A Linguagem MiniJava

MiniJava é um subconjunto da linguagem Java, usado para ensino de projeto e implementação de compiladores. Como é um subconjunto, todo programa MiniJava é um programa Java válido que pode ser executado pela JVM, fornecendo um meio simples dos alunos testarem a saída dos seus compiladores sem ter um compilador MiniJava disponível.

MiniJava restringe a linguagem Java para ter apenas inteiros, booleanos, vetores de inteiros e classes, removendo interfaces, números de ponto flutuante, classes abstratas, strings, vetores de outros tipos etc. Também não há sobrecarga de métodos, ou métodos estáticos, exceto pelo método **main** da classe principal do programa. O comando `System.out.println` de MiniJava só pode imprimir números.

Um exemplo simples de MiniJava:

```
class Factorial{
    public static void main(String[] a){
        System.out.println(new Fac().ComputeFac(10));
    }
}

class Fac {
    public int ComputeFac(int num){
        int num_aux;
        if (num < 1)
            num_aux = 1;
        else
            num_aux = num * (this.ComputeFac(num-1));
        return num_aux ;
    }
}
```

Um programa MiniJava está restrito a um único arquivo fonte, não existe o conceito de pacote. Existem outros programas exemplo na página de MiniJava.

Especificação Léxica

Note que MiniJava trata `System.out.println` como uma *palavra reservada*, não como uma chamada do método `println`. Isso facilita o restante do compilador. Também não há um operador de divisão.

- Espaços em branco: `[ \n\t\r\f]`
- Comentários: dois tipos de comentário, um começando com `//` e indo até o final da linha, o outro começando com `/*` e terminando com `*/`, sem aninhamento
- Palavras reservadas: `boolean`, `class`, `extends`, `public`, `static`, `void`, `main`, `String`, `return`, `int`, `if`, `else`, `while`, `System.out.println`, `length`, `true`, `false`, `this`, `new`, `null`
- Identificadores: uma letra, seguido de zero ou mais letras, dígitos ou _
- Numerais: apenas números inteiros
- Operadores e pontuação: `(`, `)`, `[`, `]`, `{`, `}`, `;`, `.`, `,`, `=`, `<`, `==`, `!=`, `+`, `-`, `*`, `/`, `&&`, `!`

Sintaxe

A sintaxe é dada usando EBNF. Meta-símbolos EBNF usados como tokens estão entre aspas simples.

```
PROG   → MAIN {CLASSE}
MAIN   → class id '{' public static void main ( String [ ] id ) '{'
CMD '}' '}'
CLASSE → class id [extends id] '{' {VAR} {METODO} '}'
VAR    → TIPO id ;
METODO → public TIPO id '(' [PARAMS] ')' '{' {VAR} {CMD} return EXP ;
'}'
PARAMS → TIPO id {, TIPO id}
TIPO   → int '[' ']'
        | boolean
        | int
        | id
CMD    → '{' {CMD} '}'
        | if '(' EXP ')' CMD
        | if '(' EXP ')' CMD else CMD
        | while '(' EXP ')' CMD
        | System.out.println '(' EXP ')' ;
        | id = EXP ;
        | id '[' EXP ']' = EXP ;
EXP    → EXP && REXP
        | REXP
REXP   → REXP < AEXP
        | REXP == AEXP
        | REXP != AEXP
        | AEXP
AEXP   → AEXP + MEXP
        | AEXP - MEXP
        | MEXP
MEXP   → MEXP * SEXP
        | MEXP / SEXP
        | SEXP
SEXP   → ! SEXP
        | - SEXP
        | true
        | false
        | num
              | null
        | new int '[' EXP ']'
        | PEXP . length
        | PEXP '[' EXP ']'
        | PEXP
PEXP   → id
        | this
        | new id '(' ')'
        | '(' EXP ')'
        | PEXP . id
        | PEXP . id '(' [EXPS] ')'
EXPS   → EXP {, EXP}
```