```java
package Parser;

import Parser.Lexer.*;
import java.io.File;
import java.util.LinkedList;
import java.util.List;

/**
 * Created by rubenspessoa on 18/10/16.
 */
public class Parser {

    private LinkedList<Token> tokens;
    private Token lookahead;
    private LinkedList<String> output = new LinkedList<String>();

    public LinkedList<String> parse(File file) throws Exception {
        Lexer lexer = Lexer.getLexer();
        lexer.lex(file);
        parse(lexer.getTokens());
        return this.output;
    }

    private void parse(List<Token> tokens) throws ParserException {
        this.tokens = (LinkedList<Token>) tokens;
        this.lookahead = this.tokens.getFirst();

        // first production
        program();

        if (lookahead.getTokenCategory() != Token.TokenCategory.EOF) {
            throw new ParserException("Unexpected symbol " + lookahead + "
                found!");
        }

        for (String prod : output) {
            System.out.println(prod);
        }
    }

    private void program() throws ParserException {
        if (this.lookahead.getTokenCategory() == Token.TokenCategory.PR_VOID
            ) {
            output.add("<program> ::= " + this.lookahead.getSequence() + "
                <program_aux>");
            this.nextToken();
            this.program_aux();
        } else if (this.lookahead.getTokenCategory() == Token.TokenCategory.
            TYPE_VALUE) {
            output.add("<program> ::=" + lookahead.getSequence() + "
                <function_declaration> <program>");
            this.nextToken();
            this.function_declaration();
            this.program();
        } else {
            output.add("<program> ::= empty");
        }
    }

    private void program_aux() throws ParserException {
        if (this.lookahead.getTokenCategory() == Token.TokenCategory.PR_MAIN
            ) {
            String prod = "<program_aux> ::= " + lookahead.getSequence() + "
                AB_PAR FEC_PAR <scope> ";
            this.nextToken();
            if (this.lookahead.getTokenCategory() == Token.TokenCategory.
                AB_PAR) {
                this.nextToken();
                if (this.lookahead.getTokenCategory() == Token.TokenCategory
                    .FEC_PAR) {
                    output.add(prod);
                    this.nextToken();
                    this.scope();
                } else {
                    throw new ParserException("Unexpected symbol " +
                        lookahead + " found!");
                }
            } else {
                throw new ParserException("Unexpected symbol " + lookahead +
                    " found!");
            }
        } else {
            output.add("<program_aux> ::= <function_declaration> <program>")
                ;
            this.function_declaration();
            this.program();
        }
    }

    private void scope() throws ParserException {
        if (lookahead.getTokenCategory() == Token.TokenCategory.AB_CH) {
            String prod = "<scope> ::= " + lookahead.getSequence() + "
                <commands> FEC_CH SP";
            nextToken();
            this.commands();
            if (lookahead.getTokenCategory() == Token.TokenCategory.FEC_CH)
                {
                nextToken();
                if (lookahead.getTokenCategory() == Token.TokenCategory.SP)
                    {
                    output.add(prod);
                    nextToken();
                }  else {
                    throw new ParserException("Unexpected symbol " +
                        lookahead + " found!");
                }
            } else {
                throw new ParserException("Unexpected symbol " + lookahead +
                    " found!");
            }
        } else {
            throw new ParserException("Unexpected symbol " + lookahead + "
                found!");
        }
    }

    private void function_declaration() throws ParserException {
        if (lookahead.getTokenCategory() == Token.TokenCategory.ID) {
            nextToken();
            if (lookahead.getTokenCategory() == Token.TokenCategory.AB_PAR)
```

```
                {
                nextToken();
                this.parameters();
                if (lookahead.getTokenCategory() == Token.TokenCategory.
