1 Operationen für Zeichenketten

Für die Arbeit mit Zeichenketten im Rahmen von zentralen Prüfungsaufgaben wird der Umfang der zu verwendenden Operationen auf die folgenden eingeschränkt:

- Bestimmen der Länge einer Zeichenkette
- Auslesen eines Zeichens an einer bestimmten Position
- Ersetzen eines Zeichens an einer bestimmten Position
- Verbinden von zwei Zeichenketten zu einer
- Prüfen des Inhalts von zwei Zeichenketten auf Gleichheit
- Lexikographisches Vergleichen von zwei Zeichenketten

Die Prüflinge müssen den Umgang mit den in der Auflistung genannten Operationen in der im Unterricht verwendeten Programmiersprache beherrschen. Die Verwendung weiterer Zeichenkettenoperationen ist nicht zulässig.

2 Operationen für dynamische Reihungen, Stapel, Schlangen und Binärbäume

Im Rahmen von zentralen Prüfungsaufgaben werden für die aufgeführten Klassen die folgenden Operationen verwendet. Die Klassen verwenden Inhaltstypen (bzw. Inhaltsklassen), die jeweils der aktuellen Aufgabenstellung angepasst werden. Mögliche Laufzeitfehler bei der Anwendung der Operationen, z. B. Entnehmen bei einem leeren Stapel oder Zugriff auf eine nicht existierende Position einer dynamischen Reihung, müssen bei der Verwendung durch entsprechende Abfragen explizit ausgeschlossen werden.

Dynamische Reihung

Die Nummerierung der Elemente der dynamischen Reihung beginnt mit dem Index 0.

`DynArray()`
Eine leere dynamische Reihung wird angelegt.

`isEmpty(): Wahrheitswert`
Wenn die dynamische Reihung kein Element enthält, wird der Wert *wahr* zurückgegeben, sonst der Wert *falsch*.

`getItem(index: Ganzzahl): Inhaltstyp`
Der Inhalt des Elements an der Position `index` wird zurückgegeben.

`append(inhalt: Inhaltstyp)`
Ein neues Element mit dem übergebenen Inhalt wird am Ende der dynamischen Reihung angefügt.

`insertAt(index: Ganzzahl, inhalt: Inhaltstyp)`
Ein neues Element mit dem übergebenen Inhalt wird an der Position `index` in die dynamische Reihung eingefügt. Das Element, das sich vorher an dieser Position befunden hat, und alle nachfolgenden werden nach hinten verschoben. Entspricht der Wert von `index` der Länge der dynamischen Reihung, so wird ein neues Element am Ende der dynamischen Reihung angefügt.

`setItem(index: Ganzzahl, inhalt: Inhaltstyp)`
Der Inhalt des Elementes an der Position `index` wird durch den übergebenen Inhalt ersetzt.

`delete(index: Ganzzahl)`
Das Element an der Position `index` wird entfernt. Alle folgenden Elemente werden um eine Position nach vorne geschoben.

`getLength(): Ganzzahl`
Die Anzahl der Elemente der dynamischen Reihung wird zurückgegeben.

Stapel

`Stack()`
Ein leerer Stapel wird angelegt.

`isEmpty(): Wahrheitswert`
Wenn der Stapel kein Element enthält, wird der Wert *wahr* zurückgegeben, sonst der Wert *falsch*.

`top(): Inhaltstyp`
Der Inhalt des obersten Elements des Stapels wird zurückgegeben, das Element aber nicht entfernt.

`push(inhalt: Inhaltstyp)`
Ein neues Element mit dem übergebenen Inhalt wird auf den Stapel gelegt.

`pop(): Inhaltstyp`
Der Inhalt des obersten Elements wird zurückgegeben und das Element wird aus dem Stapel entfernt.

Schlange

`Queue()`
Eine leere Schlange wird angelegt.

`isEmpty(): Wahrheitswert`
Wenn die Schlange kein Element enthält, wird der Wert *wahr* zurückgegeben, sonst der Wert *falsch*.

`head(): Inhaltstyp`
Der Inhalt des ersten Elements der Schlange wird zurückgegeben, das Element aber nicht entfernt.

`enqueue(inhalt: Inhaltstyp)`
Ein neues Element mit dem angegebenen Inhalt wird am Ende an die Schlange angehängt.

`dequeue(): Inhaltstyp`
Der Inhalt des ersten Elements wird zurückgegeben und das Element wird aus der Schlange entfernt.

