Algorithmen und Datenstrukturen

Foliensatz, 08.12.05
wird noch ergänzt!

Wintersemester 2005/2006

Ausdrücke

Ausdrücke

- Gleichungen in der Mathematik
  - $A = 3 \cdot A - B$
  - $A = B / 2$
- Wertzuweisungen in der Informatik
  - A := 3*A – B
  - Neuer Wert von A ist dreifacher alter Wert von A - B
- Wertzuweisungen in 3AA
  - adr A := con 3 * val A – val B
  - Bedeutung eines Objekts hängt von Umgebung ab.

Ausdrücke

- Definition
  - Objekt
    - Merkmal eines realen Systems im informatischen Modell
  - Wertebelegung oder Objektausprägung
    - Typ (type) oder Klasse (class von classification)
  - Typ (type) oder Instanz (instantiation)
    - Beschreibung eines Objekts

Ausdrücke

- Ausdruck in Mathematik
  - (Kosten + Preis) · Steuer
  - Kosten + Preis · Steuer
- Vorrang (precedence)
  - von Operatoren in Ausdrücken durch Klammerung
  - Assoziativitätsregel
    - (Mehl + Wasser) + Salz = Mehl + (Wasser + Salz)
      - Rundungsfehler!
    - Linksassoziativ
      - linker Operator vor rechtem ausführen
      - Mehl – Wasser – Salz = (Mehl – Wasser) – Salz
    - Rechtsassoziativ
      - rechten Operator vor linkem ausführen
      - Mehl – Wasser – Salz = Mehl – (Wasser – Salz) ?

Ausdrücke

- Ausdruck in Mathematik
  - sin Winkel ^ 2 + cos Winkel ^ 2
    - `Math.sqare(Math.sin(Winkel))+`
      `Math.sqare(Math.sin(Winkel))`
  - Potenz meist nicht üblich (syntaktisch möglich) $(9^9)^9 = 9^{81}$, $9^{(9^9)}$
    - Basic, openCalc
    - 9^8^7 = 9^(8^7) (oft rechtsassoziativ)
- Bruchstrich als implizite Klammerung
  - $\frac{A+B}{C \cdot D} = (A+B)/(C \cdot D)$

