Tecnológico de Monterrey

Campus Monterrey

Documentación de Proyecto: Sofi

Microsoft Office Signature Line...

Microsoft Office Signature Line...

20 de Noviembre de 2012

Índice

Descripción del Proyecto 4

Visión 4

Objetivo 4

Alcance del Proyecto 4

Análisis de Requerimientos y Casos de Uso generales 4

Casos de usos: 5

Descripción de TestCases 7

Descipción del Proceso de desarrollo 8

Reflexión Silvia 8

Reflexión Victor 8

Descripción del Lenguaje 9

Nombre del lenguaje 9

Características generales de Sofi 9

Errores que pueden ocurrir en compilación y ejecución 9

Compilación 9

Ejecución 9

Descripción del Compilador 10

Equipo de cómputo 10

Lenguaje y utilerías especiales 10

Análisis Léxico 11

Patrones de Construcción 11

Tokens del Lenguaje 11

Análisis de Sintaxis 12

Gramática Formal 12

Generación de Código Intermedio y Análisis Semántico 13

Códigos de operación y direcciones virtuales asociadas a elementos de código 13

Diagrama de Sintaxis con acciones correspondientes 14

Tabla de consideraciones semánticas 20

Administración de Memoria 20

Estructuras Utilizadas 20

Máquina Virtual 21

Equipo de Cómputo 21

Lenguaje y Utilerías utilizadas 21

Administración de Memoria en Ejecución 21

Especificación Gráfica de Estrcturas Utilizadas 21

Asociación entre Direcciones Virtuales y Reales 22

Pruebas 23

Factorial Iterativo 23

Factorial Recursivo 24

Fibonacci Recursivo 27

Listados del Proyecto 29

Compilador 29

Analizador de Léxico (FLEX) 29

Sintaxis y Semántica (Bison) 30

Estructura de Cuadruplos 37

Estructuras de Datos Generales, Dir de Procedimientos y Cubo Semantico 37

Estructura de Stack 41

Estructura de Stack para Constantes 41

Máquina Virtual 42

Manual de Usuario 44

Requisitos 44

Generación de Código por Gráficos 44

Compilación del Programa 44

Ejecución del Programa 44

Anexos 44

Descripción del Proyecto

# Descripción del Proyecto

## Visión

La visión de nuestro programa es ser el programa más utilizado en secundarias y preparatorias, tanto públicas como privadas, para la enseñanza de lógica y metodología para la programación, sentando las bases de un lenguaje genérico para que el aprendizaje de programas más sofisticados sea más fácil y amigable.

## Objetivo

El objetivo de nuestro lenguaje es ayudar a adolecentes en secundaria y preparatoria, a desarrollar la lógica necesaria para que puedan enfrentarse a lenguajes más completos y puedan sacar todo el provecho de dichos lenguajes, sin tener que ir aprendiendo desde lo más sencillo a lo más complejo. Esto con la finalidad de ser un impulso al crecimiento de matrículas en el área de la ingeniería de sistemas y que los recién ingresados cuenten con bases en la programación.

## Alcance del Proyecto

El alcance de nuestro proyecto es terminarlo completamente, con las funciones básicas de cualquier lenguaje de programación, declaración de variables, asignaciones, lectura, escritura, ciclos, condicionales, funciones y expresiones aritméticas y lógicas, añadiéndole una parte gráfica de input, para facilitarles a los jóvenes su uso.

## Análisis de Requerimientos y Casos de Uso generales

Sofi cuenta con los siguientes elementos: asignaciones, operaciones aritméticas, operaciones de comparación, operaciones booleanas, decisiones, while, for, y funciones. Son elementos básicos que todo lenguaje tiene que tener, aunque Sofi maneja estos elementos de manera sencilla y en español para que a los niños les sea más fácil entender lo que hace cada elemento. Aunque para programadores más experimentados, Sofi también tiene la capacidad de manejar funciones más complejas, como Fibonacci, entre otras.

### Casos de usos:

|  |  |
| --- | --- |
| **Nombre:** | Códifica en la Interfaz |
| **Descripción**  Se hace la codificación en la interfaz gráfica para después exportar el archivo de código. Hace más simple que el usuario recuerde la estructura del lenguaje. | |
| **Actores**  Usuario | |
| **Precondiciones** | |
| **Postcondiciones**  Se genera un archivo con el código que se dibujo en la interfaz. | |
| **Flujo Normal**   1. El usuario accede a la página de la interfaz 2. El usuario diseña su programa 3. El usuario guarda el archivo de código que diseño a su computadora | |
| **Flujo Alternativo**  [A: inicia nuevo programa]   * A1. Aparece una ventana de confirmación que su diseño actual se da eliminado * A2. Si confirma se limpia el código | |

|  |  |
| --- | --- |
| **Nombre:** | Códifica en un Editor de Texto |
| **Descripción**  El usuario alcanza un nivel en el cual conoce el lenguaje y desea poder tener mas flexibilidad que la ofrecida por la interfaz gráfica. | |
| **Actores**  Usuario | |
| **Precondiciones**  El usuario conoce el lenguaje mas a fondo. | |
| **Postcondiciones**  El archivo resultante es un archivo con código capaz de ser leido por el compilador. | |
| **Flujo Normal**   1. El usuario escribe el código en un editor 2. El usuario guarda el archivo en su computadora | |
| **Flujo Alternativo** | |

|  |  |
| --- | --- |
| **Nombre:** | Compila el Código |
| **Descripción**  El código del archivo fuente es pasado por un analizador de léxico y sintaxis para después compilar y generar código intermedio entendible por la máquina virtual. | |
| **Actores**  Usuario | |
| **Precondiciones**  El usuario cuenta con un archivo de código. | |
| **Postcondiciones**  Se genera un archivo objeto con código intermedio entendible por la máquina virtual. | |
| **Flujo Normal**   1. El usuario compila el código que diseño 2. El código intermedio es generado automaticamente | |
| **Flujo Alternativo**  [A: error]   * A1. Aparece un mensaje de error * A2. Se cancela la compilación | |

|  |  |
| --- | --- |
| **Nombre:** | Ejecuta en Máquina Virtual |
| **Descripción**  El código intermedio es interpretado y ejecutado por la máquina virtual para obtener los resultados esperados por el usuario. | |
| **Actores**  Usuario | |
| **Precondiciones**  Se cuenta con un archivo de código intermedio. | |
| **Postcondiciones**  El código empieza a ser ejecutado en la terminal del usuario. | |
| **Flujo Normal**   1. El usuario ejecuta su código intermedio 2. El usuario pasa por la ejecución del programa que diseño 3. Termina la ejecucion | |
| **Flujo Alternativo**  [A: Error]   * A1. Aparece un mensaje de que ocurrio un error. * A2. Se sale de la ejecución | |

## Descripción de TestCases

Los principales casos de prueba es comprobar que los elementos básicos funcionen perfectamente.

Aquí tenemos un ejemplo para probar que funcionen correctamente los elementos como, PARA y FUNCIONES.

func numero fibonacci(numero a)

realiza

SI (a MENOR 1) REALIZA

a = 0;

FIN

SI (a IGUAL 1) REALIZA

a = 1;

FIN

SI (a MAYOR 1) REALIZA

a = fibonacci(a-1)+fibonacci(a-2);

FIN

regresa(a);

fin

programa viko;

var numero i;

REALIZA

para (i = 0; i menor 10; i = i + 1;) REALIZA

imprime(fibonacci(i));

FIN

FIN

## Descipción del Proceso de desarrollo

El proceso de desarrollo del lenguaje se dividió en secciones a lo largo del semestre. Al inicio nos concentramos en la parte de léxico y sintaxis. Después de eso empezamos con cuádruplos, y esa parte nos tomó mucho tiempo, la mayor parte del semestre, ya que hubieron muchos errores y complicaciones en la parte de la implementación y ya después solucionado eso comenzamos con el proceso de hacer la máquina virtual.

### Reflexión Silvia

Realizar este compilador fue toda una experiencia para mi, ya que nunca había realizado algo así, y fue un reto muy grande no sólo por la parte de la programación sino también en la administración del tiempo. También este proyecto me hizo reflexionar en todas las operaciones que se tienen que hacer cuando se esta compilando un programa, es por eso que cada que este programando no voy a desperdiciar los arreglos y mucho menos matrices.

### Reflexión Victor

La clase de compiladores es una clase muy completa y sin duda un integrador de la Carrera. Sin duda, es una material difícil pero entretenida debido a su nivel de complejidad. Lo que mas me gusto es que hacer un compilador es una tarea que nos regresa a clases de estructuras de datos, la maquina de Von Neumann, algo de lenguaje ensamblador y manejo de apuntadores, cosas que son básicas para un programador pero no muchas veces se recuerdan. Lo que se me hizo mas relevante, fue el poder ver el todo de una función tan simple como una asignación de una variable nueva a nivel máquina y como poder mejorar la velocidad de nuestro código. Un ejemplo que se me quedo mucho en la cabeza es cuando hacemos un conteo de objetos contenidos por un arreglo dentro de un for en donde si ese conteo se realiza en el estatuto de comparación la maquina realizara el conteo cada ves que quiera verificar esa comparación para saber si finalizar el for o no. En sí un compilador integra muy bien lo aprendido por la carrera, incluso a documentar, y aunque el tiempo siento es reducido para realizar un proyecto excelente a la primera aprendes mucho y es algo que no olvidaras fácilmente.

Descripción del Lenguaje

# Descripción del Lenguaje

## Nombre del lenguaje

Lenguaje Sofi

## Características generales de Sofi

Es un lenguaje de input gráfico. En la interfaz, se va construyendo el programa a través de botones que generan los diferentes bloques de código. Para nombrar variables en la parte derecha debajo de los botones viene un submenú donde se podrá poner nombre a las variables, y todas las características que se necesiten cambiar. Los bloques de código que podrán deslizar para acomodarlos en el lugar que mejor convenga a los programadores, dándoles la facilidad moverlos cuando necesiten. Además esta herramienta es una página de internet por lo que es muy accesible y también muy rápida.

Un programa en Sofi se divide en 2 bloques: primero es la declaración de funciones y después la declaración de la función principal y dentro de esa función principal están las declaraciones de variables globales.

## Errores que pueden ocurrir en compilación y ejecución

### Compilación

Errores léxicos:

* Declaración de una variable con un tipo que no existe
* Nombre no válido para una variable
* Usar funciones inexistentes
* Declaración de un identificador inválida

Errores semánticos:

* Operación entre tipos incompatibles
* Asignación a una variable con un tipo inválido

### Ejecución

* Ciclos infinitos
* División entre 0
* Uso de int o floats de un tamaño más grande de lo permitido

Descripción del Compilador

# Descripción del Compilador

## Equipo de cómputo

**MacBook Pro**

**13 pulgadas: 2.9 GHz**

* Dual core Intel Core i7 de 2.9 GHz
* Turbo Boost de hasta 3.6 GHz
* Memoria de 8 GB de 1600 MHz
* Disco duro de 750 GB de 5400 rpm1
* Intel HD Graphics 4000

**MacBook**

**13 pulgadas: 2.24 GHz**

* Core 2 Duo de Intel de 2.24 GHz
* Tarjeta gráfica GeForce 9400M de NVIDIA con 256 MB de SDRAM DDR3 compartida con la memoria principal.
* Disco duro Serial ATA de 250GB a 5.400 rpm; disco opcional de 250, 320 o 500 GB a 5.400 rpm.
* 2 GB (dos módulos de un 1 GB) de SDRAM DDR3 (PC3-8500) a 1.066 MHz

## Lenguaje y utilerías especiales

Usamos los lenguajes Flex y Bison para programar léxico y sintaxis.

Para la semántica usamos C, con listas para realizar el cubo semántico así como la tabla de variables y procedimientos.

Para la interfaz gráfica, usamos HTML para realizar la página de Internet, y para logar el efecto de arrastrar los elementos usamos librerías de JQuery.

