Republic of the Philippines
POLYTECHNIC UNIVERSITY OF THE PHILIPPINES

PaL
(Python and Ladder)

Certeza, Diana Rose

Hernandez, Vince Ivan

Innovejas, Eden Grace

Mendoza, Rhabi Ezra

Ramos, Alexson

Ramos, Mark Ciedrick

BSCS 3 – 1N

I. INTRODUCTION

PaL is a high-level programming language designed with the aim of providing a child-friendly syntax. This is created for children from the age of six and up. It encompasses the basic concept of a programming language that will help in building the foundation of children in programming. The goal of this programming language is to provide a basic experience for the target users of writing simple instructions on computers.

The target users will have an easy and enjoyable learning experience through the simplified words and syntax used here. These words, adapted from the Python language, were customized using terms that children commonly used and easily understood. This can help them become familiar with the use of words here and visualize them by remembering how certain keywords and reserved words are used in reality.

The idea that programming will soon be as necessary as knowing how to read and write led the developers to create a programming language that can be used by children from age six and up. Programming languages function nearly the same as the languages that people would normally think of. Children are adaptive to their environment. They learned speaking, reading, and writing because of what was taught to them and because of what they could see and hear. The developers thought of the age of six as a good age to start learning a programming language because this is usually the age when they are so curious and open about things. They are like sponges, soaking up information and learning quickly. Also, children at this age are already exposed to technology. Learning a programming language while still young can be beneficial to a child, and they can master it when they grow up, just like they master reading and writing in the language they are used to.

PaL was named after the famous chance-based board game Snakes and Ladders. Since this language was mostly based on the syntax of Python, the developers decided to use Python instead of Snakes. PaL holds the principle of Don't Avoid the Snakes or errors and Avoid the Ladder or shortcuts. Errors and mistakes are inevitable in the learning and problem-solving process, especially in programming. Instead of avoiding errors, which is the goal of the game, view them as opportunities to learn and improve. Understanding why an error occurred and fixing it helps a programmer develop strong debugging skills.

PaL also comes from the term pal, which means friend or playmate. PaL aims to be a friend for the target users as they delve into the world of programming. PaL will serve as the target users' buddy in learning and experiencing the basics of programming, as it will be an easy and enjoyable language for them to use. The main goal of PaL is to allow children to learn to read and write code in a simple and straightforward manner. With its easy-to-read code syntax, the target users will be able to grasp and use the language PaL in learning and experiencing basic programming. The language was designed for the target users to have a medium that they can use to get to know programming, like knowing a pal or a friend. PaL is not intended for building software but is solely a medium for target users to understand the basic concept of programming and to gain experience. The language will only serve as a practice buddy for the target users.

II. SYNTATIC ELEMENTS OF LANGUAGE

1. Character Set

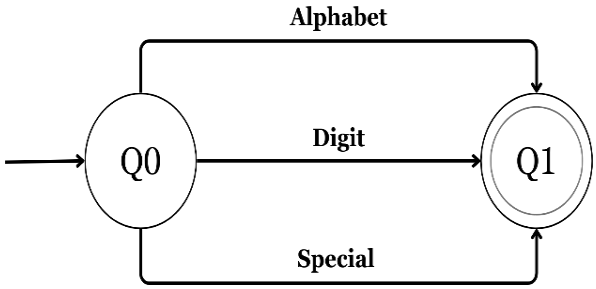
Characters = {Alphabet, Digit, Special}

Alphabet = {a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z}

Digit = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

Special = {$, _, ~, `, !, @, #, %, ^, &, *, (,), -, =, +, {, [, },], \, |, :, ;, ', ", <, ,, >, ., ?, /, ,}

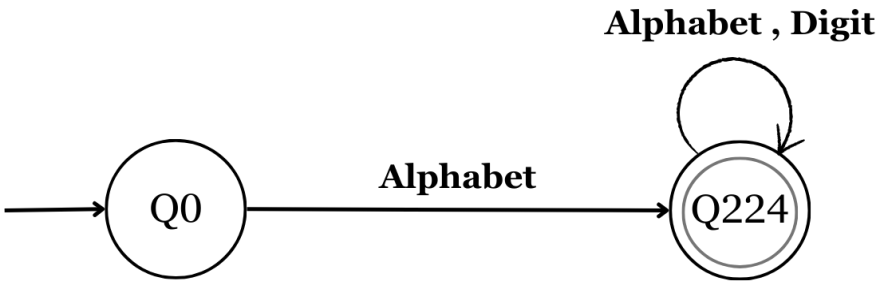
Machine for Character Set:



2. Identifier

- Rules:
  a. The identifier can only consist of alphabets which is either uppercase or lowercase.
  b. Identifiers are case sensitive.
  c. Keywords and reserved words cannot be used as an identifier.

