Instituto Tecnológico y de Estudios Superiores de Monterrey

Diseño de Compiladores

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

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

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

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

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

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

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

Fernando Garza Conde A01191305

Javier Guzmán Araiza A01175626

25 de noviembre de 2015

Tabla de Contenidos

[Visión y Objetivos 5](#_Toc436184335)

[Análisis de Requerimientos 5](#_Toc436184336)

[Requerimientos generales 5](#_Toc436184337)

[Requerimientos propios del lenguaje 6](#_Toc436184338)

[Diagrama de Casos de Uso 7](#_Toc436184339)

[Descripción de los Test Cases 8](#_Toc436184340)

[Descripción del Proceso 9](#_Toc436184341)

[Avance #1 9](#_Toc436184342)

[Avance #2 9](#_Toc436184343)

[Avance #3 9](#_Toc436184344)

[Avance #4 9](#_Toc436184345)

[Avance #5 9](#_Toc436184346)

[Avance #6 10](#_Toc436184347)

[Avance #7 10](#_Toc436184348)

[Avance #8 10](#_Toc436184349)

[Lecciones aprendidas – Fernando 10](#_Toc436184350)

[Lecciones aprendidas – Javier 10](#_Toc436184351)

[Nombre del lenguaje 11](#_Toc436184352)

[Descripción de sus características 11](#_Toc436184353)

[Descripción de errores 11](#_Toc436184354)

[Durante compilación 11](#_Toc436184355)

[Durante ejecución 12](#_Toc436184356)

[Equipo de Cómputo, Lenguaje y Utilerías 13](#_Toc436184357)

[Equipo de Cómputo 13](#_Toc436184358)

[Lenguaje de programación 13](#_Toc436184359)

[Utilerías 13](#_Toc436184360)

[Análisis de Léxico 13](#_Toc436184361)

[Expresiones regulares principales 13](#_Toc436184362)

[Tokens 13](#_Toc436184363)

[Palabras reservadas 14](#_Toc436184364)

[Jerarquía 14](#_Toc436184365)

[Análisis de Sintaxis 15](#_Toc436184366)

[Gramatica Formal 15](#_Toc436184367)

[Análisis Semántico y Generación de Código Intermedio 18](#_Toc436184368)

[Cuádruplos generados 18](#_Toc436184369)

[Asignación 18](#_Toc436184370)

[Aritméticas 18](#_Toc436184371)

[Lógicas 19](#_Toc436184372)

[Booleanas 19](#_Toc436184373)

[Brincos y funciones 19](#_Toc436184374)

[Operaciones específicas del lenguaje 19](#_Toc436184375)

[Arreglos 19](#_Toc436184376)

[Diagramas de Sintaxis 19](#_Toc436184377)

[Cubo Semántico 26](#_Toc436184378)

[Administración de Memoria - Compilación 27](#_Toc436184379)

[Cubo Semántico 27](#_Toc436184380)

[Clase variable 27](#_Toc436184381)

[Clase directorio 28](#_Toc436184382)

[Memoria Virtual 28](#_Toc436184383)

[Directorio Actual 29](#_Toc436184384)

[Cuádruplo 29](#_Toc436184385)

[Generador de Cuádruplos 29](#_Toc436184386)

[Administración de Memoria - Ejecución 30](#_Toc436184387)

[Memoria Global 30](#_Toc436184388)

[Traducción direcciones Virtuales 🡪 Reales 31](#_Toc436184389)

[Pruebas 31](#_Toc436184390)

[Factorial Iterativo 31](#_Toc436184391)

[Código en Cheese ++ 31](#_Toc436184392)

[Código Intermedio 31](#_Toc436184393)

[Resultado 32](#_Toc436184394)

[Factorial Recursivo 32](#_Toc436184395)

[Código en Cheese ++ 32](#_Toc436184396)

[Código Intermedio 33](#_Toc436184397)

[Resultado 33](#_Toc436184398)

[Fibonacci 34](#_Toc436184399)

[Código en Cheese ++ 34](#_Toc436184400)

[Código Intermedio 34](#_Toc436184401)

[Resultado 35](#_Toc436184402)

[Bubble Sort 35](#_Toc436184403)

[Código en Cheese ++ 35](#_Toc436184404)

[Código Intermedio 36](#_Toc436184405)

[Resultado 38](#_Toc436184406)

[Search 38](#_Toc436184407)

[Código en Cheese ++ 38](#_Toc436184408)

[Código Intermedio 38](#_Toc436184409)

[Resultado 39](#_Toc436184410)

[Dibujo – Sierpinski 39](#_Toc436184411)

[Código en Cheese ++ 39](#_Toc436184412)

[Código Intermedio 40](#_Toc436184413)

[Resultado 42](#_Toc436184414)

[Listados 43](#_Toc436184415)

[Declaración Arreglo 43](#_Toc436184416)

[Cubo Semántico 44](#_Toc436184417)

[Clase Variables y Directorio 48](#_Toc436184418)

[Máquina Virtual 54](#_Toc436184419)

[Máquina Virtual – Resta 56](#_Toc436184420)

[Manual de Usuario 58](#_Toc436184421)

[Tipos de variables 58](#_Toc436184422)

[Declaración de variables 58](#_Toc436184423)

[Operaciones permitidas 58](#_Toc436184424)

[Asignación de valores: 58](#_Toc436184425)

[Estructura general de un programa 58](#_Toc436184426)

[Estructura general de un if 58](#_Toc436184427)

[Estructura general de un for 59](#_Toc436184428)

[Estructura general de un while 59](#_Toc436184429)

[Funciones predeterminadas 59](#_Toc436184430)

# Visión y Objetivos

Nuestra visión es que este lenguaje preparará y motivará a las nuevas generaciones a adentrarse en el mundo de la programación para que en un futuro continúen este camino como desarrolladores. Se cree que la manera más fácil es a través de la programación gráfica, la cual permitirá que los jóvenes comprendan los fundamentos básicos de cualquier lenguaje de programación.

Cheese++ es un lenguaje de gráfico diseñado particularmente para jóvenes de secundaria y preparatoria. Tiene como objetivo ayudar a aprender los fundamentos de la programación a través de una experiencia interactiva y divertida. El lenguaje les permitirá usar elementos gráficos como líneas, círculos, curvas y otras figuras geométricas. Se espera que los jóvenes aprendan de una manera más intuitiva los conceptos básicos, como lo son los estatutos de ciclos, condición, y la lógica de programación para así obtener mejores y más rápidos resultados de aprendizaje.

# Análisis de Requerimientos

## Requerimientos generales

|  |  |
| --- | --- |
| ID | RF-001 |
| Título | Estatutos de asignación |
| Descripción | El usuario deberá de poder asignar valores a las variables que cree, ya sea de tipo: *int*, *double* o *string*. |

|  |  |
| --- | --- |
| ID | RF-002 |
| Título | Estatutos de condición |
| Descripción | El usuario deberá de poder crear flujos de condición simples, como: *if* o *if-else* |

|  |  |
| --- | --- |
| ID | RF-003 |
| Título | Estatutos de Ciclo |
| Descripción | El usuario deberá de poder crear estatutos de repetición, como: *for* o *while*. |

|  |  |
| --- | --- |
| ID | RF-004 |
| Título | Estatuto de Lectura |
| Descripción | El usuario deberá de poder desplegar a pantalla a través de la función: *print*. |

|  |  |
| --- | --- |
| ID | RF-005 |
| Título | Estatuto de Escritura |
| Descripción | El usuario deberá de poder introducir valores durante el tiempo de ejecución a través del teclado con la función: *scanf* |

|  |  |
| --- | --- |
| ID | RF-006 |
| Título | Expresiones Aritméticas |
| Descripción | El usuario deberá de poder realizar las operaciones aritméticas básicas como: *+*, *-*, *\** y */* |

|  |  |
| --- | --- |
| ID | RF-007 |
| Título | Expresiones Lógicas |
| Descripción | El usuario deberá de poder realizar operaciones lógicas simples como: *&&* (and) y *||* (or). |

|  |  |
| --- | --- |
| ID | RF-008 |
| Título | Funciones |
| Descripción | El usuario deberá de poder declarar funciones adicionales al bloque principal. Estos pueden ser de tipo: *int*, *double*, *string* o *void*. |

|  |  |
| --- | --- |
| ID | RF-009 |
| Título | Vectores/Arreglos |
| Descripción | El usuario deberá de poder declarar arreglos de tamaño fijo, inicializados en 0. Acceder y asignar valores a cualquiera de los índices de dicha estructura. |

## Requerimientos propios del lenguaje

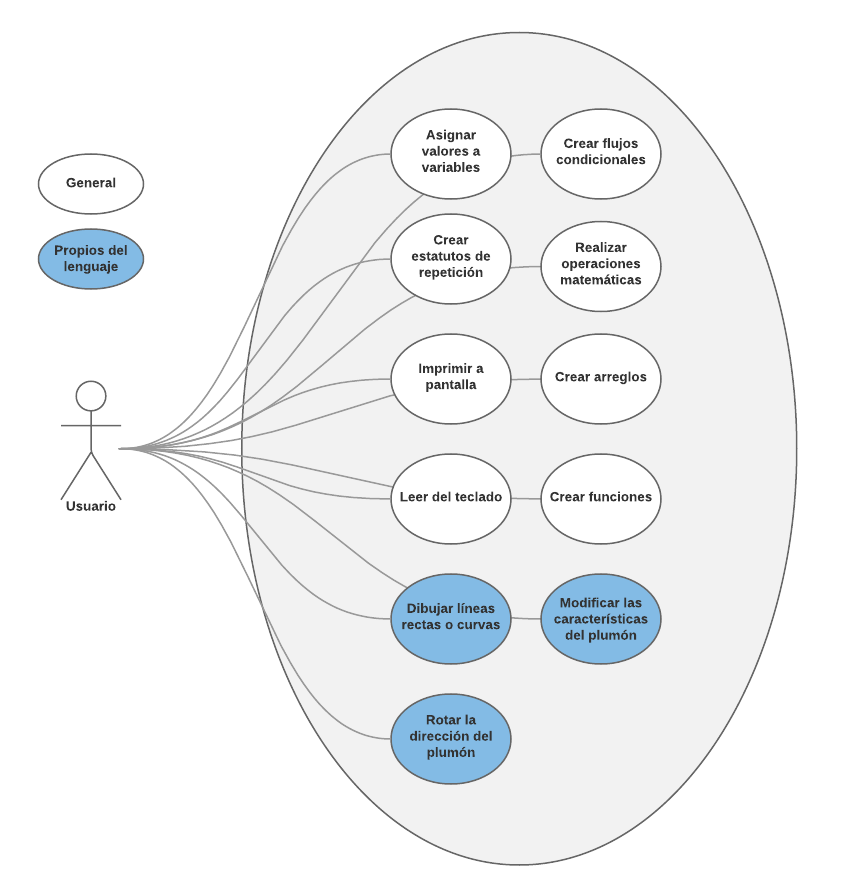
|  |  |
| --- | --- |
| ID | RF-010 |
| Título | Movimiento |
| Descripción | El usuario deberá de poder mover el plumón en línea recta con el cual ira trazando figuras a través de la pantalla con la función: *move.* |

|  |  |
| --- | --- |
| ID | RF-011 |
| Título | Rotaciones |
| Descripción | El usuario deberá de poder cambiar la dirección hacia la cual apunta el plumón con la función: *rotate.* |

|  |  |
| --- | --- |
| ID | RF-012 |
| Título | Arcos/Círculos |
| Descripción | El usuario deberá de poder mover el plumón de forma curveada con el cual ira trazando figuras a través de la pantalla con la función: *arc.* |

|  |  |
| --- | --- |
| ID | RF-013 |
| Título | Funciones Pluma |
| Descripción | El usuario deberá de poder modificar ciertas características de su plumón, como su tamaño, color, posición, estado actual (pintando o no pintando) con las funciones: *psize, setp, pcolor, pup, pdown, home.* |

## Diagrama de Casos de Uso



# Descripción de los Test Cases

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre De la Prueba | Descripción | Archivo | Output Esperado |
| Factorial Recursivo | Esta prueba busca demostrar el uso de funciones y llamadas recursivas en el lenguaje. Usa una función recursiva que devuelve el factorial de un número que se da como parámetro (7). Este se calcula multiplicando el valor del parámetro por la respuesta de una llamada recursiva al método, disminuyendo el valor del parámetro hasta llegar al caso base (0). | Factorial Recursivo.txt | 5040 |
| Factorial Iterativo | Esta prueba sirve para mostrar el uso de funciones, ciclos ‘for’ y condicionales en el lenguaje. Utiliza una función que regresa el factorial de un número que se da como parámetro (7). Este se calcula multiplicando todos los valores desde uno hasta el parámetro, iterando en un ciclo ‘for’. | Factorial Iterativo.txt | 5040 |
| Fibonacci | Esta prueba sirve para mostrar el uso de funciones y ciclos ‘for’. Usa una función que calcula e imprime una cierta cantidad de datos de la secuencia de Fibonacci. La cantidad se especifica como parámetro. Inicia con dos valores (0 y 1), imprimiendo en cada iteración de un ciclo el valor del segundo número y luego ajustando estos de forma que el segundo tenga el valor de la suma de ambos, y el primero pase a tener el valor anterior del segundo número. | Fibonacci.txt | 1  1  2  3  5  8  13 |
| Search/Find | Esta prueba demuestra el uso de variables globales y arreglos. Tiene una función “search” la cual recibe dos enteros como parámetros (n y k). La función itera sobre los primeros k elementos de un arreglo global, regresando el índice de la primera posición en la que se encuentre el entero n, o -1 si no lo encontró. | Search.txt | 5 |
| Bubble Sort | Esta prueba demuestra el uso de variables globales y arreglos. Tiene una función ‘sort’, la cual recibe dos enteros como parámetros, k y n, que denotan la posición inicial en el arreglo, y la cantidad de elementos a ordenar. En la función simplemente itera con un doble ‘for’ sobre los primeros n elementos a partir de la posición k de un arreglo global, ordenando conforme al algoritmo de ordenamiento en burbuja. El arreglo se inicializa con los valores de 99 hasta 0, después se ordenan los primeros 6 elementos con la función mencionada, y finalmente se ordenan otros 6 elementos pero empezando en la posición 4 del arreglo. Por último se imprimen los valores del arreglo. | Bubble Sort.txt | 94  95  96  97  90  91  92  93  98  99  .  .  . |

# Descripción del Proceso

## Avance #1

Contenido:  
Léxico y Sintaxis completo.

