Universidade da Beira Interior

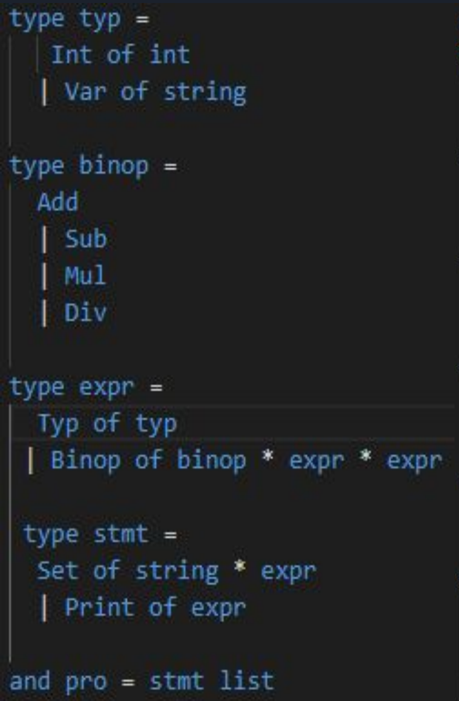
Processamento de Linguagens

Compilador de linguagem aritmética para arquitetura MIPS

Elaborado por:
Miguel Gregório
João Domingos

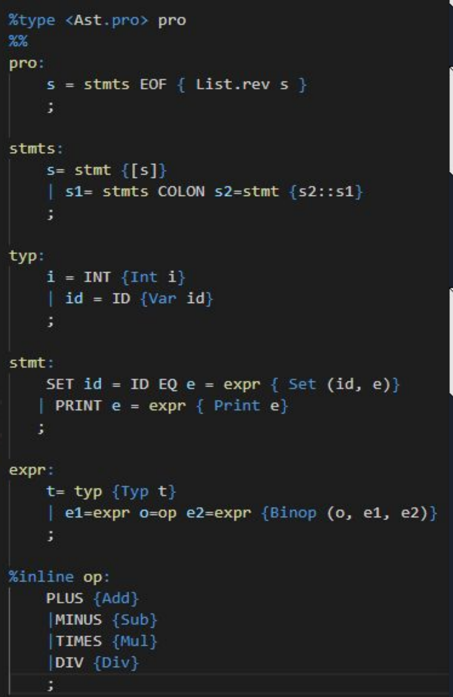
Professor:
Simão Melo de Sousa

