Vorstellung des LW-Simulators

Inhalt

- Die LW-Sprache
  - Geläufiges
  - Pre-execution commands
  - Makros
    - Aufbau von Makros
    - Verwendung von Makros
    - Wichtiges
- Vorführung des Tools

Geläufiges

- Zahlen:
  - Es gibt nur eine Konstante: *0*
  - Andere Zahlen werden durch Anhängen von *succ* gebildet:

    z.B. *succ(0)* ist *1*, *succ(succ(0))* ist *2,* ...

  - Implementierte Sprache bietet zusätzlich *pred*:

    pred(succ(x)) ist *x; pred(0)* ist *0*

Geläufiges

- Variablen sind *x0*, *x1*, …, *xi*
- Zuweisungen:

  xa := Wert

  (Wert kann Zahl oder Variable sei)
- Anweisungen:
  - Sind Zuweisungen oder Schleifen
  - Zwischen zwei Anweisungen ist ein Semikolon

    ⇒ Letzte Anweisung endet nicht mit Semikolon

Geläufiges

Beispiel für Anweisungen

```
3
4       x0 := 0;
5       x1 := succ( 0 );
6       x2 := succ( succ( 0 ) );
7       x3 := x1;
8       x4 := succ(x1 );
9       x2 := pred (x2 );
10      x0 := succ( pred ( pred(x4) ) );
11      x0 := pred( pred(x0 ) )
```

Geläufiges

- Schleifen:
  - *loop* Wert *do* P *end*

    Führe P Wert-mal aus

  - *while* Variable *do* P *end*

    Führe P solange aus, wie Variable nicht 0 ist.

Geläufiges

- Beispiel für Schleifen

```
3       loop succ(0) do
4           x1 := succ(x1)
5       end;
6
7       loop x1 do
8           x0 := succ(0)
9       end;
10
11      while x1 do
12          x0 := succ(0);
13          x1 := pred(x1)
14      end
```

Geläufiges

- Code bekommt Eingabevariablen
- Ergebnis soll am Ende in Variable x0 stehen

Pre-execution commands

- *#LOOP* / *#WHILE*:
  - Muss als allererster Befehl immer in jedem Code stehen
  - *#LOOP* erlaubt nur LOOP-Schleifen im Code
  - *#WHILE* erlaubt beide Sorten von Schleifen
  - notwendig

Pre-execution commands

- *#MAXREG* [Zahl]:
  - fordert eine gewisse Anzahl von Variablen an:
    - Ist dies mehr als der Benutzer erlaubt, startet der Code nicht
    - Ist dies nur mehr als Eingabevariablen, so füllt der Simulator die Variablen auf
    - Ansonsten keine Warnung oder Fehler
  - stellt sicher, dass bei Ausführung Variablen *x0* bis *x*(Zahl-1) bereit stehen
  - nicht notwendig

