```
Implementierung einer (einfachen) Programmiersprache
François Bry
30.3.2009

-------------------------------------
FOLIEN:
Implementierung einer (einfachen) Programmiersprache
François Bry
-------------------------------------

Für 10.-12. Jahrgangsstufen (16-18 Jahre),
Dauer: 90 mn

Vorsicht: Kennen lernen wichtiger als Stoff!

Vorbemerkungen (5 mn)

Vorstellung:
   - Name, Alter, Verheiratet, eine 3-jährige Tochter
   - Mathematikstudium und -promotion in Paris
   - zufällig zur Informatik (in der Industrie) gekommen
   - Stelle bei einem industriellen Forschungszentrum in München
   - nach der Schließung des Forschungszentrums Professur an der LMU
     München
   - Interesse:
       - Webinformationssystemen insbesondere soziale Medien (wie wiki,
         digitale Netzwerke, social bookmarking)
       - automatisches Schließen
       - alles neue in der Informatik
       - Deutsche Sprache und Kultur
       - Ostasien insbesondere Japan
       - Wirtschaft und Politik
   - mehr über mich: Google: bry
   - nicht scheuen, meine Sprechstunde zu besuchen

Vorbemerkungen (5 mn):
   - Programmiersprachen sind unumgängliche Werkzeuge der Informatik.
     Die Vorlesung soll einen Eindruck von ihrer Realisierung geben
   - Welche Programmiersprachen kennen Sie?

-------------------------------------
FOLIEN
   - Heutige Vorlesung:
      1. Rechnen mit einem Keller (10 mn)
      2. Variablen (15 mn)
      3. Wiederholungsschleife (10 mn)
      4. Einfache Prozeduren (30 mn)
      5. Verbesserung des Prozedurkonzeptes (10 mn, eventuell auslassen)
      6. Schlussbemerkungen (10 mn)
-------------------------------------

---------------------------------------------------------------------
1. Rechnen mit einem Keller (10 mn)

-------------------------------------
FOLIEN
1.1 Postfix-Notation
1.2 Keller zum Rechnen
1.3 Rechnen mit Boole'schen Werten
1.4 Abstraktion
-------------------------------------

1.1 Postfix-Notation
```

```
1 + (2 * 3) ausdruckbar als 1 2 3 * +
   postfix
   klammerfreie oder lineare Sprache
   eindeutige Sprache wenn die Stelligkeiten der Operationen festgelegt
   sind

(-1) + (2 - 3) ausdruckbar als 1 ~ 2 3 - +
(~ zur Darstellung der einstelligen Minusoperation, und - zur Darstellung
der zweistelligen Minusoperation

(1 + (2 * 3)) - ((4 * 5) + 6) ausdruckbar als 1 2 3 * + 4 5 * 6 + -

Die Reihenfolge der Operationen, die die Klammerung ausdruckt, geht nicht
verloren:

((1 - 2) - 3) (= ~4) ausdruckbar als 1 2 - 3 -
(1 - (2 - 3)) (= 2)  ausdruckbar als 1 2 3 - -

1.2 Keller zum Rechnen

Der Hauptspeicher ist ein lineares direktadressierbares Bereich, d.h. man
kann kostengünstig zu (u.a. nächsten oder zur vorherigen) Speicherzelle
gelangen. Die folgende Berechnungsweise nutzt dies aus.

1 2 3 * +  (entspricht: (1 + (2 * 3)) )

1  -*-> 1  -+-> 7
2       6
3

1 2 3 * + 4 5 * 6 + -  (entspricht: (1 + (2 * 3)) - ((4 * 5) + 6) )

1   1  7   7   7   7   7  -19
2   6      4  20  20  26
3          5       6

1 2 3 * + ---> pushlit 1,  pushlit 2, pushlit 3, op *, op +

push
literal

Kellerverwaltung mit einem Register T (Top of stack):

2 3 *

 T -> 0: 2         0: 2     T-> 0: 6
      1: ?     T-> 1: 3         1: 3
      2: ?         2: ?         2: ?

