Technischer Report

Algorithmen und Datenstrukturen im WS 2005/06

Datentypen

Datentypen

- Datentypen
  - Mengen von Werte
  - Operatoren auf diesen Werten
- Einfache Datentypen
  - Integer, Float, Boolean
- Komplexe Datentypen
  - Arrays, Strings, Klassen
- Abstrakte Datentypen
  - Verallgemeinerung auf allgemeine Eigenschaften
    - ohne konkreten Anwendungsbezug
    - Datentyp allein durch Verhalten charakterisiert
    - sehr mathematische Auffassung: Formales System
    - in Informatik durch Klasse mit folgenden Eigenschaften realisiert

Datentypen

- Datentypen
  - Fortran (1954 bis 1957)
    - Speicherplatz
    - effiziente Ausnutzung des vorhandenen Speicherplatzes
    - kann im Prinzip vom Compiler erledigt werden
      - Lisp
    - Deklaration
    - Sicherheit stand nicht im Vordergrund
    - „implizite“ Deklaration von Daten
      - `Integer: IZAHL, JMAL, KWERT, LUDWIG, MNVWZ, NORDPL`
      - `Real: ALPHA, HELIUM, OMEGA, ZERO`
      - `NO IMPLICIT`

Datentypen

- Datentypen
  - Feldzugriff
    - ```
      integer array Feld [1..10];
      integer Wert;
      Feld[11] := 7; // Jetzt ist Wert == 7
      ```
  - Keine Fehlererkennung in Fortran
    - ```
      real RealWert;
      integer IntWert;
      RealWert := 2.0;
      IntWert := RealWert; -- Falsches Ergebnis, keine Fehlermeldung
                           -- oder Warnung
      ```
  - Fehlererkennung in typsicheren modernen Sprachen
    - ```
      boolean Logisch;
      integer Zahl;
      Zahl := Logisch;     -- Fehlermeldung oder Warnung d. Compiler
      ```
      ```

Datentypen

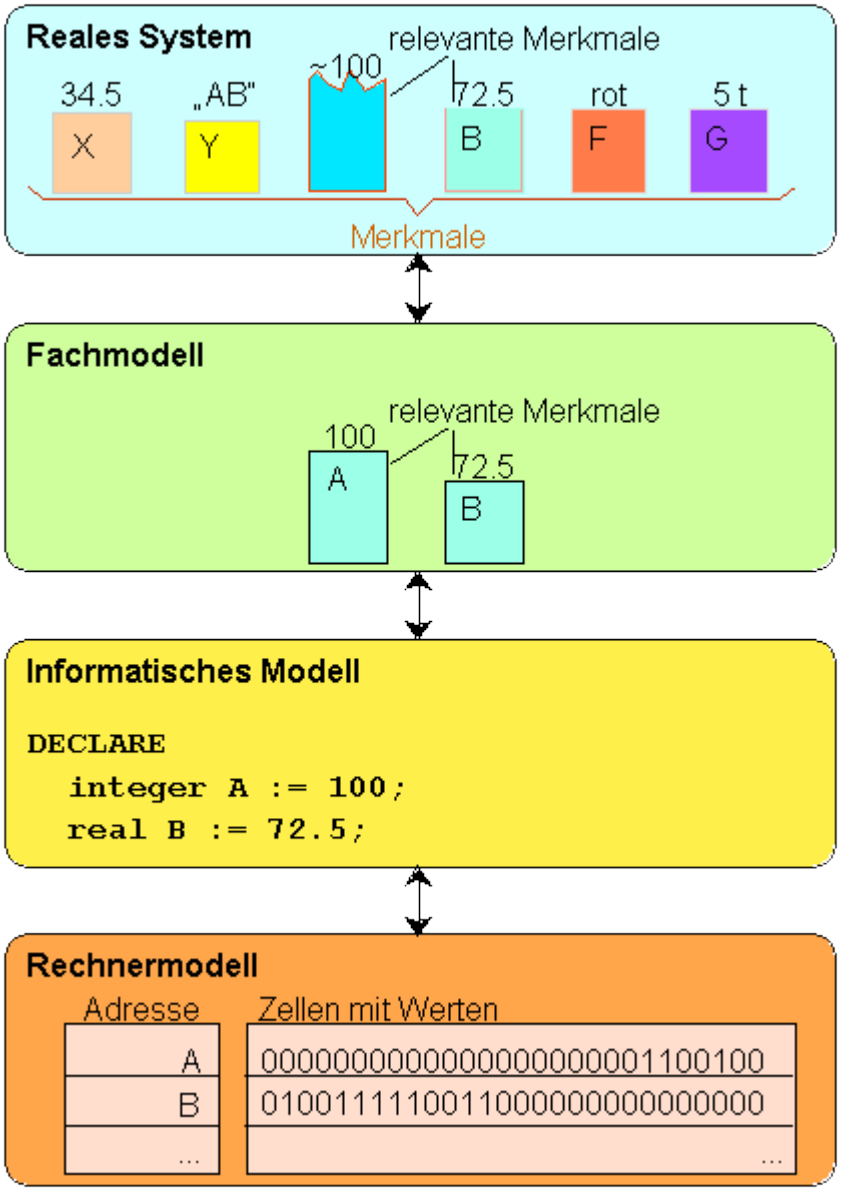
- Anforderungen an Datentyp
  - Notation eines Ausdrucks zeigt Bedeutung des Ausdrucks
    - ist Ausdruck sinnvoll?
  - Ausdruck in Programmiersprache stimmt mit analoger Bedeutung in wissenschaftlicher Fachsprache überein
    - häufig mathematische Notation
  - Fachsemantik: Bedeutung eines Ausdrucks in Fachsprache
  - Programmsemantik: Bedeutung eines Ausdrucks in Programmiersprache
  - Beispiel Addition
    - 1 + 2
    - 1.0 + 2.0
    - 1.0 + 2
    - 1 + true

