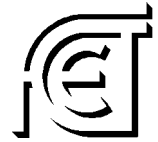




Facultad de Ciencias Exactas y Tecnologías
Universidad Nacional de Santiago del Estero



Implementación de un Intérprete Conversacional para Operaciones con Números Fraccionarios

Categoría: Trabajos de cátedra
Área: Algoritmos, lenguajes y programación

Autores:

Correa, Liliana

FerML_1@hotmail.com

Galeano, Luciano Jesús

galeluje@gmail.com

Leguizamòn, Fernanda Magdalena

fernadamagdalena@gmail.com

Docentes a Cargo:

Ms. Ing. Graciela Barchini de Jiménez

grael@unse.edu.ar

Ms. Ing. Margarita Alvarez de Benítez

malvarez@unse.edu.ar

Ing. Mario Montalvetti

mmontalv@unse.edu.ar

IMPLEMENTACIÓN DE UN INTÉRPRETE CONVERSACIONAL PARA OPERACIONES CON NÚMEROS FRACCIONARIOS

RESUMEN

La representación de los números fraccionarios en la computadora se realiza como el cociente de la división de dos números enteros. Esto ocasiona muchos problemas ya que una fracción simple no siempre tiene una representación decimal exacta.

En la aritmética de punto fijo: $5/3$ es igual a 1 (por el redondeo por truncamiento), en cambio en aritmética de punto flotante $5/3$ es igual a 1.666 ó a 1.667 (con una precisión de 3 dígitos).

Por otra parte una fracción simple no tiene representación decimal exacta: por ejemplo, $1/3$ se representa como una sucesión infinita de números 3. Además muchas fracciones que tienen una representación finita en algún sistema numérico no la tienen en otro sistema. Ejemplo: $1/10$ en el sistema decimal es 0.1 y en el sistema binario adquiere una forma repetitiva infinita: 0.000110011001100....

Por la naturaleza necesaria aproximada de la representación, desde el ingreso y posterior tratamiento de los números fraccionarios surgen errores de distinto tipo, por redondeo simétrico y truncamiento, errores absolutos y errores relativos. La propagación de errores en los cálculos no permiten predecir la confiabilidad de los resultados. Es necesario implementar un mecanismo que permita representar y tratar los números fraccionarios como tales para optimizar la exactitud y precisión de los resultados.

En este trabajo se diseña y desarrolla un intérprete NFrac para trabajar con números fraccionarios, las principales funciones que se implementan son: ingreso de datos e impresión (por pantalla) de los resultados de diversas operaciones tales como operaciones algebraicas, pasaje de un número fraccionario a decimal y viceversa y determinación del máximo y mínimo de una secuencia de fracciones ingresadas. Además, NFrac puede ser usado con fines didácticos.

Para el desarrollo de NFrac, se utilizaron generadores automáticos para el análisis léxico y sintáctico y la programación de las operaciones especificadas se realizó en el lenguaje C++ Builder 5.

La realización de este intérprete nos permitió sistematizar los conocimientos teóricos y metodológicos adquiridos en la asignatura. Además pudimos detectar y solucionar los problemas que se producen al implementar un intérprete.

1. INTRODUCCIÓN

En la asignatura Programación II de la carrera de Licenciatura en Sistemas de Información, se abordan temas pertenecientes a la Teoría de Algoritmos, Teoría de Lenguajes Formales y Autómatas. Esta asignatura proporciona la justificación teórica de los instrumentos necesarios para la construcción de compiladores.

Durante el segundo cuatrimestre del año académico 2004, la cátedra solicitó en el cuarto taller el diseño y desarrollo de un intérprete que realice operaciones con números fraccionarios, mediante la aceptación de ciertos comandos por pantalla y la devolución, también por pantalla, del resultado de efectuar las operaciones especificadas por dichos comandos. Se utilizaron generadores automáticos para el análisis léxico y sintáctico y la programación para las operaciones especificadas se realizó en el lenguaje C++ Builder 5.

Planteamiento del problema

La representación de los números fraccionarios en la computadora se almacena como el cociente de la división de dos números enteros. Esto ocasiona muchos problemas ya que una fracción simple no siempre tiene una representación decimal exacta.

Por la naturaleza necesaria aproximada de la representación, desde el ingreso y posterior tratamiento de los números fraccionarios surgen errores de distinto tipo, por redondeo simétrico y truncamiento, errores absolutos y errores relativos. La propagación de errores en los cálculos no permiten predecir la confiabilidad de los resultados.

En base a lo expuesto precedentemente, se implementó un intérprete **NFrac** que permita representar y tratar los números fraccionarios como tales para optimizar la exactitud y precisión de los resultados.

NFrac realiza operaciones tales como: determinar el máximo y el mínimo de una secuencia de fracciones, obtener un número decimal a partir del ingreso de un número fraccionario y viceversa, obtener el resultado de operaciones combinadas y en todos los casos, mostrar el error de representación.

Metodología

Para el diseño y desarrollo de **NFrac** se siguió la siguiente metodología:

1. **Exploración bibliográfica** sobre los temas involucrados en este trabajo: números fraccionarios, representaciones y operaciones, traductores, etapas de un compilador, generadores automáticos léxicos y sintácticos.

