Instituto Tecnológico de Estudios Superiores de Monterrey

**Compiladores**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ing. Elda Guadalupe Quiroga

**Solstice**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Jorge Salazar Saltijeral

A00945665

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Verónica Alejandra González González

A01087523

8 de Mayo de 2013

Tabla de Contenido

[Tabla de Contenido i](#_Toc351369622)

[1 Descripción del Proyecto 1](#_Toc351369623)

[1.1 Visión 1](#_Toc351369624)

[1.2 Objetivos 1](#_Toc351369625)

[1.3 Alcance 1](#_Toc351369626)

[1.4 Análisis de Requerimientos 1](#_Toc351369627)

[1.5 Casos de Uso 1](#_Toc351369628)

[1.6 Casos de Prueba 1](#_Toc351369629)

[1.7 Proceso de Desarrollo 1](#_Toc351369630)

[2 Descripción del Lenguaje 1](#_Toc351369631)

[2.1 Características Principales 1](#_Toc351369632)

[2.2 Principales Errores 1](#_Toc351369633)

[3 Descripción del Compilador 2](#_Toc351369634)

[3.1 Desarrollo 2](#_Toc351369635)

[3.2 Análisis Léxico 2](#_Toc351369636)

[3.3 Análisis Sintáctico 2](#_Toc351369637)

[3.4 Generación de Código Intermedio y Análisis Semántico 2](#_Toc351369638)

[3.5 Administración de Memoria en Compilación 2](#_Toc351369639)

[4 Descripción de la Máquina Virtual 2](#_Toc351369640)

[4.1 Desarrollo 2](#_Toc351369641)

[4.2 Administración de Memoria en Ejecución 2](#_Toc351369642)

[5 Pruebas de Funcionamiento de Lenguaje 2](#_Toc351369643)

[6 Código 2](#_Toc351369644)

[7 Bibliografía 2](#_Toc351369645)

# Descripción del Proyecto

## Visión

Con este proyecto se espera desarrollar un nuevo lenguaje de programación enfocado al paradigma Orientado a Objetos en una forma simplificada.

## Objetivos

El proyecto tiene como objetivo construir un compilador con las características principales del paradigma Orientado a Objetos. Donde cada objeto tiene diferentes atributos y métodos que pueden utilizar a lo largo del programa.

## Alcance

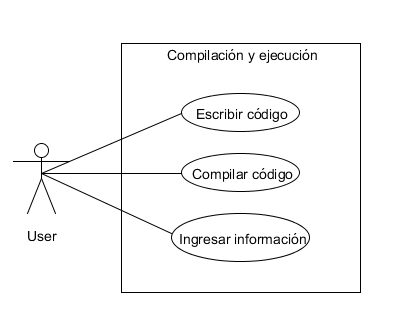
…

## Análisis de Requerimientos

* El usuario tendrá que escribir el código en un archivo con extensión .sol para que este pueda ser compilado
* El lenguaje deberá aceptar variables de tipo entera, flotante, boolean y string
* El lenguaje deberá aceptar ciclos (while)
* El lenguaje deberá aceptar condicionales (if-elseif-else)
* El lenguaje deberá aceptar arreglos como estructura de datos
* El lenguaje deberá aceptar funciones de escritura y lectura
* El lenguaje deberá aceptar herencia simple, donde las clases hijas pueden utilizar los atributos no privados de la clase padre
* El lenguaje deberá aceptar la declaración de variables globales como locales
* El lenguaje deberá aceptar la declaración de métodos tanto con valor de retorno como de tipo void

## Casos de Uso

Debido a que nuestro compilador no tiene una parte gráfica los únicos casos de uso que tiene nuestro compilador son los que se muestran a continuación:



|  |  |
| --- | --- |
| **ID:** CU1 | **Caso de uso:** Escribir código |
| **Actor:** Usuario | |
| **Propósito:** El usuario escribirá el código que desea compilar y ejecutar | |
| **Precondición:** Conocer el léxico así como las reglas semánticas y sintácticas del lenguaje Solstice | |
| **Flujo de eventos** | |
| **Acción de actor**  Escribir código de acuerdo al léxico, sintaxis y semántica del lenguaje Solstice guardando el archivo con extensión correcta (.sol) | **Reacción de sistema**  No marcar error y pasar a la compilación del código |

|  |  |
| --- | --- |
| **ID:** CU2 | **Caso de uso:** Compilar código |
| **Actor:** Usuario | |
| **Propósito:** El usuario podrá compilar el código | |
| **Precondición:** Haber creado código en lenguaje Solstice | |
| **Flujo de eventos** | |
| **Acción de actor**  Cargar el archivo y correr el compilador | **Reacción de sistema**  El sistema compilará el archivo mostrando errores en caso de que encuentre alguno, de otra forma terminará de compilarlo mostrando los resultados en caso de que hubiera lectura o escritura dentro del programa cargado |

|  |  |
| --- | --- |
| **ID:** CU3 | **Caso de uso:** Ingresar información |
| **Actor:** Usuario | |
| **Propósito:** El usuario podrá ingresar información correctamente a través de la consola | |
| **Precondición:** Haber compilado el código y que haya estatutos de lectura en el código | |
| **Flujo de eventos** | |
| **Acción de actor**  Ingresar la información que se quiere leer | **Reacción de sistema**  Guardar y validar que la información que se haya ingresado sea la que se está esperando y en caso contrario marcar error |

