

Deklarierte Namen in MicroJava



Programm	Program()	
Konstanten	ConstDecl()	
Globale Variablen	VarDecl()	level = 0
Klassen	ClassDecl()	
Felder	VarDecl()	level = 1
Methoden	MethodDecl(clazz)	
Formale Parameter	FormPars()	
Lokale Variablen	VarDecl()	level = 2
Methoden	MethodDecl(null)	
Formale Parameter	FormPars()	
Lokale Variablen	VarDecl()	level = 1

Wird ein Name deklariert, wird er in die Symbolliste eingefügt

Objektknoten



```
class Obj {  
  enum Kind {  
    Con, Var, Type, Meth, Prog  
  }  
  
  Kind kind;  
  String name;  
  Struct type;  
  int val;    // Con: value  
  int adr;    // Var, Meth: address  
  int level;  // Var: 0 → global, ≥1 → local  
  int nPars;  // Meth: number of parameters  
  // Meth: parameters and local objects  
  // Prog: global variables, constants,  
  //      classes and methods  
  Map<String, Obj> locals;  
}
```

Strukturknoten und Scope-Knoten



```
class Struct {  
    enum Kind {  
        None, Int, Char, Arr, Class  
    }  
    Kind kind;  
    Struct elemType; // Arr: element type  
    Map<String, Obj> fields; // Class: list of fields and methods  
  
    void addMethod(Obj m) {...} // Class: Inserts a method  
    int nrFields() { ... } // Class: number of fields (without methods)  
}  
  
class Scope {  
    Scope outer; // next outer scope  
    Map<String, 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 curScope; // current scope  
    private int curLevel; // nesting level of current scope  
  
    public Tab(Parser parser);  
}  
  
class TabImpl extends Tab {  
    public void openScope();  
    public void closeScope();  
    public Obj insert(Obj.Kind kind, String name, Struct type);  
    public Obj find(String name);  
    public Obj findField(String name, Struct type);  
}
```

Beispiel: Symbollistenaufbau



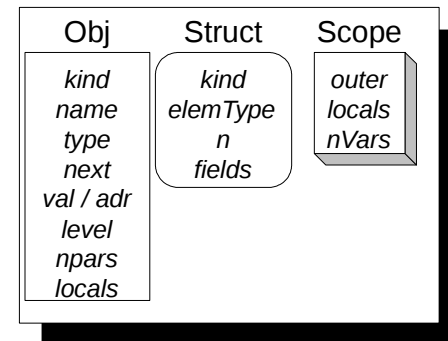
```
program ABC ①  
  class C { ②  
    char[] c;  
    int max;  
    char npp; ③  
    public int put ④ (int x) { ⑤  
      x++;  
      print(x, 5);  
      npp = 'C';  
      return x;  
    } ⑥  
  } ⑦  
} ⑧
```

Beispiel: Symbollistenaufbau

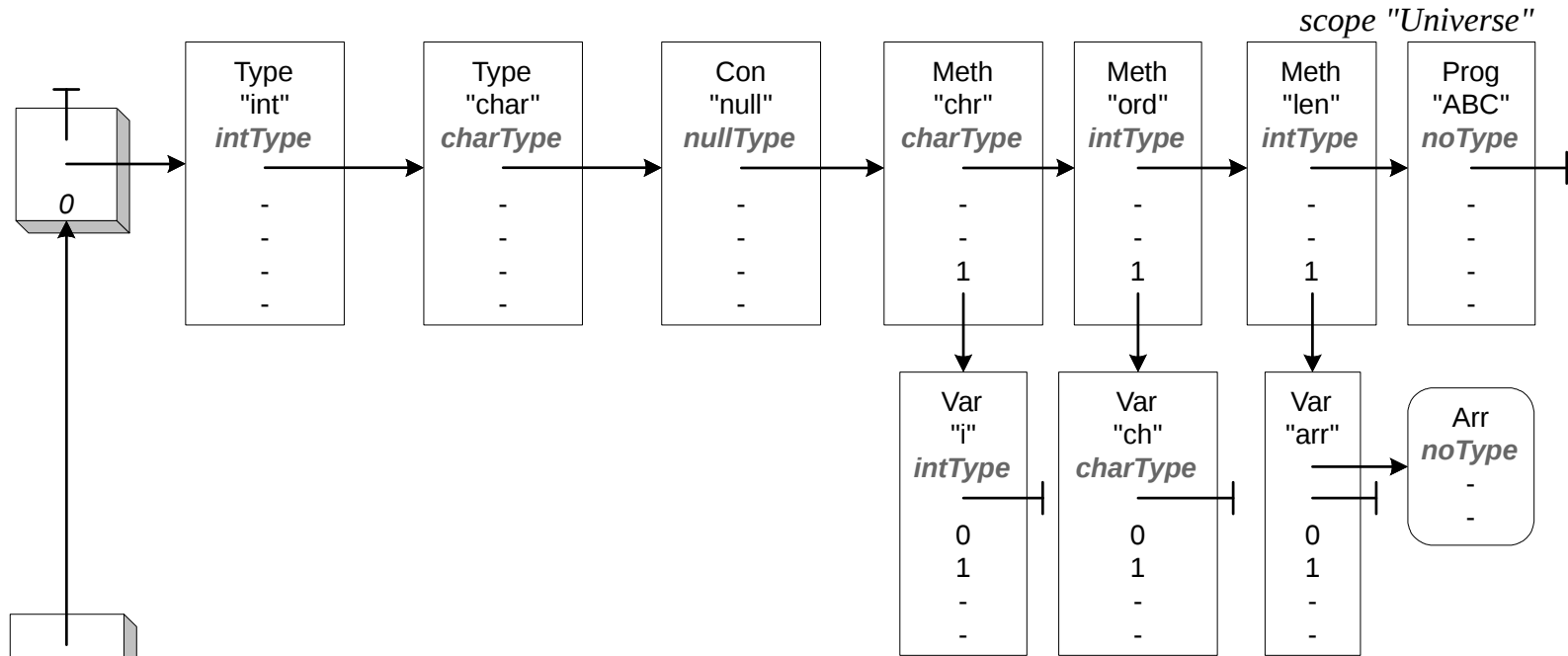


```
program ABC ①  
  class C { ②  
    char[] c;  
    int max;  
    char npp; ③  
    public int put ④ (int x) { ⑤  
      x++;  
      print(x, 5);  
      npp = 'C';  
      return x;  
    } ⑥  
  } ⑦  
}
```

