1. **Transformed grammar into LL(1)**

In the original grammar, there are some left recursions and ambiguities. Using the tool (grammartool.jar), 13 optionality EBNF construct () and 3 zero-or-more repetition EBNF constructs can be found and transformed.

**2 left-recursive constructs** can also be found and replaced:

1. Original left-recursive rule: <term>

<term> ::= <term> <multOp> <factor>

<term> ::= <factor>

Replaced by rules: <term> and <rightrec-term>

<term> ::= <factor> <rightrec-term>

<rightrec-term> ::= EPSILON

<rightrec-term> ::= <multOp> <factor> <rightrec-term>

1. Original left-recursive rule: < arithExpr >

<arithExpr> ::= <arithExpr> <addOp> <term>

<arithExpr> ::= <term>

Replaced by rules: < arithExpr > and < rightrec-arithExpr >

<arithExpr> ::= <term> <rightrec-arithExpr>

<rightrec-arithExpr> ::= EPSILON

<rightrec-arithExpr> ::= <addOp> <term> <rightrec-arithExpr>

**Ambiguities** can be detected by the online ucalgary tool, but they need to be fixed manually.

<arraySize> ::= '[' 'intNum' ']'

<arraySize> ::= '[' ']'

Replaced by:

<arraySize> ::= '[' <arraySize1>

<arraySize1> ::= 'intNum' ']'

<arraySize1> ::= ']'

<expr> ::= <arithExpr>

<expr> ::= <relExpr> (<relExpr> ::= <arithExpr> <relOp> <arithExpr>)

Replaced by:

<expr> ::= <arithExpr> <expr1>

<expr1> ::= EPSILON

<expr1> ::= <relOp> <arithExpr>

<factor> ::= <variable>

<factor> ::= <functionCall> (both have id as first right hand side)

Replaced by:

<factor> ::= 'id' <factor1>

<factor1> ::= <variableTail> <factor2>

<factor1> ::= <functionCallTail> <factor2>

<factor2> ::= '.' 'id' <factor1>

<factor2> ::= EPSILON

<funcDecl> ::= 'func' 'id' '(' <fParams> ')' ':' <type> ';'

<funcDecl> ::= 'func' 'id' '(' <fParams> ')' ':' 'void' ';'

Replaced by:

<funcDecl> ::= 'func' <funcDecl0>

<funcDecl0> ::= 'id' <funcDecl1>

<funcDecl1> ::= '(' <funcDecl2>

<funcDecl2> ::= <fParams> <funcDecl3>

<funcDecl3> ::= ')' <funcDecl4>

<funcDecl4> ::= ':' <funcDecl5>

<funcDecl5> ::= <type> ';'

<funcDecl5> ::= 'void' ';'

<funcHead> ::= 'func' <opt-funcHead1> 'id' '(' <fParams> ')' ':' <type>

<funcHead> ::= 'func' <opt-funcHead1> 'id' '(' <fParams> ')' ':' 'void'

Replaced by:

<funcHead> ::= 'func' 'id' <funcHead1>

<funcHead1> ::= <opt-funcHead1> <funcHead2>

<funcHead2> ::= '(' <funcHead3>

<funcHead3> ::= <fParams> <funcHead4>

<funcHead4> ::= ')' <funcHead5>

<funcHead5> ::= ':' <funcHead6>

<funcHead6> ::= <type>

<funcHead6> ::= 'void'

<statement> ::= <assignStat> ';'

<statement> ::= <functionCall> ';'

Replaced by:

<statement> ::= 'id' <statement1> ';'

<statement1> ::= <variableTail> <statement2>

<statement1> ::= <functionCallTail> <statement3>

<statement2> ::= '.' 'id' <statement1>

<statement2> ::= <assignOp> <expr>

<statement3> ::= '.' 'id' <statement1>

<statement3> ::= EPSILON

<variable> ::= <rept-variable0> 'id' <rept-variable2>

<rept-variable0> ::= <idnest> <rept-variable0>

<rept-variable0> ::= EPSILON

<rept-variable2> ::= <indice> <rept-variable2>

<rept-variable2> ::= EPSILON

Replaced by:

<variable> ::= 'id' <variable1>

<variable1> ::= <variableTail> <variable2>

<variable1> ::= <functionCallTail> <variable3>

<variable2> ::= '.' 'id' <variable1>

<variable2> ::= EPSILON

<variable3> ::= '.' 'id' <variable1>

<functionCallTail> ::= '(' <aParams> ')'

<variableTail> ::= <indice> <variableTail>

<variableTail> ::= EPSILON

In order to keep consistent with the course setup, I used the LL(1) grammar provided by the professor. The grammar is listed in the **appendix** section.

