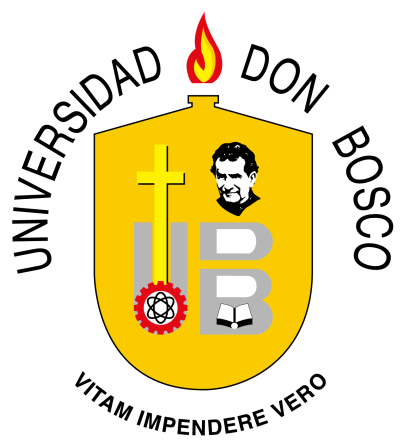
**UNIVERSIDAD DON BOSCO**

**FACULTAD DE INGENIERIA**

**ESCUELA DE COMPUTACION**



***Fase Final - Proyecto de Catedra Compilador DFD***

***Periodo III***

**Presenta:**

**PM090058 De Paz Muñoz, Silvia Johanna**

**FA090693 Franco Arévalo, Jaime Alejandro**

**FR090010 Fuentes Ruiz, Mario Miguel**

**Materia:**

**Compiladores**

**Docente:**

**Ing. Jaime Anaya**

**Fecha:**

**Viernes 21 de septiembre de 2012**

INDICE

**INTRODUCCION** 3

**OBJETIVOS DEL PROYECTO** 4

General 4

Específicos 4

**DESARROLLO** 5

1. MARCO TEORICO 5

1.1 Compiladores en la historia 5

1.2 ¿Qué es un compilador? 6

1.2.1 Elementos de un Compilador 6

1.2.2 Fases del proceso de compilación 7

1.3 ¿Qué es un Diagrama de Flujo? 9

1.3.1 Características 9

1.3.2 Acciones previas a la construcción de un diagrama de flujo 9

1.3.3 Construcción de un diagrama de flujo 9

1.3.4 Ventajas de los diagramas de flujo 10

1.3.5 Símbolos representativos 10

1.4 MSIL 11

1.5 Librería IL Generator 11

2. PROPUESTA DE PROYECTO 12

2.1 Idea General del Proyecto 12

2.2 Propósito del proyecto 12

2.3 Resultados Esperados 12

2.4 Lenguaje de programación a utilizar 13

2.5 Planificación 13

3. DESARROLLO DEL PROGRAMA 14

3.1 Clases 15

3.1.1 Tipos 15

3.1.2 UCDFD 16

3.1.3 Tabla de errores 38

3.1.4 Tabla de símbolos 40

3.1.5 Analizador Lexico 41

3.1.6 Analizador Sintactico 48

3.1.7 Analizador Semantico 62

3.1.8 ElementoDFD 67

3.1.9 Código fuente de la clase Token 69

3.1.10 Código Fuente del Diccionario de tokens 70

3.1.11 Clase de valores globales 72

3.1.12 Clase Generar Codigo 73

3.2 Formularios 82

3.2.1 Formulario Principal 82

4. IMPRESIONES DE PANTALLA 85

**CONCLUSIONES** 87

**REFERENCIAS** 88

Manual de Usuario DFD 89

# **INTRODUCCION**

En el presente documento se desarrollan los principios teóricos necesarios para la creacion de un compilador, los elementos y las fases que se necesitan en un proceso de compilación. Posteriormente se define el planteamiento de un proyecto de cátedra que lleva como elemento principal un compilador. El proyecto propuesto es un sistema que permite la creación de un ejecutable a partir de un DFD. A continuacion se desarrollan todos los componentes de manera teorica y codificada.

# **OBJETIVOS DEL PROYECTO**

## General

* Desarrollar un compilador DFD , como herramienta educativa y de investigacion.

## Específicos

* Poner en práctica el contenido desarrollado en la materia por medio de un proyecto de cátedra.
* Comprender el funcionamiento de un compilador desde sus fundamentos más básicos.
* Desarrollar un compilador en todas sus etapas.

# **DESARROLLO**

## MARCO TEORICO

### Compiladores en la historia

En 1946 se desarrolló la primera computadora digital. En un principio, estas máquinas ejecutaban instrucciones consistentes en códigos numéricos que señalaban a los circuitos de la máquina los estados correspondientes a cada operación, lo que se denominó lenguaje máquina.

Pronto los primeros usuarios de estos ordenadores descubrieron la ventaja de escribir sus programas mediante claves más fáciles de recordar que esos códigos; al final, todas esas claves juntas se traducían manualmente a lenguaje máquina. Estas claves constituyen los llamados lenguajes ensambladores.

Pese a todo, el lenguaje ensamblador seguía siendo el de una máquina, pero más fácil de manejar. Los trabajos de investigación se orientaron hacia la creación de un lenguaje que expresara las distintas acciones a realizar de una manera lo más sencilla posible para una persona. El primer compilador fue escrito por Grace Hopper, en 1952 para el lenguaje de programación A-0. En 1950 John Backus dirigió una investigación en IBM sobre un lenguaje algebraico. En 1954 se empezó a desarrollar un lenguaje que permitía escribir fórmulas matemáticas de manera traducible por un ordenador; le llamaron FORTRAN (FORmulae TRANslator). Fue el primer lenguaje de alto nivel y se introdujo en 1957 para el uso de la computadora IBM modelo 704.

Surgió así por primera vez el concepto de un traductor como un programa que traducía un lenguaje a otro lenguaje. En el caso particular de que el lenguaje a traducir es un lenguaje de alto nivel y el lenguaje traducido de bajo nivel, se emplea el término compilador.

La tarea de realizar un compilador no fue fácil. El primer compilador de FORTRAN tardó 18 años-persona en realizarse y era muy sencillo. Este desarrollo de FORTRAN estaba muy influenciado por la máquina objeto en la que iba a ser implementado. Como un ejemplo de ello tenemos el hecho de que los espacios en blanco fuesen ignorados, debido a que el periférico que se utilizaba como entrada de programas (una lectora de tarjetas perforadas) no contaba correctamente los espacios en blanco.

El primer compilador autocontenido, es decir, capaz de compilar su propio código fuente fue el creado para Lisp por Hart y Levin en el MIT en 1962. Desde 1970 se ha convertido en una práctica común escribir el compilador en el mismo lenguaje que este compila, aunque Pascal y C han sido alternativas muy usadas.

Crear un compilador auto contenido genera un problema llamado bootstrapping, es decir el primer compilador creado para un lenguaje tiene que o bien ser compilado por un compilador escrito en otro lenguaje o bien compilado al ejecutar el compilador en un intérprete.

### ¿Qué es un compilador?

Un compilador es un programa informático que traduce un programa escrito en un lenguaje de programación a otro lenguaje de programación, generando un programa equivalente que la máquina será capaz de interpretar. Usualmente el segundo lenguaje es lenguaje de máquina, pero también puede ser un código intermedio (bytecode), o simplemente texto. Este proceso de traducción se conoce como compilación.

Un compilador es un programa que permite traducir el código fuente de un programa en lenguaje de alto nivel, a otro lenguaje de nivel inferior (típicamente lenguaje de máquina). De esta manera un programador puede diseñar un programa en un lenguaje mucho más cercano a cómo piensa un ser humano, para luego compilarlo a un programa más manejable por una computadora.

### Elementos de un Compilador

Es frecuente que, además del compilador, se utilicen otros programas para crear un código objeto ejecutable. Un esquema típico es el que se describe a continuación.

La estructura del programa fuente se escribe, usando algún programa de edición de texto y puede incluir texto en lenguaje fuente y algunas órdenes para el preprocesador. Este realizará algunas tareas, como eliminación de comentarios, expansión de macros (#IF. . .), inclusión de archivos (#include . . .), sustitución de constantes (#define . . .), o algunas extensiones del lenguaje fuente.

El compilador traducirá el resultado del preproceso obteniendo un programa equivalente en lenguaje ensamblador, que a su vez será traducido por el ensamblador a código máquina relocalizable, en el cual las direcciones serán relativas a ciertas posiciones de origen, y quizás algunas llamadas a rutinas no estén resueltas.

Finalmente, el editor de carga y enlace (o “montador”, o “link”) resolverá las llamadas a rutinas, incluyéndolas a partir de otros objetos de biblioteca si procede, y obtendrá direcciones absolutas, de modo que ya se dispondrá del código máquina absoluto ejecutable.

### Fases del proceso de compilación

* Analizador Léxico:

Realiza un análisis del archivo. La cadena de entrada se lee e izquierda a derecha y se va agrupando en componentes léxicos, que son secuencias de caracteres con un significado colectivo. Por ejemplo, identificadores, palabras reservadas, signos de final de instrucción. Cada componente es asociada a la categoría que pertenece.

* Analizador Sintáctico

Realiza un análisis jerárquico agrupado de los componentes léxicos en frases gramaticales que el compilador utiliza.

* Analizador Semántico

Busca errores semánticos, reúne información sobre los tipos; identifica operadores en base al árbol sintáctico producido en el análisis anterior Ejemplo de error: operación entre tipos de datos incompatibles, rangos permitidos existencia de variables. En cualquiera de estos tres análisis pueden producirse errores.

* Generador de código intermedio

Algunos compiladores generan una representación explicita del programa fuente. Este código es independiente de la máquina y a veces se usa en un conjunto con intérpretes, en lenguajes independientes de la plataforma como JAVA. Esta representación debe ser fácil de producir, ayudar a la optimización y fácil de traducir al programa objeto.

* Optimización

Esta fase trata de mejorar el código intermedio, o las estructuras que generaran el código definitivo de modo de que resulte un código de maquina más rápido de ejecutar para guardarlos valores calculados por cada instrucción.

* Generador de código

Esta fase final de un compilador. Genera el código objeto, que por lo general consiste en un código de maquina relocalizable o código ensamblador. Las posiciones de memoria relativas se seleccionan para cada variable. El uso de los registros de la CPU es relevante.

### ¿Qué es un Diagrama de Flujo?

Un diagrama de flujo es una representación gráfica de un algoritmo o proceso. Se utiliza en disciplinas como la programación, la economía, los procesos industriales y la psicología cognitiva. Estos diagramas utilizan símbolos con significados bien definidos que representan los pasos del algoritmo, y representan el flujo de ejecución mediante flechas que conectan los puntos de inicio y de término.

### Características

Un diagrama de flujo siempre tiene un único punto de inicio y un único punto de término. Además, todo camino de ejecución debe permitir llegar desde el inicio hasta el término.

### Acciones previas a la construcción de un diagrama de flujo

* Identificar las ideas principales a ser incluidas en el diagrama de flujo. Deben estar presentes el dueño o responsable del proceso, los dueños o responsables del proceso anterior y posterior y de otros procesos interrelacionados, otras partes interesadas.
* Definir qué se espera obtener del diagrama de flujo.
* Identificar quién lo empleará y cómo.
* Establecer el nivel de detalle requerido.
* Determinar los límites del proceso a describir.

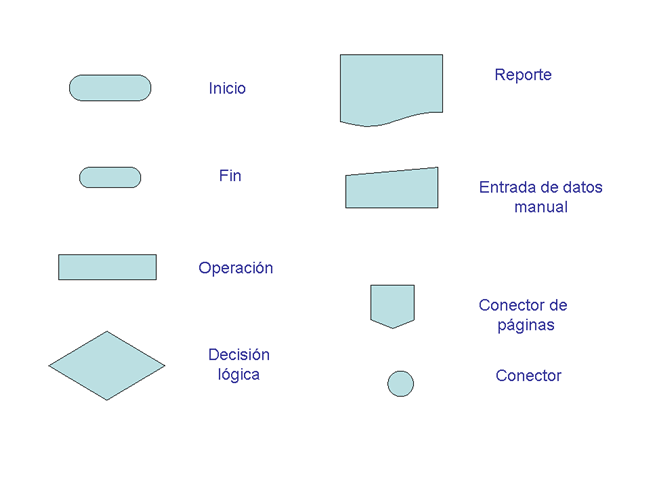
### Construcción de un diagrama de flujo

* Establecer el alcance del proceso a describir. De esta manera quedará fijado el comienzo y el final del diagrama. Frecuentemente el comienzo es la salida del proceso previo y el final la entrada al proceso siguiente.
* Identificar y listar las principales actividades/subprocesos que están incluidos en el proceso a describir y su orden cronológico.
* Si el nivel de detalle definido incluye actividades menores, listarlas también.
* Identificar y listar los puntos de decisión.
* Construir el diagrama respetando la secuencia cronológica y asignando los correspondientes símbolos.
* Asignar un título al diagrama y verificar que esté completo y describa con exactitud el proceso elegido.

### Ventajas de los diagramas de flujo

* Favorecen la comprensión del proceso a través de mostrarlo como un dibujo. El cerebro humano reconoce fácilmente los dibujos. Un buen diagrama de flujo reemplaza varias páginas de texto.
* Permiten identificar los problemas y las oportunidades de mejora del proceso. Se identifican los pasos redundantes, los flujos de los re-procesos , los conflictos de autoridad, las responsabilidades, los cuellos de botella, y los puntos de decisión.
* Muestran las interfaces cliente-proveedor y las transacciones que en ellas se realizan, facilitando a los empleados el análisis de las mismas.
* Son una excelente herramienta para capacitar a los nuevos empleados y también a los que desarrollan la tarea, cuando se realizan mejoras en el proceso.

### Símbolos representativos



### MSIL

MSIL es el acrónimo de Microsoft Intermediate Language, los lenguajes de programación basados en el .Net Framework tales como C#, VisualBasic.Net compilan a MSIL (también llamado CIL, Common Intermediate Language en la actualidad). Este es un lenguaje ensamblador de bajo nivel orientado a objetos y además basado en pilas.

Al momento de compilar lenguajes de programación .Net se convierte el código fuente a código CIL y no en código objeto, CIL es independiente de la plataforma, es un lenguaje intermedio común a todos los sistemas operativos que soporten .Net Framework y puede ser ejecutado en cualquier CPU.

MSIL incluye instrucciones para cargar, almacenar, inicializar y llamar a métodos en los objetos, así como instrucciones para operaciones lógicas y aritméticas, flujo de control, acceso directo a la memoria, control de excepciones y otras operaciones.

Para poder ejecutar código, se convierte el código MSIL a código específico del CPU, esto gracias al CLR (Common Language Runtime), quien vuelve a compilar optimizando generando código nativo por medio uno o varios compiladores JIT (Just in Time) proporcionados por CLR. JIT compila en tiempo de ejecución a medida que corre el programa. El compilador JIT examina los metadatos (información acerca de las clases compiladas) del ensamblado en busca de accesos ilegales y maneja las violaciones apropiadamente. El proceso de lectura de metadatos se llama reflexión.

### Librería IL Generator

ILGenerator es una clase que genera instrucciones maquina del lenguaje intermedio de Microsoft (MSIL). ILGenerator hace parte del espacio de nombres Reflection.Emit el cual nos permite emitir código MSIL de manera que se ejecute directamente, es decir se puede embeber código de ensamblador en el código y luego invocarlo para ser ejecutado. Cuando se compila cualquier lenguaje de alto nivel basado en .NET framework este pasa primero a MSIL, luego cuando se invoca un ensamblado MSIL se convierte y es ejecutado por el lenguaje máquina del CPU.

En el espacio de nombres System.Reflection se encuentran clases que nos permiten acceder a cualquier tipo de estructuras dentro de un programa tales como clases, tipos, campos, enumeradores, miembros y métodos.

La clase ILGenerrator.Opcodes genera las instrucciones MSIL y contiene todos los códigos de operación MSIL necesarios. Las instrucciones mas utilizadas son Emit (OpCode, Type) el cual coloca la instrucción máquina especificada en la secuencia del Lenguaje intermedio de Microsoft (MSIL) seguida del símbolo (token) de metadatos del tipo indicado; EmitCall la cual coloca una instrucción máquina call o callvirt en la secuencia del Lenguaje intermedio de Microsoft (MSIL) para llamar a un método varargs.

## PROPUESTA DE PROYECTO

### Idea General del Proyecto

Se pretende implementar un compilador con interfaz grafica que permita generar un archivo ejecutable a partir de un diagrama de flujo de datos o DFD, realizando todas las etapas de análisis y generación de codigo de dicho compilador.

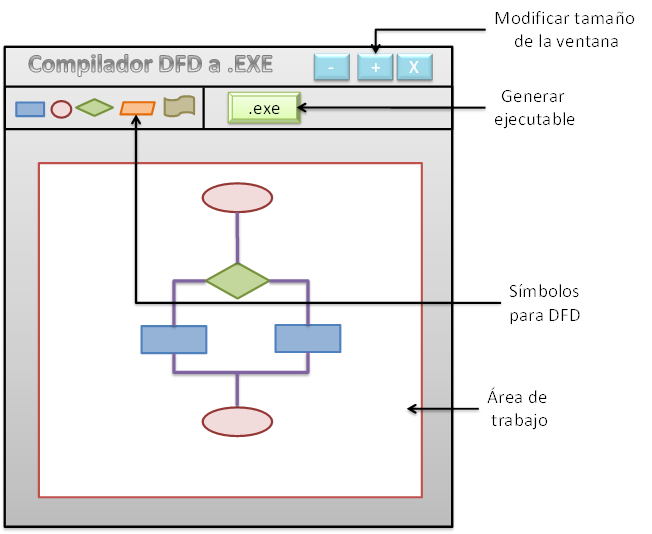
### Propósito del proyecto

Generar una herramienta que permita generar ejecutables con solo diseñar el algoritmo y realizar un diagrama de flujo de datos.

### Resultados Esperados

Una aplicación con un entorno amigable de desarrollo, que sea fácil de entender y usar , ademas que permita generar un ejecutable de un programa a partir de un DFD.

A continuación se muestra un prototipo de dicha aplicación:



### Lenguaje de programación a utilizar

Para el desarrollo de esta aplicación se utilizara C# de Visual Studio 2010, en el Framework .NET, debido al dominio de dicho lenguaje y de la facilidad de crear aplicaciones con entorno gráfico, ademas de poder explotar todas las ventajas de lo que es .NET.

### Planificación



## DESARROLLO DEL PROGRAMA

Como antes se menciona se realizara el proyecto de un DFD para ello tambien se debe poseer cierto codigo para poder procesar el diagrama por lo cual una vez hecho el diagrama este generara codigo para poder ser compilado una vez generado el codigo pasaremos al analisis Lexico, para ello se necesita una tabla de token iniciales que seran nuestros delimitantes al analizar el codigo tales como variables, datos, palabras reservadas entre otros.

Ahora bien para comenzar se presenta la tabla de todos los tokens considerados hasta el momento para nuestro compilador de DFD.

|  |  |
| --- | --- |
| Simbolo | Tipo/Token |
| ( | Signo Separador-SS |
| ) | Signo Separador-SS |
| ! |  |
| + | Operador -OP |
| - | Operador -OP |
| \* | Operador -OP |
| / | Operador -OP |
| && | Operador -OP |
| || | Operador-OP |
| ++ | Operador -OP |
| -- | Operador -OP |
| = | Operador -OP |
| == | Signo Comparador -SCOM |
| <= | Signo Comparador -SCOM |
| >= | Signo Comparador -SCOM |
| < | Signo Comparador -SCOM |
| > | Signo Comparador -SCOM |
| != | Signo Comparador -SCOM |
| ! | Signo Relacionador –SR |
| , | Signo Puntuacion –SP |
| . | Signo Punto Flotante -SF |
| identificador | Variable o identificador –ID |
| numero entero | Numero entero-NE |
| numero decimal | Numero decimal-ND |
|  | Simbolo desconocido –SD |
| cadena | Cadena-CA |
| variableNumerica | Variable numerica -VN |
| variableCadena | Variable cadena -VC |

Ahora una vez definidos nuestros caracteres se procede a crear un diccionario de datos para alamacenar los tokens y tener una forma de controlar las palabras y simbolos una vez el analizador lexico los pida.