Árvore de Sintaxe



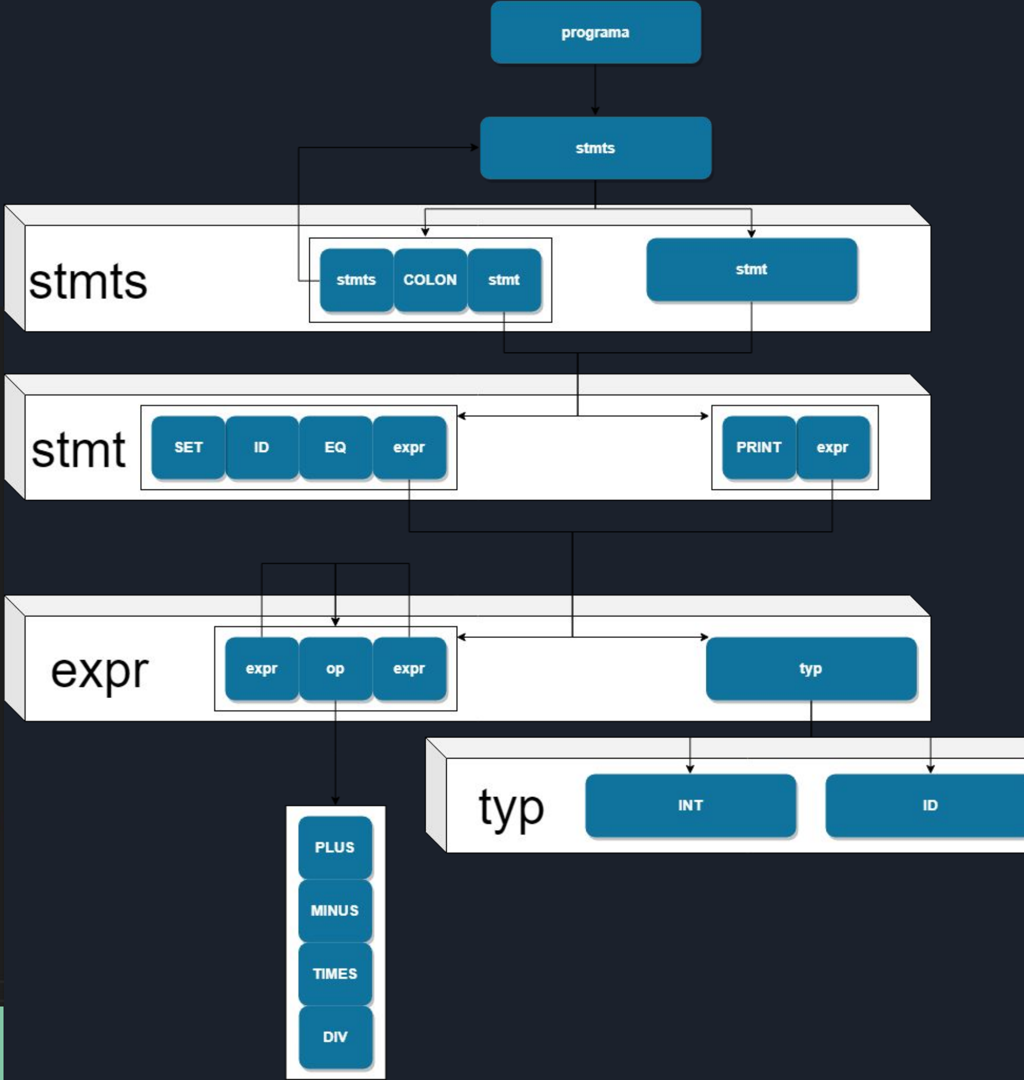
```
type typ =
  Int of int
  | Var of string

type binop =
  Add
  | Sub
  | Mul
  | Div

type expr =
  Typ of typ
  | Binop of binop * expr * expr

type stmt =
  Set of string * expr
  | Print of expr

and pro = stmt list
```

