

Deklarierte Namen in MicroJava



- Programm
 - Konstanten ConstDecl ()
 - Globale Variablen VarDecl () level = 0
 - Klassen
 - Felder VarDecl () level = 1
 - Methoden
 - Formale Parameter FormPars()
 - Lokale Variablen VarDecl () level = 1

- Wo werden die Namen deklariert
= wo werden sie in die Symbolliste eingefügt

Objektknoten



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

    int    kind;        // Con, Var, Type, Meth, Prog
    String name;
    Struct type;
    Obj    next;
    int    val;         // Con: value
    int    adr;         // Var, Meth: address
    int    level;      // Var: 0 = global, 1 = local
    int    nPars;      // Meth: number of parameters
    Obj    locals;     // Meth: parameters and local objects
                    // Prog: global variables, constants,
                    //
                    // classes and methods
}
```

Strukturknoten und Scope-Knoten



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

    int    kind;           // None, Int, Char, Arr, Class
    Struct elementType;   // Arr: element type
    int    n;              // Class: number of fields
    Obj    fields;        // Class: list of fields
}

class Scope {
    Scope outer;          // next outer scope
    Obj    locals;        // list of objects in this scope
    int    nVars;         // number of variables in this scope
}
```

Symboltabelle



```
class Tab {
    public static final Struct
        noType, intType, charType, nullType;
    public Obj noObj, chrObj, ordObj, lenObj;

    public Parser parser; // target for errors
    public Scope topScope; // current scope
    private int level; // nesting level of current scope

    public Tab(Parser parser);
    public void openScope();
    public void closeScope();
    public Obj insert(int kind, String name, Struct type);
    public Obj find(String name);
    public Obj findField(String name, Struct type);
}
```

Füllen der Symbolliste

```
/** VarDecl = Type ident { "," ident } ";" . */
private void VarDecl () {
    Struct type = Type();
    check(ident);
    tab.insert(Obj.Var, t.str, type);
    while (sym == comma) {
        scan();
        check(ident);
        tab.insert(Obj.Var, t.str, type);
    }
    check(semi colon);
}
```

Symbolliste verwenden



```
/** Type = ident [ "[" "]" ] . */
private Struct Type() {
    check(ident);
    Obj o = tab.find(t.str);
    if (o.kind != Obj.Type) error("NO_TYPE");
    Struct type = o.type;
    if (sym == lbrack) {
        scan();
        check(rbrack);
        type = new Struct(Struct.Arr, type);
    }
    return type;
}
```