Machine for Identifier:



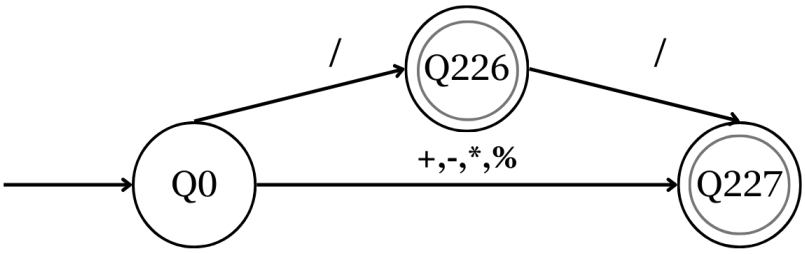
Regular Expression: Alphabet (Alphabet +Digits)*

3. Operation Symbols

OP_Arithmetic = {+, -, *, /, %, //}

OP_Arithmetic	Example Expression	Description
+ (Addition Operator)	X + Y	Adds the value of X and Y
- (Subtraction Operator)	X - Y	Subtracts the value of Y from X
* (Multiplication Operator)	X * Y	Multiplies the value of X by Y
/ (Division Operator)	X / Y	Divides the value of X by Y
% (Modulo Operator)	X % Y	Divides the value of X by Y and returns the remainder
// (Integer Division Operator)	X // Y	Divides the integer value of X by Y

Machine for OP_Arithmetic:



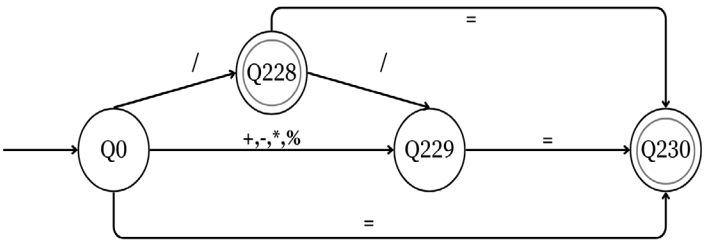
Regular Expression: **+** + **-** + ***** + **/** + **%** + (/(/))

OP_Assign = {=, +=, -=, *=, /=, %=, //=}

OP_Assign	Example Expression	Description
= (Assignment Operator)	X = 5	Assigns the value of 5 to variable X

+= (Addition Assignment Operator)	X += 5	Adds 5 to the current value of X
-= (Subtraction Assignment Operator)	X -= 5	Subtracts 5 to the current value of X
*= (Multiplication Assignment Operator)	X *= 5	Multiplies the current value of X by 5
/= (Division Assignment Operator)	X /= 5	Divides the current value of X by 5
%= (Remainder Assignment Operator)	X %= 5	Assign the remainder of dividing the current value of X by 5 to X
//= (Floor Division Assignment Operator)	X //= 5	Divides the integer value of X by 5 and assign it to X

Machine for OP_Assign:



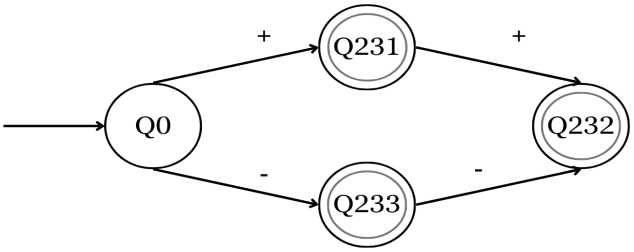
Regular Expression: = + (+ + - + * + / + % + /(/)) =

OP_Unary = {+, -, ++, –}

OP_Unary	Example Expression	Description
+ (Unary Plus Operator)	+X	Indicates that the value of X is positive

- (Unary Minus Operator)	-X	Indicates that the value of X is negative
++ (Increment Operator)	++X or X++	Prefix Increment Increments the value of x by 5 before using it in an expression Postfix Increment Increments the value of x by 5 after using it in an expression
-- (Decrement Operator)	--X or X--	Prefix Decrement Decrements the value of x by 5 before using it in an expression Postfix Decrement Decrements the value of x by 5 after using it in an expression

Machine for OP_Unary:



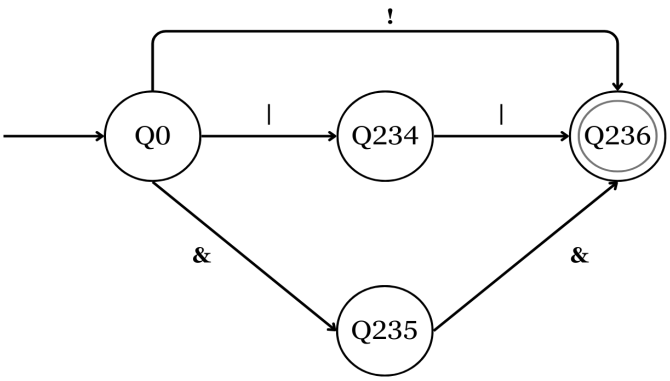
Regular Expression: + + **-** + ++ + **-(-)**

OP_LogBool = {!, ||, &&}

OP_LogBool	**Example Expression**	**Description**

! (Logical NOT Operator)	X=true !(X)	Returns the opposite value of the expression [!(X) returns false]
|| (Logical OR Operator)	X=1 (X>0||X< 4)	Returns a value true if one or more statements in a condition are true, otherwise, returns false [(X>0||X< 4) returns false]
&& (Logical AND Operator)	X=2 (X>0&&X<4)	Returns a value true if both statements in the conditions are true, otherwise, returns false [(X>0&&X< 4) returns true]

Machine for OP_LogBool:



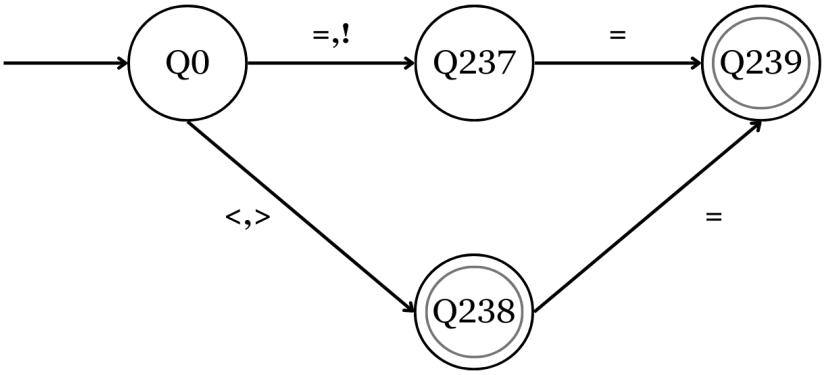
Regular Expression: **!** + || + **&(&)**

OP_RelBool = {==, !=. >, <, >=, <=}

OP_RelBool	**Example Expression**	**Description**
== (Equal To Operator)	X=2,Y=2 (X==Y)	Checks if the value of operands is equal then returns true, else returns false [(X==Y) returns true]
!= (Not Equal To Operator)	X=7,Y=7 (X!=Y)	Checks if the value of operands is not equal then returns true, else returns false [(X!=Y) returns false]

> (Greater than Operator)	X=10,Y=11 (X >Y)	Checks if the value of the left operand is greater than the value of the right operand then returns true, else returns false [(X>Y) returns false]
< (Less than Operator)	X=12,Y=13 (X<Y)	Checks if the value of the left operand is less than the value of the right operand then returns true, else returns false [(X>Y) returns true]
>= (Greater than or Equal Operator)	X=10,Y=10 (X>=Y)	Checks if the value of the left operand is greater than or equal to the value of the right operand, then returns true, else returns false [(X>=Y) returns true]
<= (Less than or Equal Operator)	X=4,Y=5 (X<=Y)	Checks if the value of the left operand is less than or equal to the value of the right operand, then returns true, else returns false [(X>=Y) returns false]

Machine for OP_RelBool:



Regular Expression: ((< + >) + (< + >) =) + (= + !) =

4. Keywords and Reserved Words

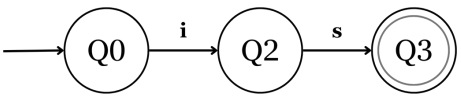
Control Flow and Repetition

Keywords and Reserved Words	Definition
is	Used to create conditional statements.

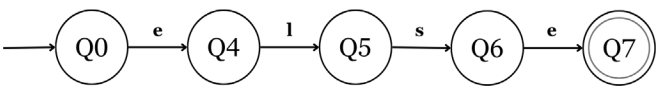
else	Used to create a default statement in an is statement.
iselse	Used to create additional conditional statements in an is statement.
for	Used to create for loops.
while	Used to create while loops.
countup	Create counting up statement.
countdown	Create counting down statement.
to	Indicate range of counting statement.
add	Create direct addition calculation.
sub	Create direct subtraction calculation.
multi	Create direct multiplication calculation.
cond	Indicate shorten conditional statement.
showsquare	Display a square.
showrectangle	Display a rectangle.
showtriangle	Display a triangle.
perisquare	Calculate square perimeter.
perirectangle	Calculate rectangle perimeter.
peritriangle	Calculate triangle perimeter.
arsquare	Calculate square area.
arrectangle	Calculate rectangle area.
artriangle	Calculate triangle area.

