

Diseño de Compiladores

**MATRUSHKA**

Jose Luis Loya Cabrera A01111663

1. de Mayo, 2016

**Índice**

Descripción del Proyecto …………………………………………………………………….2

Visión y propósito del proyecto …..…..…..…..…..…..…..…..…..…..…..………..2

Objetivos del Proyecto …..…..…..…..…..…..…..…..…..…..…..…..…..…..……..2

Alcance del Proyecto…..…..…..…..…..…..…..…..…..…..…..…..…..…..…….....2

Análisis de Requerimientos…..…..…..…..…..…..…..…..…..…..…..…..…….......2

Casos de Uso…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..……3

Descripción de las pruebas…..…..…..…..…..…..…..…..…..…..…..…..…..….....4

Descripción del proceso…..…..…..…..…..…..…..…..…..…..…..…..…..…..…….4

Descripción del Lenguaje ……..…..…..…..…..…..…..…..…..…..…..…..…..…..…..……5

Nombre…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..…...5

Descripción…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..5

Errores que pueden ocurrir…..…..…..…..…..…..…..…..…..…..…..…..…..……...5

Descripción del compilador…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..……..6

Análisis Léxico…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..……6

Descripción de Análisis de Sintaxis…..…..…..…..…..…..…..…..…..…..…..….....7

Descripción de generación de código…..…..…..…..…..…..…..…..…..…..…..….9

Diagrama de sintaxis con las acciones…..…..…..…..…..…..…..…..…..…..……10

Descripción de las acciones…..…..…..…..…..…..…..…..…..…..…..…..…..……17

Tabla Semántica…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..…….18

Descripción de la Administración de Memoria…..…..…..…..…..…..…..…..…….19

Descripción de la Máquina Virtual…..…..…..…..…..…..…..…..…..…..…..…..…..……..20

Descripción de la Memoria y del Runtime…..…..…..…..…..…..…..…..…..……..20

Pruebas de Funcionalidad…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..….21

Prueba1…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..….21

Prueba2…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..….22

Lista de Documentación…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..……23

COMPILADOR…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..23

MAQUINA VIRTUAL…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..…….28

Manual de Usuario…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..…..………20

**Descripción del Proyecto**

**Visión y propósito del proyecto**

La visión de matrushka es poder encriptar y desencriptar programas enteros. Actualmente personas que quieren comunicarse de manera segura necesitan saber del tema y saber utilizar las herramientas. Con matrushka, la barra de dificultad disminuye. El usuario que recibe o ejecuta el programa solo tiene que introducir la contraseña para poder ejecutarlo, no ocupa conocimientos especiales.

**Objetivos del Proyecto**

Mi objetivo es llevar la encriptación a más lugares. Me fue imperativo dado la época que vivimos de vigilancia masiva. Otro objetivo es que fuera sencillo de utilizar. No quiero hacer la encriptación más complicada, si no más accesible.

**Alcance del Proyecto**

El proyecto fue diseñado para que la configuración del cifrado fuese sencilla. Es lo primero que vez cuando vez el código fuente. El lenguaje funciona con o sin cifrado. Tiene las funcionalidades que esperarías de un lenguaje educativo.

**Análisis de Requerimientos**

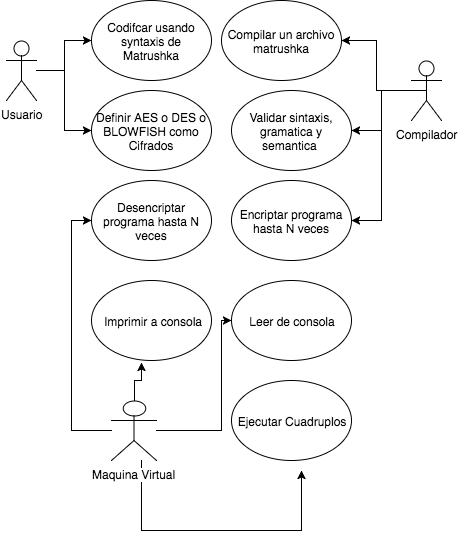
Requerimientos Funcionales:

* El lenguaje debe soportar los algoritmos de cifrado DES, AES y BLOWFISH
* El compilador deberá poder compilar código escrito con gramática valida.
* El compilador deberá detenerse al detectar errores.
* El compilador deberá cifrar el programa de 0 a N veces. (N siendo la cantidad de cifrados especificados).
* El lenguaje debe soportar los tipos entero, flotante, booleanas y string.
* El lenguaje debe soportar variables, vectores y matrices
* El lenguaje debe soportar condicionales if/else y ciclos while
* El lenguaje debe soportar funciones y parámetros
* El lenguaje debe soportar variables locales y globales
* El lenguaje debe soportar scope de variables
* El lenguaje debe soportar operaciones de concatenación en strings
* El lenguaje debe soportar lectura de stdin
* El lenguaje debe soportar escritura a stdout

Requerimientos No Funcionales:

* La gramática deberá ser sencilla
* La máquina virtual y el compilador deberán utilizar OpenSSL para cifrados
* El compilador deberá estar codificado en ANSI C
* La máquina virtual deberá estar codificada en Python3

**Casos de uso**



**Descripción de las pruebas**

Los archivos dentro de “/tests/” consideran las siguientes cosas:

* Prueba de ciclos y condiciones
* Prueba de llamadas a funciones
* Prueba de expresiones aritméticas
* Prueba de expresiones con strings
* Prueba de variables, vectores y matrices
* Prueba de compilación y ejecución con 0 cifrados
* Prueba de compilación y ejecución con 1 cifrado
* Prueba de compilación y ejecución con 3 cifrados

**Descripción del Proceso**

|  |  |
| --- | --- |
| **Fecha** | **Comentario** |
| Thu May 4 | Codigo ya limpio y con comentarios |
| Thu May 4 | Cleaned about 80% of the code |
| Thu May 4 | Cleaned about 40% of the code |
| Thu May 4 | Cleaned about 20% of the code |
| Tue May 3 | Removed old initialization script |
| Tue May 3 | Moved source files. Created Makefile |
| Tue May 3 | Encryption works |
| Tue May 2 | Ya funciones vectores y matrices |
| Tue May 2 | Máquina virtual lista. Arreglos masomenos |
| Mon May 1 | Added era\_size reference at bottom |
| Mon May 1 | New standard for opcodes. Deleted deadcode. |
| Sun Apr 30 | Code generation for CYCLES READY |
| Sun Apr 30 | Code generation for condicionales |
| Sun Apr 30 | Generacion de codigo secuencial listo |
| Sat Apr 29 | IR Code for simple if's |
| Sat Apr 29 | Added memory addressing to symbol table |
| Sat Apr 29 | Do semantics analysis and code generation in just one attribute for expressions |
| Sat Apr 29 | Cleaned up even more. Better naming and structure |
| Sat Apr 29 | Cleaning up work: RENAMING, DELETING DAEDCODE, ADDING COMMENTS |
| Fri Apr 28 | Generacion de codigo sencillo para expresiones |
| Fri Apr 28 | Semantica de Variables y Expresiones Terminada |
| Fri Apr 28 | Switched 1era y 2da entrega folder. They were in the wrong order |
| Thu Apr 27 | Entrega 2 terminada |
| Thu Apr 27 | Trabajando en la segunda entrega |
| Tue Apr 25 | Primera entrega lista |

Fue el proyecto más largo, complicado y gratificante que hice en mi carrera. Combine mis dos lenguajes favoritos para programarlo y también de inspiración. El hecho de que el lenguaje tenía que ser temático, me hizo escoger algo que valiera la pena. La verdad lo empecé muy tarde porque trabajo y estudio y hubo varias veces que pensé en tirar la toalla. Sin embargo, lo termine y esto me da mucho orgullo.

.

Jose Luis Loya Cabrera

**Descripción del Lenguaje**

**Nombre**

El lenguaje se llama Matrushka. Está inspirado en la muñeca rusa que al abrirla tiene otra muñeca que al abrirla trae otra muñeca y así recursivamente hasta llegar al final. De ahí nace la inspiración. Los programas de este lenguaje pueden cifrarse múltiples veces y para ejecutarlo tienes que descifrarlo múltiples veces.