Binärbaum

`BinTree()`
Ein Baum wird erzeugt. Der Baum besitzt keine Teilbäume. Die Wurzel besitzt keinen Inhaltswert.

`BinTree(inhalt: Inhaltstyp)`
Ein Baum wird erzeugt. Der Baum besitzt keine Teilbäume. Die Wurzel erhält den übergebenen Inhalt als Wert.

`hasItem(): Wahrheitswert`
Wenn die Wurzel des Baums einen Inhaltswert besitzt, wird der Wert *wahr* zurückgegeben, sonst der Wert *falsch*.

`getItem(): Inhaltstyp`
Die Operation gibt den Inhaltswert der Wurzel des Baums zurück.

`setItem(inhalt: Inhaltstyp)`
Die Wurzel des Baums erhält den übergebenen Inhalt als Wert.

`deleteItem()`
Die Operation löscht den Inhaltswert der Wurzel des Baums.

`isLeaf(): Wahrheitswert`
Wenn der Baum keine Teilbäume besitzt, die Wurzel des Baums also ein Blatt ist, wird der Wert *wahr* zurückgegeben, sonst der Wert *falsch*.

`hasLeft(): Wahrheitswert`
Wenn der Baum einen linken Teilbaum besitzt, wird der Wert *wahr* zurückgegeben, sonst der Wert *falsch*.

`getLeft(): Binärbaum`
Die Operation gibt den linken Teilbaum zurück.

`setLeft(b: Binärbaum)`
Der übergebene Baum wird als linker Teilbaum gesetzt.

`deleteLeft()`
Die Operation löscht den linken Teilbaum.

`hasRight(): Wahrheitswert`
Wenn der Baum einen rechten Teilbaum besitzt, wird der Wert *wahr* zurückgegeben, sonst der Wert *falsch*.

`getRight(): Binärbaum`
Die Operation gibt den rechten Teilbaum zurück.

`setRight(b: Binärbaum)`
Der übergebene Baum wird als rechter Teilbaum gesetzt.

`deleteRight()`
Die Operation löscht den rechten Teilbaum.

3 Datenbankabfragen

Im Rahmen von zentralen Prüfungsaufgaben wird in den Aufgabenstellungen und in den erwarteten Lösungen die Abfragesprache SQL mit dem folgenden Sprachumfang unter Berücksichtigung der angegebenen Operatoren und Gruppenfunktionen verwendet. Verschachtelte SQL-Anweisungen werden im Rahmen der zentralen Prüfungsaufgaben nicht thematisiert.

SELECT-Anweisung:

```
SELECT [DISTINCT | ALL] * | spalte1 [AS alias1], spalte2 [AS alias2],
       …, spalten [AS aliasn]

FROM tabelle1, tabelle2, …, tabellem

[WHERE bedingung1 (AND | OR) bedingung2 … (AND|OR) bedingungk]

[GROUP BY spalte1 , spalte2 , … , spalte1]

[HAVING gruppenBedingung1 (AND | OR) gruppenBedingung2 … (AND | OR)
        gruppenBedingungs]

[ORDER BY spalte1 [ASC | DESC], spalte2 [ASC | DESC], ..., spaltet
[ASC | DESC] ]

[LIMIT anzahl]
```

Angaben in eckigen Klammern sind optional. Spalten können Attribute, Aggregatfunktionen oder Berechnungen sein. Bei `GROUP BY` und `ORDER BY` ist auch die Angabe eines Alias möglich.

Operatoren für Berechnungen: `+`, `-`, `*`, `/`

Operatoren für Vergleiche in Bedingungen: `=`, `!=` (ungleich), `>`, `<`, `>=`, `<=`, `NOT`, `LIKE` (mit den Platzhaltern _ und `%`) , `BETWEEN`, `IN`, `IS NULL`