## Casos de Prueba

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Elemento** | **Entrada** | **Salida** |
| 1 | Declaración de variables locales | public void $funcion() {  int \_i = 0;  string \_s = “Hola”;  bool \_a = true;  double \_r = 1.0;  int arr[5];  } | * Validación semántica * Creación de cuádruplos * Registro en memoria |
| 2 | Declaración de variables globales | protected double \_peso;  public string \_nombre;  private int \_edad;  public bool \_carrera; | * Validación semántica * Creación de cuádruplos * Registro en memoria |
| 3 | Funciones que regresa valor entero | public int #getEdad() {  return \_edad;  } | * Validación semántica * Creación de cuádruplos * Asociar parámetros en caso de que haya * Registro en memoria |
| 4 | Funciones que regresa valor double | private double #getPeso() {  return \_peso;  } | * Validación semántica * Creación de cuádruplos * Asociar parámetros en caso de que haya * Registro en memoria |
| 5 | Funciones que regresa valor boolean | public bool #getCarrera() {  return \_carrera;  } | * Validación semántica * Creación de cuádruplos * Asociar parámetros en caso de que haya * Registro en memoria |
| 6 | Funciones que regresa valor string | public string #getNombre() {  return \_nombre;  } | * Validación semántica * Creación de cuádruplos * Asociar parámetros en caso de que haya * Registro en memoria |
| 7 | Funciones void | public void #setNombre(string \_n) {  \_nombre = \_n;  } | * Validación semántica * Creación de cuádruplos * Asociar parámetros en caso de que haya * Registro en memoria |
| 8 | Funciones recursivas | public int #factR(int \_x){  int \_y = 0;  ?(\_x <> 1){  \_y = \_x + 1;  \_x = \_x \* #factR(\_y);  }  return \_x;  } | * Validación semántica * Creación de cuádruplos * Asociar parámetros en caso de que haya * Registro en memoria |
| 9 | Asignación a constante | \_a = 0;  \_b = -1.2;  \_c = false;  \_d = “prueba”; | * Validación semántica * Creación de cuádruplos * Asociación de memoria |
| 10 | Asignación a función | \_a = \_objetoA.#fibonnacci(2); | * Validación semántica * Creación de cuádruplos * Asociación de memoria |
| 11 | Asignación a expresión | \_a = (1 + 5 / 6 \* 9 % 10) + 8; | * Validación semántica * Creación de cuádruplos * Asociación de memoria |
| 12 | Asignación a arreglo | \_arr[0] = arrB[3];  \_arr = \_arrB; | * Validación semántica * Creación de cuádruplos * Asociación de memoria |
| 13 | Asignación de objetos | \_obejtoA = \_objetoB; | * Validación semántica * Creación de cuádruplos * Asociación de memoria |
| 14 | Ciclo (while) | @(\_x > 0){  \_r = \_r \* \_x;  \_x--;  } | * Validación semántica * Creación de cuádruplos |
| 15 | Condicional (if) | ?(\_x <> 1){  \_y = \_x + 1;  \_x = \_x \* #factR(\_y);  } | * Validación semántica * Creación de cuádruplos |
| 16 | Suma | int \_a = 1 + 2;  double \_b = 1.0 + 2.3; | * Validación semántica * Creación de cuádruplos |
| 17 | Resta | int \_a = 1 - 2;  double \_b = 1.0 - 2.3; | * Validación semántica * Creación de cuádruplos |
| 18 | Multiplicación | int \_a = 1 \* 2;  double \_b = 1.0 \* 2.3; | * Validación semántica * Creación de cuádruplos |
| 19 | División | int \_a = 1 / 2;  double \_b = 1.0 / 2.3; | * Validación semántica * Creación de cuádruplos |
| 20 | Módulo | int \_a = 1 % 2;  int \_b = 1.0 % 2.3; | * Validación semántica * Creación de cuádruplos |
| 21 | Incremento | int \_a = 0;  \_a++; | * Validación semántica * Creación de cuádruplos |
| 23 | Decremento | int \_a = 0;  \_a--; | * Validación semántica * Creación de cuádruplos |
| 24 | Relaciones | bool \_a = 1 < 2;  bool \_b = 3.3 >= 2.0;  bool \_c = “prueba” <> “hola”;  bool \_d = true == false; | * Validación semántica * Creación de cuádruplos |
| 25 | Lógicos | bool \_a = true | false;  bool \_b = true & false; | * Validación semántica * Creación de cuádruplos |
| 26 | Escritura y concatenación | >> “hola mundo”;  >> “prueba” ^ 2.9;  >> “prueba” ^ true;  >> “prueba” ^ 2; | * Validación semántica * Creación de cuádruplos |
| 27 | Lectura | << (“Dime un número”, int); | * Validación semántica * Creación de cuádruplos |
| 29 | Declaración de clases | class Prog {  public int \_prueba;  $Prog(){  }  public int #factI(int \_x){  int \_r = 1;  @(\_x > 0){  \_r = \_r \* \_x;  \_x--;  }  return \_r;  } | * Validación semántica * Creación de cuádruplos * Registro de memoria |
| 30 | Llamada a funciones | \_num2 = \_p.#factR(\_num1);  \_objeto.#setEdad(\_edad); | * Validación semántica * Creación de cuádruplos * Asociar parámetros en caso de que haya * Registro en memoria |

## Proceso de Desarrollo

Describir PROCESO…

A continuación se presentan cada una de las bitácoras que se generaron durante cada junta:

**Bitácora 1**

**Fecha:** Febrero 16 de 2013

**Hora inicio:** 11:30 a.m.

**Hora fin:** 3:00 p.m.

**Lugar:** CETEC del Campus Monterrey

**Participantes:**

Verónica Alejandra González González A01087523

Jorge Salazar Saltijeral A00945665

**Asuntos tratados**

En la sesión que se tuvo el día sábado 16 de febrero, nos enfocamos a definir el tipo de aplicación que se iba a desarrollar como parte del proyecto final de la clase de compiladores. Se estableció la visión y objetivo del proyecto, así como la plataforma en la que se desarrollara.

**Acuerdos tomados**

Se tomó la decisión de desarrollar una aplicación de un lenguaje básico orientado a objetos basado en Java y C++. Se eligió C++ como lenguaje de programación y Coco/R como herramienta para la creación del lenguaje. Asimismo, se acordó que el próximo sábado nos juntaríamos para realizar los diagramas de sintaxis y poder terminar la propuesta.

**Bitácora 2**

**Fecha:** Febrero 23 de 2013

**Hora inicio:** 10:45 a.m.

**Hora fin:** 7:00 p.m.

**Lugar:** CETEC del Campus Monterrey

**Participantes:**

Verónica Alejandra González González A01087523

Jorge Salazar Saltijeral A00945665

**Asuntos tratados**

En la sesión que se tuvo el día sábado 23 de febrero, se estableció el léxico a utilizar y se realizaron los diagramas de sintaxis en la herramienta Visio. Para poder identificar los elementos del lenguaje se realizó un ejemplo del código fuente que nuestro lenguaje permitirá.

**Acuerdos tomados**

Se tomó la decisión de terminar los diagramas de sintaxis de forma virtual el día domingo para poder entregar la propuesta el lunes para la revisión de esta misma y se empezó a desarrollar el léxico en Coco/R.

**Bitácora 3**

**Fecha:** Marzo 2 de 2013

**Hora inicio:** 11:00 a.m.

**Hora fin:** 10:00 p.m.

**Lugar:** CETEC del Campus Monterrey

**Participantes:**

Verónica Alejandra González González A01087523

Jorge Salazar Saltijeral A00945665

**Asuntos tratados**

En la sesión que se tuvo el día sábado 2 de marzo, se hicieron las correcciones de los diagramas de sintaxis de acuerdo a los comentarios que nos dio la Ing. Elda, asimismo, se desarrolló la gramática del lenguaje en Coco/R.

**Acuerdos tomados**

Cada quien haría códigos fuente de prueba para revisar que la gramática que desarrollamos en Coco/R funciona correctamente y acepta solo lo que se esperaba.