Machine for Keywords and Reserved Words of Control Flow and Repetition:

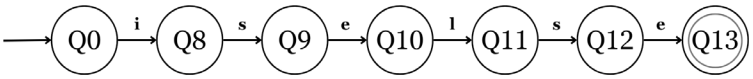
is



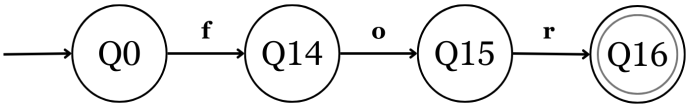
else



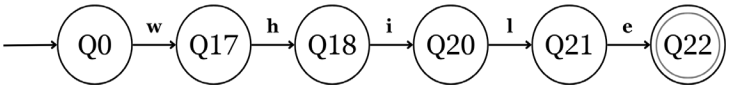
iselse



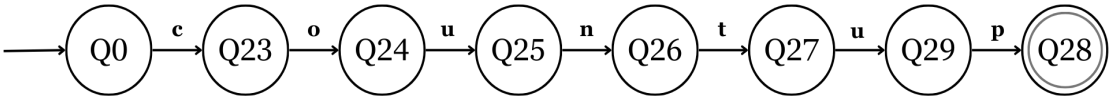
for



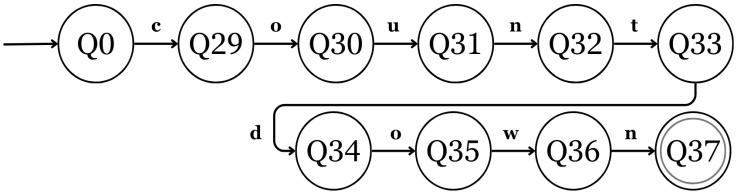
while



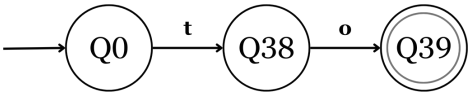
countup



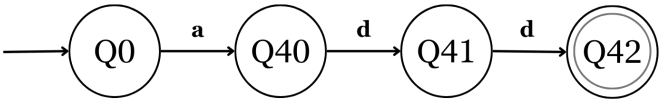
countdown



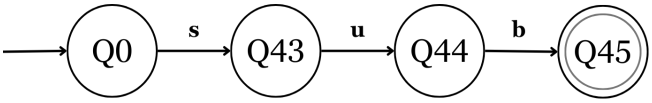
to



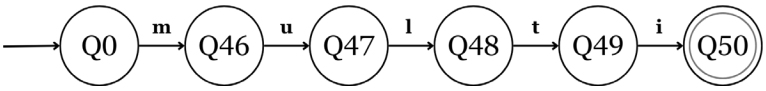
add



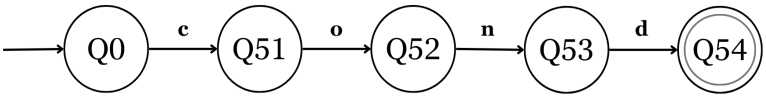
sub



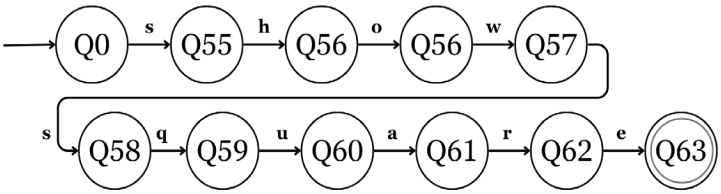
multi



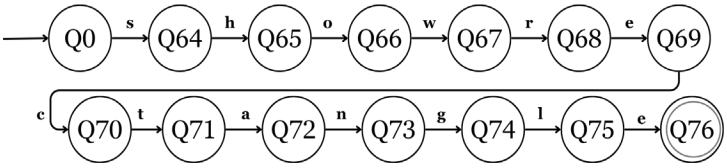
cond



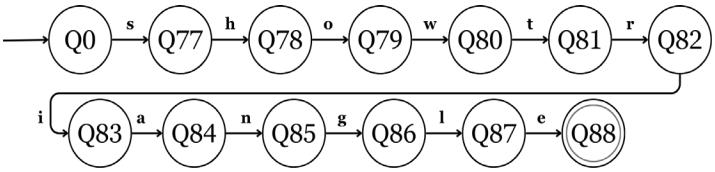
showsquare



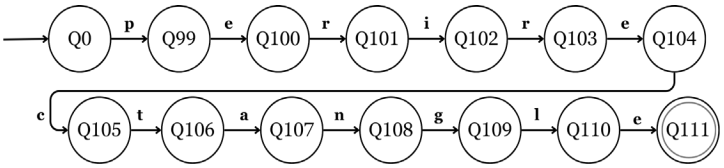
showrectangle



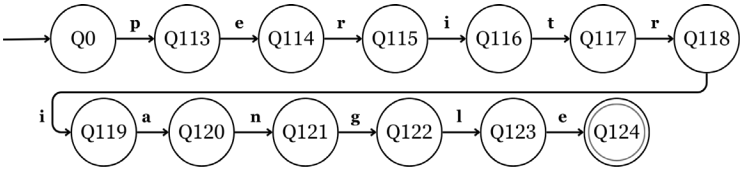
showtriangle



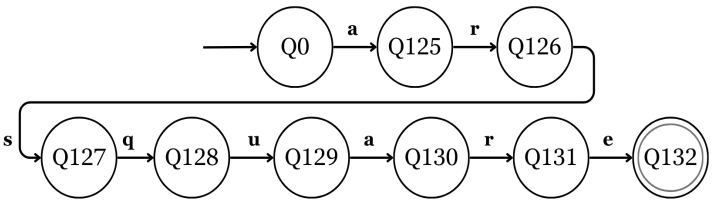
perisquare



