Inhaltsverzeichnis dieser Seite:

10.1 Klassifikation von Fehlern

Zur wirksamen Vermeidung oder Beseitigung von Fehlern ist ein klares Bild ihrer Ursachen notwendig.

10.1.1 Syntaxfehler

- Die einfachste Art von Fehlern: wird vom Compiler meistens schnell und zuverlässig aufgedeckt;
- Einfaches Beispiel:

```
if (x == 0) then
       x++;
```

  Schlüsselwort "`then`" in C(++) syntaktisch falsch;
- Compiler kann inhaltliche Fehlerursache i.d.R. nicht lokalisieren, sondern weist auf ersten Punkt mit Schwierigkeiten hin;

  Beispiel: GNU-C++-Compiler V2.7.2 meldet:

```
parse error before `++'
```

- Fehlerhaftes Beispielprogramm enthält (u.a.) einen undurchsichtigen Syntaxfehler;
- Compilermeldungen erfordern Interpretation seitens des Lesers; Qualität von Fehlermeldungen ist heikler Punkt im Compilerbau; bis heute nicht befriedigend gelöst;

10.1.2 Fehler der statischen Semantik

- Betreffen Abhängigkeiten zwischen entfernt liegenden Quelltext-Abschnitten;
- Beispiel: Verwendung eines Bezeichners muß im Einklang mit der Typdefinition stehen;

```
int x = 0;      // x ist als Variable definiert
int y;

y = 2*x(2);     // x wird als Funktion aufgerufen
```

GNU-C++-Compiler V2.7.2 meldet zielsicher:

```
`x' cannot be used as a function
```

- Fehlerhaftes Beispielprogramm enthält Verstoß gegen statische Semantik (wird von den meisten Compilern treffend moniert);
- Fehler der statischen Semantik nicht scharf von Syntaxfehlern abgrenzbar;

  Beispiel: Definition eines Bezeichners mit dem Namen eines Schlüsselwortes, wie z.B. in:

```
int if = 0;
if(if == 0)
        if++;
else
        if--;
```

  - wird durch *Grammatik* der Sprache nicht ausgeschlossen, demnach streng genommen kein Syntaxfehler;
  - wird i.d.R. durch umgangssprachliche Begleittexte zur Grammatik verboten; also in der Zuständigkeit der statischen Semantik;

  Mit viel Aufwand könnten Schlüsselworter *syntaktisch*, d.h. in der Grammatik, [1]von benutzerdefinierbaren Bezeichnern abgegrenzt werden;

  Damit wäre obiger Fehler wieder zum Syntaxfehler geworden;

10.1.3 Fehler der dynamischen Semantik (Laufzeitfehler)

- Betreffen Verwendung von Sprachkonstrukten, die vom Compiler (zur Übersetzungszeit) nicht gefunden werden *können;*
- Einfaches Beispiel: Semantik des Divisionsoperators `/` verbietet Division durch Null;

```
int x;
cin >> x;
x = 1/x;        // potentielle Division durch Null;
                // erst zur Laufzeit feststellbar
```

- Programm mit Fehlern der dynamischen Semantik wird vom Compiler *ohne Beanstandung* übersetzt;
- Verstöße gegen die dynamische Semantik äußern sich als **"Laufzeitfehler";**

  Mit GNU-C++-Compiler V2.7.2 übersetztes Programm meldet bei der *Ausführung* nach Eingabe von 0:

```
Floating point exception
```

  und bricht dann ab ("stürzt ab"); [2]

- Behandlung von Laufzeitfehlern problematisch, weil...

- im übersetzten Programm Bezeichner nicht mehr enthalten sind (Treffende Fehlermeldungen können nicht mehr formuliert werden);
- die ursprüngliche Programmstruktur (Zeilennummern etc.) im übersetzten Programm verschwunden ist (Folge: wie oben); [3]
- das Programm möglicherweise in einer Umgebung abläuft, in der Textausgaben den Benutzer nicht erreichen (Start eines Programms über Popup-Menü in einem Fenstersystem);
- der Fehler nicht in jedem Fall auftritt, sondern nur in einem bestimmten Kontext (Benutzereingaben, Kommandozeilen-Parameter, Eingabedaten, Uhrzeit und Datum etc.); Der Fehler ist dann nicht mehr reproduzierbar; [4]

- Fehlerhaftes Beispielprogramm enthält Verstoß gegen dynamische Semantik, der bei Eingabe von "0/1" auftritt (leicht zu finden, weniger leicht zu beheben);
- Grenze zwischen statischer und dynamischer Semantik ist fließend, weil bestimmte Zusammenhänge theoretisch vom Compiler entdeckt werden könnten, wenn er nur gründlich genug zu Werke würde;

  Beispiel:

  ```
  int x = 1;
  x = 2*x - 1;
  x = 1/(--x);    // zwangsläufig Division durch 0
  ```

  In letzter Konsequenz würde das bedeuten, daß der Compiler das Programm schon beim Übersetzen "im Geiste" ausführen müßte, um die Folgen derartiger Berechnungen herauszufinden.

  Das ist natürlich kein praktikabler Ansatz, so daß derartige Fehlkonstruktionen pauschal in die Verantwortung der dynamischen Semantik delegiert werden.

10.1.4 Logische Fehler

- Selbst ein syntaktisch und (im Sinne der Programmiersprache) sematisch korrektes Programm liefert kaum auf anhieb die erwarteten Ergebnisse;
- Die hohe Kunst des Programmierens liegt zum guten Teil in der korrekten und leserlichen Wiedergabe von Entwurfsgedanken im Formalismus einer Programmiersprache; [5]
- Logische Fehler äußern sich nicht in Fehlermeldungen, sondern in falschen Resultaten oder unerwartetem Verhalten. [6]
- Fehlerhaftes Beispielprogramm ist logisch falsch; Eingabe von "1/1" liefert Ergebnis "1/1"; Betrachten des kurzen Hauptprogramms zeigt sofort, daß das nicht stimmen kann; (Fehler läßt sich mit minimalem Aufwand beheben, aber nur schwer finden;)

10.1.5 Warnungen

- Einige Konstrukte sind syntaktisch und semantisch korrekt, aber *erfahrungsgemäß* in den seltensten Fällen logisch korrekt;
- Beispiele:

1. Wertzuweisung in Bedingung

```
if(x = 0)
        ...
```

2. Funktion, die manchmal ein Ergebnis liefert, manchmal keines:

```
int Foo(int x)
{
        if(x == 0)
                return(x + 1);
}
```

3. unbenutzter formaler Parameter:

```
void Foo(int x)
{
        return(2);
}
```

- (Brauchbare) Compiler melden solche Konstrukte als "Warnung", übersetzen das Programm aber trotzdem;

  GNU-C++-Compiler V2.7.2 meldet:

  1. `warning: suggest parentheses around assignment used as truth value`
  2. `warning: control reaches end of non-void function `Foo(int)'`
  3. `warning: unused parameter `int x'`

- Warnungen eines Compilers lassen sich pauschal und einzeln an- und abschalten;

  Empfehlung: Pauschal alle Warnungen **anschalten;** Programm solange korrigieren, bis alle Warnungen behoben sind;

  GNU-C++-Compiler V2.7.2 aufrufen mit

  ```
  g++ -Wall ...
  ```

  Aktiviert (fast) alle Prüfungen; Weitere Informationen über die *Manual page* zum Compiler;

[Previous] [Top of page] [Next]

Letzte Änderung: Sat Oct 19 20:30:41 1996