### Clases

### Tipos

La clase tipos nos permite determinar el tipo es en la cual se encuentra los tipos de elementos que existen dentro del DFD siendo estos el punto de partida para la identificacion de las acciones que se realizaran en analisis sintactico y semantico.

Ademas de ser de vital importancia para generar una adeacuada traduccion a lo que es el MSIL.

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace CompiladorDFD  {  //Elementos utilizados para declarar el tipo de elemento que repredsentara  public enum Elemento {  none,//No tiene un tipo definido  inicio,//Inicio del grafo  Asignacion,//Asignacion de variables  Eif,//Estructura If  EndIf,//Fin del if  EWhile,//Estructura While  EndWhile,//End While  Efor,//Estructura for  Endfor,//FIn de la estructura for  Lectura,//Leer un dato de teclado  Escritura,//Escribir datos en pantalla  fin  }  } |

### UCDFD

Se utiliza para generar el analisis sintactico y generar la estructura del DFD de tal manera que se permita obtener una estructura acorde a las necesidades que genera la gramatica de un DFD ademas de generar toda la interfaz grafica que se utiliza dentro del programa.

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Drawing;  using System.Data;  using System.Linq;  using System.Text;  using System.Windows.Forms;  using System.Drawing.Drawing2D;  namespace CompiladorDFD  {  public partial class UCDFD : UserControl  {//Enum utilizado para saber el estado del control  private enum Estado {  Normal,  Agregando,  Eliminando,  Moviendo,  }  //Posiciones del Mouse dentro del control  private int posMouseX=0;  private int posMouseY=0;  //Estado dentro del control segun accion a desempeniar  private Estado estado;  //Variable utilizada para saber cuando se esta agregando un elemento  private ElementoDFD elementoAgregado = null;  //Declaracion de variables a utilizar para controlar las lineas  private Color colorLinea = Color.Black;  private int grosorLinea = 1;  private int grosorFiguras = 1;  private Color colorFondo = Color.White;  private Color colorLetra = Color.Black;  private Color colorEliminar = Color.Red;  private Font fontLetra = new Font("Arial", 8);  //Declaracion de las variables a ser utilizadas por el control  //private UCElementos raiz;//Raiz del grafo a crear desde un nodo inicio  private ElementoDFD elementoRaiz;//Raiz del grafo utilizado para tener una referencia hacia todos ya que es el inicio  //Lista para utilizarse para recorrer todos los elementos creados  //Variables para declarar los valores de las dimensiones de los elementos  private Size tamanioAsignacion = new Size(120, 60);  private Size tamanioIf = new Size(90, 60);  private Size tamanioFor = new Size(140, 30);  private Size tamanioWhile = new Size(140, 30);  private Size tamanioInicio = new Size(60, 60);  private Size tamanioFin = new Size(60, 60);  private Size tamanioLectura = new Size(120, 60);  private Size tamanioEscritura = new Size(120, 60);  private Size tamanioEndIF = new Size(50, 0);  private Size tamanioEndFor = new Size(40, 40);  private Size tamanioEndWhile = new Size(40, 40);  private List<ElementoDFD> listadoElementos = new List<ElementoDFD>();  //---------------------------------------------------------------  // Elementos para utilizar el doble buffer  //---------------------------------------------------------------  Graphics GX;  BufferedGraphicsContext bgc = new BufferedGraphicsContext();  BufferedGraphics bg;  //---------------------------------------------------------------  public UCDFD()  {  InitializeComponent();  //Utilizado para permitir el doble buffer este optimizado  this.SetStyle(ControlStyles.OptimizedDoubleBuffer, true);  //Se crean los elementos iniales dentro del grafo  this.BackColor = colorFondo;  }  public ElementoDFD ObtenerRaiz(){  return elementoRaiz;  }  private void UCDFD\_Load(object sender, EventArgs e)  { //Se colocan las barras de dezplazamiento  this.AutoScrollMinSize = new Size(3000, 3000);  this.HorizontalScroll.Enabled = true;  this.HorizontalScroll.Minimum = 0;  this.HorizontalScroll.Maximum = 3000;  //Se crea el principio del grafo que se utilizara dentro del programa  //se agrega el inicio  //AgregarElemento(null, CrearElemento(Elemento.inicio, imgInicio));  AgregarElemento(null, null, CrearElementoDFD(Elemento.inicio));  AgregarElemento(elementoRaiz, elementoRaiz.centro, CrearElementoDFD(Elemento.fin));  //se agrega el final  //AgregarElemento(raiz, CrearElemento(Elemento.fin, imgFin));  //Se ajustan los elementos  //ReajustarElementos(raiz);  ReajustarElementos(elementoRaiz);  Dibujar();  }  private string CargarDataDifurcacion;  private void cargarData(ref string tempDatos,ref ElementoDFD temp) {  if (elementoAgregado != null)  {  if (temp.tipo != Elemento.Eif)  {  AgregarElmentoContinuo(temp);  temp = temp.centro;  temp.datos = tempDatos;  tempDatos = "";  elementoAgregado = null;  }  else {  if (CargarDataDifurcacion == "izquierda") {  temp.izquierda = new ElementoDFD();  AgregarElementoDentroIf(temp, ref temp.izquierda);  temp = temp.izquierda;  }  else if (CargarDataDifurcacion == "derecha") {  temp.derecha = new ElementoDFD();  AgregarElementoDentroIf(temp,ref temp.derecha);  temp = temp.derecha;  }  temp.datos = tempDatos;  if (temp.tipo == Elemento.Eif) {  temp.centro.centro = temp.padre.centro;  }  else  {  temp.centro = temp.padre.centro;  }  tempDatos = "";  elementoAgregado = null;  }  }  }  //Codigo utilizado para recargar un nuevo DFD desde un archivo  public void CargarDFD(List<string> listaCodigo) {    string tempDatos = "";  elementoRaiz = null;  elementoAgregado = null;  Stack<ElementoDFD> ElementoIF = new Stack<ElementoDFD>();  Stack<String> Direccion = new Stack<string>();  //AgregarElemento(null, CrearElemento(Elemento.inicio, imgInicio));  AgregarElemento(null, null, CrearElementoDFD(Elemento.inicio));  AgregarElemento(elementoRaiz, elementoRaiz.centro, CrearElementoDFD(Elemento.fin));  ElementoDFD temp = elementoRaiz;  foreach (string tipo in listaCodigo) {  switch (tipo) {  case "inicio"://Inicio del grafo  break;  case "Asignacion"://Asignacion de variable  cargarData(ref tempDatos, ref temp);  elementoAgregado = new ElementoDFD();  elementoAgregado.tipo = Elemento.Asignacion;  break;  case "Eif"://Estructura If  cargarData(ref tempDatos, ref temp);  elementoAgregado = new ElementoDFD();  elementoAgregado.tipo = Elemento.Eif;  break;  case "derecha":  cargarData(ref tempDatos, ref temp);  ElementoIF.Push(temp);  CargarDataDifurcacion = "derecha";  Direccion.Push("derecha");  break;  case "izquierda":  cargarData(ref tempDatos, ref temp);  temp = ElementoIF.Peek();  CargarDataDifurcacion = "izquierda";  break;  case "EndIf"://Fin del if  cargarData(ref tempDatos, ref temp);  temp =ElementoIF.Pop().centro;  Direccion.Pop();  if (Direccion.Count > 0) CargarDataDifurcacion = Direccion.Peek();  break;  case "EWhile"://Estructura While  cargarData(ref tempDatos, ref temp);  elementoAgregado = new ElementoDFD();  elementoAgregado.tipo = Elemento.EWhile;  break;  case "EndWhile"://End While  cargarData(ref tempDatos, ref temp);  break;  case "Efor"://Estructura for  cargarData(ref tempDatos, ref temp);  elementoAgregado = new ElementoDFD();  elementoAgregado.tipo = Elemento.Efor;  break;  case "Endfor"://FIn de la estructura for  cargarData(ref tempDatos, ref temp);  break;  case "Lectura"://Leer un dato de teclado  cargarData(ref tempDatos, ref temp);  elementoAgregado = new ElementoDFD();  elementoAgregado.tipo = Elemento.Lectura;  break;  case "Escritura"://Escribir datos en pantalla  cargarData(ref tempDatos, ref temp);  elementoAgregado = new ElementoDFD();  elementoAgregado.tipo = Elemento.Escritura;  break;  case "fin":  cargarData(ref tempDatos,ref temp);  break;  default:  tempDatos += tipo + '\n';  break;  }  }  ReajustarElementos(elementoRaiz);  Dibujar();  }  public void Iniciar() {  this.HorizontalScroll.Value = 1500;  ReajustarElementos(elementoRaiz);  }  //-------------------------------------------------------------------------------------  // Funciones externas para realizar acciones sobre el control  //-------------------------------------------------------------------------------------  //Funcion para permitir agregar un elemnto dentro del grafo  public void AgregandoElemento(Elemento tipo) {  elementoAgregado = new ElementoDFD();  elementoAgregado.width = 15;  elementoAgregado.height = 15;  elementoAgregado.tipo = tipo;  estado = Estado.Agregando;  }  public void Eliminar() {  estado = Estado.Eliminando;  }    private bool ConcordanciaEjeX(int posX, ElementoDFD tempElemento, ElementoDFD tempElemento2) {  //Funcion que verifica si el elemento esta entre los dos elementos dados dentro de el eje de X  return (posX > tempElemento.left && posX < tempElemento.left + tempElemento.width) || (posX > tempElemento2.left && posX < tempElemento2.left + tempElemento2.width);  }  private bool ConcordanciaEjeY(int posY, ElementoDFD tempElemento,ElementoDFD tempElemento2) {  return posY > tempElemento.top && posY < tempElemento2.top;  }  private void AgregarElmentoContinuo(ElementoDFD temp) {  switch (elementoAgregado.tipo)  {  case Elemento.Eif:  ElementoDFD elementoIF = CrearElementoDFD(elementoAgregado.tipo);  AgregarElemento(temp, temp.centro, elementoIF);  AgregarElemento(elementoIF, elementoIF.centro, CrearElementoDFD(Elemento.EndIf));  break;  case Elemento.Efor:  ElementoDFD elementoFor = CrearElementoDFD(elementoAgregado.tipo);  AgregarElemento(temp, temp.centro, elementoFor);  AgregarElemento(elementoFor, elementoFor.centro, CrearElementoDFD(Elemento.Endfor));  elementoFor.fin = elementoFor.centro;  break;  case Elemento.EWhile:  ElementoDFD elementoWhile = CrearElementoDFD(elementoAgregado.tipo);  AgregarElemento(temp, temp.centro, elementoWhile);  AgregarElemento(elementoWhile, elementoWhile.centro, CrearElementoDFD(Elemento.EndWhile));  elementoWhile.fin = elementoWhile.centro;  break;  default:  AgregarElemento(temp, temp.centro, CrearElementoDFD(elementoAgregado.tipo));  break;  }    }  private void AgregarElementoDentroIf(ElementoDFD padre,ref ElementoDFD difuracacion)  {  if (difuracacion == null)  {  switch (elementoAgregado.tipo)  {  case Elemento.Eif:  difuracacion = padre.centro;  break;  default:  difuracacion = padre.centro;  break;  }  }    switch (elementoAgregado.tipo)  {  case Elemento.Eif:  ElementoDFD elementoIF = CrearElementoDFD(elementoAgregado.tipo);  AgregarElemento(padre, difuracacion, elementoIF);  AgregarElemento(elementoIF, elementoIF.centro, CrearElementoDFD(Elemento.EndIf));  break;  case Elemento.Efor:  ElementoDFD elementoFor = CrearElementoDFD(elementoAgregado.tipo);  AgregarElemento(padre, difuracacion, elementoFor);  AgregarElemento(elementoFor, elementoFor.centro, CrearElementoDFD(Elemento.Endfor));  elementoFor.fin = elementoFor.centro;  break;  case Elemento.EWhile:  ElementoDFD elementoWhile = CrearElementoDFD(elementoAgregado.tipo);  AgregarElemento(padre, difuracacion, elementoWhile);  AgregarElemento(elementoWhile, elementoWhile.centro, CrearElementoDFD(Elemento.EndWhile));  elementoWhile.fin = elementoWhile.centro;  break;  default:  AgregarElemento(padre, difuracacion, CrearElementoDFD(elementoAgregado.tipo));  break;  }    }  private void Agregando\_Click\_if(ElementoDFD tempElemento) {  //Se calcula las posiciones en las que se desea agregar el elemento  int posX = elementoAgregado.left + elementoAgregado.width / 2;  int posY = elementoAgregado.top + elementoAgregado.height / 2;    //Variable temporal para almacenar una referencia  ElementoDFD temp = tempElemento;  //Se verifica que el elemento no sea null  if (temp != null) {  //se verifica si el elemento es un if  if (temp.tipo == Elemento.Eif)  { //Si se encuentra entre el if  if (ConcordanciaEjeY(posY, temp,temp.centro))  { //El click fue a la derecha  if (temp.left + temp.width / 2 < posX)  {  if (temp.derecha == null)  {  AgregarElementoDentroIf(temp, ref temp.derecha);  return;  }  else {  if (ConcordanciaEjeY(posY, temp,temp.derecha))  {  if(ConcordanciaEjeX(posX,temp,temp.derecha)){  AgregarElementoDentroIf(temp, ref temp.derecha);  return;  }  }  }  Agregando\_Click\_if(temp.derecha);  return;  }//El click fue a la izquierda  else {  if (temp.izquierda == null)  {  AgregarElementoDentroIf(temp, ref temp.izquierda);  return;  }  else {  if(ConcordanciaEjeY(posY,temp,temp.izquierda)){  if(ConcordanciaEjeX(posX,temp,temp.izquierda)){  AgregarElementoDentroIf(temp,ref temp.izquierda);  return;  }  }  }  }  Agregando\_Click\_if(temp.izquierda);  return;  }//De no ser asi no se encuentra dentro del if      }  else {  if (ConcordanciaEjeY(posY, temp, temp.centro))  {  if (ConcordanciaEjeX(posX, temp, temp.centro))  {  AgregarElmentoContinuo(temp);  return;  }  }  }  Agregando\_Click\_if(temp.centro);  }    }  private void Agregando\_Click() {  if (elementoAgregado != null) {  ElementoDFD tempElemento = elementoRaiz;  int posX = elementoAgregado.left + elementoAgregado.width / 2;  int posY = elementoAgregado.top + elementoAgregado.height / 2;  while (tempElemento.tipo != Elemento.fin) {  if( posY> tempElemento.top && posY<tempElemento.centro.top)  if (tempElemento.tipo == Elemento.Eif) {  Agregando\_Click\_if(tempElemento);  ReajustarElementos(elementoRaiz);  }else  if (ConcordanciaEjeX(posX,tempElemento,tempElemento.centro)){  AgregarElmentoContinuo(tempElemento);  ReajustarElementos(elementoRaiz);  break;  }  tempElemento = tempElemento.centro;  }  //Se limpia la bandera para agregar controles  estado = Estado.Normal;  elementoAgregado = null;  }  }  //------------------------------------------------------------------  // Funciones para eliminar los elementos  //------------------------------------------------------------------  private bool EliminarDifuracion(ElementoDFD tempElemento,ElementoDFD paro) {  if (tempElemento == null) return false;  if (tempElemento != paro) {  EliminarElementos(tempElemento);  }  return true;  }  private bool EliminarElementos(ElementoDFD tempElemento) {  if (tempElemento != null) {  switch (tempElemento.tipo) {  case Elemento.Eif:  //Eliminado elementos intermedios  EliminarDifuracion(tempElemento.izquierda, tempElemento.centro);  EliminarDifuracion(tempElemento.derecha, tempElemento);  EliminarElementos(tempElemento.centro);  //Eliminado elemento if  EliminarElemento(tempElemento);  break;  case Elemento.Efor:  case Elemento.EWhile:  ElementoDFD temp = tempElemento.centro;  //Mientras no llege al end if  while (temp != tempElemento.fin) {  ElementoDFD sig;  if (temp.tipo == Elemento.Eif) sig = temp.centro.centro;  else if (temp.tipo == Elemento.Efor) sig = temp.fin.centro;  else if (temp.tipo == Elemento.EWhile) sig = temp.fin.centro;  else sig = temp.centro;    EliminarElementos(temp);  temp = sig;  }  EliminarElemento(tempElemento.fin);  EliminarElemento(tempElemento);  break;  default:  EliminarElemento(tempElemento);  break;    }  return true;  }  return false;  }  private void EliminarElemento(ElementoDFD tempElemento) {    if (tempElemento.padre.tipo == Elemento.Eif && tempElemento.tipo != Elemento.EndIf)  {  if (tempElemento.centro == tempElemento.padre.centro)  {  if (tempElemento.padre.derecha == tempElemento) tempElemento.padre.derecha = null;  else if (tempElemento.padre.izquierda == tempElemento) tempElemento.padre.izquierda = null;  }  else  {  if (tempElemento.padre.derecha == tempElemento) tempElemento.padre.derecha = tempElemento.centro;  else if (tempElemento.padre.izquierda == tempElemento) tempElemento.padre.derecha = tempElemento.centro;  }  tempElemento.centro.padre = tempElemento.padre;  listadoElementos.Remove(tempElemento);  tempElemento = null;  }  else  {  tempElemento.padre.centro = tempElemento.centro;  tempElemento.centro.padre = tempElemento.padre;  listadoElementos.Remove(tempElemento);  tempElemento = null;  }    }  //------------------------------------------------------------------  // Funciones para crear los elementos  //------------------------------------------------------------------  private ElementoDFD CrearElementoDFD(Elemento tipo) {  //Se crea el elemeto a manejar  ElementoDFD tempElemento = new ElementoDFD();  switch (tipo) {  case Elemento.inicio:  tempElemento.Tamanio(tamanioInicio);  break;  case Elemento.fin:  tempElemento.Tamanio(tamanioFin);  break;  case Elemento.Asignacion:  tempElemento.Tamanio(tamanioAsignacion);  break;  case Elemento.Lectura:  tempElemento.Tamanio(tamanioLectura);  break;  case Elemento.Escritura:  tempElemento.Tamanio(tamanioEscritura);  break;  case Elemento.Eif:  tempElemento.Tamanio(tamanioIf);  break;  case Elemento.EndIf:  tempElemento.Tamanio(tamanioEndIF);  break;  case Elemento.Efor:  tempElemento.Tamanio(tamanioFor);  break;  case Elemento.Endfor:  tempElemento.Tamanio(tamanioEndFor);  break;  case Elemento.EWhile:  tempElemento.Tamanio(tamanioWhile);  break;  case Elemento.EndWhile:  tempElemento.Tamanio(tamanioEndWhile);  break;  }  tempElemento.visible = true;  tempElemento.tipo = tipo;  listadoElementos.Add(tempElemento);  //Se retorna el elemento  return tempElemento;  }    //------------------------------------------------------------------  // Funciones para ingresar los elementos dentro del grafo  //------------------------------------------------------------------  //Se pasan los elementos para poder ser asignados correctamente deltro del grafo  private void AgregarElemento(ElementoDFD padre,ElementoDFD hijo,ElementoDFD nuevo) {  //Si no existe un padre se coloca como raiz del grafo entero  if (padre == null)  {  elementoRaiz = nuevo;  }  else  {//Se verifica si existe un hijo  if (hijo != null)  {//Se comienza a verificar que hijo es de los 3 posibles que existen  if (padre.izquierda == hijo) padre.izquierda = nuevo;  else if (padre.derecha == hijo) padre.derecha = nuevo;  else if (padre.centro == hijo) padre.centro = nuevo;  //Se pasa al hijo como referencia  nuevo.centro = hijo;  }  else  {  padre.centro = nuevo;  }  //Se le asigna la referencia la padre  nuevo.padre = padre;  }  }  //------------------------------------------------------------------  // Eventos creados para manejar las funciones hechas sobre los controles  //------------------------------------------------------------------  private void UCDFD\_MouseDoubleClick(object sender, MouseEventArgs e)  { ElementoDFD tempElemento;  switch(estado){  case Estado.Normal:    if ((tempElemento = VerificarAccion(e.X, e.Y)) != null)  {  switch (tempElemento.tipo) {  case Elemento.Asignacion:  case Elemento.Eif:  case Elemento.Escritura:  case Elemento.Lectura:  case Elemento.EWhile:  tempElemento.LlamarFormulario();  break;  }    }  break;  }  }  //--------------------------------------------------------------------------------------  // Acciones hechas para manejar a los tipos de controles  //--------------------------------------------------------------------------------------  //-------------------------------------------------------------------------------------  // Funciones a utilizar para verificar los objetos sobre los que se ejecutan acciones  //-------------------------------------------------------------------------------------  private ElementoDFD VerificarAccion(int x , int y)  {  ElementoDFD resultante = null;  foreach (ElementoDFD temp in listadoElementos)  {  if (temp.VerificarInteraccion(x, y))  {  resultante = temp;  }  }  return resultante;    }    //---------------------------------------------------------------------  // Funciones a utilizar para reacomodar los elementos dentro del grafo  //---------------------------------------------------------------------  void ReajustarElementos(ElementoDFD elemento)  {  ElementoDFD tempElemento = elemento;  //Dezplazamientos realizados  int espaciado = 30;  int x = this.Width / 2+ this.AutoScrollPosition.X + 3000/2 ; //Indica el inicio del dezplazamiento  int y;    y = 10 + this.AutoScrollPosition.Y; //centro de la ventana contenedora  int yMax = 10;  int xMax = x;//Variable a utilizar para los scrools  while (tempElemento != null)  {//Si el elemento es diferente de visible no se dibuja ni integra dentro del diagrama  //Evitando que se ajuste dentro de este  if (tempElemento.visible == true)  {  switch (tempElemento.tipo)  {  case Elemento.Eif:  tempElemento.left = x - tempElemento.width / 2;  tempElemento.top = y;  y += tempElemento.height + espaciado;  ReajustarIf(tempElemento, x, ref y, espaciado);  tempElemento= tempElemento.centro;  break;  default:  tempElemento.left = x - tempElemento.width / 2;  tempElemento.top = y;  y += tempElemento.height + espaciado;  break;  }  }  tempElemento = tempElemento.centro;  }  yMax= y - this.AutoScrollPosition.Y;  //Configuracion de los scrools a utilizar    }  private void ReajustarIf(ElementoDFD tempElemento,int centro,ref int y,int espaciado) {  ElementoDFD temp = tempElemento.izquierda;  int separacion= CalcularSeparacionIzquierda(tempElemento.izquierda);  int tempY = y ;  //Calculando linea izquierda  if (temp != null)  {  while (temp != tempElemento.centro)  {  temp.left = centro - separacion - temp.width / 2;  temp.top = tempY;  tempY += temp.height + espaciado;  if (temp.tipo == Elemento.Eif) ReajustarIf(temp, temp.left + temp.width / 2, ref tempY, espaciado);  temp = temp.centro;  }  }  int tempY2 = y;  //Calculando linea derecha  temp = tempElemento.derecha;  if (temp != null)  {  separacion = CalcularSeparacionDerecha(tempElemento.derecha);  while (temp != tempElemento.centro)  {  temp.left = centro + separacion - temp.width / 2;  temp.top = tempY2;  tempY2 += temp.height + espaciado;  if (temp.tipo == Elemento.Eif) ReajustarIf(temp, temp.left + temp.width / 2, ref tempY2, espaciado);  temp = temp.centro;      }  }  if (tempY > tempY2)  {  y = tempY;  }  else {  y = tempY2;  }    //Confugurando el End if  tempElemento.centro.left = tempElemento.left+ tempElemento.width/2- tempElemento.centro.width/2;  tempElemento.centro.top = y;  y += tempElemento.centro.height + espaciado;    }  //Se calcualan las separaciones dentro de lo que son los if  private int CalcularSeparacionIzquierda(ElementoDFD tempElemento) {  int separacion = 0;  if (tempElemento == null) return 20;  ElementoDFD temp = tempElemento;  while (temp != tempElemento.centro) {  if (temp.padre.tipo == Elemento.Eif) separacion += temp.width;  if (temp.tipo == Elemento.Eif) separacion += CalcularSeparacionIzquierda(temp.derecha);  temp = temp.centro;  }  return separacion;  }  private int CalcularSeparacionDerecha(ElementoDFD tempElemento)  {  int separacion = 0;  if (tempElemento == null) return 20;  ElementoDFD temp = tempElemento;  while (temp != tempElemento.centro )  {  if (temp.padre.tipo == Elemento.Eif) separacion += temp.width;  if (temp.tipo == Elemento.Eif) separacion += CalcularSeparacionDerecha(temp.izquierda);  temp = temp.centro;  }  return separacion;  }  //---------------------------------------------------------------------------  // Funciones para realizar las lineas sobre el grafo y los elementos  //---------------------------------------------------------------------------  private void Dibujar()  {  //Se crea el lienzo en el cual se trabajara  GX = this.CreateGraphics();  bg = bgc.Allocate(GX, new Rectangle(0, 0, this.Width, this.Height));  //Se dibuja un rectangulo en el fondo  SolidBrush Brectangulo = new SolidBrush(colorFondo);  bg.Graphics.FillRectangle(Brectangulo, new Rectangle(0, 0, this.Width, this.Height));  //Se comienzan a pintar las lineas dentro del control  ElementoDFD tempElemento = elementoRaiz; //Se obtiene una referencia al elemento inicial  //se crean variables para manejar los puntos en los cuales se crearan las lineas  //Se crea el lapiza a ser utilizado para dibujar las lineas  Pen lapiz = new Pen(colorLinea, grosorLinea);  Pen lapizFiguras = new Pen(colorLinea, grosorFiguras);  Pen lapizEliminar = new Pen(colorEliminar, grosorFiguras);  //Se recorre todo el grafo hasta llegar al elemento final del grafo que  //es de tipo fin indicando el fin del dibujo del grafo  while (tempElemento.tipo != Elemento.fin)  {  //Se dibuja el elemento a utilizar  if (estado == Estado.Eliminando && tempElemento.tipo!= Elemento.inicio &&tempElemento.VerificarInteraccion(posMouseX,posMouseY))  DibujarElemento(ref bg, tempElemento, lapizEliminar);  else  DibujarElemento(ref bg, tempElemento, lapizFiguras);    //Se dibujan elementos lineales es decir que solo tienen un camnino central  if (tempElemento.centro.tipo != Elemento.none)  {  DibujarLinea(ref bg, tempElemento, lapiz);  }  else  {  }  tempElemento = tempElemento.centro;  }  DibujarElemento(ref bg, tempElemento, lapizFiguras);  //Se dibuja el control que se desea agregar si es que se posee ese evento habilitado  if (estado==Estado.Agregando && elementoAgregado != null) DibujarElemento(ref bg, elementoAgregado, lapizFiguras);  //Se realiza el render de las lineas para poder ser visualizadas todas a la vez  bg.Render();  //Se procede a realizar un dispose sobre todos los elementos que se dejaron de usar para  //poder asi liberar los recursos utilizados por estos  lapiz.Dispose();  bg.Dispose();  GX.Dispose();  Brectangulo.Dispose();  }  //------------------------------------------------------------------  // Funciones para dibujar los elementos a crear  //------------------------------------------------------------------  private void DibujarLinea(ref BufferedGraphics tempbf, ElementoDFD tempElemento, Pen lapiz ) {  //punto inicial y punto final  Point puntoInicial;  Point puntoFinal;  switch (tempElemento.tipo) {  case Elemento.Eif:  //Elementos hacia la izquierda    if (tempElemento.izquierda == null)  {  lapiz.EndCap = LineCap.NoAnchor;    Point izq = new Point(tempElemento.left,tempElemento.top+tempElemento.height/2);  Point izqPlus = new Point(izq.X - 10,izq.Y);  Point izqDown = new Point(izqPlus.X, tempElemento.centro.top);  Point izqFinal = new Point(tempElemento.centro.left,tempElemento.centro.top);  Point[] izqPoints = {izq,izqPlus,izqDown,izqFinal};  bg.Graphics.DrawLines(lapiz, izqPoints);  }  else {  lapiz.EndCap = LineCap.ArrowAnchor;  Point izq2 = new Point(tempElemento.left, tempElemento.top + tempElemento.height / 2);  Point izqPlus2 = new Point(tempElemento.izquierda.left + tempElemento.izquierda.width / 2, tempElemento.top + tempElemento.height / 2);  Point izqDown = new Point(izqPlus2.X, tempElemento.izquierda.top);  Point[] izqPoints2 = { izq2,izqPlus2, izqDown };  bg.Graphics.DrawLines(lapiz, izqPoints2);  }  //Elementos hacia la derecha  if (tempElemento.derecha == null)  {  lapiz.EndCap = LineCap.NoAnchor;  Point der = new Point(tempElemento.left + tempElemento.width, tempElemento.height / 2 + tempElemento.top);  Point derPlus = new Point(der.X+10, der.Y);  Point derDown = new Point(derPlus.X,tempElemento.centro.top);  Point derFinal = new Point(tempElemento.centro.left,tempElemento.centro.top);  Point[] derPoints = {der,derPlus,derDown,derFinal};  bg.Graphics.DrawLines(lapiz, derPoints);  }  else {  lapiz.EndCap = LineCap.ArrowAnchor;  Point der2 = new Point(tempElemento.left + tempElemento.width, tempElemento.height / 2 + tempElemento.top);  Point derPlus2 = new Point(tempElemento.derecha.left + tempElemento.derecha.width / 2, tempElemento.top + tempElemento.height / 2);  Point derDown2 = new Point(derPlus2.X, tempElemento.derecha.top);  Point[] derPoints2 = { der2,derPlus2, derDown2 };  bg.Graphics.DrawLines(lapiz, derPoints2);  }  break;  default:  //Se agrega al lapiz al final una flecha  lapiz.EndCap = LineCap.ArrowAnchor;  //Codigo para dibujar lineas  //Se calculan los puntos a utilizar para redibujar la linea  puntoInicial = new Point(tempElemento.width / 2 + tempElemento.left, tempElemento.height + tempElemento.top);  //variable temporal para almacenar el elemento del punto final  ElementoDFD tempElemento2 = tempElemento.centro;  //while (tempElemento2.tipo!= Elemento.fin && tempElemento2.visible != true) tempElemento2 = tempElemento2.centro;  //Se calcula el punto final a utilizar para realizar el dibujo de la linea  if (tempElemento2.tipo == Elemento.EndIf) {  lapiz.EndCap = LineCap.NoAnchor;  Point segundo = new Point(puntoInicial.X, tempElemento2.top);  Point final = new Point(tempElemento2.left, tempElemento2.top);  Point[] tempPoins = {puntoInicial,segundo,final};  bg.Graphics.DrawLines(lapiz, tempPoins);  }  else  {  puntoFinal = new Point(tempElemento2.width / 2 + tempElemento2.left, tempElemento2.top);  bg.Graphics.DrawLine(lapiz, puntoInicial, puntoFinal);  }  break;  }  }  private void DibujarElemento(ref BufferedGraphics tempbf, ElementoDFD tempElemento, Pen tempPen)  {  if (tempElemento.visible == false) return;//si el elemto es invisible no se dibuja y se regresa  //Rectangulo utilizado por la funcion de dibujo  Rectangle tempRectangle = new Rectangle(tempElemento.left, tempElemento.top, tempElemento.width, tempElemento.height);  StringFormat formato =new StringFormat();  formato.Alignment= StringAlignment.Center;  SolidBrush brocha = new SolidBrush(colorLetra);  switch (tempElemento.tipo)  {  case Elemento.inicio:  tempbf.Graphics.DrawEllipse(tempPen, tempRectangle);  //Agregando Texto al control  tempbf.Graphics.DrawString("\n\nInicio", fontLetra, brocha, tempRectangle, formato);  break;  case Elemento.Asignacion:  tempbf.Graphics.DrawRectangle(tempPen, tempRectangle);  //Agregando Texto al control  tempbf.Graphics.DrawString("\n"+tempElemento.datos, fontLetra, brocha, tempRectangle, formato);  break;  case Elemento.Escritura:  Point ep1 = new Point(tempElemento.left,tempElemento.top+tempElemento.height/2);  Point ep2 = new Point(tempElemento.left+tempElemento.width,tempElemento.top);  Point ep3 = new Point(tempElemento.left+tempElemento.width,tempElemento.top+tempElemento.height);  Point[] esc = { ep1,ep2,ep3};  tempbf.Graphics.DrawPolygon(tempPen, esc);  tempRectangle = new Rectangle(tempElemento.left+10, tempElemento.top+24, tempElemento.width-10, 15);  tempbf.Graphics.DrawString( tempElemento.datos, fontLetra, brocha, tempRectangle, formato);  break;  case Elemento.Lectura:  Point sizq = new Point(tempElemento.left,tempElemento.top+tempElemento.height/2);  Point sder = new Point(tempElemento.left+tempElemento.width,tempElemento.top);  Point ider = new Point(tempElemento.left+tempElemento.width,tempElemento.top+tempElemento.height);  Point iizq = new Point(tempElemento.left,tempElemento.top+tempElemento.height);  Point[] lect = { sizq, sder, ider, iizq };  tempbf.Graphics.DrawPolygon(tempPen, lect);  tempRectangle = new Rectangle(tempElemento.left+10, tempElemento.top+24, tempElemento.width-10, 15);  tempbf.Graphics.DrawString( tempElemento.datos, fontLetra, brocha, tempRectangle, formato);  break;  case Elemento.Eif:  Point sup = new Point(tempElemento.left + tempElemento.width / 2, tempElemento.top);  Point izq = new Point(tempElemento.left, tempElemento.top + tempElemento.height / 2);  Point inf = new Point(tempElemento.left + tempElemento.width / 2, tempElemento.top + tempElemento.height);  Point der = new Point(tempElemento.left + tempElemento.width, tempElemento.top + tempElemento.height / 2);  Point[] Eif = { sup, izq, inf, der };  tempbf.Graphics.DrawPolygon(tempPen, Eif);  tempRectangle = new Rectangle(tempElemento.left+10, tempElemento.top, tempElemento.width-10, tempElemento.height);  tempbf.Graphics.DrawString("\n\n" + tempElemento.datos, fontLetra, brocha, tempRectangle, formato);  tempRectangle = new Rectangle(tempElemento.left -20, tempElemento.top, 20, 20);  tempbf.Graphics.DrawString("Si" , fontLetra, brocha, tempRectangle, formato);  tempRectangle = new Rectangle(tempElemento.left + tempElemento.width+5, tempElemento.top, 20, 20);  tempbf.Graphics.DrawString("No" , fontLetra, brocha, tempRectangle, formato);  ElementoDFD temp = tempElemento.izquierda;  Pen lapizFiguras = new Pen(colorLinea, grosorFiguras);  Pen lapiz = new Pen(colorLinea, grosorLinea);  Pen lapizEliminar = new Pen(colorEliminar, grosorFiguras);  if (temp != null) {  while (temp != tempElemento.centro) {  //Se dibuja el elemento a utilizar  if (estado == Estado.Eliminando && temp.tipo != Elemento.inicio && temp.VerificarInteraccion(posMouseX, posMouseY))  DibujarElemento(ref bg, temp, lapizEliminar);  else  DibujarElemento(ref bg, temp, lapizFiguras);    DibujarLinea(ref tempbf, temp, lapiz);  temp = temp.centro;  }  }  temp = tempElemento.derecha;  if (temp != null) {  while (temp != tempElemento.centro)  {  //Se dibuja el elemento a utilizar  if (estado == Estado.Eliminando && temp.tipo != Elemento.inicio && temp.VerificarInteraccion(posMouseX, posMouseY))  DibujarElemento(ref bg, temp, lapizEliminar);  else  DibujarElemento(ref bg, temp, lapizFiguras);    DibujarLinea(ref tempbf, temp, lapiz);  temp = temp.centro;  }  }  break;  case Elemento.Efor:  Point P1 = new Point(tempElemento.left, tempElemento.height / 2 + tempElemento.top);  Point P2 = new Point(P1.X + tempElemento.width / 10, tempElemento.top);  Point P3 = new Point(P1.X + tempElemento.width - tempElemento.width / 10, tempElemento.top);  Point P4 = new Point(P1.X + tempElemento.width, P1.Y);  Point P5 = new Point(P3.X, tempElemento.top + tempElemento.height);  Point P6 = new Point(P2.X, P5.Y);  Point[] Efor = { P1, P2, P3, P4, P5, P6 };  tempbf.Graphics.DrawPolygon(tempPen, Efor);  break;  case Elemento.Endfor:  tempbf.Graphics.DrawEllipse(tempPen, tempRectangle);  //Agregando Texto al control  tempbf.Graphics.DrawString("\nFin For", fontLetra, brocha, tempRectangle, formato);    break;  case Elemento.EWhile:  Point PW1 = new Point(tempElemento.left, tempElemento.height / 2 + tempElemento.top);  Point PW2 = new Point(PW1.X + tempElemento.width / 10, tempElemento.top);  Point PW3 = new Point(PW1.X + tempElemento.width - tempElemento.width / 10, tempElemento.top);  Point PW4 = new Point(PW1.X + tempElemento.width, PW1.Y);  Point PW5 = new Point(PW3.X, tempElemento.top + tempElemento.height);  Point PW6 = new Point(PW2.X, PW5.Y);  Point[] EWhile = { PW1, PW2, PW3, PW4, PW5, PW6 };  tempbf.Graphics.DrawPolygon(tempPen, EWhile);  tempRectangle = new Rectangle(tempElemento.left+2, tempElemento.top, tempElemento.width, 15);  tempbf.Graphics.DrawString( "While : "+tempElemento.datos, fontLetra, brocha, tempRectangle, formato);    break;  case Elemento.EndWhile:  tempbf.Graphics.DrawEllipse(tempPen, tempRectangle);  //Agregando Texto al control  tempbf.Graphics.DrawString("\nFin While", fontLetra, brocha, tempRectangle, formato);  break;  case Elemento.fin:  tempbf.Graphics.DrawEllipse(tempPen, tempRectangle);  //Agregando Texto al control  tempbf.Graphics.DrawString("\n\nFin", fontLetra, brocha, tempRectangle, formato);  break;  }  }  //------------------------------------------------------------------  //Declaracion de propiedades publicas para utilizar con los controles  //------------------------------------------------------------------  [Category("Lineas")]  [Description("Color de las lineas del grafo")]  public Color Color  {  set { colorLinea = value; }  get { return colorLinea; }  }  [Category("Lineas")]  [Description("Color del fondo")]  public Color ColorFondo  {  set { colorFondo = value;  this.BackColor = value;  }  get { return colorFondo; }  }  [Category("Lineas")]  [Description("Grosor de las lineas del grafo")]  public int Grosor  {  set { grosorLinea = value; }  get { return grosorLinea; }  }  [Category("Dimensiones")]  [Description("Dimensiones del elemento Asignacion")]  public Size DimensionAsignacion  {  set { tamanioAsignacion = value; }  get { return tamanioAsignacion; }  }  [Category("Dimensiones")]  [Description("Dimensiones del elemento If")]  public Size DimensionIf  {  set { tamanioIf = value; }  get { return tamanioIf; }  }  //----------------------------------------------------------------------------------------------------------------------------  private void TimerDibujar\_Tick(object sender, EventArgs e)  {  Dibujar();  }  private void UCDFD\_DoubleClick(object sender, EventArgs e)  {    }    //Funcion utilizada para poder reajustar los elementos dentro del contenedor cada vez que realice un movimiento dentro del scroll  private void UCDFD\_Scroll(object sender, ScrollEventArgs e)  {  ReajustarElementos(elementoRaiz);  }  private void UCDFD\_MouseMove(object sender, MouseEventArgs e)  {  switch (estado)  {  case Estado.Agregando:  if (elementoAgregado != null)  { //se calcula la posicion que tendria que tener dentro de la ventana  elementoAgregado.left = e.X - elementoAgregado.width / 2;  elementoAgregado.top = e.Y - elementoAgregado.height / 2;  }  break;  case Estado.Eliminando:  posMouseX = e.X;  posMouseY = e.Y;  break;  }    }  private void UCDFD\_Click(object sender, EventArgs e)  {      }  private void UCDFD\_MouseEnter(object sender, EventArgs e)  {  switch (estado)  {  case Estado.Agregando:  if (elementoAgregado != null) elementoAgregado.visible = true;  break;  }    }  private void UCDFD\_MouseLeave(object sender, EventArgs e)  {  switch (estado) {  case Estado.Agregando:  if (elementoAgregado != null) elementoAgregado.visible = false;  break;  }      }  private void UCDFD\_MouseClick(object sender, MouseEventArgs e)  {  ElementoDFD tempElemento;  switch (estado)  {  case Estado.Agregando:  Agregando\_Click();  break;  case Estado.Eliminando:  if((tempElemento= VerificarAccion(e.X,e.Y))!=null){  if (tempElemento.tipo != Elemento.inicio && tempElemento.tipo != Elemento.Endfor && tempElemento.tipo != Elemento.EndWhile && tempElemento.tipo != Elemento.fin && tempElemento.tipo != Elemento.EndIf)  EliminarElementos(tempElemento);  estado=Estado.Normal;  ReajustarElementos(elementoRaiz);  }  break;  }  }  }  } |

### Tabla de errores

Se utiliza para poseer una estructura que nos permita manejar los errores y almacenarlos para mostrarlos al usuario si es que estos sedan en el proceso de compilacion o analisis, permitiendo tener una referencia hacia el elemento que genero el error ademas de permetir la deteccion de futuros problemas dentro del analsis y detener la compilacion.