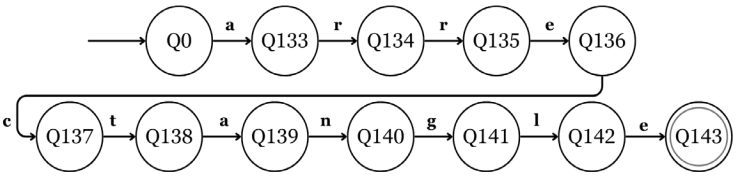
peritriangle



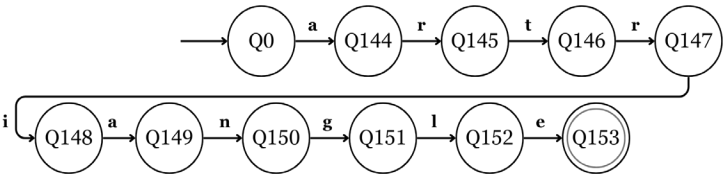
arsquare



arrectangle



artriangle

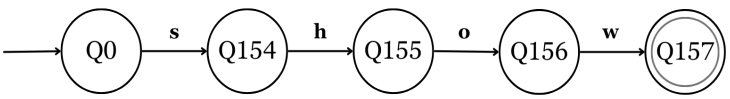


Output and Input

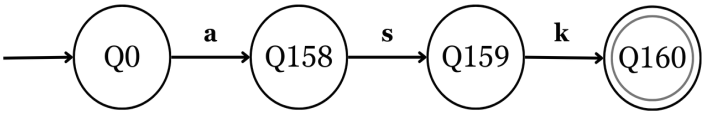
Keywords and Reserved Words	Definition
show	Used to print to the console.
ask	Used to get input from the console.

Machine for Keywords and Reserved Words of Output and Input:

show



ask

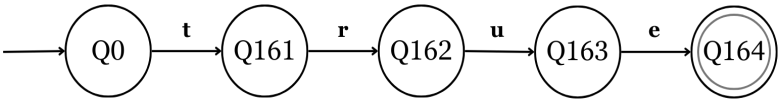


Value

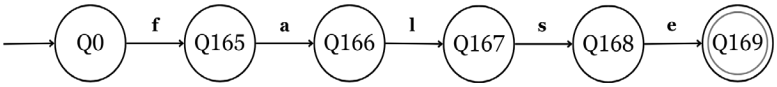
Keywords and Reserved Words	Definition
true	Value true.
false	Value false.

Machine for Keywords and Reserved Words of Value:

true



false

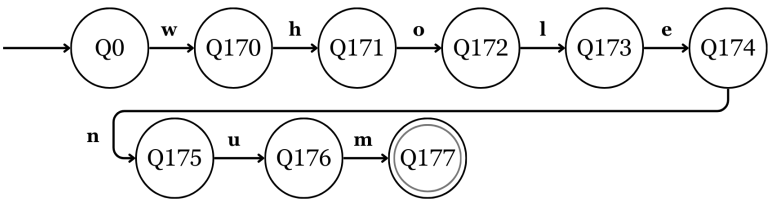


Data Types

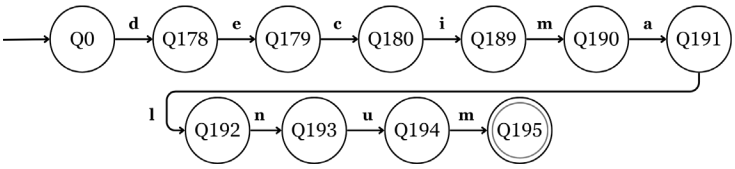
Keywords and Reserved Words	Definition
wholenum	Whole numbers.
decimalnum	Decimal numbers.
text	Collection of characters.
bool	True or false.

