Bsp: Deklarationen


```
DeclPart        = { ForwardDecl } "{" Body "}" .
ForwardDecl     = "void" ident "(" ")" ";" .
Body            = ... .
```

Damit lassen sich folgende Deklarationen erzeugen:

```
void p1();
void p2();
void p3();
...
{
    ...
}
```

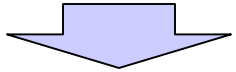
Bsp: Fehler in *ForwardDecl*


```
void p [];
{ ... }
```

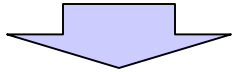
```
                          Erkenne DeclPart
next()  →  voidKW             Erkenne ForwardDecl
                                  voidKW erkannt
next()  →  ident                  ident erkannt
next()  →  lbrack                 ERROR: "( expected"
                                  ERROR: ") expected"
                                  ERROR: "; expected"
                              ERROR: "{ expected"
                              ...
                              ERROR: "} expected"
```

Bsp: First/Follow-BitSets


```
DeclPart        = { ForwardDecl } "{" Body "}" .
ForwardDecl     = "void" ident "(" ")" ";" .
Body            = ... .
```



First(ForwardDecl) = {voidKW}
Follow(ForwardDecl) = First(ForwardDecl) + {lbrace} = {voidKW, lbrace}



```
static BitSet firstFwdDecl = new BitSet();
static BitSet followFwdDecl = new BitSet();

firstFwdDecl.set(voidKW);

followFwdDecl.or(firstFwdDecl);
followFwdDecl.set(lbrace);
followFwdDecl.set(eof);          // Wichtig!!!
```

Bsp: Fehler in *ForwardDecl* (2)


```
void p [];
{ ... }
```

		Erkenne DeclPart
next()	→ voidKW	Erkenne ForwardDecl
		voidKW erkannt
next()	→ ident	ident erkannt
next()	→ lbrack	ERROR: "(expected"
		ERROR: ") expected"
		ERROR: "; expected"
		ERROR: "invalid forward declaration"
next()	→ rpar	
next()	→ semicolon	
next()	→ lbrace	lbrace erkannt
next()	→ ...	Erkenne Body
...		...
next()	→ rbrace	rbrace erkannt

LL(1)-Bedingung

- keine Alternativen mit gleichen terminalen Anfängen
- keine Linksrekursionen

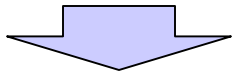
⇨ Bei Top-Down-Analyse:
mit einem Vorgriffssymbol entscheiden,
welche Alternative ausgewählt werden muss.

- Abhilfen:
  - gleiche Anfänge ⇨ Faktorisieren
  - Linksrekursionen ⇨ Umwandlung in Iteration

Regel *Statement*


```
Statement
    = Assignment
     | ProcedureCall
     | Increment | Decrement
     | ... .
```

gut lesbar, aber nicht LL(1), weil alle Alternativen mit ident beginnen



Abhilfe: Faktorisieren

```
Statement
    = Designator
        ( "=" Expr                  // Assignment
        | "(" [ ActPars ] ")"       // ProcedureCall
        | "++" | "--"               // Increment | Decrement
        ) ";"
     | ... .
```