https://folk.ntnu.no/michaeng/tdt4205_21/
michael.engel@ntnu.no

Practical exercises
Spring 2021

Practical Exercises 6

Code generation

Please submit solutions on Blackboard by Monday, 26.04.2021 23:59

Notice: Please submit solutions on Blackboard in groups of two or three students.
The practical exercises will be graded and count as part of your final grade.

6.1 Code generation

This final practical exercise realizes the backend of your compiler by generating x86-64 assembler source code from your intermediate representation.
The VSL compiler in the provided archive is extended with a function `generate_program` in `generator.c`; this function is called from `main.c` after syntax tree and symbol table construction.
Implement this function so that it generates x86-64 assembly source code for the following constructs:

a. Global string table (0.5 p.)

   Strings should be given numbered labels in a data segment.

b. Global variables (1 p.)

   Global variables should be given names corresponding to their declarations, prefixed with an underscore character "_" (see the guidline slides if you want to write your backend on macOS) to avoid names that clash with names from the system libraries.

c. Functions (1 p.)

   Functions should be placed in the text segment, named in the same manner as global variables, and set up/remove a stack frame. Furthermore, they should initiate a recursive traversal of their syntax subtrees, so that the remaining constructs can be generated.

d. Function parameters (1 p.)

   Function parameters should follow the standard calling convention covered in the course. Copies can be placed at the bottom of the function's stack frame to make their runtime address computable from their sequence number and liberate the registers for further function calls.

e. Arithmetic expressions (1 p.)

   Arithmetic expressions should be translated so as to leave their result in the RAX register, and remove any intermediate calculations from the generated program's run time stack.

f. Assignment statements (0.5 p.)

   Assignment statements should copy the result of an expression to the address of the assigned variable.

g. PRINT statements (0.5 p.)

   PRINT statements can be translated into a sequence of `printf` system calls, with one call per item in the PRINT statement's list.

h. RETURN statements (0.5 p.)

   RETURN statements should leave the result of their expression in the RAX register, remove the function's stack frame, and return control to the caller.

i. Local variables and function calls (2 p.)

   Implement local variables and function calls in `generator.c`.

j. Control structures (2 p.)

   Implement conditionals, while loops and continue