2. **Construcción del Intérprete**

- 2.1 **Análisis**

- 2.1.1 **Análisis léxico (AL):** especificación del vocabulario, de los componentes léxicos (identificadores, palabras claves, constates, etc.), de las expresiones regulares para cada componente léxico. Codificación mediante el empleo del generador automático de analizadores léxicos tipo *Lex*. Prueba del analizador léxico.

- 2.1.2 **Análisis sintáctico (AS):** elaboración de la gramática y prueba manual de la misma, codificación mediante el empleo del generador de analizador sintáctico tipo *Yacc*. Prueba del analizador sintáctico.

- 2.2 **Síntesis**

- A partir de la gramática generada para las funciones de **NFrac**, se realiza la implementación del intérprete en el lenguaje C++ Builder 5.

3. Evaluación del Intérprete

- 3.1. Pruebas unitarias para todas las operaciones detalladas.
- 3.2. Pruebas de integración.

2. DESARROLLO DE NFRAC

2.1. ANÁLISIS

El intérprete NFrac debe realizar una serie de operaciones que se describen a continuación.

- **Ingreso de números fraccionarios**
- **Impresión por pantalla de los diferentes resultados**
- **Operaciones aritméticas:** el intérprete debe ser capaz de realizar cualquier tipo de operaciones combinadas con números fraccionarios. Las operaciones que se pueden realizar son: suma, resta, división, multiplicación, potencia, raíz y valor absoluto
- **Cálculo del máximo y del mínimo de una secuencia de fracciones** ingresadas.
- **Pasaje de un número fraccionario a número decimal**
- **Pasaje número decimal a número fraccionario:** se deben considerar los casos de números fraccionarios periódicos puros, periódicos mixtos y decimales exactos.

Errores de representación: Esta operación permite mostrar los resultados de las operaciones combinadas con la representación interna de la computadora, y el cociente de dos números enteros.

2.1.1. Análisis Léxico

2.1.1.1. Descripción de componentes léxicos

En el análisis léxico es conveniente diferenciar, los términos “componente léxico” y “patrón” que tienen significados específicos. En la tabla 1 aparecen ejemplos de dichos usos. En general, hay un conjunto de cadenas en la entrada para el cual se produce como salida el mismo componente léxico. Este conjunto de cadenas se describe mediante una regla llamada patrón asociada al componente léxico. Se dice que el patrón concuerda con cada cadena del conjunto. Un lexema es una secuencia de caracteres en el programa fuente con la que concuerda el patrón para un componente léxico.

En la mayoría de los lenguajes de programación se consideran componentes léxicos las siguientes construcciones: palabras claves, operadores, identificadores, constantes, cadenas literales y signos de puntuación. La devolución de un componente léxico a menudo se realiza mediante un paso de un número entero correspondiente al componente léxico.

Un patrón es una regla que describe un conjunto de lexemas que pueden representar a un determinado componente léxico en los programas fuentes.

En la tabla 1 se describen los componentes léxicos y las expresiones regulares de Nfrac.

Tabla 1. Descripción de los componentes léxicos y expresiones regulares de NFrac

Componente léxico	Descripción formal (ER)	Descripción informal del patron
<i>Asignación</i>	=	Igual
<i>Barra</i>	/	Barra que separa denominador y numerador
<i>BarraA-BarraC</i>	[]	Valor absoluto
<i>Coma</i>	,	Coma
<i>División</i>	%	División
<i>Final</i>	-----	Terminal que representa el fin del cuerpo del programa
<i>Inicio</i>	-----	Terminal que representa el principio del cuerpo del programa
<i>Maximin</i>	-----	Palabra reservada para el máximo y mínimo de un secuencia de fracciones
<i>Nombre</i>	-----	Representa que se le asigna al programa o a una variable
<i>Num</i>	[0-9]+	Números enteros sin signo
<i>Parentesisa</i>	(Paréntesis que abre
<i>Parentesisc</i>)	Paréntesis que cierra
<i>PasajeDF</i>	-----	Palabra reservada para el pasaje de decimal a fracción
<i>PasajeFD</i>	-----	Palabra reservada para el pasaje de fracción a decimal
<i>Potencia</i>	.	Potencia
<i>Producto</i>	*	Producto
<i>Programa</i>	-----	Palabra reservada para comenzar a escribir los comandos
<i>PtoComa</i>	;	Punto y coma
<i>Raiz</i>	Ç	Raíz
<i>Resta</i>	-	Resta
<i>Suma</i>	+	Suma

En Nfrac los componentes léxicos se tratan como símbolos terminales de la gramática del lenguaje fuente, con nombres en negrita para representarlos. Los lexemas para el componente léxico que concuerda con patrón representan cadenas de caracteres en el programa fuente que se pueden tratar juntos como una unidad léxica.

2.1.1.2. Implementación del Análisis léxico

Para la implementación se utilizó el “Generador de Analizadores Léxicos AFD”¹, el cual genera el autómata finito determinístico a partir de una ER fuente que se encuentra en un archivo de texto. En síntesis, se ingresan al AFD las ER especificadas en la tabla 1 y se obtienen los autómatas finitos determinísticos correspondientes al analizador léxico.