    $(A+B)/C \cdot D = \frac{A+B}{C} \cdot D.$

Ausdrücke

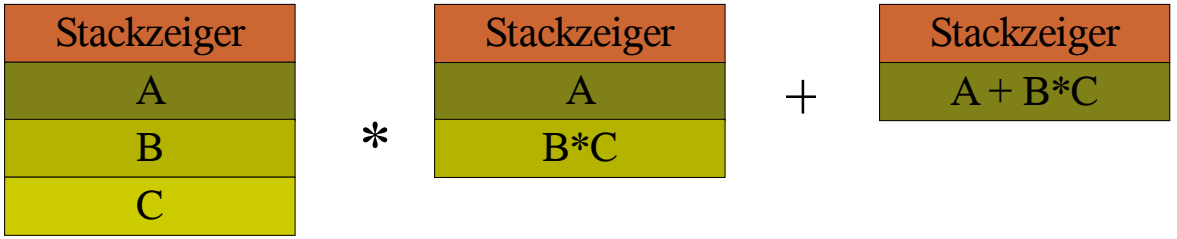
- Auswertender Operator
  - In Mathematik
    - $y = \begin{cases} -x & x < 0 \\ x & sonst \end{cases}$
  - In C, Java
    - <bedingung> ? <wert1> : <wert2>;
  - `y = x<0 ? -x : x;`
  - ```
    String result =
        number < 0
        ? "Result negative!"
        : "Result positive!";
    ```
    ```

Ausdrücke

- Postfix-Notation
  - auch umgekehrte polnische Notation nach dem polnischen Logiker Jan Łukasiewicz, 1878-1956
- In Mathematik Infix-Notation üblich
  - Operator zwischen Operanden
    - A + B
    - (A + B) * C
    - A + B * C
    - sin A * B ^ 2 / (2 * PI)
  - Vorrangregeln nötig (*, / vor +, - usw.)
    - Smalltalk
  - oder Klammern
  - Operator hinter Operanden wahren Eindeutigkeit
    - A B +
    - A B + C *
    - A B C * +
    - A sin B 2 ^ * 2 PI * /

Ausdrücke

- Postfix-Notation
  - In Informatik zur Auswertung von Ausdrücken verwendet
    - Java-Emulator
    - Keller-Automat
      - LR(k)-Sprachen
    - Stack-Architektur
      - A B C * +



Ausdrücke

- Aussagenlogik
  - „Junktorenlogik“ [Lorenzen70]

Par1	Par2	¬ Par1	Par1 ∧ Par2	Par1 ∨ Par2	Par1 ⊗ Par2	Par1 ⇒ Par2
		(not)	(and)	(or)	(xor)	(Wenn ... dann ...)
false	false	true	false	false	false	true
false	true	true	false	true	true	true
true	false	false	false	true	true	false
true	true	false	true	true	false	true
Java		!	&&	||	^	
Java			&	|		

Ausdrücke

- Aussagenlogik
  - Ausschließendes (*conditional*) Und in Java: **&&**
  - Einschließendes (*boolean*) Und in Java: **&**
  - Bitweises Und in Java: **&**
  - Ausschließendes (*conditional*) Oder in Java: **||**
  - Einschließendes (*boolean*) Oder in Java: **|**
  - Bitweises Oder in Java: **|**
  - Einschließendes (*boolean*) Xoder in Java: **^**
  - Bitweises Xoder in Java: **^**

Ausdrücke

- Aussagenlogik
  - Vergleich von Zahlenwerten

mathematische Notation	=	≠	<	>	≤	≥
informatische Notation	==	<>, ><, !=	<	>	<=, =<	>=, =>

  - FORTRAN
    - .EQ. entspricht ==
    - .NE. entspricht <>
    - .LT. entspricht <
    - .AND. entspricht and
  - Zeichen
    - 'A' < 'B'
    - 'A' < 'a'
    - '0' < '1'

Ausdrücke

- Aussagenlogik
  - Texte vergleichen
    - lexikalische Ordnung der Zeichen vom ersten zum letzten Zeichen der zu vergleichenden Wörter
    - Zeichen an gleicher Stelle verschieden bestimmen Ordnung
    - kürzeres Wort kleiner als längeres, wenn beide bis zur Länge der kürzeren Worts übereinstimmen
      - `"Hans" < "Karl"`
      - `"Helga" < "Hertha"`
      - `"Anne" < "Annemarie"`
      - `"Annamarie" < "Anne"`
    - In Java: `"Anne".compareTo("Annemarie")<0;`

Ausdrücke

- Bit-Operationen in Java

Operator	Definition	Priorität	Assoz.
~	Bitweises NOT Setzt für jedes Bit des Operanden den entgegengesetzten Wert ein, so dass 0 zu 1 wird und umgekehrt.	2	Rechts
<<	Linksverschiebung mit Vorzeichen. Verschiebt die Bits des linken Operanden um die im rechten Operanden angegebene Anzahl an Stellen nach links und füllt die freiwerdenden Stellen mit Nullen auf. High-Order-Bits gehen verloren.	6	Links
>>	Rechtsverschiebung mit Vorzeichen. Verschiebt die Bits des linken Operanden um die auf der rechten Seite angegebene Anzahl an Stellen nach rechts. Falls der linke Operand negativ ist, werden die freiwerdenden Stellen mit Einsen aufgefüllt, falls es sich um einen positiven Operanden handelt, mit Nullen. Dadurch bleibt das ursprüngliche Vorzeichen erhalten.	6	Links
>>>	Rechtsverschiebung und Auffüllung mit Nullen. Verschiebt nach rechts und füllt immer mit Nullen auf.	6	Links
&	Bitweises AND. Kann zusammen mit = verwendet werden, um den Wert zuzuweisen.	9	Links
|	Bitweises OR. Kann zusammen mit = verwendet werden, um den Wert zuzuweisen.	10	Links
^	Bitweises XOR. Kann zusammen mit = verwendet werden, um den Wert zuzuweisen.	11	Links
<<=	Verschiebung nach links mit Zuweisung	15	Links
>>=	Verschiebung nach rechts mit Zuweisung	15	Links
>>>=	Nullen)	15	Links

Anweisungsblöcke

- Anweisungsblöcke
  - Folge von Anweisungen zu Anweisungsblock zusammenfassen
    - gleiche Bedeutung wie eine einzelne Anweisung
    - Funktionalität entsprechend erweitert
  - beginnt und endet mit eigenem Schlüsselwort
    - **`begin Anweisungsfolge end`** — z.B. Algol60, Pascal, Ada
    - **`{ Anweisungsfolge }`** — C, C++, Java

Anweisungsblöcke

- Anweisungsblöcke
  - Anweisungsblock lässt sich Name zuweisen
    - andere Operationen einfacher ausführbar
    - **`Name: begin Anweisungsfolge end`** — Ada
    - **`Name: { Anweisungsfolge }`** — Java
      - (*labeled statement*), für **`break`** und **`continue`**

