Namensliste

- Teil der semantischen Aktionen
- Führt eine Liste über alle gültigen Bezeichner und die Eigenschaften der benannten Sprachkonstrukte
- Realisiert nebenbei die Gültigkeitsbereiche von Bezeichnern
- Operationen der Namensliste:
  - einfügen eines Bezeichners in die aktuelle Namensliste
  - Lokale Suche eines Bezeichners in der aktuellen Namensliste
  - globale Suche eines Bezeichners „von innen nach außen“

Namensliste

Es gibt bestimmt sehr viele Möglichkeiten, die in einem Prgramm verwendeten Bezeichner und die bezeichneten Sprachelemente in einem Compiler zu verwalten. Im diesem Foliensatz werden zwei einander ähnelnde Möglichkeiten vorgestellt, die sich in der Compilerbaupraxis durchaus bewährt haben.

Sie beruhen auf Listen. In den meisten Fällen reicht es, immer am Listenkopf einzuketten, das ist der einfachste Fall der Arbeit mit verketteten Listen.

Implementationsvarianten

- Namensliste für jede Prozedur
  - Jede Prozedur verfügt über ihre eigene Namensliste.
  - Über Eltern-Kind-Beziehungen ist die globale Suche von „innen nach außen“ möglich.
- Pulsierender Keller
  - Alle Namen werden in einem Keller geführt.
  - Anfang einer jeden lokalen Namensliste muss gekennzeichnet und mitgeführt werden.
- In beiden Fällen gibt es Namenseinträge, die auf Beschreibungen der benannten Sprachelemente (hier Konstanten, Variablen und Prozeduren) verweisen.

Namenslisteneintrag

Aufbau eines Namens-listeneintrags

(KzName, nicht wirklich nötig, hilfreich beim debugging)

Prozedurnummer (zu der dieser Bezeichner gehört)

Pointer auf benanntes Objekt (Var-, Const-, Procbeschreibung)

Pointer auf Name

Kennzeichen und Länge sind redundant

```
typedef struct
{
  tKz   Kz;
  short IdxProc;
  void* pObj;
  int   Len;
  char* pName;
}tBez;
```

Anmerkungen zur Variablenbeschreibung

- Die folgenden Betrachtungen beziehen sich auf die verwendete virtuelle Maschine.
- Variablen werden zur Laufzeit im Stack angelegt.
- In jeder neuen Prozedur beginnt die Adressierung der Variablen mit 0.
- Da es in PL0 keine Datentypen gibt sondern nur die Variablenart Zahl, sind alle Variablen gleich lang (4 byte).
- Ist eine Sprache mit Datentypen zu implementieren, so mss es Typbeschreibungen geben, die dann die Länge des zu reservierenden Speichers für eine Variable des Typs enthalten.

Anmerkungen zur Variablenbeschreibung

- In unserer Implementierung von PL0 seien alle Variablen ganzzahlig in der Länge 4 Byte.
- Mit jeder neuen Variablen wird ein Speicher-platzzuordnungszähler (SpzzVar) um die Länge der Variablen (hier 4) erhöht. Somit ergibt sich eine relative Adresse der Variablen innerhalb des Stackframes der umgebeneden Prozedur.

1. Variable: relAddr=0
2. Variable: relAddr=4
3. Variable: relAddr:=8 . . .

Variablenbeschreibung

Variablenbeschreibung

- KzVar
- Displacement oder Relativadresse

```
typedef struct tVAR
{
  tKz Kz;
  int Dspl; // Relativadresse
}tVar;
```

Variablenbeschreibung

In einer Implementierung des Compilers in einer objektorientierten Sprache kann das Kennzeichen entfallen. Hier kann man einen Basistyp für Namenslistenobjekte und abgeleitete Klassen für Variablen-, Konstanten- und Prozedurbeschreibungen vorsehen. Beim Debugging ist das Kennzeichen auch mitunter hilfreich.