                    FEC_PAR) {
                    output.add("<function_declaration> ::= ID AB_PAR
                        <parameters> FEC_PAR <scope>");
                    nextToken();
                    this.scope();
                } else {
                    throw new ParserException("Unexpected symbol " +
                        lookahead + " found!");
                }
            } else {
                throw new ParserException("Unexpected symbol " + lookahead +
                    " found!");
            }
        } else {
            throw new ParserException("Unexpected symbol " + lookahead + "
                found!");
        }
    }

    private void commands() throws ParserException {
        if (lookahead.getTokenCategory() == Token.TokenCategory.TYPE_VALUE)
            {
            String prod = "<commands> ::= " + lookahead.getSequence() + " ID
                <declaration> <commands>";
            nextToken();
            if (lookahead.getTokenCategory() == Token.TokenCategory.ID) {
                output.add(prod);
                nextToken();
                this.declaration();
                this.commands();
            } else {
                throw new ParserException("Unexpected symbol " + lookahead +
                    " found!");
            }
        } else if (lookahead.getTokenCategory() == Token.TokenCategory.ID) {
            String prod = "<commands> ::= " + lookahead.getSequence() + "
                <attribution_or_function_call> <commands>";
            output.add(prod);
            nextToken();
            this.attribution_or_function_call();
            this.commands();
        } else if (lookahead.getTokenCategory() == Token.TokenCategory.PR_IO
            ) {
            String prod = "<commands> ::= " + lookahead.getSequence() + "
                AB_PAR <printout_or_readin> <commands>";
            nextToken();
            if (lookahead.getTokenCategory() == Token.TokenCategory.AB_PAR)
                {
                output.add(prod);
                nextToken();
                this.printout_or_readin();
                this.commands();
            }
        } else if (lookahead.getTokenCategory() == Token.TokenCategory.PR_IF
            ) {
            output.add("<commands> ::= PR_IF <ifelse> <commands>");
            nextToken();
            this.ifelse();
            this.commands();
        } else if (lookahead.getTokenCategory() == Token.TokenCategory.
            PR_WHILE) {
            output.add("<commands> ::= PR_WHILE <while> <commands>");
            nextToken();
            this.while_prod();
            this.commands();
        } else if (lookahead.getTokenCategory() == Token.TokenCategory.
            PR_FOR) {
            output.add("<commands> ::= PR_FOR <for> <commands>");
            nextToken();
            this.for_prod();
            this.commands();
        } else if (lookahead.getTokenCategory() == Token.TokenCategory.
            PR_SHOOT) {
            nextToken();
            this.shoot();
            if (lookahead.getTokenCategory() == Token.TokenCategory.SP) {
                output.add("<commands> ::= PR_SHOOT <shoot> SP");
                nextToken();
            } else {
                throw new ParserException("Unexpected symbol " + lookahead +
                    " found!");
            }
        } else {
            output.add("<commands> ::= empty");
        }
    }

    private void while_prod() throws ParserException {
        if (lookahead.getTokenCategory() == Token.TokenCategory.AB_PAR) {
            nextToken();
            this.expression();
            if (lookahead.getTokenCategory() == Token.TokenCategory.FEC_PAR)
                {
                output.add("<while> ::= AB_PAR <expression> FEC_PAR <scope>"
                    );
                nextToken();
                this.scope();
            } else {
                throw new ParserException("Unexpected symbol " + lookahead +
                    " found!");
            }
        } else {
            throw new ParserException("Unexpected symbol " + lookahead + "
                found!");
        }
    }

    private void for_prod() throws ParserException {
        if (lookahead.getTokenCategory() == Token.TokenCategory.AB_PAR) {
            nextToken();
            this.for_steps();
            if (lookahead.getTokenCategory() == Token.TokenCategory.FEC_PAR)
                {
                output.add("<for> ::= AB_PAR <for_steps> FEC_PAR <scope>");
                nextToken();
                this.scope();
            } else {
                throw new ParserException("Unexpected symbol " + lookahead +
                    " found!");
            }
        } else {
            throw new ParserException("Unexpected symbol " + lookahead + "
                found!");
        }
    }

    private void for_steps() throws ParserException {

        if (lookahead.getTokenCategory() == Token.TokenCategory.TYPE_VALUE)
            {
            nextToken();
            if (lookahead.getTokenCategory() == Token.TokenCategory.ID) {
                nextToken();
                if (lookahead.getTokenCategory() == Token.TokenCategory.
                    OP_ATR) {
                    nextToken();
                    if (lookahead.getTokenCategory() == Token.TokenCategory.
                        CTE_INT) {
                        nextToken();
                        if (lookahead.getTokenCategory() == Token.
                            TokenCategory.SP) {
                            nextToken();
                            if (lookahead.getTokenCategory() == Token.
                                TokenCategory.ID) {
                                nextToken();
                                if (lookahead.getTokenCategory() == Token.
                                    TokenCategory.OP_REL1) {
                                    nextToken();
                                    if (lookahead.getTokenCategory() ==
                                        Token.TokenCategory.CTE_INT) {
                                        nextToken();
                                        if (lookahead.getTokenCategory() ==
                                            Token.TokenCategory.SP) {
                                            nextToken();
                                            if (lookahead.getTokenCategory()
                                                == Token.TokenCategory.ID) {
                                                nextToken();
                                                if (lookahead.
                                                    getTokenCategory() ==
                                                    Token.TokenCategory.
                                                    OP_ATR) {
                                                    nextToken();
                                                    if (lookahead.
