**Introduction:**

Below are the steps taken to convert this left recursive grammar to right recursive grammar. First and Follow sets are created at the end. The predictive parsing table is present in the separate excel file. Code for the algorithm is written in main.cpp file. 2 files tokens.txt and dec.txt are provided as sample token files.

Here is the sample output generated by the code.

A screenshot of a computer program

Description automatically generatedA computer screen shot of a code

Description automatically generated

**Original Grammer (# represents null)**

Function -> Type identifier (ArgList) CompoundStmt

ArgList -> Arg | ArgList, Arg

Arg -> Type identifier

Stmt -> WhileStmt | Expr; | CompoundStmt | Declaration | IfStmt | ;

Declaration -> Type identifier;

Type -> int | float

WhileStmt -> while (BoolExpr) Stmt

IfStmt -> if (BoolExpr) Stmt ElsePart

ElsePart -> else Stmt | #

CompoundStmt -> {StmtList}

StmtList -> StmtList Stmt | #

BoolExpr -> identifier Compare Mag

Compare -> == | < | > | <= | >= | <>

Expr -> identifier = Mag

Mag -> Mag + Term | Mag – Term | Term

Term -> Term \* Factor | Term / Factor | Factor

Factor -> (Expr) | identifier | number

* **No Cycles in the Grammer**
* **Removing Null Production from ElsePart**

Function -> Type identifier (ArgList) CompoundStmt

ArgList -> Arg | ArgList, Arg

Arg -> Type identifier

Stmt -> WhileStmt | Expr; | CompoundStmt | Declaration | IfStmt | ;

Declaration -> Type identifier;

Type -> int | float

WhileStmt -> while (BoolExpr) Stmt

IfStmt -> if (BoolExpr) Stmt ElsePart | if (BoolExpr) Stmt

ElsePart -> else Stmt

CompoundStmt -> {StmtList}

StmtList -> StmtList Stmt | #

BoolExpr -> identifier Compare Mag

Compare -> == | < | > | <= | >= | <>

Expr -> identifier = Mag

Mag -> Mag + Term | Mag – Term | Term

Term -> Term \* Factor | Term / Factor | Factor

Factor -> (Expr) | identifier | number

* **Removing Null Production from StmtList**

Function -> Type identifier (ArgList) CompoundStmt

ArgList -> Arg | ArgList, Arg

Arg -> Type identifier

Stmt -> WhileStmt | Expr; | CompoundStmt | Declaration | IfStmt | ;

Declaration -> Type identifier;

Type -> int | float

WhileStmt -> while (BoolExpr) Stmt

IfStmt -> if (BoolExpr) Stmt ElsePart | if (BoolExpr) Stmt

ElsePart -> else Stmt

CompoundStmt -> {StmtList} | {}

StmtList -> StmtList Stmt | Stmt

BoolExpr -> identifier Compare Mag

Compare -> == | < | > | <= | >= | <>

Expr -> identifier = Mag

Mag -> Mag + Term | Mag – Term | Term

Term -> Term \* Factor | Term / Factor | Factor

Factor -> (Expr) | identifier | number

* **No Indirect Left Recursion**
* **Removing Direct Left Recursion from StmtList**

Function -> Type identifier (ArgList) CompoundStmt

ArgList -> Arg | ArgList, Arg

Arg -> Type identifier

Stmt -> WhileStmt | Expr; | CompoundStmt | Declaration | IfStmt | ;

Declaration -> Type identifier;

Type -> int | float

WhileStmt -> while (BoolExpr) Stmt

IfStmt -> if (BoolExpr) Stmt ElsePart | if (BoolExpr) Stmt

ElsePart -> else Stmt

CompoundStmt -> {StmtList} | {}

StmtList -> Stmt StmtList’

StmtList’ -> Stmt StmtList’ | #

BoolExpr -> identifier Compare Mag

Compare -> == | < | > | <= | >= | <>

Expr -> identifier = Mag

Mag -> Mag + Term | Mag – Term | Term

Term -> Term \* Factor | Term / Factor | Factor

Factor -> (Expr) | identifier | number

* **Removing Direct Left Recursion from Mag, Term, ArgList**

Function -> Type identifier (ArgList) CompoundStmt

ArgList -> Arg ArgList’

ArgList’ -> , Arg ArgList’ | #

Arg -> Type identifier

Stmt -> WhileStmt | Expr; | CompoundStmt | Declaration | IfStmt | ;

Declaration -> Type identifier;

Type -> int | float

WhileStmt -> while (BoolExpr) Stmt

IfStmt -> if (BoolExpr) Stmt ElsePart | if (BoolExpr) Stmt

ElsePart -> else Stmt

CompoundStmt -> {StmtList} | {}

StmtList -> Stmt StmtList’

StmtList’ -> Stmt StmtList’ | #

BoolExpr -> identifier Compare Mag

Compare -> == | < | > | <= | >= | <>

Expr -> identifier = Mag

Mag -> Term Mag’

Mag’ -> + Term Mag’ | – Term Mag’ | #

Term -> Factor Term’

Term’ -> \* Factor Term’ | / Factor Term’ | #

Factor -> (Expr) | identifier | number

* **LL1 Grammer**

Function -> Type identifier (ArgList) CompoundStmt

ArgList -> Arg ArgList’

ArgList’ -> , Arg ArgList’ | #

Arg -> Type identifier

Stmt -> WhileStmt | Expr; | CompoundStmt | Declaration | IfStmt | ;