Anmerkungen zur Konstantenbeschreibung

Alle Konstanten werden in einem Konstantenblock gesammelt, der am Ende an den generierten Code angehängt wird. Jede Konstante ist nur ein mal im Konstantenblock enthalten, gleichgültig, wie oft die Konstante im Quelltext vorkommt.

Die Konstanten sind 4 Byte groß

Die Konstanten werden indiziert

1. Konstante: ConstBlock[0]
2. Konstante: ConstBlock[1]

Konstanten am Ende des Codes

Pl0 Beispiel

```
var a;
begin
  a:=-1+2*3
end.
```

Der generierte Code als Ausgabe von outCl0 in mnemonischer Form

```
0000: 1A EntryProc                0018,0000,0004
0007: 04 PushAdrVarMain           0000
000A: 06 PushConst                0000
000D: 0A VzMinus
000E: 06 PushConst                0001
0011: 06 PushConst                0002
0014: 0E Mul
0015: 0C Add
0016: 07 StoreVal
0017: 17 ReturnProc
Const 0000:0001
Const 0001:0002
Const 0002:0003
```

Der generierte Code als Hexdump, wie er in der Codedatei steht und von der virtuellen Maschine ausgeführt wird (Fett: die Konstanten).

```
0000  01 00 00 00 1a 18 00 00  00 04 00 04 00 00 06 00  |................|
0010  00 0a 06 01 00 06 02 00  0e 0c 07 17 01 00 00 00  |................|
0020  02 00 00 00 03 00 00 00                          |........|
```

Konstantenbeschreibung

Member der Konstantenbe-schreibung

- KzConst
- Wert der Konstanten
- Index der Konstanten im ConstBlock

```
typedef struct tCONST
{
  tKz  Kz;
  long Val;
  int  Idx;
}tConst;
```

Anmerkungen zur Prozedurbeschreibung

- Jede Prozedur und das Hauptprogramm erhalten eine Nummer, das Hauptprogramm hat die Nummer 0 (Prozedurnummer idxProc).
- Die Nummern werden fortlaufend vergeben.
- Jede Prozedurbeschreibung enthält eine Namensliste mit den lokalen Namen.
- Der Name einer Prozedur steht dabei immer in der übergeordneten Namensliste.

Prozedurbeschreibung

- KzProc
- Prozedurnummer
- Pointer auf die Prozedurbeschreibung der Parentprozedur
- Lokale Namensliste der Prozedur
- Speicherplatzzuordnungszähler für Variablen