Weitere Register:
   - I (Instruction, Codeadresse)
   - B (Base of segment, Kelleradresse)

Erstes Befehl zur Initialisierung: reset (T:= 0; I := 0; B:= 0)

Keller dient als Schmierpapier für Zeischenberechnungen
Die Speicherbereinigung ist dabei eingebaut
Klammerfreie oder lineare Befehlsprache
      klammerfrei = durchführbar ohne Berücksichtigung eines
      Kontextes, also Befehlsprache

1.3 Rechnen mit Booles'schen Werten

(true or ((false and true) or false)) ---> true false true and false or or
```

```
ausdrucktbar als
reset, pushlit true, pushlit false, pushlit true, op and, pushlit false,
op or, op or

1.4 Abstraktion

Jedes Wert belegt hier eine Speicherzelle - in der Realität ist es anders
Abstrakte vs. reale Befehlsprache und abstrakte Maschine

-----------------------------------------------------------------
2. Variablen (10 mn)

Variablen sind nutzlich, um Algorithmen unabhängig von bestimmten Werten
auszudrücken.

2.1 Programmbeispiele

Berechnung des arithmetischen Durchschnitts zweier Werte:

-----------------------------------
FOLIEN
   var Wert1;
   var Wert2;
   var arithmetischerDurchschnitt;
   arithmetischerDurchschnitt := (Wert1 + Wert2) / 2;
-----------------------------------
FOLIEN
Vertauschen der Werte zweier Variablen:

   var Var1;
   var Var2;
   var Hilfsvariable;
   Hilfsvariable := Var1;
   Var1 := Var2;
   Var2 := Hilfsvariable;
-----------------------------------
FOLIEN
Eine weitere Berechnung:

   var x;
   var y,
   x := (1 + 2) * 3;
   y := 2;
   x := x + y;
-----------------------------------

:= Zuweisung
Konstanten und Variablen

2.3 Darstellung von Variablen auf dem Keller

Durchführung des letzten Programms mit einen Keller:

Symboltabelle:   Keller:      Befehle zur Darstellunf auf dem Keller:

x: 0                0:        inc 1
y: 1           T-> 1:        inc 1

var x; ---> inc 1 (während dieser Übersetzung Eintrag von x in die
                   Symboltabelle)

var y; ---> inc 1 (während dieser Übersetzung Eintrag von y in die
                   Symboltabelle)
```

```
x :=  ((1 + 2) * 3)  -->
              pushlit 1, pushlit 2, op +, pushlit 3, op *, store 0 0
y := 2  ----> pushlit 2, store 0 1

store 0 0: Erste 0 bedeutet 1. Segment (momentan nur 1 Segment folglich
momentan irrelevant)

-----------------------------------
FOLIEN
Symboltabelle
Befehle:
   - reset
   - inc 1
   - pushlit
   - store 0 1
   - op *
-----------------------------------

2.4 Benannte Konstanten

   nutzlich um Werte auszudrücken
      - die im Algorithmus unverändert bleiben
      - wovon jedoch der Algorithmus abhängt
   eine Änderungen einer Konstant findet nur an einem Ort im
   Quellprogramm statt
      - Verständlichkeit erhöht
      - Fehlerrisiko bei Änderungen vermindert

Fiktives Beispiel, in dem eine kindesbezogene Steuerentlastung
komplementär zum Steuersatz ist:

-----------------------------------
FOLIEN
   const Steuersatz := 25%;
   const Kindergeld := 1848;

   var Einnahmen;
   var Kinderzahl;
   var absetzbareKosten;
   var Steuerschuld;
   var kindesbezogeneSteuerentlastung;

   kindesbezogeneSteuerentlastung :=
          (Kinderzahl * Kindergeld * (100 - Steuersatz));
   Steuerschuld :=
          (((Einnahmen - absetzbareKosten) * Steuersatz) -
          kindesbezogeneSteuerentlastung);
-----------------------------------

Benannte Konstanten werden wie Variablen übersetzt, außer dass bei der
Übersetzung überprüft wird, dass im Quellprogramm ihnen keine Werte
zugewiesen werden.

