**Solita Programming Language**

Submitted by:

Lim, Ciana

Soria, Spencer

Tan, Luis

Solita Programming Language is a bit like C, but uses different keywords, and has a slightly different structure.

Special keywords must be typed in all uppercase letters. This is to easily differentiate it with the variables used. White spaces will be ignored when tokenizing.

Requirements to use Solita:

1. Must be able to run at least Python 3.4
2. Must have PLY library installed

To install the PLY library to Python:

1. Copy the ply folder inside the extracted ply-3.10 folder
2. Paste this folder inside the Python folder (i.e. C:\Python34)

To run Solita, simply type python solita.py <filename>

Solita Programming Language supports:

1. Variables

Variables are to be declared starting with a lowercase letter, followed by either a lowercase letter, an integer, or an underscore. Uppercase letters are not allowed.

To declare variables, they must be declared at the start. They must be declared like so:

VARIABLES {

INTEGER ivar

FLOAT fvar

STRING svar

ARRAY avar

}

where *ivar* is an integer variable, *fvar* is a float variable, *svar* is a string variable, and *avar* is an array variable. An array need not be declared with its type and size, because it is dynamic.

Initializations are not allowed during declaration.

Variable declarations cannot be empty like so:

VARIABLES { }

However, variable declarations are not required.

To initialize or assign variables after declaration, it will be like so:

ivar = 1

fvar = 0.1

svar = “hello world”

For string literals, Solita detects “hello” + “world” as one string literal hello” + “world.

For variables of type array, see 4. Array Operations

1. Integer and Float Operations

The integer and float operations supported are the 4 basic arithmetic operations (+-\*/), power (^), and modulo (%).

To use these operations, the statement must look like something like this:

ivar = ivar + 1;

where the operand can either be +, -, \*, /, ^, or %.

Variables that are declared as a float can store integer numbers, and variables declared as an integer can store float numbers, floored down.

Precedence follows the regular precedence rules.

1. String Operations

A number of string operations are supported in this language such as string concatenation, getting the string length, searching for a substring and getting a character of a string by index.

String concatenation can be done as follows:

newstring = SAMA(string1, string2)

The SAMA function accepts two respective parameters of string type. string2 is appended to the end of string1. The resultant string is returned and is stored to the newstring in the example given.

However, a limitation to the SAMA function is that it cannot accept two string literals. At least one of the parameters have to be a variable of string type.

To get a string’s length, do this:

len = HABA(string)

The HABA function accepts a parameter of string type. The length of string is returned as an integer and in the example given is stored into the variable len.

To search for a substring, do this:

var = HANAP(longstring, substring)

The HANAP function accepts two parameters of string type. The longstring is searched if the substring is contained in it. The integer 1 is returned to var if the substring is found and 0 otherwise.

However, a limitation to the HANAP function is that it cannot accept two string literals. At least one of the parameters have to be a variable of string type.

To get a character from a string by index, do this:

string1 = SA(string2, index)

The SA function function accepts two parameters of string and integer type respectively. It returns the index-th character of string2 in string form. In the example given, it is stored in string1.

1. Array Operations

There are a number of array of operations we can do such as accessing by index, assigning values by index, appending to an array, getting the length of an array, and deleting an array element by index.

We can access an array element by index as follows:

var = array[index]

We can assign values to an array element by index as follows:

array[index] = var

We can append to an array as follows:

DAGDAG(array, element)

We can get the length of an array as follows:  
 var = HABA(array)

We can also delete an array element by index as follows:

BURA(array, index)

1. I/O Operations

To print into the console, do this:

PAKITA(“string”) \*\* to print string literals \*\*

PAKITA(var) \*\* to print variable values \*\*

To get input from terminal:

BASA(var)

Note that the user input will be stored into var and will immediately be checked if its actual type satisfies the expected type of the variable.

1. Loops

Loops and iterations can be done in two ways. One is by the use of HABANG loop, This can be done by this syntax :

HABANG (condition){ \*\* while <condition> {} \*\*

\*\* code here \*\*

}

The way it looks and the way it function is just the same as how the while loop in c works. Everything inside the { and } is the code block to be iterated while the logical <condition> stated is still satisfied.

1. Conditionals

Conditionals in our language will work like the conditionals in C, just with a different syntax. Conditionals is used through this syntax:

PAG (condition){

\*\*codeblock

}

KUNDIPAG (condition){

\*\*codeblock

}

KUNDI{

\*\*codeblock

}

PAG statement is equivalent to the if statement in C while the KUNDIPAG statement is equivalent to the else if statement and the KUNDI statement is equivalent to the else statement.

1. Conditions

Logical conditions are handled like how they are done in other programming languages. The syntax, uses and examples are stated below :

|  |  |  |
| --- | --- | --- |
| Solita Syntax | Use | Example |
| == | To test if a is equal to b | a == b |
| >= | To test if a is equal or greater than b | a >= b |
| <= | To test if a is equal or less than b | a <= b |
| != | To test if a is not equal to b | a != b |
| > | To test if a is greater than b | a > b |
| < | To test if a is less than b | a < b |

These logical statements returns only two values, True or False.

Logical conditions can handle integers, floats, and strings so long as the lefthand side and the righthand side are both the same data type.

1. Type Checking for all

Type checking will always be done. Syntactical errors are guaranteed to be checked. The error details will be printed in the terminal. Since it is an interpreted language, the error can only be identified during runtime and when it is encountered, the program will terminate after notifying the user.