Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Monterrey

Compilador: ABC

A01283076 Antonio Díaz Ramos (firma digital)

A01275834 Citlalli Rosario Alonzo Mateos

Diseño de Compiladores

M. C. Elda Guadalupe Quiroga

Dr. Héctor Ceballos

22 de noviembre de 2022

Índice

Descripción y Documentación Técnica del Proyecto

Descripción del Proyecto

Propósito y Alcance del Proyecto

El proyecto final de compiladores tiene como objetivo que nosotros como estudiantes de la carrera de ingeniería en tecnologías computacionales seamos capaces de crear un lenguaje de programación, junto con su compilador, el archivo que se genera en compilación y la máquina virtual. Todo lo anterior haciendo uso de los conocimientos que hemos adquirido a lo largo de nuestro plan de estudios y los temas vistos en esta materia.

Debido al tiempo que tenemos para llevar a cabo esta tarea, el "universo" de nuestro lenguaje es muy pequeño a comparación de los que existen actualmente y de los que hacemos uso (C++, Java, JavaScript, Python, etc.). Sin embargo, si se toman en cuenta los elementos básicos y que son importantes para una programación básica, por lo tanto se incluye el uso y declaración de variables, operaciones aritméticas (suma, resta, multiplicación, división), operaciones lógicas (or, and, not), operaciones relacionales, estatutos secuenciales (read, write), estatutos condicionales (if, while), estatutos cíclicos (for), funciones, arreglos y matrices.

Análisis de Requerimientos

Es necesario hacer una recopilación de los aspectos más importantes que se deben tener en cuenta para medir el éxito del proyecto y saber si nuestro lenguaje ABC cumple con lo que se espera al final del semestre. Los requerimientos que se tomarán en cuenta son los siguientes:

- La gramática para el lenguaje debe ser no ambigua.
- El lenguaje debe soportar la declaración de variables locales.
- El lenguaje debe soportar la declaración de variables globales.
- El lenguaje debe soportar el uso de variables locales.
- El lenguaje debe soportar el uso de variables globales.

- El lenguaje debe soportar estatutos de asignación.
- El lenguaje debe soportar estatutos de lectura.
- El lenguaje debe soportar estatutos de escritura.
- El lenguaje debe soportar operaciones aritméticas.
- El lenguaje debe soportar operaciones lógicas.
- El lenguaje debe soportar operaciones relacionales.
- El lenguaje debe soportar estatutos condicionales.
- El lenguaje debe soportar estatutos cíclicos.
- El lenguaje debe soportar la declaración de funciones.
- El lenguaje debe soportar el uso de funciones.
- El lenguaje debe soportar la llamada recursiva a funciones.
- El lenguaje debe soportar la declaración de elementos estructurados.
- El lenguaje debe soportar el uso de elementos estructurados.
- El lenguaje debe estar orientado a un propósito específico.
- El compilador debe detectar errores léxicos, sintácticos y semánticos.
- El compilador debe generar código intermedio.
- El compilador debe ser capaz de destruir las estructuras que ya no son requeridas.
- El compilador debe ser eficiente en tiempo y memoria.
- La máquina virtual debe leer y almacenar la información del código intermedio.
- La máquina virtual debe manejar de manera dinámica la memoria local, global y de constantes.

Principales Test Cases

asdasd

Proceso General para el Desarrollo del Proyecto

Nosotros decidimos trabajar de manera presencial en todo el proyecto, es decir, ambos estuvimos pensando en cómo hacer cada especificación del proyecto desde el inicio hasta la finalización del mismo. Lo anterior con el propósito de que ambos

conociéramos el código, entendiéramos todo lo que hace cada archivo, la participación fuera activa de ambas partes y de esta manera tener el mejor resultado posible.

Cada semana, desde el inicio del segundo parcial, se trabajó en el proyecto. De igual manera se hizo una entrega con el avance que teníamos del proyecto y lo explicamos en el ReadMe para tener una noción más clara de lo que fuimos logrando. Nuestro plan de trabajo fue el siguiente:

- Diseño de los diagramas de sintaxis.
- Creación del Scanner y Parser.
- Almacenamiento de las variables en la tabla de variables.
- Creación del Cubo Semántico.
- Creación del directorio de Funciones.
- Generación de código intermedio.
- Hacer que el compilador genere un archivo con el código intermedio.
- Creación de la máquina virtual.
- Hacer que la máquina virtual empiece a ejecutar.
- Generación de código intermedio para arreglos y matrices.
- Corrección de errores del compilador y máquina virtual.

Los avances que realizamos cada semana, explicados de manera general, fueron los que se describirán a continuación, también se pueden encontrar en la rama master y main del repositorio StrictName/Compilador:

Avance 1

Sintaxis completa del programa, incluye declaración de clases (con sus atributos y métodos, y herencia simple), variables, funciones y el main. Hemos realizado diversas pruebas y consideramos que toda la sintaxis funciona y está completa, al igual que contempla lo necesario para que el compilador funcione al momento de revisar la sintaxis del lenguaje de programación.

La sintaxis que actualmente el programa maneja es la declaración del nombre de programa, clases, atributos, métodos, funciones, variables y estatutos como if, if-else, for, while-do, read y write.