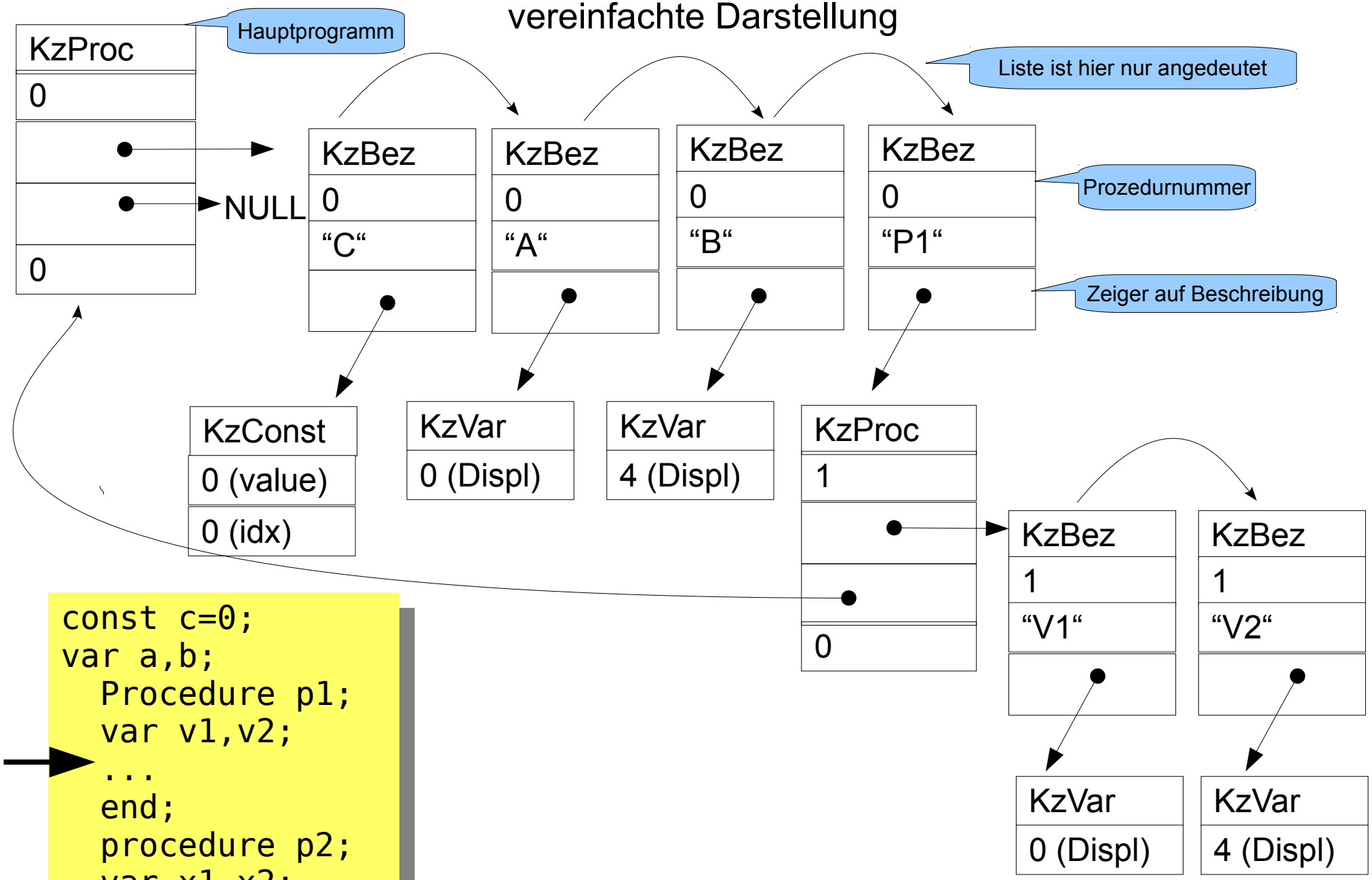
```
typedef struct tPROC
{
  tKz    Kz;
  short  IdxProc;
  struct tPROC*pParent;
  tList *pLBez;
  int    SpzzVar;
}tProc;
```

