Language Specification Document

Muhammad Danish

Language Name: **Pedestal**

Extension of Pedestal: .pd

PREFACE:

Table of Contents

Introduction

Pedestal is a programming language that is designed in a Contrary approach to esoteric languages.

But what are esoteric languages? An esoteric programming language (sometimes shortened to esolang) is a programming language designed to test the boundaries of computer programming language design, as a proof of concept, as software art, as a hacking interface to another language (particularly functional programming or procedural programming languages), or as a joke.

Hello world program in some of the languages are written as,

Esoteric Language Name	Code
Acronym	{{>>{~~~~{-<}~~~~~~~~~~{-<-<}}<<} </(<<<){[<]}:>:{>>{~~~~~~~~{<}~{>}}<<}\ ~>{{~{v}}>>>v{~}^<<<}/(<<<){[<<]}:>:{>>{~~~ ~~~~~{<<}~{>>}}<<}\ ~{>>{vvvvvvvv~~~~~~~~~~~~~~~}<<}~{>>{v vvv~~~~~~~~~~~~~~}<<}~ {>>{^^^^^^^^^^~}<<}~{>>{v~~~~~~~{{<<}~} v{~}vvvvvvv{~{>>}}^^^^^^~~~ {{<<}~}vvvv{~{>>}}v~~~}<<}~<{{^^^}}~ {>>{vvvvvv~{{<<}~{>>}}^^^^^^~~~~~~~~~~ ~{{<<}~{v>}}^^^^^^}} /{{()}}{[<<<<]}:>:{{~v}}\}
ArnoldC	IT'S SHOWTIME TALK TO THE HAND "hello world" YOU HAVE BEEN TERMINATED

On the other hand, Pedestal syntax is similar to natural English but designed with programming fundamentals in mind. It is best starter language for school kids and new learners to programming.

The Learner will not feel conceptual hurdle when shifting to a higher language like C++. Because pedestal syntax is designed in a way as a teacher explains the code to the student.

Motive

As esoteric languages are a fun way to push the boundaries of language design, but most of the time it does not benefit to anyone. Because they are complex and far different from higher level programming syntax.

For example in **Befunge**(an esoteric language), we print hello world like this.
0455**:3/\:8+:6+:3-:8+48*:66++:3*37*-:3-::7-89*>:#,_@

In contrast Pedestal provides a stepping stone to the fresh learners. It provides Conceptual understanding of code while you are coding it. Flow control of program is well understood and users will get quick grasp on flow charts and code execution.

Lexical Components

a) **Rule for identifier**

1. Identifier is any name given to the element in program, to a variable, function name or class name etc. The name can range from 0-9 | A-Z | a-z.
2. No keywords can be name of identifier.
3. Identifier must start from a letter.
4. Integer Constant is a sequence of one or more digits.
5. String constant is a series of characters, digits, spaces, or escape sequences between “.
6. Escape sequence starts with \

b) **Data Types**

Pedestal includes programming fundamental data types. Pedestal only includes primitive data types.

- Int
- Float
- String

c) **Keywords**

Pedestal includes keywords that best defines the concepts of code control flow. Hence, they are described in natural language (English only.)

All keywords start with capital letters and follow camel case.

Keyword	Derivation
RepeatUntil	For
Show	Print
IntegerContainer	Int
StringContainer	String
floatContainer	float
Update	i++

d) **Operators**

keyword	derivation
Add	+
Sub	-
IsLessthan	<

e) **Punctuations and Escape Sequences**

keyword	derivation
NewLine	\n
Tab	\t

Punctuations
(
)
{
}
[
]
;

Syntax Literals

Declarations:

a) ***integer***	Put INT into integerContainer ID ;
b) ***float***	Put FLOAT into floatContainer ID ;
c) ***string***	Put “STRING_LITERAL” into stringContainer ID ;

Loops

a) **for loop**	repeatUntil (ID isLessThan INT) { body.. }

Print statement

a) **show**	show(“printStmt ” , str) ;

Operations

a) **add**	add ID and ID into ID;
b) **sub**	sub ID and ID into ID;
c) **update**	updateInc ID;

Functions

Define void Function_name() { }

Define as

Define void Function_name();

called as:

Function_name();

Code Example

C++	Pedestal
Int i=0; ***Int n=10;*** ***Int a=0;*** ***For(i=0; i<n; i++)*** ***{*** ***a++;*** ***}*** ***cout << a;***	Put 0 into IntegerContainer i, Put 0 into IntegerContainer n, Put 0 into IntegerContainer a, RepeatUntil i IsLessThan n, Update a, Show a,

g) Grammar:

A pedestal program is divided into three parts a header, body section and a footer. Header and Footer part are necessary for the program to run, while the body part can be empty. The body part contains none, one or many statements and these statements can be deceleration, loops, print statements or for loop. Keeping this in mind CFG of pedestal is designed.

