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

1. (5.0) Acrescente à linguagem do compilador 10 um comando "declaração de tipo":

TypeDec ::= "type" IDENT "=" Type ";"

Como exemplo,

```
type Color = integer;
```

cria o tipo Color que é sinônimo de integer. Uma variável pode ter o tipo Color. A regra para Type é

Type ::= "integer" | "boolean" | "char"

Naturalmente, integer, boolean e char são terminais.

Faça o método typeDec da classe Compiler que faz a análise sintática e semântica desta produção. Assuma que, quando este método for chamado, o token corrente é `Symbol.TYPE`. Assuma a classe Compiler possua um método nextToken. Este método coloca o token encontrado na análise léxica na variável de instância token da classe Compiler. Tudo igual aos exemplos que vimos. Não é necessário declarar as constantes correspondentes aos terminais.

O método `typeDec` deve retornar o objeto da ASA correspondente à produção `TypeDec`. Naturalmente, faça a classe da ASA. Não se esqueça do construtor. Adicione um método gen() que gera na saída padrão código igual ao da entrada.

A análise semântica deve conferir se o nome do novo tipo é diferente dos outros tipos criados com a regra "TypeDec". Os tipos criados pelo usuário não devem ser colocados na tabela de símbolos.

Se `token == IDENT`, então assuma que na variável de instância stringValue de Compiler é colocada a string correspondente ao identificador.

Assume que exista uma classe Type com subclasses IntegerType, BooleanType e CharType. As variáveis estáticas integerType, booleanType e charType de Type apontam para o único objeto de cada uma destas subclasses. Há um método chamado type() que analisa se o token corrente é um dos tipos básicos retornando o objeto correspondente a aquele tipo (como nos compiladores vistos em aula).

Você pode utilizar as classes Hashtable, Vector e Enumeration.

2. (2.0) Faça analisadores sintáticos para as gramáticas abaixo. Utilize um método para cada não terminal. Assuma que há métodos já prontos para os não terminais cujas regras não são mostradas (como D e E). Assuma que o método nextToken() do analisador léxico já está pronto. Não crie a ASA. Teste pela presença de um terminal usando algo como

```
if ( token == 'a' ) ...
```

a) P ::= A [C] | B
A ::= 'a' D
B ::= { 'b' E }
C ::= 'c'

b) A ::= B { C }
B ::= 'b' | 'c' B
C ::= 'e' D

3. (1.5) Explique como funciona a tabela de símbolos. Utilize trechos de código para exemplificar o seu raciocínio, mostrando quando objetos (e que objetos) são inseridos, quando é feita uma busca e quando elementos são retirados da tabela.

4. (1.5) Faça um trecho de código em Java que monta a ASA para a variável months declarada como:

```
var months : array[1..12] of integer;
```

months é um vetor de inteiros cujos índices inferior e superior são 1 e 12, respectivamente. Não é preciso fazer nenhuma classe da ASA, apenas faça o código que monta a ASA.