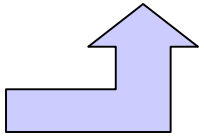
Deklarierte Name in MicroJava

- Klasse → Program ()
  - Konstanten → ConstDecl ()
  - globale Variablen → VarDecl () (level == 0)
  - innere Klassen → ClassDecl ()
    - Felder → VarDecl ()
  - Methoden → MethDecl ()
    - formale Parameter → FormPars ()
    - lokale Variablen → VarDecl () (level > 0)



- Wo werden die Namen deklariert

  = wo werden sie in die Symbolliste eingefügt

Knotenarten der Symbolliste (1)


```
class Obj {
  static final int Con=0, Var=1, Type=2, Fld=3, Meth=4, Prog=5;

  int kind;        // Art des Objekts: Con, Var, Typ, Fld, Meth, Prog
  String name;
  Struct type;
  Obj next;        // Zeiger auf nächstes Objekt
  int adr;         // Con: Wert;   Meth, Var, Fld: Adresse
  int level;       // Var: Deklarationsstufe;   Meth: Anzahl der Parameter
  Obj locals;      // Meth: Referenz auf lokale Variablen der Methode
}
```

Knotenarten der Symbolliste (2)

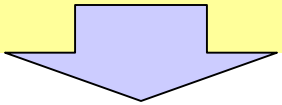

```
class Struct {
  static final int None=0, Int=1, Char=2, Arr=3, Class=4;

  int kind;          // Art des Typs: None, Int , Char, Arr, Class
  Struct elemType;   // Arr: Elementtyp
  int n;             // Class: Anzahl der Felder
  Obj fields;        // Class: Liste der Felder
}

class Scope {
  Scope outer;       // Referenz auf äußeren Gültigkeitsbereich
  Obj locals;        // Symbolliste dieses Gültigkeitsbereichs
  int nVars;         // Anzahl d. Variablen dieses Gültigkeitsbereichs
}
```

Symbollisten-Klasse *Tab*


```
class Tab {
  static final Struct noType, intType,
                      charType, nullType;        // predefined types
  static final Obj noObj;                        // predefined objects
  static Obj chrObj, ordObj, lenObj;             // predefined objects

  static Scope topScope;          // current scope
  static int level;               // nesting level of current scope

  static void init ();
  static void openScope ();
  static void closeScope ();
  static Obj insert (int kind, String name, Struct type);
  static Obj find (String name);
  static Obj findField (String name, Struct type);
}
```

Einbau von semantischen Aktionen zum Füllen der Symbolliste

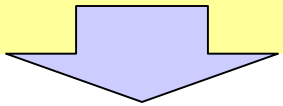

```
/** VarDecl = Type ident { "," ident } ";" . */
private static void VarDecl () {
    Type();
    check(ident);
    while (sym == comma) { scan(); check(ident); }
    check(semicolon);
}
```



```
private static void VarDecl () {
    Struct type = Type();
    check(ident);
    Tab.insert(Obj.Var, t.string, type);
    while (sym == comma) { scan(); check(ident);
        Tab.insert(Obj.Var, t.string, type);
    }
    check(semicolon);
}
```

Einbau von semantischen Aktionen, die Infos aus Symbolliste verwenden


```
/** Type = ident [ "[" "]" ] . */
private static void Type () {
    check(ident);
    if (sym == lbrack) { scan(); check(rbrack); }
}
```



```
private static Struct Type () {
    Struct type = Tab.noType;
    check(ident);
    Obj o = Tab.find(t.string);
    if (o.kind != Obj.Type)
        Parser.Errors.semError("type expected");
    type = o.type;
    if (sym == lbrack) { scan(); check(rbrack);
        type = new Struct(Struct.Arr, type);
    }
    return type;
}
```

Innere Klassen

Beispiel aus VO-Skript (Folie 5.16):

