“DFC 1.0” HERRAMIENTA SOFTWARE PARA LA CONSTRUCCIÓN Y EJECUCIÓN DE DIAGRAMAS DE FLUJO Y LA GENERACIÓN DEL CÓDIGO FUENTE EN LENGUAJE C++

LENCY GLUSBEY GAMBA MARTÍNEZ
JESÚS DAVID RUEDA POLO

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2004

“DFC 1.0” HERRAMIENTA SOFTWARE PARA LA CONSTRUCCIÓN Y EJECUCIÓN DE DIAGRAMAS DE FLUJO Y LA GENERACIÓN DEL CÓDIGO FUENTE EN LENGUAJE C++

LENCY GLUSBEY GAMBA MARTÍNEZ
JESÚS DAVID RUEDA POLO

Proyecto de grado para optar al título de
Ingeniero de Sistemas

Director
DEA. HÉCTOR NIÑO QUIÑÓNEZ

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FISICOMECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2004

A Dios,

A mis padres, por ayudarme a alcanzar mis metas,

A mis hermanos y familiares porque sembraron en mi una esperanza,

A Mr. Price por estar presente en momentos importantes de mi vida,

A mis amigas y amigos, de quienes siempre tuve algo que aprender...

LENCY

A Dios por ser la principal fuente de vida,
A mis madres Yenis y Tata, por que sin ellas no soy nada,
A mis padres Jesús y Wilfrido por enseñarme el valor de la amistad,
A mis hermanos por su respeto,
A la Negra por tantas cosas ...,
A EDWIN y a KAILY por ser mi fuente de inspiración y
A Yenny por su incondicionalidad.

JESÚS DAVID

AGRADECIMIENTOS

A la Universidad Industrial de Santander porque nos enseñó que la vida es de otra manera,

Al profesor Héctor Niño por convertirse en una base fundamental para el desarrollo de este proyecto y sobre todo por ayudar a materializar este sueño,

A Zúñiga y Eder por ser nuestros hermanos y por su incondicional apoyo y colaboración en las duras y en las suaves,

A nuestros amigos: Zulma, Laura, Mayoly, Marco, Jaider, Yovan, Lilian, Gina, Dora, Jorge, Leo y Tasco, por ser los de siempre,

A los padres políticos Teddys, Juan Francisco y Boris por enseñarnos que cuando el tiempo acabe y el mundo reclame, la historia dirá que estuvimos aquí,

A Jaime Almeyda por su confianza y porque lo bueno va por dentro,

Al Ing. Hermes Ortega porque es cierto que los buenos programadores resuelven los problemas, pero los mejores los evitan,

A What happens por ser el maestro de las pruebas de DFC,

A todos los que faltaron nombrar, porque no caben en esta hoja, pero están en nuestros recuerdos.

CONTENIDO

LISTA DE FIGURAS

pág.

LISTA DE ANEXOS

pág.

RESUMEN

TITULO: "DFC 1.0" HERRAMIENTA SOFTWARE PARA LA CONSTRUCCIÓN Y EJECUCIÓN DE DIAGRAMAS DE FLUJO Y LA GENERACIÓN DEL CÓDIGO FUENTE EN LENGUAJE C++[*]

AUTORES: GAMBA MARTÍNEZ, Lency Glusbey.
RUEDA POLO, Jesús David. [**]

PALABRAS CLAVES: Algoritmo, Diagrama de Flujo, Código Fuente, Lógica, Programación, Entrega por Etapas, Modelado Orientado a Objetos, Manejo Dinámico de Memoria.

DESCRIPCIÓN:

Una de las mayores desventajas en la optimización de las soluciones a los problemas informáticos es la dificultad de relacionar sus tres conceptos básicos como son la parte algorítmica, la diagramación y la codificación; de esta manera DFC 1.0 contribuye como herramienta software en el proceso de consolidación de las técnicas de la programación, actuando como un excelente apoyo para la verificación de lo factible, eficiente y posteriormente lo óptimo de las soluciones propuestas.

Este proyecto de investigación está orientado aquellas personas que desean iniciarse en el mundo de la programación pasando desde sus conceptos básicos hasta llegar a la codificación, permitiendo una adecuación de las técnicas necesarias para la solución de problemas e incursionar en el aprendizaje de un lenguaje de programación como lo es C++.

Como metodología de desarrollo se utilizó entrega por etapas ya que permite desarrollo de subconjuntos útiles del producto; y además el producto para el usuario final se puede definir de tal forma que se pueden hacer entregas intermedias significativas antes de entregar el producto final. Es importante analizar que esta metodología propicia un intercambio de conocimientos y de autocrítica al sistema, lo que conlleva a que se produzcan muchas pruebas antes de liberar una nueva versión, así como mejoras rápidas a problemas que puedan surgir durante su uso. De igual forma se aprovecharon todas las potencialidades del manejo dinámico de memoria y del modelado orientado a objetos.

---

[*] Proyecto de grado en la modalidad investigación.

[**] Facultad de Ingenierías Fisicomecánicas. Ingeniería de Sistemas e Informática.
Director del Proyecto: DEA. Héctor Niño Quiñónez.

SUMMARY

TITLE: "DFC 1.0" SOFTWARE TO BUILD AND RUN FLOW CHARTS AND GENERATE C++ SOURCES. [*]

AUTHORS: GAMBA MARTÍNEZ, Lency Glusbey.
RUEDA POLO, Jesús David. [**]

KEY WORDS: Algorithm, Flow Charts, Sources, Logic, Programming, Delivery by Phases, Object Oriented Programming, Dynamic Usage of Memory.

DESCRIPTION:

One of the major disadvantages in the solution optimization in computing problems is the difficulty of linking their three fundamental concepts which are Algorithm, Charting, and Source. DFC 1.0 contributes as a tool in the consolidation process of programming techniques, working as an excellent support to verify the feasible and efficient of the proposed solutions and later on their optimization.

This project is addressed to people that wish to begin in the programming field starting on basic concepts until sourcing, allowing the suitability of the necessary techniques for the problem's solutions and come into a programming language learning as C++.

The methodology of the project was so called "delivery by phases" because it allows the development of useful subsets of the package and defines it for the end user in a way that it makes possible significant intermediate deliveries before handing in the final product. It is important to analyze that this methodology allows a knowledge exchange and self-critic of the system taking to produce a lot of tests before releasing a new version, at the same time, quick improvements to problems that may emerge during its usage. Additionally, the potentialities of the dynamic usage of memory and the object oriented modeling were taken in account in this project.

---

[*] Graduate Project Investigation Modality

[**] Physical and Mechanical Engineering Faculty, Systems Engineering and Computing School. Project Director: DEA. Héctor Niño Quiñónez.

INTRODUCCIÓN

Actualmente la informática juega un papel fundamental en la resolución de muchos problemas de la sociedad y de ahí la importancia de comprender sus diferentes términos y aprovechar de la manera más óptima su utilización.

De esta forma en la parte de la programación de computadores se deben relacionar una gran cantidad de conceptos para alcanzar la mejor solución de los problemas. Actualmente las diferentes facetas que involucra un problema informático como lo es la parte algorítmica, la diagramación y la codificación se usan de manera experimental y no interrelacionada, lo cual lleva a que las soluciones planteadas puedan ser eficientes, pero en la mayoría de los casos no son óptimas.

La programación puede considerarse como un arte y es así como exige técnicas adecuadas que faciliten su uso y una mejor utilización de sus potencialidades. Se debe estructurar el pensamiento en esta área de tal forma que se garantice que cada etapa va a hacer evaluada y se analizará su aporte a la solución total.

Si no se garantiza lo anterior se seguirá actuando de una manera experimental, no habrá un apropiado análisis de la situación y posiblemente la solución de los problemas demandaría un mayor esfuerzo de lo realmente necesario.

Para atacar un problema, debemos hacer una evaluación estricta de sus diferentes implicaciones y formular un conjunto de pasos que permitan su adecuada solución; en esta parte la experiencia juega un papel fundamental

y facilita en cierta medida hacer un correcto uso de esta etapa; inmediatamente se procede a la validación de este conjunto de pasos, conocido como algoritmo, llevándolo a una nomenclatura clara como los son los diagramas de flujo, después se procede a comprobar que satisface adecuadamente las necesidades de nuestro problema y finalmente si es requerido se procede a la implementación de esta solución en un determinado lenguaje de programación.

En este sentido aparece la herramienta software desarrollada en este proyecto como una ayuda en el proceso de aprendizaje para relacionar de una forma adecuada los tres conceptos básicos de los problemas informáticos; las personas que quieran avanzar en el área de la programación desarrollaran sus técnicas mediante un constante uso de la herramienta, la cual actúa como un excelente apoyo para verificar lo factible y posteriormente lo óptimo de las soluciones propuestas, de esta forma mediante un uso constante de la herramienta, el usuario aprende a interactuar con los conceptos y gana la experiencia necesaria para satisfacer problemas de este tipo.

La información del presente documento se clasifica en tres capítulos: En el primer capítulo se encuentran los fundamentos generales del proyecto, se plantean los objetivos y los alcances del proyecto. En el segundo capítulo se presenta la información teórica general y especifica necesaria para el desarrollo del proyecto. En el tercer capítulo muestra el desarrollo del proyecto a través de cada una de las fases de la metodología escogida.

Por ultimo se presentan las conclusiones y recomendaciones del proyecto y todos los anexos de la herramienta.

1. PRESENTACIÓN

1.1. TÍTULO DEL PROYECTO

“DFC 1.0” Herramienta software para la construcción y ejecución de diagramas de flujo y la generación del código fuente en lenguaje C++.

1.2. OBJETIVOS

❖ GENERAL

Contribuir con el mejoramiento de los procesos de enseñanza-aprendizaje, que posibiliten un mejor desempeño en la construcción de algoritmos mediante el desarrollo de una herramienta informática.

❖ ESPECÍFICOS

La herramienta software debe permitir:

- La construcción de diagramas de flujo según las técnicas de desarrollo de algoritmos.

- La ejecución de la lógica representada en el diagrama y la generación del código fuente en lenguaje C++, permitiendo una corroboración de los resultados esperados y del posible código que el usuario haya realizado, convirtiéndose en una herramienta que ayude a la consolidación de los conocimientos en el área de la programación.

- Obtener las sentencias en lenguaje C++ de una estructura del diagrama de flujo, facilitando el correcto análisis en cuanto a código de la parte que se quiere implementar.
- Favorecer el proceso enseñanza-aprendizaje y no solo como un solucionador de problemas.

- Salvar el trabajo realizado y poder reconocer la información guardada, con el objeto de poder editarla y profundizar en las posibles soluciones o modificaciones que presente un problema.

1.3. JUSTIFICACIÓN

Últimamente se presentan dificultades para poder relacionar adecuadamente y de una manera eficiente los conceptos básicos para la programación, hay que recalcar que se considera esta rama como un arte y por lo tanto necesita de cierto ingenio y dedicación, pero sobre todo de una técnica específica.

Para poder enfrentar un problema que implique una solución informática, debemos partir de la elaboración de un conjunto de pasos lógicos que lleven a obtener los resultados esperados; es aquí donde la herramienta brinda una ayuda fundamental para poder empezar a enlazar los conceptos y caminar hacia una meta directa y concisa.

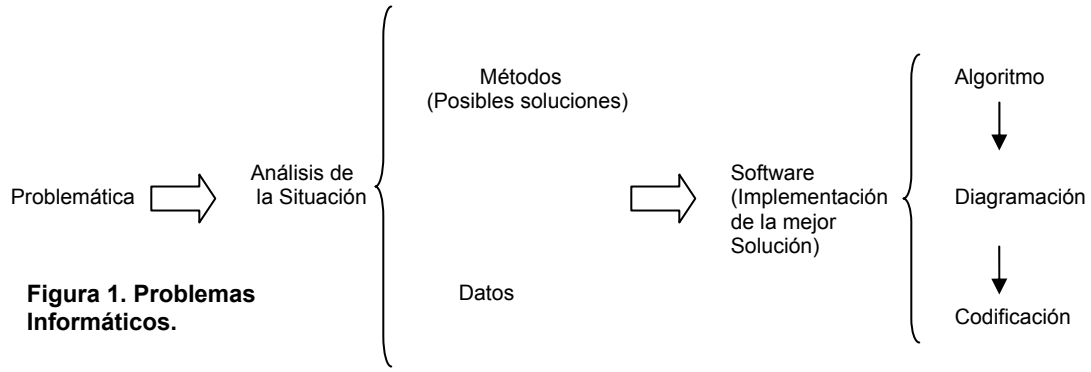

Figura 1. Problemas Informáticos.

Mediante la construcción del diagrama de flujo los usuarios plasman sus ideas en un lenguaje gráfico y de esta forma se puede dar el salto hacia el siguiente paso. En este momento las ideas estarán debidamente organizadas y se podrá observar la coherencia de la posible solución. Al poder ejecutar los diagramas analizaremos si el conjunto de pasos planteados, satisface las necesidades del problema, hay que tener presente que en la utilización de herramientas software se pueden generar dos tipos de errores: los errores que obedecen a un incorrecto uso de la herramienta y no llegando a ningún resultado y los errores que nacen de la incorrecta visión de la problemática planteada. Por lo tanto nos encontramos en un punto donde el usuario es capaz de analizar qué tipo de errores se le presentaron y cómo debe solucionarlos, sin plasmar su idea en un lenguaje especifico, pues aún se encuentra en la diagramación y ésta brinda mayor oportunidad de análisis.

Antes que el usuario llegue a la parte de la codificación, puede editar su trabajo y de esta forma mirar qué algoritmo se adapta mejor a la solución esperada, así podemos analizar que un problema puede tener muchas soluciones y cultivar una técnica que facilite la obtención de un algoritmo para un determinado caso.

Con la generación de código el usuario ha llegado a la fase final de la solución, pasando por todas las etapas intermedias y afianzando sus técnicas sobre algoritmos. En esta parte se puede confrontar la codificación planteada por la herramienta con la codificación que el usuario ha realizado, esto contribuye a que cada usuario utilice la herramienta como medio de validación de su trabajo y favorezca los aspectos de aprendizaje.

Cabe resaltar que actualmente la codificación es uno de los mayores problemas de las personas que quieren avanzar en el campo de la programación, esta herramienta quiere facilitar en primera instancia la

obtención de código fuente y luego mediante el constante uso y el desarrollo de una técnica adecuada, se podrá usar como una herramienta auxiliar, pues aspectos importantes de la herramienta como poder generar el código de una parte del diagrama de flujo, permitirá sin lugar a duda que el usuario reconozca la sintaxis de cada parte del diagrama y comprenda como la interrelación de cada parte aporta a la solución del problema y de esta forma analice cómo enfrentar un determinado problema y pueda seguir perfeccionando su técnica.

En la parte técnica será un gran avance para el área de desarrollo de software, pues se profundiza en la parte de generación de código a partir de una estructura dada, convirtiéndose en un recurso novedoso para nuestra sociedad, además cabe rescatar que es un proyecto que puede seguir su desarrollo incursionando en la generación de código en otro lenguaje.

2. MARCO TEÓRICO

2.1. TEORÍA DE ALGORITMOS

Un algoritmo es el conjunto de operaciones y procedimientos a realizar que deben seguirse en un orden para resolver un problema. La palabra "algoritmo" deriva del nombre latinizado del gran matemático árabe Mohamed Ibn Moussa Al Kow Rizmi, el cual escribió sobre entre los años 800 y 825 su obra Quitab Al Jabr Al Mugabala, donde se recogía el sistema de numeración hindú y el concepto del cero. Fue Fibonacci, el que tradujo su obra al latín y la inició con las palabras: Algoritmi dicit.

El lenguaje algorítmico es aquel por medio del cual se realiza un análisis previo del problema a resolver y encontrar un método que permita resolverlo.

El lenguaje informático es aquel por medio del cual dicho algoritmo se codifica a un sistema comprensible por el ordenador o computadora. Este tipo de lenguaje es más cercano a la máquina que al ser humano y podemos distinguir distintos tipos dependiendo de la proximidad a la máquina. Se denomina lenguaje de alto nivel aquel que es más cercano a la comprensión humana y lenguaje de bajo nivel a aquellos que son más comprensibles por la máquina.

La lógica de los algoritmos se fundamenta en la teoría causa-efecto y en la idea de que podemos controlar los efectos si controlamos las causas. Desde esta óptica, entendemos que toda actividad es susceptible de un análisis retrospectivo.

Para analizar un algoritmo se plantea un problema y la experiencia nos dice cuáles son los pasos mínimos para solucionarlo. Si el problema puede ser solucionado por un computador, el algoritmo es aún más sencillo cuando conocemos la lógica involucrada en estas máquinas.

En esencia, todo problema se puede describir por medio de un algoritmo, Para llegar a la realización de un programa es necesario el diseño previo de un algoritmo, de modo que sin algoritmo no puede existir un programa. El diseño de algoritmos requiere creatividad y conocimientos profundos de la técnica de programación. Los algoritmos son independientes de los lenguajes de programación. En cada problema el algoritmo puede escribirse y luego ejecutarse en un lenguaje diferente de programación. El algoritmo es la infraestructura de cualquier solución, escrita en cualquier lenguaje.

2.1.1. Características de los algoritmos. Se caracterizan por ser:

- Preciso. Definirse de manera rigurosa, sin dar lugar a ambigüedades.

- Definido. Si se sigue un algoritmo dos veces, se obtendrá el mismo resultado, siempre y cuando no sea un problema estocástico.

- Finito. Debe terminar en algún momento.

- Debe tener uno o más elementos de entrada, es decir, debe tener por lo menos una instrucción que ordene averiguar el dato o los datos.

- Debe producir un resultado. Los datos de salida serán los resultados de efectuar las instrucciones. Los datos de entrada pueden ser ninguno, pero los de la salida deben ser alguno o algunos.

Se concluye que un algoritmo debe ser suficiente y breve, es decir, no exceder en las instrucciones ni quedarse corto. Entre dos algoritmos que lleven a un mismo objetivo, siempre será mejor el más corto**.**

2.1.2. Etapas para la solución de un problema por medio del computador.

1. Análisis del problema, definición y delimitación (macroalgoritmo). Considerar los datos de entrada, el proceso que debe realizar el computador y los datos de salida.

2. Diseño y desarrollo del algoritmo. Pseudocódigo o escritura natural del algoritmo, diagramas de flujo, Diagramas rectangulares.

3. Prueba de escritorio. Seguimiento manual de los pasos descritos en el algoritmo. Se hace con valores bajos y tiene como fin detectar errores.

4. Codificación. Selección de un lenguaje y digitación del pseudocódigo haciendo uso de la sintaxis y estructura gramatical del lenguaje seleccionado.

5. Compilación o interpretación del programa. El software elegido convierte las instrucciones escritas en el lenguaje a las universales comprendidas por el computador.

6. Ejecución. El programa es ejecutado por la máquina para llegar a los resultados esperados.

7. Depuración (debug). Operación de detectar, localizar y eliminar errores de mal funcionamiento del programa.

8. Evaluación de resultados. Obtenidos los resultados se les evalúa para verificar si son correctos. Un programa puede arrojar resultados incorrectos aún cuando la ejecución es perfecta.

2.2. DIAGRAMAS DE FLUJO

Un Diagrama de Flujo es la representación gráfica de los pasos a seguir para lograr un objetivo, que habitualmente es la solución de un problema. Por lógica se entiende, en algunos libros, a la componente lógica de un programa y que se puede representar en un diagrama de flujo o de otra forma.

De acuerdo a lo anterior, la componente lógica de un programa se puede expresar en un diagrama de flujo, en un programa estructurado, en un programa codificado en un lenguaje de programación, o de alguna otra manera.

2.2.1. Elementos e instrucciones a usar en un diagrama de flujo. A continuación se detalla y establece lo que se puede hacer en un diagrama de flujo y como indicarlo.

2.2.1.1. Constantes. Por constante numérica se entiende un número, y se representa por su símbolo habitual en base 10. Son constantes numéricas:
12 13.5 0 3.14159.

Observamos que se usa punto y no coma para la parte decimal de un número.
Por constante alfanumérica se entiende un conjunto de caracteres alfabéticos o numéricos de nuestro uso habitual; es decir las letras del alfabeto, los diez dígitos, los símbolos de puntuación habitual, operaciones matemáticas, y

otros símbolos. Una constante alfanumérica se expresa dejando ese conjunto de caracteres escrito entre comillas dobles.

Son constantes alfanuméricas:
"INGENIEROS" "El promedio vale = 5.3 "

2.2.1.2. Variables. Por variable se entiende una cantidad a la que se alude por su nombre y no por su contenido, ya que su contenido seguramente cambiará a medida que el programa se vaya ejecutando.

Cada variable tendrá su ubicación en la memoria principal del computador, lugar en que se almacenará el contenido que ella tiene. Así, cada variable tiene una dirección en la memoria del computador; de conocer y manejar esa dirección se encarga el sistema operativo. Se conoce y alude a esa zona por el nombre que se le ha dado.

2.2.1.3. Proposición de asignación. La forma de una proposición de asignación es: a = b; donde b es una constante, variable o expresión permitida, y a es el nombre de la variable donde se dejará el valor resultante de evaluar b.

Por expresión permitida se entiende a aquella expresión matemática que ocupa las operaciones y funciones nombradas como existentes, y escritas de la forma que para ellas se señala, respetando así la sintaxis en cada instrucción.

El signo igual, " = ", de *a* = *b* tiene el sentido de asignar a la variable *a* el valor que resulte de evaluar la expresión permitida *b*. Es decir, *a* = *b* tiene el sentido:

a ← b

que no es la misma definición usada en matemáticas para el signo: = .

El computador, al ejecutar una proposición de asignación evalúa la expresión del lado derecho del signo igual con los valores que en ese momento tengan las variables ahí ocupadas. Así, la expresión entregará valores probablemente distintos en las diversas ocasiones que por programa se evalúe esa expresión.

2.2.1.4. Lectura. Para leer uno o varios datos e incorporarlos al programa, se usará la instrucción de lectura, representada por el símbolo de la lectora de tarjetas y adentro el nombre de las variables en que se almacenarán los datos leídos.

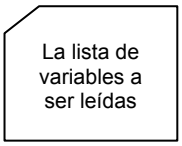

Figura 2. Bloque de lectura.

Ejemplos

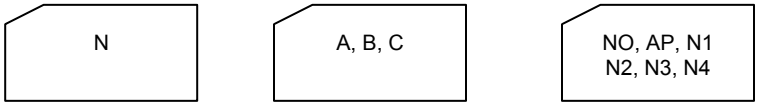

Figura 3. Ejemplos de bloques de lectura.

2.2.1.5. Impresión. Para imprimir valores calculados en el programa se usará la instrucción de impresión representada por el símbolo de la impresora, y adentro el nombre de las variables.

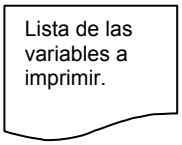

Figura 4. Bloque de impresión.

Ejemplos

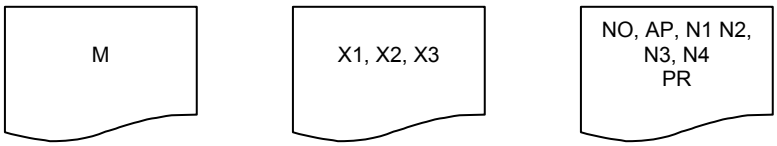

Figura 5. Ejemplos de bloques de impresión.

2.2.1.6. Símbolos usados en los diagramas de flujo. Los principales símbolos usados en los diagramas de flujo se muestran a continuación con lo que cada uno indica.

a) Ovalo de inicio y fin

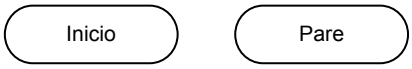

Figura 6. Ovalo de inicio y fin.

b) Flecha de dirección del flujo

Figura 7. Flechas de dirección.

c) Rectángulo o caja de operaciones

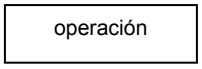

Figura 8. Operaciones.

d) Lectura de datos (símbolo de la lectora de tarjetas perforadas)

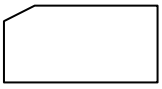

Figura 9. Lectura de datos.

e) Impresión (símbolo de la impresora de papel)

Figura 10. Impresión de datos.

f) Caja de decisiones (rombo)

Tendrán 2 salidas posibles, indicadas una por SI y la otra por NO.

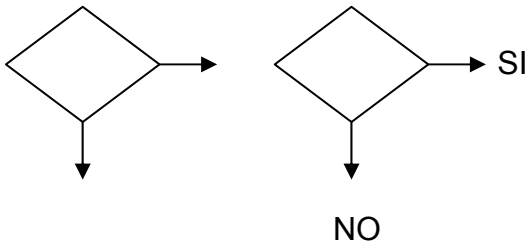

Figura 11. Caja de decisiones.

2.3. TEORÍA DE AUTOMATAS Y LENGUAJES FORMALES

La teoría de autómatas y lenguajes formales es una herramienta matemática que permite abordar con rigor el diseño de lenguajes de programación. Además desarrolla los conceptos necesarios para la construcción de autómatas para el reconocimiento de lenguajes de programación.

La teoría de los lenguajes formales tiene su origen aparentemente en un campo bastante alejado de la informática: la lingüística.

En el campo de la informática, el concepto de gramática formal adquirió gran importancia para la especificación de lenguajes de programación, concretamente se definió con sus teorías la sintaxis del lenguaje ALGOL 60,

usándose una gramática libre del contexto, lo que condujo rápidamente al diseño riguroso de algoritmos de traducción y compilación.

La teoría de lenguajes formales es de gran utilidad para el trabajo en otros campos de la informática, por ejemplo, en informática teórica, Inteligencia Artificial, procesamiento de lenguajes naturales (comprensión, generación y traducción) y reconocimiento del habla.

La teoría de autómatas proviene del campo de la Ingeniería Eléctrica, donde los autómatas son sistemas que reciben información, la transforman y producen otra información que se transmite al entorno.

La teoría de autómatas tiene su aplicación en campos muy diversos:

- Lógica de los circuitos secuénciales
- Teoría de control de sistemas
- Teoría de la comunicación
- Arquitectura de ordenadores
- Redes conmutadoras y codificadoras
- Teoría de los sistemas evolutivos y auto-reproductivos
- Reconocimiento de patrones
- Redes neuronales
- Reconocimiento y procesado de lenguajes de programación
- Traducciones de lenguajes
- Teoría de lenguajes formales.

2.3.1. Lenguajes, gramáticas y autómatas. Un lenguaje se puede definir, desde el punto de vista lingüístico, como un conjunto finito o infinito de oraciones, cada una de ellas de longitud finita y construidas por concatenación a partir de un número finito de elementos. Desde el punto de

vista informático, un lenguaje es una notación formal para describir algoritmos o funciones que serán ejecutadas por un ordenador. No obstante la primera definición, debida a Chomsky, es valida para cualquier lenguaje, sea natural o de programación.

El concepto de gramática fue acuñado por los lingüistas en sus estudios sobre lenguajes naturales. Los objetivos de una gramática son:

- Definir si una sentencia pertenece o no al lenguaje.
- Describir estructuralmente las sentencias.

En los lenguajes de programación los dos objetivos anteriores siguen vigentes.

Una gramática es un ente formal que especifica de una manera finita, un conjunto de sentencias, o cadenas de símbolos, potencialmente finitos (y que constituye un lenguaje).

Un lenguaje es un conjunto de cadenas de símbolos de un alfabeto determinado. Entendiéndose por símbolo una entidad abstracta. Alfabeto o vocabulario es un conjunto finito de símbolos. Cadena es una secuencia finita de símbolos de un determinado alfabeto.

Una gramática se puede definir formalmente como una cuadrupla formada por un vocabulario terminal VT, un vocabulario no terminal VN, un símbolo inicial S y un conjunto de producciones o reglas de derivación P.

$$G = (VT, VN, S, P)$$

Todas las sentencias del lenguaje definido por la gramática están formadas con símbolos del vocabulario terminal VT o símbolos terminales.

El vocabulario no terminal VN es un conjunto de símbolos introducidos como elementos auxiliares para la definición de las producciones de la gramática, y que no figuran en las sentencias del lenguaje. Evidentemente el conjunto de símbolos no terminales y el conjunto de símbolos terminales, no tienen ningún elemento en común.

El símbolo inicial S es un símbolo perteneciente al vocabulario no terminal, y a partir del cual pueden obtenerse todas las sentencias del lenguaje generado por la gramática.

Las producciones o reglas de derivación P son transformaciones de cadenas de símbolos, que se expresan mediante una pareja de expresiones separadas por una flecha. A la izquierda de la flecha, esta el símbolo o conjunto de símbolos a transformar y a la derecha se sitúan los símbolos resultado de la transformación.

Las reglas de derivación pueden aplicarse sucesivamente, desde el símbolo inicial hasta obtener una cadena de símbolos terminales, con lo que se obtiene una sentencia del lenguaje.

La definición formal de lenguaje, es el conjunto de todas las sentencias formadas por símbolos terminales que se pueden generar a partir de una gramática. Así si se denota L(G), al lenguaje generado por una gramática G, se puede expresar de la siguiente forma:

$$L(G)=\{\alpha Є VT/\ S \rightarrow \alpha\ \}$$

Donde α representa una cadena de símbolos terminales.

Entonces se dice que una sentencia pertenece a un lenguaje L(G) si:

- Esta compuesta de símbolos terminales.
- Puede derivarse del símbolo inicial S, por medio de las distintas producciones de la gramática G.

Se dice que dos gramáticas son equivalentes si generan el mismo lenguaje.

La tarea de comprobar si una sentencia o instrucción pertenece o no a un determinado lenguaje se encomienda a los autómatas. La palabra autómata evoca algo que pretende imitar las funciones propias de los seres vivos, especialmente relacionadas con el movimiento.

La información se codifica en cadenas de símbolos, y un autómata es un dispositivo que manipula esas cadenas que se le presentan a su entrada, produciendo otras cadenas a su salida. El autómata recibe los símbolos de entrada, uno detrás de otro, es decir secuencialmente. El símbolo de salida que en un instante determinado produce un autómata, no sólo depende del último símbolo recibido a la entrada sino de toda la secuencia o cadena de símbolos, recibida hasta ese instante. Lo anterior conduce a definir un concepto fundamental: estado de un autómata. El estado de un autómata es toda la información necesaria en un momento dado, para poder deducir, dado un símbolo de entrada en ese momento, cuál será el símbolo de salida.

