- Implementar um parser para a linguagem Karloff usando o Javacc (Esse trabalho já foi feito pela maioria dos alunos)
- 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 Karloff
- Implementar um pretty printer para a linguagem, ou seja, um método que recebe a árvore sintática e a partir dela reconstrói o programa original
- Alternativamente, o aluno pode, ao invés de gerar novamente o código Karloff, gerar código equivalente em uma outra linguagem de programação
- O `main` do Javacc deve ficar mais ou menos assim:

```
public class Karloff {

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

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

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

    Karloff parser = new Karloff(fs);

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

    ArvoreKarloff arvore =parser.Karloff();

  // Passar a árvore para o pretty printer, ou gerador de código

    pprint(arvore);

  }

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


}
```

- Linguagme Karloff, definições léxicas e sintáticas:

```
Linguagem Karloff
^^^^^^^^^ ^^^^^^


KARLOFF -> MAIN FUNC?

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

```
VARDECL -> VARDECL "newVar" TIPO TOKEN_id ";" | vazio

TIPO -> "int" | "bool"

SEQCOMANDOS -> SEQCOMANDOS COMANDO | vazio

COMANDO ->  TOKEN_id "=" EXP ";"
   | TOKEN_id "(" LISTAEXP? ")" ";"
   | "if" "(" EXP ")" "then" "{" SEQCOMANDOS "}"  ";"
   | "while" "(" EXP ")"  "{" SEQCOMANDOS "}"  ";"
   | "repeat" "{" SEQCOMANDOS "}" "until" "(" EXP ")"  ";"
   | "return" EXP ";"
   | "System.output" "(" EXP ")"  ";"


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

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

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

LISTAEXP -> EXP | LISTAEXP "," EXP


FUNC -> FUNC "func" TIPO TOKEN_id "(" LISTAARG? ")" "{" VARDECL SEQCOMANDOS "}"
       | "func" 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]
```