                                                    getTokenCategory() ==
                                                    Token.TokenCategory.ID)
                                                    {
                                                        nextToken();
                                                        if (lookahead.
                                                    getTokenCategory() ==
                                                    Token.TokenCategory.
                                                    OP_AD) {
                                                            nextToken();
                                                            if (lookahead.
                                                    getTokenCategory() ==
                                                    Token.TokenCategory.
                                                    CTE_INT) {
                                                    output.add("<for_steps>
                                                    ::= TYPE_VALUE ID OP_ATR
                                                    CTE_INT SP ID OP_REL1
                                                    CTE_INT SP ID OP_ATR ID
                                                    OP_AD CTE_INT");
                                                                nextToken();
                                                            } else {
                                                                throw new
                                                    ParserException("Unexpec
                                                    ted symbol " + lookahead
                                                    + " found!");
                                                            }
                                                        } else {
                                                            throw new
                                                    ParserException("Unexpec
                                                    ted symbol " + lookahead
                                                    + " found!");
                                                        }
                                                    } else {
                                                        throw new
                                                    ParserException("Unexpec
                                                    ted symbol " + lookahead
                                                    + " found!");
                                                    }
                                                } else {
                                                    throw new
                                                    ParserException("Unexpec
                                                    ted symbol " + lookahead
                                                    + " found!");
                                                }
                                            } else {
                                                throw new
                                                    ParserException("Unexpec
                                                    ted symbol " + lookahead
                                                    + " found!");
                                            }
                                        } else {
                                            throw new
                                                ParserException("Unexpected
                                                symbol " + lookahead + "
                                                found!");
                                        }
                                    } else if (lookahead.getTokenCategory()
                                        == Token.TokenCategory.ID) {
                                        nextToken();
                                        if (lookahead.getTokenCategory() ==
                                            Token.TokenCategory.SP) {
                                            nextToken();
                                            if (lookahead.getTokenCategory()
                                                == Token.TokenCategory.ID) {
                                                nextToken();
                                                if (lookahead.
                                                    getTokenCategory() ==
                                                    Token.TokenCategory.
                                                    OP_ATR) {
                                                    nextToken();
                                                    if (lookahead.
                                                    getTokenCategory() ==
                                                    Token.TokenCategory.ID)
```

```
                                {
                                    nextToken();
                                    if (lookahead.
                                getTokenCategory() ==
                                Token.TokenCategory.
                                OP_AD) {
                                        nextToken();
                                        if (lookahead.
                                getTokenCategory() ==
                                Token.TokenCategory.
                                CTE_INT) {

                                output.add("<for_steps>
                                ::= TYPE_VALUE ID OP_ATR
                                CTE_INT SP ID OP_REL1 ID
                                SP ID OP_ATR ID OP_AD
                                CTE_INT");
                                            nextToken();
                                        } else {
                                            throw new
                                ParserException("Unexpec
                                ted symbol " + lookahead
                                + " found!");
                                        }
                                    } else {
                                        throw new
                                ParserException("Unexpec
                                ted symbol " + lookahead
                                + " found!");
                                    }
                                } else {
                                    throw new
                                ParserException("Unexpec
                                ted symbol " + lookahead
                                + " found!");
                                }
                            } else {
                                throw new
                                ParserException("Unexpec
                                ted symbol " + lookahead
                                + " found!");
                            }
                        } else {
                            throw new
                                ParserException("Unexpec
                                ted symbol " + lookahead
                                + " found!");
                        }
                    } else {
                        throw new
                            ParserException("Unexpected
                            symbol " + lookahead + "
                            found!");
                    }
                } else {
                    throw new
                        ParserException("Unexpected
                        symbol " + lookahead + " found!"
                        );
                }
```

```
                                } else {
                                    throw new ParserException("Unexpected
                                        symbol " + lookahead + " found!");
                                }
                            } else {
                                throw new ParserException("Unexpected symbol
                                    " + lookahead + " found!");
                            }
                        } else {
                            throw new ParserException("Unexpected symbol " +
                                lookahead + " found!");
                        }
                    } else if (lookahead.getTokenCategory() == Token.
                        TokenCategory.ID) {
                        nextToken();
                        if (lookahead.getTokenCategory() == Token.
                            TokenCategory.SP) {
                            nextToken();
                            if (lookahead.getTokenCategory() == Token.
                                TokenCategory.ID) {
                                nextToken();
                                if (lookahead.getTokenCategory() == Token.