El autómata tendrá un determinado número de estados (pudiendo ser infinito) y se encontrará en uno u otro según sea la historia de símbolos que le han llegado.

Se define configuración de un autómata a una situación en un instante. Si un autómata se encuentra en una configuración y recibe un símbolo, producirá un símbolo de salida y efectuará un cambio o transición a otra configuración.

Un autómata se puede definir como una quíntupla A = (E, Q, f, q_0, F), donde E es el conjunto de entradas o vocabulario de entradas, Q es el conjunto de estados, f : E x Q→Q que es la función de transición o función del estado siguiente, q_0 es el estado inicial, F es el estado final o de aceptación.

Ejemplo:
Sea la gramática G = { {S}, {a, b}, P ,S}, donde P son las reglas de producción:

S→ab
S→aSb

La única forma de generar sentencias con esta gramática es aplicando cualquier número de veces la segunda producción, y terminando con la aplicación de la primera.
S→ab
S→aSb →aabb
S→aSb→aaSbb →aaabbb
s→aSb→...→a^n b^n
El lenguaje generado es L(G) = { a^n b^n / n>=1}.

2.3.1.1. Léxico. El léxico de un lenguaje natural está constituido por todas las palabras y símbolos que lo componen. Para un lenguaje de programación u otro lenguaje usado en informática, la definición anterior también es válida.

En los lenguajes de programación el léxico lo constituyen todos los elementos individuales del lenguaje, denominados frecuentemente en inglés tokens. Así son tokens: las palabras reservadas del lenguaje, los símbolos que denotan los distintos tipos de operadores, identificadores(de variables, de funciones, de procedimientos, de tipos, etc..), separadores de sentencias y otros símbolos empleados en las distintas construcciones de un lenguaje.

2.3.1.2. Sintaxis. En lingüística, sintaxis es el estudio de la función que desempeña cada palabra en el entorno de una frase. Mientras que semántica es el estudio del significado de una palabra tanto a nivel individual como en el contexto de una frase.

En los lenguajes de programación, sintaxis es un conjunto de reglas formales que especifican la composición de los programas a base de letras, dígitos u otros caracteres.

2.3.1.3. Semántica. En lingüística es el estudio del significado de una palabra tanto a nivel individual como en el contexto de una frase.

En los lenguajes de programación es el conjunto de reglas que especifican el significado de cualquier sentencia, sintácticamente correcta y escrita en un determinado lenguaje.

2.3.2. Metalenguajes. Los lenguajes de programación son un conjunto finito o infinito de sentencias. Los lenguajes de programación con un número de sentencias finito se pueden especificar exhaustivamente enumerando todas sus sentencias. Sin embargo, para los lenguajes con un número de sentencias infinito esto no es posible, pues los medios que tenemos para describirlo son finitos. El medio habitual para describir un lenguaje es su gramática, pero las gramáticas que se utilizan en los lenguajes de programación necesitan una modelización matemática que haga factible su implementación en los compiladores.

Los metalenguajes son herramientas para la descripción formal de los lenguajes, facilitando no sólo la compresión del lenguaje, sino también el desarrollo del compilador correspondiente.

Las características semánticas de los lenguajes pueden describirse en lenguaje natural, pero éste es demasiado impreciso, por lo que se utilizan especificaciones formales.

Las gramáticas con atributos o atributadas son una extensión de la notación BNF que incluyen aspectos semánticos. Existen también otros metalenguajes orientados a la definición de las características semánticas, basados habitualmente en los lenguajes de especificación formal o herramientas de ingeniería del software.

2.3.2.1. Expresiones regulares. Es un metalenguaje para describir los elementos léxicos de un lenguaje o tokens. Las expresiones regulares utilizan cuatro tipos de operadores para definir los componentes léxicos, que se definen a continuación de menor a mayor precedencia:

- Paréntesis, para agrupar símbolos.
- Operaciones de cierre o cierre positivo. Se define la operación de cierre o cierre de estrella sobre una cadena α, es decir α*, como el conjunto de cadenas: λ, α, αα, ααα... α. Es decir representa: la cadena vacía λ, la propia cadena, una repetición, dos, etc. Un valor indefinido de repeticiones.

  El cierre positivo sobre una cadena α, es decir α^+, como el conjunto de cadenas α, αα, ααα... α. Es decir representa: la propia cadena, una repetición, dos, etc. Un valor indefinido de repeticiones pero sin incluir la cadena vacía.
- Operaciones de concatenación: se permite la concatenación de cadenas.
- Operación de alternativa: se representa por │, y permite la elección entre dos alternativas.

2.3.3. La notación BNF (Backus-Naur Form). Se debe a John Backus, esta notación se uso por primer vez en el lenguaje ALGOL .

La notación BNF utiliza los siguientes metasímbolos:

- < > encierra conceptos definidos o por definir. Se usa para los símbolos no terminales.
- :: = sirve para definir o indicar equivalencia.
- | separa las distintas alternativas
- " " indica que el metasímbolo que aparece entre comillas es un carácter que forma parte de la sintaxis del lenguaje
- ( ) Se permite el uso del paréntesis para hacer agrupaciones.

Existen símbolos con entidad propia llamados símbolos terminales, también existen otros que se deben definir y se denominan no terminales.

Ejemplo:

La definición de un identificador en BNF es la siguiente, aquí se permiten las definiciones recursivas

< ident > :: = <letra> | < ident > <letra> | < ident > <digito>

<letra> ::= a | b | c | ... | y | z

<digito> ::= 0 | 1 | 2 | ... | 8 | 9

En este caso a, b, c,...,y, z y 0,1,2,...,9 son los símbolos terminales, y el resto son los símbolos no terminales. El identificador se ha definido recursivamente.

2.3.4. Notación EBNF. La notación EBNF (Extend BNF), añade el metasímbolo { } a la notación BNF, para indicar que lo que aparece entre las llaves puede repetirse cero o mas veces. En algunos casos se indica con subíndices y superíndices el intervalo de repeticiones.

Ejemplo:

<identFORTRAN> ::= <letra>| {<alfanumérico>}$_0^5$

<alfanumérico> ::= <letra> | <digito>

<letra> ::= a | b | c | ... | y | z

<digito> ::= 0 | 1 | 2 | ... | 8 | 9

2.4 INGENIERÍA DEL SOFTWARE

Se encarga que la construcción de software cumpla con: obtención de requisitos software, diseño, implementación, pruebas, entrega, producción y mantenimiento.

2.4.1. Proceso de desarrollo de software. Los campos en que se utilizan las computadoras hoy en día son innumerables. Se ha llegado al punto en que es posible decir que no hay un solo campo del conocimiento humano en el que no se haya aplicado la computación, pues se han creado aplicaciones para áreas del conocimiento humano tan distintas como lo pueden ser la astrofísica y la cocina. Por esta razón el software que se puede crear es tan variado como las ideas que se forman en las mentes de las personas. Aún así, todos los proyectos de creación de software siguen un proceso muy parecido para llegar desde la idea original hasta un producto terminado y funcional. Este proceso básico está conformado por análisis, diseño, implementación, pruebas y mantenimiento.

Ahora bien, este ciclo de vida básico, como a veces se le llama, es tan sólo la idea general tras cualquier proceso particular de desarrollo de software. Existen diversos modelos, o métodos, para lograr cumplir con los pasos necesarios de este ciclo, cada uno con un enfoque distinto. Así pues, está el modelo de Construcción de Prototipos, el modelo de Desarrollo Rápido de Aplicaciones, el de Procesos Evolutivos - que a su vez se divide en los modelos incrementales, en espiral, de ensamblaje de componentes y de desarrollo concurrente - el modelo de Métodos Formales, y por último las Técnicas de Cuarta Generación.

Todos estos modelos atacan el problema desde un ángulo distinto, y cada uno tiene su forma particular de aplicación, pero a fin de cuentas, todos ellos ayudan a realizar la función para la que fueron diseñados: el desarrollo de software mediante técnicas de ingeniería.

2.4.2. Herramientas case y conceptos básicos. Para desarrollar software se necesita de otros tipos de software, es decir, para construir sistemas de cómputo se utilizan otros sistemas de cómputo.

Los medios sistematizados que se utilizaron por mucho tiempo estaban limitados a los tradicionales editores de texto para la codificación, y los compiladores del lenguaje respectivo. Fuera de éstos era poco el soporte que un programador o desarrollador de sistemas obtenía por parte de su ambiente de trabajo.

Debido a esta escasez de herramientas adecuadas para el desarrollo de sistemas surgió la lógica necesidad de crear sistemas que se pudieran utilizar verdaderamente como herramientas de soporte en la construcción de software. De ahí surge la Ingeniería de Software Asistida por Computadora, o en inglés, Computer-Aided Software Engineering (CASE). Así, una

herramienta CASE es un producto computacional enfocado a apoyar una o mas técnicas dentro del un método de desarrollo de software.

A pesar de que las herramientas CASE no tienen una historia extremadamente larga, pues empiezan a surgir a partir de principios de la década de los ochenta, ya se han extendido a la mayor parte de las fases y actividades involucradas en el desarrollo de software. Existen diversas taxonomías de las herramientas CASE, que utilizan varios criterios para su clasificación. Una clasificación por función se divide en dos grandes áreas: CASE superiores (U-CASE) y CASE inferiores (L-CASE). Los U-CASE abarcan las etapas de planeación, análisis y diseño, mientras que los L-CASE comprenden las de codificación, pruebas y mantenimiento. De esta manera se cubren las grandes áreas del desarrollo de software.

Las herramientas CASE individuales pueden estar enfocadas a un área de ingeniería de software más específica, como lo puede ser la ingeniería de información, el modelado de procesos, planificación y administración de proyectos, análisis de riesgos, seguimiento de requisitos, métricas, documentación, control de calidad, gestión de bases de datos, de desarrollo de interfaz o de generación de prototipos entre otros. El tipo específico de herramienta que se utilice depende de los requerimientos tanto del sistema a implementar como de los desarrolladores.

2.4.3. Expectativas del uso de un case. El propósito de una herramienta CASE es dar soporte automatizado para la aplicación de todas o algunas técnicas usadas por una o varias metodologías. Cualquier mejora orientada a resolver problemas asociados a la Crisis del Software se enmarca dentro de los siguientes niveles de solución, desde el más general al más particular:

1. Enfrentar el proceso de desarrollo de software como un proyecto de Ingeniería de Software.

2. Aplicar una o varios métodos de forma integrada cubriendo todas las actividades del ciclo de vida del software.

3. Usar una herramienta CASE para apoyar la aplicación de los métodos utilizados. Si consideramos que cada nivel debe implicar al anterior, se pone de manifiesto que la sola utilización de una herramienta CASE no garantiza una mejora en el proceso de desarrollo de software.

Por otra parte, las metodologías incluyen gran cantidad de técnicas, y el esfuerzo de documentación (y actualización de dicha documentación) es por lo general considerable. Por lo tanto, es difícil aplicar una metodología sin la ayuda de una herramienta CASE. Así, los beneficios de utilización de un CASE se entremezclan con los beneficios de aplicar una metodología con éxito. El valor agregado indudable de utilizar un CASE es el aumento en la productividad en las actividades soportadas por la herramienta.

Mientras los costos del hardware han ido en continuo descenso, sucede todo lo contrario con los costos del software. Las exigencias en complejidad y envergadura han sobrepasado a las mejoras en cuanto a métodos de desarrollo. El uso de metodologías de desarrollo junto a herramientas CASE no es una panacea, pero sin lugar a dudas ofrece la mejor alternativa actual para enfrentar proyectos de desarrollo de software de complejidad o envergadura.

El énfasis en planificación, análisis y diseño promovido por una herramienta CASE tiene un fuerte impacto y recompensa en la mejora de la calidad del

producto obtenido y en el aumento de productividad (disminución de tiempos, costes y esfuerzos) en las actividades de desarrollo y mantenimiento.

El beneficio adicional obtenido por la utilización de un CASE actual (si se compara con la utilización de una metodología sin el uso de un CASE) se representa en los siguientes aspectos:

- Facilita la verificación y mantenimiento de la consistencia de la información del proyecto.
- Facilita el establecimiento de estándares en el procesos de desarrollo y documentación.
- Facilita el mantenimiento del sistema y las actualizaciones de su documentación.
- Facilita la aplicación de las técnicas de una metodología.
- Disponibilidad de funciones automatizadas tales como: obtención de prototipos, generación de código, generación de pantallas e informes, generación de diseños físicos de bases de datos, verificadores automáticos de consistencia.
- Facilita la aplicación de técnicas de reutilización y reingeniería.
- Facilita la planificación y gestión del proyecto informático.

2.5. MODELADO ORIENTADO A OBJETO

El modelado, o modelo de objetos, describe los conceptos principales de la orientación a objetos: las estructuras estáticas y sus relaciones. Las principales estructuras estáticas son los objetos y clases, los cuales están compuestos de atributos y operaciones, mientras que las principales relaciones entre objetos y entre clases corresponden a las ligas y asociaciones, respectivamente. Estos temas y otros serán descritos en este

capítulo, en término de los objetos, clases, atributos, operaciones, asociaciones, composición, herencia y módulos.

2.5.1 Objetos. Los objetos son las entidades básicas del modelo de objeto. La palabra objeto proviene del latín objectus, donde ob significa hacia, y jacere significa arrojar; o sea que teóricamente un objeto es cualquier cosa que se pueda arrojar. Los objetos son más que simples cosas que se puedan arrojar, son conceptos pudiendo ser abstractos o concretos. Ejemplo: Una mesa es un objeto concreto, mientras que un viaje es un objeto abstracto.

Los objetos corresponden por lo general a sustantivos, pero no a gerundios, cualquier cosa que incorpore una estructura y un comportamiento se le puede considerar como un objeto, este debe tener una identidad coherente, al que se le puede asignar un nombre razonable y conciso. La existencia de un objeto depende del contexto del problema. Lo que puede ser un objeto apropiado en una aplicación puede no ser apropiado en otra, y al revés. Los objetos deben ser entidades que existen de forma independiente. Se debe distinguir entre los objetos, los cuales contienen características o propiedades, y las propias características. El objeto integra una estructura de datos (atributos) y un comportamiento (operaciones).

Los objetos se describen gráficamente por medio de un diagrama de objetos o diagrama de instancias. La notación general para un objeto es una caja rectangular conteniendo el nombre del objeto subrayado, el cual sirve para identificar al objeto, como se muestra en la Figura 12.

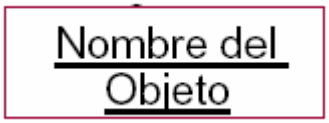

Figura 12. Diagrama de objeto.

2.5.2. Clases. Una clase describe un grupo de objetos con estructura y comportamiento común. (Clase y tipo no son necesariamente equivalentes, tipo se define por las manipulaciones que se le puede dar a un objeto dentro de un lenguaje y clase involucra una estructura, pudiendo corresponder a una implementación particular de un tipo.). Las estructuras o propiedades de la clase se conocen como atributos y el comportamiento como operaciones. Una clase define uno o más objetos, donde los objetos pertenecen a la clase, teniendo características comunes.

Ejemplo: Juan Pérez y María López se consideran miembros de la clase persona, donde todas las personas tienen una edad y un nombre. El ITAM y la UNAM pertenecen a la clase universidad, donde todas las universidades tienen una dirección y un grado máximo. Una clase se considera un "molde" del cual se crean múltiples objetos. Al definir múltiples objetos en clases se logra una abstracción del problema. Se generaliza de los casos específicos definiciones comunes, como nombres de la clase, atributos, y operaciones.

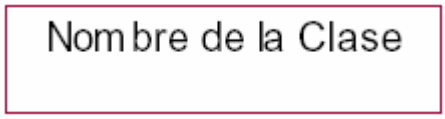

Figura 13. Diagrama de clase.

2.5.3. Instanciación. El proceso de crear objetos pertenecientes a una clase se conoce como instanciación, donde los objetos son las instancias de la clase. El objeto es la instancia de la clase a la que pertenece.

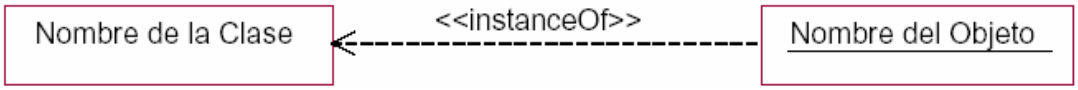

Figura 14. Notación para instanciación de objetos.

2.5.4. Atributos. Los atributos definen la estructura de una clase y de sus correspondientes objetos. El atributo define el valor de un dato para todos los objetos pertenecientes a una clase. Ejemplo: Nombre, edad, peso, son atributos de la clase persona. Color, precio, modelo, son atributos de la clase automóvil.

Los atributos corresponden a sustantivos y sus valores pueden ser sustantivos o adjetivos. Se debe definir un valor para cada atributo de una clase. Los valores pueden ser iguales o distintos en los diferentes objetos. No se puede dar un valor en un objeto si no existe un atributo correspondiente en la clase. Dentro de una clase, los nombres de los atributos deben ser únicos (aunque puede aparecer el mismo nombre de atributo en diferentes clases). Los atributos no tienen ninguna identidad, al contrario de los objetos.

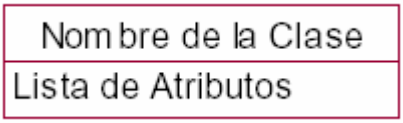

Figura 15. Diagrama de clase, conteniendo atributos

2.5.5. Operaciones. Las operaciones son funciones o transformaciones que se aplican a todos los objetos de una clase particular. La operación puede ser una acción ejecutada por el objeto o sobre el objeto, deben ser únicas dentro de una misma clase, aunque no necesariamente para diferentes clases. No se debe utilizar el mismo nombre para operaciones que tengan un significado totalmente diferente.

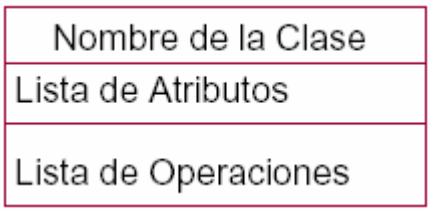

Figura 16. Diagrama de clase, conteniendo atributos y operaciones.

2.5.6. Generalización y herencia. Las clases con atributos y operaciones comunes se pueden organizar de forma jerárquica, mediante la herencia. La herencia es una abstracción importante para compartir similitudes entre clases, donde todos los atributos y operaciones comunes a varias clases se pueden compartir por medio de la superclase, una clase más general. Las clases más refinadas se conocen como las subclases. Ejemplo: Las Impresoras Láser, de Burbuja, y de Matriz, son todas subclases de la superclase Impresora. Los atributos generales de una Impresora son el Modelo, Velocidad, y Resolución, mientras que sus operaciones son Imprimir y Alimentar.

Herencia es una relación "es-una" entre las clases las más refinadas y más generales, es útil para el modelo conceptual al igual que para la implementación. Como modelo conceptual da buena estructuración a las clases. Como modelo de implementación es un buen vehículo para no replicar innecesariamente el código.

Generalización define una relación entre una clase más generalizada, y una o más versiones refinadas de ella. Ejemplo: La clase Impresora es una generalización de las clases Impresoras Láser, de Burbuja, y de Matriz.

Especialización define una relación entre una clase más general, y una o más versiones especializadas de ella. Ejemplo: Impresoras Láser, de Burbuja, y de Matriz, son todas especializaciones de Impresoras.

La superclase generaliza a sus subclases, y las subclases especializan a la superclase. El proceso de especialización es el inverso de generalización. Una instancia de una subclase, o sea un objeto, es también una instancia de su superclase. Ejemplo: Cuando se crea un objeto de tipo Impresora Láser, este objeto incluye toda la información descrita en la subclase Impresora

Láser, al igual que en la superclase Impresora; por lo tanto se considera que el objeto es una instancia de ambas.

La herencia es transitiva a través de un número arbitrario de niveles. Los ancestros de una clase son las superclases de una clase en cualquier nivel superior de la jerarquía, y los descendientes de una clase son las subclases de una clase en cualquier nivel inferior de la jerarquía. Ejemplo: Si además de Impresora de Burbuja, se define una clase más especializada como Impresora de Burbuja Portátil, entonces Impresora e Impresora de Burbuja son ancestros de la clase Impresora de Burbuja Portátil, mientras que Impresora de Burbuja e Impresora de Burbuja Portátil son descendientes de Impresora.

Las siguientes características se aplican a clases en una jerarquía de herencia:

- Los valores de una instancia incluye valores para cada atributo de cada clase ancestra.
- Cualquier operación de cualquier clase ancestra, se puede aplicar a una instancia.
- Cada subclase no solo hereda todas las características de sus ancestros sino también añade sus propios atributos y operaciones.

La generalización se puede extender a múltiples niveles de jerarquías, donde una clase hereda de su superclase, que a su vez hereda de otra superclase, hacia arriba en la jerarquía. En otras palabras, las relaciones entre subclases y superclases son relativas. La herencia define una jerarquía de clases donde existen ancestros y descendientes, que pueden ser directos o no.

Para representar herencia y generalización, se utiliza un triángulo conectando a la superclase con sus subclases. La superclase está del lado

superior del vértice del triángulo, mientras que las subclases están en la parte inferior de la base del triángulo.

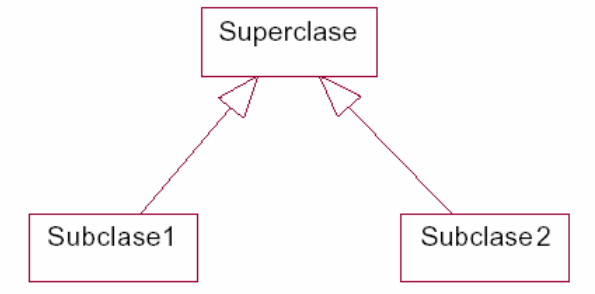

Figura 17. Diagrama de herencia y generalización.

2.5.7. Ligas y asociación. La relación entre objetos se conoce como liga. Una asociación describe la relación entre clases de objetos, y describe posibles ligas, donde una liga es una instancia de una asociación, al igual que un objeto es una instancia de una clase.

Tipos de asociaciones entre clases:

- Una asociación de conocimiento es una asociación estática entre instancias y significa que una instancia conoce de la existencia de otra instancia. Denota conocimiento entre clases durante largos periodos.
- Una asociación de comunicación es una asociación dinámica que modela la comunicación entre dos objetos, sirviendo para intercambiar información entre objetos. Denota relación entre clases cuando existe una comunicación de ellos a través de estas asociaciones, un objeto envía y recibe eventos.

El nombre de una liga debe ser igual al nombre de la correspondiente asociación. La asociación, al igual que la liga, es por naturaleza bidireccional, por lo general, el nombre de la liga o asociación implica una dirección, pero puede ser invertida para mostrar la dirección opuesta. Cualquiera de las dos

direcciones es igualmente correcta, aunque por lo general se acostumbra a leer de izquierda a derecha.

La notación describiendo una liga es una línea conectando los dos objetos, conteniendo el nombre de la liga.

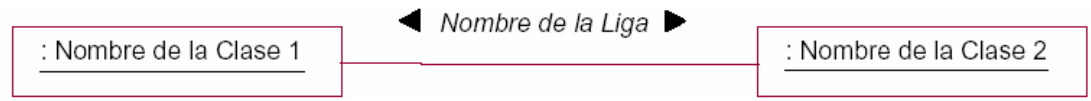

Figura 18. Notación para diagramas de objetos contenidos en una liga.

2.5.8 Lenguaje Unificado de Modelado. Para el análisis y diseño de DFC 1.0 se trabaja con UML ya que permite expresar un modelo de análisis utilizando una notación de modelado con reglas sintácticas, semánticas y prácticas de tal manera que pueden mostrar y combinar símbolos conociendo con anterioridad su significado e interpretación y comparándolas con el lenguaje natural para poder ser comprensible a otras personas.

UML representa un sistema mediante cinco vistas diferentes representadas mediante un conjunto de diagramas; estas vistas son: la del usuario con los diagramas de casos de uso, la vista estructural mostrada con clases, objetos y relaciones, la vista de comportamiento mediante diagramas de estados y actividad, la vista de implementación con diagramas de componentes y despliegue y la vista de entorno.

Durante el modelo de análisis se representan las vistas del modelo de usuario y estructural, proporcionando una visión interna al uso de las situaciones para el sistema. UML se organiza en dos actividades mayores: el diseño del sistema y el diseño de objetos representando la arquitectura del software y extendiéndose para considerar el diseño de interfaces,

administración de datos con el sistema que se va a construir y administración de tareas para los subsistemas que se hayan especificado.

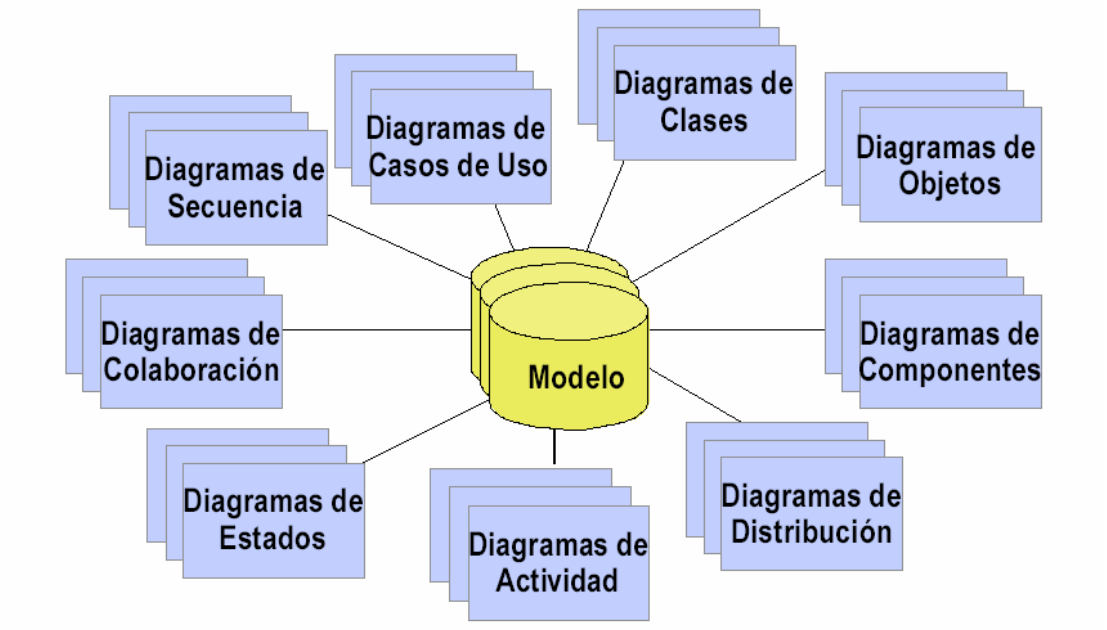

Figura 19. Diagramas de UML.

3. DESARROLLO DEL PROYECTO

3.1. METODOLOGÍA

DFC 1.0 abarca los tres conceptos básicos de la programación. Es por esto que se escogió la Metodología de Entrega por Etapas, en la cual no se entrega el producto total al final del proyecto sino que se muestra al usuario en etapas refinadas sucesivamente, proporcionando una funcionalidad útil antes de entregar el cien por ciento del proyecto.

Primero se realiza la definición de concepto del software, el análisis de requerimientos y la creación del diseño global de la arquitectura del modelo en cascada. A continuación se procede a realizar el diseño detallado, la codificación, depuración y prueba dentro de cada etapa.

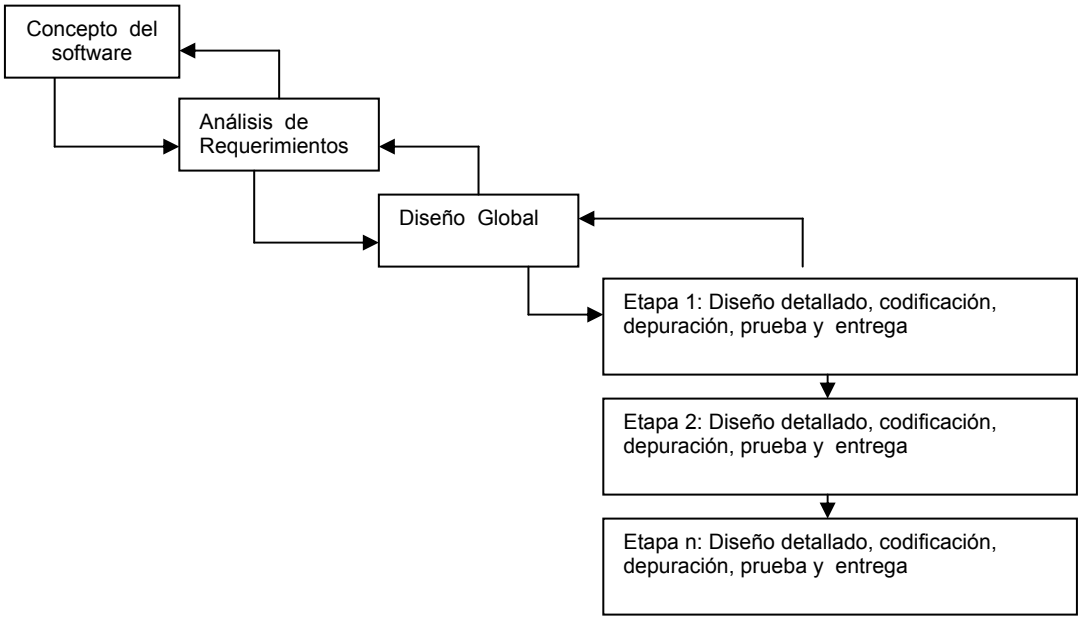

Figura 20. Modelo de entrega por etapas.

Con esta metodología se desarrollan las capacidades más importantes reduciendo el tiempo necesario para la construcción de un producto; entre sus beneficios tenemos:

- Se detectan los problemas antes y no hasta la única entrega final del proyecto.
- Se elimina el tiempo en informes debido a que cada versión es un avance.
- Se hacen varias estimaciones de tiempo por versión, evitando errores en la estimación del proyecto general.
- Cumplimiento a la fecha por los desarrolladores.

En el modelo de Entrega por Etapas se recomienda su uso para problemas que se conocen bien y que pueden ser tratados descomponiéndolos en problemas más pequeños. En el desarrollo de DFC 1.0, se pueden notar claramente las siguientes etapas:

- Construcción de los diagramas por parte de la herramienta.
- Validación de los diagramas y edición de los mismos.
- Ejecución de los diagramas.
- Almacenamiento de la información.
- Generación de código fuente.