Código de prueba:

```
program proyectocompilador;

class Animal
{
   attribute:
   public int x;
   private float s;

   method:
   private void prueba (int ys)
   {
      x = 5;
      y[3] = x;


      if(x+2 equal 2)
      {
         p = x;
      }

      else
      {
         metodo(x);
      }

   }
```

```
   public int hello (int y, int z)
   {
      y=3;

      while (x+2*3) do
      {
         mato(2);
         y = 2;
      }
      return e;
   }
};

class Dog : public Animal
{
   attribute:
   public int x;
   private float s;

   method:
   private void prueba (int y)
   {
      if(5>3)
      {
         x=w;
      }

      return x*2;
   }
```

```
   public int hello (int y, int z)
   {
      x=3;

      return e;
   }
};

var
   Dog perro;
   int edad;
   int edad2;
   float peso;
   char caracter;
   int array[1];
   float array2[2][3];


func int suma (int y);
var
   int edad3;
{

   x = 5.3;
   y = x;
   z = suma(x) + a - 6;

   read (x);
   read (x[2], x);
   read (x,w,x);
   write ('t', "jajaja");

   x='w';
   return x+2;
}
```

```
func void suma (int x, float y);
{
   x=2;
   y = p.ladra(temp1);

   y = p.ladra;

   p.come(x);
   p.duerma(3, y);
   p.juega(4.8);
   p.corre('a');
   p.atri1;

   write("buenas", p.edad, p.sum(7, 3.4));

   if (x > p.edad and y < p.anos ) {
      y[1] = p.edad;
      if (v < p.calculo) {
         u = 1 + p;
      }
   }

   while (p.animal) do {
      x = p.corre(3);
   }

   from s = p.t > 2 to p.ala equal 5 do {
      m1(x, 8);
   }
}

func void suma (char z);
{
   y=3;
}
```

```
main()
{
   x=2;
   write(x+2);

   p.come(x);
   p.duerma(3, y);
   p.juega(4.8);
   p.corre('a');
   p.atri1;

   from x = 2+3 to y do
   {
      read(w);
   }

   y = p.ladra(temp);

   y = p.ladra;

   write("buenas", p.edad, p.sum(7, 3.4));

   if (x > p.edad and y < p.anos ) {
      y[1] = p.edad;
      if (v < p.calculo) {
         u = 1 + p;
      }
   }

   while (p.animal) do {
      x = p.corre(3);
   }

   from s = p.t > 2 to p.ala equal 5 do {
      m1(x, 8);
   }

}
```

Avance 2

Se agregó el diccionario del cubo semántico, la tabla de variables y el directorio de funciones. El cubo semántico ya tiene los operadores derechos, operadores izquierdos,

y operando con su respectivo resultado, además, se hizo la función que devuelve el resultado. La tabla de variables ya guarda las variables de todo el programa junto con su scope y tipo. El directorio de funciones aún no está conectado a la tabla de variables pero ya guarda el nombre de la función.

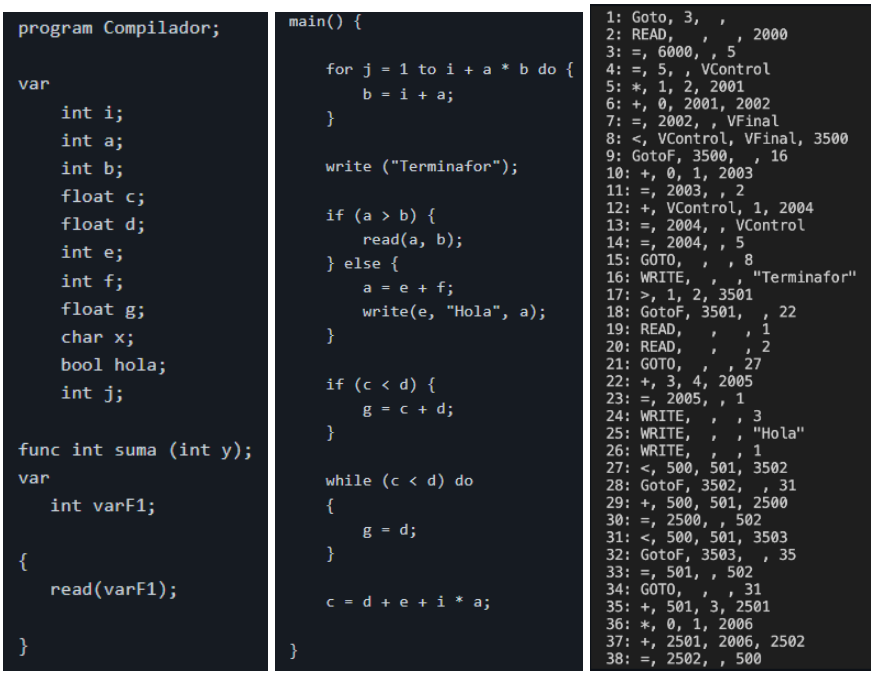
Avance 3

Se empiezan a generar los cuádruplos de las expresiones de asignación, lectura y escritura. También se generan los cuádruplos de while.

Avance 4

Se realizó la asignación de direcciones virtuales a las variables y constantes. Ya se genera el código intermedio de if y for, además, los cuádruplos se crean tomando en cuenta las direcciones virtuales.

Código de prueba:



```
program Compilador;

var
    int i;
    int a;
    int b;
    float c;
    float d;
    int e;
    int f;
    float g;
    char x;
    bool hola;
    int j;

func int suma (int y);
var
   int varF1;

{
   read(varF1);

}
```

```
main() {

    for j = 1 to i + a * b do {
        b = i + a;
    }

    write ("Terminafor");

    if (a > b) {
        read(a, b);
    } else {
        a = e + f;
        write(e, "Hola", a);
    }

    if (c < d) {
        g = c + d;
    }

    while (c < d) do
    {
        g = d;
    }

    c = d + e + i * a;

}
```

```
1: Goto, 3,  ,
2: READ,  ,  , 2000
3: =, 6000, , 5
4: =, 5, , VControl
5: *, 1, 2, 2001
6: +, 0, 2001, 2002
7: =, 2002, , VFinal
8: <, VControl, VFinal, 3500
9: GotoF, 3500,  , 16
10: +, 0, 1, 2003
11: =, 2003, , 2
12: +, VControl, 1, 2004
13: =, 2004, , VControl
14: =, 2004, , 5
15: GOTO,  ,  , 8
16: WRITE,  ,  , "Terminafor"
17: >, 1, 2, 3501
18: GotoF, 3501,  , 22
19: READ,  ,  , 1
20: READ,  ,  , 2
21: GOTO,  ,  , 27
22: +, 3, 4, 2005
23: =, 2005, , 1
24: WRITE,  ,  , 3
25: WRITE,  ,  , "Hola"
26: WRITE,  ,  , 1
27: <, 500, 501, 3502
28: GotoF, 3502,  , 31
29: +, 500, 501, 2500
30: =, 2500, , 502
31: <, 500, 501, 3503
32: GotoF, 3503,  , 35
33: =, 501, , 502
34: GOTO,  ,  , 31
35: +, 501, 3, 2501
36: *, 0, 1, 2006
37: +, 2501, 2006, 2502
38: =, 2502, , 500
```