Datentypen

- Datentyp
  - Menge von Werten (den Merkmalsausprägungen) zusammen mit den möglichen Änderungen, insbesondere Verknüpfungen dieser Werte.
  - Syntax eines Datentyps
    - Notation der Werte
    - Notation für Veränderungen der Werte
  - Semantik eines Datentyps
    - Information, die Werten im Programm zukommt.

Datentypen

- Darstellungsebenen von Werten
  - Speicherdarstellung
  - Programmdarstellung in Programmiersprache
  - Fachdarstellung
  - Reales System



Datentypen

- Variablendeklaration
  - In Pascal (und ähnlich in Ada)
    - `var Zahl: integer;`
    - `var Wert: real;`
    - `var Zahl2: integer := 23; // Ada`
  - Algol, C, Java, Fortran:
    - `integer Zahl;`
    - `real Wert;`
    - `real Wert2 = 1.5, Wert3 = 25;`
  - Standardbelegung mit Werten
    - In Java
      - `int zahl; // zahl == 0!`
    - In C
      - `int zahl; // zahl ==????`

Datentypen

- Speichersemantik und Wertesemantik
  - Wert im Sinne der Fachsemantik
    - Ergebnis := 2 + Pi + Delta;
    - 2 + 3.14159265 + 0.01
  - Speichersemantik
    - Werte werden nur als Bitmuster interpretiert
  - Wertesemantik
    - Programmsemantik und Fachsemantik stimmen überein
    - Darstellung des Werts im Speicher ist völlig unerheblich
    - in modernen Programmiersprachen allgemein üblich
    - deutlich kompliziertere Compiler

Datentypen

- Typumwandlung
  - 2 + 3.14
    - Ganzzahl 2 mit Wert in Gleitkommazahl 2.0
    - geeignetes Rechenwerk führt Addition durch
  - Typumwandlung
    - Änderung des Datentyps
    - Änderung der Speicherdarstellung
    - Erhaltung der Wertesemantik
  - Automatische Typumwandlung
    - implizite Typumwandlung oder Typanpassung
  - explizite Typumwandlung
    - Funktion für Typumwandlung nötig