La entrega por etapas funciona en el proyecto porque se desarrollan independientemente subconjuntos útiles del producto; y además el producto para el usuario final se puede definir de tal forma que se pueden hacer entregas intermedias significativas antes de entregar el producto final. Es importante analizar que esta metodología propicia un intercambio de conocimientos y de autocrítica al sistema, lo que conlleva a que se

produzcan muchas pruebas antes de liberar una nueva versión, así como mejoras rápidas a problemas que puedan surgir durante su uso.

3.2. FASES DEL PROYECTO

3.2.1. Fase 1: Concepto del Software. Se parte de todas las circunstancias que involucran el proyecto, lo cual implica un análisis de todo el marco teórico y la forma como cada uno de los temas analizados, influyen y edifican la herramienta.

Para un correcto uso de la herramienta, el usuario debe tener claro los conceptos básicos en el área de la programación, medianamente debe distinguir un algoritmo, de un diagrama de flujo y de la codificación en un determinado lenguaje, sin embargo la herramienta permite que con un constante uso, el usuario pueda distinguir si su percepción mental de un determinado problema al ser plasmado en un diagrama de flujo, lleva a la solución que el esperaba; si esto no se presenta, puede empezar después de un análisis, a realizar cambios, conllevando a un proceso de retroalimentación y aprendizaje gradual.

DFC 1.0 utiliza las técnicas del modelado orientado a objeto, garantizando una eficiente utilización de los recursos y permitiendo una expansión para futuras mejoras a la herramienta; se apoya en la teoría de autómatas y lenguajes formales para el análisis sintáctico y semántico de las expresiones creadas por el usuario y usadas en diferentes partes del diagrama; también usa los conceptos de la programación dinámica haciéndolo efectivo en el uso de la memoria y plantea una interfaz agradable y entendible para sus usuarios.

Desde este punto de vista, se pretende mirar el alcance de la herramienta analizado las funcionalidades que el usuario puede obtener de la misma.

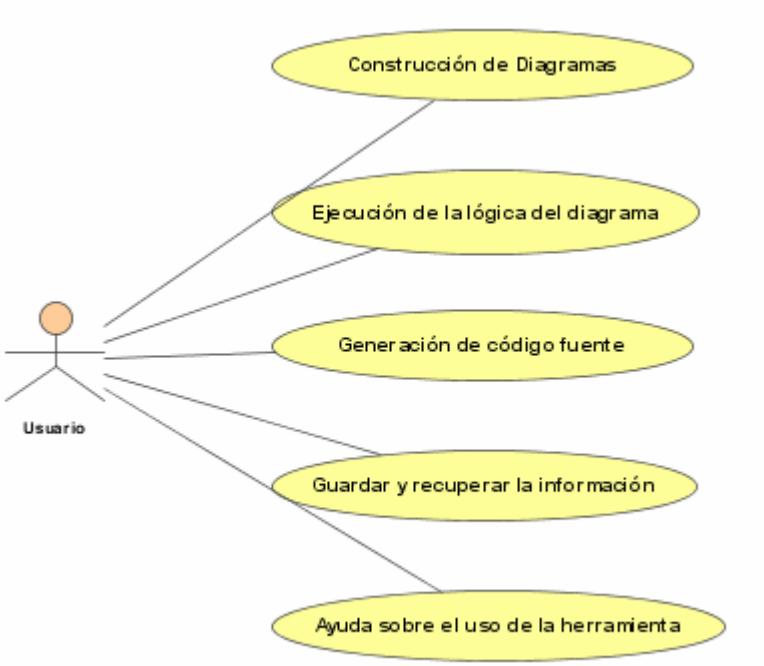

Figura 21. Diagrama caso de uso. Alcance del proyecto.

3.2.2. Fase 2: Análisis de Requerimientos. Esta fase según la metodología trabajada consta principalmente de dos partes fundamentales: la valoración de la bibliografía y el análisis de requisitos. Nos centraremos en la ultima sin desconocer la primera, la cual ya fue analizada en la parte introductoria.

3.2.2.1. Análisis de Requisitos. La especificación de los requisitos del software se produce en la culminación de la tarea de análisis y se deben tener en cuenta ciertos aspectos importantes y que marcarán la pauta durante todo el desarrollo del proyecto. Los requisitos se representan de manera que como fin último lleven al éxito de la implementación del

software, es por esto muy importante tener en cuenta los siguientes principios:[1]

- Separar la funcionalidad de la implementación.
- Desarrollar un modelo del comportamiento deseado de un sistema que comprenda datos y las respuestas funcionales de un sistema a varios estímulos del entorno.
- Establecer el contexto en que opera el software especificando la manera en que otros componentes del sistema interactúan entre con él.
- Definir el entorno en que va a operar el sistema.
- Crear un modelo intuitivo en vez de un diseño o modelo de implementación.
- Reconocer que la especificación debe ser tolerante a un posible crecimiento si no es completa.
- Establecer el contenido y la estructura de una especificación de manera que acepte cambios.

De esta forma siguiendo el caso de uso de la Figura 21, los requisitos giran en torno a cinco aspectos fundamentales

3.2.2.1.1. Construcción de diagramas de flujo. DFC 1.0 permite la creación de diagramas de flujo con la siguiente notación y que en la actualidad es la más usada.

1 **Bloque de inicio y fin**. Se representan con dos círculos rotulados cada uno de ellos según su funcionalidad al interior del programa.

[1] PRESMAN, Roger S. Ingeniería del Software - Un enfoque práctico. Quinta Edición. Editorial McGraw Hill. Madrid. 2002.

Figura 22. Bloque Inicio Figura 23. Bloque Fin

2 **Bloque de Asignación.** Está representado con un rectángulo y el cual puede contener hasta cuatro asignaciones en un mismo bloque. Las asignaciones pueden estar expresadas como variables que aceptan valores constantes numéricos (a=5) o una expresión numérica compuesta (a=c*3+d/5).

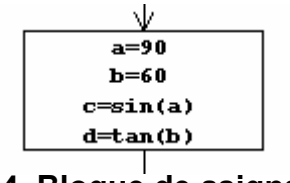

Figura 24. Bloque de asignación

3 **Bloque de escritura.** De la misma forma en que el bloque de asignación puede operar con valores constantes o con expresiones compuestas, el bloque de escritura puede mostrar estos formatos o una mezcla de números y caracteres, usando el operador de concatenación **'&'.**

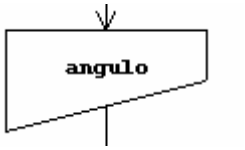

Figura 25. Bloque de escritura

4 **Bloque de lectura.** Este bloque puede contener una o más variables, separadas por comas, las cuales se pueden alimentar una por una. Las variables deben contener valores numéricos constantes.

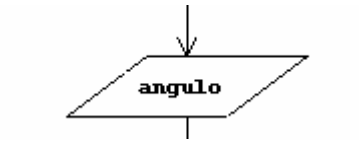

Figura 26. Bloque de lectura

5 **Bloques de decisión.** El bloque de decisión más usado es el if, el cual se usa con operandos y operadores. Los operandos pueden ser cualquier variable o valor sobre los cuales se quiere representar una relación.

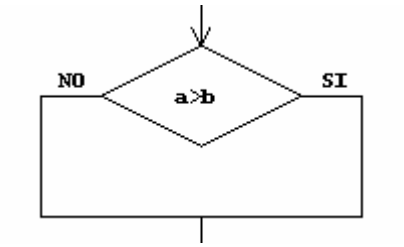

Figura 27. Bloque de Decisión

6 **Bloques de repetición.** Son los encargados de realizar ciclos al interior de un algoritmo; los podemos tener de tres formas: *ciclo para*, *ciclo mientras que,* y el *ciclo hacer mientras*. El *ciclo para* se compone de tres condiciones: un inicio, una salida y un incremento de una determinada variable en mención; el *ciclo mientras que* se compone de una expresión al igual que los bloques de decisión; el *ciclo hacer mientras* se compone de una condición al final de este.

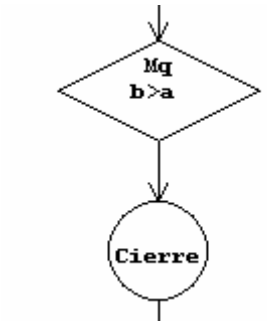

Figura 28. Bloque mientras que

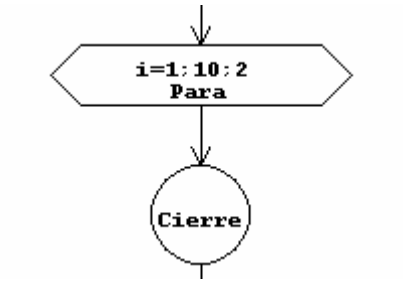

Figura 29. Bloque para

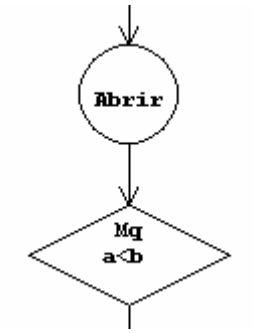

Figura 30. Bloque hacer mientras

7 **Bloques de cierre.** Cada uno de los ciclos mencionados (excepto el ciclo *hacer mientras*) lleva su bloque de cierre, el cual le indica hasta donde es su campo de acción.

Figura 31. Bloque de cierre

8 **Bloque abrir.** Es utilizado para el ciclo *hacer mientras*, el cual indica desde donde se inicia su campo de acción.

Figura 32. Bloque abrir

9 **Bloque de subllamada.** Indica que un procedimiento ha sido invocado; los parámetros que se pasen al procedimiento, siempre serán por valor, es decir no serán alterados por el subprograma, si se quiere que estos sean alterados, es decir paso por referencia se debe anteponer un * al nombre de la variable.

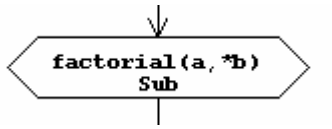

Figura 33. Bloque de subllamada

Los diagramas de flujo, además de su nomenclatura necesitan saber con que tipo de datos operan, los operadores lógicos, matemáticos y las funciones matemáticas y de cadenas con las cuales trabajan.

3.2.2.1.1.1. Tipos de datos. Como se mencionó, se manejan dos tipos de datos: el numérico y el no numérico. Dentro del formato numérico tenemos los arreglos que podemos manejar y que comúnmente son conocidos como matrices, podemos tener matrices de hasta dos dimensiones y se puede acceder usando los corchetes (matriz[i][j]). El máximo valor que se puede almacenar en formato numérico es de: **1.7E +/- 308 (15 dígitos)**. El formato no numérico incluye todas las cadenas de caracteres alfabéticas y alfanuméricas. Para representar las cadenas de caracteres podemos usar las comillas dobles **“ ”**. El formato no numérico solo es permitido en el bloque mostrar.

3.2.2.1.1.2. Operadores. Los operadores los podemos dividir en lógicos relacionales y matemáticos.

1 **Lógicos relacionales.**

- >. Mayor que
- <. Menor que
- >=. Mayor o igual que
- <=. Menor o igual que
- ==. Igual que
- !=. Diferente de
- && Operación and lógica
- ||. Operación or lógica

- !|. Operación negación or lógica
- NOT. Negación lógica
- !&. Negación del and.

2 **Matemáticos.**

- +. Adición
- -. Sustracción
- *. Multiplicación
- /. División
- %. Modulo de la división
- ^. Exponenciación
- =. Asignación

3.2.2.1.1.3. Funciones. La manipulación de datos ocurre prácticamente en el formato numérico, sin embargo ambos formatos son usados en el objeto mostrar, por lo tanto se cuenta con la siguiente gama de funciones:

- INC (X). Incrementa la variable X en 1, afectando su valor
- DEC (X). Decrementa la variable X en 1, afectando su valor
- RANDOM (X). Devuelve un numérico randómico entre 0 y X
- ROUND (X). Redondea al entero mas cercano de X
- TRUNC (X). Extrae la parte entera de X
- ABS (X). Devuelve el valor absoluto de X
- LN (X). Devuelve el logaritmo natural de X
- EXP (X). Devuelve e ^ X. Donde e = 2.718281828 aprox.
- SQRT (X). Devuelve la raíz cuadrada de X
- SIN (X). Devuelve el seno de X en radianes
- COS (X). Devuelve el coseno de X en radianes
- TAN (X). Devuelve la tangente de X en radianes

- ASIN (X). Devuelve el seno inverso de X
- ACOS (X). Devuelve el coseno inverso de X
- ATAN (X). Devuelve la tangente inversa de X

3.2.2.1.2. Ejecución de la lógica del diagrama. Una vez construidos los diagramas, el usuario puede probar, si el algoritmo implementado responde a las necesidades de su problemática. Para esto cuenta con una serie de opciones, de la siguiente manera:

- Ejecutar. Ejecuta la lógica del diagrama hasta el final del mismo.

- Depuración. Permite analizar la lógica de manera detallada, según las opciones definidas:
    - Paso a paso. Permite avanzar por cada elemento del diagrama
    - Ejecutar hasta. Permite ejecutar la lógica del diagrama hasta un punto estipulado.
    - Pausar. Permite hacer una pausa a medida que se ejecuta la lógica del diagrama.
    - Detener. Esta opción facilita la paralización de la lógica del diagrama en un punto dado.

- Evaluar. Facilita observar el valor de determinadas variables del diagrama.

3.2.2.1.3. Generación de código fuente. DFC 1.0 genera el código fuente del diagrama desarrollado en lenguaje C++, este código puede ser mostrado junto con el diagrama y también puede ser llevado a disco. De igual forma puede generar un fragmento del código correspondiente a un bloque del

diagrama. Es un código cien por ciento compilable y no genera ningún tipo de warning o error.

3.2.2.1.4. Guardar y recuperar la información. Para garantizar una integridad de la información DFC 1.0, permite que el diagrama pueda ser guardado y recuperado de disco, brindado así la facilidad de seguir modificando y adaptando un determinado algoritmo a las circunstancias del medio; para esto contamos con las siguientes opciones:

- Guardar como. Da la facilidad de guardar el trabajo con cualquier nombre y automáticamente se colocará con extensión dfc.
- Guardar. Actualiza un trabajo ya guardado con los últimos cambios realizados.
- Abrir. Colocará un trabajo guardado, listo para ser manipulado.
- Nuevo. Brinda la opción de diseñar otro diagrama.
- Imprimir. Permite que el diagrama con el cual se esta trabajando actualmente pueda ser impreso.
- Salir. Permite abandonar la aplicación.
- Edición. Ofrece todas las posibilidades de manipular el diagrama:
    - Cortar
    - Copiar
    - Pegar
    - Eliminar
    - Rehacer
    - Deshacer.

3.2.2.1.5. Ayuda sobre el uso de la herramienta. Esta opción brinda la facilidad de tener un formato de ayuda sobre DFC 1.0, de igual forma esta opción contiene información sobre la licencia de uso y sobre los créditos del programa.

3.2.3. Fase 3: Diseño Global. El diseño global está divido en análisis y diseño, para esta parte usamos la metodología orientada a objetos, para conservar coherencia entre el análisis, diseño e implementación.

3.2.3.1. Análisis orientado a objetos. El propósito del AOO es definir todas las clases que son relevantes al problema que se va a resolver, las operaciones y atributos asociados, las relaciones y comportamientos asociados con ellas. Para cumplirlo se deben ajustar las siguientes tareas:[2]

1 Los requisitos básicos del usuario deben comunicarse entre el cliente y el ingeniero del software.
2 Identificar las clases (es decir, definir atributos y métodos).
3 Se debe especificar una jerarquía de clases.
4 Representar las relaciones objeto a objeto.
5 Modelar el comportamiento del objeto.
6 Repetir iterativamente las tareas de la 1 a la 5 hasta completar el modelo.

El objetivo del análisis orientado a objeto es desarrollar una serie de modelos que describen el software de computadora al trabajar para satisfacer un conjunto de requisitos definidos por el cliente.

De esta forma se desarrollan con mas detalle cada parte del caso de uso de la Figura 21 y se hace un bosquejo de todas las acciones que debe desarrollar DFC 1.0 usando un diagrama de secuencia según la notación de UML[3].

[2] PRESMAN, Roger S. Ingeniería del Software - Un enfoque práctico. Quinta Edición. Editorial McGraw Hill. Madrid. 2002.

[3] BOOCH, Grady. RUMBAUGH, James. JACOBSON Ivar. El Lenguaje Unificado de Modelado. Primera Edición. Editorial Addison Wesley Iberoamericana. Madrid. 1999.

Secuencia general.

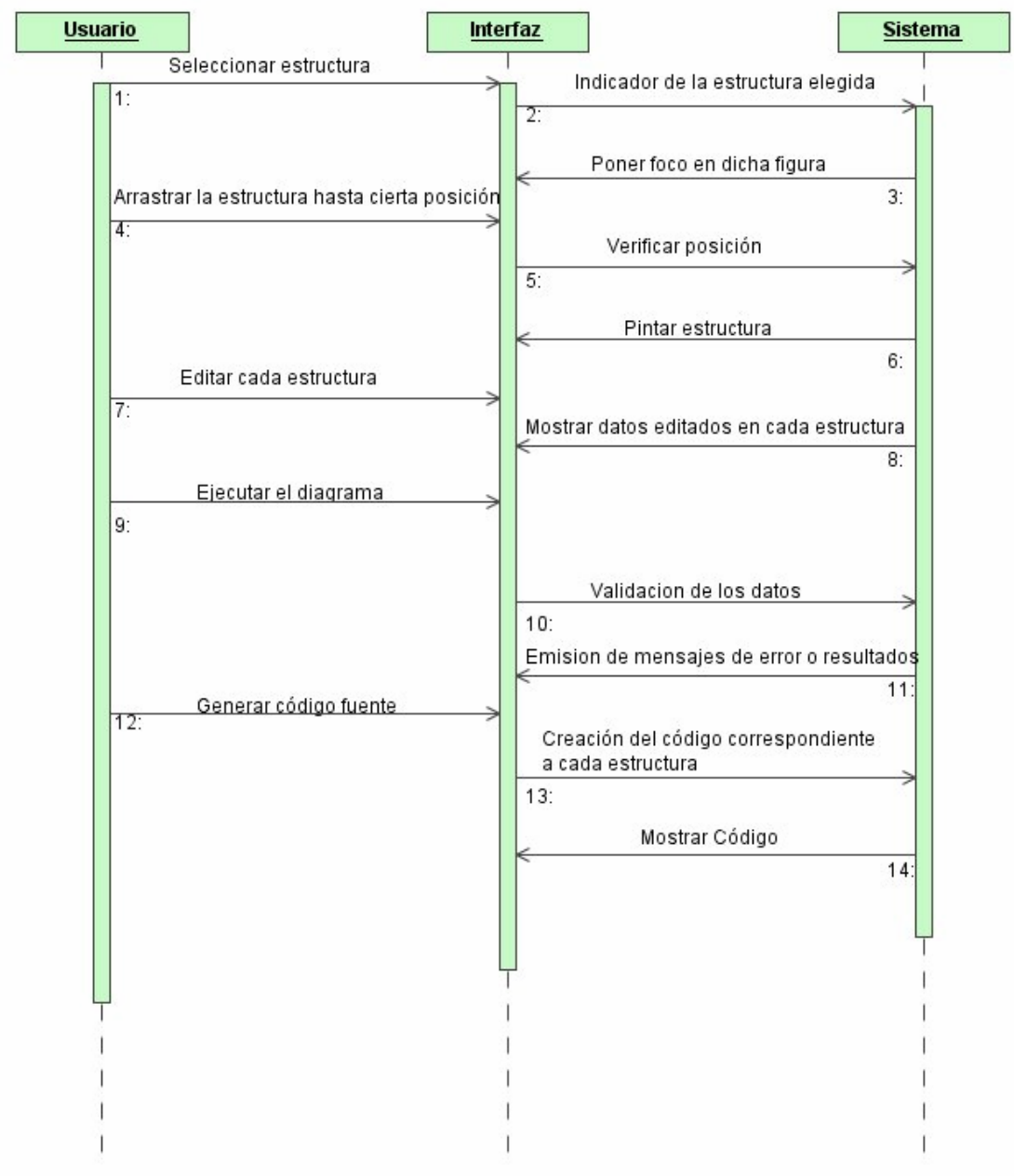

Figura 34. Diagrama de secuencia. Secuencia general.

Se representa el conjunto de acciones que tiene que realizar un determinado usuario para explotar la funcionalidad de la herramienta, aquí se nota que se incluyen las operaciones básicas, y se excluye la ayuda, por no considerarla un requisito funcional, le damos un enfoque no funcional.

Construcción de Diagramas.

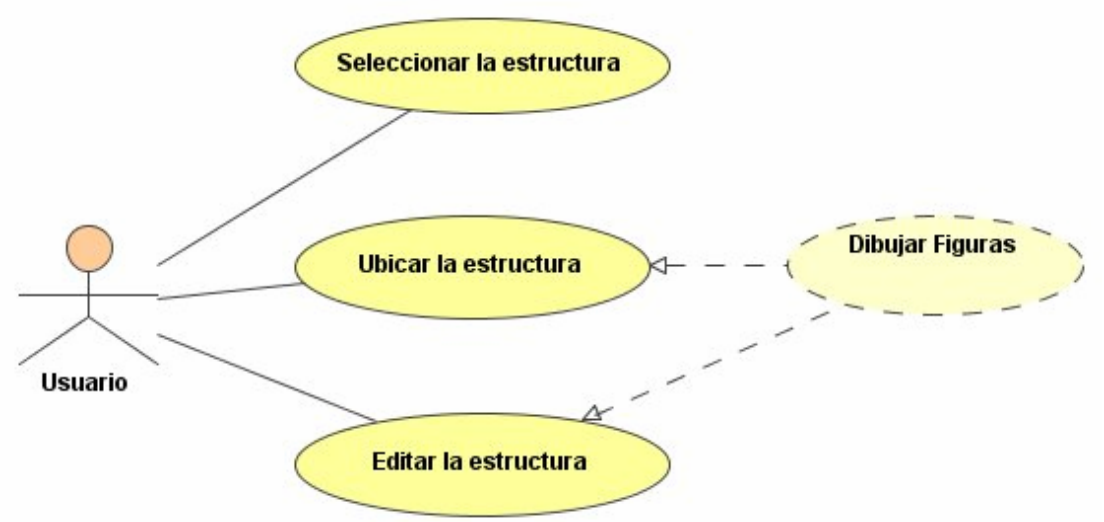

Figura 35 Diagrama caso de uso. Construcción de diagramas.

La construcción de los diagramas se lleva a cabo mediante tres operaciones básicas, las cuales están identificadas en el programa: seleccionar, arrastrar y editar; además de esto se trabaja con la colaboración de dibujar figura, la cual es realizada por la aplicación (llamada sistema en el diagrama de secuencia) y entra a actuar cada vez que una estructura es manipulada.

Ejecutar Diagramas.

Para ejecutar la lógica de un diagrama, esta se puede desarrollar de manera completa o se puede optar por pausarla o detenerla, tal cual como es indicado en el caso de uso. De la misma manera se puede entrar a depurar su lógica, esta se logra hacer paso a paso o realizarla hasta un determinado

punto, se ve como se puede apoyar de “include” y “extend”, para realizar actividades opcionales o que aumentan las cualidades. En la depuración paso a paso o realizarla hasta un punto de parada, se pueden mirar los valores de las variables en cualquier momento.

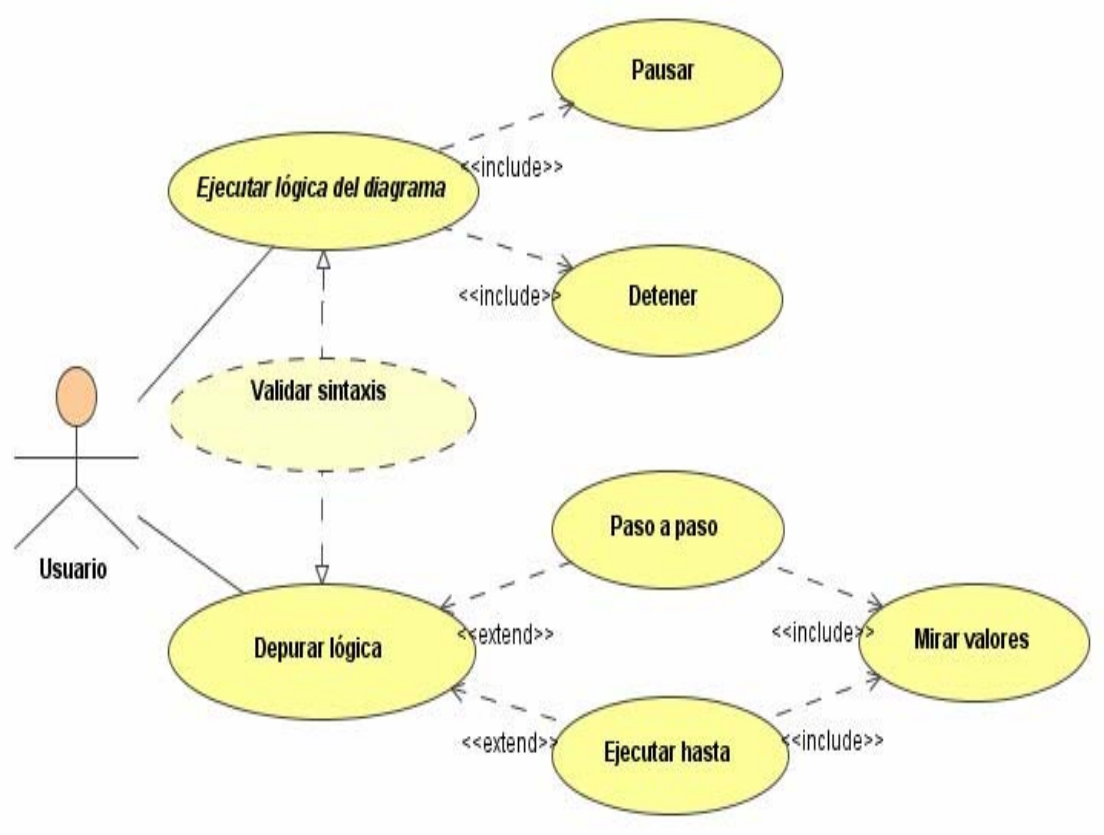

Figura 36 Diagrama caso de uso. Ejecución de la lógica del diagrama.

Generar código fuente

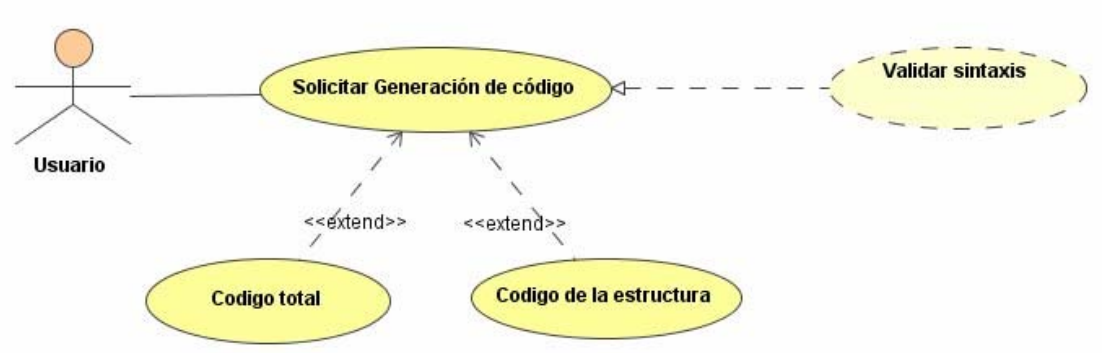

Figura 37 Diagrama caso de uso. Generar código fuente.

En la generación del código, este puede adquirir dos comportamientos: código total o código de una estructura determinada que forme parte del diagrama, en este caso se recurre a la colaboracion de validar sintaxis, la cual verifica que cada una de las estructuras del diagrama sean correctas.

Edición del diagrama

Figura 38 Diagrama caso de uso. Edición de diagramas.

La edición se considera como uno de los procesos fundamentales en la herramienta, pues mediante ella, podemos perfeccionar nuestro diagrama y probarlo con los diferentes escenarios que podamos imaginar. La parte de rehacer y deshacer almacena los cambios realizados, esto permite que si el usuario en cualquier momento considera que no ha hecho la elección más correcta puede recuperar diagramas anteriores.

Visualización de la ayuda.

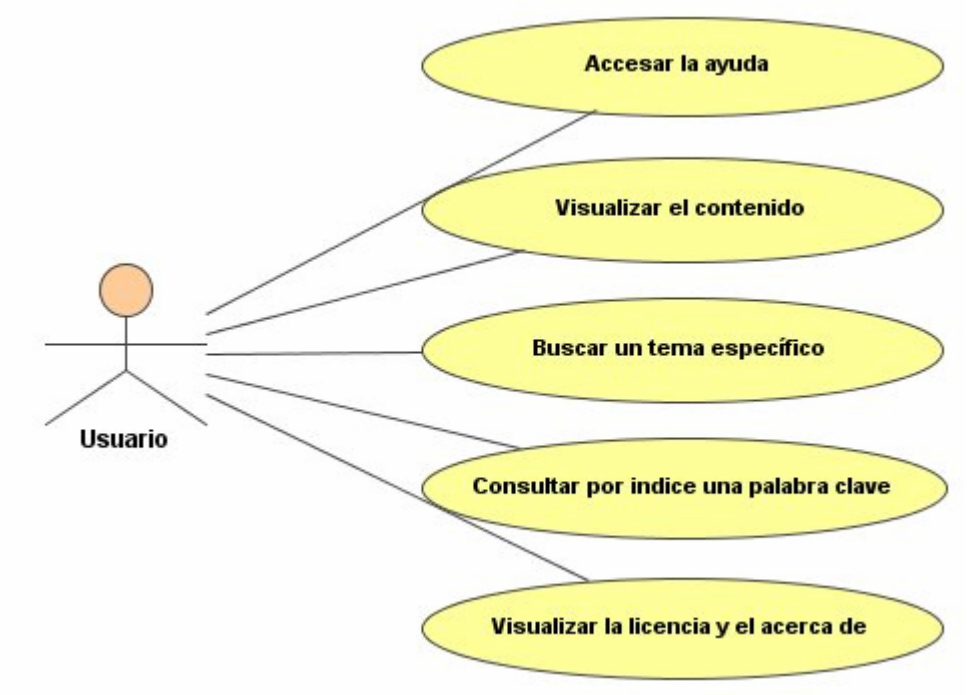

Figura 39 Diagrama caso de uso. Visualización de la ayuda.

