# Fehlerbehandlung


- Panic Mode
  - Abbruch beim ersten Fehler
  - **Übung 3**
- Allgemeine Fangsymbole
  - Synchronisation der restlichen Eingabe mit der Grammatik
  - Parser kennt an jeder Stelle alle gültigen Nachfolge-Symbole
  - Aufwendig
- Spezielle Fangsymbole
  - Synchronisation nur an besonders "sicheren" Stellen.
  - Beispiele: Schlüsselwörter, Strichpunkte, …
  - **Übung 4**

# Beispiel: Deklarationen


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

Damit lassen sich folgende Deklarationen erzeugen:

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

# Beispiel: Deklarationen


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

```
private void DeclPart () {
  while (sym == Token.void_) {
    ForwardDecl();
  }
  check(Token.lbrace); Body(); check(Token.rbrace);
}
```

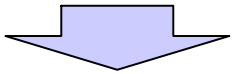
# Bsp: Fehler in *ForwardDecl*

```
void p [);
{ … }
```

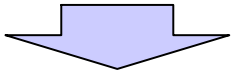
```
                        Erkenne DeclPart
next() → void_             Erkenne ForwardDecl
                              void_ erkannt
next() → ident                ident erkannt
next() → lbrack               ERROR: "( expected"
                                  ERROR: ") expected"
                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

# Bsp: First/Follow-BitSets


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



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



```
private BitSet followFwdDecl = new BitSet();

followFwdDecl.set(Token.void_);
followFwdDecl.set(Token.lbrace);
followFwdDecl.set(Token.eof); // Wichtig!!!
```

# Beispiel: Deklarationen


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

```
private void DeclPart () {
  for (;;) {
    if (sym == Token.void_) ForwardDecl();
    else if (followFwdDecl.get(sym)) break;
    else recoverFwdDecl();
  }
  check(Token.lbrace); Body(); check(Token.rbrace);
}

private void recoverFwdDecl() {
  error("invalid forward declaration");
  do {
    scan();
  } while (!followFwdDecl.get(sym));
}
```

# Bsp: Fehler in *ForwardDecl* (2)


```
void p [);
{ … }
```

```
                        Erkenne DeclPart
next() → void_             Erkenne ForwardDecl
                              void_ erkannt
next() → ident                ident erkannt
next() → lbrack               ERROR: "( expected"
                                  ERROR: ") expected"
                                  ERROR: "; expected"
                              ERROR: "invalid forward decl."
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

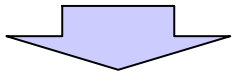
⇨ Bei Top-Down-Analyse:
mit einem Vorgriffssymbol entscheiden,
welche Alternative ausgewählt werden muss.

- 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
  ( AssignOp Expr          // Assignment
  | ActPars                // ProcedureCall
  | "++" | "--“            // Increment | Decrement
  ) ";"
| … .
```

# Beispiel: Kein LL(1)-Konflikt


```
S = a B B B | b C.
B = b B | a C.
C = S S | c.
```

( $S = S_1 \mid S_2.$ , $S_1 = aBBB.$ , $S_2 = bC.$ )
( $B = B_1 \mid B_2.$ , $B_1 = bB.$ , $B_2 = aC.$ )
( $C = C_1 \mid C_2.$ , $C_1 = SS.$ , $C_2 = c.$ )

$$first(S1) \cap first(S2) = \{a\} \cap \{b\} = \{\}$$
$$first(B1) \cap first(B2) = \{b\} \cap \{a\} = \{\}$$
$$first(C1) \cap first(C2) = first(S) \cap \{c\} = \{a, b\} \cap \{c\} = \{\}$$

# Beispiel: LL(1)-Konflikt


```
S = a B B B | b C.   (S = S1 | S2. S1 = aBBB. S2 = bC.)
B = b B | a C d.     (B = B1 | B2. B1 = bB. B2 = aCd.)
C = [ S S | c ].     (C = C1 | C2 | C3. C1 = SS. C2 = c. C3= ε.)
```

$FC1 = first(C_1) = first(S) = \{a, b\}$

$FC2 = first(C_2) = \{c\}$

$FC3 = first(C_3) = follow(C) =$

$= \{d\} \cup follow(S) =$

$= \{d\} \cup first(S) \cup follow(C) =$

$= \{d\} \cup \{a, b\} =$

$= \{a, b, d\}$

$FC1 \cap FC2 = \{\}$

$FC2 \cap FC3 = \{\}$

**$FC1 \cap FC3 = \{a, b\}$**

# Beispiel: LL(1)-Konflikt


```
S = a B B B | b C.
B = b B | a C d.
C = [ S S | c ].
```

Beispielsatz: `a a b b a d a d`

```
S = a B             B B
B =   a C         d
C =     S       S
S =     b C
C =       S …
```

# UE 3: Syntaxanalyse (*Parser*)


- Keine neuen Angabe- und Test-Klassen

- Abgabe
  - siehe Abgabeanleitung auf Homepage!
  - elektronisch bis Mi, 22.11.2006, 18:00
    - alle zum Ausführen benötigten Dateien
  - auf Papier bis Mi, 22.11.2006, 18:00
    - nur Parser.java