Übersetzung von Datenstrukturen

- **bisher: Übersetzung von**
  - Mini-Java-Exp: Datenkeller zur Auswertung arithm. Ausdrücke
  - Mini-Java: Sprünge zur Simulation von Kontrollstrukturen
  - Mini-Java-FunProc: rekursive Funktionen und Prozeduren
- **jetzt:**
  - Mini-Java-DS:

    Erweiterung von Mini-Java um
    - statische Datenstrukturen
      - Felder: konst. Anzahl von Daten gleichen Typs
      - Strukturen: konst. Anzahl von Daten verschiedenen Typs
    - dynamische Datenstrukturen
      - Zeiger (Referenzen)
  - üblicherweise: Verschachtelung der DS **hier nicht**

  -> dafür jedoch auch: Typdefinitionen

Syntax

- **statische Strukturen müssen in Typ-Def. angegeben werden**
- **Felder statisch**

```
type arrT   = int[3];
```

- **Strukturen statisch**

```
type strT  = struct { int anfang, ende };
```

- **Pointer dynamisch**

```
direkt in Deklaration
```

- **Deklarationen**

```
int i, arrT a, strT s, int* p;
```

  - für `a` 3 int-Speicherplätze initialisiert mit 0
  - für `s` 2 int-Speicherplätze initialisiert mit 0
  - für `p` 1 int-Speicherplatz initialisiert mit null

Zugriff auf Variablen

- **Felder**

```
a[0], a[1], a[2]
```

- **Strukturen**

```
s.anfang s.ende
```

- **Pointer**

```
*p
```

```
p = new int[1]
```

Codebeispiel

```
type arrT  = int[3];
type strT  = struct { int anfang, ende };

int i, arrT a, strT s, int* p;
{
    i = 0;
    s.anfang = i;
    p = new int[1];
    *p = 42;
    while i < 3 {
        a[i] = *p * i;
        i = i + 1;
    }
    s.ende = i;
}
```

Semantik?

var	sig(rho(var))
i	3
a[0]	0
a[1]	42
a[2]	84
s.anfang	0
s.ende	3
*p	42

Symboltabellenerweiterung

- **bisher**

```
[c1/(const,5), v1/(var,1), v2/(var,2), ... ]
```

- **jetzt neu: Typdefinitionen**

```
[arrT/(type,array,3),
 strT/(type,struct,anfang,0,ende,1,2)]
```

- **jetzt neu: Typinformation für jede Variable**

```
[ i/(var,int,1),
  a/(var,arrT,2),
  s/(var,strT,5),
  p/(var,int*,7)]
```

- **brauchen in update Parameter, der nächsten zu vergebenden Speicherplatz adressiert**

Symboltabelle für Array-Typ

- **Bsp.:**
  - `[..., arrT/(type,array,3), ...]`
- **Komponenten:**
  - `arrT` : Name des Typs
  - `type` : Schlüsselwort für Typ-Definition
  - `array` : Schlüsselwort für Array
  - `3` : Speicherbedarf des Arrays
- **entspricht:**

```
type arrT = int[3];
```

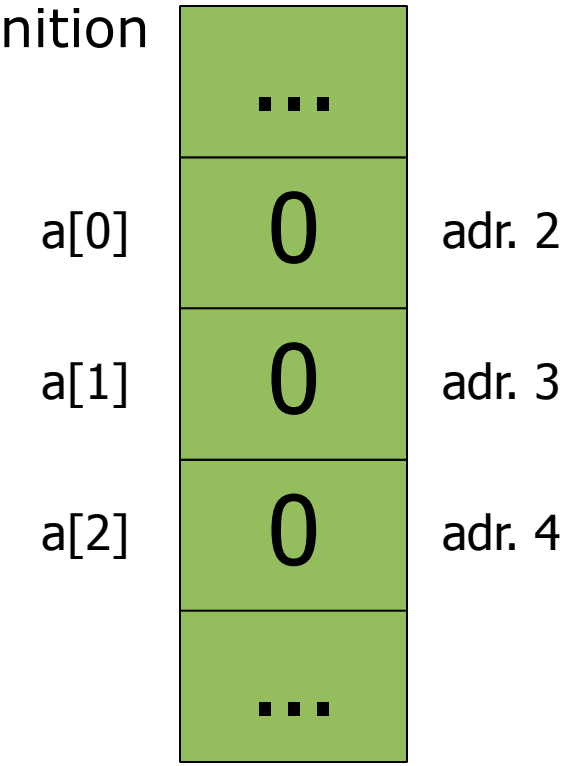
- **Festhalten des Speicherbedarfs zur späteren Adressberechnung**

Symboltabelle für Arrayvariable

- **Bsp.:**
  - `[..., a/(var,arrT,2), ...]`
- **Komponenten:**
  - `a` : Name der Variable
  - `var` : Schlüsselwort für Variablendefinition
  - `arrT` : Typ der Variable
  - `2` : Startadresse der Variable
- **entspricht:**

