**A4 – Parser – LR -> LL, Left Factored, And Left Recursion Eliminated Grammar**

**1.1 – PLATYPUS Program**

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

<opt\_statements> -> <statements> |

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

**LR -> LL**

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

Removal of Left Recursion

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

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

**FIRST Construction**

FIRST(<program>) = {PLATYPUS}

FIRST(PLATYPUS) = {PLATYPUS}

FIRST(opt\_statements) = {FIRST(statements),}

= {AVID,SVID,IF,USING,INPUT,OUTPUT,}

FIRST(statements) = {FIRST(statement)}

= {AVID,SVID,IF,USING,INPUT,OUTPUT}

FIRST(statements’) = {FIRST(<statement>),}

= {AVID,SVID,IF,USING,INPUT,OUTPUT,}

FIRST(<statement>) = {AVID,SVID,IF,USING,INPUT,OUTPUT}

**1.2 – Statements**

- <statement> -> <assignment statement>

|<selection statement>

|<iteration statement>

|<input statement>

|<output statement>

**FIRST Construction**

FIRST(<statement>) = {FIRST(<assignment statement>),FIRST(<selection statement>),

FIRST(<iteration statement>),FIRST(<input statement>),

FIRST(<output statement>)}

= {AVID,SVID,IF,USING,INPUT,OUTPUT}

**1.2.1 – Assignment Statement**

- <assignment statement> -> <assignment expression>;

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

**FIRST Construction**

FIRST(<assignment statement>) = {AVID,SVID}

FIRST(<assignment expression>) = {AVID,SVID}

**1.2.2 – Selection Statement**

- <selection statement> -> IF (<conditional expression>) THEN <opt\_statements>

ELSE {<opt\_statements>};

FIRST(<selection statement>) = {IF}

**1.2.3 – Iteration Statement**

- <iteration statement> -> USING (<assignment expression>, <conditional expression>,

<assignment expression>)

REPEAT { <opt\_statements> };

**FIRST Construction**

FIRST(<iteration statement>) = {USING}

**1.2.4 – Input Statement**

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

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

**LR -> LL**

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

**Removal of Left Recursion**

<variable list> -> <variable identifier> <variable list’>

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

<variable identifier> -> AVID\_T | SVID\_T

**FIRST Construction**

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

FIRST(<variable list’>) = {, ,FIRST(<variable identifier>),}

= {, ,}

FIRST(<variable list>) = {FIRST(<variable identifier>)} = {AVID\_T,SVID\_T}

FIRST(<input statement>) = {INPUT}

**1.2.5 – Output Statement**

- <output statement> -> OUTPUT (<opt\_variable list>); | OUTPUT (<string literal>);

**Left Factoring**

<output statement> -> OUTPUT (<output list>);

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

<opt\_variable list> -> <variable list> |

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

<variable list> -> <variable identifier> <variable list’>

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

<variable identifier> -> AVID\_T | SVID\_T

<string literal> -> STR\_T

**FIRST Construction**

FIRST(<string literal>) = {STR\_T}

FIRST(<variable identifier>) = {AVID\_T | SVID\_T}

FIRST(<variable list’>) = {, ,}

FIRST(<variable list>) = {FIRST(<variable identifier>)} = {AVID\_T,SVID\_T}

FIRST(<opt\_variable list>) = {FIRST(<variable list>),} = {AVID\_T,SVID\_T,}

FIRST(<output list>) = {FIRST(<opt\_variable list>), STR\_T} = {AVID\_T,SVID\_T,,STR\_T}

FIRST(<output statement>) = {OUTPUT}

**2 – Expressions**

**2.1 – Arithmetic Expressions**

<arithmetic expression> -> <unary arithmetic expression> | <additive arithmetic expression>

<unary arithmetic expression> -> - <primary arithmetic expression>

| + <primary arithmetic expression>

**Left Factoring**

<unary arithmetic expression> -> <additive operators> <primary arithmetic expression>

