Hai Nguyen

CPSC 323

Professor Anthony Le

November 7, 2019

**Assignment 2 (Syntax Analyzer) Documentation**

Operating System: Windows 10

Language: C++ (Visual Studio 2017)

Description: This program reads a source code file or 1 line of code and analyze the syntax (predictive RDP). The result is printed to the screen and "parse\_result.txt" file.

**1. Problem Statement:**

Build a syntax analyzer using a top down parser (predictive RDP).

**2. How to use the program:**

- User enters a source code file or a single line of code.

- The program generates a stream of tokens and analyze the syntax.

- The program prints the tokens and production rules to the screen and “parse\_result.txt”  
 file.

- If the code is syntactically correct, the program will print “Finished.” at the end.  
 Otherwise, it will print an error message.

**3. Design of the program**:

Terminals:

int\_num, real\_num : an int or real number

id : a valid identifier

int, float, bool, true, false, if, then, else, endif: keywords

+, -, \*, /, >, <, = : operators

(, ), “,”, {, } : seperators

epsilon

**Production rules:**

<Statement> -> <Expression statement> | <Assign> | <Declare> | <If> | <While>

First(<Statement>) = {-, id, int\_num, real\_num, (, true, false, int, float, bool, if, while }

Assign statement:

<Assign> -> id = <Expression>;

<Expression> -> <Term> <ExpressionPrime>

<ExpressionPrime> -> + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <Empty>

<Term> -> <Factor> <TermPrime>

<TermPrime> -> \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <Empty>

<Factor> -> - <Primary> | <Primary>

<Primary> -> id | int\_num | ( <Expression> ) | real\_num | true | false

<Empty> -> epsilon

First(<Assign>) = { id}

First(<Expression>) = { -, id, int\_num, (, real\_num, true, false}

First(<ExpressionPrime>) = {+, -, epsilon}

First(<Term>) = { -, id, int\_num, (, real\_num, true, false}

First(<TermPrime>) = { \*, /, epsilon}

First(<Factor>) = { -, id, int\_num, (, real\_num, true, false}

First(<Primary>) = {id, int\_num, (, real\_num, true, false}

Example: a = b + c;

Expression statement:

<Expression Statement> -> <Expression>;

<Expression> -> <Term> <ExpressionPrime>

<ExpressionPrime> -> + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <Empty>

<Term> -> <Factor> <TermPrime>

<TermPrime> -> \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <Empty>

<Factor> -> - <Primary> | <Primary>

<Primary> -> id | int\_num | ( <Expression> ) | real\_num | true | false

<Empty> -> epsilon

First(<Expression Statement>) = {-, id, int\_num, real\_num, (, true, false }

Example: 1 – a;

b;

Declare statement:

<Declare> -> <Type> id <MoreIDs>;

<Type> -> int | float | bool

<MoreIDs> -> , id <MoreIDs> | <Empty>

First(<Declare>) = {int, float, bool}

First(<Type>) = {int, float, bool}

First(<MoreIDs>) = {“,” , epsilon}

Example: float a, b;

If statement:

<If> -> if <Conditional> then <StatementBlock> <ElseBlock> endif

<Conditional> -> <Expression> <ConditionalPrime>

<ConditionalPrime> -> <Relop> <Expression> | <Empty>

<Relop> -> < | <= | == | <> | >= | >

<StatementBlock> -> { <Statement> <moreStatement>

<moreStatement> -> <Statement> <moreStatement> | }

<ElseBlock> -> else <StatementBlock> | <Empty>

First(<If>) = {if}

First(<Conditional>) = { -, id, int\_num, (, real\_num, true, false}

First(<ConditionalPrime>) = {<, =, >, epsilon}

First(<Relop>) = {<, =, >}

First(<StatementBlock>) = { “{“ }

First(<moreStatement>) = { -, id, int\_num, real\_num, (, true, false, int, float, bool, if, while, “}”}

First(<ElseBlock>) = {else, epsilon }

Example:

if a > max then {

max = a;