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace CompiladorDFD.Datos\_Externos  {//Clase utilizada para manejar lo que son las tablas de los errores generados a lo largo de los analisis  public class TablaDeErrores  {//Lista de errores que se generaron  public List<Error> errores = new List<Error>();  //Funcion para agregar los errores dentro de la tabla de errores  public void AgregarError(Error error){  errores.Add(error);  }  //Funcion para verificar si existe o no errores dentro de la tabla de errores  public bool Existen() {  if (errores.Count > 0) return true;  return false;  }  }  //Clase que permite alamacenar los errores para poder especificar de una manera mas clara y  //Precisa el elemento que genero el error , en que etapa del analisis y ademas que lo causo  public class Error{  public string detalle; //Detalle del error  public ElementoDFD ElementoError;//Elemento en el que se genero el error  public string faseAnalisis;//Fase en la que se genero el error  public Error() { }//Constructor vacio  //----------------------------------------------------------------------------------------------------  // FUNCIONES PARA INGRESAR LOS DIFERENTES TIPOS DE ERRORES ENCONTRADOS DENTRO DE LA EJECUCION  //----------------------------------------------------------------------------------------------------  public Error(int id, string fase, ElementoDFD elemento,TokenData tempTokendata) {  Token tempToken = ValoresGlobales.valores().tablaDeTokens.ObtenerIdToken(id);  detalle = "Se esperaba un "+ tempToken.nombre+ " en vez de " +tempTokendata.codigo+" Dentro del elemento "+ elemento.tipo.ToString();  ElementoError = elemento;  faseAnalisis = fase;  }  public Error(int id, string fase, ElementoDFD elemento)  {  Token tempToken = ValoresGlobales.valores().tablaDeTokens.ObtenerIdToken(id);  detalle = "Se esperaba un " + tempToken.nombre + " Dentro del elemento " + elemento.tipo.ToString();  ElementoError = elemento;  faseAnalisis = fase;  }  public void NoSeEsperaba(int id,string fase, ElementoDFD elemento){  Token tempToken = ValoresGlobales.valores().tablaDeTokens.ObtenerIdToken(id);  detalle = "No se esperaba un " + tempToken.nombre + " Dentro del elemento " + elemento.tipo.ToString();  ElementoError = elemento;  faseAnalisis = fase;  }  public void ErrorCustom(string mensaje, string fase, ElementoDFD elemento) {  detalle = mensaje;  ElementoError = elemento;  faseAnalisis = fase;  }    }  } |

### Tabla de símbolos

La tabla de simbolos es una tabla que nos proporciona una estructura para almacenar las variables tanto en el analisis lexico como semantico , como tambien es de vital importancia para el analizador semantico que es el encargado de determinar tipos de datos y ademas de cambiar los identificadores por su tipo de dato respectivo, y sobre todo es la base para poder declarar las variables dentro del lenguaje MSIL.

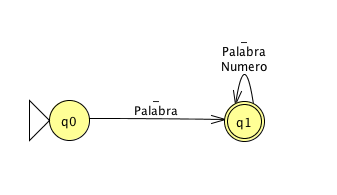
|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace CompiladorDFD.Datos\_Externos  {//En esta tabla se almacena las varibles encontradas dentro del programa esto e utilizado tanto por el  //Analizador lexico, sintactico y semantico para la generacion de cambios de tipos , ingresos y reconocimiento  // de variables y tipos de datos  public class TablaDeSimbolos  {//Diccionario en el que se almacena el nombre de la variable y sus tokenData que es la informacion  //acerca de ella  Dictionary<string, TokenData> Simbolos = new Dictionary<string, TokenData>();  //Funcion para verificar si una variable ya se encuentra dentro de la tabla de simbolos  public bool VerificarSimbolo(string nombre) {  TokenData tempTokenData = new TokenData();  if(Simbolos.TryGetValue(nombre,out tempTokenData))  return true;  else  return false;  }  //Funcion para agregar nuevos simbolos con su respectiva dataToken  public void AgregarToken(string nombre, TokenData tokenData) {  Simbolos.Add(nombre, tokenData);  }  //Funcion para obtener los datos del token apartir del nombre de la variable que se  //Ingreso dentro de la tabla de simbolos  public Token ObtenerToken(string nombre){  TokenData tempTokenData = new TokenData();  if (Simbolos.TryGetValue(nombre, out tempTokenData))  return tempTokenData.tokenInfo;  else  return null;  }  //Funcion utilizada por el analizador semantico para cambiar el tipo de las variables una vez evaluado  //Todas las asignaciones determinando asi el tipo que le corresponde  public void CambiarTipo(string variable,int id) {  Simbolos[variable].tokenInfo = ValoresGlobales.valores().tablaDeTokens.ObtenerIdToken(id);  }  }  } |

### Analizador Lexico

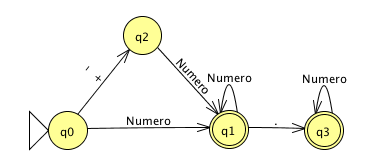
Bien definidos nuestros token ahora podemos procegir a el analisis de los patrones que aceptara nuestro Analizador Lexico.

*Patron para los tokens e identificadores alfanumericos*

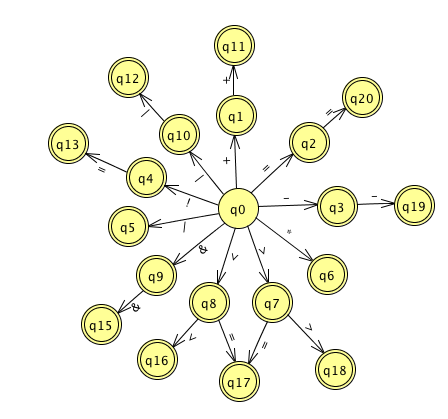
Se define palabra como A-Z y a-z y Numeros como 0-9



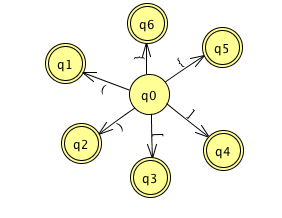
*Patron para los Numeros Enteros y Flotantes*



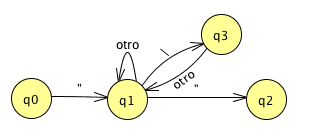
*Patrones para Operadores*



Patron para los signos de Agrupacion o llaves



Patron para las cadenas



Codigo Fuente de la implementacion de los patrones dentro del analisis lexico:

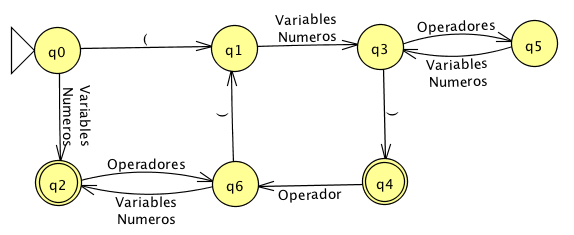
|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using CompiladorDFD.Datos\_Externos;  namespace CompiladorDFD.Analizadores  {//Clase utilizada para realizar el Analisis Lexico sobre el Grafo (DFD)  class AnalizadorLexico  {  int pos = 0; //Posicion dentro de la cadena a analizar  char[] codigo;  string union = "";//String utilizado para unir las cadenas  public ElementoDFD tempElemento= null;  public TokenData returnToken = new TokenData();  //Funcion para agregar el codigo Fuente  public void AgregarCodigo(string cod)  {  //Se convierte el string pasado a una coleccion de  //Caracteres para poder poder recorrer el arreglo.  codigo = cod.ToCharArray();  //Se hace un reset a la posicion de lectura  pos = 0;  }  //Funcion para comenzar el analisis  public bool ObtenerToken() {  //Se verifica si ya se llego al final de los datos a analizar  if (pos == codigo.Length) return false;  else {  //Mientras se encuentren caracters que leer  while (pos < codigo.Length) {  if (IsNextToken())  { //Se incrementa la posicion  pos++;  }  else {  //Se verifica si es una letra o \_  if (char.IsLetter(codigo[pos]) || codigo[pos] == '\_') {  if(!verificarVariables())  { Error error = new Error();  error.ErrorCustom("La variable "+ union + "No es valida","Lexico",tempElemento);  ValoresGlobales.valores().tablaDeErrores.AgregarError(error);  }  return true;  }else if(char.IsNumber(codigo[pos])){//Se verifica si es un numero  if (!verificarNumero()) {  Error error = new Error();  error.ErrorCustom("El siguiente parametro no es valido " + union +" en el elemento"+ tempElemento.tipo.ToString(), "Lexico", tempElemento);  ValoresGlobales.valores().tablaDeErrores.AgregarError(error);    }  return true;  }  else if (codigo[pos] == '"')//Se verifica si es una cadena  {  if (!verificarCadena())  {  Error error = new Error();  error.ErrorCustom("La variable " + union + "No es valida", "Lexico", tempElemento);  ValoresGlobales.valores().tablaDeErrores.AgregarError(error);  }  return true;  }  else if (codigo[pos] == '(') {//Se ingresa como token "("  union = "(";  returnToken = new TokenData(union, ValoresGlobales.valores().tablaDeTokens.ObtenerToken(union));  pos++;  return true;  }  else if (codigo[pos] == ')')  {//Se ingresa como token ")"  union = ")";  returnToken = new TokenData(union, ValoresGlobales.valores().tablaDeTokens.ObtenerToken(union));  pos++;  return true;  }  else if (codigo[pos] == ',')  {//Se ingresa como token ","  union = ",";  returnToken = new TokenData(union, ValoresGlobales.valores().tablaDeTokens.ObtenerToken(union));  pos++;  return true;  }  else//De no ser ninguno de los anteriores se busca una concordancia con los operadores  {//Para ello se verifica si esta dentro de los operadores  if (!VerificarOperador())  {  Error error = new Error();  error.ErrorCustom("El caractrer " + union + "No es valida no se encuentra registrado", "Lexico", tempElemento);  ValoresGlobales.valores().tablaDeErrores.AgregarError(error);  }  else  {  return true;  }  }  }  }  }  return false;//Si no existe un token que retornar  }  private bool VerificarOperador()  {//Verificacion de operadores Matematicos y Logicos  returnToken = null;  union = "";  switch (codigo[pos])  {  case '-':  union += codigo[pos];  //if (codigo[pos + 1] == '-') union += codigo[pos];  break;  case '+':  union += codigo[pos];  //if (codigo[pos + 1] == '+') union += codigo[pos];  break;  case '/':  union += codigo[pos];  break;  case '\*':  union += codigo[pos];  break;  case '>':  union += codigo[pos];  //if (codigo[pos + 1] == '>') union += codigo[pos];  if (codigo.Length > pos + 1)  if (codigo[pos + 1] == '=') union += codigo[++pos];  break;  case '<':  union += codigo[pos];  //if (codigo[pos + 1] == '<') union += codigo[pos];  if (codigo.Length > pos + 1)  if (codigo[pos + 1] == '=') union += codigo[++pos];  break;  case '=':  union += codigo[pos];  if (codigo.Length > pos+1)  if (codigo[pos + 1] == '=') union += codigo[++pos];  break;  case '|':  union += codigo[pos];  if (codigo.Length > pos + 1)  if (codigo[pos + 1] == '|') union += codigo[++pos];  break;  case '&':  union += codigo[pos];  if (codigo.Length > pos + 1)  if (codigo[pos + 1] == '&') union += codigo[++pos];  break;  case '!':  union += codigo[pos];  if (codigo.Length > pos + 1)  if (codigo[pos + 1] == '=') union += codigo[++pos];  break;  default:  pos++;  return false;  break;  }  //Se genera el token a regresar  returnToken = new TokenData(union, ValoresGlobales.valores().tablaDeTokens.ObtenerToken(union));  pos++;  return true;  }  public bool verificarNumero() {  returnToken = null;  int punto = 0;  union = "";  union += codigo[pos]; //Se comienza a formar la palabras  pos++;//Se pasa a la siguiente posicion  if (pos < codigo.Length )  {//Se verifica que no sea el final de la cadena  //Ahora como comenzamos a verificar que sea una palabra  //Verificamos si es un numero o "."  while (char.IsDigit(codigo[pos]) || codigo[pos] == '.')  {  if (codigo[pos] == '.') punto++;  //Procegimos a concatenar y verificar el siguiente caracter  union += codigo[pos];  pos++;  if (pos == codigo.Length) break; // si llegamos al final salimos de ella  }  }  if (punto > 1) return false;  returnToken = new TokenData(union, ValoresGlobales.valores().tablaDeTokens.ObtenerToken("numDecimal"));  return true;  }  //Se verifica si se ha llegado al siguiente token o no  public bool IsNextToken(){  return codigo[pos] == ' ' || codigo[pos] == '\n' || codigo[pos] == '\t';  }  public bool verificarCadena() {  returnToken = null;  union = "";  union += codigo[pos];//Se comienza a formar el codigo de la palabra  //Se incremente la posicion de la palabra  pos++;  if (pos+1 > codigo.Length) return false;  //Se recorre la cadena para verificar que sea un dato  while (codigo[pos] != '"' && pos < codigo.Length)  {  if (pos < codigo.Length - 1)  {  if (codigo[pos] == '\\' && codigo[pos + 1] == 'n')  {  union += '\n';  pos += 2;  }else  union += codigo[pos++];  }  else {  break;  }    }  union+=codigo[pos];  if (codigo[pos] != '"') return false;  pos++;  returnToken = new TokenData(union, ValoresGlobales.valores().tablaDeTokens.ObtenerToken("cadena"));  return true;  }  public bool verificarVariables()  {  union = "";  union += codigo[pos];//Se comienza a formar el codigo de la palabra  //Se incremente la posicion de la palabra  pos++;  //Se recorre la cadena para verificar que sea un dato  if (!(pos < codigo.Length)) goto continuar;  while (codigo[pos] != ' ')  {  if (!IsNextToken())  {  if (char.IsLetterOrDigit(codigo[pos]) || codigo[pos] == '\_')  {//Se continua agregando datos  union += codigo[pos++];  if (!(pos < codigo.Length)) break;  }  else  { //Codigos de escape al encontrar otro caracter a la par  if (codigo[pos] == '=') break;  if (codigo[pos] == '+') break;  if (codigo[pos] == '-') break;  if (codigo[pos] == '\*') break;  if (codigo[pos] == '/') break;  if (codigo[pos] == '(') break;  if (codigo[pos] == ')') break;  if (codigo[pos] == ',') break;  if (codigo[pos] == '>') break;  if (codigo[pos] == '<') break;  if (codigo[pos] == '!') break;    else return false;  }  }  else  {  break;  }  }  continuar:  //Se procede a verificar si las variables seran registradas o no en la tabla de simbolos  if (ValoresGlobales.valores().tablaDeSimbolos.VerificarSimbolo(union))  {//Se obtiene la informacion del token si esque se encontro dentro de la tabla de simbolos  returnToken = new TokenData(union, ValoresGlobales.valores().tablaDeSimbolos.ObtenerToken(union));  }else  {//De no estar en la tabla de simbolos se procede a ingresarlo dentro de ella con los parametros  //De un identificador para poder darle seguimiento a dicha variable  TokenData newToken = new TokenData(union, ValoresGlobales.valores().tablaDeTokens.ObtenerToken("identificador"));  ValoresGlobales.valores().tablaDeSimbolos.AgregarToken(union,newToken);  returnToken = new TokenData(union, newToken.tokenInfo);  }  return true;  }  }  } |

### Analizador Sintactico

En esta parte se usan patrones para reconocer las expresiones que seran reconocidas por lo que es el compilador DFD utilizando como principal estructura el grafo creado por le control “ucdfd” que nos permite obtener un grafo según los elementos insertados.

Estructuras basicas a utilizar para definir patrones aceptados por los elementos

Expresiones

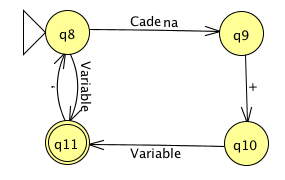


Patrones aceptados por los elementos :

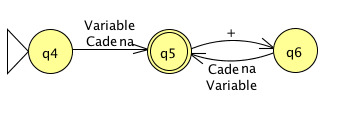
Elemento de Asignacion :



Elemento Lectura :



Elemento Escritura :



Elemento IF y While :



A partir de los patrones anteriores se procede a realizar las comparaciones dentro de lo que es el analizador sintectico para encontrar las concordancias de cada elemento dentro de cada elemento del grafo y asi crear lo que serian los tokenData que contiene toda la informacion hacerca del codigo y ademas la informacion sobre el tipo de token y id que recibe dentro de la tabla de token.

Una vez obtenido cada una de las verificaciones de los patrones se procede a formar un grafo atachado al grado del DFD para ser pasado al analizador semantico.