**Descripción**

El lenguaje le permite al usuario definir un cifrado y una clave para descifrarlo. Ese es el feature principal. La sintaxis está inspirada en C y Python. Dándole esa vibra de simplicidad y seriedad al mismo tiempo. El lenguaje le permite al usuario usar variables, vectores, matrices, funciones, whiles, condiciones IFs, lectura y escritura en consola. En cuestión de tipos solo son 4: int, double, bol y string.

**Errores que pueden ocurrir**

El compilador detecta y avisa de los siguientes errores:

* Uso de variables o funciones previamente no declaradas
* Doble declaración de variables con el mismo nombre
* Usos no permitidos de operandos en ciertos operadores
* Regresar un dato distinto al de la función
* Usar un algoritmo de cifrado distinto a “AES”, “DES” o “BLOWFISH”
* Expresiones distintas a booleanas en condicionales y ciclos
* Errores de gramática

La máquina virtual avisa sobre los siguientes errores:

* La clave de cifrado es incorrecta
* Cuando detecta un cuádruplo no registrado

**Descripción del Compilador**

Matrushka Compiler fue desarrollado en una computadora Mac OS 10.12 utilizando ANSI C. Software necesario para compilar a Matrushka es Lex, Bison y GCC. Matrushka espera además la presencia de la librería OpenSSL en el sistema.

**Descripción del Análisis Léxico**

Expresiones Regulares

integer [0-9]+

double [0-9]+\.[0-9]+

string \"(\\.|[^\\"])\*\"

identifier [a-zA-Z]+[0-9]\*

whitespace [ \t]+

newline \n

comments \/\/.\*

Palabras reservadas

|  |  |
| --- | --- |
| "if" = IFTKN  "endi" = ENDIFTKN  "else" = ELSETKN  "func" = FUNCTKN  "endf" = ENDFUNCTKN  "while" = WHILETKN  "endw" = ENDWHILETKN  "cipher" = CIPHERTKN | "read" = READTKN  "write" = WRITETKN  "int" = INTTKN  "double" = DOUBLETKN  "string" = STRINGTKN  "bool" = BOOLTKN  "true" = FALSETKN  "false" = FALSETKN |

TOKENS

|  |  |
| --- | --- |
| "=" = ASSIGNTKN  "+" = PLUSTKN  "-" = MINUSTKN  "\*" = TIMESTKN  "/" = DIVTKN  "<" = LTTKN  "<=" = LTETKN  ">" = GTTKN  "[" = LEFTBTKN  "]" = RIGHTBTKN  "," = COMMATKN  ":" = OPENBLOCKTKN | ">=" = GTETKN  "==" = EQUALTKN  "!=" = NOTEQUALTKN  "<<" = ASSIGNSECRETTKN  "||" = ORTKN  "&&" = ANDTKN  "(" = LEFTPTKN  ")" = RIGHTPTKN  ";" = SEMICOLONTKN  {integer} = INTVALTKN  {double} = DOUBLEVALTKN  {string} = STRINGVALTKN  {identifier} = IDTKN |

**Descripción de Sintaxis**

Program : secrets a\_1 vardeclarations functions a\_2

| secrets a\_1 functions a\_2

secrets : secrets secret | secret

secret : **ε** | CIPHERTKN STRINGVALTKN COMMATKN STRINGVALTKN a\_3 SEMICOLONTKN

functions : functions function | function;

function : FUNCTKN type OPENBLOCKTKN IDTKN a\_4 LEFTPTKN params RIGHTPTKN

OPENBLOCKTKN funcbody a\_5 ENDFUNCTKN;

funcbody : **ε** | vardeclarations blockstmts | blockstmts

params : **ε** | paramlist;

paramlist : paramlist COMMATKN param | param;

param : type IDTKN a\_6

vardeclarations : vardeclarations vardeclaration SEMICOLONTKN

| vardeclaration SEMICOLONTKN;

vardeclaration : type IDTKN a\_7

| type IDTKN LEFTBTKN INTVALTKN RIGHTBTKN a\_8

| type IDTKN LEFTBTKN INTVALTKN RIGHTBTKN LEFTBTKN INTVALTKN

RIGHTBTKN a\_9

type : INTTKN | DOUBLETKN | STRINGTKN | BOOLTKN;

blockstmts : blockstmts blockstmt | blockstmt;

blockstmt : assignstmt SEMICOLONTKN | returnstmt SEMICOLONTKN | ifstmt

| iterstmt | iostmt SEMICOLONTKN | callstmt SEMICOLONTKN;

ifstmt : IFTKN LEFTPTKN expstmt a\_9 RIGHTPTKN OPENBLOCKTKN

blockstmts ENDIFTKN a\_11

| IFTKN LEFTPTKN expstmt a\_9 RIGHTPTKN OPENBLOCKTKN blockstmts

ELSETKN a\_10 blockstmts ENDIFTKN a\_12;

iterstmt : WHILETKN a\_13 LEFTPTKN expstmt a\_14 RIGHTPTKN OPENBLOCKTKN

blockstmts ENDWHILETKN a\_15

assignstmt : IDTKN ASSIGNTKN expstmt a\_16

| IDTKN LEFTBTKN expstmt RIGHTBTKN a\_17 ASSIGNTKN expstmt a\_18

| IDTKN LEFTBTKN expstmt RIGHTBTKN LEFTBTKN expstmt a\_19

RIGHTBTKN ASSIGNTKN expstmt a\_18

iostmt : READTKN var a\_19 | WRITETKN expstmt a\_20

var : identifier | identifier LEFTBTKN expstmt RIGHTBTKN a\_21

| identifier LEFTBTKN expstmt RIGHTBTKN LEFTBTKN expstmt a\_22 RIGHTBTKN;

identifier : IDTKN;

expstmt : expstmt compoperator a\_23 exp a\_24 | exp;

compoperator : LTTKN | LTETKN | GTTKN | GTETKN | EQUALTKN | NOTEQUALTKN |

ORTKN | ANDTKN;

exp : exp additiveoperator a\_23 term a\_24 | term;

additiveoperator: MINUSTKN | PLUSTKN;

term : term TIMESTKN a\_23 factor a\_24

| term DIVTKN a\_23 factor a\_24

| factor

values : INTVALTKN | DOUBLEVALTKN | STRINGVALTKN | TRUETKN | FALSETKN;

factor : LEFTPTKN exp RIGHTPTKN

| var a\_25

| values a\_26

| callstmt a\_27

callstmt : IDTKN a\_28 LEFTPTKN args RIGHTPTKN a\_29

args : **ε** | arglist

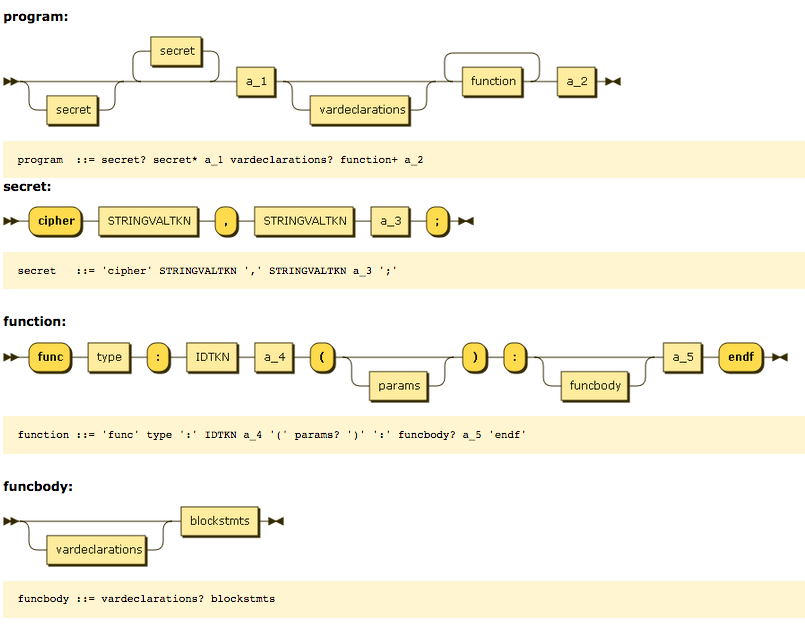
arglist : arglist COMMATKN exp a\_30 | exp a\_31

