Name: *Obaid-Ur-Rahman Siddiqui*

Reg. No: *L1S19BSCS0001*

Section: *(G1)*

Compiler Construction (CC)

Assignment # 2

Submitted to: Sir Muhammad Usman Afzal

Due Date: 12-06-2022

Question :-

onvert the following complete grammar into LL(1) predictive grammar :-

Function → Type identifier (Arglist) | CompoundStmt

ArgList → Arg | ArgList , Arg

Arg → Type identifier

Declaration → Type Identlist ;

Type → int | float

Identlist → identifier , Identlist | identifier

Stmt → whileStmt | Rvalue ; | IfStmt | Compound Stmt | Declaration | ;

WhileStmt → while (Rvalue) Stmt

IfStmt → if (Rvalue) Stmt ElsePart

ElsePart → else Stmt | ε

Compound Stmt → { StmtList }

StmtList → StmtList Stmt | ε

Rvalue → Rvalue Compare Mag | Mag

Compare → == | < | > | <= | >= | != | < >

Mag → Mag + Term | Mag - Term | Term

Term → Term * Factor | Term / Factor | Factor

Factor → (Mag) | identifier | number

Answer :-

There are 3 steps to convert the grammar into LL (1) predi grammar which are as follows :-

- Remove Ambiguity
- Remove Left recursion
- Left Factoring.

Step-01 : Remove Ambiguity.

There is no ambiguity in the given grammar so we move to step 02.

Step - 02 : Remove Left Recursion :

Function → Type identifier (ArgList) CompoundStmt

ArgList → Arg ArgList′

ArgList′ → , Arg ArgList′ | ε

Arg → Type identifier

Declaration → Type IdentList ;

Type → int | float

IdentList → identifier , IdentList | identifier

Stmt → WhileStmt | Rvalue ; | IfStmt | CompoundStmt | Declaration | ;

WhileStmt → while (Rvalue) Stmt

IfStmt → if (Rvalue) Stmt ElsePart

ElsePart → else Stmt | ε

CompoundStmt → { StmtList }

predictive

List → ε StmtList'

List' → Stmt StmtList' | ε

value → Mag Rvalue'

Rvalue' → Compare Mag Rvalue' | ε

Compare → == | < | > | <= | >= | != | <>

Mag → Term Mag'

Mag' → + Term Mag' | - Term Mag' | ε

Term → Factor Term'

Term' → * Factor Term' | / Factor Term' | ε

Factor → (Mag) | identifier | numler

→ Minimal Grammar

Step-03 : Left Factoring

Function → Type identifier (ArgList) Compound Stmt

Arglist → Arg Arglist'

Arglist' → , Arg Arglist' | ε

Arg → Type identifier

Declaration → Type IdentList ;

Type → int | float

Identlist → identifier IdentList'

Identlist' → , Identlist | ε

Stmt → WhileStmt | Rvalue ; | IfStmt | CompoundStmt | Declaration | ;

WhileStmt → while (Rvalue) Stmt

IfStmt → if (Rvalue) Stmt ElsePart

ElsePart → else Stmt | ε

Compound Stmt → { Stmt List }

Stmt List → ε StmtList′

StmtList′ → Stmt StmtList′ | ε

Rvalue → Mag Rvalue′

Rvalue′ → Compare Mag Rvalue′ | ε

Compare → ==|<|>|<=|>=|!=|<>

Mag → Term Mag′

Mag′ → + Term Mag′ | - Term Mag′ | ε

Term → Factor Term′

Term′ → * Factor Term′ | / Factor Term′ | ε

Factor → (Mag) | identifier | number

→ Minimal Grammar

Therefore, the given grammar has been converted to LL(1) predictive Grammar.

☆ Implementation Part ☆

The corrected LL(1) predictive minimal grammar is given as follows :-

1. Mag → Term Mag′
2. Mag′ → + Term Mag′
3. Mag′ → - Term Mag′
4. Mag′ → ε
5. Term → Factor Term′
6. Term′ → * Factor Term′

m′ → / Factor Term′

Term′ → ε

9. Factor → (Mag)

10. Factor → identifier

11. Factor → number

We first find the FIRST and FOLLOW sets of all Variables :-

Variable	FIRST	FOLLOW
Mag	{ (, identifier , number }	{ $,) }
Mag′	{ +, -, ε }	{ $,) }
Term	{ (, identifier , number }	{ +, -, $,) }
Term′	{ *, /, ε }	{ +, -, $,) }
Factor	{ (, identifier, number }	{ *, /, +, -, $,) }

Now we create and Fill the LL(1) parsing Table as follows :-

Variable	identifier	number	+	-	*	/	(	)	$
Mag	1	1	skip	skip	skip	skip	1	PoP	PoP
Mag′	skip	skip	2	3	skip	skip	skip	4	4
Term	5	5	PoP	PoP	skip	skip	5	PoP	PoP
Term′	skip	skip	8	8	6	7	skip	8	8
Factor	10	11	PoP	PoP	PoP	PoP	9	PoP	PoP