Übung zur Vorlesung Compiler 1: Grundlagen

Prof. Dr. Andreas Koch
Jens Huthmann

TECHNISCHE UNIVERSITÄT DARMSTADT

Wintersemester 12/13
Aufgabenblatt 3 - Lösungsvorschlag

Abgabemodalitäten

Die Aufgabe sind in 2er Gruppen zu bearbeiten. Ihre Lösungen reichen Sie als PDF-Datei per E-Mail bis zum 3.02.2013 um 23:59 MEZ an der Adresse oc@esa.informatik.tu-darmstadt.de ein. Die E-Mail muss als Betreff "Compiler 1 Aufgabe 3" haben. Geben Sie in Ihrem Lösungsdokument den Namen und die Matrikelnummer aller Gruppenmitglieder an.

Aufgabe 3.1 Allgemeine Fragen — 9 Punkte

a) Beschreiben Sie kurz die Aufgabe des Routinenprotokolls. Begründen Sie auch warum ein Routinenprotokoll notwendig ist.
Das Routinenprotkoll gibt Aufschluss über Lage und Sortierung von Parametern beim Aufruf von Routinen. Zusätzlich bestimmt es ob der Aufrufer oder Aufgerufene Register sichert und wer den Stack und die Register bei Routinenende aufräumt. Kompatible Bibliotheken sind nur denkbar, wenn sie das gleiche Routinenprotokoll implementieren.

b) Wieso ist es notwendig einen Stack und einen Heap in der Speicherverwaltung zu verwenden? Beschreiben Sie hierzu auch kurz welche Daten im Stack bzw. Heap gespeichert werden.
Der Stack wird dazu verwendet alle statisch allozierbaren Variablen zu speichern. Dabei ist aber nur die Größe dieses Bereichs für jede einzelne Routine immer der gleiche. Da die Reihenfolge der Aufrufe während des Programmablaufs dynamisch varieren kann ist die Größe des Stacks nicht statisch. Da die TAM eine Stackmaschine ist liegen auch alle Operanden auf dem Stack. Der Heap hingegen wird verwendet um Datenstrukturen abzuspeichern deren Größe zur Compilierungszeit nicht bekannt ist und die unabhängig von der Aufrufhierachie weiterbestehen sollen, wie z.b. verkettete Listen.

c) Wofür wird der dynamische bzw. statische Link innerhalb der Speicherverwaltung verwendet?
Der dynamic link dient dazu das Stackframe der aufrufenden Routine zu merken. Die ist notwendig um nach Fertigstellung der aufgerufenen Routine das Stackframe der aufgerufen Routine abzubauen. Somit wird das aktuelle Stackframe auf das der aufrufenden Routine zurückgesetzt. Der static link hingegen ist ein Verweis auf die im Quelltext umgebende Routine, welcher verwendet wird um auf nicht lokale Variablen zuzugreifen.

Aufgabe 3.2 Routinen als Parameter — 20 Punkte

Triangle bietet die Möglichkeit auch Funktionen oder Prozeduren als Parameter bei einem Aufruf zu verwenden. Hierzu wird ein Paar bestehende aus Static Link und der Adresse der Routine übergeben. Dieses Paar nennt man Closure.

Untersuchen Sie das folgende Programm mit Hilfe des Triangle Disassemblers. Kommentieren Sie im disassemblierten Code die Position an welcher die Clossure erzeugt wird und wie die parametrisierte Funktion aufgerufen wird. Geben Sie auch an wo die einzelnen Routinen beginnen.

```
let
  func twice(func doit(x : Integer): Integer, i : Integer): Integer ~ doit(doit(i));
  func double(d : Integer): Integer ~ d*2;
  var x: Integer
in begin
  x := twice(func double, 10);
  putint(x)
end
```

