

# Grundlagen Programmiersprachen und Compilerbau

Wintersemester 2006/2007

## 6. Übungsblatt

22. Januar 2007

### Organisatorische Hinweise

Dieses Übungsblatt wird in der 5. Kalenderwoche 2007 besprochen.

### Aufgabe 6.1

1. Gegeben ist ein Schleifenkonstrukt `repeat C until B`. Geben Sie hierzu die operationelle, die denotationelle sowie die axiomatische Semantik analog zur jeweiligen Semantik der `while`-Schleife an.
2. Das Axiom der Wertzuweisung lautet  $\{P[id \leftarrow exp]\} id := exp \{P\}$ . Warum ist die vereinfachte Regel  $\{true\} id := exp \{id = exp\}$  nicht korrekt?
3. Betrachten Sie das Tripel  $\{X = Y\} C \{X = 2 * Y\}$  und nehmen sie an, dass die Ausführung von `C` immer terminiert.

- a) Warum bedeutet dieses Tripel *nicht*, dass „`C` den Wert von `X` verdoppelt“?
- b) Geben Sie ein Tripel  $\{P\} C \{Q\}$  an, das diese Aussage macht.

4. Gegeben sei das Programm:

```
I := 0; Y := 1; while I ≤ X do I:=I+1; Y:=Y+1; od
```

und die Vorbedingung  $\{X \geq 0\}$ . Bestimmen Sie die stärkste Nachbedingung mit Hilfe der im Skript abgedruckten Regeln für axiomatische Semantik.

*Tipp:* Gilt nach der Ausführung  $\{Y = X!\}$ ?

### Aufgabe 6.2

Gegeben sei das folgende Fragment aus einem Ada-ähnlichen Programm:

```
for i in 1..1_000_000 loop
  A := new T7; B := new T9; C := new T3; D := new T5;
  ... E := D; ...
  Free(A); Free(B); Free(C); Free(D);
  -- Free(E); -- (1)
end loop;
```

Die Halde wird durch eine Freispeicherliste nach dem „first-fit“-Verfahren verwaltet. Die Größe der Allokationen sei durch die Typnamen gekennzeichnet, d.h., ein Objekt vom Typ `T7` benötigt 7 Worte, vom Typ `T3` 3 Worte, usw.

1. Zeichnen Sie die Freispeicherliste, die nach dem erstmaligen Durchlauf der Schleife entstanden ist.
2. Charakterisieren Sie das Verhalten der Haldenverwaltung für die weiteren Durchläufe der Schleife. „Funktioniert“ dieses Programm? Begründen Sie Ihre Antwort.
3. Die bislang auskommentierte Zeile bei (1), d.h., „Free(E)“ sei nun nicht mehr auskommentiert. Welchen Effekt hat die Ausführung dieses „Free“-Aufrufs und wie werden sich die weiteren Durchläufe der Schleife verhalten?

### Aufgabe 6.3

Gegeben sei das folgende Ada95 Programm. Die Parameterübergabe sei „by-value“. Der Compiler implementiert die Methodik des „Reference Counting“ (über eine für den Benutzer transparente Zählerkomponente in den Haldenobjekten).

Das Programm erzeugt insgesamt vier Haldenobjekte O1 bis O4 an den im Code kommentierten Stellen. (Guter Rat: Machen Sie sich Skizzen der Haldenzustände.)

```
( 0)  type Node;
( 1)  type Node_Acc is access Node;
( 2)  type Node is
( 3)      record
( 4)          Content : Integer;
( 5)          Next    : Node_Acc;
( 6)      end record;
( 7)  procedure test is
( 8)      P,Q,R : Node_Acc;
( 9)      procedure Inner ( A, B, C : Node_Acc ) is
(10)      begin
(11)          declare
(12)              S : Node_Acc := new Node'(8, C); -- Objekt O4
(13)          begin
(14)              P := B;
(15)              S := A;
(16)              B.Next := C;
(17)              ... -- Verwendung, aber keine Änderung der Zeiger
(18)          end;
(19)              ... -- Verwendung, aber keine Änderung der Zeiger
(20)          end Inner;
(21)  begin
(22)      P := new Node'(5, null); -- Objekt O1
(23)      Q := new Node'(6, null); -- Objekt O2
(24)      P.Next := Q;
(25)      R := new Node'(7, Q); -- Objekt O3
(26)      Inner(P, Q, R);
(27)      return;
(28)  end test;
(29)  test;
```

- a) Geben Sie in der folgenden Tabelle den momentanen Wert des Referenzzählers für jedes der Objekte an, nachdem die genannte Zeile und alle durch sie implizierten Aktionen der „Reference Counting“ Methode ausgeführt wurden.

	O1	O2	O3	O4
nach Zeile (16):				
nach Zeile (18):				
nach Zeile (26):				

- b) Geben Sie ggf. die Nummern der Zeilen an, nach deren Ausführung die Reference-Counting Methode aufgrund eines Referenzzählers mit dem Wert 0 das entsprechende Objekt freigibt. Erfolgt keine Freigabe, markieren sie dies mit „X“.

Freigabe von	O1	O2	O3	O4
nach Zeile:				

### Aufgabe 6.4

Gegeben seien folgende Deklarationen:

```

type Feld1 = array[1..10] of BOOLEAN;
   Feld2 = array[1..10] of BOOLEAN;
var A      : Feld1;
   B, C    : array[1..10] of BOOLEAN;
   D, E    : Feld2;
   F      : Feld2;
   X      : array[1..10] of Feld1;
   Z, Y   : array[1..10] of Feld2;

```

- a) Welche Variablen sind strukturäquivalent?
- b) Welche Variablen sind namensäquivalent? (Hier gibt es eine Klasse von Fällen, die in verschiedenen Programmiersprachen unterschiedlich gelöst ist — welche?)
- c) Welche Vor- und Nachteile haben die beiden Konzepte der Typäquivalenz aus der Sicht eines Programmierers und aus der eines Compilerbauers?