La ayuda no la consideramos como un requisito funcional, pero si de vital importancia para explotar al máximo todas y cada una de las características que nos ofrece DFC 1.0.

3.2.3.1.1. Jerarquía de clases. Se muestran las clases que existen en la herramienta y las relaciones entre ellas.

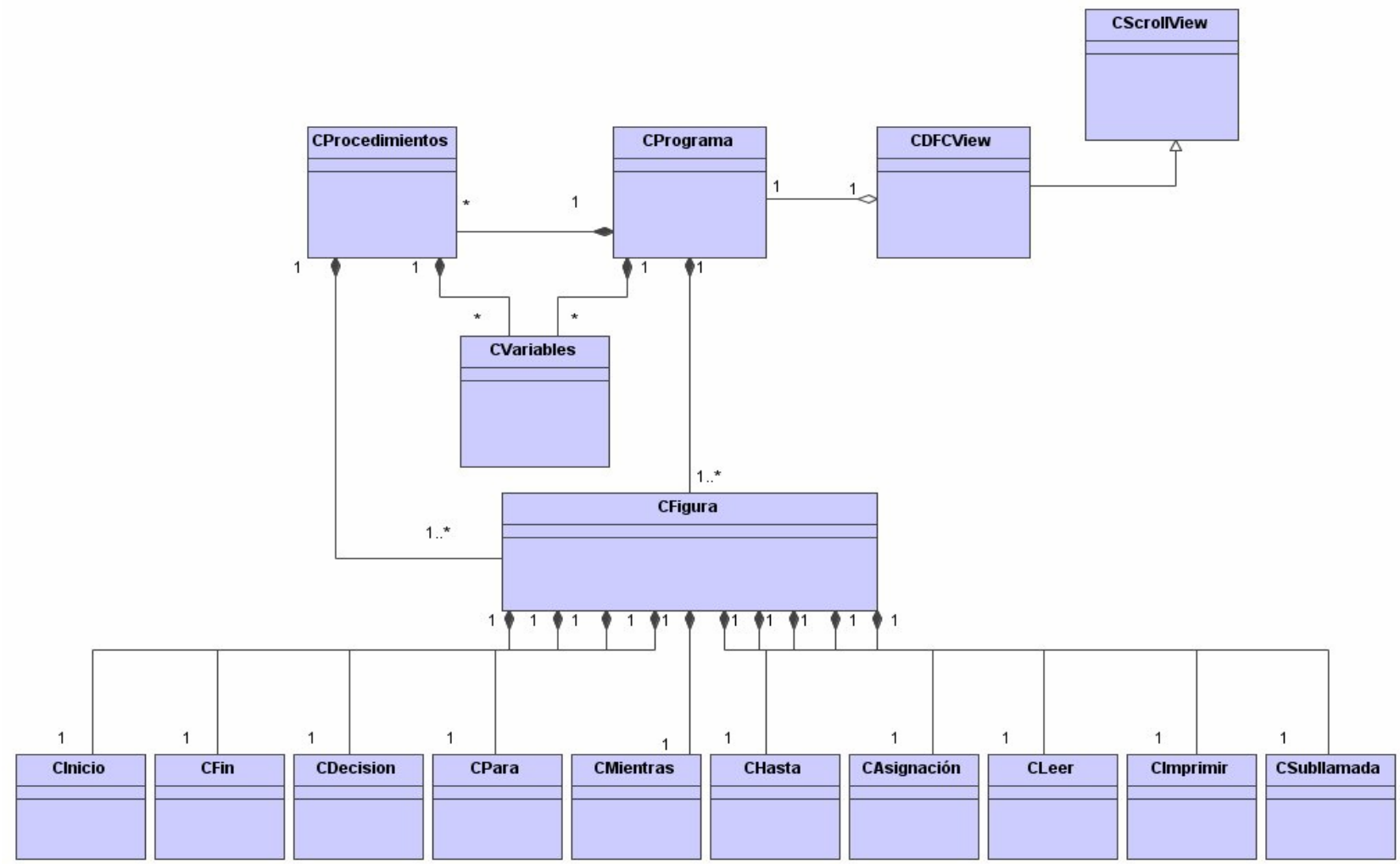

Figura 40 Diagrama de clases. Jerarquía de clases

Después de analizar todos los casos de uso que representan las características de la herramienta, se condensa en un diagrama de clases todas las funcionalidades mostradas.

Cabe resaltar que usando la metodología orienta a objetos, un problema se puede representar de diferentes maneras y cada forma de resolverlo se mueve en un contexto determinado, de tal forma se presenta un análisis de la estrategia empleada para obtener el diagrama de la figura 40.

Se identifican diez figuras básicas (Inicio, Fin, Decisión, Para, Mientras, Hacer mientras, Asignación, Subllamada, Leer e Imprimir; las cuales formaran cada una, una clase diferente) que se condensan utilizando composición en una clase denominada CFigura. Una agregación de composición es aquella que posee a sus partes y entraña una fuerte dependencia de propiedad. Las partes viven en el todo, de modo que se destruyen cuando se destruye el todo. La multiplicidad del lado del todo debe ser cero o uno, pero la multiplicidad en el lado parte puede ser un intervalo.[4]

De esta forma la clase CFigura posee una instancia de cada figura en concreto y en cualquier momento puede usarla. La clase CFigura se convierte en un nodo de una lista enlazada que se encuentra en la clase CPrograma. Se usan las listan enlazadas, porque cada nodo se comporta como una de las diez figuras básicas, además no sabemos de cuantas figuras se compone cada programa y de ahí la utilización de listas enlazadas. CPrograma es la clase principal que contiene la lista enlazada con las figuras y otra lista enlazada con todos lo procedimientos que el usuario implemente en su diagrama, de igual manera contiene en forma de lista, todas las variables a las cuales se hizo referencia.

[4] JOYANES, Luis. Programación Orientada a Objetos. Segunda Edición. Editorial Mc Graw Hill. Madrid. 1998.

La clase CProcedimiento es en realidad un nuevo diagrama y por lo tanto tiene todas las características en cuanto a instancias de CPrograma, vemos en ella el uso de la lista enlazada para CFigura y para CVariables, esta clase es la encargada de todo el manejo en su parte grafica y de variables de los subprogramas.

En los casos de uso analizados anteriormente nos encontramos con relaciones de colaboración, include y extends que no se evidencian en el diagrama de clases, pero si son desarrolladas como métodos de las clases a las cual afectan.

Se podría pensar que la clase CFiguras bien podía tener una relación de generalización con las diez clases que la componen, la herencia se usa cuando una clase es un tipo de otra y la composición cuando una clase tiene otras clases.[5] En este caso es muy difícil encontrar características comunes entre cada figura, pues obedecen a diferentes formas y a diferentes atributos, aunque pueden responder a iguales métodos, en resumen no cumple a cabalidad con los requisitos para implementar generalización, sin embargo se puede utilizar una clase abstracta y tratar de implementar los métodos virtuales semejantes, pero en realidad se llegó a una mayor simplicidad usando composición, para lograr transparencia al manejar las clases.

3.2.3.2. Diseño orientado a objetos. El diseño orientado a objetos requiere la definición de una arquitectura multicapa, la especificación de subsistemas que realizan funciones necesarias y proveen soporte de infraestructura, una descripción de clases, que son los bloques de construcción del sistema, y una descripción de los mecanismos de comunicación, que permiten que los

[5] JOYANES, Luis. Programación Orientada a Objetos. Segunda Edición. Editorial Mc Graw Hill. Madrid. 1998.

datos fluyan entre las capas subsistemas y objetos.[6] De esta forma se analizan los atributos y métodos mas importantes de cada clase.

Hay que recordar que se usan dos clases que no son implementadas como parte del análisis de la herramienta, sino que hacen parte del lenguaje de desarrollo, como lo son CSrollView que es una clase que nos permite interactuar con las barras de desplazamientos; y nuestra vista de la aplicación CDFCView.

La clase CDFCView se compone de dos atributos fundamentales como lo son: diagramaPpal, el cual es una instancia de CPrograma y es el encargado del manejo de todo el diagrama de flujo; y para manipular un determinado objeto, se tiene la variable objectoActivo, la cual guarda el código de la figura que se está manejando actualmente.

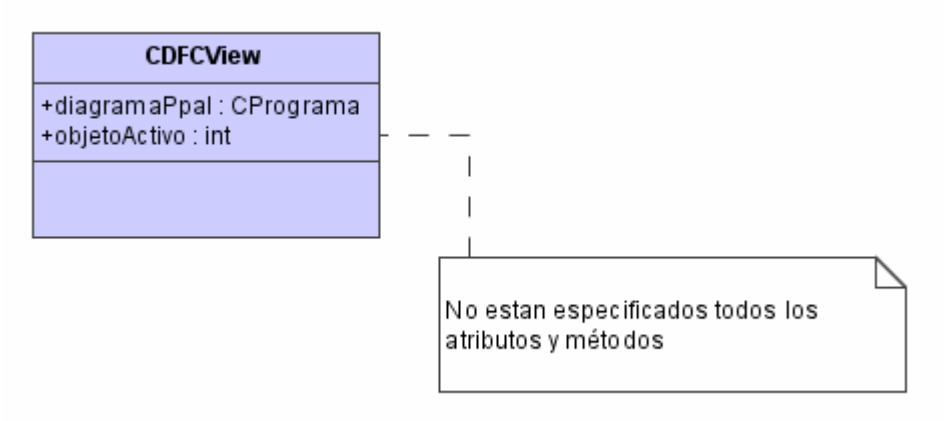

Figura 41. Diagrama de clase. Clase CDFCView.

Con la clase CPrograma se tienen tres variables muy importantes: figurasP, la cual es la encargada de mantener información sobre todas las figuras, es una lista simplemente enlazada y su tipado obedece a una estructura

[6] PRESMAN, Roger S. Ingeniería del Software - Un enfoque práctico. Quinta Edición. Editorial McGraw Hill. Madrid. 2002.

denominada punteroF, de la misma forma son listas enlazadas procedimientosP y variablesP; las cuales manejan toda la estructura de los procedimientos que posee el diagrama y las variables que pertenecen al mismo, respectivamente.

Así mismo, posee cuatro métodos: agregarFigura el cual añade los nodos necesarios para guardar información de la figura, agregarProcedimiento el cual agrega los nodos necesarios para manejar los procedimientos, agregarVariable que hace la misma acción para guardar las variables, dibujarFigura que se encarga del manejo gráfico y un constructor para inicializar variables y dar valores iniciales.

En esta clase se manejan tres tipados que obedecen a estructuras y un tipo enumerado, las cuales son especificadas de la siguiente forma:

```
enum{CURSOR, ASIGNACION, LECTURA, IMPRESION, DECISION, HASTA, PARA, SUBLLAMADA}objetoActivo;
typedef struct estructuraV{
        CVariable variable;
        estructuraV *sig;
}*punteroV;
typedef struct estructuraP{
        CProcedimiento procedimiento;
        estructuraP *sig,*ant;
}*punteroP;
typedef struct estructuraF{
        CFigura figura;
        estructuraF *sig, *ant;
}*punteroF;
```

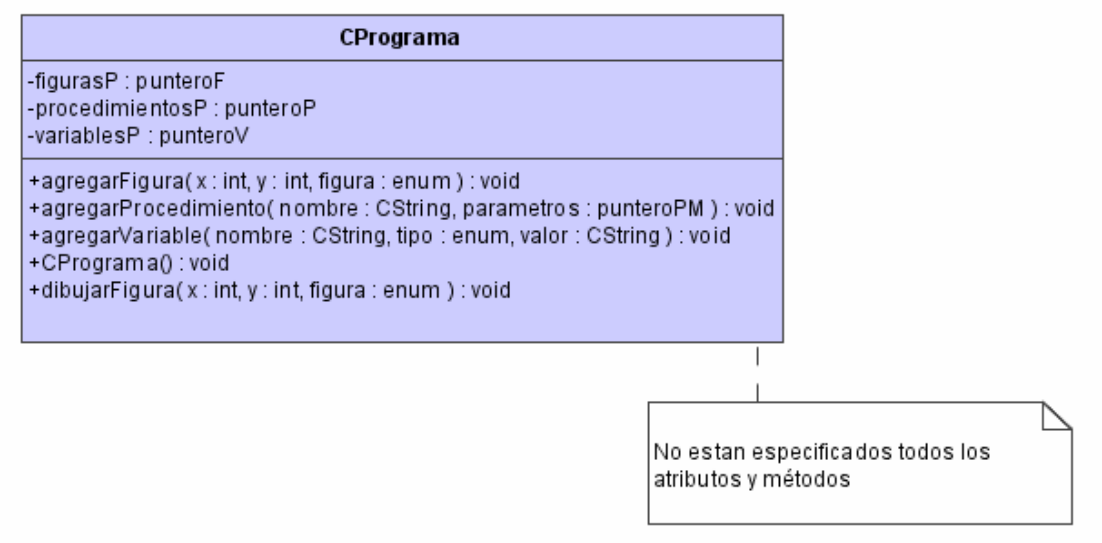

Figura 42. Diagrama de clase. Clase CPrograma.

La clase CVariables ofrece todo el manejo necesario para manipular las variables de la aplicación; se compone de tres atributos: el nombre de la variable, su tipo que es una enumeración y su valor. Sus operaciones permiten manipular la información de sus atributos.

Especificación de la enumeración: enum{NO_ NUMERICO, NUMERICO }tipoDato;

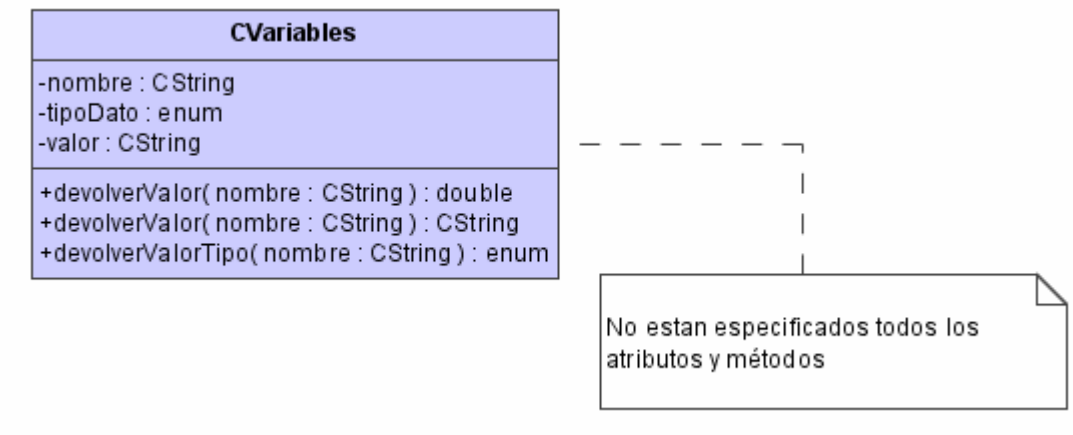

Figura 43. Diagrama de clase. Clase CVariables.

La clase CProcedimientos, posee una estructura muy similar a CPrograma, pues los procedimientos son pequeños programas que ejecutan una labor definida; de esta manera se poseen prácticamente las mismas variables y cabe resaltar que aunque en el diagrama de clases general, se puede observar que de las mismas clases, CProcedimiento y CPrograma son partes CVariables y CFiguras, pero nunca la misma instancia se encuentra en ambas clases. Se puede pensar en una generalización pero sería algo cíclico, pues CProcedimiento sería clase generalizada y a su vez parte de CPrograma.

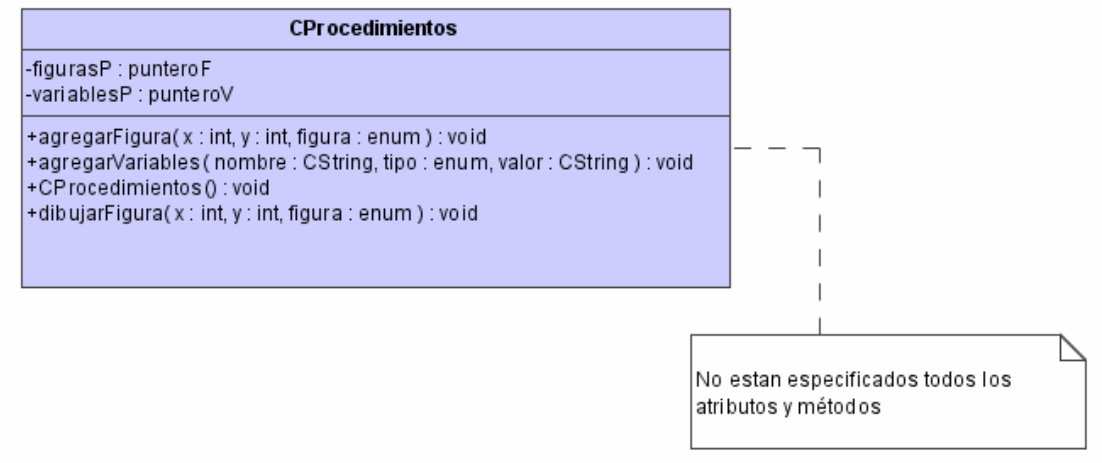

Figura 44. Diagrama de clase. Clase CVariables.

Las clases CInicio y CFin son muy parecidas en cuanto a su funcionamiento y pintan una elipse según las coordenadas dadas, las cuales tienen la leyenda "**Inicio**" y "**Fin**", respectivamente.

Se manejan dos operaciones fundamentales, como lo son dibujarFigura y calcularValores, esta última se encarga de dar valores a todas las variables necesarias para pintar la estructura en la vista; dibujarFigura es común en todas las clases que representan figuras, pero su forma de implementación es muy distinta y varía mucho de una clase a otra.

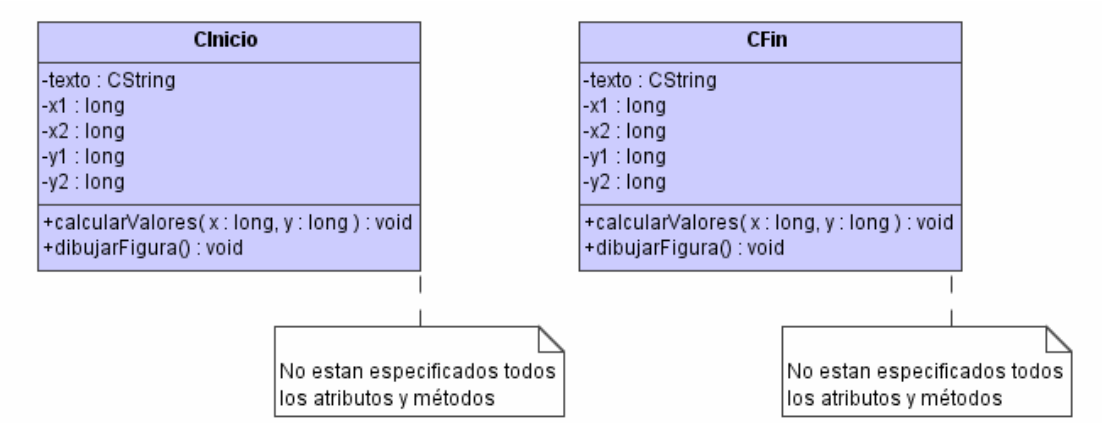

Figura 45. Diagrama de clase. Clases CInicio y CFin.

La clase CDecision es la encargada de simular la operación **if,** en realidad es un tanto complejo su gráfico y por esa forma solo fueron condensados los atributos y operaciones mas importantes. Aquí tomamos como marco de referencia los valores x, y en los cuales el usuario ha hecho click, a partir de esos valores se obtienen todos los valores necesarios para pintar correctamente la figura.

Cabe destacar, que Decisión es la figura que causa mayor complejidad en su tratamiento, pues mientras las otras crecen hacia abajo, esta lo puede hacer hacia abajo y hacia los lados, por lo tanto le es dado un trato especial.

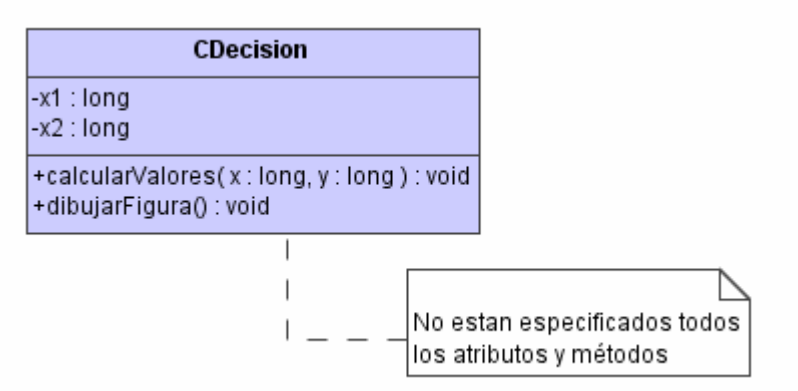

Figura 46. Diagrama de clase. Clase CDecision.

La clase CPara se encarga de la operación **for** y activa el uso de la lista enlazada de variables.

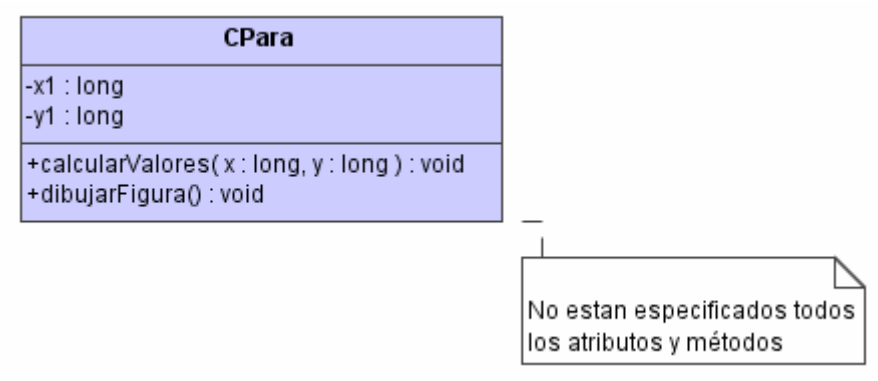

Figura 47. Diagrama de clase. Clase CPara.

Las clases CMientras y CHasta son las encargadas de realizar los ciclos **while** y **do while,** al igual que todas las clases que se han usado, con las variables x1, y1, se simula y se recogen todos los demás valores para pintar la figura.

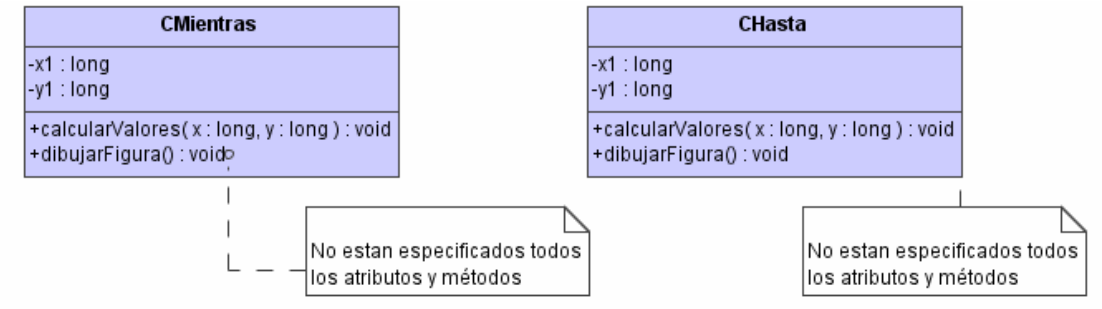

Figura 48. Diagrama de clase. Clases CMientras y CHasta.

La clase CAsignacion es por naturaleza la que mas trabaja con la CVariables, pues en esencia es junto con CLectura las que mueven las variables.

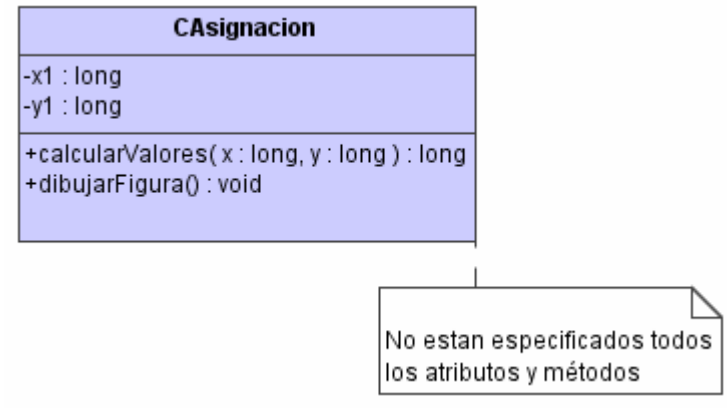

Figura 49. Diagrama de clase. Clase CAsignacion.

La clases CLectura y CImprimir, se encargan del contacto con el mundo exterior.

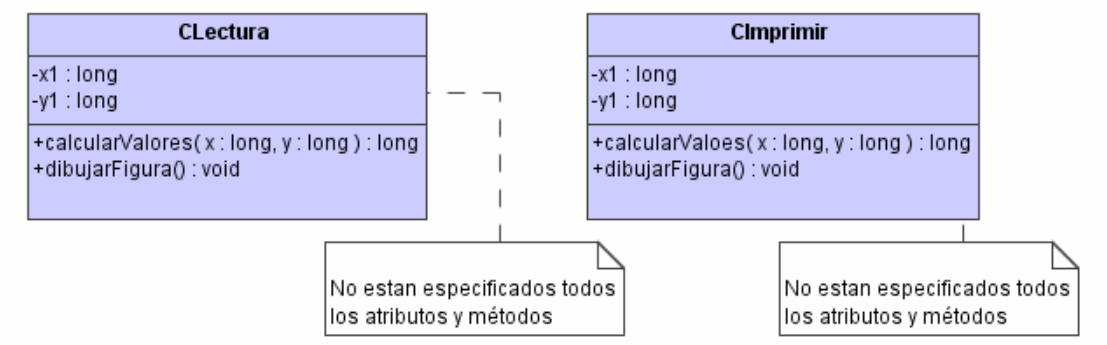

Figura 50. Diagrama de clase. Clases CLectura y CImprimir.

La clase CSubllamada es la encargada de los procedimientos.

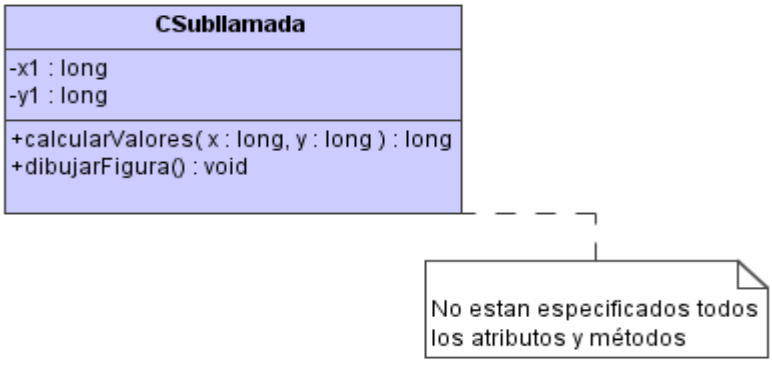

Figura 51. Diagrama de clase. Clase CSubllamada.

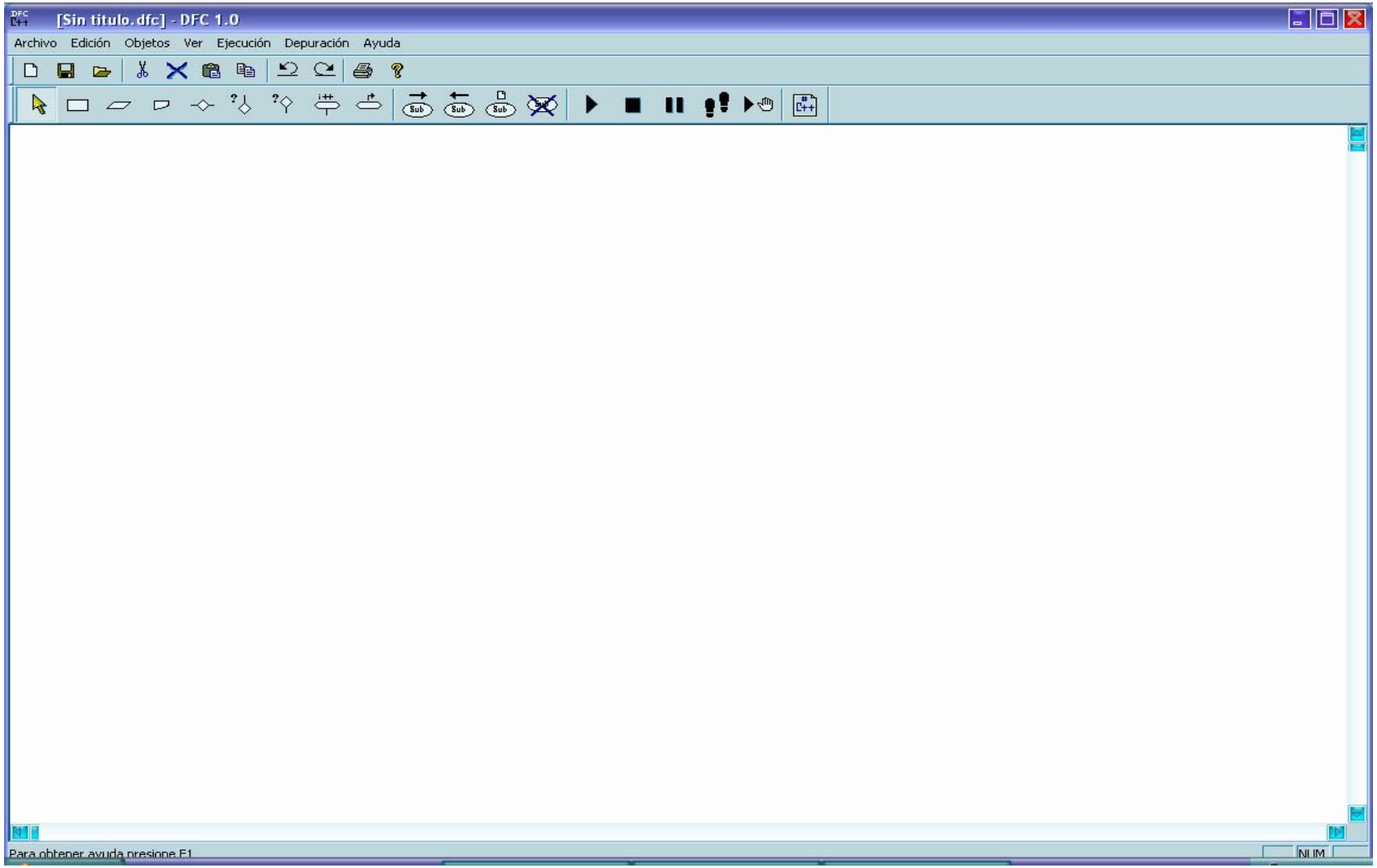

Figura 52. Interfaz de DFC 1.0.

