PLATYPUS Syntactic Specification

Program:

<program> → PROGRAM {<opt\_statements>}

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

Optional Statements:

<opt\_statements> → <statements> | e

FIRST(<opt\_statements>) = {AVID\_T, SVID\_T, KW\_T(IF), KW\_T(WHILE), KW\_T(INPUT), KW\_T(OUTPUT),∈ }

Statements:

< statements > → <statement> <statementPrime>

<statementPrime> → <statements> <statementPrime> | e

FIRST (<statementPrime>) = {AVID\_T, SVID\_T, KW\_T(IF), KW\_T(WHILE), KW\_T(INPUT), KW\_T(OUTPUT),∈ }

Statement:

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

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

Assignment Statement:

<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:

<selection statement> → if <pre-condition> (<conditional expression>) THEN {<opt\_statements>} ELSE {<opt\_statements>};

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

Iteration Statement:

<iteration statement> → WHILE <pre-condition> (<conditional expression>) DO {<statements>}

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

<pre-condition> → TRUE | FALSE

FIRST (<pre-condition>) = {KW\_T(TRUE), KW\_T(FALSE)}

Input Statement:

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

Variable List:

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

Fixed LR

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

<variable list> → <variable identifier> <variable list prime>

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

<variable list Prime> → <variable identifier> <variable list Prime> | e

FIRST (<variable list prime >) = {COM\_T, e}

<variable identifier> → AVID | SVID

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

Output Statement:

<output statement> → OUTPUT (<opt\_statements>); | OUTPUT (STR\_T);

Fixed LR

<output statement> → OUTPUT (<output statementPrime>);

FIRST (<output statement >) = {KW\_T(OUTPUT)}

<output statement Prime> → <opt\_variable list> | STR\_T

FIRST (<output statement Prime>) = {AVID\_T, SVID\_T, STR\_T}

<opt\_variable list> → <variable list> | e

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

Arithmetic Expression:

<arithmetic expression> → <unary arithmetic expression> | <additive arithmetic expression>

FIRST (<arithmetic expression>) = {-, +, AVID\_T, FPL\_T, INT\_L}

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

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

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

Fixed LR

<additive arithmetic expression> → + <multiplicative arithmetic expression> < additive arithmetic expression prime>

FIRST (<additive arithmetic expression>) = {AVID\_T, FPL\_T, INT\_L, (}

<additive arithmetic expression prime> → + <multiplicative arithmetic expression> < additive arithmetic expression prime> | - <multiplicative arithmetic expression> < additive arithmetic expression prime> | e

FIRST (<additive arithmetic expression prime>) = {+, -, e}

<multiplicative arithmetic expression> → <multiplicative arithmetic expression> \* <primary arithmetic expression> | <multiplicative arithmetic expression> / <primary arithmetic expression> | <primary arithmetic expression>

Fixed LR

<multiplicative arithmetic expression> → <primary arithmetic expression> <multiplicative arithmetic expression prime>

FIRST (<multiplicative arithmetic expression>) = {AVID\_T, FPL\_T, INT\_L, (}

<multiplicative arithmetic expression prime> → \* <primary arithmetic expression> <multiplicative arithmetic expression prime> | / <primary arithmetic expression> <multiplicative arithmetic expression prime> | 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, INT\_L, (}

String Expression:

<string expression> → <primary string expression> | <string expression> $$ <primary string expression>

Fixed LR

<string expression> → <string expression> $$ <primary string expression> | <primary string expression>

<string expression> → <primary string expression> <string expression prime>

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

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

FIRST (<string expression prime>) = {$$, e}

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

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

Conditional Expression:

<conditional expression> → <logical OR expression>

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

<logical OR expression> → <logical AND expression> | <logical OR expression> \_OR\_ <logical AND expression>

Fixed LR

<logical OR expression> → <logical OR expression> \_OR\_ <logical AND expression> | <logical AND expression>

<logical OR expression> → <logical AND expression> <logical OR expression prime>

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

<logical OR expression prime> → \_OR\_ <logical AND expression> <logical OR expression prime> | e

FIRST (<logical OR expression prime>) = {\_OR\_, e}

<logical AND expression> → <logical NOT expression> | <logical AND expression> \_AND\_ <logical NOT expression>

Fixed LR

<logical AND expression> → <logical AND expression> \_AND\_ <logical NOT expression> | <logical NOT expression>

<logical AND expression> → <logical NOT expression> <logical AND expression prime>

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

<logical AND expression prime> → \_AND\_ <logical NOT expression> <logical AND expression prime> | e

FIRST (<logical AND expression prime>) = {\_AND\_, e}

<logical NOT expression> → \_NOT\_ <relational expression> | <relational expression>

FIRST (<logical NOT expression>) = {\_NOT\_, e}

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> |

Applying LF

<relational expression> → <primary a\_relational expression> <primary a\_relational expression prime> |

<primary s\_relational expression><primary s\_relational expression prime>

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

<primary a\_relational expression prime> → == <primary a\_relational expression>|

<> <primary a\_relational expression> |

> <primary a\_relational expression> |

< <primary a\_relational expression>

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

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

<> <primary s \_relational expression> |

> <primary s \_relational expression> |

< <primary s \_relational expression>

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

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

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

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

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