Aggregatfunktionen: `AVG( )`, `COUNT( )`, `MAX( )`, `MIN( )`, `SUM( )`

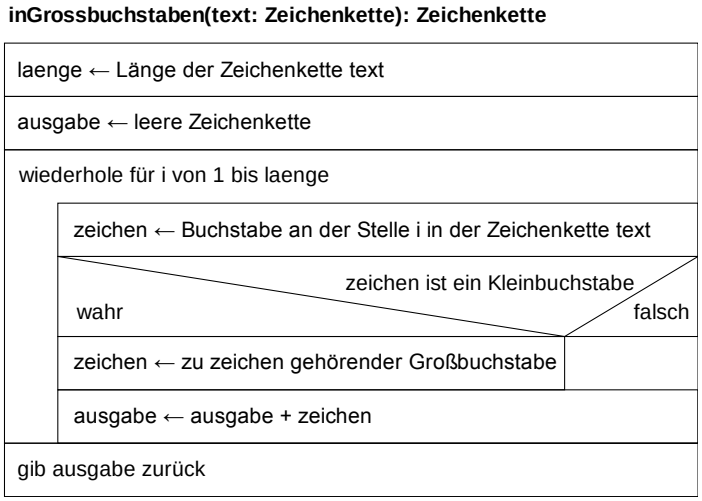
4 Standardisierte Darstellung von Operationen und Algorithmen

- Operationen werden in den Aufgabenstellungen vergleichbar zur Darstellung in den Klassendiagrammen wie folgt notiert, um die Bezeichnung, die Übergabeparameter und den Rückgabewert zu kennzeichnen:

  `Bezeichnung(Parameterbezeichnung: Parametertyp, ...): Rückgabetyp`

  Beispiel: `istGerade(x: Ganzzahl): Wahrheitswert`

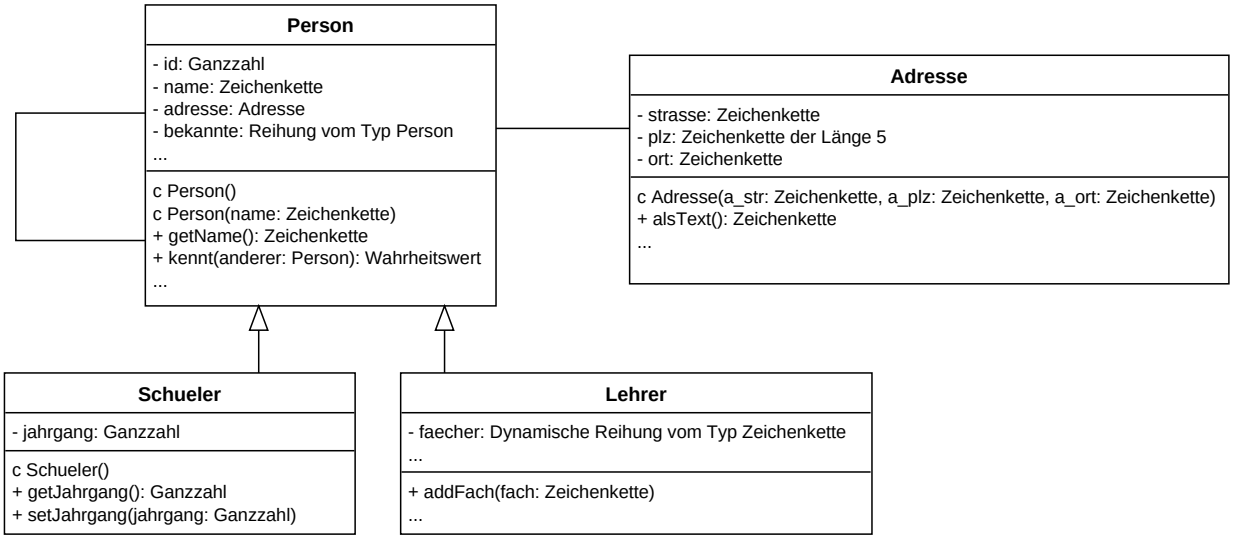
- Zur Darstellung von Algorithmen werden in den zentralen Prüfungsaufgaben Struktogramme (Nassi-Shneiderman-Diagramme) verwendet, wie im folgenden Beispiel exemplarisch dargestellt:



- Die Nummerierung der Elemente von (statischen) Reihungen beginnt mit 0. In Struktogrammen kann der Zugriff auf ein bestimmtes Element einer Reihung durch eine sprachliche Formulierung („n-tes Element von zeichen“) oder durch eine Kurzschreibweise mit eckigen Klammern („zeichen[n]“) dargestellt werden. Die Kurzschreibweise findet insbesondere bei der Verwendung von zweidimensionalen Reihungen Anwendung.

