Projet : Vers un mini compilateur d'un mini JAVA (MiniJAVA)

Cadre générale

Le but de ce projet est de construire un mini compilateur java **minijava**.
L'implémentation de ce compilateur sera réalisé avec le langage C.
La grammaire de **MiniJAVA** est décrite sous forme BNF comme suit, l'axiome de départ est `Program`.

BNF de MiniJAVA

```
Program ::= MainClass ( ClassDeclaration )* <EOF>

MainClass ::= "class" Identifier "{" "public" "static" "void" "main" "(" "String" "["
              "]" Identifier ")" "{" Statement "}" "}"
ClassDeclaration ::= "class" Identifier ( "extends" Identifier )? "{" ( VarDeclaration )*
              ( MethodDeclaration )* "}"
VarDeclaration ::= Type Identifier ";"

MethodDeclaration ::= "public" Type Identifier "(" ( Type Identifier ( "," Type Identifier )* )? ")"
              "{" ( VarDeclaration )* ( Statement )* "return" Expression ";" "}"
Type ::= "int" "[" "]"

       | "boolean"
       | "int"
       | Identifier
Statement ::= "{" ( Statement )* "}"

       | "if" "(" Expression ")" Statement "else" Statement
       | "while" "(" Expression ")" Statement
       | "System.out.println" "(" Expression ")" ";"
       | Identifier "=" Expression ";"
       | Identifier "[" Expression "]" "=" Expression ";"
Expression ::= Expression ( "&&" | "<" | "+" | "-" | "*" ) Expression

       | Expression "[" Expression "]"
       | Expression "." "length"
       | Expression "." Identifier "(" ( Expression ( "," Expression )* )? ")"
       | <INTEGER_LITERAL>
       | <BOOLEAN_LITERAL>

       | Identifier
       | "this"
       | "new" "int" "[" Expression "]"
       | "new" Identifier "(" ")"
```

| "!" Expression
| "(" Expression ")"
Identifier ::= <IDENTIFIER>

CONVENTIONS LEXICALES

1. Les éléments ::=, |, () sont les symboles du métalangage :

   - ::= est le symbole de réecriture. Il défi
   - `()` indique quelque chose d'optionnel; ex: `S -> A ( B C )` est l'équivalent de:

     - `S -> A`
     - `S -> A B C`

   - `|` indique le choix

2. Les commentaires sont entourés par /* et */ pour commenter un bloc ou bien par // pour commenter une ligne. Un commentaire peut apparaître après une unité lexicale quelconque

3. Le lexème de l'unité lexicale affectation est =.
4. Les éléments en caractères gras sont les mots-clés réservés du langage (que votre analyseur lexical aura à charge de reconnaître).
5. Les éléments en rouge indiquent des unités (tokens) reconnues par l'analyseur lexical que vous devez écrire :

   - `Identifier` est un nom d'au moins un caractère qui commence par une lettre et qui est suivi d'un nombre quelconque (éventuellement nul) de lettre ou de chiffres.
   - `<INTEGER_LITERAL>` désigne une suite (non vide) de chiffres
   - `<BOOLEAN_LITERAL>` désigne l'une des deux valeurs TRUE ou FALSE

```
identifier        = ? /([A-Za-z_][A-Za-z0-9_]*)/ ? ;
<INTEGER_LITERAL> = ? /(-?[1-9][0-9]*)/ ? ;
<BOOLEAN_LITERAL> = ? /(true|false)/ ? ;
```

6. Enfin une règle de la forme: `A ::=`

QUESTIONS

1- Développer un analyseur lexical.
2- Mettre la grammaire sous forme LL(1) et développer un analyseur syntaxique descendant
3- Intégrer des traitements d'erreurs lexicales et syntaxiques
4- Intégrer des contrôles sémantiques

5- Générer un code intermédiaire pour une machine à 3 adresses.
6- Produire le code pour une machine cible.
7- Réaliser une interface graphique pour permettre à l'utilisateur d'écrire un code java et le compiler

Pour chaque question, il est demandé de rendre un rapport expliquant la démarche adoptée, qui sera évalué ainsi que la réalisation.