an die Funktion die uns den Index (wie im Punkt Hashfunktion beschrieben) ausrechnet und zurückgibt weiter. Dieser errechnete Index wird dann verwendet, um die entsprechende Stelle des Hashtable Arrays zu erreichen und dort abzuspeichern. Wir haben uns dazu entschieden den Head des Hashtable unsichtbar zu lassen, damit es in späterer Folge beim Löschen dann nicht zu Schwierigkeiten kommt, wenn der Kopf der Liste gelöscht wird.

Die SEARCH function rechnet sich den gesucht Index aus indem sie die WKN nummer hasht und % rechnet wie im oberen Punkt Hashfunktion beschrieben. Durch diese Rechenoperation wissen wir, an welcher Stelle des Hashtablearrays unseren Eintrag abgespeichert werden soll. Wir gehen dann die verkettete Liste so lange durch, bis wir den entsprechenden Eintrag finden und geben die gefunden daten des Listenelements aus.

Die DELETE function berechnet ebenfalls den Index mittel der obrigen Hashfunktion und geht dann die Liste bis zum gesuchten Element durch um das entsprechende Element freizugeben. Die Liste wird dann mit dem übernächsten Element (oder NULL falls es nur ein Element war) verknüpft und ist somit intakt.

**Löschalgorithmus:**Wir haben uns beim Löschen an der SEARCH function orientiert und ihre Funktion erweitert. Das Löschen entspricht den normalen Ablauf, wie wenn bei einer verketteten Liste ein Element entfernt wird. Der Head unseres Hashtables wird dabei nie gelöscht.

**Aufwandabschätzung:**

Der Aufwand eines Hastables beträgt O(1). Der Grund dafür ist, dass der Hashtable selbst einen größeren Datensatz an unsortierten Werten entgegennimmt und diesen in ein Array schreibt, damit dieses zugänglich ist. So erreicht man bei der richtigen Hashfunktion, dass der Index ausgerechnet werden kann und sofort auf das gesuchte Element zugegriffen werden kann.

Wir gehen außerdem davon aus, dass es zu wenigen Kollisionen kommt, womit wir die zweidimensionale Natur eines Hastables aus Listen ignorieren können.

Auf Grund der wenigen Kollisionen ist die O-Notation fast immer konstant.

Wenn es zu Kollisionen kommen sollte, wird die O-Notation von O(1) auf O(n) geändert. (Für diesen Index). Da solche Kollisionen aber sehr selten sind und man allgemein davon ausgehen kann, dass selbst im Falle von regelmäßigen Kollisionen die Zeitgleichung n / arraygröße (indexgröße) \* t ist, so ist die Länge eines Index immer sehr gering, nahezu eins.