                                    TokenCategory.OP_REL1) {
                                    nextToken();
                                    if (lookahead.getTokenCategory() ==
                                        Token.TokenCategory.CTE_INT) {
                                        nextToken();
                                        if (lookahead.getTokenCategory() ==
                                            Token.TokenCategory.SP) {
                                            nextToken();
                                            if (lookahead.getTokenCategory()
                                                == Token.TokenCategory.ID) {
                                                nextToken();
                                                if (lookahead.
                                                    getTokenCategory() ==
                                                    Token.TokenCategory.
                                                    OP_ATR) {
                                                    nextToken();
                                                    if (lookahead.
                                                    getTokenCategory() ==
                                                    Token.TokenCategory.ID)
                                                    {
                                                        nextToken();
                                                        if (lookahead.
                                                    getTokenCategory() ==
                                                    Token.TokenCategory.
                                                    OP_AD) {
                                                            nextToken();
                                                            if (lookahead.
                                                    getTokenCategory() ==
                                                    Token.TokenCategory.
                                                    CTE_INT) {

                                                    output.add("<for_steps>
                                                    ::= TYPE_VALUE ID OP_ATR
                                                    CTE_INT SP ID OP_REL1
                                                    CTE_INT SP ID OP_ATR ID
                                                    OP_AD CTE_INT");
                                                                nextToken();
                                                            } else {
```

```
                                                            throw new
                                                  ParserException("Unexpec
                                                  ted symbol " + lookahead
                                                  + " found!");
                                                        }
                                                     } else {
                                                        throw new
                                                  ParserException("Unexpec
                                                  ted symbol " + lookahead
                                                  + " found!");
                                                     }
                                                  } else {
                                                     throw new
                                                  ParserException("Unexpec
                                                  ted symbol " + lookahead
                                                  + " found!");
                                                  }
                                               } else {
                                                  throw new
                                                  ParserException("Unexpec
                                                  ted symbol " + lookahead
                                                  + " found!");
                                               }
                                            } else {
                                               throw new
                                                  ParserException("Unexpec
                                                  ted symbol " + lookahead
                                                  + " found!");
                                            }
                                         } else {
                                            throw new
                                               ParserException("Unexpected
                                               symbol " + lookahead + "
                                               found!");
                                         }
                                      } else if (lookahead.getTokenCategory()
                                         == Token.TokenCategory.ID) {
                                         nextToken();
                                         if (lookahead.getTokenCategory() ==
                                            Token.TokenCategory.SP) {
                                            nextToken();
                                            if (lookahead.getTokenCategory()
                                               == Token.TokenCategory.ID) {
                                               nextToken();
                                               if (lookahead.
                                                  getTokenCategory() ==
                                                  Token.TokenCategory.
                                                  OP_ATR) {
                                                  nextToken();
                                                  if (lookahead.
                                                  getTokenCategory() ==
                                                  Token.TokenCategory.ID)
                                                  {
                                                     nextToken();
                                                     if (lookahead.
                                                  getTokenCategory() ==
                                                  Token.TokenCategory.
                                                  OP_AD) {
                                                        nextToken();
                                                        if (lookahead.
                                                  getTokenCategory() ==
                                                  Token.TokenCategory.
                                                  CTE_INT) {

                                                  output.add("<for_steps>
                                                  ::= TYPE_VALUE ID OP_ATR
                                                  CTE_INT SP ID OP_REL1 ID
                                                  SP ID OP_ATR ID OP_AD
                                                  CTE_INT");
                                                        nextToken();
                                                        } else {
                                                        throw new
                                                  ParserException("Unexpec
                                                  ted symbol " + lookahead
                                                  + " found!");
                                                        }
                                                     } else {
                                                        throw new
                                                  ParserException("Unexpec
                                                  ted symbol " + lookahead
                                                  + " found!");
                                                     }
                                                  } else {
                                                     throw new
                                                  ParserException("Unexpec
                                                  ted symbol " + lookahead
                                                  + " found!");
                                                  }
                                               } else {
                                                  throw new
                                                  ParserException("Unexpec
                                                  ted symbol " + lookahead
                                                  + " found!");
                                               }
                                            } else {
                                               throw new
                                                  ParserException("Unexpec
                                                  ted symbol " + lookahead
                                                  + " found!");
                                            }
                                         } else {
                                            throw new
                                               ParserException("Unexpected
                                               symbol " + lookahead + "
                                               found!");
                                         }
                                      } else {
                                         throw new
                                            ParserException("Unexpected
                                            symbol " + lookahead + " found!"