Se encuentra el scanner y parser en el mismo archivo. Por el momento, el input tiene que ser a través de línea de comandos. Queda pendiente agregar una función para leer input de archivos.  
Hay también 5 Warnings por conflictos de Shift/Reduce que serán corregidos para la entrega del Domingo 11 de Octubre.

## Avance #2

Contenido:  
Contiene al 100% el dirProc y tablaVars  
Implementación:  
- Se tiene un hash de las con las variables globales.  
- Una tabla con el nombre del proc y el índice en donde se encuentran las vars. Locales del mismo.  
- Y una lista de hashes para las variables locales de cada función.  
- Se le están asignando dir. virtuales de una vez, al principio del código viene la 'división' que se estará haciendo en la memoria. Ej. 1000 - 4999 variables globales int, etc.  
- También se tiene un hash de variables constantes.  
- Ya se tiene implementada una estructura para las variables temporales, pero aún no hay código para asignarle valores a esta estructura (hasta que se empiecen a crear los cuádruplos)    
Falta agregar que el compilador pueda leer archivos en lugar de input a través del teclado.

## Avance #3

Cubo Semántico completo

Se hicieron cambios en la implementación de las tablas de variables. Se añadieron clase: variables, directorios. Se separó el scanner y el parser, además ya se puede leer desde un archivo.  
Generación de cuádruplos para expresiones casi completo, falto generar cuádruplos para los operadores lógicos y de asignación '='  
Se corrigieron 4 de los 5 warnings que arrojaba PLY. El warning que queda se quedará así mientras que no cause problemas en la gramática.

## Avance #4

Cosas faltantes:  
Cuádruplo del FOR loop.  
Todo lo demás está completo. Se implementó una clase: cuádruplos y generador de cuádruplos.

## Avance #5

Contenido: cuádruplos de funciones.

Falta checar un detalle con las funciones void.

## Avance #6

Cuádruplos 100% completo  
Máquina virtual: trabajando en el mapa de memoria.

## Avance #7

Contenido: Incorporación de arreglos a los cuádruplos. Nuevos cuádruplos para verificar el índice del arreglo e inicializarlo.

## Avance #8

Contenido: Compilador 100% listo. Falta correr pruebas para verificar que todo funcione correctamente.

## Lecciones aprendidas – Fernando

Durante la construcción del compilador aprendí diversas cosas, no obstante entre las que más destacan se encuentran la complejidad de trabajar en un proyecto extenso. Por lo general, los proyectos que había realizado con anterioridad se realizaban en períodos cortos de tiempo, sin embargo el compilador requirió de esfuerzo semanal para poder tenerlo a tiempo. Uno de los mayores enfoques que se tuvo fue el dar un mensaje preciso del error de compilación/ejecución. Al hacer esto aprendí todo el trabajo que tiene que hacer el compilador detrás de escena para que pueda ser así de específico. Finalmente el haber creado la máquina virtual me dejó como aprendizaje un mejor manejo de apuntadores, dado a que constantemente teníamos que manipular accesos directos e indirectos a nuestra estructura de memoria.

## Lecciones aprendidas – Javier

A lo largo de este semestre y durante el desarrollo de este proyecto aprendí muchas cosas sobre la creación de un lenguaje y su compilador. Desde la definición del lenguaje y su sintaxis hasta la administración de la memoria. La mayoría de las cosas que vimos en clase y usamos en este proyecto son cosas que ya conocía, pero solo como una idea de cómo funcionaban. Al desarrollar este proyecto te das cuenta de todo lo que conlleva y todos los pequeños detalles que surgen en el desarrollo. Muchas cosas suenan relativamente simples como idea o concepto pero al momento de aplicarlas tienen muchas implicaciones que van complicando el desarrollo. Aprendí que la creación de un compilador para incluso un lenguaje tan básico como el nuestro necesita de mucho trabajo y dedicación. Como pudimos ver que cada cosa que le agregas complica tanto el desarrollo, me doy cuenta que hacer un proyecto como este para un lenguaje completo sería un trabajo de una gran magnitud.

# Nombre del lenguaje

Cheese++

# Descripción de sus características

Cheese ++ es un lenguaje de programación de alto nivel, imperativo y de muy fácil aprendizaje. Es por esto que se espera que el lenguaje sea utilizado mayormente por jóvenes y niños. Cheese ++ fue creado con la finalidad de usarlo para enseñar programación y puede usarse para enseñar la mayoría de los principales conceptos de programación: asignación de variables, flujos condicionales, flujos de repetición, manejo de I/O y listas simples.

El que sea un lenguaje de output gráfico, les permite a los jóvenes aprender de una manera más sencilla a través del trazado de líneas, arcos, figuras, y demás. Esto permite que el usuario obtenga una retroalimentación más directa cuando intente realizar operaciones más complicadas, como lo es la recursividad. El lenguaje soporta además cambios en el plumón, como su color, tamaño, etc.

# Descripción de errores

## Durante compilación

|  |  |  |
| --- | --- | --- |
| **Tipo** | **Descripción** | **Ejemplo** |
| Error de genérico de sintaxis | Error que ocurre cuando el scanner procesa un carácter no esperado por algún Token. | ERROR: Syntax error at '/' in line 4! |
| Uso de palabras reservadas | Error que ocurre cuando se usa una palabra reservada como nombre de variable o función. | ERROR: Syntax error at 'main' in line 4. Cannot use reserved word 'main' as a variable or function! |
| Variable no declarada – Asignación | Error que ocurre cuando se le asigna un valor a una variable no declarada dentro del alcance. | ERROR: Variable 'b' not declared in this scope, line 4! |
| Variable no declarada – Referencia | Error que ocurre cuando se hace referencia a una variable no existente dentro del alcance. | ERROR: Variable 'c' not declared in this scope! |
| Tipos incompatibles – Asignación | Error que ocurre cuando se le asigna un tipo distinto a los permitidos a cierta variable. | ERROR: Invalid types in line 4. Variable 'b' of type '<class 'int'>' cannot store '10.0'! |
| Tipos incompatibles – Operaciones lógicas | Error que ocurre cuando se intenta realizar una operación lógica entre dos variables que no son del tipo booleano. | ERROR: Operation '&&' between '<class 'bool'>' and '<class 'int'>' cannot be performed. Incompatible types! |
| Tipos incompatibles – Operaciones aritméticas | Error que ocurre cuando se intenta realizar una operación aritmética entre dos variables de dominio diferente. | ERROR: Operation '>' between '<class 'int'>' and '<class 'str'>' cannot be performed. Incompatible types! |
| Tipos incompatibles – Índice del arreglo | Error que ocurre cuando se intenta acceder a un índice de un arreglo con un valor distinto a un entero. | ERROR: Type error. Array index must be of type int, found <class 'str'> at line 5! |
| Tipos incompatibles – Condicional Flujos | Error que ocurre cuando la condicional que evaluará que flujo toma el programa no es de tipo booleano. (For, While, If) | ERROR: Expected type bool, but found <class 'int'> in FOR Loop! |
| Tipos incompatibles – Funciones específicas del lenguaje | Error que ocurre cuando se intenta pasar como parámetro un tipo de valor no permitido por la función predeterminada. | ERROR: Expected type int or double, but found <class 'str'> in line! |
| Llamadas a funciones – Falta de argumentos | Error que ocurre cuando se llama una función y se le envían menos argumentos de los requeridos. | ERROR: Function 'test' received 2 arguments, expected 3 in line 8! |
| Llamadas a funciones – Exceso de argumentos | Error que ocurre cuando se llama una función y se le envían más argumentos de los requeridos. | ERROR: Function 'test' received more arguments than expected! |
| Llamadas a funciones – No definidas | Error que ocurre cuando se llama una función que no ha sido definida en el código. | ERROR: Undeclared function: 'test2'! |
| Llamadas a funciones – Argumentos no compatibles | Error que ocurre cuando se llama una función sin embargo alguno de sus parámetros no es del tipo previamente definido. | ERROR: Incompatible arguments. Received '<class 'float'>', expected '<class 'int'>'! |
| Llamadas a funciones – Valor de retorno | Error que ocurre cuando se regresa un tipo de valor diferente al especificado en la firma de la función. | ERROR: Incompatible return types! Received <class 'float'>, expected <class 'int'>! |
| Redefinición - Variables | Error que ocurre cuando se redefine una variable que ya había sido declarada previamente. | ERROR: Variable 'a' already exists in current scope 'main cheese'! |
| Redefinición - Funciones | Error que ocurre cuando se redefine una función que ya había sido declarada previamente. | ERROR: Function 'uno' already exists in scope: 'global'! |
| Asignación – Arreglos | Error que ocurre cuando se intenta inicializar un arreglo con cierto valor. (Funcionalidad no soportada) | ERROR: Variable 'a' cannot be initialized since it is an array, in line 2! |

## Durante ejecución

|  |  |  |
| --- | --- | --- |
| Tipo | Descripción | Ejemplo |
| Arreglos – Índice | Error que ocurre cuando se intenta acceder a un índice | ERROR: Index out of bounds, cannot access index '10' of an array of size '5'! |
| División – Cero | Error que ocurre cuando se intenta realizar una división en donde el divisor es 0. | ERROR: Attempting to divide by 0! |
| Tipos incompatibles | Se vuelven a verificar todos los tipos nuevamente en la máquina virtual, para toda operación. (>, +, \*, etc).  NOTA: Se espera que el error se detecte en compilación. | TypeError: Operation invalid for specified operand types |
| ValueError | Error que ocurre cuando se tiene un número fuera del rango permitido en las operaciones específicas del lenguaje. Rangos de color: 0 - 255 | Value mismatch, expected an integer between 0 and 255! |

# Equipo de Cómputo, Lenguaje y Utilerías

## Equipo de Cómputo

ASUS Q550LF – Windows 10

## Lenguaje de programación

Python v3.5.0

## Utilerías

Python Lex & Yacc (PLY) v3.8

Turtle Graphics v.3.5.0 (ya contenido dentro del lenguaje)

# Análisis de Léxico

## Expresiones regulares principales

|  |  |
| --- | --- |
| **Tipo** | **Expresión regular** |
| ID | [a-zA-Z\_][a-zA-Z0-9\_]\* |
| Constante entera | [-]?[0-9]+ |
| Constante flotante | [-]?[0-9]+.[0-9]+ |
| Constante cadena de caracteres | ".\*?" |
| Comentarios | //.\* |

## Tokens

|  |  |
| --- | --- |
| **Token** | **Valor** |
| ID | [a-zA-Z\_][a-zA-Z0-9\_]\* |
| RBRACKET | } |
| LBRACKET | { |
| RSQUARE | ] |
| LSQUARE | [ |
| CGT | > |
| CLT | < |
| CGE | >= |
| CLE | <= |
| CEQ | == |
| CNE | != |
| CSTDOUBLE | [-]?[0-9]+.[0-9]+ |
| CSTSTRING | [-]?[0-9]+ |
| COMMA | , |
| SEMICOLON | ; |
| AND | && |
| OR | || |
| = | = |
| + | + |
| - | - |
| \* | \* |
| / | / |
| ( | ( |
| ) | ) |

## Palabras reservadas

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| * main | * cheese | * int | * double | * string | * If |
| * Else | * For | * While | * Scanf | * Print | * Move |
| * Rotate | * Arc | * Home | * Pdown | * Pup | * Setp |
| * Pcolor | * Psize | * Pclear | * Void | * Return | * func |