Innere Klassen



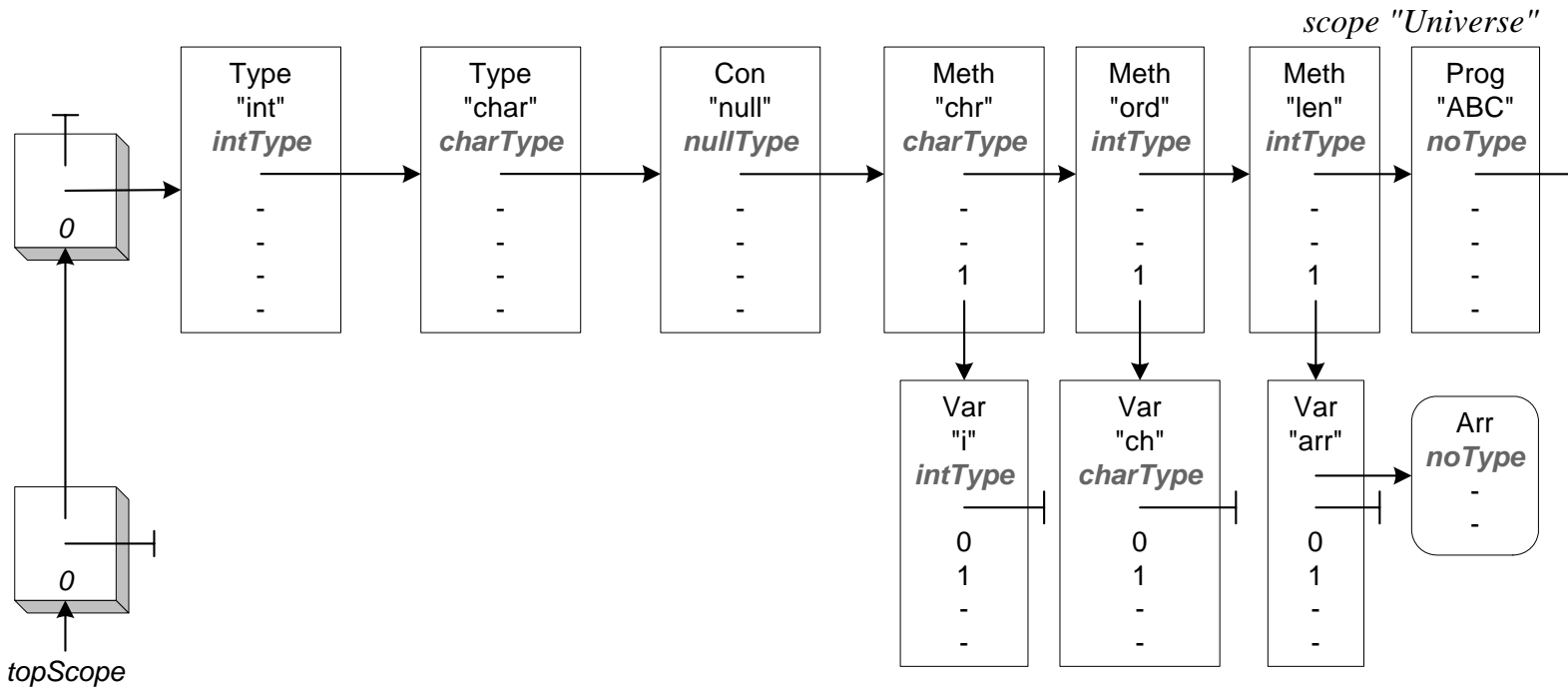
```
/** ClassDecl = "class" ident "{" {VarDecl} "}" . */
private void ClassDecl () {
    check(class_);
    check(ident);
    Obj clazz =
        tab.insert(Obj.Type, t.str, new Struct(Struct.Class));
    check(lbrace);
    tab.openScope();
    while (sym == ident) VarDecl ();
    check(rbrace);
    clazz.type.fields = tab.topScope.locals;
    clazz.type.n = tab.topScope.nVars;
    tab.closeScope();
}
```

Beispiel: Symbollistenaufbau

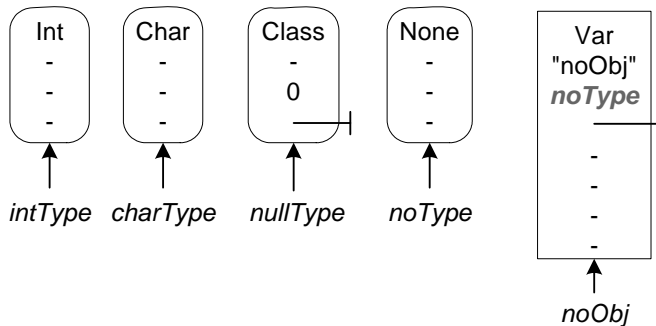


```
program ABC ①
  char[] c;
  int max;
  char npp;
{
  int put ② (int x) { ③
    x++;
    print(x, 5);
    npp = 'C';
    return x;
  } ④
} ⑤
```

