Assessed Exercise 2, Task 3

Implement `codegenTask3` for the whole language.

```
PROG → DEC | DEC; PROG
DEC → def ID (VARDEC) = E
VARDEC →  ε | VARDECNE
VARDECNE → ID | VARDECNE, ID
ID → ... (identifiers)
INT → ... (Integers)
E →  INT
   | ID
   | if E COMP E then E else E endif
   | (E BINOP E)
   | (E)
   | skip
   | (E; E)
   | while E COMP E do E endwhile
   | repeat E until E COMP E endrepeat
   | ID := E
   | ID(ARGS)
   | break
   | continue
ARGS → ε | ARGSNE
ARGSNE → E | ARGSNE, E
COMP → == | < | > | <= | >=
BINOP → + | - | * | /
```

Recall that the relevant definitions are here, here and here. If you don't want to implement a feature, simply throw `CodegenException` when the code generator encounters this feature.

This task is more difficult than the previous two in the sense that we have not discussed how to implement division, `break` and `continue` in the lectures. The actual changes to the code generator are small and simple. Note that / is integer division, i.e. remainders are ignored (e.g. 7/5=1).

Meaning of `break` and `continue`. The commands `break` and `continue` work just as the eponymous statements do in Java, see e.g. here in their unlabelled versions). They always occur within a `while` or `repeat` loop. This is checked by semantic analysis. You can assume that the test suite will only contain correct uses of `break` and `continue`.

When inside a loop, `break` will immediately break out of the innermost containing loop and execute the command following that innermost loop. For example consider the following declaration.

```
def f ( x ) = (
   while x > 0 do
      if x > 500 then
         break
      else
         x := (x-1)
      endif
   endwhile;
   x )
```

With this definition, `f ( 1000 )` will return 1000, while `f ( 432 )` will return 0. Relatedly, the outer loop in

```
while x > 0 do (
   repeat (
      break; x := (x+1) )
   until x > 0 endrepeat;
   x := (x-1) )
```

```
endwhile
```

will be executed exactly x times (assuming x ≥ 0).

When inside a loop, `continue` will immediately abandon the current round of the innermost containing loop and go to checking the condition. So the following loop never terminates whenever x &geq; 0.

```
repeat
   ( continue; x = ( x - 1 ) )
until x < 0 endrepeat
```

Both, `break` and `continue`, can leave anything in the accumulator.

Important side-condition. The grammar allows us to have expressions like `1 + break`. You do not have to cater for this. All uses of `break` and `continue` will be 'normal', subject to the restrictions `break` and `continue` must meet in Java.