Primeira Prova de Construção de Compiladores
Primeiro Semestre de 2003, DC-UFSCar
Prof. José de Oliveira Guimarães
Turma B (Terça feira).

1. (5.0) Tipos enumerados de uma linguagem fictícia são formados pela seguinte produção:

```
EnumType ::= "enum" "(" ID { "," ID } ")"
```

Faça um método `EnumType enumType()` que reconheça esta produção e o correspondente analisador semântico. Assuma que quando este método for chamado o token corrente é `Symbol.ENUM`. A função enumType deve estar dentro de uma classe. Assuma que o analisador léxico esteja dentro desta classe também, como um método. Este método coloca o token encontrado na análise léxica na variável de instância token da classe que você fazer. Tudo igual aos exemplos que vimos. Não é necessário declarar as constantes correspondentes aos terminais.

O método `enumType` deve retornar o objeto da ASA correspondente à produção `EnumType`. Naturalmente, faça a classe da ASA, que deve herdar de `Type`. 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 uma constante do enumerado já não foi declarada no mesmo tipo. Ex:

```
type color = enum( blue, red, green); // ok
type cor = enum ( azul,
                  red, // ok, tipos diferentes
                  azul ); // erro: repetiu azul
```

Se `token == IDENT`, então assuma que na variável de instância stringValue do analisador é colocada a string correspondente à palavra.

Você pode utilizar as classes Hashtable, Vector e Enumeration.

2. (2.0) Faça o analisador sintático para a gramática abaixo.

P ::= A | B
A ::= 'a' C | 'b' D
B ::= qualquer dígito entre 0 e 9
C ::= B { B }
D ::= B ['c'] B

Importante: utilize um método para cada não terminal. Assuma que o analisador léxico já está pronto. Não crie a ASA. Teste pela presença de um terminal usando algo como

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

3. (2.0) Crie uma classe SymbolTable para ser utilizada em um compilador de Java. Não é necessário fazer o corpo dos métodos. Naturalmente, coloque nomes dos métodos e das variáveis auto-explicativos. Assuma que todas as variáveis declaradas dentro de um método de Java estão no mesmo escopo.

4. (1.0) O programa C++ abaixo não funciona corretamente. Um compilador ***realmente*** esperto poderia avisar um problema. Qual ? Este programa deveria imprimir 26*25, 25*24, ... 6, 2.

```
#include <iostream.h>

int a, b, c, d, ...,  w, x, y, z;
int *V[] = { &a, &b, &c, &d, &e,
  ... &x, &y, &z };

int mult(int *x, int *y) {
  return *x * *y;
}

void main() {
   z = 26, y = 25, x = 24,
   ... c = 3, b = 2, a = 1;
   int ind = sizeof(V)/
             sizeof(int*);
   ind--;
   while ( ind > 0 ) {
      cout << mult(V[ind],
         V[ind-1]) << endl;
      ind--;
   }
}
```

5. (1.5) Faça um desenho representando a ASA do seguinte trecho de código da linguagem do compilador 10.

```
if n > 0
then write( calc(n, 0) );
else read(n);
```

Coloque os nomes das classes da ASA ao lado dos objetos. Assuma que write só escreve *uma* expressão, que read só lê *uma* variável e que calc toma dois inteiros como parâmetros. Represente a *função* "calc" apenas pela string "calc".