

# 6 Fehlerbehandlung

## 6.1 Allgemeine Gesichtspunkte

Fehler bei der Programmerstellung können in allen Phasen des Entwicklungsprozesses auftreten.

### Einteilung

- Spezifikationsfehler (falsche Vorgaben)
- Algorithmische Fehler (Verfahren bietet keine korrekte Lösung)
- Codierungsfehler (fehlerhafte Umsetzung des Algorithmus in ein Programm)
- \* Schreibfehler
- \* Mißachtung von Sprach- und Compilereigenschaften
- Compilerfehler (liegen vor, wenn zu einem korrekten Quellprogramm kein korrektes Zielprogramm erzeugt wird)

Compiler bieten Hilfe nur bei den Fehlerarten \* .

### Mögliche Reaktionen des Compilers im Fehlerfall

- Abbruch der Compilierung (*mit* oder *ohne* Fehlermeldung)
- Fehlerbehandlung (Feststellung und Meldung, Fortsetzung der Compilation)
- Fehlerkorrektur (in einfachen Fehlersituationen können Fehler korrigiert werden)

### Forderung

Angemessene und benutzerfreundliche Fehlerreaktion. Dazu gehören:

- quellsprachenbezogene Identifikation der Fehlerstelle (wo?)
- aussagekräftige Erklärung (was?)
- Vermeidung unnötiger Redundanz in den Fehlermeldungen (wie oft?)

Wie kann man Redundanz vermeiden?

(Fall: "Missing Declaration", mehrfache Ausfertigung vermeiden durch Aufnahme der nichtdeklarierten Größe in die Symboltabelle)

### Einteilung der Fehler (bzgl. Compiler)

- lexikalische Fehler (Scanner)
- syntaktische Fehler (Parser)
- semantische Fehler (Parser, Typprüfer, ...)
- Laufzeitfehler (Laufzeitsystem)

## 6.2 Lexikalische Fehler

Sie werden von Scanner entdeckt. Typische lexikalische Fehler sind:

- Verwendung von Zeichen, die nicht zum Alphabet der Sprache gehören
- Zahlbereichüberschreitungen bei Konstanten
- Nichteinhaltung von Namenskonventionen

Viele Probleme entstehen dadurch, daß der Scanner ein Schlüsselwort *nicht erkennt*: wegen Schreibfehlern. Sie können ansatzweise dadurch behoben werden, daß man im Parser eine Fehlerkorrektur versucht: erwartet der Parser ein Schlüsselwort und liefert der Scanner keins, so vergleicht man das vom Scanner gelieferte Symbol (Name) mit allen Schlüsselwörtern und prüft, ob sie aus diesem Symbol hervorgehen durch:

- Hinzufügen oder Weglassen eines Zeichens,
- Vertauschen zweier Zeichen,
- Abändern eines Zeichens.

## 6.3 Syntaktische Fehler

Sie werden vom Parser entdeckt. Wir behandeln exemplarisch nur den Fall der *Top-Down-Analyse* mit einem *Recursive Descent Parser*.

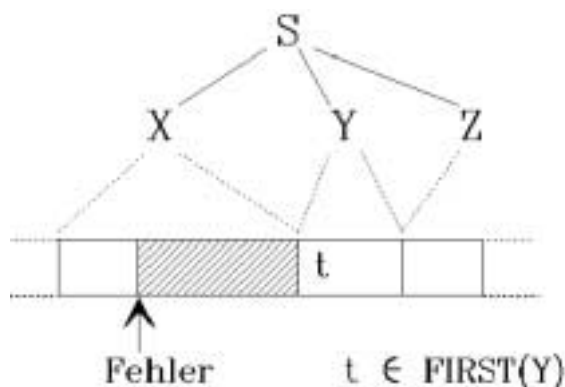
Als Beispiel dient der PL/0-Parser in C (ohne Fehlerbehandlung). Dort wird bei Fehlern die Syntaxanalyse abgebrochen. Ziel einer sinnvollen Fehlerbehandlung muß jedoch sein, eine Fortsetzung der Analyse zu ermöglichen: *Recovery*.

### Recovery-Strategien:

- panische Recovery
- konstruktorientierte Recovery
- Fehlerproduktion

### 1. Panische Recovery

Wenn der Parser einen Fehler entdeckt hat, überliebt er die Eingabesymbole solange, bis er ein Symbol findet, das zu einer Menge "synchronisierender" Symbole gehört:



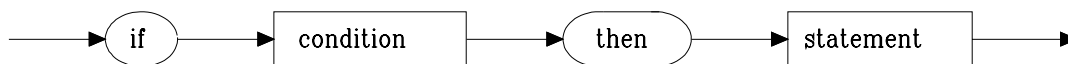
### Programmtechnische Realisation

Jede Prozedur des Recursive Descent Parsers besitzt einen Parameter, dem die Menge der erlaubten Folgesymbole übergeben wird.