¹ Tesis “Un generador de analizadores sintácticos ascendentes y un generador de analizadores lexicográficos”.

Domingo Eduardo Becker .
UCSE, 1996.

En la figura 1 se muestra como se ingresa la ER correspondiente al terminal “Nombre”.



Figura 1. Pantalla de AFD

Para cada ER ingresada se obtiene como salida del proceso dos archivos con extensiones .CMM y .TXT con el mismo nombre de la entrada. En la figura 2 se visualiza la salida correspondiente al archivo con extensión .CMM generado a partir de la ER del terminal “Nombre”.

```
# include "expreg.h"
# endif

static const char er[] = "// Exp reg para el identificador Nombre:\r\n# Letra [a-
zA-Z_áéíóúñÑüÛ]\r\n# Dígito [0-9]\r\nLetra (Letra | Dígito)*\r\n";
static const char * vc[] = {
    "a-zA-Z_áéíóúñÑüÛ", "0-9"
};
static const Ent16ns t[] = {
    1, TNDEF,
    1, 1
};
static const Ent16ns ef[] = {
    1, 1
};
AutomFinDet afd = {
    er, vc, 2, t, ef
};
```

Figura 2. Archivo generado con AFD

Antes de pasar al análisis sintáctico se realizaron las pruebas correspondientes para verificar si los AFD generados no contienen errores y si fueron generados todos los autómatas finitos determinísticos a partir de las ER especificadas en la tabla 1.

2.1.2. Análisis Sintáctico

2.1.2.1. Elaboración de la Gramática

Para el intérprete Nfrac se define la siguiente Gramática libre de contexto:

$$G = (V_N, V_T, P, S)$$

donde:

$$V_N =$$

{S, A, I, D, J, H, L, M, Asignación, O, E, F, T, K, G, N, P, B, C, R}

$$V_T =$$

{Programa, Nombre, PtoComa, inicio, Final, Maximin, PasajeDF, PasajeFD, Igual, Suma, Resta, Producto, Division, Potencia, Raiz, ParentesisA, ParentesisC, barraA, barraC, Num, barra}

P :

S → *Programa* A;

A → *Nombre PtoComa* I;

I → *inicio* D *Final*;

D → J | H | L | M | *Asignacion PtoComa* | *Maximin* B | *PasajeDF PtoComa* | *PasajeFD* B;

J → *Asignacion PtoComa* D;

H → *Maximin* B D;

L → *PasajeDF PtoComa* D;

M → *PasajeFD* B D;

Asignación → *Nombre* O;

O → *Igual* E;

E → E *Suma* F | E *Resta* F | F | *Resta* F;

F → F *Producto* T | F *Division* T | T | F *Potencia* G | F *Raiz* G;

T → *ParentesisA* E *ParentesisC* | *barraA* E *barraC* | G;

K → *Num* N;

G → *Num* N | *Nombre* | *ParentesisA* R *ParentesisC*;

N → *barra* P;

P → *Num*;

B → *Igual* C;

C → K *Coma* C | K *PtoComa*;

R → *Resta* *Num* N;

S: símbolo inicial de la gramática.

Esta gramática es utilizada por el analizador sintáctico para luego saber si reconoce o no las hileras ingresadas en el intérprete Nfrac.

2.1.2.2. Implementación del AS

Se realizó la implementación usando el “Generador Automático de Analizadores Sintáctico SLR1 versión 2.3”².

El SLR1 es un programa para generar tablas de análisis sintáctico LR(1) simples a partir de la especificación de la gramática del lenguaje. Las tablas generadas por este programa serán usadas para la realización del análisis sintáctico de oraciones del lenguaje.

La entrada de SLR1 v2 es un archivo de texto plano que contiene la gramática y las acciones semánticas para las reglas. El archivo que contiene la gramática es un archivo .GLC .

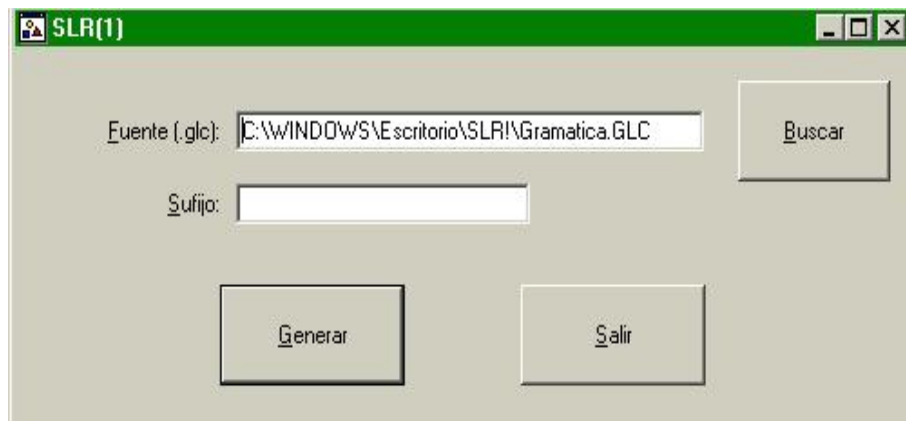


Figura 3. Pantalla de SLR(1)

Las salidas del programa son tres archivos:

- un archivo **.cmm**, con las acciones semánticas y el analizador léxico (en C++),
- un archivo **.tab**, con las tablas en formato C++ que se utilizan para el análisis sintáctico.
- un archivo **.est** con las tablas de análisis sintáctico, con los estados y los códigos asignados a los símbolos terminales y no terminales.