## Jerarquía

1. () Left
2. \*, / Left
3. +, - Left
4. >, <, <=, >=, ==, != Left

**Funciones Especiales:**

|  |  |  |
| --- | --- | --- |
| Nombre | Descripción | Ejemplo |
| scanf() | Permite la lectura del teclado. | scanf(varTemp); |
| print() | Permite imprimir texto a pantalla. | print(“hola”); |
| move() | Permite trazar líneas a través de un movimiento hacia adelante o atrás, de acuerdo al valor dado. | move(30); move(-50.5); |
| rotate() | Permite girar la dirección de la pluma en ciertos grados. Por default se rota a favor de las manecillas del reloj. | rotate(90); rotate(-180);  clockwise counterclockwise |
| arc() | Permite dibujar un arco con cierto radio, también se debe de proveer el ángulo que cubre el arco. | arc(radius, angle); |
| home() | Regresa a la pluma a las coordenadas (0,0) | home(); |
| pdown() | Baja la pluma para que se puedan trazar líneas y demás. Por default la pluma siempre esta abajo. | pdown(); |
| pup() | Levanta la pluma para que no se dibuje en la pantalla. | pup(); |
| setp() | Reposiciona el lugar de la pluma a unas nuevas coordenadas. | setp(x, y); |
| pcolor() | Cambia el color de la pluma. | pcolor(255, 0, 245); |
| psize() | Cambia el grosor de la pluma. | psize(10); |
| pclear() | Limpia la pantalla completamente. | pclear(); |

# Análisis de Sintaxis

Gramatica Formal:

PROGRAM -> PROGRAM2 PROGRAM3 MAIN

PROGRAM2 🡪 VARIABLE\_DECLARATION PROGRAM2

PROGRAM2 🡪 EMPTY

PROGRAM3 🡪 METHOD\_DECLARATION PROGRAM3

PROGRAM3 🡪 EMPTY

VARIABLE\_DECLARATION 🡪 TYPE VARIABLE\_DECLARATION2

VARIABLE\_DECLARATION2 🡪 VARIABLE\_DECLARATOR VARIABLE\_DECLARATION3

VARIABLE\_DECLARATION3 🡪 comma VARIABLE\_DECLARATION2

VARIABLE\_DECLARATION3 🡪 semicolon

VARIABLE\_DECLARATOR 🡪 id VARIABLE\_DECLARATOR2 VARIABLE\_DECLARATOR3

VARIABLE\_DECLARATOR2 🡪 lsquare cstint rsquare

VARIABLE\_DECLARATOR2 🡪 EMPTY

VARIABLE\_DECLARATOR3 🡪 equal SUPEREXPRESSION

VARIABLE\_DECLARATOR3 🡪 EMPTY

METHOD\_DECLARATION 🡪 func TYPE id lparen METHOD\_DECLARATION2 METHOD\_DECLARATION3

METHOD\_DECLARATION2 🡪 PARAMETER\_LIST

METHOD\_DECLARATION2 🡪 EMPTY

METHOD\_DECLARATION3 🡪 rparen BLOCK\_STATEMENT

PARAMETER\_LIST 🡪 PARAMETER PARAMETER\_LIST2

PARAMETER\_LIST2 🡪 comma PARAMETER\_LIST

PARAMETER\_LIST2 🡪 EMPTY

PARAMETER 🡪 TYPE id

MAIN 🡪 main cheese lparen rparen BLOCK\_STATEMENT

TYPE 🡪 int

TYPE 🡪 double

TYPE 🡪 string

TYPE 🡪 void

VARIABLE\_ASSIGNMENT 🡪 id ID\_OR\_ARRAY

ID\_OR\_ARRAY 🡪 equal SUPEREXPRESSION

ID\_OR\_ARRAY 🡪 lsquare SUPEREXPRESSION rsquare equal SUPEREXPRESSION

SUPEREXPRESSION 🡪 EXPRESSION SUPEREXPRESSION2

SUPEREXPRESSION2 🡪 ANDOR SUPEREXPRESSION

SUPEREXPRESSION2 🡪 EMPTY

ANDOR 🡪 and

ANDOR 🡪 or

EXPRESSION 🡪 NUMERIC\_EXPRESSION COMPARE

COMPARE 🡪 COMPARE\_SYMBOLS NUMERIC\_EXPRESSION

COMPARE 🡪 EMPTY

COMPARE\_SYMBOLS 🡪 <

COMPARE\_SYMBOLS 🡪 >

COMPARE\_SYMBOLS 🡪 <=

COMPARE\_SYMBOLS 🡪 >=

COMPARE\_SYMBOLS 🡪 ==

COMPARE\_SYMBOLS 🡪 !=

NUMERIC\_EXPRESSION 🡪 TERM NUMERIC\_EXPRESSION2

NUMERIC\_EXPRESSION2 🡪 ARITHOP NUMERIC\_EXPRESSION

NUMERIC\_EXPRESSION2 🡪 EMPTY

ARITHOP 🡪 +

ARITHOP 🡪 -

TERM 🡪 FACTOR TERM2

TERM2 🡪 GEOMOP TERM

TERM2 🡪 EMPTY

GEOMOP 🡪 \*

GEOMOP 🡪 /

FACTOR 🡪 CST\_EXPRESSION

FACTOR 🡪 lparen SUPEREXPRESSION rparen

CST\_EXPRESSION 🡪 cstdouble

CST\_EXPRESSION 🡪 cstint

CST\_EXPRESSION 🡪 cststring

CST\_EXPRESSION 🡪 id CST\_EXPRESSION2

CST\_EXPRESSION2 🡪 lsquare SUPEREXPRESSION rsquare

CST\_EXPRESSION2 🡪 FUNCTIONCALL

CST\_EXPRESSION2 🡪 EMPTY

FUNCTION\_CALL 🡪 lparen ARGS rparen

ARGS 🡪 ARG

ARGS 🡪 EMPTY

ARG 🡪 NUMERIC\_EXPRESSION MORE\_ARGS

MORE\_ARGS 🡪 comma ARG

MORE\_ARGS 🡪 EMPTY

LOOP\_STATEMENT 🡪 LOOPHEAD BLOCK\_STATEMENT

LOOPHEAD 🡪 for lparen VARIABLE\_ASSIGNMENT semicolon SUPEREXPRESSION semicolon VARIABLE\_ASSIGNMENT rparen

LOOPHEAD 🡪 while lparen SUPEREXPRESSION rparen

IF\_STATEMENT 🡪 if lparen SUPEREXPRESSION rparen BLOCK\_STATEMENT IF\_STATEMENT2

IF\_STATEMENT2 🡪 else BLOCK\_STATEMENT

IF\_STATEMENT2 🡪 EMPTY

BLOCK\_STATEMENT 🡪 lbracket BLOCK\_STATEMENT2 rbracket

BLOCK\_STATEMENT2 🡪 STATEMENT BLOCK\_STATEMENT2

BLOCK\_STATEMENT2 🡪 EMPTY

READ\_STATEMENT 🡪 scan lparen id rparen semicolon

PRINT\_STATEMENT 🡪 print lparen SUPEREXPRESSION rparen semicolon

GEOMETRY\_STATEMENT 🡪 MOVE

GEOMETRY\_STATEMENT 🡪 ROTATE

GEOMETRY\_STATEMENT 🡪 ARC

GEOMETRY\_STATEMENT 🡪 HOME

GEOMETRY\_STATEMENT 🡪 PDOWN

GEOMETRY\_STATEMENT 🡪 PUP

GEOMETRY\_STATEMENT 🡪 SETP

GEOMETRY\_STATEMENT 🡪 PCOLOR

GEOMETRY\_STATEMENT 🡪 PSIZE

GEOMETRY\_STATEMENT 🡪 PCLEAR

MOVE 🡪 move lparen SUPEREXPRESSION rparen

ROTATE 🡪 rotate lparen SUPEREXPRESSION rparen

ARC 🡪 arc lparen SUPEREXPRESSION comma SUPEREXPRESSION rparen

HOME 🡪 home lparen rparen

PDOWN 🡪 pdown lparen rparen

PUP 🡪 pup lparen rparen

SETP 🡪 setp lparen SUPEREXPRESSION comma SUPEREXPRESSION rparen

PCOLOR 🡪 pcolor lparen SUPEREXPRESSION comma SUPEREXPRESSION comma SUPEREXPRESSION rparen

PSIZE 🡪 psize lparen SUPEREXPRESSION rparen

PCLEAR 🡪 pclear lparen rparen

STATEMENT 🡪 VARIABLE\_DECLARATION

STATEMENT 🡪 VARIABLE\_ASSIGNMENT semicolon

STATEMENT 🡪 SUPEREXPRESSION semicolon

STATEMENT 🡪 IF\_STATEMENT

STATEMENT 🡪 LOOP\_STATEMENT

STATEMENT 🡪 RETURN

STATEMENT 🡪 PRINT\_STATEMENT

STATEMENT 🡪 READ\_STATEMENT

STATEMENT 🡪 GEOMETRY\_STATEMENT semicolon

RETURN 🡪 return SUPEREXPRESSION semicolon

# Análisis Semántico y Generación de Código Intermedio

## Cuádruplos generados

### Asignación

**Cuádruplo de asignación**

EQU op1 op2 resultado

### Aritméticas

**Cuádruplo para la suma**

ADD op1 op2 resultado

**Cuádruplo para la resta**

SUB op1 op2 resultado

**Cuádruplo para la multiplicación**

MUL op1 op2 resultado

**Cuádruplo para la división**

DIV op1 op2 resultado

### Lógicas

**Cuádruplo para checar si es igual**

CEQ op1 op2 resultado

**Cuádruplo para checar no es igual**

CNE op1 op2 resultado

**Cuádruplo para checar menor que**

CLT op1 op2 resultado

**Cuádruplo para checar menor o igual**

CLE op1 op2 resultado

**Cuádruplo para checar mayor que**

CGT op1 op2 resultado

**Cuádruplo para checar mayor o igual**

CGE op1 op2 resultado

### Booleanas

**Cuádruplo para checar &&**

AND op1 op2 resultado

**Cuádruplo para checar ||**

ORR op1 op2 resultado

### Brincos y funciones

**Cuádruplo para realizar un salto**

GTO op1 op2 resultado

**Cuádruplo para realizar un salto en falso**

GTF op1 op2 resultado

**Cuádruplo para generar la función local**

ERA op1 op2 resultado

**Cuádruplo para realizar un salto a la función**

CAL op1 op2 resultado

**Cuádruplo para regresar al PC anterior**

RET op1 op2 resultado

### Operaciones específicas del lenguaje

**Cuádruplo para imprimir en pantalla**

PRT op1 op2 resultado

**Cuádruplo para leer del teclado**

RED op1 op2 resultado

**Cuádruplo para mover el plumón**

MVT op1 op2 resultado

**Cuádruplo para dibujar un arco**

ARC op1 op2 resultado

**Cuádruplo para regresar el plumón al origen**

HOM op1 op2 resultado

**Cuádruplo para poner el plumón a pintar**

PDO op1 op2 resultado

**Cuádruplo para quitar el plumón**

PUP op1 op2 resultado

**Cuádruplo para reposicionar el plumón**

SET op1 op2 resultado

**Cuádruplo para cambiar el color del plumón**

PCO op1 op2 resultado

**Cuádruplo para cambiar el tamaño del plumón**

PSZ op1 op2 resultado

**Cuádruplo para limpiar la pantalla**

PCL op1 op2 resultado

### Arreglos

**Cuádruplo para inicializar el arreglo en 0s**

INI op1 op2 resultado

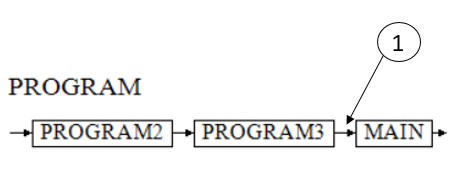
**Cuádruplo para calcular el offset + dirBase**

OFF op1 op2 resultado

**Cuádruplo para verificar índice del arreglo**

VER op1 op2 resultado

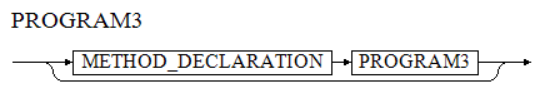
## Diagramas de Sintaxis

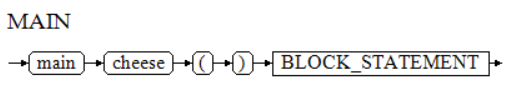


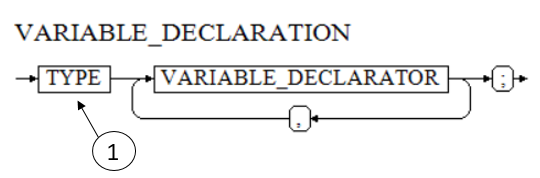
1. Fijar todos los saltos pendientes a la siguiente instrucción.

Agregar directorio ‘main cheese’ y cambiar a ese ámbito.