<additive operators> -> + | -

<additive arithmetic expression> ->

<additive arithmetic expression> + <multiplicative arithmetic expression>

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

| <multiplicative arithmetic expression>

**Removal of Left Recursion**

<additive arithmetic expression> ->

<multiplicative arithmetic expression> <additive arithmetic expression’>

<additive arithmetic expression’> ->

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

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

|

**Left Factoring**

<additive arithmetic expression’> ->

<additive operators><multiplicative arithmetic expression> <additive arithmetic expression’>

|

\_\_\_\_\_\_\_\_\_\_\_­\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

<multiplicative arithmetic expression> ->

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

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

| <primary arithmetic expression>

**Removal of Left Recursion**

<multiplicative arithmetic expression> ->

<primary arithmetic expression> <multiplicative arithmetic expression’>

<multiplicative arithmetic expression’> ->

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

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

|

**Left Factoring**

<multiplicative arithmetic expression’> ->

<multiplicative operators> <primary arithmetic expression>

<multiplicative arithmetic expression’> |

<multiplicative operators> -> \* | /

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

<primary arithmetic expression> -> AVID\_T | FPL\_T | INL\_T | (<arithmetic expression>)

**FIRST Construction**

FIRST(<primary arithmetic expression>) = {AVID\_T,FPL\_T,INL\_T,(}

FIRST(<multiplicative arithmetic expression>) = {FIRST(<primary arithmetic expression>)}

= {AVID\_T,FPL\_T,INL\_T,( }

FIRST(<multiplicative operators>) = {\*,/}

FIRST(<multiplicative arithmetic expression’>) = {FIRST(<multiplicative operators>),}

= {\*,/,}

FIRST(<additive arithmetic expression>) = {FIRST(<multiplicative arithmetic expression>)

= {AVID\_T, FPL\_T, INL\_T, (}

FIRST(<additive operators>) = {+,-}

FIRST(<additive arithmetic expression’>) = {FIRST(<additive operators>),}

= {+,-,}

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

FIRST(<arithmetic expression>)

= {FIRST(<unary arithmetic expression>),FIRST(<additive arithmetic expression>)}

= {AVID\_T,FPL\_T,INL\_T, (, +, - }

**2.2 – String Expressions**

<string expression> ->

<primary string expression>

|<string expression> # <primary string expression>

**LR -> LL**

<string expression> ->

<string expression> # <primary string expression>

|<primary string expression>

**Removal of Left Recursion**

<string expression> -> <primary string expression> <string expression’>

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

<primary string expression> -> SVID\_T | STR\_T

**FIRST Construction**

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

FIRST(<string expression’>) = {#,}

FIRST(<string expression>) = {FIRST(<primary string expression>),FIRST(<string expression’>)}

= {SVID\_T,STR\_T}

**2.3 – Conditional Expressions**

<conditional expression> -> <logical OR expression>

<logical OR expression> ->

<logical AND expression>

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

**LR -> LL**

<logical OR expression> ->

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

|<logical AND expression>

**Removal of Left Recursion**

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

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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

<logical AND expression> ->

<relational expression>

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

**LR -> LL**

<logical AND expression> ->

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

|<relational expression>

**Removal of Left Recursion**

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

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

**FIRST Construction**

FIRST(<logical AND expression’>) = {.AND.,}

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

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

FIRST(<logical AND expression>)

= {FIRST(<relational expression>)}

= {AVID\_T,FPL\_T,INL\_T,SVID\_T,STR\_T}

FIRST(<logical OR expression>) =

{FIRST(<logical AND expression>)}

= {AVID\_T,FPL\_T,INL\_T,SVID\_T,STR\_T}

FIRST(<conditional expression>) = {FIRST(<logical OR expression>)}

= {AVID\_T,FPL\_T,INL\_T,SVID\_T,STR\_T}

**2.4 – Relational Expressions**

<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>

**Left Factoring**

<relational expression> ->

<primary a\_relational expression> <rel\_ops> <primary a\_relational expression>

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

<rel\_ops> -> == | <> | > | <

<primary a\_relational expression> -> AVID\_T | FPL\_T | INL\_T

<primary s\_relational expression> -> <primary string expression>

**FIRST Construction**

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

FIRST(<primary s\_relational expression>) = {FIRST(<primary string expression>)}

= {SVID\_T,STR\_T}

FIRST(<primary a\_relational expression>) = {AVID\_T,FPL\_T,INL\_T}

FIRST(<relational expression>) = {FIRST(<primary a\_relational expression>),

FIRST(<primary s\_relational expression>)}

= {AVID\_T,FPL\_T,INL\_T,SVID\_T,STR\_T}