En la figura 4 se muestra parte del archivo **.est** generado por SLR1 v2 a partir de la gramática ingresada.

² Tesis “Un generador de analizadores sintácticos ascendentes y un generador de analizadores lexicográficos”.

Domingo Eduardo Becker .
UCSE, 1996.

```

NoTerm = { S, A, I, D, J, H, L, M, Asignacion, O, E, F, T, K, G, N, P, B,
C, R }
Term = { Programa, Nombre, PtoComa, inicio, Final, Maximin, PasajeDF,
PasajeFD, Igual, Suma, Resta, Producto, Division, Potencia, Raiz, arentesis,
Parentesisc, barraA, barraC, Num, barra, Coma }
Gramática:
S ---> Programa A
A ---> Nombre PtoComa I
I ---> inicio D Final
D ---> J
D ---> H
D ---> L
D ---> M
D ---> Asignacion PtoComa
D ---> Maximin B
D ---> PasajeDF PtoComa
D ---> PasajeFD B
J ---> Asignacion PtoComa D
H ---> Maximin B D
L ---> PasajeDF PtoComa D

```

Figura 4. Programa parcial de AS

2.2. SÍNTESIS

Las acciones semánticas para las reglas se codificaron en C ++ Builder 5.0. En la figura 5 se incluye parte del código correspondiente a la operación de expresiones algebraicas.

```

void Numerodecimal (float &numef, float &denof)
{
String nu,de, numer,denom;
int c, icad,dimecade,aux;
dimecade=numedecimal.Length();
icad=1;
if (numedecimal[icad] != '-')
{
//Concatena el numerador y lo convierte a entero
nu='';
while (numedecimal[icad] != '/')
{
nu=nu+numedecimal[icad];
icad++;
}
//Concatena el denominador y lo convierte a entero
de='';
icad++;
while (icad <= dimecade)
{

```

Figura 5. Programa parcial de las expresiones algebraicas

En la figura 6 se sintetiza por medio de un esquema las etapas de análisis y síntesis que se siguieron para el desarrollo del interprete NFrac.