## Análisis Léxico

### Patrones de Construcción

**Id:** [a-zA-Z]+[0-9]\*

**cte.Numero:** [0-9]+

**cte.Decimal:** [0-9]+. [0-9]+

**cte.Texto:** [a-zA-Z]+

**cte.Char:** [a-zA-Z]

### Tokens del Lenguaje

**Palabras reservadas**

• PROGRAMA

• VAR

• IMPRIME

• SI

• SINO

• MIENTRAS

• PARA

• REALIZA

• FIN

• NUMERO

• DECIMAL

• CARACTER

• TEXTO

• IGUAL

• DIFERENTE

• MENOR

• Y

• O

**Operadores**

• +

• -

• \*

• /

**Separadores y Especiales**

• {

• }

• (

• )

• ;

• ,

## Análisis de Sintaxis

### Gramática Formal

<PROGRAM> 🡪 PROGRAMA2 programa id ; PROGRAM1 BLOQUE

<PROGRAM1> 🡪 vars PROMGRAM1

<PROGRAM1> 🡪 E

<PROGRAM2> 🡪 FUNC PROGRAM2 ;

<PROGRAM2> 🡪 E

<VARS> 🡪 var TIPO VARS1 id ;

<VARS1> 🡪 [ cte.Num ]

<VARS1> 🡪 E

<TIPO> 🡪 numero

<TIPO> 🡪 decimal

<TIPO> 🡪 caracter

<TIPO> 🡪 texto

<TIPO> 🡪 booleano

<BLOQUE> 🡪 realiza BLOQUE1 fin

<BLOQUE1> 🡪 ESTATUTO BLOQUE1

<BLOQUE1> 🡪 E

<ESTATUTO> 🡪 ASIGNACION

<ESTATUTO> 🡪 CONDICION

<ESTATUTO> 🡪 ESCRITURA

<ESTATUTO> 🡪 LECTURA

<ESTATUTO> 🡪 MIENTRAS

<ESTATUTO> 🡪 PARA

<ESTATUTO> 🡪 LLAMADA

<ASIGNACION> 🡪 id = EXPRESION ;

<CONDICION> 🡪 si ( EXPRESION ) realiza BLOQUE1 CONDICION1 fin

<CONDICION1> 🡪 sino BLOQUE1

<CONDICION1> 🡪 E

<ESCRITURA> 🡪 imprime ( ESCRITURA1 ) ;

<ESCRITURA1> 🡪 EXPRESION ESCRITURA2

<ESCRITURA2> 🡪 , ESCRITURA1

<ESCRITURA2> 🡪 E

<LECTURA> 🡪 lee ( VARIABLE ) ;

<MIENTRAS> 🡪 mientras ( EXPRESION ) BLOQUE

<PARA> 🡪 para ( ASIGNACION EXPRESION ; ASIGNACION ) BLOQUE

<FUNC> 🡪 func TIPO id ( FUNC1 ) VARSFUNC realiza BLOQUE1 FUNCRET fin

<FUNC1> 🡪 TIPO id FUNC2

<FUNC1> 🡪 E

<FUNC2> 🡪 , FUNC1

<FUNC2> 🡪 E

<VARSFUNC> 🡪 VARS VARSFUNC ;

<VARSFUNC>

<FUNCRET> 🡪 ret (EXP ) ;

<FUNCRET> 🡪 E

<LLAMADA> 🡪 id ( LLAMADA2 )

<LLAMADA2> 🡪 EXP LLAMADA3

<LLAMADA2> 🡪 E

<LLAMADA3> 🡪, LLAMADA2

<LLAMADA3> 🡪 E

<EXPRESION> 🡪 COND EXPRESION1

<EXPRESION1> 🡪 EXPRESION2 COND

<EXPRESION1> 🡪 E

<EXPRESION2> 🡪 mayor | menor | diferente | igual

<COND> 🡪 EXP COND1

<COND1> 🡪 y COND

<COND1> 🡪 o COND

<COND1> 🡪 E

<EXP> 🡪 TERMINO EXP1

<EXP1> 🡪 + EXP

<EXP1> 🡪 - EXP

<EXP1 > 🡪 E

<TERMINO> 🡪 FACTOR TERMINO1

<TERMINO1> 🡪 \* TERMINO

<TERMINO1> 🡪 / TERMINO

<TERMINO1> 🡪 E

<FACTOR> 🡪 ( EXPRESION )

<FACTOR> 🡪 VARCTE

<FACTOR1> 🡪 LLAMADA

<VARCTE> 🡪 ctenum | ctedec | ctex | ccar | true | false

<VARCTE> 🡪 VARIABLE

<VARIABLE> 🡪 id VARIABLE1

<VARIABLE1> 🡪 [ EXPRESION ]

<VARIABLE1> 🡪 E

## Generación de Código Intermedio y Análisis Semántico

### Códigos de operación y direcciones virtuales asociadas a elementos de código

**Direcciones de memoria**

Direcciones Globales

0 – 999 🡪 Caracter

1000 – 1999 🡪 Texto

2000 – 2999 🡪 Número

3000 – 3999 🡪 Decimal

4000 – 4999 🡪 Booleano

Direcciones Locales

5000 – 5999 🡪 Caracter

6000 – 6999 🡪 Texto

7000 – 7999 🡪 Número

8000 – 8999 🡪 Decimal

9000 – 9999 🡪 Booleano

Direcciones Temporales

10000 – 10999 🡪 Caracter

11000 – 11999 🡪 Texto

12000 – 12999 🡪 Número

13000 – 13999 🡪 Decimal

14000 – 14999 🡪 Booleano

Direcciones Constantes

15000 –15999 🡪 Caracter

16000 – 16999 🡪 Texto

17000 –17999 🡪 Número

18000 – 18999 🡪 Decimal

19000 –19999 🡪 Booleano

**Códigos de operación**

+ 🡪10

- 🡪 20

\* 🡪 30

/ 🡪 40

% 🡪 50

Y 🡪 60

O 🡪 70

Mayor 🡪 80

Menor 🡪90

Igual 🡪 110

Diferente 🡪100

= 🡪 120

GoTo 🡪 130

GoToF 🡪 140

GoToV 🡪 150

GoSub 🡪 160

Era 🡪 170

Return 🡪 180

Escribe 🡪 190

Lee 🡪 200

Println 🡪210

End 🡪 220

Param 🡪 230

Regresa 🡪 240

### Diagrama de Sintaxis con acciones correspondientes



1. Guardamos el tipo de variable que sería el nombre del método o función, en este caso seria “NP”.

Guardamos el alcance de la función: Global en este caso.

1. Guardamos el nombre la variable.

Guardamos en que función apareció esa variable.

Metemos el nombre de la variable (el nombre de la función) a la pila de ejecución

1. Mandamos llamar el método addProc, que es para agregar la función al directorio de procedimientos y asignar la tabla de variables.



1. Se guarda el nombre de la variable en una variable.
2. Se agrega esa variable a la tabla de variables asociando al directorio de procedimientos correspondiente.









1. Meter el id a la PilaOperadores
2. Meter el = a la PilaOperandos
3. Mandar llamar la función que se encarga de generar los cuadruplos
   1. Verificar que el tope de la pila de operadores será un 120, SI SÍ
   2. Sacar el operador
   3. Sacar los tipos de los id’s
   4. Si los tipos son iguales, ENTONCES,
   5. Sacarmos los tipos de la pila
   6. Sacamos los operandos de las pilas
   7. Los agregamos a los cuadruplos



1. Sacar el tipo de la ultima operación
   1. Si el tipo es igual a Booleano ENTONCES
   2. Sacamos el id de la pila de operadores
   3. Se genera el cuádruplo del GoToFalso
   4. Se hace un push a la PiladeSaltos
2. Genera el cuádruplo del GoTo

Sacar el salto en falso de la PiladeSaltos

Meter la línea del código falso

Hacer un push de la PiladeSaltos

1. Sacar el fin de la PiladeSaltos

Llenar el fin con el contador de líneas de código



1. Agrego el código de operación a la PilaOperadores
2. Agrego la expresión o resultado de la expresión al cuádruplo



1. Se genera el cuádruplo de la lectura, sacando la el operando de la pilaOperandos



1. Meter el cont a la PiladeSaltos
2. Sacar el temporal de la PilaTipos

Si el temporal es igual a booleano entonces

Sacar la variable de la PilaOperandos

Hacer el GoToF

Push a la PiladeSaltos

1. Sacar el falso de la PiladeSaltos.

Sacar el retorno de la PiladeSaltos

Generar el GoTo retorno

Rellenar el falso con la línea de código en cont

1. Dar de alta el nombre del procedimiento en el Directorio de Procedimientos
2. Ligar cada parámetro a la tabla de parámetros del Directorio de Procedimientos y dar de alta el tipo de parámetro en la tabla de Parámetros
3. Dar de alta en el Directorio de Procedimientos, el número de parámetros declarados
4. Dar de alta en el Directorio de Procedimientos, el número de variables locales definidas
5. Dar de alta en el Directorio de Procedimientos, el número de cuádruplo en el que inicia el procedimiento
6. Liberar la tabla de variables locales del procedimiento. Generar una acción de retorno



1. Verificar que el procedimiento exista en el Directorio de Procedimientos
2. Generar el cuádruplo ERA. Iniciar el contador de parámetros en 1
3. Obtener el argumento y el tipo del argumento de las pilas correspondientes. Verificar que el tipo del argumentos corresponda con el del parámetro K. Generar el cuádruplo de PARAMETRO
4. Apuntar al siguiente parámetro
5. Verificar que el último parámetro apunte a Nulo
6. Generar el cuádruplo de GOSUB













(Desde el diagrama de EXPRESION hasta el de variable, aplica los siguientes códigos de operación)

1. Meter la variable a la PilaOperandos y los tipos a la PiladeTipos
2. Meter los operadores a la PilaOperadores
3. Si el tope de la PilaOperadores es cualquier operador (+,-,\*,/,mayor, menor, diferente, igual) ENTONCES

Si el tipo de retorno de la variable no es “error” entonces

Sacamos los 2 tipos de la PiladeTipos

Sacamos los 2 operandos

Sacamos el operador de la PilaOperadores

Generamos la dirección virtual de la variable temporal

Agregamos el cuádruplo

Metemos la variable temporal a la PilaOperandos

Metemos el tipo a la PiladeTipos

1. Meter a la PilaOperadores el fondo falso
2. Sacar el fondo falso de la PilaOperadores

### Tabla de consideraciones semánticas

Está en el archivo de Excel adjunto en el zip

## Administración de Memoria

### Estructuras Utilizadas

#### Directorio de Procedimientos

Es una lista que contiene los siguientes:

|  |  |  |
| --- | --- | --- |
| Char \* | name | El nombre de la función o procedimiento |
| Char \* | returnType | El tipo de retorno de esta función |
| Int | virtualAddress | La dirección virtual de inicio de dicha función |
| \_node \* | localAddress | Apuntador al header de la lista de variables locales de esta función |
| \_node \* | next | Apuntador al siguiente apuntador. Nulo si es el último. |

La misma lista se usará la lista de las variables locales de un procedimiento con la diferencia de que virtualAddress es nulo porque no se sabe aún y localAddress es nulo porque tampoco se pueden tener variables locales de algo que no es función.

#### Cuádruplos

Es una lista que contiene lo siguiente:

|  |  |  |
| --- | --- | --- |
| int | operador | El código de operación a realizer. |
| int | dirOperando1 | Dirección del primer operando. |
| int | dirOperando2 | Dirección del Segundo operando. |
| int | temporal | Dirección de la variable responsable de almacenar el resultado. |
| \_cuadruplos \* | next | Apuntador al siguiente cuadruplo. |

#### Stacks

Cuando es necesario el manejo de stacks se uso la siguiente estructura.

|  |  |  |
| --- | --- | --- |
| void \* | ptr | Apuntador a la dirección del dato almacenado. |
| int | ptrType | Tipo del dato almacenado ya al ser void puede ser un int, char, char\*, etc. Esto para cuando necesitamos recuperar la información y el stack es heterogeneo. |
| Stack | next | Apuntador al siguiente nodo del stack. |

Máquina Virtual

# Máquina Virtual

## Equipo de Cómputo

**MacBook Pro**

**13 pulgadas: 2.9 GHz**

* Dual core Intel Core i7 de 2.9 GHz
* Turbo Boost de hasta 3.6 GHz
* Memoria de 8 GB de 1600 MHz
* Disco duro de 750 GB de 5400 rpm1
* Intel HD Graphics 4000

**MacBook**

**13 pulgadas: 2.24 GHz**

* Core 2 Duo de Intel de 2.24 GHz
* Tarjeta gráfica GeForce 9400M de NVIDIA con 256 MB de SDRAM DDR3 compartida con la memoria principal.
* Disco duro Serial ATA de 250GB a 5.400 rpm; disco opcional de 250, 320 o 500 GB a 5.400 rpm.
* 2 GB (dos módulos de un 1 GB) de SDRAM DDR3 (PC3-8500) a 1.066 MHz

## Lenguaje y Utilerías utilizadas

La maquina virtual fue realizada en el lenguaje Ruby sin utilizar librerias adicionales. Simplemente se usaron las librerias ya incluidas sobre todo las clases de Hash y Arreglos. Se utilizaron computadoras Mac debido a que el compilador de Ruby ya viene incluido en OSX.

## Administración de Memoria en Ejecución

### Especificación Gráfica de Estrcturas Utilizadas

#### Cuadruplos

Los hash de cuadruplos utilizan la llave para indicar el numero de cuadruplo mientras que el valor es una clase creada especialmente de la siguiente manera:

|  |  |
| --- | --- |
| **Cuadruplo** | |
| int | opId |
| int | oper1 |
| Int | oper2 |
| int | result |

#### Directorio de Procedimientos

Para esta estructura se utiliza un array en el cual la posición indica el número del procedimiento mientras que el valor contiene una estructura de la siguiente manera:

|  |  |
| --- | --- |
| **Procs** | |
| string | Name |
| string | returnType |
| int | Params |
| int | initCuad |
| int | returnAddress |

### Asociación entre Direcciones Virtuales y Reales

La memoria virtual es manejada de manera de Hash en el cual la llave es la dirección de memoria y el valor es el valor de dicha memoria. Es una memoria totalmente homogénea por los valores del hash pueden tener cualquier tipo de valor. Las variables globales, locales, temporales, constantes y sus diferentes tipos están delimitadas de la misma manera que el compilador, de manera que la dirección es la cual indica que tipo de variable es y si es local, global, etc.