} else {

a = 0;

} endif

if a > b then {

a = b;

} endif

While statement:

<While> -> while <Conditional> do <StatementBlock> whileend

<Conditional> -> <Expression> <ConditionalPrime>

<ConditionalPrime> -> <Relop> <Expression> | <Empty>

<Relop> -> < | <= | == | <> | >= | >

<StatementBlock> -> { <Statement> <moreStatement>

<moreStatement> -> <Statement> <moreStatement> | }

First(<While>) = {while}

Example:

while a < 5 do {

a = a +1;

} whileend

**4. Any Limitation:**

The program exits if it finds an error in the source code.

**5. Any Shortcomings:**

None

**6. Test cases:**

Case 1:

x=a + b;

z= (c - d)\*2;

-Token: identifier -Lexeme: x

<Statement> -> <Expression-statement> | <Assign> | <If> | <Declare> | <While>

<Assign> -> identifier = <Expression>;

-Token: operator -Lexeme: =

-Token: identifier -Lexeme: a

<Expression> -> <Term> <ExpressionPrime>

<Term> -> <Factor> <TermPrime>

<Factor> -> - <Primary> | <Primary>

<Primary> -> identifier | int | ( <Expression> ) | real | true | false

-Token: operator -Lexeme: +

<TermPrime> -> \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <Empty>

<Empty> -> epsilon

<ExpressionPrime> -> + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <Empty>

-Token: identifier -Lexeme: b

<Term> -> <Factor> <TermPrime>

<Factor> -> - <Primary> | <Primary>

<Primary> -> identifier | int | ( <Expression> ) | real | true | false

-Token: separator -Lexeme: ;

<TermPrime> -> \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <Empty>

<Empty> -> epsilon

<ExpressionPrime> -> + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <Empty>

<Empty> -> epsilon

-Token: identifier -Lexeme: z

<Statement> -> <Expression-statement> | <Assign> | <If> | <Declare> | <While>

<Assign> -> identifier = <Expression>;

-Token: operator -Lexeme: =

-Token: separator -Lexeme: (

<Expression> -> <Term> <ExpressionPrime>

<Term> -> <Factor> <TermPrime>

<Factor> -> - <Primary> | <Primary>

<Primary> -> identifier | int | ( <Expression> ) | real | true | false

-Token: identifier -Lexeme: c

<Expression> -> <Term> <ExpressionPrime>

<Term> -> <Factor> <TermPrime>

<Factor> -> - <Primary> | <Primary>

<Primary> -> identifier | int | ( <Expression> ) | real | true | false

-Token: operator -Lexeme: -

<TermPrime> -> \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <Empty>

<Empty> -> epsilon

<ExpressionPrime> -> + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <Empty>

-Token: identifier -Lexeme: d

<Term> -> <Factor> <TermPrime>

<Factor> -> - <Primary> | <Primary>

<Primary> -> identifier | int | ( <Expression> ) | real | true | false

-Token: separator -Lexeme: )

<TermPrime> -> \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <Empty>

<Empty> -> epsilon

<ExpressionPrime> -> + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <Empty>

<Empty> -> epsilon

-Token: operator -Lexeme: \*

<TermPrime> -> \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <Empty>

-Token: int -Lexeme: 2

<Factor> -> - <Primary> | <Primary>

<Primary> -> identifier | int | ( <Expression> ) | real | true | false

-Token: separator -Lexeme: ;

<TermPrime> -> \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <Empty>

<Empty> -> epsilon

<ExpressionPrime> -> + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <Empty>

<Empty> -> epsilon

Finished.