Se presenta un pantallazo sobre el diseño global de la interfaz de la aplicación y resta expresar que solo se hizo un enfoque muy global sobre los atributos y operaciones de cada clase, pero jamás el estudio fue descontextualizado y poco evaluador de la situación.

3.2.3.3. Etapa 1: Construcción de los diagramas por la herramienta. Para la realización de DFC 1.0 se utilizó el lenguaje de programación Visual C++ 6.0 de Microsoft Corporation y para la elaboración de los distintos diagramas de UML se usó MagicDraw UML 7.1 de Sun MicroSystems Inc.

Esta etapa abarca la construcción de diagramas usando todas las figuras previstas y la construcción de subprogramas asociados a los diagramas. La manipulación permitida hasta el momento solo acepta inserción de figuras e insertar, eliminar y visualizar subprogramas.

La jerarquía de clases trabajada usa las clases que hacen composición con CFiguras, explotando la característica de la POO como es el paso de mensajes para lograr una comunicación mas efectiva y segura entre las diferentes instancias.

Cada figura usada representa una clase que posee sus atributos y operaciones que permiten la manipulación al interior del diagrama, como ejemplo veamos la definición de la clase CLectura:

```
class CLeer
{
private:
        char texto[80];
        long x1,y1,x2,y2,xT,yT,x3,y3,x4,y4;
        long xM,yM,xMF,yMF;
        int ancho,largoFlecha;
```

```
public:
        CLeer();
        virtual ~CLeer();
        //Se encarga de llenar todas las variables de la clase
        void calcularValores(long x, long y);
        void dibujarFigura(CDC *pDC);
        void escribirTexto(CDC *pDC);
        RECT getZonaActiva();
        POINT getPuntoFinal();
        POINT getValoresXY();
};
```

En sus atributos se pueden destacar la variable texto, usada para recoger y mostrar la leyenda que se quiere colocar en el diagrama y que posteriormente será usada en la ejecución del diagrama; las variables de tipo long son usadas para manipular gráficamente la figura, las cuales son calculadas a partir de dos valores iniciales pasados a la operación calcularValores la cual los toma de la última coordenada arrojada por la figura anterior con ayuda de la operación getPuntoFinal. La operación getZonaActiva devuelve un rectángulo que nos indica, cuando la actual figura puede aceptar una figura después de ella, para así devolver su valor final.

Cada figura maneja en cierta forma los mismos atributos y operaciones para la manipulación de sus datos. Ahora bien la clase CFigura es la encargada de poner a trabajar a cada figura; posee entre sus atributos:

```
void agregarFigura(long x,long y, short figura);
void dibujarFigura(short figura,CDC *pDC);
```

De esta forma se observa cómo se le debe indicar qué figura se quiere manipular, y así hacer las respectivas llamadas según sea la clase correspondiente.

La clase CPrograma es quizás la más importante en todo este proceso. Esta formada por listas doblemente enlazadas que permiten la manipulación de

las figuras y de los distintos subprogramas, por lo tanto hay que tener especial cuidado en el uso de la memoria, pues es la fuente principal de las listas dinámicas.

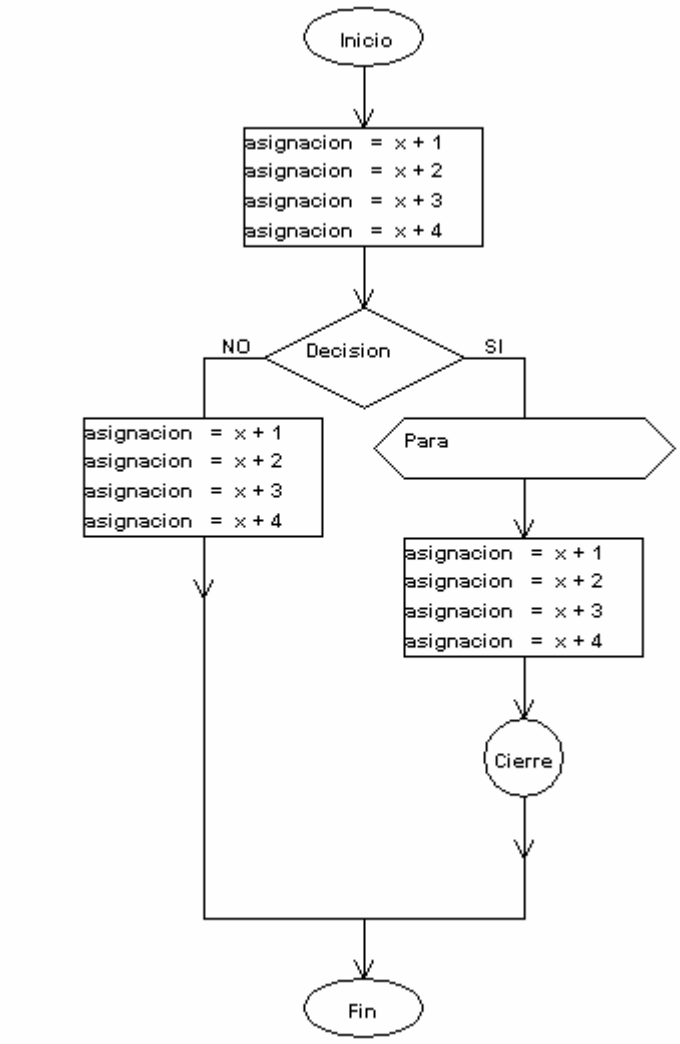

Figura 53. Diagrama generado por DFC 1.0.

La clase CDFCView se encarga de manipular todos los dispositivos para el pintado de las figuras y por lo tanto posee una instancia de la clase CPrograma.

3.2.3.3.1. Pintar figuras. Las figuras posibles para pintar son manipulables con esta barra:

Figura 54. Barra de figuras.

Tenemos de izquierda a derecha: cursor, asignación, lectura, impresión, decisión, hacer mientras, mientras que, para, subprograma o subllamada, siguiente subprograma, anterior subprograma, nuevo subprograma y eliminar subprograma.

Para pintar figuras el usuario hace click en la vista soportada por la clase CView de Visual C++, de esta manera CDFCView en su operación:
void CDFCView::OnLButtonDown(UINT nFlags, CPoint point), mediante su mensaje WM_LBUTTONDOWN, que manipula el click derecho del mouse cuando el botón se encuentra abajo; de esta forma podemos adquirir las coordenadas mediante la variable point, que pertenece a la clase CPoint de la MFC (Microsoft Foundation Class Library – biblioteca de clases base de Microsoft). Con estas variables llamamos a la operación agregarFigura la cual debe verificar que en las coordenadas actuales se puede ubicar una figura, esto se logra analizando si esas variables pertenecen a la zona activa de alguna figura, es decir si existe una figura que pueda recoger en su final a esta figura; si el permiso es autorizado se sigue con el paso de mensajes o sencillamente el usuario debe retomar su actividad.

Si el permiso se acepta se le comunica a CPrograma, para que pueda ubicar los correspondientes nodos que aceptaran la nueva figura y puede inicializar sus valores, CPrograma debe entonces invocar a CFigura para que esta última manipule la figura determinada.

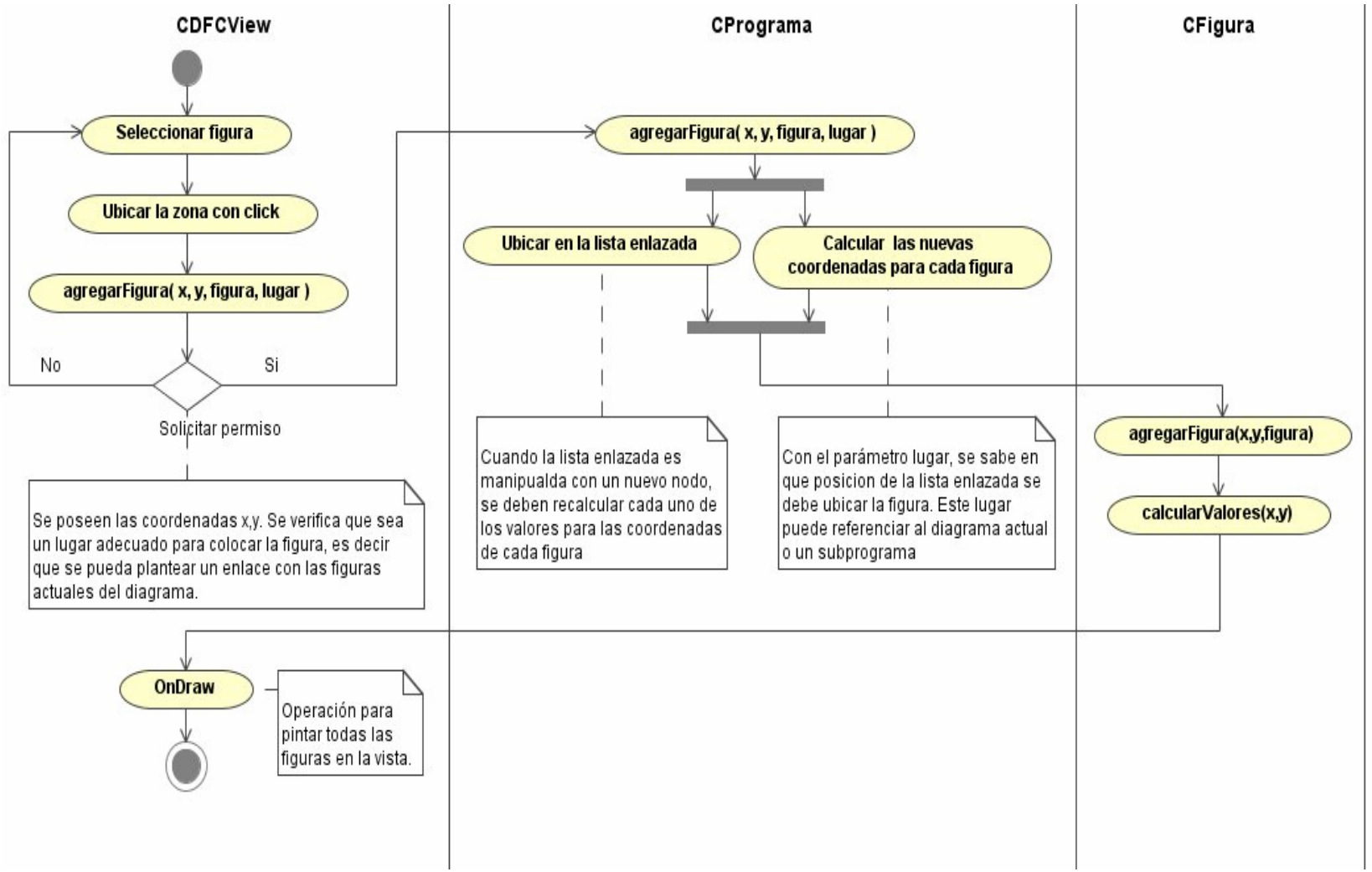

Figura 55. Diagrama de Actividades. Pintar figuras.

Después de agregar los nuevos nodos, se deben recalcular todos los valores de las figuras para que sean pintados en sus nuevas posiciones en la vista, después de este proceso, se invoca la operación OnDraw para el pintado final.

3.2.3.3.2. Manipulación de la lista enlazada. Las listas enlazadas usadas por DFC 1.0, son de doble enlace: *ant y *sig lo cual permite un doble enlace y sobre todo poder recorrer la lista de atrás hacia delante o en el camino contrario. Dependiendo de la figura que se quiere adicionar se agregan tantos nodos como sean necesarios.

Las figuras asignación, lectura, impresión y subllamada generan un solo nodo. Cada nodo posee las siguientes variables:

```
typedef struct estructuraF{
        CFigura figura;
        bool visitado;
        estructuraF *sig;
        estructuraF *ant;
        estructuraF *padre;
}*punteroF;
```

La variable figura como ya se ha expresado es la encargada de la manipulación en si de cada elemento, tenemos los apuntadores de avances: *sig, *ant; el apuntador *padre y la variable booleana *visitado,* se usan para la manipulación de los if.

Las figuras *hacer mientras*, *mientras que* y *para* generan dos nodos, esto para indicar que todo lo que se encuentre entre esos nodos pertenece a ese bloque, de esa forma podemos afirmar que estas figuras poseen un inicio y un fin; cada una de esas figuras tiene entre sus atributos un variable denominada consecutivo que hace las veces de llave principal y permite enlazar mediante un número de tipo long el inicio con el fin; esto posee una

gran desventaja, la variable consecutivo es incrementable y no retoma valores anteriores, es decir, siempre va en aumento, esto provoca que se pueda manipular un máximo valor para el consecutivo de 64 bits (9.223.372.036.854.775.807) según información de Visual C++.

La figura if es tal vez la más difícil de manipular, pues esta hace que el diagrama no solo aumente de arriba hacia abajo, si no que aumente para los lados, esto lleva a que el tamaño de los if, es manipulable según las figuras que posea. Su formación consta de cinco nodos, para representar el si, el fin del si, el no, el fin del no y el fin del if completo, todas comparten el mismo consecutivo y además se usa la variable padre; esta variable permite saber en cualquier momento si un if le pertenece o no a otro if, si le pertenece guarda la dirección de su padre, si no tiene padre guarda NULL. Los padres van a permitir estirar los if, pues esto lo hacen de acuerdo a los hijos que posean, es decir, la relación padre-hijo solo esta diseñada para los if, pues son las figuras que crecen horizontal y verticalmente, todas las demás figuras lo hacen de manera vertical.

Si un if tiene hijos (otros if) debe crecer horizontalmente tanto como sea necesario, para que se conserve lo entendible del diagrama, si no tiene hijos solo crece de forma vertical, pero hay que tener muy presente que un if, puede tener figuras de ambos lados, pero siempre debe ser simétrico verticalmente, es decir, sus lados deben conservar la misma longitud, caso contrario de forma horizontal, donde puede o no existir la simetría.

Para los subprogramas se maneja otra lista enlazada de forma muy similar a esta. Cada subprograma es un nuevo diagrama, de esta forma cada nodo de un subprograma, será una variable de tipo CPrograma, luego la manipulación de las figuras siguen el mismo proceso, estas son las ventajas de la programación orientada a objetos.

3.2.3.4. Etapa 2: Validación y edición de los diagramas. En esta etapa se puede introducir la información a los diagramas y manejar la parte de edición, que nos permite la manipulación de las figuras.

Figura 56. Barra de edición.

De izquierda a derecha tenemos: cortar, copiar, pegar, eliminar, deshacer y rehacer. Para la captura de información, se ofrecen cuadros de diálogos que permiten colocar los correspondientes valores.

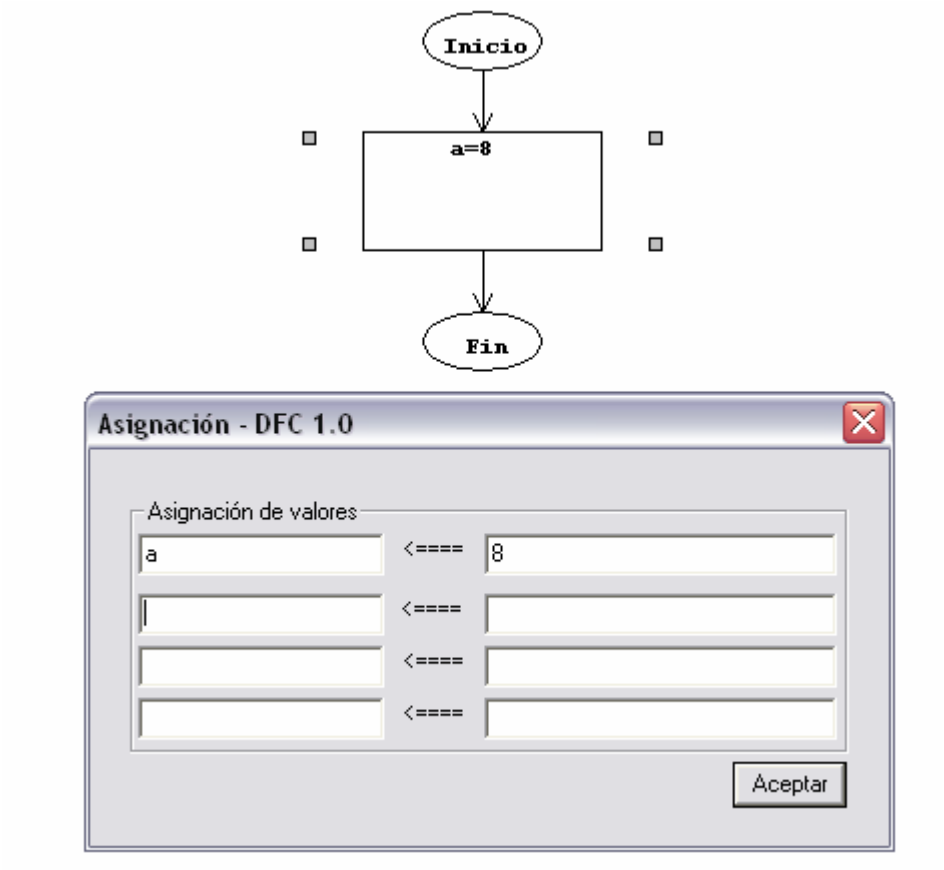

Figura 57. Captura de información.

Se observa la selección de la figura asignación, mediante cuatro puntos que la encierran, esto indica que la manipulación que se haga, se refiere a la selección actual. Si no hay figura seleccionada las opciones de edición no se activarán.

Estas opciones manejan directamente la lista de figura, si estamos en un subprograma, hay que recordar que su operación es idéntica a la del diagrama principal. La opción eliminar, lo que hace es liberar nodos de la lista enlazada, los nodos necesarios que la figura ha agregado, por eso en este caso es fundamental el consecutivo, pues permite encontrar el inicio y fin de la figura en cuestión; después de eliminar hay que liberar memoria, para que estas direcciones ocupadas sean reasignadas nuevamente. Si no hay una liberación de memoria, sino solo un rompimiento de enlaces, DFC 1.0 sigue funcionando perfectamente, el problema es que colapsará en algún instante por falta de memoria, de aquí que el uso de listas enlazadas, es de especial cuidado, si su manipulación no es la adecuada.

Cortar y copiar obedecen en cierto modo al mismo algoritmo, la diferencia es que cortar elimina los nodos, mientras que copiar no hace esta operación. Para realizar estas operaciones se debe hacer una imagen exacta de lo que queremos manipular, recordemos que la información importante en cada figura es: código, consecutivo, texto y padre; pues con las coordenadas de la figura Inicio, se puede armar todo el diagrama. Hay que tener claro que como se trabaja listas, cada vez que el portapapeles se use se deben generar nuevos nodos, pues al momento de pegar estos nuevos nodos formarán parte de la lista. El otro punto para tener en cuenta, es el consecutivo; cuando una figura entra al portapapeles se debe generar un nuevo consecutivo que no existente ni en el diagrama actual, ni en otra figura del portapapeles, recordemos que el consecutivo, es como la llave primaria, y por ende debe ser única.

Así cada vez que se agreguen figuras, sea por edición, o por nuevas figuras, el portapapeles debe ser actualizado con referencia a los consecutivos, en resumen no pueden existir consecutivos iguales, dentro de un mismo diagrama (solo pueden ser iguales, si pertenecen a la misma figura). Cuando en la manipulación de edición, existen if con padres, hay que tener claro que las direcciones apuntadas por estos hijos a sus padres, en el portapapeles debe cambiar, pues si se generan nuevos nodos, los hijos apuntan a nuevos padres; esto se logra llevando un historial de viejas direcciones con las actuales.

3.2.3.4.1. Rehacer y deshacer. Se hace especial énfasis en estos dos aspectos, por considerarlos de vital importancia, tanto para la manipulación de los diagramas, como para los recursos manejados. La implementación de estas operaciones se hace de manera tradicional, mediante el manejo de pilas (un caso especial de las listas enlazadas; ultimo en entrar, primero en salir); cada nodo de la pila permite guardar el diagrama principal y todos sus procedimientos, de esta forma cada vez que se agregue una figura o se use el menú de edición, se añadirá un nuevo nodo a la pila, teniendo un deshacer y un rehacer que no tienen limite, excepto la memoria permitida por el equipo. Por esto se ha considerado que estas opciones son de especial cuidado, pues aunque ofrecen una facilidad para la edición, consumen muchos recursos. Su definición es la siguiente:

```
typedef struct listaPila{
        punteroF figuras,cabezaF;
        punteroP procedimientos,cabezaP;
        listaPila *sig;
}*pila;
```

Los procesos de deshacer y rehacer es un paso de información entre la pila deshacer y la pila rehacer y que se ilustra en el diagrama de actividades.

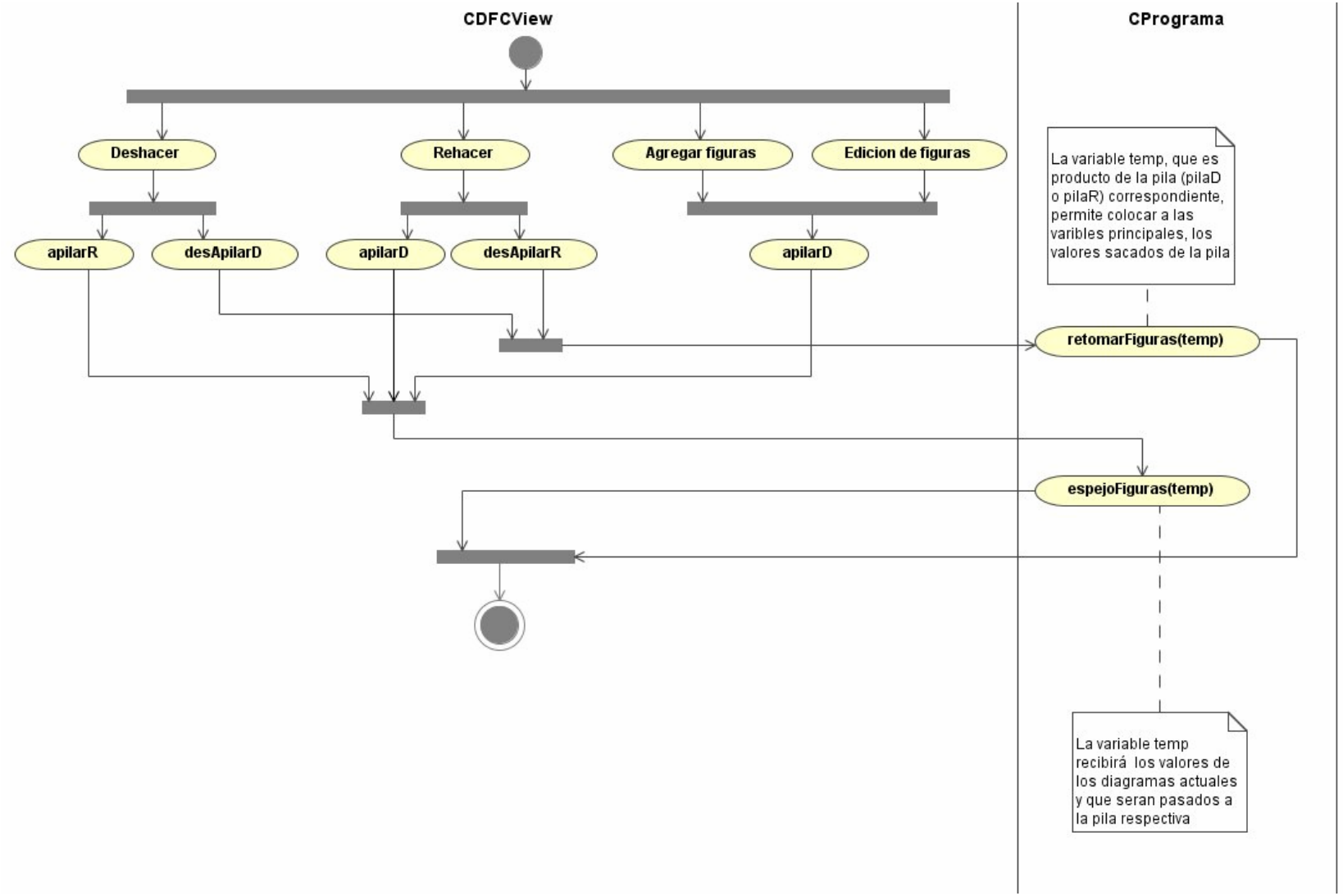

Figura 58. Diagrama de Actividades. Edición de figuras.

3.2.3.5. Etapa 3: Ejecución de los diagramas. Quizás se puede considerar como la parte más importante y difícil de este proyecto; hay que analizar que con lo valiosos e importantes que son los compiladores, la creación de un compilador puede ser un proceso largo y complicado. De hecho simplemente crear una biblioteca de tiempo de ejecución de un compilador es por sí misma una enorme tarea. Como contraste, la creación de un interprete de lenguaje es una tarea más sencilla y más fácil de tratar. Además, si está diseñado correctamente, el funcionamiento de un interprete es más fácil de entender que el de un compilador comparable. Sin despreciar la facilidad de desarrollo, los interpretes de lenguajes ofrecen una interesante característica que no se encuentra en los compiladores; un motor que realmente ejecuta el programa.[7]

La ejecución de una aplicación puede ser resumida como tres procesos básicos: leer variables, mostrar leyendas y variables y modificar las variables; bajo estos pilares se desarrolla esta etapa en DFC 1.0. Lo primero que se debe tener en cuenta es un buen evaluador de expresiones que permita las operaciones adecuadas y sobre todo precisas que se quieran realizar.

3.2.3.5.1. Evaluador de expresiones. En casi todos los lenguajes de programación, las expresiones se definen recursivamente utilizando un conjunto de reglas de producciones y echando mano de la teoría de autómatas y lenguajes formales. Por tanto DFC 1.0 utiliza el siguiente bosquejo de reglas de producciones:

<expresión> ::= <asignación>|<valord>
<asignación> ::= <valori>=<valord>
<valori> ::= <variable >

[7] SCHILDT, Herbert. C Manual de Referencia. Cuarta Edición. Editorial McGraw Hill. Madrid. 2001.

<valord> ::= <parte>|<op-rel> < parte>
<parte> ::= <término>|<+><término>|<-><término>
<término> ::= <factor>|<*><factor>|</><factor>|<%><factor>
<factor> ::= <+>|<->|<átomo>
<átomo> ::= <variable>|<números>|<función>|<expresión>

Op-rel se refiere a cualquiera de los operadores definidos en DFC 1.0. Los términos valori y valord se refieren a objetos que pueden aparecer, respectivamente en la parte izquierda y derecha de una instrucción de asignación. Una cosa muy importante es la precedencia de operadores, la cual esta incorporada en las reglas de producción. A mayor precedencia, más abajo en la lista estará el operador.

Para evaluar una expresión, esta se toma como una cadena de caracteres donde cada unidad indivisible y con significado se denomina token; así pues nos encontramos con que un expresión puede contener: FUNCIÓN, VARIABLES, NÚMEROS, DELIMITADORES.

Las funciones son aquellas expresiones que fueron desarrolladas como complementos a las operaciones básicas con las que puede trabajar el usuario, así pues expresiones como sin ,cos, tan; se tratan como funciones.

Las variables son las expresiones que siguen la siguiente ley de producción:
<variable> ::= <letra >|<letra><sVariable>
<sVariable> ::= <letra >|<letra><sVariable>|<numero>|<numero><sVariable>
<letra> ::= a|b...|z|A|B..Z
<numero> ::= 0|1..|9
así vemos que las variables contienen letras y números, pero que necesariamente deben empezar por una letra.

Los números son los dígitos utilizados desde 0 a 9 y los delimitadores son todos los caracteres que marcan un cambio de token: +,-,*,/,%,^,=,(,),!,&,|,<,>,[,].

Si tenemos a+b/8, los token encontrados son:

a -> VARIABLES

+ -> DELIMITADOR

b -> VARIABLES

/ -> DELIMITADOR

8 ->NÚMEROS

Según las reglas de producciones, tenemos término + término; donde el primer término es una variable “a” y el segundo término es factor/factor; el primer factor representa una variable “b”, luego tenemos un delimitador / y por último tenemos el otro factor que representa un número, de esta forma por la recursividad del interprete se empieza a retroceder y se evalúa b/8, luego la variable a es sumada con el resultado de esta expresión.

En este orden de ideas, el evaluador toma una expresión y empieza sus llamadas recursivas, entra por una función denominada iniciar, luego se llama a eval_exp0 que se encarga de analizar si es una asignación o una evaluación normal, esta a su vez llama a eval_exp1, que llama a eval_exp2, luego eval_exp3, eval_exp4, eval_exp5, eval_exp6; esta última función se encarga de mirar si hay paréntesis en la expresión, si esto es cierto captura el primer token ‘(‘ y llama a eval_exp0, al regresar debe terminar en ‘)’, sino supone que el token es un átomo, es decir, una expresión que necesita ser evaluada inmediatamente como son las variables, funciones o números.

Cuando sale de eval_exp6 y llega a eval_exp5, siguiendo la precedencia de operadores verifica si es un + o – monario para su evaluación, luego regresa

a eval_exp4 para comprobar si hay exponente, cuando llega a eval_exp3, analiza si es *,/ o %, al llegar a eval_exp2 suma o resta los términos y al llegar a eval_exp1 analiza si son operadores relacionales; de esta manera el procedimiento se repite hasta que se terminen con todos los token.