```
RowLoop: {
  for(int Row=0; Row < Rows.length; Row++) {
            // Java (labeled loop)
  ColLoop: for(int Col=0; Col < Columns.length; Col++) {
            // Another labeled loop
            break RowLoop;
} }
```

Anweisungsblöcke

- Anweisungsblöcke
  - Sprachen ohne Anweisungsblöcke
    - Fortran, BASIC (dialektabhängig), Assembler, 3AA
      - "Goto" zur Ablaufsteuerung
      - `DO 10 I=1,20` -- Die Schleife wird für Werte
      - ... -- I=1..20 durchlaufen.
      - `10 CONTINUE` -- 10 ist die Sprungmarke

Sprunganweisungen

- Unbedingtes Goto
  - Meist durch Schlüsselwort goto ausgewiesen
  - ```
    goto Ende;
    ...
    Ende: ...
    ```
  - Sprungmarke kann sein
    - Bezeichner (C, Algol, Ada) (ohne Deklaration)
    - Zahl (Label in Pascal) (zu deklarieren)
  - Viele Sprachen verbieten das Goto (Modula2, Java)
    - Java kennt Schlüsselwort **`goto`**, verwendet es aber nicht.
    ```

Sprunganweisungen

- Bedingtes Goto
  - Bedingung führt Sprung wahlweise aus
    - `if x<0 then goto Ende;`
  - Bei Alternativen Mehrdeutigkeiten möglich
    - ```
      if x<=0 then
        if x==0 then goto Anfang;
          else goto Restart;
      ```
  - Zwei Interpretationsmöglichkeiten (*dangling else*)
    - ```
      if x<=0 then
        {if x==0 then goto Anfang;
            else goto Restart;}
      ```
  - oder
    - ```
      if x<=0 then
        { if x==0 then goto Anfang;}
      else goto Restart;
      ```
  - Erste Interpretation üblich!
  - Bedingung durch Schlüsselwort abgrenzen (nicht in C, Java, Pascal usw.)
      ```

Sprunganweisungen

- Bedingtes Goto
  - ```
    if Bedingung1
      then Anweisungsfolge1
      else
      if Bedingung2
        then Anweisungsfolge2
        else
        if Bedingung3
          then Anweisungsfolge3
          else
          if Bedingung4
            then Anweisungsfolge4
            else Anweisungsfolge5
          endif
        endif
      endif
    endif ;
    ```
    ```

Sprunganweisungen

- Bedingtes Goto
  - Mehrfache Alternative
    - ```
      if     Bedingung1 then Anweisungsfolge1
      elseif Bedingung2 then Anweisungsfolge2
      elseif Bedingung3 then Anweisungsfolge3
      elseif Bedingung4 then Anweisungsfolge4
                        else Anweisungsfolge5
      endif ;
      ```
  - Guarded Command: Alle wahren Bedingungen ausführen
    - ```
      guard
        Bedingung1 do Anweisungsfolge1 done;
        Bedingung2 do Anweisungsfolge2 done;
        ...
        BedingungN do AnweisungsfolgeN done;
      endguard;
      ```

Sprunganweisungen

- Bedingtes Goto
  - Guarded Command in Ada
    - ```
      select
        accept Wert auf erstem Kanal eingetroffen ...
      or
        accept Wert auf zweitem Kanal eingetroffen ...
      else
        sonst tue etwas anderes
      end select;
      ```
  - Prozesskommunikation
    - Auf welchem Kanal trifft Signal ein?
    - synchrone Kommunikation
      - (i.d.R. sehr umständlich – CSP von Hoare)
    - asynchrone Kommunikation verwendet Puffer
      ```

Sprunganweisungen

- Bedingtes Goto
  - Alternative mit konstanten Werten
    - ```
      if  Zeichen == 'A' then Anweisungsfolge1
      elseif Zeichen == 'B' then Anweisungsfolge2
      elseif Zeichen == 'C' then Anweisungsfolge3
      elseif Zeichen == 'D' then Anweisungsfolge4
      else Anweisungsfolge5
      endif ;
      ```
  - case/switch-Statement
    - ```
      switch(Zeichen) {
        case 'A': Anweisungsfolge1; break;
        case 'B': Anweisungsfolge2; break;
        case 'C': case 'f': Anweisungsfolge3; break
        case 'D': Anweisungsfolge4; break;
        default: Anweisungsfolge5; break;
      }
      ```
    - i.d.R. effizienter als if then else if ...

Sprunganweisungen

- Bedingtes Goto
  - Entscheidungstabelle (*decision table*)
    - *Chill* (*CCITT High Level Language*)