Avance 5

Se realizaron avances en cuanto a las funciones. Ya se guardan los datos completos de la función, como el cuádruplo en donde inicia su ejecución o la cantidad de recursos

que ocupa cada función. De igual manera, se generan de los módulos como ENDFunc y RETURN. Ya se conecta la tabla de variables con su respectiva función.

Avance 6

Corrección de los cuádruplos de las funciones. Agregamos el "parche guadalupano", creación de los cuádruplos para una llamada recursiva. Creación de la máquina virtual: creación de la memoria de las constantes y almacenar valores, creación de la memoria global y local.

La lista de commits realizadas durante el desarrollo del proyecto es la siguiente:

Fecha	**Nombre**	**Breve descripción**
25 Octubre 2022	Actualizado	Ya estaban implementadas las reglas gramaticales, pero no habíamos actualizado nada en el github, y empezamos a generar la tabla de variables, el cubo semántico y el directorio de funciones. Empezamos a guardar, con puntos neurálgicos, los símbolos de expresiones aritméticas.
	Última versión	Eliminamos errores
	Datos funciones (sin variables)	Empezar a guardar datos de la función en el directorio de funciones.
26 Octubre 2022	Direcciones de memoria	Creamos los rangos que llevarán nuestras direcciones de memoria, creación de la función que asigna la dirección de acuerdo el tipo al scope de la variable/temporal.
	Memoria virtual	Agregamos a la función de asignar_direccion_memoria los rangos para las funciones.

	Asignar direcciones	Corrección de errores.
31 Octubre 2022	PilaO y PTye Funcionando	Hicimos que las pilas PilaO y PType ya guardaran los datos que les corresponden con los puntos neurálgicos para resolver las operaciones aritméticas, lógicas y relacionales.
01 Noviembre 2022	Inicio de cuádruplos	Agregar los puntos neurálgicos que eran necesarios para guardar información en las pilas y así empezar la generación de cuádruplos. También los cuádruplos ya sustituyen el nombre de la variable por su dirección de memoria. Aquí solo se imprimían los cuádruplos pero no se guardaban.
	Clase cuádruplo creada	Creación de la clase cuádruplo que guardaba los cuádruplos. Se editó en el compiler.py que los cuadruplos ya no se imprimieran y ahora se guardarán en la clase.
	Continuación cuadruplos	Creación de la función que guardaba el cuádruplo de asignación en la clase cuádruplos. Cambios en los puntos neurálgicos de asignación.
	Cuadruplos simples	Llamar al cubo semántico en los puntos neurálgicos de las operaciones aritméticas, relacionales y lógicas para comprobar que las operaciones puedan llevarse a cabo. Los cuádruplos de las operaciones ya eran guardados con la función de

		la clase de cuádruplos.
02 Noviembre 2022	Cuádruplo IF	Creación de los puntos neurálgicos gotoMain_np que realiza el primer cuádruplo “GOTO (main)”. Creación de los puntos neurálgicos y las funciones en la clase cuadruplos necesarios para el IF.
03 Noviembre 2022	While, read, write cuadruplos	Generación de cuádruplos de las instrucciones while, read, write, if/else.
	Fill main	El cuádruplo de main ya era rellenado con la línea en donde empieza el main.
04 Noviembre 2022	Ciclo for y asignación de dir virtuales constantes	Generación de cuádruplos del for, asignar dirección a los cuádruplos del for. Creación de la clase constantes que guarda y asigna la dirección de memoria correspondiente a las constantes.
07 Noviembre 2022	Implementación funciones	Editar el directorio de funciones para agregar la línea en donde comienza la función. Editar la tabla de variables para asociar la variable a la dirección de la función que pertenece.
	Cuádruplos de funciones	Creación de los cuádruplos de las funciones con dirección de memoria.
08 Noviembre 2022	Mapa de memoria y cuádruplos de funciones	Creación del archivo tipo txt que exporta cuadruplos, información del directorio de funciones y tabla de constantes. Creación del cuadruplo END. Contar el tamaño que tiene la función.

