**PLATYPUS Compiler - Parser**

**3 The PLATYPUS Syntactic Specification**

**3.1 PLATYPUS Program**

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

FIRST(<program>) = {KW\_T(PLATYPUS)}

<opt\_statements> -> <statements> | ε

FIRST(<opt\_statements>) = {KW\_T(IF), KW\_T(WHILE), KW\_T(READ), KW\_T(WRITE), AVID\_T, SVID\_T, and ε}

**3.2 Statements**

<statements> ->

<statements><statement> |

<statement>

***[Recursion Elimination]***

<statements> ->

<statement><statements\_p>

<statements\_p>->

<statement><statements\_p> |

ε

<statement> ->

<assignment statement> |

<selection statement> |

<iteration statement> |

<input statement> |

<output statement>

FIRST(<statements>) = {<statement>, <statements\_p>} = {AVID\_T, SVID\_T, KW\_T(IF), KW\_T(WHILE), KW\_T(READ), KW\_T(WRITE), ε}

FIRST(<statements\_p>)= {<statement>, ε} = {AVID\_T, SVID\_T, KW\_T(IF), KW\_T(WHILE), KW\_T(READ), KW\_T(WRITE), ε}

FIRST(<statement>) = {AVID\_T, SVID\_T, KW\_T(IF), KW\_T(WHILE), KW\_T(READ), KW\_T(WRITE)}

* + 1. **Assignment Statement**

<assignment statement> ->

<assignment expression>;

< assignment expression> ->

AVID = <arithmetic expression>

| SVID = <string expression>

FIRST(<assignment statement>) = {<assignment expression>}

= {AVID\_T, SVID\_T}

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

**3.2.2 Selection Statement( the if statement)**

<selection statement> ->

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

ELSE { <opt\_statements> } ;

FIRST(<selection statement>) = KW\_T(IF)

**3.2.3 Iteration Statement (the loop statement)**

<iteration statement> ->

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

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

**<**pre-condition> ->

TRUE | FALSE

FIRST(<iteration statement>) = KW\_T(WHILE)

**3.2.4 Input Statement**

<input statement> ->

READ (<variable list>);

FIRST(<input statement>) -> KW\_T(READ)

<variable list> ->

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

***[Recursion Elimination]***

<variable list> -> <variable identifier><variable list\_p>

<variable list\_p>->

,<variable identifier><variable list\_p> |

ε

<variable identifier> ->

AVID\_T |

SVID\_T

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

FIRST(<variable list\_p>) = {AVID\_T, SVID\_T,COM\_T , ε}

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

**3.2.5 Output Statement**

<output statement> ->

WRITE(<opt\_variable list>); |

WRITE(STR\_T);

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

<output list> -> <variable list> | STR\_T | ε

FIRST(<output statement>) = KW\_T(WRITE)

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

**3.3 Expressions**

**3.3.1 Arithmetic Expression**

<arithmetic expression> - >

<unary arithmetic expression>

| <additive arithmetic expression>

FIRST(<arithmetic expression) ->

{ FIRST(<unary arithmetic expression>) |

FIRST(<additive arithmetic expression>) }

= {+,-, ε, 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>

***[Recusion Elimination]***

<additive arithmetic expression> ->

<multiplicative arithmetic expression><additive arithmetic expression\_p>

<additive arithmetic expression\_p> ->

+ <multiplicative arithmetic expression><additive arithmetic expression\_p> |

* <multiplicative arithmetic expression><additive arithmetic expression\_p> |

ε

FIRST(<additive arithmetic expression>) ->

{ FIRST(<additive arithmetic expression\_p>, AVID\_T, FPL\_T, INL\_T, (}

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

FIRST(<additive arithmetic expression\_p>) -> {+,-, ε}

<multiplicative arithmetic expression> ->

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

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

| <primary arithmetic expression>

***[Recusion Elimination]***

<multiplicative arithmetic expression> ->

<primary arithmetic expression><multiplicative arithmetic expression\_p>

FIRST(<multiplicative arithmetic expression>) ->

{ FIRST(<primary arithmetic expression>),

FIRST(<multiplicative arithmetic expression\_p>)}

= {AVID\_T, FPL\_T, INL\_T, (, \*, /, ε}

FIRST(<multiplicative arithmetic expression\_p>) -> {\*,/,ε}

<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>

***[Recursion Elimination]***

<string expression> ->

<primary string expression><string expression\_p>

<string expression\_p> ->

<< <primary string expression><string expression\_p> |

ε

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

FIRST(<string expression\_p> -> {<<, ε }

<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>

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

<logical OR expression> ->

<logical AND expression>

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

***[Recursion Elimination]***

<logical OR expression> ->

<logical AND expression><logical OR expression\_p>

<logical OR expression\_p> ->

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

ε

<logical AND expression> ->

<relational expression>

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

***[Recursion Elimination]***

<logical AND expression> ->

<relational expression><logical AND expression\_p>

<logical AND expression\_p> ->

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

ε

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

FIRST(<logical OR expression\_p)> -> {.OR., ε }

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

FIRST(<logical AND expression\_p>) -> {.AND., ε }

**3.3.4 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>

***[Recursion Elimination]***

<relational expression> ->

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

<primary s\_relational expression><primary s\_relational expression\_p>

<primary a\_relational expression\_p> ->

== <primary a\_relational expression> |

<> <primary a\_relational expression> |

> <primary a\_relational expression> |

< <primary a\_relational expression>

<primary s\_relational expression\_p> ->

== <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>

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

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

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

FIRST(<primary a\_relational expression\_p) -> {==, <>, >, <}

FIRST(<primary s\_relational expression\_p) -> {==, <>, >, <}