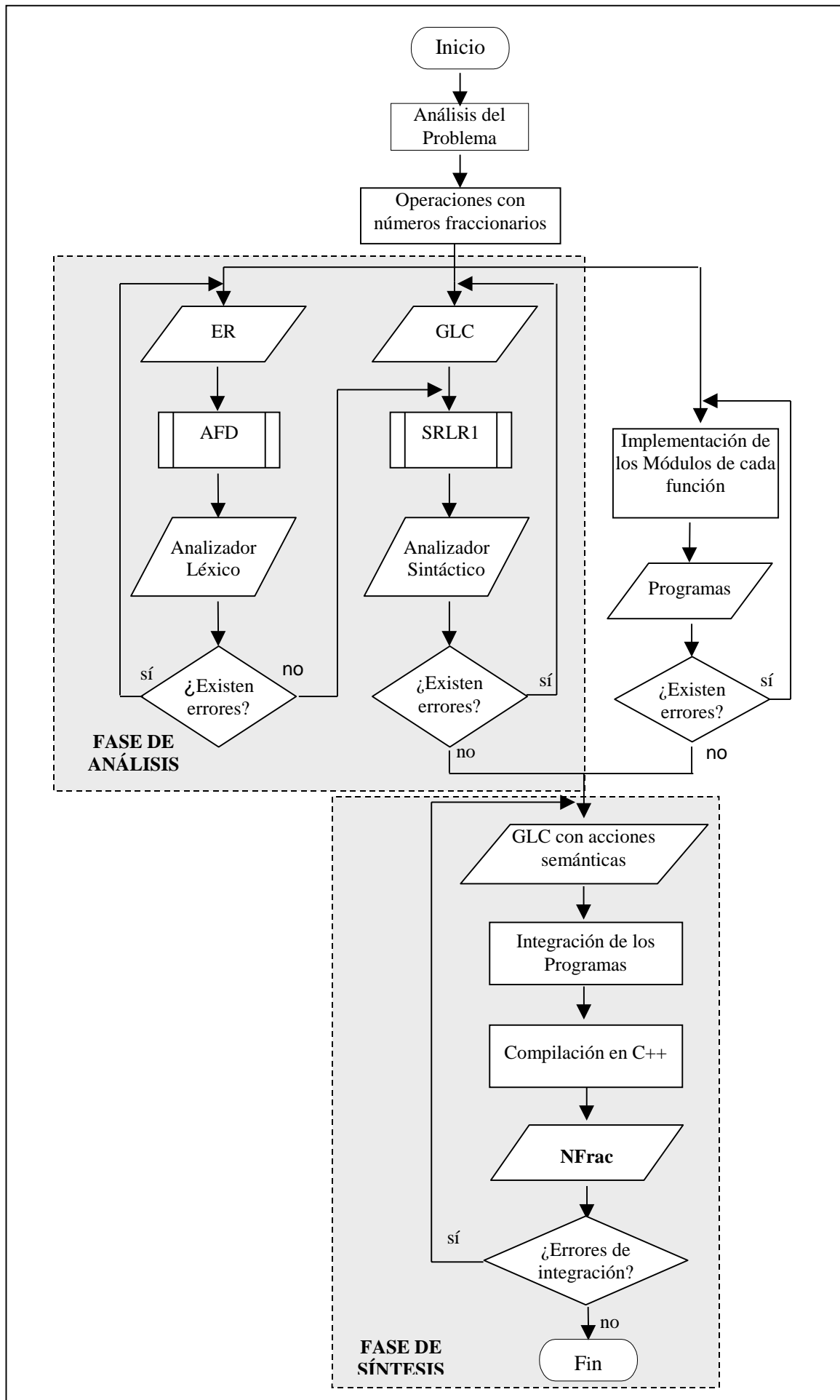


Figura 6. Metodología para la construcción de NFrac

3. NFRAC EN ACCIÓN

En la figura 7 se muestra un esquema del funcionamiento del intérprete Nfrac.

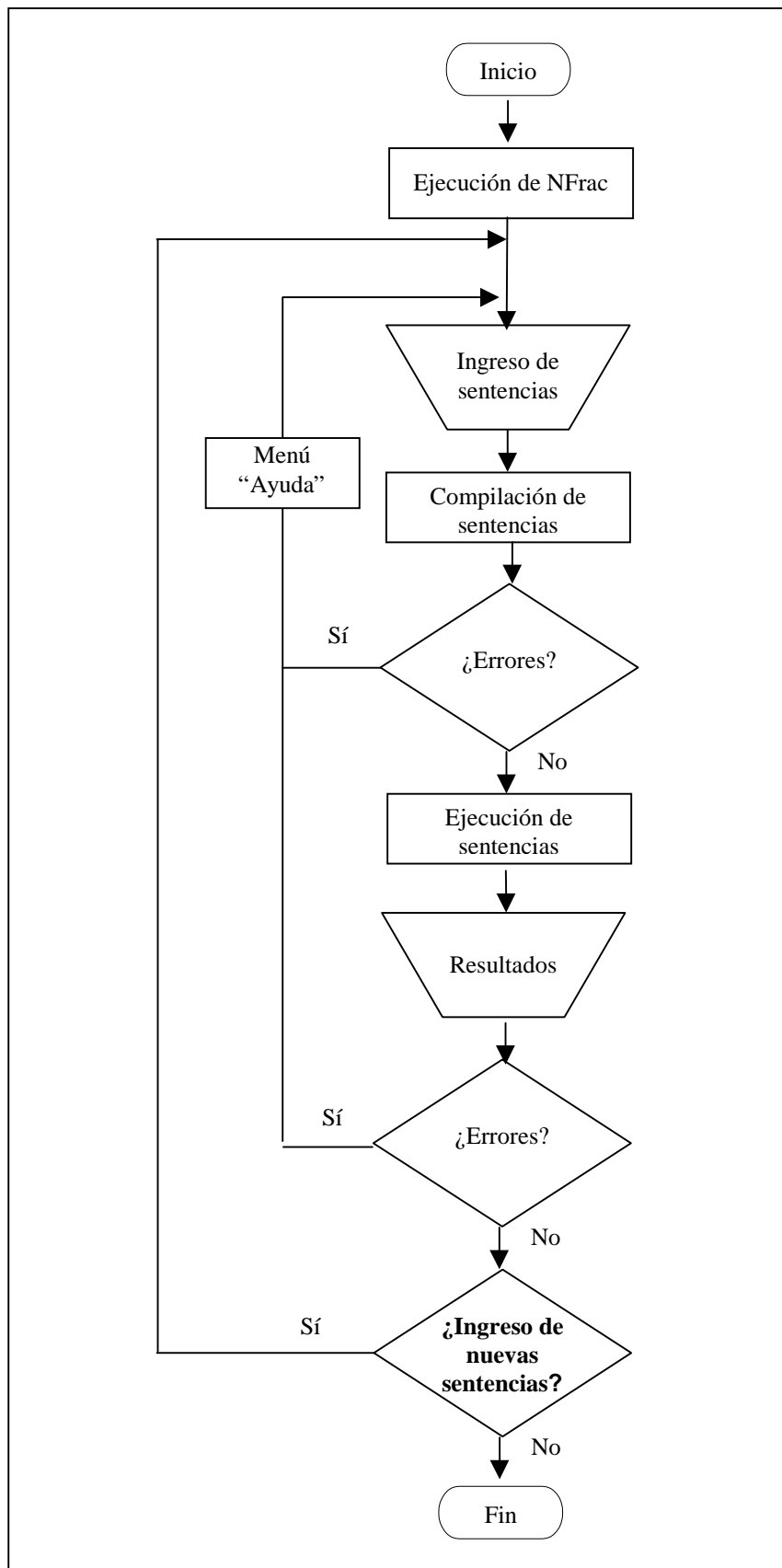


Figura 7. Esquema de funcionamiento de Nfrac

La figura 8 muestra el editor de texto en el cual el usuario ingresa las sentencias necesarias para realizar las operaciones.

