PROJECT PROPOSAL

*Yaotecatl*

**Vision and Purpose**

Currently there’s a huge demand for programmers, but it’s still very hard for young boys and girls to get involved with it. Through this project, we want to create an easy way for people of all ages to learn and experience the fun of programming, with the hope that when they’re faced with the decision of what major to choose, they pick one related to computer science.

**Objective**

Our principal objective is to develop a programming language that can help people who want to start learning to program. Our language needs to be easy to understand, needs to cover the basic concepts of programming so that the user can understand the simple concepts of programming. To help the people understand better how programming works we are going to use a tool known as “Blockly” and utilize “PLY” which is an implementation of lex and yacc that uses Python. This tool will help us do the syntactic and lexical analysis of our compiler.

**Language requirements**

**Basic Elements (TOKENS)**

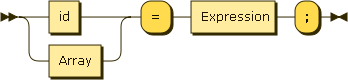
|  |  |
| --- | --- |
| Token | Description |
| Int | Stores integer elements.  Example: 14 |
| Array[] | Stores a set of data with the same data type. We can have arrays of Integers, Booleans, etc.  Example: ArrayInt = [4,5,6,7] |
| String | Stores a set of char type values.  Example: “tec de monterrey” |
| Char | Stores elements of type char.  Example: ‘b’ |
| Float | Stores elements of the type float.  Example: 10.45 |
| Bool | Stores elements of type boolean.  Example: bool A = false |
| * - \* / | They are the arithmetic operators to help with the calculations.  Example: 12 \* 3 + 1 |
| <= >= > < | Comparison operators to help with the order  Example: 10 > 1 |
| = | Character for all the assignments  Example: X = 9 |
| ; | Character for all the terminations  Example: X = 9; |
| && == != || | Logical operators  Examples:  4 && 5 = 4 and 5  3 == 3 = 3 is equal to 3  a != b = a is different than b  a || b = a or b |
| “ “ | Characters that help define a string  Example: “This is a string” |
| { } ( ) [ ] | This set of operators helps us divide pieces of code.  Example: if ( a != b ) |
| ‘ ’ | Characters that help define a char.  Example: ‘a’ |

