TDT 4205
Problem Set 6

The deadline for this problem set is friday, November 16th. Submissions will only be accepted through *It's learning*.

1 Control flow graphs

1.1 10%

Create a control flow graph for the following program:

```
i = j = 1;
b = c = true;
while (i <= j) {
    j++;
    if (j % 2 == 0)
        b = false;
    else if (j % 3 == 0)
        c = false;
    else
        b = c = true;
    if (b && c)
        i++;
    else if (b)
        i += 2;
    else
        i += 3;
}
```

2 Assembler programming warm-up

2.1 15%

Write an assembler program which writes the lyrics of the song "99 Bottles of Coke" on standard output. (The lyrics of a very similar work can be found at *http://en.wikipedia.org/wiki/99_Bottles_of_Beer*, but we won't be held responsible for combining alcohol and assembler...) The program may terminate at 0 without any final verse.

3 Simple VSL expressions

These tasks finalize the contents of the symbol table in *symtab.c*, and begin an implementation of a code generator in `generator.c`.

3.1 20%

Extend the function `find_symbols` to assign the `stack_offset` value of each variable entered in the symbol table to a suitable location in an activation record, by the following rules:

In a list of n parameters, each should receive a positive `stack_offset` from $8 + (n - 1) * 4$ with a stride of -4, e.g. finding a parameter list (a,b,c), the offset 16 should be given to a, 12 to b, and 8 to c).

Stack offsets for variable declarations in functions should begin at -4 for each function, proceeding with a stride of -4 for each variable declared (irrespective of nested blocks).

As an example, in the program

```
FUNC foo ( n )
{
    VAR a,b
    IF ( n ) THEN
    {
        VAR c
        c := n/2
    }
    FI
    RETURN bar()
}
FUNC bar (n)
{
   VAR d
   RETURN 12
}
```

the stack offsets should be -4 for a, -8 for b, -12 for c, and -4 for d (as it is declared in another function).

3.2 25%

Begin an implementation of the function `generate ( node_t *root )` in `generator.c` by covering the base case for a node of type `PROGRAM_N`. The code generated for this node should be placed on standard output, and

- start a data segment (see the macro `DATA_HEAD` in `asm_macros.h`)
- populate the data segment with labelled copies of the program's string constants (see the additional notes on assembly programming)
- create an entry point for the assembly code (see the macro `TEXT_HEAD` in `asm_macros.h`)

- recursively call itself to proceed with the `FUNCTION_LIST_N` node which embodies the rest of the program.

3.3 30%

Extend your generator to write code for single-function programs which contain only RETURN statements, variables and simple expressions/assignments, such as

```
FUNC main()
{
    VAR x,y
    x := 16
    y := 5
    x := 2*x
    y := x/y
    RETURN y
}
```

This translation will require generating code such that

- the function activation record is set up with locations to match each local variable
- assignment statements copy the result of the evaluated expression into the location for the variable which is assigned
- expressions evaluate to leave only their result at the top of the stack
- return statements leave the function value in EAX and return correctly

To test whether values are returned correctly (and to see any output from this program), note that the return value from an executed program is stored in the environment variable `$?` of the calling shell. Hence, executing `echo $?` after running your produced executable should produce "6" for a correctly compiled version of the program above.