PLATYPUS LANGUAGE SPECIFICATION

3 The PLATYPUS Syntactic Specification

3.1 PLATYPUS Program

<program> ->

PLATYPUS {<opt\_statements>}

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

<opt\_statements> -> <statements> | E

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

<statements> ->

<statement> | <statements> <statement>

LEFT RECURSION

<statements> ->

<statement> <statements> | <statement>

<statements> ->

<statement> <statements’>

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

<statements’> ->

<statement> <statements’>

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

3.2 Statements

<statement> ->

<assignment statement>

| <selection statement>

| <iteration statement>

| <input statement>

| <output statement>

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

* + 1. Assignment Statement

<assignment statement> ->

<assignment expression>;

FIRST < assignment statement> -> {<AVID> , <SVID>}

< assignment expression> ->

AVID = <arithmetic expression>

| SVID = <string expression>

FIRST <assignment expression> ->{ <AVID> , <SVID>}

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>};

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

<pre-condition> ->

TRUE | FALSE

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

3.2.4 Input Statement

<input statement> ->

READ (<variable list>);

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

LEFT RECURSION

<variable list> ->

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

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

FIRST(<variable list>)

= {FIRST(variable identifier)}

= { AVID\_T, SVID\_T }

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

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

<variable identifier> -> {AVID | SVID}

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

3.2.5 Output Statement

<output statement> ->

WRITE (<opt\_variable list>);

| WRITE (STR\_T);

LEFT FACTORING

<output statement> -> WRITE (<output\_list>);

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

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

FIRST (<output\_list>) = {AVID\_T, SVID\_T, STR\_T, e }

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

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

3.3 Expressions

3.3.1 Arithmetic Expression

<arithmetic expression> - >

<unary arithmetic expression>

| <additive arithmetic expression>

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

LEFT RECURSION

<additive arithmetic expression> ->

<multiplicative arithmetic expression><additive arithmetic expression’>

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

<additive arithmetic expression’> ->

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

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

| e

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

<multiplicative arithmetic expression> ->

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

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

| <primary arithmetic expression>

LEFT RECURSION

<multiplicative arithmetic expression> ->

<primary arithmetic expression><multiplicative arithmetic expression’>

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

<multiplicative arithmetic expression’> ->

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

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

| 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, INL\_T, ( }

3.3.2 String Expression

<string expression> ->

<primary string expression>

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

LEFT RECURSION

<string expression> -> <string expression> << <primary string expression> | <primary string expression>

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

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

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

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

<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, STR\_T }

<logical OR expression> ->

<logical AND expression>

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

LEFT RECURSION

<logical OR expression> ->

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

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

<logical OR expression’> -> .OR. <logical AND expression><logical OR expression’> | e FIRST (<logical OR expression’>) = { .OR. , e }

<logical AND expression> ->

<relational expression>

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

LEFT RECURSION

<logical AND expression> ->

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

| <relational expression>

<logical AND expression> ->

<relational expression> <logical AND expression’>

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

<logical AND expression’> -> .AND. <relational expression><logical AND expression’> | e FIRST (<logical AND expression’>) = {.AND. , e }

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>

Left Factoring

<relational expression> ->

<primary a\_relational expression> <primary a\_relational expression’>

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

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

<primary a\_relational expression’> ->

== <primary a\_relational expression>

| <> <primary a\_relational expression>

| > <primary a\_relational expression>

| < <primary a\_relational expression>

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

<primary s\_relational expression’> ->

== <primary s\_relational expression>

| <> <primary s\_relational expression>

| > <primary s\_relational expression>

| < <primary s\_relational expression>

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

<relational expression> ->

<primary a\_relational expression> <primary a\_relational expression’>

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

<primary a\_relational expression> ->

AVID\_T

| FPL\_T

| INL\_T

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

<primary s\_relational expression> ->

<primary string expression>

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