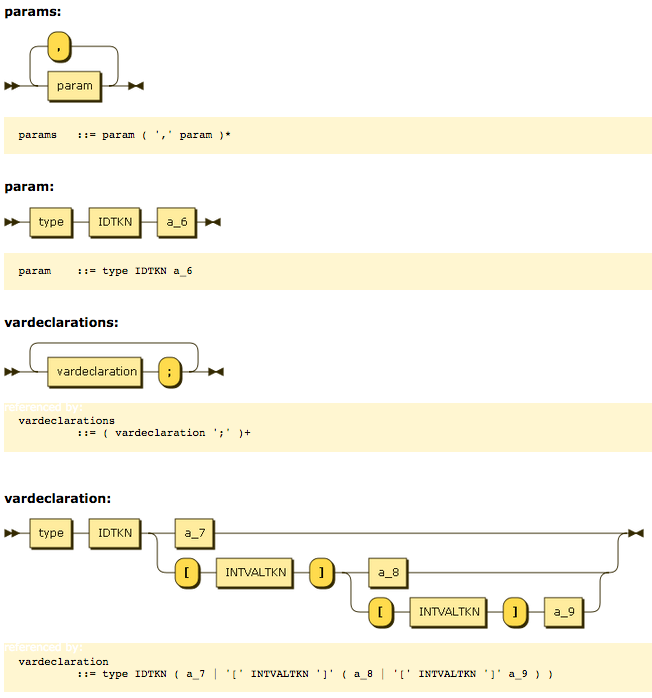
returnstmt : RETURNFUNCTKN expstmt a\_32

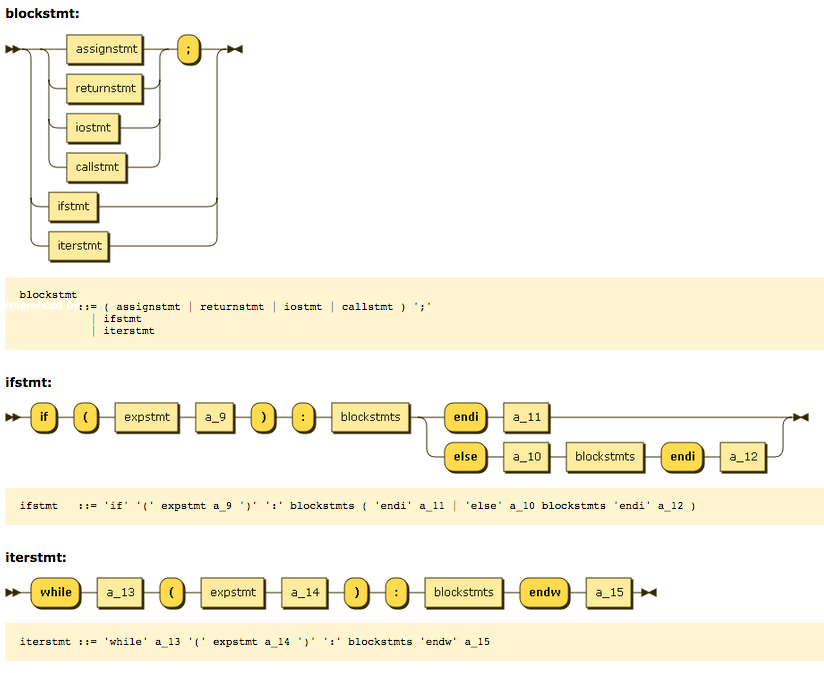
**Descripción de la Generación de Código** *( C=constant, A=address )*

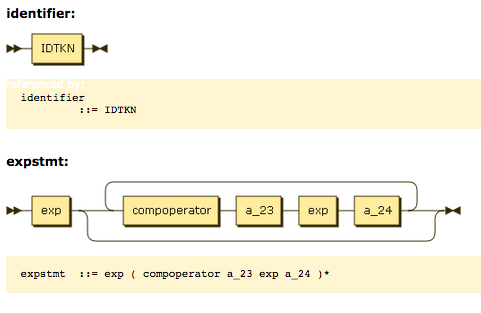
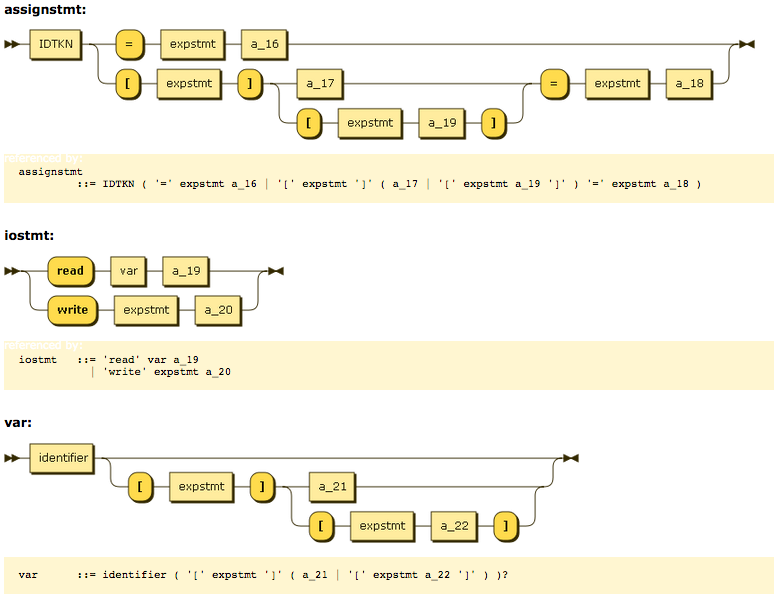
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| OPCODE | OP A | OP B | OP C | COMMENT |
| CALL | c |  |  | GOTO FUNCTION WITH THAT LABEL |
| ERAS | c |  |  | RESERVER SPACE IN STACK |
| PUSH | c/a |  |  | PUSH VALUE INTO STACK |
| AVER | c/a |  |  | OFFSET FOR ARRAY ACCESS |
| MVER | c/a | c/a | c | OFFSET FOR MATRIX ACCESS |
| SADD | c/a | c/a | a | CONCAT TWO STRINGS |
| IADD | c/a | c/a | a | ADD TWO INTEGERS |
| ISUB | c/a | c/a | a | SUBSTRACT TWO INTEGERS |
| IMUL | c/a | c/a | a | MULTIPLY TWO INTEGERS |
| IDIV | c/a | c/a | a | DO INTEGER DIVISION |
| DADD | c/a | c/a | a | ADD TWO DOUBLES |
| DSUB | c/a | c/a | a | SUBSTRACT TWO DOUBLES |
| DMUL | c/a | c/a | a | MULTIPLY TWO DOUBLES |
| DDIV | c/a | c/a | a | DO DOUBLE DIVISON |
| ORVL | c/a | c/a | a | LOGICAL OR |
| ANDV | c/a | c/a | a | LOGICAL AND |
| EQLV | c/a | c/a | a | TEST FOR EQUALITY |
| NEQL | c/a | c/a | a | TEST FOR INEQUALITY |
| EQST | c/a | c/a | a | TEST FOR EQUALITY OF TWO STRINGS |
| NEQS | c/a | c/a | a | TEST FOR INEQUALITY OF TWO STRINGS |
| ENDF |  |  |  | RETURNS 0 |
| RETV | c/a |  |  | RETURNS OP A |
| POPV | c/a |  |  | POP STACK AND SAVE IT ON OP A |
| MOVE | c/a | a |  | MOVE VALUE ON A TO ADDRES B |
| LTVL | c/a | c/a | a | LESS THAN |
| LTEV | c/a | c/a | a | LESS THAN OR EQUAL |
| GTVL | c/a | c/a | a | GREATER THAN |
| GTEV | c/a | c/a | a | GREATER THAN OR EQUAL |
| GTOF | c/a | a |  | JUMP TO OPB ON FALSE OPA |
| GOTO | a |  |  | JUMP TO ADDRESS A |
| IGET | a |  |  | READ INT FROM STDIN |
| DGET | a |  |  | READ DOUBLE FROM STDIN |
| SGET | a |  |  | READ STRING FROM STDIN |
| BGET | a |  |  | READ BOOLEAN FROM STDIN |
| IPUT | c/a |  |  | PUT INT INTO STDOUT |
| DPUT | c/a |  |  | PUT DOUBLE INTO STDOUT |
| BPUT | c/a |  |  | PUT BOOLEAN INTO STDOUT |
| SPUT | c/a |  |  | PUT STRING INTO STDOUT |