**Bitácora 4**

**Fecha:** Marzo 10 de 2013

**Hora inicio:** 11:00 a.m.

**Hora fin:** 7:30 p.m.

**Lugar:** CETEC del Campus Monterrey

**Participantes:**

Verónica Alejandra González González A01087523

Jorge Salazar Saltijeral A00945665

**Asuntos tratados**

En la sesión que se tuvo el día domingo 10 de marzo, se comenzó con la parte de semántica, donde realizamos el cubo semántico y se empezó a codificar la tabla de variables.

**Acuerdos tomados**

Continuar con la tabla de variables.

**Bitácora 5**

**Fecha:** Marzo 13 de 2013

**Hora inicio:** 7:20 p.m.

**Hora fin:** 11:00p.m.

**Lugar:** CETEC del Campus Monterrey

**Participantes:**

Verónica Alejandra González González A01087523

Jorge Salazar Saltijeral A00945665

**Asuntos tratados**

 Se corrigieron algunos errores que se presentaron en la gramática. Se implementó el cubo semántico para las operaciones que se puedan llevar acabo entre los diversos tipos de variables. Se analizó las implicaciones de las variables globales y locales, los métodos y como estos deben ser manejados al momento de herencia. Se crearon las estructuras de las tablas de variables y procedimientos. Se empezó a implementar en el archivo de las producciones sintácticas.

**Acuerdos tomados**

Se tomó la decisión de construir una tabla general dividida de acuerdo con la visibilidad (public, protect, private) de los atributos de la clase.

**Bitácora 6**

**Fecha:** Marzo 16 de 2013

**Hora inicio:** 12:00 p.m.

**Hora fin:** 7:00p.m.

**Lugar:** CETEC del Campus Monterrey

**Participantes:**

Verónica Alejandra González González A01087523

Jorge Salazar Saltijeral A00945665

**Asuntos tratados**

En la sesión se continuó con la instanciación de la tabla de variables y el cubo semántico. Nos dimos cuenta que el separar la tabla en tres (public, private y protect) aumentaba el tiempo de búsqueda para variables globales, por lo que solo se implementó una sola tabla de procedimientos y de variables.

**Acuerdos tomados**

Se tomó la decisión de juntarnos el lunes para empezar con la construcción de los cuádruplos.

**Bitácora 7**

**Fecha:** Marzo 18 de 2013

**Hora inicio:** 11:00 a.m.

**Hora fin:** 6:00p.m.

**Lugar:** CETEC del Campus Monterrey

**Participantes:**

Verónica Alejandra González González A01087523

Jorge Salazar Saltijeral A00945665

**Asuntos tratados**

En la sesión se empezó con la parte de documentación. Asimismo, se hicieron correcciones al cubo semántico. Se empezaron a construir las estructuras de la generación de los cuádruplos de las expresiones y se implementaron los algoritmos para la creación de los mismos.

**Acuerdos tomados**

Se tomó la decisión de juntarnos de manera virtual durante semana santa para poder avanzar con lo que quedó pendiente de los cuádruplos y poder terminarlo completamente.

**Bitácora 8**

**Fecha:** Marzo 27 de 2013

**Hora inicio:** 6:00 p.m.

**Hora fin:** 10:00p.m.

**Lugar:** Virtual por Gmail Chat y Skype

**Participantes:**

Verónica Alejandra González González A01087523

Jorge Salazar Saltijeral A00945665

**Asuntos tratados**

Se terminó la parte de los cuádruplos de expresiones y se agregó la tabla para administrar las clases de los objetos para poder manejar de mejor manera la herencia de estos. Asimismo, se empezó a construir los cuádruplos de asignación y de las condiciones (if y while).

**Acuerdos tomados**

Se tomó la decisión de juntarnos el lunes regresando de semana santa para poder hacer casos prueba y continuar con el avance que se entrega el miércoles.

**Bitácora 9**

**Fecha:** Abril 6 de 2013

**Hora inicio:** 11:00 a.m.

**Hora fin:** 6:00p.m.

**Lugar:** CETEC Campus Monterrey

**Participantes:**

Verónica Alejandra González González A01087523

Jorge Salazar Saltijeral A00945665

**Asuntos tratados**

Se terminaron de hacer los cuádruplos de asignación y estatutos condicionales (if & while). Asimismo, se implementó las operaciones de incremento (++), decremento (- -) y concatenación (^), además de los estatutos secuenciales de escritura y lectura.

**Acuerdos tomados**

Se tomó la decisión de juntarnos el día domingo para implementar el else if y empezar con la generación de código de funciones.

**Bitácora 10**

**Fecha:** Abril 7 de 2013

**Hora inicio:** 11:30 a.m.

**Hora fin:** 5:00p.m.

**Lugar:** CETEC Campus Monterrey

**Participantes:**

Verónica Alejandra González González A01087523

Jorge Salazar Saltijeral A00945665

**Asuntos tratados**

Se terminaron de hacer los cuádruplos del if ya que ayer solo se alcanzó a hacer el if – else sin embargo hoy se implementó la parte del if – elseif – else. Asimismo, se empezó a construir la estructura para la definición y llamadas de un procedimiento.

**Bitácora 11**

**Fecha:** Abril 9 de 2013

**Hora inicio:** 10:00 a.m.

**Hora fin:** 10:20p.m.

**Lugar:** CETEC 6° Piso Torre Norte

**Participantes:**

Verónica Alejandra González González A01087523

Jorge Salazar Saltijeral A00945665

Dr. José Ignacio Icaza

**Asuntos tratados**

Fuimos a la oficina del Dr. Icaza donde se le explico nuestro proyecto además de que reviso los diagramas de sintaxis y código fuente. Asimismo, le explicamos lo que llevábamos del proyecto y lo que teníamos pendiente. Le pedimos su recomendación sobre el lenguaje a usar en la máquina virtual donde nos dijo que si queríamos aprender un nuevo lenguaje nos recomendaba Python.

**Bitácora 12**

**Fecha:** Abril 12 de 2013

**Hora inicio:** 5:30 p.m.

**Hora fin:** 7:30p.m.

**Lugar:** CETEC Campus Monterrey

**Participantes:**

Verónica Alejandra González González A01087523

Jorge Salazar Saltijeral A00945665

**Asuntos tratados**

Se continuo con la parte de las funciones donde lo que hicimos fue la parte de la llamada a la función donde se hizo la relación con los parámetros.

**Acuerdos tomados**

Se tomó la decisión de juntarnos el día Sábado 13 para continuar con el proyecto.

**Bitácora 13**

**Fecha:** Abril 13 de 2013

**Hora inicio:** 11:00 a.m.

**Hora fin:** 4:30p.m.

**Lugar:** CETEC Campus Monterrey

**Participantes:**

Verónica Alejandra González González A01087523

Jorge Salazar Saltijeral A00945665

**Asuntos tratados**

Se terminó la parte de las funciones, se modificó el diagrama de sintaxis de la lectura pues ahora también recibe como parámetro el tipo de dato que está leyendo. Asimismo, se hizo limpieza del código pues había ciertas variables que no se estaban inicializando al terminar la llamada y se creó la tabla de constantes.