5 Klassendiagramme

Klassendiagramme werden in den zentralen Prüfungsaufgaben wie im folgenden Beispiel exemplarisch notiert. Im Sinne einer übersichtlicheren Darstellung können vereinfacht nur die für die konkrete Aufgabenstellung wichtigen Attribute und Operationen angegeben werden, wie im Beispiel in der Klasse Lehrer dokumentiert. Nicht aufgeführte Attribute bzw. Operationen werden durch eine Pünktchen-Schreibweise verdeutlicht. Konstruktor-Operationen werden durch ein vorangestelltes c notiert. Assoziationen zwischen Klassen werden stets unspezifiziert durch einfache Verbindungslinien angegeben.



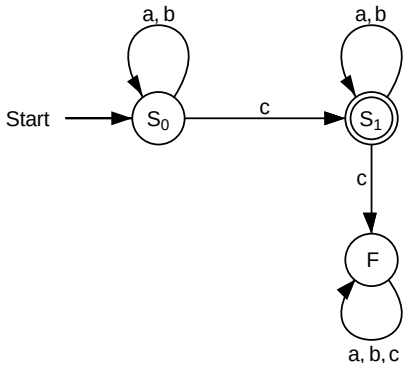
6 Notation von endlichen Automaten, von Mealy-Automaten und von Kellerautomaten

Ein **endlicher Automat** wird durch die Angabe eines Eingabealphabets Σ in Mengenschreibweise und durch einen Übergangsgraphen dargestellt.

Die zusätzliche Angabe der Zustandsmenge, der Menge der akzeptierenden Zustände, des Startzustandes und der Übergangsfunktion ist nicht notwendig, wenn die entsprechenden Informationen dem Übergangsgraphen entnommen werden können. Die grafische Darstellung entspricht dem nebenstehenden Beispiel. Akzeptierende Zustände werden mit einem doppelten Rand markiert.

Endliche Automaten werden, wenn nicht explizit anders vermerkt, vollständig angegeben. Auf die Darstellung eines Fehlerzustandes im Übergangsgraphen kann verzichtet werden, wenn in einem erläuternden Text auf die fehlenden Übergänge und die Existenz eines Fehlerzustandes hingewiesen wird.

Eingabealphabet Σ = {a, b, c}

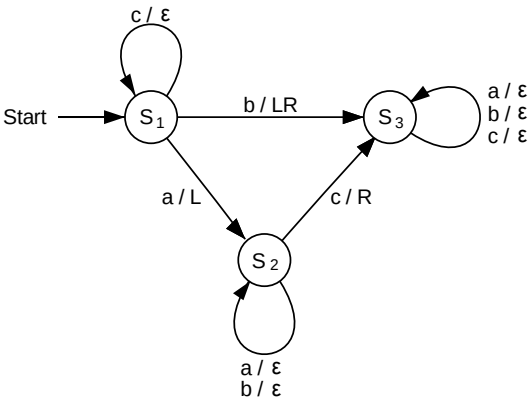


Die obigen Vorgaben gelten analog für **Mealy-Automaten**, allerdings haben sie keine akzeptierenden Zustände. Die Übergänge eines Mealy-Automaten werden wie nebenstehend dargestellt. Am Zustandsübergang erfolgt die Schreibweise in der Reihenfolge Eingabezeichen / Ausgabezeichen.

Zur Vereinfachung sind an den Zustandsübergängen statt eines einzelnen Ausgabezeichens auch Wörter über dem Ausgabealphabet erlaubt.

Es sind auch Übergänge möglich, bei denen keine Ausgabe erfolgt. Eine leere Ausgabe wird mit dem Zeichen ε gekennzeichnet.

Eingabealphabet Σ = {a, b, c}
Ausgabealphabet Ω = {L, R}

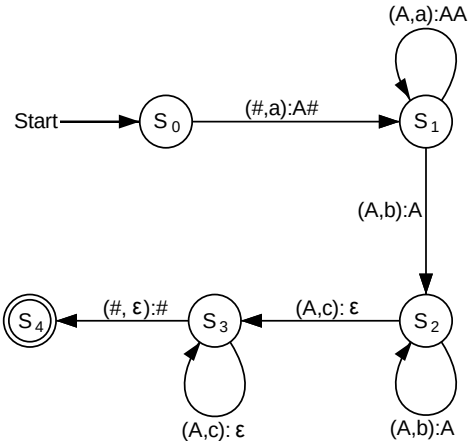


Bei deterministischen **Kellerautomaten** wird am Zustandsübergang in Klammern zuerst das oberste Zeichen des Kellerspeichers gefolgt vom aktuellen Eingabezeichen notiert. Das vom Speicher gelesene Zeichen wird dabei aus dem Keller entfernt. Hinter dem Doppelpunkt wird notiert, welches Zeichen wieder im Kellerspeicher abgelegt wird. Dabei ist auch das Speichern mehrerer Zeichen möglich. Das am weitesten rechts notierte Zeichen wird dann zuerst auf den Keller gelegt.

