Language Specification (Full Compilation)

Tahmid Khan Arnab

[Corrosion]

|  |  |
| --- | --- |
| **Part**  **1** | **Basic Language Reference** |

* 1. **User Manual**

**Element 1: Name / Extension**

*Name: Corrosion*

*Extension: .cor*

**Element 2 – Comments**

*Comments in my language are one-line comments that start with // and end at the end of the line (\n).*

*Example: // Comment*

*<code>*

**Element 3 – Keywords**

*All keywords are prefixed with underscore(\_):*

*main, i32, str, if, else, let, fn, print, input*

**Element 4 – Datatypes**

|  |  |  |
| --- | --- | --- |
| *Datatype* | *Size* | *Range* |
| *i32* | *32 bit integer* | *-231 till 231-1* |
| *str* | *Depends on number of chars* | *As much storage is available in memory* |

**Element 5 – Variables**

*The basic syntax is:*

*\_let <variable\_id>: <datatype> = <value>;*

*Datatypes: \_i32 or \_str*

*Value: based on type, “string” or integer*

**Element 6 - Commands**

* ***Attribution / assignment****: My language allows assignment as such:*
  + ***<variable\_id> = <value>****;*

*String concatenation is not allowed.*

*My language handles math as one line arithmetic assignments and doesn’t allow multiple operations in one line.*

* ***Selection****: My language has a simple if else clause which uses a logical condition:*
  + ***\_if(<variable> <rel\_op> <variable>) {*** *} else { }*

***<rel\_op>: =, !=, <, >***

* ***Input****: My language uses input statements with a variable to assign the input to.*
  + ***\_input(<variable\_name>)****;*
* ***Output****: My language uses the print keyword with a String variable or value.*
  + ***\_print(<string\_variable\_name>* | *<string\_literal>);***
* ***Functions****:*
  + *Syntax: \_function%(){<statements>};*
  + *My functions will only perform very basic functions which does not require parameters or a return value.*
  1. **Implementation challenges**

*[Describe the main challenges that you faced about front-end compilation]*

* *Main difficulties: Confusing errors and recursion*
* *How I solved them: By setting up the debugging and reading through the code several times.*
  1. **Real Examples**

// Corrosion square area calculator program

\_fn \_main(){

\_let area: \_i32 = 0;

\_print("Enter breadth:");

\_input(a);

\_print("Enter width:");

\_input(b);

area = a \* b;

\_print(area);

}

// Corrosion simple number comparison program

\_fn \_main(){

\_print("Enter two numbers (a and b) to compare:");

\_input(a);

\_input(b);

\_if (a < b) {

\_print("First number is smaller!");

} \_else {

\_print("Second number is smaller!");

}

}

|  |  |
| --- | --- |
| **Part**  **2** | **Formal Models (RE, Automaton)** |

* 1. **Define the RE elements**

**Element 1: Lexeme Classes**

*L = [A-Za-z]*

*D = [0-9]*

*U = \_*

*M = %*

*Q = “*

*S = /*

*T = \n*

**Element 2: Define the RE for main kinds of tokens**

Comments: S S [^T] T

Variables: L+ [ L | D | U]\*

Methods: U [ L | D | U]\* M

Literals:

Integers: D+

Strings: Q [^Q]\* Q

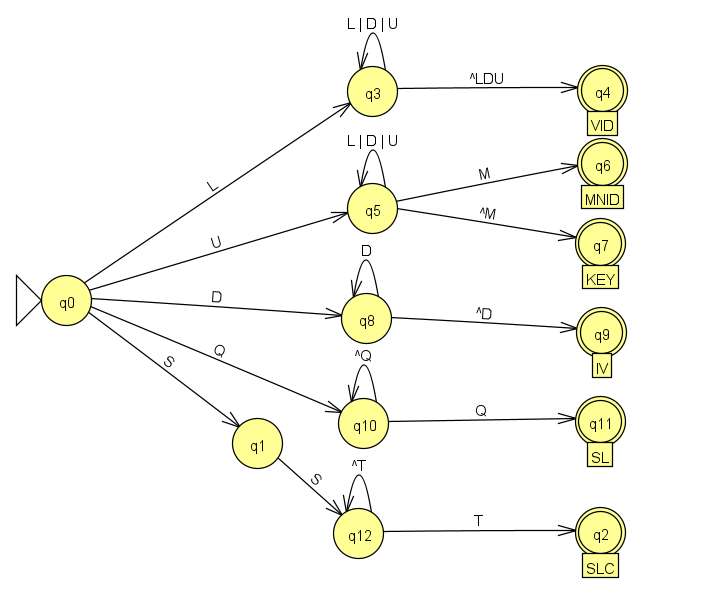
**Element 3: Keyword lists**

*All keywords are prefixed with an underscore(\_) in the code:*

*fn, main, i32, str, if, else, let, fn, print, input*

* 1. **Automaton and transition table**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Input | Input Symbol | | | | | | | | Output |
| q | L | D | U | M | Q | S | T | O | Type |
| 0 | 3 | 8 | 5 | ES | 10 | 1 | ES | ES | NOAS[0] |
| 1 | ES | ES | ES | ES | ES | 12 | ES | ES | NOAS[1] |
| 2 | IS | IS | IS | IS | IS | IS | IS | IS | ASNR(CMT)[2] |
| 3 | 3 | 3 | 3 | 4 | 4 | 4 | 4 | 4 | NOAS[3] |
| 4 | IS | IS | IS | IS | IS | IS | IS | IS | ASWR(VID)[4] |
| 5 | 5 | 5 | 5 | 6 | 7 | 7 | 7 | 7 | NOAS[5] |
| 6 | IS | IS | IS | IS | IS | IS | IS | IS | ASNR(MNID)[6] |
| 7 | IS | IS | IS | IS | IS | IS | IS | IS | ASWR(KEY)[7] |
| 8 | 9 | 8 | 9 | 9 | 9 | 9 | 9 | 9 | NOAS[8] |
| 9 | IS | IS | IS | IS | IS | IS | IS | IS | ASWR(SL)[9] |
| 10 | 10 | 10 | 10 | 10 | 11 | 10 | 10 | 10 | NOAS[10] |
| 11 | IS | IS | IS | IS | IS | IS | IS | IS | ASNR(SL)[11] |
| 12 | 12 | 12 | 12 | 12 | 12 | 12 | 2 | 12 | NOAS[12] |



|  |  |
| --- | --- |
| **Part**  **3** | **Language Grammar** |

* 1. **Define the Language BNF**

1. **The Corrosion LANGUAGE Lexical Specification**
   1. **White Space**

White spaceis defined as the ASCII space, horizontal and vertical tabs, and form feed characters, as well as line terminators. White space is discarded by the scanner.

**<white space>** → *one of* { SPACE, TAB, FF, NL, CR, NLCR }

* 1. **Comments**

**Corrosion** LANGUAGE supports only single-line comments: all the text from the ASCII characters **!!** to the end of the line is ignored by the scanner:

**<comments>** → // { sequence of ASCII chars } \n

* 1. **Variable Identifiers**

The following variable identifier (VID) tokens are produced by the scanner: one kind of token for both arithmetic and strings: **VID\_T**.

**<variable identifier>** → VID\_T

* 1. **Keywords**

The scanner produces a single token: **KW\_T**. The type of the keyword is defined by the attribute of the token (the index of the keywordTable[]). Remember that the list of keywords in **Corrosion** language is given by:

**\_fn, \_main, \_let, \_mut, \_i32, \_str, \_if, \_else, \_input, \_print**

* 1. **Integer Literals**

The scanner produces a single token: **INL\_T** with an integer value as an attribute.

**<integer\_literal>** → INL\_T

* 1. **String Literals**

**STR\_T** token is produced by the scanner.

**<string\_literal>** → STR\_T

* 1. **Separators**

**<separator>** → *one of* {( ){ } ; }

Some different tokens are produced by the scanner - **LPR\_T**, **RPR\_T**, **LBR\_T**, **RBR\_T**, **EOS\_T**.