Pre-execution commands

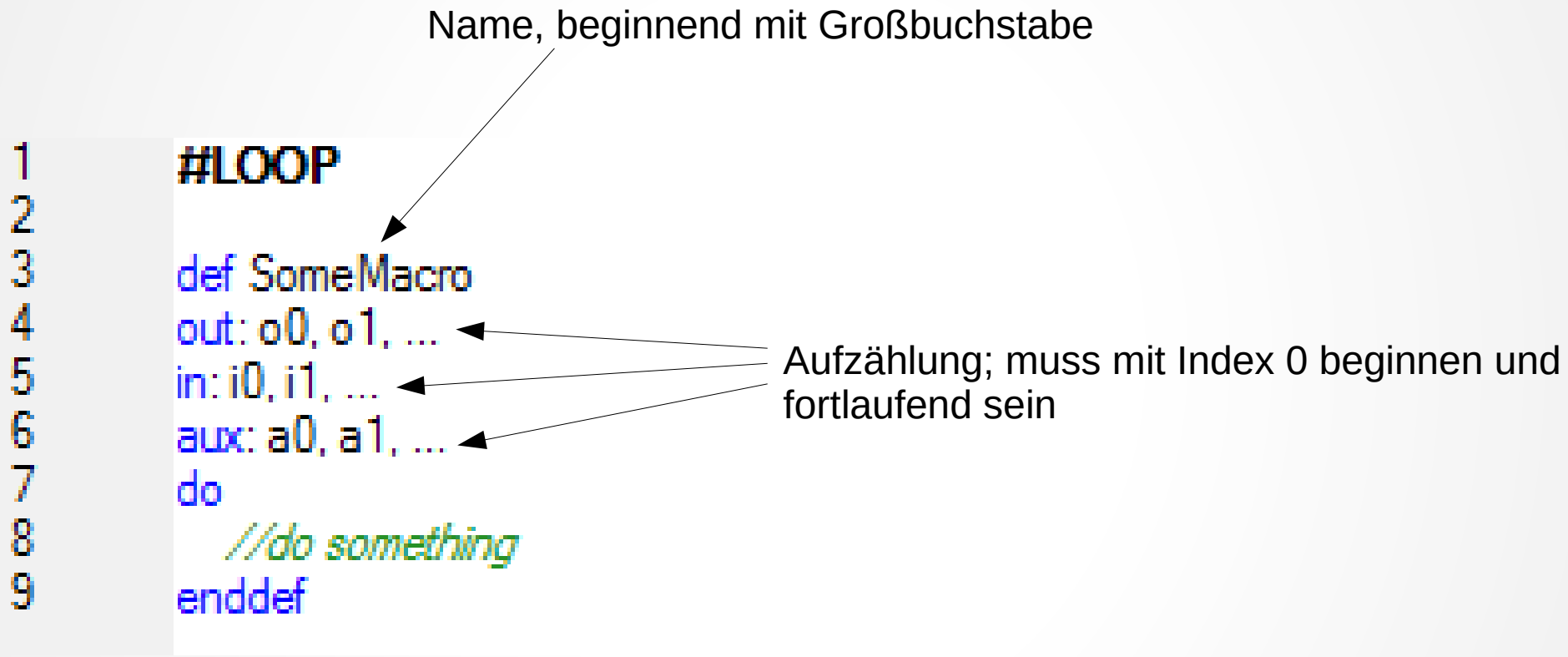
- *#IMPORT* [Datei]:
  - Lädt alle Makros, die in Datei enthalten sind
  - Wenn der derzeitige Code *#LOOP* enthält, darf Datei nicht *#WHILE* enthalten
  - Datei wird im aktuellen Verzeichnis gesucht, dann als absoluter Pfad
  - nicht notwendig

Pre-execution commands

Beispiel:

```
1    #LOOP
2    #MAXREG 6
3    #IMPORT OneImport
4    #IMPORT 2ndImport.txt
```

Makros - Aufbau



WICHTIG: Makros stehen immer vor dem eigentlichen Programm

Makros - Aufbau

Beispiele

```
1   #WHILE
2
3   def Greater
4   out: o0
5   in: i0,i1
6   aux:
7   do
8      o0 := i0;
9      loop i1 do
10        o0 := pred(o0)
11     end
12  enddef
13
14  def Xth_Neighbor
15  out: o0,o1
16  in: i0,i1
17  aux: a0
18  do
19     o0 := i0;
20     o1 := i0;
21     a0 := i1;
22     while a0 do
23        o0 := pred(o0);
24        o1 := succ(o1);
25        a0 := pred(a0)
26     end
27  enddef
```

Makros - Verwendung

- Makros als Zuweisungen:

```
31
32      (x2,x3) := Xth_Neighbor(x0,x1)(x4);
33
```

Outputvariablen Inputvariablen Hilfsvariablen

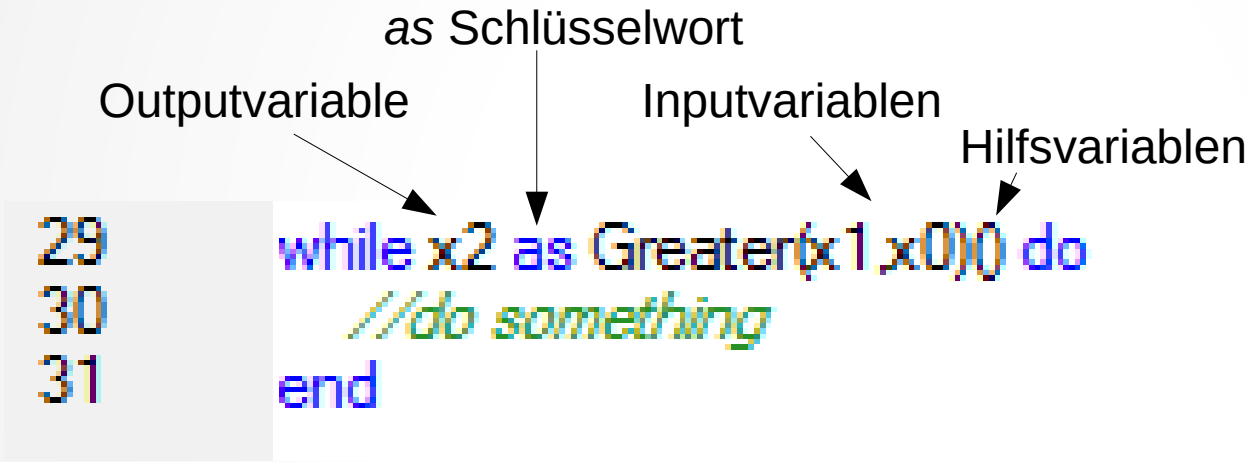
Klammern können weggelassen werden, wenn nur ein Outputvariable

```
26
27      x1 := Hundert()();
28
```

```
1       #WHILE
2
3       def Greater
4       out: o0
5       in: i0,i1
6       aux:
7       do
8          o0 := i0;
9          loop i1 do
10            o0 := pred(o0)
11         end
12      enddef
13
14      def Xth_Neighbor
15      out: o0,o1
16      in: i0,i1
17      aux: a0
18      do
19         o0 := i0;
20         o1 := i0;
21         a0 := i1;
22         while a0 do
23            o0 := pred(o0);
24            o1 := succ(o1);
25            a0 := pred(a0)
26         end
27      enddef
```