Codigo de la implementacion de los patrones para el analisis sintactico:

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using CompiladorDFD.Datos\_Externos;  namespace CompiladorDFD.Analizadores  {  class AnalizadorSintactico  { //Elemento raiz del grafo  private ElementoDFD raiz;  private ElementoDFD tempElemento;  //Elemento para realizar el analisis lexico  private AnalizadorLexico analizadorLexico = new AnalizadorLexico();  public AnalizadorSintactico() { }  //Se recorre todo el arbol y se va verificando los elementos agregados a este  public void GenerarArbol() {  Stack<ElementoDFD> elementoIf = new Stack<ElementoDFD>();  tempElemento = ValoresGlobales.valores().elementoRaiz;  while(tempElemento.tipo != Elemento.fin){  switch (tempElemento.tipo) {  case Elemento.Asignacion:  //Se limpian las referencias del arbol  tempElemento.tokenDataRef.Clear();  //Se obtiene los parametros de las cadenas para poder proceder a realizar el analisis lexico  if (tempElemento.datos != null)  {  string[] cadena = tempElemento.datos.Split('\n');  if (cadena.Length > 1) tempElemento.tokenDataRef.Add(SintaxisAsignacion(cadena[0])); ;  if (cadena.Length > 2) tempElemento.tokenDataRef.Add(SintaxisAsignacion(cadena[1])); ;  if (cadena.Length > 3) tempElemento.tokenDataRef.Add(SintaxisAsignacion(cadena[2])); ;  }  break;  case Elemento.Escritura:  tempElemento.tokenDataRef.Clear();  tempElemento.tokenDataRef.Add( SintaxisExpresionSalida(tempElemento.datos));  break;  case Elemento.Lectura:  tempElemento.tokenDataRef.Clear();  string[] cadenaLectura = tempElemento.datos.Split(',');  int i = 0;  while (i < cadenaLectura.Length) tempElemento.tokenDataRef.Add(SintaxisExpresionLectura(cadenaLectura[i++]));  break;  case Elemento.Eif:  tempElemento.tokenDataRef.Clear();  //Se agregan los datos para que el analizador trabaje  analizadorLexico.AgregarCodigo(tempElemento.datos);  analizadorLexico.tempElemento = tempElemento;  //Se procede a verifiar la sintaxis  tempElemento.tokenDataRef.Add(SintaxisExpresionIf());  if (analizadorLexico.returnToken != null)  {//Se verifica si el token ingresado es uno de los esperados  switch (analizadorLexico.returnToken.tokenInfo.id)  {  case 38: // ==  case 39: // <=  case 40: // >=  case 41: // <  case 42: // >  case 43: // !=  tempElemento.tokenDataRef.Add(analizadorLexico.returnToken);  break;  default:  Error error2 = new Error();  error2.NoSeEsperaba(analizadorLexico.returnToken.tokenInfo.id, "Sintactico", tempElemento);  ValoresGlobales.valores().tablaDeErrores.AgregarError(error2);  break;  }  tempElemento.tokenDataRef.Add(SintaxisExpresionIf());  if (tempElemento.derecha != null)  {  tempElemento = tempElemento.derecha;  elementoIf.Push(tempElemento);  continue;  }  else if (tempElemento.izquierda != null)  {  tempElemento = tempElemento.izquierda;  elementoIf.Push(tempElemento);  continue;  }  }  else  {  Error error = new Error();  error.ErrorCustom("No se han obtendido la gramatica esperada 'EXP COMPARADOR EXP' en el elemento" + tempElemento.tipo.ToString(), "Sintactico", tempElemento);  ValoresGlobales.valores().tablaDeErrores.AgregarError(error);  }  break;  case Elemento.EndIf:  if (elementoIf.Count > 0)  {  if (tempElemento.padre.derecha == elementoIf.Peek())  {  tempElemento = tempElemento.padre.izquierda;  elementoIf.Pop();  continue;  }  }  break;  case Elemento.EWhile:  tempElemento.tokenDataRef.Clear();  //Se agregan los datos para que el analizador trabaje  analizadorLexico.AgregarCodigo(tempElemento.datos);  analizadorLexico.tempElemento = tempElemento;  //Se procede a verifiar la sintaxis  tempElemento.tokenDataRef.Add(SintaxisExpresionIf());  if (analizadorLexico.returnToken != null)  {//Se verifica si el token ingresado es uno de los esperados  switch (analizadorLexico.returnToken.tokenInfo.id)  {  case 38: // ==  case 39: // <=  case 40: // >=  case 41: // <  case 42: // >  case 43: // !=  tempElemento.tokenDataRef.Add(analizadorLexico.returnToken);  break;  default:  Error error2 = new Error();  error2.NoSeEsperaba(analizadorLexico.returnToken.tokenInfo.id, "Sintactico", tempElemento);  ValoresGlobales.valores().tablaDeErrores.AgregarError(error2);  break;  }  tempElemento.tokenDataRef.Add(SintaxisExpresionIf());  }  else  {  Error error = new Error();  error.ErrorCustom("No se han obtendido la gramatica esperada 'EXP COMPARADOR EXP' en el elemento" + tempElemento.tipo.ToString(), "Sintactico", tempElemento);  ValoresGlobales.valores().tablaDeErrores.AgregarError(error);  }  break;  default:  break;  }  tempElemento = tempElemento.centro;  }  }  private TokenData SintaxisIf(string cadena) {  TokenData returnTokenData = new TokenData();  //Se agrega el codigo a ser analizado por el analizador lexico  analizadorLexico.tempElemento = tempElemento;  analizadorLexico.AgregarCodigo(cadena);    return returnTokenData;  }  private TokenData SintaxisAsignacion(string cadena) {  TokenData returnTokenData = new TokenData();  //Se agrega el codigo a ser analizado por el analizador lexico  analizadorLexico.tempElemento = tempElemento;  analizadorLexico.AgregarCodigo(cadena);  //Variable temporal para almacenar el tokendata  TokenData tempTokenData;  int[] patron = { 53, 37 };  int posPatron = 0;  bool er = true;  while (analizadorLexico.ObtenerToken()) {  tempTokenData =analizadorLexico.returnToken;  if (posPatron < patron.Length)  {  if (patron[posPatron] == tempTokenData.tokenInfo.id)  {  if(posPatron==0) returnTokenData = tempTokenData;  else returnTokenData.tokenDataRef = tempTokenData;  posPatron++;  }  else {  //Agregando el error correspondiente  ValoresGlobales.valores().tablaDeErrores.errores.Add(new Error(patron[posPatron],"Sintactico",tempElemento,tempTokenData));  }  }  else {  returnTokenData.tokenDataRef.tokenDataRef= SintaxisExpresionesAsignacion();  if(returnTokenData.tokenDataRef.tokenDataRef != null) er = false;  }  }  if (er == true) {  Error error = new Error();  error.ErrorCustom("Expresion ingresada incorrecta dentro del elemento" + tempElemento.tipo.ToString(), "Sintactico", tempElemento);  ValoresGlobales.valores().tablaDeErrores.AgregarError(error);  }  return returnTokenData;  }  public void AgregarTokenData(ref TokenData tempRaiz, ref TokenData temp){  if(temp==null && tempRaiz==null){  //Se coloca el elemento raiz  tempRaiz = analizadorLexico.returnToken;  temp=analizadorLexico.returnToken;  }else{  //Agregamos el nuevo elemento al arbol y luego nos movemos para dejar espacio  //para el siguiete elemento  temp.tokenDataRef=analizadorLexico.returnToken;  temp= temp.tokenDataRef;  }  }  //Nos lleva hasta el final de la lista  private void AvanzarFinal(TokenData tempToken){  while (tempToken.tokenDataRef != null)  tempToken = tempToken.tokenDataRef;  }  private TokenData SintaxisExpresionLectura(string cadena) {  analizadorLexico.AgregarCodigo(cadena);  analizadorLexico.tempElemento = tempElemento;  TokenData raizReturn = null;  TokenData tempReturn = null;  Stack<int> obligatorio = new Stack<int>();  List<int> opcional = new List<int>();  //Valores opcionales iniciales  opcional.Add(53);//variables  opcional.Add(58);//Cadena  analizadorLexico.ObtenerToken();  do  {  if (opcional.Contains(analizadorLexico.returnToken.tokenInfo.id))  {  switch (analizadorLexico.returnToken.tokenInfo.id)  {  case 58://cadenas  opcional.Clear();  opcional.Add(29);  AgregarTokenData(ref raizReturn, ref tempReturn);  //Se colocan los datos obligatorios dentro del stack  obligatorio.Push(29);  break;  case 53:// Variables    //Se colocan los valores opcionales a continuacion  opcional.Clear();  AgregarTokenData(ref raizReturn, ref tempReturn);  //Se colocan los datos obligatorios dentro del stack  if (obligatorio.Count > 0)  {  if (obligatorio.Peek() == 53) obligatorio.Pop();  }  break;  case 29:  //Se colocan los valores opcionales a continuacion  opcional.Clear();  opcional.Add(53);//variables  AgregarTokenData(ref raizReturn, ref tempReturn);  //Datos que son obligatorios  if (obligatorio.Count > 0)  {  if (obligatorio.Peek() == 29) obligatorio.Pop();  }  obligatorio.Push(53);  break;  default:  Error error2 = new Error();  error2.NoSeEsperaba(analizadorLexico.returnToken.tokenInfo.id, "Sintactico", tempElemento);  ValoresGlobales.valores().tablaDeErrores.AgregarError(error2);  break;  }  }  else  {  Error error2 = new Error();  error2.NoSeEsperaba(analizadorLexico.returnToken.tokenInfo.id, "Sintactico", tempElemento);  ValoresGlobales.valores().tablaDeErrores.AgregarError(error2);  break;  }  //Mientras existan tokens  } while (analizadorLexico.ObtenerToken());  if (obligatorio.Count > 0)  {  Error error = new Error();  error.ErrorCustom("Expresion ingresada incorrecta dentro del elemento" + tempElemento.tipo.ToString(), "Sintactico", tempElemento);  ValoresGlobales.valores().tablaDeErrores.AgregarError(error);  }  return raizReturn;  }//Fin SintaxisExpresionLectura  private TokenData SintaxisExpresionSalida( string cadena) {  analizadorLexico.AgregarCodigo(cadena);  analizadorLexico.tempElemento = tempElemento;  TokenData raizReturn = null;  TokenData tempReturn = null;  Stack<int> obligatorio = new Stack<int>();  List<int> opcional = new List<int>();  //Valores opcionales iniciales  opcional.Add(53);//variables  opcional.Add(55);//numero  opcional.Add(58);//Cadena  analizadorLexico.ObtenerToken();  do  { if(opcional.Contains(analizadorLexico.returnToken.tokenInfo.id)){  switch (analizadorLexico.returnToken.tokenInfo.id)  {  case 53:// Variables  case 55:// Numeros  case 58://cadenas  //Se colocan los valores opcionales a continuacion  opcional.Clear();  opcional.Add(29);//variable  AgregarTokenData(ref raizReturn, ref tempReturn);  //Se colocan los datos obligatorios dentro del stack  if (obligatorio.Count > 0)  { if(obligatorio.Peek()==58) obligatorio.Pop();  if (obligatorio.Peek() == 55) obligatorio.Pop();  if (obligatorio.Peek() == 53) obligatorio.Pop();  }  break;  case 29:  //Se colocan los valores opcionales a continuacion  opcional.Clear();  opcional.Add(53);//variables  opcional.Add(55);//numero  opcional.Add(58);//Cadena  AgregarTokenData(ref raizReturn, ref tempReturn);  //Datos que son obligatorios  obligatorio.Push(53);  obligatorio.Push(55);  obligatorio.Push(58);  break;  default:  Error error2 = new Error();  error2.NoSeEsperaba(analizadorLexico.returnToken.tokenInfo.id, "Sintactico", tempElemento);  ValoresGlobales.valores().tablaDeErrores.AgregarError(error2);  break;  }  }else{  Error error2 = new Error();  error2.NoSeEsperaba(analizadorLexico.returnToken.tokenInfo.id, "Sintactico", tempElemento);  ValoresGlobales.valores().tablaDeErrores.AgregarError(error2);  break;    }  //Mientras existan tokens  } while (analizadorLexico.ObtenerToken());  if (obligatorio.Count > 0)  {  Error error = new Error();  error.ErrorCustom("Expresion ingresada incorrecta dentro del elemento" + tempElemento.tipo.ToString(), "Sintactico", tempElemento);  ValoresGlobales.valores().tablaDeErrores.AgregarError(error);  }  return raizReturn;  }  private TokenData SintaxisExpresion() {  TokenData raizReturn = null;  TokenData tempReturn = null;  Stack<int> obligatorio = new Stack<int>();  List<int> opcional = new List<int>();  //Valores opcionales iniciales  opcional.Add(23);  opcional.Add(53);  opcional.Add(55);  opcional.Add(30);  do  {  switch (analizadorLexico.returnToken.tokenInfo.id) {  case 23:// (  //Se colocan los valores opcionales a continuacion  opcional.Clear();  opcional.Add(23);  opcional.Add(53);  opcional.Add(55);  opcional.Add(30);  AgregarTokenData(ref raizReturn, ref tempReturn);  //Se colocan los datos obligatorios dentro del stack  if (obligatorio.Count > 0)  {  if (obligatorio.Peek() == 55) obligatorio.Pop();  if (obligatorio.Peek() == 53) obligatorio.Pop();  }  obligatorio.Push(24);  break;  case 24:// )  AgregarTokenData(ref raizReturn, ref tempReturn);  if (obligatorio.Count > 0)  {  if (obligatorio.Peek() != 24)  {  ValoresGlobales.valores().tablaDeErrores.errores.Add(new Error(24, "Sintactico", tempElemento));  }  else  {  obligatorio.Pop();  }  }  else {  Error error = new Error();  error.ErrorCustom("No se espera caracter )", "Sintactico", tempElemento);  ValoresGlobales.valores().tablaDeErrores.errores.Add(error);    }  //Datos opcionales  opcional.Clear();  opcional.Add(29);  opcional.Add(30);  opcional.Add(31);  opcional.Add(32);  break;  case 29:// +  case 30:// -  case 31:// \*  case 32:// /  AgregarTokenData(ref raizReturn,ref tempReturn);  //Datos obligatorios  obligatorio.Push(53);  obligatorio.Push(55);  //Datos opcionales  opcional.Clear();  opcional.Add(23);  opcional.Add(53);  opcional.Add(55);  break;  case 53://Variables  case 55://Numeros  AgregarTokenData(ref raizReturn,ref tempReturn);  opcional.Clear();  opcional.Add(29);  opcional.Add(30);  opcional.Add(31);  opcional.Add(32);  if (obligatorio.Count > 0)  {  if (obligatorio.Peek() == 55) obligatorio.Pop();  if (obligatorio.Peek() == 53) obligatorio.Pop();  }  break;  default:  Error error2 = new Error();  error2.NoSeEsperaba(analizadorLexico.returnToken.tokenInfo.id,"Sintactico",tempElemento);  ValoresGlobales.valores().tablaDeErrores.AgregarError(error2);  break;  }  //Mientras existan tokens  } while (analizadorLexico.ObtenerToken());  if (obligatorio.Count > 0) {  Error error = new Error();  error.ErrorCustom("Expresion ingresada incorrecta dentro del elemento" + tempElemento.tipo.ToString(), "Sintactico", tempElemento);  }  return raizReturn;  }//Fin SintaxisExpresion  private TokenData SintaxisExpresionIf()  {  TokenData raizReturn = null;  TokenData tempReturn = null;  Stack<int> obligatorio = new Stack<int>();  List<int> opcional = new List<int>();  //Valores opcionales iniciales  opcional.Add(23);  opcional.Add(53);  opcional.Add(55);  opcional.Add(30);  //Se obtiene el primer token  if (!analizadorLexico.ObtenerToken() || analizadorLexico.returnToken == null)  {  Error error = new Error();  error.ErrorCustom("No se han obtendido la gramatica esperada 'EXP COMPARADOR EXP' en el elemento" + tempElemento.tipo.ToString(), "Sintactico", tempElemento);    }  else  {  do  {  switch (analizadorLexico.returnToken.tokenInfo.id)  {  case 23:// (  //Se colocan los valores opcionales a continuacion  opcional.Clear();  opcional.Add(23);  opcional.Add(53);  opcional.Add(55);  opcional.Add(30);  AgregarTokenData(ref raizReturn, ref tempReturn);  //Se colocan los datos obligatorios dentro del stack  if (obligatorio.Count > 0)  {  if (obligatorio.Peek() == 55) obligatorio.Pop();  if (obligatorio.Peek() == 53) obligatorio.Pop();  }  obligatorio.Push(24);  break;  case 24:// )  AgregarTokenData(ref raizReturn, ref tempReturn);  if (obligatorio.Count > 0)  {  if (obligatorio.Peek() != 24)  {  ValoresGlobales.valores().tablaDeErrores.errores.Add(new Error(24, "Sintactico", tempElemento));  }  else  {  obligatorio.Pop();  }  }  else  {  Error error = new Error();  error.ErrorCustom("No se espera caracter )", "Sintactico", tempElemento);  ValoresGlobales.valores().tablaDeErrores.errores.Add(error);  }  //Datos opcionales  opcional.Clear();  opcional.Add(29);  opcional.Add(30);  opcional.Add(31);  opcional.Add(32);  break;  case 29:// +  case 30:// -  case 31:// \*  case 32:// /  AgregarTokenData(ref raizReturn, ref tempReturn);  //Datos obligatorios  obligatorio.Push(53);  obligatorio.Push(55);  //Datos opcionales  opcional.Clear();  opcional.Add(23);  opcional.Add(53);  opcional.Add(55);  break;  case 53://Variables  case 55://Numeros  AgregarTokenData(ref raizReturn, ref tempReturn);  opcional.Clear();  opcional.Add(29);  opcional.Add(30);  opcional.Add(31);  opcional.Add(32);  if (obligatorio.Count > 0)  {  if (obligatorio.Peek() == 55) obligatorio.Pop();  if (obligatorio.Peek() == 53) obligatorio.Pop();  }  break;  case 38: // ==  case 39: // <=  case 40: // >=  case 41: // <  case 42: // >  case 43: // !=  goto cont;//Se sale del analisis  break;  default:  Error error2 = new Error();  error2.NoSeEsperaba(analizadorLexico.returnToken.tokenInfo.id, "Sintactico", tempElemento);  ValoresGlobales.valores().tablaDeErrores.AgregarError(error2);  break;  }  //Mientras existan tokens  } while (analizadorLexico.ObtenerToken());  }  cont:  if (obligatorio.Count > 0)  {  Error error = new Error();  error.ErrorCustom("Expresion ingresada incorrecta dentro del elemento" + tempElemento.tipo.ToString(), "Sintactico", tempElemento);  }  return raizReturn;  }//Fin de SintaxisExpresionIf()  private TokenData SintaxisExpresionesAsignacion() {  if (analizadorLexico.returnToken == null)  {  Error error = new Error();  error.ErrorCustom("Expresion ingresada No cumple la sintaxis en el elemento" + tempElemento.tipo.ToString(), "Sintactico", tempElemento);  ValoresGlobales.valores().tablaDeErrores.AgregarError(error);  return null;  }  switch(analizadorLexico.returnToken.tokenInfo.id){  //Caso en el que inicie con un parentesis ( ,variable o un numero  case 23:  case 53:  case 55:  case 30:  return SintaxisExpresion();  break;  //En caso sea una cadena  case 58:  TokenData temp = analizadorLexico.returnToken;  if(analizadorLexico.ObtenerToken()){  Error error = new Error();  error.ErrorCustom("Expresion ingreada no cumple requisitos de ser solamente una cadena","Sintactico",tempElemento);  ValoresGlobales.valores().tablaDeErrores.AgregarError(error);  }  return temp;  break;  break;  }  return null;  }  }  } |

### Analizador Semantico

En esta parte el analizador semantico nos sirve para reconocer los tipos dedatos ingresados dentro del DFD permitendonos reconocer tipos de datos y asi poder realizar el reconocimiento de tipos de variables, y asignar el id correspondiente al tipo de variable, ya sea esta numerica o cadena de caracteres.

