

Diseño de Compiladores

**MATRUSHKA**

Jose Luis Loya Cabrera A01111663

1. de Mayo, 2016

**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 Requerimiento**

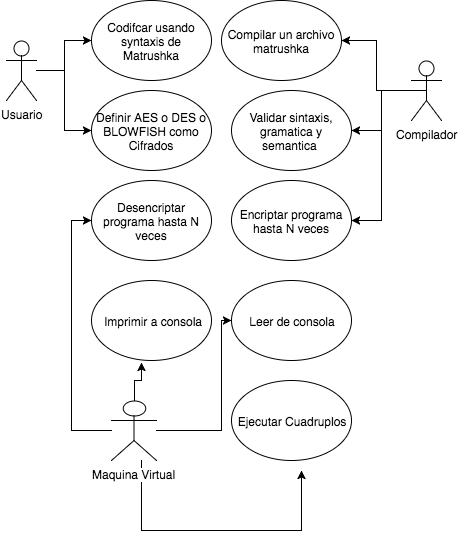
Requerimientos Funcionales:

* 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

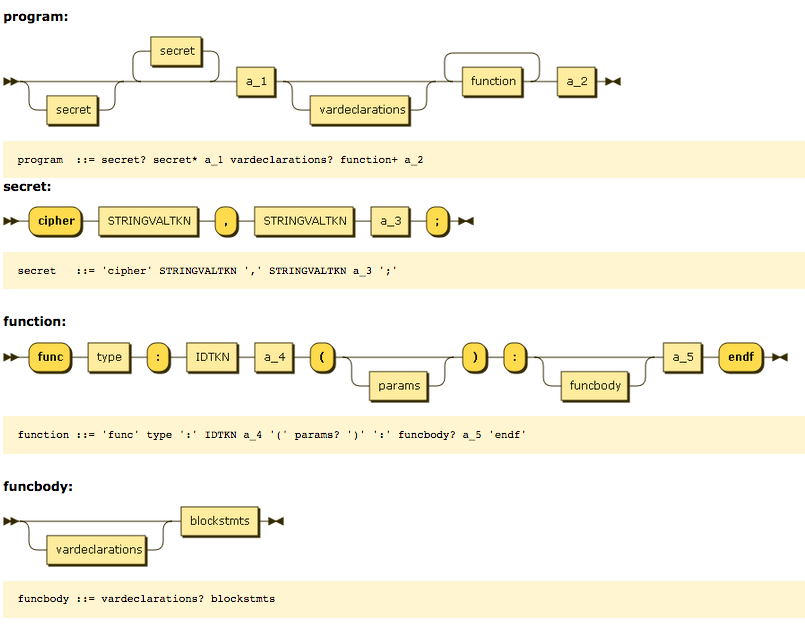
arglist : arglist COMMATKN exp a\_30 | exp a\_31

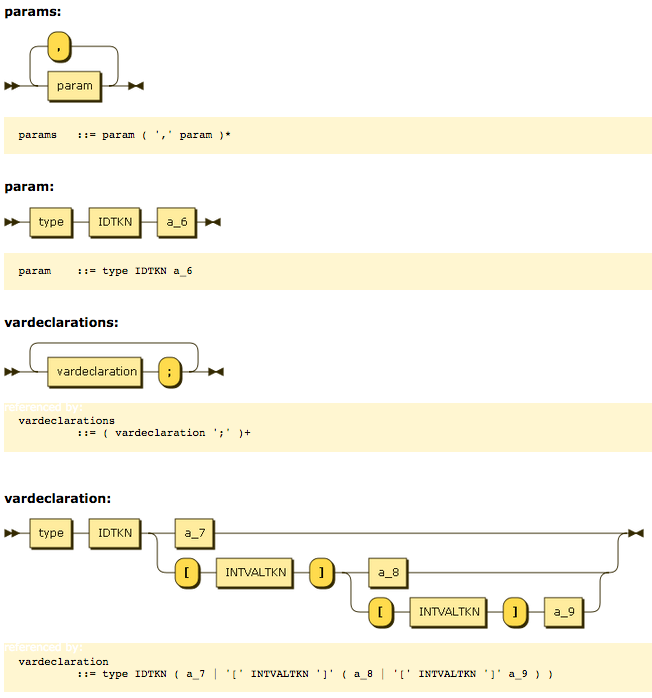
returnstmt : RETURNFUNCTKN expstmt a\_32