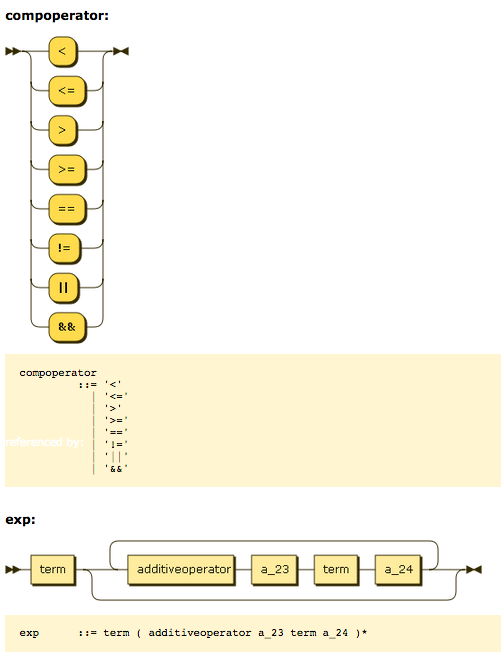
**Diagramas de Sintaxis con las acciones**

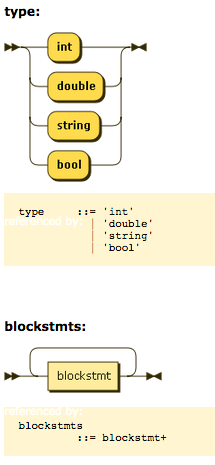
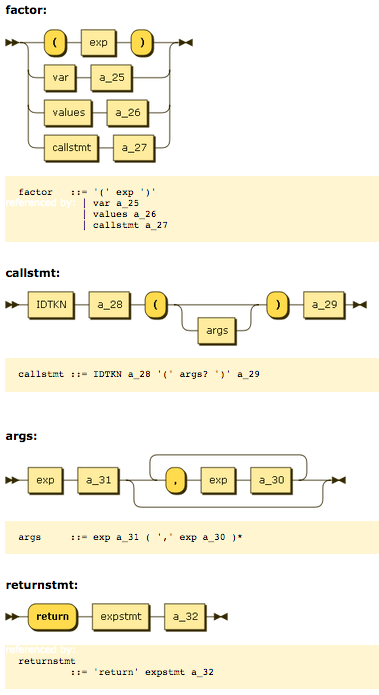
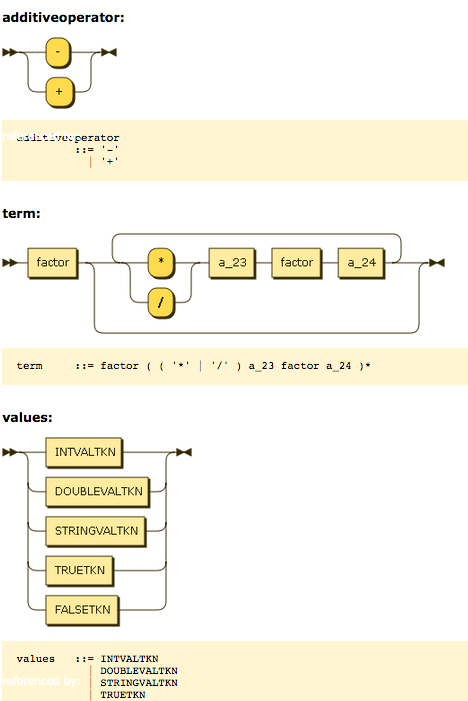
****

****

****

****

****

****

**Descripción de las acciones**

**a1:** Inicializa las estructuras. Genera el CALL a la función MAIN

**a2:** Genera el código encriptado. Guarda el código en el archivo de salida

**a3:** Guarda las opciones de cifrado en cipherStack

**a4:** Guarda el identificador de la función junto con su tipo. Actualiza “scope”

**a5:** Genera código de retorno vacío

**a6:** Guarda el identificador y su tipo en la tabla de Símbolos como “parámetro”

**a7:** Guarda el identificador y su tipo en la tabla de Símbolos como “variable”

**a8:** Guarda el identificador, su tipo y su tamaño en la tabla de Símbolos

**a9:** Guarda el identificador, su tipo y sus dimensiones en la tabla de Símbolos

**a9b:** Genera el código GOTOF para condicional

**a10:** Genera el código GOTO para else de condicional

**a11:** Cierra el condicional y actualiza el salto previo

**a12:** Cierra el condicional y actualiza el salto previo

**a13:** Guarda la dirección de memoria antes de empezar el ciclo

**a14**: Genera el código de inicio del ciclo

**a15:** Genera un GOTO hacia a13 y actualiza a14 con la dirección actual

**a16:** Genera código y checa semántica de asignación de variable

**a17:** Genera código para informar al VM offset del vector

**a18:** Genera código para informar al VM offset de la matriz

**a19:** Genera código y checa semántica de asignación de vector/matriz

**a19b:** Genera código para lectura de consola

**a20:** Genera código para escritura a consola

**a21:** Lo mismo que a17

**a22**: Lo mismo que a18

**a23:** Empuja operador a stack de operadores

**a24:** Genera código y checa semántica para evaluación de expresiones

**a25:** Empujar operando tipo constante a stack de operandos

**a26:** Empujar operando tipo variable a stack de operandos

**a27:** Empujar operando tipo función a stack de operandos

**a28:** Genera código para informar al VM el número de variables de la función

**a29:** Genera código y checa semántica para llamar a una función

**a30:** Genera código para empujar argumentos al stack del VM

**a31:** Lo mismo que a30

**a32:** Genera código y checa semántica para retornar de una función

**Tabla Semántica**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **op1** | **op2** | **+** | **-** | **\*** | **/** | **<** | **<=** | **>** | **>=** | **==** | **!=** | **&&** | **||** |
| int | int | int | int | int | int | bool | bool | bool | bool | bool | bool | err | err |
| int | double | double | double | double | double | bool | bool | bool | bool | bool | bool | err | err |
| int | bool | err | err | err | err | err | err | err | err | err | err | err | err |
| double | int | double | double | double | double | bool | bool | bool | bool | bool | bool | err | err |
| double | double | double | double | double | double | bool | bool | bool | bool | bool | bool | err | err |
| double | bool | err | err | err | err | err | err | err | err | err | err | err | err |
| bool | int | err | err | err | err | err | err | err | err | err | err | err | err |
| bool | double | err | err | err | err | err | err | err | err | err | err | err | err |
| bool | bool | err | err | err | err | err | err | err | err | bool | bool | bool | bool |
| string | string | string | err | errror | err | err | err | err | err | bool | bool | err | err |
| string | int | err | err | err | err | err | err | err | err | err | err | err | err |
| string | double | err | err | err | err | err | err | err | err | err | err | err | err |
| string | bool | err | err | err | err | err | err | err | err | err | err | err | err |
| int | string | err | err | err | err | err | err | err | err | err | err | err | err |
| double | string | err | err | err | err | err | err | err | err | err | err | err | err |
| bool | string | err | err | err | err | err | err | err | err | err | err | err | err |

**Descripción de la Administración de Memoria**

Para administrar la memoria solo existieron dos tipos de estructuras básicas. Una estructura de Hash que contenía todos los atributos que puede llegar a tener un identificador. Y estructuras tipo PILA para guardar operandos, operadores, tipos de los operadores, saltos y opciones de cifrado.