#### Stack de Funciones

Se utilizo la clase Array de manera en que se almacena objetos con la misma estructura de la memoria virtual pero que almacena solamente las partes de memoria local y temporal para después poder ser recuperadas.

Pruebas

# Pruebas

## Factorial Iterativo

**Código**

PROGRAMA factorialIterarivo;

VAR numero x;VAR numero n;

REALIZA

IMPRIME("Numero a sacar su factorial:");

LEE(n); x = 1;

MIENTRAS (n MAYOR 1)

REALIZA

x = x \* n;

n = n - 1;

FIN

IMPRIME("Resultado: ",x);

FIN

**Cuádruplos**

16001,"Resultado: "

17002,1

17001,1

17000,1

16000,"Numero a sacar su factorial:"

#

130,-1,-1,2

190,-1,-1,16000

210,-1,-1,-1

200,-1,-1,2001

120,17000,-1,2000

80,2001,17001,14000

140,14000,-1,13

30,2000,2001,12000

120,12000,-1,2000

20,2001,17002,12001

120,12001,-1,2001

130,-1,-1,6

190,-1,-1,16001

190,-1,-1,2000

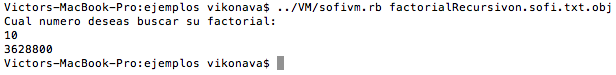
210,-1,-1,-1

220,-1,-1,-1

#

factorialIterarivo,NP,0,0,0

**Pantalla**



## Factorial Recursivo

****

**Código**

FUNC numero factorial(numero a)

REALIZA

SI (a MAYOR 1)

REALIZA

a = factorial(a-1) \* a;

SINO

FIN

REGRESA(a);

FIN

PROGRAMA factorialRecursivo;

VAR numero n;

REALIZA

IMPRIME("Cual numero deseas buscar su factorial: ");

LEE(n);

IMPRIME(factorial(n));

FIN

**Cuádruplos**

16000,"Cual numero deseas buscar su factorial: "

17001,1

17000,1

#

130,-1,-1,14

80,7000,17000,14000

140,14000,-1,12

20,7000,17001,12000

170,0,-1,-1

230,12000,-1,1

160,0,-1,-1

120,2000,-1,12001

30,12001,7000,12002

120,12002,-1,7000

130,-1,-1,12

240,7000,-1,-1

180,-1,-1,-1

190,-1,-1,16000

210,-1,-1,-1

200,-1,-1,2001

170,0,-1,-1

230,2001,-1,1

160,0,-1,-1

120,2000,-1,12003

190,-1,-1,12003

210,-1,-1,-1

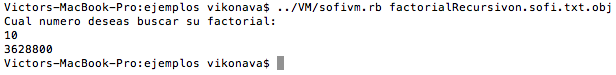
220,-1,-1,-1

#

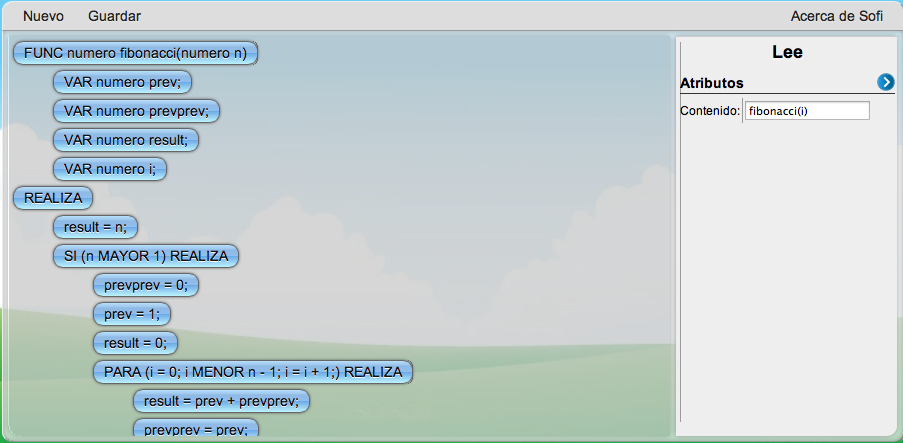
factorial,numero,1,1,2000

factorialRecursivo,NP,0,0,0

**Pantalla**

****

**Fibonacci Iterativo**

****

**Código**

FUNC numero fibonacci(numero n)

VAR numero prev;

VAR numero prevprev;

VAR numero result;

VAR numero i;

REALIZA

result = n;

SI (n MAYOR 1)

REALIZA

prevprev = 0;

prev = 1;

result = 0;

PARA (i = 0; i MENOR n - 1; i = i + 1;)

REALIZA

result = prev + prevprev;

prevprev = prev;

prev = result;

FIN

SINO

FIN

REGRESA(result);

FIN

PROGRAMA fibonacciIterativo;

VAR numero i;

VAR numero x;

REALIZA

IMPRIME("Numero maximo de Fibonacci: ");

LEE(x);

PARA (i = 0; i MENOR x + 1; i = i + 1;)

REALIZA

IMPRIME(fibonacci(i));

FIN

IMPRIME("");

FIN

**Cuádruplos**

16001,""

17009,1

17008,1

17007,0

16000,"Numero maximo de Fibonacci: "

17006,1

17005,1

17004,0

17003,0

17002,1

17001,0

17000,1

#

130,-1,-1,24

120,7000,-1,7003

80,7000,17000,14000

140,14000,-1,22

120,17001,-1,7002

120,17002,-1,7001

120,17003,-1,7003

120,17004,-1,7004

20,7000,17005,12000

90,7004,12000,14001

140,14001,-1,21

130,-1,-1,16

10,7004,17006,12001

120,12001,-1,7004

130,-1,-1,9

10,7001,7002,12002

120,12002,-1,7003

120,7001,-1,7002

120,7003,-1,7001

130,-1,-1,13

130,-1,-1,22

240,7003,-1,-1

180,-1,-1,-1

190,-1,-1,16000

210,-1,-1,-1

200,-1,-1,2002

120,17007,-1,2001

10,2002,17008,12003

90,2001,12003,14002

140,14002,-1,42

130,-1,-1,35

10,2001,17009,12004

120,12004,-1,2001

130,-1,-1,28

170,0,-1,-1

230,2001,-1,1

160,0,-1,-1

120,2000,-1,12005

190,-1,-1,12005

210,-1,-1,-1

130,-1,-1,32

190,-1,-1,16001

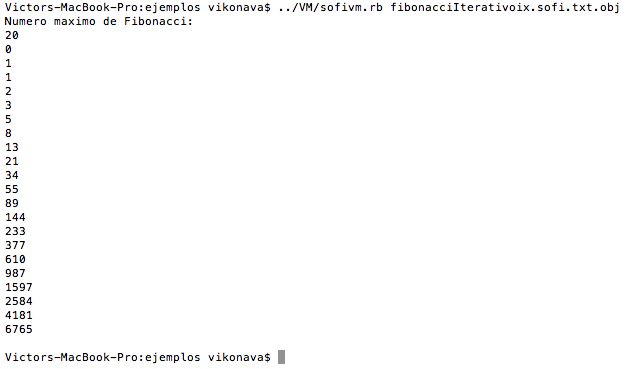
210,-1,-1,-1

220,-1,-1,-1

#

fibonacci,numero,1,1,2000

fibonacciIterativo,NP,0,0,0

**Pantalla**

## Fibonacci Recursivo



**Código**

FUNC numero fibonacci(numero a)

REALIZA

SI (a MENOR 1)

REALIZA

a = 0;

SINO

FIN

SI (a MAYOR 1)

REALIZA

a = fibonacci(a-1) + fibonacci(a-2);

SINO

FIN

REGRESA(a);

FIN

PROGRAMA fibonacciRecursivo;

VAR numero x;

VAR numero i;

REALIZA

IMPRIME("Numero maximo a buscar fibonacci:");

LEE(x);

PARA (i = 0; i MENOR x + 1; i = i + 1;)

REALIZA

IMPRIME(fibonacci(i));

FIN

FIN

**Cuádruplos**

17007,1

17006,1

17005,0

16000,"Numero maximo a buscar fibonacci:"

17004,2

17003,1

17002,1

17001,0

17000,1

#

130,-1,-1,23

90,7000,17000,14000

140,14000,-1,6

120,17001,-1,7000

130,-1,-1,6

80,7000,17002,14001

140,14001,-1,21

20,7000,17003,12000

170,0,-1,-1

230,12000,-1,1

160,0,-1,-1

120,2000,-1,12001

20,7000,17004,12002

170,0,-1,-1

230,12002,-1,1

160,0,-1,-1

120,2000,-1,12003

10,12001,12003,12004

120,12004,-1,7000

130,-1,-1,21

240,7000,-1,-1

180,-1,-1,-1

190,-1,-1,16000

210,-1,-1,-1

200,-1,-1,2001

120,17005,-1,2002

10,2001,17006,12005

90,2002,12005,14002

140,14002,-1,41

130,-1,-1,34

10,2002,17007,12006

120,12006,-1,2002

130,-1,-1,27

170,0,-1,-1

230,2002,-1,1

160,0,-1,-1

120,2000,-1,12007

190,-1,-1,12007

210,-1,-1,-1

130,-1,-1,31

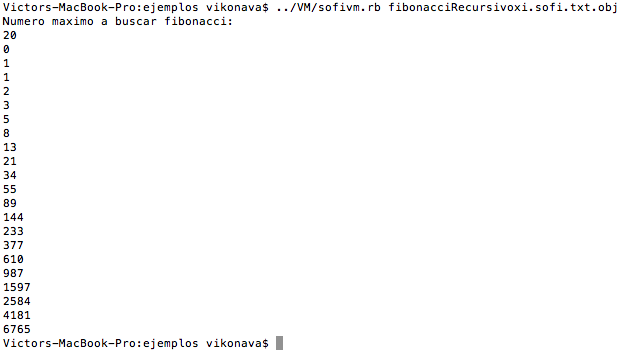
220,-1,-1,-1

#

fibonacci,numero,1,1,2000

fibonacciRecursivo,NP,0,0,0

**Pantalla**

****

# Listados del Proyecto

## Compilador

### Analizador de Léxico (FLEX)

Archivo: lexico.l

/\*

Silvia Fosado Garcia 1032432

Victor Nava 617610

Lexico del Lenguaje

\*/

%{

#include <stdio.h>

#include "sintaxis.tab.h"

using namespace std;

#define YY\_DECL extern "C" int yylex()

%}

%option yylineno

%%

programa {return PROGRAMA;}

PROGRAMA {return PROGRAMA;}

var {return VAR;}

VAR {return VAR;}

numero {return NUMERO;}

NUMERO {return NUMERO;}

regresa {return RET;}

REGRESA {return RET;}

decimal {return DECIMAL;}

DECIMAL {return DECIMAL;}

texto {return TEXTO;}

TEXTO {return TEXTO;}

caracter {return CARACTER;}

CARACTER {return CARACTER;}

booleano {return BOOLEANO;}

BOOLEANO {return BOOLEANO;}

imprime {return IMPRIME;}

IMPRIME {return IMPRIME;}

lee { return LEE;}

LEE { return LEE;}

si {return SI;}

SI {return SI;}

sino {return SINO;}

SINO {return SINO;}

y {return Y;}

Y {return Y;}

o {return O;}

O {return O;}

realiza {return REALIZA;}

REALIZA {return REALIZA;}

fin {return FIN;}

FIN {return FIN;}

mientras {return MIENTRAS;}

MIENTRAS {return MIENTRAS;}

para {return PARA;}

PARA {return PARA;}

func {return FUNC;}

FUNC {return FUNC;}

igual {return IGUAL;}

IGUAL {return IGUAL;}

mayor {return MAYOR;}

MAYOR {return MAYOR;}

menor {return MENOR;}

MENOR {return MENOR;}

diferente {return DIFERENTE;}

DIFERENTE {return DIFERENTE;}

verdadero {return TRUE;}

falso {return FALSE;}

VERDADERO {return TRUE;}

FALSO {return FALSE;}

[a-zA-Z][a-zA-Z]\* { yylval.str = strdup(yytext); return ID;}

[0-9][0-9]\* {yylval.str = strdup(yytext); return CTENUM;}

[0-9][0-9]\*\.[0-9][0-9]\* {yylval.str = strdup(yytext); return CTEDEC;}

\"[^\"]\*\" {yylval.str = strdup(yytext); return CTEX;}

\'[a-zA-Z]\' {yylval.str = strdup(yytext); return CCAR;}

"," {return COMA;}

";" {return PCOMA;}

"(" {return ABREPA;}

")" {return CIERRAPA;}

"[" {return ABRECORCH;}

"]" {return CIERRACORCH;}

"+" {return MAS;}

"-" {return MENOS;}

"\*" {return POR;}

"/" {return ENTRE;}

"=" {return SIGUAL;}

\n

\t

\b

\r

.

