Linguagem TPL-02

Especificação

Para o desenvolvimento dos trabalhos práticos, utiliza-se uma linguagem designada por TPL-02 (*Trivial Programming Language 02*) na qual se encontram características diversas das linguagens de programação imperativas, que exercitam um leque alargado de situações.

1 Sintaxe

A linguagem TPL-02 é apresentada informalmente pela gramática das figuras 1 a 5 (ver páginas 1 a 3). Esta gramática já se encontra numa forma fácilmente adaptável para especificar como input para um gerador de parsers LALR(1) como o CUP. Por uma questão de legibilidade (e tipografia) a gramática foi repartida em várias secções.

```
program ::= decl

decls ::=                              /* Lista de declarações */
        | decl decls

decl ::=                               /* Declaração dum nome: */
       ids '=' type                    /* Definição de tipo */
     | ids ':' type                    /* Variável, tipo explícito */
     | ids ':' type ASSIGN exp         /* Variável, tipo explícito, init */
     | ids ':' type EQ exp             /* Constante, tipo explícito */
     | ids          EQ exp             /* Constante, tipo implícito */

formals ::=                            /* Lista de parâmetros formais */
          | formal_decl formals

formal_decl ::=                        /* Parâmetro formal: */
              ids                      /* Tipo implícito */
            | ids ':' type             /* Tipo explícito */

ids ::= id                             /* lista de identificadores */
      | id ',' ids

id ::= ID                              /* um NOME */
     | OP op                           /* um nome como OPERADOR */

op ::= '+' | '-' | '*' | '/' | '%'     /* Os operadores */
     | AND | OR | NOT
     | LT | LE | EQ | NE | GE | GT
```

Figura 1: EBNF para a linguagem TPL-02 – Declarações

2 Elementos lexicais

As convenções lexicais são as habituais. O analisador lexical deverá reconhecer constantes (literais) dos 3 tipos apresentados, nomeadamente:

Constantes inteiras (`INT_LIT`). São constantes inteiras decimais, expressas pela definição habitual. Só são contempladas as este nível as constantes positivas, ie. sem sinal.

```
type ::=                          /* -- ASSINATURA DE TIPO -- */
         typx                       /* Um só tipo (fim de lista) */
       | '(' type ')'               /* Agrupamento sintactico */
       | typx ',' type              /* Tuplo de tipos (lista) */

typx ::=                          /* -- EXPRESSÃO DE TIPO -- */
         ID                         /* Identificador de tipo */
       | INT                        /* Inteiro */
       | REAL                       /* Vírgula flutuante */
       | BOOL                       /* Booleano */
       | VOID                       /* Void (ex. instruções de controle) */
       | type RETURNS type          /* Tipo funcional */
       | '[' exp ']' type           /* Tipo "Array" */
       | '{' formals '}'            /* Tipo agregado (classe) */
```

Figura 2: EBNF para a linguagem TPL-02 – Declarações de tipo

```
exp ::= sexp                            /* -- EXPRESSÃO -- */
      | sexp ',' exp
      | '(' exp ')'

sexp ::= sexp OR sexp                     /* Operadores booleanos */
       | sexp AND sexp
       | NOT sexp

       | sexp LT sexp                     /* Operadores de comparação */
       | sexp LE sexp
       | sexp EQ sexp
       | sexp NE sexp
       | sexp GE sexp
       | sexp GT sexp

       | sexp '+' sexp                    /* Operadores aritméticos */
       | sexp '-' sexp
       | sexp '*' sexp
       | sexp '/' sexp
       | sexp '%' sexp
       | '-' sexp

       | sexp '.' id                      /* Nomes qualificados */
       | sexp '[' exp ']'                 /* Referências a arrays */
       | sexp '(' exp ')'                 /* Aplicação funcional */

       | ID                               /* Nome simples */

       | INT_LIT                          /* Constante inteira */
       | REAL_LIT                         /* Constante em vírgula flutuante */
       | BOOL_LIT                         /* Constante booleana */

       | '[' exp ']'                      /* Literal de array */
       | MAP '(' formals ')' '[' stats ']' /* Literal funcional */
       | MAP '(' formals ')' RETURNS type /* Idem, com tipo explícito */
                             '[' stats ']'
       | CLASS '(' formals ')' '[' stats ']' /* Literal de classe */
```

Figura 3: EBNF para a linguagem TPL-02 – Expressões

```
primary ::= prim
          | prim ',' primary
          | '(' primary ')'


prim ::= id
       | primary '.' primary
       | primary '[' exp ']'
```

Figura 4: EBNF para a linguagem TPL-02 – Expressões restritas

```
stats ::= /* VAZIA */                  /* -- INSTRUÇÕES -- */
        | stat stats

stat ::= decl                            /* Declaração */
       | primary ASSIGN exp              /* Afectação */
       | primary '(' exp ')'             /* Chamada de função */
       | RETURN exp                      /* Retorno de função */
       | BREAK                           /* Saída de ciclo */
       | SKIP                            /* Próxima iteração */
       | RETRY                           /* Re-iniciar iteração */
       | COND '[' clauses ']'            /* Instrução condicional */
       | WHILE '[' clauses ']'           /* Instrução de ciclo condicional */
       | '[' stats ']'                   /* Agrupamento de instruções */

clauses ::= exp RETURNS stats           /* Instrução com guarda */
          | exp RETURNS stats '|' clauses
          | exp RETURNS stats '|' ELSE RETURNS stats
```

Figura 5: EBNF para a linguagem TPL-02 – Instruções

Constantes de vírgula flutuante (`REAL_LIT`). Tal como as anteriores, estas seguem as convenções habituais. No entanto, deverão ser reconhecidos, por exemplo, valores nas seguintes formas: "`.8`", "`0.005`", "`123e+17`", "`1.5e2`", "`3e-5`" e "`.200284E3`".

Constantes booleanas (`BOOL_LIT`). As constantes booleanas, literalmente `true` e `false`.

3 Semântica

As definições que antecedem o não-terminal `statement` dum programa serão designadas como definições *globais*, pelo que são reconhecidas em todo o programa.

Não é necessário definir um nome antes de o usar, bastando para tal que este esteja definido no mesmo "bloco" (âmbito ou "scope"), mesmo que posteriormente ao uso.