En un principio se generaron direcciones fijas, pero después de analizar el funcionamiento de GASM y ver como hacían referencias con registros se decidió utilizar únicamente direcciones relativas. La máquina virtual se encarga de reservar espacio antes de entrar a cada función y las variables están en posiciones relativas a la memoria de la función. Por ejemplo.

//CODIGO // DIRECCION ABSOLUTA // DIRECION RELATIVA

---------------- ------------------------------------- ---------------------------------

int globali; // 0x1000 // (%GLOBALS+1)

func(): // 0x1004

int x; // 0x1008 // (%STACK+1)

int y; // 0x1012 // (%STACK+2)

double a; // 0x2000 // (%STACK+3)

endf

Como la Máquina Virtual esta programada en Python, se decidió usar offsets contiguos. Python te permite guardar objetos de distintos tipos en una misma lista. Mientras el ordenamiento y el posicionamiento del stack se mantuviera correcto se podían tratar a todos los tipos como si fueran del mismo tamaño. En la tabla de abajo se ve de donde proviene la información puesta arriba. Usando la columna num podemos generar las direcciones relativas.

SYMBOL TABLE

---------- ---------- ------ ----- ------- ----

identifier context type kind address num

---------- ---------- ------ ----- ------- ----

y main int var 1012 2

globali global int var 1000 1

main global int func 1004 2

a main double var 2000 3

x main int var 1008 1

**Descripción de la Máquina Virtual**

Matrushka Virtual Machine fue desarrollado en Python3 sin ninguna librería exterior. Se utilizó también MacOS 10.12 como plataforma de desarrollo. La decisión para usar Python en la Máquina Virtual era por su simplicidad y flexibilidad a la hora de transformar y manejar tipos de datos. La máquina virtual, al igual que el compilador, espera que esté instalado OpenSSL en el sistema para poder ejecutarse.

**Descripción de la Memoria y del Runtime**

Lo primero que hace la máquina virtual es checar si el código IR está cifrado o no. Si este encriptado procede a desencriptarlo hasta que ya no quede código cifrado alguno.

Una vez que ya no hay código cifrado hace un primer pase entero del código. Con esto inicializa los diccionarios *function\_address* y *function\_era*. Dado que el código intermedio usa labels para las funciones en vez de sus direcciones absolutas, la máquina virtual debe ubicar donde están posicionadas las funciones antes de saltar hacia ellas. A su vez el direccionario function\_era guarda información acerca de cuantas variables hay que reservar en el stack cada que se mande a llamar a función.

Después entra en un loop infinito donde ejecuta cuádruplo por cuádruplo de acuerdo a la especificación de OPCodes en la pagina 9. Este loop se apoya de las PILAS callstack, globalstack, stack y esp para ejecutar el código. Callstack siempre tiene información acerca de donde saltar cuando retorne una función. Globalstack tiene en memoria todo el tiempo los datos de la memoria global. Stack simula literalmente ser un stack de cualquier programa. Esp tiene información que sirve para apuntar al stack correspondiente a la función actual. La máquina se detiene una vez que ya no hay cuádruplos por ejecutar.

Para acceder a valores en memoria la máquina virtual utiliza una función llamada getValue(). La función se encarga de la lógica de referenciar al scope global o local, se encarga de referenciar a una variable normal o dimensionada. Todo está guardado en memoria como STRING. Es hasta que se realizan las operaciones que se castean al tipo requerido por la operación.

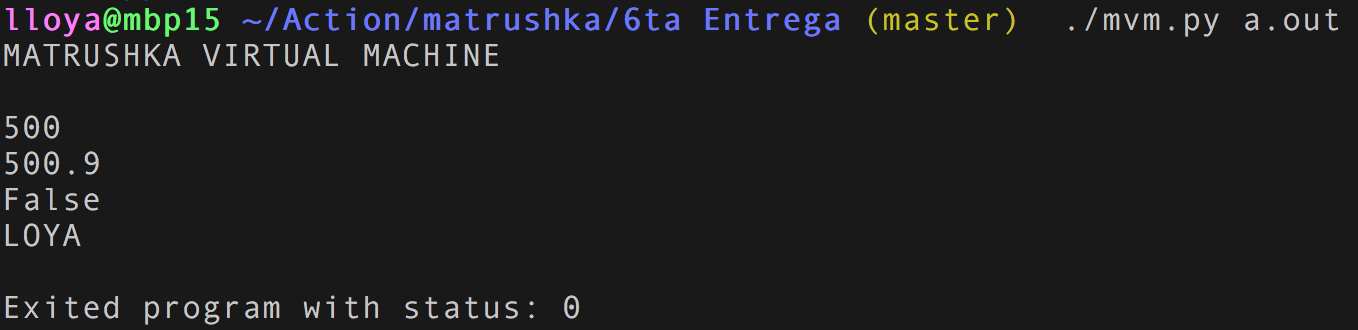
**Pruebas de Funcionalidad**

**Prueba1**

Esta prueba tiene todos los tipos, tiene llamadas a funciones y tiene IO

|  |  |
| --- | --- |
| **prueba1.mka**  **int** b;  func **bool**:orexclusivo(**bool** a,**bool** b):  **bool** c;  **bool** d;  c = a||b;  d = a!=b;  **return** c && d;  endf  func **double**:sumad (**double** a,**double** b):  **return** a + b;  endf  func **int**:sumai(**int** param1,**int** param2):  **return** param1 + param2;  endf  func **int**:main():  **int** resultado;  **double** resultadodobles;  **bool** resultadobool;  string test;  test = "LOYA"  resultado = sumai(300,200);  resultadodobles = sumad (300.5,200.4);  resultadobool = orexclusivo(**true**,**true**);  write resultado;  write resultadodobles;  write resultadobool;  write test;  endf | **prueba1.bab**  **ERAS** \*era\_size\_main\*  **CALL** \_main  \_orexclusivo**:**  **ORVL** (**%STACK+**2) (**%STACK+**1) (**%STACK+**5)  **MOVE** (**%STACK+**5) (**%STACK+**3)  **NEQL** (**%STACK+**2) (**%STACK+**1) (**%STACK+**6)  **MOVE** (**%STACK+**6) (**%STACK+**4)  **ANDV** (**%STACK+**4) (**%STACK+**3) (**%STACK+**7)  **RETV** (**%STACK+**7)  **ENDF**  \_sumad**:**  **DADD** (**%STACK+**2) (**%STACK+**1) (**%STACK+**3)  **RETV** (**%STACK+**3)  **ENDF**  \_sumai**:**  **IADD** (**%STACK+**2) (**%STACK+**1) (**%STACK+**3)  **RETV** (**%STACK+**3)  **ENDF**  \_main**:**  **MOVE** "LOYA" (**%STACK+**4)  **ERAS** \*era\_size\_sumai\*  **PUSH** 300  **PUSH** 200  **CALL** \_sumai  **POPV** (**%STACK+**5)  **MOVE** (**%STACK+**5) (**%STACK+**1)  **ERAS** \*era\_size\_sumad\*  **PUSH** 300.5  **PUSH** 200.4  **CALL** \_sumad  **POPV** (**%STACK+**6)  **MOVE** (**%STACK+**6) (**%STACK+**2)  **ERAS** \*era\_size\_orexclusivo\*  **PUSH** true  **PUSH** true  **CALL** \_orexclusivo  **POPV** (**%STACK+**7)  **MOVE** (**%STACK+**7) (**%STACK+**3)  **IPUT** (**%STACK+**1)  **DPUT** (**%STACK+**2)  **BPUT** (**%STACK+**3)  **SPUT** (**%STACK+**4)  **ENDF**  **.data:**  era\_size\_sumai**:** 3  era\_size\_main**:** 7  era\_size\_orexclusivo**:** 7  era\_size\_sumad**:** 3 |