%%

### Sintaxis y Semántica (Bison)

Archivo: sintaxis.y

/\*

Silvia Fosado Garcia 1032432

Victor Nava 617610

Programa BISON para el Manejo de Sintaxis

\*/

%{

#include <stdio.h>

#include <iostream>

#include "estructura.c"

#include "cuadruplos.c"

#include "stack.c"

#include "stackConstantes.c"

using namespace std;

#define YYERROR\_VERBOSE 1

extern "C" int yylex();

extern "C" int yyparse();

extern "C" FILE \*yyin;

extern int yylineno;

extern void insert\_to\_dirprocs(char \*name, char \*returnType);

void yyerror(const char \*s);

typedef struct {

char \*name;

char \*returnType;

int local\_variables\_address;

int next;

} dir\_procs;

// Base de Direcciones de Memoria Virtuales

int GCAR = 0, GTEXTO = 1000, GINT = 2000, GFLOAT = 3000, GBOOL = 4000, LCAR = 5000, LTEXTO = 6000, LINT = 7000, LFLOAT = 8000, LBOOL = 9000, TCAR = 10000, TTEXTO = 11000, TINT = 12000, TFLOAT = 13000, TBOOL = 14000, CTCAR = 15000, CTTEXTO = 16000, CTINT = 17000, CTFLOAT = 18000, CTBOOL = 19000;

// Offsets de Direcciones de Memoria

int global\_car\_offset = 0, global\_text\_offset = 0, global\_int\_offset = 0, global\_float\_offset = 0, global\_bool\_offset = 0, local\_car\_offset = 0, local\_text\_offset = 0, local\_int\_offset = 0, local\_float\_offset = 0, local\_bool\_offset = 0, temp\_car\_offset = 0, temp\_text\_offset = 0, temp\_int\_offset = 0, temp\_float\_offset = 0, temp\_bool\_offset = 0, const\_car\_offset = 0, const\_text\_offset = 0, const\_int\_offset = 0, const\_float\_offset = 0, const\_bool\_offset = 0;

// Declaracion Variables

Stack \*PilaOperandos; // Creamos la pila de Operandos

Stack \*PilaOperadores; // Creamos la pila de Operadores

Stack \*PiladeSaltos; // Creamos la pila Saltos ints de linea de cuadruplos

Stack \*PiladeSaltosCuad; // Creamos la pila de Saltos que almacena cuadruplos

Stack \*PiladeTipos; //Creamos la pila de los tipos para checar la semantica

Stack \*PilaEjecucion; // Se guarda los procedimientos en los que estamos

StackConstantes \*PilaConstantes; // Creamos la pila de constantes

char \*variable\_type; // Tipo de la ultima variable conocida

char \*variable\_name; // Nombre de la ultima variable conocida

char \*last\_func; // Guarda el nombre de la ultima funcion utilizada

char \*tipo\_func; // Guarda el tipo de retorno de la funcion declarada

int contadorCuad = 0; // Guarda el contador de los cuadruplos

int contadorK; // Guarda el contador de los parametros

Cuadruplos \*cuadruplo; // Crea el apuntador al primer pointer de los cuadruplos

Node \*dirProcsInit; // Crea el apuntador al primer pointer del dir procs

SemanticCubeNode \*semanticCube;; // Apuntador al primer nodo del cubo semantico

// FUNCTION: alloc\_virtual\_address(char \*returnType, char \*scope)

// RETURN: int

//

// Crea y regresa la direccion virtual para los diferentes tipos de variables

//

int alloc\_virtual\_address(char \*returnType, char \*scope) {

if (strcmp(scope,(char \*)"global") == 0) {

if (strcmp(returnType,(char \*)"caracter") == 0) {

global\_car\_offset++;

return GCAR+global\_car\_offset-1;

} else if (strcmp(returnType,(char \*)"texto") == 0) {

global\_text\_offset++;

return GTEXTO+global\_text\_offset-1;

} else if (strcmp(returnType,(char \*)"numero") == 0) {

global\_int\_offset++;

return GINT+global\_int\_offset-1;

} else if (strcmp(returnType,(char \*)"decimal") == 0) {

global\_float\_offset++;

return GFLOAT+global\_float\_offset-1;

} else if (strcmp(returnType,(char \*)"booleano") == 0) {

global\_bool\_offset++;

return GBOOL+global\_bool\_offset-1;

} else return -1;

} else if (strcmp(scope,(char \*)"local") == 0) {

if (strcmp(returnType,(char \*)"caracter") == 0) {

local\_car\_offset++;

return LCAR+local\_car\_offset-1;

} else if (strcmp(returnType,(char \*)"texto") == 0) {

local\_text\_offset++;

return LTEXTO+local\_text\_offset-1;

} else if (strcmp(returnType,(char \*)"numero") == 0) {

local\_int\_offset++;

return LINT+local\_int\_offset-1;

} else if (strcmp(returnType,(char \*)"decimal") == 0) {

local\_float\_offset++;

return LFLOAT+local\_float\_offset-1;

} else if (strcmp(returnType,(char \*)"booleano") == 0) {

local\_bool\_offset++;

return LBOOL+local\_bool\_offset-1;

} else return -1;

} else if (strcmp(scope,(char \*)"temporal") == 0) {

if (strcmp(returnType,(char \*)"caracter") == 0) {

temp\_car\_offset++;

return TCAR+temp\_car\_offset-1;

} else if (strcmp(returnType,(char \*)"texto") == 0) {

temp\_text\_offset++;

return TTEXTO+temp\_text\_offset-1;

} else if (strcmp(returnType,(char \*)"numero") == 0) {

temp\_int\_offset++;

return TINT+temp\_int\_offset-1;

} else if (strcmp(returnType,(char \*)"decimal") == 0) {

temp\_float\_offset++;

return TFLOAT+temp\_float\_offset-1;

} else if (strcmp(returnType,(char \*)"booleano") == 0) {

temp\_bool\_offset++;

return TBOOL+temp\_bool\_offset-1;

} else return -1;

} else if (strcmp(scope,(char \*)"constante") == 0) {

if (strcmp(returnType,(char \*)"caracter") == 0) {

const\_car\_offset++;

return CTCAR+const\_car\_offset-1;

} else if (strcmp(returnType,(char \*)"texto") == 0) {

const\_text\_offset++;

return CTTEXTO+const\_text\_offset-1;

} else if (strcmp(returnType,(char \*)"numero") == 0) {

const\_int\_offset++;

return CTINT+const\_int\_offset-1;

} else if (strcmp(returnType,(char \*)"decimal") == 0) {

const\_float\_offset++;

return CTFLOAT+const\_float\_offset-1;

} else if (strcmp(returnType,(char \*)"booleano") == 0) {

const\_bool\_offset++;

return CTBOOL+const\_bool\_offset-1;

} else return -1;

} else return -1;

}

// FUNCTION: meterAPilaOperandos(Node \*dirProcsInit, char\* variable\_name, char\* last\_func)

// RETURN: void

//

// Funcion encargada de meter los ID's a la PilaOperandos

//

void meterAPilaOperandos(Node \*dirProcsInit, char\* variable\_name, char\* last\_func){

Node \*tNode = dirProcsInit;

Node \*variableEncontrada = NULL;

char \*tipoVariable;

variableEncontrada = findLocalVariableInProc(tNode, static\_cast<char\*>(PilaEjecucion->ptr),variable\_name);

tipoVariable = variableEncontrada->returnType;

stack\_push(&PilaOperandos, &variableEncontrada->virtual\_address, 2);

stack\_push(&PiladeTipos, (findLocalVariableInProc(tNode, static\_cast<char\*>(PilaEjecucion->ptr),variable\_name))->returnType, 1);

}

// FUNCTION: meterConstanteAPilaOperandos(char\* tipo, char \*constante)

// RETURN: void

//

// Funcion encargada de meter las constantes a la PilaOperandos y PilaConstantes

//

void meterConstanteAPilaOperandos(char\* tipo, char \*constante){

int \*direccion = (int \*)malloc(sizeof(int));

if(strcmp(tipo, (char \*) "numero") == 0){

\*direccion = alloc\_virtual\_address((char \*) "numero", (char \*) "constante");

stack\_push(&PilaOperandos, direccion, 2);

stack\_push(&PiladeTipos, (char \*) "numero", 1);

int \*cons = (int \*)malloc(sizeof(int));

\*cons = atoi(constante);

stackC\_push(&PilaConstantes, cons, \*direccion, 2);

}

if(strcmp(tipo, (char \*) "decimal") == 0){

\*direccion = alloc\_virtual\_address((char \*) "decimal", (char \*) "constante");

stack\_push(&PilaOperandos, direccion, 2);

stack\_push(&PiladeTipos, (char \*) "decimal", 1);

float \*cons = (float \*)malloc(sizeof(float));

\*cons = atof(constante);

stackC\_push(&PilaConstantes, cons, \*direccion, 3);

}

if(strcmp(tipo, (char \*) "caracter") == 0){

\*direccion = alloc\_virtual\_address((char \*) "caracter", (char \*) "constante");

stack\_push(&PilaOperandos, direccion, 2);

stack\_push(&PiladeTipos, (char \*) "caracter", 0);

char \*cons = (char \*)malloc(sizeof(char));

cons = constante;

stackC\_push(&PilaConstantes, cons, \*direccion, 0);

}

if(strcmp(tipo, (char \*) "texto") == 0){

\*direccion = alloc\_virtual\_address((char \*)"texto",(char \*) "constante");

stack\_push(&PilaOperandos, direccion, 2);

stack\_push(&PiladeTipos, (char \*)"texto", 1);

char \*cons = (char \*)malloc(sizeof(char \*));

cons = constante;

stackC\_push(&PilaConstantes, cons, \*direccion, 1);

}

}

// FUNCTION: meterFondoFalso()

// RETURN: void

//

// Funcion encargada de meter el fondo falso a la pilaOperadores

//

void meterFondoFalso(){

stack\_push(&PilaOperadores, (char \*) "-1", 1);

}

// FUNCTION: sacaFondoFalso()

// RETURN: void

//

// Funcion encargada de sacar el fondo falso a la pilaOperadores

//

void sacaFondoFalso(){

if(strcmp(static\_cast<char\*>(PilaOperadores->ptr), (char \*) "-1") == 0){

stack\_pop(&PilaOperadores);

}

}

// FUNCTION: funcionOperaciones(char\* operador)

// RETURN: void

//

// Funcion que se encarga de realizar la verificacion de la pilaOperadores para las expresiones y genera cuadruplos de expresiones

//

void funcionOperaciones(char\* operador){

char\* tipo1;

char\* tipo2;

char\* tipoRetorno;

int operando1;

int operando2;

int \*direccionTemporal = (int \*)malloc(sizeof(int));

int operadorInt;

operadorInt = atoi(operador);

tipo1 = static\_cast<char\*>(PiladeTipos->ptr);

if (PiladeTipos->next != NULL) {

tipo2 = static\_cast<char\*>(PiladeTipos->next->ptr);

tipoRetorno = findSemanticCubeReturnType(semanticCube, tipo1, tipo2, operador);

if (PilaOperadores != NULL){

if (strcmp(operador, static\_cast<char\*>(PilaOperadores->ptr)) == 0){

if( strcmp(tipoRetorno,"error") != 0) {

stack\_pop(&PiladeTipos);

stack\_pop(&PiladeTipos);

operando2 = \*static\_cast<int\*>(stack\_pop(&PilaOperandos)->ptr);

operando1 = \*static\_cast<int\*>(stack\_pop(&PilaOperandos)->ptr);

stack\_pop(&PilaOperadores);

\*direccionTemporal = alloc\_virtual\_address(tipoRetorno, (char \*) "temporal");

addCuad(&cuadruplo, operadorInt, operando1, operando2, \*direccionTemporal);

contadorCuad++;

stack\_push(&PilaOperandos, direccionTemporal, 2);

stack\_push(&PiladeTipos, (void \*)findSemanticCubeReturnType(semanticCube, tipo1, tipo2, operador), 1);

} else{

printf("Error Semantico: Tipos incopatibles %s,%s\n",tipo1,tipo2);

}

}

}

}

}

// FUNCTION: funcionAsignacion()

// RETURN: void

//

// Funcion encargada de generar los cuadruplos para las asignaciones

//

void funcionAsignacion(){

int operando11;

int operando12;

char\* tipo1;

char\* tipo2;

if (strcmp((char \*) "120", static\_cast<char\*>(PilaOperadores->ptr)) == 0) {

stack\_pop(&PilaOperadores);

tipo1 = static\_cast<char\*>(PiladeTipos->ptr);

tipo2 = static\_cast<char\*>(PiladeTipos->next->ptr);

if (strcmp(tipo2,tipo1) == 0){

stack\_pop(&PiladeTipos);

stack\_pop(&PiladeTipos);

operando12 = \*static\_cast<int\*>(stack\_pop(&PilaOperandos)->ptr);

operando11 = \*static\_cast<int\*>(stack\_pop(&PilaOperandos)->ptr);

addCuad(&cuadruplo, 120, operando12, -1, operando11);

contadorCuad++;

} else{

printf("v: %i, %i\n",\*static\_cast<int\*>(stack\_pop(&PilaOperandos)->ptr),\*static\_cast<int\*>(stack\_pop(&PilaOperandos)->ptr));

printf("Error Semantico: Tipos incopatibles %s,%s\n",tipo1,tipo2);