Datentypen

- Typkonversionsfunktion
  - INTREL(2) + 3.14
  - Type Casting
    - `int * Zeiger;  -- C-Syntax: Integer-Zeiger`
    - `Zeiger = (int *) malloc (sizeof(int));`
      `-- erzeugt Integer-Objekt`
    - `int ASCIIWert;`
    - `ASCIIWert = (int) 'A'`
  - Typäquivalenzregeln
    - kompatible Typen
      - GanzZahl := 'A';
      - Zeichen := 112;
    - explizite Typanpassungen
      - GanzZahl := ord('A');
      - Zeichen := chr(112);

Datentypen

- strenge Typisierung
  - keine automatische Typanpassung
    - MODULA2
    - INTEGER, CARDINAL
      - '1' mehrdeutig
      - Keine Addition erlaubt: intZahl + cardZahl
      - Zuweisung erlaubt: intZahl := cardZahl

Datentypen

- Rundung (round)
  - `Wert := 2.1;  -- Wert == 2`
    `Wert := 2.49;  -- Wert == 2`
    `Wert := 2.51;  -- Wert == 3`
    `Wert := -2.49;  -- Wert == -2`
    `Wert := -2.51;  -- Wert == -3`
- Abschneiden (truncate)
  - `Wert := 2.1;  -- Wert == 2`
  - `Wert := 2.49;  -- Wert == 2`
  - `Wert := 2.51;  -- Wert == 2`
  - `Wert := -2.49;  -- Wert == -2`
  - `Wert := -2.51;  -- Wert == -2`
  - Java
    - `Math.floor()`
    - `Math.floor()`
    - `Math.ceil()`

Datentypen

- Typüberprüfung
  - Überprüfung der korrekten Verwendung von Objekten bei einer Operation
  - statische Typüberprüfung
    - während der Compilierung
  - dynamische Typüberprüfung
    - während des Ablaufs eines Programms
  - Typinferenzverfahren
    - Typen in beliebig komplexen Ausdrücken berechnen
  - Typsystem
    - Typäquivalenzregeln
    - Typinferenzregeln
    - Regeln zur Konstruktion neuer Typen

Datentypen

- Objekt mit Werten verschiedener Typen
  - Variant-Tag Field in Pascal und Modula-2
  - Union in C
    - ```
      union MehrTypObjekt {
        int GanzZahl;
        double  GleitZahl;
        char Zeichen;
      };
      ```
  - keine Überprüfung
    - ```
      MehrTypObjekt Objekt;
      Objekt.GanzZahl := 2;
      RealZahl := Objekt.Gleitzahl;
      ```

Datentypen

- Objekt mit Werten verschiedener Typen
  - Compiler kennt Typ nicht
    - Type Casting
      - Werte der Objekte entsprechend im Speicher ausgerichtet!
  - Ada löst Problem durch Mehrfachzuweisung
    -
```
Type ZahlZeichen(Art: Boolean) is
        record case Art is
          when TRUE => Zahl: integer;
          when FALSE => Zeichen: character;
               end case
        end record;
        ...
        Buchstabe: ZahlZeichen(FALSE);
        Wert: ZahlZeichen;
        ...
        Wert := (Art => TRUE, Zahl => 123);
        Wert := (Art => FALSE, Zeichen => "A");
```

Datentypen

- strukturelle Äquivalenz
  - gleicher Speicher
  - gleiche Interpretation
  - Fortran, Algol68
    - Feld[11] = realZahl;
    - Feld[1] äquivalent zu realZahl
    - dynamische Typanpassung
- Namensäquivalenz
  - Nur Datentypen mit gleichem Namen äquivalent

Datentypen

- Zeigersemantik
  - Objekte nur über Referenzen erreichbar
    - Einheitliche Behandlung aller Objekte
    - Für elementare Objekte (int, double) ineffizient

