Primeira Prova de Construção de Compiladores 1
Primeiro Semestre de 2008, DC-UFSCar
Prof. José de Oliveira Guimarães
Quarta 14h-18h

1. (3.0) Faça um analisador léxico para a seguinte gramática

```
E ::= T "+" E | T
T ::= T "*" F | F
F ::= "-" F | Numero | Letra
```

Onde "+", "*", "-", Numero e Letra são terminais. Numero é composto por qualquer quantidade de dígitos e Letra é uma única letra minúscula. O analisador léxico deve ser um método nextToken que coloca na variável token o número do token correspondente. Assuma que exista uma classe Symbol com uma constante para cada terminal (MAIS, MULT, MENOS, NUMERO, LETRA) mais EOF. Se o token for Symbol.NUMERO, na variável de instância valorNumero inteira deve ser colocado o valor do número. Se token for Symbol.LETRA, a variável de instância valorIdent do tipo char deve conter a letra do identificador (ex: valorIdent pode ser 'x' se a variável do programa for x). Veja o exemplo no quadro (se houver !). Não se esqueça de:

a) saltar espaços em branco (\r, \n, \t e ' ')
b) sinalizar EOF ao fim da entrada. Assuma que a entrada é dada em um vetor de char chamado input, indexado pela variável k e que termina com '\0';
c) valorNumero não pode ter mais do que cinco dígitos (0 a 99999).

As questões seguintes da prova utilizam a seguinte gramática:

```
Program ::= [ VarDec { VarDec } ] CompositeCommand
CompositeCommand ::= "{" Command  { Command } "}"
Command ::=  PrintCommand | AssignmentCommand
           | WhileCommand | IfCommand
           | CompositeCommand
VarDec ::= Type Id
Type ::= "integer" | "boolean"
AssignmentCommand ::= Id "=" OrExpr ";"
PrintCommand ::=  "print" OrExpr { OrExpr } ";"
WhileCommand ::= "while"  OrExpr "do"  Command
IfCommand ::= "if"  OrExpr "then"  Command
      [ "else" Command ]
```

As produções para expressões, OrExpr e outras, não serão necessárias para fazer esta prova. Para fazer as questões abaixo, é necessário saber que o comando print só imprime expressões inteiras.

Assuma que já existam todos os métodos de análise sintática que não são exigidos nesta prova. Por exemplo, assuma que exista um método

```
Expr orExpr()
```

que faz a análise sintática de uma expressão e retorne o objeto correspondente. Da mesma forma, existem métodos CommandList compositeCommand() e Variable varDec().

2. (7.0) Baseado nesta gramática, faça:
   a) as classes da ASA Command, Program, PrintCommand e Variable. Coloque as variáveis de instância e heranças (se houver). Não é necessário nenhum método;
   b) os métodos program, printCommand e varDec do analisador sintático com a inserção das variáveis na tabela de símbolos (quando necessário) e todas as conferências semânticas. Naturalmente, construa a ASA durante esta análise.

Os métodos do item b) estão na classe Compiler que possui um método nextToken responsável pela análise léxica. O token corrente é colocado na variável de instância token de Compiler, do tipo int. As constantes da análise léxica estão na classe Symbol (IDENT para Id, VAR, FOR, etc). 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 (que não precisam ser significativas --- chame error sem parâmetros). Se o token corrente for Symbol.IDENT, a variável de instância stringValue de Compiler guarda o identificador (uma string). Admita que exista uma classe abstrata Expr superclasse de todas as classes de expressões. Esta classe possui um método Type getType() que retorna o tipo da expressão. A classe Type é superclasse das classes IntegerType e BooleanType. Há um único objeto de cada uma delas e este objeto representa o tipo correspondente através de toda a compilação. Cada um destes objetos pode ser manipulado fazendo-se "Type.integerType" e "Type.booleanType", onde integerType e booleanType são variáveis estáticas da classe Type. O ";" é representado por Symbol.SEMICOLON. Qualquer outra dúvida a respeito de métodos/classes do compilador que você precise poderá ser esclarecida pelo professor.

3. (2.0) Faça o método genC da classe PrintCommand. O código deve ser gerado em C. Utilize

```
printf("%d\n", expr);
```

para imprimir uma expressão inteira expr. Valores booleanos não podem ser impressos.