                                            );
                                      }
                                    } else {
                                      throw new ParserException("Unexpected
                                         symbol " + lookahead + " found!");
                                    }
                                  } else {
                                    throw new ParserException("Unexpected symbol
                                      " + lookahead + " found!");
                                  }
```

```
                    } else {
                        throw new ParserException("Unexpected symbol " +
                            lookahead + " found!");
                    }
                } else {
                    throw new ParserException("Unexpected symbol " +
                        lookahead + " found!");
                }
            } else {
                throw new ParserException("Unexpected symbol " +
                    lookahead + " found!");
            }
        } else {
            throw new ParserException("Unexpected symbol " + lookahead +
                " found!");
        }
    } else {
        throw new ParserException("Unexpected symbol " + lookahead + "
            found!");
    }
}

private void shoot() throws ParserException {
    if (lookahead.getTokenCategory() == Token.TokenCategory.ID) {
        output.add("<shoot> ::= ID");
        nextToken();
    } else if (lookahead.getTokenCategory() == Token.TokenCategory.
        CTE_INT) {
        output.add("<shoot> ::= CTE_INT");
        nextToken();
    } else if (lookahead.getTokenCategory() == Token.TokenCategory.
        CTE_STR) {
        output.add("<shoot> ::= CTE_STR");
        nextToken();
    } else if (lookahead.getTokenCategory() == Token.TokenCategory.
        CTE_FLOAT) {
        output.add("<shoot> ::= CTE_FLOAT");
        nextToken();
    } else if (lookahead.getTokenCategory() == Token.TokenCategory.
        BOOL_VALUE) {
        output.add("<shoot> ::= BOOL_VALUE");
        nextToken();
    } else {
        throw new ParserException("Unexpected symbol " + lookahead + "
            found!");
    }
}

private void ifelse() throws ParserException {
    if (lookahead.getTokenCategory() == Token.TokenCategory.AB_PAR) {
        nextToken();
        this.expression();
        if (lookahead.getTokenCategory() == Token.TokenCategory.FEC_PAR)
            {
            nextToken();
            if (lookahead.getTokenCategory() == Token.TokenCategory.
                AB_CH) {
                nextToken();
                this.commands();
                if (lookahead.getTokenCategory() == Token.TokenCategory.
                    FEC_CH) {
                    nextToken();
                    this.else_prod();
                    if (lookahead.getTokenCategory() == Token.
                        TokenCategory.SP) {
                        output.add("<ifelse> ::= AB_AR <expression>
                            FEC_PAR AB_CH <commands> FEC_CH <else> SP");
                        nextToken();
                    } else {
                        throw new ParserException("Unexpected symbol " +
                            lookahead + " found!");
                    }
                } else {
                    throw new ParserException("Unexpected symbol " +
                        lookahead + " found!");
                }
            } else {
                throw new ParserException("Unexpected symbol " +
                    lookahead + " found!");
            }
        } else {
            throw new ParserException("Unexpected symbol " + lookahead +
                " found!");
        }
    } else {
        throw new ParserException("Unexpected symbol " + lookahead + "
            found!");
    }
}

private void else_prod() throws ParserException {
    if (lookahead.getTokenCategory() == Token.TokenCategory.PR_ELSE) {
        nextToken();
        if (lookahead.getTokenCategory() == Token.TokenCategory.AB_CH) {
            nextToken();
            this.commands();
            if (lookahead.getTokenCategory() == Token.TokenCategory.
                FEC_CH) {
                output.add("<else> ::= PR_ELSE AB_CH <commands> FEC_CH")
                    ;
                nextToken();
            } else {
                throw new ParserException("Unexpected symbol " +
                    lookahead + " found!");
            }
        } else {
            throw new ParserException("Unexpected symbol " + lookahead +
                " found!");
        }
    } else {
        output.add("<else> ::= empty");
    }
}

private void printout_or_readin() throws ParserException {
    if (lookahead.getTokenCategory() == Token.TokenCategory.ID) {
        nextToken();
        if (lookahead.getTokenCategory() == Token.TokenCategory.FEC_PAR)
            {
            nextToken();
            if (lookahead.getTokenCategory() == Token.TokenCategory.SP)
                {
                output.add("<printout_or_readin> ::= ID FEC_PAR SP");
                nextToken();
            } else {
                throw new ParserException("Unexpected symbol " +
                    lookahead + " found!");
            }
        } else {
            throw new ParserException("Unexpected symbol " + lookahead +
                " found!");
        }
    } else {
        this.msg();
        if (lookahead.getTokenCategory() == Token.TokenCategory.FEC_PAR)
            {
            nextToken();
            if (lookahead.getTokenCategory() == Token.TokenCategory.SP)
                {
                output.add("<printout_or_readin> ::= <msg> FEC_PAR SP");
                nextToken();
            } else {
                throw new ParserException("Unexpected symbol " +
                    lookahead + " found!");
            }
        } else {
            throw new ParserException("Unexpected symbol " + lookahead +
                " found!");
        }
    }
}

private void msg() {
    if (lookahead.getTokenCategory() == Token.TokenCategory.CTE_STR) {
        nextToken();
        if (lookahead.getTokenCategory() == Token.TokenCategory.OP_AD) {
            output.add("<msg> ::= CTE_STR OP_AD <msg>");
            nextToken();
            this.msg();
        } else {
          output.add("<msg> ::= CTE_STR");
        }
    }
}

private void attribution_or_function_call() throws ParserException {
    if (lookahead.getTokenCategory() == Token.TokenCategory.VECTOR_AUX)
        {
        String prod = "<attribution_or_function_call> ::= " + lookahead.