  `arrT a;`
- **Typeintrag war:**

  `[.., arrT/(type,array,3), ...]`



Symboltabelle für Struct-Typ

- **Bsp.:**
  - `[...,strT/(type,struct,anfang,0,ende,1,2),...]`
- **Komponenten:**
  - **`strT`** : Name des Typs
  - **`type`** : Schlüsselwort für Typ-Definition
  - **`struct`** : Schlüsselwort für Struct
  - **`anfang, ende`** : Name der 1. bzw. 2. Komponente
  - **`0, 1`** : Offset der 1. bzw. 2. Komponente
  - **`2`** : Speicherbedarf der Struktur
- **entspricht der Typdefinition:**

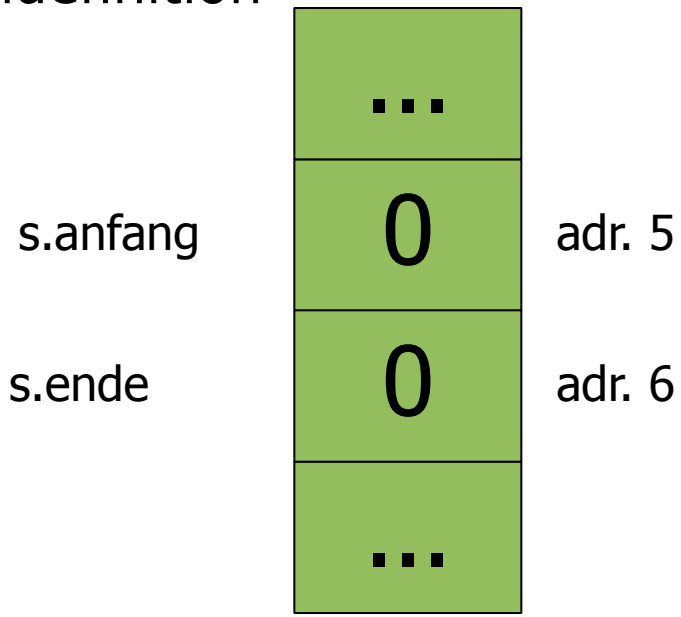
```
type strT  = struct { int anfang, ende };
```

- **Festhalten des Speicherbedarfs zur späteren Adressberechnung**

Symboltabelle für Struct-Variable

- **Bsp.:**
  - `[..., s/(var,strT,5), ...]`
- **Komponenten:**
  - **`s`** : Name der Variable
  - **`var`** : Schlüsselwort für Variablendefinition
  - **`strT`** : Typ der Variable
  - **`5`** : Startadresse der Variable
- **entspricht:**

```
strT s;
```



- **Typeintrag war:**

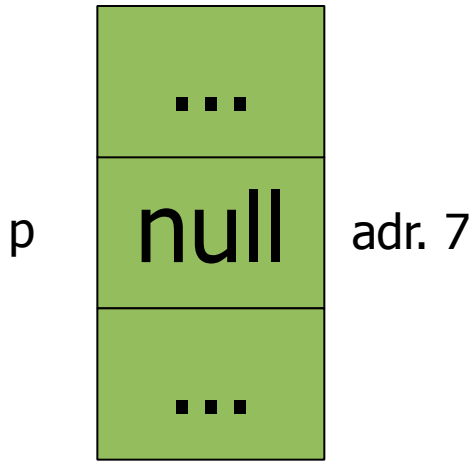
`[...,strT/(type,struct,anfang,0,ende,1,2),...]`

Symboltabelle für Pointer-Variable

- **Bsp.:**
  - `[..., p/(var,int*,7), ...]`
- **Komponenten:**
  - `p` : Name der Variable
  - `var` : Schlüsselwort für Variablendefinition
  - `int*` : Typ der Variable
  - `7` : Startadresse der Variable
- **entspricht:**

```
int* p;
```