3.2.3.5.2. Ejecutar, ejecutar hasta y paso a paso. El desarrollo de estos tres procesos se lleva a cabo mediante el uso de la barra de ejecución, en ella podemos encontrar de izquierda a derecha: ejecutar, detener, pausar, paso a paso y ejecutar hasta.

Figura 59. Barra de ejecución

La opción ejecutar desarrolla toda la lógica del diagrama de manera no interrumpida e interactuando con el usuario donde sea necesario; ejecutar hasta desarrolla la lógica hasta el lugar determinado y paso a paso va objeto por objeto realizando lo que es necesario, aquí hay que destacar que existe una relación directa entre estas tres opciones, y estando paso a paso, si se desea se puede ejecutar toda la lógica desde el lugar donde nos encontramos hasta el final, de la misma forma sucede con el ejecutar hasta, que puede interactuar con paso a paso y ejecutar.

Para mantener una interacción con el usuario se usan dos cuadros de diálogos que permiten leer y mostrar valores en pantalla.

En la figura 60, se observa como se pretende leer la variable “longitud” y como el cuadro de diálogo nos ofrece una pequeña ayuda con su leyenda; en la figura 61, se aprecia como se muestra un leyenda en la aplicación.

Figura 60. Leer valores

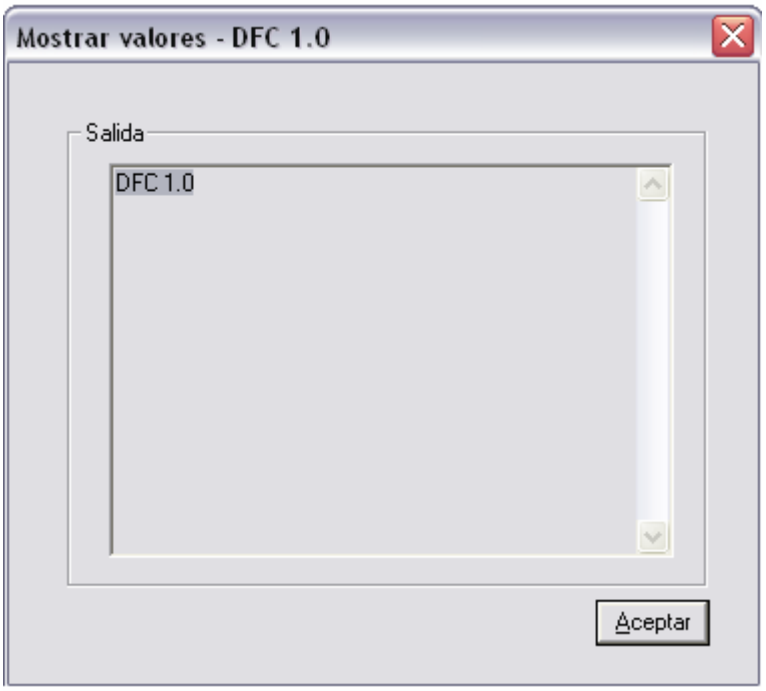

Figura 61. Mostrar valores

Los tres procesos mencionados operan de forma similar, todos poseen dos variables necesarias para su funcionamiento, las cuales son: inicio y fin cuyo tipado es punteroF o un nodo de la lista de figuras. Con esta información el proceso se ejecuta pasando por los diferentes objetos a los que haya lugar.

Si la ejecución pasa por el objeto asignación, se hace un llamado al evaluador de expresiones para devuelva un valor y modifique las variables a las que haya lugar. Si tenemos un objeto lectura o impresión se llaman a los cuadro de diálogos de las figuras 60 y 61 respectivamente. Para poder manejar variables están deben haber sido inicializadas con anterioridad.

Para los objetos decisión, hacer mientras, mientras que, para y subllamada el proceso es un poco diferente. Si analizamos el objeto de decisión este se compone de cinco nodos: decisión, fin del si, else, fin del else y fin del if, identifiquémoslo respectivamente como DECISIÓN, FINIFS, ELSE, FINIFN, FINIF.

Al evaluar la expresión de la decisión esta puede ser o verdadera o falsa; hay que recordar que según C++, verdadero es todo valor diferente de cero y falso es lo contrario. Si la expresión es verdadera, de DECISIÓN se debe avanzar a la siguiente instrucción, si en llegado caso por este camino se ubica en FINIFS, se debe saltar una posición por delante de FINIF. En caso de que la expresión es falsa, de DECISIÓN se debe llegar un paso delante de ELSE y continuar con el proceso, cuando se encuentre FINIFN se debe saltar a FINIF, por lo tanto vemos como la posición actual de la lista enlazada debe saltar de un nodo a otro nodo; en este caso es sólo hacia delante, pero en otros toca retroceder y de ahí la importancia de hacer listas doblemente enlazadas.

El proceso de ejecutar un mientras que, puede describirse de la siguiente forma: en primer lugar el mientras es referenciado por MIENTRAS y el fin del mientras por CIERRE. Cuando se llega a un MIENTRAS y su condición es evaluada como verdadera se debe pasar al siguiente nodo después del MIENTRAS, en caso contrario se debe llegar al CIERRE, si una expresión llega al CIERRE debe regresarse a su MIENTRAS y de ahí la importancia de las listas doblemente enlazadas; para su correspondiente evaluación.

Si analizamos el objeto hacer mientras, conocido como ABRIR y HASTA su proceso es muy similar al anterior; si llegamos al ABRIR se debe analizar el siguiente nodo; cuando se encuentre un HASTA se evalúa su condición, si es verdadera lo devuelve al ABRIR, si es falsa pasa al siguiente nodo.

EL objeto para, usado como PARA y CIERRE, complica un poco las cosas; al llegar a PARA se debe analizar si es la primera vez que visita el ciclo, esto se sabe con un miembro booleano de los nodos nombrado como “visitado”, si este es verdadero PARA ya fue inicializado, de lo contrario de inicializa y se coloca visitado en falso. Al mirar si debe entrar al ciclo, se evalúa la condición del ciclo dando como resultado verdadero o falso, si es falso se salta a CIERRE y se limpia el visitado de PARA en falso, si es verdadero se pasa a la siguiente instrucción. Al llegar al cierre se aumenta el PARA y se devuelve para continuar con el proceso.

Al estudiar diferentes tópicos de programación de computadores, se encuentra que al momento de realizar una subllamada se guardan ciertos valores en la pila y la siguiente instrucción después de la llamada, para que al momento de regresar se pueda seguir con el proceso, pues DFC 1.0 implementa un sistema de pila con una lista simplemente enlazada manejada como pila (último en entrar, primero en salir. LIFO o UEPS) y dos operaciones básicas.

La declaración de la pila con sus operaciones es la siguiente:

```
typedef struct listaLlamadas{
        short accion;
        punteroP procedimiento;
        punteroF figura;
        punteroV variables;
        listaLlamadas *sig;
}*punteroL;

void push(short accion, punteroF figura, punteroP procedimiento, punteroV
variables);
void pop(short &accion, punteroF &figura, punteroP &procedimiento,
punteroV &variables);
```

Las operaciones push y pop agregan a la pila y sacan de ella respectivamente, de esta forma al momento de ejecutar una llamada se mete a la pila la siguiente instrucción desde de la posición actual, con sus variables e información adicional, como lo es acción que se usa para el pintado de las figuras y procedimiento que permite saber en que procedimiento nos encontramos actualmente o si estamos en el procedimiento principal, esto con el objeto de pasar los valores cuando se invoca la subllamada a los parámetros del procedimiento a realizar. Después de finalizada la lógica contenida en un procedimiento se procede a sacar de la pila para continuar con la lógica del actual diagrama.

Como se mencionó un procedimiento no es mas que el programa principal con otros valores y esto es lo que hacen las tres operaciones de ejecución sobrescribir valores para luego ejecutar apoyándose de la pila de llamadas.

Para que el proceso de ejecución tenga éxito, se pretende tratar de analizar todos los errores que el usuario puede tener en la construcción de su diagrama, tratando de esta forma que no se ejecute una lógica si hay errores de construcción; si existen errores lógicos, es decir situaciones no favorables

por el análisis no adecuado que hizo el usuario, DFC 1.0 empieza su ejecución.

3.2.3.5.3. Precedencia de Funciones y Operadores. Las funciones matemáticas tienen todas la misma prioridad, y se evalúan de izquierda a derecha. Cuando hay una expresión con varios operadores, se evalúan primero los matemáticos y luego los operadores lógicos relacionales.

Los operadores lógicos y relacionales tienen todos la misma prioridad, se evalúan de izquierda a derecha. Esto se hace con la intención de motivar al usuario que se inicia en el arte de la programación al uso de los paréntesis en las expresiones lógicas. De esta forma a>b||c>b se considera una expresión no válida para estimular a usarla así: (a>b)||(c>b).

Los operadores matemáticos tienen la siguiente precedencia:

Mayor
+ y - monarios
^
* / %
+ -
=
Menor

Es importante tener en cuenta que la finalidad de DFC 1.0 es impulsar la adquisición de destrezas para resolver problemas informáticos y por eso se ha hecho un excesivo énfasis en que el usuario puede desarrollar expresiones claras y coherentes; de ahí el marcado uso del paréntesis.

3.2.3.6. Etapa 4: Almacenamiento de la información. Esta etapa le da al usuario la facilidad de poder conservar su trabajo como proceso de aprendizaje y luego poder editarlo como parte del aspecto de retroalimentación.

Se usan los cuadros de diálogos de Windows para capturar la ruta en la cual se pretende guardar o leer un archivo; después de tener la ruta del archivo se declara una variable de tipo FILE de la librería stdio.h que soporta el almacenamiento en disco. Se define entonces la información relevante y con la cual DFC 1.0 es capaz de construir nuevamente el diagrama. La manipulación es permitida por la siguiente barra:

Figura 62. Barra de almacenamiento

La estructura definida para almacenar la información es:

```
typedef struct listaFiguras{
        short codigo;
        char texto1[120];
        char texto2[120];
        char texto3[120];
        char texto4[120];
        long consecutivo,cPadre;
        long x,y;
        bool padreIF;
}lista;
```

La variable codigo permite capturar la figura que se está manipulando, los textos, las diferentes expresiones que contiene el objeto; consecutivo es su llave primaria y que la identifica en la lista; x,y sus coordenadas de inicio para el pintado y cPadre y padreIF que soportan el trabajo con los IF anidados, que como fue analizado, es un poco complicado de trabajar.

3.2.3.6.1. Guardar. Para guardar la información, después de tener la ruta de archivo, se recorre la lista enlazada y se pasa la información necesaria a la estructura. Como medida de reconocer que el formato es válido el primer registro del archivo contiene -1 en sus variables numéricas, false en sus variables boolenas y “DFC 1.0” en sus variables texto.

Se continúa con cada uno de los procedimientos, que están compuestos de figuras, para este caso la cabecera de registro es igual al caso anterior, pero contienen la leyenda "PROCEDIMIENTOS-C++".

El tamaño del archivo que resulte depende del tamaño de la estructura que se guarde, así que éste es directamente proporcional a la cantidad de figuras usadas.

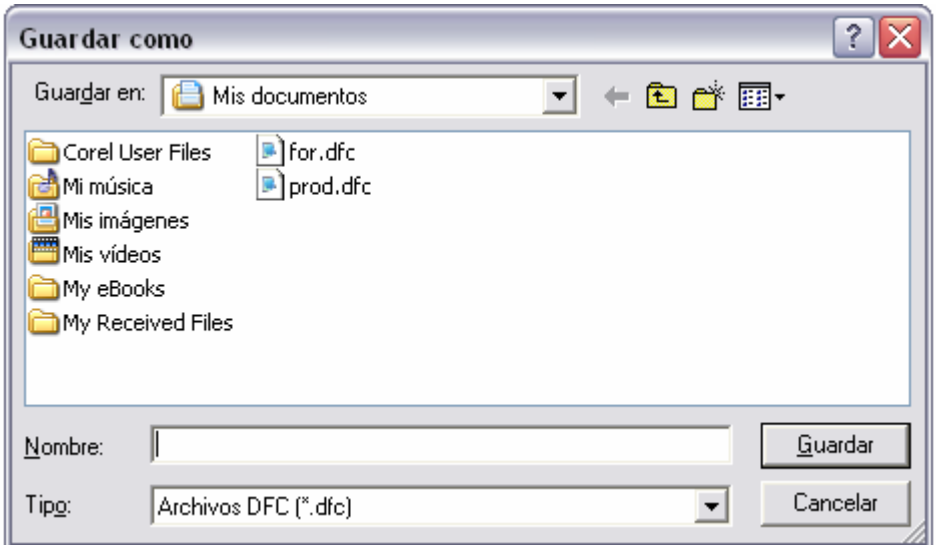

Figura 63. Diálogo guardar

3.2.3.6.2. Abrir. Para leer la información; después de capturada la ruta del archivo, se comprueba si el formato es válido. Se debe liberar toda la memoria que se esté usando en ese momento y empezar a crear los nuevos nodos que contendrá el diagrama que se quiere cargar.

De esta manera se recorre el archivo y se empiezan a crear figuras hasta que se encuentre el bloque de "PROCEDIMIENTOS-C++", el cual marca el comienzo de procedimientos y el enlace de las figuras siguientes con el procedimiento que se crea.

Los if, manejan una información adicional necesaria para su pintado y la cual es la dirección del padre al que pertenecen. En este caso se recupera el consecutivo del padre y se busca ese consecutivo en la nueva lista creada, al momento de encontrarlo, es esa, la nueva dirección que debe ser copiada en el campo padre. Encontrar este padre depende del lado en el cual se encuentre el hijo, por esa forma la variable booleana padreIF, indica si la figura es hija del if o del else. Esta información es de vital importancia a la hora de pintar las figuras.

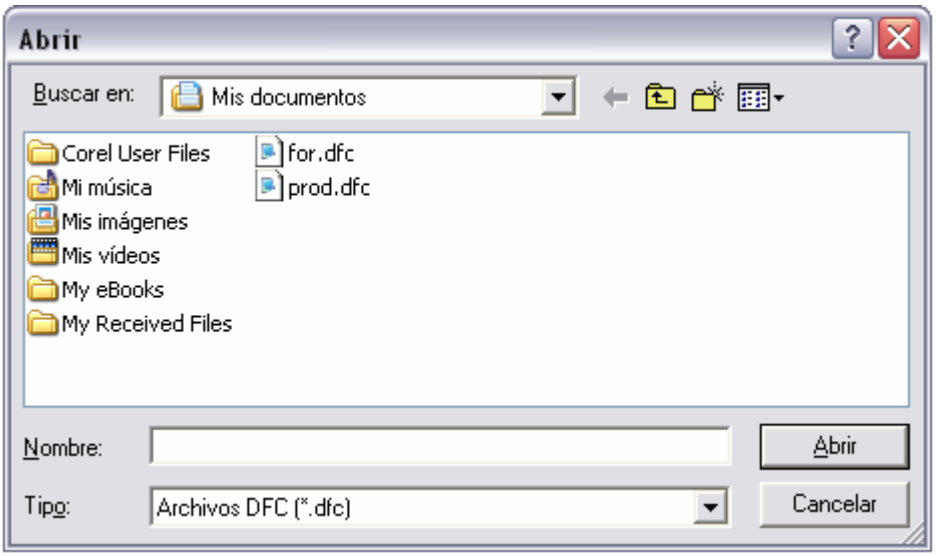

Figura 64. Diálogo abrir

3.2.3.7. Etapa 5: Generación de código fuente. Es un proceso que está separado en dos partes: generar código para una estructura o generar el código total del diagrama y los cuales son manipulados por el menú ver y esta barra de herramientas:

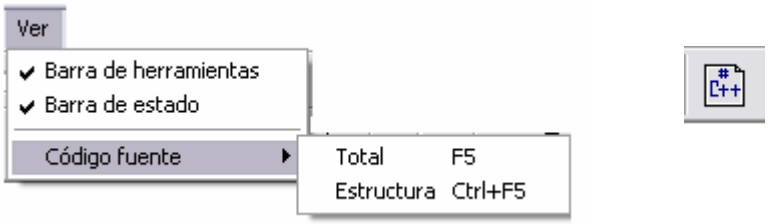

Figura 65. Ver código fuente

Antes de mostrarle código al usuario se realiza el proceso de compilación adecuado, para luego tratar de generar un código cien por ciento libre de errores de sintaxis. De esta forma el código fuente de una determinada estructura, bien sea sencilla: INICIO, FIN, ASIGNACION, LECTURA, IMPRESIÓN, SUBLLAMADA o compuesta: DECISIÓN, PARA, MIENTRAS, HASTA; es mostrado en un cuadro de diálogo. El código total es generado en un archivo de texto con extensión .cpp guardado en el lugar escogido por el usuario.

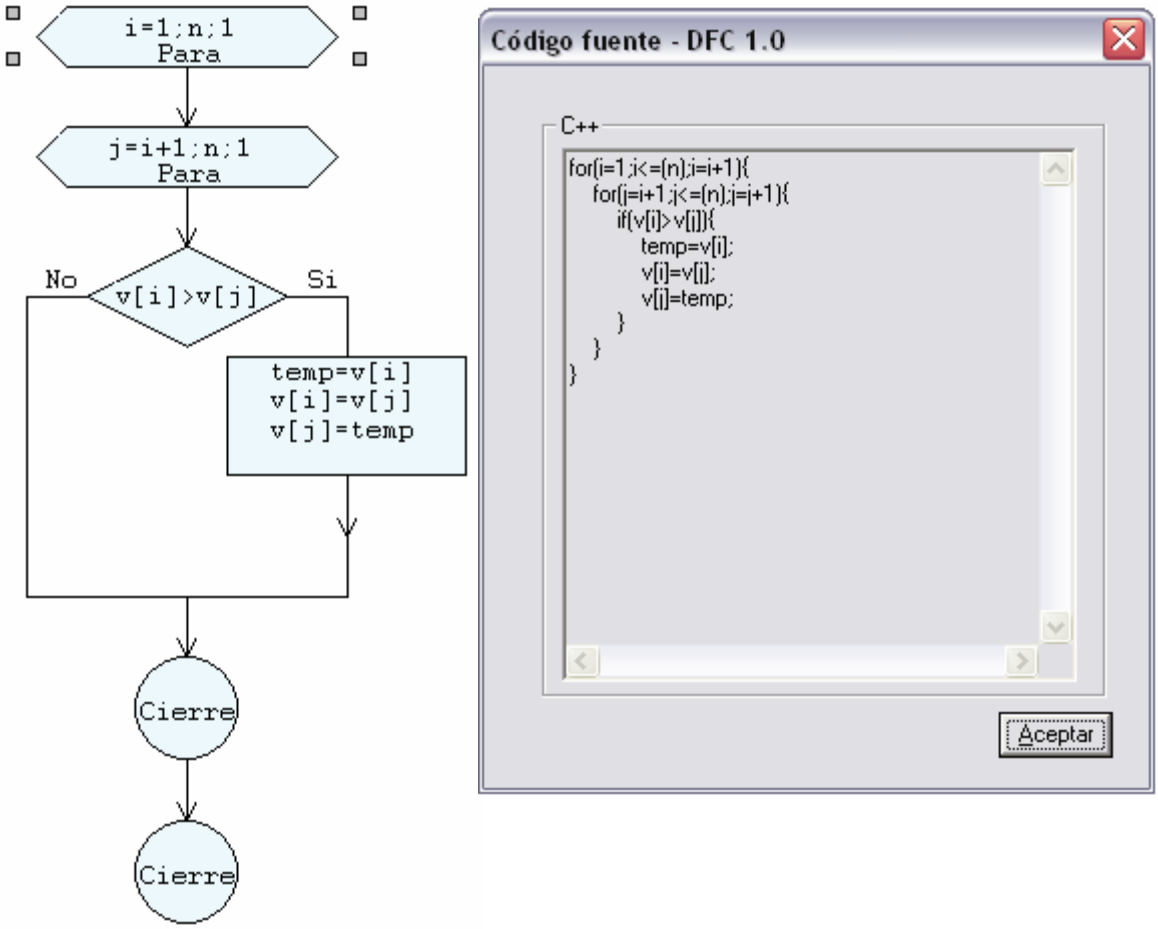

Figura 66. Generación de código fuente

En la figura 66 se observa como es generado el código de la estructura seleccionada, en este caso la figura PARA, la cual busca a su figura CIERRE

y genera el código, de ahí que se conoce como un objeto compuesto. Para este proceso se recorren todas las figuras en cuestión y a cada uno se le genera su franja de código. Para tabular, se tienen en cuenta las figuras que generan tabulaciones (las compuestas) y se lleva un contador de tabulaciones, y cada vez que aparece un cierre, el contador se disminuye en uno.

DFC 1.0 usó palabras reservadas que no pertenecen al lenguaje C++, por ejemplo a!|b es una NOR entre a y b y su equivalente en C++ es !(a||b), de esta forma lo asume la herramienta, se hace esta conversión solo para generación de código.

Con las funciones implementadas como INC y DEC entre otras, DFC 1.0, contribuye a que el usuario afiance su lógica y aquí no hace la transformación directa, pero cuando genera el código total INC, DEC, TRUNC,... son implementadas como funciones, de esta forma si el usuario emplea DEC(A) en una expresión, DFC 1.0 agrega al código fuente:

```
double dec (double a){
        a=a+1;
        return a;
}
```

Esto lleva a que el usuario se familiarice con el uso de funciones, que es una de las ventajas de C++.

DFC 1.0 no distingue entre mayúsculas y minúsculas, por eso la variable numero y NumEro son iguales, pero al momento de generar código en C++, que si distingue de mayúsculas y minúsculas, se lleva todo a un formato familiar con C++, como es el de minúsculas, pero las cadenas “ “, se conservan intactas.

4. CONCLUSIONES

Con la realización de este proyecto se logró verificar que la Programación Orientada a Objetos es una de las metodologías mas fuertes cuando de desarrollo de software se trata, y usada de una forma adecuada puede reducir los tiempos estipulados de trabajo, permitiendo aspectos fundamentales como la reutilización del software.

La teoría de autómatas y lenguajes formales se convierte en un tópico esencial al momento de desarrollar interpretes de lenguajes y querer avanzar en el área de los compiladores.

El uso de memoria dinámica es de vital importancia, cuando se desean programas eficientes, pero ante todo óptimos, que aprovechen al máximo cada uno de los recursos que consumen, liberándolos cuando no los utilizan.

De un correcto algoritmo, los problemas más complejos adquieren solución, por eso lo mas importante a la hora de empezar a desarrollar software es mantener una lógica acertada; y esto se logra solo con la práctica. De esta forma DFC 1.0 es el inicio de una práctica constante y segura.

La solución de problemas informáticos no requiere únicamente expertos en un determinado lenguaje, requiere personas que sean capaces de expresar mediante pasos lógicos, un algoritmo que pueda ser solución. Así DFC 1.0 recalca el factor de la lógica como aspecto fundamental y genera código como aspecto complementario; lo cual conlleva a que el usuario pueda interrelacionar de forma adecuada los tres conceptos básicos del arte de programar.

5. RECOMENDACIONES

Para futuras versiones de la herramienta, es propicio avanzar inicialmente en la generación de código fuente en otros lenguajes de programación como puede ser Basic, por su manera sencilla de operar; y para desligar al usuario de la percepción, que aprender a programar es saber usar las palabras reservadas del lenguaje.

Trabajar con otros tipos de datos diferentes al numérico (double) y al formato no numérico, para permitir un mejor aprovechamiento de los recursos.

Permitir la manipulación de cadenas y el paso de arreglos como parámetros de funciones, para contribuir con el proceso de aprendizaje de los usuarios.

Realizar el proceso inverso a lo ejecutado por DFC 1.0, lo cual conlleva a que el usuario digite código fuente y se genere el respetivo diagrama de flujo.

Continuar con procesos que conlleven a la adquisición de destreza lógica antes que aprendizaje de un determinado lenguaje.

BIBLIOGRAFÍA

BOOCH, Grady. RUMBAUGH, James. JACOBSON Ivar. El Lenguaje Unificado de Modelado. Primera Edición. Editorial Addison Wesley Iberoamericana. Madrid. 1999.

CEBALLOS, Francisco Javier. Programación Orientada a Objetos con C++. Primera Edición. Editorial Ra-ma. Madrid. 1993.

CEBALLOS, Francisco Javier. Visual C++. Aplicaciones para Win32. Primera Edición. Editorial Ra-ma. Madrid. 1998.

GRECH, Pablo. Introducción a la Ingeniería - Un enfoque a través del diseño. Primera Edición. Editorial Prentice Hall. Bogotá. 2001.

GREGORY, Kate. Microsoft Visual C++ 6. Edición especial. Editorial Prentice hall. Madrid 1999.

JOYANES, Luis. Programación Orientada a Objetos. Segunda Edición. Editorial Mc Graw Hill. Madrid. 1998.

KELLEY, Dean. Teoría de Autómatas y Lenguajes Formales. Primera Edición. Editorial Prentice Hall. Madrid. 1998.

KRUGLINSKI, David. Programación avanzada con Microsoft Visual C++. Primera Edicion, Mc Graw Hill. Madrid 1998.

MCCONNELL, Steve. Desarrollo y Gestión de Proyectos Informáticos. Primera Edición. Editorial McGraw Hill. Madrid. 1997.

PRESMAN, Roger S. Ingeniería del Software - Un enfoque práctico. Quinta Edición. Editorial McGraw Hill. Madrid. 2002.

SCHILDT, Herbert. C Manual de Referencia. Cuarta Edición. Editorial McGraw Hill. Madrid. 2001.

VILLALOBOS, Jorge. Diseño y Manejo de Estructuras de Datos en C. Primera Edición. Editorial McGraw Hill. Bogotá. 1996.

ANEXOS

ANEXO A. MANUAL DE USUARIO

DFC 1.0

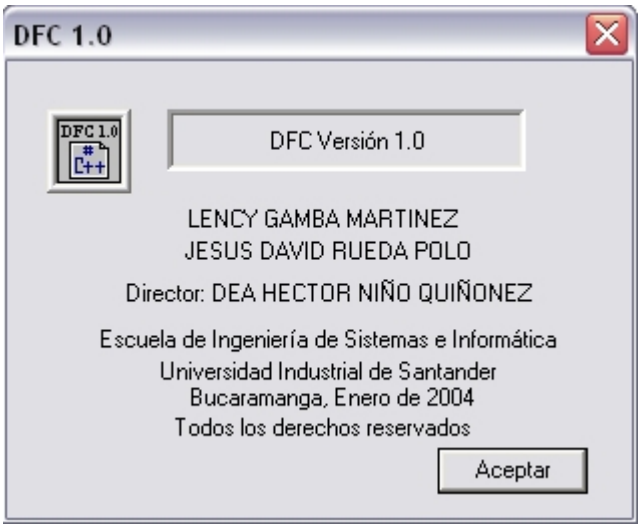

Figura 1. Acerca de

Conceptos Básicos

¿Qué es DFC 1.0?

DFC 1.0 es un editor de diagramas de flujo, que contribuye con el perfeccionamiento de las técnicas de programación, permitiendo sintetizar la lógica de un problema en un diagrama y mostrando los resultados que esta lógica representa y como complemento es capaz de generar el código fuente de un diagrama semántica y sintácticamente correcto.

Diagrama de flujo

El diagrama de flujo hace uso de elementos gráficos para expresar las secuencias de operaciones que deben realizarse para resolver un problema.
Un diagrama de flujo muestra la lógica de un algoritmo, haciendo énfasis en los pasos individuales y sus interrelaciones.

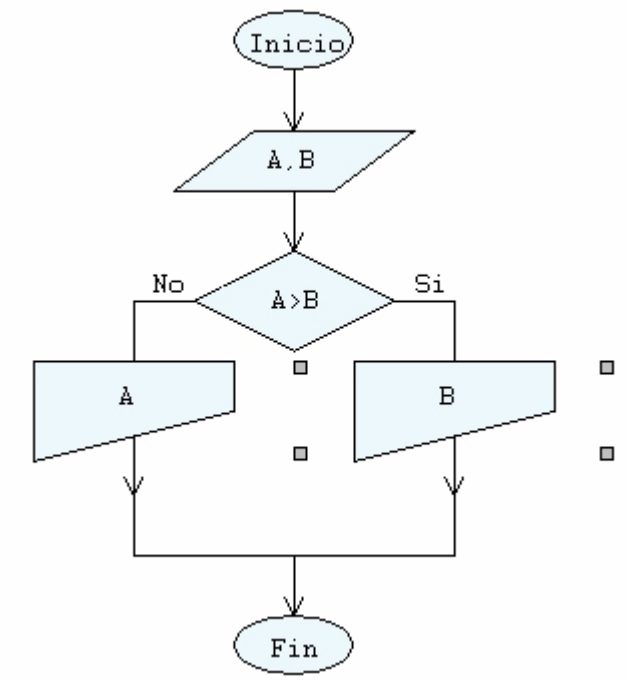

Figura 2. Diagrama de flujo

El diagrama de flujo asociado, muestra un algoritmo donde inicialmente se leen las variables A y B, se selecciona una alternativa A>B, si esta condición es verdadera se imprime el valor de la variable A, de lo contrario se imprime el valor de la variable B.

Algoritmo

Un algoritmo es el conjunto de operaciones y procedimientos a realizar que deben seguirse en un orden para resolver un problema.

Los algoritmos se caracterizan por ser:

Preciso. Definirse de manera rigurosa, sin dar lugar a ambigüedades.

Definido. Si se sigue un algoritmo dos veces, se obtendrá el mismo resultado.

Finito. Debe terminar en algún momento.

Debe tener cero o más elementos de entrada, es decir, debe tener por lo menos una instrucción que ordene averiguar el dato o los datos.

Debe producir un resultado. Los datos de salida serán los resultados de efectuar las instrucciones. Los datos de entrada pueden ser ninguno, pero los de la salida deben ser alguno o algunos.

Se concluye que un algoritmo debe ser suficiente y breve, es decir, no exceder en las instrucciones ni quedarse corto. Entre dos algoritmos que lleven a un mismo objetivo, siempre será mejor el más corto.

Código Fuente

Es la forma de expresar un algoritmo en un lenguaje de programación, en donde se tienen en cuenta aspectos sintácticos y semánticos propios del lenguaje.

Operandos

Pueden ser cualquier variable o valor constante involucradas en el algoritmo; sobre los cuales se quiere representar una relación.

Operadores

Son símbolos que muestran una relación entre operandos.