1. **FIRST and FOLLOW sets**

Using the online ucalgary tool as well as the grammartool, I derived the FIRST and FOLLOW sets for each non-terminal. They are listed in the **appendix** section.

1. **Design**

Class diagram as follows:

A picture containing diagram

Description automatically generated

I followed the approach of **Table-driven predictive parsing**, imported the parser table generated by the online ucalgary tool, implemented the parser algorithm in slide 4.SyntaxII-page 22. The parsing table which bases on FIRST and FOLLOW sets tells the parser which right-hand-side of the rule to choose when the top of stack is non-terminal.

The benefit of Table-driven parsing is that the algorithm is universal. I first implemented the parser using the example grammar in lecture slides to make sure the parser works. Then I applied the project LL(1) grammar to the parser. The same process is with creation of AST.

The **Rule class** and **SemanticAction** class store rules and sematic actions in the form of left-hand-side part and right-hand-side part.

The **Grammar class** separate information about the grammar with the parser. It imports parsing table, FIRST and FOLLOW sets from files that derive from the use of tools. The files are placed in resource folder. The one called “LL1.attrbute.grm” is the grammar rules injected with sematic actions.

The **NodeFactory** class is dealing with the creation of different node depending on the different parameters. A package of AST is created with a group of nodes which represent the abstract syntactic structure of source code. **Node** class is the parent class that implements several methods(makeSiblings, adoptChildren, etc).

The **class of SyntacticAnalyzer(parser)** is responsible for parsing every file, generating a final abstract syntax tree and writing to output files (derivation output, AST output, error reporting). It contains a LexicalAnalyzer object which is implemented in assignment #1 for tokenizing the src file. A Grammar object and a NodeFactory object are aggregated in it.Two stacks (parsing stack and semantic stack) are utilized to implement the parsing and sematic actions.

The **SyntacticAnalyzerDriver** opens every file(.src) in a folder or a single file(.src), creates a SyntacticAnalyzer object to handle the task of parsing and generating output files.

More detailed explanation can be seen in the source code.

1. **Use of tools**

I used grammartool.jar that provided by the professor to modify the grammar and changed it to the [ucalgary](https://smlweb.cpsc.ucalgary.ca/start.html) online tool form. Although I used the given LL(1) grammar at last, the tool is very useful for dealing with grammar. The ucalgary online tool is also useful in validating the LL(1) grammar and generating FIRST/FOLLOW sets and parsing table.

[Jsoup](https://jsoup.org/), a Java library for parsing HTML, was used in Grammar class the import IRST/FOLLOW sets and parsing table. These files are HTML files, so Jsoup helped me extract and manipulating data, letting me focus on the implementation of parser and AST part.

[Graphviz Online](https://dreampuf.github.io/GraphvizOnline) is a great website for displaying .dot file in the structure of a tree. It is easy to output .dot file and no other software needs to be installed.

As for writing codes, I used [IntelliJ IDEA](https://www.jetbrains.com/idea/), which is great at coding auto completion, code debugging, etc. The class diagram is generated by [PlantUML](https://plantuml.com/), which can be integrated in the IDE conveniently using a plugin called [sketch it](https://plugins.jetbrains.com/plugin/10387-sketch-it-). Through this workflow, the drawing of UML diagram is greatly simplified.