- **Typeintrag nicht explizit**

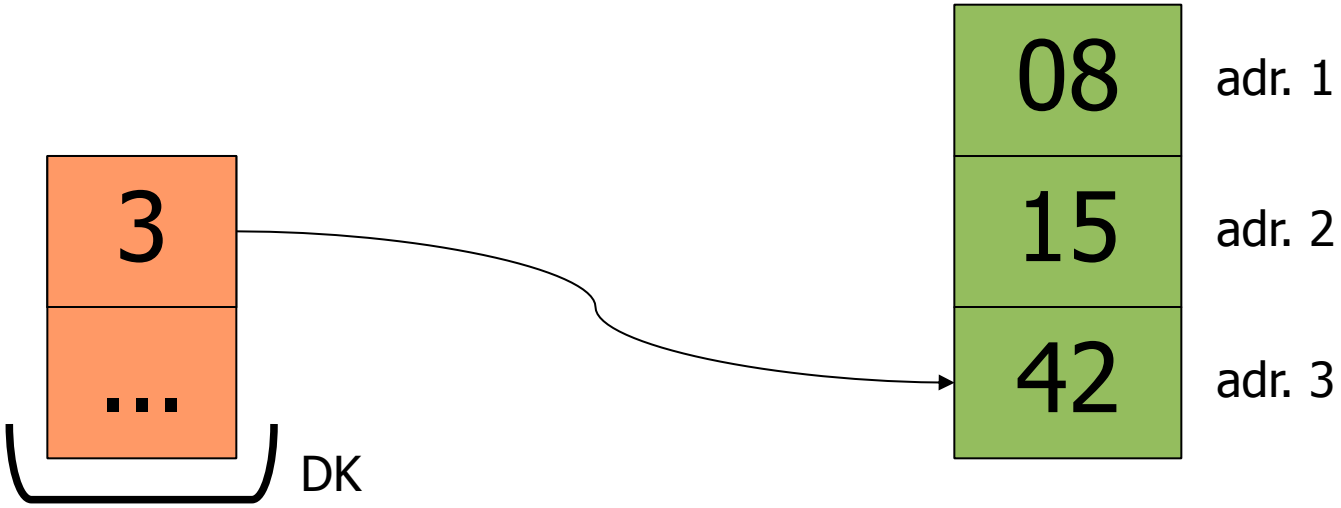


Neue Befehle für Z

- Idee: Berechne Hauptspeicher-Adressen auf DK und verwende indirektes Laden und indirektes Speichern

- Indirektes Laden

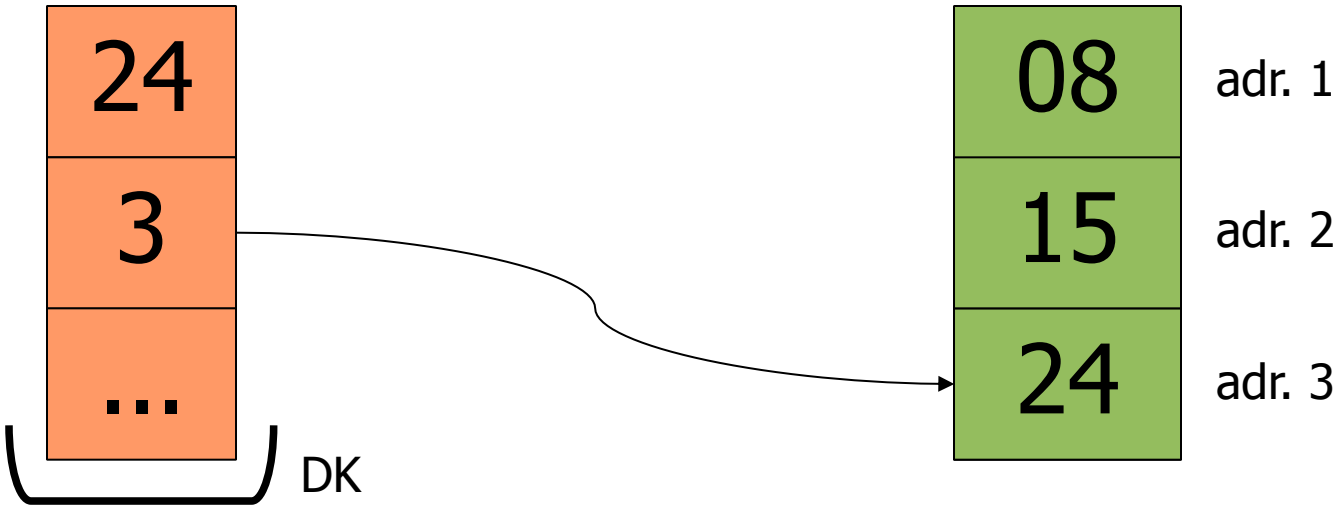
  $\mathbb{C}$ (LODI) (m, a:d, h) = (m+1, h(a):d, h)



Neue Befehle für Z 2

- Indirektes Speichern

  $\mathbb{C}$ (STOI) (m, z a:d, h) = (m+1, d, h[a/z])



- **Überprüfung der Arraygrenzen**

  $\mathbb{C}$ (AC n) (m, i:d, h) = **if** 0 <= i < n **then** (m+1, i:d, h)
  else throw Exception

Erweiterung von exptrans

- indirekte Adressierung nach Speicherplatzauswertung auf DK
  - *exptrans* (*ide[exp],* st) =

    if st(*ide*) **=** (var,typ,start) && st(typ) = (type,array,n)

    then LIT start; //Array-Basis auf DK
    *exptrans(exp,*st); //Werte Index aus
    AC n; //ArrayIndexOutOfBounds?
    ADD; //Adresse des Array-Elements
    LODI; //indirektes Laden

  - *exptrans* (*ide.el,* st) =

    if st(*ide*) **=** (var,typ,start) && st(typ) = (type,struct,...,el,n,...)

    then LIT start; //Struct-Basis auf DK
    LIT n; //Offset auf DK
    ADD; //Adresse der Struct-Komponente
    LODI; //indirektes Laden