Machine for Keywords and Reserved Words of Data Types:

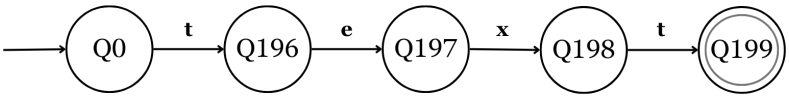
wholenum



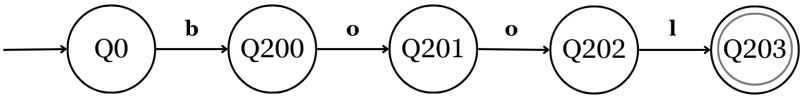
decimalnum



text



bool

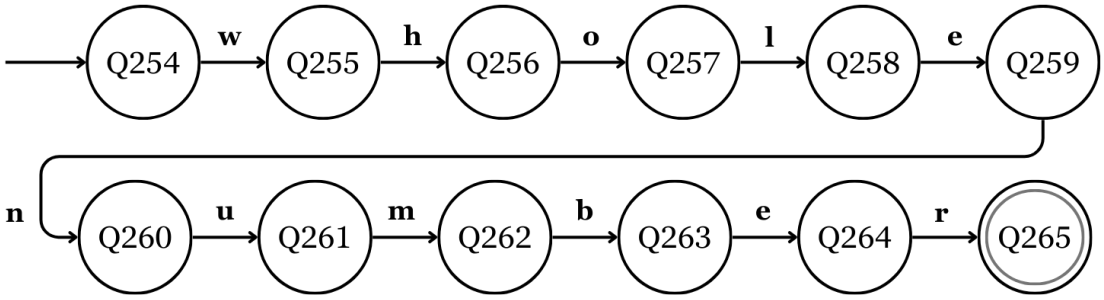


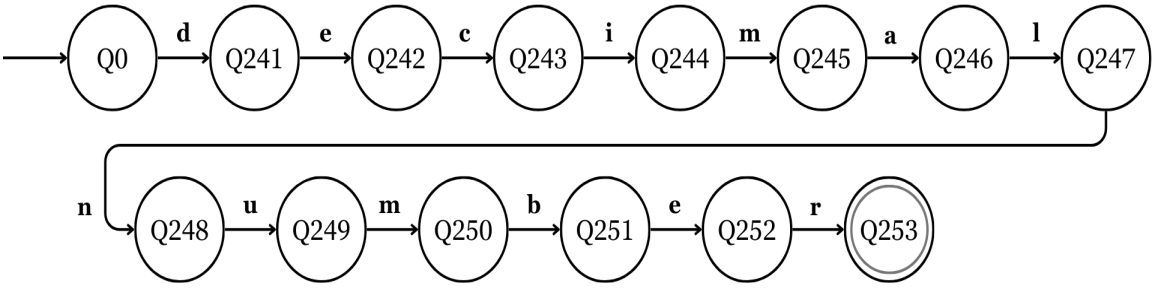
5. Noise Words

Noise Words	Shorthand	Original Notation	Definition
ber	wholenum, decimalnum	wholenumber, decimalnumber	The shorthand for the original notation of wholenumber and decimalnumber is wholenum and decimalnum. The noise word "ber" can be typed to make it easier to read and define.
ean	bool	boolean	The shorhand for the original notation of boolean is bool. The noise word "ean" can be typed to make it easier to read and define.

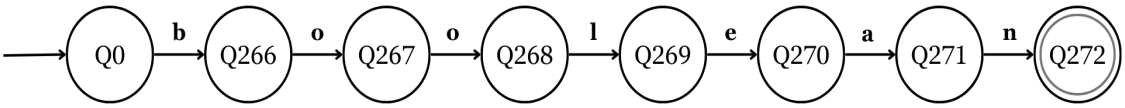
Machine for Noise Words:

wholenumber and decimalnumber (wholenum + ber and decimalnum + ber)