Anmerkungen zu den Listen

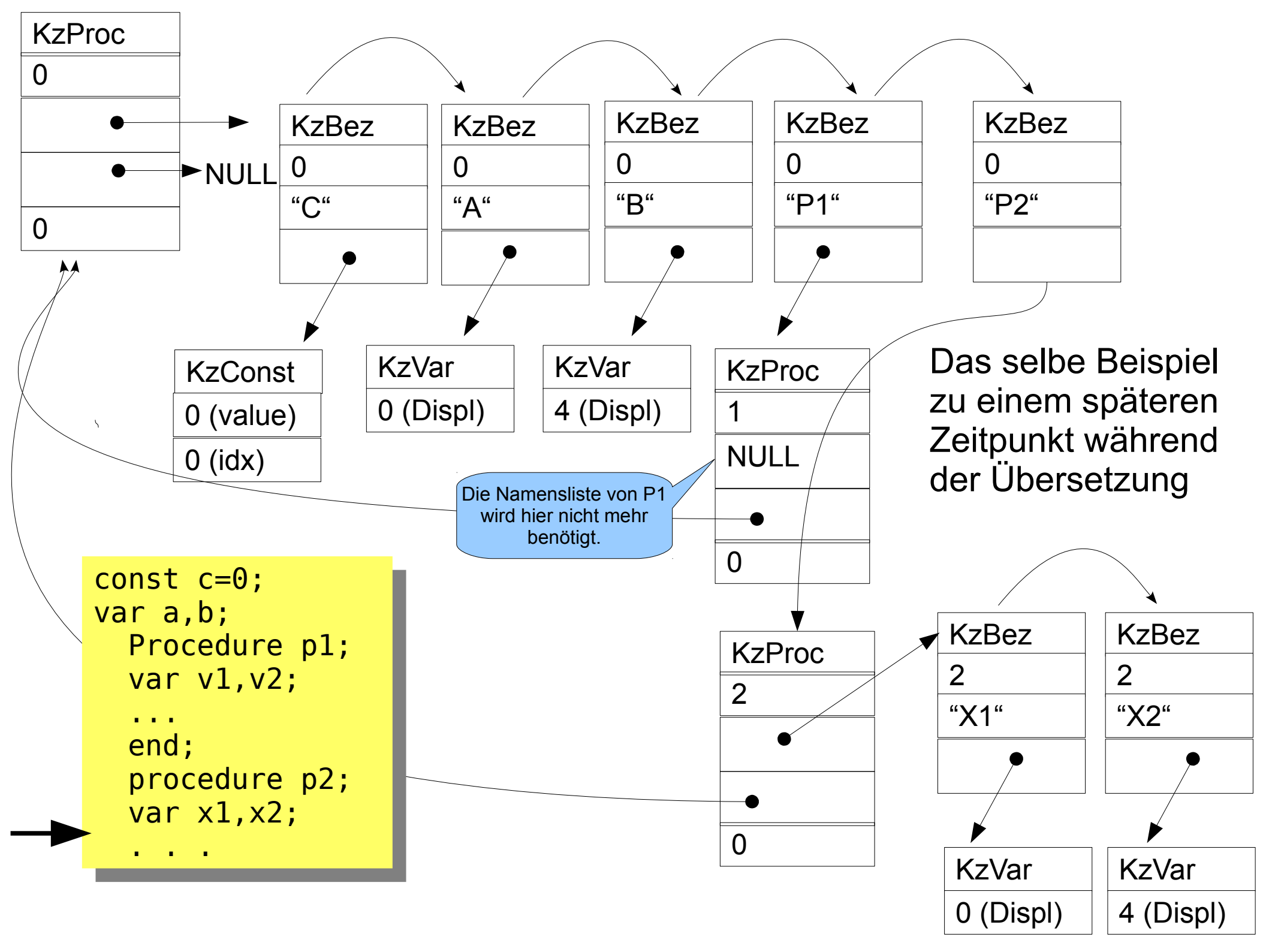
- Wie bereits oben angemerkt, reichen einfach verkettete, offene Listen, bei denen am Listenkopf ein- und ausgekettet wird.
- Die Elemente der Namensliste können auch eine intrusive Liste bilden, dh. Sie können den Zeiger auf das jeweils nächste Namensobjekt selbst enthalten.
- Wenn aus dem Modul Programmierung I/Praktikum 12 noch eine Listenimplementation verfügbar ist, kann diese auch verwendet werden.

Namensliste (Schnappschuss)