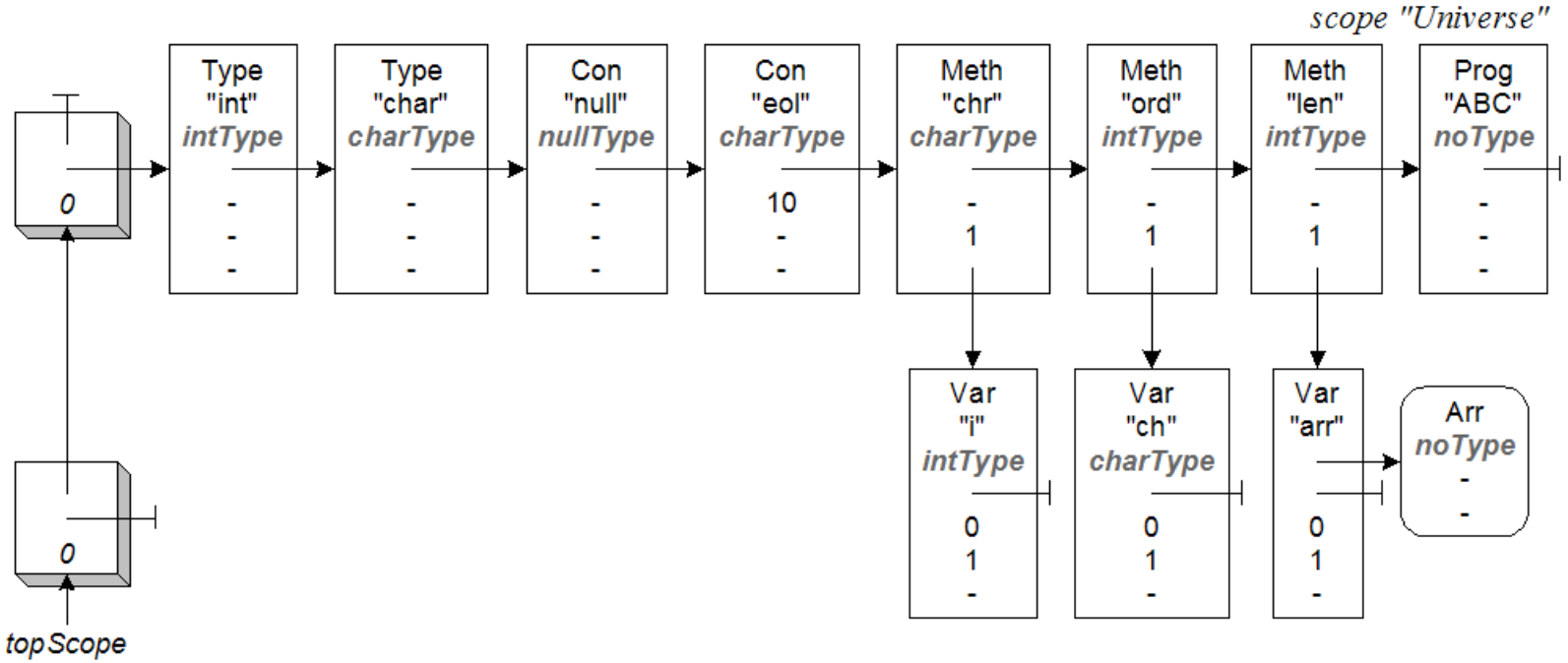
```
class C {
  int x, y, z;
}

C v;
```

```
/** ClassDecl = "class" ident "{" { VarDecl } "}" . */
private static void ClassDecl () {
    scan();
    check(ident);
    Obj c = Tab.insert(Obj.Type, t.string, new Struct(Struct.Class));
    check(lbrace);
    Tab.openScope();
    while (sym == ident) VarDecl();
    check(rbrace);
    c.type.fields = Tab.topScope.locals;
    c.type.n = Tab.topScope.nVars;
    for (Obj cur = c.type.fields; cur != null; cur = cur.next)
       cur.kind = Obj.Fld;  // change from Obj.Var to Obj.Fld
    Tab.closeScope();
}
```

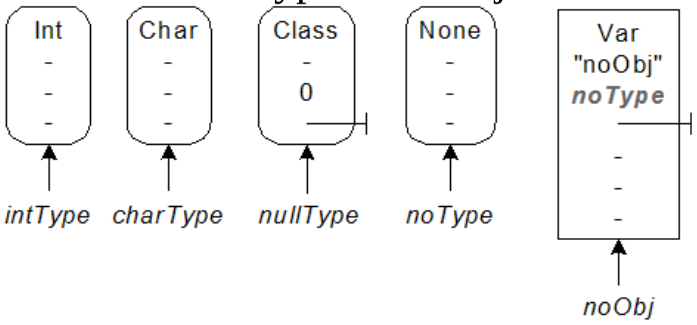
Beispiel: Symbollistenaufbau


```
class ABC (** 1 **)
        char[] c;
        int max;
        char npp;
{
        int put (** 2 **) (int x)
        { (** 3 **)
                x++;
                print(x, 5);
                npp = 'C';
                return x;
        } (** 4 **)
} (** 5 **)
```

