

# Grundlagen Programmiersprachen und Compilerbau

Wintersemester 2005/2006

## 7. Übungsblatt

01.02.2006

Dieses Übungsblatt wird noch dieses Jahr besprochen :-)

### Aufgabe 7.1

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.

- Zeichnen Sie die Freispeicherliste, die nach dem erstmaligen Durchlauf der Schleife entstanden ist.
- Charakterisieren Sie das Verhalten der Haldenverwaltung für die weiteren Durchläufe der Schleife. "Funktioniert" dieses Programm? Begründen Sie Ihre Antwort.
- 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?

(Klausuraufgabe)

### Aufgabe 7.2

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:				

(Klausuraufgabe)

### Aufgabe 7.3

In vielen Programmiersprachen wird ein Laufzeitdeskriptor (“dope”) für Arrays benötigt. Gegeben sei nun das folgende Programm:

```
type Arr is array(Integer range <>, Integer range <>) of Integer;

procedure test(ArrPara: in out Arr) is
begin
  ArrPara(3,2):= 1000; -- (1)
end test;

procedure driver(X: in out Integer; K: Integer) is
  ArrObj: Arr(K..4, 1..X);
begin
  X := 4;
  test(ArrObj);
end driver;

driver(2,2);
```

Multidimensionale Arrays seien spaltenorientiert gespeichert; Integer benötigen 4 Bytes; die Adressierung sei Byte-orientiert.

- Zeichnen Sie ein Bild des Laufzeitdeskriptors zu 'ArrPara', geben Sie zu jedem Feld einen kurzen Kommentar, der dessen Inhalt charakterisiert, und tragen Sie die Werte ein, die gelten, wenn am Punkt (1) die Zuweisung an ArrPara(3,2) erfolgt.
- Erläutern und begründen Sie, welche Teile des Laufzeitdeskriptors am Punkt (1) im Beispiel für die Adressierung des Arrayelements definitiv benötigt werden.
- Erläutern Sie, welche Teile des Laufzeitdeskriptors unbedingt benötigt werden, um ungültige Speicherzugriffe durch Indizierungen zur Laufzeit zu verhindern, und wie diese Information benützt wird.
- Zeichnen Sie ein Bild des Speicherbereichs für den Wert von 'ArrObj' (ohne Laufzeitdeskriptor). Die Abfolge der Elemente muss aus Ihrer Zeichnung eindeutig hervorgehen.

(Klausuraufgabe)

## Aufgabe 7.4

Gegeben sei das folgende Programmfragment:

```
type Ptr is pointer to some_type;
procedure clean_assign (A: Ptr; B: Ptr) is
begin
  Free (A);
  A:= B;
end clean_assign;
```

**Free** veranlasst die Freigabe der Speichers für das vom Zeiger designierte Objekt und setzt den Zeigerparameter auf **null**, führt aber keine weiteren Prüfungen oder Maßnahmen durch. Der Parameterübergabe-Mechanismus der Sprache sei zunächst unspezifiziert.

- a. Zeigen Sie durch ein kleines Programmfragment, das **clean\_assign** aufruft, wie *unabhängig vom Parameterübergabe-Mechanismus* durch den Aufruf von **clean\_assign** sogenannte ‘dangling references’ im aufrufenden Programm entstehen können.
- b. Der Parameterübergabe-Mechanismus sei nun ‘by value-result’ für alle Parameter. Entstehen durch diesen Mechanismus *weitere* Möglichkeiten für ‘dangling references’ im aufrufenden Programm? Wenn ja, wie? Wenn nein, warum nicht?
- c. Der Parameterübergabe-Mechanismus sei nun ‘by reference’ für alle Parameter. Entstehen durch diesen Mechanismus gegenüber a) *weitere* Möglichkeiten für ‘dangling references’ im aufrufenden Programm? Wenn ja, wie? Wenn nein, warum nicht?

(Klausuraufgabe)