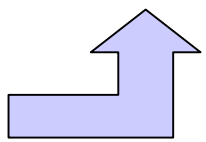
Deklarierte Name in MicroJava

- 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*

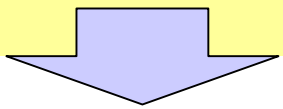

```
class Tab {
  static final Struct noType, intType,
                      charType, nullType;        // predefined types
  static final Obj noObj;                        // predefined objects
  static Obj chrObj, ordObj, lenObj;             // predefined objects

  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

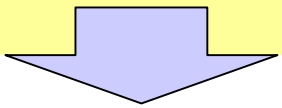

```
/** VarDecl = Type ident { "," ident } ";" . */
private static void VarDecl () {
    Type();
    check(ident);
    while (sym == comma) { scan(); check(ident); }
    check(semicolon);
}
```



```
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); }
}
```

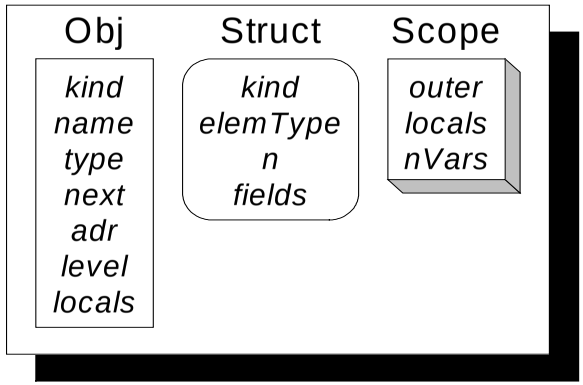


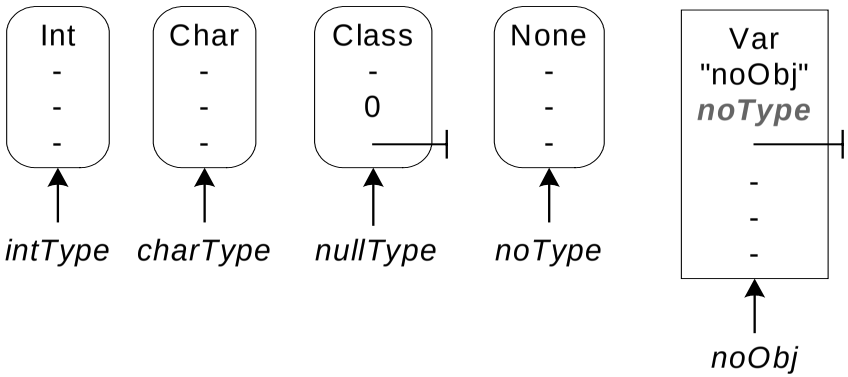
```
private static Struct Type () {
    Struct type = Tab.noType;
    check(ident);
    Obj o = Tab.find(t.string);
    if (o.kind != Obj.Type) semError("type expected");
    type = o.type;
    if (sym == lbrack) { scan(); check(rbrack);
        type = new Struct(Struct.Arr, type);
    }
    return type;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