Datentypen

- Überladung
  - gleicher Operator für verschiedene Funktionalitäten
  - jeweilige Funktionalität hängt vom Typ der Parameter ab
    - Operator „+“
      - Addition ganzer Zahlen,
      - rationaler Zahlen,
      - reeller Zahlen,
      - komplexer Zahlen,
      - hyperkomplexer Zahlen
      - Verknüpfung in (abelschen) Gruppen
      - disjunkte Mengenvereinigung
      - usw.
    - Klammer |<Objekt>|
      - Absolutwert ganzer, rationaler oder reeller Zahlen
      - Betrag komplexer Zahlen (Abstand zwischen Ursprung und Zahl)
      - Anzahl von Elementen in einer Menge (Mächtigkeit) usw.

Datentypen

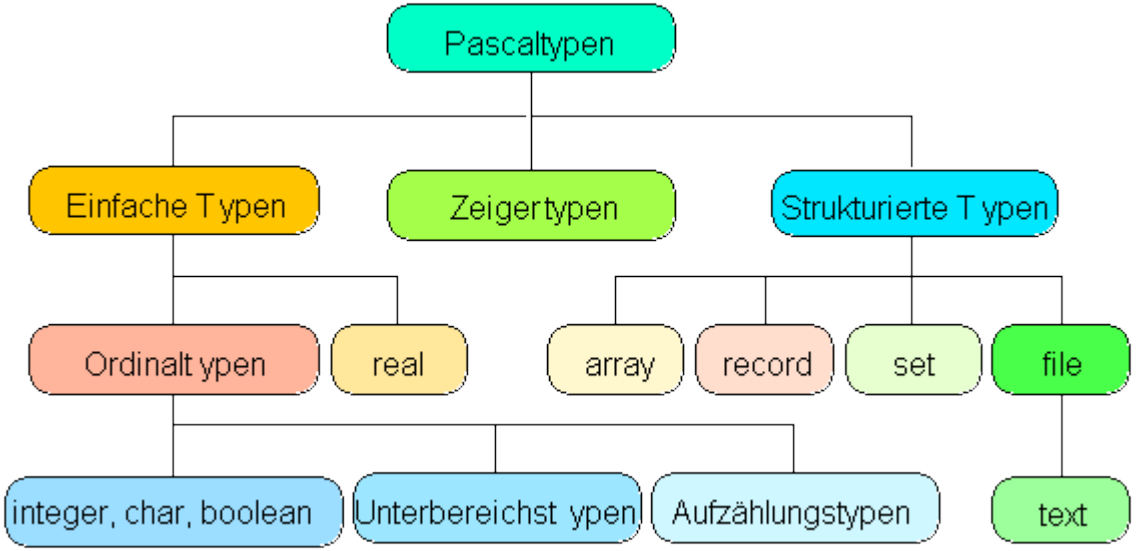
- Funktionalität hängt ab
  - vom Typ der Operanden
  - von Anzahl der Operanden
  - Typ des Ergebnisses
    - Operator „–“
      - Vorzeichen (unärer Operator: „–3“, 1 Operand)
      - Subtraktionszeichen (binärer Operator: „5–2“, 2 Operaden)
  - **Überladen (overloading)** eines Operators
    - Verschiedene Funktionalitäten mit gleichem Operator
      - +, -, *, /, :=, usw.
      - Funktionsname
    - Addition verschiedener numerischer Typen
      - unterschiedliche Operation aufgefasst werden. Seien AddIntInt, AddIntReal usw. Funktionen zur Addition zweier Ganzzahlen, von Ganzzahlen und Gleitkommazahlen usw., so gilt:

Datentypen

- Addition verschiedener numerischer Typen
  - AddIntInt, AddIntReal usw.
  - ```
    2 + 3 == AddIntInt(2 ,3);
    2.0 + 3.0 == AddRealReal(2.0 ,3.0);
    2 + 3.0 == AddIntReal(2 ,3.0);
    2.0 + 3 == AddRealInt(2.0 ,3);
    ```
  - Statt Typanpassung eigenen Operation
    - durch Typtransformation realisiert
  - Operation bestimmt sich durch
    - Funktionsnamen („+“)
    - Typen der Parameter
    - Anzahl der Parameter
  - jeder Operator mit beliebigem Operanden
    - ```
      "A" + 1 == "B"; // Nachfolger eines Zeichens
      "a" * 0 == "a";
      "a" * 1 == "A";
      "A" * 0 == "a";
      "A" * 1 == "A";
      ```
    - allzu exotische Zuordnungen vermeiden!

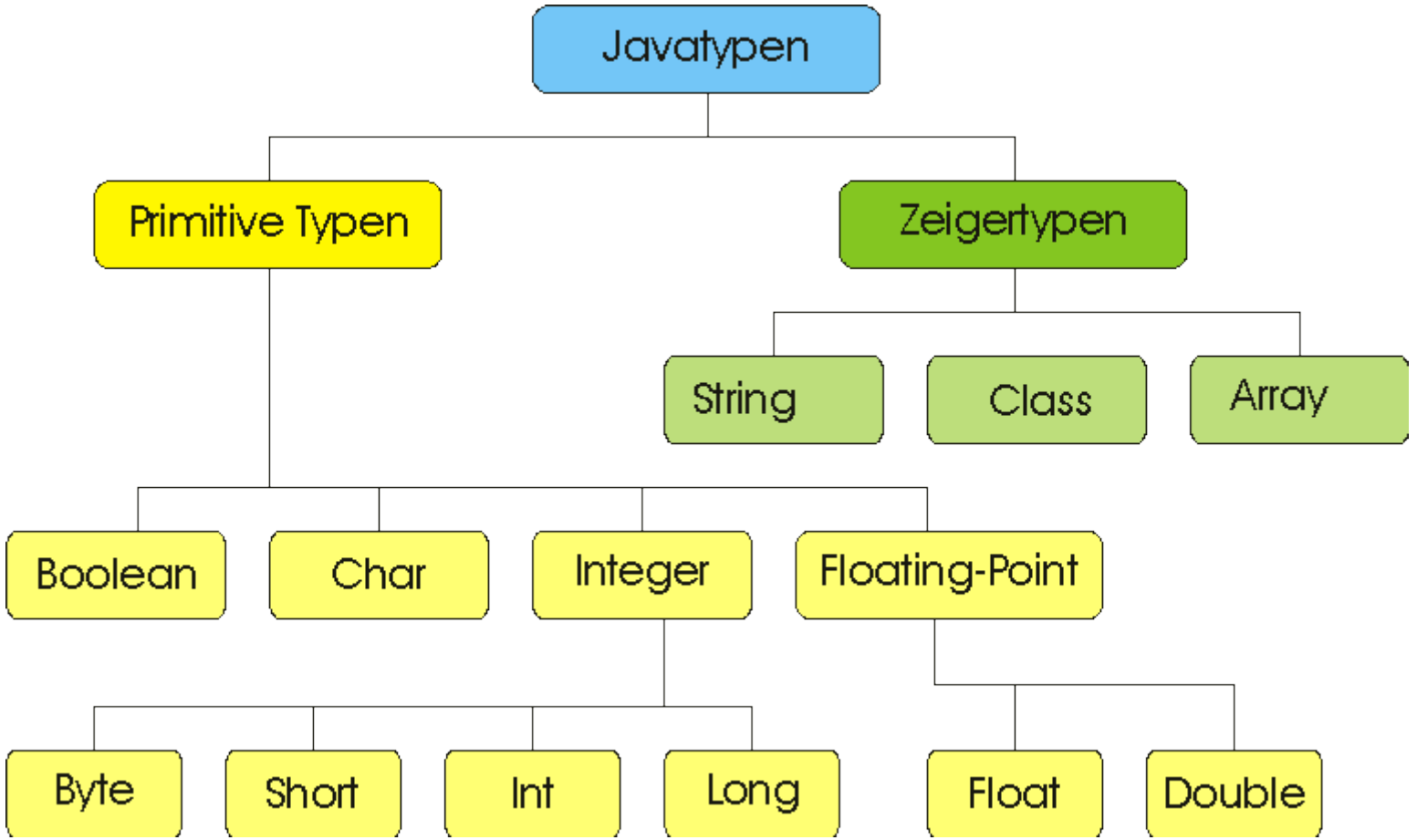
Datentypen

- Typstrukturen von Programmiersprache
  - In Sprachen sehr unterschiedlich
  - Uneinheitliche Benennung
- Pascal



Datentypen

- Typstrukturen von Programmiersprache
  - Java



Datentypen

- Einfache Typen
  - In meisten Sprachen vordefinierte Typen
    - übliche arithmetische Struktur
    - Ordnung zwischen den Werten
    - Speicherplatz fester Länge
    - vollständige lineare Ordnung
  - Beispiele
    - Ganzzahlen (integer)
    - Gleitkommazahlen (real)
    - Zeichen (character)
    - Ordnung
      - Operatoren <, <=, >, >=, ==, <>
      - numerischen Datentypen Operatoren +, –, *
      - Division: /
      - ganzzahlige Division oft anderer Operator (z.B. // oder div)

Datentypen

- Zeigertypen
  - Erst in modernen Typsystemen
    - nicht in Fortran, Algol60, Basic (dialektabhängig)
    - keine dynamischen Variablen
    - in Simula67, Pascal und Algol68
  - Werte von Zeigertypen sind Adressen
  - Algol68
    - **`ref real Name = loc real;`**
      - Name ist Konstante
      - Wert von Name ist Adresse einer Real-Variablen
      - loc real entsprechender Speicherplatz für real-Wert
      - äquivalent auch
    - **`real Name;`**
    - Initialisierung
      - **`real Name := 3.14;`**
        `real Pi = 3.14;`

Datentypen

- Zeigertypen
  - Simula 67