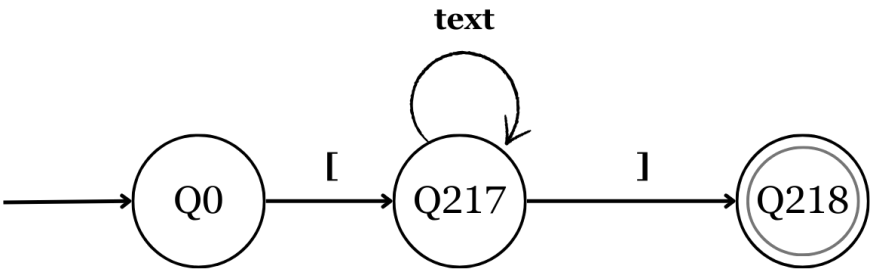
boolean (bool + ean)



6. **Comments**
   - Single or Multi-Line Comments

     [comments]

   Machine for Comments:

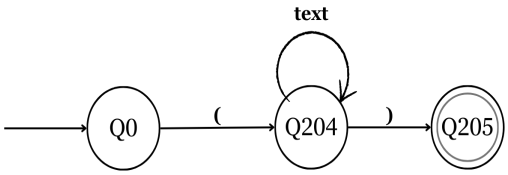


7. **Blanks**
   - Spaces are used for indentation in order to organize blocks of code, such as loops, conditional statements, and functions.
   - Spaces are used to distinguish various elements within a statement, like identifiers, operators, and keywords and reserved words.
   - Allow spaces within formatted strings to manage the alignment of text.
   - Enable the creation of multi-line comments or documentation strings.
   - Blank lines can be used to separate blocks of code, thereby enhancing readability.
   - Allow for the inclusion of spaces after reserved words and keywords to enhance code readability.

8. Delimiters and Brackets

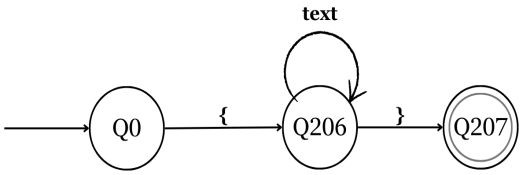
- **Parentheses (( ))** - used to enclose operations.

  Machine for Parentheses



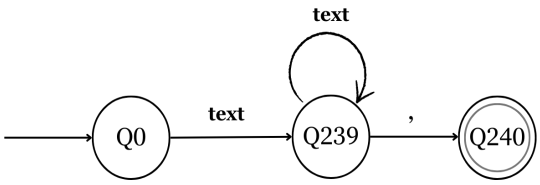
- **Curly Braces ({})** - used to enclose blocks of statements.

  Machine for Curly Braces



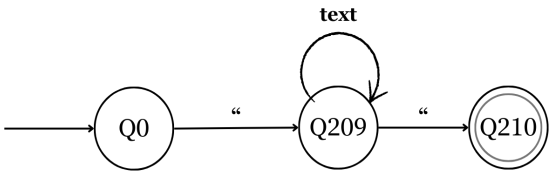
- **Comma (,)** - used to separate arguments and list items.

  Machine for Comma



- **Double Quote (“ “)** - used to enclose text data type characters.

  Machine for Double Quote



9. Free-Fixed-Field Formats

PaL is a free-field format language. A code can start on any line and statements can reside anywhere on a line and is still valid. With this, target users will have a much easier time typing in a code without having to worry certain positioning of the statements. Despite

the fact that statements can be placed anywhere, PaL codes can be organized using line breaks and spacing.

10. Expression

Arithmetic Expression

Operator	Description	Precedence	Associativity
()	Parentheses	1	left to right
* / % //	Multiplication, Division, Modulo, Integer Division	2	left to right
+ -	Addition, Subtraction	3	left to right

Unary Expression

Operator	Description	Precedence	Associativity
++ --	Prefix increment and decrement	1	left to right
++ --	Post fix increment and decrement	2	right to left

Boolean Expression

Operator	Description	Precedence	Associativity
!	Logical NOT	1	left to right
> >= < <=	Relational greater than/greater than or equal to Relational less than/greater than or equal to	2	left to right
== !=	Relational is equal to/is not equal to	3	left to right
&&	Logical AND	4	left to right
||	Logical OR	5	left to right

11. Statements

Declaration Statement

Data_Type = {wholenum, decimalnum, text, bool}

Syntax	**Example**
<Data_Type><identifier>	wholenum a
<Data_Type><identifier> = <value>	bool val = true
<Data_Type><identifier>, ..., <identifier>	decimalnum x, y, z
<Data_Type><identifier> = <value>, …, <identifier> = <value>	text firstName = “rhabi”, lastName = “mendoza”

Input Statement

Syntax	**Example**
<identifier> = ask()	name = ask()

Output Statement

Syntax	**Example**
show(<identifier>)	show(name)
show(“<value>” + <identifier>)	show(“hello” , name)
show(<identifier> + “<value>”)	show(name , “, how are you?”)
show(“<value>”)	show(“hello, pup!”)