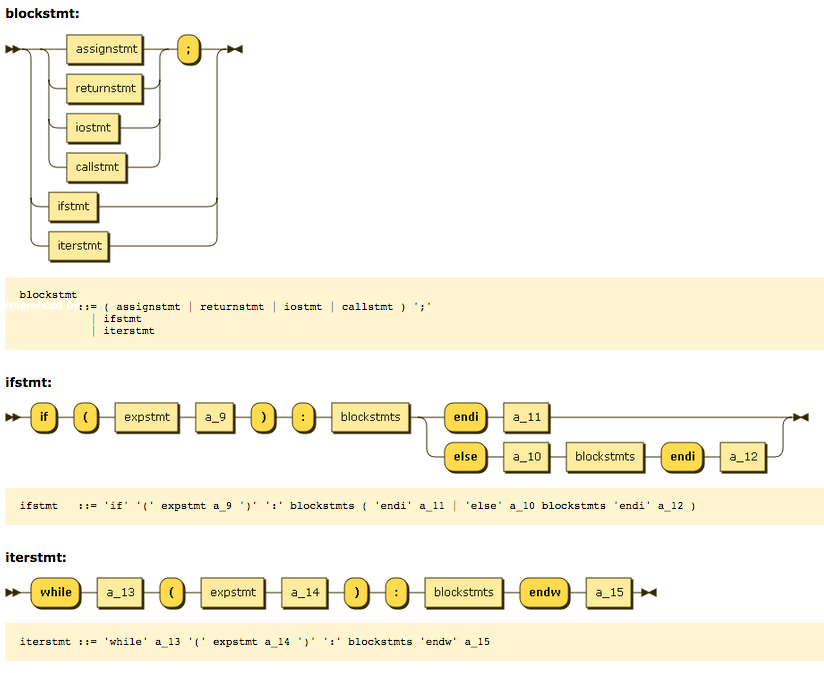
**Descripción de la Generación de Código**

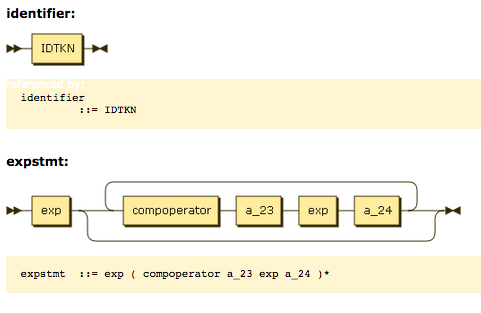
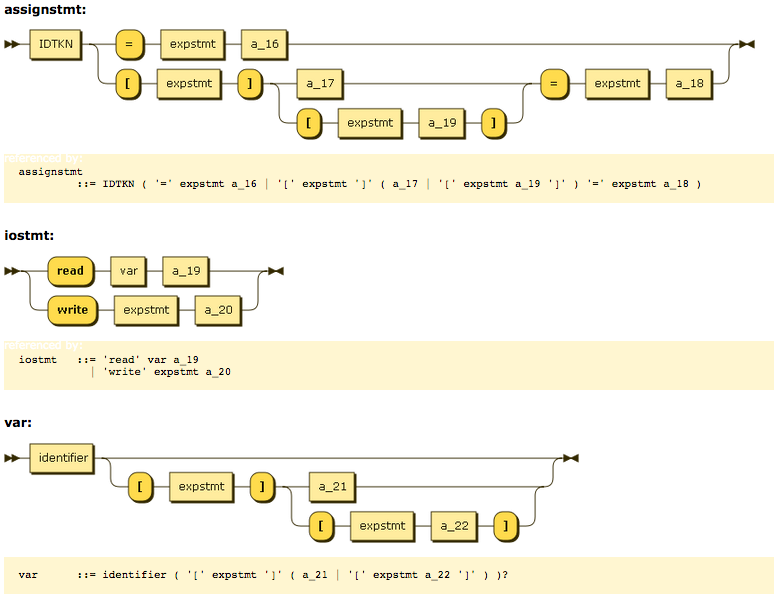
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| OPCODE | OP A | OP B | OP C | COMMENT |
| CALL | c |  |  | GOTO FUNCTION WITH THAT LABEL |
| ERAS | c |  |  | RESERVER SPACE IN STACK |
| PUSH | c/a |  |  | PUSH VALUE INTO STACK |
| AVER | c/a |  |  | OFFSET FOR ARRAY ACCESS |
| MVER | c/a | c/a | c | 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 |