Erweiterung von cmdtrans

– *cmdtrans* (*ide[exp1] = exp2;*, st) =

if st(*ide*) **=** (var,typ,start) && st(typ) = (type,array,n)

then
```
LIT start;              //Array-Basis auf DK
exptrans(exp1,st);      //Werte Index aus
AC n;                   //ArrayIndexOutOfBounds?
ADD;                    //Adresse des Array-Elements
exptrans(exp2,st);      //Werte rechte Seite aus
STOI;                   //indirektes Speichern
```

– *cmdtrans* (*ide.el = exp;*, st) =

if st(*ide*) **=** (var,typ,start) && st(typ) = (type,struct,...,el,n,...)

then
```
LIT start;              //Struct-Basis auf DK
LIT n;                  //Offset auf DK
ADD;                    //Adresse der Struct-Komponente
exptrans(exp,st);       //Werte rechte Seite aus
STOI;                   //indirektes Speichern
```

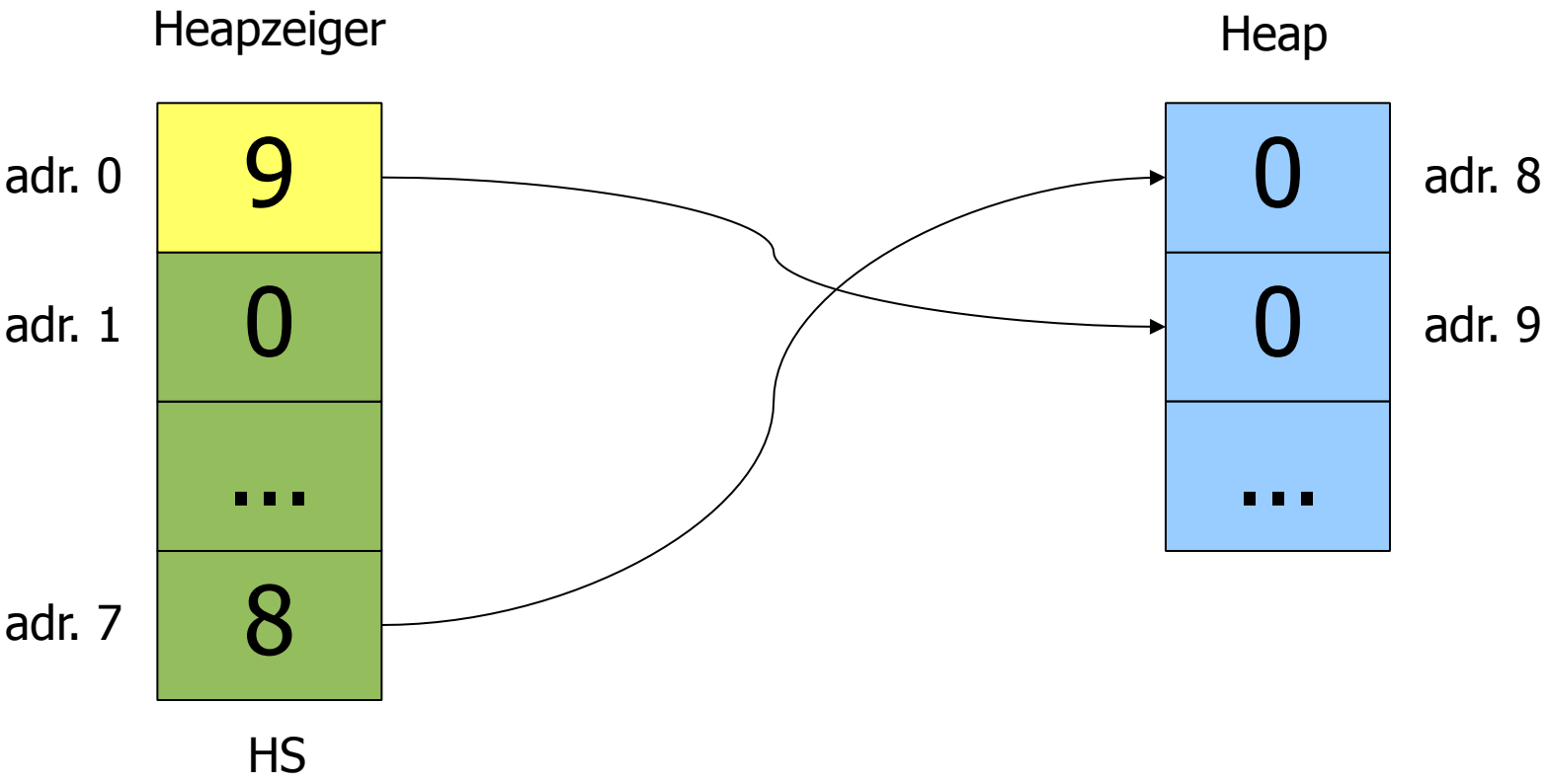
Übersetzung von Pointern

- **in Symboltabelle Adresse des Pointers:**
  - `[..., p/(var,int*,7), ...]`

- **Umgang mit Pointern**
  - Stack reicht nicht mehr -> **Heap**
  - Heap im HS realisiert
  - Heap-Zeiger an Adresse 0 im HS
  - nach Var-Dekl. Heap-Zeiger initialisiert auf 1. freien Platz in HS
  - Allokation zur Laufzeit
  - Weitere Indirektionsstufe -> **Dereferenzieren**

