**Universität Trier**
LDV / Computerlinguistik
Seminar Softwaretechnik, Wintersemester 2005/2006
Leitung: Prof. Dr. Reinhard Köhler
Referent: Kai Kugler

26.1.2006

# Abstrakte Datenstrukturen und -typen

## 1. Datentypen

### 1.1 Konkrete Datentypen

Ein Datentyp besteht aus Objektmengen und den darauf definierten Operationen (Syntax). Bei Konkreten Datentypen - wie man sie aus den Programmiersprachen kennt - sind die Objektmengen konkrete Wertebereiche mit darauf definierten Funktionen (Syntax und Semantik). Als Beispiel eine (sehr vereinfachte) Spezifikation eines Typs der natürlichen Zahlen (meist 'word' oder 'unsigned integer'):

Syntax:

```
Name          nat
Operationen   new  :            -> nat    (Konstruktor)
              +    : nat x nat -> nat
              -    : nat x nat -> nat
```

Sematik:

```
Menge         nat : Menge der natürlichen Zahlen
Funktionen    new  = {}
              +    : nat x nat (Summe zweier natürlicher Zahlen)
              -    : nat x nat (Differenz zweier natürlicher Zahlen)
```

Die meisten Programmiersprachen bieten vordefinierten elementare ('`int`', '`set`', '`boolean`', '`char`') und zusammengesetzte ('`array`', '`string`', '`struct`') Datentypen, deren Operationen und interne Repräsentation programmiersprachen- und maschinenabhängig sind. Beispiele: ein einfacher boolscher Wert (0 oder 1) belegt in der einen Sprache ein Bit, in einer anderen ein ganzes Byte; Der Vergleich eines Verbundes ('`struct`') mit einem anderen kann in zwei verschiedenen Sprachen anders definiert sein; Eine Fließkommazahl ('`real`') belegt in der selben Sprache auf verschiedenen Systemen unterschiedlich viel Speicher; Eine ganze Zahl ('`integer`') mit 4 Byte wird auf der einen Maschine mit dem höchstwertigen Byte an der niedrigsten Speicheradresse abgelegt ('big endian' – z.B. Motorola 68000), auf einer anderen Maschine genau umgekehrt ('little endian' – z.B. Intel x86-Prozessoren).

### 1.2 Typen von Operatoren

- Konstruktoren: erzeugen Instanzen (z.B. für ein Objekt in Java `Auto mein_Auto = new Auto();` oder auch `String str = "Datentypen";` für Java.lang.String)
- Modifikatoren: verändern eine Instanz (z.B. Elemente hinzufügen, löschen, sortieren)
- Observatoren/Selektoren: liefern Informationen über eine Instanz [z.B. ob eine Instanz 'leer' ist] oder erlauben das 'Auslesen' von Elementen der Instanz [vgl. Funktionen wie `charAt(int index)` für Strings in Java]

### 1.3 Abstrakte Datenstrukturen und -typen (ADS/ADT)

Abstrakte Datenstrukturen und -typen sind von dem konkreten Problem verallgemeinerte (komplexe) Datenstrukturen. Ihre Spezifikation ist unabhängig von ihrer Implementierung und der konkreten Verwendung. Sie bieten eine Schnittstelle mit den erforderlichen Operationen. Wichtige Prinzipien sind:

- Geheimnisprinzip (Interne Realisierung [Daten und Mechanismus der Operatoren] verborgen; Implementierung des ADT kann ohne Auswirkungen auf Programme, die diese benutzen, geändert werden)
- Kapselung (Nutzung nur über die Schnittstelle)

Anzustrebende Eigenschaften einer ADS sind u.a.:

- *Universalität/Wiederverwendbarkeit* (ein implementierter Datentyp kann auch für andere Programme benutzt werden)
- *präzise Beschreibung der Schnittstelle* (der Benutzer braucht die Kenntnis über Vor- und Nachbedingungen von Operationen)
- *Geschütztheit der internen Struktur der Daten nach außen*
- *Modularität*

Möglichkeiten zur Erstellung von ADTs z.B. durch objektorientierte Programmierung oder 'generische Typen' (Datentypen mit der Möglichkeit zur Angabe von Typparametern => Typensicherheit) (vgl. Java ab Version 5, Templates in C++).

## 2 Beispiele

### 2.1 Stapelspeicher (Stack)

- Ein Stapel kann eine Menge von Objekten aufnehmen und gibt diese entgegengesetzt zur Reihenfolge der Eingabe wieder zurück ('Last In, First Out-Prinzip' = 'LIFO') => Position der Elemente von Reihenfolge der Aufnahme abhängig.
- Operationen (erlauben ausschließlich den Zugriff auf das oberste Element des Stapelspeichers):
  - *push* (Modifikator zum Hinzufügen eines Objektes)
  - *pop* (Modifikator zum Entfernen eines Objektes)
  - *top* (Selektor zum Auslesen eines Objektes)
- Anwendungen (z.B.): Auswertung von Klammerausdrücken, Auflösung von arithmetischen Ausdrücken in Infix-Notation, Parsing.

### 2.2 Warteschlange (Queue)

- Ein Queue kann eine Menge von Objekten aufnehmen und gibt diese in der Reihenfolge ihres Einfügens wieder zurück ('First In, First Out-Prinzip' = 'FIFO') => Position der Elemente von Reihenfolge der Aufnahme abhängig.
- Operationen:
  - *enqueue* (Modifikator zum Hinzufügen eines Objektes)
  - *dequeue* (Modifikator zum Zurückholen und Entfernen eines Objektes)
- Anwendungen (z.B.): Datenübergabe zwischen asynchronen Prozessen (Tastaturpuffer o.ä.)