**Acuerdos tomados**

Se definieron los rangos de memoria para las variables locales, globales, temporales y las constantes.

**Bitácora 14**

**Fecha:** Abril 19 de 2013

**Hora inicio:** 3:30 p.m.

**Hora fin:** 6:00p.m.

**Lugar:** CETEC Campus Monterrey

**Participantes:**

Verónica Alejandra González González A01087523

Jorge Salazar Saltijeral A00945665

**Asuntos tratados**

Se empezó a construir la estructura para los arreglos, tanto para la declaración como para la asignación de valores. Asimismo, se modificó el diagrama de sintaxis del arreglo pues se tenía que se podía declarar un arreglo de tamaño que no es entero, es decir se podía aceptar arr[id] sin embargo esto estaba por fuera del alcance del proyecto.

**Acuerdos tomados**

Se tomó la decisión de juntarnos el sábado 20 de abril para agregar las validaciones de dimensión e instanciar la tabla de constantes.

**Bitácora 15**

**Fecha:** Abril 20 de 2013

**Hora inicio:** 11:30 a.m.

**Hora fin:** 6:00p.m.

**Lugar:** CETEC Campus Monterrey

**Participantes:**

Verónica Alejandra González González A01087523

Jorge Salazar Saltijeral A00945665

**Asuntos tratados**

Se implementaron las validaciones de dimensión, pues nos dimos cuenta que no validábamos el hecho de que se pudiera asignar un arreglo a un int (int i = arr). Asimismo, se instancio la tabla de constantes pues hasta ahora solo se tenía la estructura pero no guardábamos nada. Y se empezó con la documentación.

**Acuerdos tomados**

Se tomó la decisión de juntarnos el día 21 de abril para empezar con el desarrollo de la máquina virtual.

**Bitácora 16**

**Fecha:** Abril 21 de 2013

**Hora inicio:** 11:30 a.m.

**Hora fin:** 7:00p.m.

**Lugar:** CETEC Campus Monterrey

**Participantes:**

Verónica Alejandra González González A01087523

Jorge Salazar Saltijeral A00945665

**Asuntos tratados**

Se empezó con la estructura de la máquina virtual para que realice las operaciones aritméticas.

**Acuerdos tomados**

Se tomó la decisión de cambiar la estructura de la memoria a un map para que fuera más sencillo el mapeo de las direcciones de memoria.

**Bitácora 17**

**Fecha:** Abril 24 de 2013

**Hora inicio:** 7:30 p.m.

**Hora fin:** 9:45p.m.

**Lugar:** CETEC Campus Monterrey

**Participantes:**

Verónica Alejandra González González A01087523

Jorge Salazar Saltijeral A00945665

**Asuntos tratados**

Se corrigió un error que se tenía en la parte del cubo semántico, pues al hacer la suma con números double este no funcionaba ya que marcaba type mismatch. Se adelantó también la parte del append de la función write.

**Acuerdos tomados**

Se decidió juntarnos el Sábado 27 de abril para continuar con la generación de código de estatutos condicionales.

**Bitácora 18**

**Fecha:** Abril 27 de 2013

**Hora inicio:** 11:30 a.m.

**Hora fin:** 6:30p.m.

**Lugar:** CETEC Campus Monterrey

**Participantes:**

Verónica Alejandra González González A01087523

Jorge Salazar Saltijeral A00945665

**Asuntos tratados**

Se adelantó la parte de la documentación, se terminó la parte de escritura y lectura. Asimismo, se agregó la validación de división entre cero. Se terminaron los códigos de operación de los operadores relacionales, lógicos, concatenación, GTT, GTF, GTO, fin de módulo (END), verificación del arreglo (VER) y la asignación.

**Acuerdos tomados**

Decidimos juntarnos el domingo 28 de abril para continuar con la parte de funciones en la máquina virtual.

**Bitácora 19**

**Fecha:** Abril 28 de 2013

**Hora inicio:** 11:00 a.m.

**Hora fin:** 4:00p.m.

**Lugar:** CETEC Campus Monterrey

**Participantes:**

Verónica Alejandra González González A01087523

Jorge Salazar Saltijeral A00945665

**Asuntos tratados**

Se corrigieron unos detalles que teníamos en la parte de los operadores relacionales y lógicos pues no habíamos casteado correctamente a valor boolean. Asimismo, se trabajó en la parte de funciones en la máquina virtual.

**Acuerdos tomados**

Se tomó la decisión de juntarnos el miércoles 1 de mayo para continuar con lo que queda del proyecto.

**Bitácora 20**

**Fecha:** Mayo 1 de 2013

**Hora inicio:** 11:00 a.m.

**Hora fin:** 6:00p.m.

**Lugar:** CETEC Campus Monterrey

**Participantes:**

Verónica Alejandra González González A01087523

Jorge Salazar Saltijeral A00945665

**Asuntos tratados**

Se corrigió la parte de los arreglos en la máquina virtual ya que no estaba funcionando correctamente la parte del acceso a los elementos del arreglo. Asimismo, se trabajó en la parte de documentación y se empezó a estructurar la parte de los objetos en la máquina virtual.

**Acuerdos tomados**

Se decidió juntarse el viernes 3 de mayo para seguir con lo que queda del proyecto.

**Bitácora 21**

**Fecha:** Mayo41 de 2013

**Hora inicio:** 11:45 a.m.

**Hora fin:** 7:00p.m.

**Lugar:** CETEC Campus Monterrey

**Participantes:**

Verónica Alejandra González González A01087523

Jorge Salazar Saltijeral A00945665

**Asuntos tratados**

Debido a que el viernes 3 de mayo nos juntamos para hacer la actividad de puntos extra, nos quedamos en juntar al día siguiente 4 de mayo. Se adelantó la parte de documentación y se modificó la parte de los arreglos, ya que al querer sumar dos arreglos no funcionaba correctamente.

**Acuerdos tomados**

Se decidió juntarse el día 5 de mayo para continuar con el proyecto.

Por último se presentan las reflexiones de cada alumno así como los aprendizajes obtenidos.

* Verónica González \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Después de haber terminado este proyecto me siento más preparada en el ámbito de los compiladores, asimismo, me di cuenta de que son muchas cosas las que se desencadenan cuando nosotros compilamos y ejecutamos nuestro código y que no es tan sencillo de lo que parece pues hay muchas cosas como la memoria que se necesita manejar correctamente para que funcione adecuadamente. El proyecto también me dio la oportunidad de poder aprender más a fondo el lenguaje C++ pues tanto el compilador como la máquina virtual están codificados en este lenguaje.