Dynamische Speicherallokation

- Bei Pointer in HS: Adressen des Heaps
- Bsp. für: `[..., p/(var,int*,7), ...]`



Erweiterung von cmdtrans

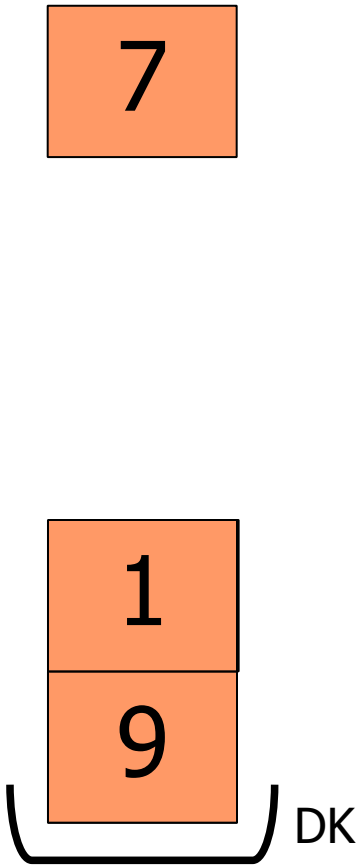
cmdtrans (*p = new int[1];*, st) =

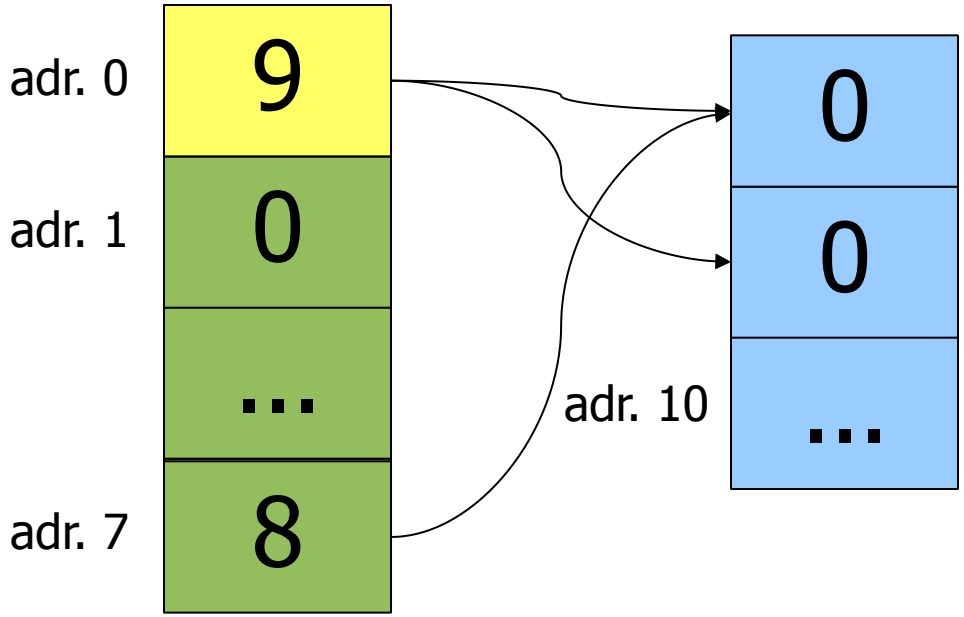
```
if st(p) = (var,int*,n)
then   LIT n;          //Pointer-Adresse aus HS auf DK
       LOD 0;          //Heap-Zeiger auf DK
       STOI;           //Heap-Zeiger -> Pointer-Adresse
       LOD 0;          //nochmal Heap-Zeiger auf DK
       LIT 1;          //1 auf DK
       ADD;            //Heap-Zeiger um 1 erhöhen
       STO 0;          //neuen Heap-Zeiger abspeichern
```

Beispiel für dyn. Speicherallokation

cmdtrans (p = new int[1];, st):
(st = `[p/(var,int*,7)]`)

```
LIT 7;
LOD 0;
STOI;
LOD 0;
LIT 1;
ADD;
STO 0;
```



