Grammar

//first set = PLATYPUS

<program> ->

PLATYPUS { opt_statements }

//first set = first(statements)

<opt_statements> ->

<statements> | e

//first set = first(statement), e

<statements>

<statements'><statement>

//first set = first(statement), e

<statements'>

<statement><statements'> | e

//first set = first(assignment statement), first(selection statement), first(iteration statement), first(input statement), first(output statement)

<statement>->

<assignment statement> | <selection statement > | <iteration statement> | <input statement> | <output statement>

//first set = first(assignment expression)

<assignment statement>->

<assignment expression> ;

//first set = AVID_T, SVID_T

<assignment expression>->

AVID_T = <arithmetic expression> | SVID_T = <string expression>

//first set = IF

<selection statement>->

IF <pre-condition> (<conditional expression>) THEN { opt_statements } ELSE { opt_statements };

//first set = WHILE

<iteration statement>->

WHILE <pre-condition> (<conditional_expression>) REPEAT { <statements> } ;

//first set = TRUE, FALSE

<pre-condition>->

TRUE | FALSE

//first set = READ

<input statement>->

READ (<variable list>) ;

<variable list>->

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

Convert to LL

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

Remove Left recursion

//first set = first(variable identifier)

<variable list>->

<variable identifier><variable list prime>

//first set = , e

<variable list prime>->

, <variable identifier><variable list prime> | e

//first set = AVID_T, SVID_T

<variable identifier> ->

AVID_T | SVID_T

//first set = WRITE

<output statement>->

WRITE (<opt_variable list>) ; | WRITE (STR_T) ;

Condense to one path

WRITE (<opt_variable list>) ;

//first set = first(variable list), e

<opt_variable list>->

<variable list> | e

//first set = first(unary…), first(additive…)

<arithmetic expression>->

<unary arithmetic expression> | <additive arithmetic expression>

//first set = -, +

<unary arithmetic expression>

-<primary arithmetic expression> | + <primary arithmetic expression>

Condense into one path

<unary operators><primary arithmetic expression>

//first set = -, +

<unary operator>

+ | -

<additive arithmetic expression>->

<additive arithmertic expression > + <multiplicative arithmetic expression>

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

| <multiplicative arithmetic expression>

Remove Left recursion

//first set = first(multiplicative arithmetic expression)

< additive arithmetic expression>->

<multiplicative arithmetic expression><additive arithmetic expression prime>

//first set = first(unary operator), e

<additive arithmetic expression prime>->

<unary operator><multiplicative arithmetic expression><additive arithmetic expression prime>
| e

<multiplicative arithmetic expression>->

<multiplicative arithmetic expression> * <primary arithmetic expression>

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

| <primary arithmetic expression>

Remove left recursion

//first set = first(primary arithmetic expression)

<multiplicative arithmetic expression>->

<primary arithmetic expression><multiplicative arithmetic expression prime>

//first set = first(mult operators), e

<multiplicative arithmetic expression prime>->

<mult operators><primary arithmetic expression><multiplicative arithmetic expression prime>

| e

//first set = / , *

<mult operators>

/ | *

//first set = AVID_T, FPL_T, INL_T, (

<primary arithmetic expression>->

AVID_T | FPL_T | INL_T | (<arithmetic expression>)

<string expression>->

<primary string expression> | <string expression > # <primary string expression>

Convert to LL

<string expression > # <primary string expression> | <primary string expression>

Remove Left Recursion

//first set = first(primary string expression)

<string expression>->

<primary string expression><string expression prime>

//first set = #, e

<string expression prime>->

<primary string expression ><string expression prime> | e

//first set = SVID_T, STR_T

<primary string expression>->

SVID_T | STR_T

//first set = first(logical or expression)

<conditional expression>->

<logical or expression>

<logical or expression>->

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

Change to LL

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

Remove Recursion

//first set = first(logical AND expression)

<logical OR expression>->

<logical AND expression><logical or expression prime>

//first set = .OR. , e

<logical or expression prime>->

.OR. <logical AND expression><logical or expression prime> | e

<logical and expression>->

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

Change to LL

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

Remove Left Recursion

//first set = first(relational expression)

<logical AND expression>

<relational expression><logical AND expression prime>

//first set = .AND. , e

<logical AND expression prime>

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

<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>

Condense statements

//first set = ==, <>, >, <

<relational operator>->

== | <> | > | <

//first set = AVID_T, FPL_T, INL_T

<primary a_relational expression>->

AVID_T | FPL_T | INL_T

//first set = first(primary string expression)

<primary a_relational expression>->

<primary string expression>

//first set = first(primary a_relational expression), first(primary s_relational expression)

<relational expression>->

<primary a_relational expression><relational operator><primary a_relational expression>

| <primary s_relational expression><relational operator><primary s_relational expression>