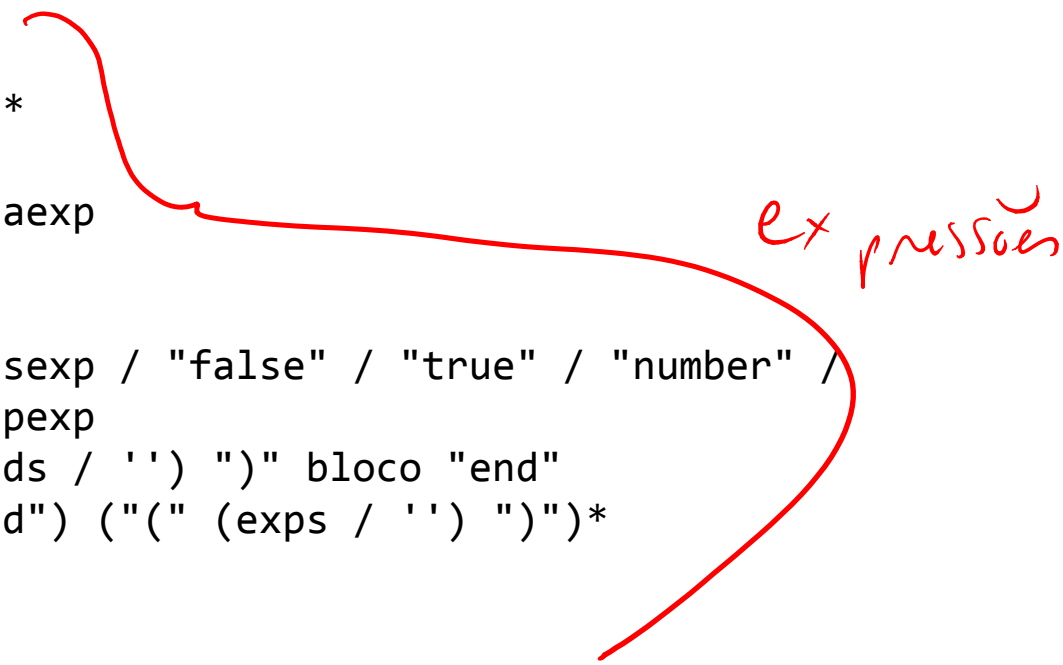
Compiladores II

Fabio Mascarenhas - 2014.2

http://www.dcc.ufrj.br/~fabiom/comp2

SmallLua - sintaxe

```
bloco <- stat* (ret / '')
stat  <- "while" exp "do" bloco "end" / "local" "id" "=" exp /
         "id" "=" exp / "function" "id" "(" (ids / '') ")" bloco "end" /
         "if" exp "then" bloco ("else" bloco / '') "end" /
         pexp
ret   <- "return" exp
ids   <- "id" ("," "id")*
exps  <- exp ("," exp)*
exp   <- lexp ("or" lexp)*
lexp  <- rexp ("and" rexp)*
rexp  <- cexp (rop cexp)*
cexp  <- aexp ".." cexp / aexp
aexp  <- mexp (aop mexp)*
mexp  <- sexp (mop sexp)*
sexp  <- "-" sexp / "not" sexp / "false" / "true" / "number" /
         "string" / lmb / pexp
lmb   <- "function" "(" (ids / '') ")" bloco "end"
pexp  <- ("(" exp ")" / "id") ("(" (exps / '') ")")*
rop   <- "<" / "==" / "~="
aop   <- "+" / "-"
mop   <- "*" / "/"
```



Funções definidas por casos

- Várias operações que vamos fazer serão operações em nós de uma árvore sintática

- Cada tipo de nó tem uma implementação específica de cada operação

```
local tos = casef("tos", tostring)

tos["while"] = function (node, ident)
    local out = {
        (" "):rep(ident), "while ",
        tos(node.cond), " do\n"
    }
    for _, stat in ipairs(node.body) do
        out[#out+1] = tos(stat, ident+4)
    end
    out[#out+1] = (" "):rep(ident) ..
        "end\n"
    return table.concat(out)
end
```

```
function tos.set(node, ident)
    return (" "):rep(ident) ..
        tos(node.rval) ..
        " = " .. tos(node.rval) .. "\n"
end

function tos.add(node)
    return "(" .. tos(node.left) ..
        " + " .. tos(node.right) .. ")"
end
```

Dojo

- Completar a função por casos `tos` incluindo os casos que estão faltando

- Implementar uma função por casos fixpos que converte a posição em cada nó de “quantos tokens faltam para o final” para linha e coluna na entrada (dica: você vai precisar da entrada e da lista de tokens)