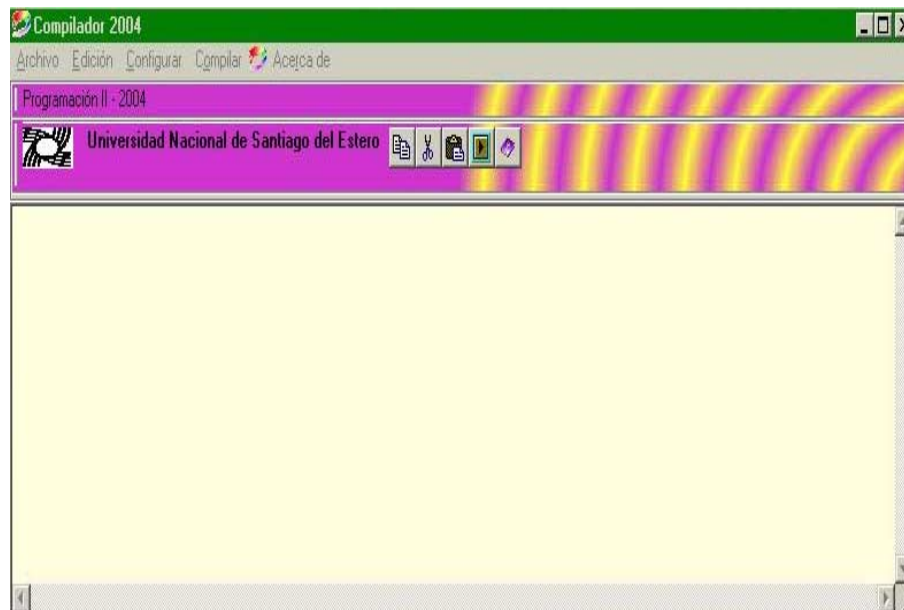


Figura 8. Pantalla principal de NFRAC

La figura 9 muestra el menú de opciones del intérprete Nfrac.



Figura 9. Barra de menú

3.1. USO DE NFRAC PARA EJECUTAR OPERACIONES

A continuación se detallan algunos ejemplos del uso del intérprete Nfrac.

❖ Maximin

La figura 10 muestra la operación *Maximin*, la cual permite obtener el máximo y el mínimo de una secuencia de fracciones. Dicha secuencia debe ser ingresada, los números deben estar separados por una coma y no debe haber espacios en blancos entre número y número. El fin de la secuencia viene dado por el punto y coma.

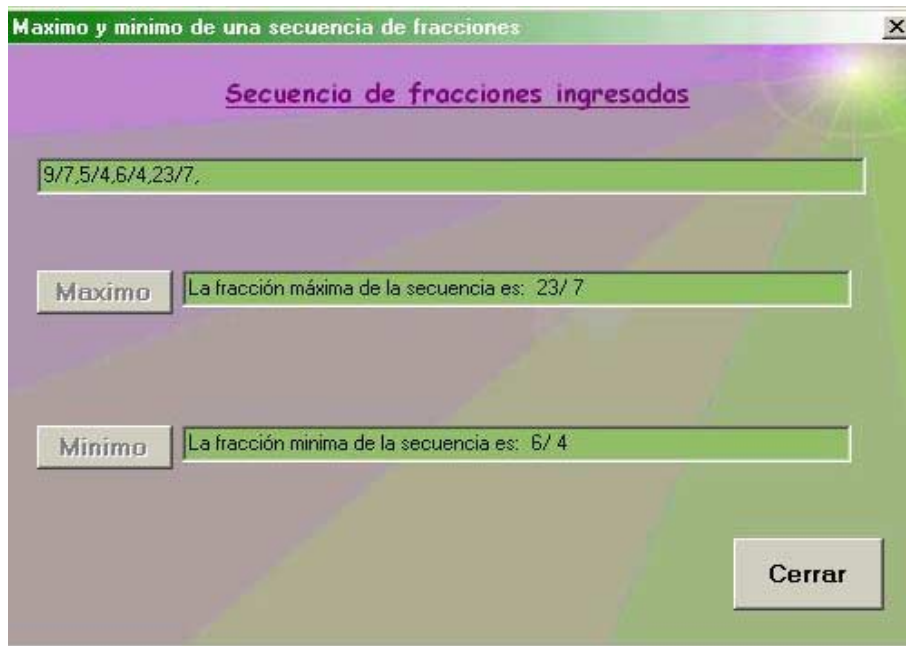


Figura 10. Máximo y mínimo de una secuencia

❖ **PasajeDF**

En la figura 11 se muestra la operación *PasajeDF*, la que realiza el pasaje de números decimales a números fraccionarios.

Como se puede observar los números que están después de las comillas representan la parte periódica de un numero decimal.

