

Konzepte der Programmiersprachen

Sommersemester 2007

Eduard Wiebe

Übungsblatt 5

Organisatorisches

Dieses Übungsblatt wird am 6. Juli besprochen.

Aufgabe 5.1: Das „dangling else“-Problem

Das „dangling else“-Problem bei der Definition von Programmiersprachen beruht darauf, dass Grammatiken mit zwei Produktionen der Struktur

```
Anweisung -> IF Bedingung THEN Anweisung
              | IF Bedingung THEN Anweisung ELSE Anweisung
```

mehrdeutig sind.

1. Zeigen Sie anhand der Anweisung
IF b1 THEN IF b2 THEN a1 ELSE a2,
welche Probleme durch diese Mehrdeutigkeit entstehen.
2. Welche Möglichkeiten gibt es zum Beseitigen der Mehrdeutigkeit des „dangling else“
(z. B. in Algol 60, Pascal, Modula-2 und Occam)?

Aufgabe 5.2: Äquivalenz von Schleifen

Eine Programmiersprache (etwa Modula-2) verfüge über WHILE-Schleifen, REPEAT-UNTIL-Schleifen und FOR-Schleifen sowie bedingte Anweisungen, Zuweisungen und Ausdrücke mit der „üblichen“ Semantik.

Insbesondere werden die Kontrollausdrücke der FOR-Schleife nur bei Schleifeneintritt ausgewertet; die Schleifenvariable kann im Schleifenrumpf nur gelesen werden.

Die Syntax sei:

```
Schleife -> WHILE Bedingung DO Anweisungsfolge END
            | REPEAT Anweisungsfolge UNTIL Bedingung
            | FOR Bezeichner := Ausdruck TO Ausdruck DO
              Anweisungsfolge END .
```

Welche dieser Schleifenkonstrukte können semantisch äquivalent durch die jeweils anderen Schleifenkonstrukte (evtl. unter Verwendung der oben genannten, zusätzlichen Sprachelemente) ersetzt werden?

Geben Sie entsprechende Programmfragmente an. Begründen Sie die Fälle, in denen keine solche Äquivalenz möglich ist.

Aufgabe 5.3: For-Schleifen

Vergleichen Sie die Konzepte von For-Schleifen in den folgenden Programmiersprachen: Basic, Pascal, Ada, C/C++, Python.

Wird die Terminierung der Schleife garantiert? Wie sieht die Lebensdauer der Schleifenvariable aus, und welchen Wert hat sie ggf. nach dem Austritt aus der Schleife? Wird die Schleifenvariable vor Schreibzugriffen innerhalb des Schleifenkörpers geschützt? Welche Vor- und Nachteile haben die jeweiligen Varianten von Schleifen?

Aufgabe 5.4: Ausdrücke

1. Nennen Sie Beispiele für Operatoren und Funktionen aus der Mathematik, für die Infix-, Präfix-, Postfix- oder Funktions-Schreibweise üblich sind.
2. Stellen Sie den Infix-Ausdruck $((a + b) / (c - d) + e) * (f - g)$ in Postfix- und Präfix-Schreibweise dar. Gibt es ein allgemein anwendbares Verfahren für diese Transformationen?
3. Die ersten kommerziell erhältlichen Taschenrechner verlangten Eingaben im Postfix-Format. Was könnte der Grund dafür sein?
4. Die Postfix-Notation wird auch als umgekehrt polnische Notation (abgekürzt UPN) bezeichnet. Warum?
5. Welche Programmiersprachen verwenden Infix-, Präfix- bzw. Postfix-Notation für Ausdrücke? Welche Art der Notation ist aus Sicht des Programmierers wünschenswert?
6. Wie werden Ausdrücke in COBOL notiert?

Aufgabe 5.5: Parameterübergabe-Semantik

Gegeben sei das folgende Programm in einer Ada-ähnlichen Sprache:

```
I: Integer;
A: array(1..3) of Integer := (0,0,0);

procedure Test (X: Integer) is
begin
  A(2) := 7;
  I := 3;
  X := X + 1;
end Test;

begin
  I := 2;
```

```

    Test (A(I));
    Print (A);
    Print (I);
end;

```

1. Welche Ausgabe liefert dieses Programm, wenn als Parameterübergabe-Mechanismus
 - a) call by value
 - b) call by value-result
 - c) call by reference
 - d) call by name (ALGOL-Regel)
 - e) call by name (originale Namensersetzungsregel)
 verwendet wird?

2. Die Prozedur `Test` im obigen Beispiel sei nun implementiert durch:

```

procedure Test (X: Integer) is
  I: Integer := 1;
begin
  A(2) := 7;
  X := X + 1;
end Test;

```

Ändern sich damit die Antworten auf die Fragen in Teil 1? Wenn ja, welche, mit welchem Resultat und warum?

Aufgabe 5.6: Parameterübergabe in verschiedenen Sprachen

Viele Programmiersprachen unterscheiden sich in der Art, wie Parameterübergabe definiert ist.

1. Pascal und Modula-2 bieten zur Parameterübergabe nur Call-By-Value und Call-By-Reference an. Welche Vor- und Nachteile ergeben sich daraus?
(Hinweis: Entwerfen Sie ein Unterprogramm zur Multiplikation von Matrizen und überlegen Sie, welche Probleme bei der Verwendung dieses Unterprogrammes auftreten können.)
2. Wie hat man versucht, dieses Problem in den Sprachen C++ und Modula-3 zu lösen?
3. Wie unterscheiden sich die Werteparameter in Modula-2 von den IN-Parametern in Ada?

4. In Ada 83 ist vorgeschrieben, dass formale OUT-Parameter nicht in Ausdrücken verwendet werden dürfen. Diese Regel ist aufgrund der beabsichtigten Semantik von OUT-Parametern naheliegend, aber ist sie auch sinnvoll, notwendig und effizient? Welche Verbesserung enthält Ada 95?
5. Die Sprache Ada schreibt als Implementierung von IN-OUT-Parametern für skalare Datentypen Copy-In-Copy-Out vor, während es dem Implementierer überlassen bleibt, strukturierte Datentypen wahlweise auch durch Call-By-Reference zu übergeben. Welche Vor- und Nachteile ergeben sich dadurch?
6. *Es ist nicht allgemein entscheidbar, ob die Ausgabe eines Programmes von der zur Implementierung von IN-OUT-Parametern verwendeten Übergabemethode abhängt.*
Beweisen Sie diese Aussage!