**Syntax Diagrams**

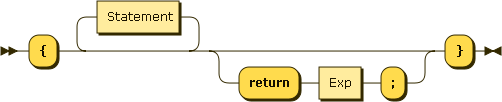
Array



Assignment



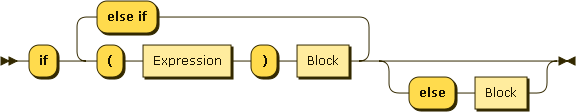
Block\_Return



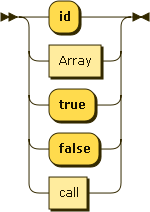
Block



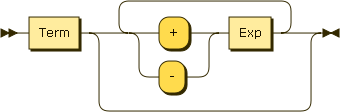
Condition



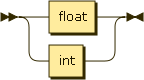
Constant



Exp



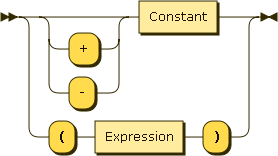
cteN



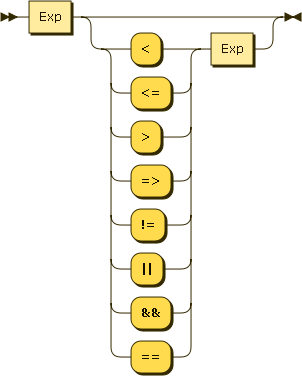
cteS

../../../Downloads/diagram/diagram/cteS.png

Factor



Expression



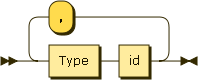
Loop



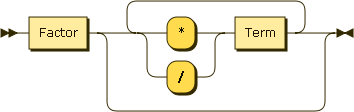
Write



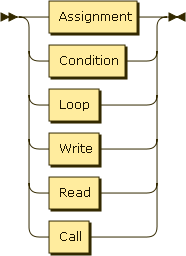
Parameter



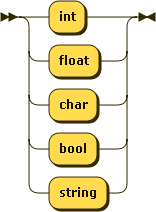
Term



Statement



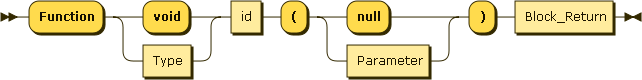
Type



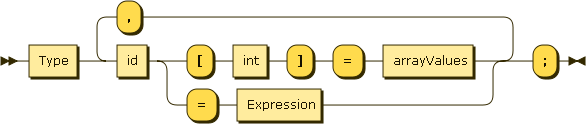
Main



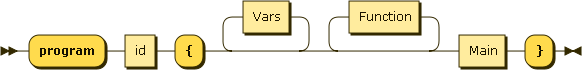
Function



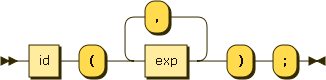
Vars



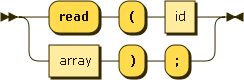
Program



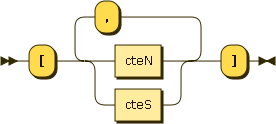
Call



Read



Arrayvalues



|  |  |
| --- | --- |
| Name | Syntax Grammar |
| Array | Array ::= ( id '[' Exp ']' ) |
| Assignment | Assignment ::= ( id | Array ) '=' Expression ';' |
| Block\_Return | Block\_Return ::= ( '{' (Statement)? ( 'return' Exp ';')? '}' ) |
| Block | Block ::= ('{' (Statement)? '}') |
| Condition | Condition ::= ( 'if' ( ('(' Expression ')' Block) ('else if' '(' Expression ')' Block)\* ) ('else' Block)? ) |
| Constant | Constant ::= ( 'id' | Array | 'true' | 'false' | call ) |
| Exp | Exp ::= ( Term (('+' | '-') Exp )\* ) |
| Expression | Expression ::= ( Exp (('<' | '<=' | '>' | '=>' | '!=' | '||' | '&&' | '==') Exp )? ) |
| Factor | Factor ::= ( (('+' | '-')? Constant) | ( '(' Expression ')' ) ) |
| Function | Function ::= 'Function' ( 'void' | Type ) id '(' ( 'null' | Parameter ) ')' Block\_Return |
| Loop | Loop ::= ( 'while' '(' Expression ')' Block ) |
| Main | Main ::= ( 'main' Block) |
| Parameter | Parameter ::= ( (Type id) (',' Type id)\* ) |
| Statement | Statement::= ( Assignment | Condition | Loop | Write | Read| Call) |
| Term | Term ::= ( Factor (('\*' | '/') Term )\*) |
| Type | Type ::= ( "int" | "float" | "char" | "bool" | "string") |
| Vars | Vars ::= Type id ( '[' int ']' '=' arrayValues | '=' Expression ) ( ',' id ( '[' int ']' '=' arrayValues | '=' Expression ) )\* ';' |
| Write | Write ::= ( 'print' '(' Exp ')' ';') |
| Program | Program ::= 'program' id '{' (Vars)\* (Function)\* Main '}' |
| Call | Call ::= id '(' exp ( ',' exp )\* ')' ';' |
| Read | Read ::= 'read' '(' (id | array) ')' ';' |
| Arrayvalues | Arrayvalues ::= '[' ( cteN | cteS ) ( ',' ( cteN | cteS ) )\* ']' |

**Principal Characteristics**

* Different functions can be created; they can be of type void or of any of the types we defined on our grammar. The functions can contain any amount of instruction blocks.
* Variables can be defined globally and locally. If two variables are named the same, the local variable will be use first.
* A function can have any number of parameter the user wants.
* You cannot use reserved words for a variable name.

**Special functions and use instructions in our language**

* Priomh(): This is what we are going to call our main function

**Data Types**

|  |
| --- |
| Int |
| Bool |
| Float |
| Array |
| Char |

**Language and Computer Specifications**

We are going to use Python 2.7.12 and we both are going to use a Macbook to develop our programming language.

**Bibliography**

"Blockly | Google Developers." *Google Developers*. Web. 11 Feb. 2017. <<https://developers.google.com/blockly/>>.

Beazley, David M. "PLY (Python Lex-Yacc)." Web. 11 Feb. 2017. <<http://www.dabeaz.com/ply/>>.

Gunther Rademache. "Railroad Diagram Generator." *Railroad Diagram Generator*. Web. 11 Feb. 2017. <http://www.bottlecaps.de/rr/ui>.

**Signatures**



Gonzalo Gutierrez Javier Guajardo