T1 – 2ª parte

Implemente um compilador que gere *bytecodes* Java para a linguagem definida pela gramática abaixo, a saída deve ser um arquivo texto com mnemônicos que representem as instruções. O arquivo gerado deve ser montado pelo *Jasmin.* A linguagem deve manipular apenas três tipos de dados: *int, float* e *string*. As produções para expressões (lógicas, relacionais e aritméticas) devem ser definidas.

<Programa> → <ListaFuncoes> <BlocoPrincipal>
| <BlocoPrincipal>

<ListaFuncoes> → <ListaFuncoes> <Função>
| <Funcao>

<Funcao> → <TipoRetorno> **id** (<DeclParametros>) <BlocoPrincipal>
| <TipoRetorno> **id ( )** <BlocoPrincipal>

<TipoRetoro> → <Tipo>
| **void**

<DeclParametros>→ <DeclParametros>**,** <Parametro>
| <Parametro>

<Parametro> → <Tipo> **id**

<BlocoPrincipal> → **{**<Declaracoes> <ListaCmd>**}**
| {<ListaCmd>**}**

<Declaracoes> → <Declaracoes> <Declaracao>
| <Declaração>

<Declaracao> → <Tipo> <ListaId>**;**

<Tipo> → **int**
| **string**
| float

<ListaId> → <ListaId>, **id**
| **id**

<Bloco> → **{** <ListaCmd> **}**

<ListaCmd> → <ListaCmd> <Comando>
| <Comando>

<Comando> → <CmdSe>
| <CmdEnquanto>
| <CmdAtrib>
| <CmdEscrita>
| <CmdLeitura>
| <ChamadaProc>
| <Retorno>

<Retorno> → **return** <ExpressaoAritimetica>**;**

		| return literal;
<CmdSe>	→	**if (**<ExpressaoLogica>**)** <Bloco> | **if (**<ExpressaoLogica>**)** <Bloco> **else** <Bloco>
<CmdEquanto>	→	**while** (<ExpressaoLogica>) <Bloco>
<CmdAtrib>	→	**id =** <ExpressaoAritmetica>**;** **| id = literal;**
<CmdEscrita>	→	**print (**<ExpressaoAritmetica>**);** **| print (literal);**
<CmdLeitura>	→	**read (id);**
<ChamadaProc>	→	<ChamaFunção>**;**
<ChamadaFuncao>	→	**id (**<ListaParametros>**)** | **id ( )**
<ListaParametros>	→	<ListaParametros>**,** <ExpressaoAritmetica> | <ListaParametros>**, literal** | <ExpressaoAritmetica> | **literal**

- Uma expressão relacional tem como termos expressões aritméticas e envolve os operadores: <, >, <=, >=, ==, !=.

- Uma expressão lógica tem como termos expressões relacionais e envolve os seguintes operadores: && (conjunção), || (disjunção), ! (negação). Os operadores binários && e || têm a mesma precedência e a associatividade é da esquerda para a direita, o operador ! é um operador unário e possui a maior precedência.

- Os operadores aritméticos (+, -, *, /) têm associatividade da esquerda para direita e a precedência usual.

- Uma expressão aritmética term como termos: identificadores de variáveis, constantes inteiras, constantes com ponto flutuante ou chamadas de funções.

- Nas expressões lógicas ou aritméticas os parênteses alteram a ordem de avaliação.

- Os *tokens* identificador (**id**), constante inteira, constante com ponto flutuante e constante cadeia de caracteres (**literal**) devem ser definidos como ocorrem usualmente em linguagens de programação.