Beispiel: Symbollistenaufbau f. Klasse ABC

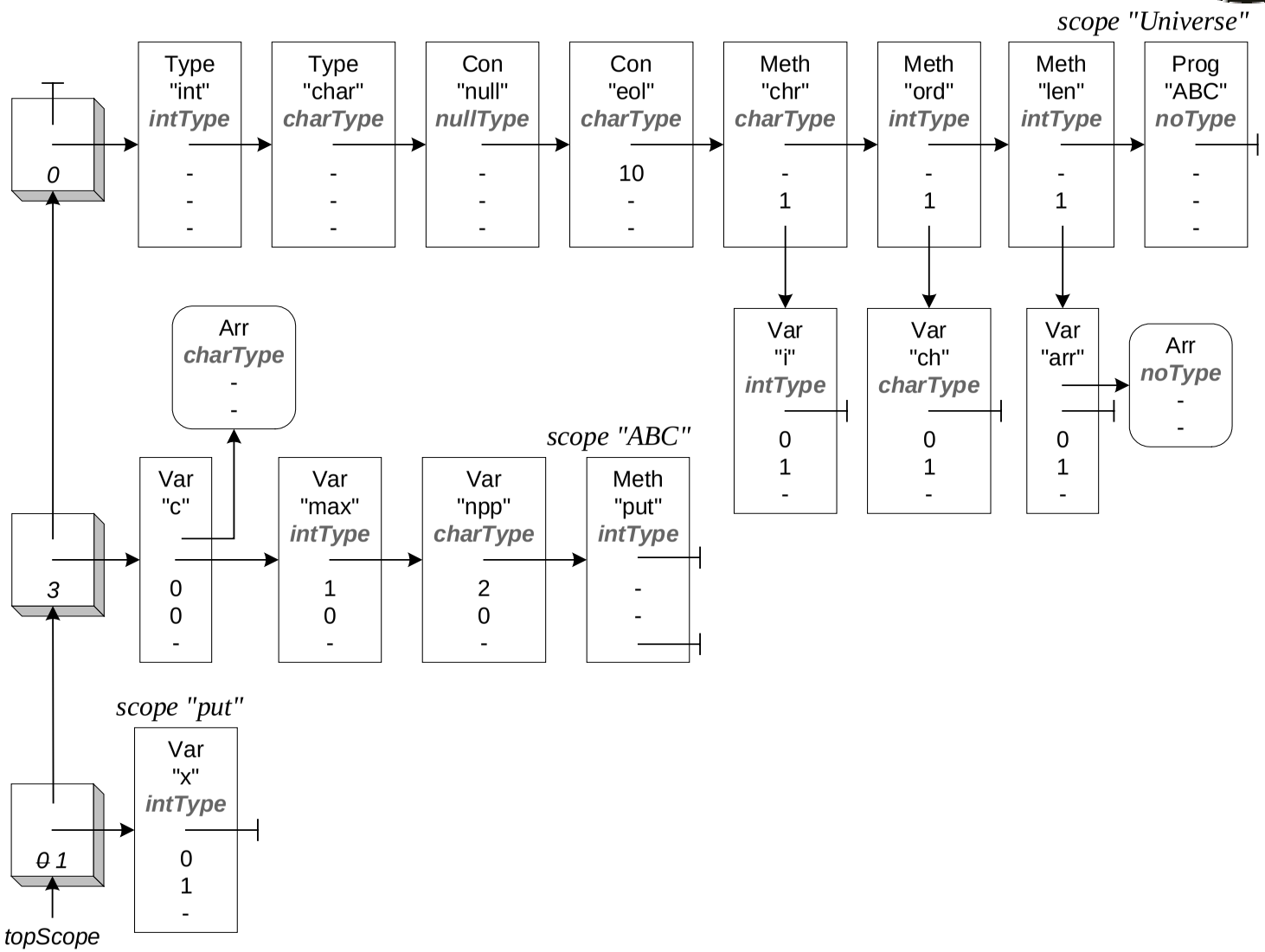

```
class ABC (** 1 **)
        char[] c;
        int max;
        char npp;
{
        int put (** 2 **) (int x)
        { (** 3 **)
                x++;
                print(x, 5);
                npp = 'C';
                return x;
        } (** 4 **)
} (** 5 **)
```