Se dividen en lógicos, relacionales y matemáticos.

- **Relacionales.**

  >. Mayor que

  >=. Mayor o igual que

  <=. Menor o igual que

  ==. Igual que

  !=. Diferente de

- **Lógicos:**

  && Operación and lógica

  ||. Operación or lógica

!|. Operación negación or lógica

NOT. Negación lógica

!&. Negación del and.

Matemáticos.

+. Adición

-. Sustracción

*. Multiplicación

/. División

%. Modulo de la división

^. Exponenciación

=. Asignación.

. Separador fraccionario

& operador de concatenación para el objeto impresión.

Funciones Matemáticas

Se cuenta con la siguiente gama de funciones matemáticas:

INC (X). Incrementa la variable X en 1, afectando su valor

DEC (X). Decrementa la variable X en 1, afectando su valor

RANDOM (X). Devuelve un numérico randómico entre 0 y X

ROUND (X). Redondea al entero más cercano de X

TRUNC (X). Extrae la parte entera de X

ABS (X). Devuelve el valor absoluto de X

LN (X). Devuelve el logaritmo natural de X

EXP (X). Devuelve e ^ X. Donde e = 2.718281828 aprox.

SQRT (X). Devuelve la raíz cuadrada de X

SIN (X). Devuelve el seno de X en radianes

COS (X). Devuelve el coseno de X en radianes

TAN (X). Devuelve la tangente de X en radianes

ASIN (X). Devuelve el seno inverso de X

ACOS (X). Devuelve el coseno inverso de X

ATAN (X). Devuelve la tangente inversa de X

Las funciones matemáticas en DFC se trabajan en radianes. Para convertir radianes en grados, multiplique los radianes por 180/PI.

Para convertir grados en radianes multiplique los grados por PI/180.

Expresión

Es cualquier combinación válida de operadores, constantes y variables.

Constantes

Las constantes refieren valores fijos que no pueden ser modificados por el programa.

Variable

Por variable se entiende una cantidad a la que se alude por su nombre y no por su contenido, ya que su contenido seguramente cambiará a medida que el programa se vaya ejecutando.

Cada variable tiene su ubicación en la memoria principal del computador, lugar en que se almacena su contenido. Así, cada variable tiene una dirección en la memoria del computador; de conocer y manejar esa dirección se encarga el sistema operativo. Se conoce y alude a esa zona por el nombre que se le ha dado.

Tipos de Datos

En DFC se manejan dos tipos de datos: el numérico y el no numérico. Dentro del formato numérico se tienen los arreglos que se pueden manejar y que comúnmente son conocidos como matrices, se pueden tener matrices hasta de dos dimensiones y se pueden acceder usando los corchetes (matriz[i][j]).

El máximo valor que se puede almacenar en formato numérico es de: 1.7E +/- 308 (15 dígitos). El formato no numérico incluye todas las cadenas de caracteres alfabéticas y alfanuméricas. Para representar las cadenas de caracteres se pueden usar las comillas dobles " ". El formato no numérico solo es permitido en la figura imprimir.

Parámetros

Son las variables que se definen en un Subprograma para la comunicación con el programa que lo invoca; estos deben ir escritos sin paréntesis y separados por comas.

Si se declara un parámetro como referencia, el argumento de este debe ser una variable.

Argumentos

Son variables existentes en el programa que se pasan mediante el objeto subllamada al Subprograma para su correcto funcionamiento.

En DFC un argumento es pasado por referencia cuando es un nombre de variable y su correspondiente parámetro es antecedido por un *, en ningún caso ni los argumentos ni los parámetros deben ser vectores o matrices. Cuando un argumento es pasado por referencia, los cambios efectuados a la variable dentro del subprograma invocado serán reflejados en el programa que realizó la llamada. En caso contrario, si los argumentos son pasados por valor, los cambios efectuados al parámetro que recibe el valor del argumento en el subprograma, no se reflejarán en el programa que hizo la llamada.

Recursividad

La recursividad es el proceso de definir algo en términos de sí mismo; así por ejemplo los procedimientos pueden llamarse a sí mismos.

Cuando un procedimiento se llama a sí mismo, se asigna un espacio en la pila para los datos, y el código del procedimiento se ejecuta con estos nuevos datos desde el principio. Una llamada recursiva no produce una nueva copia del procedimiento; sólo los valores con los que trabaja son nuevos. Al volver de la llamada recursiva se recuperan de la pila los datos y la ejecución se reanuda inmediatamente después del punto donde se llamó recursivamente al procedimiento dentro del procedimiento.

En DFC para efectos de recursividad, no importa la forma de declaración de los parámetros (valor o referencia), es decir, siempre serán operados como referencia en los procedimientos recursivos, por tanto, los cambios efectuados a la variable parámetro dentro del procedimiento invocado serán reflejados en el subprograma que realizó la llamada recursiva.

Ejemplo:

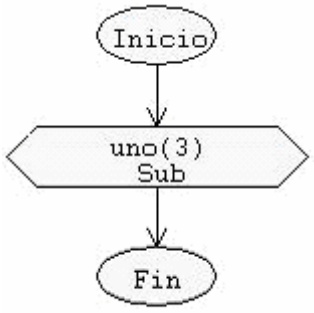

Figura 3. Programa principal

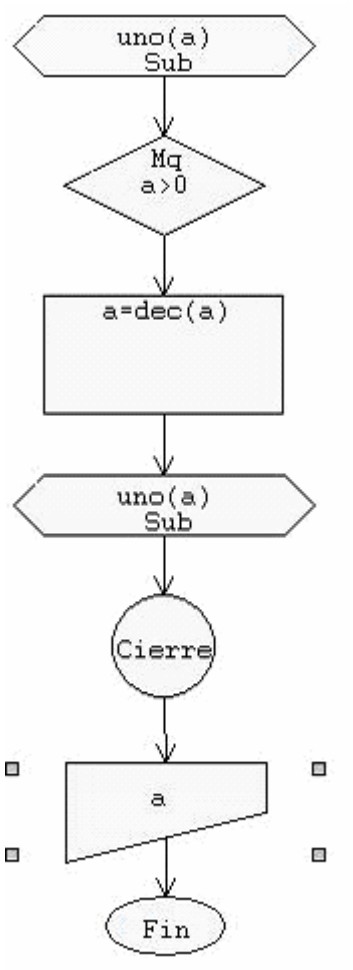

Figura 4. Subprograma

En el programa principal se hace una llamada a un subprograma "Uno", con un argumento por valor "3". El subprograma "Uno" tiene un parámetro "a" el cual recibe el valor de 3, por el argumento por valor pasado por la subllamada, el procedimiento del subprograma se hace mientras la variable "a" sea mayor que 0, se decrementa y se llama recursivamente el mismo procedimiento con la misma variable por referencia como parámetro; esta variable se decrementa al paso recursivo de la llamada hasta que es igual a 0; en este punto empieza a retornar el procedimiento, produciendo los siguientes valores como resultado:

a=0

Retorno 1: a=0

Retorno 2: a=0

Retorno 3: a=0

Estos retornos del procedimiento se producen porque la variable "a" se pasa por referencia.

OPCIONES DE MENÚ DE DFC 1.0

Figura 5. Opciones de menú de DFC 1.0

Menú Archivo

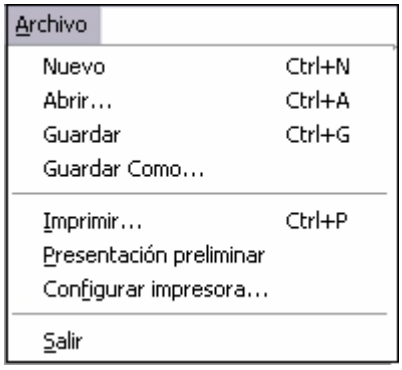

Figura 6. Menú archivo

Nuevo

Este comando permite crear un diagrama de flujo.

Formas de acceder al comando:

Barra de herramientas: 

Teclado: CTRL + N

Al acceder a este comando, se muestra su cuadro de diálogo, donde advierte acerca de la perdida de los cambios no guardados.

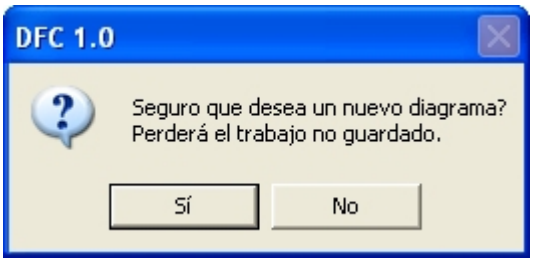

Figura 7. Cuadro de diálogo del comando nuevo

Abrir

Este comando permite abrir un diagrama de flujo anteriormente creado, listo para editar.

Formas de acceder al comando:

Barra de herramientas: 

Teclado: CTRL + A

Al acceder a este comando, se muestra su cuadro de diálogo, para indicar la ruta donde se encuentra el archivo .dfc que se desea abrir.

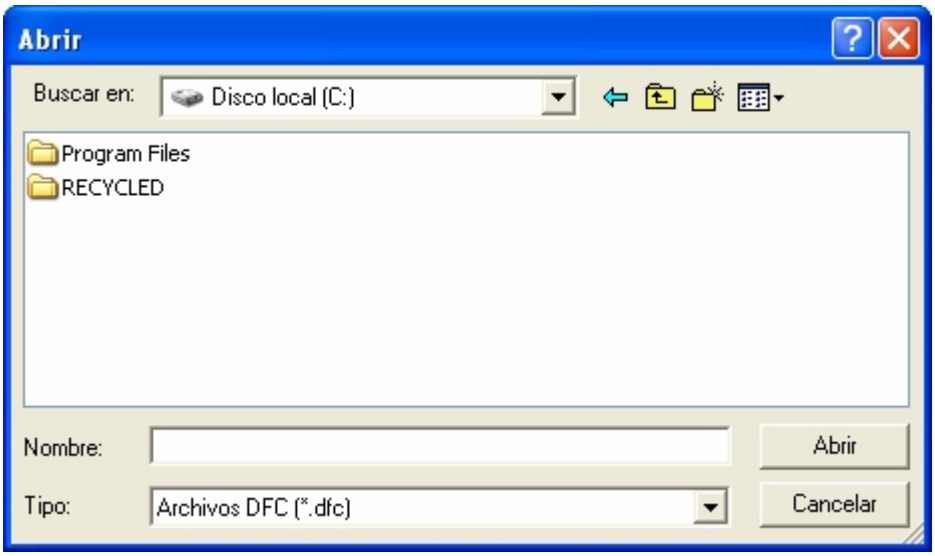

Figura 8. Cuadro de diálogo del comando abrir

Guardar

Este comando permite Guardar en cualquier unidad un diagrama de flujo. que se esta editando.

Formas de acceder al comando:

Barra de herramientas: 

Teclado: CTRL + G

Guardar Como

Este comando permite guardar en una unidad específica con un nuevo nombre un diagrama de flujo que se esta editando; mediante el despliegue del cuadro de diálogo del sistema.

La forma de acceder al comando se hace mediante el despliegue del menú Archivo (Click o Alt + A).

Al acceder a este comando, se muestra su cuadro de diálogo, para indicar el nombre y la ruta donde se desea guardar el archivo .dfc que se está trabajando.

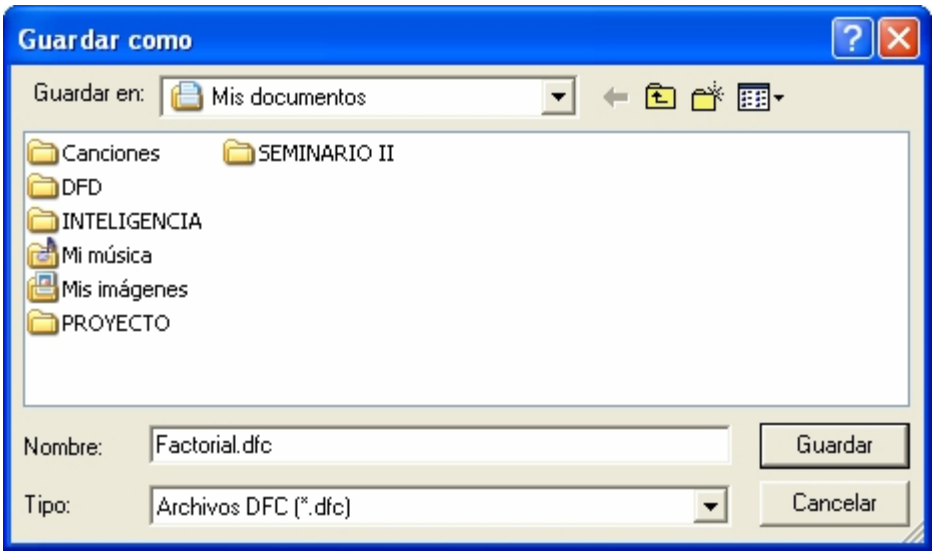

Figura 9. Cuadro de diálogo del comando guardar

Imprimir

Este comando permite Imprimir un diagrama de flujo mediante el despliegue del cuadro de diálogo de impresión del sistema.

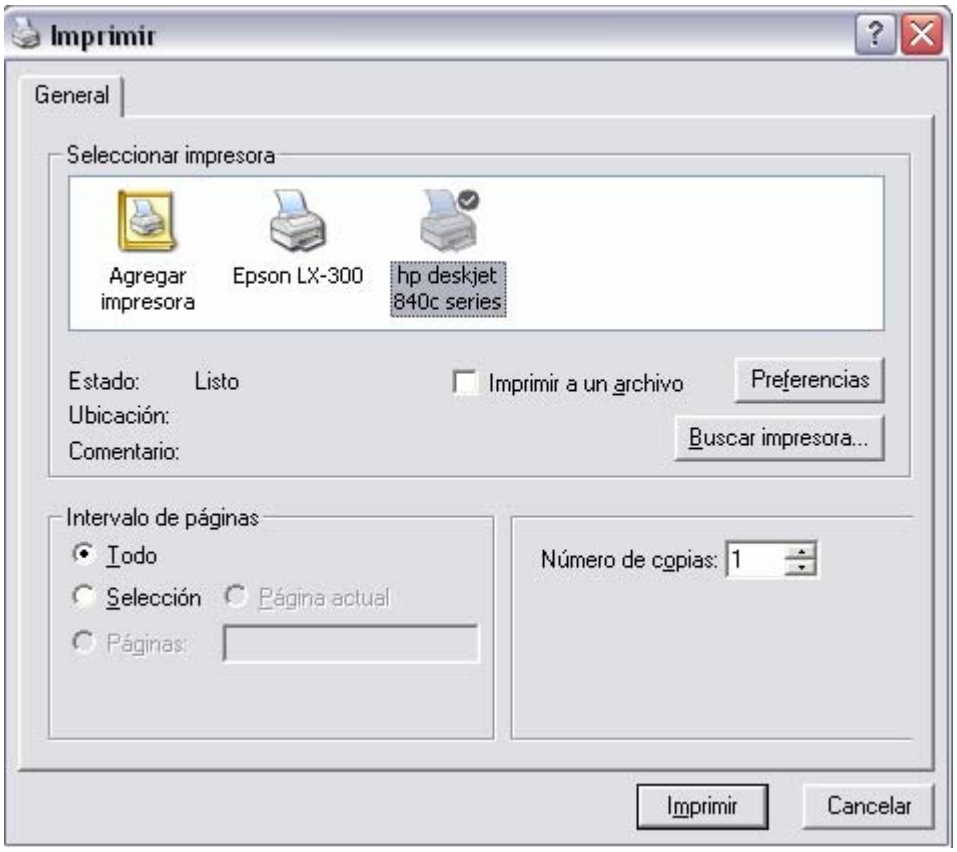

Figura 10. Cuadro de diálogo del comando imprimir

Formas de acceder al comando:

Barra de herramientas: 

Teclado: CTRL + P

Presentación Preliminar

Este comando permite Ver un diagrama de flujo en páginas completas, antes de Imprimir.

La forma de acceder al comando se hace mediante el despliegue del menú Archivo.

Configurar Impresora

Este comando permite elegir la impresora que se desee usar, mediante el despliegue del cuadro de diálogo del sistema.

La forma de acceder al comando se hace mediante el despliegue del menú Archivo.

Salir

Este comando permite abandonar la aplicación.

La forma de acceder al comando se hace mediante el despliegue del menú Archivo ó mediante el icono de cerrar de la ventana activa.

Menú Edición

Figura 11. Menú edición

Copiar

Este comando permite obtener una copia de un objeto seleccionado en el portapapeles.

Las estructuras que representan ciclos, son copiadas con todas las figuras que la contienen.

En el portapapeles es actualizado con un sólo objeto cada vez que se ejecute la acción copiar.

Formas de acceder al comando:

Barra de herramientas: 

Teclado: CTRL + C

Pegar

Este comando permite insertar lo que se encuentra en el portapapeles, delante del objeto seleccionado.

El portapapeles contiene aún el objeto que fue pegado, para poderlo pegar las veces que se desee.

Formas de acceder al comando:

Barra de herramientas: 

Teclado: CTRL + V

Cortar

Este comando se usa para quitar un objeto seleccionado de un diagrama de flujo, y se lleva al portapapeles.

Al cortar un objeto, este remplaza el contenido del portapapeles.

Las estructuras de ciclos son cortadas con todas las demás estructuras que la conforman.

Formas de acceder al comando:

Barra de herramientas: 

Teclado: CTRL + X

Eliminar

Este comando se usa para quitar un objeto seleccionado de un diagrama de flujo.

Las estructuras de ciclos son eliminadas con todas las demás estructuras que la conforman; si se trata de un subprograma, el comando a usar es Eliminar Subprograma.

Formas de acceder al comando:

Barra de herramientas: 

Teclado: Supr

Deshacer

Este comando Deshace la última acción de edición ejecutada.

Formas de acceder al comando:

Barra de herramientas: 

Teclado: CRTL + Z

Rehacer

Este comando rehace el último cambio de edición ejecutado.

Formas de acceder al comando:

Barra de herramientas: 

Teclado: CRTL + Y

Menú Objetos

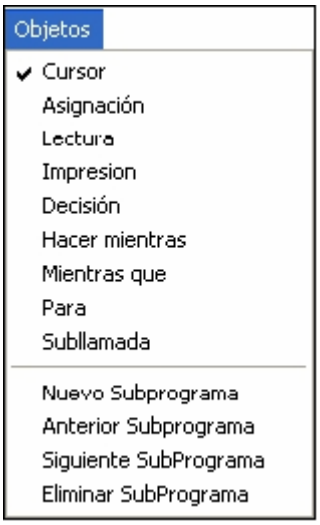

Figura 12. Menú objetos

Inicio

Inicio es el primer objeto al diseñar y al ejecutar un diagrama de flujo; este transfiere el control al siguiente objeto.

Figura 13. Objeto inicio

Fin

Fin es el último objeto al diseñar y al ejecutar un diagrama de flujo.

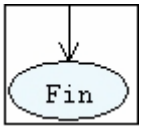

Figura 14. Objeto fin

Cursor

Este comando sirve para seleccionar un objeto determinado.

Formas de acceder al comando:

Mediante el despliegue del cuadro de diálogo del menú Objetos,

Barra de herramientas: 

Asignación

Inserta un objeto que permite asignar valores a las variables.

Formas de acceder al comando:

Mediante el despliegue del menú Objetos,

Barra de herramientas: 

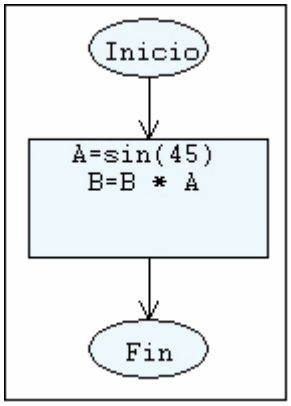

Figura 15. Diagrama con objeto Asignación

El objeto Asignación se puede editar mediante su cuadro de diálogo, haciendo doble click sobre él o presionando enter siempre y cuando esté seleccionado.

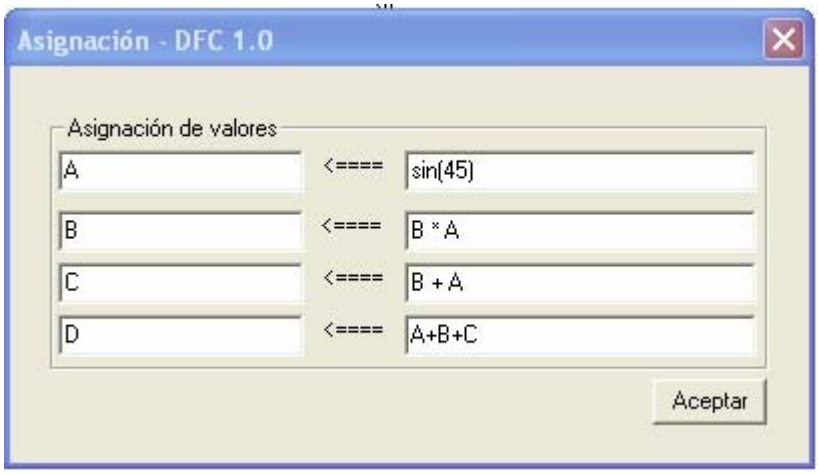

Figura 16. Cuadro de diálogo del objeto Asignación

En el cuadro de diálogo del objeto asignación, se pueden realizar hasta cuatro asignaciones. Cada asignación consta de un espacio para el nombre de la variable, situado a la izquierda, y a su respectiva derecha se encuentra el campo para la expresión; Lo que indica que a la variable se le asigna el resultado de la expresión que a su derecha contiene. Este cuadro de diálogo debe contener al menos una asignación, y si el usuario necesita más de cuatro asignaciones, es necesario que inserte otro objeto de asignación.

Lectura

Inserta un objeto que permite la lectura de datos.

Formas de acceder al comando:

Mediante el despliegue del menú Objetos,

Barra de herramientas: 

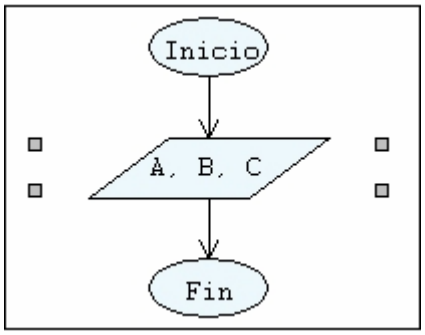

Figura 17. Diagrama con objeto Lectura

El objeto Lectura se puede editar mediante su cuadro de diálogo, haciendo doble click sobre él o presionando enter siempre y cuando esté seleccionado.

Este bloque puede contener una o más variables, separadas por comas, las cuales se pueden alimentar una por una.

Las variables deben contener valores numéricos constantes.

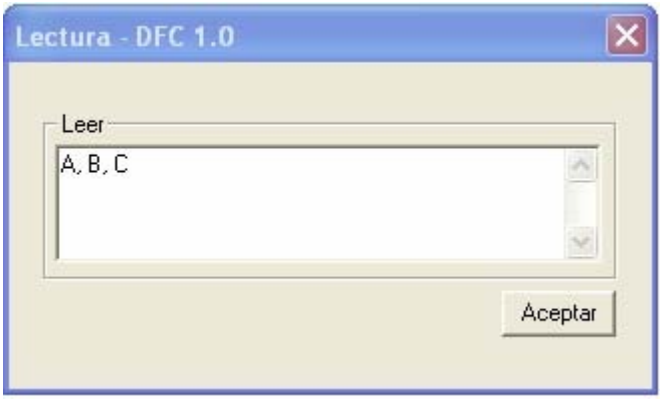

Figura 18. Cuadro de diálogo del objeto Lectura

Impresión

Inserta un objeto que permite visualizar resultados y leyendas. Si se desea imprimir el resultado de varias variables, estas deben ir concatenadas con el operador &

Formas de acceder al comando:

Barra de herramientas:

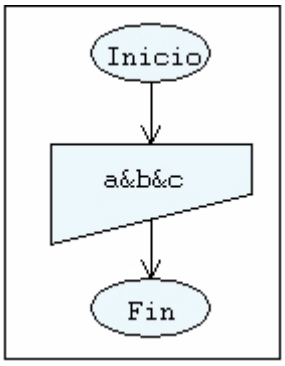

Figura 19. Diagrama con objeto Impresión

El objeto Impresión se puede editar mediante su cuadro de diálogo, haciendo doble click sobre él o presionando enter siempre y cuando esté seleccionado.

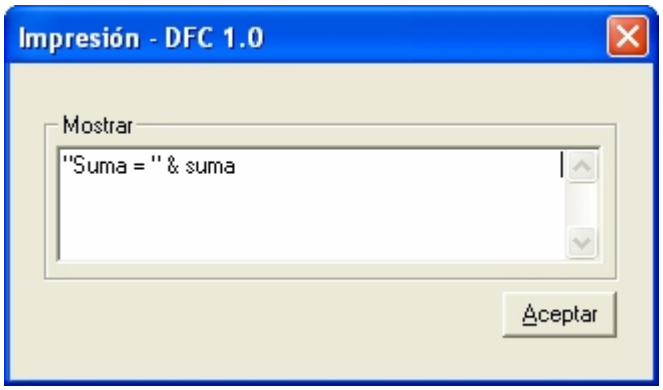

Figura 20. Cuadro de diálogo del objeto Impresión

De la misma forma en que el bloque de asignación puede operar con valores constantes o con expresiones compuestas, el bloque de Impresión puede mostrar estos formatos o una mezcla de números y caracteres, usando el operador de concatenación '&'.

Decisión

Inserta un objeto que permite realizar unas u otras acciones dependiendo del valor de la condición.

Formas de acceder al comando:

Mediante el despliegue del menú Objetos,

Barra de herramientas: 

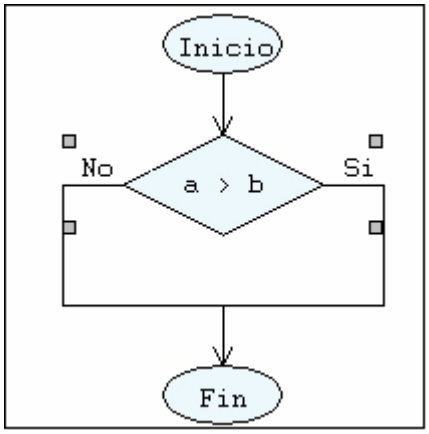

Figura 21. Diagrama con objeto Decisión

El objeto Decisión se puede editar mediante su cuadro de diálogo, haciendo doble click sobre él o presionando enter siempre y cuando esté seleccionado.

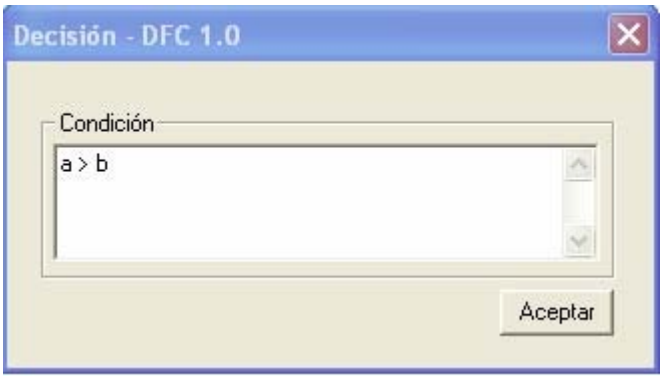

Figura 22. Cuadro de diálogo del objeto Decisión

El bloque Decisión se usa con operandos y operadores. Los operandos pueden ser cualquier variable o valor sobre los cuales se quiere representar una relación; el resultado de la expresión debe ser un tipo de dato lógico si se usan operadores lógicos, de lo contrario se considera verdadera si el resultado de la expresión es diferente de cero.

Ejemplos:

a >(3*sin(45))

a=4*8

3<=A

a!=b%c

El flujo a seguir del objeto Decisión, esta ligado al resultado de la condición; por ello cuenta con dos bloques, uno a su derecha en caso de ser verdadera la expresión y en caso contrario cuenta con el bloque de izquierda; y por último, en ambos casos encuentra el cierre del bloque.

Bloques de Repetición

Son los encargados de realizar ciclos al interior de un algoritmo; se encuentran de tres formas: ciclo para, ciclo mientras que, y el ciclo hacer mientras.

Hacer Mientras

Inserta un objeto que permite realizar acciones mientras la condición sea verdadera.

Este bloque primero realiza las acciones y luego evalúa la condición.

Formas de acceder al comando:

Barra de herramientas: 

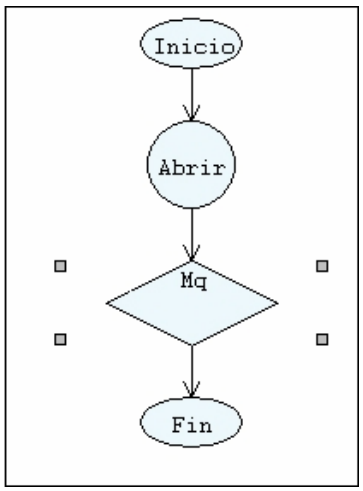

Figura 23. Diagrama con objeto Hacer Mientras

El objeto Hacer Mientras se puede editar mediante su cuadro de diálogo, haciendo doble click sobre él o presionando enter siempre y cuando esté seleccionado.

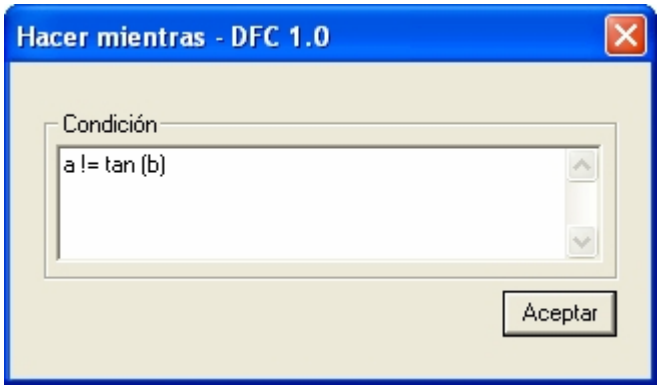

Figura 24. Cuadro de diálogo del objeto Hacer Mientras

El bloque de repetición Hacer Mientras, ejecuta un bloque de objetos mientras la condición sea verdadera.

Consta de un objeto Abrir, para indicar desde donde comienza el bloque que le corresponde, y finaliza donde se encuentra la condición.

Mientras Que

Inserta un objeto que permite realizar acciones mientras la condición sea verdadera.