	Arreglos a cuádruplos de módulos	Errores de cuádruplos de funciones arreglados.
11 Noviembre 2022	Inicio de máquina virtual	Empezar la máquina virtual. Lectura del archivo txt generado en compilación dataVirtualMachine.txt
15 Noviembre 2022	Máquina virtual: memoria ctes, globales y temp	Máquina virtual, ya empieza a leer líneas y a separar los elementos de los cuádruplos para poder guardarlos en los diccionarios correspondientes. Creación del “parche guadalupano” en las funciones.
17 Noviembre 2022	Corrección operaciones aritméticas MV	Cambiar la dirección que se le asignaba a los temporales porque tenían un error de scope.
	Corrección op aritmética MV	Creación de la función convert_type en la MV, para mandar la dirección y convertir al tipo que le corresponde de acuerdo al rango de direcciones. Hacer más pequeño el código que realiza las operaciones aritméticas en la MV.
	Asignación con función	Crear la función search_dict que regresa el valor guardado en esa dirección. Ocupar esa función para hacer más pequeño el código de operaciones en la MV.
	Op Aritm cortas	Reducción del código.
	Read, write, op aritm	Ejecución de operaciones aritméticas, write y read en la MV.
	GOTO, GOTOF, for?	Ejecución del if y while en MV, inicio de ejecución del

		for.
	For 100%	Ejecución del for al 100.
	Errors included	Exit() en compilación cuando se encuentre un error.
18 Noviembre 2022	Reset de temporales locales	Creación de otro rango de direcciones únicamente para las funciones y así distinguir más fácil los temporales, variables y parámetros.
	Ejecución parcial de funciones	Inicio de ejecución de las funciones. Aún existen errores.

Párrafo de Reflexión: Citlalli

Párrafo de Reflexión: Antonio

Descripción del Lenguaje

Nombre del Lenguaje

Nuestro lenguaje de programación lleva el nombre **ABC.** Este surgió al inicio de juntar las iniciales de nuestros nombres A+C. Después notamos que si le agregamos la letra b en el medio es como las primeras letras del alfabeto, y pudiera ser una buena analogía ya que como nuestro lenguaje es muy básico solamente podría ser usado para personas que comienzan en el mundo de la programación.

Características del Lenguaje

En el lenguaje ABC es aceptada la declaración y uso de variables de tipo int, float, char y bool. Se pueden hacer operaciones aritméticas, lógicas y relaciones, no obstante, tiene algunas limitaciones nuestro lenguaje ya que deben realizarse de forma jerárquica. De igual manera la declaración y uso de estatutos como if, if/else, while do, for, read y write. Asimismo, se pueden declarar funciones de tipo int, float, char, bool y void. Para el caso de las funciones void no se regresa nada, mientras que las que son de int, float, char y bool si tienen un return. Las funciones te permiten hacer recursividad lo que facilita el desarrollo de código.

Listado de Errores

En ABC se clasifican los errores en dos etapas: compilación y ejecución.

Compilación

Error	**Descripción**
Error sintáctico en '%s'	Cuando no se estructura o define bien el lenguaje de acuerdo con las reglas gramaticales y de sintaxis.
ERROR: Function undeclared	No se pueden mandar a llamar funciones que no están declaradas.
ERROR: Wrong type of parameter	Avisa cuando mandas como parámetro un tipo diferente al que se usó cuando se definió la función.
ERROR: Wrong quantity of parameters	Avisa cuando mandas un número incorrecto de parámetros a la función llamada.
No se encontró la dirección de la variable	Marca error cuando se realiza un estatuto a una variable no declarada.
No se encontró el type de la variable	Marca error cuando se realiza un estatuto a una variable no declarada.
ERROR: Type mismatch	Marca error cuando se quieren realizar operaciones entre tipos de datos que no son compatibles.
ERROR: Se excedió el máximo de variables enteras globales	Avisa cuando se han declarado más variables **int global** de las que soporta el lenguaje.

ERROR: Se excedió el máximo de variables float globales	Avisa cuando se han declarado más variables **float global** de las que soporta el lenguaje.
ERROR: Se excedió el máximo de variables char globales	Avisa cuando se han declarado más variables **char global** de las que soporta el lenguaje.
ERROR: Se excedió el máximo de variables bool globales	Avisa cuando se han declarado más variables **bool global** de las que soporta el lenguaje.
ERROR: Se excedió el máximo de variables/funciones enteras	Avisa cuando se han declarado más variables **int local** de las que soporta el lenguaje.
ERROR: Se excedió el máximo de variables/funciones float	Avisa cuando se han declarado más variables **float local** de las que soporta el lenguaje.
ERROR: Se excedió el máximo de variables/funciones char	Avisa cuando se han declarado más variables **char local** de las que soporta el lenguaje.
ERROR: Se excedió el máximo de variables/funciones bool	Avisa cuando se han declarado más variables **bool local** de las que soporta el lenguaje.
ERROR: Se excedió el máximo de variables/clases enteras	Avisa cuando se han declarado más variables **int class** de las que soporta el lenguaje.
ERROR: Se excedió el máximo de variables/clases flotantes	Avisa cuando se han declarado más variables **float class** de las que soporta el lenguaje.
ERROR: Se excedió el máximo de variables/clases char	Avisa cuando se han declarado más variables **char class** de las que soporta el lenguaje.
ERROR: Se excedió el máximo de variables/clases bool	Avisa cuando se han declarado más variables **bool class** de las que soporta el lenguaje.
ERROR: Se excedió el máximo de constantes	Avisa cuando se han declarado más constantes **int** de las que soporta el lenguaje.

int	
ERROR: Se excedió el máximo de constantes float	Avisa cuando se han declarado más constantes **float** de las que soporta el lenguaje.
ERROR: Se excedió el máximo de constantes char	Avisa cuando se han declarado más constantes **char** de las que soporta el lenguaje.
ERROR: Se excedió el máximo de funciones int	Avisa cuando se han declarado más funciones **int** de las que soporta el lenguaje.
ERROR: Se excedió el máximo de funciones float	Avisa cuando se han declarado más funciones **float** de las que soporta el lenguaje.
ERROR: Se excedió el máximo de funciones char	Avisa cuando se han declarado más funciones **char** de las que soporta el lenguaje.
ERROR: Se excedió el máximo de funciones bool	Avisa cuando se han declarado más funciones **bool** de las que soporta el lenguaje.
ERROR: Se excedió el máximo de funciones void	Avisa cuando se han declarado más funciones **void** de las que soporta el lenguaje.
Archivo no encontrado	Avisa cuando no se encontró el archivo que quiera ser compilado.
The variable {name} already exists	Revisa que el nombre de las variables no se repita.
Variable undeclared	Revisa que no se pueda hacer uso de una variable no declarada.

