SAARLAND UNIVERSITY
COMPUTER SCIENCE

Prof. Dr. Reinhard Wilhelm
Prof. Dr. Sebastian Hack
Dipl.-Inform. Daniel Grund
Michael Jacobs, M.Sc.

Compiler Construction WS11/12

Exercise Sheet 7

Please hand in the solutions to the theoretical exercises until the beginning of the lecture next Friday 2011-12-09, 12:00. Please write the number of your tutorial group or the name of your tutor on the first sheet of your solution.

Exercise 7.1 Statements and Expressions (Points: 2+2+2+2)

Translate the following C-statements into valid CMa-instruction sequences. Assume `i`, `x` and `y` to be global variables.

- `i++;`
- `x = &y;`
- `while(e_1) {s_1; if (e_2) continue; s_2;}`
- `while(e_1) {s; if (e_2) break;}`

Exercise 7.2 Extreme Pointer (Points: 4+4)

We have seen in the lecture that stack and heap grow towards each other in the store S. SP and NP denote the Stack and the New pointer, respectively. During execution of a program we must make sure that SP and NP do not pass each other. Without any extra provisions we would have to compare SP and NP on every update of one of the two values. To save some work we introduce EP, the Extreme Pointer, which denotes the uppermost cell to which SP may point to during the execution of the current function. This relieves us from checking for a collision whenever SP is manipulated. EP can be determined statically. It depends on the maximal stack usage during expression evaluation. Let $t(e)$ denote the number of stack cells needed to evaluate expression e. Assume e to be of the following form:

$$e ::= x \mid e_1[e_2] \mid e_1 = e_2 \mid e_1\ op_b\ e_2 \mid op_u e_1$$

- Give a recursive definition for the computation of $t(e)$! Explain your definition briefly.
- Consider the following expressions:

$$\begin{aligned} e_a &\equiv (\ldots((x_1 + x_2) + x_3)\ldots + x_n) \\ e_b &\equiv (x_1 + \ldots(x_{n-2} + (x_{n-1} + x_n))\ldots) \end{aligned}$$

  Assume $n \in \mathbb{N}$ and $n \geq 2$. Provide formulas to calculate $t(e_a)$ and $t(e_b)$ depending on n. Prove the correctness of your formulas by induction.

Exercise 7.3 Switch (Points: 2+2+2+2)

Translate the following switch statement into valid CMa code using the code generation rule in the lecture slides. Use a context with $\rho(n) = (L, 3)$ and $\rho(i) = (G, 4)$.

```
switch (n)
{
    case 0: i = n+2; break;
    case 1: i = -n; break;
    case 2: i = 1; break;
    default: break;
}
```

Consider the following questions regarding switch statements.

1. Is this code generation able to handle switch statements where gaps between case statements exist (i.e. cases are undefined)?
2. Is it always *feasible* to use jump tables to implement switch statements? Explain your answer!
3. Give a different alternative to implement code s for a general switch statement s. Discuss the (dis)advantages of your scheme.

Exercise 7.4 CMa Code Generation (Points: 6, Bonus-Points: 4)

- Translate the following C code into CMa code using the algorithm presented in the lecture. Remember to correctly compute the EP. Additional translation rules considering functions and whole programs can be taken from the lecture slides.

```
1    int sum;
2
3    int summarize (int n) {
4      int i, sum;
5      for (i = 0; i < n; ++i)
6        sum = i+sum;
7      return sum;
8    }
9
10   void main () {
11     sum = summarize(42);
12     return;
13   }
```

- Determine the state of the CMa before the execution of line 5 and after the execution of line 11. In each case, label variables in the stack and mark the stack frames.