}

}

}

// FUNCTION: funcionIfUno()

// RETURN: void

//

// Funcion encargada de generar los cuadruplos para la expresion de la condicion del IF y el goToF

//

void funcionIfUno(){

Cuadruplos \*temp;

char\* aux;

int resultado;

aux = static\_cast<char\*>(PiladeTipos->ptr);

if(strcmp(aux, (char \*) "booleano") == 0){

resultado = \*static\_cast<int\*>(stack\_pop(&PilaOperandos)->ptr);

temp = addCuad(&cuadruplo, 140, resultado, -1, -1);

contadorCuad++;

stack\_push(&PiladeSaltosCuad, temp, 2);

}

}

// FUNCTION: funcionIfDos()

// RETURN: void

//

// Funcion encargada de generar el cuadruplos del goTo y rellenar el cuadruolo de goToF

//

void funcionIfDos(){

Cuadruplos \*falso;

Cuadruplos \*temp;

temp = addCuad(&cuadruplo, 130, -1, -1, -1);

contadorCuad++;

falso = static\_cast<Cuadruplos\*>(stack\_pop(&PiladeSaltosCuad)->ptr);

falso->temporal = contadorCuad+1;

stack\_push(&PiladeSaltosCuad, temp, 2);

}

// FUNCTION: funcionIfTres()

// RETURN: void

//

// Funcion encargada de generar el cuadruplo del goTo y rellenar el FIN

//

void funcionIfTres(){

Cuadruplos \*temp;

temp = static\_cast<Cuadruplos\*>(stack\_pop(&PiladeSaltosCuad)->ptr);

temp->temporal = contadorCuad+1;

}

// FUNCTION: funcionWhileUno()

// RETURN: void

//

// Funcion encargada de meter el contador a la PiladeSaltos

//

void funcionWhileUno(){

int \*cont = (int \*)malloc(sizeof(int));

\*cont = contadorCuad;

stack\_push(&PiladeSaltos, cont, 2);

}

// FUNCTION: funcionWhileDos()

// RETURN: void

//

// Funcion encargada de generar los cuadruplos de goToF

//

void funcionWhileDos(){

Cuadruplos \*temp;

char\* aux;

int resultado;

aux = static\_cast<char\*>(PiladeTipos->ptr);

if(strcmp(aux, (char \*) "booleano") == 0){

resultado = \*static\_cast<int\*>(stack\_pop(&PilaOperandos)->ptr);

temp = addCuad(&cuadruplo, 140, resultado, -1, -1);

contadorCuad++;

stack\_push(&PiladeSaltosCuad, temp, 2);

}

}

// FUNCTION: funcionWhileTres()

// RETURN: void

//

// Funcion encargada de generar el cuadruplo de goTo y rellenar el cuadruplo del goToF

//

void funcionWhileTres(){

Cuadruplos \*temp;

int retorno;

temp = static\_cast<Cuadruplos\*>(stack\_pop(&PiladeSaltosCuad)->ptr);

retorno = \*static\_cast<int\*>(stack\_pop(&PiladeSaltos)->ptr);

addCuad(&cuadruplo, 130, -1, -1, retorno+1);

contadorCuad++;

temp->temporal = contadorCuad+1;

}

// FUNCTION: funcionParaUno()

// RETURN: void

//

//

//

void funcionParaUno() {

int \*cont = (int \*)malloc(sizeof(int));

\*cont = contadorCuad;

stack\_push(&PiladeSaltos, cont, 2);

}

// FUNCTION: funcionParaDos()

// RETURN: void

//

//

//

void funcionParaDos() {

Cuadruplos \*temp;

char\* aux;

int resultado;

aux = static\_cast<char\*>(PiladeTipos->ptr);

if(strcmp(aux, (char \*) "booleano") == 0){

resultado = \*static\_cast<int\*>(stack\_pop(&PilaOperandos)->ptr);

temp = addCuad(&cuadruplo, 140, resultado, -1, -1);

contadorCuad++;

stack\_push(&PiladeSaltosCuad, temp, 2);

temp = addCuad(&cuadruplo, 130, -1, -1, -1);

contadorCuad++;

stack\_push(&PiladeSaltosCuad, temp, 2);

int \*cont = (int \*)malloc(sizeof(int));

\*cont = contadorCuad;

stack\_push(&PiladeSaltos, cont, 2);

}

}

// FUNCTION: funcionParaTres()

// RETURN: void

//

//

//

void funcionParaTres() {

Cuadruplos \*temp;

Cuadruplos \*temp2;

temp = addCuad(&cuadruplo, 130, -1, -1, -1);

contadorCuad++;

temp2 = static\_cast<Cuadruplos\*>(stack\_pop(&PiladeSaltosCuad)->ptr);

temp2->temporal = contadorCuad+1;

stack\_push(&PiladeSaltosCuad, temp, 2);

}

// FUNCTION: funcionParaCuatro()

// RETURN: void

//

//

//

void funcionParaCuatro() {

int retorno;

Cuadruplos \*temp;

retorno = \*static\_cast<int\*>(stack\_pop(&PiladeSaltos)->ptr);

addCuad(&cuadruplo, 130, -1, -1, retorno+1);

contadorCuad++;

temp = static\_cast<Cuadruplos\*>(stack\_pop(&PiladeSaltosCuad)->ptr);

retorno = \*static\_cast<int\*>(stack\_pop(&PiladeSaltos)->ptr);

temp->temporal = retorno+1;

temp = static\_cast<Cuadruplos\*>(stack\_pop(&PiladeSaltosCuad)->ptr);

temp->temporal = contadorCuad+1;

}

// FUNCTION: funcionLectura()

// RETURN: void

//

// Funcion que se encarga de generar el cuadruplo para la funcion de Lectura

//

void funcionLectura() {

addCuad(&cuadruplo, 200, -1, -1, \*static\_cast<int\*>(stack\_pop(&PilaOperandos)->ptr));

contadorCuad++;

}

// FUNCTION: funcionEscritura()

// RETURN: void

//

// Funcion que se encarga de generar el cuadruplo para la funcion de Escritura

//

void funcionEscritura() {

addCuad(&cuadruplo, 190, -1, -1, \*static\_cast<int\*>(stack\_pop(&PilaOperandos)->ptr));

contadorCuad++;

}

// FUNCTION: funcionEscrituraFin()

// RETURN: void

//

// Funcion que se encarga de generar el cuadruplo para la escribir un println

//

void funcionEscrituraFin() {

addCuad(&cuadruplo, 210, -1, -1, -1);

contadorCuad++;

}

void funcReturnUno() {

addCuad(&cuadruplo, 240, \*static\_cast<int\*>(PilaOperandos->ptr), -1, -1);

contadorCuad++;

}

void funcionFuncionesUno(char\* funcion) {

Node \*tNode = findProc(dirProcsInit, funcion);

if(tNode != NULL){

tNode->virtual\_address = alloc\_virtual\_address(tNode->returnType,(char \*)"global");

}

}

// FUNCTION: agregaParam(char\* proc\_name, char\* tipoParam)

// RETURN: void

//

// Funcion que se encarga de agregar un nuevo parametro en un procedimiento

//

void agregaParam(char\* proc\_name, char\* tipoParam){

Node \*tNode = findProc(dirProcsInit, proc\_name);

addParameter(&tNode->apunta, tipoParam);

tNode->numeroCuadruplo = contadorCuad;

}

// FUNCTION: eliminaTabla(char\* funcion)

// RETURN: void

//

// Funcion que se encarga de eliminar la tabla de variables al momento de terminar la declaracion de la funcion y genera el cuadruplo de Retorno

//

void eliminaTabla(char\* funcion){

Node \*tNode = findProc(dirProcsInit, funcion);

if(tNode != NULL){

addCuad(&cuadruplo, 180, -1, -1, -1);

contadorCuad++;

}

}

// FUNCTION: generaEra(char\* proc\_name)

// RETURN: void

//

// Funcion que se encarga de generar el cuadruplo de Era al momento de mandar llamar a una funcion

//

void generaEra(char\* proc\_name){

Node \*tNode = findProc(dirProcsInit, proc\_name);

int cont;

if(tNode != NULL){

cont = findProcPos(dirProcsInit, proc\_name);

addCuad(&cuadruplo, 170, cont, -1, -1);

contadorCuad++;

}

}

// FUNCTION: generaParametro(char\* proc\_name)

// RETURN: void

//

// Funcion que se encarga de generar el cuadruplo de los parametros mandados

//

void generaParametros(char\* proc\_name){

int argumento;

char\* tipoArg;

Node \*tNode = findProc(dirProcsInit, proc\_name);

for(int i = 1; i <= contadorK; i++) {

argumento = \*static\_cast<int\*>(stack\_pop(&PilaOperandos)->ptr);

tipoArg = static\_cast<char\*>(stack\_pop(&PiladeTipos)->ptr);

// No checa si queda con el tipo que manda

//if(strcmp(tipoArg, tNode->apunta->tipo) == 0){

addCuad(&cuadruplo, 230, argumento, -1, i);

contadorCuad++;

//}

}

stack\_push(&PilaOperandos, &(tNode->virtual\_address), 2);

stack\_push(&PiladeTipos, tNode->returnType, 0);

}

// FUNCTION: generaCuadPrincipal()

// RETURN: void

//

// Genera el cuadruplo de salto al main al inicio de la lista de cuadruplos

// y la guarda en el stack

//

void generaCuadPrincipal() {

Cuadruplos \*temp;

temp = addCuad(&cuadruplo, 130, -1, -1, -1);

contadorCuad++;

stack\_push(&PiladeSaltosCuad, temp, 2);

}

void addSaltoAlPrincipal() {

Cuadruplos \*temp;

temp = static\_cast<Cuadruplos\*>(stack\_pop(&PiladeSaltosCuad)->ptr);

temp->temporal = contadorCuad+1;

}

void meterReturnATemp() {

int \*temp = (int \*)malloc(sizeof(int));;

\*temp = alloc\_virtual\_address(static\_cast<char\*>(PiladeTipos->ptr),(char\*)"temporal");

addCuad(&cuadruplo,120,\*static\_cast<int\*>(stack\_pop(&PilaOperandos)->ptr),-1,\*temp);

contadorCuad++;

stack\_push(&PilaOperandos, temp, 2);

}

%}

%union {

char \*str;

}

%token PROGRAMA VAR NUMERO DECIMAL TEXTO CARACTER BOOLEANO TRUE FALSE IMPRIME LEE SI SINO Y O REALIZA FIN MIENTRAS PARA FUNC IGUAL MAYOR MENOR DIFERENTE ID CTENUM CTEDEC CTEX CCAR COMA PCOMA ABREPA CIERRAPA ABRECORCH CIERRACORCH SIGUAL MAS MENOS POR ENTRE RET

%start program

%%

program: { generaCuadPrincipal(); } program2 PROGRAMA {variable\_type = (char \*)"NP"; tipo\_func = (char \*)"global"; } ID { variable\_name = yylval.str; last\_func = variable\_name; stack\_push(&PilaEjecucion, variable\_name, 1);} PCOMA { addProc(&dirProcsInit,variable\_name,variable\_type); addSaltoAlPrincipal();} program1 bloque {addCuad(&cuadruplo, 220, -1, -1, -1); contadorCuad++;};

program1: vars program1;

program1: ;

program2: func program2;

program2: ;

vars: VAR tipo vars1 ID { variable\_name = yylval.str; } PCOMA {addLocalVariableToProc(&dirProcsInit,last\_func,variable\_name,variable\_type,alloc\_virtual\_address(variable\_type, tipo\_func)); } ;

vars1: ABRECORCH CTENUM CIERRACORCH;

vars1: ;

tipo: NUMERO { variable\_type = (char \*)"numero"; } | DECIMAL { variable\_type = (char \*)"decimal"; } | CARACTER { variable\_type = (char \*)"caracter"; } | TEXTO { variable\_type = (char \*)"texto"; } | BOOLEANO { variable\_type = (char \*)"booleano"; };

bloque: REALIZA bloque1 FIN;

bloque1: estatuto bloque1;

bloque1: ;

estatuto: asignacion | condicion | escritura | lectura | mientras | para | llamada PCOMA;

asignacion: ID {meterAPilaOperandos(dirProcsInit, yylval.str, last\_func); } SIGUAL {stack\_push(&PilaOperadores, (char \*) "120", 1);} expresion PCOMA {funcionAsignacion();};

condicion: SI ABREPA expresion CIERRAPA {funcionIfUno();} REALIZA bloque1 condicion1 FIN {funcionIfTres();};

condicion1: SINO {funcionIfDos();} bloque1;

condicion1: ;

escritura: IMPRIME ABREPA escritura1 CIERRAPA { funcionEscrituraFin(); } PCOMA;

escritura1: expresion { funcionEscritura(); } escritura2;

escritura2: COMA escritura1;

escritura2: ;

lectura: LEE ABREPA variable { funcionLectura(); } CIERRAPA PCOMA;

mientras: MIENTRAS {funcionWhileUno();} ABREPA expresion CIERRAPA {funcionWhileDos();} bloque {funcionWhileTres();};