Descripción del Compilador

Herramientas Utilizadas

Para la creación de ABC se hizo uso de las siguientes herramientas:

- Python: Python es un lenguaje de programación ampliamente utilizado en las aplicaciones web, el desarrollo de software, la ciencia de datos y el machine learning (ML).
- PLY: PLY es una implementación de lex y yacc herramientas de análisis para Python.
- Visual Studio Code: Visual Studio Code es un editor de código fuente desarrollado por Microsoft para Windows, Linux, macOS y Web.
- Windows: Microsoft Windows es el nombre del Sistema Operativo desarrollado por Microsoft para PC de escritorio, servidores y sistemas empotrados.
- MacOS: MacOS es un sistema operativo diseñado por Apple que está instalado en todos los equipos creados por la compañía Apple Inc., y son conocidos generalmente como Mac.

Descripción del Análisis Léxico

Patrones de Construcción

Las expresiones regulares son patrones que se utilizan para hacer coincidir combinaciones de caracteres en cadenas. En nuestro lenguaje de programación tenemos las siguientes:

- ID: [a-zA-Z_][a-zA-Z_0-9]*
- CTECH: [a-zA-Z_][a-zA-Z_0-9]?
- CTEF: [0-9]+ \.[0-9]+
- CTEI: [0-9]+
- LETRERO: "[a-zA-Z_][a-zA-Z_0-9]*"
- newline: \n+

Tokens del Lenguaje

Los tokens son los elementos equivalentes a las palabras y signos de puntuación en el lenguaje natural escrito. Para el caso de ABC son:

PUNTOCOMA	;
PARENTESISIZQ	(
PARENTESIDER	)
LLAVEIZQ	{
LLAVEDER	}
CORCHETEIZQ	[
CORCHETEDER	]
COMA	,
DOSPUNTOS	:
IGUAL	=
LESSTHAN	<
GREATERTHAN	>
MAS	+
MENOS	-
POR	*
DIV	/
PUNTO	.

Palabras Reservadas

Las palabras reservadas tienen un significado especial para realizar ciertas tareas del compilador, las de nuestro lenguaje son:

'program'	'main'	'class'	'var'	'int'
'float'	'char'	'bool'	'func'	'void'
'return'	'public'	'private'	'protected'	'attribute'
'method'	'read'	'write'	'while'	'do'
'for'	'to'	'and'	'or'	'not'

'equal' 'if' 'else'

Descripción del Análisis de Sintaxis

Gramática Formal

La gramática formal es un conjunto de reglas para reescribir cadenas, junto con un símbolo de inicio a partir del cual comienza la reescritura. La gramática formal definida para nuestro lenguaje es la siguiente:

program:
PROGRAM ID PUNTOCOMA main
| PROGRAM ID PUNTOCOMA clase main
| PROGRAM ID PUNTOCOMA clase var main
| PROGRAM ID PUNTOCOMA clase var funcion main
| PROGRAM ID PUNTOCOMA clase funcion main
| PROGRAM ID PUNTOCOMA var main
| PROGRAM ID PUNTOCOMA var funcion main
| PROGRAM ID PUNTOCOMA funcion main

main:
MAIN PARENTESISIZQ PARENTESISDER LLAVEIZQ LLAVEDER
| MAIN PARENTESISIZQ PARENTESISDER LLAVEIZQ estatuto LLAVEDER

clase:
CLASS ID DOSPUNTOS tipo_clase ID LLAVEIZQ bloque_clase LLAVEDER PUNTOCOMA
| CLASS ID DOSPUNTOS tipo_clase ID LLAVEIZQ bloque_clase LLAVEDER PUNTOCOMA clase
| CLASS ID LLAVEIZQ bloque_clase LLAVEDER PUNTOCOMA
| CLASS ID LLAVEIZQ bloque_clase LLAVEDER PUNTOCOMA clase

tipo_clase:

PUBLIC

| PROTECTED

| PRIVATE

var:

VAR varp

varp:

tipo_compuesto ID PUNTOCOMA

| tipo_compuesto ID PUNTOCOMA varp

| tipo_simple ID PUNTOCOMA

| tipo_simple ID PUNTOCOMA varp

| tipo_simple ID CORCHETEIZQ CTEI CORCHETEDER PUNTOCOMA

| tipo_simple ID CORCHETEIZQ CTEI CORCHETEDER PUNTOCOMA varp

| tipo_simple ID CORCHETEIZQ CTEI CORCHETEDER CORCHETEIZQ CTEI CORCHETEDER PUNTOCOMA

| tipo_simple ID CORCHETEIZQ CTEI CORCHETEDER CORCHETEIZQ CTEI CORCHETEDER PUNTOCOMA varp

tipo_simple:

INT

| FLOAT

| CHAR

| BOOL

tipo_simple_func:

INT

| FLOAT

| CHAR

| BOOL

tipo_compuesto:
ID

funcion:
FUNC tipo_simple_func ID PARENTESISIZQ parametros PARENTESISDER PUNTOCOMA dec_var cuerpo
| FUNC tipo_simple_func ID PARENTESISIZQ parametros PARENTESISDER PUNTOCOMA dec_var cuerpo funcion
| FUNC tipo_simple_func ID PARENTESISIZQ parametros PARENTESISDER PUNTOCOMA cuerpo
| FUNC tipo_simple_func ID PARENTESISIZQ parametros PARENTESISDER PUNTOCOMA cuerpo funcion
| FUNC VOID ID PARENTESISIZQ parametros PARENTESISDER PUNTOCOMA dec_var cuerpo_void
| FUNC VOID ID PARENTESISIZQ parametros PARENTESISDER PUNTOCOMA dec_var cuerpo_void funcion
| FUNC VOID ID PARENTESISIZQ parametros PARENTESISDER PUNTOCOMA cuerpo_void
| FUNC VOID ID PARENTESISIZQ parametros PARENTESISDER PUNTOCOMA cuerpo_void funcion

dec_var:
VAR dec_varp

dec_varp:
tipo_simple ID PUNTOCOMA dec_varp
| tipo_simple ID PUNTOCOMA
| tipo_simple ID CORCHETEIZQ CTEI CORCHETEDER PUNTOCOMA dec_varp
| tipo_simple ID CORCHETEIZQ CTEI CORCHETEDER PUNTOCOMA

| tipo_simple ID CORCHETEIZQ CTEI CORCHETEDER CORCHETEIZQ CTEI CORCHETEDER PUNTOCOMA dec_varp
| tipo_simple ID CORCHETEIZQ CTEI CORCHETEDER CORCHETEIZQ CTEI CORCHETEDER PUNTOCOMA

parametros:
INT ID
| INT ID COMA parametros
| FLOAT ID
| FLOAT ID COMA parametros
| CHAR ID
| CHAR ID COMA parametros
| BOOL ID
| BOOL ID COMA parametros

cuerpo:
LLAVEIZQ estatuto RETURN exp PUNTOCOMA LLAVEDER
| LLAVEIZQ estatuto LLAVEDER

cuerpo_void:
LLAVEIZQ estatuto LLAVEDER

bloque_clase:
ATTRIBUTE DOSPUNTOS atributo METHOD DOSPUNTOS metodo

atributo:
tipo_clase tipo_simple ID PUNTOCOMA
| tipo_clase tipo_simple ID PUNTOCOMA atributo

metodo:
tipo_clase tipo_simple ID PARENTESISIZQ parametros PARENTESISDER cuerpo

| tipo_clase tipo_simple ID PARENTESISIZQ parametros PARENTESISDER cuerpo metodo
| tipo_clase VOID ID PARENTESISIZQ parametros PARENTESISDER cuerpo
| tipo_clase VOID ID PARENTESISIZQ parametros PARENTESISDER cuerpo metodo

estatuto:
asignacion PUNTOCOMA
| asignacion PUNTOCOMA estatuto
| llamada PUNTOCOMA
| llamada PUNTOCOMA estatuto
| lee PUNTOCOMA
| lee PUNTOCOMA estatuto
| escribe PUNTOCOMA
| escribe PUNTOCOMA estatuto
| condicion
| condicion estatuto
| ciclo_w
| ciclo_w estatuto
| ciclo_f
| ciclo_f estatuto
| llamada_metodo PUNTOCOMA estatuto
| llamada_atributo PUNTOCOMA estatuto

asignacion:
variable IGUAL exp

llamada:
ID PARENTESISIZQ llamadap PARENTESISDER

llamadap:

exp
| exp COMA llamadap

lee:
READ PARENTESISIZQ leep PARENTESISDER

leep:
variable
| variable COMA leep

variable:
ID
| ID CORCHETEIZQ exp CORCHETEDER
| ID CORCHETEIZQ exp CORCHETEDER CORCHETEIZQ exp CORCHETEDER

escribe:
WRITE PARENTESISIZQ escribep PARENTESISDER

escribep:
exp
| exp COMA escribep
| LETRERO
| LETRERO COMA escribep

condicion:
IF PARENTESISIZQ exp PARENTESISDER LLAVEIZQ estatuto LLAVEDER
| IF PARENTESISIZQ exp PARENTESISDER LLAVEIZQ estatuto LLAVEDER ELSE LLAVEIZQ estatuto LLAVEDER

ciclo_w:

WHILE PARENTESISIZQ exp PARENTESISDER DO LLAVEIZQ estatuto LLAVEDER'

ciclo_f:

FOR variable_for IGUAL exp TO exp DO LLAVEIZQ estatuto LLAVEDER

variable_for:

ID

exp:

t_exp

| t_exp OR exp

t_exp:

g_exp

| g_exp AND t_exp

g_exp:

m_exp

| m_exp EQUAL m_exp

| m_exp NOT m_exp

| m_exp GREATERTHAN m_exp

| m_exp LESSTHAN m_exp

m_exp:

t

| t MAS m_exp

| t MENOS m_exp

t:

f

| f POR t

| f DIV t

f:

PARENTESISIZQ exp PARENTESISDER

| CTEI

| CTEF

| CTECH

| variable

| llamada

| llamada_metodo

| llamada_atributo

llamada_metodo:

ID PUNTO ID PARENTESISIZQ llamada_metodop PARENTESISDER

llamada_metodop:

CTEI

| CTEI COMA llamada_metodop

| CTEF

| CTEF COMA llamada_metodop

| CTECH

| CTECH COMA llamada_metodop

| ID

| ID COMA llamada_metodop

llamada_atributo:

ID PUNTO ID

Descripción de Generación de Código Intermedio y Análisis Semántico

Código Intermedio

El compilador de ABC logra transformar el código que le da el usuario en código intermedio usando el método de cuádruplos. El código intermedio se encuentra en el archivo dataVirtualMachine.txt una vez que termina de compilarse y no cuenta con errores.

Los cuádruplos que se generan tienen la siguiente estructura:

Operador Argumento1 Argumento2 Temporal

Las direcciones asociadas a los temporales dependen de si son locales o globales.