Appendix

**LL(1) grammar:**

<AParams> ::= <Expr> <AParamsTail>

<AParams> ::= EPSILON

<AParamsTail> ::= ',' <Expr> <AParamsTail>

<AParamsTail> ::= EPSILON

<AddOp> ::= '+'

<AddOp> ::= '-'

<AddOp> ::= 'or'

<ArithExpr> ::= <Term> <ArithExprTail>

<ArithExprTail> ::= <AddOp> <Term> <ArithExprTail>

<ArithExprTail> ::= EPSILON

<ArraySizeRept> ::= '[' <IntNum> ']' <ArraySizeRept>

<ArraySizeRept> ::= EPSILON

<AssignOp> ::= 'assign'

<AssignStatTail> ::= <AssignOp> <Expr>

<ClassDecl> ::= 'class' 'id' <Inherit> '{' <ClassDeclBody> '}' ';' <ClassDecl>

<ClassDecl> ::= EPSILON

<ClassDeclBody> ::= <Visibility> <MemberDecl> <ClassDeclBody>

<ClassDeclBody> ::= EPSILON

<ClassMethod> ::= 'sr' 'id'

<ClassMethod> ::= EPSILON

<Expr> ::= <ArithExpr> <ExprTail>

<ExprTail> ::= <RelOp> <ArithExpr>

<ExprTail> ::= EPSILON

<FParams> ::= <Type> 'id' <ArraySizeRept> <FParamsTail>

<FParams> ::= EPSILON

<FParamsTail> ::= ',' <Type> 'id' <ArraySizeRept> <FParamsTail>

<FParamsTail> ::= EPSILON

<Factor> ::= <FuncOrVar>

<Factor> ::= 'intnum'

<Factor> ::= 'floatnum'

<Factor> ::= 'stringlit'

<Factor> ::= '(' <Expr> ')'

<Factor> ::= 'not' <Factor>

<Factor> ::= <Sign> <Factor>

<Factor> ::= 'qm' '[' <Expr> ':' <Expr> ':' <Expr> ']'

<FuncBody> ::= '{' <MethodBodyVar> <StatementList> '}'

<FuncDecl> ::= 'func' 'id' '(' <FParams> ')' ':' <FuncDeclTail> ';'

<FuncDeclTail> ::= <Type>

<FuncDeclTail> ::= 'void'

<FuncDef> ::= <Function> <FuncDef>

<FuncDef> ::= EPSILON

<FuncHead> ::= 'func' 'id' <ClassMethod> '(' <FParams> ')' ':' <FuncDeclTail>

<FuncOrAssignStat> ::= 'id' <FuncOrAssignStatIdnest>

<FuncOrAssignStatIdnest> ::= <IndiceRep> <FuncOrAssignStatIdnestVarTail>

<FuncOrAssignStatIdnest> ::= '(' <AParams> ')' <FuncOrAssignStatIdnestFuncTail>

<FuncOrAssignStatIdnestFuncTail> ::= '.' 'id' <FuncStatTail>

<FuncOrAssignStatIdnestFuncTail> ::= EPSILON

<FuncStatTail> ::= <IndiceRep> '.' 'id' <FuncStatTail>

<FuncStatTail> ::= '(' <AParams> ')' <FuncStatTailIdnest>

<FuncStatTailIdnest> ::= '.' 'id' <FuncStatTail>

<FuncStatTailIdnest> ::= EPSILON

<FuncOrAssignStatIdnestVarTail> ::= '.' 'id' <FuncOrAssignStatIdnest>

<FuncOrAssignStatIdnestVarTail> ::= <AssignStatTail>

<FuncOrVar> ::= 'id' <FuncOrVarIdnest>

<FuncOrVarIdnest> ::= <IndiceRep> <FuncOrVarIdnestTail>

<FuncOrVarIdnest> ::= '(' <AParams> ')' <FuncOrVarIdnestTail>

<FuncOrVarIdnestTail> ::= '.' 'id' <FuncOrVarIdnest>

<FuncOrVarIdnestTail> ::= EPSILON

<Function> ::= <FuncHead> <FuncBody>

<IndiceRep> ::= '[' <Expr> ']' <IndiceRep>

<IndiceRep> ::= EPSILON

<Inherit> ::= 'inherits' 'id' <NestedId>

<Inherit> ::= EPSILON

<IntNum> ::= 'intnum'

<IntNum> ::= EPSILON

<MemberDecl> ::= <FuncDecl>

<MemberDecl> ::= <VarDecl>

<MethodBodyVar> ::= 'var' '{' <VarDeclRep> '}'

<MethodBodyVar> ::= EPSILON

<MultOp> ::= '\*'

<MultOp> ::= '/'

<MultOp> ::= 'and'