para: PARA ABREPA asignacion { funcionParaUno(); } expresion PCOMA { funcionParaDos(); } asignacion { funcionParaTres(); } CIERRAPA bloque { funcionParaCuatro(); } ;

func: FUNC {tipo\_func = (char \*)"local";} tipo ID {last\_func = yylval.str; stack\_push(&PilaEjecucion, last\_func, 1); addProc(&dirProcsInit, last\_func, variable\_type);funcionFuncionesUno(last\_func); } ABREPA func1 CIERRAPA varsfunc REALIZA bloque1 funcret FIN {eliminaTabla(last\_func);};

func1: tipo {agregaParam(static\_cast<char\*>(PilaEjecucion->ptr), variable\_type);} ID {variable\_name = yylval.str; addLocalVariableToProc(&dirProcsInit, static\_cast<char\*>(PilaEjecucion->ptr), variable\_name, variable\_type, alloc\_virtual\_address(variable\_type, (char \*)"local"));} func2;

func1: ;

func2: COMA func1;

func2: ;

varsfunc: vars varsfunc;

varsfunc: ;

funcret: RET ABREPA exp { funcReturnUno(); } CIERRAPA PCOMA;

funcret: ;

llamada: ID { meterFondoFalso();last\_func = yylval.str; contadorK = 0; } ABREPA llamada2 CIERRAPA { generaEra(last\_func); generaParametros(last\_func); addCuad(&cuadruplo, 160, findProcPos(dirProcsInit, last\_func), -1, -1); contadorCuad++; meterReturnATemp(); sacaFondoFalso();};

llamada2: exp { contadorK++; } llamada3;

llamada2: ;

llamada3: COMA llamada2;

llamada3: ;

expresion: cond {funcionOperaciones((char\*)"80"); funcionOperaciones((char\*)"90"); funcionOperaciones((char\*)"100"); funcionOperaciones((char\*)"110");} expresion1 ;

expresion1: expresion2 cond {funcionOperaciones((char\*)"80"); funcionOperaciones((char\*)"90"); funcionOperaciones((char\*)"100"); funcionOperaciones((char\*)"110");};

expresion1: ;

expresion2: {funcionOperaciones((char\*)"80");} MAYOR {stack\_push(&PilaOperadores, (char \*) "80", 1);}

| {funcionOperaciones((char\*)"90");} MENOR {stack\_push(&PilaOperadores, (char \*) "90", 1);}

| {funcionOperaciones((char\*)"100");} DIFERENTE {stack\_push(&PilaOperadores, (char \*) "100", 1);}

| {funcionOperaciones((char\*)"110");} IGUAL {stack\_push(&PilaOperadores, (char \*) "110", 1);};

cond: exp {funcionOperaciones((char\*)"60"); funcionOperaciones((char\*)"70");} cond1;

cond1: {funcionOperaciones((char\*)"60");} Y {stack\_push(&PilaOperadores, (char \*) "60", 1);} cond;

cond1: {funcionOperaciones((char\*)"70");} O {stack\_push(&PilaOperadores, (char \*) "70", 1);} cond;

cond1: ;

exp: termino exp1 {funcionOperaciones((char\*)"10"); funcionOperaciones((char\*)"20");};

exp1: {funcionOperaciones((char\*)"10");} MAS {stack\_push(&PilaOperadores, (char \*) "10", 1);} exp;

exp1: {funcionOperaciones((char \*)"20");} MENOS {stack\_push(&PilaOperadores, (char \*) "20", 1);} exp;

exp1: ;

termino: factor {funcionOperaciones((char\*)"30"); funcionOperaciones((char\*)"40");} termino1;

termino1: {funcionOperaciones((char\*)"30");} POR {stack\_push(&PilaOperadores, (char \*) "30", 1);} termino;

termino1: {funcionOperaciones((char\*)"40");} ENTRE {stack\_push(&PilaOperadores, (char \*) "40", 1);} termino;

termino1: ;

factor: ABREPA {meterFondoFalso();} expresion CIERRAPA {sacaFondoFalso();};

factor: varcte;

factor: llamada;

varcte: CTENUM {meterConstanteAPilaOperandos((char \*)"numero", yylval.str);} | CTEDEC {meterConstanteAPilaOperandos((char \*)"decimal", yylval.str);} | CTEX {meterConstanteAPilaOperandos((char \*)"texto", yylval.str);} | CCAR {meterConstanteAPilaOperandos((char \*)"caracter", yylval.str);} | TRUE | FALSE;

varcte: variable ;

variable: ID {meterAPilaOperandos(dirProcsInit, yylval.str, last\_func);} variable1;

variable1: ABRECORCH expresion CIERRACORCH;

variable1: ;

%%

void yyerror(const char \*s) /\* Llamada por yyparse ante un error \*/

{

printf ("Error on line #%i: %s\n", yylineno, s);

exit(-1);

}

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

if (argc == 2) {

FILE \*myfile = fopen(argv[1], "r");

if (!myfile) {

printf("Cant open file %s!\n", argv[1]);

return -1;

}

yyin = myfile;

initSemanticCube(&semanticCube);

do {

yyparse();

} while (!feof(yyin));

//printf("#\n");

//debugList(dirProcsInit);

//printf("#\n");

//imprimeListaConstantes(PilaConstantes);

//printf("#\n");

//imprimeCuad(cuadruplo);

strcat(argv[1],(char \*)".obj");

/\* WRITE OBJ FILE \*/

FILE \*file;

file = fopen(argv[1],"w+"); /\* apend file (add text to a file or create a file if it does not exist.\*/

imprimeConstantesToFile(&file, PilaConstantes);

fprintf(file,"#\n");

imprimeCuadruploToFile(&file, cuadruplo);

fprintf(file,"#\n");

imprimeDirProcsToFile(&file, dirProcsInit);

fclose(file);

deallocSemanticCube(&semanticCube);

deallocProcDir(&dirProcsInit);

} else {

printf("Invalid parameter count: %s <filename>\n", argv[0]);

}

}

### Estructura de Cuadruplos

Archivo: cuadruplos.c

/\*

Silvia Fosado Garcia 1032432

Victor Nava 617610

Estructuras de datos utilizadas en el compilador

\*/

#include <stdio.h>

#include <stdlib.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Cuadruplos \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

typedef struct \_cuadruplos {

int operador;

int dirOperando1;

int dirOperando2;

int temporal;

struct \_cuadruplos \*next;

} Cuadruplos;

// FUNCTION: addCuad(lista, operador, dirOperando1, dirOperando2, temporal);

// RETURN: void

//

// Inserta un Cuadruplo en una lista

//

Cuadruplos\* addCuad(Cuadruplos \*\*list, int operador, int dirOp1, int dirOp2, int temporal) {

Cuadruplos \*cuad;

cuad = \*list;

Cuadruplos \*cuad\_tmp = (struct \_cuadruplos \*)malloc(sizeof(struct \_cuadruplos));

cuad\_tmp->operador = operador;

cuad\_tmp->dirOperando1 = dirOp1;

cuad\_tmp->dirOperando2 = dirOp2;

cuad\_tmp->temporal = temporal;

cuad\_tmp->next = NULL;

if( cuad == NULL ){

\*list = cuad\_tmp;

}else{

while(cuad->next != NULL) {

cuad = cuad->next;

}

cuad->next = cuad\_tmp;

}

return cuad\_tmp;

}

// FUNCTION: imprimeCuad(Cuadruplos \*cuadruplo)

// RETURN: void

//

// Imprime los cuadruplos en el formato requerido

//

void imprimeCuad(Cuadruplos \*cuadruplo) {

Cuadruplos \*cuad;

cuad = cuadruplo;

int contador = 1;

while(cuad != NULL) {

printf("%d,%d,%d,%d\n", cuad->operador, cuad->dirOperando1, cuad->dirOperando2, cuad->temporal);

contador++;

cuad = cuad->next;

}

}

// FUNCTION: buscaCuad(Cuadruplos \*cuadruplo, contador)

// RETURN: Cuadruplos

//

// Funcion que busca un cuadruplo dado su numero de cuadruplo y regresa ese mismo cuadruplo encontrado para ser modificado

//

Cuadruplos\* buscarCuad(Cuadruplos \*cuadruplo, int contador){

Cuadruplos \*temp = cuadruplo;

for(int cont = 0; cont < contador && temp != NULL; cont++){

temp = temp->next;

}

return temp;

}

// FUNCTION: imprimeCuadToFile(FILE \*\*file, Cuadruplos \*cuadruplo)

// RETURN: void

//

// Imprime los cuadruplos en el formato requerido en un archivo especifico mandado como parametro

//

void imprimeCuadruploToFile(FILE \*\*file, Cuadruplos \*cuadruplo) {

Cuadruplos \*cuad;

cuad = cuadruplo;

while(cuad != NULL) {

fprintf(\*file, "%d,%d,%d,%d\n", cuad->operador, cuad->dirOperando1, cuad->dirOperando2, cuad->temporal);

cuad = cuad->next;

}

}

### Estructuras de Datos Generales, Dir de Procedimientos y Cubo Semantico

Archivo: estructura.c

/\*

Silvia Fosado Garcia 1032432

Victor Nava 617610

Estructuras de datos utilizadas en el compilador

\*/

#include <stdio.h>

#include <stdlib.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Tabla Parametros \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

typedef struct Parametros {

int numero;

char\* tipo;

struct Parametros \*next;

} Parametros;

// FUNCTION: addParameter(parametro, tipoParametro)

// RETURN: void

//

// Inserta un nodo nuevo de un parametro a la lista que contiene todos los parametros de la funcion

//

void addParameter(Parametros \*\*parametro, char\* tipo){

Parametros \*param;

param = \*parametro;

Parametros \*param\_temp = (struct Parametros \*)malloc(sizeof(struct Parametros));

param\_temp->numero = param\_temp->numero++;

param\_temp->tipo = tipo;

param\_temp->next = NULL;

if (param == NULL) {

\*parametro = param\_temp;

} else {

while(param->next != NULL){

param = param->next;

}

param->next = param\_temp;

}

}

// FUNCTION: cuentaParametros(parametro)

// RETURN: int

//

// Regresa la cantidad de parametros que hay en la lista de parametros

//

int cuentaParametros(Parametros \*head){

Parametros \*param = head;

int cont = 0;

while (param != NULL) {

cont++;

param = param -> next;

}

return cont;

}

/\*void imprime(Parametros \*head){

Parametros \*param = head;

while (param != NULL) {

printf("%s \n", param->tipo);

param = param -> next;

}

}\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Cubo Semantico \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

typedef struct \_fopernode {

char\* data;

\_fopernode \*contents;

\_fopernode \*next;

} SemanticCubeNode;

// FUNCTION: addOperNode(lista, operador)

// RETURN: void

//

// Inserta un nudo nuevo de un operador en una lista

//

void addOperNode(SemanticCubeNode \*\*list, char\* operador) {

SemanticCubeNode \*sNode;

sNode = (struct \_fopernode \*)malloc(sizeof(struct \_fopernode));

sNode->data = operador;

sNode->contents = NULL;

sNode->next = \*list;

\*list = sNode;

}

// FUNCTION: addSecondOperNode(lista, operador1, operador2)

// RETURN: void

//

// Inserta el segundo operador a la lista de un primer operador

//

void addSecondOperNode(SemanticCubeNode \*\*list, char\* op1, char\* op2) {

SemanticCubeNode \*sNode = \*list;

while (sNode != NULL) {

if (sNode->data == op1) {

SemanticCubeNode \*stNode = (struct \_fopernode \*)malloc(sizeof(struct \_fopernode));

stNode->data = op2;

stNode->contents = NULL;

stNode->next = sNode->contents;

sNode->contents = stNode;

break;

}

sNode = sNode->next;

}

}

// FUNCTION: addOperationNode(list, op1, op2, operation, returnType)

// Return: void

//

// Inserta la operacion en conjunto con su respuesta a la lista de nodos

//

void addOperationNode(SemanticCubeNode \*\*list, char\* op1, char \*op2, char\* operation, char\* returnType) {

SemanticCubeNode \*sNode = \*list;

SemanticCubeNode \*stNode;

while (sNode != NULL) {

if (sNode->data == op1) {

stNode = sNode->contents;

while (stNode != NULL) {

if (stNode->data == op2) {

SemanticCubeNode \*otNode = (struct \_fopernode \*)malloc(sizeof(struct \_fopernode));

SemanticCubeNode \*rtNode = (struct \_fopernode \*)malloc(sizeof(struct \_fopernode));

// Nodo con info del return type

rtNode->data = returnType;

rtNode->contents = NULL;

rtNode->next = NULL;

// Nodo con info de la operacion

otNode->data = operation;

otNode->contents = rtNode;

otNode->next = stNode->contents;

stNode->contents = otNode;

break;

}

stNode = stNode->next;

}

break;

}

sNode = sNode->next;

}

}

// FUNCTION: findSemanticCubeReturnType(lista, op1, op2, operation)

// RETURN: char\*

//

// Devuelve el tipo de retorno que generaria la operacion "operation" entre los operadores op1 y op2

// en caso de no encontrarlo devuelve (char \*)"error"