Ein unkontrolliertes Überlesen wird durch Abbrechen bei bestimmten, zusätzlichen *Stoppzeichen* verhindert. Im PL/0-Compiler geschieht dies mit Hilfe der Prozedur Test. Ihre Parameter sind:

- Menge der erweiterten Folgezeichen
- Menge der zusätzlichen Stop-Zeichen
- für die Situation charakteristische Fehlernummer

*Bsp.:* Die einseitige if-Anweisung in PL/0 hat die Syntax



### Fehlerbehandlung des PL/0-Parsers (alte Pascal-Version)

```

IF sym = ifsym THEN
BEGIN
  getsym();
  (*) condition([thensym,ident] + statestart);
  (**) IF sym = thensym THEN Getsym ELSE Error(...);
  Statement(...)
END
  
```

Falls in einer IF-Anweisung das Wortsymbol THEN fehlt oder falsch geschrieben ist, wird dies durch (\*\*) als Fehler gemeldet. In diesem Fall hat der Aufruf von (\*) schon dafür gesorgt, daß die Analyseprozedur Statement einen sinnvollen Aufsetzpunkt erreicht.

## 2. Konstruktorientierte Recovery

Der Parser versucht bei bestimmten Fehlern, lokale Korrekturen anzubringen.

*Bsp.:* Ersetzung eines Kommas durch ein Semikolon.  
Einfügen eines fehlenden Semikolons.

Diese Korrekturen dürfen nicht zu Endlosschleifen führen!

## 3. Fehlerproduktionen

Wenn einigermaßen klar ist, welche Fehler gewöhnlich auftreten, so kann man einer Grammatik Produktionen zur Erzeugung dieser *fehlerhaften Konstrukte* hinzufügen. Für diese erweiterte Grammatik wird dann der Parser erstellt. *Vorteil:* Bessere Fehlererkennung, leichtere Synchronisierung nach Fehlern.

<if-Anweisung>           —> if <condition> then <Anweisung> else <Anweisung>  
<falsche if-Anweisung> —> if <condition> then <Anweisung>; else <Anweisung>

Auf den folgenden Seiten ist das Listing des PL/0-Compilers bei der Analyse eines fehlerhaften Programms wiedergegeben.

```

C O M P I L E R   L I S T I N G
=====

  1      const a:=15  b=7, c=  ;
****      ^
**** Use "=" instead of "!=" !
****      ^
**** Semicolon or comma missing !
****      ^
**** A number must follow after "=" !

  2      var  min  max
****      ^
**** Semicolon or comma missing !

  3      begin
****      ^
**** Semicolon or comma missing !

  4      if a<b then begin
  5          min:=a
  6          max:=b
****      ^
**** This symbol cannot follow a factor !

  7          end;
****      ^
**** Text overpassed until here !

  8      if a>=b  begin
****      ^
**** "then" expectet !

  9          min:=b;
 10          max =a
****      ^
**** Assignment operator is "!=" !
****      ^
**** An Expression may not start with this symbol !
****      ^
**** Text overpassed until here !

 11          end;
 12      if c<min then min:=c
****      ^
**** Undeclared identifier !
****      ^
**** Undeclared identifier !

 13      if c>max then  :=c
****      ^
**** Incorrect use of a symbol within a statement !
****      ^
**** Undeclared identifier !
****      ^
**** Illegal symbol following a statement !

 14      end.
****      ^
**** Text overpassed until here !

**** 18 lines completed with errors

```

```
line   1 Error   1   Use "=" instead of ":= !
line   1 Error   5   Semicolon or comma missing !
line   1 Error   2   A number must follow after "=" !
line   2 Error   5   Semicolon or comma missing !
line   3 Error   5   Semicolon or comma missing !
line   6 Error  23   This symbol cannot follow a factor !
line   8 Error  16   "then " expected !
line  10 Error  13   Assignment operator is ":= " !
line  10 Error  24   An Expression may not start with this
        symbol !
line  12 Error  11   Undeclared identifier !
line  12 Error  11   Undeclared identifier !
line  13 Error  10   Incorrect use of a symbol within a
        statement !
line  13 Error  11   Undeclared identifier !
line  13 Error  19   Illegal symbol following a statement !

number of errors   14
```

## 6.4 Semantische Fehler

Sie werden von der semantischen Analyse entdeckt. Es handelt sich dann im wesentlichen um Fehler gegenüber den Typisierungsregeln. Man kann dann weiterkompilieren, muß darauf achten, dass die Fehler nicht wiederholt auftreten und zu viele Fehlermeldungen erzeugen. Siehe dazu das Beispiel auf Seite 5-2.