Primeira Prova de Construção de Compiladores. Primeiro Semestre de 2004, DC-UFSCar
Prof. José de Oliveira Guimarães
Turma B (Quinta feira).

1.(5.0) Dada a gramática

```
Program ::=  "let" VarDecList "in"  Expr
VarDecList ::= VarDec { VarDec }
VarDec ::= IDENT "=" NUMBER
Expr ::= SimpleExpr { AddOper SimpleExpr }
AddOper ::= "+" | "-"
SimpleExpr ::= IDENT | NUMBER | IfExpr
IfExpr ::= "if" Expr "then" Expr "else" Expr
```

Faça:
a) os métodos `program`, `varDec` e `ifExpr` da classe `Compiler` do analisador sintático com a inserção das variáveis na tabela de símbolos e todas as conferências semânticas (que são relacionadas à declaração de variáveis). Naturalmente, construa a ASA durante esta análise. Assuma que, ao chamar `ifExpr`, o token corrente seja `Symbol.IF`. A classe compiler possui um método `nextToken` responsável pela análise sintática. O token corrente é colocado na variável de instância `token` de Compiler. As constantes da análise léxica estão na classe `Symbol`. O método `varDecList` retorna um vetor com objetos da classe `Variable`. Há uma variável de instância `symbolTable` na classe `Compiler` do tipo `Hashtable` (lembre-se de que esta classe possui métodos `Object get(Object key)` e `Object put(Object key, Object value)`). O método `error` de `Compiler` deve ser utilizado para emitir mensagens de erro. `IDENT` e `NUMBER` são terminais. Se o token corrente for `Symbol.IDENT`, a variável de instância `stringValue` de `Compiler` guarda o identificador (uma string). Se token for `Symbol.NUMBER`, a variável de instância `numberValue` de Compiler guarda o valor do número, que é inteiro;

b) as classes `Program` e `IfExpr` (que representa uma expressão if). Coloque apenas as variáveis de instância nas classes. Assuma que exista uma classe abstrata `Expr`. Não se esqueça das heranças.

2. Nos compiladores estudados nesta disciplina, utilizamos objetos para representar os tipos. Poderia ser diferente: as constantes inteiras `Symbol.INTEGER`, `Symbol.CHAR` e `Symbol.BOOLEAN` poderiam representar os tipos. Por exemplo, a classe `Variable` teria uma variável de instância "`int type`" e `getType()` simplesmente retornaria `type`. Naturalmente, a classe `Expr` define um método abstrato "`int getType()`". Na classe `Compiler`, a TS é referenciada pela variável `symbolTable`. Assumindo isto, faça:

a) as conferências semânticas do trecho de código que aparece a seguir. Copie este código na folha de respostas. Evidentemente, somas e subtrações só podem ser feitas com inteiros;
b) o método `getType()` da classe `CompositeExpr`. Admita que a linguagem possua operadores `+`, `-`, `*`, `/`, `and` e `or`. A classe `CompositeExpr` possui três variáveis de instância:

```
Expr left, right;
int oper;
```

`oper` guarda um dos seguintes inteiros: `Symbol.PLUS`, `Symbol.MINUS`, `Symbol.MULT`, `Symbol.DIV`, `Symbol.AND` ou `Symbol.OR`.

```
// assuma que não existe tipo undefinedType
private Expr addExpr() {
  int op;   Expr left, right;
  left = multExpr();
  while ( (op = lexer.token) == Symbol.PLUS ||
            op == Symbol.MINUS ) {
    lexer.nextToken();
    right = multExpr();
    left = new CompositeExpr( left, op, right );
  }
  return left;
}
```

3. (2.5) Complemente o seguinte analisador léxico para que ele :
a) elimine comentários que começam com # e terminam no final da linha. Não se esqueça que o último caráter do vetor `in` é '\0'. Assuma que '\0' é tratado no switch do código abaixo;
b) considere os caracteres '\t', '\n' e '\r' como espaços em branco;
c) incremente a variável de instância numLinhas a cada nova linha.

```
void nextToken() {
  while ( in[p] == ' ' )
     p++;
  if(Character.isDigit(in[p]))
        reconheceDigito();
  switch ( in[p] ) {
    case '+' : ...
  }
}
```

Copie o código acima na sua resposta.