Der Kellerspeicher besitzt das Vorbelegungszeichen #. Es sind ε-Übergänge möglich. Wenn ein ε-Übergang von einem Zustand ausgeht, so darf von diesem kein weiterer Übergang mit demselben obersten Kellersymbol ausgehen.

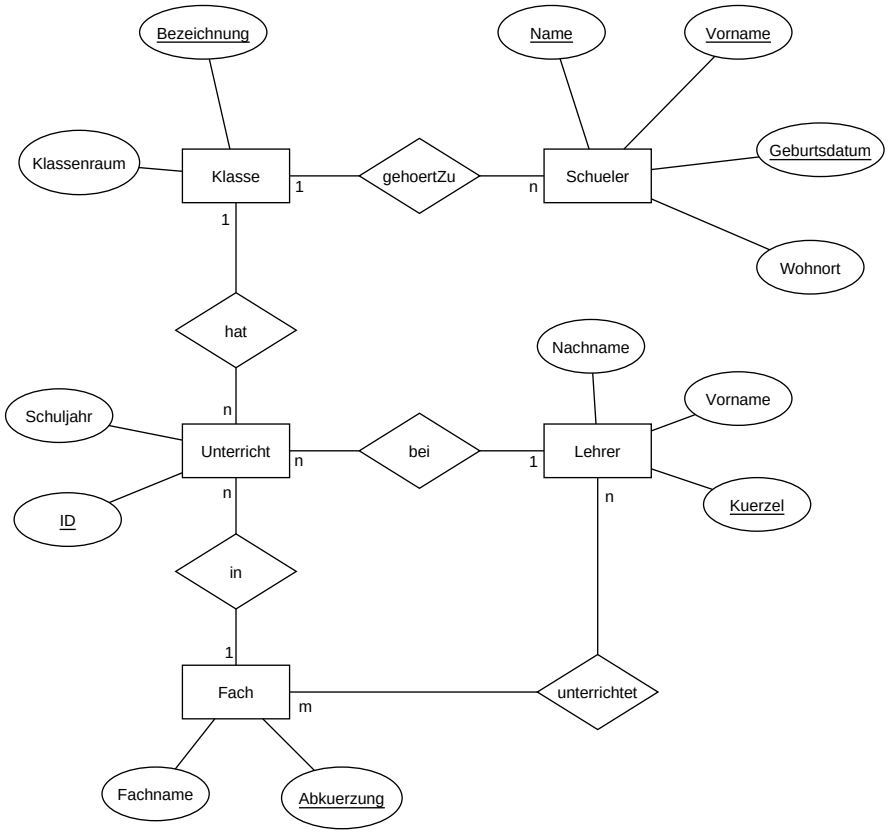
Eine Eingabe wird genau dann akzeptiert, wenn sich der Automat nach vollständiger Verarbeitung der Eingabe in einem Endzustand befindet.

Eingabealphabet Σ = {a, b, c}
Kelleralphabet Γ = {A, #}



7 ER-Diagramme / Kurzschreibweise von Tabellen

Im folgenden Beispiel sind die Komponenten eines ER-Diagramms dargestellt, wie sie in den zentralen Prüfungsaufgaben verwendet werden. Attribute können sowohl Entitätstypen als auch Beziehungstypen zugeordnet werden. Auch rekursive Beziehungen sind möglich. Die Darstellung der Kardinalitäten an den Kanten und die Kennzeichnung von Schlüsselattributen der Entitätstypen sind im Normalfall obligatorisch.



Für **Tabellen in Kurzschreibweise** werden Primärschlüssel durch Unterstreichen und Fremdschlüssel durch Voranstellen eines Pfeils gekennzeichnet.

Beispiel:
Schueler (Vorname, Name, Geburtsdatum, Wohnort, ↑Klassenbezeichnung)
Klasse (Bezeichnung, Klassenraum)