1. Transformations:

All {{E}} repetitions have been transformed into a new variable E’ that produces -> ε| E E’

Examples: RepFuncDef, RepClassDef

All [[E]] optionals have been transformed into a new variable E’ that produces ε | E

Examples: OptId, OptLocalVD

Newly added variable to remove ambiguity include: VoidOrType, OptIntNum

Resulting Transformed grammar:

START → Prog

Prog → RepClassDecl RepFuncDef 'main' FuncBody

RepFuncDef → ε

RepFuncDef → RepFuncDef FuncDef

RepClassDecl → ε

RepClassDecl → RepClassDecl ClassDecl

ClassDecl → 'class' 'id' OptInherits '{' RepMembers '}' ';'

OptInherits → 'inherits' 'id' RepExtraInherits

OptInherits → ε

RepExtraInherits → ε

RepExtraInherits → RepExtraInherits ',' 'id'

RepMembers → ε

RepMembers → RepMembers Visibility MemberDecl

Visibility → 'public'

Visibility → 'private'

MemberDecl → FuncDecl

MemberDecl → VarDecl

FuncDecl → 'id' '(' FParams ')' ':' VoidOrType ';'

FuncHead → OptId 'id' '(' FParams ')' ':' VoidOrType

OptId →'id' 'sr'

OptId → ε

FuncDef → FuncHead FuncBody

FuncBody → OptLocalVD 'do' RepStatement 'end'

OptLocalVD → 'local' RepVarDecl

OptLocalVD → ε

VarDecl → Type 'id' RepArraySize ';'

RepVarDecl → ε

RepVarDecl → RepVarDecl VarDecl

Statement → AssignStat ';'

Statement → 'if' '(' RelExpr ')' 'then' StatBlock 'else' StatBlock ';'

Statement → 'while' '(' RelExpr ')' StatBlock ';'

Statement → 'read' '(' Variable ')' ';'

Statement → 'write' '(' Expr ')' ';'

Statement → 'return' '(' Expr ')' ';'

Statement → FunctionCall ';'

RepStatement → ε

RepStatement → RepStatement Statement

AssignStat → Variable AssignOp Expr

StatBlock → 'do' RepStatement 'end'

StatBlock → Statement

StatBlock → ε

Expr → ArithExpr

Expr → RelExpr

RelExpr → ArithExpr RelOp ArithExpr

ArithExpr → ArithExpr AddOp Term

ArithExpr → Term

Sign → '+'

Sign → '-'

Term → Term MultOp Factor

Term → Factor

Factor → Variable

Factor → FunctionCall

Factor → 'intNum'

Factor → 'floatNum'

Factor → '(' ArithExpr ')'

Factor → 'not' Factor

Factor → Sign Factor

Variable → RepIdnest 'id' RepIndice

FunctionCall → RepIdnest 'id' '(' AParams ')'

Idnest → 'id' RepIndice '.'

Idnest → 'id' '(' AParams ')' '.'

RepIdnest → ε

RepIdnest → RepIdnest Idnest

Indice → '[' ArithExpr ']'

RepIndice → ε

RepIndice → RepIndice Indice

ArraySize → '[' OptIntNum ']'

RepArraySize → ε

RepArraySize → RepArraySize ArraySize

Type → 'integer'

Type → 'float'

Type → 'id'

FParams → Type 'id' RepArraySize RepFParamsTail

FParams → ε

AParams → Expr RepAParamsTail

AParams → ε

AParamsTail → ',' Expr

RepAParamsTail → ε

RepAParamsTail → RepAParamsTail AParamsTail

FParamsTail → ',' Type 'id' RepArraySize

RepFParamsTail → ε

RepFParamsTail → RepFParamsTail FParamsTail

AssignOp → '='

RelOp → 'eq'

RelOp → 'neq'

RelOp → 'lt'

RelOp → 'gt'

RelOp → 'leq'

RelOp → 'geq'

AddOp → '+'

AddOp → '-'

AddOp → 'or'

MultOp → '\*'

MultOp → '/'

MultOp → 'and'

VoidOrType → Type

VoidOrType → 'void'

OptIntNum → intnum

OptIntNum → ε

2. using mike.device.github.io/first-follow/ we obtains first and follow sets:

| # | First sets |
| --- | --- |
| START | 'main''class''id' |
| Prog | 'main''class''id' |
| RepFuncDef | ε'id' |
| RepClassDecl | ε'class' |
| ClassDecl | 'class' |
| OptInherits | 'inherits'ε |
| RepExtraInherits | ε',' |
| RepMembers | ε'public''private' |
| Visibility | 'public''private' |
| MemberDecl | 'id''integer''float' |
| FuncDecl | 'id' |
| FuncHead | 'id' |
| OptId | 'id'ε |
| FuncDef | 'id' |
| FuncBody | 'local''do' |
| OptLocalVD | 'local'ε |
| VarDecl | 'integer''float''id' |
| RepVarDecl | ε'integer''float''id' |
| Statement | 'if''while''read''write''return''id' |
| RepStatement | ε'if''while''read''write''return''id' |
| AssignStat | 'id' |
| StatBlock | 'do''if''while''read''write''return'ε'id' |
| Expr | 'intNum''floatNum''(''not''+''-''id' |
| RelExpr | 'intNum''floatNum''(''not''+''-''id' |
| ArithExpr | 'intNum''floatNum''(''not''+''-''id' |
| Sign | '+''-' |
| Term | 'intNum''floatNum''(''not''+''-''id' |
| Factor | 'intNum''floatNum''(''not''+''-''id' |
| Variable | 'id' |
| FunctionCall | 'id' |
| Idnest | 'id' |
| RepIdnest | ε'id' |
| Indice | '[' |
| RepIndice | ε'[' |
| ArraySize | '[' |
| RepArraySize | ε'[' |
| Type | 'integer''float''id' |
| FParams | 'integer''float''id'ε |
| AParams | ε'intNum''floatNum''(''not''+''-''id' |
| AParamsTail | ',' |
| RepAParamsTail | ε',' |
| FParamsTail | ',' |
| RepFParamsTail | ε',' |
| AssignOp | '=' |
| RelOp | 'eq''neq''lt''gt''leq''geq' |
| AddOp | '+''-''or' |
| MultOp | '\*''/''and' |
| VoidOrType | 'integer''float''id''void' |
| OptIntNum | intnumε |
| # | Follow sets |
| START | ┤ |
| Prog | ┤ |
| RepFuncDef | 'main''id' |
| RepClassDecl | 'id''main''class' |
| ClassDecl | 'id''main''class' |
| OptInherits | '{' |
| RepExtraInherits | '{'',' |
| RepMembers | '}''public''private' |
| Visibility | 'id''integer''float' |
| MemberDecl | '}''public''private' |
| FuncDecl | '}''public''private' |
| FuncHead | 'local''do' |
| OptId | 'id' |
| FuncDef | 'main''id' |
| FuncBody | ┤'main''id' |
| OptLocalVD | 'do' |
| VarDecl | '}''public''private''do''integer''float''id' |
| RepVarDecl | 'do''integer''float''id' |
| Statement | 'end''if''while''read''write''return''id''else'';' |
| RepStatement | 'end''if''while''read''write''return''id' |
| AssignStat | ';' |
| StatBlock | 'else'';' |
| Expr | ')'';'',' |
| RelExpr | ')'';'',' |
| ArithExpr | ')'';''eq''neq''lt''gt''leq''geq''+''-''or'']'',' |
| Sign | 'intNum''floatNum''(''not''+''-''id' |
| Term | ')'';''eq''neq''lt''gt''leq''geq''+''-''or''\*''/''and'']'',' |
| Factor | ')'';''eq''neq''lt''gt''leq''geq''+''-''or''\*''/''and'']'',' |
| Variable | ')''='';''eq''neq''lt''gt''leq''geq''+''-''or''\*''/''and'']'',' |
| FunctionCall | ';'')''eq''neq''lt''gt''leq''geq''+''-''or''\*''/''and'']'',' |
| Idnest | '['')''='';''eq''neq''lt''gt''leq''geq''+''-''or''\*''/''and''(''id'']'',' |
| RepIdnest | '['')''='';''eq''neq''lt''gt''leq''geq''+''-''or''\*''/''and''(''id'']'',' |
| Indice | ')''='';''eq''neq''lt''gt''leq''geq''+''-''or''\*''/''and''.''['']'',' |
| RepIndice | ')''='';''eq''neq''lt''gt''leq''geq''+''-''or''\*''/''and''.''['']'',' |
| ArraySize | ';''['','')' |
| RepArraySize | ';''['','')' |
| Type | 'id'';''local''do' |
| FParams | ')' |
| AParams | ')' |
| AParamsTail | ')'',' |
| RepAParamsTail | ')'',' |
| FParamsTail | ')'',' |
| RepFParamsTail | ')'',' |
| AssignOp | 'intNum''floatNum''(''not''+''-''id' |
| RelOp | 'intNum''floatNum''(''not''+''-''id' |
| AddOp | 'intNum''floatNum''(''not''+''-''id' |
| MultOp | 'intNum''floatNum''(''not''+''-''id' |
| VoidOrType | ';''local''do' |
| OptIntNum | ']' |

3.

Decided on using a Table-Driven parser, after discussion with TA during the first assignment demo.

Overall not sure if this was the best decision because it was exceptionally tedious, and quite hard to debug.

To do this I need to remove ambiguities from the given grammar, which I mostly accomplished by hand, afterwards I used the aforementioned to tool to generate first and follow sets for the grammar, the follow sets were used to fill the parse table with its productions and afterwards the follow set to complete the closure of productions.

To begin parsing my Lexical Tokens, I read the .outlextokens file and reconstructed the tokens and sent the batch of tokens through to the parse method.

The parse method compares the current productions stack to the current tokens that need to be parsed, if the production and token are present in the parse table the production is poped from the stack and the new productions is added (it is also added to the Parse Tree). If the production equals the current token of the stack, the stack is poped and the current token moved on to the next token (the parse tree is also updated to follow the path of the parse). This is done until no tokens are left or no productions are left.

For the creation of the parse tree, I used a tree data structure with tree nodes. Nodes were only added to the tree when a production was added, and when a terminal replace a production.

For the abstact syntax tree, a recursive function was used to trim the tree until there is only terminal symbols in the tree (not perfect solution).

4. Tools used include: mike.device.github.io/first-follow/

The course lecture slides, multiple youtube videos on creating a LL parsing table

And some C# libraries:

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text.RegularExpressions;