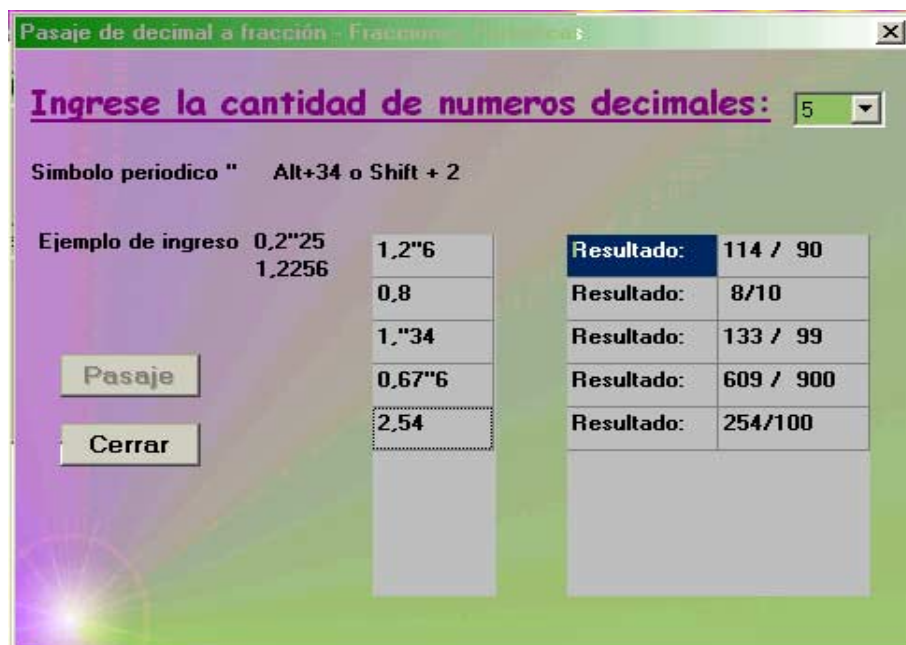


Figura 11. Pasaje de decimal a fracción

❖ PasajeFD

En la figura 12 se puede observar la operación *PasajeFD*, la cual realiza el pasaje de un número fraccional a un número decimal.

La secuencia de números deben ingresarse en el editor de texto y estar separados por una coma y no debe haber espacios en blancos entre número y número. El fin de la secuencia viene dado por el punto y coma.

Es decir si se quiere realizar el pasaje de la secuencia $2/4, 5/9, 7/8$ se debe escribir en el editor de texto: PasajeFD=2/4,5/9,7/8;

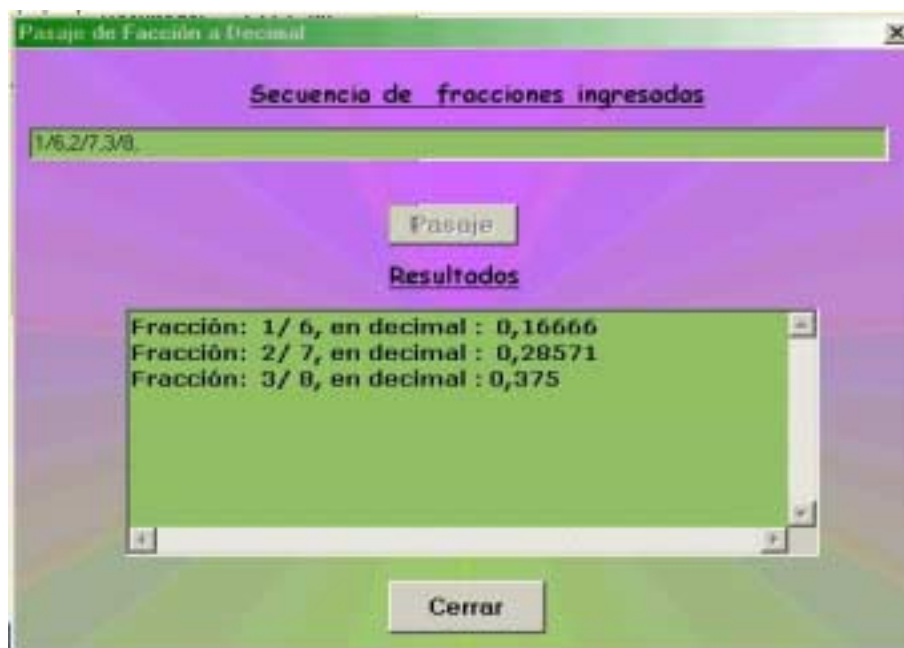


Figura 12. Pasaje de fracción a decimal

❖ Expresiones Algebraicas

En la figura 13 se muestra el resultado de las operaciones combinadas con números fraccionarios También se permite realizar asignaciones a variables, tal es el caso de $C=78/7$;

. Las operaciones que se pueden realizar son:

+ : Suma

- : Resta

* : Producto

% : División

· = Potencia , el símbolo "." se lo obtiene presionando (Shift + 3)

Ç = Raíz , el símbolo "Ç" se lo obtiene presionando (Alt + 128)

[] = Valor Absoluto, los símbolos son los corchetes

Si se quiere obtener el resultado de una expresión algebraica tiene que se ingresado en el editor de texto, luego que se realizo la compilación se ejecuta la ventana en donde se mostrará el resultado.