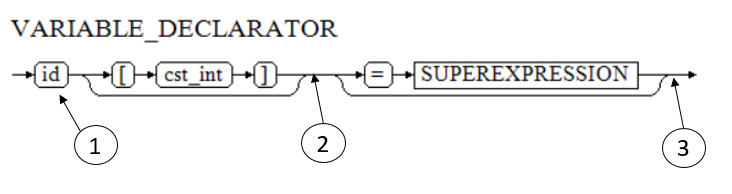








1. Guardar el tipo en la variable ‘seenType’.



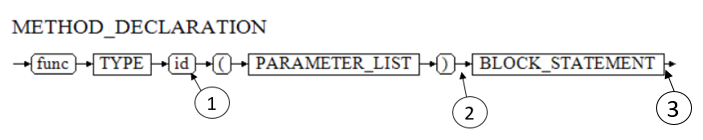
1. Guardar la variable temporalmente (Hasta saber si tendrá dimensiones).
2. Sacar la variable guardada, checar si la variable tenía dimensiones.

Agregar variable al directorio actual (Variable, tipo, dimensión).

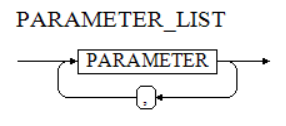
Si fue arreglo, generar cuádruplo: [INI, Variable, dimensión, 0] (Inicializa el arreglo en 0s).

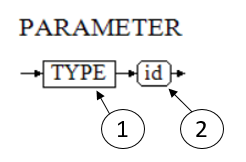
1. Si no es arreglo:

Pop de la pida de operando, checar tipos, generar cuádruplo [EQU, op1, 0, variable]

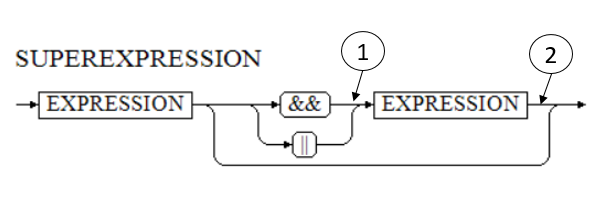


1. Registrar la nueva función y su tipo.
2. Guardar dirección de inicio de la función.
3. Generar cuádruplo [RET, 0, 0, 0] y cambiar al directorio global.

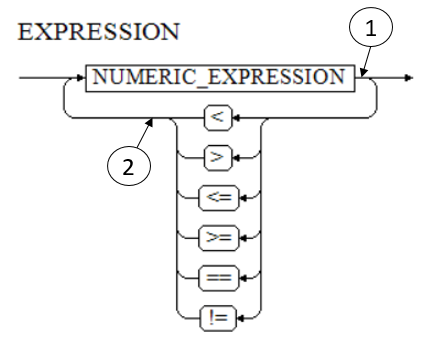




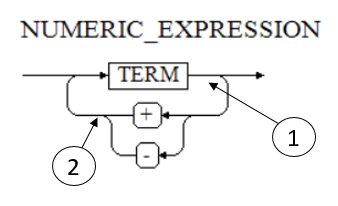
1. Guardar el tipo en la variable ‘seenType’.
2. Agregar al id como parámetro y variable de la función.



1. Meter el operador en la pila de Operadores.
2. Checar tope de la pila de Operadores, si es && o ||, obtener cada uno de los operandos haciendo pop de la pila de Operandos. Checar tipos, pedir temporal y generar cuádruplo: [AND/ORR, op1, op2, result].



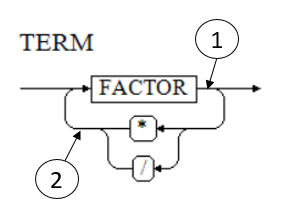
1. Checar tope de la pila de Operadores, si es <, >, <=, >=, == o !=, obtener cada uno de los operandos haciendo pop de la pila de Operandos. Checar tipos, pedir temporal y generar cuádruplo: [CGE/CGT/CLE/CLT/CEQ/CNE, op1, op2, result].
2. Meter el operador en la pila de Operadores.



1. Checar tope de la pila de Operadores, si es + o -, obtener cada uno de los operandos haciendo pop de la pila de Operandos. Checar tipos, pedir temporal y generar cuádruplo:

[ADD/SUB, op1, op2, result].

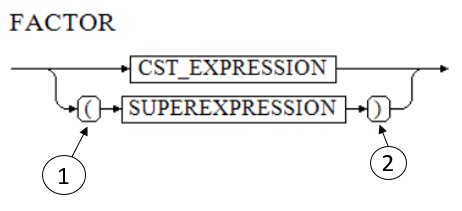
1. Meter el operador en la pila de Operadores.



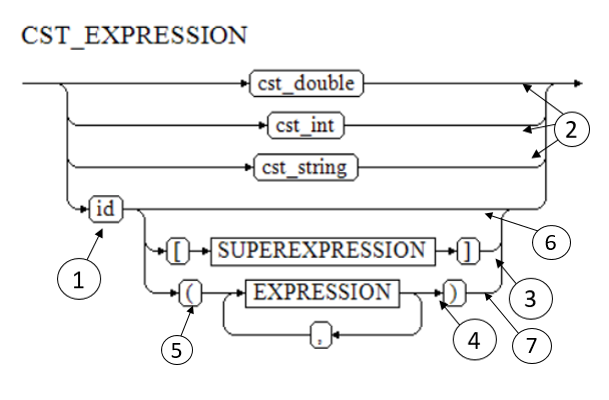
1. Checar tope de la pila de Operadores, si es \* o /, obtener cada uno de los operandos haciendo pop de la pila de Operandos. Checar tipos, pedir temporal y generar cuádruplo:

[ADD/SUB, op1, op2, result].

1. Meter el operador en la pila de Operadores.



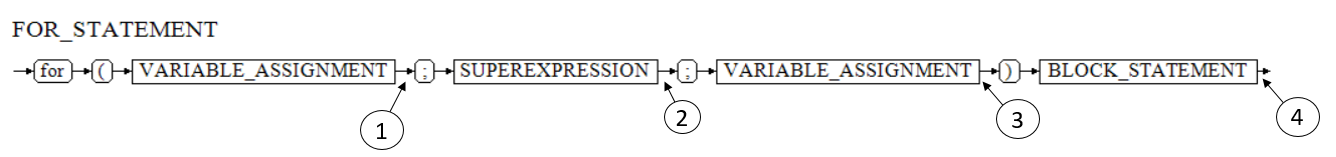
1. Meter el operador ‘(‘ en la pila de Operadores.
2. Pop del ‘(‘ de la pila de Operadores.



1. Guardar la variable temporalmente y verificar que este previamente definida (Hasta saber si tendrá dimensiones o es una llamada).
2. Agregar constante si es que no existe. Meter constante en la pila de Operandos.
3. Popear el índice de la pila de Operandos, checar que sea entero, generar cuádruplo:

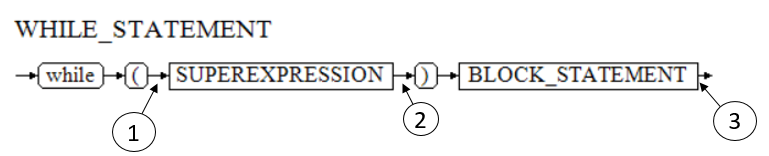
[VER, índice, dimensión, 0] y [OFF, dirección, índice, resultado]. Meter resultado en la pila de Operandos.

1. Para cada argumento, checar que sea del mismo tipo que el declarado previamente. Generar cuádruplo [EQU, variable, 0, param] para cada uno.
2. Checar que el directorio exista, generar cuádruplo: [ERA, 7050, 0, 0].
3. Meter el id normal en la pila de Operandos.
4. Generar cuádruplo: [CAL, 0, 0, dirInicio].

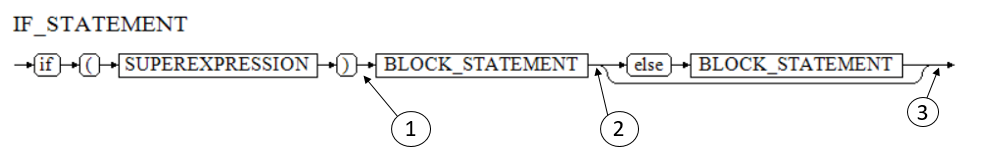


1. Meter el número de la siguiente instrucción en la pila de Saltos.
2. Sacar el resultado de la pila de Operandos. Crear dos brincos: [GTF, condición, 0, 0] y [GTO, 0, 0, 0]. Brinco de salida y brinco para el inicio del ciclo.
3. Generar brinco al cuádruplo con la condición [GTO, 0, 0, jump]. Asignar la dirección a la que puede saltar (inicio del ciclo for).
4. Completar el brinco pendiente, dado a que ya se sabe dónde termina el ciclo.

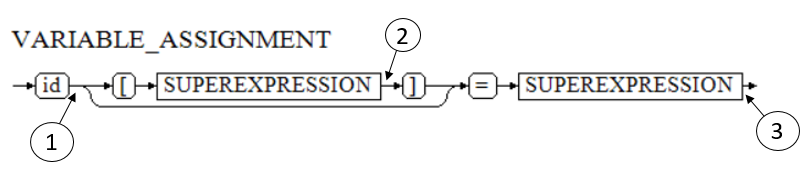
Generar [GTO, dirInicioLoop, 0, 0]



1. Meter el número de la siguiente instrucción en la pila de Saltos.
2. Sacar el resultado de la condición de la pila de Operandos. Generar el cuádruplo: [GTF, condition, 0, 0]
3. Completar el brinco pendiente, dado a que ya se sabe dónde termina el ciclo. Generar [GTO, dirInicioLoop, 0, 0]



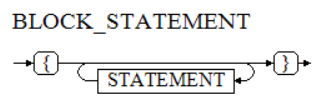
1. Obtener el resultado de la expresión de la pila de Operandos, generar el cuádruplo: [GTF, condición, 0, 0]. Guardar instrucción donde se checa la condición.
2. Generar cuádruplo: [GTO, 0, 0, 0]. Completar el brinco del cuádruplo GTF.
3. Completar el brinco GTO (brinco para saltarse el else).

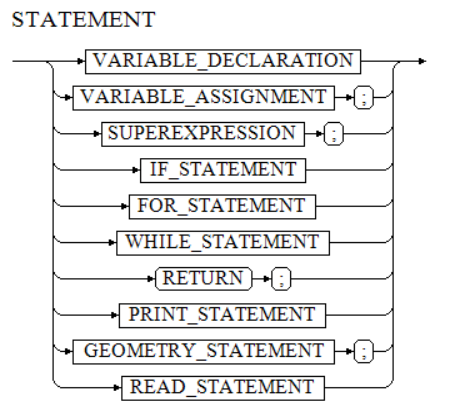


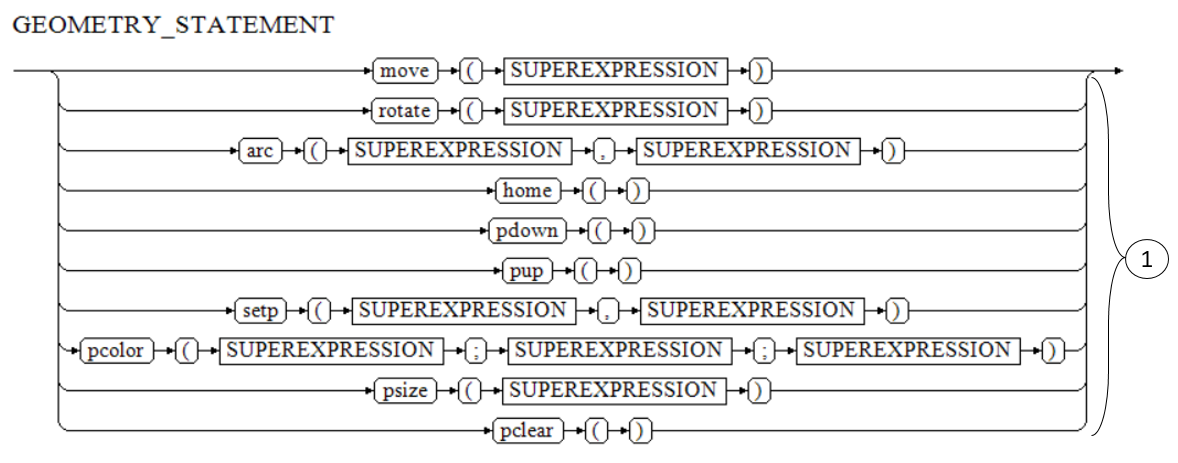
1. Guardar la variable temporalmente y verificar que haya sido declarada previamente (Hasta saber si tendrá dimensiones).

Checar que el resultado de la expresión es entero, generar cuádruplo: [VER, índice, dimensión, 0] y [OFF, dirección, índice, resultado].

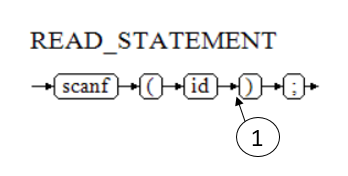
1. Sacar el valor del operando de la pila de Operandos, checar tipo con el ID. Generar cuádruplo: [EQU, op1, 0, result]



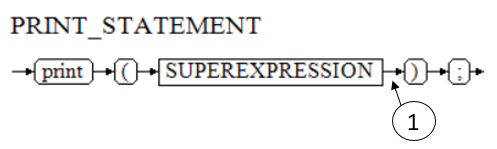




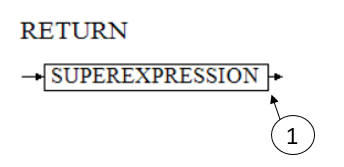
1. Para cada función predefinida, validar que el resultado de la expresión sea entera o flotante, según sea el caso. Generar cuádruplo: [FUNC, op1, op2, op3].