            getSequence();
        nextToken();
        if (lookahead.getTokenCategory() == Token.TokenCategory.CTE_INT)
            {
            prod += "CTE_INT <attribution>";
            output.add(prod);
            nextToken();
            this.attribution();
        } else if (lookahead.getTokenCategory() == Token.TokenCategory.
            ID) {
            prod += "ID <attribution>";
                output.add(prod);
                nextToken();
                this.attribution();
            } else {
                throw new ParserException("Unexpected symbol " + lookahead +
                    " found!");
            }
        } else if (lookahead.getTokenCategory() == Token.TokenCategory.
            AB_PAR) {
            String prod = "<attribution_or_function_call> ::= " + lookahead.
                getSequence() + " <parameters_call> FEC_PAR SP";
            nextToken();
            this.parameters_call();
            if (lookahead.getTokenCategory() == Token.TokenCategory.FEC_PAR)
                {
                nextToken();
                if (lookahead.getTokenCategory() == Token.TokenCategory.SP)
                    {
                    output.add(prod);
                    nextToken();
                } else {
                    throw new ParserException("Unexpected symbol " +
                        lookahead + " found!");
                }
            } else {
                throw new ParserException("Unexpected symbol " + lookahead +
                    " found!");
            }
        } else {
            output.add("<attribution_or_function_call> ::= <attribution>");
            this.attribution();
        }
    }

    private void parameters_call() {
        if (lookahead.getTokenCategory() == Token.TokenCategory.ID) {
            output.add("<parameters_call> ::= ID <parameters_call>");
            nextToken();
            this.parameters_call();
        } else if (lookahead.getTokenCategory() == Token.TokenCategory.
            CTE_STR) {
            output.add("<parameters_call> ::= CTE_STR <parameters_call>");
            nextToken();
            this.parameters_call();
        } else if (lookahead.getTokenCategory() == Token.TokenCategory.
            CTE_FLOAT) {
            output.add("<parameters_call> ::= CTE_FLOAT <parameters_call>");
            nextToken();
            this.parameters_call();
        } else if (lookahead.getTokenCategory() == Token.TokenCategory.
            CTE_INT) {
            output.add("<parameters_call> ::= CTE_INT <parameters_call>");
            nextToken();
            this.parameters_call();
        } else if (lookahead.getTokenCategory() == Token.TokenCategory.SP) {
            output.add("<parameters_call> ::= SP <parameters_call>");
            nextToken();
            this.parameters_call();
        } else {
            output.add("<parameters_call> ::= empty");
        }
    }

    private void declaration() throws ParserException {
        if (lookahead.getTokenCategory() == Token.TokenCategory.SP) {
            output.add("<declaration> ::= SP");
            nextToken();
        } else if (lookahead.getTokenCategory() == Token.TokenCategory.
            VECTOR_AUX) {
            String prod = "<declaration> ::= " + lookahead.getSequence() + "
                CTE_INT <declaration_aux>";
            nextToken();
            if (lookahead.getTokenCategory() == Token.TokenCategory.CTE_INT)
                {
                output.add(prod);
                nextToken();
                this.declaration_aux();
            } else if (lookahead.getTokenCategory() == Token.TokenCategory.