Case 2:

int a;

float b, c;

bool isValid, isEmpty, d;

-Token: keyword -Lexeme: int

<Statement> -> <Expression-statement> | <Assign> | <If> | <Declare> | <While>

<Decalre> -> <Type> identifier <MoreIDs>;

<Type> -> int | float | bool

-Token: identifier -Lexeme: a

-Token: separator -Lexeme: ;

<MoreIDs> -> , id <MoreIDs> | <Empty>

<Empty> -> epsilon

-Token: keyword -Lexeme: float

<Statement> -> <Expression-statement> | <Assign> | <If> | <Declare> | <While>

<Decalre> -> <Type> identifier <MoreIDs>;

<Type> -> int | float | bool

-Token: identifier -Lexeme: b

-Token: separator -Lexeme: ,

<MoreIDs> -> , id <MoreIDs> | <Empty>

-Token: identifier -Lexeme: c

-Token: separator -Lexeme: ;

<MoreIDs> -> , id <MoreIDs> | <Empty>

<Empty> -> epsilon

-Token: keyword -Lexeme: bool

<Statement> -> <Expression-statement> | <Assign> | <If> | <Declare> | <While>

<Decalre> -> <Type> identifier <MoreIDs>;

<Type> -> int | float | bool

-Token: identifier -Lexeme: isValid

-Token: separator -Lexeme: ,

<MoreIDs> -> , id <MoreIDs> | <Empty>

-Token: identifier -Lexeme: isEmpty

-Token: separator -Lexeme: ,

<MoreIDs> -> , id <MoreIDs> | <Empty>

-Token: identifier -Lexeme: d

-Token: separator -Lexeme: ;

<MoreIDs> -> , id <MoreIDs> | <Empty>

<Empty> -> epsilon

Finished.

Case 3:

while a > b do {

if a == 1 then {

a = a+b;

b = 1;

} endif

bool b;

} whileend

-Token: keyword -Lexeme: while

<Statement> -> <Expression-statement> | <Assign> | <If> | <Declare> | <While>

<While> -> while <Conditional> do <StatementBlock> whileend

-Token: identifier -Lexeme: a

<Conditional> -> <Expression> <ConditionalPrime>

<Expression> -> <Term> <ExpressionPrime>

<Term> -> <Factor> <TermPrime>

<Factor> -> - <Primary> | <Primary>

<Primary> -> identifier | int | ( <Expression> ) | real | true | false

-Token: operator -Lexeme: >

<TermPrime> -> \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <Empty>

<Empty> -> epsilon

<ExpressionPrime> -> + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <Empty>

<Empty> -> epsilon

<ConditionalPrime> -> <Relop> <Expression> | <Empty>

<Relop> -> < | <= | == | <> | >= | >

-Token: identifier -Lexeme: b

<Expression> -> <Term> <ExpressionPrime>

<Term> -> <Factor> <TermPrime>

<Factor> -> - <Primary> | <Primary>

<Primary> -> identifier | int | ( <Expression> ) | real | true | false

-Token: keyword -Lexeme: do

<TermPrime> -> \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <Empty>

<Empty> -> epsilon

<ExpressionPrime> -> + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <Empty>

<Empty> -> epsilon

-Token: separator -Lexeme: {

<StatementBlock> -> { <Statement> <moreStatement>

-Token: keyword -Lexeme: if

<Statement> -> <Expression-statement> | <Assign> | <If> | <Declare> | <While>

<If> -> if <Conditional> then <StatementBlock> <ElseBlock> endif

-Token: identifier -Lexeme: a

<Conditional> -> <Expression> <ConditionalPrime>

<Expression> -> <Term> <ExpressionPrime>

<Term> -> <Factor> <TermPrime>

<Factor> -> - <Primary> | <Primary>

<Primary> -> identifier | int | ( <Expression> ) | real | true | false

-Token: operator -Lexeme: =

<TermPrime> -> \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <Empty>

<Empty> -> epsilon

<ExpressionPrime> -> + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <Empty>