1. Checar que el id este definido previamente, generar cuádruplo: [RED, op1, 0, 0]



1. Sacar el resultado de la expresión de la pila de Operandos, generar el cuádruplo: [PRT, op1, 0, 0].



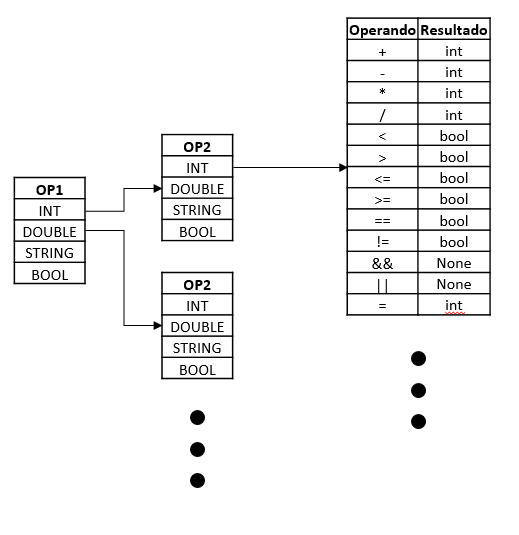
1. Sacar el resultado de la expresión de la pila de Operandos, checar tipos y generar cuádruplo: [RET, var, 0, 0].

## Cubo Semántico

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Operando 1 | Operando 2 | +, - | \*, / | >, <, <=, >=, ==, != | = (op1 = op2) | &&, || |
| INT | INT | INT | INT | BOOLEAN | INT | X |
| INT | DOUBLE | DOUBLE | DOUBLE | BOOLEAN | X |  |
| INT | STRING | X | X | X | X | X |
| INT | BOOLEAN | X | X | X | X | X |
| DOUBLE | INT | DOUBLE | DOUBLE | BOOLEAN | DOUBLE | X |
| DOUBLE | DOUBLE | DOUBLE | DOUBLE | BOOLEAN | DOUBLE | X |
| DOUBLE | STRING | X | X | X | X | X |
| DOUBLE | BOOLEAN | X | X | X | X | X |
| STRING | INT | X | X | X | X | X |
| STRING | DOUBLE | X | X | X | X | X |
| STRING | STRING | STRING | X | X | STRING | X |
| STRING | BOOLEAN | X | X | X | X | X |
| BOOLEAN | INT | X | X | X | X | X |
| BOOLEAN | DOUBLE | X | X | X | X | X |
| BOOLEAN | STRING | X | X | X | X | X |
| BOOLEAN | BOOLEAN | X | X | X | BOOLEAN | BOOLEAN |

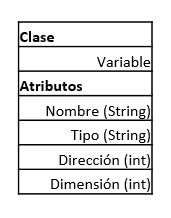
# Administración de Memoria - Compilación

## Cubo Semántico

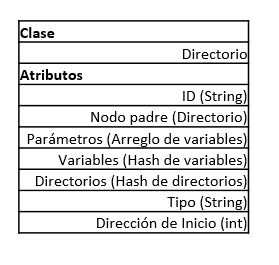


Es un triple diccionario (hash), en donde la llave completa es OP1, OP2, Operando, y como resultado regresa el tipo de valor que espera de la operación. None representa todos los errores.

## Clase variable

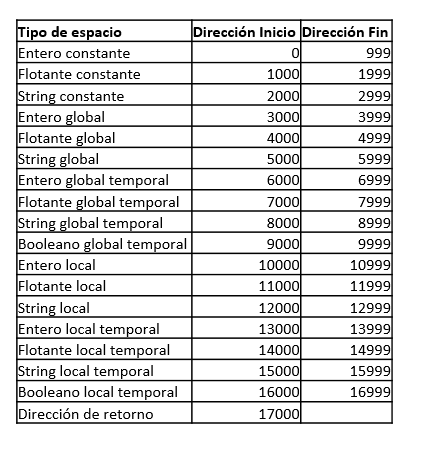


## Clase directorio

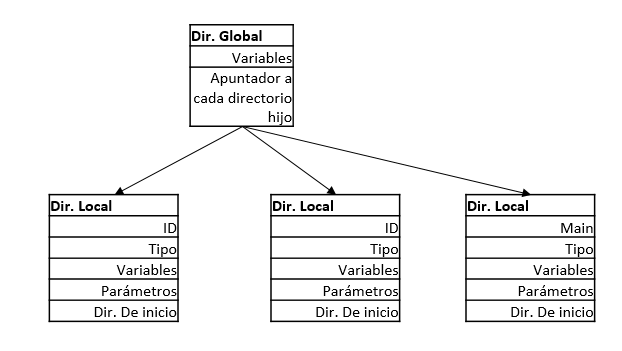


La clase además contiene métodos para añadir variables normales, temporales, constantes, calcular la siguiente dirección de memoria virtual disponible, agregar directorio, etc.

## Memoria Virtual

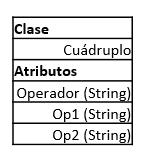


## Directorio Actual

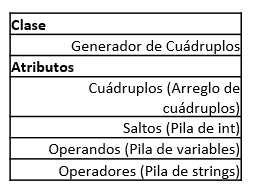


Estructura que se crea al ligar varios directorios juntos. El ‘main’ es considerado también como una función local.

## Cuádruplo



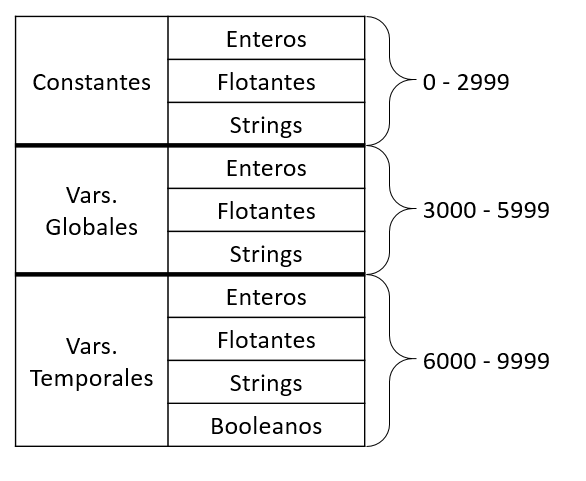
## Generador de Cuádruplos



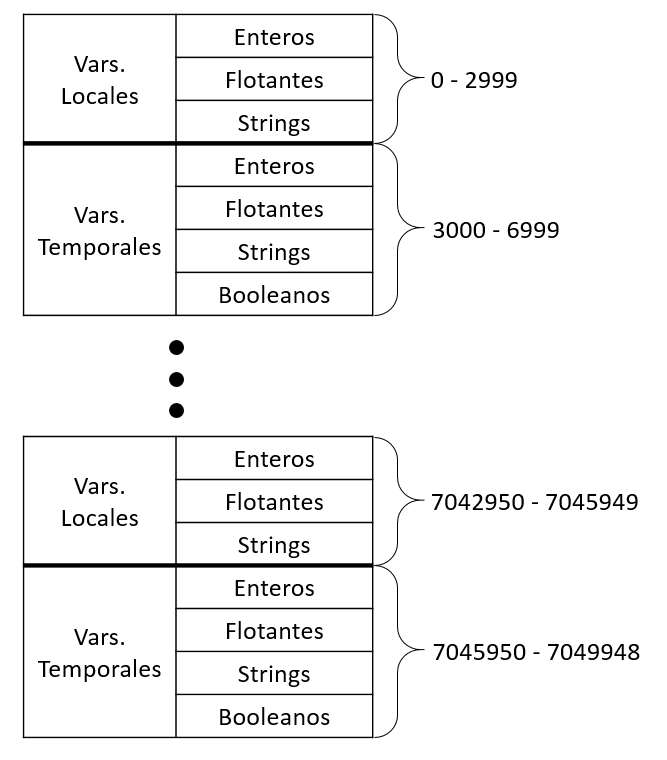
La clase además contiene métodos para agregar/sacar operadores, saltos, operandos, generar cuádruplos, completar cuádruplos de los brincos, inicializar los cuádruplos con un salto pendiente al ‘main’ y generar cuádruplos con las variables constantes.

# Administración de Memoria - Ejecución

## Memoria Global



Memoria Local



## Traducción direcciones Virtuales 🡪 Reales

Forma en la que se extraen los valores de las variables, según su dirección de memoria:

If OP1 == 17000:

OP1 = returnStack.pop()

Else if OP1 >= 10000:

OP1 = (OP1 – 10000) + segmentLength\*(stackPointer)

OP1 = localMemory[OP1]

Else if OP1 < 10000:

OP1 = globalMemory[OP1]

En donde, segmentLength es el tamaño de cada función (predeterminado en 7050). StackPointer apunta al bloque de memoria local en el que se está actualmente trabajando (El main reside en el bloque 0). Recordar que la dir. 17000 es la del valor de retorno.

# Pruebas

## Factorial Iterativo

### Código en Cheese ++

//Función que regresa el factorial de un número

//calculado de forma iterativa.

func int iterativeFactorial(int n) {

int i;

int ans = 1;

if(n<0) {

return -1;

}

for(i = 1; i <= n; i = i + 1) {

ans = ans \* i;

}

return ans;

}

main cheese() {

print (iterativeFactorial(7));

}

### Código Intermedio

CST -1 0 2

CST 1 0 1

CST 0 0 0

CST 7 0 3

GTO 0 0 18

EQU 1 0 10002

CLT 10000 0 16000

GTF 16000 0 6

RET 2 0 0

GTO 0 0 6

EQU 1 0 10001

CLE 10001 10000 16001

GTF 16001 0 16

GTO 0 0 13

ADD 10001 1 13000

EQU 13000 0 10001

GTO 0 0 7

MUL 10002 10001 13001

EQU 13001 0 10002

GTO 0 0 10

RET 10002 0 0

RET 0 0 0

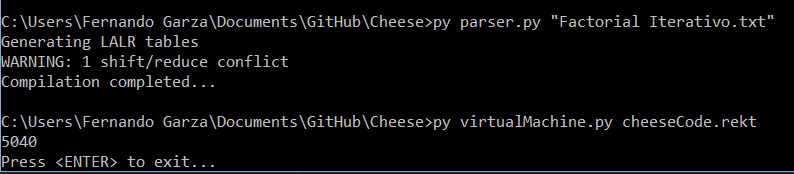
ERA 7050 0 0

EQU 3 0 10000

CAL 0 0 1

PRT 17000 0 0

### Resultado



## Factorial Recursivo

### Código en Cheese ++

//Función que regresa el factorial de un número

//calculado de forma recursiva

func int recursiveFactorial(int n) {

if(n<0) {

return -1;

}

if(n==0) {

return 1;

}

return n\*recursiveFactorial(n - 1);

}

main cheese() {

print (recursiveFactorial(7));

}

### Código Intermedio

CST 0 0 0

CST 7 0 3

CST 1 0 2

CST -1 0 1

GTO 0 0 16

CLT 10000 0 16000

GTF 16000 0 5

RET 1 0 0

GTO 0 0 5

CEQ 10000 0 16001

GTF 16001 0 9

RET 2 0 0

GTO 0 0 9

ERA 7050 0 0

SUB 10000 2 13000

EQU 13000 0 10000

CAL 0 0 1

MUL 10000 17000 13001

RET 13001 0 0

RET 0 0 0

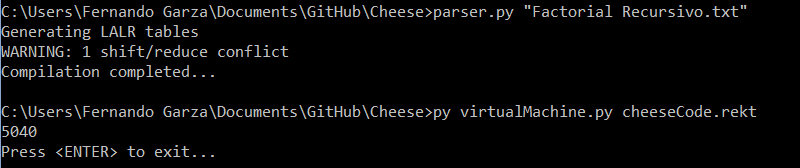
ERA 7050 0 0

EQU 3 0 10000

CAL 0 0 1

PRT 17000 0 0

### Resultado



## Fibonacci

### Código en Cheese ++

//Funcion que imprime los primeros 'n'

//valores de la secuencia de fibonacci

func void fibonacci(int n){

int i, a, b;

a = 0; b = 1;

for(i = 0; i < n; i = i + 1) {

print(b);

b = a + b;

a = b - a;

}

}

main cheese() {

fibonacci(7);

}

### Código Intermedio

CST 0 0 0

CST 7 0 2

CST 1 0 1

GTO 0 0 17

EQU 0 0 10002

EQU 1 0 10003

EQU 0 0 10001

CLT 10001 10000 16000

GTF 16000 0 16

GTO 0 0 10

ADD 10001 1 13000

EQU 13000 0 10001

GTO 0 0 4

PRT 10003 0 0

ADD 10002 10003 13001

EQU 13001 0 10003

SUB 10003 10002 13002

EQU 13002 0 10002

GTO 0 0 7

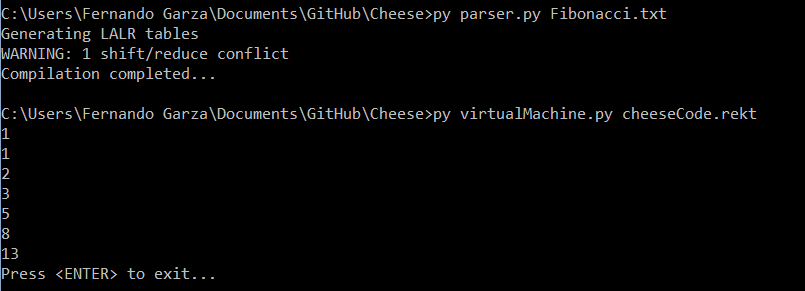
RET 0 0 0

ERA 7050 0 0

EQU 2 0 10000

CAL 0 0 1

### Resultado



## Bubble Sort

### Código en Cheese ++

int myArr [100];

//función que ordena los primeros 'n' números

//de un arreglo de enteros GLOBAL

//empezando desde la posición 'k'

//llamado myArr, de menor a mayor

func void sort(int k, int n){

int i, j, tmp;

for(i = k; i < (k + n - 1); i = i + 1) {

for(j = i+1; j < (k+n); j = j + 1) {

if(myArr[j]<myArr[i]){

tmp = myArr[i];

myArr[i] = myArr[j];

myArr[j]=tmp;

}

}

}

}

main cheese() {

int i;

//preparar el arreglo con los numeros descendiendo de 99 a 0

for(i = 0; i < 100; i = i + 1) {

myArr[99-i]=i;

// 99 98 97 96 95 94 93 92 91 90...