Códigos de Operación

Un código de operación es la parte de una instrucción en lenguaje máquina que especifica la operación a realizar. Los códigos de operación que pueden encontrarse en los cuádruplos que ABC genera son los siguientes:

GOTO	Te indica a qué número de línea debes moverte, puede ser una anterior o una posterior. La línea está indicada en la última posición del cuádruplo.
GOTOF	Te indica a qué línea debes moverte si el valor del temporal es falso.
=	Realiza la asignación del argumento 1 al temporal.
-	Realiza la resta del argumento 1 y argumento 2 y lo guarda en el temporal.
+	Realiza la suma del argumento 1 y argumento 2 y lo guarda en el temporal.
*	Realiza la multiplicación del argumento 1 y argumento 2 y lo guarda en el temporal.
/	Realiza la división del argumento 1 y argumento 2 y lo

	guarda en el temporal.
>	Realiza la comparación de argumento 1 > argumento 2 y lo guarda en el temporal.
<	Realiza la comparación de argumento 1 < argumento 2 y lo guarda en el temporal.
equal	Realiza la comparación de argumento 1 == argumento 2 y lo guarda en el temporal.
not	Realiza la comparación de argumento 1 != argumento 2 y lo guarda en el temporal.
and	Realiza la comparación de argumento 1 && argumento 2 y lo guarda en el temporal.
or	Realiza la comparación de argumento 1 | argumento 2 y lo guarda en el temporal.
WRITE	Imprime lo que se le manda en el temporal.
READ	Lee lo que se guarda en el temporal.
ERA	Crea el tamaño de memoria necesario para la función que se va a llamar.
PARAMETER	Manda como parámetro el argumento 1, a la función que previamente se llamó en ERA. La posición de temporal indica el número de parámetro que es.
GOSUB	Te indica en la posición del temporal en que línea empieza la función que se va a mandar a llamar.
RETURN	Regresa el valor que está indicado en la posición del temporal.
ENDFunc	Indica la línea en donde termina la función.

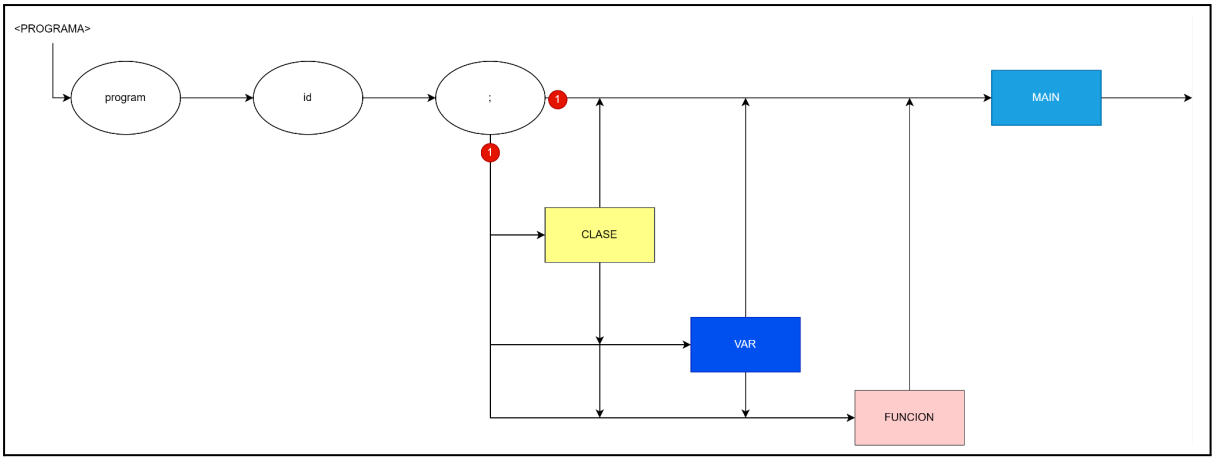
Direcciones Virtuales

Una dirección virtual no representa la ubicación física real de un objeto en memoria, en cambio, en la memoria es un puntero para un espacio. Las direcciones que ABC asigna están divididas por scope y tipo de dato. Los rangos de direcciones son de la siguiente manera:

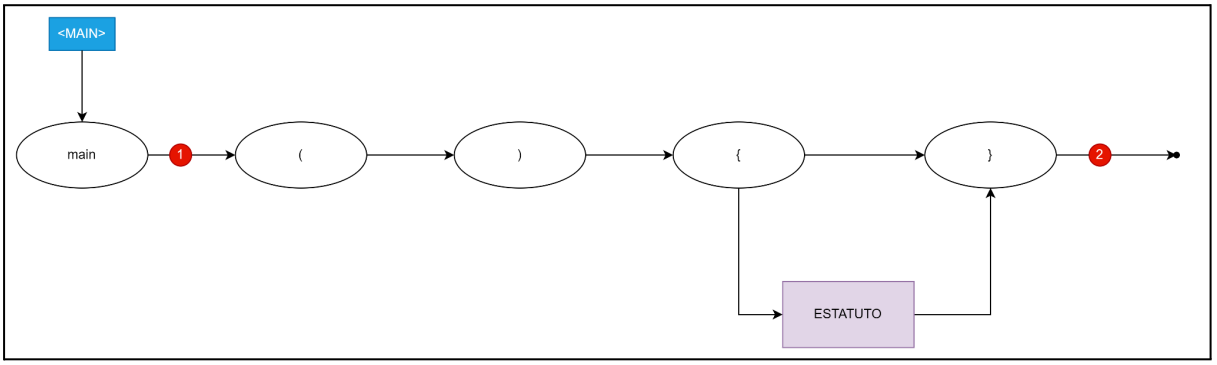
- 0: programa
- 1 - 999: variables enteras globales
- 1000 - 1999: variables flotantes globales
- 2000 - 2999: variables char globales
- 3000 - 3999: variables bool globales
- 4000 - 4999: variables enteras locales
- 5000 - 5999: variables flotantes locales
- 6000 - 6999: variables char locales
- 7000 - 7999: variables bool locales
- 8000 - 8999: funciones void
- 9000 - 9999: variables int clase
- 10000 - 10999: variables float clase
- 11000 - 11999: variables char clase
- 12000 - 12999: variables bool clase
- 13000 - 13999: constantes enteras
- 14000 - 14999: constantes flotantes
- 15000 - 15999: constantes char
- 16000 - 16999: funciones int
- 17000 - 17999: funciones float
- 18000 - 18999: funciones char
- 19000 - 19999: funciones bool