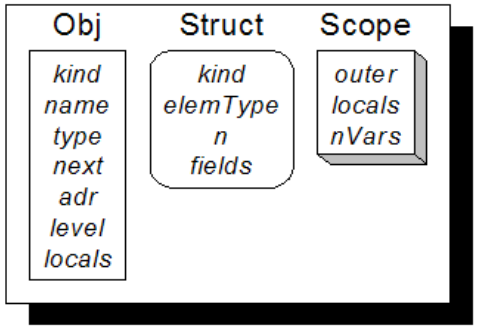
Bsp: Bei Punkt (** 1 **)



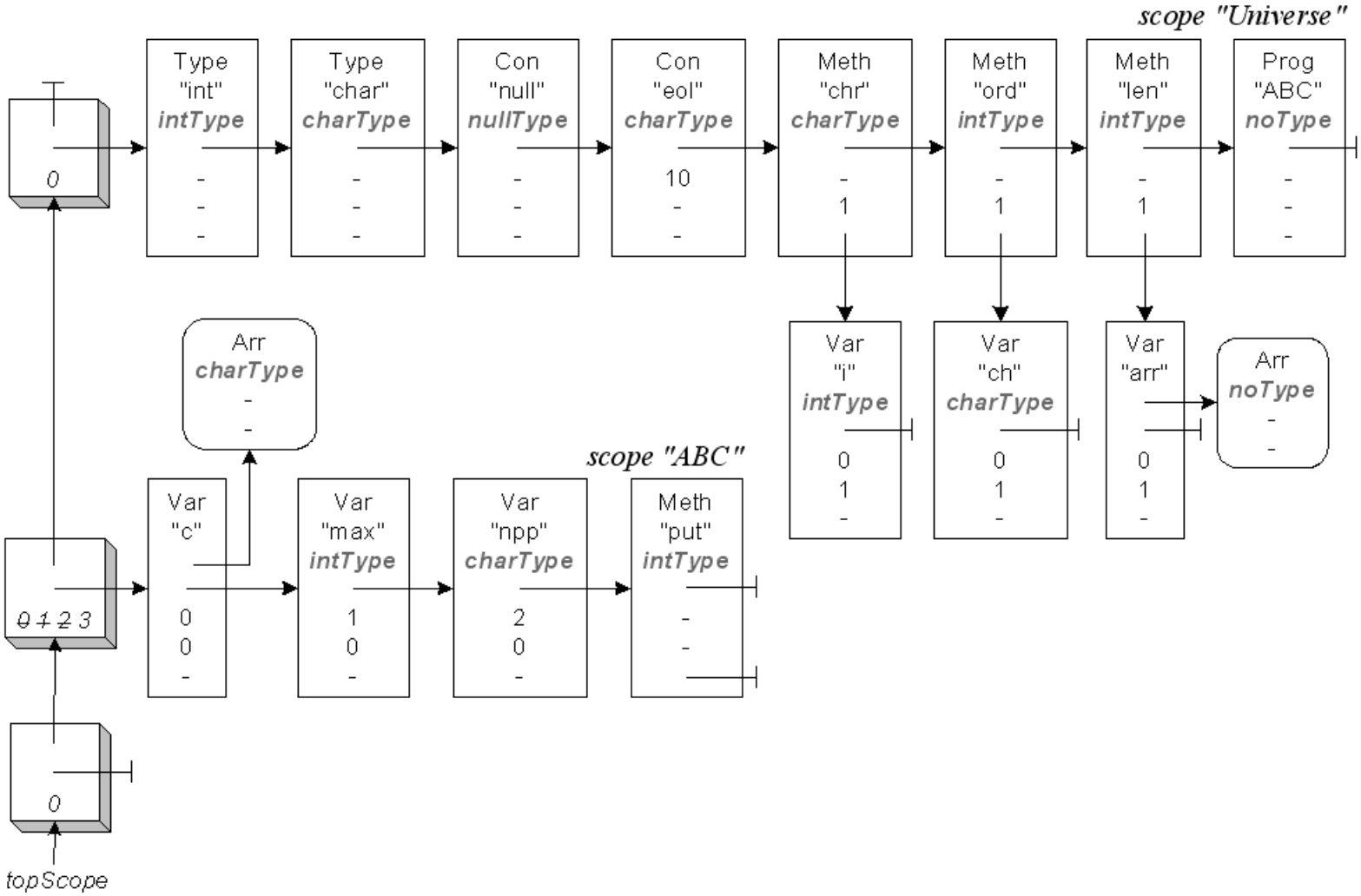
Vordefinierte Typen und Objekte:



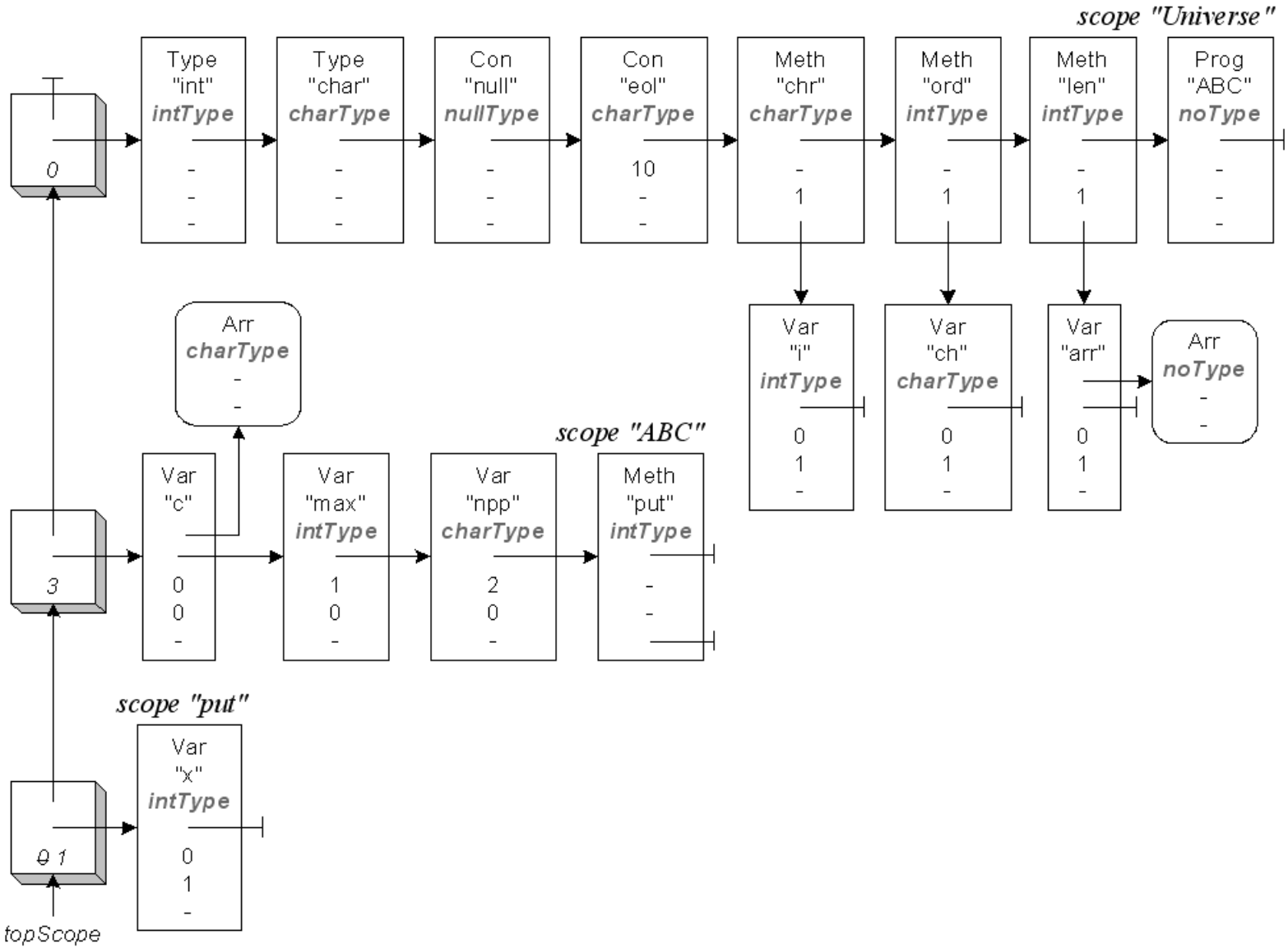
Struktur der 3 Knotenarten:



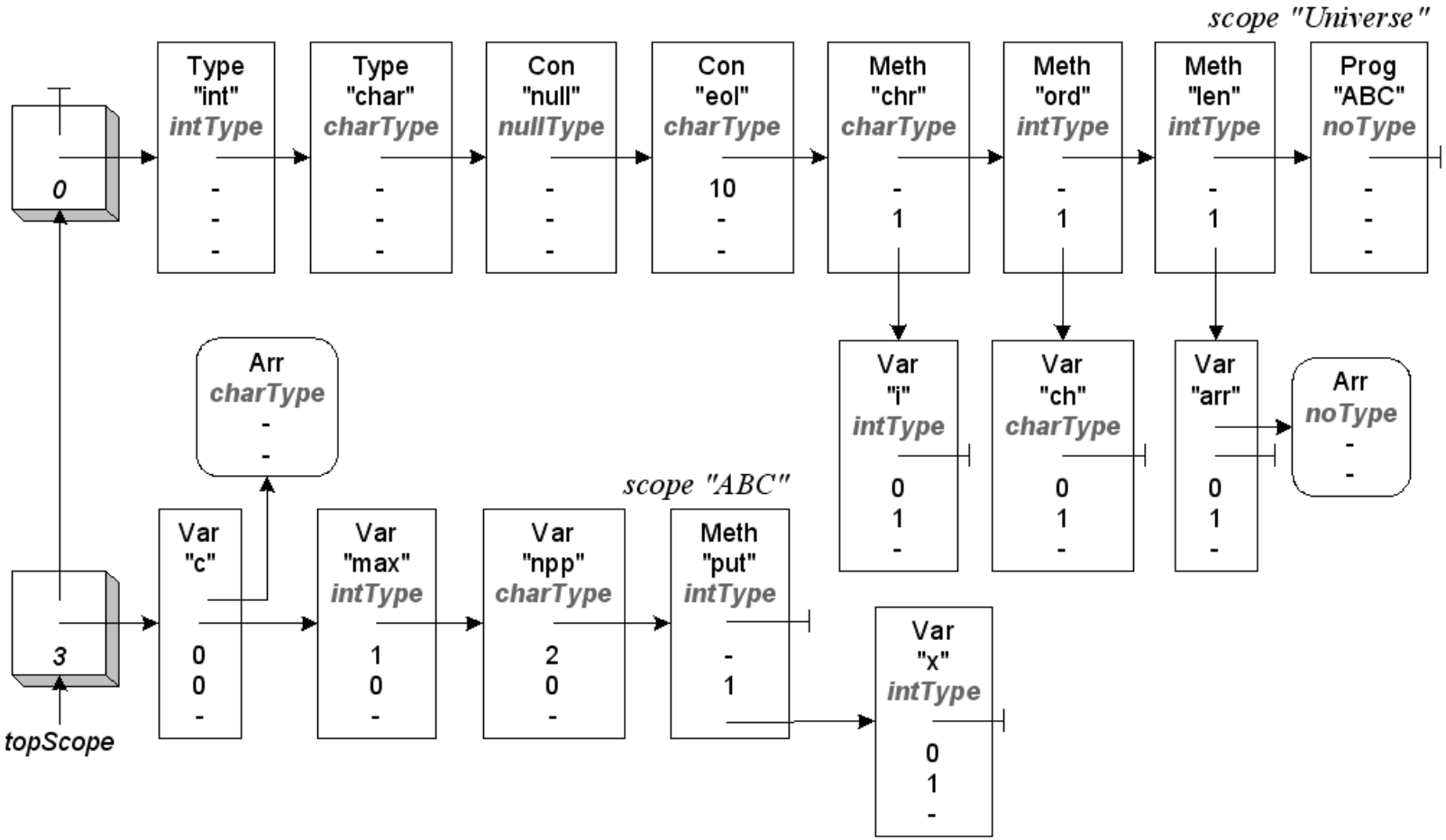
Bsp: Bei Punkt (** 2 **)