}

sort(0, 6);

//94 95 96 97 98 99 93 92 91 90...

sort(4, 6);

//94 95 96 97 90 91 92 93 98 99

for(i = 0; i < 100; i = i + 1) {

print(myArr[i]);

}

}

### Código Intermedio

CST 99 0 3

CST 100 0 2

CST 4 0 5

CST 6 0 4

CST 1 0 1

CST 0 0 0

GTO 0 0 40

EQU 10000 0 10002

ADD 10000 10001 13000

SUB 13000 1 13001

CLT 10002 13001 16000

GTF 16000 0 39

GTO 0 0 10

ADD 10002 1 13002

EQU 13002 0 10002

GTO 0 0 2

ADD 10002 1 13003

EQU 13003 0 10003

ADD 10000 10001 13004

CLT 10003 13004 16001

GTF 16001 0 38

GTO 0 0 19

ADD 10003 1 13005

EQU 13005 0 10003

GTO 0 0 12

VER 10003 100 3000

OFF 3000 10003 13006

VER 10002 100 3000

OFF 3000 10002 13007

CLT (13006) (13007) 16002

GTF 16002 0 37

VER 10002 100 3000

OFF 3000 10002 13008

EQU (13008) 0 10004

VER 10003 100 3000

OFF 3000 10003 13009

VER 10002 100 3000

OFF 3000 10002 13010

EQU (13009) 0 (13010)

VER 10003 100 3000

OFF 3000 10003 13011

EQU 10004 0 (13011)

GTO 0 0 37

GTO 0 0 16

GTO 0 0 7

RET 0 0 0

EQU 0 0 10000

CLT 10000 2 16000

GTF 16000 0 52

GTO 0 0 47

ADD 10000 1 13000

EQU 13000 0 10000

GTO 0 0 41

SUB 3 10000 13001

VER 13001 100 3000

OFF 3000 13001 13002

EQU 10000 0 (13002)

GTO 0 0 44

ERA 7050 0 0

EQU 0 0 10000

EQU 4 0 10001

CAL 0 0 1

ERA 7050 0 0

EQU 5 0 10000

EQU 4 0 10001

CAL 0 0 1

EQU 0 0 10000

CLT 10000 2 16001

GTF 16001 0 71

GTO 0 0 67

ADD 10000 1 13003

EQU 13003 0 10000

GTO 0 0 61

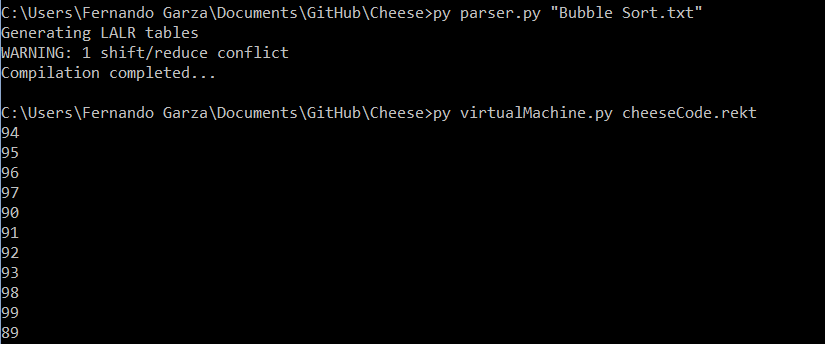
VER 10000 100 3000

OFF 3000 10000 13004

PRT (13004) 0 0

GTO 0 0 64

### Resultado



## Search

### Código en Cheese ++

int arr[10];

//función que que regresa el indice de la

//primera aparición de un numero 'n'

//en los primeros k numeros de un arreglo

//regresa -1 si no se encuentra

func int search(int n, int k){

int i;

for(i = 0; i < k; i = i + 1) {

if(arr[i] == n){

return i;

}

}

return -1;

}

main cheese() {

arr[5] = 1;

print(search(1, 7));

}

### Código Intermedio

CST 5 0 3

CST -1 0 2

CST 0 0 0

CST 1 0 1

CST 7 0 4

GTO 0 0 17

EQU 0 0 10002

CLT 10002 10001 16000

GTF 16000 0 15

GTO 0 0 8

ADD 10002 1 13000

EQU 13000 0 10002

GTO 0 0 2

VER 10002 10 3000

OFF 3000 10002 13001

CEQ (13001) 10000 16001

GTF 16001 0 14

RET 10002 0 0

GTO 0 0 14

GTO 0 0 5

RET 2 0 0

RET 0 0 0

VER 3 10 3000

OFF 3000 3 13000

EQU 1 0 (13000)

ERA 7050 0 0

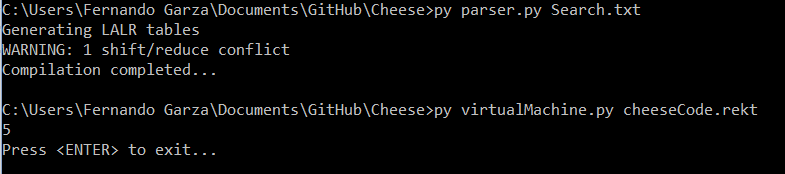
EQU 1 0 10000

EQU 4 0 10001

CAL 0 0 1

PRT 17000 0 0

### Resultado



## Dibujo – Sierpinski

### Código en Cheese ++

func void sierpinski (double length, int depth) {

if (depth == 0) {

int i;

for(i = 0; i < 3; i = i + 1) {

move(length);

rotate(-120);

}

}

else {

sierpinski(length/2, depth - 1);

move(length/2);

sierpinski(length/2, depth - 1);

move(length/-2);

rotate(-60);

move(length/2);

rotate(60);

sierpinski(length/2, depth - 1);

rotate(-60);

move(length/-2);

rotate(60);

}

}

main cheese() {

pup();

move(-100);

pdown();

sierpinski(300, 4);

}

### Código Intermedio

CST -100 0 8

CST 60 0 7

CST -2 0 5

CST -60 0 6

CST -120 0 3

CST 3 0 1

CST 1 0 2

CST 0 0 0

CST 300 0 9

CST 4 0 10

CST 2 0 4

GTO 0 0 45

CEQ 10000 0 16000

GTF 16000 0 14

EQU 0 0 10001

CLT 10001 1 16001

GTF 16001 0 13

GTO 0 0 10

ADD 10001 2 13000

EQU 13000 0 10001

GTO 0 0 4

MVT 11000 0 0

ROT 3 0 0

GTO 0 0 7

GTO 0 0 44

ERA 7050 0 0

DIV 11000 4 14000

EQU 14000 0 11000

SUB 10000 2 13001

EQU 13001 0 10000

CAL 0 0 1

DIV 11000 4 14001

MVT 14001 0 0

ERA 7050 0 0

DIV 11000 4 14002

EQU 14002 0 11000

SUB 10000 2 13002

EQU 13002 0 10000

CAL 0 0 1

DIV 11000 5 14003

MVT 14003 0 0

ROT 6 0 0

DIV 11000 4 14004

MVT 14004 0 0

ROT 7 0 0

ERA 7050 0 0

DIV 11000 4 14005

EQU 14005 0 11000

SUB 10000 2 13003

EQU 13003 0 10000

CAL 0 0 1

ROT 6 0 0

DIV 11000 5 14006

MVT 14006 0 0

ROT 7 0 0

RET 0 0 0

PUP 0 0 0

MVT 8 0 0

PDO 0 0 0

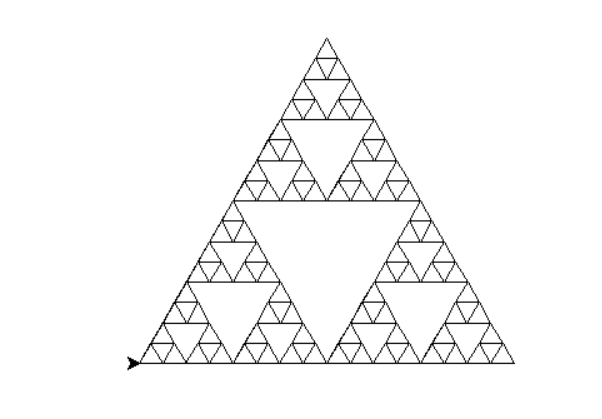
ERA 7050 0 0

EQU 9 0 11000

EQU 10 0 10000

CAL 0 0 1

### Resultado



# Listados

## Declaración Arreglo

#Código del parser

def p\_variable\_declarator(p):

'''variable\_declarator : ID variable\_declarator2 variable\_declarator3'''

#Regla inicial con la que se declaran variables

def p\_variable\_declarator2(p):

'''variable\_declarator2 : LSQUARE CSTINT RSQUARE

| empty'''

#Regla en la que se identifica si la variable es un arreglo o no.

global currentDirectory, instructions

#Se revisa si p[1] contiene algun valor, si tiene uno es arreglo.

if p[1]:

#Se agrega la variable al directorio actual, del tipo previamente definido

#y con una dimension especificada.

currentDirectory.add\_variable(p[-2], seenType, p[2])

#Se obtiene la variable del directorio,

#sirve para verificar que se guardo correctamente y ademas acceder a ciertos valores.

variable = currentDirectory.get\_variable(p[-2])

#Se genera el cuadruplo INI que inicializa el arreglo ya sea en 0s o "" segun sea el tipo.

if variable.Type is int:

instructions.generateQuadruple('INI', variable.Address, variable.Dimension, 1)

elif variable.Type is float:

instructions.generateQuadruple('INI', variable.Address, variable.Dimension, 2)

elif variable.Type is str:

instructions.generateQuadruple('INI', variable.Address, variable.Dimension, 3)

else:

#Si la variable no fue arreglo, agregarse como una variable normal de dimension 0

currentDirectory.add\_variable(p[-2], seenType, 0)

#Código de la máquina virtual

#Metodo: INI

#Parámetros de entrada: op1, op2, result.

#op1: dirección de la variable

#op2: dimensión de la variable (tam máximo)

#result: código para saber el tipo de la variable

#Output: inicializa el arreglo en 0s o "".

def INI(op1, op2, result):

#Parsear los valores de op1 y op2 a int.

op2 = int(op2)

op1 = int(op1)

#Checar si la dirección de la variable es local o global.

#Las variables globales se encuentran < 10,000.

if op1 >= inferiorLimit:

#Dado a que es una variable local, calcular la dirección real.

op1 = (op1 - inferiorLimit) + segmentLength\*stackPointer

#Inicializar el arreglo.

for i in range(0, op2):

if result == 3:

methodsMemory[op1+i] = ""

else:

methodsMemory[op1+i] = 0

elif op1 < inferiorLimit:

#Dado a que es una variable global, no es necesario hacer ningún cambio

#para obtener su dirección real.

for i in range(0, op2):

if result == 3:

globalMemory[op1+i] = ""

else:

globalMemory[op1+i] = 0

## Cubo Semántico

#Método: Cubo semántico

#Estructura del cubo semántico: triple hash con llave op1, op2, operador

#Parámetros de entrada: op1, op2, operator

#op1: primer operando de la operación.

#op2: segundo operando de la operación.

#operator: operador a aplicar a ambos operandos.

#Output: int, float, str, bool, None

#Si regresa None, implica que la operación no es válida.

#También regresa error si la combinación de llaves no fue encontrada.

#Esta estructura se utiliza siempre que se tiene que verificar el tipo resultante de una operación.