Diagramas de Sintaxis y Acciones Semánticas

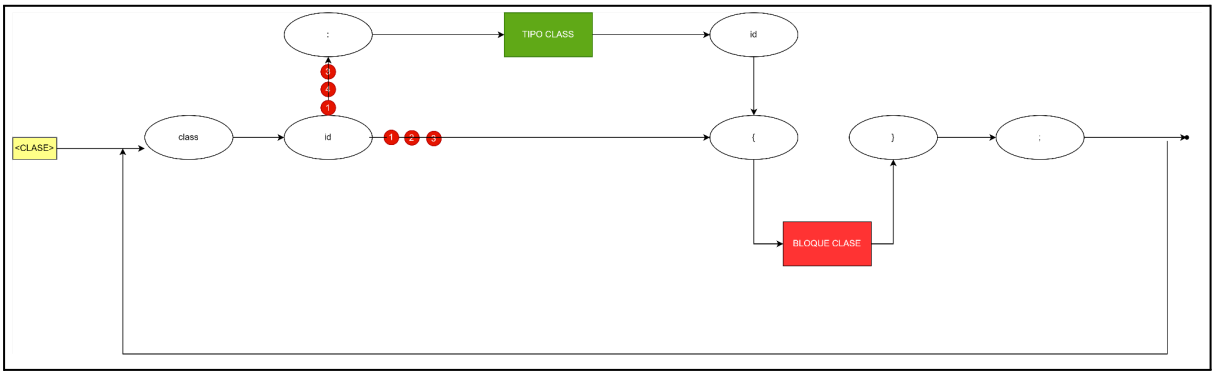
Los diagramas sintácticos son una forma de representar una gramática libre de contexto. Los que fueron usados para la implementación de nuestro lenguaje son los que se mostraran a continuación y se dará una breve explicación de las acciones semánticas.



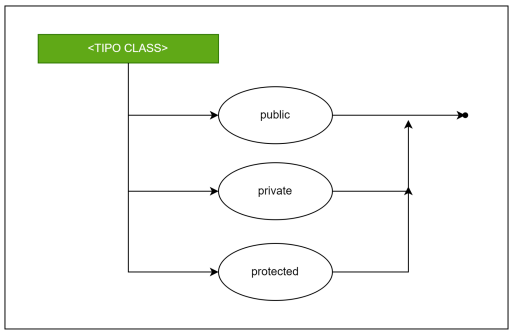
Punto	Descripción
1	Crea el primer cuádruplo necesario, es decir, el GOTO main.

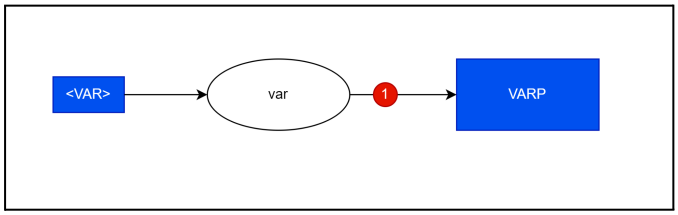


Punto	Descripción
1	Rellena el cuádruplo de main con la línea en donde empieza el main.
2	Crea el último cuádruplo del programa. También escribe en el archivodataVirtualMachine.txt la tabla de constantes, los cuadrupos y el directorio de funciones.

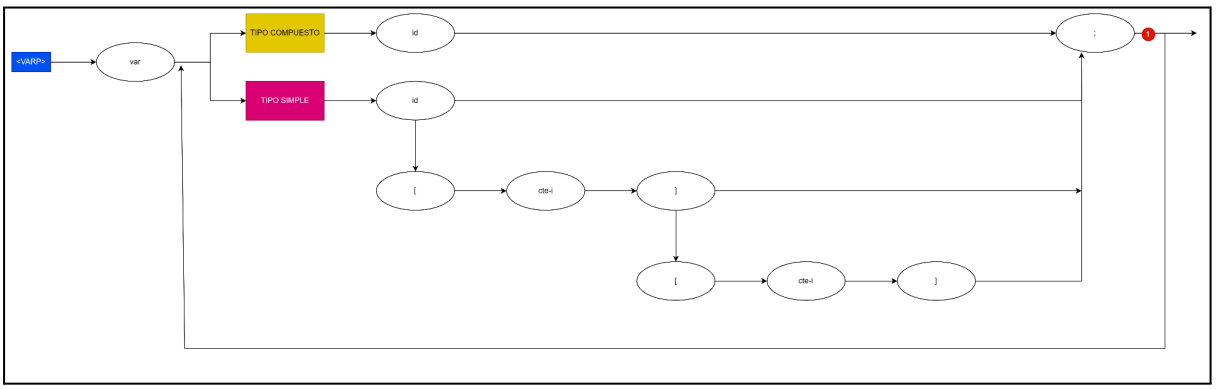


Punto	Descripción
1	Guarda el nombre de la clase
2	Asigna que la clase es de tipo padre
3	Guarda el nombre de la clase y el tipo de clase en la tabla de clases
4	Asigna que la clase es de tipo hijo





Punto	Descripción
1	Asigna un scope global a esa variable



Punto	Descripción
1	Asigna la dirección de memoria a la variable y la guarda en el diccionario correspondiente.