LepLang User Manual

Gustavo Hernández Sánchez

Luis Miguel Maawad Hinojosa

Compiler Design (Group 1)

OBJ 11

June 2nd 2021

Data Types

boolean	Boolean Type can be true or false.
int	4-byte signed integer.
float	Single-precision fraction, 6 significant figures.
string	An array of bytes representing Unicode characters.

Operators

+ - / *	Arithmetic operators
> != < ==	Relational operators
and	Logical AND operator
or	Logical OR operator

Reserved Words

class	public	blocked	main
func	void	int	float
string	bool	print	read
if	else	loop	true
false	and	or	return
program	interface	implements	

Statements

A statement must be defined on a single line.

More than one statement can be on a single line.

Comments

Comments start with “/*” and end with “*/”.

LepLang accepts multiline comments.

Console Output

LepLang programs can output simple messages to the console as follows:

```
print("Hello World");
```

To print numeric values, they need to be converted to a string, this is done by adding an empty string:

```
print(3 + "");
```

Console Input

LepLang supports user input, to store the value, a variable needs to be declared before:

```
int userInput;
read(userInput);
```

Important note: the input and the destination variable should be of the same data type.

Control Flow

If-else statement

If statements are formed as follows, else block is optional.

```
...
if (a > b){
  print(a + “”);
}else{
  print(b + “”);
}
...
```

Loop

The conditional loop is formed as follows. It can be read as: “loop until the condition is true”.

```
...
loop(a < b){
a = a + 1;
}
print(a + "");
...
```

Functions / Modules

Function declaration

Functions can only be declared inside a class.

The keyword func is required and followed by the data type and variable name.

The parameters of the function are declared inside the parenthesis.

For returning value functions, only one return statement is allowed.

Returning value functions and void functions are declared as follows:

```
...
/* Void Function Declaration */
func void sayHello(){
   print("Hello There!");
}

/* Returning Value Function Declaration */
func int getDouble(int a){
   return a * 2;
}

/* Using a single return statement */
func int solveRecursive(int n){
  int result;
  if(n == 0){
    result = 1;
  }
  else{
    result = n*solveRecursive(n - 1);
  }
  return result;
}
...
```

Function call

Function calls depend on the return type of the function.

Void function calls can only be called outside expressions.

Return type functions can only be called inside expressions.

Both types of function calls are structured as follows.

```
...
/* Void Type Function Calls */
array.sort()
/* Return Type Function Calls */
int i  = factorial.solveLinear(n);
print("Factorial Linear is: " +  factorial.solveLinear(n));
...
```

Arrays

LepLang supports multidimensional variables, limited to two-dimensional arrays.

Arrays are created and manipulated as follows.

```
...
/* Array declaration */
int a[2][2];

/* Array initialization */
a[0][0] = 5;
a[0][1] = 4;

/* Array value access */
print("The result is: " + (a[0][0] + a[0][1] * 2));
...
```

Classes

Class Structure

Each class declaration is composed of its visibility, the keyword class, and its id.

Inside a class, attributes can only be declared or assigned. Class modules must be declared at the end of each class. Further details are expressed below.

```
...
/* Class declaration */
public class Animal{
       /* Attributes declarations */
        int legs = 0;

       /* Module declarations*/
        func int getLegs(){
            return legs;
        }
}
...
```

Visibility

LepLang implements a different type of visibility from a traditional programming language. Visibility can either be public or blocked. Public visibility behaves similarly to other programming languages, where all attributes are accessible. Blocked visibility prevents the extension of the current class. An example of a blocked class declaration is detailed below.

```
...
blocked class Factorial{
        func int factorialLinear(int a){
            int res = 1;
            int i = 1;

            loop(i < a + 1){
                res = res * i;
                i = i + 1;
            }

            return res;
        }
}
...
```

Extension

Classes can be extended similarly to regular oriented programming languages, where the attributes and methods are passed from the parent class to the child class. The class extension is structured as follows where the class Rino extends from Animal.

```
...
/* Parent Class*/
public class Animal{
    int legs = 0;

    func int getLegs(){
        return legs;
    }
}
/* Child Class: attribute legs can be modified */
public class Rino : Animal{
    legs = 4;
}
...
```

Object Call

Objects can be instantiated inside a class or the main section. The declarations and access to attributes and modules are as described in the following code.

```
...
public class Animal{
    int legs = 0;

    func int getLegs(){
        return legs;
    }
}
main(){
    Animal duck, spider;
    spider.legs = 4
    print("Duck total legs: " + duck.getLegs());
}
...
```

Interfaces

LepLang supports the creation of interfaces that may be declared at the beginning of a program as follows.

```
program InterfaceExample{
    interface Legs{
        func int getLegs();
    }
...
```

To implement a class the keyword implements and the interface name is required after the definition of the class id. An example of this implementation is described in the following code.

```
program ObjectTest{
    interface Legs{
        func int getLegs();
    }

    public class Animal implements Legs{
        int legs = 0;

        func int getLegs(){
            return legs;
        }
    }
...
```

Program Structure

A program is made up of a series of interfaces, classes that may be present or not and a section named main. This last section is the entry point for every program after the definition of global variables. An example of a complete structure of a program is described in the following code.

```
program ObjectTest{

    /* Global Variables */
    int total = 0;

    /* Interface Definitions */
    interface Legs{
        func int getLegs();
    }
    /* Class Definitions */
    public class Animal implements Legs{
        int legs = 0;

        func int getLegs(){
            return legs;
        }
    }
    public class Spider : Animal{
        legs = 8;
        string venom = "Medium";
    }
    public class Tarantula : Spider{
        venom = "Low";
    }
    public class Monkey : Animal{
        legs = 2;
        float popularity = 0.95;
    }
    public class Zoo{
        Monkey mon;
        Tarantula tr;

        func void printAnimals(){
            print("Monkey legs: " + mon.getLegs());
            print("Monkey popularity: " + mon.popularity);
            print("Tarantula legs: " + tr.legs);
        }
    }
    /* Main Section */
    main(){
        Zoo zoo;
        zoo.printAnimals();
    }
}
```