Fachhochschule
Münster University of
Applied Sciences

Einführung in die Informatik I

Einige wichtige Datenstrukturen:
Vektor, Matrix, Liste, Stapelspeicher, Warteschlange

Prof. Dr. Nikolaus Wulff

Datenstruktur / Datentyp

- Programme benötigen nicht nur effiziente Algorithmen, genauso wichtig sind die Datenstrukturen auf denen diese operieren.
- Bis lang kennen Sie nur die „primitiven Datentypen“ wie Ganzzahl und Gleitkommazahl (int, long oder float, double in C).
- Daten werden im Rechner intern als Binärfolgen repräsentiert und je nach Kontext geeignet interpretiert.
- Eine Datenstruktur (oder ein Datentyp) entsteht, wenn zusätzlich geeignete Operationen und Algorithmen angeboten werden (z.B. +, *, / ... für int und double).

Felder

- Der einfachste Datentyp, der sich aus primitiven Datentypen bilden lässt ist ein Feld.
- Ein Feld besteht aus einer *n*-fachen Hintereinanderreihung eines primitiven Datentyps:
  - Zeichenketten lassen sich als Felder von Einzelzeichen darstellen:

    „Hugo...“ ≡ { 'H', 'u', 'g', 'o' ... }

  - Vektoren $\vec{x} \in \mathbb{R}^n$ des *n*-dimensionalen Raums können als geordnete Tupel $\vec{x} = \{x_1, \ldots, x_n\}^T$ realisiert werden.
- Damit daraus echte Datentypen werden müssen noch Operationen wie $\vec{x}^T \cdot \vec{y} = \sum_j x_j \cdot y_j$ definiert werden.

Feldindizierung

- Mathematisch werden die einzelnen Feldelemente durch einen Index k gekennzeichnet: Das k-te Element des Vektors $\vec{x}$ ist x_k .
- In der Informatik werden die Elemente meist durch eckige Klammern innerhalb des Felds adressiert:

$$(\vec{x})_k = x_k \equiv x[k]$$

- Die Addition zweier Vektoren erfolgt komponentenweise:

$$\vec{z} = \vec{x} + \vec{y} \Rightarrow$$

```
for(k=0; k<N; k++) {
    z[k] = x[k] + y[k];
}
```

C-Syntax

Mehrfach indizierte Felder

- Felder können mehrere Indizes haben. Somit lassen sich recht bequem Matrizen und ähnliche Strukturen abbilden: $A \in M(n \times m, \mathbb{R})$

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{pmatrix} \equiv \begin{pmatrix} \vec{a}_1^T \\ \vec{a}_2^T \\ \vdots \\ \vec{a}_n^T \end{pmatrix} \qquad (A)_{jk} = a_{jk} \equiv a[j][k]$$

- D.h. die Matrix A ist ein rechteckiges $n \times m$ Schema bestehend aus n Zeilenvektoren $\vec{a}_k^T \in \mathbb{R}^m$, die jeweils eine Dimension von m haben. (C-Notation!)

Operationen mit Feldern

- Indizierte Felder sind geeignete Datencontainer für Vektoren, Matrizen, etc.
- Um vollwertige Datentypen/strukturen zu erhalten, muss noch definiert werden, welche Operationen auf das jeweilige Datum angewandt werden dürfen.
- Sprachen wie C++ und C# gestatten es den Symbolen/ Operatoren +, -, *, ... eine neue Bedeutung zu geben, genau passend zum Datentyp: die Operatoren werden überladen. Sprachen wie C oder Java kennen dieses Konzept nicht, dort müssen Methoden wie *plus*, *minus*, *multiply* etc. definiert werden, um einen vollwertigen Datentyp zu erhalten.

Listen

- Felder haben eine fest definierte Größe. Listen können dynamisch wachsen und beliebig groß werden.
- Listen sind definiert als eine *geordnete Menge* eines beliebigen Typen *T*.
- Je nach Programmiersprache erfolgt die Realisierung unterschiedlich:
  - In C++ sind dies in der STL vordefinierte Templates.
  - Java verwendet mit dem Typen parametrisierte Klassen, z.B. ArrayList<Double> ist eine Liste mit Elementen vom Typ *T*=Double.
  - C kennt nicht das Konzept einer Liste, allerdings lassen sich entsprechende Datenstrukturen entwickeln.

