# INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

## Documentación

# OBSIDIAN


César Armando Galván Valles
A00814038

Luis Mario Díaz Rincón
A00343755

# Tabla de Contenidos

# Descripción del Proyecto

## Propósito y Visión del Lenguaje

El lenguaje de programación *Obsidian,* que se implementará como proyecto de esta clase, tiene como propósito el ser utilizado por jóvenes para que puedan aprender sobre la lógica de la programación al arrastrar bloques de código de una manera gráfica sobre un plano.

## Objetivo del Lenguaje

Nuestro objetivo es implementar una solución efectiva y simple que ayude a que los jóvenes aprendan de una manera diferente, permitiendo un nuevo método efectivo de enseñanza para que los jóvenes aprendan el uso de ciclos, condiciones, estatutos de asignación, expresiones matemáticas, tipos de datos y funciones.

## Alcance del Proyecto

El alcance del lenguaje *Obsidian* va de la mano con nuestro objetivo de hacer un lenguaje simple en el cual sea fácil aprender las bases de programación, en este permitimos al usuario declarar variables, hacer operaciones matemáticas con ellas, definir ciclos ¨while¨, utilizar operadores lógicos en condicionales, definir funciones normales y recursivas. También ofrecerle al usuario la capacidad de utilizar funciones de I/O como Read y Write.

## Análisis de Requerimientos

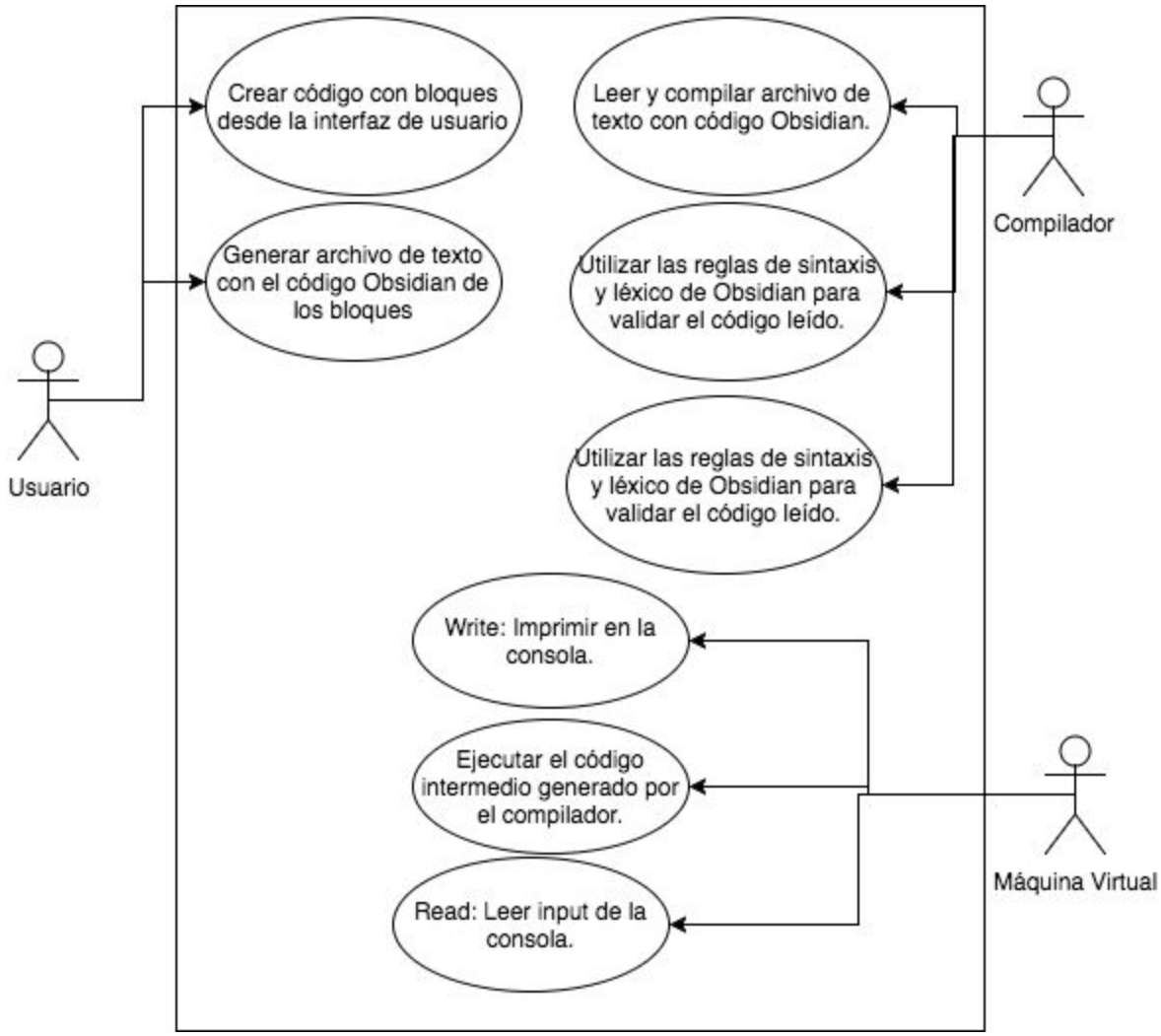
### Requerimientos funcionales

- El Compilador debe de leer y compilar código *Obsidian* válido.
- El Compilador debe generar código intermedio para que sea ejecutado por la máquina virtual.
- En caso de error, el compilador debe parar su ejecución e indicar al usuario el tipo de error y en que linea se generó el mismo.
- La Máquina Virtual debe ejecutar el código intermedio generado por el compilador.
- La Interfaz de Usuario debe permitir generar código *Obsidian* válido a partir de bloques, siempre y cuando el usuario sigue las reglas del lenguaje.