<NestedId> ::= ',' 'id' <NestedId>

<NestedId> ::= EPSILON

<Prog> ::= <ClassDecl> <FuncDef> 'main' <FuncBody>

<RelOp> ::= 'eq'

<RelOp> ::= 'neq'

<RelOp> ::= 'lt'

<RelOp> ::= 'gt'

<RelOp> ::= 'leq'

<RelOp> ::= 'geq'

<START> ::= <Prog>

<Sign> ::= '+'

<Sign> ::= '-'

<StatBlock> ::= '{' <StatementList> '}'

<StatBlock> ::= <Statement>

<StatBlock> ::= EPSILON

<Statement> ::= <FuncOrAssignStat> ';'

<Statement> ::= 'if' '(' <Expr> ')' 'then' <StatBlock> 'else' <StatBlock> ';'

<Statement> ::= 'while' '(' <Expr> ')' <StatBlock> ';'

<Statement> ::= 'read' '(' <Variable> ')' ';'

<Statement> ::= 'write' '(' <Expr> ')' ';'

<Statement> ::= 'return' '(' <Expr> ')' ';'

<Statement> ::= 'break' ';'

<Statement> ::= 'continue' ';'

<StatementList> ::= <Statement> <StatementList>

<StatementList> ::= EPSILON

<Term> ::= <Factor> <TermTail>

<TermTail> ::= <MultOp> <Factor> <TermTail>

<TermTail> ::= EPSILON

<Type> ::= 'integer'

<Type> ::= 'float'

<Type> ::= 'string'

<Type> ::= 'id'

<VarDecl> ::= <Type> 'id' <ArraySizeRept> ';'

<VarDeclRep> ::= <VarDecl> <VarDeclRep>

<VarDeclRep> ::= EPSILON

<Variable> ::= 'id' <VariableIdnest>

<VariableIdnest> ::= <IndiceRep> <VariableIdnestTail>

<VariableIdnestTail> ::= '.' 'id' <VariableIdnest>

<VariableIdnestTail> ::= EPSILON

<Visibility> ::= 'public'

<Visibility> ::= 'private'

<Visibility> ::= EPSILON

**FIRST set**

FIRST(<Intnum>)= ['intnum', EPSILON]

FIRST(<Vardeclrep>)= ['integer', 'float', 'string', 'id', EPSILON]

FIRST(<Start>)= ['main', 'class', 'func']

FIRST(<Statementlist>)= ['if', 'while', 'read', 'write', 'return', 'break', 'continue', 'id', EPSILON]

FIRST(<Funcorassignstatidnestfunctail>)= ['dot', EPSILON]

FIRST(<Memberdecl>)= ['func', 'integer', 'float', 'string', 'id']

FIRST(<Funcorvaridnest>)= ['lpar', 'lsqbr', 'dot', EPSILON]

FIRST(<Factor>)= ['intnum', 'floatnum', 'stringlit', 'lpar', 'not', 'qm', 'id', 'plus', 'minus']

FIRST(<Inherit>)= ['inherits', EPSILON]

FIRST(<Funcorvaridnesttail>)= ['dot', EPSILON]

FIRST(<Classmethod>)= ['sr', EPSILON]

FIRST(<Aparams>)= ['intnum', 'floatnum', 'stringlit', 'lpar', 'not', 'qm', 'id', 'plus', 'minus', EPSILON]

FIRST(<Funcbody>)= ['lcurbr']

FIRST(<Variableidnesttail>)= ['dot', EPSILON]

FIRST(<Relop>)= ['eq', 'neq', 'lt', 'gt', 'leq', 'geq']

FIRST(<Termtail>)= ['mult', 'div', 'and', EPSILON]

FIRST(<Term>)= ['intnum', 'floatnum', 'stringlit', 'lpar', 'not', 'qm', 'id', 'plus', 'minus']

FIRST(<Funcdecl>)= ['func']

FIRST(<Nestedid>)= ['comma', EPSILON]

FIRST(<Indicerep>)= ['lsqbr', EPSILON]

FIRST(<Expr>)= ['intnum', 'floatnum', 'stringlit', 'lpar', 'not', 'qm', 'id', 'plus', 'minus']

FIRST(<Fparamstail>)= ['comma', EPSILON]

FIRST(<Multop>)= ['mult', 'div', 'and']

FIRST(<Visibility>)= ['public', 'private', EPSILON]

