UNIVERSIDAD SAN IGNACIO DE LOYOLA

PROGRAMACIÓN Y ESTRUCTURAS DE DATOS

"DESARROLLO DE UNA CALCULADORA DE INTEGRALES INDEFINIDAS PARA MEJORAR LA EFICIENCIA ACADÉMICA EN EL ÁMBITO EDUCATIVO"

AUTOR(ES)

PEREZ CONCHA JUAN ANTONIO

Ingeniería de Software

Profesor

Escobar Aguirre, Jaime Luis

Lima - Perú

2024

Tabla de contenidos

1 Introducción **3**

2 Descripción de caso de estudio **5**

3 Objetivos **6**

3.1 Objetivos específicos 6

4 Marco teórico **7**

5 Desarrollo **11**

5.1 Identificación de requerimientos 11

5.1.1 Requerimientos funcionales: 11

5.1.2 Requerimientos no funcionales: 11

5.2 Solución del caso 12

5.2.1 Análisis y descomposición 12

5.2.2 Diseño 19

5.2.3 Implementación 29

5.2.4 Pruebas 47

6 Conclusiones y recomendaciones **49**

7 Referencias **50**

1 Introducción

El estudio del cálculo, tanto de una variable como de varias, constituye una base fundamental en la formación de ingenieros y científicos, proporcionando herramientas imprescindibles como los límites, derivadas e integrales. Estos conceptos son la piedra angular del análisis matemático desde Newton y Leibniz. Su comprensión es crucial para el modelado y la resolución de problemas complejos en diversas ramas de la ingeniería y las ciencias aplicadas.

Las integrales, en sus diversas manifestaciones, permiten el cálculo de áreas bajo curvas, la determinación de volúmenes de sólidos de revolución, la obtención de antiderivadas generales y la solución de problemas en campos como la física, la ingeniería y la economía, entre otros. Dada la amplitud de sus aplicaciones, resulta indispensable que la enseñanza del cálculo esté orientada no solo a la comprensión técnica de los métodos, sino también a la integración de estos conocimientos en escenarios prácticos y aplicados. De este modo, se fomenta un aprendizaje significativo y contextualizado, que permita a los estudiantes adquirir las competencias necesarias para enfrentar desafíos profesionales y científicos.

En este contexto, la implementación e integración de nuevas herramientas tecnológicas en el ámbito educativo se ha vuelto una necesidad cada vez más evidente. A lo largo de la historia, estas innovaciones han transformado la manera en que los docentes transmiten el conocimiento y los estudiantes lo asimilan. En particular, el uso de herramientas tecnológicas ha permitido no solo la automatización de cálculos complejos, sino también la oportunidad de innovar y mejorar los procesos pedagógicos, brindando un enfoque más interactivo y dinámico a la enseñanza de conceptos abstractos. Este avance en el proceso educativo facilita la personalización del aprendizaje, permitiendo a los estudiantes desarrollar un mayor nivel de autonomía en la resolución de problemas y en la adquisición de competencias.

En el Perú, la calidad educativa en los colegios sigue siendo la preocupación central que afecta de manera directa al desempeño de los estudiantes en el nivel universitario. La brecha en la calidad educativa entre instituciones públicas y privadas,

así como entre las zonas urbanas y rurales, genera desigualdades significativas en la preparación. Este problema se manifiesta claramente en las diferencias en los niveles de competencias académicas, especialmente en áreas como matemáticas.

Las universidades, especialmente las privadas, tienen el reto de subsanar estas deficiencias académicas gestadas desde etapas tempranas de la formación académica. Muchas de ellas, destinan recursos en la implementación de programas de nivelación en matemáticas o lenguaje, con el objetivo de aminorar estas deficiencias. A pesar de ello, un número considerable de estudiantes siguen teniendo problemas al afrontar materias básicas para la formación en ingeniería, como cálculo de una variable, cálculo de varias variables, ecuaciones difernciales entre otras. Por otro lado, esta situación pone presión adicional sobre las universidades, dado que su misión es garantizar la formación de profesionales competentes y con la capacidad de desenvolverse en sus respectivos campos laborales.