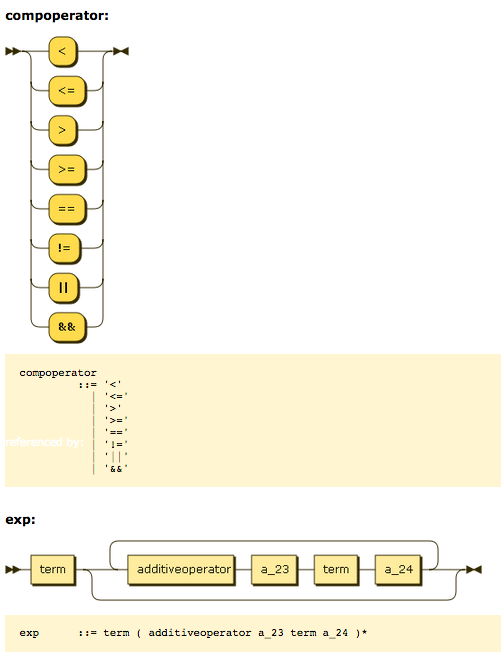
**Diagramas de Sintaxis con las acciones**

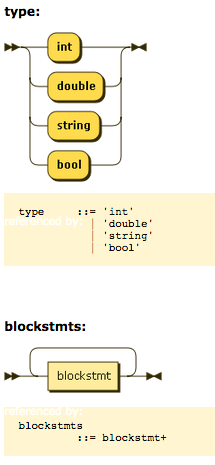
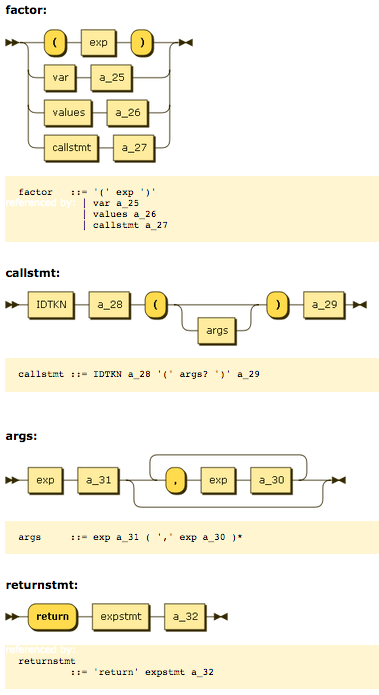
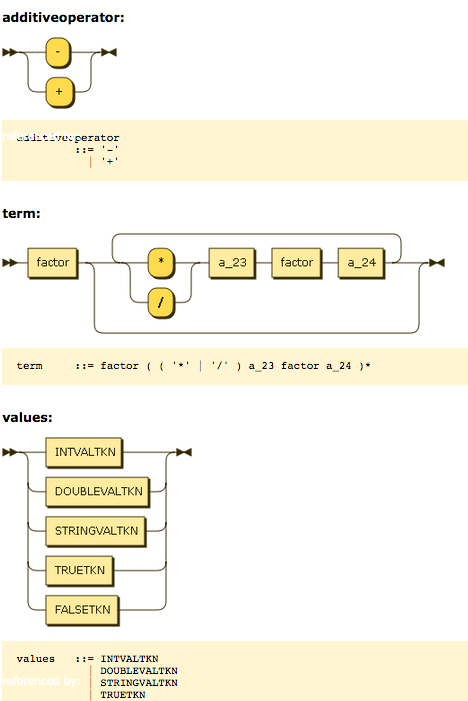
****

****

****

****

****

****

**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 Maquina Virtual esta programa en Python, se decidio 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 del Compilador**

Descripcion de la Maquina Virtual

Equipo de computo, lenguaje y utilerías especiales usadas

Descripcion detallada del proceso de Administracion de memoria

Estructguras y mapeo de virtuales a reales

Pruebas del funcionamiento del lenguaje

Listaods documentados del proyecto

Manual de Usuarios “Quick reference Manual para programadores”