### Requerimientos no funcionales

- La Interfaz de Usuario debe ser de fácil uso para personas con poca experiencia programando.
- La ejecución del código debe ser rápida.

## Casos de Uso Generales




## Descripción de Casos de Prueba

- Prueba de generación de cuádruplos de sentencias if/else
- Pruebas de léxico
- Pruebas de sintaxis
- Prueba de generación de cuádruplos de sentencias while
- Prueba de generación de cuádruplos de declaración de variables
- Prueba de generación de cuádruplos de declaración de funciones
- Prueba de generación de cuádruplos de llamadas de funciones
- Prueba de generación de cuádruplos de declaración de arreglos
- Prueba de generación de cuádruplos de acceso a arreglos

- Prueba de ejecución de sentencias if, else, while en la máquina virtual
- Prueba de ejecución de arreglos en la máquina virtual
- Fibonacci Recursivo
- Fibonacci Ciclico
- Factorial Recursivo
- Factorial Ciclico
- Bubble Sort
- Multiplicación de Matrices
- Prueba de Print()
- Prueba de Read()

### Logs del Desarrollo del Proyecto

Lorem ipsum

Aprendí muchísimo durante el desarrollo de Obsidian. Ha sido creo, uno de los proyectos más interesantes durante la carrera y me ayudó a comprender lo que pasa detrás cuando corro código para el trabajo o para cualquier otra clase. El proceso de desarrollo fue interesante, decidimos utilizar pair programming como nuestro método de desarrollo pues el compilador es complicado y creíamos que encontraríamos mejores soluciones si estas eran diseñadas en conjunto, mismo caso con los problemas que se nos iban presentando. Ya en la última etapa, cuando empezó a entrar la presión del tiempo empezamos a dividir cargas para asegurar la entrega a tiempo del proyecto.

_____

Luis Mario Díaz
A00343755

dfsa

_____

Cesar Armando Galvan

## Descripción del Lenguaje

### Nombre

Obsidian

## Descripción de Características Principales del Lenguaje

Los lenguajes más utilizados en el mundo tienen una sintaxis basada en C que puede llegar a ser algo compleja para principiantes. En Obsidian decidimos basarnos en el lenguaje C, sin embargo intentamos hacerlo mucho más simple ya que estamos enfocados a personas principiantes programando. Básicamente el objetivo de Obsidian es introducir a novatos programando a la sintaxis de programación C, para que una vez que estén familiarizados con este tipo de sintaxis, ya puedan dar el paso a algo más complejo como lo es C o Java.

Obsidian tiene las funciones más importantes para que un principiante pueda empezar a programar sin verse agobiado por la cantidad de opciones que ofrecen lenguajes más avanzados. Declaraciones de variables, ciclos while, creación y llamadas a funciones, llamadas a funciones, declaración y uso de arreglos, y funciones fundamentales de I/O como write y print.

## Descripción de Errores

**Errores de compilación:**

- Error de Sintaxis
- Error: Uso de variable o función no declarada
- Error: Llamar a una función con el número incorrecto de parámetros
- Error: Inicializar un arreglo de tamaño incompatible
- Error: Utilizar operador con operando no compatible

**Errores en la máquina virtual:**

-

# Descripción del Compilador

## Lenguaje, Equipo y Utilidades utilizados en Desarrollo

Durante el desarrollo del lenguaje utilizamos dos Macbook Pro de 13 pulgadas, una con la versión 10.12 del sistema operativo y otra con la versión 10.11.

El lenguaje que se utilizó fue Python 2.7 y para la interfaz de usuario utilizamos HTML, Javascript y CSS, esto en conjunto con la librería Blockly.

## Descripción del Análisis Léxico

### Expresiones Regulares

```
t_ignore = ' \t'                t_PLUS = r'\+'
```

```
t_MINUS = r'-'                  t_MULTIPLICATION = r'\*'
t_DIVISION = r'/'               t_MOD = r'%'
t_EQUALS = r'='                 t_EQUALEQUALS = r'=='
t_DIFFERENT = r'!='             t_GREATER = r'>'
t_LESS = r'<'                   t_GREATEROREQUAL = r'>='
t_LESSOREQUAL = r'<='           t_AND = r'&&'
t_OR = r'\|\|'                  t_LPAR = r'\('
t_RPAR = r'\)'                  t_LBRACKET = r'{'
t_RBRACKET = r'}'               t_LSQRTBRACKET = r'\['
t_RSQRTBRACKET = r'\]'          t_COMMA = r','
t_SEMICOLON = r';'              t_CTEDOUBLE = r'-?[0-9]+\.[0-9]+'
t_CTEINT = r'-?[0-9]+'          t_CTEBOOL = r'false|true'
t_ID = r'[a-z][a-zA-Z0-9]*'     t_newline = r'\n+'
```