//

char\* findSemanticCubeReturnType(SemanticCubeNode \*list, char\* op1, char\* op2, char\* operation) {

SemanticCubeNode \*sNode = list;

while (sNode != NULL) {

if (strcmp(sNode->data,op1) == 0) {

sNode = sNode->contents;

while (sNode != NULL) {

if (strcmp(sNode->data,op2) == 0) {

sNode = sNode->contents;

while (sNode != NULL) {

if (sNode->data == operation) return sNode->contents->data;

sNode = sNode->next;

}

break;

}

sNode = sNode->next;

}

break;

}

sNode = sNode->next;

}

return (char \*)"error";

}

// FUNCTION: debugSemanticCube(lista)

// Return: void

//

// Imprime el cubo semantico para fines de debugging

//

void debugSemanticCube(SemanticCubeNode \*list) {

SemanticCubeNode \*sNode = list;

SemanticCubeNode \*stNode;

SemanticCubeNode \*otNode;

while (sNode != NULL) {

stNode = sNode->contents;

while (stNode != NULL) {

otNode = stNode->contents;

while (otNode != NULL) {

printf("OP1:%s, OP2:%s, OPER:%s, RETURN:%s\n", sNode->data, stNode->data, otNode->data, otNode->contents->data);

otNode = otNode->next;

}

stNode = stNode->next;

}

sNode = sNode->next;

printf("\n");

}

printf("FIN\n");

}

// FUNCTION: deallocSemanticCube(lista)

// Return: void

//

// Libera memoria del cubo semantico

//

void deallocSemanticCube(SemanticCubeNode \*\*list) {

SemanticCubeNode \*sNode = \*list;

SemanticCubeNode \*stNode;

SemanticCubeNode \*otNode;

SemanticCubeNode \*fNode;

while (sNode != NULL) {

stNode = sNode->contents;

while (stNode != NULL) {

otNode = stNode->contents;

while (otNode != NULL) {

free(otNode->contents);

fNode = otNode;

otNode = otNode->next;

free(fNode);

}

fNode = stNode;

stNode = stNode->next;

free(fNode);

}

fNode = sNode;

sNode = sNode->next;

free(fNode);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\* Init Semantic Cube \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void initSemanticCube(SemanticCubeNode \*\*list) {

addOperNode(list,(char \*)"numero");

addSecondOperNode(list, (char \*)"numero", (char \*)"numero");

addOperationNode(list, (char \*)"numero", (char \*)"numero", (char \*)"10", (char \*)"numero");

addOperationNode(list, (char \*)"numero", (char \*)"numero", (char \*)"20", (char \*)"numero");

addOperationNode(list, (char \*)"numero", (char \*)"numero", (char \*)"30", (char \*)"numero");

addOperationNode(list, (char \*)"numero", (char \*)"numero", (char \*)"40", (char \*)"decimal");

addOperationNode(list, (char \*)"numero", (char \*)"numero", (char \*)"50", (char \*)"numero");

addOperationNode(list, (char \*)"numero", (char \*)"numero", (char \*)"80", (char \*)"booleano");

addOperationNode(list, (char \*)"numero", (char \*)"numero", (char \*)"90", (char \*)"booleano");

addOperationNode(list, (char \*)"numero", (char \*)"numero", (char \*)"110", (char \*)"booleano");

addOperationNode(list, (char \*)"numero", (char \*)"numero", (char \*)"100", (char \*)"booleano");

addSecondOperNode(list, (char \*)"numero", (char \*)"decimal");

addOperationNode(list, (char \*)"numero", (char \*)"decimal", (char \*)"10", (char \*)"decimal");

addOperationNode(list, (char \*)"numero", (char \*)"decimal", (char \*)"20", (char \*)"decimal");

addOperationNode(list, (char \*)"numero", (char \*)"decimal", (char \*)"30", (char \*)"decimal");

addOperationNode(list, (char \*)"numero", (char \*)"decimal", (char \*)"40", (char \*)"decimal");

addOperationNode(list, (char \*)"numero", (char \*)"decimal", (char \*)"50", (char \*)"numero");

addOperationNode(list, (char \*)"numero", (char \*)"decimal", (char \*)"80", (char \*)"booleano");

addOperationNode(list, (char \*)"numero", (char \*)"decimal", (char \*)"90", (char \*)"booleano");

addOperNode(list,(char \*)"decimal");

addSecondOperNode(list, (char \*)"decimal", (char \*)"numero");

addOperationNode(list, (char \*)"decimal", (char \*)"numero", (char \*)"10", (char \*)"decimal");

addOperationNode(list, (char \*)"decimal", (char \*)"numero", (char \*)"20", (char \*)"decimal");

addOperationNode(list, (char \*)"decimal", (char \*)"numero", (char \*)"30", (char \*)"decimal");

addOperationNode(list, (char \*)"decimal", (char \*)"numero", (char \*)"40", (char \*)"decimal");

addOperationNode(list, (char \*)"decimal", (char \*)"numero", (char \*)"50", (char \*)"numero");

addOperationNode(list, (char \*)"decimal", (char \*)"numero", (char \*)"80", (char \*)"booleano");

addOperationNode(list, (char \*)"decimal", (char \*)"numero", (char \*)"90", (char \*)"booleano");

addSecondOperNode(list, (char \*)"decimal", (char \*)"decimal");

addOperationNode(list, (char \*)"decimal", (char \*)"decimal", (char \*)"10", (char \*)"decimal");

addOperationNode(list, (char \*)"decimal", (char \*)"decimal", (char \*)"20", (char \*)"decimal");

addOperationNode(list, (char \*)"decimal", (char \*)"decimal", (char \*)"30", (char \*)"decimal");

addOperationNode(list, (char \*)"decimal", (char \*)"decimal", (char \*)"40", (char \*)"decimal");

addOperationNode(list, (char \*)"decimal", (char \*)"decimal", (char \*)"50", (char \*)"numero");

addOperationNode(list, (char \*)"decimal", (char \*)"decimal", (char \*)"80", (char \*)"booleano");

addOperationNode(list, (char \*)"decimal", (char \*)"decimal", (char \*)"90", (char \*)"booleano");

addOperationNode(list, (char \*)"decimal", (char \*)"decimal", (char \*)"110", (char \*)"booleano");

addOperationNode(list, (char \*)"decimal", (char \*)"decimal", (char \*)"100", (char \*)"booleano");

addOperNode(list,(char \*)"texto");

addSecondOperNode(list, (char \*)"texto", (char \*)"texto");

addOperationNode(list, (char \*)"texto", (char \*)"texto", (char \*)"110", (char \*)"booleano");

addOperationNode(list, (char \*)"texto", (char \*)"texto", (char \*)"100", (char \*)"booleano");

addOperNode(list,(char \*)"caracter");

addSecondOperNode(list, (char \*)"caracter", (char \*)"caracter");

addOperationNode(list, (char \*)"caracter", (char \*)"caracter", (char \*)"110", (char \*)"booleano");

addOperationNode(list, (char \*)"caracter", (char \*)"caracter", (char \*)"100", (char \*)"booleano");

addOperNode(list,(char \*)"booleano");

addSecondOperNode(list, (char \*)"booleano", (char \*)"booleano");

addOperationNode(list, (char \*)"booleano", (char \*)"booleano", (char \*)"110", (char \*)"booleano");

addOperationNode(list, (char \*)"booleano", (char \*)"booleano", (char \*)"100", (char \*)"booleano");

addOperationNode(list, (char \*)"booleano", (char \*)"booleano", (char \*)"60", (char \*)"booleano");

addOperationNode(list, (char \*)"booleano", (char \*)"booleano", (char \*)"70", (char \*)"booleano");

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Dir Procs \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

typedef struct \_node {

char\* name;

char\* returnType;

int virtual\_address;

int numeroCuadruplo;

Parametros \*apunta;

\_node \*local\_variables;

\_node \*next;

} Node;

// FUNCTION: addProc(lista, name, returnType);

// RETURN: void

//

// Inserta un Proc en una lista junto con su nombre, tipo de retorno y

// variable local.

//

void addProc(Node \*\*list, char\* name, char \*returnType) {

Node \*tNode, \*temp;

temp = \*list;

tNode = (struct \_node \*)malloc(sizeof(struct \_node));

tNode->name = name;

tNode->returnType = returnType;

tNode->apunta = NULL;

tNode->local\_variables = NULL;

if (temp == NULL) {

\*list = tNode;

} else {

while (temp->next != NULL) temp = temp->next;

temp->next = tNode;

}

}

// FUNCTION: findProc(lista, name)

// RETURN: Node

//

// Busca un Proc en una lista y lo devuelve. Devuelve nulo en caso

// de no existir.

//

Node\* findProc(Node \*list, char\* name) {

Node \*tNode = list;

while (tNode != NULL) {

if (strcmp(tNode->name,name) == 0) return tNode;

tNode = tNode->next;

}

return NULL;

}

// FUNCTION: findProcPos(directorioProcedimientos, nombreProceso)

// RETURN: int

//

// Regresa la cantidad total de parametros que tiene una funcion

//

int findProcPos(Node \*list, char\* name) {

Node \*tNode = list;

int contador = 0;

while (tNode != NULL) {

if (strcmp(tNode->name,name) == 0) return contador;

tNode = tNode->next;

contador++; }

return -1;

}

// FUNCTION: findLocalVariableInProc(list, proc\_name, var\_name);

// RETURN: Node

//

// Busca una variable en un proc

//

Node\* findLocalVariableInProc(Node \*list, char\* proc\_name, char\* var\_name) {

// Busca en las variables del Proc

Node \*tNode = findProc(list, proc\_name);

tNode = tNode->local\_variables;

while (tNode != NULL) {

if (strcmp(tNode->name,var\_name) == 0) return tNode;

tNode = tNode->next;

}

return NULL;

}

// FUNCTION: addLocalVariableToProc(list, proc\_name, var\_name, returnType);

// RETURN: void

//

// Agrega una variable local a un Proc especifico en una lista

//

void addLocalVariableToProc(Node \*\*list, char\* proc\_name, char\* var\_name, char\* returnType, int virtual\_address) {

Node \*procNode = findProc(\*list, proc\_name);

Node \*tNode = (struct \_node \*)malloc(sizeof(struct \_node));

tNode->name = var\_name;

tNode->returnType = returnType;

tNode->local\_variables = NULL;

tNode->virtual\_address = virtual\_address;

tNode->next = procNode->local\_variables;

procNode->local\_variables = tNode;

}

// FUNCTION: dealloc(list)

// RETURN: void

//

// Libera la memoria de la lista automaticamente

//

void deallocProcDir(Node \*\*list) {

Node \*tNode = \*list;

Node \*sNode;

while (tNode != NULL) {

sNode = tNode;

tNode = tNode->next;

free(sNode);

}

\*list = NULL;

}

// FUNCTION: debugList(lista)

// RETURN: void

//

// Imprime la informacion disponible de la lista de procs. Esto

// simplemente para uso de debugging.

//

void debugList(Node \*list) {

Node \*tNode = list;

Node \*local\_variables;

int numParams = 0;

while (tNode != NULL) {

numParams = cuentaParametros(tNode->apunta);

printf("%s, %s, %d, %d\n", tNode->name, tNode->returnType, numParams, tNode->numeroCuadruplo);

/\*printf("%s\n", tNode->name);

imprime(tNode->apunta);\*/

tNode = tNode->next;

}

}

// FUNCTION: imprimeCuadToFile(FILE \*\*file, Cuadruplos \*cuadruplo)

// RETURN: void

//

// Imprime los cuadruplos en el formato requerido en un archivo especifico mandado como parametro

//

void imprimeDirProcsToFile(FILE \*\*file, Node \*list) {

Node \*tNode = list;

int numParams = 0;

while(tNode != NULL) {

numParams = cuentaParametros(tNode->apunta);

fprintf(\*file, "%s,%s,%d,%d,%d\n", tNode->name, tNode->returnType, numParams, tNode->numeroCuadruplo, tNode->virtual\_address);

tNode = tNode->next;

}

}

### Estructura de Stack

Archivo: stack.c

#include <stdlib.h>

#include <stdio.h>

typedef struct Stack {

void \*ptr;

int ptrType;

struct Stack \*next;

} Stack;

bool stack\_push(Stack \*\*head, void \*ptr, int ptrType) {

Stack \*temp;

temp = (Stack \*)malloc(sizeof(Stack));

temp->ptr = ptr;

temp->ptrType = ptrType;

temp->next = \*head;

\*head = temp;

return true;

}

Stack \*stack\_pop(Stack \*\*head) {

if( head != NULL ){

Stack \*temp;

temp = \*head;

\*head = temp->next;

return temp;

}else{

return NULL;

}

}

### Estructura de Stack para Constantes

Archivo: stackConstantes.c

#include <stdlib.h>

#include <stdio.h>

typedef struct StackConstantes {

void \*ptr;

int direccionMemoria;

int ptrType;

struct StackConstantes \*next;

} StackConstantes;

bool stackC\_push(StackConstantes \*\*head, void \*ptr, int direccionMemoria, int ptrType) {