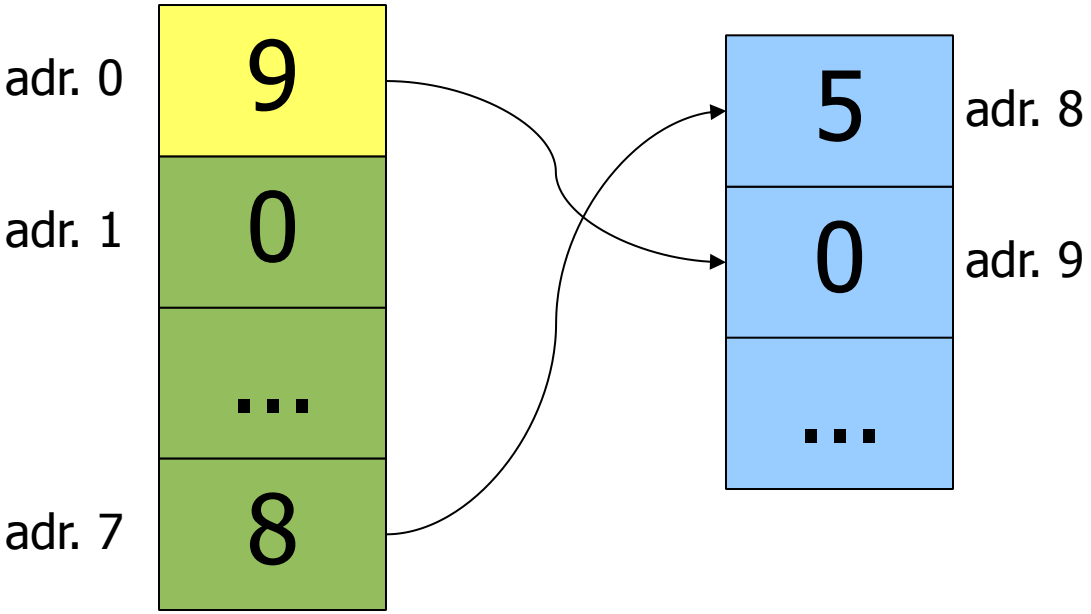


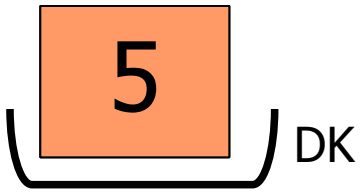
Erweiterung von exptrans

- *exptrans (*p, st)* =

  if st(p) **=** (var,int*,n)
  then LOD n;
  LODI;

```
LOD  7;
LODI;
```





Erweiterung von cmdtrans

- *cmdtrans (*p = exp, st)* =

```
        if st(p) = (var,int*,n)
        then   LOD n;                  //Adresse vom Heap auf DK
               exptrans(exp,st)        //Werte rechte Seite aus
               STOI;                   //Speicher auf Heap ab
```

Zusätze für Praxis

- **in Praxis üblicherweise geschachtelte DS**
- **müssten Definitionen erweitern auf Verschachtelung**

  -> komplexere Berechnungen für Symboltabelle

  mit rekursiven Definitionen jedoch unproblematisch
- **Semantik-Definition analog**

Beispielübersetzung

- **für Programm auf Folie 4 & Symboltabelle auf Folie 5**
- **nur für neue Statements**

```
s.anfang = i;      -> 3: LIT 5; 4: LIT 0; 5: ADD;
                      6: LOD 1; 7: STOI;

p= new int[1];     -> 8: LIT 7; 9: LOD 0; 10: STOI;
                      11: LOD 0; 12: LIT 1; 13: ADD;
                      14: STO 0;

*p = 42;           -> 15: LOD 7; 16: LIT 42; 17: STOI;

a[i] = *p * i;     -> 22: LIT 2; 23: LOD 1; 24: AC 3;
                      25: ADD; 26: LOD 7; 27: LODI;
                      28: LOD 1; 29: MULT; 30: STOI;

s.ende = i;        -> 36: LIT 5; 37: LIT 1; 38: ADD;
                      39: LOD 1; 40: STOI;
```

Beispielübersetzung komplett

1: LIT 0;	11: LOD 0;	21: JMC 36;	31: LOD 1;
2: STO 1;	12: LIT 1;	22: LIT 2;	32: LIT 1;
3: LIT 5;	13: ADD;	23: LOD 1;	33: ADD;
4: LIT 0;	14: STO 0;	24: AC 3;	34: STO 1;
5: ADD;	15: LOD 7;	25: ADD;	35: JMP 18;
6: LOD 1;	16: LIT 42;	26: LOD 7;	36: LIT 5;
7: STOI;	17: STOI;	27: LODI;	37: LIT 1;
8: LIT 7;	18: LOD 1;	28: LOD 1;	38: ADD;
9: LOD 0;	19: LIT 3;	29: MULT;	39: LOD 1;
10: STOI;	20: LT;	30: STOI;	40: STOI;

