

# JER Language SPECIFICATION[[1]](#footnote-1)

A context-free grammar is used to define the lexical and syntactical parts of the **JER LANGUAGE** and the lexical and syntactic structure of a **JER LANGUAGE** program.

1. **The JER 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**

JER LANGUAGE supports only multi-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 } ##

* 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 JER LANGUAGE is given by:

**return,false,true,null,local,global,and,or,continue,def,del,if,elseif,finally,for,while,in,pass,**

**with ,yield, try, expect, as, class, raise, int, char, int, char, bool, float, string.**

* 1. **Integer Literals**

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

**<integer\_literal>** → INL\_T

* 1. **Floating-point Literals**

**FPL\_T** token with a real decimal value as an attribute is produced by the scanner.

**<float\_literal>** → FL\_T

* 1. **String Literals**

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

**<string\_literal>** → STR\_T

* 1. **Separators**

**<separator>** → *one of* {( ): }

Seven different tokens are produced by the scanner - **LPR\_T**, **RPR\_T**, **COM\_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: **SCC\_OP\_T**.

**<string concatenation operator>** → ++

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 JER LANGUAGE Syntactic Specification**
   1. **JER LANGUAGE Program**
      1. **Program**

JER LANGUAGE program is composed by one special function: “main” (Method name) defined as follows.

**<program>**  **<id>(parms)**:

<indent><body>

<end\_indent><id>**(parms)**

* + 1. **Body**

The first part (**body**) is the place we declare the variables:

**<body>** <opt\_staments>

**Indent:**

**<indent>** → **<white space>**

**End of Indent:**

**<end\_indent>** → **<\b>**

Id:

Params:

**<params>** → **< opt\_varlist\_declarations >**

**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 Lists**

The optional variable list declarations is used to define several datatype declarations:

**<opt\_varlist\_declarations>** <varlist\_declarations> | ϵ

**Variable Declarations**

**<varlist\_declarations>** → <varlist\_declaration>

| <varlist\_declarations><varlist\_declaration>

* **PROBLEM DETECTED: Left recursion – SOLVING FOR YOU:**

**New Grammar**

**<varlist\_declarations>** → <varlist\_declaration> <varlist\_declarationsPrime>

**<varlist\_declarationsPrime>** → <varlist\_declaration> <varlist\_declarationsPrime> | ϵ

Each variable declaration can be done as follows:

**<varlist\_declaration>** → <integer\_varlist\_declaration>

| <float\_varlist\_declaration>

| <string\_varlist\_declaration>

* + 1. **Declaration of Lists:**

The variables list declaration is defined here:

**<integer\_varlist\_declaration>** <integer\_variable\_list>

**<float\_varlist\_declaration>** <float\_variable\_list>

**<string\_varlist\_declaration>** <string\_variable\_list>

* + 1. **List of Variables:**

The list of variables is defined here:

**Integers:**

**<integer\_variable\_list>** <integer\_variable>

| <integer\_variable\_list>, <integer\_variable>

**<integer\_variable>**  **INL\_T**

**Float-points:**

**<float\_variable\_list>** <float\_variable>

| <float\_variable\_list>, <float\_variable>

**<float\_variable>** FL\_T

**Strings:**

**<string\_variable\_list>** <string\_variable>

| <float\_variable\_list>, <string\_variable>

**<string\_variable>** STR\_T

* + 1. **Assignment Statement**

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

* + 1. **Assignment Expression**

**<assignment expression>** → <integer\_variable> = <arithmetic expression>

| <float\_variable> = <arithmetic expression>

| <string\_variable>= <string expression>

* + 1. **Selection Statement (if statement)**

**<selection statement>** → **if** <conditional expression>:

<indent> <opt\_statements>

<end\_indent>**else**:

<indent><opt\_statement>

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

**<iteration statement>** → **while** <conditional expression>:

<indent><opt\_statement>

<end\_indent>**else**:

<indent><opt\_statement>

* + 1. **Input Statement**

**<input statement>** → **input** (<opt\_string>)

Option string:

**<** opt\_string **>** → **<STR\_T>** | ϵ

* + 1. **Output Statement**

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

**Optional Variable List:**

**<opt\_variable list>** →<variable list> | ϵ

**Variable List:**

**<variable list>** → <variable identifier> | <variable list>,<variable identifier>

**Variable Identifier:**

**<variable identifier>** →<integer\_variable>

| <float\_variable>

| <string\_variable>

**Integer variable:**

**<integer\_variable>** →INL\_T

**Float variable:**

**<float\_variable>** →FL\_T

**String variable:**

**<string\_variable>** →STR\_T

* 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>** → <integer\_variable>

| <float\_variable>

| FL\_T | INL\_T

| (<arithmetic expression>)

* + 1. **String Expression**

**<string expression>** →

<primary string expression>

**Primary String Expression:**

**<primary string expression>** → <string\_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>** →<integer\_variable> | <float\_variable> | FPL\_T | INL\_T

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

Ex:

**## JER Example 3:**

**Volume of a sphere ##**

**def main ():**

**PI = 3.14**

**r = int (input("Write a number:"))**

**Vol = 4.0 / 3.0 \* PI \* (r \* r \* r)**

**print (Vol)**

**main ()**

**End of Assignment 3.1!**

1. [↑](#footnote-ref-1)