Compiler Design
COMP442 - Winter 2021

Assignment 2

Report

Kresten Ordekian 40030197

Section 1. Transformed grammar into LL(1)

<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
```

Section 2. FIRST and FOLLOW sets

Symbol	First set	Follow set
ADDOP	plus, minus, or	plus, minus, id, intnum, floatnum, stringlit, lpar, not, qm, $
APARAMS	plus, minus, id, intnum, floatnum, stringlit, lpar, not, qm	rpar
APARAMSTAIL	comma	rpar
ARITHEXPR	plus, minus, id, intnum, floatnum, stringlit, lpar, not, qm	comma, rsqbr, semi, rpar, colon, eq, neq, lt, gt, leq, geq
ARITHEXPRTAIL	plus, minus, or	comma, rsqbr, semi, rpar, colon, eq, neq, lt, gt, leq, geq
ARRAYSIZEREPT	lsqbr	comma, semi, rpar
ASSIGNOP	assign	plus, minus, id, intnum, floatnum, stringlit, lpar, not, qm
ASSIGNSTATTAIL	assign	semi
CLASSDECL	class	func, main
CLASSDECLBODY	id, func, integer, float, string, public, private	rcurbr
CLASSMETHOD	sr	lpar
EXPR	plus, minus, id, intnum, floatnum, stringlit, lpar, not, qm	comma, rsqbr, semi, rpar, colon
EXPRTAIL	eq, neq, lt, gt, leq, geq	comma, rsqbr, semi, rpar, colon
FACTOR	plus, minus, id, intnum, floatnum, stringlit, lpar, not, qm	plus, minus, or, comma, rsqbr, semi, rpar, colon, mult, div, and, eq, neq, lt, gt, leq, geq
FPARAMS	id, integer, float, string	rpar
FPARAMSTAIL	comma	rpar
FUNCBODY	lcurbr	func, main
FUNCDECL	func	id, rcurbr, func, integer, float, string, public, private

FUNCDECLTAIL	**id, void, integer, float, string**	**lcurbr, semi**
FUNCDEF	**func**	**main**
FUNCHEAD	**func**	**lcurbr**
FUNCORASSIGNSTAT	**id**	**semi**
FUNCORASSIGNSTATIDNEST	**lsqbr, assign, lpar, dot**	**semi**
FUNCORASSIGNSTATIDNESTFUNCTAIL	**dot**	**semi**
FUNCORASSIGNSTATIDNESTVARTAIL	**assign, dot**	**semi**
FUNCORVAR	**id**	**plus, minus, or, comma, rsqbr, semi, rpar, colon, mult, div, and, eq, neq, lt, gt, leq, geq**
FUNCORVARIDNEST	**lsqbr, lpar, dot**	**plus, minus, or, comma, rsqbr, semi, rpar, colon, mult, div, and, eq, neq, lt, gt, leq, geq**
FUNCORVARIDNESTTAIL	**dot**	**plus, minus, or, comma, rsqbr, semi, rpar, colon, mult, div, and, eq, neq, lt, gt, leq, geq**
FUNCSTATTAIL	**lsqbr, lpar, dot**	**semi**
FUNCSTATTAILIDNEST	**dot**	**semi**
FUNCTION	**func**	**func, main**
INDICEREP	**lsqbr**	**plus, minus, or, comma, rsqbr, assign, semi, rpar, colon, dot, mult, div, and, eq, neq, lt, gt, leq, geq**
INHERIT	**inherits**	**lcurbr**
INTNUM	**intnum**	**rsqbr**
MEMBERDECL	**id, func, integer, float, string**	**id, rcurbr, func, integer, float, string, public, private**
METHODBODYVAR	**var**	**id, rcurbr, if, while, read, write, return, break, continue**
MULTOP	**mult, div, and**	**plus, minus, id, intnum, floatnum, stringlit, lpar, not, qm**

NESTEDID	**comma**	**lcurbr**
PROG	**class, func, main**	
RELOP	**eq, neq, lt, gt, leq, geq**	**plus, minus, id, intnum, floatnum, stringlit, lpar, not, qm**
SIGN	**plus, minus**	**plus, minus, id, intnum, floatnum, stringlit, lpar, not, qm**
START	**class, func, main**	
STATBLOCK	**id, lcurbr, if, while, read, write, return, break, continue**	**semi, else**
STATEMENT	**id, if, while, read, write, return, break, continue**	**id, rcurbr, semi, if, else, while, read, write, return, break, continue**
STATEMENTLIST	**id, if, while, read, write, return, break, continue**	**rcurbr**
TERM	**plus, minus, id, intnum, floatnum, stringlit, lpar, not, qm**	**plus, minus, or, comma, rsqbr, semi, rpar, colon, eq, neq, lt, gt, leq, geq**
TERMTAIL	**mult, div, and**	**plus, minus, or, comma, rsqbr, semi, rpar, colon, eq, neq, lt, gt, leq, geq**
TYPE	**id, integer, float, string**	**id, lcurbr, semi**
VARDECL	**id, integer, float, string**	**id, rcurbr, func, integer, float, string, public, private**
VARDECLREP	**id, integer, float, string**	**rcurbr**
VARIABLE	**id**	**rpar**
VARIABLEIDNEST	**lsqbr, dot**	**rpar**
VARIABLEIDNESTTAIL	**dot**	**rpar**
VISIBILITY	**public, private**	**id, func, integer, float, string**

Section 3. Design

I chose to implement a recursive descent predictive parser. I chose it because I wanted to be able to debug and see everything step by step.

I wrote a function for each of the non terminal symbols. I wrote a match function to check if the terminal symbols and tokens match.

I wrote skipErrors method to find and display a message of the line of the error to the user and recover from the error

I have a stack that keeps track of the function calls and I get the derivation output using the reverse of the stack and terminal symbols.

I have a separate ast.py file that has all the ast data structure methods.

Section 4. Use of tools

I used graphviz to visualize the tree.