* Jorge Salazar \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Descripción del Lenguaje

## Características Principales

* Nombre del lenguaje: Solstice
* El léxico y sintaxis del lenguaje está desarrollado en CocoR
* La estructura semántica está desarrollada en C++
* Se utilizaron las siguientes estructuras de datos: Map, Stack, Vector
* La máquina virtual está desarrollada en C++
* Es un lenguaje orientado a objetos
* Permite declarar y asignar variables
* Permite declarar funciones y arreglos
* Permite tener expresiones aritméticas, lógicas y relacionales
* Permite tener ciclos y estatutos condicionales

## Principales Errores

# Descripción del Compilador

## Desarrollo

El compilador está hecho utilizando el lenguaje CocoR/C++ usando librerías como lo son <stack>, <map>, <vector> para las estructuras que se utilizaron para la administración de las direcciones. Para el desarrollo de este se utilizó Visual Studio 201/2012 así como el uso de Github y Dropbox para el manejo de versiones como para el respaldo de la documentación.

## Análisis Léxico

Operadores Relacionales

* Equivalente ==
* Mayor que >
* Menor que <
* Mayor o igual que >=
* Menor o igual que <=
* Diferente <>
* Asignación =

Operadores Matemáticos

* Suma +
* Resta -
* Multiplicación \*
* División /
* Módulo %
* Incremento ++
* Decremento --

Operadores Lógicos

* And &
* Or |
* Negación ~

Delimitadores

* Llamada método .
* Punto y coma ;
* Coma ,
* Concatenación ^
* Paréntesis ( )
* Corchetes [ ]
* Llaves { }
* Comentarios /\* \*/

Constante

* String “ ”
* Entera (0-9)+ 🡪 4 bytes, rango –2,147,483,648 a 2,147,483,647
* Double (0-9)+ ‘.’ (0-9)+ 🡪 8 bytes, rango 1.7E +/- 308
* Boolean true, false 🡪 1 byte

Identificador

* Variables ‘\_’ A-Za-z(A-Za-z|0-9)\*
* Métodos ‘#’ A-Za-z(A-Za-z|0-9)\*
* Método Constructor ‘$’ A-Z(A-Za-z|0-9)\*

Lectura >>

Escritura <<

Herencia ->

Condicional while @

Ciclo if ?, elseif :?, else :

Palabras reservadas

class false

public true

private string

protect int

return double

new bool

void if

while else

CHARACTERS

letra = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz".

letraU = "ABCDEFGHIJKLMNOPQRSTUVWXYZ".

digito = "0123456789".

rc = '\r'.

sl = '\n'.

tab = '\t'.

otrosC = ANY - '/' - '\*'.

otrosS = ANY - '"' - rc - sl - tab.

TOKENS

idV = '\_'letra {letra | digito}. /\* Id Variable \*/

idM = '#' letra {letra | digito}. /\* Id Metodo \*/

idO = letraU {letra | digito}. /\* Id Objeto Nombre de la Clase \*/

idC = '$' letraU {letra | digito}. /\* Id Constructor \*/

cteI = digito {digito}.

cteD = digito {digito} "." digito {digito} .

cteS = '\"' otrosS {otrosS} '\"'.

cteB = "true" | "false".

tCla = "class".

tPub = "public".

tPri = "private".

tPro = "protect".

tRet = "return".

tStr = "string".

tInt = "int".

tDou = "double".

tBoo = "bool".

tNew = "new".

tVoi = "void".

tCon = "if" | "?".

tEls = "else" | ":".

tEif = "elseif" | ":?".

tWhi = "while" | "@".

tExt = '-''>'.

tMai = "main".

tLec = '>''>'.

tEsc = '<''<'.

tInc = '+''+'.

tDec = '-''-'.

tMay = '>''='.

tMen = '<''='.

tDif = '<''>'.

tEqu = '=''='.

PRAGMAS

comentarios = "/\*" {'/' | otrosC | '\*' {'\*'} otrosC} '\*' {'\*'} '/'.

## Análisis Sintáctico

A continuación se presenta la gramática formal para el lenguaje Solstice:

<programa>

Solstice = Class Main.

<arreglo>

Arr = '[' cteI ']'.

Arr2 = '['ExpOY ']'.

<asignación>

Asig = ((tInc | tDec | Asig2) | Arr2 Asig2)';'.

Asig2 = '=' ExpOY.

<atributo>

Atributo = Vis Tipo idV ';'.

<bloque>

Bloque = Atributo{Atributo} Constructor {Vis (MetodoR | MetodoV)}.

<ciclo>

Ciclo = tWhi '(' ExpOY ')' '{' {Estatuto} '}'.

<clase>

Class = tCla idO [tExt idO] '{' Bloque '}' [Class].

<condición>

Con = tCon '(' ExpOY ')' '{' Estatuto {Estatuto} '}' { tEif '(' ExpOY ')' '{' Estatuto {Estatuto} '}'}.

Con2 = tEls '{' Estatuto {Estatuto} '}'.

ConG = Con [Con2].

<constructor>

Constructor = idC '(' [Param] ')' '{' {Init} '}'.

<constantes>

CTE = (cteS | cteI | cteD | cteB).

CTES = idV | CTE.

<declaración>

Decl = (New | Tipo idV ( '=' CTES | Arr) {',' idV ( '=' CTES | Arr )})';'.

<escritura>

Esc = tEsc '(' ExpOY {'^'ExpOY}')'';'.

<estatuto>

Estatuto = (idV (Asig | Llamada)) | Metodo ';' | Ciclo | Esc | Lec | ConG.

<expresión>

Exp = Termino {('+' | '-')) Termino}.

ExpOY = Expresion {('&' | '|' ) Expresion}.

Expresion = Exp (tMay | tMen | tDif | '<' | '>' | tEqu) Exp].

<factor>

Factor = ['~'|'-'] ( '(' ExpOY ')' | idV [Llamada|Arr2] | CTE | Metodo).

<inicializa>

Init = idV '=' CTES ';'.

<lectura>

Lec = tLec '(' idV ',' Tipo ')' ';'.

<lista>

Lista = CTES [',' Lista].

<main>

Main = tMai '(' ')' '{' {Decl} {Estatuto} '}'.

<método>

Metodo = idM '(' [Lista] ')'.

MetodoR = Tipo idM '(' [Param] ')' '{' {Decl} {Estatuto} Return '}'.

MetodoV = tVoi idM '(' [Param] ')' '{' {Decl} {Estatuto} '}'.

<new>

New = idO idV '=' tNew idC '(' [Lista] ')'.

<parámetros>

Param = Tipo idV [',' Param].

<llamada>

Llamada = '.' (Metodo | idV).

<return>

Return = tRet ExpOY ';'.

<termino>

Termino = Factor {( '\*' | '/' | '%' ) Factor}.

<tipo>

Tipo = tInt | tDou | tStr | tBoo.

<visibilidad>

Vis = tPub | tPri | tPro.