Si el analizador semantico encuentra algun error dentro de el procede a reportarlo a la tabla de errores, y si hay cambios del tipo de dato de los “identificadores” se procede a realizar los cambios dentro de la tabla de las variables.

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using CompiladorDFD.Datos\_Externos;  namespace CompiladorDFD.Analizadores  {//Clase utilizada para realizar el analisis semant  class AnalizadorSemantico  {  public AnalizadorSemantico(){}  private ElementoDFD tempElemento;  public void AnalizarTipos(){  Stack<ElementoDFD> elementoIf = new Stack<ElementoDFD>();  //Se obtien la regerencia al grafo formado para por el analizador sintactico  tempElemento = ValoresGlobales.valores().elementoRaiz;  //A continuacion se procede a realizar un analisis semantico el cual consistira en esta etapa  //En la verificacion de los tipos de datos dentro de las cadenas  while (tempElemento.tipo != Elemento.fin)  {//Se verificaa el tipo de elemento que se esta analizando para luego proceder a analizar la validez de las variables  switch (tempElemento.tipo)  {  case Elemento.Asignacion:  VerificarAsignacion();  break;  case Elemento.Escritura:  verificarEscritura();  break;  case Elemento.Lectura:  verificarLectura();  break;  case Elemento.Eif: //En este caso se procede a realizar un analisis con un stack para recorrer los elementos internos dentro del if  VerificarIf();  if (tempElemento.derecha != null)  {  tempElemento = tempElemento.derecha;  elementoIf.Push(tempElemento);  continue;  }  else if (tempElemento.izquierda != null)  {  tempElemento = tempElemento.izquierda;  elementoIf.Push(tempElemento);  continue;  }  break;  case Elemento.EndIf: //Se espera que el elemento sea el fin del if para indicar el fin del analisis recursivo  if (elementoIf.Count > 0)  {  if (tempElemento.padre.derecha == elementoIf.Peek())  {  tempElemento = tempElemento.padre.izquierda;  elementoIf.Pop();  continue;  }  }  break;  case Elemento.EWhile: //Para la funcion while se utiliza la funcion de verificacion del if ya que esta  //es identica a la del if por la forma EXP COMP EXP  VerificarIf();  break;  default:  break;  }  tempElemento = tempElemento.centro; //Se procede a analizar el siguiente elemento del grafo  }  }  //Funcion utilizada para verificar que los id que se pasan como parametros sean iguales  //Esto para comprobar el tipo de dato que se va a obtener al final de las comparaciones  private bool VerificarID(ref int id,int idObtenido) {  if (id == 0) id = idObtenido;  else {  if (id != idObtenido)  return false;  }  return true;  }  private void verificarLectura() {  TokenData tempTokenData;  foreach (TokenData tokenref in tempElemento.tokenDataRef)  {  tempTokenData = tokenref;  //Mientras no se llege al final  while (tempTokenData != null)  {  if (tempTokenData.tokenInfo.id == 53) tempTokenData.tokenInfo = ValoresGlobales.valores().tablaDeSimbolos.ObtenerToken(tempTokenData.codigo);  switch (tempTokenData.tokenInfo.id)  {  case 58://cadena  case 29:// +  case 59://Variable numeros  case 60://Variable caracteres  break;  case 53://Identificador  //Se verifica el tipo de dato para ver si coincide o no  Error errorIdentificador = new Error();  errorIdentificador.ErrorCustom("La variable : '" + tempTokenData.codigo + "' No se ha declarado", "Semantico", tempElemento);  ValoresGlobales.valores().tablaDeErrores.AgregarError(errorIdentificador);  break;  }  tempTokenData = tempTokenData.tokenDataRef;  }  }  }  private void VerificarIf() {  TokenData tempTokenData;  foreach (TokenData tokenref in tempElemento.tokenDataRef)  {  tempTokenData = tokenref;  //Mientras no se llege al final  while (tempTokenData != null)  {  if (tempTokenData.tokenInfo.id == 53) tempTokenData.tokenInfo = ValoresGlobales.valores().tablaDeSimbolos.ObtenerToken(tempTokenData.codigo);  switch (tempTokenData.tokenInfo.id)  {    case 55://Numeros  case 59://Variable numeros  break;  case 60://Variable caracteres  case 58://Cadena  //Se verifica el tipo de dato para ver si coincide o no  Error errorIdentificador1 = new Error();  errorIdentificador1.ErrorCustom("Las cadenas no estan permitidas en la estructura" + tempElemento.tipo.ToString(), "Semantico", tempElemento);  ValoresGlobales.valores().tablaDeErrores.AgregarError(errorIdentificador1);  break;  break;  case 53://Identificador  //Se verifica el tipo de dato para ver si coincide o no  Error errorIdentificador = new Error();  errorIdentificador.ErrorCustom("La variable : '" + tempTokenData.codigo + "' No se ha declarado", "Semantico", tempElemento);  ValoresGlobales.valores().tablaDeErrores.AgregarError(errorIdentificador);  break;  }  tempTokenData = tempTokenData.tokenDataRef;  }  }    }  private void verificarEscritura() {  TokenData tempTokenData;  foreach (TokenData tokenref in tempElemento.tokenDataRef) {  tempTokenData = tokenref;  //Mientras no se llege al final  while (tempTokenData != null) {  if( tempTokenData.tokenInfo.id==53) tempTokenData.tokenInfo = ValoresGlobales.valores().tablaDeSimbolos.ObtenerToken(tempTokenData.codigo);  switch (tempTokenData.tokenInfo.id) {  case 58://cadena  case 55://Numeros  case 59://Variable numeros  case 60://Variable caracteres  break;  case 53://Identificador  //Se verifica el tipo de dato para ver si coincide o no  Error errorIdentificador = new Error();  errorIdentificador.ErrorCustom("La variable : '" + tempTokenData.codigo + "' No se ha declarado", "Semantico", tempElemento);  ValoresGlobales.valores().tablaDeErrores.AgregarError(errorIdentificador);  break;  }  tempTokenData = tempTokenData.tokenDataRef;  }  }  }//Fin VerificarEscritura()  private void VerificarAsignacion() {  TokenData tempTokenData;  bool error = false;  int id=0;  foreach (TokenData tokenref in tempElemento.tokenDataRef) {  error = false;  tempTokenData = tokenref.tokenDataRef;  //Mientras no se llege al final  while (tempTokenData != null) {  if( tempTokenData.tokenInfo.id==53) tempTokenData.tokenInfo = ValoresGlobales.valores().tablaDeSimbolos.ObtenerToken(tempTokenData.codigo);  switch (tempTokenData.tokenInfo.id) {  case 53://Identificador  //Se verifica el tipo de dato para ver si coincide o no  Error errorIdentificador = new Error();  errorIdentificador.ErrorCustom("La variable : '" + tempTokenData.codigo + "' No se ha declarado", "Semantico", tempElemento);  ValoresGlobales.valores().tablaDeErrores.AgregarError(errorIdentificador);  error = true;  break;  case 55://Numero  if(!VerificarID(ref id,59)){  Error error1 = new Error();  error1.ErrorCustom("Los tipos de datos no coinciden en"+tempElemento.tipo.ToString(), "Semantico", tempElemento);  ValoresGlobales.valores().tablaDeErrores.AgregarError(error1);  error = true;  }  break;  case 58://Cadena  if (!VerificarID(ref id, 60))  {  Error error1 = new Error();  error1.ErrorCustom("Los tipos de datos no coinciden en" + tempElemento.tipo.ToString(), "Semantico", tempElemento);  ValoresGlobales.valores().tablaDeErrores.AgregarError(error1);  error = true;  }  break;  case 59://Variable Numerica  if (!VerificarID(ref id, 59))  {  Error error1 = new Error();  error1.ErrorCustom("Los tipos de datos no coinciden en" + tempElemento.tipo.ToString(), "Semantico", tempElemento);  ValoresGlobales.valores().tablaDeErrores.AgregarError(error1);  error = true;  }  break;  case 60://Variable cadena  if (!VerificarID(ref id, 60))  {  Error error1 = new Error();  error1.ErrorCustom("Los tipos de datos no coinciden en" + tempElemento.tipo.ToString(), "Semantico", tempElemento);  ValoresGlobales.valores().tablaDeErrores.AgregarError(error1);  error = true;  }  break;  }  tempTokenData = tempTokenData.tokenDataRef;  }  if (!error) {  if (tokenref.codigo != null)  {  //Se asigna el valor a la variable segun los parametros asignados  ValoresGlobales.valores().tablaDeSimbolos.CambiarTipo(tokenref.codigo, id);  tokenref.tokenInfo = ValoresGlobales.valores().tablaDeSimbolos.ObtenerToken(tokenref.codigo);  }  else {  Error error1 = new Error();  error1.ErrorCustom("Los tipos de datos no coinciden en" + tempElemento.tipo.ToString(), "Semantico", tempElemento);  ValoresGlobales.valores().tablaDeErrores.AgregarError(error1);  error = true;  }  }  }  }  }  } |

### ElementoDFD

Representa un elemento relacional usado dentro de la jerarquia del grafo para ser utilizado posteriormente por el analizador sintactico y semantico , como tambien por el generador de codigo.

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Drawing;  using CompiladorDFD.Datos\_Externos;  namespace CompiladorDFD  {  public class ElementoDFD  {  //-------------------------------------------------------------------------------------------------  // Propiedas necesarias para el elemento a ser mostradas  //-------------------------------------------------------------------------------------------------  //Para definir las dimensiones  public int width;  public int height;  //para definir puntos inicales  public int top;  public int left;  //para difinir tipos de datos dentro del elemento DFD  public Elemento tipo;  public string datos;  //Para definir caracteristicas explicitas dentro de los elementos  public Color color;  public int grosor;  public bool visible;  public bool errores;  //Se crean las referncias hacia los posibles hijos que pueda obtener el elemento  public ElementoDFD izquierda;  public ElementoDFD derecha;  //Elementos que no tengan difuracasiones  public ElementoDFD centro;  //Se crea una referencia hacia el padre del objeto  public ElementoDFD padre;  //se crea una referencia para conocer el fin de un bloque  public ElementoDFD fin;  //Referencia hacia la libreria de tokens para generar el arbol sintanctico  public List<TokenData> tokenDataRef = new List<TokenData>();  //-------------------------------------------------------------------------------------------------  // Constructores de la clase  //-------------------------------------------------------------------------------------------------  public ElementoDFD() {  //No hace nada :D  }  //-------------------------------------------------------------------------------------------------  // Funciones necesarias para el funcionamiento  //-------------------------------------------------------------------------------------------------    public bool VerificarInteraccion(int x,int y ){  //Se verifica si concuerda en el eje de las y  if( top < y && height > (y - top))  //Luego se veridica si concuerda con el eje de las x  if(left< x && width>(x-left))  return true;  //De lo contrario no se hizo click dentro del objeto o el contenido del objeto  return false;  }  public void Tamanio(Size size) {  width = size.Width;  height = size.Height;  }  //Funciones utilizadas para crear y llamar al formulario respectivo para el ingreso de los datos  //Dentro de cada elementoDFD  public void LlamarFormulario(){  switch (this.tipo) {  case Elemento.Asignacion:  FrmAsignacion frmAsignacion = new FrmAsignacion();  frmAsignacion.PasarElemento( this);  frmAsignacion.ShowDialog();  break;  case Elemento.Eif:  FrmIf frmIf = new FrmIf();  frmIf.pasarElemento(this);  frmIf.ShowDialog();  break;  case Elemento.Escritura:  FrmEscritura frmEscritura = new FrmEscritura();  frmEscritura.pasarElemento(this);  frmEscritura.ShowDialog();  break;  case Elemento.Lectura:  FrmLectura frmLectura = new FrmLectura();  frmLectura.pasarElemento(this);  frmLectura.ShowDialog();  break;  case Elemento.EWhile:  FrmWhile frmWhile = new FrmWhile();  frmWhile.pasarElemento(this);  frmWhile.ShowDialog();  break;  }    }  }  } |

### Clase Token

Nos permite determinar el tipo de token asignado según las caracteristicas antes mencionadas dentro de la tabla de token presentada , ademas permite crear un listado de referencias una clase llamda TokenData que es la que se utiliza para manejar toda la informacion acerca del token como lo es su codigo , tipo y id.

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace CompiladorDFD.Datos\_Externos  {  public class Token  {  public enum TipoToken  {  PR,//Palabra Reservada  SS,//Signo de Separacion  SES,//Signo Especial Separacion  OP,//Operador  SCOM,//Signo de Comparacion  SR,//Signo de Relacionador  SP,//Signo de Puntuacion  ID,//Identificador usado para variables  NE,//Usado para numeros enteros  ND,//Usado para numeros decimales  SD,//Simbolo Desconocido  CL,//Comentario Linea  CP,//Comentario Parrafo  CA,//Cadena  CR,//Caracter  VN,//Variable Numerica  VC,//Variable Cadena  }  public int id;  public string nombre;  public TipoToken tipoToken;  //Constructor de la funcion  //Se asignan los parametros del token  public Token(int idT, string nombreT, TipoToken tipoT)  {  id = idT;  nombre = nombreT;  tipoToken = tipoT;  }  public Token()  {  }  //Estructura para guardar tipos de tokens que existen en el lexico definido  }  //Clase para manejar los datos del token  public class TokenData {  public string codigo; //Codigo que posee  public Token tokenInfo;//Informacion mas detallada del token  public TokenData tokenDataRef = null;//Referencia al token que le precede  //Constructor de la clase para inicializar los componentes  public TokenData(string dato,Token token) {  codigo = dato;  tokenInfo = token;  }  //Constructor vacio  public TokenData() { }  }  } |

### Diccionario de tokens

El diccionario de tokens como su nombre lo dice es un diccionario que posee todos los tokens que pueden ser reconocidos por el analizador lexico, permitiendo asi facilitar el reconocimiento de errores y acortar codificacion con solo utilizar el identificador de cada Token como tambien verificar la valides de estos mismos.

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace CompiladorDFD.Datos\_Externos  {//Clase utilizada para tener la referncia de todos los tokens que seran aceptados por el  //Analizador lexico , ademas de ser utilizados tambien para comprobacion de sintaxis y semantica  //Estos Se presentan a continuacion y estan separado segun su utilidad  public class TablaDeTokens  {  Dictionary<string,Token> dToken = new Dictionary<string, Token>();  Dictionary<int, Token> idToken = new Dictionary<int, Token>();  //Se registran todos los tokens existentes  public TablaDeTokens() {  //Tokens de palabras reservadas  //Token de signos de separacion  dToken.Add("(",new Token(23,"(",Token.TipoToken.SS));  dToken.Add(")",new Token(24,")",Token.TipoToken.SS));  //Token de Operadores  dToken.Add("!", new Token(28, "!", Token.TipoToken.OP));  dToken.Add("+",new Token(29,"+",Token.TipoToken.OP));  dToken.Add("-",new Token(30,"-",Token.TipoToken.OP));  dToken.Add("\*",new Token(31,"\*",Token.TipoToken.OP));  dToken.Add("/",new Token(32,"/",Token.TipoToken.OP));  dToken.Add("&&",new Token(33,"&&",Token.TipoToken.OP));  dToken.Add("||",new Token(34,"||",Token.TipoToken.OP));  dToken.Add("++", new Token(35, "++", Token.TipoToken.OP));  dToken.Add("--", new Token(36, "--", Token.TipoToken.OP));  dToken.Add("=", new Token(37, "=", Token.TipoToken.OP));  //Operadores de comparacion  dToken.Add("==", new Token(38, "==", Token.TipoToken.SCOM));  dToken.Add("<=", new Token(39, "<=", Token.TipoToken.SCOM));  dToken.Add(">=", new Token(40, ">=", Token.TipoToken.SCOM));  dToken.Add("<", new Token(41, "<", Token.TipoToken.SCOM));  dToken.Add(">", new Token(42, ">", Token.TipoToken.SCOM));  dToken.Add("!=", new Token(43, "!=", Token.TipoToken.SCOM));  //Signos Relacionales  //Signos de Puntuacion  dToken.Add(",", new Token(51, ",", Token.TipoToken.SP));  dToken.Add(".", new Token(52, ".", Token.TipoToken.SP));  //Identificadores para numeros y cadenas  dToken.Add("identificador", new Token(53, "identificador", Token.TipoToken.ID));  dToken.Add("numEntero", new Token(54, "numEntero", Token.TipoToken.NE));  dToken.Add("numDecimal", new Token(55, "numDecimal", Token.TipoToken.ND));  dToken.Add("", new Token(56, "", Token.TipoToken.ND));  dToken.Add("/\*", new Token(57, "//", Token.TipoToken.CL));  dToken.Add("cadena", new Token(58, "cadena", Token.TipoToken.CA));  dToken.Add("variableNumerica",new Token(59,"variableNumerica",Token.TipoToken.VN));  dToken.Add("variabeCadena",new Token(60,"variableCadena",Token.TipoToken.VC));  foreach (KeyValuePair<string,Token> tempdToken in dToken) {  //Se agregan todos los datos necesarios  idToken.Add(tempdToken.Value.id, tempdToken.Value);  }  }  //Funcion para obtener el tipo de token  public Token ObtenerToken(string cadena) {  Token tempToken= new Token();  if(dToken.TryGetValue(cadena,out tempToken))  return tempToken;  return null;  }  //Funcion utilizada para obtener el token apartir de su id  public Token ObtenerIdToken(int id) {  Token tempToken = new Token();  if (idToken.TryGetValue(id, out tempToken))  return tempToken;  return null;  }  }  } |

### Clase de valores globales

Esta clase es una clase declarada usando el patron Singleton que permite ser instanciada solo una vez y ademas de eso nos da la ventaja que pude ser accedida desde cualquier parte del codigo haciendo posible tener referencias a las tablas :

* Simbolos.
* Tokens.
* Errores

y ademas nos permite tener una referencia directa hacia la raiz del grafo que se ha creado permitiendo asi el acceso de todos los analizadores y el generador de codigo.

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  namespace CompiladorDFD.Datos\_Externos  {//Clase con patron Singleton, que permite instanciar solo una vez la clase y ademas de eso  //Nos permite tener acceso a ella desde cualquier parte del codigo sin crear ningun objeto  public class ValoresGlobales  {  //Variables globales a utilizar por el compilador  //Tabla para almacenar un registro de todas las variables que se han ingresado  public TablaDeSimbolos tablaDeSimbolos = new TablaDeSimbolos();  //Tabla de tokens que posee todos los tokens a ser utilizados por el programa  public TablaDeTokens tablaDeTokens = new TablaDeTokens();  //Tabla utilizada para almacenar todos los posibles errores dentro del compilador  public TablaDeErrores tablaDeErrores = new TablaDeErrores();  //Referencia hacia el elemento inicial del grafo que representa toda la estructura del DFD  public ElementoDFD elementoRaiz;  //Varible utilizada para tener acceso a la instancia de la clase  private static ValoresGlobales instance = null;  //Constructor de la clase  private ValoresGlobales() { }  //Funcion para limpiar las tablas una vez se inicia una nueva fase de compilacion  //y asi evitar problemas  public void LimpiarDatos(){  tablaDeSimbolos = new TablaDeSimbolos();  tablaDeErrores = new TablaDeErrores();    }  //Funcion en patron singleton para instanciar la clase y tener acceso a ella  //desde cualquier parte del codigo  public static ValoresGlobales valores()  {  if (instance == null)  {  instance = new ValoresGlobales();  }  return instance;  }  }  } |

### Clase Generar Codigo

En esta parte como ya se hablo antes de MSIL se procede a generar el codigo MSIL para nuestro ejecutable utilizando la libreria de .NET System.Reflection, donde se encuentra ILGenerator que nos permite generar paso a paso el codigo MSIL.