Aufgabe 3.2 Lösung

```
0:  JUMP        7[CB]
1:  LOAD  (1)   -1[LB]; func twice; load first parameter
2:  LOAD  (2)   -3[LB]; load closure
3:  CALLI             ; call loaded closure
4:  LOAD  (2)   -3[LB]; load closure
5:  CALLI             ; call loaded closure
6:  RETURN(1)   3
7:  JUMP        12[CB]
8:  LOAD  (1)   -1[LB]; func double
9:  LOADL       2
10:  CALL  (SB)  10[PB]; call mult
11:  RETURN(1)   1
12:  PUSH        1; main program
13:  LOADA       0[SB]; closure start - static link
14:  LOADA       8[CB]; closure end   - address of double
15:  LOADL       10
16:  CALL  (SB)  1[CB]; call twice
17:  STORE (1)   0[SB]
18:  LOAD  (1)   0[SB]
19:  CALL  (SB)  26[PB]; call putint
20:  POP   (0)   1
21:  HALT
```

Aufgabe 3.3 Codeschablonen — 16 Punkte

Triangle soll um ein "`foreach ARRAY do C`" Kommando erweitert werden. Dieser Befehl soll auf jedes Element des Arrays das Kommando `C` ausführen. Hierbei wird das aktuelle Element von `ARRAY` innerhalb `C` durch `current` dargestellt. In `size` steht die Anzahl der Elemente von `ARRAY`. Beachten Sie hierbei, dass `C` auch ein "`begin ... end`" Block sein kann. Ein `return` beendet ganz normal die Ausführung der aktuellen Prozedur.

Bestimmen Sie für diesen Befehl die Codeschablone, welche das Kommando ausführt. Sie dürfen hierzu bestehende Schablonen verwenden.

Aufgabe 3.3 Lösung

```
execute[foreach ARRAY do C] =
        elaborate[var _i : Integer]
        execute[_i := 0]
        execute[while _i < size do current := ARRAY[_i]; C; _i := _i + 1]
        POP(0) 1
```

Aufgabe 3.4 Statische Speicherverwaltung — 20 Punkte

Nachfolgend finden Sie die zur Definition eines Go-Spiels notwendigen Datentypen. Geben Sie für dieses Programm die Speicherorganisation der Variable *state* gemäß Folie 40 des 4. Vorlesungsblocks an. Ihre Lösung sollte für jeden Wort-Index des Speichers, beginnend ab der fiktiven Anfangsadresse 0, dessen benutzerdefinierten- (Player, Point,...) und Basistyp (Integer, Boolean) angeben. Markieren Sie auch welche Worte zu welchem Record bzw. Array gehören.

```
let
  type Player ~ record
    number : Integer
  end;
  type Point ~ record
    empty : Boolean,
    owner : Player
  end;
  type Board ~ record
    board : array 361 of Point
```

```
  end;
  type State ~ record
    moves : Integer
    next : Player,
    board : Board,
  end;
  var state : State
in
  ...
```

Listing 1: Beispiel zur statischen Speicherverwaltung

Aufgabe 3.4 Lösung

Adresse	Variablenname	Typ	Strukturhierarchie
0	next.number	Integer	
1	moves	Integer	
2	board.board[0].empty	Boolean	
3	board.board[0].owner.number	Integer	
...	...		
722	board.board[360].empty	Boolean	
723	board.board[360].owner.number	Integer	

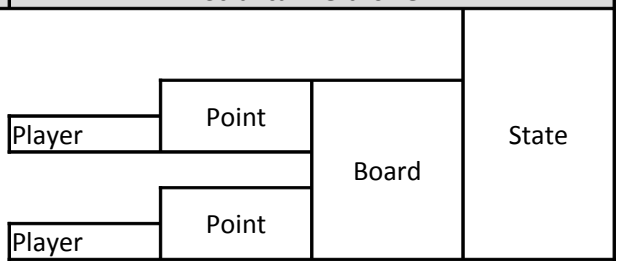

Abbildung 1: Lösung zu Aufgabe 3.3

Aufgabe 3.5 Routinenprotokoll — 10 Punkte

Das nachfolgende Programm implementiert die notwendigen Funktionen zur Berechnung der Fibonacci-Folge. Zeichnen Sie hierfür stack frames, die Argumente und Resultate vor Eintritt in eine Funktion und nach Austritt aus einer Funktion zeigen für die Auswertung folgender Ausdrücke:

a) $mfib(4)$

```
  let
        var n : Integer;

        proc mfib(n : Integer, var r : Integer) ~
        let
              var a : Integer;
              var b : Integer
        in
        if n <= 2 then r := n + 1
        else begin
               mfib(n-1,var a);
               mfib(n-2,var b);
               r := a+b;
        end

  in begin
        getint(var n);
        mfib(n,var n);
        putint(n);
  end
```