## Generación de Código Intermedio y Análisis Semántico

A continuación se muestra la tabla con los identificadores (código de operación) que se utilizó para cada uno de los operadores en el lenguaje:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operador** | **Código de operación** | **Operador** | **Código de operación** |
| + | 0 | GOTOV | 15 |
| - | 1 | GOTOF | 16 |
| \* | 2 | GOTO | 17 |
| / | 3 | read >> | 18 |
| % | 4 | write << | 19 |
| & | 5 | ERA | 20 |
| | | 6 | GOSUB | 21 |
| < | 7 | RET | 22 |
| > | 8 | return | 23 |
| == | 9 | PARAM | 24 |
| <> | 10 | VER | 25 |
| >= | 11 | MAIN | 26 |
| <= | 12 | ACC | 27 |
| ^ | 13 | NOT | 28 |
| = | 14 | END | 29 |

A continuación se muestra los rangos de las direcciones virtuales que se utilizaron para cada una de las variables:

**Variables Globales**

* Entero 0 - 999
* Double 1000 - 1999
* String 2000 - 2999
* Boolean 3000 - 3999

**Variables Locales**

* Entero 4000 - 5999
* Double 6000 - 7999
* String 8000 - 9999
* Boolean 10000 - 11999

**Variables Temporales**

* Entero 12000 - 13999
* Double 14000 - 15999
* String 16000 - 17999
* Boolean 18000 - 19999
* Pointer 23002 - 23999

**Constantes**

* Entero 20000 - 20999
* Double 21000 - 21999
* String 22000 - 22999
* Boolean 23000 - 23001

**Objetos**

* Objeto 24000 – 29999

Diagramas de sintaxis









A continuación se muestra la tabla de consideraciones semánticas

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **op1** | **op2** | **+** | **-** | **\*** | **/** | **%** | **&** | **|** | **<** | **>** | **==** | **<>** | **>=** | **<=** | **^** | **=** |
| int | int | int | int | int | int | int | E | E | bool | bool | bool | bool | bool | bool | string | int |
| int | double | E | E | E | E | E | E | E | E | E | E | E | E | E | string | E |
| int | string | E | E | E | E | E | E | E | E | E | E | E | E | E | string | E |
| int | bool | E | E | E | E | E | E | E | E | E | E | E | E | E | string | E |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **op1** | **op2** | **+** | **-** | **\*** | **/** | **%** | **&** | **|** | **<** | **>** | **==** | **<>** | **>=** | **<=** | **^** | **=** |
| double | int | E | E | E | E | E | E | E | E | E | E | E | E | E | string | E |
| double | double | double | double | double | double | int | E | E | bool | bool | bool | bool | bool | bool | string | double |
| double | string | E | E | E | E | E | E | E | E | E | E | E | E | E | string | E |
| double | bool | E | E | E | E | E | E | E | E | E | E | E | E | E | string | E |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **op1** | **op2** | **+** | **-** | **\*** | **/** | **%** | **&** | **|** | **<** | **>** | **==** | **<>** | **>=** | **<=** | **^** | **=** |
| string | int | E | E | E | E | E | E | E | E | E | E | E | E | E | string | E |
| string | double | E | E | E | E | E | E | E | E | E | E | E | E | E | string | E |
| string | string | E | E | E | E | E | E | E | E | E | bool | bool | E | E | string | string |
| string | bool | E | E | E | E | E | E | E | E | E | E | E | E | E | string | E |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **op1** | **op2** | **+** | **-** | **\*** | **/** | **%** | **&** | **|** | **<** | **>** | **==** | **<>** | **>=** | **<=** | **^** | **=** |
| bool | int | E | E | E | E | E | E | E | E | E | E | E | E | E | string | E |
| bool | double | E | E | E | E | E | E | E | E | E | E | E | E | E | string | E |
| bool | string | E | E | E | E | E | E | E | E | E | E | E | E | E | string | E |
| bool | bool | E | E | E | E | E | bool | bool | E | E | bool | bool | E | E | string | bool |

Por último se muestra la descripción de los cuádruplos que se utilizan a lo largo del compilador:

1. Suma: este cuádruplo se encarga de hacer la suma entre dos operandos que son del mismo tipo de dato numérico, guardando el resultado en una dirección de memoria temporal (SUM, op1, op2, temp)
2. Resta: este cuádruplo se encarga de hacer la resta entre dos operandos que son del mismo tipo de dato numérico, guardando el resultado en una dirección de memoria temporal (SUB, op1, op2, temp)
3. Multiplicación: este cuádruplo se encarga de hacer la multiplicación entre dos operandos que son del mismo tipo de dato numérico, guardando el resultado en una dirección de memoria temporal (MUL, op1, op2, temp)
4. División: este cuádruplo se encarga de hacer la división entre dos operandos que son del mismo tipo de dato numérico, guardando el resultado en una dirección de memoria temporal (DIV, op1, op2, temp)
5. Módulo: este cuádruplo se encarga de hacer el módulo entre dos operandos que son del mismo tipo de dato numérico, guardando el resultado en una dirección de memoria temporal (MOD, op1, op2, temp)
6. AND: este cuádruplo se encarga de hacer el AND entre dos operandos que son del mismo tipo de dato lógico, guardando el resultado en una dirección de memoria temporal (AND, op1, op2, temp)
7. OR: este cuádruplo se encarga de hacer el OR entre dos operandos que son del mismo tipo de dato lógico, guardando el resultado en una dirección de memoria temporal (OR, op1, op2, temp)
8. Menor que: este cuádruplo se encarga de hacer la operación de Menor que entre dos operandos que son del mismo tipo numérico, guardando el resultado en una dirección de memoria temporal (LT, op1, op2, temp)
9. Mayor que: este cuádruplo se encarga de hacer la comparación de Mayor que entre dos operandos que son del mismo tipo numérico, guardando el resultado en una dirección de memoria temporal (GT, op1, op2, temp)
10. Menor o igual que: este cuádruplo se encarga de hacer la comparación de Menor o igual que entre dos operandos que son del mismo tipo numérico, guardando el resultado en una dirección de memoria temporal (LEQ, op1, op2, temp)
11. Mayor o igual que: este cuádruplo se encarga de hacer la comparación de Mayor o igual que entre dos operandos que son del mismo tipo numérico, guardando el resultado en una dirección de memoria temporal (GET, op1, op2, temp)
12. Igual: este cuádruplo se encarga de hacer la comparación de igual entre dos operandos que son del mismo tipo de dato, guardando el resultado en una dirección de memoria temporal (EQ, op1, op2, temp)
13. Diferente de: este cuádruplo se encarga de hacer la comparación de diferente entre dos operandos que son del mismo tipo de datos, guardando el resultado en una dirección de memoria temporal (DIF, op1, op2, temp)
14. Concatenación: este cuádruplo se encarga de hacer la concatenación entre dos operandos sin importar el tipo de dato, guardando el resultado en una dirección de memoria temporal (APP, op1, op2, temp)
15. Asignación: este cuádruplo se encarga de hacer la asignación de un operando a otra variables siempre y cuando sean del mismo tipo de dato, guardando el resultado en una dirección de memoria de la variable a la que quieren asignar (ASI, op1, -1, mem)
16. GOTV: no fue necesario implementarlo
17. GOTOF: este cuádruplo se encarga de hacer un salto hacia el número de cuádruplo indicado siempre y cuando el operando sea falso (GTF, op1, -1, mem)
18. GOTO: este cuádruplo se encarga de hacer un salto hacia el número de cuádruplo indicado (GTO, -1, -1, mem)
19. Lectura: este cuádruplo se encarga de leer datos desde consola, donde recibe el dato de entrada y el tipo de dato (REA, dato, tipo, -1)
20. Escritura: este cuádruplo se encarga de escribir datos en la consola donde recibe el dato que desea escribir (WRI, dato, -1, -1)
21. ERA: este cuádruplo se encarga de identificar a la función que es llamada, en donde recibe la memoria del procedimiento esto con el fin de crear el espacio en memoria para esta función (ERA, mem, -1, -1)
22. GOSUB: este cuádruplo se encarga de hacer un salto a la función que está siendo llamada (GSU, mem, -1, -1)
23. RET: este cuádruplo se encarga de indicar a la máquina virtual que se ha terminado la función (RET, -1, -1, -1)
24. Retorno: este cuádruplo se encarga de regresar el valor de retorno de la función enviándole el operando y la dirección de memoria del procedimiento (MR, op, -1, mem)
25. PARAM: este cuádruplo se encarga de mandar como parámetro el valor que está en la dirección de memoria del parámetro (PARAM, mem, -1, param)
26. VER: este cuádruplo se encarga de revisar que el índice del arreglo este entre el rango (VER, mem, -1, rango)
27. Main: este cuádruplo se encarga de hacer un salto hacia el número de cuádruplo indicado (MAI, mem, -1, -1)
28. Acceso: este cuádruplo se encarga de sumar el operador con la dirección base del arreglo y guardarlo en el apuntador de memoria (ACC, op2, op1, mem)
29. Not:
30. End: este cuádruplo se encarga de indicar a la máquina virtual que el programa a llegado a su fin (END, -1, -1, -1)