## Enumeración de Tokens del Lenguaje y Código Asociado

### Palabras Reservadas:

```
'int' : 'INT'
'double' : 'DOUBLE',
'bool' : 'BOOL',
'func' : 'FUNC',
'void' : 'VOID',
'if' : 'IF',
'else' : 'ELSE',
'main' : 'MAIN',
'while' : 'WHILE',
'read' : 'READ',
'write' : 'WRITE',
'return' : 'RETURN'
```

### Tokens

```
‘PLUS’  :  ` + `                ‘MINUS’  :  ` - `
‘DIVISION’  :  ` / `            ‘MULTIPLICATION’  :  ` * `
‘MOD’  :  ` % `                 ‘EQUALS’  :  ` = `
‘EQUALSEQUALS’  :  ` == `       ‘DIFFERENT’  :  ` != `
‘GREATER’  : ` > `              ‘LESS’  :  ` < `
‘GREATEROREQUAL’  :  ` >= `     ‘LESSOREQUAL’  :  ` <= `
‘AND’  :  ` && `                ‘OR’  :  ` || `
‘LPAR’  :  ` ( `                ‘RPAR’  : ` ) `
‘LBRACKET’  :  `  {  `          ‘RBRACKET’  :  ` } `
```

‘LSQRTBRACKET’ : ` ] ` ‘RSQRTBRACKET’ : ` [ `
‘COMMA’ : ` , ` ‘SEMICOLON’ : ` ; `

## Descripción del Análisis de Sintaxis

### Gramática usada para representar estructuras sintácticas

```
PROGRAM  ::= VARS* FUNC* MAIN

VARS ::= VAR_TYPE 'id' ('[' 'cte' ']')* (('=' VAR_CTE) | ) (',' 'id' ('[' 'cte' ']')* (('='
VAR_CTE) | ))* ';'

VAR_TYPE ::= ('bool' | ' int' | 'double')

VAR_CTE ::= ('cte_int' | 'cte_double' | ' cte_bool' | 'id' ('[' 'cte' ']')* | FUNC_CALL)

FUNC ::= 'func' FUNC_TYPE id '(' ( VAR_TYPE 'id' (',' VAR_TYPE 'id')*)? ')' FUNC_BLOCK

FUNC_TYPE ::= ('void' | 'bool' | ' int' | 'double')

FUNC_BLOCK ::= '{' VARS* STATEMENT* '}'

STATEMENT ::= ( READ | WRITE | CYCLE | CONDITION | ASSIGNATION | FUNC_CALL | RETURN)

READ ::= 'read' '(' 'id' ('[' EXP ']')* ')' ';'

WRITE ::= 'write' '(' EXP ')' ';'

CYCLE ::= 'while' '(' EXPRESSION ')' BLOCK

CONDITION ::= 'if' '(' EXPRESSION ')' BLOCK ("else" BLOCK)?

ASSIGNATION ::= 'id' ('[' EXP ']')* '=' EXPRESSION ';'

FUNC_CALL ::= 'id' '(' (EXP (',' EXP)*)? ')' ';'

RETURN ::= 'return' EXP ';'

BLOCK ::= '{' STATEMENT* '}'

EXPRESSION ::= CONC ( (AO CONC)+)?

CONC ::= EXP (COMP EXP)?

EXP ::= TERM (PL TERM)*
```

```
TERM ::= FACTOR (DM FACTOR)*

FACTOR ::= ( '(' EXPRESSION ')' ) | VAR_CTE

AO ::= '&&' | '||'

COMP ::= ('<' | '>' | '<=' | '>=' | '==' | '!=')

PL ::= '+' | '-'

DM ::= '*' | '/'

MAIN ::= 'main' MAIN_BLOCK

MAIN_BLOCK ::= '{' VARS* STATEMENT* '}'
```

## Descripción de Generación de Código intermedio y Análisis Semántico

### Código de operación y direcciones virtuales asociadas a elementos del código

Lorem ipsum

### Diagramas de Sintaxis con las acciones correspondientes

Lorem ipsum

### Breve descripción de las acciones semánticas y de código

Lorem ipsum

### Tabla de consideraciones semánticas

Lorem ipsum

## Descripción del proceso de Administración de Memoria

### Especificación gráfica de las estructuras de datos utilizadas

# Descripción de la Máquina Virtual

## Lenguaje, Equipo y Utilidades utilizados en el Desarrollo

Lorem ipsum

## Descripción del proceso de Administración de Memoria en Ejecución

### Gráfica de Estructuras de Datos Utilizados

Lorem ipsum

### Asociación entre Direcciones Virtuales y Reales

Lorem ipsum

# Pruebas del Funcionamiento del Lenguaje

## Pruebas de Funcionamiento