Listing 2: Beispielprogramm zum Routinenprotokoll zu Aufgabe 3.4a

Listing 3: before getint(var n)

```
      ST--> |////////|
            |--------|
1:       |0|
0: SB-->|0|
            |--------|
```

Listing 4: after getint(var n)

```
      ST--> |////////|
            |--------|
0: SB-->|4|
            |--------|
```

Listing 5: before mfib(4, var n)

```
      ST--> |////////|
            |--------|
2:       |0|
1:       |4|
0: SB-->|4|
            |--------|
```

Listing 6: before mfib(3, var a)

```
      ST--> |////////|
            |--------|
9:       |6|
8:       |3|
7:       |0|
6:       |0|
5:       |RA=36|
4:       |DL=0|
3: LB-->|SL=0|
            |--------|
2:       |0|
1:       |4|
0: SB-->|4|
            |--------|
```

Listing 7: before mfib(2, var a)

```
      ST--> |////////|
            |--------|
16:      |13|
15:      |2|
14:      |0|
13:      |0|
12:      |RA=19|
11:      |DL=3|
10: LB-->|SL=0|
            |--------|
9:       |6|
8:       |3|
7:       |0|
6:       |0|
5:       |RA=36|
4:       |DL=0|
3:       |SL=0|
            |--------|
```

```
2:       |0|
1:       |4|
0: SB-->|4|
            |--------|
```

Listing 8: after mfib(2, var a)

```
      ST--> |////////|
            |--------|
14:       |0|
13:       |3|
12:       |RA=19|
11:       |DL=3|
10: LB-->|SL=0|
            |--------|
9:       |6|
8:       |3|
7:       |0|
6:       |0|
5:       |RA=36|
4:       |DL=0|
3:       |SL=0|
            |--------|
2:       |0|
1:       |4|
0: SB-->|4|
            |--------|
```

Listing 9: before mfib(1, var b)

```
      ST--> |////////|
            |--------|
16:       |14|
15:       |1|
14:       |0|
13:       |3|
12:       |RA=19|
11:       |DL=3|
10: LB-->|SL=0|
            |--------|
9:       |6|
8:       |3|
7:       |0|
6:       |0|
5:       |RA=36|
4:       |DL=0|
3:       |SL=0|
            |--------|
2:       |0|
1:       |4|
0: SB-->|4|
            |--------|
```

Listing 10: after mfib(1, var b)

```
      ST--> |////////|
            |--------|
14:       |2|
13:       |3|
12:       |RA=19|
```

```
11:       |DL=3|
10: LB-->|SL=0|
             |--------|
9:        |6|
8:        |3|
7:        |0|
6:        |0|
5:        |RA=36|
4:        |DL=0|
3:        |SL=0|
             |--------|
2:        |0|
1:        |4|
0: SB-->|4|
             |--------|
```

Listing 11: after mfib(3, var a)

```
      ST--> |////////|
             |--------|
7:        |0|
6:        |5|
5:        |RA=36|
4:        |DL=0|
3: LB-->|SL=0|
             |--------|
2:        |0|
1:        |4|
0: SB-->|4|
             |--------|
```

Listing 12: before mfib(2, var b)

```
      ST--> |////////|
             |--------|
9:        |7|
8:        |2|
7:        |0|
6:        |5|
5:        |RA=36|
4:        |DL=0|
3: LB-->|SL=0|
             |--------|
2:        |0|
1:        |4|
0: SB-->|4|
             |--------|
```

Listing 13: after mfib(2, var b)

```
      ST--> |////////|
             |--------|
7:        |3|
6:        |5|
5:        |RA=36|
4:        |DL=0|
3: LB-->|SL=0|
             |--------|
2:        |0|
1:        |4|
```

```
0: SB-->|4|
        |--------|
```

Listing 14: after mfib(4, var n)

```
      ST--> |////////|
            |--------|
0: SB-->|8|
        |--------|
```

Listing 15: before putint(n)

```
      ST--> |////////|
            |--------|
1:      |8|
0: SB-->|8|
        |--------|
```

Listing 16: after putint(n)

```
      ST--> |////////|
            |--------|
0: SB-->|8|
        |--------|
```