## Administración de Memoria en Compilación

A continuación se presenta una descripción de las diferentes estructuras que se utilizaron para el manejo de memoria en la parte de compilación:

Pila de Operandos: pila en la que se guardan las direcciones de los operandos



Pila de Operadores: pila en la que se guardan los códigos de operación de las expresiones



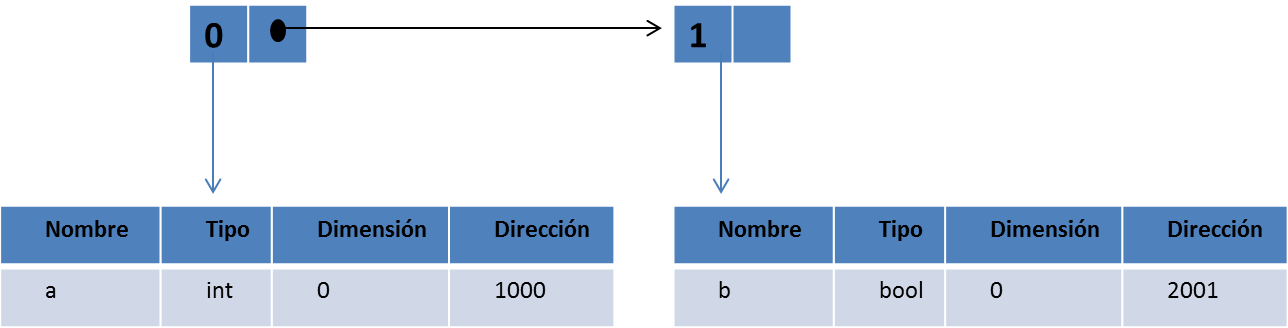
Pila de Saltos: pila en la que se guardan los saltos que son utilizados en los códigos de operación GOTO y GOTOF



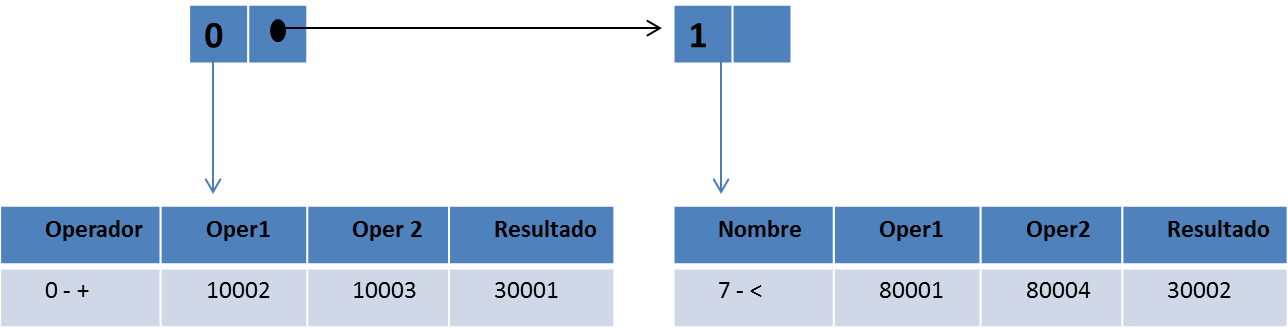
Pila de IF: pila en la que se guardan la cantidad de if’s anidados que se tienen