                ID) {
                output.add(prod);
                nextToken();
                this.declaration_aux();
            } else {
                throw new ParserException("Unexpected symbol " + lookahead +
                    " found!");
            }
        } else {
            output.add("<declaration> ::= <attribution>");
            this.attribution();
        }
    }

    private void declaration_aux() throws ParserException {
        if (lookahead.getTokenCategory() == Token.TokenCategory.SP) {
            output.add("<declaration_aux> ::= SP");
            nextToken();
        } else {
            output.add("<declaration_aux> ::= <attribution>");
            this.attribution();
        }
    }

    private void attribution() throws ParserException {
        if (lookahead.getTokenCategory() == Token.TokenCategory.OP_ATR) {
            nextToken();
            this.value();
            if (lookahead.getTokenCategory() == Token.TokenCategory.SP) {
                output.add("<attribution> ::= OP_ATR <value> SP");
                nextToken();
            } else {
                throw new ParserException("Unexpected symbol " + lookahead +
                    " found!");
            }
        } else {
            throw new ParserException("Unexpected symbol " + lookahead + "
                found!");
        }
    }

    private void value() throws ParserException {
```

```
        if (lookahead.getTokenCategory() == Token.TokenCategory.AB_CH) {
            output.add("<value> ::= AB_CH <array>");
            nextToken();
            this.array();
        } else {
            output.add("<value> ::= <expression>");
            this.expression();
        }
    }

    private void array() throws ParserException {
        if (lookahead.getTokenCategory() == Token.TokenCategory.FEC_CH) {
            output.add("<array> ::= FEC_CH");
            nextToken();
        } else {
            this.elements();
            if (lookahead.getTokenCategory() == Token.TokenCategory.FEC_CH)
                {
                nextToken();
                output.add("<array> ::= <elements> FEC_CH");
            } else {
                throw new ParserException("Unexpected symbol " + lookahead +
                    " found!");
            }
        }
    }

    private void elements() throws ParserException {
        if (lookahead.getTokenCategory() == Token.TokenCategory.ID) {
            output.add("<elements> ::= ID");
            nextToken();
            this.elements();
        } else if (lookahead.getTokenCategory() == Token.TokenCategory.
            CTE_STR) {
            output.add("<elements> ::= CTE_STR");
            nextToken();
            this.elements();
        } else if (lookahead.getTokenCategory() == Token.TokenCategory.
            CTE_FLOAT) {
            output.add("<elements> ::= CTE_FLOAT");
            nextToken();
            this.elements();
        } else if (lookahead.getTokenCategory() == Token.TokenCategory.
            CTE_INT) {
            output.add("<elements> ::= CTE_INT");
            nextToken();
            this.elements();
        } else if (lookahead.getTokenCategory() == Token.TokenCategory.SP) {
            output.add("<elements> ::= SP <elements>");
            nextToken();
            this.elements();
        } else {
            output.add("<elements> ::= empty");

        }
    }

    private void parameters() throws ParserException {
        if (lookahead.getTokenCategory() == Token.TokenCategory.TYPE_VALUE)
            {
```

```
            String prod = "<parameters>:: " + lookahead.getSequence() + "
                ID";
            nextToken();
            if (lookahead.getTokenCategory() == Token.TokenCategory.ID) {
                output.add(prod);
                nextToken();
            } else {
                throw new ParserException("Unexpected symbol " + lookahead +
                    " found!");
            }
        } else if (lookahead.getTokenCategory() == Token.TokenCategory.SP) {
            String prod = "<parameters>:: " + lookahead.getSequence() + "
                TYPE_VALUE ID <parameters>";
            nextToken();
            if (lookahead.getTokenCategory() == Token.TokenCategory.
                TYPE_VALUE) {
                nextToken();
                if (lookahead.getTokenCategory() == Token.TokenCategory.ID)
                    {
                    output.add(prod);
                    nextToken();
                    this.parameters();
                } else {
                    throw new ParserException("Unexpected symbol " +
                        lookahead + " found!");
                }
            } else {
                throw new ParserException("Unexpected symbol " + lookahead +
                    " found!");
            }
        } else {
            output.add("<parameters>:: empty");
        }
    }

    private void expression() throws ParserException {
        output.add("<expression>:: <eq_expression> <expression_aux>");
        this.eq_expression();
        this.expression_aux();
    }

    private void expression_aux() throws ParserException {
        if (lookahead.getTokenCategory() == Token.TokenCategory.OP_AND) {
            output.add("<expression_aux>:: OP_AND <expression>");
            nextToken();
            this.expression();
        } else if (lookahead.getTokenCategory() == Token.TokenCategory.OP_OR
            ) {
            output.add("<expression_aux>:: OP_OR <expression>");
            nextToken();
            this.expression();
        } else {