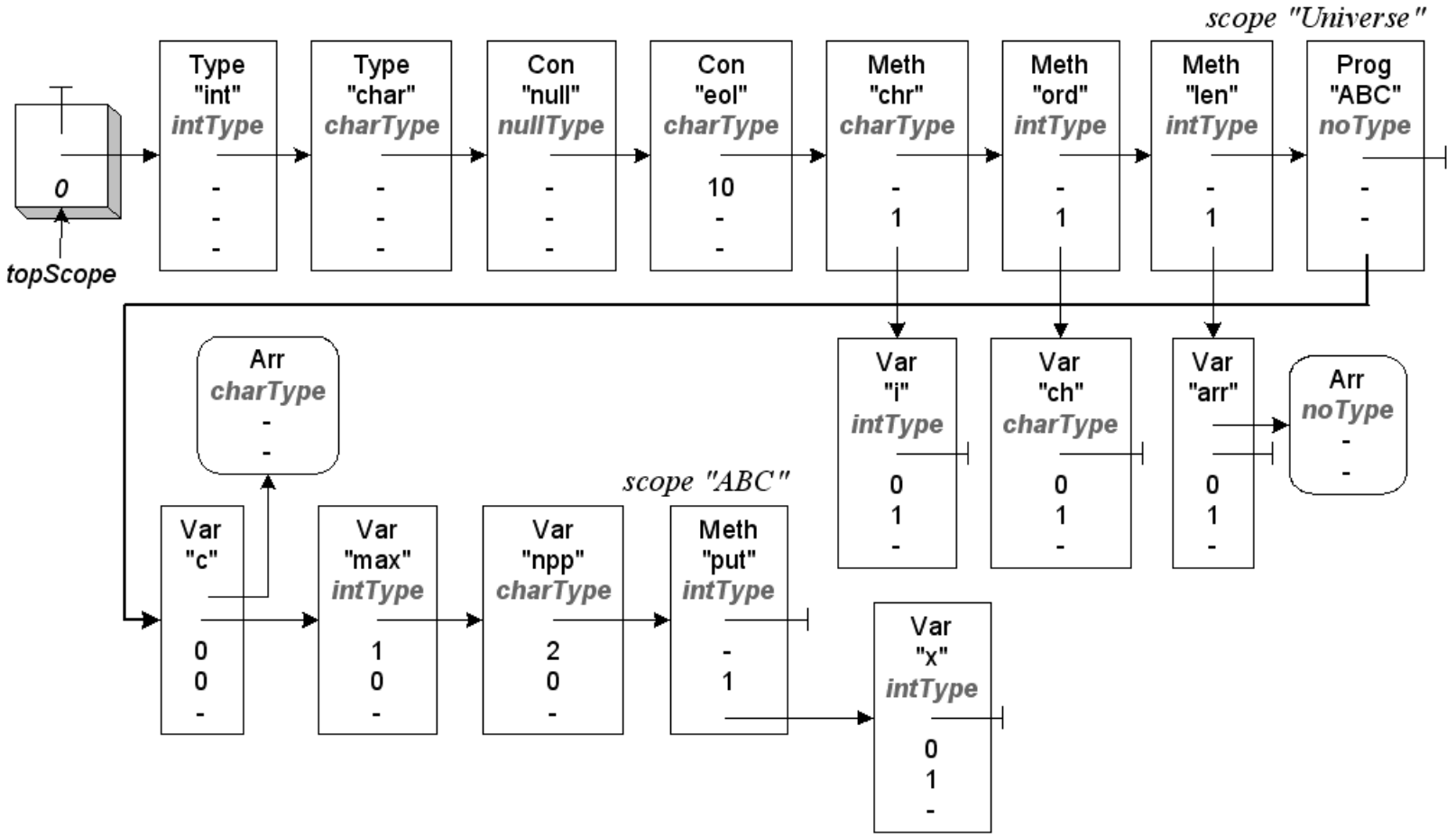
Bsp: Bei Punkt (** 3 **)



Bsp: Bei Punkt (** 4 **)



Bsp: Bei Punkt (** 5 **)



UE 4: Symbolliste & Fehlerbehandlung

UB-UE4-Angabe.zip

- Implementierung:
  - Token.java: toString() gibt nun ALLE Felder aus! (für klarere Failure-Meldungen)
  - Scanner.java: Gerüst verwendet nun Parser.Errors
  - Parser.java: Gerüst + innere Klasse Errors
  - Compiler.java: verwendet nun Parser.Errors
  - Symbollistenklassen:
    - Obj.java, Struct.java, Scope.java: vollständige Implementierungen (müssen nicht mehr verändert werden)
    - Tab.java: Gerüst für Symbollistenverwaltungsklasse
- Testfälle:
  - SymTabTest.java: spezielle Tests für Symbolliste
  - ParserTest.java: zusätzliche Symbollistentests & semantische Checks