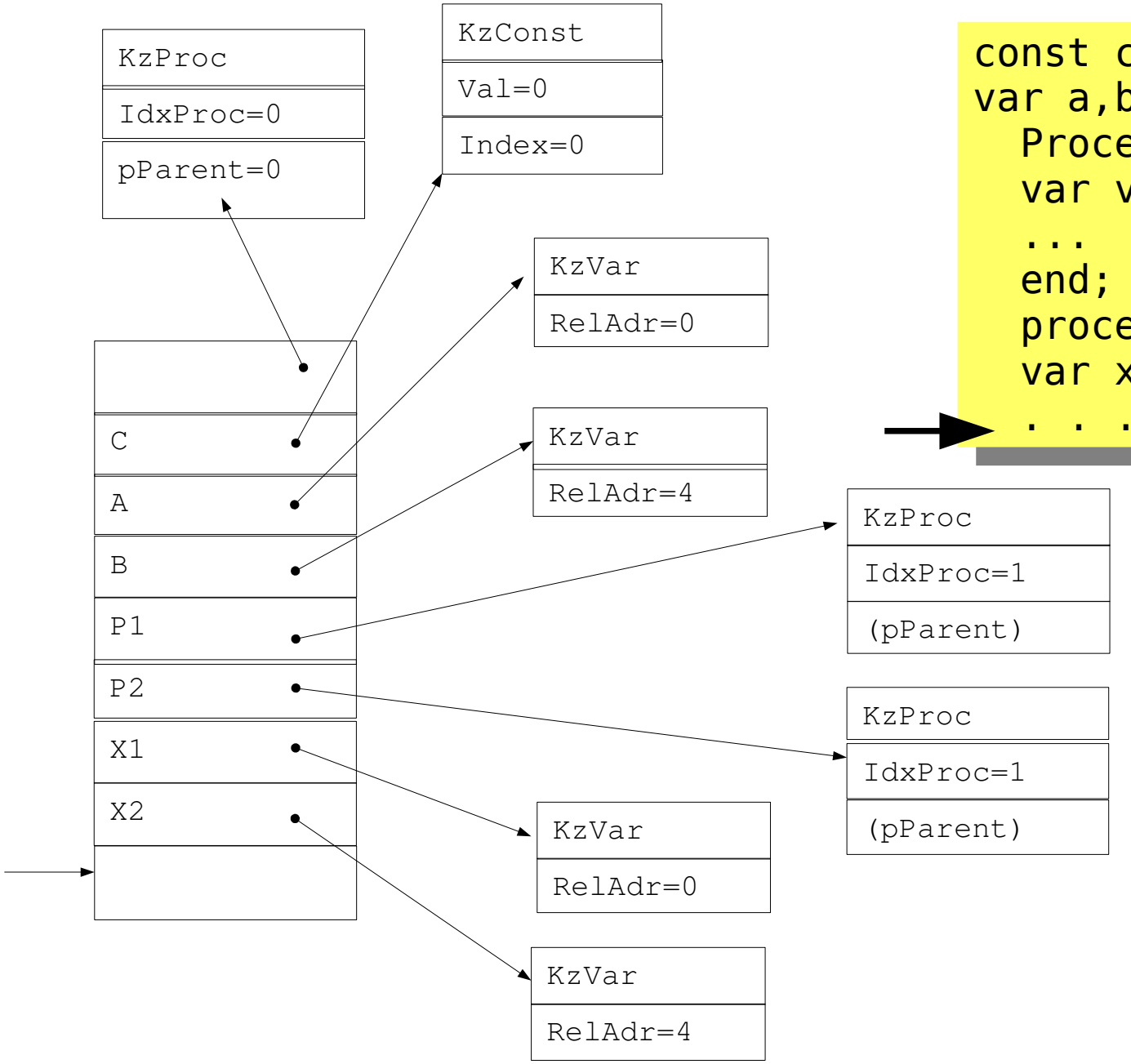
vereinfachte Darstellung




```
const c=0;
var a,b;
  Procedure p1;
  var v1,v2;
  ...
  end;
  procedure p2;
  var x1,x2;
  . . .
```


KzProc
0
NULL
0
KzBez
0
“C“
KzBez
0
“A“
KzBez
0
“B“
KzBez
0
“P1“
KzBez
0
“P2“
KzConst
0 (value)
0 (idx)
KzVar
0 (Displ)
KzVar
4 (Displ)
KzProc
1
NULL
0
Die Namensliste von P1 wird hier nicht mehr benötigt.
Das selbe Beispiel zu einem späteren Zeitpunkt während der Übersetzung
const c=0;
var a,b;
Procedure p1;
var v1,v2;
...
end;
procedure p2;
var x1,x2;
. . .
KzProc
2
0
KzBez
2
“X1“
KzBez
2
“X2“
KzVar
0 (Displ)
KzVar
4 (Displ)

Realisierung als Stack




```
const c=0;
var a,b;
  Procedure p1;
  var v1,v2;
  ...
  end;
  procedure p2;
  var x1,x2;
  . . .
```