```
ref(integer) Zeiger1, Zeiger2;
Zeiger1 :- NEW integer;// Adresse von Integer Speicher
Zeiger2 :- NEW integer;// Adresse von Integer Speicher
Zeiger1 := 1;          // adr val Zeiger1 := con 1
Zeiger2 := 2;          // adr val Zeiger2 := con 2
Zeiger1 := Zeiger2;// adr val Zeiger1 := val val Zeiger2
Zeiger1 :- Zeiger2;// adr Zeiger1 := val Zeiger2
```

  - C

```
int * Zeiger1, * Zeiger2;
Zeiger1 = new int; /* Andere Syntax als in C */
Zeiger2 = new int; /* Andere Syntax als in C */
*Zeiger1 = 1;      // adr val Zeiger1 := con 1
*Zeiger2 = 2;      // adr val Zeiger2 := con 2
*Zeiger1 = *Zeiger2;//adr val Zeiger1 := val val Zeiger2
Zeiger1 = Zeiger2; // adr Zeiger1 := val Zeiger2
```

Datentypen

- Zeigertypen
  - Simula 67

```
ref(integer) Zeiger1, Zeiger2;
Zeiger1 :- NEW integer;// Adresse von Integer Speicher
Zeiger2 :- NEW integer;// Adresse von Integer Speicher
Zeiger1 := 1;          // adr val Zeiger1 := con 1
Zeiger2 := 2;          // adr val Zeiger2 := con 2
Zeiger1 := Zeiger2;// adr val Zeiger1 := val val Zeiger2
Zeiger1 :- Zeiger2;// adr Zeiger1 := val Zeiger2
```

  - Pascal