```
case Zeichen, Nummer of
  'A', 1..4: Anweisungsfolge1;
  'B', 2..5: Anweisungsfolge2;
  'C', 1   : Anweisungsfolge3;
  'D', 1..3: Anweisungsfolge4;
  others     Anweisungsfolge5;
end case ;
```

  - Entspricht dem Konzept der Entscheidungstabelle in COBOL

	Zeichen=='A'	Zeichen=='B'	Zeichen=='C'	Zeichen=='D'
Nummer=1	Anwfolge1	Anwfolge5	Anwfolge3	Anwfolge4
Nummer=2	Anwfolge1	Anwfolge2	Anwfolge5	Anwfolge4
Nummer=3	Anwfolge1	Anwfolge2	Anwfolge5	Anwfolge4
Nummer=4	Anwfolge1	Anwfolge2	Anwfolge5	Anwfolge5
Nummer=5	Anwfolge5	Anwfolge2	Anwfolge5	Anwfolge5

Wiederholungen

- Wiederholte Abläufe des gleichen Programms
  - In Maschinensprache nur mit **goto** ausdrückbar
    - ```
              adr fak := con 1
              adr zahl := con 1
      Weiter  if val zahl > val N then goto Ende
              adr fak *= val zahl
              adr zahl += con 1
              goto con Weiter
      ```
  - In modernen Sprachen durch höhere Konstrukte ausgedrückt
    - Algol 68
    - ```
      for Var from Ausdr1 by Ausdr2 to Ausdr3
         while Bedingung do
        ... // Schleifenrumpf
      od;
      ```

Wiederholungen

- Algol 68
  - Bis auf **`do .. od`** darf alles wegfallen
    - `do ... od;`
    - `for Var from 0 by 5 to 100 do .. od;`
    - `to 256 do ... od;`
    - `for Var to 99 do ... od;`
    - `while ref!=null do .. od;`
  - In Java
    - `for(;;) ..`
    - `for(int var=0; var<=100; var+=5) ..`
    - `for(int var=1; var<=256; var++) ..`
    - `for(int var=1; var<=99;  var++) ..`
    - `for(; ref!=null; ) ..`
    - `while(ref!=null) ..`

Wiederholungen

- Allgemeine Schleife
  - Name für **`for`**-Schleife
    - `OUTER:`