#Sirve para validar si una operación es permitida, de acuerdo al tipo de los operandos.

#Sirve para determinar el tipo del temporal que se tendrá que pedir al AVAIL.

#Se usa a lo largo del parser.

def getResultingType(operator, op1, op2):

semanticCube = {

int: {

int: {

'+' : int,

'-' : int,

'\*' : int,

'/' : int,

'<' : bool,

'<=' : bool,

'>' : bool,

'>=' : bool,

'==' : bool,

'!=' : bool,

'&&' : None,

'||' : None,

'=' : int

},

float: {

'+' : float,

'-' : float,

'\*' : float,

'/' : float,

'<' : bool,

'<=' : bool,

'>' : bool,

'>=' : bool,

'==' : bool,

'!=' : bool,

'&&' : None,

'||' : None,

'=' : None

},

str: {

'+' : None,

'-' : None,

'\*' : None,

'/' : None,

'<' : None,

'<=' : None,

'>' : None,

'>=' : None,

'==' : None,

'!=' : None,

'&&' : None,

'||' : None,

'=' : None

},

bool: {

'+' : None,

'-' : None,

'\*' : None,

'/' : None,

'<' : None,

'<=' : None,

'>' : None,

'>=' : None,

'==' : None,

'!=' : None,

'&&' : None,

'||' : None,

'=' : None

}

},

float: {

int: {

'+' : float,

'-' : float,

'\*' : float,

'/' : float,

'<' : bool,

'<=' : bool,

'>' : bool,

'>=' : bool,

'==' : bool,

'!=' : bool,

'&&' : None,

'||' : None,

'=' : float

},

float: {

'+' : float,

'-' : float,

'\*' : float,

'/' : float,

'<' : bool,

'<=' : bool,

'>' : bool,

'>=' : bool,

'==' : bool,

'!=' : bool,

'&&' : None,

'||' : None,

'=' : float

},

str: {

'+' : None,

'-' : None,

'\*' : None,

'/' : None,

'<' : None,

'<=' : None,

'>' : None,

'>=' : None,

'==' : None,

'!=' : None,

'&&' : None,

'||' : None,

'=' : None

},

bool: {

'+' : None,

'-' : None,

'\*' : None,

'/' : None,

'<' : None,

'<=' : None,

'>' : None,

'>=' : None,

'==' : None,

'!=' : None,

'&&' : None,

'||' : None,

'=' : None

}

},

str: {

int: {

'+' : None,

'-' : None,

'\*' : None,

'/' : None,

'<' : None,

'<=' : None,

'>' : None,

'>=' : None,

'==' : None,

'!=' : None,

'&&' : None,

'||' : None,

'=' : None

},

float: {

'+' : None,

'-' : None,

'\*' : None,

'/' : None,

'<' : None,

'<=' : None,

'>' : None,

'>=' : None,

'==' : None,

'!=' : None,

'&&' : None,

'||' : None,

'=' : None

},

str: {

'+' : str,

'-' : None,

'\*' : None,

'/' : None,

'<' : None,

'<=' : None,

'>' : None,

'>=' : None,

'==' : None,

'!=' : None,

'&&' : None,

'||' : None,

'=' : str

},

bool: {

'+' : None,

'-' : None,

'\*' : None,

'/' : None,

'<' : None,

'<=' : None,

'>' : None,

'>=' : None,

'==' : None,

'!=' : None,

'&&' : None,

'||' : None,

'=' : None

}

},

bool: {

int: {

'+' : None,

'-' : None,

'\*' : None,

'/' : None,

'<' : None,

'<=' : None,

'>' : None,

'>=' : None,

'==' : None,

'!=' : None,

'&&' : None,

'||' : None,

'=' : None

},

float: {

'+' : None,

'-' : None,

'\*' : None,

'/' : None,

'<' : None,

'<=' : None,

'>' : None,

'>=' : None,

'==' : None,

'!=' : None,

'&&' : None,

'||' : None,

'=' : None

},

str: {

'+' : None,

'-' : None,

'\*' : None,

'/' : None,

'<' : None,

'<=' : None,

'>' : None,

'>=' : None,

'==' : None,

'!=' : None,

'&&' : None,

'||' : None,

'=' : None

},

bool: {

'+' : None,

'-' : None,

'\*' : None,

'/' : None,

'<' : None,

'<=' : None,

'>' : None,

'>=' : None,

'==' : None,

'!=' : None,

'&&' : bool,

'||' : bool,

'=' : bool

}

}

}

try:

return semanticCube[op1][op2][operator]

except KeyError:

raise KeyError("WARNING! KeyError in getResultingType: semanticCube[{}][{}][{}]".format(op1, op2, operator))

return None

## Clase Variables y Directorio

#Begin class Variable

#Clase: Variable

#Atributos: Name, Type, Address, Dimension

class Variable:

def \_\_init\_\_(self, Name='0', Type=int, Address='0', Dimension='0'):

self.Name = Name

self.Type = Type

self.Address = Address

self.Dimension = Dimension

def \_\_str\_\_(self):

return "{}\t{}\t{}\t{}".format(str(self.Name), str(self.Type), str(self.Address), str(self.Dimension))

#End class Variable

from quadrupleGenerator import quadruple

#Begin class procedureDirectory

#Clase: procedureDirectory

#Atributos: diccionario de constantes, id, parent, lista de parametros,

# diccionario de variables, diccionario de directorios hijos,

# tipo y dirección de inicio.

class procedureDirectory:

constants = {} #stores the value of constants

def \_\_init\_\_(self, identifier, parent=None):

self.identifier = identifier #directory identifier

self.parent = parent #pointer to the directory's parent

self.parameters = [] #current directory's parameter table

self.variables = {} #current directory's variable table

self.directories = {} #current directory's children

self.Type = None

self.startAddress = 0

#Note: works with preincrement, so variables start one index below specification

if parent:

self.nextIntVarAddress = 9999 #10,000 - 10,999

self.nextDoubleVarAddress = 10999 #11,000 - 11,999

self.nextStringVarAddress = 11999 #12,000 - 12,999

self.nextIntTempAddress = 12999 #13,000 - 13,999

self.nextDoubleTempAddress = 13999 #14,000 - 14,999

self.nextStringTempAddress = 14999 #15,000 - 15,999

self.nextBoolTempAddress = 15999 #16,000 - 16,999

else:

self.nextIntVarAddress = 2999 #3,000 - 3,999

self.nextDoubleVarAddress = 3999 #4,000 - 4,999

self.nextStringVarAddress = 4999 #5,000 - 5,999

self.nextIntTempAddress = 5999 #6,000 - 6,999

self.nextDoubleTempAddress = 6999 #7,000 - 7,999

self.nextStringTempAddress = 7999 #8,000 - 8,999

self.nextBoolTempAddress = 8999 #9,000 - 9,999

#Note: zero-const is zero and all variables and constants are numbered with preincrement

self.nextIntConst = -1 #0 - 999

self.nextDoubleConst = 999 #1,000 - 1,999

self.nextStringConst = 1999 #2,000 - 2,999

self.add\_const(int, 0)

def \_\_str\_\_(self):

return self.to\_string()

#Método: to\_string

#Parámetros de entrada:

#Output: Imprime toda la estructura.

#Utilizada primordialmente para depurar el código.

def to\_string(self,):

string = str(self.identifier) + "{\n"

#Imprimir las variables globales primero

if(self.variables):

string = string + "variables:\n"

for identifier in self.variables:

string = string + str(self.variables[identifier])

#Append the constant value to constant variables

try:

string = string + " := " + str(procedureDirectory.constants[identifier])

except KeyError:

# print ("warning: constant key error")

pass

string = string + "\n"

#Imprimir los directorios hijos y sus contenidos

if(self.directories):

string = string + "\ndirectories:\n"

for identifier in self.directories:

string = string + str(self.directories[identifier])

string = string + "}\n\n"

string = string + "Starting Address: " + str(self.startAddress) + "\n\n"

return string

#Método: getReturnVariable

#Parámetros de entrada:

#Output: regresa la variable que se crea en la dir. 17,000

# la cual es la variable de retorno de cada función.

#Utilizado para asignar el valor de la funcion de retorno a otra variable.

def getReturnVariable(self):

return Variable("retVar", self.Type, 17000)

#Método: get\_variable

#Parámetros de entrada: id

#id: identificador de la variable.

#Output: regresa la variable si es que la encuentra.

#Utilizado frecuentemente en el parser para checar si la varaible existe.

def get\_variable(self, identifier):

currDir = self

#Se busca primero en el directorio actual, sino se checa en el global

while currDir:

if identifier in currDir.variables:

return currDir.variables[identifier]

currDir = currDir.parent

return None

#Método: add\_variable

#Parámetros de entrada: id, tipo, dimensiones, Clase

#id: identificador de la variable

#tipo: int, float o str

#dimensiones: 0 si no es arreglo, sino tam del arreglo.

#Clase: diferenciar entre const, variables, params, temporals.

#Output: regresa True si la operación fue realizada correctamente.

# Error si algo esta mal.

#Utilizado siempre que se va a insertar una variable normal desde el parser

# o desde dentro de esta estructura, cuando se agregan constantes o temporales.

def add\_variable(self, identifier, variableType, variableDimensions, variableClass = "variable"):

if identifier in self.variables:

print ("ERROR: Variable '{}' already exists in current scope '{}'!".format(str(identifier), str(self.identifier)))

raise SystemExit

else:

#Se crea la variable en la siguiente posicion de memoria virtual disponible

self.variables[identifier] = Variable(identifier, variableType, self.next\_address(variableClass, variableType), variableDimensions)

if(variableDimensions):

if variableType is int:

self.nextIntVarAddress += variableDimensions - 1

elif variableType is float:

self.nextDoubleVarAddress += variableDimensions - 1

elif variableType is str:

self.nextStringVarAddress += variableDimensions - 1

return True

#Método: add\_temp

#Parámetros de entrada: tipo

#tipo: int, float o str

#Output: regresa True si la operación fue realizada correctamente.

# Error si algo está mal.

#Utilizado siempre que se pide una variable temporal desde el parser.

def add\_temp(self, variableType):

variableClass = "temporal"

#Se le asigna el siguiente espacio de memoria segun sea su tipo.

if variableType is int:

identifier = "T{}".format(self.nextIntTempAddress+1)

elif variableType is float:

identifier = "T{}".format(self.nextDoubleTempAddress+1)

elif variableType is str:

identifier = "T{}".format(self.nextStringTempAddress+1)

elif variableType is bool:

identifier = "T{}".format(self.nextBoolTempAddress+1)

#Agrega la variable temporal al diccionario de variables.

if self.add\_variable(identifier, variableType, 0, variableClass):

return self.get\_variable(identifier)

else:

return False

#Método: add\_const

#Parámetros de entrada: tipo, valor

#tipo: int, float o str

#valor: valor numérico de la constante.

#Output: regresa True si la operación fue realizada correctamente.

# Error si algo esta mal.

#Utilizado siempre que se va a agregar una constante, sea nueva o no.

def add\_const(self, variableType, variableValue):

""" Adds a constant to the Procedure Directory.

Constants are always added to the global Procedure Directory.

Return: "Variable" referencing the constant value, or false if an error occurred.

"""

variableClass = "constant"

identifier = str(variableValue)

#if the constant already exists, return

if self.get\_variable(identifier):

return self.get\_variable(identifier)

#else:

#find global directory

currDir = self

while currDir.parent:

currDir = currDir.parent

#currDir is now the global directory

#add constant to the global directory

if currDir.add\_variable(identifier, variableType, 0, variableClass):

procedureDirectory.constants[identifier] = variableValue

return self.get\_variable(identifier)

#Método: next\_address

#Parámetros de entrada: clase, tipo

#clase: variable, temporal, constante

#tipo: int, float o str

#Output: regresa el siguiente espacio de memoria disponible.

# funciona con preincremento.

#Utilizado siempre que se necesita pedir el siguiente espacio disponible.

def next\_address(self, variableClass, variableType):

if variableClass == "variable":

if variableType is int:

self.nextIntVarAddress += 1

return self.nextIntVarAddress

elif variableType is float:

self.nextDoubleVarAddress += 1

return self.nextDoubleVarAddress

elif variableType is str:

self.nextStringVarAddress += 1

return self.nextStringVarAddress

elif variableClass == "temporal":

if variableType is int:

self.nextIntTempAddress += 1

return self.nextIntTempAddress

elif variableType is float:

self.nextDoubleTempAddress += 1

return self.nextDoubleTempAddress

elif variableType is str:

self.nextStringTempAddress += 1

return self.nextStringTempAddress

elif variableType is bool:

self.nextBoolTempAddress += 1

return self.nextBoolTempAddress

elif variableClass == "constant":

if variableType is int:

self.nextIntConst += 1

return self.nextIntConst

elif variableType is float:

self.nextDoubleConst += 1

return self.nextDoubleConst

elif variableType is str:

self.nextStringConst += 1

return self.nextStringConst

#Método: rem\_variable

#Parámetros de entrada: id

#id: identificador de la variable

#Output: regresa True si la operación fue realizada correctamente.