Listenoperationen

- In Listen können Elemente eingefügt und entfernt werden, ferner lässt sich mit einem Index gezielt ein Element referenzieren.
- Im einfachsten Fall werden Elemente mit *add(obj)* am Ende der Liste angehängt, ansonsten an beliebiger Stelle mit *addAt(index, obj)*.
- Ein Element lässt sich mit *remove(obj)* – falls die Liste das Element nur einmal enhält – bzw. gezielt mit *removeAt(index)* aus der Liste entfernen.
- und mit *obj=getAt(index)* referenzieren.

Listenbeispiel

- Eine Liste mit umgangssprachlichen Wörtern:

Liste: | Ein | Buch | in | einer | Liste |

- Operationen: *add*(“von“) und *add*(“Wörtern“)

Liste: | Ein | Buch | in | einer | Liste | von | Wörtern |

- Operationen: *remove*(“in“) und *remove*(“einer“)

Liste: | Ein | Buch | Liste | von | Wörtern |

- Operationen *addAt*(3,“ist“) und *addAt*(4, “eine“)

Liste: | Ein | Buch | ist | eine | Liste | von | Wörtern |

  - Die Operation *addAt*(3,“ist“) verschiebt das Wort “Liste“ um einen Platz nach hinten.
  - Die Interna hängen von der Programmiersprache ab.

Listen von Zahlen

- Genauso wie mit Wörtern können Listen auch mit jedem anderen im Rechner darstellbaren Datentypen wie z.B. Zahlen umgehen:

- *add*(1), *addAt*(1,2):

- Bemerkung: Auch in C kann eine solche Datenstruktur mit entsprechend programmierten Methoden implementiert werden.

Warteschlangen

- Ein wichtiges häufig vorkommendes Konzept ist eine *Warteschlange*:
  - Anstehen beim Metzger oder vor einem Telefonhaus, Kinoplatzkartenvergabe, Warten in der Telefon-Hotline.
- Eine Warteschlange (engl. *Queue*) funktioniert nach der einfachen Regel: „*wer zuerst kommt wird zuerst bedient*“. In der Informatik wird dies als **FIFO** Prinzip abgekürzt: „First In, First Out“.
- Eine Warteschlange lässt sich intern durch eine Liste realisieren, die nur die Operation *add* und *get* zulässt, wobei letztere intern als *getAt*(1) realisiert ist. So wird immer der Zugriff auf das 1.te Element gewährleistet.

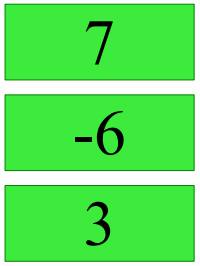
Stapelspeicher

- Ebenso wichtig ist das Konzept des Stapels:
  - Bücher oder Akten auf dem Schreibtisch, etc.
- Ein Stapel (engl. Stack) funktioniert nach der einfachen Regel „*was zuletzt darauf gelegt wurde, muss auch zuerst wieder entfernt werden*“.
- Ein Stapel liefert die zuletzt hinzugefügten Element zuerst. In der Informatik wird dies als **LIFO** Prinzip „Last in, First Out“ abgekürzt.
- Die grundlegenden Operationen einer Queue sind *push* zum auf den Stapel legen und *pop* zum Entfernen eines Elements vom Stapel. Auch die Queue lässt sich intern durch eine Liste realisieren.

Vergleich von Queue und Stack

- Ausgangssituation:

lifo

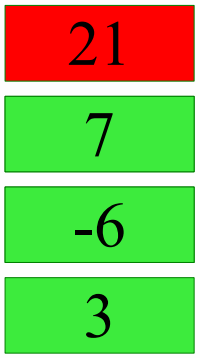


oben drauf legen...

- fifo

rechts anstellen...

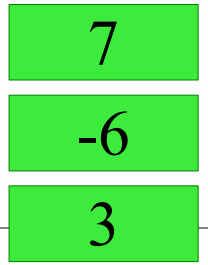
- fifo add(21) bzw. lifo push(21)



links entfernen...

oben herunter-nehmen...

- fifo get → 3 und lifo pop → 21



Stack und Queue in der Informatik

- Im Kapitel über Assembler hatten wir gesehen, das die Parameter für ein Unterprogramm per ***push*** Befehl auf den „Stapel gelegt wurden“.
- Die gerufene Methode holte sich diese per ***pop*** Befehl wieder zurück. Ebenso wurden die CPU Register per ***push*** und ***pop*** Operationen gesichert.
- Beim Multitasking werden die einzelnen Prozesse vom Betriebssystem in eine Warteschlange gestellt. Jeder Prozess erhält eine gewisse Zeitspanne zum Arbeiten, danach wird er wieder hinten in die Schlange eingereiht und muss warten...