```
var Zeiger1, Zeiger2: ^integer;
new(Zeiger1);
new(Zeiger2);
Zeiger1^ := 1;
Zeiger2^ := 2;
Zeiger1^ := Zeiger2^;//adr val Zeiger1:= val val Zeiger2
Zeiger1 := Zeiger2;    //adr Zeiger1 := val Zeiger2
```

Datentypen

- Zeigertypen
  - Vergleich auf Identität: Simula67
    - `Zeiger1 = Zeiger2; // val val Zeiger1 == val val Zeiger2`
    - `Zeiger1 == Zeiger2;// val Zeiger1 == val Zeiger2`
  - Vergleich auf Identität: String in Java
    - `string1 == string2; // val string1 == val string2`
    - `string1.compareTo(string2)==0;`
      `// val val string1 == val val string2`

Datentypen

- Zeigertypen
  - Inkrementierungsoperationen für Adressen in C
    Kopiere Text
    - `char * Zeiger1; char * Zeiger2 = "Hallo";`
    - `while (*Zeiger1++ = *Zeiger2++);`
  - Zeiger1 und Zeiger2 jeweils Adressen einer Zeichenfolge (oder Text)
    - Werte, auf die Zeiger weisen, kopiert
    - Zeiger (Adressen) inkrementiert (durch „++“)
    - letztes Zeichen eines Texts ist NUL (=0)
      - Wert wird als falsch interpretiert
      - Schleife hält an
    - berüchtigte Unlesbarkeit von C-Code
    - Feldnamen nach Operation zeigen hinter letztes Element des Texts
    - In C sind Feldnamen Zeiger auf erstes Element des Feldes
  - Arbeiten mit Zeigern sehr anspruchsvoll!
  - Zeiger für dynamische Datenstrukturen wie Listen oder Bäume

Abstrakte Datentypen

- Abstrakte Datentypen
  - Werte (Variable) + Operationen (Funktionen) auf diesen Werten
  - Schnittstelle
    - Menge (nicht privater) Funktionen und Variablen
  - Anfangswerte bei Initialisierung
    - Konstruktor
    - Destruktor (nicht in Java)
  - Werte unterteilt in
    - schreibbar (write), // setter-Methode: setVariablenName(..);
    - lesbar (read) und // getter-Methode: getVariablenName(..);
    - privat (private)
    - Geschützt (protected; in abgeleiteten Klassen zugreifbar)
  - Werte durch Funktionen des abstrakten Datentyps veränderbar
  - Funktionsnamen können überladen werden
  - Standardoperatoren können überladen werden (nicht in Java)

Abstrakte Datentypen

- Abstrakter Datentyp (ADT: abstract data type)
  - Werte und Operationen nicht für konkretes System entwickelt
  - zur Modellierung der Merkmale mehrerer Systeme geeignet
- Beispiel
  Konkreter Datentyp
  - ```
    class EidotterFarbe
    {final static int Weiß=0, Gelb=1, Orange=2, Rot=3;
     int value = Weiß; }
    boolean isDunkler(EidotterFarbe Ei1, Ei2)
    {return (Ei1.value > Ei2.value;}
    ```
  - Damit ist dieser konkrete Datentyp vollständig spezifiziert
  - Abstrakt: Allgemeiner Farbvergleich
    - Eidotterfarbe
    - Anlassfarben von Schmiedeeisen
    - RAL-Farbpalette
    - Wandfarben
    ```

Abstrakte Datentypen

- Mathematik definiert Datentyp meist durch Eigenschaften
  - Ganze Zahlen durch Peano-Axiome definiert
  - Unterschiede zur Informatik
    - Endlichkeit der Menge der Zahlen
    - Division durch null nicht verboten (durch Exception abfangbar)
    - Anzahl der Berechnungsschritte endlich

      'endlich' in Mathematik ≠ 'endlich' in der Informatik!

      siehe rekursive Berechnung der Fibonacci-Zahlen
  - Begriff ADT stammt aus der (reinen) theoretischen Informatik
  - Beispiele
    - Zahlen
    - Stack (unendlich!)
      - push a
      - pull a
      - a == pull(push a)
      - ...
    - Queue (Doubled Ended Queue = DEQUE)