# Error si algo está mal.

#Remover la variable del diccionario.

def rem\_variable(self, identifier):

if identifier in self.variables:

del self.variables[identifier]

return True

else:

return False

#Método: list\_all\_variables

#Parámetros de entrada:

#Output: listado de las variables locales.

#Desplegar las variables locales, debugging.

def list\_all\_variables(self):

currDirr = self

string = ""

while(currDirr):

string = string + str(currDirr.identifier) + "{\n"

if(currDirr.variables):

for identifier in currDirr.variables:

string = string + str(currDirr.variables[identifier]) + "\n"

string = string + "}\n"

currDirr = currDirr.parent

return string

#Método: get\_all\_variables

#Parámetros de entrada:

#Output: diccionario completo de variables.

#Regresa la estructura completa de las variables del procedimiento.

#Utilizado para generar los cuádruplos de las variables constantes.

def get\_all\_variables(self):

currDir = self

allVariables = []

for var in procedureDirectory.constants:

if not var in allVariables:

allVariables.append(var)

return allVariables

#Método: get\_directory

#Parámetros de entrada: id

#id: identificador del directorio.

#Output: apuntador al directorio.

#Regresa la estructura completa del directorio.

#Utilizado para cambiar de contexto siempre que se encuentra una función nueva

# o se llego al main method.

def get\_directory(self, identifier):

currDir = self

while currDir:

if identifier in currDir.directories:

return currDir.directories[identifier]

currDir = currDir.parent

return None

#Método: add\_directory

#Parámetros de entrada: id

#id: identificador del directorio.

#Output: True si agrego el directorio correctamente.

#Utilizado cada vez que se quiera definir una nueva función, empleada desde

# el parser.

def add\_directory(self, identifier):

if identifier in self.directories:

print ("ERROR: Function '{}' already exists in scope: '{}'!".format(str(identifier), str(self.identifier)))

raise SystemExit

else:

self.directories[identifier] = procedureDirectory(identifier, self)

return True

#Método: rem\_directory

#Parámetros de entrada: id

#id: identificador del directorio.

#Output: True si borro el directorio correctamente.

#Utilizado para liberar espacio de memoria, cuando el directorio ya no es requerido.

def rem\_directory(self, identifier):

if identifier in self.directories:

del self.directories[identifier]

return True

else:

return False

#Método: rem\_directory

#Parámetros de entrada:

#Output: Genera todos los cuádruplos de las variables constantes.

#Utilizado al final del parser para generar todas los cuádruplos de todas las

# variables constantes. Dado a que esta es la única información que se manda a la

# máquina virtual.

def getConstantDeclarations(self):

constantDeclarations = []

for var in self.get\_all\_variables():

var = self.get\_variable(var)

if var.Address <= self.nextStringConst:

constantDeclarations.append(quadruple('CST', var, Variable(0), var))

string = ""

for const in constantDeclarations:

const.printFormat = "Constants"

string += '{}\n'.format(str(const))

return string

## Máquina Virtual

#Método runVM

#Input:

#Output: ejecutar el método correspondiente al operador

#Utilizado en todos los cuádruplos que genera el código.

def runVM():

global PC, operator, op1, op2, result, passingArray, resultIsArray

PC = 0

methods = globals().copy()

methods.update(locals())

#Ejecutarse mientras tenga instrucciones.

while PC < len(instructionMemory):

#Read quadruple

operator, op1, op2, result = instructionMemory[PC]

#intenta parsear result a int.

#Cuando es un dir. indirecto, va en ().

try:

result = int(result)

except:

result = result[1:-1]

result = int(result)

resultIsArray = True

#Translate addresses

#menos los que están en la lista, dado a que requieren de parseo especial.

if operator not in ["OFF", "CAL", "RET", "INI", "RED"]:

#Checar si se están pasando parametros en ese momento.

if passingParameters:

#intenta parsear op2 a int.

#Cuando es un dir. indirecto, va en ().

try:

op2 = int(op2)

except:

op2 = op2[1:-1]

op2 = int(op2)

passingArray = 2

#Dir 17000 es la dir. de retorno

if op2 == 17000:

op2 = returnStack.pop()

#Dir locales 10,000 - 16,999

elif op2 >= inferiorLimit:

op2 = (op2 - inferiorLimit) + segmentLength\*(stackPointer-1)

op2 = methodsMemory[op2]

#dir global 0 - 9999

elif op2 < inferiorLimit:

op2 = globalMemory[op2]

try:

op1 = int(op1)

except:

op1 = op1[1:-1]

op1 = int(op1)

if passingArray == 2:

passingArray = 3

else:

passingArray = 1

if op1 == 17000:

op1 = returnStack.pop()

elif op1 >= inferiorLimit:

op1 = (op1 - inferiorLimit) + segmentLength\*(stackPointer-1)

op1 = methodsMemory[op1]

elif op1 < inferiorLimit:

op1 = globalMemory[op1]

else:

#caso cuando no se esta pasando parametros.

#el único cambio es la traducción de las variables locales.

#dado a que el stackPointer hace referencia a un bloque anterior.

try:

op2 = int(op2)

except:

op2 = op2[1:-1]

op2 = int(op2)

passingArray = 2

if op2 == 17000:

op2 = returnStack.pop()

elif op2 >= inferiorLimit:

op2 = (op2 - inferiorLimit) + segmentLength\*(stackPointer)

op2 = methodsMemory[op2]

elif op2 < inferiorLimit:

op2 = globalMemory[op2]

try:

op1 = int(op1)

except:

op1 = op1[1:-1]

op1 = int(op1)

if passingArray == 2:

passingArray = 3

else:

passingArray = 1

if op1 == 17000:

op1 = returnStack.pop()

elif op1 >= inferiorLimit:

op1 = (op1 - inferiorLimit) + segmentLength\*(stackPointer)

op1 = methodsMemory[op1]

elif op1 < inferiorLimit:

op1 = globalMemory[op1]

#Execute quadruple

method = methods.get(str(operator))

if not method:

raise Exception("ERROR: Method \"{}\" not implemented!".format(str(operator)))

#Stack overflow si el stackpointer intenta acceder a un espacio de memoria no disponible.

if stackPointer >= 1000:

raise Exception("ERROR: Stack overflow!")

# print("OPERATOR: ", operator)

# print("OP1: ", op1)

# print("OP2: ", op2)

# print("RESULT: ", result)

# print("Pointer: ", stackPointer)

method(op1, op2, result)

#Increment PC to execute next instruction

PC += 1

input("Press <ENTER> to exit...")

#Debugging code

'''

for i,n in enumerate(instructionMemory):

print "{}\t{}\t{}\t{}\t{}\n".format(i, n[0], n[1], n[2], n[3])

raise SystemExit

'''

## Máquina Virtual – Resta

#Código de la máquina virtual

#Metodo: SUB

#Parámetros de entrada: op1, op2, result.

#op1: valor a restar

#op2: valor a restar

#result: dirección en donde se guardará el resutlado.

#Output: guarda el resultado de la resta en la localidad de memoria correspondiente.

def SUB(op1, op2, result):

global passingArray

try:

#Se checa si resultado es una dirección global o local.

if result >= inferiorLimit:

#Si se están pasando parámetros se tiene que hacer referencia al

#bloque anterior de memoria.

if passingParameters:

result = (result-inferiorLimit) + segmentLength\*(stackPointer-1);

else:

result = (result-inferiorLimit) + segmentLength\*stackPointer;

#Checar si uno de los valores recibidos es una dir. indirecta.

#Esto es el caso cuando son accessos a indices de arreglos.

#1: op1 es arreglo

#2: op2 es arreglo

#3: op1 y op2 son arreglos

if passingArray == 1:

if op1 < inferiorLimit:

methodsMemory[result] = globalMemory[op1] - op2

else:

op1 = methodsMemory[(op1 - inferiorLimit) + segmentLength\*stackPointer]

methodsMemory[result] = methodsMemory[op1] - op2

elif passingArray == 2:

if op2 < inferiorLimit:

methodsMemory[result] = op1 - globalMemory[op2]

else:

op2 = methodsMemory[(op2 - inferiorLimit) + segmentLength\*stackPointer]

methodsMemory[result] = op1 - methodsMemory[op2]

elif passingArray == 3:

if op1 < inferiorLimit and op2 < inferiorLimit:

methodsMemory[result] = globalMemory[op1] - globalMemory[op2]

elif op1 < inferiorLimit:

op2 = methodsMemory[(op2 - inferiorLimit) + segmentLength\*stackPointer]

methodsMemory[result] = globalMemory[op1] - methodsMemory[op2]

elif op2 < inferiorLimit:

op1 = methodsMemory[(op1 - inferiorLimit) + segmentLength\*stackPointer]

methodsMemory[result] = methodsMemory[op1] - globalMemory[op2]

else:

op1 = methodsMemory[(op1 - inferiorLimit) + segmentLength\*stackPointer]

op2 = methodsMemory[(op2 - inferiorLimit) + segmentLength\*stackPointer]

methodsMemory[result] = methodsMemory[op1] - methodsMemory[op2]

else:

methodsMemory[result] = op1 - op2

elif result < inferiorLimit:

if passingArray == 1:

if op1 < inferiorLimit:

methodsMemory[result] = globalMemory[op1] - op2

else:

op1 = methodsMemory[(op1 - inferiorLimit) + segmentLength\*stackPointer]

methodsMemory[result] = methodsMemory[op1] - op2

elif passingArray == 2:

if op2 < inferiorLimit:

methodsMemory[result] = op1 - globalMemory[op2]

else:

op2 = methodsMemory[(op2 - inferiorLimit) + segmentLength\*stackPointer]

methodsMemory[result] = op1 - methodsMemory[op2]

elif passingArray == 3:

if op1 < inferiorLimit and op2 < inferiorLimit:

methodsMemory[result] = globalMemory[op1] - globalMemory[op2]

elif op1 < inferiorLimit:

op2 = methodsMemory[(op2 - inferiorLimit) + segmentLength\*stackPointer]

methodsMemory[result] = globalMemory[op1] - methodsMemory[op2]

elif op2 < inferiorLimit:

op1 = methodsMemory[(op1 - inferiorLimit) + segmentLength\*stackPointer]

methodsMemory[result] = methodsMemory[op1] - globalMemory[op2]

else:

op1 = methodsMemory[(op1 - inferiorLimit) + segmentLength\*stackPointer]

op2 = methodsMemory[(op2 - inferiorLimit) + segmentLength\*stackPointer]

methodsMemory[result] = methodsMemory[op1] - methodsMemory[op2]

else:

methodsMemory[result] = op1 - op2

except:

raise TypeError("Operation invalid for specified operand types")

passingArray = 0

# Manual de Usuario

## Tipos de variables

* Entero 🡪 int
* Flotante 🡪 double
* Cadena de caracteres 🡪 string

## Declaración de variables

* int a, b, c;
* str a = “hola”;
* double arr [10];

Operaciones permitidas:

* Aritméticas: +, -, \*, /
* Lógicas: &&, ||
* Booleanas: <, >, <=, >=, ==, !=

## Asignación de valores:

* a = 5;
* abc[3] = 15.5;
* hilo = “world”;

## Estructura general de un programa

* 0 o más variables globales
* 0 o más funciones
* Un método main cheese

Int globalEntero;

func int uno () {

<acciones>

}

main cheese () {

<acciones>

}

## Estructura general de un if

* If

If(condición > 5) {

}

* If-Else

If(condición > 10) {

<acciones>

}

Else {

<acciones>

}

## Estructura general de un for

* Se debe declarar el contador previamente.

Int i;

for(i = 0; i < 5; i = i + 1) {

<acciones>

}

## Estructura general de un while

* Se debe declarar el contador o variable de escape previamente.

int i = 0;

while(i <= 10) {

<acciones>

}

## Funciones predeterminadas

|  |  |  |
| --- | --- | --- |
| Nombre | Descripción | Ejemplo |
| scanf() | Permite la lectura del teclado. | scanf(varTemp); |
| print() | Permite imprimir texto a pantalla. | print(“hola”); |
| move() | Permite trazar líneas a través de un movimiento hacia adelante o atrás, de acuerdo al valor dado. | move(30); move(-50.5); |
| rotate() | Permite girar la dirección de la pluma en ciertos grados. Por default se rota a favor de las manecillas del reloj. | rotate(90); rotate(-180);  clockwise counterclockwise |
| arc() | Permite dibujar un arco con cierto radio, también se debe de proveer el ángulo que cubre el arco. | arc(radius, angle); |
| home() | Regresa a la pluma a las coordenadas (0,0) | home(); |
| pdown() | Baja la pluma para que se puedan trazar líneas y demás. Por default la pluma siempre esta abajo. | pdown(); |
| pup() | Levanta la pluma para que no se dibuje en la pantalla. | pup(); |
| setp() | Reposiciona el lugar de la pluma a unas nuevas coordenadas. | setp(x, y); |
| pcolor() | Cambia el color de la pluma. | pcolor(255, 0, 245); |
| psize() | Cambia el grosor de la pluma. | psize(10); |
| pclear() | Limpia la pantalla completamente. | pclear(); |