Primeira Prova de Construção de Compiladores.
Primeiro Semestre de 2005, DC-UFSCar
Prof. José de Oliveira Guimarães
Turma C (Quinta-feira).

1.(5.0) Dada a gramática

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

Faça:
a) os métodos `program`, `varDecList` e `assign` 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. A classe compiler possui um método `nextToken` responsável pela análise léxica. O token corrente é colocado na variável de instância `token` de Compiler. As constantes da análise léxica estão na classe `Symbol`. 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`, `Assign` 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. (3.0) Faça :
a) (1.5) o desenho da ASA correspondente ao seguinte trecho de código da linguagem do compilador 9:

```
var a : integer;
begin
read(a);
a = a + 1;
write(a + 1);
end
```

b) (1.5) as conferências semânticas do comando write. Copie o trecho de código dado abaixo para a folha de respostas e adicione as conferências — assuma que expressões booleanas não possam ser escritas. Admita que os tipos são representados por objetos de IntegerType, BooleanType e CharType e que existe um único objeto de cada um dos tipos, referenciados pelas variáveis estáticas Type.integerType, Type.booleanType e Type.charType.

```
private WriteStatement writeStatement() {
   lexer.nextToken();
   if ( lexer.token != Symbol.LEFTPAR )
     error.signal("( expected");
   lexer.nextToken();
   Expr e = orExpr();
   if ( lexer.token != Symbol.RIGHTPAR )
     error.signal(") expected");
   lexer.nextToken();
   return new WriteStatement( e );
}
```

3. (1.5) Explique porque, nos compiladores 8 e 9 estudados em aula, utilizamos VariableExpr para representar variáveis que aparecem onde se espera uma expressão. Não poderíamos utilizar Variable, já que esta classe representa uma Variável ? Para auxiliar a sua resposta, recomendamos declarar a classe VariableExpr com as variáveis de instância e métodos (sem o corpo).

4. (2.5) Faça um analisador léxico para uma gramática que possui os seguintes terminais: "L", "R", "U", "D", "I" e qualquer dígito (0 a 9). Assuma que o analisador é um método nextToken de uma classe Compiler que possui variáveis de instância input (um vetor de char), tokenPos (a próxima posição de input a ser analisada) e token (o token corrente). O método nextToken, que é o que você deve fazer, deve saltar espaços em branco (`' '` e `'\n'` somente) e colocar em token uma constante da classe Symbol correspondente a cada um dos terminais:
public class Symbol { public final static int EOF = 0, L = 1, R = 2, U = 3, D = 4, I = 5, Number = 6;}
Você pode utilizar o método Character.isDigit(ch) para verificar se ch é dígito. Lembre-se de que input termina com '\0'.