# **The PLATYPUS Syntactic Specification**

## 3.1 PLATYPUS Program

<program> ->

PLATYPUS {<opt\_statements>}

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

<opt\_statements> ->

<statements><statements’>

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

<statements> ->

<statement> | <statements> <statement>

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

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

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

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

## 3.2 Statements

<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(READ), KW\_T(WRITE) }

## 3.2.1 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}

## 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)}

<variable list> ->

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

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

FIRST(<variable list>) -> {AVID\_T, SVID\_T, COM\_T, Ɛ }

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

FIRST(<variable list’> -> { Ɛ , SVID\_T, AVID\_T}

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

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

## 3.2.5 Output Statement

<output statement> ->

WRITE (<*opt\_variable list>*);

| WRITE (STR\_T);

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

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

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

FIRST(<output list>) -> { Ɛ, AVID\_T, SVID\_T, STR\_T}

## 3.3 Expressions

## 3.3.1 Arithmetic Expression

<arithmetic expression> - >

<unary arithmetic expression>

| <additive arithmetic expression>

FIRST(<arithmetic expression>) -> {ART\_OP\_T(PLUS), ART\_OP\_T(MINUS), AVID\_T, FPL\_T, INL\_T}

<unary arithmetic expression> ->

- <primary arithmetic expression>

| + <primary arithmetic expression>

FIRST(<unary arithmetic expression>) -> {ART\_OP\_T(PLUS), ART\_OP\_T(MINUS)}

<additive arithmetic expression> ->

<additive arithmetic expression> + <multiplicative arithmetic expression>

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

| <multiplicative arithmetic expression>

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

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

<additive arithmetic expression’> ->

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

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

| Ɛ

FIRST(<additive arithmetic expression’> -> { Ɛ, ART\_OP\_T(PLUS), ART\_OP\_T(MINUS)}

<multiplicative arithmetic expression> ->

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

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

| <primary arithmetic expression>

<multiplicative arithmetic expression> ->

<primary arithmetic expression><multiplicative arithmetic expression’>

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

<multiplicative arithmetic expression’> ->

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

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

| Ɛ

FIRST(<multiplicative arithmetic expression’>) -> { Ɛ, ART\_OP\_T(DIV), ART\_OP\_T(MULT)}

<primary arithmetic expression> ->

AVID\_T

| FPL\_T

| INL\_T

| (<arithmetic expression>)

FIRST(<primary arithmetic expression> -> {AVID\_T, FPL\_T, INL\_T, LPR\_T}

## 3.3.2 String Expression

<string expression> ->

<primary string expression>

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

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

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

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

FIRST(<string expression’> -> { Ɛ, SCC\_OP\_T}

<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>) -> {STR\_T, SVID\_T, AVID\_T, FPL\_T, INL\_T}

<logical OR expression> ->

<logical AND expression>

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

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

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

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

FIRST(<logical OR expression’>) -> { Ɛ, LOG\_OP\_T(OR)}

<logical AND expression> ->

<relational expression>

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

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

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

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

FIRST(<logical AND expression’>) -> { Ɛ, LOG\_OP\_T(AND)}

## 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>

<relational expression> ->

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

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

FIRST(<relational expression>) -> {STR\_T, SVID\_T, AVID\_T, FPL\_T, INL\_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’>) -> {REL\_OP\_T(EQ), REL\_OP\_T(NE), REL\_OP\_T(LT), REL\_OP\_T(GT)}

<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’>) -> {REL\_OP\_T(EQ), REL\_OP\_T(NE), REL\_OP\_T(LT), REL\_OP\_T(GT)}

<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>) -> {STR\_T, SVID\_T}

<primary string expression> ->

STR\_T | SVID\_T

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