* 1. **Operators:**

**<separator>** → *one of* { (, ), {, }, ; }

A single token is produced by the scanner: **ART\_OP\_T**. The type of the operator is defined by the attribute of the token.

**<arithmetic operator>** → *one of* { +, -, \*, / }

A single token is produced by the scanner: **REL\_OP\_T**. The type of the operator is defined by the attribute of the token.

**<relational operator>** → *one of* { >, <, ==, != }

A single token is produced by the scanner: **LOG\_OP\_T**. The type of the operator is defined by the attribute of the token.

**<logical operator>** → *one of* { **&&** , **||** , **!** }

A single token is produced by the scanner: **ASS\_OP\_T**.

**<assignment operator>** → =

1. **The Corrosion LANGUAGE Syntactic Specification**
   1. **Corrosion LANGUAGE Program**
      1. **Program**

**Corrosion** LANGUAGE program is composed by one special function: “**main**” (Keyword) defined as follows:

**<program>**  \_fn **\_main** () {

<opt\_statements>

}

* + 1. **Optional Statements:**

**Optional Statements:**

**<opt\_statements>** → <statements> | ϵ

* + 1. **Statements**

**<statements>** → <statement> | <statements> <statement>

* 1. **Statement**

**<statement>** → <assignment statement> | <selection statement> | <iteration statement>

| <input statement> | <output statement> | <variable\_declaration>

* + 1. **Assignment Statement**

**<assignment statement>** → <assignment expression>

* + 1. **Variable declaration**

**<variable\_declaration>** → KW(\_let) **<variable>**: **<datatype>** = **<value>**;

**<datatype>** → KW(\_i32) | KW(\_str)

**<variable>** → VID\_T

**<value>** → VID\_T | INL\_T | STR\_T

* + 1. **Assignment Expression**

**<assignment expression>** → <variable> = <arithmetic expression>

| <variable> = <string expression>

* + 1. **Selection Statement (if and else statements),**

**<selection statement>** → **\_if** (<conditional expression>)

{ <opt\_statements> }

**\_else** { <opt\_statements> } ;

* + 1. **Iteration Statement (the loop statement)**

**<iteration statement>** → **\_while** (<conditional expression>)

**\_do** { <statements>};

* + 1. **Input Statement**

**<input statement>** → \_**input** (<variable>);

* + 1. **Output Statement**

**<output statement>** → \_**print** (<opt\_variable>);

**Optional Variable List:**

**<opt\_variable>** →<variable> | ϵ

* 1. **Expressions**
     1. **Arithmetic Expression**

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

**Unary Arithmetic Expression:**

**<unary arithmetic expression>** → - <primary arithmetic expression>

| + <primary arithmetic expression>

**Additive Arithmetic Expression:**

**<additive arithmetic expression>** →

<additive arithmetic expression> + <multiplicative arithmetic expression>

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

| <multiplicative arithmetic expression>

**Multiplicative Arithmetic Expression:**

**<multiplicative arithmetic expression>** →

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

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

| <primary arithmetic expression>

**Primary Arithmetic Expression:**

**<primary arithmetic expression>** → <variable>

| INL\_T

| (<arithmetic expression>)

* + 1. **String Expression**

**<string expression>** →

<primary string expression>

**Primary String Expression:**

**<primary string expression>** → <variable> | STR\_T

* + 1. **Conditional Expression**

**<conditional expression>** → <logical OR expression>

**Logical OR Expression:**

**<logical OR expression>** → <logical AND expression>

| <logical OR expression> **||** <logical AND expression>

**Logical AND Expression:**

**<logical AND expression>** → <logical NOT expression>

| <logical AND expression> **&&** <logical NOT expression>

**Logical NOT Expression:**

**<logical NOT expression>** → **!** <relational expression>

| <relational expression>

* + 1. **Relational Expression**

**<relational expression>** →

<relational a\_expression> | <relational s\_expression>

**Relational Arithmetic Expression:**

**<relational a\_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>

**Relational String Expression:**

**<relational s\_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>

**Primary Arithmetic Relational Expression:**

**<primary a\_relational expression>** →<variable> | INL\_T

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