Funktionen zur Namensliste

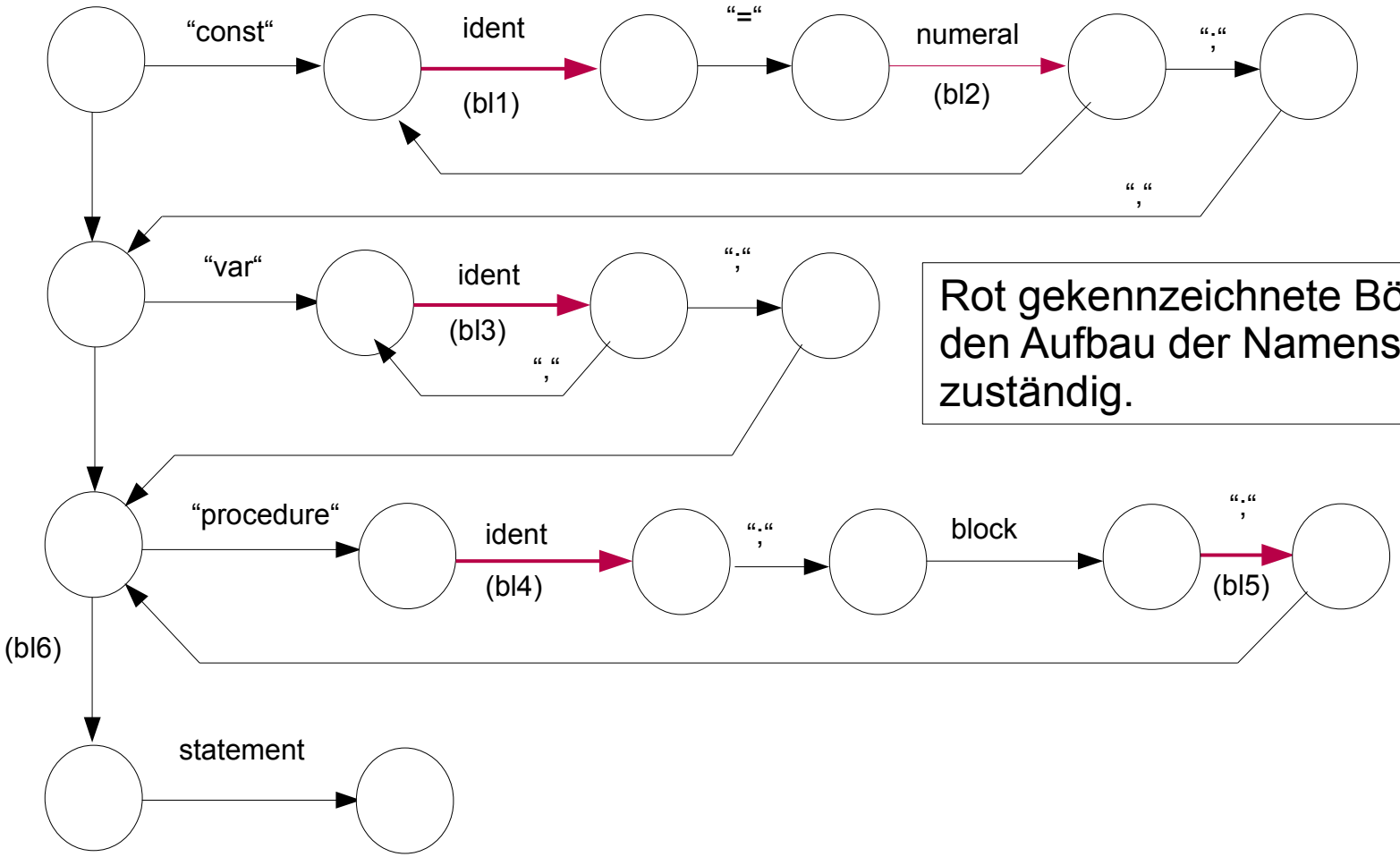
```
tBez* createBez(char*pBez);
tConst* createConst(long Val);
tConst* searchConst(long Val);
int createVar(void);
tProc* createProc(tProc*pParent);
tBez* searchBez(tProc*pProc,char* pBez);
tBez* searchBezGlobal(char* pBez);
```