Para la generacion de codigo se toma en cuenta diferentes factores que son de vital importancia :

* La realizacion de operaciones, impresión de cadenas , sumas , restas ,ect. Depende del stack , ya que se deben ingresar al stack para poder ser ejecutadas.
* Para realizacion de operaciones matematicas se necesita ingresar a la pila los dos elementos a operar y luego ejecutar la operación a realizar ( add , sub , mul , div).
* Para utilizar variable estas deben ser declaradas antes de ser ingresadas para poder definir un tipo de datos especifico a utilizar.
* Se debe tener cuidado con los saltos y etiquetas ya que MSIL es sencible a esto.
* Se debe Poseer la definicion de todo el espacio de trabajo, Ensamblado, Ambito y sobre todo se deben declarar todos los parametros para la clase principal o clase de inicio de MSIL.

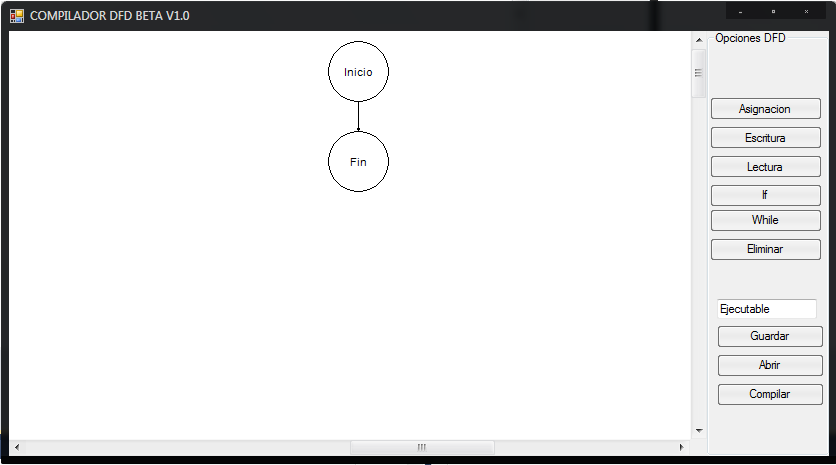
Para la generacion de codigo los elementos utilizados y mas importantes fueron :

* Tabla de simbolos : nos permite saber las variables declaradas y ademas los tipos.
* Grafo DFD : contiene toda la estructura de los token y estructuras (if ,asignacion , entre otras ).
* Tabla de errores : Nos permite determinar si es posible o no continuar con la etapa de compilacion.

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Runtime;  using System.Reflection;  using System.Reflection.Emit;  using CompiladorDFD.Datos\_Externos;  namespace CompiladorDFD.Generacion\_de\_Codigo  {  class GenerarCodigo  {  //-----------------------------------------------------------------------------------------  // METODOS UTILIZADOS PARA LA GENERACION DE CODIGO  //-----------------------------------------------------------------------------------------  //Metodo para poder realizar leer de la consola  MethodInfo metodoLeer = typeof(Console).GetMethod("ReadLine",new Type[0]);  MethodInfo metodoPausa = typeof(Console).GetMethod("ReadLine", new Type[0]);  MethodInfo metodoEscribir = typeof(Console).GetMethod("Write", new Type[]{typeof(string)});  MethodInfo metodoEscribirNum = typeof(Console).GetMethod("Write", new Type[] { typeof(Double) });  MethodInfo convertirDouble = typeof(Convert).GetMethod("ToDouble", new Type[] { typeof(string) });  //------------------------------------------------------------------------------------------  // VARIABLES DE LA CLASE  //------------------------------------------------------------------------------------------  //Clase que permite la generacion de codigo MSIL (Microsoft Intermediate language)  ILGenerator il = null;  //Elemento utilizado para tener una referencia hacia el elemento actual dentro del recorrido  ElementoDFD tempElemento;  //Diccionario utilizado para obtener control de todas las variables utilizadas  Dictionary<string, LocalBuilder> TablaDeVariables = new Dictionary<string, LocalBuilder>();  public GenerarCodigo() {  }  public void GenerarEjecutable(string nombre) {  //Se prepara el entorno para la generacion de codigo MSIL para luego ser compilado  //Se establece el dominio del aplicativo en el cual va a ser compilado en este caso  //se trabaja con el mismo del programa  AppDomain appDomain = AppDomain.CurrentDomain;  //Esta se utiliza para dar los parametros acerca de la informacion del compilable  //En este caso nos sirve para poner el nombre asignado al ensamblado  AssemblyName assemblyName = new AssemblyName();  assemblyName.Name = nombre;  //Esta clase nos permite crear un ejecutable dinamicamente es decir crear un ejecutable  //Desde el programa siempre y cuando se utilice MSLI y las clases ILGenerator para poder  //Proporcionar el lenguaje intermedio con el cual trabaja  AssemblyBuilder assemblyBuilder = appDomain.DefineDynamicAssembly(assemblyName, AssemblyBuilderAccess.Save);  //Esto nos permite crear un modulo dinamico que luego sera utilizado por TypeBuilder  ModuleBuilder moduleBuilder = assemblyBuilder.DefineDynamicModule("Modulo", nombre+".exe");  //Esta se encarga de crear la definicion de la ruta y el metodo inicial dentro de nuestro programa  TypeBuilder typeBuilder = moduleBuilder.DefineType("MiTipo", TypeAttributes.Public);  //Este es el discreptor principal que engloba todos los parametros anteriores  //Para ser utilizado con el MSIL  MethodBuilder methodBuilder = typeBuilder.DefineMethod("Main", MethodAttributes.Public | MethodAttributes.Static, null, null);  //Linea de codigo que proporciona la linea de entrada en la cual se ejecutar  //La funcion principal en este caso el "Main"  assemblyBuilder.SetEntryPoint(methodBuilder);  //Se entrega a la clase ILGenerator los parametros para poder crear un ejecutable una vez terminado  //EL ingreso de todo el codigo para crear el codigo MSLI  il = methodBuilder.GetILGenerator();  //Se comienza a formar el codigo apartir del Arbol que se creo en los analizadores obteniendo el elemento Raiz  ElementoDFD raiz = ValoresGlobales.valores().elementoRaiz;  //Ahora se comienza a recorrer todo el arbol para generar el codigo de acuerdo a la estructura planteada  Stack<ElementoDFD> elementoIf = new Stack<ElementoDFD>();  Stack<Label> ifSi = new Stack<Label>();  Stack<Label> ifEnd = new Stack<Label>();  Stack<Label> EndWhile = new Stack<Label>();  Stack<Label> BeginWhile = new Stack<Label>();  tempElemento = raiz;  //Mientras no se llege al final del grafo  while (tempElemento.tipo != Elemento.fin) {  switch (tempElemento.tipo) {  case Elemento.Asignacion:  GenerarAsignaciones();  break;  case Elemento.Escritura:  GenerarEscritura();  break;  case Elemento.Lectura:  GenerarLectura();  break;  case Elemento.Eif:  IngresarOperaciones(PolacaInversa(tempElemento.tokenDataRef[0]));  IngresarOperaciones(PolacaInversa(tempElemento.tokenDataRef[2]));  Label if\_si = il.DefineLabel();  Label if\_fin = il.DefineLabel();  ifSi.Push(if\_si);  ifEnd.Push(if\_fin);  switch (tempElemento.tokenDataRef[1].tokenInfo.id) {  case 38: // ==  il.Emit(OpCodes.Beq, if\_si);  break;  case 39: // <=  il.Emit(OpCodes.Ble, if\_si);  break;  case 40: // >=  il.Emit(OpCodes.Bge, if\_si);  break;  case 41: // <  il.Emit(OpCodes.Blt, if\_si);  break;  case 42: // >  il.Emit(OpCodes.Bgt, if\_si);  break;  case 43: // !=  il.Emit(OpCodes.Bne\_Un, if\_si);  break;  }  if (tempElemento.derecha != null)  {  tempElemento = tempElemento.derecha;  elementoIf.Push(tempElemento);  continue;  }  else if (tempElemento.izquierda != null)  {  il.Emit(OpCodes.Br, ifEnd.Peek());  il.MarkLabel(ifSi.Pop());  tempElemento = tempElemento.izquierda;  continue;  }  else {  il.Emit(OpCodes.Br, ifEnd.Peek());  il.MarkLabel(ifSi.Pop());  }  break;  case Elemento.EndIf:  if (elementoIf.Count > 0)  {  if (tempElemento.padre.derecha == elementoIf.Peek())  {  il.Emit(OpCodes.Br, ifEnd.Peek());  il.MarkLabel(ifSi.Pop());  tempElemento = tempElemento.padre.izquierda;  elementoIf.Pop();  continue;  }  }  il.MarkLabel(ifEnd.Pop());  break;  case Elemento.EWhile:  Label end\_while = il.DefineLabel();  Label begin\_while = il.DefineLabel();  Label si\_while = il.DefineLabel();  EndWhile.Push(end\_while);  BeginWhile.Push(begin\_while);    il.MarkLabel(begin\_while);  IngresarOperaciones(PolacaInversa(tempElemento.tokenDataRef[0]));  IngresarOperaciones(PolacaInversa(tempElemento.tokenDataRef[2]));    switch (tempElemento.tokenDataRef[1].tokenInfo.id)  {  case 38: // ==  il.Emit(OpCodes.Beq, si\_while);  break;  case 39: // <=  il.Emit(OpCodes.Ble, si\_while);  break;  case 40: // >=  il.Emit(OpCodes.Bge, si\_while);  break;  case 41: // <  il.Emit(OpCodes.Blt, si\_while);  break;  case 42: // >  il.Emit(OpCodes.Bgt, si\_while);  break;  case 43: // !=  il.Emit(OpCodes.Bne\_Un, si\_while);  break;  }  il.Emit(OpCodes.Br, end\_while);  il.MarkLabel(si\_while);  break;  case Elemento.EndWhile:  il.Emit(OpCodes.Br, BeginWhile.Pop());  il.MarkLabel(EndWhile.Pop());  break;  }  tempElemento = tempElemento.centro;  }  //Codigo para hacer una pausa al final del codigo mediante una lectura de teclado  il.Emit(OpCodes.Call, metodoLeer);  //Hace la devolucion del CPU a la computadora  il.Emit(OpCodes.Ret);  //Se crea el ejecutable apartir del codigo MSLI creado anteriormente  typeBuilder.CreateType();    assemblyBuilder.Save(nombre+".exe");  }  private void GenerarAsignaciones() {  foreach (TokenData raizTokenData in tempElemento.tokenDataRef)  {  TokenData tempTokenData = raizTokenData.tokenDataRef.tokenDataRef;  //Se obtiene la variable sobre la que se guardara el dato  LocalBuilder variable = ObtenerVariable(raizTokenData);  switch (tempTokenData.tokenInfo.id)  {  case 58://Cadena  //Se limpia las comillas extra que posee  string texto = tempTokenData.codigo.Substring(1, tempTokenData.codigo.Length - 2);  il.Emit(OpCodes.Ldstr, texto);  il.Emit(OpCodes.Stloc, variable);  break;  case 59:// Variable Numerica  case 55:// Numero  case 23:// (  IngresarOperaciones(PolacaInversa(tempTokenData));  il.Emit(OpCodes.Stloc, variable);  break;  case 60://Variable Cadena  break;  }    }  }//Fin GenerarAsignaciones  private void GenerarLectura() {  foreach (TokenData raizTokenData in tempElemento.tokenDataRef)  {  TokenData tempTokenData = raizTokenData;  while (tempTokenData != null)  {  switch (tempTokenData.tokenInfo.id)  {  case 29:// +  break;  case 58://Cadena  //Se limpia las comillas extra que posee  string texto = tempTokenData.codigo.Substring(1, tempTokenData.codigo.Length - 2);  //Se almacena el texto  il.Emit(OpCodes.Ldstr, texto);  //Se imprime en pantalla el texto  il.EmitCall(OpCodes.Call, metodoEscribir, null);  break;  case 59://Variable Numerica  il.EmitCall(OpCodes.Call, metodoLeer, null);  il.EmitCall(OpCodes.Call, convertirDouble, null);  il.Emit(OpCodes.Stloc,ObtenerVariable(tempTokenData));  break;  case 60://Variable Cadena  //Se lee del teclado  il.EmitCall(OpCodes.Call, metodoLeer, null);  il.Emit(OpCodes.Stloc, ObtenerVariable(tempTokenData));  break;  }  tempTokenData = tempTokenData.tokenDataRef;  }  }  }//Fin GeneracionLectura  private void GenerarEscritura() {  //Se recorren todos los token dentro del token de escritura    foreach (TokenData raizTokenData in tempElemento.tokenDataRef)  { TokenData tempTokenData = raizTokenData;    while(tempTokenData != null){  switch (tempTokenData.tokenInfo.id) {  case 55://Numero  break;  case 58://Cadena  //Se limpia las comillas extra que posee  string texto = tempTokenData.codigo.Substring(1, tempTokenData.codigo.Length - 2);  //Se almacena el texto  il.Emit(OpCodes.Ldstr, texto);  //Se imprime en pantalla el texto  il.EmitCall(OpCodes.Call, metodoEscribir,null);  break;  case 59://Variable Numerica  il.Emit(OpCodes.Ldloc, ObtenerVariable(tempTokenData));  il.EmitCall(OpCodes.Call, metodoEscribirNum, null);  break;  case 60://Variable Cadena  il.Emit(OpCodes.Ldloc, ObtenerVariable(tempTokenData));  il.EmitCall(OpCodes.Call, metodoEscribir, null);  break;  }  tempTokenData = tempTokenData.tokenDataRef;  }  }  }//Fin GenerarEscritura  private LocalBuilder ObtenerVariable(TokenData tokenData){  LocalBuilder temp;  //Se verifica si la variable ya fue registrada  if (!TablaDeVariables.TryGetValue(tokenData.codigo, out temp)) {  temp=CreacionDeVariable(tokenData.tokenInfo.id);  //De lo contrario se crea y se registra  TablaDeVariables.Add(tokenData.codigo, temp);  }  return temp;  }  private LocalBuilder CreacionDeVariable(int id) {  switch (id) {  case 59: //Variable Numerica  return il.DeclareLocal(typeof(Double));  break;  case 60: //Variable de tipo Cadena  return il.DeclareLocal(typeof(string));  break;  }  return null;  }//Fin CreacionVariable  //Funcion para insertar las operacione en el orden adecuado  private void IngresarOperaciones(List<TokenData> tokenDataList){  //Pila utilizada para los operadores  Stack<TokenData> operadores = new Stack<TokenData>();  int num = 0;  foreach (TokenData tempTokenData in tokenDataList)  {//Switch para identificar el parametro agregado  switch (tempTokenData.tokenInfo.id)  {  case 55://Numero  il.Emit(OpCodes.Ldc\_R8, Convert.ToDouble(tempTokenData.codigo));  num++;  break;  case 59: //Variable  il.Emit(OpCodes.Ldloc,ObtenerVariable(tempTokenData));  num++;  break;  default://Operadores  operadores.Push(tempTokenData);  break;  }  if (operadores.Count > 0 && num>=2)  { num--;  switch (operadores.Pop().tokenInfo.id)  {  case 29: // +  il.Emit(OpCodes.Add);  break;  case 30: // -  il.Emit(OpCodes.Sub);  break;  case 31: // \*  il.Emit(OpCodes.Mul);  break;  case 32: // /  il.Emit(OpCodes.Div);  break;  }//End switch  }//End if  }//End foreach  }  //Funcion para reacomodar las operaciones realizadas dentro del programa  private List<TokenData> PolacaInversa(TokenData tokenData){  //Pilas para realizar la polaca inversa  Stack<TokenData> operadores = new Stack<TokenData>();  //Lista para almacenar los resultados  List<TokenData> salida = new List<TokenData>();  //Si el token no es vacio  if (tokenData != null) {  TokenData tempTokenData = tokenData;  while (tempTokenData != null) {  switch (tempTokenData.tokenInfo.id)  {//En caso que sea un numero se agrega directamente a la pila  case 59:  case 55:  salida.Add(tempTokenData);  break;  //En caso que sea un (  case 23:  operadores.Push(tempTokenData);  break;  //En caso que sea un )  case 24:  while (operadores.Peek().tokenInfo.id != 23) {  salida.Add(operadores.Pop());  }  operadores.Pop();  break;  //En caso que sea + o -  case 29:// +  case 30:// -  if (operadores.Count == 0) operadores.Push(tempTokenData);  else {  while (operadores.Peek().tokenInfo.id == 31 || operadores.Peek().tokenInfo.id == 32) {  //Se saca el elemento que se encuentra dentro de la pila  salida.Add(operadores.Pop());  //Si ya no hay elementos nos salimos de la pila  if (operadores.Count == 0) break;  }  //Se agrega el elemento adentro de la pila  operadores.Push(tempTokenData);  }  //Se verofoca si existe un operador de mayor presedencia dentro de la pila para sacarlo de la pila y agregarlo  break;  //En caso que sea \* o /  case 31:// \*  case 32:// /  if (operadores.Count == 0) operadores.Push(tempTokenData);  if (operadores.Peek().tokenInfo.id == 31 || operadores.Peek().tokenInfo.id == 32) {  salida.Add(operadores.Pop());  //Si ya no hay datos dentro de la pila se sale  if (operadores.Count == 0) break;  }  operadores.Push(tempTokenData);  break;  }  //se avanza a la siguiente posicion  tempTokenData = tempTokenData.tokenDataRef;  }  }  while (operadores.Count > 0) salida.Add(operadores.Pop());  return salida;  }//Fin Polaca Inversa  }  } |