Declaration -> Type identifier;

Type -> int | float

WhileStmt -> while (BoolExpr) Stmt

IfStmt -> if (BoolExpr) Stmt Else

Else -> ElsePart | #

ElsePart -> else Stmt

CompoundStmt -> {CompoundStmt’

CompoundStmt’ -> StmtList} | }

StmtList -> Stmt StmtList’

StmtList’ -> Stmt StmtList’ | #

BoolExpr -> identifier Compare Mag

Compare -> == | <Compare’ | >Compare’’ | <Compare’ | >Compare’’ | <Compare’

Compare’ -> = | # | >

Compare’’ -> = | #

Expr -> identifier = Mag

Mag -> Term Mag’

Mag’ -> + Term Mag’ | – Term Mag’ | #

Term -> Factor Term’

Term’ -> \* Factor Term’ | / Factor Term’ | #

Factor -> (Expr) | identifier | number

**First Set**

first(Factor) -> (, identifier, number

first(Term’) -> \*, /, #

first(Term) -> first(Factor) = (, identifier, number

first(Mag’) -> +, -, #

first(Mag) -> first(Term) = (, identifier, number

first(Expr) -> identifier

first(Compare) -> =,<,>

first(Compare’) -> =,#,>

first(Compare’’) -> =,#

first(BoolExpr) -> identifier

first(StmtList’) -> first(Stmt), # = while, identifier, {, int, float, if, ;, #

first(StmtList) -> first(Stmt) = while, identifier, {, int, float, if, ;

first(CompoundStmt) -> {

first(CompoundStmt’) -> first(StmtList), } -> while, identifier, {, int, float, if, ;, }

first(ElsePart) -> else

first(Else) -> first(ElsePart), # = else, #

first(IfStmt) -> if

first(WhileStmt) -> while

first(Type) -> int, float

first(Declaration) -> first(Type) = int, float

first(Stmt) -> first(WhileStmt), first(Expr), first(CompoundStmt), first(Declaration), first(IfStmt), ;

-> while, identifier, {, int, float, if, ;

first(Arg) -> first(Type) = int, float

first(ArgList’) -> comma, #

first(ArgList) -> first(Arg) = int, float

first(Function) -> first(Type) = int, float

**Follow Set**

**Step 1:**

follow(Function) -> $

follow(ArgList) -> )

follow(ArgList’) -> follow(ArgList), follow(ArgList’) = )

follow(Arg) -> first(ArgList’), first(ArgList’)

-> comma, follow(ArgList), follow(ArgList’) = comma, )

follow(Stmt) -> follow(WhileStmt), first(Else), follow(ElsePart), first(StmtList’), first(StmtList’)

-> follow(WhileStmt), else, follow(IfStmt), follow(ElsePart), while, identifier, {, int, float, if, ;, follow(StmtList), follow(StmtList’)

follow(Declaration) -> follow(Stmt)

follow(Type) -> identifier

follow(WhileStmt) -> follow(Stmt)

follow(IfStmt) -> follow(Stmt)

follow(Else) -> follow(IfStmt)

follow(ElsePart) -> follow(Else)

follow(CompoundStmt) -> follow(Stmt)

follow(CompoundStmt’)-> follow(CompoundStmt)

follow(StmtList) -> }

follow(StmtList’) -> follow(StmtList), follow(StmtList’) = }

follow(BoolExpr) -> )

follow(Compare) -> first(Mag) = (, identifier, number

follow(Compare’) -> follow(Compare) = (, identifier, number

follow(Compare’’) -> follow(Compare) = (, identifier, number

follow(Expr) -> ;

follow(Mag) -> follow(BoolExpr), follow(Expr) = ), ;

follow(Mag’) -> follow(Mag), follow(Mag’) = ), ;

follow(Term) -> first(Mag’), first(Mag’), first(Mag’)

-> +, -, follow(Mag), follow(Mag’), follow(Mag’) = +, -, ), ;

follow(Term’) -> follow(Term), follow(Term’) = +, -, ), ;

follow(Factor) -> first(Term’), first(Term’), first(Term’)

-> \*, /, follow(Term), follow(Term’), follow(Term’) = \*, /, +, -, ), ;

**Step 2:**

follow(Function) -> $

follow(ArgList) -> )

follow(ArgList’) ->)

follow(Arg) -> comma, )

follow(Stmt) -> else , while, identifier, {, int, float, if, ;, }

follow(Declaration) -> else , while, identifier, {, int, float, if, ;, }

follow(Type) -> identifier

follow(WhileStmt) -> else , while, identifier, {, int, float, if, ;, }

follow(IfStmt) -> else , while, identifier, {, int, float, if, ;, }

follow(Else) -> else , while, identifier, {, int, float, if, ;, }

follow(ElsePart) -> else , while, identifier, {, int, float, if, ;, }

follow(CompoundStmt) -> else , while, identifier, {, int, float, if, ;, }

follow(CompoundStmt’)-> else , while, identifier, {, int, float, if, ;, }

follow(StmtList) -> }

follow(StmtList’) -> }

follow(BoolExpr) -> )

follow(Compare) -> (, identifier, number

follow(Compare’) -> (, identifier, number

follow(Compare’’) -> (, identifier, number

follow(Expr) -> ;

follow(Mag) -> ), ;

follow(Mag’) -> ), ;

follow(Term) -> +, -, ), ;

follow(Term’) -> +, -, ), ;

follow(Factor) -> \*, /, +, -, ), ;