Quellprogramm
Zielprogramm oder Code
Codesprache oder (abstrahierte) Prozessorsprache

2.5 Typ einer Variable oder Konstante

Der Typ (ganze Zahl, reale Zahl, Boole'sche Wert, usw. wird unterschieden)

-----------------------------------
FOLIEN
```

```
   const real Steuersatz := 0.25;
   var integer Kinderzahl;
   var boolean steuerpflichtig := true;
-----------------------------------

Typrüfung
   - statisch
   - dynamisch
Typanpassung
   - implizit
   - explizit

-----------------------------------------------------------------
3. Wiederholungsschleife (10 mn)

-----------------------------------
FOLIEN
   var Kapital;
   var Zinssatz; (* Dezimalwert *)
   var Dauer;
   var i := 1;   (* Schleifenindex *)

   read Kapital;
   read Dauer;
   while (i =< Dauer)
      { Kapital := Kapital * (1 + Zinsatz); i := i + 1;  }
   write Kapital;
-----------------------------------
ZUERT AN DER TAFFEL
FOLIEN
Übersetzung:

   0: inc 1          Symboltabelle: Kapital:  0
   1: inc 1                         Zinssatz: 1
   2: inc 1                         Dauer:    2
   3: inc 1                         i:        3

   4: pushlit 1
   5: store 0 3

   6: read
   7: store 0 0
   8: read
   9: store 0 2
  10: goto 21              Unbedingter Sprung zur Schleifenbedingung

  11: push 0 0             Schleifenrumpf
  12: pushlit 1
  13: push 0 1
  14: op +
  15: op *
  16: store 0 0
  17: push 0 3
  18: pushlit 1
  19: op +
  20: store 0 3

  21: push 0 3             Schleifenbedingung
  22: push 0 2
  23: op =<
  24: jumpontrue 11         Bedingter Sprung zum Schleifenrumpf

  24: write 0 0
-----------------------------------
```

```
Wird die Schleife n Mal durchgelafen, so finden n+1 Sprünge statt.
(Eine unmittelbare Übersetzung egibt 2n + 1 Sprünge.)

Sprungbefehle in Prozessorsprachen:
   - goto (unbedingter Sprung)
   - jumpontrue (bedingter Sprung)
   - jumponfalse (bedingter Sprung)

Weitere Wiederholungschleifen:
   - Until-Schleifen
   - For-Schleifen

-----------------------------------------------------------------
4. Einfache Prozeduren (30 mn)

Einfache Prozeduren bedeutet:
   - keine Aufrufparameter
   - keinen gelieferten Wert
   - keine Rekursion

Prozeduren dienen zum Ausdruck von Teilberechnungen
   - der Lesbarkeit halber
   - der Wiederverwendbarkeit halber
   - der Programwartung halber

4.1 Beispiel eines Programms mit Prozeduren

-----------------------------------
FOLIEN
program Schuldenrechner
   { var Schuld;
     var Zinssatz;
     var Dauer;
     procedure Schuldberechnung
        { var Zinsen := 0;
          procedure Zinsenberechnung
              { var Index := 1;
                while (Index =< Dauer)
                   { Zinsen := (Schuld * Zinssatz);
                     Index := Index + 1; } }
           call Zinsenberechnung;
           Schuld := Schuld + Zinsen; }
     read Schuld;
     read Zinssatz;
     read Dauer;
     call Schuldberechnung;
     write Schuld; }
-----------------------------------

Blockstruktur
Zinsenberechnung ist lokal zur Prozedur Schulberechnung

4.2 Code für das Programmbeispiel

-----------------------------------
ZUERST AN DER TAFFEL
FOLIEN
   0: reset
   1: inc 1      in die Symboltabelle: Schuld:   0  (Kelleradresse)
   2: inc 1                            Zinssatz: 1
   3: inc 1                            Dauer:    2
   4: goto 30    Sprung zum Programmrumpf
```

```
                                              Umgebung:
                                                 Schuldberechnung 5
   5: inc 1      in die Symboltabelle  Zinsen:   0
   6: goto 24    Sprung zum Prozeduranfang

                                               Umgebung:
                                                 Zinsenberechnung: 8
   7: inc 1      in die Symboltabelle  Index:    0
   8: pushlit 1:
   9: store 0 0  1. Segement relativ zum derzeitigen Segment: rückwärts)
  10: goto 19    Sprung zum Schleifenbedingung 19

  11: push 2 0   Schleifenrumpf         push 2 0: 2. Segment höher, 1. Zelle
  12: push 2 1
  13: op *
  14: store 1 0
  15: push 0 0
  16: pushlit 1
  17: op +
  18: store 0 0

  19: push 0 0   Schleifenbedingung
  20: push 2 2
  21: op =<
  22: jumpontrue 13  Sprung zum Schleifenrumpf
  23: return     aus der Symboltabelle: Index:   0
                 return: I erhält den im Segementdeskriptor gereteten Wert
                         T erhält den im Segementdeskriptor gereteten Wert
                            (somit wird das Prozedursegment vom Keller
                            entfernt)

  24: call 8     Rettung der gegenwertigen Werten von I und B und:
                    B := T, I := 8 (Sprung zum Prozeduranfang)
  25: push 1 0
  26: push 0 0
  27: op +
  28: store 1 0
  29: return     aus der Symboltabelle: Zinsen: 0     aus der Umgebung:
                                                    Zinsenberechnung: 8

  30: read 0 0   Progarammrumpf
  31: read 0 1
  32: read 0 2
  33: call 5     Rettung der gegenwertigen Werten von I und B und:
                    B := T, I := 5 (Sprung zum Prozeduranfang)
  34: write 0 0
  35: halt
-------------------------------------
```

Befehl halt nötig, damit nicht nach Ende vom Zielprogramm weiter gelesen wird.

