Mohammed Umaruddin Assignment 2:

Section 1: Transformed grammar LL(1)

ARRAYSIZE has a first set conflict

EXPR has a first set conflict

FACTOR has a first set conflict

IDNEST has a first set conflict

STATEMENT has a first set conflict

There were various additional ambiguities while I was processing the grammar. Like aParms, intersecting first and follow sets.

Grammar fixed to LL(1):

START -> prog

prog -> rept-prog0

rept-prog0 -> classOrImplOrFunc rept-prog0

rept-prog0 -> EPSILON

classOrImplOrFunc -> classDecl

classOrImplOrFunc -> implDef

classOrImplOrFunc -> funcDef

classDecl -> 'class' 'id' opt-classDecl2 '{' rept-classDecl4 '}' ';'

opt-classDecl2 -> 'isa' 'id' rept-opt-classDecl22

opt-classDecl2 -> EPSILON

rept-opt-classDecl22 -> ',' 'id' rept-opt-classDecl22

rept-opt-classDecl22 -> EPSILON

rept-classDecl4 -> visibility memberDecl rept-classDecl4

rept-classDecl4 -> EPSILON

implDef -> 'implementation' 'id' '{' rept-implDef3 '}'

rept-implDef3 -> funcDef rept-implDef3

rept-implDef3 -> EPSILON

funcDef -> funcHead funcBody

funcHead -> 'function' 'id' '(' fParams ')' 'arrow' returnType

funcHead -> 'constructor' '(' fParams ')'

funcBody -> '{' rept-funcBody1 '}'

rept-funcBody1 -> localVarDeclOrStat rept-funcBody1

rept-funcBody1 -> EPSILON

localVarDeclOrStat -> localVarDecl

localVarDeclOrStat -> statement

arithExpr -> term rightrec-arithExpr

rightrec-arithExpr -> EPSILON

rightrec-arithExpr -> addOp term rightrec-arithExpr

term -> factor rightrec-term

rightrec-term -> EPSILON

rightrec-term -> multOp factor rightrec-term

factor -> idOrSelf factor2 VarOrFunc

factor2 -> '(' aParams ')'

factor2 -> rept-idnest10

VarOrFunc -> idnest VarOrFunc

VarOrFunc -> EPSILON

factor -> 'intLit'

factor -> 'floatLit'

factor -> '(' arithExpr ')'

factor -> 'not' factor

factor -> sign factor

arraySize -> '[' arraySizeTail

arraySizeTail -> 'intNum' ']'

arraySizeTail -> ']'

type -> 'int'

type -> 'float'

type -> 'id'

returnType -> type

returnType -> 'void'

memberDecl -> funcDecl

memberDecl -> attributeDecl

funcDecl -> funcHead ';'

attributeDecl -> 'attribute' varDecl

varDecl -> 'id' ':' type rept-varDecl3 ';'

rept-varDecl3 -> arraySize rept-varDecl3

rept-varDecl3 -> EPSILON

expr -> arithExpr relExprRest

relExprRest -> relOp arithExpr

relExprRest -> EPSILON

fParams -> 'id' ':' type rept-fParams3 rept-fParams4

fParams -> EPSILON

rept-fParams3 -> arraySize rept-fParams3

rept-fParams3 -> EPSILON

rept-fParams4 -> fParamsTail rept-fParams4

rept-fParams4 -> EPSILON

fParamsTail -> ',' 'id' ':' type rept-fParamsTail4

rept-fParamsTail4 -> arraySize rept-fParamsTail4

rept-fParamsTail4 -> EPSILON

aParams -> expr rept-aParams1

aParams -> EPSILON

rept-aParams1 -> aParamsTail rept-aParams1

rept-aParams1 -> EPSILON

aParamsTail -> ',' expr

idOrSelf -> 'id'

idOrSelf -> 'self'

idnest -> '.' 'id' idnest2

idnest2 -> '(' aParams ')'

idnest2 -> rept-idnest10

rept-idnest10 -> indice rept-idnest10

rept-idnest10 -> EPSILON

indice -> '[' arithExpr ']'

variable -> idOrSelf Variable2

Variable2 -> rept-idnest10 rept-variable

Variable2 -> '(' aParams ')' idnest

rept-variable -> varidnest rept-variable

rept-variable -> EPSILON

varidnest -> '.' 'id' varidnest2

varidnest2 -> '(' aParams ')' varidnest

varidnest2 -> rept-idnest10

localVarDecl -> 'local' varDecl

relExpr -> arithExpr relOp arithExpr

assignOp -> ':='

relOp -> '=='

relOp -> '<>'

relOp -> '<'

relOp -> '>'

relOp -> '<='

relOp -> '>='

multOp -> '\*'

multOp -> '/'

multOp -> 'and'

addOp -> '+'

addOp -> '-'

addOp -> 'or'

sign -> '+'

sign -> '-'

statBlock -> '{' rept-statBlock1 '}'

statBlock -> statement

rept-statBlock1 -> statement rept-statBlock1

rept-statBlock1 -> EPSILON

statBlock -> EPSILON

statement -> variable assignOp expr ';'

statement -> 'if' '(' relExpr ')' 'then' statBlock 'else' statBlock ';'

statement -> 'while' '(' relExpr ')' statBlock ';'

statement -> 'read' '(' variable ')' ';'

statement -> 'write' '(' expr ')' ';'

statement -> 'return' '(' expr ')' ';'

visibility -> 'public'

visibility -> 'private'

First and Follow Sets:

Attached in the file called First\_Follow Sets

Section 3:

I used the recursive descent approach. I made an unordered\_map for first and follow sets, where the name of the function is the key and it holds array of arrays of tokentype(the first and follow sets terminals). With the standard approach of recursive descent, the parser goes from function to function, searches the first and follow sets and returns the result which is true or false.

Section 4:

I have used the grammar tool provided by the professor. Although useful, it needed some changes in the input file to first run. I have also used atoCC to generate the first and follow sets for the grammar.