Datenstrukturen

- Folgende Variablen/Objekte sind nötig:
  - Prozedurbeschreibung für das Hauptprogramm als Anker für die gesamte Namensliste, diese Prozedurbeschreibung hat keinen Namen.
  - Pointer auf die aktuelle Prozedurbeschreibung
  - Vraiable für die Prozedurnummer der aktuellen Prozedur
- Dierse Variablen werden später auch für die Codegenerierung benötigt und sollten deshalb global sein.

Parser und Namensliste

- Die Namensliste wird durch Routinen, die vom Parser aufgerufen werden, relisiert.
- An den Bögen der Syntaxgraphen gibt es Funktions-pointer, die auf Funktionen verweisen, die aufgerufen werden, wenn der Bogen akzeptiert ist.
- Die Namensliste wird nun durch Funktionen aufgebaut und von Funktionen verwendet, die vom Parser aus-gerufen werden.
- Auf den nachfolgenden Folien wird der Graph Block gezeigt. Ergänzt wurde, an welchen Stellen jetzt eine Routine aufgerufen wird, um die Namensliste aufzu-bauen und was die einzelnen Routinen ausführen.



Rot gekennzeichnete Bögen sind für den Aufbau der Namensliste zuständig.

bl1(Konstantenbezeichner): lokale Suche nach dem Bezeichner

- gefunden -> Fehlerbehandlung
- nicht gefunden -> Bezeichner anlegen

bl2 (Konstantenwert):

- Konstantenbeschreibung anlegen
- Suche nach Konstante im Konstantenblock
  - Gefunden → Index der Konstanten in Konstantenbeschreibung eintragen
  - Nicht gefunden → Konstante im Konstantenblock anlegen und Index der Konstanten in Konstantenbeschreibung eintragen
- In letzten Bezeichner Zeiger auf diese Konstantenbeschreibung eintragen

bl3 (Variablenbezeichner):
lokale Suche nach dem Bezeichner

- Gefunden -> Fehlerbehandlung
- Nicht gefunden -> Bezeichner anlegen
  - Variablenbeschreibung anlegen und Pointer in Bezeichner eintragen
  - Relativadresse ermitteln aus SpzzVar, SpzzVar um 4 erhöhen (Virtuelle Maschine arbeitet mit 4 Byte langen long-Werten)

bl4(Prozedurbezeichner):

- lokale Suche nach dem Bezeichner
  - gefunden -> Fehlerbehandlung
  - nicht gefunden -> Bezeichner anlegen
- Prozedurbeschreibung anlegen
- Pointer auf Parent-Prozedur eintragen
- Pointer auf Prozedurbeschreibung in letzten Bezeichner eintragen
- Neue Prozedur ist jetzt aktuelle Prozedur

