Technische Universität München — *Summer Semester 08*
Fakultät für Informatik — *Vesal Vojdani*
Prof. Dr. H. Seidl — *vojdanig@in.tum.de*

Compiler Construction

Exercise Sheet 3

Deadline: 5. May 2008, at the lecture, in room 02.07.053, or by e-mail.

Exercise 1: *Variable initialization* *6 Points*

Extend the translation function *code* to initialization constructions, such as `int x = 5`. If no initializer is given, then you should set the variable to its null value, i.e., set the entire allocated area to zeroes. Structs and arrays initializers are given as a list of constant expressions, e.g., `int a[3] = {1, 2, 4}`, which you can assume to be of the right size.

You now have to define cases like *code* $(\tau x = e;\ ss)\ (\rho, r)$. The code function can be expanded to take the parameter r, which indicates the next free cell on the stack.

Exercise 2: *Break and Continue* *4 Points*

Modify the scheme for translation of loops, to take care of the `break` statement, for the immediate termination of a loop, and `continue` statements for skipping to the next iteration of the loop. For this purpose, you may wish to extend the code generation function to take a further argument, a pair of labels (l_b, l_c), which characterize the labels that exit and continue the loop. Your solution need only give the translation function for one of the looping constructs, e.g., the `while`-statment.

Exercise 3: *Code generation: Function calls* *8 Points*

Generate the code for the following. Make sure it runs on the CMM (C Machine München) of the VAM system. It should run for a fairly long time, 111 recursive calls. Send the solution by e-mail as a nicely commented .cmm file.

```
int f(int n) {
  if (n % 2 == 0) return n/2;
  else return (3*n + 1);
}

int c(int n) {
  if (n <= 1) return 1;
  else return c(f(n));
}

int main() {
  return c(27);
}
```

Exercise 4: *Call-by-reference* *6 Points*

Change the way paramters are passed to procedures to adopt a call-by-reference translation scheme. This is how reference types in C++ and non-primitive types in Java are handled. Consider the following example:

```
int x;
int f(int i) {
  i = 27;
}

int main() {
  f(x);
}
```

The value of x should be 27 after the function call. If an expression is given as parameter, then a new memory location is created on the heap and reference to it is passed. However, since the caller has no access to it, this aspect is not so important. You can assume only variables are used as arguments.

a) Give the translation-scheme for function calls.

b) Write a VAM executable .cmm file for this example using your new translation function.