Assignment Statement

Assign_Operator = {=, +=, -=, *=, /=, %=, //=}

Syntax	**Example**
<identifier> <Assign_Op> <value>	x = 5
<identifier> <Assign_Op> <identifier>	y += x
<identifier> <Assign_Op> <expression>	z = x + y

Republic of the Philippines
POLYTECHNIC UNIVERSITY OF THE PHILIPPINES

Conditional Statements

Name	Syntax	Rule	Example	Output

Is	is(<condition>){
	<statements>
	}
		The statement will be executed only if the condition is true.	is(5 == 5){
			show("yes")
			}
				yes

Is Else	is(<condition>){
	<statements>
	}
	else{
	<statements>
	}
		When the is condition is true, it will execute the statements inside its block and when the condition is false, it will execute the statements inside the else block.	is(5 <= 4){
			show("yes")
			}
			else{
			show("no")
			}
				no

Is
IsElse
Else
	is(<condition>){
	<statements>
	}
	iselse(<condition>){
	<statements>
	}
	else{
	<statements>
	}
		Iselse section is another condition that can be executed if its condition is true and the is condition is false.	is(10 <= 4){
			show("first condition")
			}
			is(10 >= 4){
			show("second condition")
			}
			else{
			show("none")
			}
				second condition

Repetition Statements

Name	Syntax	Rule	Example	Output

For	for(<declaration>, <condition>, <update>){
	<statements>
	}
		First part is declaration of identifier, second part is the condition for the iteration to start, and third part is the update of the identifier used for iteration to prevent infinite iteration.	for(wholenum a = 1, a <= 5, a++){
			show(a)
			}
				12345

While	while(<condition>){
	<statements>
	}
		Repetition statement where the block inside will make the condition false at one point.	wholenum a = 5
			while(a > 0){
			show(a)
			a--
			}
				54321

Counting Statements

Name	Syntax	Rule	Example	Output

Count Up	countup(<wholenum> to <wholenum>){
	<statements>
	}
		Value on the left should be less than the value on the right.
		Parameters can be a whole number variable or a literal.
			countup(1 to 5){
			show("hello")
			}
				hello
				hello
				hello
				hello
				hello

Count Down	countdown(<wholenum> to <wholenum>){
	<statements>
	}
		Value on the left should be greater than the value on the right.
		Parameters can be a whole number variable or a literal.
			countdown(5 to 1){
			show("hi")
			}
				hi
				hi
				hi
				hi
				hi

Direct Calculation Statements

Name	Syntax	Rule	Example	Output
Addition Direct Calculation	add(<value>, <value>, …, <value)	Values enclosed in parentheses should be separated by comma.	x = 2 add(2, 3, 4, 5, x)	16
Subtraction Direct Calculation	sub(<value>, <value>, …, <value)	Values enclosed in parentheses should be separated by comma.	x = 2 sub(2, 3, 4, 5, x)	-12
Multiplication Direct Calculation	multi(<value>, <value>, …, <value)	Values enclosed in parentheses should be separated by comma.	x = 2 multi(2, 3, 4, 5, x)	240

Shorten Conditional Statement

Name	Syntax	Rule	Example	Output
Direct Conditional	cond(<condition>, <output_string>, <output_string>)	First option will be printed if the condition is true, otherwise it will	cond(10 > 5, "yes", "no")	yes

		go to the second option.		

Drawing Shapes

Name	Syntax	Rule	Example	Output
Draw square	showsquare(<value>)	Value should be a whole number.	showsquare(2)	*** ***
Draw rectangle	showrectangle(<value>, <value>)	Value should be a whole number.	showrectangle(2,5)	***** *****
Draw triangle	showtriangle(<value>)	Value should be a whole number.	showtriangle(5)	* ** *** **** *****

Perimeter Calculator

Name	Syntax	Rule	Example	Output
Square perimeter	perisquare(<value>)	Value should be numerical.	perisquare(5)	20
Rectangle perimeter	perirectangle(<value>, <value>)	Value should be numerical.	perirectangle(3, 4)	14
Triangle perimeter	peritriangle(<value>, <value>, <value>)	Value should be numerical.	peritriangle(5, 6, 7)	18

Area Calculator

Name	Syntax	Rule	Example	Output
Square area	arsquare(<value>)	Value should be numerical.	arsquare(5)	25
Rectangle area	arrectangle(<value>, <value>)	Value should be numerical.	arrectangle(3, 4)	12
Triangle area	artriangle(<value>, <value>)	Value should be numerical.	artriangle(5, 6)	15