Prozedursegment
- Segmentdescriptor: um beim call
  - Wert von I zu retten
    (Zur Fortsetzung der Berechnung nach dem Prozeduraufruf)
  - Wert von B zu retten
    (Kelleradresse des Segmentanfanges, damit die Vartiablen gefunden werden)
  - neuer Wert von B: B := T
    (Kellkeradresse des Segmentanfanges, damit die Vartiablen gefunden werden)
- Rest des Segements:
  - Keller für die Berechnungen des Prozedurrumpfes

```
Symboltabelle und Umgebung werden nur währen der Übersetzung verwendet

Überschatten eingebaut
   - was ist Überschatten
   - warum es eine gute Sache ist

-----------------------------------------------------------------
5. Verbesserung des Prozedurkonzeptes (10 mn, eventuell auslassen)

-----------------------------------
FOLIEN
5.1 Aufrufparameter
5.2 Funktionen
5.3 Rekursion
-----------------------------------

5.1 Aufrufparameter:

-----------------------------------
FOLIEN
program Schuldenrechner
   { var Schuld;
     var Zinssatz;
     var Dauer;
     procedure Schuldberechnung(S, ZS, D)
        { var Zinsen := 0;
          procedure Zinsenberechnung(S1, Z1, ZS1; D1)
             { var Index := 1;
               while (Index =< D1)
                  { Z1 := (S1 * ZS1);
                    Index := Index + 1;} }
             call Zinsenberechnung(Zinsen, ZS, D);
             S:= S + Zinsen; }
     read Schuld;
     read Zinssatz;
     read Dauer;
     call Schuldberechnung(Schuld, Zinssatz);
     write Schuld; }
-----------------------------------

Realisierung:
   - Prozedurparameter werden wie Variablen behandelt
   - Zwei Ansätze
        - call by name: keine neue Zelle für einen Aufrufparameter
        - call by value: neue Zelle, die den Wert des Aufrufparameters
          erhält
        - pros und cons?
             call by name natürlich für Vertauschen(x, y)
             call by value natürlich für eine Schuldberechnung bei
             optionalen Zinsen

5.2 Funktionen

Die folgende Schreibweise ist natürlich.

-----------------------------------
FOLIEN
    Schuld := Schuld + Zinsenberechnung(Schuld, Zinssatz, Dauer);
-----------------------------------

Realisierung:
   der Wert des Aufrufes belegt die 1. Zelle des Prozedursegementes
```

```
2.3 Rekursion

Eine natürliche form der Wiederholung:

   Fakultaet(n) := if n = 1 then 1
                           else n * Fakultaet(n - 1);

Realisierung:
    Problem:
       da die Anzahl der rekursive Aufrufe zur Übersetzungzeit unbekannt
       ist, können Variablen, die nicht lokal zur rekursiven Prozedur nicht
       wie bei den einfachen (nicht-rekursiven Prozeduren) behndelt werden.
    Lösung:
       static link: in jedem Segementdeskriptor wird die Adresse des
       Segmentes der Ugebenden Prozedur im Quellprogramm behalten. Diese
       Adresse ist bei einem rekursiven Aufruf dieselbe wie beim aufrufenden
       Segement.

-----------------------------------------------------------------
6. Schlussbemerkungen (10 mn)

-----------------------------------
FOLIEN
- Es gibt viele Programmierparadigmen
- Es gibt noch mehr Programmiersprachen und es werden noch sehr viele
  kommen:

    - Die Programmiersprachen des Studiums werden nicht lange diejenigen
      des Berufes sein
    - wesentliche Veränderungen finden statt (etwa "web programming
      framework")
    - Beispiel einer ungeöhnlichen Programmiersprache (z.B. wie ein Kreis
      damit gebildet wird):
      Mitchel Resnick: Turtles, Termites, and Traffic Jams, MIT  Press,
      1997
-----------------------------------
FOLIEN
- Ein Programm ist für Menschen, nicht für Computer.
- Wer nicht programmiert, kann kein fähiger Informatiker sein.
- Eleganz ist in der Programmierung nicht optional.
- Man programmiert erst dann gut, wenn man weiß, wie die Sprache implementiert ist.
-----------------------------------
```