<Empty> -> epsilon

<ConditionalPrime> -> <Relop> <Expression> | <Empty>

<Relop> -> < | <= | == | <> | >= | >

-Token: operator -Lexeme: =

-Token: int -Lexeme: 1

<Expression> -> <Term> <ExpressionPrime>

<Term> -> <Factor> <TermPrime>

<Factor> -> - <Primary> | <Primary>

<Primary> -> identifier | int | ( <Expression> ) | real | true | false

-Token: keyword -Lexeme: then

<TermPrime> -> \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <Empty>

<Empty> -> epsilon

<ExpressionPrime> -> + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <Empty>

<Empty> -> epsilon

-Token: separator -Lexeme: {

<StatementBlock> -> { <Statement> <moreStatement>

-Token: identifier -Lexeme: a

<Statement> -> <Expression-statement> | <Assign> | <If> | <Declare> | <While>

<Assign> -> identifier = <Expression>;

-Token: operator -Lexeme: =

-Token: identifier -Lexeme: a

<Expression> -> <Term> <ExpressionPrime>

<Term> -> <Factor> <TermPrime>

<Factor> -> - <Primary> | <Primary>

<Primary> -> identifier | int | ( <Expression> ) | real | true | false

-Token: operator -Lexeme: +

<TermPrime> -> \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <Empty>

<Empty> -> epsilon

<ExpressionPrime> -> + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <Empty>

-Token: identifier -Lexeme: b

<Term> -> <Factor> <TermPrime>

<Factor> -> - <Primary> | <Primary>

<Primary> -> identifier | int | ( <Expression> ) | real | true | false

-Token: separator -Lexeme: ;

<TermPrime> -> \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <Empty>

<Empty> -> epsilon

<ExpressionPrime> -> + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <Empty>

<Empty> -> epsilon

-Token: identifier -Lexeme: b

<moreStatement> -> <Statement> <moreStatement> | }

<Statement> -> <Expression-statement> | <Assign> | <If> | <Declare> | <While>

<Assign> -> identifier = <Expression>;

-Token: operator -Lexeme: =

-Token: int -Lexeme: 1

<Expression> -> <Term> <ExpressionPrime>

<Term> -> <Factor> <TermPrime>

<Factor> -> - <Primary> | <Primary>

<Primary> -> identifier | int | ( <Expression> ) | real | true | false

-Token: separator -Lexeme: ;

<TermPrime> -> \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <Empty>

<Empty> -> epsilon

<ExpressionPrime> -> + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <Empty>

<Empty> -> epsilon

-Token: separator -Lexeme: }

<moreStatement> -> <Statement> <moreStatement> | }

-Token: keyword -Lexeme: endif

<ElseBlock> -> else <StatementBlock> | <Empty>

<Empty> -> epsilon

-Token: keyword -Lexeme: bool

<moreStatement> -> <Statement> <moreStatement> | }

<Statement> -> <Expression-statement> | <Assign> | <If> | <Declare> | <While>

<Decalre> -> <Type> identifier <MoreIDs>;

<Type> -> int | float | bool

-Token: identifier -Lexeme: b

-Token: separator -Lexeme: ;

<MoreIDs> -> , id <MoreIDs> | <Empty>

<Empty> -> epsilon

-Token: separator -Lexeme: }

<moreStatement> -> <Statement> <moreStatement> | }

-Token: keyword -Lexeme: whileend

Finished.

Case 4:

height + width;

-Token: identifier -Lexeme: height

<Statement> -> <Expression-statement> | <Assign> | <If> | <Declare> | <While>

<Expression Statement> -> <Expression>;

<Expression> -> <Term> <ExpressionPrime>

<Term> -> <Factor> <TermPrime>

<Factor> -> - <Primary> | <Primary>

<Primary> -> identifier | int | ( <Expression> ) | real | true | false

-Token: operator -Lexeme: +

<TermPrime> -> \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <Empty>

<Empty> -> epsilon

<ExpressionPrime> -> + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <Empty>

-Token: identifier -Lexeme: width

<Term> -> <Factor> <TermPrime>

<Factor> -> - <Primary> | <Primary>

<Primary> -> identifier | int | ( <Expression> ) | real | true | false

-Token: separator -Lexeme: ;

<TermPrime> -> \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <Empty>

<Empty> -> epsilon

<ExpressionPrime> -> + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <Empty>

<Empty> -> epsilon

Finished.