            output.add("<expression_aux>:: empty");
        }
    }

    private void eq_expression() throws ParserException {
        output.add("<eq_expression>:: <comparative_exp> <eq_expression_aux>"
            );
        this.comparative_exp();
```

```
            this.eq_expression_aux();
        }

        private void eq_expression_aux() throws ParserException {
            if (lookahead.getTokenCategory() == Token.TokenCategory.OP_REL2) {
                output.add("<eq_expression_aux>:: OP_REL2 <eq_expression>");
                nextToken();
                this.eq_expression();
            } else {
                output.add("<eq_expression_aux>:: empty");
            }
        }

        private void comparative_exp() throws ParserException {
            output.add("<comparative_exp>:: <add_exp> <comparative_exp_aux>");
            this.add_exp();
            this.comparative_exp_aux();
        }

        private void comparative_exp_aux() throws ParserException {
            if (lookahead.getTokenCategory() == Token.TokenCategory.OP_REL1) {
                output.add("<comparative_exp_aux>:: OP_REL1 <comparative_exp>");
                nextToken();
                this.comparative_exp();
            } else {
                output.add("<comparative_exp_aux>:: empty");
            }
        }

        private void add_exp() throws ParserException {
            output.add("<add_exp>:: <mult_exp> <add_exp_aux>");
            this.mult_exp();
            this.add_exp_aux();
        }

        private void add_exp_aux() throws ParserException {
            if (lookahead.getTokenCategory() == Token.TokenCategory.OP_AD) {
                output.add("<add_exp_aux>:: OP_AD <add_exp>");
                nextToken();
                this.add_exp();
            } else {
                output.add("<add_exp_aux>:: empty");
            }
        }

        private void mult_exp() throws ParserException {
            output.add("<mult_exp>:: <neg_exp> <mult_exp_aux>");
            this.neg_exp();
            this.mult_exp_aux();
        }

        private void mult_exp_aux() throws ParserException {
            if (lookahead.getTokenCategory() == Token.TokenCategory.OP_MULT) {
                output.add("<mult_exp_aux>:: OP_MULT <mult_exp>");
                nextToken();
                this.mult_exp();
            } else {
                output.add("<mult_exp_aux>:: empty");
            }
        }
```

```
    private void neg_exp() throws ParserException {
        if (lookahead.getTokenCategory() == Token.TokenCategory.OP_NOT) {
            output.add("<neg_exp>:: OP_NOT <exp_aux>");
            nextToken();
            this.exp_aux();
        } else {
            output.add("<neg_exp>:: <exp_aux>");
            this.exp_aux();
        }
    }

    private void exp_aux() throws ParserException {
        if (lookahead.getTokenCategory() == Token.TokenCategory.AB_PAR) {
            nextToken();
            this.atom_exp();
            if (lookahead.getTokenCategory() == Token.TokenCategory.FEC_PAR)
                {
                output.add("<exp_aux>:: AB_PAR <atom_exp> FEC_PAR");
                nextToken();
            } else {
                throw new ParserException("Unexpected symbol " + lookahead +
                    " found!");
            }
        } else {
            output.add("<exp_aux>:: <atom_exp>");
            this.atom_exp();
        }
    }

    private void atom_exp() throws ParserException {
        if (lookahead.getTokenCategory() == Token.TokenCategory.ID) {
            nextToken();
            if (lookahead.getTokenCategory() == Token.TokenCategory.AB_PAR)
                {
                nextToken();
                this.parameters_call();
                if (lookahead.getTokenCategory() == Token.TokenCategory.
                    FEC_PAR) {
                    output.add("<atom_exp> ::= ID AB_PAR <parameters_call>
                        FEC_PAR");
                    nextToken();
                } else {
                    throw new ParserException("Unexpected symbol " +
                        lookahead + " found!");
                }
            } else {
                output.add("<atom_exp> ::= ID");
            }
        } else if (lookahead.getTokenCategory() == Token.TokenCategory.
            CTE_STR) {
            output.add("<atom_exp> ::= CTE_STR");
            nextToken();
        } else if (lookahead.getTokenCategory() == Token.TokenCategory.
            CTE_FLOAT) {
            output.add("<atom_exp> ::= CTE_FLOAT");
            nextToken();
        } else if (lookahead.getTokenCategory() == Token.TokenCategory.
            CTE_INT) {
            output.add("<atom_exp> ::= CTE_INT");
            nextToken();
        } else if (lookahead.getTokenCategory() == Token.TokenCategory.
            BOOL_VALUE) {
            output.add("<atom_exp> ::= BOOL_VALUE");
            nextToken();
        } else {
            throw new ParserException("Unexpected symbol " + lookahead + "
                found!");
        }
    }

    /**
     * Remove the first token from the list and store the next token in
         lookahead
     */

    private void nextToken() {
        this.tokens.removeFirst();

        if ( tokens.isEmpty() ) {
            lookahead = new Token( Token.TokenCategory.EOF, "", -1, -1);
        } else {
            if (tokens.getFirst().getTokenCategory() == Token.TokenCategory.
                COMMENT) {
                nextToken();
            } else {
                lookahead = tokens.getFirst();
            }
        }
    }


}
```