The L+ Programming Language

Language Designed by:
Leonardo Val & Matías Hernandez

Grammar and Implementation by:
Alejandro Segovia & Emilio Pombo

July 15, 2011

1 Introduction

This document describes the L+ programming language from a developer's point of view. It is not a document to learn how to program. Throughout this document, it is assumed that the reader is familiar with the C programming language or other C-based languages like C++, Java or even C#.

2 Keywords

void	int	float	bool	string
true	false	if	else	while
return				

3 Data Types

L+ is a strongly-typed programming language. It includes a type system composed of the following types:

1. *int* – Integer data type.
2. *float* – Single precision data type for storing real numbers.
3. *string* – Built-in string type for managing character sequences.
4. *bool* – Boolean data type (true/false).

There is no automatic conversion between types, except for operations involving a *float* operand and an *int* operand. In this case the *int* operand is temporarily "promoted" to *float* to perform the operation. The result is always a *float*[1].

No other data types are promoted in L+ operations. Conversions must be done explicitly using the standard library's conversion functions.

4 Variables and Scoping

Variables can be declared to be of any of the previously defined data types. L+ does not include any mechanisms to define new data types (*structs*, *classes* are not supported).

There are two types of variables in L+: global and local. Global variables are defined at file scope (not contained in any functions) and are visible to all functions defined after said variable.

Local variables are defined inside a function and are only visible to said function. Local variable scoping rules in L+ are very similar to C's; Furthermore, L+ allows the programmer to define arbitrary scopes where variables may be redeclared with a different type.

When using a variable that exists at different scopes, the innermost one is always the one referenced.

Consider the following program:

```
int number = 1; // Ok, number is an int.

{
    string number = "One"; // Ok as well, number is a
                           // string.
    print(number); // Prints "One"
}

print(number); // Compile error, number is an int and print
               // must receive a string parameter as
               // argument.
```

[1] This rule is known as *type promotion*.

Finally, local variables can be initialized upon declaration, just like in C. Global variables cannot be initialized upon declaration.

5 Control Structures

L+ offers a range of control structures that help you control program flow.

The structures for the current version include:

1. *if* – Branching construct.
2. *else* – Branching construct for alternative paths.
3. *while* – Looping construct.

The bodies of the *if*, *else* and *while* constructs define new scopes where variable names may overlap with variables previously defined in an outer scope.

6 Functions

L+ allows declaring functions in a similar fashion to C.

```
type function_name ( [optional parameters] )
{
    (statements)
}
```

Functions may receive zero or more parameters. Each parameter must specify its data type.

All functions must return a value, except when the return type is declared as the special type: *void*.

Functions are called using their name and specifying the arguments. If no arguments are to be passed to the function, the programmer must use an empty parameter list: "()"[2].

[2]Please refer to the L+ Grammar.

All L+ programs must contain a special *main* function with the following signature:

```
void main()
{
    (optional statements)
}
```

This function declares the entry point for the program.

7 The L+ Standard Library

The L+ standard library provides a series of convenience functions to convert between data types and to perform I/O. The default functions included in the library are:

1. *void print(string message)* – Print a string to stdout.
2. *string read()* – Read a string from stdin. This function buffers characters until a newline character is read.
3. *int trunc(float value)* – Truncate a floating point number.
4. *string toString(float value)* – Convert a floating point number to a string. Because of int-to-float promotion, this function works on integer data types as well.
5. *int toInt(string str)* – Convert a string data type to an integer data type.
6. *float toFloat(string str)* – Convert a string type to a floating point real number.

8 The L+ Grammar

This section presents the L+ Grammar [3].

The starting symbol is "program". NUM, REAL and STRING are lexer constructs. NUM refers to an integer number, REAL to a floating point number and STRING to any character sequence contained between quotes.

```
program :: function
        | function program
        | declaration
        | declaration program

function :: type <id> ( args ) body
         | type <id> () body

declaration :: type <id> ;
            | type <id> = exp ;

args :: type <id>
     | type <id> , args

params :: exp
       | exp , params

exp : NUM
    | REAL
    | ( exp )
    | exp + exp
    | exp - exp
    | exp * exp
    | exp / exp
    | true
    | false
    | ! exp
    | exp == exp
    | exp != exp
    | exp || exp
    | exp && exp
    | exp < exp
    | exp <= exp
    | exp > exp
```

[3] The grammar is specified in the *samp.y* configuration file in the L+ source code.

```
    | exp >= exp
    | STRING
    | fcall

stmtlst ::
        | stmt stmtlst

stmt :: declaration
     | if ( exp ) stmt
     | if ( exp ) stmt else stmt
     | while ( exp ) stmt
     | body
     | fcall ;
     | <id> = exp ;
     | return exp ;

body :: { stmtlist }
     | { }

fcall :: <id> ( params )
      | <id> ( )

type :: BOOL
     | INT
     | FLOAT
     | STRING
     | VOID
```

9 Putting it all Together: a complete example

This program reads a number from stdin, calculates its corresponding Fibonacci number and prints it to stdout.

```
// Calculate the Fibonacci number for n
int fibo(int n)
{
    if (n == 0)
    {
        return 0;
    }

    if (n == 1)
    {
        return 1;
    }

    return fibo(n-1) + fibo(n-2);
}

// Program fibo.lp
// Read a number, calculate the Fibonacci number for said number
// and print a message with the result.
void main()
{
    print("Enter number to calculate the Fibonacci number for:");
    string nStr = read();

    int f = fibo(toInt(nStr));
    string s = toString(f);

    print("The Fibonacci number for " + nStr + " is: " + s);
}
```