# TDT4205 Compiler Construction
# Assignment/Problem Statement 5

## 1 Theory 30%

### 1.1 Part I

Create control flow graphs for the following program fragments:

1. for ( a ; b ; c) d ; e;
2. a ; while ( b ) d ; c ; e ;
3. a ; do d ; c ; while ( b ); e ;

## 2 Programming 70%

To ensure that everyone is on track the first part of the programming exercise will not be graded and the code for it will be provided mid-way (approx 3 weeks from now).

Since this exercise is released early, there is some skeleton code available. The generator.c file is to be placed in the src/ folder. The makefile in the top level directory (the one that does the compilation of the vslc.c). The vslc.c and vslc.h in the usual places.

The actual skeleton for 5 will be made available after the submission deadline of exercise 4.

### 2.1 Part I

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 code for the following constructs:

- **Global String table**: Strings should be given numbered labels in a data segment.
- **Global variables**: Global variables should be given names corresponding to their declarations, prefixed with an underscore character ' ', so as to avoid names that clash with names from the system libraries.
- **Functions**: 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.

- **Function Parameters**: Function parameters should be expected to follow the standard calling conven- tion covered in lectures. Copies can be placed at the bottom of the function's stack frame, to make their run-time address computable from their sequence number, and liberate the registers for further function calls.
- **Arithmetic expressions**: 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.
- **Assignment statements**: Assignment statements should copy the result of an expression to the address of the assigned variable.
- **"print" statements**: "print" statements can be translated into a sequence of 'printf' calls, with one call per item in the PRINT statement's list.
- **"return" statements** "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.

## 2.2 Part II 70 %

Implement the following constructs in generator.c:

1. Local variables 20%
2. Function calls 20%
3. Conditionals (IF and relations) 15%
4. While loops 10%
5. Continue (null statement) 5%