Ejecución de prueba 1



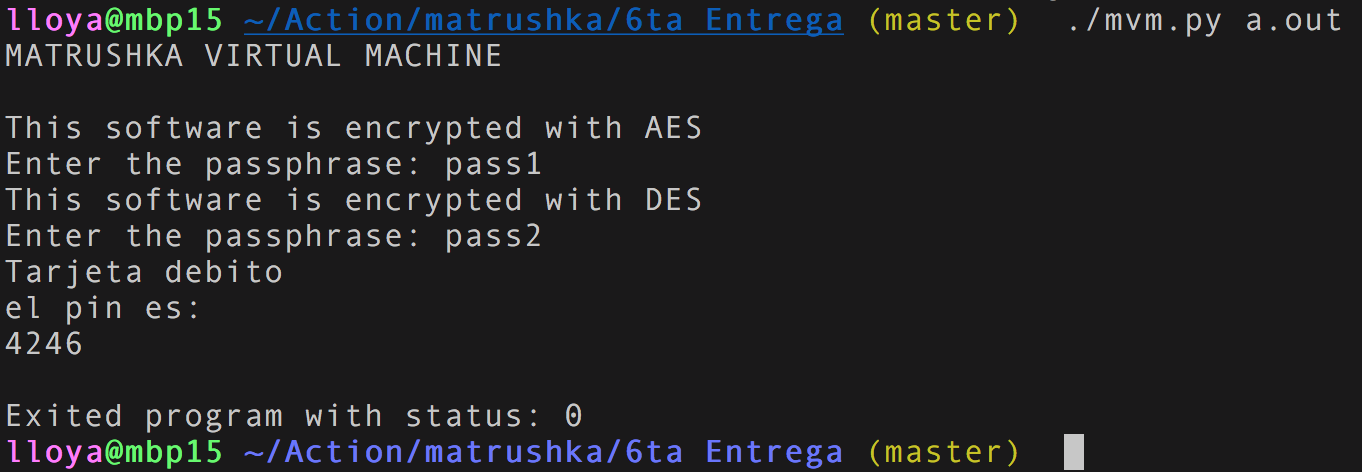
**Prueba2**

En esta prueba hacemos uso del feature principal “encriptación”

|  |  |
| --- | --- |
| **cipher** "aes256", "pass1";  **cipher** "des", "pass2";  **int** a;  func **int**:pinnumber():  **return** a;  endf  func **int**:main():  a = **4246**;  write "Tarjeta debito";  write "el pin es: ";  write pinnumber();  endf | **ASKP** AES  U2FsdGVkX1/1xirKJhaGrDTV0wXMXcI2  WNkyfItxbWuRIvAsN/8Bny15e4/ezB1c  jwuqcEflD9pKC2mSsH3lJTZQBheATDfo  HHL8lsoQQ1MB5/HRFDmf1yyfLJc/Z195  TQLTkCuC5EJRo+KuIsbYpyrt0hv3VXDy  w/YeP9beIv8pvu9o02lowEjtkJhHOIqq  W4rOMKqqLKQJoiLDR8ggMOcv1FOBdVlT  nXY7nyUSi9OG+bA2IzXQCMGRUw3lFkTh  Zki/23aEQKeL8RWfei1hjWKeXC1/DSgX  lL7IUzyK8D2LObyA6YbbHOqzeMAUkLsi  rku+s0U3S1X/1td4fd7Jqz/3LabzAV37  rDEkS+2VbwRXYJtFs+/yy1aI1HWW/p1R  GejhlGLTm/hmLYH44dFD6OCRuvnn60/j  VxIYqIfeJ0qOnveaLDMoIAtniCiMpMaB  znGR4FquOf2mcZ78Ez11bJRneFYRz+t5  C5IluDeCiQD2wGCvF/6JVgjcOJe9U5QU  bn8F5gDxMY2L0Zk+7y64mw1cg8Q+xF6w  sBk+RvT6IzY= |

Del lado izquierdo tenemos el código fuente y del lado derecho el código intermedio cifrado. Literalmente no se puede hacer análisis estático en el programa. Y solo funciona cuando se introducen las claves correctas.

Ejecución de prueba 2



**Lista de Documentación**

Los más relevantes van a estar enfatizados con **Negritas**.

/\*\*\*\*\*\*\* COMPILADOR \*\*\*\*\*\*\*\*/

*//Esta funcion lee un Lexema*

*//Regresa que tipo de dato es*

**char** \***getTypeFromValue**(char \*value);

*//Esta funcion lee un identificador (a.k.a una variable)*

*//Regresa el tipo de dato del identificador (int, double, string, boolean)*

**char** \***getTypeFromSymbol**(char \*symbol, char \*scope);

*//Esta funcion imprime la tabla de variables*

*//Solo se manda a llamar cuando la constante VERBOSE es 1*

**void** **DBG\_PrintSymbolTable**(struct symbol hashtable[]);

*//Esta funcion imprime el codigo intermedio del programa*

*//Solo se manda a llamar cuando la constante VERBOSE es 1*

**void** **DBG\_PrintIRCode**();

*// REGLA: program*

*// Esta funcion guarda el codigo en el FILE\* outputFile especificado en main*

**void** **PRG\_SaveIRCode**();

*// REGLA: expstmt, exp, term*

*// Esta funcion representa un cubo semantico que recibe un operador y sus operandos*

*// Regresa el tipo resultante*

**char** \* **PRG\_SemanticCube**(char\* op, char \*operand1, char \*operand2);

*// REGLA: program, function*

*// Esta funcion mantiene la global "scope" actualizada acerca de en que funcion*

*// se cuentra el parser.*

**void** **PRG\_SetScope**(char \*scopeName);

*// REGLA: param, vardeclaration, stmt*

*// Esta funcion accesa y regresa el valo de la global "scope".*

*// Para guardar cualquier identifier en el symbolTable es necesario tener su scope*

**char** \* **PRG\_GetScope**();

*// REGLA: callstmt*

*// Esta funcion recibe el identificador de un scope (function\_name o global)*

*// Regresa el numero de variables que se registraron para dicho scope*

**int** **PRG\_GetFunctionVariablesNum**(char \*identifier);

*// REGLA: program*

*// Esta funcion genera una seccion de datos al final del codigo intermedio*

*// La maquina virtual de matrushka necesita saber cuantas variables tiene cada funcion*

**void** **PRG\_ERAUpdate**();

*// REGLA: vardeclarations, params, factor*

*// Esta funcion se llama cada que se declara un identificador o genera una variable temporal*