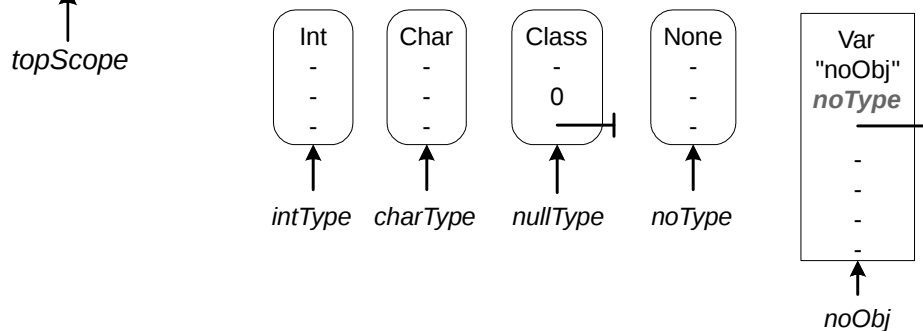
Struktur der 3 Knotenarten:



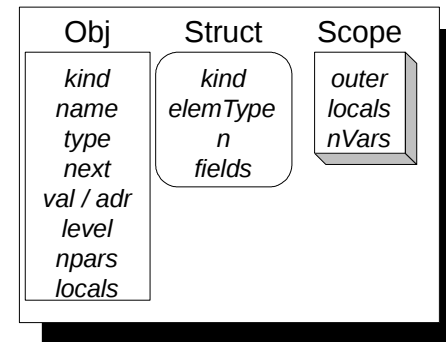
Beispiel: Bei Punkt ①



