# **Input Elements and Tokens**

< input character > -> one of ASCII characters but not SEOF

<input element > -> <white space > | <comment> | <token>

<token> ->

<variable identifier>

| <keyword>

| <floating-point literal >

| <integer literal >

| <string literal>

| <separator>

| <operator>

# **Variable Identifiers**

<variable identifier> -> <arithmetic variable identifier> | <string variable identifier>

<string variable identifier> -> <arithmetic variable identifier>@

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

# **Keywords**

<keyword> -> PLATYPUS | IF | THEN | ELSE | WHILE | REPEAT | READ | WRITE | TRUE | FALSE

# **Integer Literals**

<integer literal> -> <decimal integer literal>

# **Floating-point Literals**

<floating-point literal> -> <decimal integer literal> . <opt\_digits>

# **Separators**

<separator> -> one of ( ){ } , ; “

# **Operators**

<operator> ->

< arithmetic operator > | <string concatenation operator>

| < relational operator> | < logical operator >

| < assignment operator >

(ART\_OP\_T )<arithmetic operator> -> one of + - \* /

(SCC\_OP\_T )<string concatenation operator> -> <<

(REL\_OP\_T )<relational operator> -> one of > < == <>

(LOG\_OP\_T)<logical operator> -> .AND. | .OR.

(ASS\_OP\_T)<assignment operator> -> =

# **PLATYPUS Program**

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

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

<opt\_statements> -> <statements> | ε

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

# **Statements**

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

**Eliminate left recursion**

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

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

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

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

<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), KW\_T(WHILE)}

# **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)}

<opt\_statements> -> <statements> | ε

# **Iteration 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 )}

# **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}

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

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

FIRST(<variable list’>)= {COM\_T, ε}

# **Output Statement**

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

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

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

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

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

# **Arithmetic Expression**

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

FIRST(<arithmetic expression>)= { ART\_OP\_T( - ), ART\_OP\_T( + ), AVID\_T, FPL\_T, INL\_T, LPR\_T}

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

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

<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 expression’>

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

FIRST(<additive arithmetic expression’>)= {ART\_OP\_T(+) , ART\_OP\_T(-) , ε}

<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’>

| ε

FIRST(<multiplicative arithmetic expression’>)= {ART\_OP\_T( \* ) , ART\_OP\_T( / ) , ε}

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

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

# **String Expression**

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

FIRST(<string expression’>)= {SCC\_OP\_T , ε}

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

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

FIRST(<logical OR expression’>)={ LOG\_OP\_T(.OR.) , ε}

<logical AND expression> ->

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

FIRST(<logical AND expression’>)= {LOG\_OP\_T(.AND.) , ε}

# **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>)= { 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’>)={REL\_OP\_T( == ), REL\_OP\_T(<>), REL\_OP\_T( > ), REL\_OP\_T( < )}

<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( == ), REL\_OP\_T(<>), REL\_OP\_T( > ), REL\_OP\_T( < )}

<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}

**Joseph Trottier 040866019**

**Course: CCST8152 Compilers  
Lab Section: 12  
Assignment 3  
Professor: SV.Ranev**

**DueDate: April 18th, 2019  
Date: April 18th, 2019  
Contents:**