ast.mli



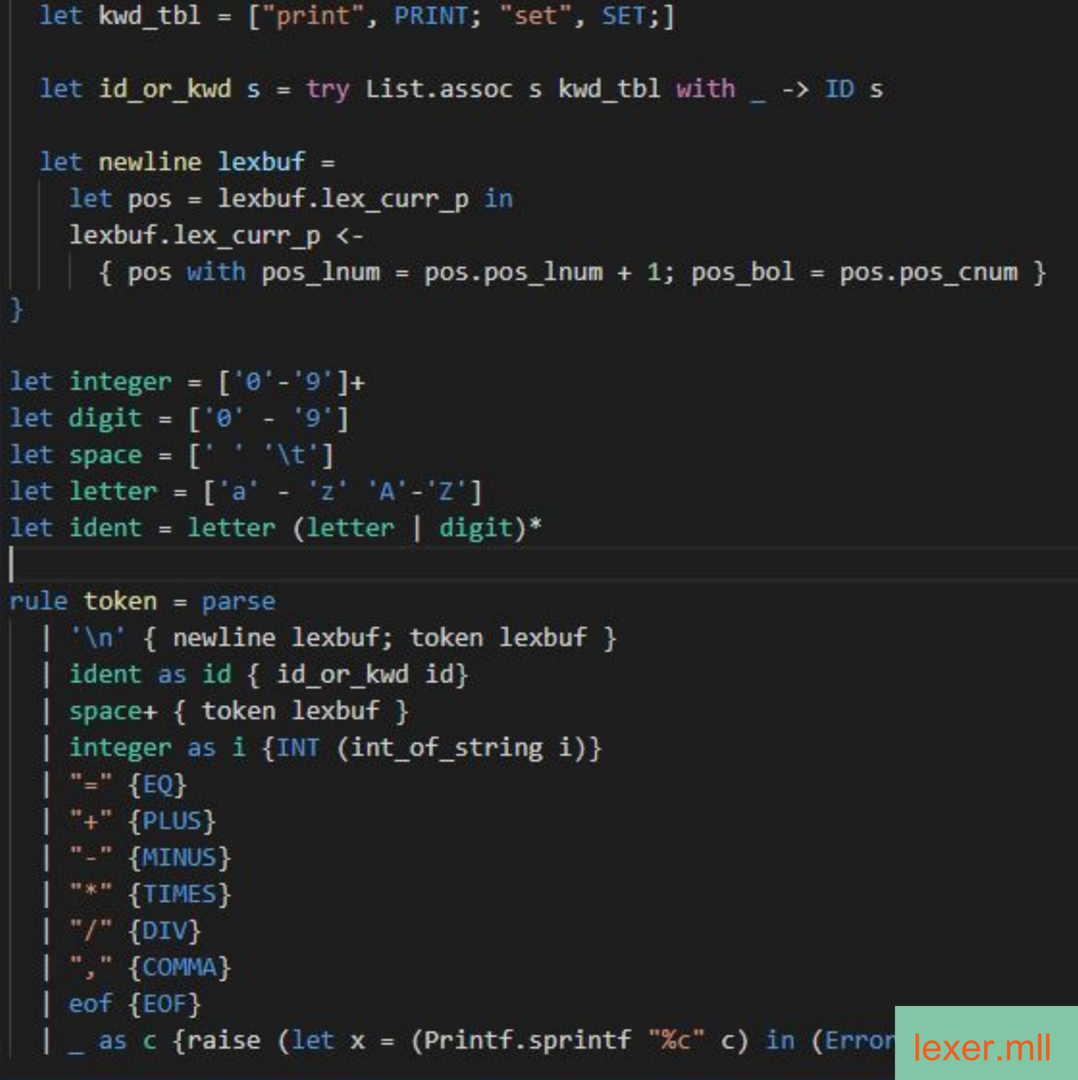
```
%type <Ast.pro> pro
%%
pro:
    s = stmts EOF { List.rev s }
    ;

stmts:
    s= stmt {[s]}
    | s1= stmts COLON s2=stmt {s2::s1}
    ;

typ:
    i = INT {Int i}
    | id = ID {Var id}
    ;

stmt:
    SET id = ID EQ e = expr { Set (id, e)}
   | PRINT e = expr { Print e}
   ;

expr:
    t= typ {Typ t}
    | e1=expr o=op e2=expr {Binop (o, e1, e2)}
    ;

%inline op:
    PLUS {Add}
    |MINUS {Sub}
    |TIMES {Mul}
    |DIV {Div}
    ;
```

parser.mly



Conversão de tokens

PRINT	print
SET	set
EQ	=
COMMA	,
INT	[0 - 9] +
ID	[a - Z] ([a-Z]| INT) *
OP	{ + , - , * , / }



```
 let kwd_tbl = ["print", PRINT; "set", SET;]

 let id_or_kwd s = try List.assoc s kwd_tbl with _ -> ID s

 let newline lexbuf =
   let pos = lexbuf.lex_curr_p in
   lexbuf.lex_curr_p <-
     { pos with pos_lnum = pos.pos_lnum + 1; pos_bol = pos.pos_cnum }
}

let integer = ['0'-'9']+
let digit = ['0' - '9']
let space = [' ' '\t']
let letter = ['a' - 'z' 'A'-'Z']
let ident = letter (letter | digit)*

rule token = parse
  | '\n' { newline lexbuf; token lexbuf }
  | ident as id { id_or_kwd id}
  | space+ { token lexbuf }
  | integer as i {INT (int_of_string i)}
  | "=" {EQ}
  | "+" {PLUS}
  | "-" {MINUS}
  | "*" {TIMES}
  | "/" {DIV}
  | "," {COMMA}
  | eof {EOF}
  | _ as c {raise (let x = (Printf.sprintf "%c" c) in (Error
```