FIRST(<Funcstattail>)= ['dot', 'lpar', 'lsqbr']

FIRST(<Variable>)= ['id']

FIRST(<Variableidnest>)= ['lsqbr', 'dot', EPSILON]

FIRST(<Arithexpr>)= ['intnum', 'floatnum', 'stringlit', 'lpar', 'not', 'qm', 'id', 'plus', 'minus']

FIRST(<Funcorvar>)= ['id']

FIRST(<Function>)= ['func']

FIRST(<Fparams>)= ['integer', 'float', 'string', 'id', EPSILON]

FIRST(<Prog>)= ['main', 'class', 'func']

FIRST(<Arraysizerept>)= ['lsqbr', EPSILON]

FIRST(<Exprtail>)= ['eq', 'neq', 'lt', 'gt', 'leq', 'geq', EPSILON]

FIRST(<Classdeclbody>)= ['public', 'private', 'func', 'integer', 'float', 'string', 'id', EPSILON]

FIRST(<Funcdecltail>)= ['void', 'integer', 'float', 'string', 'id']

FIRST(<Addop>)= ['plus', 'minus', 'or']

FIRST(<Assignstattail>)= ['assign']

FIRST(<Funchead>)= ['func']

FIRST(<Vardecl>)= ['integer', 'float', 'string', 'id']

FIRST(<Funcorassignstat>)= ['id']

FIRST(<Statement>)= ['if', 'while', 'read', 'write', 'return', 'break', 'continue', 'id']

FIRST(<Classdecl>)= ['class', EPSILON]

FIRST(<Statblock>)= ['lcurbr', 'if', 'while', 'read', 'write', 'return', 'break', 'continue', 'id', EPSILON]

FIRST(<Funcstattailidnest>)= ['dot', EPSILON]

FIRST(<Arithexprtail>)= ['plus', 'minus', 'or', EPSILON]

FIRST(<Funcorassignstatidnestvartail>)= ['dot', 'assign']

FIRST(<Type>)= ['integer', 'float', 'string', 'id']

FIRST(<Funcdef>)= ['func', EPSILON]

FIRST(<Aparamstail>)= ['comma', EPSILON]

FIRST(<Methodbodyvar>)= ['var', EPSILON]

FIRST(<Sign>)= ['plus', 'minus']

FIRST(<Funcorassignstatidnest>)= ['lpar', 'lsqbr', 'dot', 'assign']

FIRST(<Assignop>)= ['assign']

**FOLLOW set**

FOLLOW(<Intnum>)= ['rsqbr']

FOLLOW(<Vardeclrep>)= ['rcurbr']

FOLLOW(<Start>)= ['∅']

FOLLOW(<Statementlist>)= ['rcurbr']

FOLLOW(<Funcorassignstatidnestfunctail>)= ['semi']

FOLLOW(<Memberdecl>)= ['public', 'private', 'func', 'integer', 'float', 'string', 'id', 'rcurbr']

FOLLOW(<Funcorvaridnest>)= ['mult', 'div', 'and', 'semi', 'eq', 'neq', 'lt', 'gt', 'leq', 'geq', 'plus', 'minus', 'or', 'comma', 'colon', 'rsqbr', 'rpar']

FOLLOW(<Factor>)= ['mult', 'div', 'and', 'semi', 'eq', 'neq', 'lt', 'gt', 'leq', 'geq', 'plus', 'minus', 'or', 'comma', 'colon', 'rsqbr', 'rpar']

FOLLOW(<Inherit>)= ['lcurbr']

FOLLOW(<Funcorvaridnesttail>)= ['mult', 'div', 'and', 'semi', 'eq', 'neq', 'lt', 'gt', 'leq', 'geq', 'plus', 'minus', 'or', 'comma', 'colon', 'rsqbr', 'rpar']

FOLLOW(<Classmethod>)= ['lpar']

FOLLOW(<Aparams>)= ['rpar']

FOLLOW(<Funcbody>)= ['main', 'func']

FOLLOW(<Variableidnesttail>)= ['rpar']

FOLLOW(<Relop>)= ['intnum', 'floatnum', 'stringlit', 'lpar', 'not', 'qm', 'id', 'plus', 'minus']