Beispiel-Rechnung

(1, ,	[0/8,1/0,2/0,3/0,4/0,5/0,6/0,7/0])	//LIT 0
(2, 0,	[0/8,1/0,2/0,3/0,4/0,5/0,6/0,7/0])	//STO 1
(3, ,	[0/8,1/0,2/0,3/0,4/0,5/0,6/0,7/0])	//LIT 5
(4, 5,	[0/8,1/0,2/0,3/0,4/0,5/0,6/0,7/0])	//LIT 0
(5, 0 5,	[0/8,1/0,2/0,3/0,4/0,5/0,6/0,7/0])	//ADD
(6, 5,	[0/8,1/0,2/0,3/0,4/0,5/0,6/0,7/0])	//LOD 1
(7, 0 5,	[0/8,1/0,2/0,3/0,4/0,5/0,6/0,7/0])	//STOI
(8, ,	[0/8,1/0,2/0,3/0,4/0,5/0,6/0,7/0])	//LIT 7
(9, 7,	[0/8,1/0,2/0,3/0,4/0,5/0,6/0,7/0])	//LOD 0
(10, 8 7,	[0/8,1/0,2/0,3/0,4/0,5/0,6/0,7/0])	//STOI
(11, ,	[0/8,1/0,2/0,3/0,4/0,5/0,6/0,7/8,8/0])	//LOD 0
(12, 8,	[0/8,1/0,2/0,3/0,4/0,5/0,6/0,7/8,8/0])	//LIT 1
(13, 1 8,	[0/8,1/0,2/0,3/0,4/0,5/0,6/0,7/8,8/0])	//ADD
(14, 9,	[0/8,1/0,2/0,3/0,4/0,5/0,6/0,7/8,8/0])	//STO 0

Beispiel-Rechnung 2

```
(15,  ,          [0/9,1/0,2/0,3/0,4/0,5/0,6/0,7/8,8/0])    //LOD 7
(16, 8,          [0/9,1/0,2/0,3/0,4/0,5/0,6/0,7/8,8/0])    //LIT 42
(17, 42 8 ,      [0/9,1/0,2/0,3/0,4/0,5/0,6/0,7/8,8/0])    //STOI
(18,  ,          [0/9,1/0,2/0,3/0,4/0,5/0,6/0,7/8,8/42])   //LOD 1
(19,  0,         [0/9,1/0,2/0,3/0,4/0,5/0,6/0,7/8,8/42])   //LIT 3
(20, 3  0,       [0/9,1/0,2/0,3/0,4/0,5/0,6/0,7/8,8/42])   //LT
(21,  1,         [0/9,1/0,2/0,3/0,4/0,5/0,6/0,7/8,8/42])   //JMC 36
(22,  ,          [0/9,1/0,2/0,3/0,4/0,5/0,6/0,7/8,8/42])   //LIT 2
(23,  2,         [0/9,1/0,2/0,3/0,4/0,5/0,6/0,7/8,8/42])   //LOD 1
(24, 0 2,        [0/9,1/0,2/0,3/0,4/0,5/0,6/0,7/8,8/42])   //AC 3
(25, 0 2,        [0/9,1/0,2/0,3/0,4/0,5/0,6/0,7/8,8/42])   //ADD
(26, 2,          [0/9,1/0,2/0,3/0,4/0,5/0,6/0,7/8,8/42])   //LOD 7
(27, 8 2,        [0/9,1/0,2/0,3/0,4/0,5/0,6/0,7/8,8/42])   //LODI
(28, 42 2,       [0/9,1/0,2/0,3/0,4/0,5/0,6/0,7/8,8/42])   //LOD 1
```

Beispiel-Rechnung 3

(29, 0 42 2,	[0/9,1/0,2/0,3/0,4/0,5/0,6/0,7/8,8/42])	//MULT
(30, 0 2,	[0/9,1/0,2/0,3/0,4/0,5/0,6/0,7/8,8/42])	//STOI
(31, ,	[0/9,1/0,2/0,3/0,4/0,5/0,6/0,7/8,8/42])	//LOD 1
(32, 0,	[0/9,1/0,2/0,3/0,4/0,5/0,6/0,7/8,8/42])	//LIT 1
(33, 1 0,	[0/9,1/0,2/0,3/0,4/0,5/0,6/0,7/8,8/42])	//ADD
(34, 1,	[0/9,1/0,2/0,3/0,4/0,5/0,6/0,7/8,8/42])	//STO 1
(35, ,	[0/9,1/1,2/0,3/0,4/0,5/0,6/0,7/8,8/42])	//JMP 18
(18, ,	[0/9,1/1,2/0,3/0,4/0,5/0,6/0,7/8,8/42])	//LOD 1
(19, 1,	[0/9,1/1,2/0,3/0,4/0,5/0,6/0,7/8,8/42])	//LIT 3
(20, 3 1,	[0/9,1/1,2/0,3/0,4/0,5/0,6/0,7/8,8/42])	//LT
(21, 1,	[0/9,1/1,2/0,3/0,4/0,5/0,6/0,7/8,8/42])	//JMC 36
(22, ,	[0/9,1/1,2/0,3/0,4/0,5/0,6/0,7/8,8/42])	//LIT 2
(23, 2,	[0/9,1/1,2/0,3/0,4/0,5/0,6/0,7/8,8/42])	//LOD 1
(24, 1 2,	[0/9,1/1,2/0,3/0,4/0,5/0,6/0,7/8,8/42])	//AC 3