The parsing in implement from left to right and it is same as implemented in C++ parser we coded.

The body production of this CFG can be used in any body of a statement for example, inside for loop body or in main body of program.

Tokens are created by the flex file and then bison is generating the parse tree and mapping them to CFG. If a syntax error is occurred the parsing is halted with the error code.

Continued…..

// **capital letters are terminals and small letters are productions**

//**pedestal:** header body_section footer

//**header:** PEDESTAL_START STRING SEMICOLON

//**body_section:** body_statements | null;

//**body_statements:** body_statement | body_statements body_statement;

//**body_statement:** declerations | loops | operations | printstatements;

//**declerations:** integer_dec | float_dec | string_dec;

//**loops:** for_loop;

//**operations:** addition | subtraction | updation;

//**int_dec** - > PUT INT INTO INT_CON ID SEMICOLON

//**float_dec** - > PUT FLOAT INTO FLOAT_CON ID SEMICOLON

//**string_dec** -> PUT STRING INTO STR_CON ID SEMICOLON

//**for_loop** -> FOR_LOOP_KEYWORD OPEN_BRACKET_ROUND IDENTIFIER FOR_LOOP_COND INT CLOSE_BRACKET_ROUND OPEN_BRACKET_CURLY body_section CLOSE_BRACKET_CURLY

//**addition** -> ADD IDENTIFIER AND IDENTIFIER INTO IDENTIFIER SEMICOLON

//**subtraction**: SUB IDENTIFIER AND IDENTIFIER INTO IDENTIFIER SEMICOLON

//**updation:** UPDATE IDENTIFIER SEMICOLON

//**printstatements:** PRINT OPEN_BRACKET_ROUND STRING COMMA IDENTIFIER CLOSE_BRACKET_ROUND SEMICOLON

//**footer:** PEDESTAL_END STRING SEMICOLON

f) FLEX Rules:

"pedestal start" {yylval.identifier = strdup(yytext); return PEDESTAL_START;}

"pedestal end" {yylval.identifier = strdup(yytext); return PEDESTAL_END;}

[0-9]+ {yylval.integer_num = atoi(yytext); return INT; }

[0-9]*"."[0-9]+ { yylval.float_num = atof(yytext); return FLOAT; }

\"(\\.|[^"\\])*\" {yylval.string_literal = strdup(yytext); return STRING;}

"into" {yylval.identifier = strdup(yytext); return INTO;}

"put" {yylval.identifier = strdup(yytext); return PUT;}

"integerContainer" {yylval.identifier = strdup(yytext); return INT_CON;}

"floatContainer" {yylval.identifier = strdup(yytext); return FLOAT_CON;}

"stringContainer" {yylval.identifier = strdup(yytext); return STRING_CON;}

"repeatUntil" {yylval.identifier = strdup(yytext); return FOR_LOOP_KEYWORD;}

"and" {yylval.identifier = strdup(yytext); return AND;}

"show" {yylval.identifier = strdup(yytext); return PRINT;}

"isLessThan" {yylval.identifier = strdup(yytext); return FOR_LOOP_COND;}

"add" {yylval.identifier = strdup(yytext); return ADD;}

"sub" {yylval.identifier = strdup(yytext); return SUB;}

"updateInc" {yylval.identifier = strdup(yytext); return UPDATE;}

[a-zA-Z_][a-zA-Z0-9_]* {yylval.identifier = strdup(yytext); return IDENTIFIER;}

[;] {yylval.identifier = strdup(yytext); return SEMICOLON;}

[(] {yylval.identifier = strdup(yytext); return OPEN_BRACKET_ROUND;}

[)] {yylval.identifier = strdup(yytext); return CLOSE_BRACKET_ROUND;}

[{] {yylval.identifier = strdup(yytext); return OPEN_BRACKET_CURLY;}

[}] {yylval.identifier = strdup(yytext); return CLOSE_BRACKET_CURLY;}

[,] {yylval.identifier = strdup(yytext); return COMMA;}

[' '\t\n]+ { } **and** "//".* { } **and** [/][*][^*]*[*]+([^*/][^*]*[*]+)*[/] { } **and** . { yylval.identifier = strdup(yytext); return SYN_ERROR; }