Dicho lo anterior, el presente estudio propone la implementación e integración de una herramienta tecnológica que reduzca la curva de aprendizaje de los estudiantes de la facultad de ingeniería de la Universidad San Ignacio de Loyola en cursos de cálculo de una y de varias variables, a través de una calculadora de integrales simples con memoria.

2 Descripción de caso de estudio

Este caso de estudio se enfoca en el desarrollo de una herramienta que se convierta en un recurso importante para el estudiante durante el curso de cálculo 1, cálculo 2 y ecuaciones diferenciales. Se trata de una calculadora de integrales con memoria que tiene como propósito facilitar el proceso de aprendizaje y solución de problemas complejos.

La problemática que motiva este estudio radica en la falta de preparación adecuada que muchos estudiantes arrastran desde su etapa escolar, especialmente en áreas como matemáticas y lenguaje. Esta deficiencia se manifiesta en un bajo rendimiento académico en materias relacionadas, lo que genera una serie de dificultades en su formación universitaria.

En este contexto, la Universidad San Ignacio de Loyola ha destinado recursos a la implementación de cursos de nivelación en matemáticas. Sin embargo, estos cursos a menudo no garantizan el desarrollo de las competencias necesarias que los estudiantes deben poseer para enfrentar los desafíos académicos en niveles superiores. Esta situación, a la larga, puede resultar en un estancamiento académico para los estudiantes y un porcentaje significativo de reprobaciones en los cursos, lo que podría impactar negativamente en la reputación de la universidad y en la retención de nuevos estudiantes.

El objetivo es aminorar en la medida de lo posible el problema actual de la universidad, dotando a los estudiantes de una herramienta que puede llegar a mejorar su rendimiento académico en matemáticas y reducir las deficiencias educativas identificadas en etapas tempranas de su formación.

3 Objetivos

Desarrollar un programa en C++ que sirva como recurso de apoyo y consulta para estudiantes de la facultad de ingeniería, con el fin de mejorar el rendimiento académico de los mismos.

3.1 Objetivos específicos

1. Implementar funcionalidades básicas de cálculo, como integraciones de funciones polinómicas, exponenciales, trigonométricas y logarítmicas.

2. Desarrollar una interfaz gráfica intuitiva y fácil de usar para la calculadora de integrales.

4 Marco teórico

El presente estudio empleará estructuras de datos que van desde listas enlazadas hasta árboles binarios. De igual manera, se recurrirán a conceptos matemáticos, así como a gramáticas libres de contexto.

Es conocido de antemano que, una lista es una estructura de datos dinámica, cuyo tamaño puede variar durante la ejecución del programa, En una lista enlazada, los elementos se organizan en nodos, donde cada nodo tiene dos componentes, el dato almacenado y el puntero al siguiente nodo. Se le suele representar de la siguente manera:

```
struct Node
{
    int data;
    Node *next;
};
```

O mediante una clase:

```
class Node
{
    int data;
    Nodo *next;

public:
    Node(int valData) : data(valData), next(nullptr);
    Node(int valData, Nodo *nextNodo) : data(valData), next(nextNodo);
};
```

El lector se encuentra en la obligación de conocer e identificar algunas de las reglas básicas de integración que se emplearán a lo largo del trabajo.

1. $\int dx = x + c$
2. $\int x^n \, dx = \frac{x^{n+1}}{n+1} + c, n \neq -1$
3. $\int e^x \, dx = e^x + c$

Una gramática libre de contexto se define por el conjunto de variables, terminales, reglas y un punto de partida para generar o analizar expresiones.

Se plantea resolver la siguiente integral:

$$\int (x^3 + x^2 - e^x)\, dx \tag{1}$$

El cálculo de la integral planteada resulta un proceso relativamente sencillo, siempre que se apliquen de manera correcta las reglas.

