- Implementar um parser para a linguagem Lugosi usando o Javacc
- Usar tradução dirigida por sintaxe para construir a árvore sintática dos programas. Você deve pensar em um conjunto de classes que sirva para representar a árvore sintática de um programa Lugosi
- Implementar um pretty printer para a linguagem, ou seja, um método que recebe a árvore sintática e a partir dela reconstrói o programa original
- Desenvolver três programas escritos na linguagem Lugosi para testar o parser
- EXTRA: Fazer um gerardor de código para a linguagem Lugosi, ou seja, um método que recebe a árvore sintática e gera um arquivo contendo um código com a mesma semântica do programa original em Lugosi mas escrito em outra linguagem, e.g., C, Java, Python, etc..
- O `main` do Javacc deve ficar mais ou menos assim:

```
public class Lugosi {

  public static void main(String args[]) throws Exception{
   // abrir o arquivo passado por linha
   // de comando contento o código em Lugosi:

  FileInputStream fs = new FileInputStream(new File(args[0]));

   // Instanciar o parser da linguagem Lugosi passando
   // como argumento o arquivo contendo o código
   //Lugosi a ser processado:

    Lugosi parser = new Lugosi(fs);

   // Chamar a primeira regra do parser que irá
   // analisar o código e devolver a árvore sintática

    ArvoreLugosi arvore =parser.Lugosi();

  // Passar a árvore para o prety printer:

    pprint(arvore);
  // passar a árvore para o gerador de código

   geraCodigo(arvore)

  }

 public static void pprint(ArvoreLugosi prog){??????}
 public static void geraCodigo(ArvoreLugosi prog){??????}


}
```

- Linguagme Lugosi, definições léxicas e sintáticas:

Linguagem Lugosi
^^^^^^^^^ ^^^^^^

20/12/2017

```
LUGOSI -> MAIN FUNC*

MAIN ->  "main" "{" VARDECL SEQCOMANDOS "}"

VARDECL -> VARDECL "var" TIPO TOKEN_id ";" | vazio

TIPO -> "int" | "bool"

SEQCOMANDOS -> SEQCOMANDOS COMANDO | vazio

COMANDO ->  TOKEN_id ":=" EXP ";"
   | TOKEN_id "(" LISTAEXP? ")" ";"
   | "if" "(" EXP ")" "{" SEQCOMANDOS "}"  ";"
   | "while" "(" EXP ")" "do" "{" SEQCOMANDOS "}"  ";"
   | "do" "{" SEQCOMANDOS "}" "while" "(" EXP ")"  ";"
   | "return" EXP ";"
   | "print" "(" EXP ")"  ";"


EXP -> "(" EXP OP EXP ")" | FATOR

FATOR -> TOKEN_id | TOKEN_id "(" LISTAEXP? ")"
         | TOKEN_numliteral |   "true" | "false"

OP -> "+" | "-" | "*" | "/" | "&&" | "||" | "<" | ">" | "=="

LISTAEXP -> EXP | LISTAEXP "," EXP


FUNC -> "function" TIPO TOKEN_id "(" LISTAARG? ")" "{" VARDECL SEQCOMANDOS "}"

LISTAARG -> TIPO TOKEN_id | LISTAARG "," TIPO TOKEN_id
```

==========

Convenções léxicas
^^^^^^^^^^ ^^^^^^^

```
TOKEN_id -> letra letraoudigito* finalsublinhado*
TOKEN_numliteral -> digitos facao_opcional expoente_opcional


onde:
letra -> [a-zA-Z]
digito -> [0-9]
digitos -> digito+
facao_opcional -> (.digitos)?
```

```
expoente_opcional -> (E (+ | -)? digitos)?
letraoudigito -> letra | digito
finalsublinhado -> _letraoudigito+
letra -> [a-zA-Z]
digito -> [0-9]
```