lexer.mll

Tipos de Instruções

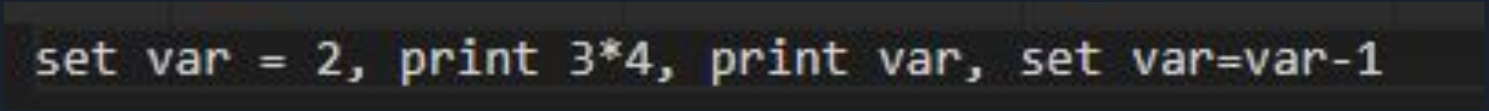
Definir variável

Operações

Imprimir variável

Redefinir variável

Inputs esperados



```
set var = 2, print 3*4, print var, set var=var-1
```

Ficheiro test.ar

Compilador

Tipos de Instruções

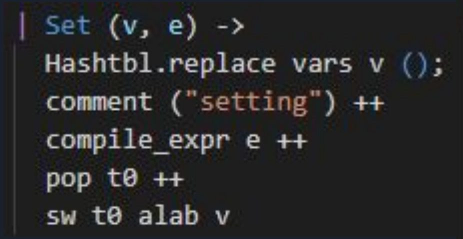
Definir variável

Operações

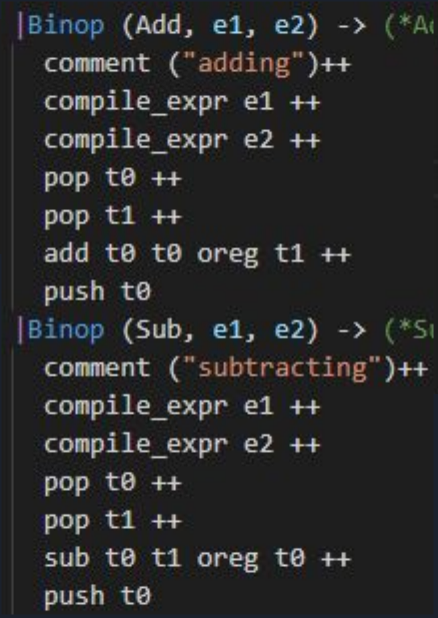
Imprimir variável

Redefinir variável

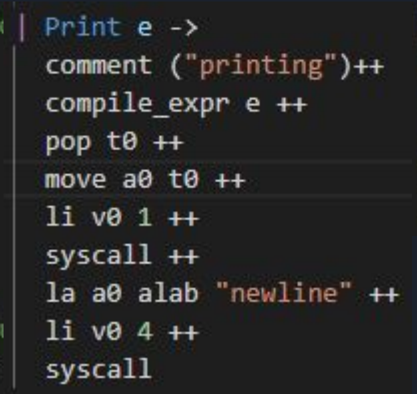
Arquitetura MIPS



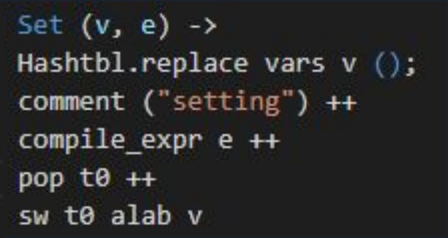
```
| Set (v, e) ->
  Hashtbl.replace vars v ();
  comment ("setting") ++
  compile_expr e ++
  pop t0 ++
  sw t0 alab v
```



```
|Binop (Add, e1, e2) -> (*A
  comment ("adding")++
  compile_expr e1 ++
  compile_expr e2 ++
  pop t0 ++
  pop t1 ++
  add t0 t0 oreg t1 ++
  push t0
|Binop (Sub, e1, e2) -> (*S
  comment ("subtracting")++
  compile_expr e1 ++
  compile_expr e2 ++
  pop t0 ++
  pop t1 ++
  sub t0 t1 oreg t0 ++
  push t0
```



```
| Print e ->
  comment ("printing")++
  compile_expr e ++
  pop t0 ++
  move a0 t0 ++
  li v0 1 ++
  syscall ++
  la a0 alab "newline" ++
  li v0 4 ++
  syscall
```



```
Set (v, e) ->
Hashtbl.replace vars v ();
comment ("setting") ++
compile_expr e ++
pop t0 ++
sw t0 alab v
```

compile.ml