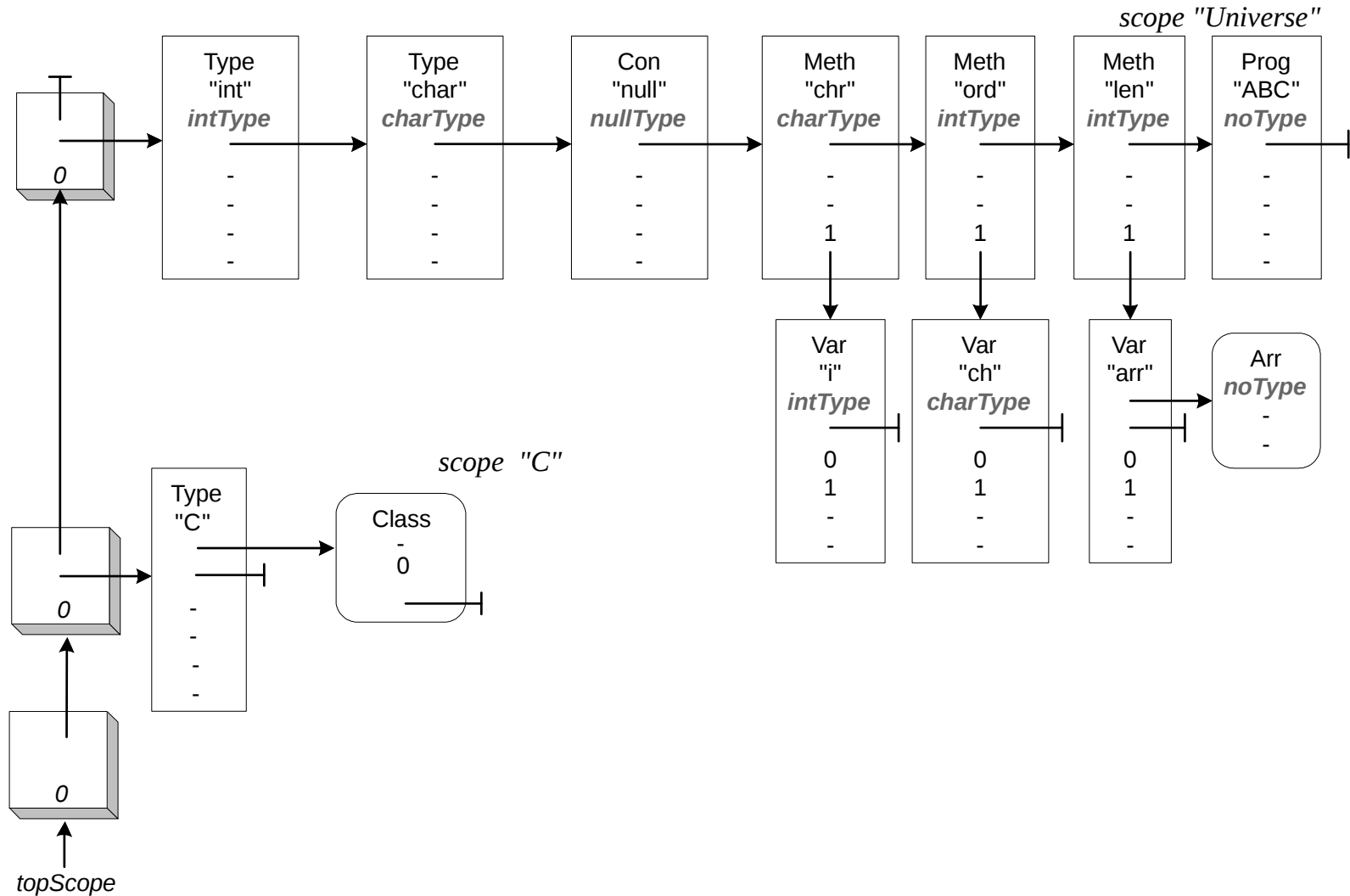
Vordefinierte Typen und Objekte:



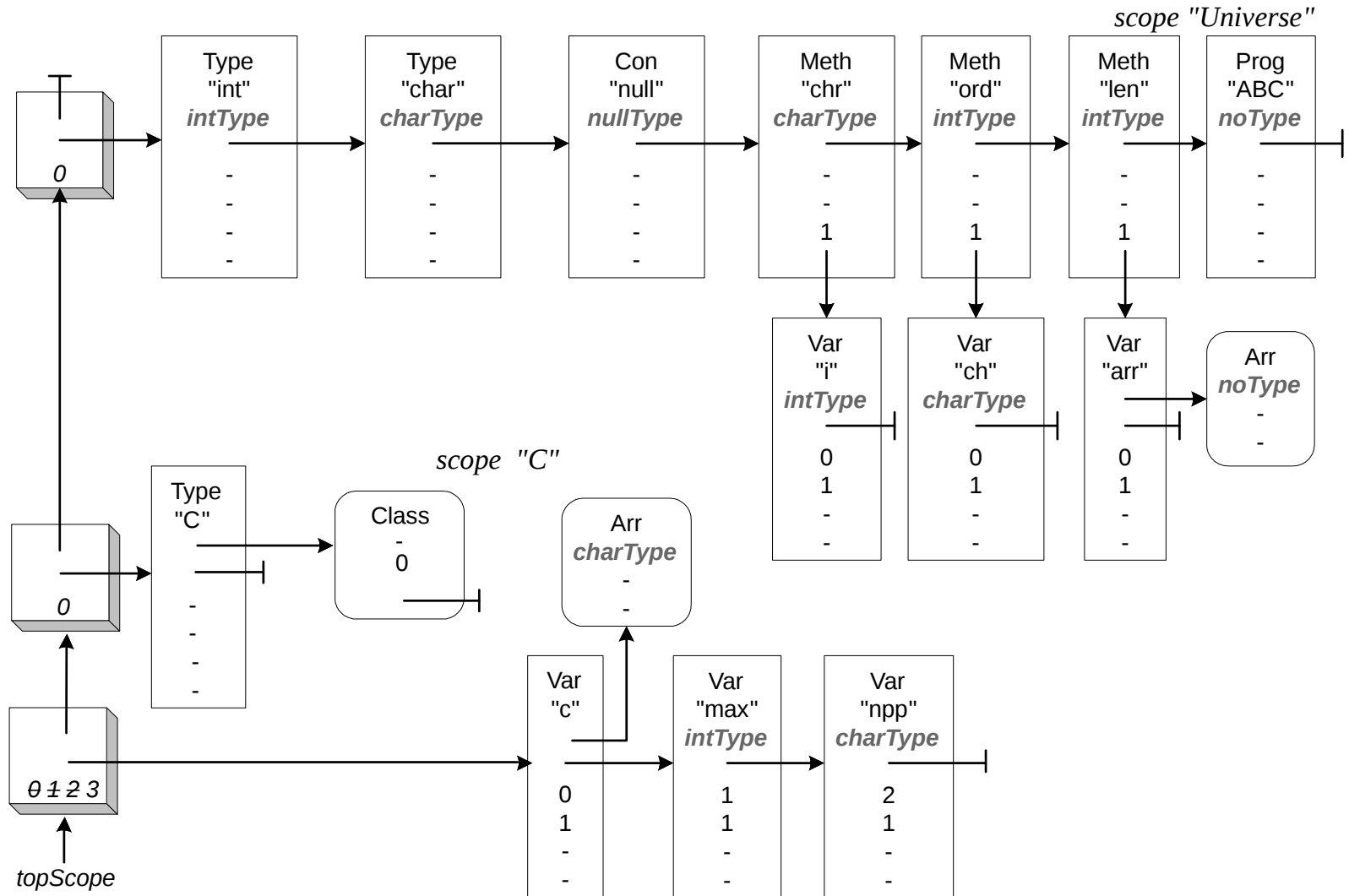
Struktur der 3 Knotenarten:



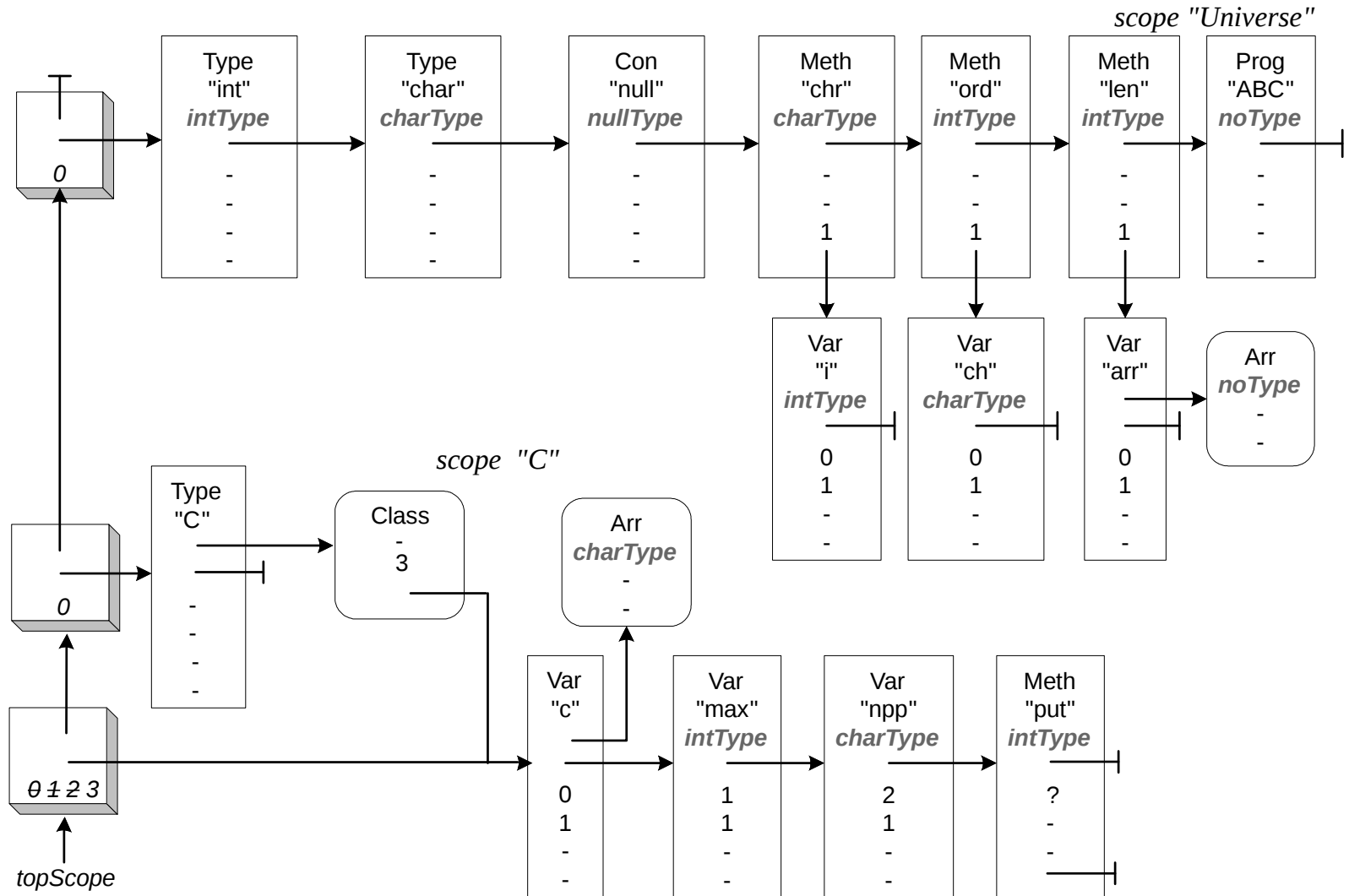
Beispiel: Bei Punkt ②



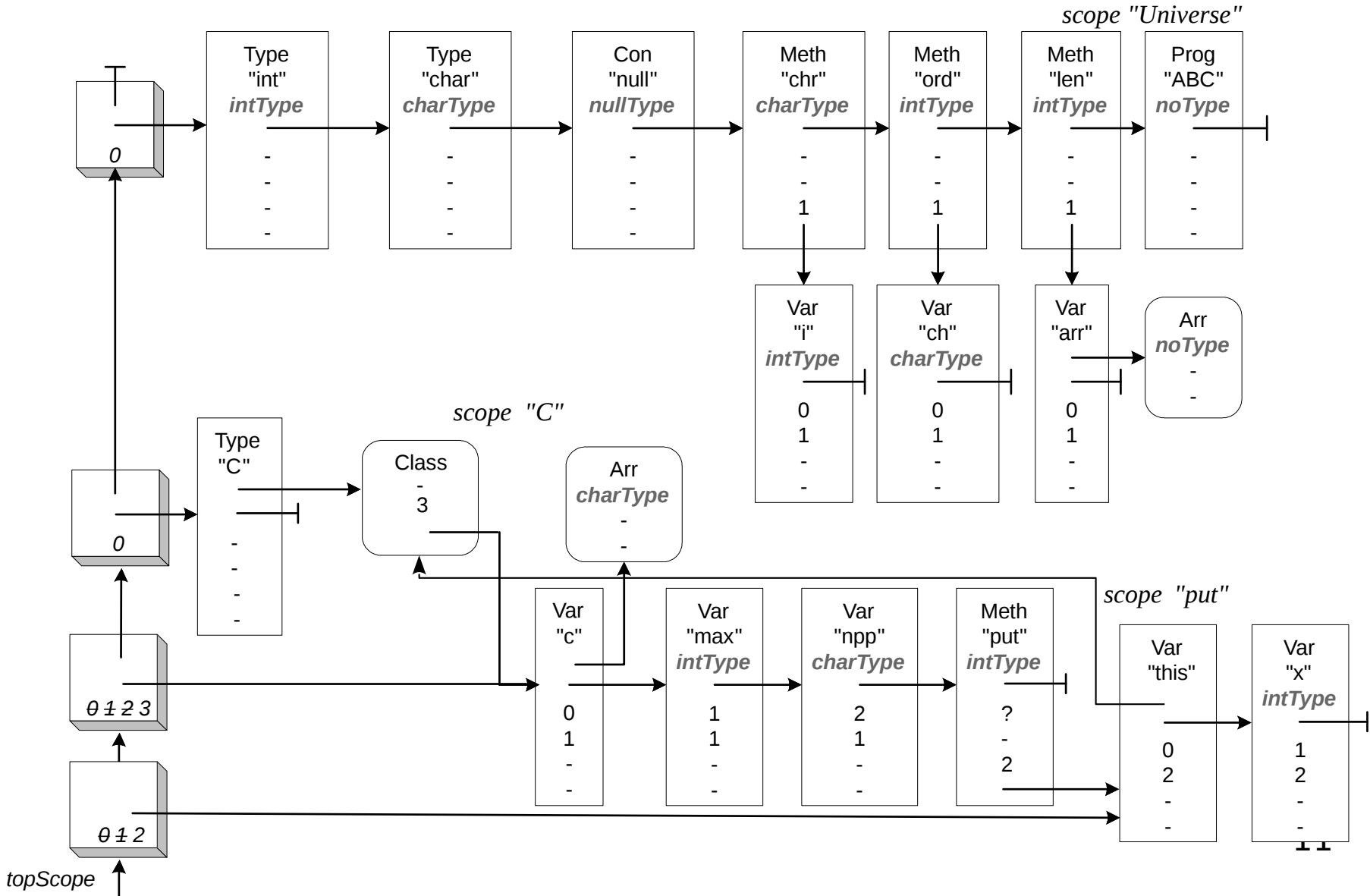
Beispiel: Bei Punkt ③



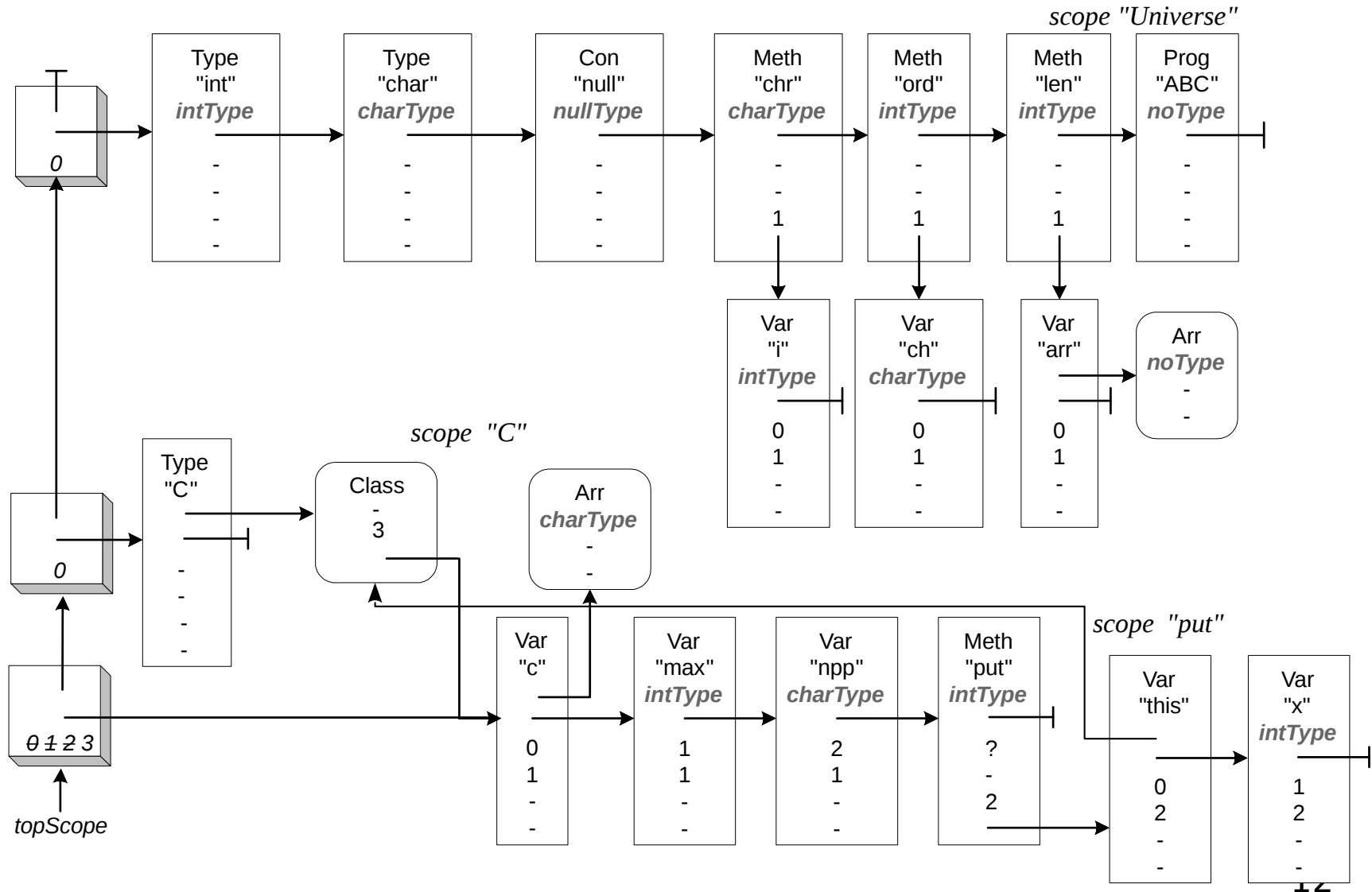
Beispiel: Bei Punkt ④



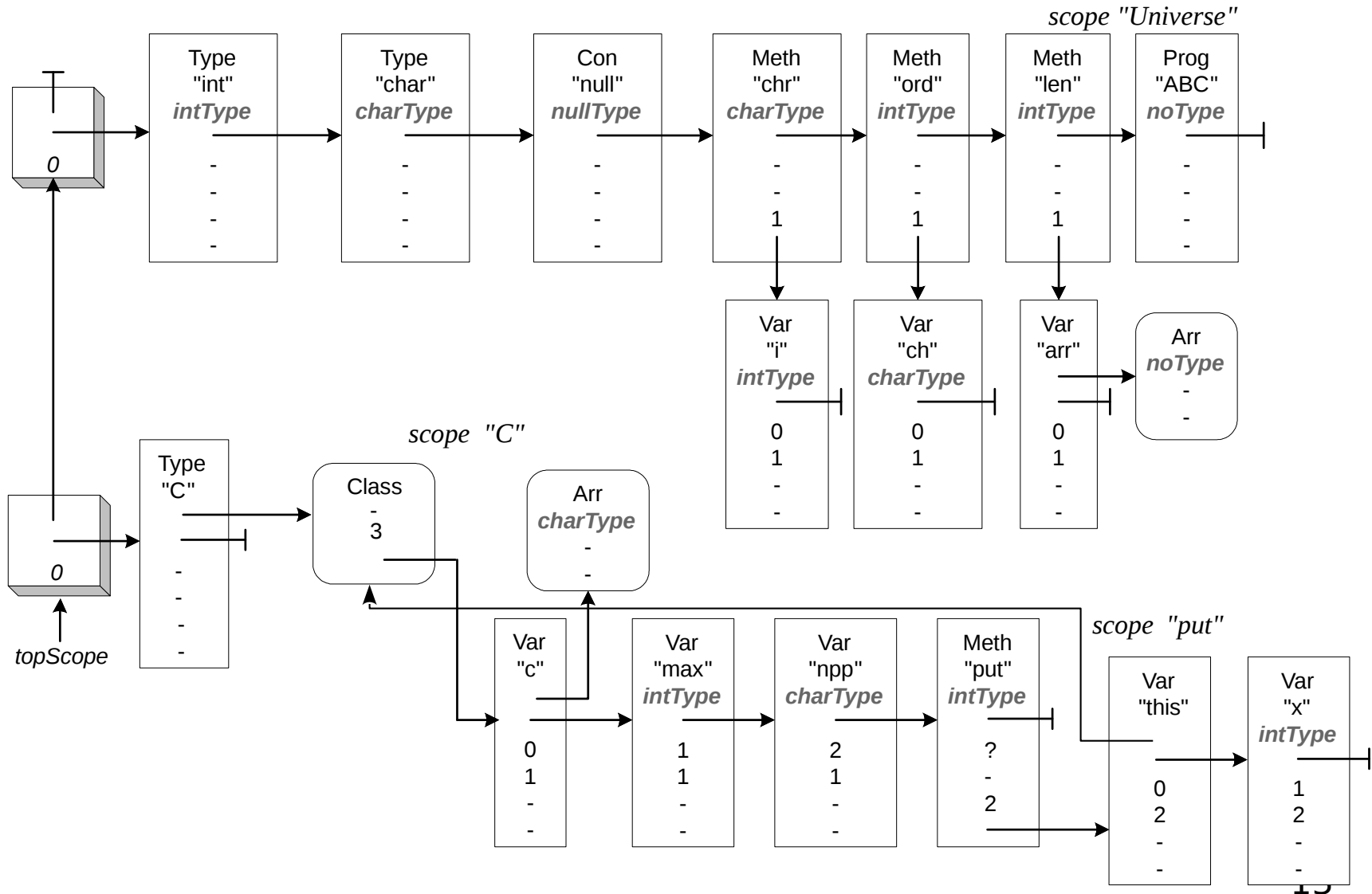
Beispiel: Bei Punkt ⑤



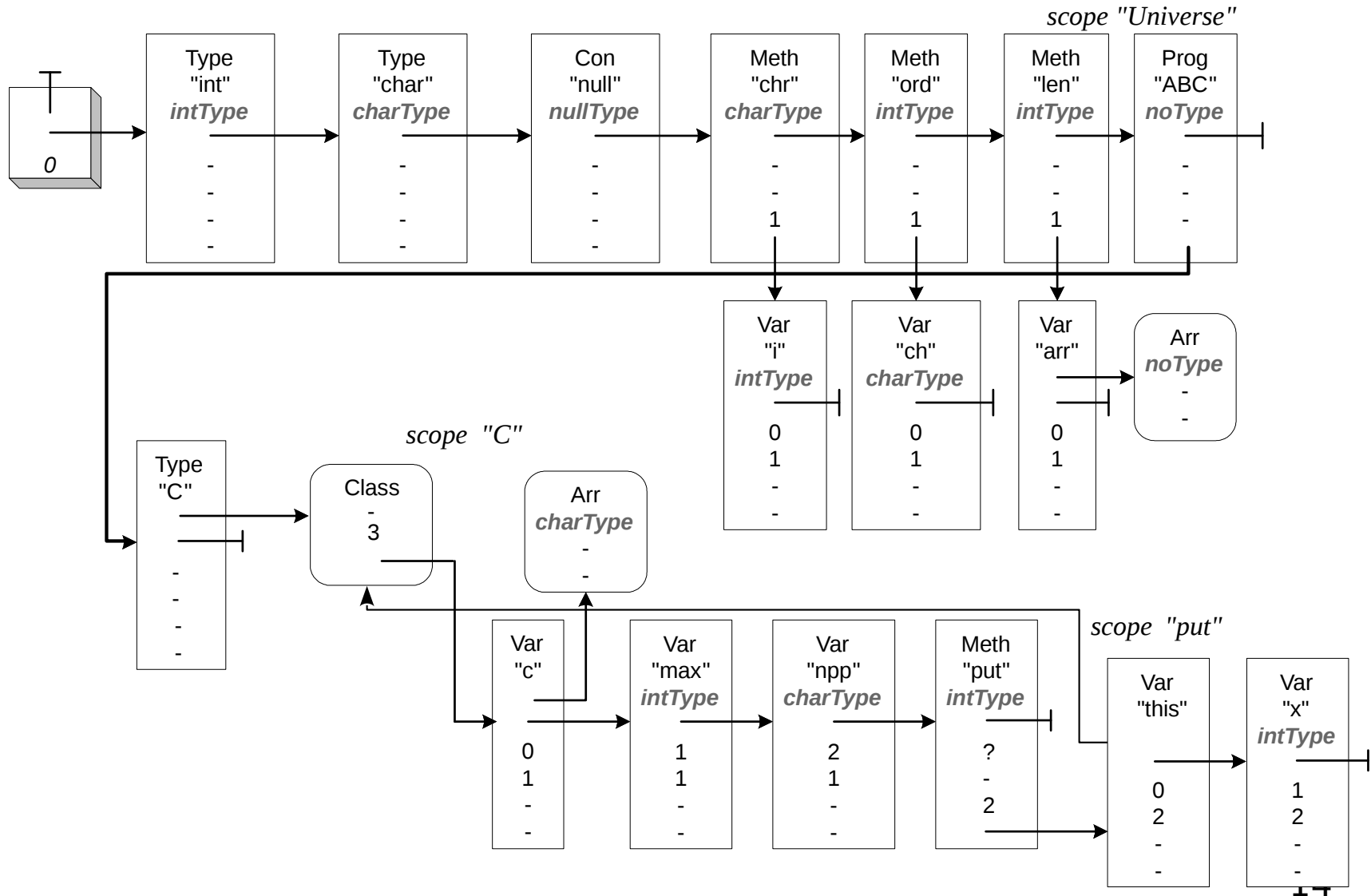
Beispiel: Bei Punkt ⑥



Beispiel: Bei Punkt ⑦



Beispiel: Bei Punkt ⑧



Füllen der Symbolliste



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

Füllen der Symbolliste



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


Symbolliste verwenden