Makros - Verwendung

- Makros als Ausdruck




```
1   #WHILE
2
3   def Greater
4   out: o0
5   in: i0,i1
6   aux:
7   do
8      o0 := i0;
9      loop i1 do
10        o0 := pred(o0)
11     end
12  enddef
13
14  def Xth_Neighbor
15  out: o0,o1
16  in: i0,i1
17  aux: a0
18  do
19     o0 := i0;
20     o1 := i0;
21     a0 := i1;
22     while a0 do
23        o0 := pred(o0);
24        o1 := succ(o1);
25        a0 := pred(a0)
26     end
27  enddef
```

Praktisch kann jedes Makro mit nur einer Outputvariable als Ausdruck verwendet werden

Makros - Verwendung

Beispiel

```
1   #WHILE
2
3   def Greater
4   out: o0
5   in: i0,i1
6   aux:
7   do
8      o0 := i0;
9      loop i1 do
10        o0 := pred(o0)
11     end
12  enddef
13
14  def Xth_Neighbor
15  out: o0,o1
16  in: i0,i1
17  aux: a0
18  do
19     o0 := i0;
20     o1 := i0;
21     a0 := i1;
22     while a0 do
23        o0 := pred(o0);
24        o1 := succ(o1);
25        a0 := pred(a0)
26     end
27  enddef
28
29  x0 := succ(succ(succ(0)));
30  x1 := succ(0);
31
32  (x2,x3) := Xth_Neighbor(x0,x1)(x4);
33
34  x0 := 0;
35
36  while x4 as Greater(x3,x0)() do
37     x0 := succ(x0)
38  end
```

Eigentlichen Programm

Makros - Wichtiges

- Makros haben keinen Stack

  ⇒ Werte werden direkt in die Variablen geschrieben

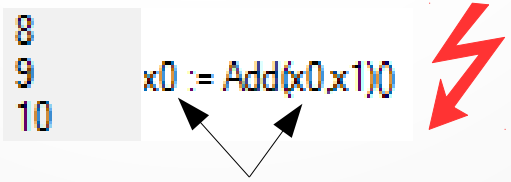
  ⇒ Es können keine neuen Variablen erzeugt werden

  Beispiel: Ich rufe aus einem Makro ein weiteres auf, dass mehr Variablen benötigt.

  Lösung: Bereits im hierarchisch höheren Makro zusätzliche Hilfsvariablen anfordern und weitergeben

- Inputvariablen dürfen nicht modifiziert werden

  ⇒ Inputvariablen können nicht gleichzeitig Output- oder Hilfsvariablen sein



Vorführung des Tools

- Das Tool wurde in C# geschrieben
  - ⇒ auf Windows kein Problem
  - ⇒ Linux / MacOS benötigen Mono
    - http://www.mono-project.com/

Vorführung des Tools

- Das Tool findet sich hier (samt ReadMe):

  https://cal8.cs.fau.de/redmine/projects/lw-simulator

- Nur die Binaries sind hier:

  https://cal8.cs.fau.de/redmine/projects/lw-simulator/repository/revisions/master/show/LWSimulatorCons/bin/Release

  (Wichtig sind MainLWSimulator.dll und LWSimulatorGUI.exe)