**RED->Corrected Grammar BLUE->First Set E->EMPTY**

**Orignal**

**3.1 PLATYPUS Program**

<program> ->

PLATYPUS { <statements> } | PLATYPUS { }

<program> ->

PLATYPUS {<opt\_statements>}

FIRST(program) = { KW\_T PLATYPUS}

<opt\_statements> ->

Empty | <statements>

FIRST (<opt\_statements>) -> {AVID, SVID, IF, WHILE, READ, WRITE, E}

<statements> ->

<statement> | <statements> <statement>

<statements> ->

<statement><statements>’

<statements>’ -> <statement><statements>’

FIRST (statements) = {AVID, SVID, IF, WHILE, READ, WRITE}

FIRST (<statements>’) = {AVID, SVID, IF, WHILE, READ, WRITE, E}

<statement> ->

<assignment statement>

| <selection statement>

| <iteration statement>

| <input statement>

| <output statement>

FIRST (statement)={ AVID\_T, SVID\_T, IF, While, READ, WRITE }

<assignment statement> ->

<assignment expression>;

FIRST(assignment statement)-> { AVID\_T, SVID\_T }

< assignment expression> ->

AVID = <arithmetic expression>

| SVID = <string expression>

FIRST (assignment expression) -> {AVID\_T, SVID\_T}

<selection statement> ->

IF TRUE (<conditional expression>) THEN {<opt statements>}

ELSE {<opt statements>};

FIRST (selection statement) -> { KW\_T(IF) }

<iteration statement> ->

WHILE **<**pre-condition> **(<**conditional expression>**)**

REPEAT **{**<statements*>***};**

FIRST (iteration statement)-> {WHILE}

**<**pre-condition> ->

TRUE | FALSE

FIRST (pre-condition) -> { TRUE , FALSE}

**3.2.4 Input Statement**

<input statement> ->

READ (<variable list>);

FIRST (input statement) -> {READ}

<variable list> ->

<variable identifier> | <variable list>,<variable identifier>

<variable list> ->

<variable identifier><variable list>’

FIRST(variable list) -> { AVID\_T, SVID\_T }

<variable list>’-> ,<variable identifier> <variable list>’ | <Empty>

FIRST (<variable list>’) -> { , , E}

FIRST (variable identifier) -> { AVID\_T, SVID\_T }

**3.2.5 Output Statement**

<output statement> ->

WRITE (<output list>);

FIRST (output statement) -> {WRITE}

< output list> ->

<*variable list>* | STR\_T | E

FIRST (output list) -> { AVID\_T, SVID\_T, STR\_T , E }

**3.3.1 Arithmetic Expression**

<arithmetic expression> - >

<unary arithmetic expression>

| <additive arithmetic expression>

FIRST (arithmetic expression) -> {+ , - , AVID\_T , FPL\_T , INL\_T , ( }

<unary arithmetic expression> ->

- <primary arithmetic expression>

| + <primary arithmetic expression>

FIRST (unary arithmetic expression) -> { + , - }

<additive arithmetic expression> ->

<additive arithmetic expression> + <multiplicative arithmetic expression>

| <additive arithmetic expression> - <multiplicative arithmetic expression>

| <multiplicative arithmetic expression>

<multiplicative arithmetic expression> ->

<multiplicative arithmetic expression> \* <primary arithmetic expression>

| <multiplicative arithmetic expression> / <primary arithmetic expression>

| <primary arithmetic expression>

<additive arithmetic expression> ->

<multiplicative arithmetic expression> | <additive arithmetic expression>’

FIRST (additive arithmetic expression) -> { AVID\_T , FPL\_T , INL\_T , ( }

<additive arithmetic expression>’

+<multiplicative arithmetic expression> <additive arithmetic expression>’

| -<multiplicative arithmetic expression> <additive arithmetic expression>’

| E

FIRST (<additive arithmetic expression>’) -> { +, - , E }

<multiplicative arithmetic expression> ->

<primary arithmetic expression><multiplicative arithmetic expression>’

FIRST (multiplicative arithmetic expression) -> (AVID\_T, FPL\_T , INL\_T , ( )

<multiplicative arithmetic expression>’->

\* <primary arithmetic expression><multiplicative arithmetic expression>’

| / <primary arithmetic expression><multiplicative arithmetic expression>’

| E

FIRST (<multiplicative arithmetic expression>’) -> {\* , / , E}

<primary arithmetic expression> ->

AVID\_T

| FPL\_T

| INL\_T

| (<arithmetic expression>)

FIRST (primary arithmetic expression) -> (AVID\_T, FPL\_T , INL\_T , ( )

**3.3.2 String Expression**

<string expression> ->

<primary string expression>

| <string expression> # <primary string expression>

<string expression> ->

<primary string expression> | <string expression>’

FIRST (string expression) -> { SVID\_T, STR\_T }

<string expression>’ ->

# <primary string expression><string expression>’

FIRST (<string expression>’) -> {#, E}

<primary string expression> ->

SVID\_T

| STR\_T

FIRST(primary string expression)-> { SVID\_T , STR\_T }

**3.3.3 Conditional Expression**

<conditional expression> ->

<logical OR expression>

<logical OR expression> ->

<logical AND expression><logical OR expression>’

FIRST (<logical OR expression>) -> { AVID\_T , FPL\_T , INL\_T , SVID\_T, STR\_T }

<logical OR expression>’->

.OR. <logical AND expression><logical OR expression>’

FIRST (<logical OR expression>’) -> { .OR. , E }

<logical AND expression> ->

<relational expression><logical AND expression>’

FIRST (logical AND expression>) -> { AVID\_T , FPL\_T , INL\_T , SVID\_T, STR\_T }

<logical AND expression>’ ->

.AND. <relational expression><logical AND expression>’

FIRST (logical AND expression>’) -> { .AND. , E }

**3.3.4 Relational Expression**

<relational expression> ->

<primary a\_relational expression> <operators> <primary a\_relational expression>

| <primary s\_relational expression> <operators> <primary s\_relational expression>

FIRST (relational expression >) -> { AVID\_T , FPL\_T , INL\_T , SVID\_T, STR\_T }

<operators> ->

== | <> | > | <

<primary a\_relational expression> ->

AVID\_T | FPL\_T | INL\_T

FIRST (primary a\_relational expression >) ->{ AVID\_T | FPL\_T | INL\_T }

<primary s\_relational expression> ->

<primary string expression>

FIRST (<primary s\_relational expression >) -> { SVID\_T, STR\_T }