**Platypus Program**

<program> -> PLATYPUS {<opt\_statements>}

<statements> -> <statement> | <statements> <statement>

<statements> -> <statements> <statement> | <statement> //rearranged grammar

<statements> -> <statement> <statements’> // left recursion eliminated

<statements’>-> <statement> <statements’> | ε

**Statements**

<statement> ->

<assignment statement>

| <selection statement>

| <iteration statement>

| <input statement>

| <output statement>

**Assignment Statement**

<assignment statement> -> <assignment expression>;

<assignment expression> -> AVID = <arithmetic expression> | SVID = <string expression>

**Selection Statement**

<selection statement>-> IF <pre-condition> (<conditional expression>) THEN { <opt\_statements> }

ELSE { < opt\_statements> }

**Iteration Statement**

<iteration statement> -> WHILE < pre-condition> (<conditional expression>)

REPEAT { <statements> };

<pre-condition> -> TRUE | FALSE

**Input Statement**

<input statement> -> READ (<variable list>);

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

<variable list> -> <variable list>, <variable identifier> | <variable identifier> // rearranged grammar

<variable list> -> <variable identifier> <variable list’> // left recursion eliminated

<variable list’> -> <variable identifier> <variable list’> |ε

**Output Statement**

<output statement > -> WRITE (< opt\_variable list>); | WRITE (STR\_T);

<output statement > -> WRITE (<output list>); // reworked grammar

<output list> -> <opt\_variable list> | STR\_T

**Expressions**

**Arithmetic Expression**

<arithmetic expression> - >

<unary arithmetic expression>

| <additive arithmetic expression>

<unary arithmetic expression> ->

- <primary arithmetic expression>

| + <primary arithmetic expression>

<additive arithmetic expression> ->

<additive arithmetic expression> + <multiplicative arithmetic expression>

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

| <multiplicative arithmetic expression>

<additive arithmetic expression> -> // left recursion eliminated

<multiplicative arithmetic expression> <additive arithmetic expression’>

<additive arithmetic expression’>->

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

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

|ε

<multiplicative arithmetic expression> ->

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

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

| <primary arithmetic expression>

<multiplicative arithmetic expression> -> // left recursion eliminated

<primary arithmetic expression> <multiplicative arithmetic expression’>

<multiplicative arithmetic expression’> ->

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

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

|ε

<primary arithmetic expression> ->

AVID\_T

| FPL\_T

| INL\_T

| (<arithmetic expression>)

**String Expression**

<string expression> ->

<primary string expression>

| <string expression> # <primary string expression>

<string expression> -> //rearranged grammar

<string expression> # <primary string expression>

|<primary string expression>

<string expression> -> //left recursion eliminated

<primary string expression> <string expression’>

<string expression’>->

# <primary string expression><string expression’> |ε

<primary string expression> ->

SVID\_T

| STR\_T

**Conditional Expression**

<conditional expression> ->

<logical OR expression>

<logical OR expression> ->

<logical AND expression>

| <logical OR expression> .OR. <logical AND expression>

<logical OR expression> -> //rearranged grammar

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

| <logical AND expression>

// left recursion eliminated

<logical OR expression> -> <logical AND expression> <logical OR expression’>

<logical OR expression’> -> .OR. <logical AND expression><logical OR expression’> |ε

<logical AND expression> ->

<relational expression>

| <logical AND expression> .AND. <relational expression>

<logical AND expression> -> //rearranged grammar

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

|<relational expression>

<logical AND expression> -> <relational expression><logical AND expression’>

<logical AND expression’>->.AND. <relational expression><logical AND expression’>|ε

**Relational Expression**

<relational expression> ->

<primary a\_relational expression> == <primary a\_relational expression>

| <primary a\_relational expression> <> <primary a\_relational expression>

| <primary a\_relational expression> > <primary a\_relational expression>

| <primary a\_relational expression> < <primary a\_relational expression>

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

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

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

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

// apply left factoring

<relational expression>-> < primary a\_relational expression> < primary a\_relational expression‘>

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

< primary a\_relational expression‘>->

== <primary a\_relational expression>

<> <primary a\_relational expression>

> <primary a\_relational expression>

< <primary a\_relational expression>

< primary s\_relational expression‘>->

== <primary s\_relational expression>

<> <primary s\_relational expression>

> <primary s\_relational expression>

< <primary s\_relational expression>

<primary a\_relational expression> ->

AVID\_T

| FPL\_T

| INL\_T

<primary s\_relational expression> ->

<primary string expression>