```
/** Type = ident [ "[" "]" ] . */
```

```
private void Type() {
```

```
    check(ident);
```

```
    if (sym == lbrack) {
```

```
        scan();
```

```
        check(rbrack);
```

```
    }
```

```
}
```

Symbolliste verwenden



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

Klassen



```
/** ClassDecl = "class" ident "{" {VarDecl} "}" . */
```

```
private void ClassDecl() {  
    check(class_);  
    check(ident);
```

```
  
    check(lbrace);
```

```
while (sym == ident) {  
    VarDecl();  
}
```

```
  
while (sym == public_) {  
    scan();  
    MethodDecl();  
  
}
```

```
  
    check(rbrace);  
}
```

Klassen



```
/** ClassDecl = "class" ident "{" {VarDecl} "}" . */
private void ClassDecl() {
    check(class_);
    check(ident);
    Obj clazz = tab.insert(Obj.Kind.Type,
        t.str, new Struct(Struct.Kind.Class));
    check(lbrace);
    tab.openScope();
    while (sym == ident) {
        VarDecl();
    }
    if (tab.curScope.nVars() > MAX_FIELDS) {
        error(TOO_MANY_FIELDS);
    }
    clazz.type.fields = tab.curScope.locals();
    while (sym == public_) {
        scan();