Beispiel: vordefinierte Typen und Objekte

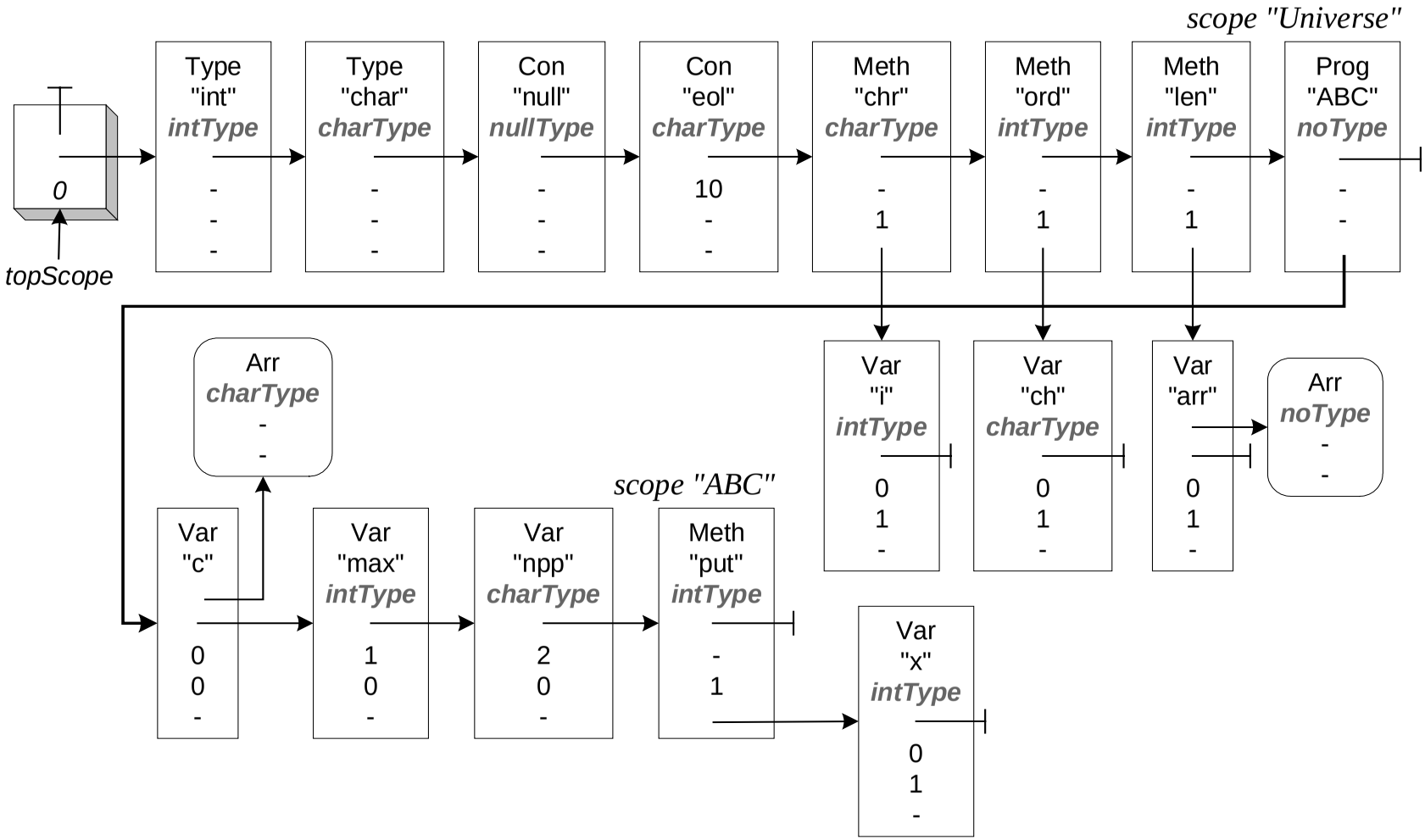




Beispiel: bei Position (** 3 **)



Beispiel: bei Position (** 5 **)



UE 4: Symbolliste & Fehlerbehandlung

UB-UE04-Angabe.zip

- Implementierung:
  - Scanner.java: Gerüst (wie bei UE 2), verwendet Parser.Errors
  - Parser.java: Gerüst (wie bei UE 3) + innere Klasse Errors
  - Compiler.java: verwendet Parser.Errors
  - Symbollistenklassen:
    - Obj.java, Struct.java, Scope.java: vollständige Implementierungen
    - Tab.java: Gerüst für Symbollistenverwaltungsklasse
- Testfälle:
  - TestPrintWriter.java: Ausgabe wird jetzt auch an die Konsole (System.out) weitergeleitet
  - SymTabTest.java: spezielle Tests für Symbolliste
  - ParserTest.java: zusätzliche Symbollistentests