Language manual

20161033 -Yahnit Sirineni.
20161232- Vishnu Tej.

We have designed the programming language **'Leng01**' with the features.

Data Types: {Signed and Unsigned integers, char, bool, 1D and 2D arrays; **Arithmetic Operations**: {add, sub, mul, div, modulo}; **Boolean Operations**: {and, or, not}; **Control Statements**: {if-then, if-then-else, for loop, while loop, break}; **Functions**: {call by value mechanism and Recursion support} and **I/O Routines:** {scan and print for stdio and files}

Syntax and Semantics:

i) Comments and spaces:

White spaces or tabs or end of lines can occur anywhere between tokens. Our language is **not space-sensitive** like **'C'.** Comments are started by // and are terminated by the end of line.

ii) main() function and function parameters:

All other functions are called by the main function from where the code execution begins. The functions are to be passed with parameters/variables either declared locally in a function or there are 'Global variables' which can be by any method without passing into the function.

iii) key words vs variables:

Variables are in upper-case or lower-case while key words are in lower-case only.

iv) Strings and char:

String consists of characters enclosed in double quotes. A character literal or char consists of a character enclosed in single quotes or double quotes.

v) I/O Routines :

scan and print for stdio; read and write to files.

vi) Integers , char, strings can take values like in C. Integers are 4 bytes and char are 1 byte in size.

Notation for CFG:

<A>	A is a non terminal symbol
foo	means that foo is a terminal i.e., a token or a part of a token
[x]	means zero or one occurrence of x, i.e., x is optional;
x*	means zero or more occurrences of x.
x+ ,	a comma-separated list of one or more x's.
{ }	large braces are used for grouping; note that braces in quotes '{' '}' are terminals. separates alternatives.

Macro syntax/CFG:

<start> -> <libraries> <global_decl> * <method_decl> *

<global_decl> -> <type> { <id> | <id> '[' <int_val> ']' } +

<method_decl> -> **def** <id> ([{ <type> <id>} + ,]:) <code>

<code> -> <var_decls> <statements>

<var_decls> -> { <type> <id> + ,; | <location> <assign_op> <expr> ;} *

<statements> -> <method_call> | <if_stat> | <loop> | **return** [<expr>] | **break** |**continue**;

<location> -> <id> | <id> '[' <expr> ']' | <id> '[' <expr> ']' '[' <expr> ']'

<method_call> -> <id> ([<expr> + ,]) |

<if_stat> -> **if** (<condition>) <code> { **else if** (<condition>)* <code>} [**else** <code>]

<loop> -> <for_loop> | <while_loop>

<for_loop> -> **for** (<id> = <expr> , <condition> <location><assign_op><expr>) <code>
End

<while_loop> -> **while**(<condition>) <code>

<declaration> -> <id> = <expr>

<condition> -> **ε** | <expr> <comp> <expr> | **not** <expr>

<expr> -> <location> | <method_call> | <literal> | <expr> <op> <expr> | <ternary> |

<ternary> -> <expr> ? <expr>:<expr>

<op> -> < **arithmetic** > | < **comp** >

<literal> -> **ε** | <**int_val**> | <**char_val**> | <**unsign_val**> |

Micro syntax:

<int_val> ->	[-] {0-9}{0-9} *
<unsign_val> ->	{0-9}{0-9} *
<char_val> ->	'<char>'
<bool_val> ->	**true|false**
<type> ->	**int|uint|bool|char**
<id> ->	{a − z; A − Z} {a − z; A − Z; 0 − 9} *
<arithmetic> ->	+ | - | * | / | %
<assign_op> ->	= | += | - =
<comp>	!=| > | <

Semantics:

i) Defining variable and types:

Syntax: <type> <variable name>

Variables names are at least one character long and the first character should be alphabet followed by any combination of alphanumerics.

For integer arrays or char arrays,

Syntax 1D array:<type> <variable name> <[size]>, where size is an integer denoting size of array.

Syntax 2D array: <type> <variable_name> <[M]><[N]>, M,N are integers denoting rows and columns

ii) Functions :

Syntax: **def** <type> <name> (input_parametes+)

The code following consists of return <variable> at the end ; A function may not return anything which is indicated by return ;

Operations :

-For characters if we do any of the operations (ex: 'A'-'B') it takes ASCII values of the characters.

-We can only perform equal and not equal operations on boolean variables. Only '&& ' , '||', '~' operations on boolean variables can be done.

- For rest of the data types we can do all the operations.

Control statements:

If:

The code following if() block executes only if the condition in the parenthesis of the if block holds true.

If elseif else:

As mentioned in the CFG there can be zero or more number of 'else if' blocks followed by an else block. The code pertaining to each block executes only if the condition in parenthesis holds true. If none of 'if'/'else-if'condition holds true, then else block executes.

While loop

Syntax: **while** followed by ( )

The condition in the parenthesis is to be checked and the loop executes indefinitely while the condition holds true.(If no condition is mentioned it is assumed to be true always).

for loop

Syntax**:** for(initialization; condition; operation on variable)

The conditions are similar as in the case of 'C' and any of the three can be left empty as well.

Additional Semantic checks:

All the rules for a programming language cannot be specified through grammar/macro syntax or micro syntax. These additional semantic checks are to be done by the compiler to output error messages.

- To check each identifier is only declared once in each function.
- No identifier is used before it is declared.
- Making sure that the libraries included are valid/defined.

- The <int_val> in an array declaration must be greater than 0.
- The number and types of arguments in a method call must be the same as the number and types of the formals, i.e., the signatures must be identical.
- If a method call is used as an expression, the method must return a result.
- The expression in a return statement must have the same type as the declared result

type of the enclosing method definition.

- The <expr> in an if statement must have type boolean.

- All break and continue statements must be contained within the body of a for.

- The type variable returned and type of variable into which output is operated should be Same.

- Making sure the types of arguments of functions in all function calls are correct, ex. An integer should not be passed where a string is required.

- To verify there are no infinite loops in the code where many functions call each other endless without properly defined terminating conditions.