*// Lo guarda en en el symbol table*

**void** **PRG\_SaveSymbol**(char \*typeName, char \*identifierName, char \*scopeName, char \*symbolKind);

*// REGLA: exp, callstmt*

*// Esta funcion se llama cada que se hace referencia a algun identificador*

*// Comprueba que el identificador se encuentre en el symbol table*

**void** **PRG\_GetSymbol**(char \*identifierName, char \*scopeName, char \*symbolKind);

*// REGLA: program*

*// Esta funcion se llama al inicio del programa, se encarga de inicializar*

*// las estructuras de datos. Tambien genera la primera instruccion de IR :)*

**void** **PRG\_Initialize**();

*// REGLA: secret*

*// Esta funcion se llama cada vez que el usuario definio algun cifrado*

*// Guarda los tipos de cifrado en un stack global.*

**void** **PRG\_PushCipher**(char \* cipher\_algorithm, char \* cipher\_key);

***// REGLA: program***

***// Esta funcion es la encarga de encriptar el codigo intermedio***

***// Utiliza el cifrado y el passphrase que el usuario definio***

***// Hace uso de openssl para cifrar.***

***// Regla de ORO: Dont role your own crypto ;)***

***void PRG\_EncryptCode(){***

***char \*cipher\_algorithm;***

***char \*cipher\_key;***

***char \*command;***

***char \*tmp\_string[2];***

***if(VERBOSE)***

***printf("LINE: %-4d PRG\_EncryptCode()\n", g\_lineno);***

***while(cipherStack.size!=0){***

***cipher\_key = stackPop(&cipherStack);***

***cipher\_algorithm = stackPop(&cipherStack);***

***//This is other level magic***

***tmp\_string[0] = concat("openssl enc -a ", cipher\_algorithm);***

***command = strdup(tmp\_string[0]);***

***free(tmp\_string[0]);***

***tmp\_string[0] = concat(" -k ", cipher\_key);***

***tmp\_string[1] = concat(command, tmp\_string[0]);***

***free(tmp\_string[0]);***

***tmp\_string[0] = concat(tmp\_string[1], " -out /tmp/babushka.sec");***

***free(command);***

***free(tmp\_string[1]);***

***command = strdup(tmp\_string[0]); // open ssl enc -a -aes-256-cbc -k password***

***free(tmp\_string[0]);***

***if(VERBOSE)***

***printf("\n%s\n", command);***

***FILE \*crypto\_engine = popen(command, "w");***

***for(int i=0;i<program\_counter;i++){***

***fprintf(crypto\_engine, "%s\n", IRCode[i]);***

***}***

***pclose(crypto\_engine);***

***//Now the encrypted code is in /tmp/babushka.sec***

***//We must read it back and replace IRCode***

***FILE \*cryptedfile;***

***char \* buffer = (char \*)malloc(100 \* sizeof(char));***

***size\_t len = 0;***

***ssize\_t linelen;***

***cryptedfile = fopen("/tmp/babushka.sec","r");***

***if (cryptedfile==NULL){***

***printf("LINE: %-4d CALL: PRG\_EncryptCode()\n", g\_lineno);***

***printf("FATAL: Could not open encrypted file /tmp/babushka.sec\n");***

***exit(EXIT\_FAILURE);***

***}***

***//Here we are going to replace IRCode with wathever it is on babushka.sec***

***program\_counter = 0;***

***if(strcmp(cipher\_algorithm,"-aes-256-cbc")==0)***

***IRCode[program\_counter] = strdup("ASKP AES");***

***if(strcmp(cipher\_algorithm,"-des-cbc")==0)***

***IRCode[program\_counter] = strdup("ASKP DES");***

***if(strcmp(cipher\_algorithm,"-bf-cbc")==0)***

***IRCode[program\_counter] = strdup("ASKP BLOWFISH");***

***program\_counter++;***

***while ((linelen = getline(&buffer,&len,cryptedfile))>0){***

***buffer[strlen(buffer) - 1] = 0;***

***IRCode[program\_counter] = strdup(buffer);***

***program\_counter++;***

***}***

***fclose(cryptedfile);***

***free(buffer);***

***}***

***}***

*// REGLA: REGLA(PushOperand)*

*// Subrutina de EXP\_PushOperand*

*// Se encarga de pushear al typeStack el tipo del identificador*

**void** **pushType**(char \*value, char \*symbolKind);

*// REGLA: REGLA(PushOperand)*

*// Subrutina de EXP\_PushOperand*

*// Se encarga de pushear al typeStack el operador*

**void** **pushOperand**(char \*operand, char \*kind)

***// REGLA: factor, expcmp, exp, term***

***// Esta funcion se encarga de pushear los operandos al stack***

**void EXP\_PushOperand(char \*operand, char \*symbolKind) {**

**if (NULL==operand || NULL==symbolKind){**

**printf("LINE: %-4d CALL: EXP\_PushOperand(NULL)\t", g\_lineno);**

**printf("FATAL: Null reference.\n");**

**exit(EXIT\_FAILURE);**

**}**

**if(VERBOSE)**

**printf("LINE: %-4d EXP\_PushOperand(%s,%s)\n", g\_lineno, operand,symbolKind);**

**//Check for the existance of a variable before even continuing.**

**if(strcmp(symbolKind,"var")==0)**

**PRG\_GetSymbol(operand, PRG\_GetScope(), "var");**

**//Later is the same for both expressions**

**pushType(operand, symbolKind);**

**pushOperand(operand, symbolKind);**

**}**

*// REGLA: expcmp, exp, term*

*// Esta funcion se encarga de pushear los operadores al stack*

**void** **EXP\_PushOperator**(char \*op);

*// Esta funcion se encarga de generar los labels de que ya "empezo una nueva funcion"*

**void** **IR\_MakeFUNCTION**(char \*identifier);

*// Esta funcion se encarga de generar el label de una funcion ya termino*

**void** **IR\_MakeFUNCTIONEND**();

*// Esta funcion se encarga de checar la semantica para el read y llama a la generacion de codigo intermedio*

**void** **IR\_MakeIOREAD**(char \*identifier);

*// Esta funcion se encarga de generar el codigo intermedio para escritura a pantalla*

**void** **IR\_MakeIOWRITE**();

*// Esta funcion se encarga de generar el codigo intermedio para asignacion a arreglos*

**void** **IR\_MakeARGYASSIGN**(char \*identifier);

*// Esta funcion se encarga de generar el codigo intermedio para asignacion normal*

**void** **IR\_MakeASSIGN**(char \*identifier);

*// Esta funcion se encarga de generar el codigo intermedio para ciclos*

**void** **IR\_MakeWHILE**();

*// Cierra el ciclo*

**void** **IR\_MakeENDWHILE**();

*// REGLA: IF*

*// Genera codigo para condicionales*

**void** **IR\_MakeIF**();

*// REGLA: IF*

*// Genera codigo para condicionales else*

**void** **IR\_MakeELSEIF**();

*// REGLA: IF*

*// Cierra la condicional*

**void** **IR\_MakeENDIF**();

*// REGLA: returnstmt*

*// Genera codigo para return de una funcion*

*// Hace el chequeo semantico tmb*

**void** **IR\_MakeRETURN**();

*// REGLA: callstmt*

*// Hace saber a la maquina virtual que ahi viene una funcion*

**void** **IR\_MakeERA**(char \*identifier);

*// REGLA: callstmt*

*// Hace saber a la maquina virtual los parametros de una funcion*

**void** **IR\_MakeSTACKPUSH**();

*// REGLA: callstmt*

*// Genera codigo para llamado de funciones*

**void** **IR\_MakeCALL**(char \*identifier);

*// Genera codigo para indicar el offset del arreglo*

**void** **IR\_MakeVER**();

***// Genera codigo para indicar el offset del arreglo multidimensionado***

**void IR\_MakeMULTIVER(char \*identifier) {**

**// Hubo una llamada a[exp][exp]**

**char \*operandB = stackPop(&operandStack);**

**stackPop(&typeStack);**

**char \*instruction;**

**char \*operandA = stackPop(&operandStack);**

**stackPop(&typeStack);**

**char \*firstDimension = memberType(SymbolTable, identifier, "\_dim");**

**// La sintaxis que genera es “MVER A B C”**

**// Para calcular el indice de la matriz se hace A\*C+B en el VM**

**instruction = concat("MVER ", concat(operandA, concat(" ", concat(operandB, concat(" ", firstDimension)))));**

**IRCode[program\_counter] = instruction;**

**program\_counter++;**

**}**

*// REGLA: expcmp, exp, term*

*// Genera codigo para expresiones*

**void** **IR\_MakeEXP**();

*// Se encarga de guardar arreglos en el symbol table*

**void** **save\_ary\_symbol**(char \*type, char \*identifier, char \*range);

*// Se encarga de guardar arregos multidimensionados en el symboltable*

**void** **save\_multiary\_symbol**(char \*type, char \*identifier, char \*first\_dimension, char \*second\_dimension);

*/\**

*Funcion de entrada al compilador*

*Realmente solo coordinada el input y el output del compilador*

*El codigo del compilador se encuentra en yacc.y*

*\*/*

**int** **main**(int argc, char \* argv[] );

/\*\*\*\*\*\*\* MAQUINA VIRTUAL \*\*\*\*\*\*\*\*/

*# Esta funcion recibe un operando que puede ser*

*# una constante o una direccion a una variable o vector o matriz*

*# El que quiero utilizar el valor no le interesa donde esta almacenado*

*# Le manda la chamba a esta funcion*

**def** **getValue**(*operand*):

*# Esta funcion recibe una direccion de memoria y un valor*

*# Se encarga de hacer el save. Una interfaz limpia para acceder a memoria*

**def** **save\_in**(*address*,*value*):

***#First decrypt in case it is encrypted***

***#Esta seccion se encarga de desencriptar el programa si es que esta encriptado***

***ceropass = open(ircode,'r').readlines()***

***while("ASKP" in ceropass[0]):***

***regs =re.findall(r'(\S+)', ceropass[0])***

***cipher = regs[1]***

***print("This software is encrypted with %s" % cipher)***

***if(cipher=="DES"):***

***cipher = "-des-cbc"***

***elif(cipher=="AES"):***

***cipher = "-aes-256-cbc"***

***elif(cipher=="BLOWFISH"):***

***cipher = "-bf-cbc"***

***else:***

***sys.exit();***

***password = input('Enter the passphrase: ')***

***//Este commando manda a llamar al programa openssl***

***//Por eso es requisite para ejecutar la aplicacion***

***proc = subprocess.Popen(["openssl","enc","-a",cipher,"-d","-k",password], stdout=subprocess.PIPE, stdin=subprocess.PIPE)***