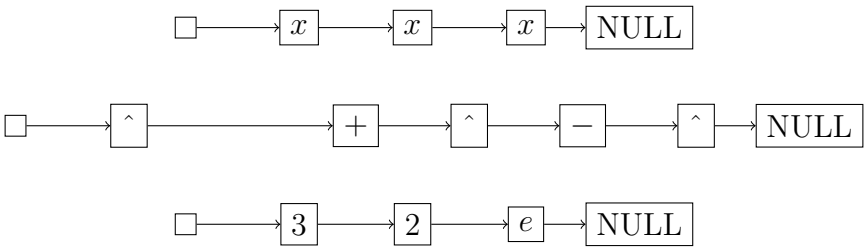
$$\int x^3\, dx + \int x^2\, dx - \int e^x\, dx$$

$$\frac{1}{4}x^4 + \frac{1}{3}x^3 - e^x + c$$

Pero, resulta complejo pensar en un algoritmo que permita resolver la integral planteada de forma general o que, pueda verificar la validez de la integral como expresión dentro de un contexto matemático adecuado. Es útil pensar en una estrategia basada en la filtración progresiva de los componentes de la función integrando, separando variables, símbolos y coeficientes en los distintos niveles. Dado que no es posible conocer de antemano la cantidad de estos elementos hasta el momento de ejecución $(a_1x_1 + a_2x_2 + \cdots + a_nx_n)$, en primera instancia se optará por listas para cada tipo de elemento a filtrar.

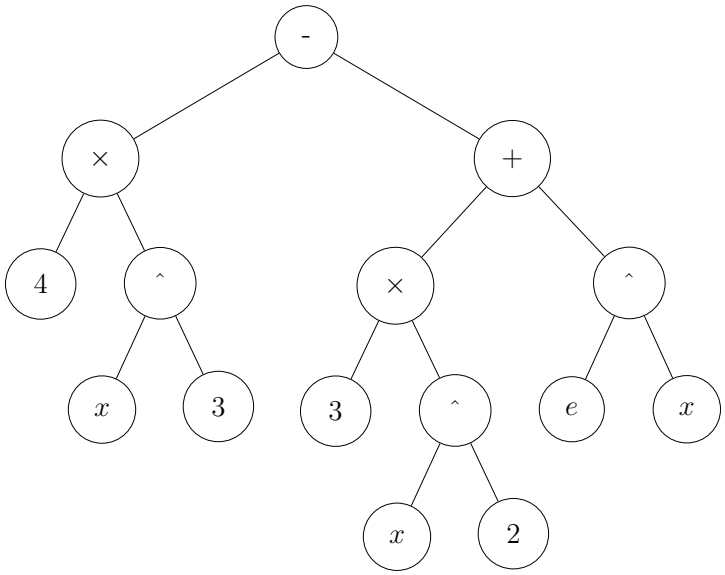
La función integrando del ejemplo anterior:

$$x^3 + x^2 - e^x$$



En segunda instancia se puede optar por un árbol de expresión, que no es más que un árbol binario que cumple con ciertas propiedades. En este tipo de estructura, cada hoja representa un operando, mientras que los nodos raíz e internos son operadores.

El árbol de expresión asociado a $4x^3 - (3x^2 + e^x)$:



Se define la gramática G. Considere la siguiente expresión para el ejemplo (1) en particular:

$$G = \{V, \Sigma, R, S\} \tag{2}$$

Donde las no terminales $V = \{\langle \text{EXPR} \rangle, \langle \text{TERM} \rangle, \langle \text{FACT} \rangle, \langle \text{NUM} \rangle, \langle \text{INT\_IND} \rangle\}$ y $\Sigma = \{\int, dx, x, e, \times, +, -, \hat{}, (,), \{1, 2, 3, \dots\}\}$, además $S = \langle \text{INT\_IND} \rangle$ y que R