## 2.3 Binärbäume

- Positionen der Elemente im Baum von ihrem Wert/Schlüssel abhängig
- Operationen:
  - Durchlaufen des kompletten Baumes (*tree traversal*) in Pre-, Post-, oder Inorder-Reihenfolge
  - Suchen nach einem Wert/Schlüssel (welche Pfade des Baumes hierfür durchlaufen werden ist nach außen gekapselt)
  - Modifikatoren zum Hinzufügen oder Löschen von Elementen (welche Knoten hierbei gelöscht werden und welche Rotationen bei z.B. AVL-Bäumen (ausgeglichene Bäume) stattfinden spielt nur innerhalb des ADT eine Rolle)
- Anwendung (z.B.): Darstellung von Daten, deren Elemente nach einem bestimmten Schlüssel zu finden sind => 'schnelle' Suche (log(n) Vergleiche bei n Elementen); Anwendungen, die Suchen und Sortieren erfordern.

## 2.4 Objekte

- Das objektorientierte Programmierparadigma unterstützt die Erstellung von ADTs
- Als Beispiel ein `interface` für eine Klasse Stack in Java:

```
public interface Stack{
      public void push (Object neuesElement);
      public Object top();
      public void pop();
}
```

Das `interface` beschreibt nur abstrakt die Schnittstelle, von ihm kann man eine Klasse ableiten und schließlich Objekte vom Typ der Klasse erzeugen. Dieses Objekt verfügt dann über den Selektor `top` und die Modifikatoren `push` und `pop`. Die Realisierung dieser Mechanismen und die Organisation der Daten ist mit der Klasse nach 'außen' gekapselt und geheim.

## 2.5 Lexika als abstrakte Datenstrukturen

- Auch bei einem digital vorliegenden Lexikon sind u.a. Wiederverwendbarkeit und der Zugriff über eine standardisierte Schnittstelle wünschenswert.
- Für Anwendungen möchte man sich keine Gedanken über die interne Organisation der Daten machen (z.B. über das Datenbankdesign), die Schnittstelle soll Selektoren bereitstellen.
- Die Daten müssen nach außen geschützt sein (vor unberechtigtem oder falschem Zugriff - es sollen ausschließlich Selektoren und Modifikatoren der Schnittstelle benutzt werden können).
- Beispiel: Das Wortschatzlexikon an der Universität Leipzig (wortschatz.uni-leipzig.de)[1] bietet als 'Webservices' eine SOAP-Schnittstelle (XML) und stellt u.a. folgende Operatoren (auch als importierbare Java-Objekte) zu Verfügung:
  - *Frequencies* (gibt die Frequenz, sowie die Häufigkeitsklasse einer Wortform)
  - *Wordforms* (gibt alle Flexionsformen der gegebenen Wortform zurück)
  - *Similarity* (gibt automatisch berechnete ähnliche Wörter (Synonyme, Oberbegriffe, Unterbegriffe, freie Assoziationen oder Kohyponyme) zurück)

1 vgl. auch Quasthoff, U.: Deutscher Wortschatz im Internet.

## 2.5 Textkorpora als abstrakte Datenstrukturen

- Maschinenlesbare Korpora werden meist auf einen bestimmten Verwendungszweck hin erstellt und optimiert und besitzen spezifische technische Merkmale (z.B. Zeichencodierung, Datenspeicherung [Datenbank, XML, ...]) =>
  - soll das Korpus für einen anderen Zweck wiederverwendet werden, muß der Nutzer diese Merkmale kennen (Verletzung des Geheimnisprinzips).
  - der Zugriff auf das Korpus richtet sich nach dessen Merkmalen, bei Änderungen müssen (in den meisten Fällen) auch die zugreifenden Prozeduren geändert werden (keine Kapselung).
  - 'jeder' Benutzer erstellt und verwendet eigene Zugriffsoperationen (ineffizient), sollen mehrere verschiedene Korpora für eine Untersuchung herangezogen werden, müssen je Korpus sogar verschiedene genutzt werden.
- Korpora sollen für verschiedene Zwecke wiederverwendbar sein.
- Dem Benutzer soll ein unkomplizierter Zugang (über eine/verschiedene Schnittstelle/n) zu allen im Korpus annotierten Daten möglich gemacht werden (z.B. Operatoren, die die jeweils nächste Einheit [Satz, Silbe, ...], deren Länge oder Position im Text/String zurückgeben).

**Literatur**


- Budd, T.: Classic Data Structures in Java. Boston u.a. 2001.
- Ehrich, H.-D.; Gogalla, M.; Lipeck, U. W.: Algebraische Spezifikation abstrakter Datentypen. Eine Einführung in die Theorie. Stuttgart 1989.
- Köhler, R.: Korpuslinguistik - zu wissenschaftstheoretischen Grundlagen und methodologischen Perspektiven. In: LDV-Forum. Zeitschrift für Computerlinguistik und Sprachtechnologie. 2/2000 (Bd. 20). (Im Druck)
- Quasthoff, U.: Deutscher Wortschatz im Internet. In: LDV-Forum. Zeitschrift für Computerlinguistik und Sprachtechnologie. 2/1998 (Bd. 15), S. 4-23.
- Singh, B.; Naps, T. L.: Introduction to Data Structures. St. Paul 1985.
- Saake, G.; Sattler, K.-U.: Algorithmen und Datenstrukturen. Eine Einführung mit Java. Stuttgart $^{2}$2004.
- Wirth, N.:Algorithmen und Datenstrukturen. Stuttgart $^{3}$1983.