        Obj m = MethodDecl(clazz);
        clazz.type.addMethod(m);
    }
    tab.closeScope();
    check(rbrace);
}
```

MethodDecl



```
// MethodDecl = ( Type | "void" ) ident "(" [ FormPars ] ")" ...  
private void MethodDecl() {
```

```
    if (sym == ident) {  
        Type();  
    } else if (sym == void_) {  
        scan();  
    } else {  
        error(METH_DECL);  
    }  
    check(ident);
```

```
    check(lpar);
```

```
    if (sym == ident) {  
        FormPars();  
    }
```

```
    ...
```

MethodDecl

```
// MethodDecl = ( Type | "void" ) ident "(" [ FormPars ] ")" ...
private Obj MethodDecl(Obj clazz) {
    StructImpl type = Tab.noType;
    if (sym == ident) {
        type = Type();
    } else if (sym == void_) {
        scan();
    } else {
        error(METH_DECL);
    }
    check(ident);
    String methodName = t.str;
    meth = tab.insert(Obj.Kind.Meth, methodName, type);
    meth.adr = code.pc;
    check(lpar);
    tab.openScope();
    if (clazz != null) { // insert this reference
        tab.insert(Obj.Kind.Var, "this", clazz.type);
    }
    if (sym == ident) {
        FormPars();
    }
    ...
}
```