apl. Prof. Dr. Thomas Noll
Friedrich Gretz, Souymodip Chakraborty

Exercise 1 (Procedure Stack): (1 Points)

Which of the following procedure stacks could be result of the execution of an EPL-programm? Why?

a) $p_1 = 13:3:9:1:4:3:2:2:4:5:5:15:1:3:2:12:0:0:0:17:3$

b) $p_2 = 13:3:9:1:4:3:2:2:5:4:5:15:1:3:2:12:0:0:0:17:3$

c) $p_3 = 13:3:9:1:4:3:2:2:8:4:5:15:1:3:2:12:0:0:0:17:3$

Exercise 2 (Abstract machine execution): (2 Points)

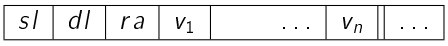
Consider the following intermediate code:

```
   ⋮
7:LOAD(1, 2);     (dif, off)
8:ADD;
9:RET;
   ⋮
26:CALL(38, 1, 3);(ca, dif, loc)
```

Give the next four states of the abstract machine starting in:

$$(ca, d, p) := (7, -3, 9:4:26:3:7:4:3:36:5:10:4:40:1:2:\ldots)$$

Recall that the procedure stack has the form:



and the *base*-function is defined as:

$$\begin{aligned} base(p, 0) &:= 1 \\ base(p, dif + 1) &:= base(p, dif) + p.base(p, dif) \end{aligned}$$

Exercise 3 (Translation function): (3 Points)

In addition to `while`-loops we want to have `for`-loops with implicit declaration of the counter variable in our example programming language:

$$\texttt{for (var } I := A \texttt{ ; } B \texttt{ ; } C_1 \texttt{) } C_2$$

a) Extend the translation function ct accordingly.

b) Generate intermediate code for

```
for (var x := 0; x < 10; x := x + 1) P();
```

without parameters for the `CALL` instruction generated for `P()`.

Exercise 4 (Code generation): (4 Points)

Implement `generator.Generator.translateWHILE(AST)` which given an abstract syntax tree returns Jasmin Code in a string. *Hint: It is a good approach to write methods for every language construct and call them recursively. Once you get the idea, it is actually less effort than you might think!*
To test your implementation you can write code in *WHILE* run it through our compiler, save the output and use Jasmin to build executable Java class files. The latter you can execute and observe their behaviour.
This exercise concludes the implementation of our "While to Jasmin" compiler!