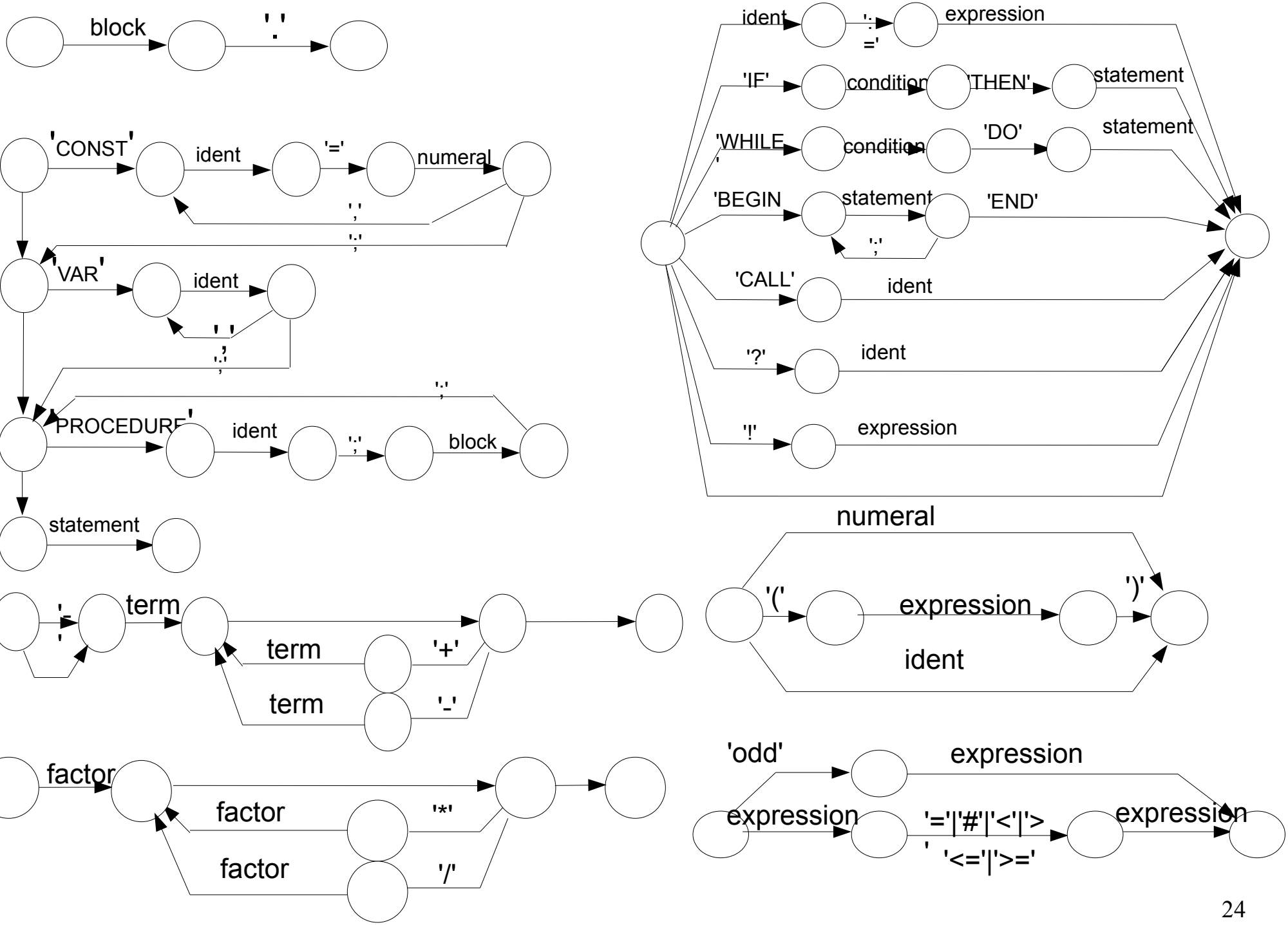
bl5 (Ende der Prozedurvereinbarung):

Codegenerierung: `retProc`

Codelänge in den Befehl `entryProc` als 1. Parameter nachtragen

Code aus dem Codepuffer in die Ausgabedatei schreiben (anfügen)

- Namensliste mit allen Konstanten-, Variablen- und Prozedurbeschreibungen auflösen; die Prozedurbeschreibung selbst muss noch erhalten bleiben.
- Die Parent-Prozedur wird die aktuelle Prozedur.


block
'.'
'CONST'
ident
'='
numeral
','
';'
'VAR'
ident
','
';'
';'
'PROCEDURE'
ident
';'
block
statement
ident
':='
expression
'IF'
condition
'THEN'
statement
'WHILE'
condition
'DO'
statement
'BEGIN'
statement
'END'
';'
'CALL'
ident
'?'
ident
'!'
expression
'-'
term
term
'+'
term
'-'
numeral
'('
expression
')'
ident
factor
factor
'*'
factor
'/'
'odd'
expression
expression
'='|'#'|'<'|'>'
'<='|'>='
expression