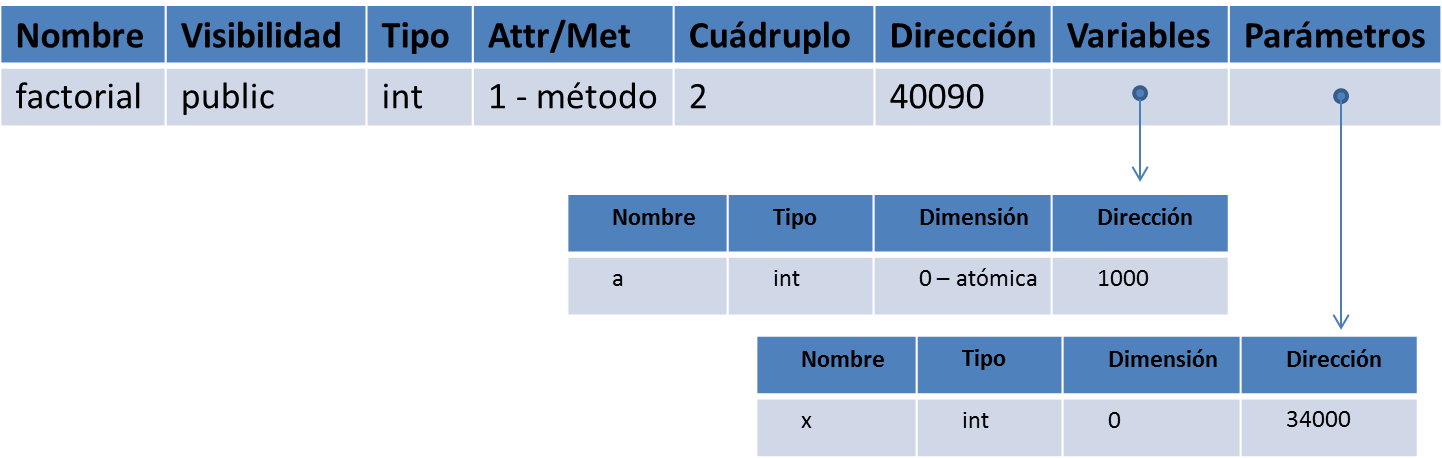
Vector de Parámetros: estructura en la que se guardan los parámetros de la función



Vector de Cuádruplos: estructura en la que se guardan los cuádruplos que se van generando



Directorio de Procedimientos: estructura de tipo Hash Map teniendo como llave el nombre de la función o atributo según sea el caso donde guarda su visibilidad, tipo de función/atributo, indicando si es atributo o función y un apuntador al vector de parámetros en caso de ser función



Directorio de Clases: estructura de tipo Hash Map teniendo como llave el nombre de la clase donde se guarda el padre de la clase en caso de que herede de alguna clase, asimismo se crea un apuntador a la tabla de atributos de la clase

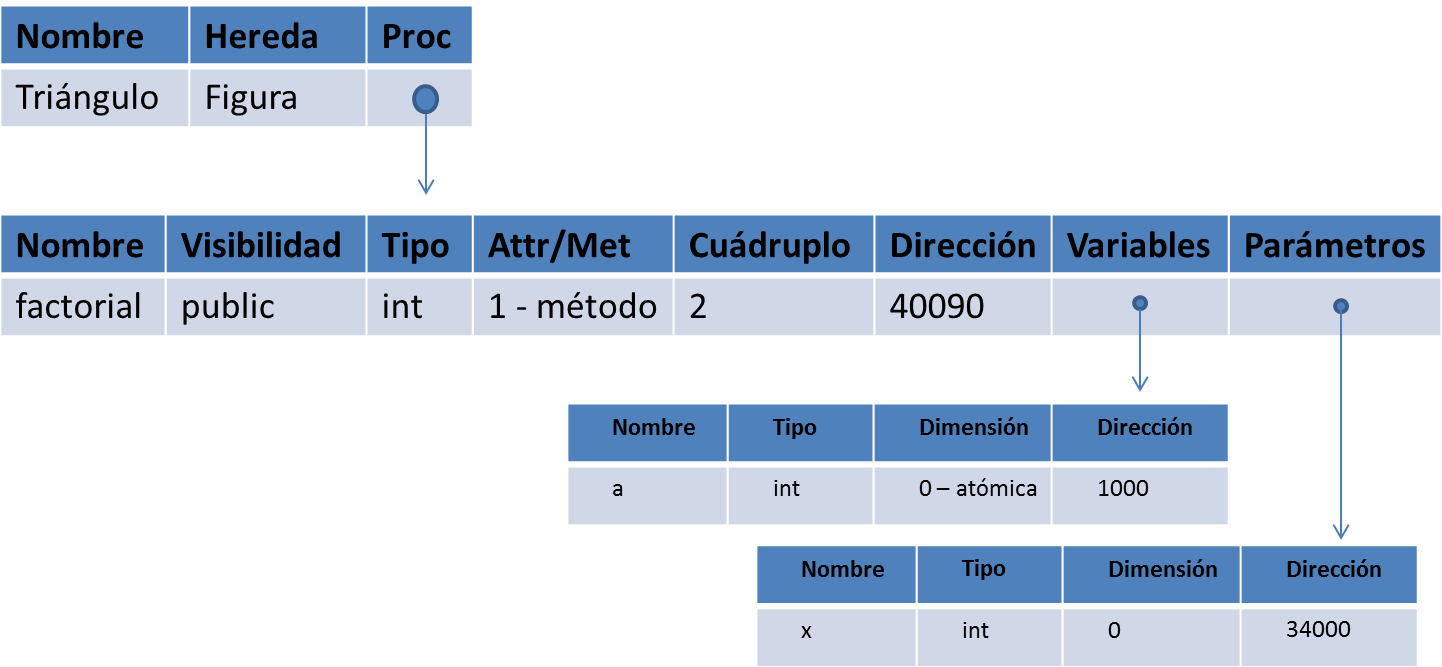


Tabla de Variables: estructura de tipo Hash Map teniendo como llave el nombre de la variable donde se guarda el tipo de dato, la dimensión (atómica o arreglo) y la dirección

|  |  |  |  |
| --- | --- | --- | --- |
| **Nombre** | **Tipo** | **Dimensión** | **Dirección** |
| a | int | 0 – atómica | 1000 |
| arr | double | 2 – arreglo dimensión 2 | 20003 |

Tabla de Constantes: estructura de tipo Hash Map donde se guarda el tipo de dato y la dirección donde la llave es el valor de la constante

|  |  |  |
| --- | --- | --- |
| **Nombre** | **Tipo** | **Dirección** |
| 45 | int | 5000 |
| 2 | int | 5002 |
| True | bool | 1400 |

# Descripción de la Máquina Virtual

## Desarrollo

La máquina virtual está hecha utilizando el lenguaje C++ usando librerías como lo son <vector> para la administración de la memoria en la máquina virtual. Para el desarrollo de este se utilizó Visual Studio 201/2012 así como el uso de Github y Dropbox para el manejo de versiones y para el respaldo de la documentación.

## Administración de Memoria en Ejecución

# Pruebas de Funcionamiento de Lenguaje

# Código

# Bibliografía

(2013). Data Type Ranges. Recuperado el 27 de abril de 2013, a partir de <http://msdn.microsoft.com/en-us/library/s3f49ktz(v=vs.80).aspx>

Mössenböck, Hanspeter, Markus Löberbauer y Albrecht Wöb. (2011). Coco/R. *University of Linz*. Recuperado el 23 de febrero de 2013, a partir de <http://www.ssw.uni-linz.ac.at/coco/Doc/UserManual.pdf>

Rueda, Antonio Jesús. (2008). Programación orientada a objetos en Java. Universidad de Jaén. Recuperado el 23 de febrero de 2013, a partir de <http://wwwdi.ujaen.es/asignaturas/progav/progav-tema2.pdf>