Este bloque primero evalúa la condición y luego realiza las acciones.

Formas de acceder al comando:

Mediante el despliegue del menú Objetos,

Barra de herramientas: 

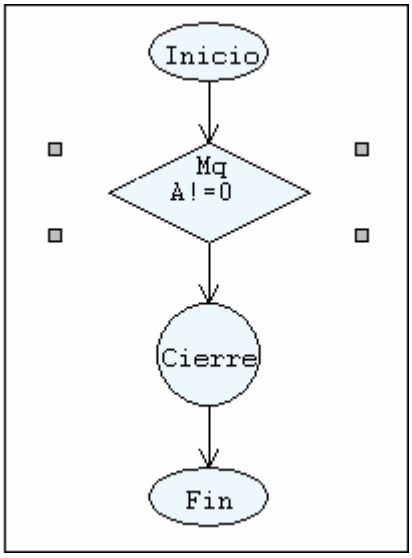

Figura 25. Diagrama con objeto Mientras Que

El objeto Mientras Que se puede editar mediante su cuadro de diálogo, haciendo doble click sobre él o presionando enter siempre y cuando esté seleccionado.

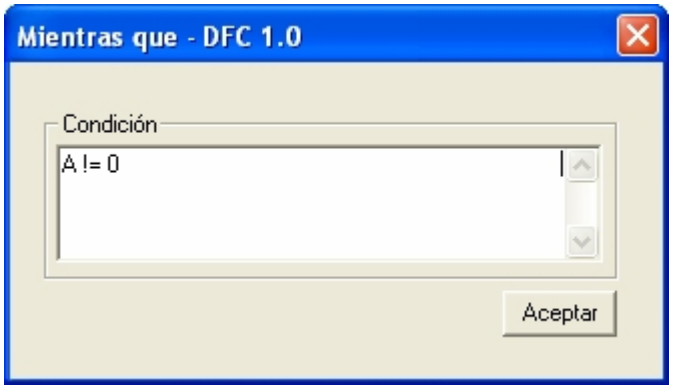

Figura 26. Cuadro de diálogo del objeto Mientras Que

El bloque de repetición Mientras Que, ejecuta un bloque de objetos mientras la condición sea verdadera.

Consta de un objeto Cierre, para indicar donde termina el bloque que le corresponde, e inicia donde se encuentra la condición.

Para

Inserta un objeto que permite realizar acciones hasta que una variable alcanza su limite.

El ciclo Para se compone de tres condiciones: un inicio, una salida y un incremento de una determinada variable en mención.

Formas de acceder al comando:

Barra de herramientas: 

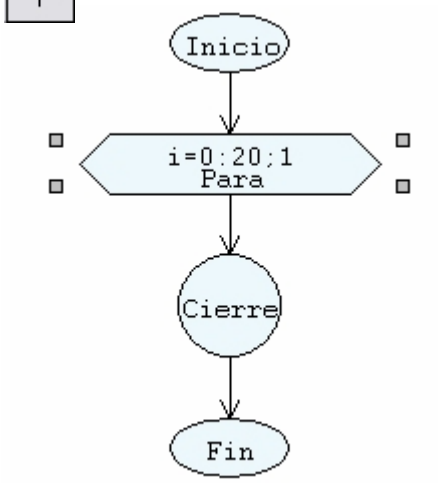

Figura 27. Diagrama con objeto Para

El objeto Para se puede editar mediante su cuadro de diálogo, haciendo doble click sobre él o presionando enter siempre y cuando esté seleccionado.

Figura 28. Cuadro de diálogo del objeto Para

El bloque de repetición Para, ejecuta un bloque de objetos mientras la variable contador alcanza su limite, establecido por la condición de salida del ciclo.

La variable contador debe tener un tipo de dato numérico. Lleva su bloque de cierre, el cual le indica hasta donde es su campo de acción, e inicia donde se encuentra la condición.

El ciclo Para contiene un valor inicial que será asignado a la variable contador al iniciar la ejecución del ciclo, un valor final para salir del ciclo y un valor de incremento para alcanzar el valor final; así se ejecuta el cuerpo del ciclo mientras la variable contador excede el valor final y la ejecución continua en el objeto siguiente del objeto Cierre del ciclo Para.

Subllamada

Inserta un objeto para invocar a un Subprograma.

El objeto Subllamada debe contener un nombre y unos argumentos (si existen) que se pasan al Subprograma; el tipo de argumentos debe coincidir con los parámetros del Subprograma.

Los parámetros que se pasen al procedimiento, siempre serán por valor, es decir no serán alterados por el Subprograma, si se quiere que estos sean alterados, es decir paso por referencia se debe anteponer un * al nombre del parámetro correspondiente.

Formas de acceder al comando:

Mediante el despliegue del menú Objetos,

Barra de herramientas: 

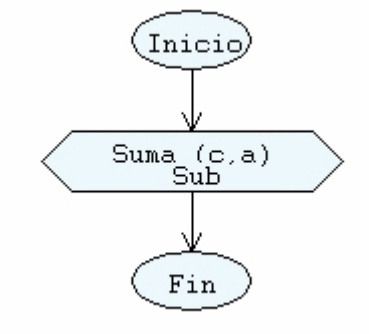

Figura 29. Diagrama con objeto Subllamada

El objeto Subllamada se puede editar mediante su cuadro de diálogo, haciendo doble click sobre él o presionando enter siempre y cuando esté seleccionado.

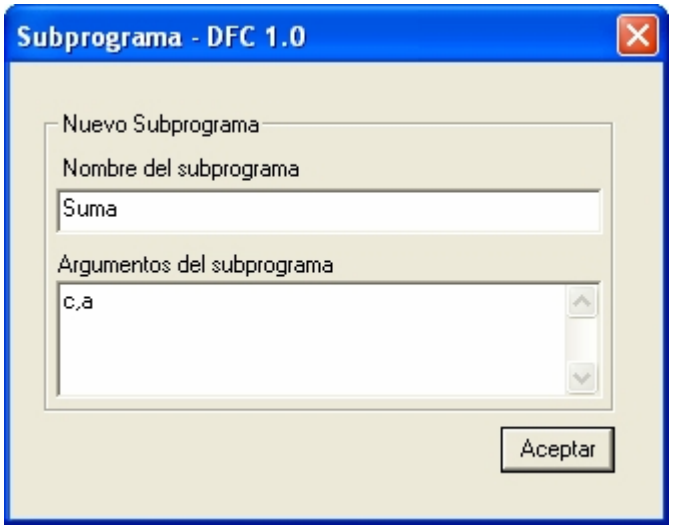

Figura 30. Cuadro de diálogo del objeto Subllamada

En el cuadro de diálogo de Subllamada, se debe insertar el nombre del Subprograma que está invocando, con sus respectivos argumentos (si existen) separados por comas y escritos sin paréntesis.
El nombre de un Subprograma debe ser único y debe comenzar por una letra seguida de letras, números. Una vez ejecutado el objeto Subllamada, se

procede a ejecutar el Subprograma y retorna para seguir ejecutando el objeto siguiente a Subllamada.

Nuevo Subprograma

Inserta un nuevo Subprograma en una nueva ventana. El objeto Nuevo Subprograma debe contener un nombre y unos parámetros que deben coincidir con los argumentos de la Subllamada.

Formas de acceder al comando:

Mediante el despliegue del menú Objetos,

Barra de herramientas:

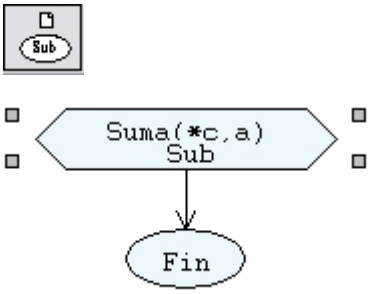

Figura 31. Diagrama con objeto Subprograma

El objeto Nuevo Subprograma se puede editar mediante su cuadro de diálogo, haciendo doble click sobre él o presionando enter siempre y cuando esté seleccionado.

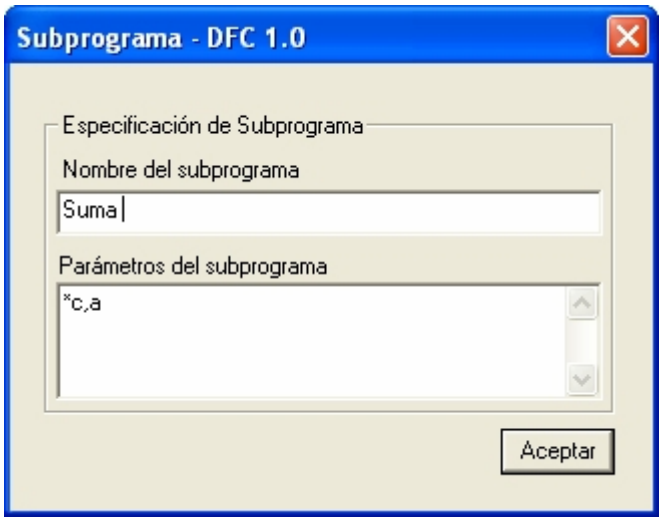

Figura 32. Cuadro de diálogo del objeto Subllamada

El cuadro de diálogo del objeto Subprograma contiene un espacio para el nombre del subprograma y un espacio para los parámetros. Estos parámetros (si existen) deben estar separados por comas. El nombre de un subprograma debe comenzar por una letra seguida de letras, números y debe ser único.

Anterior Subprograma

Muestra el Subprograma inmediatamente anterior.

El objeto Anterior Subprograma sirve para la manipulación hacia atrás de las diferentes ventanas donde se encuentra cada Subprograma.

Formas de acceder al comando:

Mediante el despliegue del menú Objetos,

Barra de herramientas:

Siguiente Subprograma

Muestra el Subprograma inmediatamente consecutivo.

El objeto Siguiente Subprograma sirve para la manipulación hacia adelante de las diferentes ventanas donde se encuentra cada Subprograma.

Formas de acceder al comando:

Mediante el despliegue del menú Objetos,

Barra de herramientas:

Eliminar Subprograma

Elimina el Subprograma actual.

Formas de acceder al comando:

Mediante el despliegue del menú Objetos,

Barra de herramientas:

Menú Ver

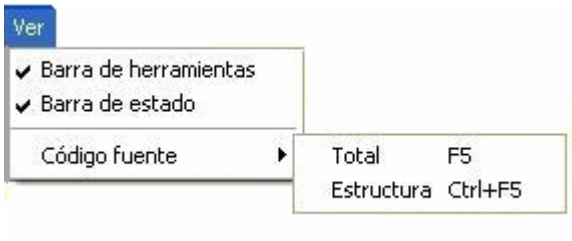

Figura 33. Menú Ver

Barra de Herramientas

Este comando sirve para mostrar u ocultar la barra de herramientas.

Figura 34. Barra de Herramientas

Formas de acceder al comando:

Mediante el despliegue del menú Ver.

La barra de herramientas se ubica debajo de la barra de menús, en ella se visualizan iconos de pulsación que proporcionan una forma fácil de de manipulación: añadir objetos al diagrama, editarlo, ejecutarlo y mostrar el código fuente.

Cada icono de la barra de herramientas tiene una forma equivalente de selección con el teclado, por tanto su uso es opcional. Al situar el puntero del ratón sobre la herramienta, aparece desplegado su nombre.

Barra de Estado

Este comando sirve para mostrar u ocultar la barra de estado de DFC.

Figura 35. Menú Ver

Formas de acceder al comando:

Mediante el despliegue del menú Ver.

La barra de estado se ubica en la parte inferior de la ventana de edición, está dividida en varios paneles: el primero de ellos se usa para visualizar mensajes de ayuda al usuario, el cual aparece cuando se sitúa el puntero del ratón sobre una herramienta de la barra de herramientas, mostrando una breve descripción sobre su función y los otros paneles sirven para mostrar distintos indicadores de estado como activación del bloqueo de mayúsculas, del bloqueo de números y del bloqueo de desplazamiento.
La barra de entrada no acepta manipulación por parte del usuario.

Código Fuente

DFC 1.0 genera el código fuente del diagrama desarrollado en lenguaje C++, este código puede ser mostrado junto con el diagrama y también puede ser llevado a disco. De igual forma puede generar un fragmento del código correspondiente a un bloque del diagrama. Es un código cien por ciento compilable y no genera ningún tipo de warning o error.

Este comando permite escoger la forma de generación del código fuente, ya sea total o de una sola estructura que este seleccionada en el diagrama de flujo.

Total

Este comando sirve para generar el código fuente en lenguaje C++, de todo un diagrama de flujo

Formas de acceder al comando:

Mediante el despliegue del menú Ver.

Teclado: F5

Estructura

Este comando sirve para generar el código fuente en lenguaje C++, de una estructura seleccionada en un diagrama de flujo; permitiendo reconocer la sintaxis de cada parte del diagrama.

Formas de acceder al comando:

Mediante el despliegue del menú Ver.

Teclado: Ctrl + F5

Menú Ejecución

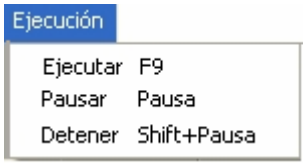

Figura 36. Menú Ver

Una vez construidos los diagramas, se puede probar, si el algoritmo implementado responde a las necesidades del problema mediante la ejecución de la lógica del diagrama; ésta se puede desarrollar de manera completa o se puede optar por pausarla o detenerla.

Ejecutar

Este comando sirve para ejecutar la lógica del diagrama hasta el final del mismo.

Formas de acceder al comando:

Barra de herramientas: 

Mediante el despliegue del menú Ejecución.

Teclado: F9

Pausar

Este comando permite hacer una pausa a medida que se ejecuta la lógica del diagrama.

Formas de acceder al comando:

Barra de herramientas: 

Mediante el despliegue del menú Ejecución.

Teclado: Pausa

Detener

Este comando sirve para paralizar la lógica del diagrama en un punto dado.

Formas de acceder al comando:

Barra de herramientas: 

Mediante el despliegue del menú Ejecución.

Teclado: Shift + F5

Menú Depuración

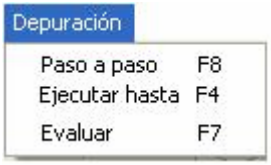

Figura 37. Menú Ver

Permite analizar la lógica de manera detallada, según las opciones definidas: paso simple, ejecutar hasta y evaluar.

Paso a Paso

Este comando sirve para ejecutar la lógica del diagrama instrucción por instrucción.

Formas de acceder al comando:

Barra de herramientas: 

Mediante el despliegue del menú Depuración.

Teclado: F8

Ejecutar Hasta

Este comando permite ejecutar la lógica del diagrama hasta un punto estipulado.

Formas de acceder al comando:

Barra de herramientas: 

Mediante el despliegue del menú Depuración.

Teclado: F4

Evaluar

Este comando facilita observar y modificar el valor de una determinada expresión, mediante su cuadro de diálogo.

Figura 38. Cuadro de diálogo del evaluador de expresiones

Formas de acceder al comando:

Mediante el despliegue del menú Depuración.

Teclado:F7

Precedencia de Funciones y Operadores

Las funciones matemáticas tienen todas la misma prioridad, se evalúan de izquierda a derecha.

Cuando hay una expresión con varios operadores, se evalúan primero los matemáticos y luego los operadores lógicos relacionales.

Los operadores lógicos y relacionales tienen todos la misma prioridad, se evalúan de izquierda a derecha. Esto se hace con la intención de motivar al programador al uso de los paréntesis en las expresiones lógicas. De esta forma a>b||c>b se considera una expresión no válida para estimular a usarla así: (a>b)||(c>b)

Los operadores matemáticos tienen la siguiente precedencia:

Mayor

+ y - monarios

^

* / %

+ -

=

Menor

Funciones matemáticas derivadas:

La siguiente es una lista de funciones matemáticas no implementadas que pueden derivarse de funciones matemáticas que se implementaron:

Función	**Derivadas equivalentes**
Secante(X)	1 / COS(X)
Cosecante(X)	1 / SIN(X)
Cotangente(X)	1 / TAN(X)
Cotangente inversa(X)	ATAN(X) + 2 * ATAN(1)
Seno hiperbólico(X)	(EXP(X) – EXP(–X)) / 2
Coseno hiperbólico(X)	(EXP(X) + EXP(–X)) / 2

Tangente hiperbólica(X)	(EXP(X) – EXP(–X)) / (EXP(X) + EXP (–X))
Secante hiperbólica(X)	2 / (EXP(X) + EXP(–X))
Cosecante hiperbólica(X)	2 / (EXP(X) – EXP(–X))
Cotangente hiperbólica(X)	(EXP(X) + EXP (–X)) / (EXP (X) – EXP (–X))
Seno hiperbólico inverso(X)	LN(X + SQRT(X * X + 1))
Coseno hiperbólico inverso(X)	LN(X + SQRT(X * X – 1))
Tangente hiperbólica inversa(X)	LN((1 + X) / (1 – X)) / 2
Secante hiperbólica inversa(X)	LN ((SQRT(–X * X + 1) + 1) / X)
Cotangente hiperbólica inversa(X)	LN ((X + 1) / (X – 1)) / 2
Logaritmo en base N(X)	LN (X) / LN (N)

Tabla 1. Funciones matemáticas derivadas

Tablas de verdad de operadores lógicos:

La clave de los conceptos de operadores relacionales y lógicos es la idea de cierto o falso. Como la finalidad de DFC es la codificación en lenguaje C++, se toma cualquier valor distinto de cero como verdadero y falso en caso contrario.

Operación AND lógica:

Sintaxis: X && Y

Entrada X,Y

X	Y	&&
v	v	v
v	f	f
f	v	f
f	f	f

Tabla 2. Tabla de verdad del operador AND

Operación OR lógica:

Sintaxis: X || Y

Entrada X,Y

X	Y	||
v	v	v
v	f	v
f	v	v
f	f	f

Tabla 3. Tabla de verdad del operador OR

Operación Negación OR lógica:

Sintaxis: X !| Y

Entrada X,Y

X	Y	!|
v	v	f
v	f	f
f	v	f
f	f	v

Tabla 4. Tabla de verdad del operador negación OR

Negación del AND:

Sintaxis: X !& Y

Entrada X,Y

X	Y	!&
v	v	f

v	f	v
f	v	v
f	f	v

Tabla 5. Tabla de verdad del operador negación del AND

Negación lógica:

Sintaxis: NOT (X)

Entrada X

X	**NOT**
v	f
f	v

Tabla 6. Tabla de verdad del operador NOT

Listado de Palabras Reservadas:

Palabras reservadas de DFC en la ejecución de los diagramas:

NOT	INC	DEC	RANDOM	ROUND	TRUNC	ABS	LN
EXP	SQRT	SIN	COS	TAN	ASIN	ACOS	ATAN

Tabla 7. Tabla de palabras reservadas en la ejecución

Palabras reservadas por DFC en la generación de código:

Estas palabras reservadas son las mismas de turbo C.

AUTO	BREAK	CASE	CHAR
DO	DOUBLE	ELSE	ENUM
GOTO	IF	INT	LONG
SIGNED	SIZEOF	STATIC	STRUCT

UNSIGNED	VOID	VOLATILE	WHILE
VIRTUAL	CATCH	NEW	TEMPLATE
DELETE	PRIVATE	THROW	FRIEND
CONST	EXTERN	REGISTER	SWITCH
MAIN	CLASS	PROTECTED	CONTINUE
FLOAT	RETURN	TYPEDEF	INLINE
OPERATOR	TRY	DEFAULT	FOR
SHORT	UNION	PUBLIC	THIS

Tabla 8 Tabla de verdad de palabras reservadas en la codificación

Listado de Errores:

Se presentan errores en el **tiempo de revisión**, el cual ocurre cuando se intenta cambiar de la acción de edición del diagrama a cualquier otra; este tipo de errores suelen ser sintácticos o errores en los argumentos o valores no definidos en operaciones matemáticas, entre otros.

- Ejemplos:
  a=6/0
  ASIN(500)
  TAN 60)
  SQRT(-9)

Si se detecta un error, se muestra el mensaje correspondiente y se enfoca la figura que lo produjo.

En el **tiempo de ejecución** del diagrama se pueden presentar errores en el algoritmo, en tal caso se suspende la ejecución, enfocando la figura que contiene el error.

- Ejemplos:
  a=b/c cuando c se inicializa en cero.

La figura debe tener algún valor o condición.

Este error se presenta al momento de revisión, cuando no se ha editado una figura por medio de su cuadro de diálogo.

Se esta usando una palabra reservada como nombre de variable.

Ver palabras reservadas.

Los nombres de las variables deben empezar por una letra (a..z , A..Z).

Este error se presenta al momento de revisión, cuando se ha declarado una variable cuyo nombre empieza por un caracter diferente de letra. Las letras iniciales para las variables son del alfabeto occidental y pueden ir en minúsculas o mayúsculas.

Los nombres de las variables solo pueden contener letras (a..z , A..Z) y dígitos (0..9).

Este error se presenta al momento de revisión, cuando se ha declarado una variable cuyo nombre posee por lo menos un carater diferente de letra del alfabeto occidental o digito.

Los nombre de las variables que contienen espacios, son manipuladas por DFC como si no los tuviera.

Ejemplo:

nom bre, nom bre, nom bre

Son tomados por DFC como si la variable fuese nombre.

Debe haber al menos un caracter diferente de espacio después del signo =.

Este error se presenta al momento de revisión, en la figura Asignación cuando se ha declarado una variable y después del operador = se ha dejado el campo vacío.

Ejemplo: contador =

Si a la variable se le es asignado un valor pero inmediatamente después del operador = hay espacios en blanco, DFC suprime estos espacios. Ejemplo:
contador= a, DFC la muestra así: contador=a.

Paréntesis no emparejados.

Este error se presenta al momento de revisión, cuando se han usado mal los paréntesis, por ejemplo no es cerrado un paréntesis que fue abierto en una expresión o viceversa.

Ejemplo:
5/(7*6 ó 5/ 7-6)

No hay expresión.

Este error se presenta cuando se pretende evaluar una cadena vacía.

Se pretende usar una variable que no ha sido inicializada.

Este error se presenta cuando se intenta usar una variable que no le ha sido asignado un valor; para solucionar el problema, a la variable se le debe dar algún valor en el objeto asignación.

No está determinada la división sobre cero.

Este error se presenta cuando se intenta realizar una división sobre cero; ya sea por asignación directa o porque la variable del dividendo esta inicializada en cero.

Ejemplo:

a=5/0

a=0; b=8/a

Los argumentos de las funciones deben estar dentro de paréntesis.

Este error se presenta cuando se usan las funciones matemáticas y sus argumentos no se encuentran entre paréntesis.

Ejemplo:

SIN 90

El formato numérico solo acepta dígitos (0..9) y un separador (.) para las fracciones.

Este error se presenta cuando se usan un valor decimal con coma o doble punto, o simplemente un número con caracteres diferentes a los dígitos del 0 al 9.

Ejemplo:

35,2 35..2 a=98v98

El logaritmo natural no esta definido para los números menores o iguales a cero (0).

Este error se presenta cuando se usa la función logaritmo natural y como argumento tiene un número negativo o el cero.

Ejemplo:

LN(0) LN(-3)

La raíz cuadrada no está definida para números negativos.

Este error se presenta cuando se usa la función SQRT y como argumento se introduce un número negativo.

Ejemplo:

SQRT(-5)

La función ASIN solo acepta valores entre -1 y 1.

Este error se presenta cuando se usa la función arco seno y como argumento se introduce un número fuera del intervalo cerrado [-1,1].

Ejemplo:

ASIN(-5) ASIN(0) ASIN(10)

La función ACOS solo acepta valores entre -1 y 1.

Este error se presenta cuando se usa la función arco coseno y como argumento se introduce un número fuera del intervalo cerrado [-1,1].

Ejemplo:

ACOS(-5) ACOS(0) ACOS(10)

Los subíndices de los arreglos deben ser mayores o iguales a cero (0).

Este error se presenta cuando se usan vectores o matrices cuyos subíndices no son enteros positivos.

Ejemplo:

Vector[-3] Matriz[-1][2]

Se pretenden usar posiciones de un arreglo que no han sido inicializadas.

Este error se presenta al trabajar con una matriz o un vector en cuyas posiciones no se ha asignado ningún valor. Para solucionar este error es necesario insertar en el objeto asignación el vector en sus respectivas posiciones y asignarle un valor.

Error en el uso de corchetes en el arreglo.

Este error se presenta por el mal uso de corchetes [ ] al trabajar con una matriz o un vector.

Ejemplo:

vector[[1]=2, Vector[[1]]=0, Matriz[[1,2]]

la forma correcta de usar los corchetes para arreglos es la siguiente:

vector[1]=2, Matriz[2][1]=0

Error en la dimensión del arreglo.

Este error se presenta al trabajar con una variable inicializada como matriz o vector y posteriormente no es llamada como arreglo y lo contrario, cuando se inicializa un variable y luego es llamada como arreglo.

Ejemplo:

vector[1]=2; Vector=0

M=0; M[1][3]=2

Los subíndices de los arreglos deben ser números enteros.

Este error se presenta al trabajar con matrices o vectores cuyos subíndices son valores decimales.

Ejemplo:

vector[1.2]=2

Los arreglos no son permitidos como parámetros de subllamadas. Estos son tratados como variables por referencia sin importar su vinculación con la subllamada.

Este error se presenta al trabajar el objeto subllamada, usando como parámetro un vector o una matriz.

Ejemplo:

ProcesoUno(a,v[8],b)

Si un arreglo esta definido en un programa principal y este es modificado en el subprograma, los cambios se ven reflejados en el programa principal ya que son tratados como variables por referencia.

El subprograma que se quiere invocar no ha sido declarado.

Este error se presenta al trabajar el objeto subllamada, sin haber insertado el objeto subprograma con sus respectivas especificaciones. De tal manera que se está llamando a un subprograma que aún no existe.

Se requiere un nombre para el procedimiento.

Este error se presenta al trabajar el objeto subprograma, cuando aún no se le ha identificado con un nombre.

El número de argumentos debe coincidir con el número de parámetros de la subllamada.

Este error se presenta al trabajar el objeto subprograma, cuando la cantidad de argumentos especificados, no corresponden a la cantidad de parámetros de la subllamada que lo invocó.

Ejemplo:
Subllamada: Uno(a,b,3,d)
Subprograma: Uno(*a,b,c)

Existe un valor numérico que se quiere pasar a un parámetro por referencia.
Este error se presenta al trabajar el objeto subprograma, cuando existe un argumento por referencia el cual fue especificado como valor numérico en los parámetros de la subllamada.

Ejemplo:
Subllamada: Uno(a,b,3)
Subprograma: Uno(*a,b,*c)

Existen dos o más subprogramas con el mismo nombre.
Este error se presenta al trabajar con varios subprogramas, cuando se les ha identificado con el mismo nombre.

El parámetro ya fue definido.
Este error se presenta al repetir un parámetro de un subprograma.

Ejemplo:
Subprograma: Uno(*a,b,b)

ANEXO B. PRUEBAS DEL SOFTWARE

Dentro de la metodología de entrega por etapas, se planean pruebas especificas al momento de concluir cada etapa, esto permite que no haya propagación de errores entre cada una de las etapas del proyecto.

Las pruebas se acostumbran a realizar con los ítem que pueden representar riesgos al momento de su implementación. Si hacemos un recorrido por DFC hallaremos las siguientes pruebas:

- Etapa 1: Construcción de figuras. Énfasis: manejo del objeto decisión.
- Etapa 2: Validación y edición. Énfasis: rehacer y deshacer .
- Etapa 3: Ejecución de los diagramas. Énfasis: ejecutar paso a paso y funciones recursivas.
- Etapa 4: Almacenamiento de la información. Énfasis: lectura de archivos para cargar los diagramas.
- Etapa 5: Generación de código. Énfasis: generar código de los procedimientos y vincular las variables.

Así se aprecia que énfasis se le hizo a cada etapa, sin embargo cuando el producto estuvo completo se implementaron pruebas que contenían los tópicos mencionados anteriormente, con el animo de mirar si era perfecto el acople de las cinco etapas.

Las pruebas que se escogieron para el producto final, fueron procedimientos recursivos con parámetros por valor y por referencia, y programas donde se daban manejo de vectores. Se encontraron una serie de errores que no

involucraban las cuatro primeras etapas, sino errores en la generación de código. Los errores mas frecuentes fueron la vinculación de parámetros de los procedimientos en la declaración de las variables del respectivo procedimiento, así los parámetros eran definidos dos veces, el otro error de consideración era la declaración de arreglos como variables normales.

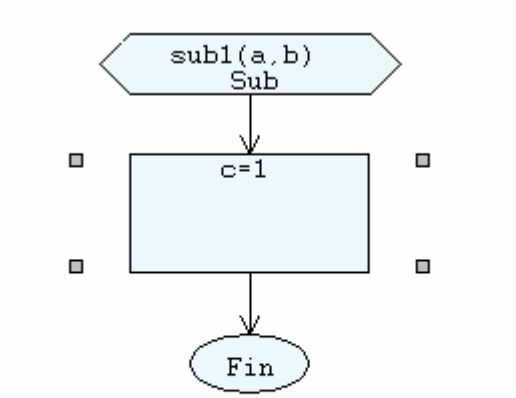

Figura 1. Corrección del error declaración de parámetros

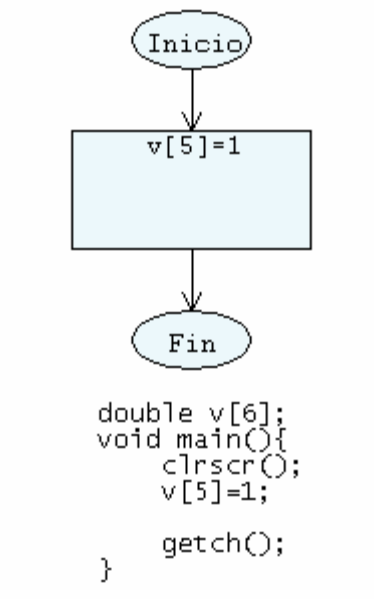

Figura 2. Corrección del error declaración de arreglos

Todos los errores fueron corregidos y revisados por personas ajenas a los autores, principalmente colaboraron en las pruebas estudiantes que cursan y que cursaron introducción a los computadores y estructuras computacionales.

Todas las correcciones y pruebas anteriores nos permiten entregar un producto confiable y que satisface todos los objetivos planteados.