

# Grundlagen der Programmiersprachen und Compilerbau

Wintersemester 2007/2008

## 7. Übungsblatt

6. Februar 2008

### Aufgabe 7.1

Die Halde sei durch das binäre Buddy-Verfahren verwaltet. Das System verfügt über  $2^\ell$  Bytes an physikalischem Speicher. Die kleinste mögliche Block-Größe beträgt  $2^k$  Bytes. Die Adressierung ist Byte-orientiert.

Wählen Sie eine Notation für die Allokationsanforderungen aus und geben Sie eine Sequenz von diesen an das System an, so dass:

- die interne Fragmentierung maximal ist,
- die externe Fragmentierung maximal ist,
- der interne und der externe Verschnitt gleich sind.

Geben Sie jeweils die Verschnittgrößen an.

### Aufgabe 7.2

In vielen Programmiersprachen wird ein Laufzeitdeskriptor (*dope*) für Arrays (Felder) benötigt. Gegeben sei nun das folgende MANGER<sup>©</sup>-Programm:

```
verbund R ist
  eins : Zahl;
  zwei : Bool;
ende R;

typ Feld ist feld(Zahl bereich <>, Zahl bereich <>) von R;

ablauf Test (feld_param: rein raus Feld) ist
  anfang
    feld_param(4,3).zwei := Richtig; -- (1)
  ende test;

ablauf Treiber (X: rein raus Zahl; K: Zahl) ist
  feld_objekt : Feld (2 .. X, K .. 4);
  anfang
    X := 7;
    Test (feld_objekt);
  ende Treiber;

Treiber (4,2);
```

Multidimensionale Felder seien spaltenorientiert gespeichert; `Zahl` benötigen 4 Bytes und `Bool` einen Byte; die Adressierung sei Byte-orientiert.

1. Zeichnen Sie ein Bild des Laufzeitdeskriptors zu `feld_param`, 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 `feld_param(4,3).zwei` erfolgt.
2. Erläutern und begründen Sie, welche Teile des Laufzeitdeskriptors am Punkt (1) im Beispiel für die Adressierung des Feldelements definitiv benötigt werden.
3. Erläutern Sie, welche Teile des Laufzeitdeskriptors unbedingt benötigt werden, um ungültige Speicherzugriffe durch Indizierungen zur Laufzeit zu verhindern, und wie diese Information benutzt wird.
4. Zeichnen Sie ein Bild des Speicherbereichs für den Wert von `feld_objekt` (ohne Laufzeitdeskriptor). Die Abfolge der Elemente muss aus Ihrer Zeichnung eindeutig hervorgehen.

### Aufgabe 7.3

Gegeben sei das folgende Programm in MANGER<sup>©</sup>:

```
ablauf Müstik
  I : Zahl;
  A : feld (1..3) von Zahl := (1, 2, 3);

  ablauf Was (X : Zahl) ist
  anfang
    I := 3;
  sei
    I : Integer := 2;
  anfang
    X := 20;
    A[1] := A[1] + I;
  ende;
  ende Was;

anfang
  I := 1;
  Was(A[I]);
  Gebe_Aus (A);
ende Müstik;
```

Welche Ausgabe liefert dieses Programm, wenn als Parameterübergabemechanismus

1. call-by-reference

2. call-by-value
3. call-by-value/result
4. call-by-name
5. call-by-name mit Algol-Regel

benutzt wird?

#### Aufgabe 7.4

Gegeben sei das folgende Programmfragment in MANGER<sup>©</sup>:

```

typ Zgr ist zeiger auf T;

ablauf Zuweisen (A: Zgr; B: Zgr) ist
  anfang
    Frei (A);
    A:= B;
  ende Zuweisen;

```

**Frei** 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.

1. Zeigen Sie durch ein kleines Programmfragment, das **Zuweisen** aufruft, wie *unabhängig vom Parameterübergabe-Mechanismus* durch den Aufruf von **Zuweisen** sogenannte „dangling references“ im aufrufenden Programm entstehen können.
2. 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?
3. Der Parameterübergabe-Mechanismus sei nun „by reference“ für alle Parameter. Entstehen durch diesen Mechanismus gegenüber 1. *weitere* Möglichkeiten für „dangling references“ im aufrufenden Programm? Wenn ja, wie? Wenn nein, warum nicht?