Case 5:

if i >= size then {

size + 1;

} else {

i = size;

} endif

-Token: keyword -Lexeme: if

<Statement> -> <Expression-statement> | <Assign> | <If> | <Declare> | <While>

<If> -> if <Conditional> then <StatementBlock> <ElseBlock> endif

-Token: identifier -Lexeme: i

<Conditional> -> <Expression> <ConditionalPrime>

<Expression> -> <Term> <ExpressionPrime>

<Term> -> <Factor> <TermPrime>

<Factor> -> - <Primary> | <Primary>

<Primary> -> identifier | int | ( <Expression> ) | real | true | false

-Token: operator -Lexeme: >

<TermPrime> -> \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <Empty>

<Empty> -> epsilon

<ExpressionPrime> -> + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <Empty>

<Empty> -> epsilon

<ConditionalPrime> -> <Relop> <Expression> | <Empty>

<Relop> -> < | <= | == | <> | >= | >

-Token: operator -Lexeme: =

-Token: identifier -Lexeme: size

<Expression> -> <Term> <ExpressionPrime>

<Term> -> <Factor> <TermPrime>

<Factor> -> - <Primary> | <Primary>

<Primary> -> identifier | int | ( <Expression> ) | real | true | false

-Token: keyword -Lexeme: then

<TermPrime> -> \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <Empty>

<Empty> -> epsilon

<ExpressionPrime> -> + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <Empty>

<Empty> -> epsilon

-Token: separator -Lexeme: {

<StatementBlock> -> { <Statement> <moreStatement>

-Token: identifier -Lexeme: size

<Statement> -> <Expression-statement> | <Assign> | <If> | <Declare> | <While>

<Expression Statement> -> <Expression>;

<Expression> -> <Term> <ExpressionPrime>

<Term> -> <Factor> <TermPrime>

<Factor> -> - <Primary> | <Primary>

<Primary> -> identifier | int | ( <Expression> ) | real | true | false

-Token: operator -Lexeme: +

<TermPrime> -> \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <Empty>

<Empty> -> epsilon

<ExpressionPrime> -> + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <Empty>

-Token: int -Lexeme: 1

<Term> -> <Factor> <TermPrime>

<Factor> -> - <Primary> | <Primary>

<Primary> -> identifier | int | ( <Expression> ) | real | true | false

-Token: separator -Lexeme: ;

<TermPrime> -> \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <Empty>

<Empty> -> epsilon

<ExpressionPrime> -> + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <Empty>

<Empty> -> epsilon

-Token: separator -Lexeme: }

<moreStatement> -> <Statement> <moreStatement> | }

-Token: keyword -Lexeme: else

<ElseBlock> -> else <StatementBlock> | <Empty>

-Token: separator -Lexeme: {

<StatementBlock> -> { <Statement> <moreStatement>

-Token: identifier -Lexeme: i

<Statement> -> <Expression-statement> | <Assign> | <If> | <Declare> | <While>

<Assign> -> identifier = <Expression>;

-Token: operator -Lexeme: =

-Token: identifier -Lexeme: size

<Expression> -> <Term> <ExpressionPrime>

<Term> -> <Factor> <TermPrime>

<Factor> -> - <Primary> | <Primary>

<Primary> -> identifier | int | ( <Expression> ) | real | true | false

-Token: separator -Lexeme: ;

<TermPrime> -> \* <Factor> <TermPrime> | / <Factor> <TermPrime> | <Empty>

<Empty> -> epsilon

<ExpressionPrime> -> + <Term> <ExpressionPrime> | - <Term> <ExpressionPrime> | <Empty>

<Empty> -> epsilon

-Token: separator -Lexeme: }

<moreStatement> -> <Statement> <moreStatement> | }

-Token: keyword -Lexeme: endif

Finished.