Assessed Exercise 2, Task 2

Implement `codegenTask2` for the part of the language in black. No need to implement code generation for the red parts. No test for this task will feature programs containing language constructs in red. Your code generator can throw `CodegenException` for the ASTs that correspond to language constructs in red, or you can return MIPS code, it does not matter.

```
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.

Assignment, as well as commands `while`, `repeat` and `skip` can return whatever they want, it's not specified. No test will depend on the return value of such commands. Sequential composition `(E; E')` returns whatever its second command returns.

Note that `if`, `while` and `repeat` constructs all carry out comparisons. I suggest to factor the translation of comparisons into a separate method to avoid code duplication.