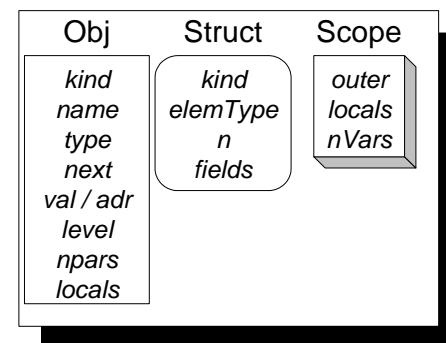

Beispiel: Bei Punkt ①



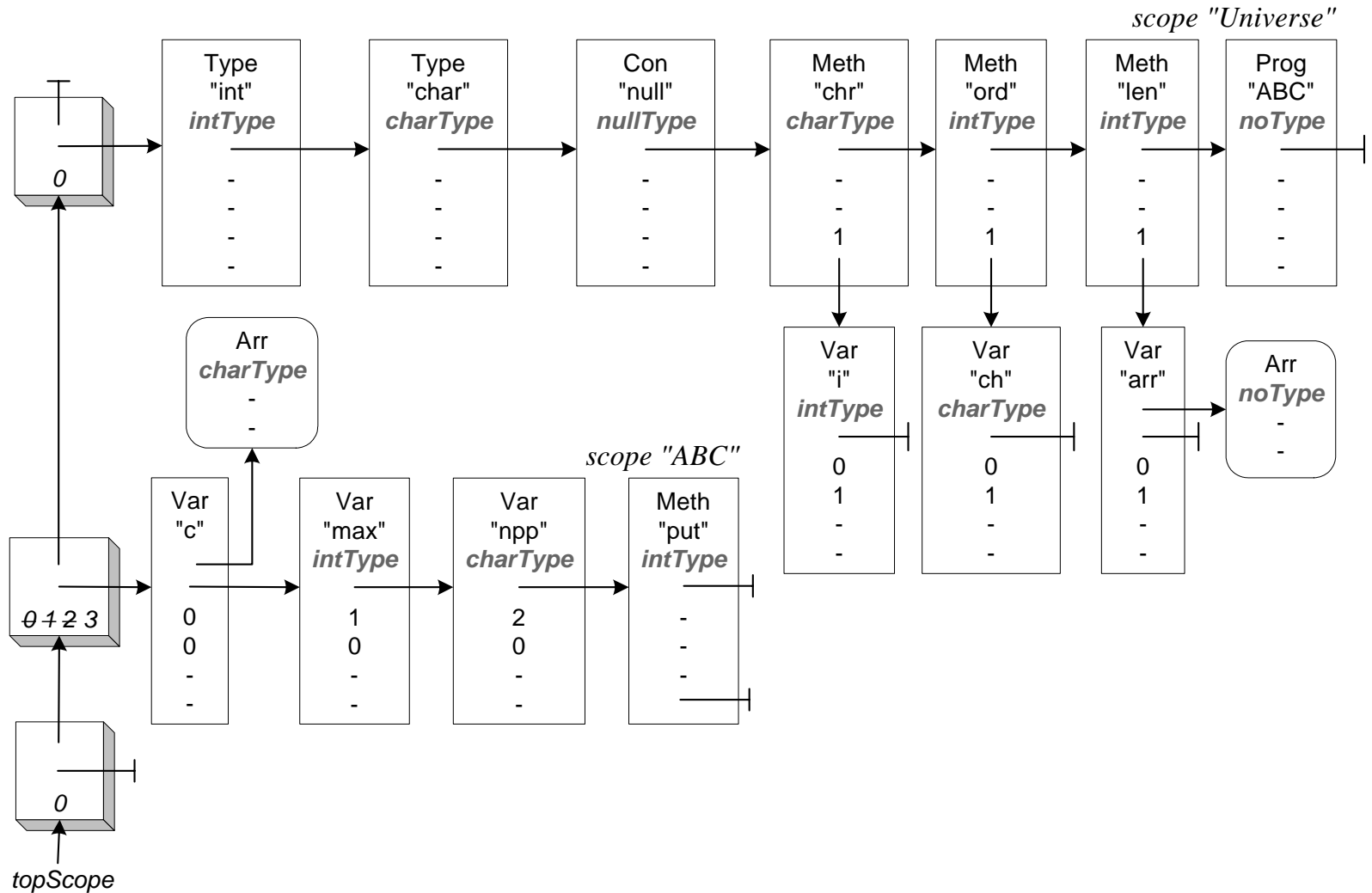
Vordefinierte Typen und Objekte:



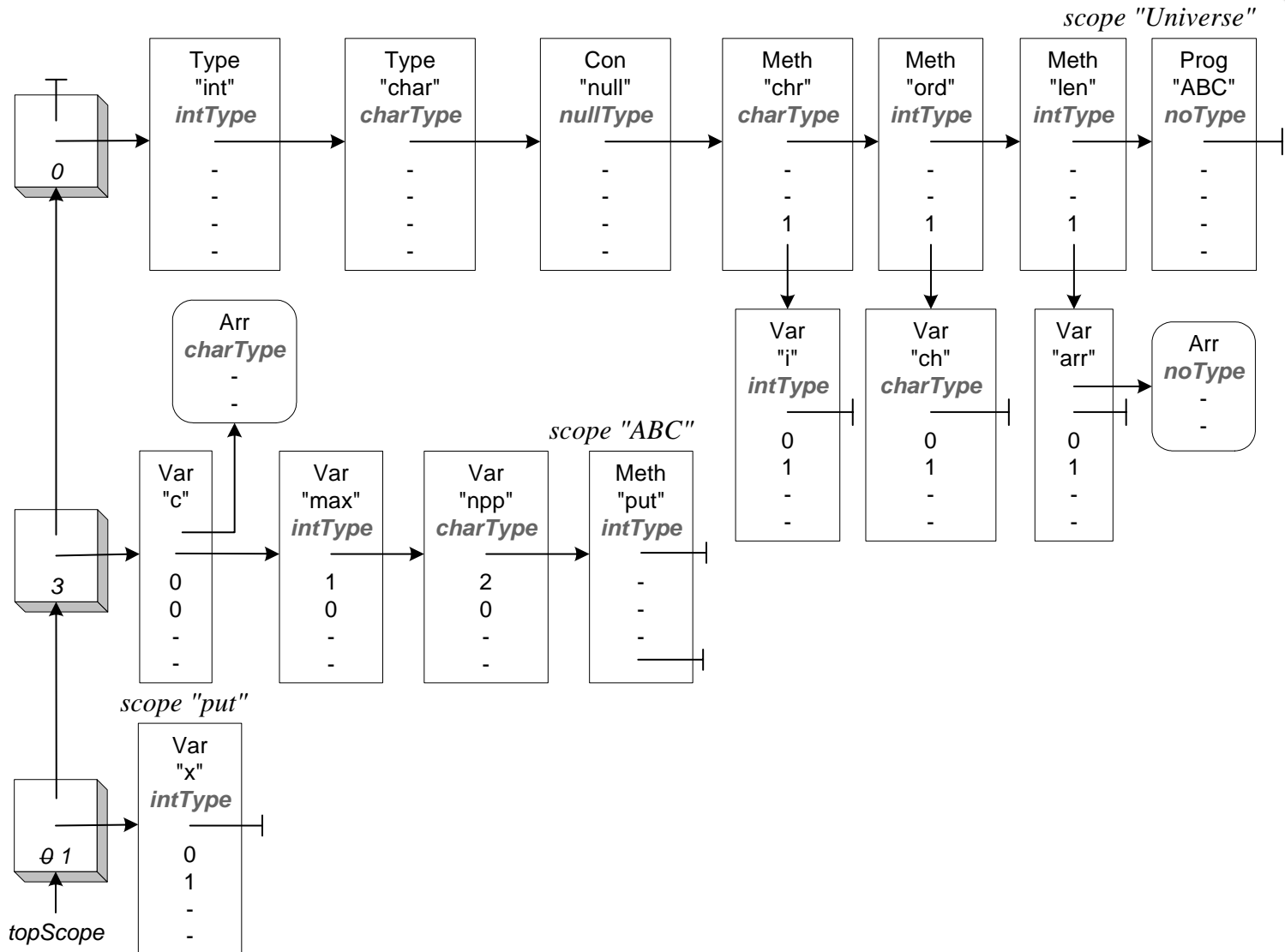
Struktur der 3 Knotenarten:



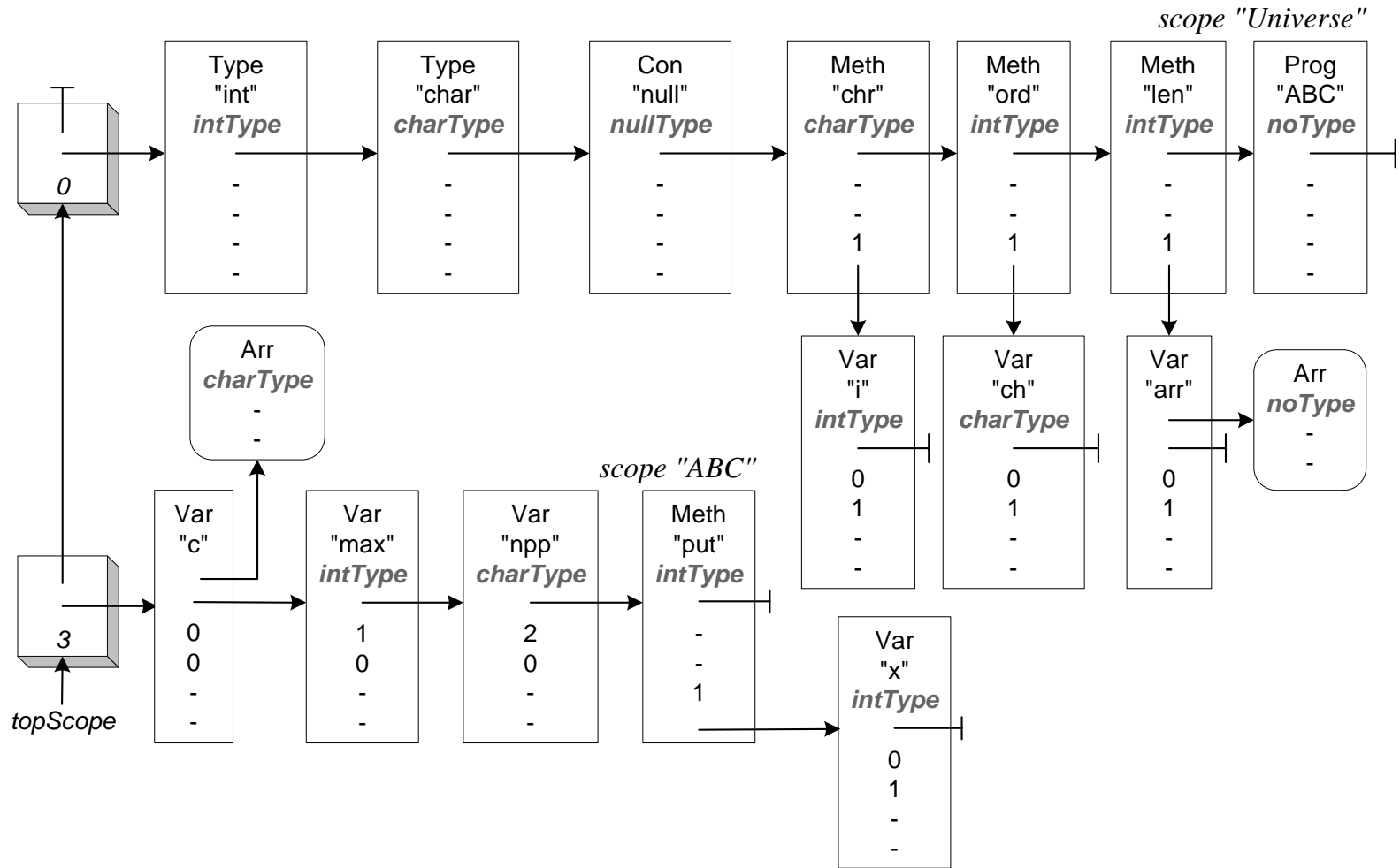
Beispiel: Bei Punkt ②



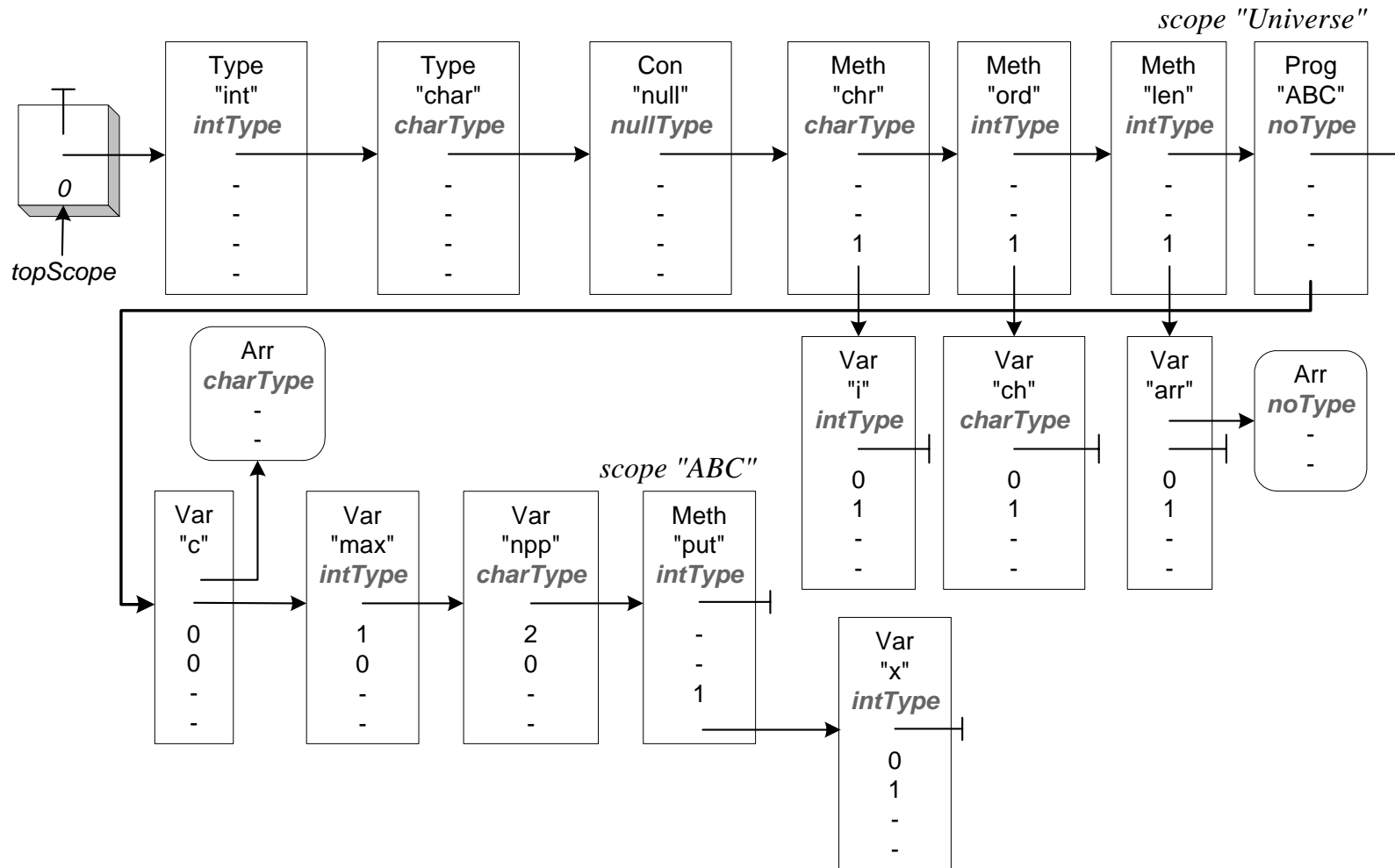
Beispiel: Bei Punkt ③



Beispiel: Bei Punkt ④



Beispiel: Bei Punkt ⑤



UE 4: Symbolliste und Fehlerbehandlung



- Keine neuen Angabe- und Test-Klassen
- Abgabe
 - siehe Abgabeanleitung auf Homepage!
 - elektronisch bis Mi, 06.12.2006, 18:00
 - alle zum Ausführen benötigten Dateien
 - auf Papier bis Mi, 06.12.2006, 18:00
 - Parser.java, Errors.java, Tab.java