***ciphertext = "\n".join(ceropass[1:])***

***try:***

***proc.stdin.write(str.encode(ciphertext))***

***proc.stdin.close()***

***result = proc.stdout.read()***

***ceropass = result.decode().splitlines(1)***

***except:***

***print("Couldn't decrypt software")***

***sys.exit()***

*# In the first pass we build the function\_adress and function\_era structures*

firstpass **=** ceropass

**for** line **in** **iter**(firstpass)

*# In the second pass we actually execute the code*

*# We keep advacing the EIP until we reach the .data section*

secondpass **=** ceropass

**while**(".data:" **not** **in** (secondpass[eip]))

**Manual de Usuario**

**Bienvenido a Matrushka!**

Matrushka es una lenguaje imperative sencillo que tiene como bonus el poder encriptar tus programas. Esto sirve para asegurarte que solo la persona que tiene que ejecutarlo pueda ejecutarlo. También sirve para enviar mensajes fácilmente descifrables. Realmente la imaginación es tu limite.

**Pre-Requisitos de Instalación**

Es necesario que tu sistema cuente con las siguientes características:

1.- POSIX Compliant.

2.- La libreria OpenSSL este instalada.

3.- Python3 este instalado.

4.- Flex y Bison estén instalados.

**Instalación**

Para instalar basta con correr el siguiente comando:

*make install*

**Sintaxis**

Una declaración formal de la sintaxis del lenguaje la puedes encontrar en el archivo “grammar.txt”

**Ejemplo1**

No hay mejor manera de aprender algo que usándolo.

Guarda el siguiente código como ***“debito.mka”***

|  |
| --- |
| int a;  func int:pinnumber():  return a;  endf  func int:main():  a = 4246;  write "Tarjeta debito";  write "el pin es: ";  write pinnumber();  endf |

Para compilar corre el siguiente comando:

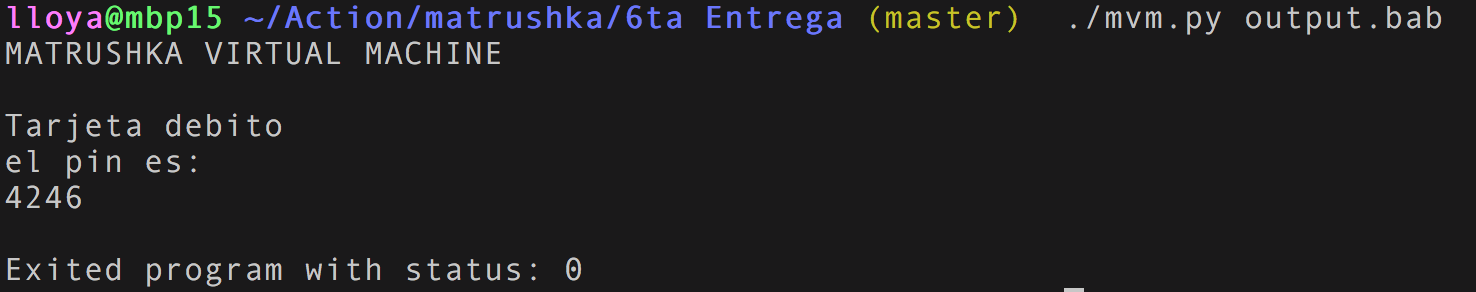
*./mcc debito.mka output.bab*

El comando debe haber compilado el archivo “debito.mka” y guardado el resultado en el archivo “output.bab”

Para ejecutar el resultado es necesario correr el siguiente comando:

./mvm.py output.bab

Resultado:



Ahora llevemos el ejemplo a otro nivel. Vamos a hacer uso del feature de encriptacion.

**Ejemplo2**

Guarda el siguiente código como ***“debito2.mka”***

|  |
| --- |
| cipher "aes256", "pass1";  cipher "des", "pass2";  cipher "blowfish", "pass3";  int a;  func int:pinnumber():  return a;  endf  func int:main():  a = 4246;  write "Tarjeta debito";  write "el pin es: ";  write pinnumber();  endf |

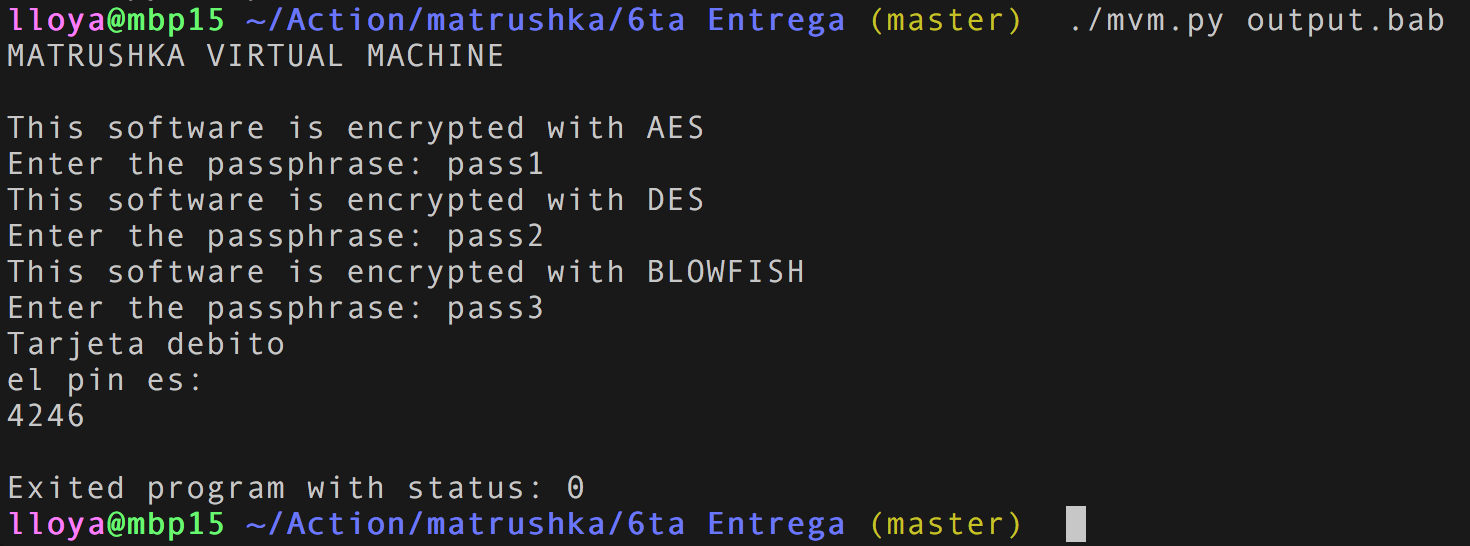
Para compilar corre el siguiente comando:

*./mcc debito2.mka output.bab*

El comando debe haber compilado el archivo “debito.mka” y guardado el resultado en el archivo “output.bab”

Para ejecutar el resultado es necesario correr el siguiente comando:

./mvm.py output.bab



Veamos cómo nos pidio ingresar la llave para poder descifrar el programa.

Para más ejemplos revisa la carpeta **/tests/** para que te des una idea de lo que se puede lograr con este lenguaje.

¡Gracias!