Ejemplos de operaciones algebraicas:

$$A=54/4+6/4; \quad B=6/4\%6/5; \quad C=78/7; \quad D=5/3+4/2-1/4;$$

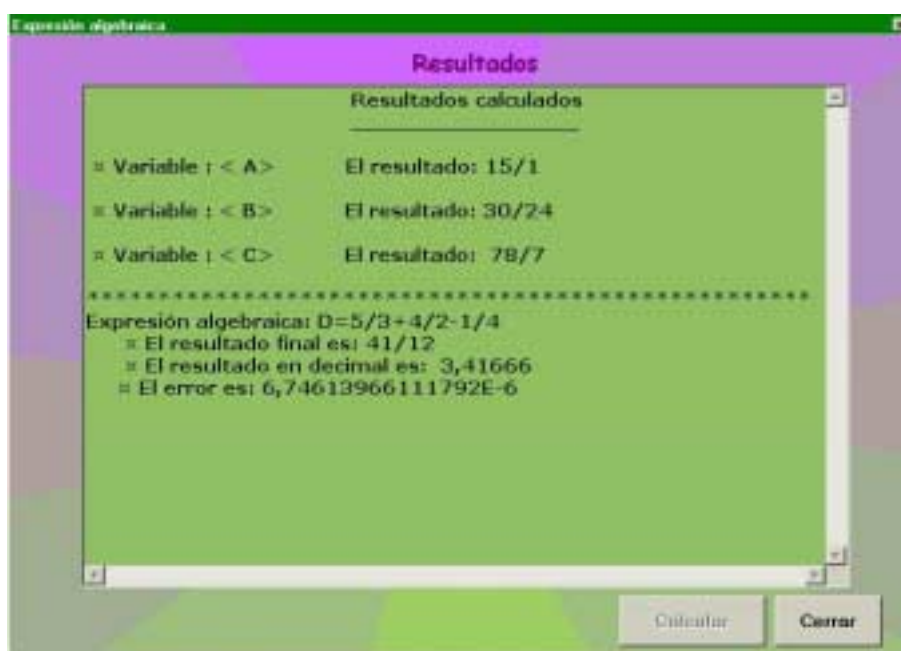


Figura 13. Ventana de resultados de operaciones algebraicas

3.2. USO DE NFRAC CON FINES DIDÁCTICOS

En la figura 14 se observa el software de ayuda construido en el programa Neobook 4.0 en donde se muestra de forma sencilla y general el uso de intérprete Nfrac.



Figura 14. Ventana de ayuda

En la figura 15 se muestra el software educativo hecho en el programa Neobook 4.0 mediante el cual se puede aprender en forma sencilla y general los diversos temas relacionados con los números fraccionarios.



Figura 15. Pantalla principal de software educativo

4. CONCLUSIONES

En este trabajo se ha desarrollado un intérprete denominado NFrac para trabajar con números fraccionarios.

A partir del análisis léxico y sintáctico realizado, se obtuvieron los componentes léxicos (identificadores, palabras claves, constantes, etc.), las expresiones regulares y el analizador sintáctico.

Se implementaron en lenguaje C++ Builder 5.0, funciones tales como: máximo y mínimo de una secuencia, resolución de expresiones algebraicas, representación del error, pasajes de decimal a fracción y viceversa.

Para todas las funciones realizadas se realizaron las pruebas estáticas y dinámicas y finalmente se realizó una prueba de integración.

Con el desarrollo de NFrac no sólo aplicamos los conocimientos teóricos adquiridos en la asignatura Programación II (Teoría de Algoritmos, Teoría de Lenguajes Formales y Autómatas, Construcción de compiladores), sino que también comprobamos cómo se puede utilizar un intérprete con fines didácticos, ya que por medio de su utilización se puede abordar el tema de números fraccionarios bien sea para mostrar o para validar resultados calculados manualmente. También de una manera sencilla se puede visualizar los errores que se cometen al trabajar con los números fraccionarios como el cociente de dos números enteros.

BIBLIOGRAFÍA

- Aho, Alfred; Ullman, Jeffrey y Sethi, Ravi *Compiladores, Principios, Técnicas y Herramientas*. Editorial Addison Wesley Iberoamericana. 1986.
- Alfonseca Manuel, Sancho, Justo y Martínez Orga *Teoría de Lenguajes, Gramáticas Y Autómatas*. Ediciones Universidad y Cultura, 1990.
- Kelly Dean *Teoría de Autómatas y Lenguajes Formales*. Editorial Prentice Hall, 1995.
- Mandrioli Dino y Ghezzi Carlo *Theoretical Foundations of Computer Science*. John Wiley & Sons, 1987.
- Charte, Francisco *Programación con C++ Builder*. Ediciones Anaya. Multimedia 1999.
- Leblanc G. *Borland C++ Builder*, Ediciones Eyrolles, 1997.
- Levine, J. ; Mason, T. y Brown *Lex & Yacc*. O'Reilly & Associates, 1.995.
- Barchini Graciela y Alvarez Margarita *Fundamentos Teóricos de las Ciencias de la Computación* – 1987
- Domingo Alberto Becker *Tesis “un generador de analizadores sintácticos ascendentes y un generador de analizadores lexicográficos”*. Trabajo final para optar por el título de Ing. en Computación. UCSE, 1996