Beispiel-Rechnung 4

(25, 1 2,	[0/9,1/1,2/0,3/0,4/0,5/0,6/0,7/8,8/42])	//ADD
(26, 3,	[0/9,1/1,2/0,3/0,4/0,5/0,6/0,7/8,8/42])	//LOD 7
(27, 8 3,	[0/9,1/1,2/0,3/0,4/0,5/0,6/0,7/8,8/42])	//LODI
(28, 42 3,	[0/9,1/1,2/0,3/0,4/0,5/0,6/0,7/8,8/42])	//LOD 1
(29, 1 42 3,	[0/9,1/1,2/0,3/0,4/0,5/0,6/0,7/8,8/42])	//MULT
(30, 42 3,	[0/9,1/1,2/0,3/0,4/0,5/0,6/0,7/8,8/42])	//STOI
(31, ,	[0/9,1/1,2/0,3/42,4/0,5/0,6/0,7/8,8/42])	//LOD 1
(32, 1,	[0/9,1/1,2/0,3/42,4/0,5/0,6/0,7/8,8/42])	//LIT 1
(33, 1 1,	[0/9,1/1,2/0,3/42,4/0,5/0,6/0,7/8,8/42])	//ADD
(34, 2,	[0/9,1/1,2/0,3/42,4/0,5/0,6/0,7/8,8/42])	//STO 1
(35, ,	[0/9,1/2,2/0,3/42,4/0,5/0,6/0,7/8,8/42])	//JMP 18
(18, ,	[0/9,1/2,2/0,3/42,4/0,5/0,6/0,7/8,8/42])	//LOD 1
(19, 2,	[0/9,1/2,2/0,3/42,4/0,5/0,6/0,7/8,8/42])	//LIT 3
(20, 3 2,	[0/9,1/2,2/0,3/42,4/0,5/0,6/0,7/8,8/42])	//LT

Beispiel-Rechnung 5

(21, 1,	[0/9,1/2,2/0,3/42,4/0,5/0,6/0,7/8,8/42])	//JMC 36
(22, ,	[0/9,1/2,2/0,3/42,4/0,5/0,6/0,7/8,8/42])	//LIT 2
(23, 2,	[0/9,1/2,2/0,3/42,4/0,5/0,6/0,7/8,8/42])	//LOD 1
(24, 2 2,	[0/9,1/2,2/0,3/42,4/0,5/0,6/0,7/8,8/42])	//AC 3
(25, 2 2,	[0/9,1/2,2/0,3/42,4/0,5/0,6/0,7/8,8/42])	//ADD
(26, 4,	[0/9,1/2,2/0,3/42,4/0,5/0,6/0,7/8,8/42])	//LOD 7
(27, 8 4,	[0/9,1/2,2/0,3/42,4/0,5/0,6/0,7/8,8/42])	//LODI
(28, 42 4,	[0/9,1/2,2/0,3/42,4/0,5/0,6/0,7/8,8/42])	//LOD 1
(29, 2 42 4,	[0/9,1/2,2/0,3/42,4/0,5/0,6/0,7/8,8/42])	//MULT
(30, 84 4,	[0/9,1/2,2/0,3/42,4/0,5/0,6/0,7/8,8/42])	//STOI
(31, ,	[0/9,1/2,2/0,3/42,4/84,5/0,6/0,7/8,8/42])	//LOD 1
(32, 2,	[0/9,1/2,2/0,3/42,4/84,5/0,6/0,7/8,8/42])	//LIT 1
(33, 1 2,	[0/9,1/2,2/0,3/42,4/84,5/0,6/0,7/8,8/42])	//ADD
(34, 3,	[0/9,1/2,2/0,3/42,4/84,5/0,6/0,7/8,8/42])	//STO 1

Beispiel-Rechnung 6

(35, ,	[0/9,1/3,2/0,3/42,4/84,5/0,6/0,7/8,8/42])	//JMP 18
(18, ,	[0/9,1/3,2/0,3/42,4/84,5/0,6/0,7/8,8/42])	//LOD 1
(19, 3,	[0/9,1/3,2/0,3/42,4/84,5/0,6/0,7/8,8/42])	//LIT 3
(20, 3 3,	[0/9,1/3,2/0,3/42,4/84,5/0,6/0,7/8,8/42])	//LT
(21, 0,	[0/9,1/3,2/0,3/42,4/84,5/0,6/0,7/8,8/42])	//JMC 36
(36, ,	[0/9,1/3,2/0,3/42,4/84,5/0,6/0,7/8,8/42])	//LIT 5
(37, 5,	[0/9,1/3,2/0,3/42,4/84,5/0,6/0,7/8,8/42])	//LIT 1
(38, 1 5,	[0/9,1/3,2/0,3/42,4/84,5/0,6/0,7/8,8/42])	//ADD
(39, 6,	[0/9,1/3,2/0,3/42,4/84,5/0,6/0,7/8,8/42])	//LOD 1
(40, 3 6,	[0/9,1/3,2/0,3/42,4/84,5/0,6/0,7/8,8/42])	//STOI
(41, ,	[0/9,1/3,2/0,3/42,4/84,5/0,6/3,7/8,8/42])	//STOPP