Abstrakte Datentypen

- Datentyp und Eigenschaften
  - reelle Zahlen in Mathematik
    - Kommutativität und Assoziativität selbstverständlich
  - Gleitkommazahlen in Informatik
    - Mantisse der Gleitkommazahlen nur drei Dezimalstellen
      - $(4 + 4) + 1000 = 8 + 1000 = 1010$
    - Wert acht auf zehn aufgerundet
      - $4 + (4 + 1000) = 4 + 1000 = 1000$
    - Wert 1004 auf 1000 abgerundet
    - bei Gleitkommazahlen gilt i.d.R. nicht das Assoziativitätsgesetz
  - BigNumbers, BigDecimal, BigInteger (s.h.)

Abstrakte Datentypen

- Datentyp
  - Spezifikation eines Datentyps über abstrakten Eigenschaften aus praktischer informatischer Sicht unzulänglich
    - mathematische Struktur gibt Verhalten informatischer Datentypen zu ungenau wieder
    - unnötig
      - informatischen Datentypen speichern und verknüpfen Werte
        - formale Struktur der Mathematik beweist abstrakte Aussagen
          - Mathematikers definiert dazu Eigenschaften seines formalen Systems
        - Informatiker benötigt Ergebnisse konkreter Berechnungen
        - Beschreibung von Datentypenbesser durch operationale oder funktionale Beschreibung der Änderung von Speicherwerten
        - zu informatischem Konstrukt Operationen in Maschinenmodell

Abstrakte Datentypen

- Datentypen in Programmiersprachen
  - Weitere Anforderungen an Datentypen
    - Kapselung, *encapsulation*
      - fasse Werte und Operationen an einer Stelle zusammen
    - Verbergung der Implementierung, *information hiding*
      - verberge Darstellung von Werte und Implementierung von Operatoren
        - komplexe Zahlen in Koordianten (x,y) oder Polar (r,φ)
    - Beispiel
      - BigNumbers, BigDecimal, BigInteger

Abstrakte Datentypen

- Datentypen in Programmiersprachen
  - Weitere Anforderungen an Datentypen
    - Erweiterung, *extension*
      - erweitere Menge an Werten und Operationen
    - Einschränkung
      - schränke Menge an Werten und Operationen ein
    - Neudefinition
      - ändere Werte und Operationen
    - Abstraktion, *abstraction*
      - fasse gemeinsame Werte und Operationen an einer Stelle zusammen
    - Polymorphisierung, *polymorphism*
      - jeder Operator kann mit jedem Operanden benutzt werden
        - ganz+real
        - complex*real
        - usw.

Abstrakte Datentypen

- Polymorphisierung, *polymorphism*
  - Operatoren (Methoden) in abgeleiteten Klassen überschreiben Methoden in Basisklassen (**`virtual`** in C++, Simula 67)
  - ```
    class A {
      void drucke(){...}
    }
    class B extends A {
      void drucke(){...} // überschreibt drucke() in A
    }
    { A a = new A(), b = new B();
      a.drucke(); // ruft drucke aus A() auf
      b.drucke(); // ruft drucke aus B() auf
      if(a instanceof A) ... // ist wahr
      if(a instanceof B) ... // ist falsch
      if(b instanceof A) ... // ist falsch
      if(b instanceof B) ... // ist wahr
    }
    ```
    ```

Abstrakte Datentypen

- Polymorphisierung, *polymorphism*
  - Anwendung: Objekthierarch
  - ```
    class Geometrie {
      void draw(){...}
    }
    class Dreieck extends Geometrie {
      void draw(){...} // überschreibt draw() in A
    }
    class Viereck extends Geometrie {
      void draw(){...} // überschreibt draw()in A
    }
    { Geometrie dreieck = new Dreieck();
      Geometrie viereck = new Viereck();
      dreieck.draw(); // zeichnet ein Dreieck
      viereck.draw(); // zeichnet ein Viereck
    }
    ```
    ```