Algonquin College Logo

# SCHOOL OF ADVANCED TECHNOLOGY

### ICT - Applications & Programming

### Computer Engineering Technology – Computing Science



A21

Model Definitions (RE/Automaton)

Lab Professor / Lab Session:

[Paulo Sousa / 012,011]

Team:

[Dhruv Hapani] - Id: [041080213] / [Shreya Gulati] - Id: [041076329]

Language Name [Gojo]

Algonquin College Logo

A screenshot of a video game

Description automatically generated with medium confidence

# Gojo Language LANGUAGE SPECIFICATION

1. **The Gojo 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**

**Gojo** LANGUAGE allows you to write both single-line and multi-line comments (using // and /\* \*/, respectively).

**<singleLinecomments>** → // This is a single-line comment in Gojo \n

**<multiLinecomments>** → /\*

This is a multi-line comment

in Gojo, spanning multiple lines.

\*/

* 1. **Variable Identifiers**

The scanner produces variable identifier tokens as **VID\_T** for variable identifier.

**<variable identifier>** → VID\_T

* 1. **Keywords**

The scanner generates a **KW\_T** token for Gojo language keywords, with the token's attribute indicating its index in a **keywordTable[],** which includes:

**default, interface, case, if, for, go, map, struct, else, type, import, goto, package, switch, const, continue**

* 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**

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

**<float\_literal>** → FLOAT\_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: **OP\_ADD, OP\_SUB, OP\_MUL, OP\_DIV**. 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: **OP\_EQ, OP\_NE, OP\_GT, OP\_LT**. 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: **OP\_AND, OP\_OR, OP\_NOT**. 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 Gojo LANGUAGE Syntactic Specification**
   1. **Gojo LANGUAGE Program**
      1. **Program**

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

**<program>**  **func main()** {

// Data session

// Here, we define and initialize your variables

var exampleInt int = 10

var exampleString string = "Hello, Gojo!"

// Code session

// This section contains the executable code

if exampleInt > 5 {

fmt.Println(exampleString)

} else {

fmt.Println("exampleInt is less than or equal to 5")

}

}

**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>

* + 1. **Assignment Statement**

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

* + 1. **Assignment Expression**

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

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

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

{ <opt\_statements> }

<optional else statement>

**<optional else statement>** → **else** { <opt\_statements> } ;

|<selection statement>

| ϵ

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

**<iteration statement>** → **for** <conditional expression> {

<statements>

}

* + 1. **Input Statement**

**<input statement>** → **println**(<variable list>);

**Variable List:**

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

**Variable Identifier:**

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

| <integer\_variable>

| <string\_variable>

* + 1. **Output Statement**

**<output statement>** → **fmt.println**(<opt\_variable list>); | **fmt.println**(STR\_T);

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

**<output statement>** → **print&** (<output statement Prime>);

**<output statement Prime>** → <opt\_variable list> | STR\_T

**Optional Variable List:**

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

| FPL\_T | INL\_T

| (<arithmetic expression>)

* + 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 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>

**Primary Arithmetic Relational Expression:**

**<primary a\_relational expression>** →<integer\_variable | FPL\_T | INL\_T

References

|  |  |
| --- | --- |
|  | Chat GPT :- <https://chat.openai.com> |