FOLLOW(<Termtail>)= ['semi', 'eq', 'neq', 'lt', 'gt', 'leq', 'geq', 'plus', 'minus', 'or', 'comma', 'colon', 'rsqbr', 'rpar']

FOLLOW(<Term>)= ['semi', 'eq', 'neq', 'lt', 'gt', 'leq', 'geq', 'plus', 'minus', 'or', 'comma', 'colon', 'rsqbr', 'rpar']

FOLLOW(<Funcdecl>)= ['public', 'private', 'func', 'integer', 'float', 'string', 'id', 'rcurbr']

FOLLOW(<Nestedid>)= ['lcurbr']

FOLLOW(<Indicerep>)= ['mult', 'div', 'and', 'semi', 'assign', 'dot', 'eq', 'neq', 'lt', 'gt', 'leq', 'geq', 'plus', 'minus', 'or', 'comma', 'colon', 'rsqbr', 'rpar']

FOLLOW(<Expr>)= ['semi', 'comma', 'colon', 'rsqbr', 'rpar']

FOLLOW(<Fparamstail>)= ['rpar']

FOLLOW(<Multop>)= ['intnum', 'floatnum', 'stringlit', 'lpar', 'not', 'qm', 'id', 'plus', 'minus']

FOLLOW(<Visibility>)= ['func', 'integer', 'float', 'string', 'id']

FOLLOW(<Funcstattail>)= ['semi']

FOLLOW(<Variable>)= ['rpar']

FOLLOW(<Variableidnest>)= ['rpar']

FOLLOW(<Arithexpr>)= ['semi', 'eq', 'neq', 'lt', 'gt', 'leq', 'geq', 'comma', 'colon', 'rsqbr', 'rpar']

FOLLOW(<Funcorvar>)= ['mult', 'div', 'and', 'semi', 'eq', 'neq', 'lt', 'gt', 'leq', 'geq', 'plus', 'minus', 'or', 'comma', 'colon', 'rsqbr', 'rpar']

FOLLOW(<Function>)= ['main', 'func']

FOLLOW(<Fparams>)= ['rpar']

FOLLOW(<Prog>)= ['∅']

FOLLOW(<Arraysizerept>)= ['rpar', 'comma', 'semi']

FOLLOW(<Exprtail>)= ['semi', 'comma', 'colon', 'rsqbr', 'rpar']

FOLLOW(<Classdeclbody>)= ['rcurbr']

FOLLOW(<Funcdecltail>)= ['lcurbr', 'semi']

FOLLOW(<Addop>)= ['intnum', 'floatnum', 'stringlit', 'lpar', 'not', 'qm', 'id', 'plus', 'minus']

FOLLOW(<Assignstattail>)= ['semi']

FOLLOW(<Funchead>)= ['lcurbr']

FOLLOW(<Vardecl>)= ['public', 'private', 'func', 'integer', 'float', 'string', 'id', 'rcurbr']

FOLLOW(<Funcorassignstat>)= ['semi']

FOLLOW(<Statement>)= ['if', 'while', 'read', 'write', 'return', 'break', 'continue', 'id', 'else', 'semi', 'rcurbr']

FOLLOW(<Classdecl>)= ['func', 'main']

FOLLOW(<Statblock>)= ['else', 'semi']

FOLLOW(<Funcstattailidnest>)= ['semi']

FOLLOW(<Arithexprtail>)= ['semi', 'eq', 'neq', 'lt', 'gt', 'leq', 'geq', 'comma', 'colon', 'rsqbr', 'rpar']

FOLLOW(<Funcorassignstatidnestvartail>)= ['semi']

FOLLOW(<Type>)= ['lcurbr', 'semi', 'id']

FOLLOW(<Funcdef>)= ['main']

FOLLOW(<Aparamstail>)= ['rpar']

FOLLOW(<Methodbodyvar>)= ['if', 'while', 'read', 'write', 'return', 'break', 'continue', 'id', 'rcurbr']

FOLLOW(<Sign>)= ['intnum', 'floatnum', 'stringlit', 'lpar', 'not', 'qm', 'id', 'plus', 'minus']

FOLLOW(<Funcorassignstatidnest>)= ['semi']

FOLLOW(<Assignop>)= ['intnum', 'floatnum', 'stringlit', 'lpar', 'not', 'qm', 'id', 'plus', 'minus']