StackConstantes \*temp;

temp = (StackConstantes \*)malloc(sizeof(StackConstantes));

temp->ptr = ptr;

temp->direccionMemoria = direccionMemoria;

temp->ptrType = ptrType;

temp->next = \*head;

\*head = temp;

return true;

}

StackConstantes \*stackC\_pop(StackConstantes \*\*head) {

if( head != NULL ){

StackConstantes \*temp;

temp = \*head;

\*head = temp->next;

return temp;

}else{

return NULL;

}

}

void imprimeListaConstantes(StackConstantes \*head){

StackConstantes \*temp;

temp = head;

while (temp != NULL) {

if(temp->ptrType == 0){

printf("%i,%c\n", temp->direccionMemoria,\*static\_cast<char\*>(temp->ptr));

}

if(temp->ptrType == 1){

printf("%i,%s\n", temp->direccionMemoria,static\_cast<char\*>(temp->ptr));

}

if(temp->ptrType == 2){

printf("%i,%i\n", temp->direccionMemoria,\*static\_cast<int\*>(temp->ptr));

}

if(temp->ptrType == 3) {

printf("%i,%f\n", temp->direccionMemoria,\*static\_cast<float\*>(temp->ptr));

}

temp = temp->next;

}

}

void imprimeConstantesToFile(FILE \*\*file, StackConstantes \*head) {

StackConstantes \*temp;

temp = head;

while (temp != NULL) {

if(temp->ptrType == 0){

fprintf(\*file, "%i,%c\n", temp->direccionMemoria,\*static\_cast<char\*>(temp->ptr));

}

if(temp->ptrType == 1){

fprintf(\*file, "%i,%s\n", temp->direccionMemoria,static\_cast<char\*>(temp->ptr));

}

if(temp->ptrType == 2){

fprintf(\*file, "%i,%i\n", temp->direccionMemoria,\*static\_cast<int\*>(temp->ptr));

}

if(temp->ptrType == 3) {

fprintf(\*file, "%i,%f\n", temp->direccionMemoria,\*static\_cast<float\*>(temp->ptr));

}

temp = temp->next;

## Máquina Virtual

#!/usr/bin/env ruby

#####################

# Clases #

#####################

class Cuadruplo

attr\_reader :opId, :oper1, :oper2, :result

attr\_writer :opId, :oper1, :oper2, :result

def initialize(cuadText)

cuadText = cuadText.split(',')

@opId = cuadText[0].to\_i

@oper1 = cuadText[1].to\_i

@oper2 = cuadText[2].to\_i

@result = cuadText[3].to\_i

end

end

class Procs

attr\_reader :name, :returnType, :params, :initCuad, :retAddress

def initialize(cuadText)

cuadText = cuadText.split(',');

@name = cuadText[0].to\_s

@returnType = cuadText[1].to\_s

@params = cuadText[2].to\_i

@initCuad = cuadText[3].to\_i

@retAddress = cuadText[4].to\_i

end

end

#####################################

# Inicializa Variables #

#####################################

@cuadruplos = Hash.new

@virtualMemory = Hash.new

@dirprocs = Array.new

@stackProcs = Array.new

@stackPointer = Array.new

@stackReturn = Array.new

@apuntador = 1; # El index de los cuadruplos inicia en el 1

#####################################

# Define Limites Memoria #

#####################################

VAR\_SPACE = 1000

GLOBAL\_CHAR = 0

GLOBAL\_TEXT = 1000

GLOBAL\_NUMBER = 2000

GLOBAL\_DECIMAL = 3000

GLOBAL\_BOOL = 4000

LOCAL\_CHAR = 5000

LOCAL\_TEXT = 6000

LOCAL\_NUMBER = 7000

LOCAL\_DECIMAL = 8000

LOCAL\_BOOL = 9000

TEMP\_CHAR = 10000

TEMP\_TEXT = 11000

TEMP\_NUMBER = 12000

TEMP\_DECIMAL = 13000

TEMP\_BOOL = 14000

CONST\_CHAR = 15000

CONST\_TEXT = 16000

CONST\_NUMBER = 17000

CONST\_DECIMAL = 18000

CONST\_BOOL = 19000

#########################

# Funciones #

#########################

# Carga archivo de los cuadruplos en el hash

def loadCuadruplos

part = 0

counter = 1

file = File.new(ARGV[0], "r")

while (line = file.gets)

if line.strip == "#"

part += 1

else

case part

when 0

temp = line.strip.split(",")

virtualAddress = temp[0].to\_i

@virtualMemory[virtualAddress] = temp[1].gsub(/\r\n?/,"").gsub(/"/, "");

# En caso de que haya comas en el string

for i in 2..temp.length-1

@virtualMemory[virtualAddress] += "," + temp[i].gsub(/\r\n?/,"").gsub(/"/, "");

end

# En caso de que sea numero o flotante

@virtualMemory[virtualAddress] = temp[1].to\_i if (virtualAddress >= GLOBAL\_NUMBER && virtualAddress <= GLOBAL\_NUMBER+VAR\_SPACE-1) || (virtualAddress >= LOCAL\_NUMBER && virtualAddress <= LOCAL\_NUMBER+VAR\_SPACE-1) || (virtualAddress >= TEMP\_NUMBER && virtualAddress <= TEMP\_NUMBER+VAR\_SPACE-1) || (virtualAddress >= CONST\_NUMBER && virtualAddress <= CONST\_NUMBER+VAR\_SPACE-1)

@virtualMemory[virtualAddress] = temp[1].to\_f if (virtualAddress >= GLOBAL\_DECIMAL && virtualAddress <= GLOBAL\_DECIMAL+VAR\_SPACE-1) || (virtualAddress >= LOCAL\_DECIMAL && virtualAddress <= LOCAL\_DECIMAL+VAR\_SPACE-1) || (virtualAddress >= TEMP\_DECIMAL && virtualAddress <= TEMP\_DECIMAL+VAR\_SPACE-1) || (virtualAddress >= CONST\_DECIMAL && virtualAddress <= CONST\_DECIMAL+VAR\_SPACE-1)

when 1

@cuadruplos[counter] = Cuadruplo.new(line.strip)

counter += 1

when 2

@dirprocs += [Procs.new(line.strip)]

end

end

end

file.close

end

#####################

# Main #

#####################

if !ARGV.empty? # Si pasa el parametro del archivo a cargar

loadCuadruplos # carga el archivo de OBJ

while @apuntador <= @cuadruplos.length do

#obten cuadruplo actual

cuadruplo = @cuadruplos[@apuntador]

# Realizar Operacion del Apuntador

case cuadruplo.opId

when 10 # Suma

@virtualMemory[cuadruplo.result] = @virtualMemory[cuadruplo.oper1] + @virtualMemory[cuadruplo.oper2]

when 20 # Resta

@virtualMemory[cuadruplo.result] = @virtualMemory[cuadruplo.oper1] - @virtualMemory[cuadruplo.oper2]

when 30 # Multiplicacion

@virtualMemory[cuadruplo.result] = @virtualMemory[cuadruplo.oper1] \* @virtualMemory[cuadruplo.oper2]

when 40 # Division

@virtualMemory[cuadruplo.result] = @virtualMemory[cuadruplo.oper1] / @virtualMemory[cuadruplo.oper2]

when 50 # Modulo

@virtualMemory[cuadruplo.result] = @virtualMemory[cuadruplo.oper1] % @virtualMemory[cuadruplo.oper2]

when 60 # Y

@virtualMemory[cuadruplo.result] = (@virtualMemory[cuadruplo.oper1] && @virtualMemory[cuadruplo.oper2] ? true : false)

when 70 # O

@virtualMemory[cuadruplo.result] = (@virtualMemory[cuadruplo.oper1] || @virtualMemory[cuadruplo.oper2] ? true : false)

when 80 # Mayor Que

@virtualMemory[cuadruplo.result] = (@virtualMemory[cuadruplo.oper1] > @virtualMemory[cuadruplo.oper2] ? true : false)

when 90 # Menor Que

@virtualMemory[cuadruplo.result] = (@virtualMemory[cuadruplo.oper1] < @virtualMemory[cuadruplo.oper2] ? true : false)

when 100 # Diferente Que

@virtualMemory[cuadruplo.result] = (@virtualMemory[cuadruplo.oper1] != @virtualMemory[cuadruplo.oper2] ? true : false)

when 110 # Igual Que

@virtualMemory[cuadruplo.result] = (@virtualMemory[cuadruplo.oper1] == @virtualMemory[cuadruplo.oper2] ? true : false)

when 120 # Igualar Variable

@virtualMemory[cuadruplo.result] = @virtualMemory[cuadruplo.oper1];

when 130 # Goto

@apuntador = cuadruplo.result-1

when 140 # Goto False

@apuntador = cuadruplo.result-1 if !@virtualMemory[cuadruplo.oper1]

when 150 # Goto True

@apuntador = cuadruplo.result-1 if @virtualMemory[cuadruplo.oper1]

when 160 # GoSub

@stackPointer.push(@apuntador)

@apuntador = @dirprocs[cuadruplo.oper1].initCuad

@stackReturn.push(@dirprocs[cuadruplo.oper1].retAddress)

when 170 # ERA

# Crear Buffer de memoria y salvar los datos al stack de procs

memory\_buffer = Hash.new

@virtualMemory.each { |key, value|

if (key >= 5000 && key < 15000)

memory\_buffer[key] = value

@virtualMemory.delete(key)

end

}

@stackProcs.push(memory\_buffer)

when 180 # RET From Function

# Devolver apuntador

@apuntador = @stackPointer.pop()

# Destruir Memoria local de funcion

@virtualMemory.each { |key, value| @virtualMemory.delete(key) if key >= 5000 && key < 15000 }

# Recupera memoria local del stack

@virtualMemory.merge!(@stackProcs.pop())

when 190 # Print

print @virtualMemory[cuadruplo.result].to\_s

STDOUT.flush

when 200 # Read

@virtualMemory[cuadruplo.result] = STDIN.gets.strip

if (cuadruplo.result >= GLOBAL\_NUMBER && cuadruplo.result <= GLOBAL\_NUMBER+VAR\_SPACE-1) || (cuadruplo.result >= LOCAL\_NUMBER && cuadruplo.result <= LOCAL\_NUMBER+VAR\_SPACE-1) || (cuadruplo.result >= TEMP\_NUMBER && cuadruplo.result <= TEMP\_NUMBER+VAR\_SPACE-1)

@virtualMemory[cuadruplo.result] = Integer(@virtualMemory[cuadruplo.result])

end

if (cuadruplo.result >= GLOBAL\_DECIMAL && cuadruplo.result <= GLOBAL\_DECIMAL+VAR\_SPACE-1) || (cuadruplo.result >= LOCAL\_DECIMAL && cuadruplo.result <= LOCAL\_DECIMAL+VAR\_SPACE-1) || (cuadruplo.result >= TEMP\_DECIMAL && cuadruplo.result <= TEMP\_DECIMAL+VAR\_SPACE-1)

@virtualMemory[cuadruplo.result] = Float(@virtualMemory[cuadruplo.result])

end

when 210 # Print New Line

puts ""

when 220 # Program Quit

@apuntador = @cuadruplos.length;

when 230 # Crea Param de una Funcion

funcmem = ((cuadruplo.oper1/VAR\_SPACE)%5)\*VAR\_SPACE+LOCAL\_CHAR

i = 0;

while @virtualMemory.has\_key?(funcmem+i)

i += 1

end

@virtualMemory[funcmem+i] = @virtualMemory[cuadruplo.oper1] if cuadruplo.oper1 < 5000 || cuadruplo.oper1 >= 15000

@virtualMemory[funcmem+i] = @stackProcs.last[cuadruplo.oper1] if cuadruplo.oper1 >= 5000 && cuadruplo.oper1 < 15000

when 240 # RETURN

@virtualMemory[@stackReturn.pop()] = @virtualMemory[cuadruplo.oper1]

else

puts "UNRecog: "+cuadruplo.opId.to\_s

end

@apuntador += 1; # Incrementa apuntador

end

#puts "\nMemoria Virtual Final:\n"+@virtualMemory.to\_s

else

puts "Debes poner el argumento del codigo objeto a ejecutar."

end

# Manual de Usuario

## Requisitos

* Contar con acceso a internet o un servidor web local
* Tener Ruby instalado en su computadora

## Generación de Código por Gráficos

1. El usuario puede acceder al desarrollador gráfico entrando a <http://sofi.vikonava.com/> donde se tiene la última versión, o bien abriendo el archivo que viene en su CD localizado en el folder *graphics/index.php* desde su servidor web local.
2. Diseñar su código en la interfaz gráfica.
3. Guardar su código dando click en el botón de *“Guardar”*.

## Compilación del Programa

1. El usuario debe ejecutar el archivo *Compilador/sofi* y mandar como parámetro el nombre del archivo de su código a compilar. (Ejemplo:compilador/sofi miprograma.sofi)

## Ejecución del Programa

1. El usuario debe ejecutar su código intermedio utilizando la máquina virtual de sofi. Para esto se ejecuta el archivo *VM/sofivm.rb* y mandar como parámetro el nombre del archivo objeto a compilar. (Ejemplo: VM/sofivm.rb miprograma.sofi.obj)

# Anexos