```
{ int count = 10;     // Prolog
  if (count == 9)     break OUTER;
  INNER: for (;;) {
    for(int i=0, j=5; i<55&&j!=13; i++, j+=7)
    if( i!=2) {
      count--;
      if (count == 0)       break INNER;
      if (count == -22)     break OUTER;
      if (count%3 == 0) continue INNER;
  } }
  // Epilog
} // end OUTER
```

Ausnahmebehandlung

- Ausnahmebehandlung
  - Fehler beim Ablauf eines Programms
    - Eine Operation mit Zahlen liefert einen Wert, der in dem deklarierten Zahlenbereich nicht mehr darstellbar ist.
    - Zugriffe in case–Anweisungen finden keine passende Konstante.
    - Der Feldzugriff liegt außerhalb des definierten Bereichs.
    - Ein Zeiger greift auf einen nicht zugeteilten Speicher zu.
    - Der dynamisch angeforderte Speicher reicht nicht aus.
    - Es werden undefinierte Operationen durchgeführt, z.B.
      - Division durch null,
      - Logarithmus von negativen Zahlen,
      - Tangens von 90˚.

Ausnahmebehandlung

- Fehlerbehandlung durch
  - Abbruch des Programmlaufs
  - Ignorieren
  - Sonderwert (**`Infinity`**, **`NaN`**)
  - Besser: Fehler systematisch behandeln und spezifizieren
    - Fehler explizit vom Programmierer abfragen
      - `TanWin = Winkel <> 90 ? tan(Winkel) : GroeZahl;`
      - `if(Index>=0&&Index<Feld.length)Feld[Index]=23;`
      - `if(Zeiger!=null) Zeiger = Zeiger.next;`
    - umständlich
    - fehleranfällig
    - nicht immer möglich
      - Wertebereichsüberlauf
      - Speicheranforderung

Ausnahmebehandlung

- Ausnahmebehandlung in Java™
  - ```
    try  {
       ... // Code, der Exception auslösen kann
    }
    catch( Exception1 e1 ) {
       ... // Exception-Behandlung.
    }
    catch( Exception2 e2 ) {
       ... // Exception-Behandlung.
    }
    finally {
       ... // Dieser Code wird immer ausgeführt
    }
    ```
  - **`Exception`** ist eine Klasse
    - **`java.lang.Throwable.Exception`**
    - **`java.lang.Throwable.Error`**
    ```

Ausnahmebehandlung

- Ausnahmebehandlung in Java™
  - **`Exception`** ist eine Klasse
  - Es gibt standardisierte Ausnahme-Klassen
    - **`java.lang.Throwable.Exception`**
      - **`IOException.WriteAbortedException`**
      - **`EOFException`**
      - **`ArrayIndexOutOfBoundsException`**
      - **`u.v.a.`**
  - Es gibt standardisierte Fehler-Klassen
    - **`java.lang.Throwable.Error`**
      - **`OutOfMemoryError`**
      - **`StackOverflowError`**
      - **`ClassFormatError`**
      - **`u.v.a.`**

Ausnahmebehandlung

- Ausnahmebehandlung in Java™
  - Jede Ausnahme enthält einen String
    - ```
      catch(Exception e) {
        System.out.println(e.getMessage());
        System.out.exit(1); // Exception-Behandlung.
      }
      ```
  - Ausnahmen können definiert werden.
    - ```
      class MyException extends IOException {
        String myExceptionText = "Das war nichts!";
        public MyException(String s) { // Konstruktor
          this.myExceptionText = s;
        }
        public void getMessage() {     // Überlade
          return myExceptionText;
      } }
      ```

Ausnahmebehandlung

- Ausnahmebehandlung in Java™
  - Java unterscheidet zwischen *checked* und *unchecked* Exceptions
    - checked Exceptions werden vom Compiler überprüft
    - unchecked Exceptions werden nicht vom Compiler überprüft
  - unchecked Exceptions müssen nicht abgefangen werden
    - Meistens Systemabbruch, wie **`Errors`**
      - **`OutOfMemoryError`**
    - Oder
      - **`RuntimeException`**
  - checked Exceptions müssen explizit abgefangen werden
    - durch einen **`try{..}catch(..){..}`**-Block
    - durch throws
      - ```
        String myMethod throws IOException (..) {
          BufferedReader in = new BufferedReader(..);
          if(..) throw new MyException("ExceptionText");
          return in.readLine();
        }
        ```
        ```

Ausnahmebehandlung

- Ausnahmebehandlung in Java™
  - ```
    long LongValue; double DoubleValue;
    boolean IsLong = true, IsDouble = true;
    try { LongValue =
       Long.parseLong(Parameter1.getText());
    } catch(Exception e){ LongValue = 0;
                          IsLong = false;
    }
    try { DoubleValue =
      Double.parseDouble(Parameter1.getText());
    } catch(Exception e){ DoubleValue = 0;
                          IsDouble=false;
    }
    ```
    ```