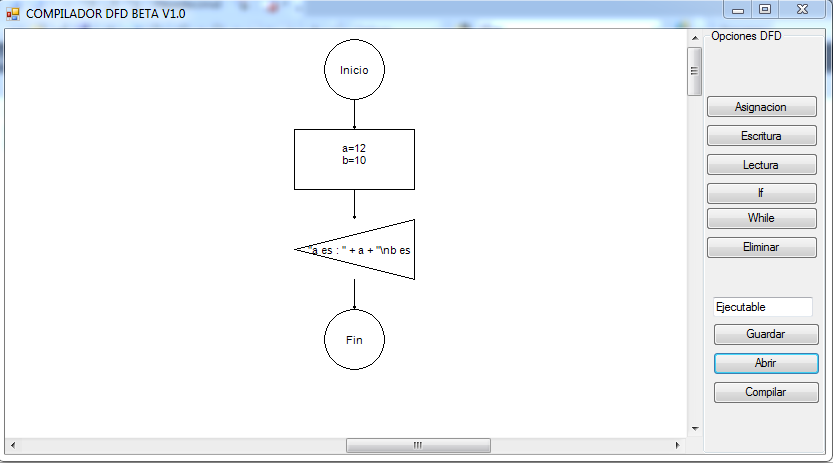
### Formularios

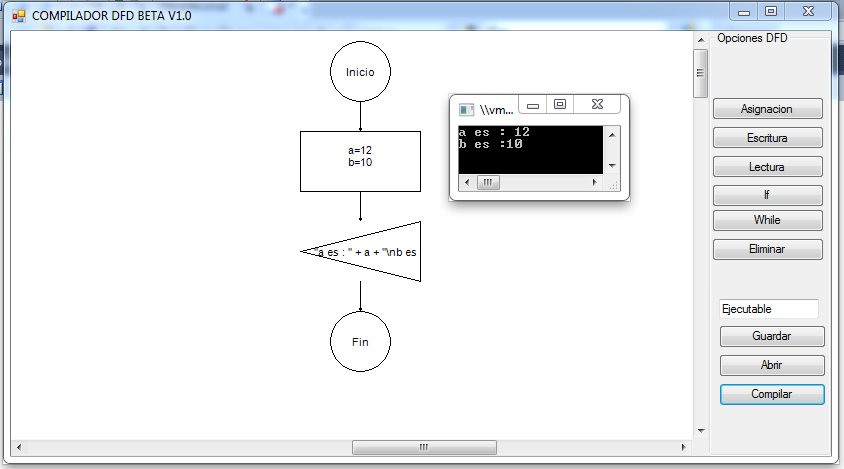
### Formulario Principal



|  |
| --- |
| using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Linq;  using System.Text;  using System.Windows.Forms;  using CompiladorDFD.Analizadores;  using CompiladorDFD.Datos\_Externos;  using CompiladorDFD.Generacion\_de\_Codigo;  namespace CompiladorDFD  {  public partial class FrmDFDCompiler : Form  {  public FrmDFDCompiler()  {  InitializeComponent();  }  private void FrmDFDCompiler\_Load(object sender, EventArgs e)  {  ucdfd1.Iniciar();  }  private void ucdfd1\_Load(object sender, EventArgs e)  {  }  private void button1\_Click(object sender, EventArgs e)  {  ucdfd1.AgregandoElemento(Elemento.Asignacion);  }  private void btnCompilar\_Click(object sender, EventArgs e)  {  ValoresGlobales.valores().LimpiarDatos();  ValoresGlobales.valores().elementoRaiz = ucdfd1.ObtenerRaiz();  AnalizadorSintactico analisSintactico = new AnalizadorSintactico();  AnalizadorSemantico analisisSemantico = new AnalizadorSemantico();  GenerarCodigo generarCodigo = new GenerarCodigo();  analisSintactico.GenerarArbol();  if (!ValoresGlobales.valores().tablaDeErrores.Existen()) analisisSemantico.AnalizarTipos();  //Se verifica si existen errores o no dentro del codigo antes de compilarlo  if (!ValoresGlobales.valores().tablaDeErrores.Existen())  {  if (txtNombre.Text == "") txtNombre.Text = "Ejecutable";  generarCodigo.GenerarEjecutable(txtNombre.Text);  System.Diagnostics.Process.Start(txtNombre.Text + ".exe");    }  else {  FrmTablaDeErrores frmTablaDeErrores = new FrmTablaDeErrores();  frmTablaDeErrores.Show();  }  }  private void btnLectura\_Click(object sender, EventArgs e)  {  ucdfd1.AgregandoElemento(Elemento.Lectura);  }  private void btnEscritura\_Click(object sender, EventArgs e)  {  ucdfd1.AgregandoElemento(Elemento.Escritura);  }  private void button2\_Click(object sender, EventArgs e)  {  ucdfd1.AgregandoElemento(Elemento.Eif);  }  private void BtnWhile\_Click(object sender, EventArgs e)  {  ucdfd1.AgregandoElemento(Elemento.EWhile);  }  private void BtnFor\_Click(object sender, EventArgs e)  {  ucdfd1.AgregandoElemento(Elemento.Efor);  }    private void BtnGuardar\_Click(object sender, EventArgs e)  {  SaveFile = new SaveFileDialog();  SaveFile.Filter = "txt files (\*.txt)|\*.txt";  SaveFile.DefaultExt = "txt";  SaveFile.Title = "Guardar Archivo DFD";  SaveFile.ShowDialog();  // If the file name is not an empty string open it for saving.  if (SaveFile.FileName != "")  {  ValoresGlobales.valores().elementoRaiz = ucdfd1.ObtenerRaiz();  GuardarCargarDFD guardar = new GuardarCargarDFD();  guardar.GuardarArchivo(SaveFile.FileName);  }  }  private void Eliminar\_Click(object sender, EventArgs e)  {  ucdfd1.Eliminar();  }  private void button2\_Click\_2(object sender, EventArgs e)  {  OpenFile = new OpenFileDialog();  OpenFile.Filter = "txt files (\*.txt)|\*.txt";  OpenFile.FilterIndex = 2;  if (OpenFile.ShowDialog() == DialogResult.OK)  {  try  {  ValoresGlobales.valores().elementoRaiz = ucdfd1.ObtenerRaiz();  GuardarCargarDFD guardar = new GuardarCargarDFD();  ucdfd1.CargarDFD(guardar.Leer(OpenFile.FileName));  }  catch (Exception ex)  {  MessageBox.Show("Error No se logro abrir el archivo deseado probablemente el formato no es el adecuado");  }  }    }  }  } |

## IMPRESIONES DE PANTALLA

****

****

# **CONCLUSIONES**

* El proyecto Del compilador DFD nos trajo nuevos concimientos sobre el funcionamiento de un compilador.
* Se pudo comprobar que se pude pasar de cualquier lenguaje en este caso un DFD a la generacion de un ejecutable por un proceso de analisis y compilacion
* Cada parte de un compilador tiene una mision fundamental, como tambien un proposito de ser.
* Se debe considera el hecho que no se necesita tenr un lengujae muy complejo para poder tener un ejecutable, basta con poseer estructuras basicas.

# **REFERENCIAS**

Libro Texto :

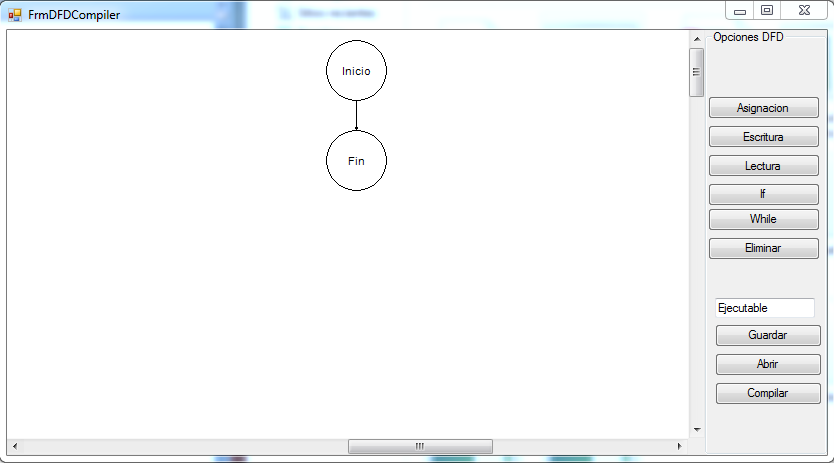
**Compiladores, Principios, Tecnicas Y Herramientas**  
*Autores: Alfred V. Aho – Ravi Sethi : At&T Bell Laboratories. Murray Hill, New Jersey – Jeffrey D. Ullman : Stanford University | Pearson*

Sitios Web :

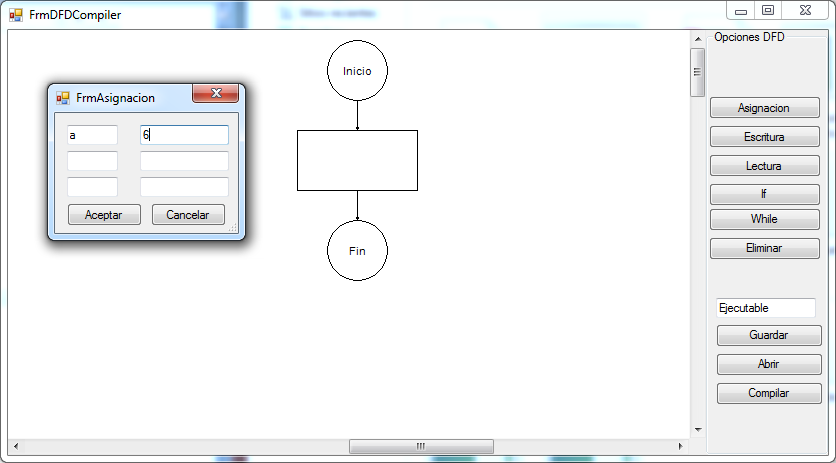
* <http://foro.elhacker.net/programacion_general/fases_de_compilacion-t302095.0.html>
* <ftp://soporte.uson.mx/publico/02_ING.SISTEMAS.DE.INFORMACION/Lenguajes/Analizadorlexico.pdf>
* <http://elrincondemerida.blogspot.com/2008/04/analisis-sintactico.html#tema2>
* <http://www.google.com.sv/url?sa=t&rct=j&q=historia%20de%20los%20compiladores&source=web&cd=2&sqi=2&ved=0CFEQFjAB&url=http%3A%2F%2Fwww.cs.buap.mx%2F~hilda%2Fnotascompila.doc&ei=tgXyT7fHLuHv0gGbk-36Ag&usg=AFQjCNFAJ65FY-EiyLJitKLzIEvjZv9YNA>
* <http://es.wikipedia.org/wiki/Diagrama_de_Flujo_de_Datos>
* <http://ing.utalca.cl/~rgarrido/recursos/dfd10/manual_dfd.pdf>
* <http://rockanlover.wikispaces.com/Simbolos>
* http://judelco.wordpress.com/2011/12/15/introduccion-al-lenguaje-ensamblador-il-de-net/
* http://www.vbdotnetheaven.com/uploadfile/manish1231/reflection-and-reflection-emit-in-vb-net-part-1/
* http://msdn.microsoft.com/en-us/magazine/cc136756.aspx

# Manual de Usuario DFD

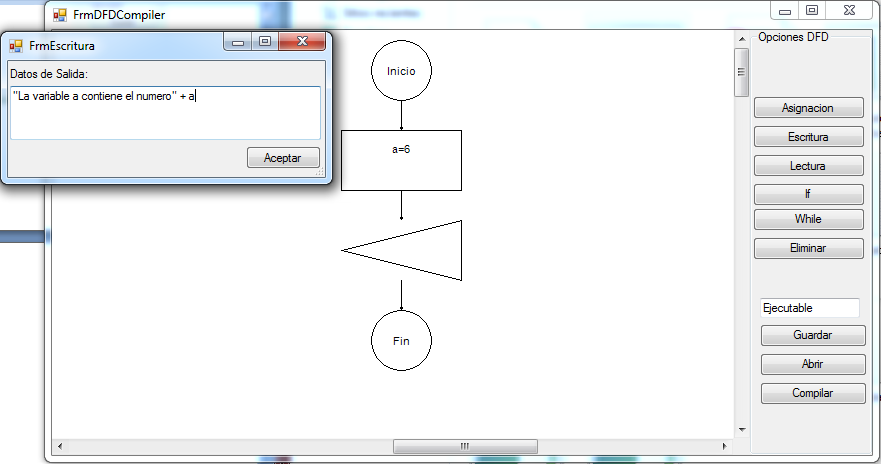
1. **Pantalla Principal**: se muestran los controles separados por funcionalidad, los botones superiores corresponden a los controles del DFD en sí, y los inferiores para manipular los archivos generados además de compilarlos. En la parte izquierda se muestra el estado inicial y final ya que estos no pueden faltar.



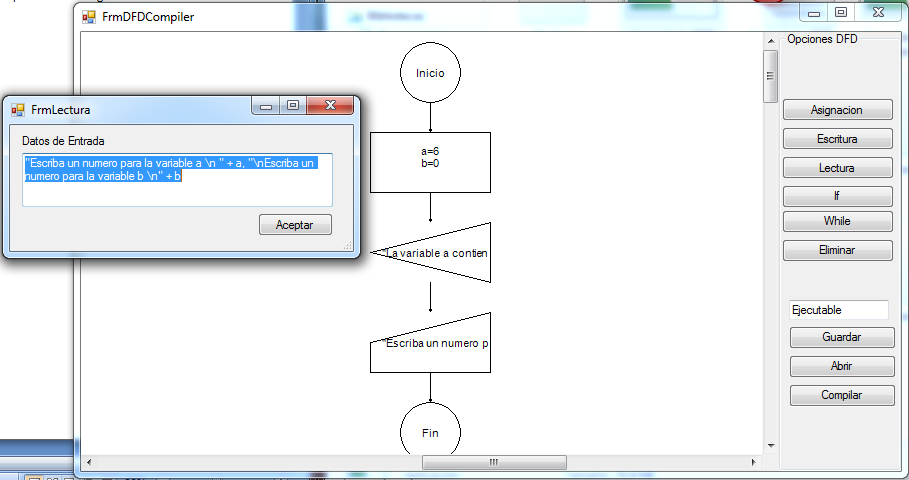
1. **Control de asignación**: este nos sirve para declarar variables y asignarles un valor en el textbox de la izquierda va la variable y en el derecho va el valor, como cada uno de los controles lo único que debemos hacer es dar clic sobre el control y luego se arrastra a la posición dentro del DFD donde se quiere utilizar.



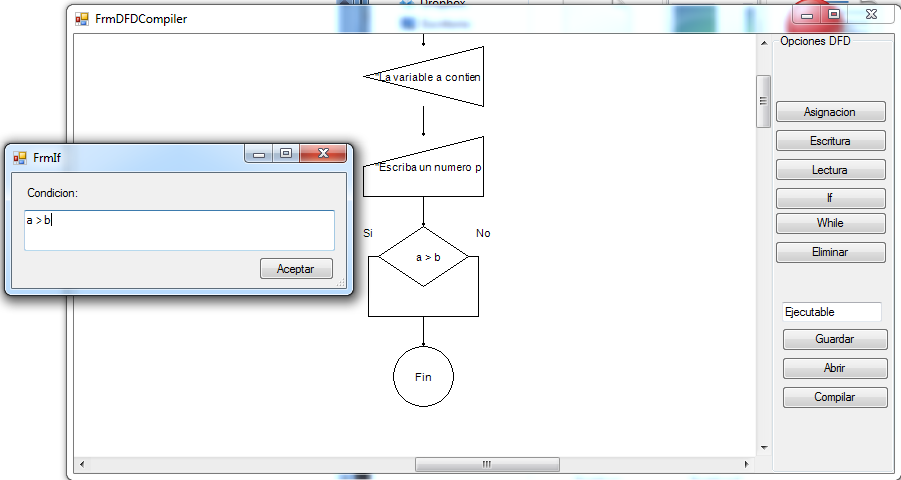
1. **Control de escritura**: este control sirve para mostrar en pantalla algún mensaje que se quiera mostrar, se pueden utilizar variables inicializadas anteriormente. La sintaxis que se debe utilizar es la siguiente: para una cadena de texto, debe ir entre comillas dobles (“ ”), para concatenar cadenas se utiliza el operador (+).



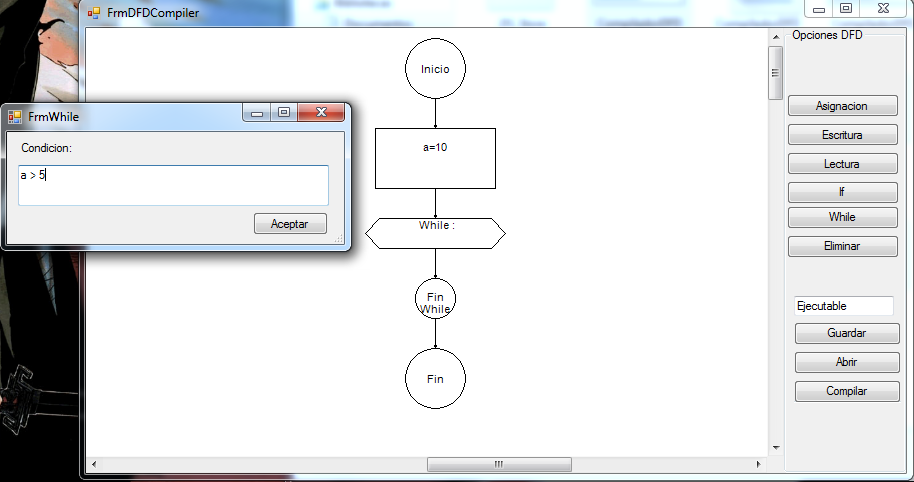
1. **Control de lectura**: este se utiliza en caso que se quiera que el usuario digite el valor de una variable, la variable tiene estrictamente que estar declarada anteriormente. La sintaxis es la siguiente:
2. “Cadena de texto” + a, “Cadena de texto2” + b.
3. A, b



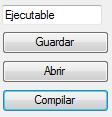
1. **Control If**: este esta apegado a una condición, se debe colocar un control de cada lado, en la parte donde sea verdadero (SI) o en la parte donde sea falso (NO). En primer lugar se debe escribir una condición. La sintaxis es la siguiente:
2. *Variable1* operador *Variable2*.



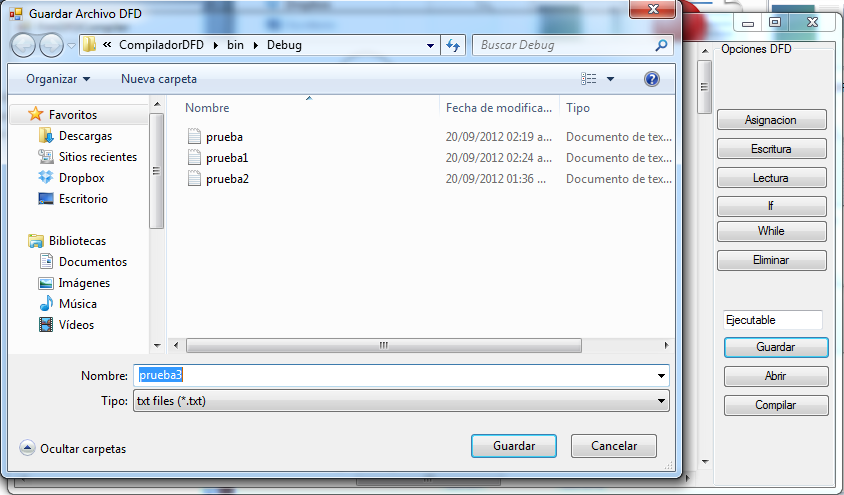
1. **Control While**: al igual que el control If, este también está ligado a una condición, la sintaxis es la misma, en este caso se termina de ejecutar esa parte cuando se deje de cumplir la condición.



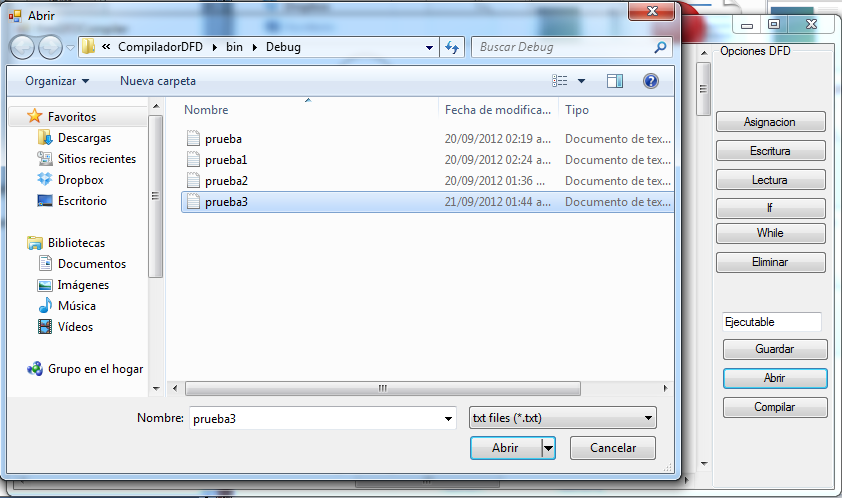
1. **Controles de manipulación de archivos y compilación**: mediante estos botones, podemos guardar bajo el nombre que deseemos, abrir un documento guardado previamente, y compilar y ejecutar el código directamente, el nombre del ejecutable es el nombre que escribamos en el textbox.



Al dar clic en guardar nos aparece la ventana de “Guardar como” donde se especifica el nombre del archivo que queremos guardar en formato de archivo plano.



Al dar clic en abrir, se puede abrir un archivo generado anteriormente y se cargará directamente en el DFD.



Al dar clic en compilar, se compila y ejecuta el código generado mediante el DFD, en este caso se ha compilado un bucle while.