$$\langle \text{INT\_IND} \rangle \to \int (\langle \text{EXPR} \rangle)\, dx$$

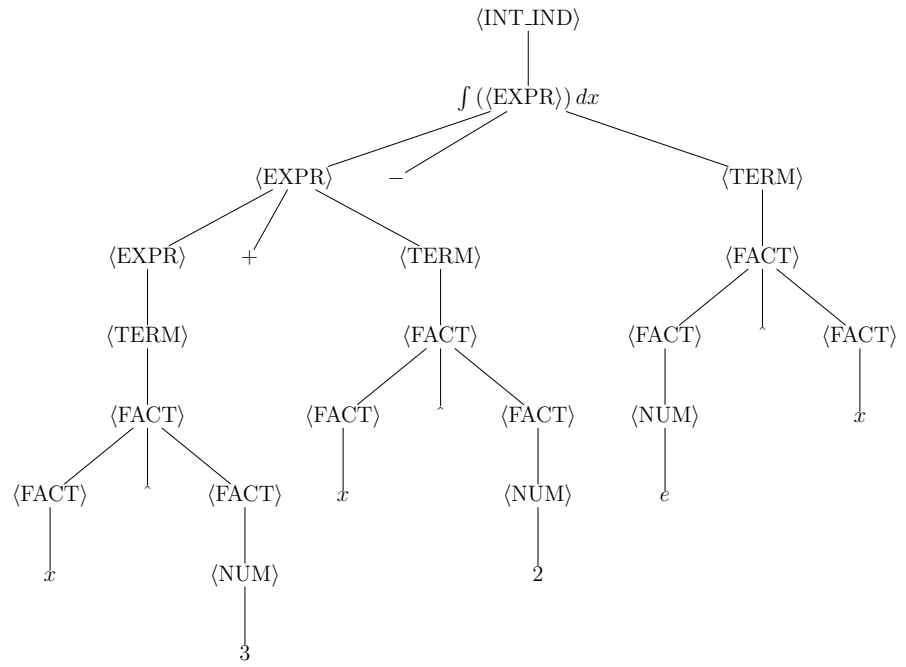
$$\langle \text{EXPR} \rangle \to \langle \text{EXPR} \rangle + \langle \text{TERM} \rangle \,|\, \langle \text{EXPR} \rangle - \langle \text{TERM} \rangle \,|\, \langle \text{TERM} \rangle$$

$$\langle \text{TERM} \rangle \to \langle \text{TERM} \rangle \times \langle \text{FACT} \rangle \,|\, \langle \text{FACT} \rangle$$

$$\langle \text{FACT} \rangle \to \langle \text{FACT} \rangle^{\langle \text{FACT} \rangle} \,|\, \langle \text{NUM} \rangle \,|\, x$$

$$\langle \text{NUM} \rangle \to \text{cualquier número real o constante}$$

Es posible validar el ingreso de la integral (1) a partir de las reglas de producción planteadas junto con un conjunto bien definido de terminales (Σ). Se presenta el árbol de derivación de (1).



Con esto, se obtiene que la cadena $\int (x^3 + x^2 - e^x)\, dx$ es parte de un contexto matemático adecuado (**Cálculo Integral)**.

Actualmente se tiene una gramática que maneja operaciones básicas como sumas, multiplicaciones y potencias. El objetivo es abarcar una cantidad más amplia de integrales, por lo tanto, se necesitan de más reglas que cubran funciones más complejas y hacerlo más generalizable evitando la ambiguedad. Un ejemplo de esto es:

$$\langle \text{MULTIPLE} \rangle \rightarrow \int \int \langle \text{EXPR} \rangle \, dx \, dy$$

$$\langle \text{EXPR} \rangle \rightarrow \langle \text{EXPR} \rangle + \langle \text{TERM} \rangle \,|\, \langle \text{EXPR} \rangle - \langle \text{TERM} \rangle \,|\, \langle \text{TERM} \rangle$$

$$\langle \text{TERM} \rangle \rightarrow \langle \text{TERM} \rangle \times \langle \text{FACT} \rangle \,|\, \langle \text{TERM} \rangle / \langle \text{FACT} \rangle \,|\, \langle \text{FACT} \rangle$$

$$\langle \text{FACT} \rangle \rightarrow \sin(\langle \text{EXPR} \rangle) \,|\, \cos(\langle \text{EXPR} \rangle) \,|\, ln(\langle \text{EXPR} \rangle) \,|\, e^{\langle \text{EXPR} \rangle} \,|\, x^n \,|\, C$$

5 Desarrollo

5.1 Identificación de requerimientos

Para el desarrollo de la problemática, se han identificado los siguientes requerimientos funcionales y no funcionales:

5.1.1 Requerimientos funcionales:

1. **Cálculo de integrales:** La calculadora permitirá el cálculo de integrales indefinidas para funciones polinómicas, exponenciales, trigonométricas y logarítmicas.
2. **Validación de expresiones:** La calculadora verificará la validez de las expresiones ingresadas antes de proceder con los cálculos.

5.1.2 Requerimientos no funcionales:

1. **Eficiencia en el procesamiento:** El sistema optimizará el procesamiento de las expresiones ingresadas $O(n \log n)$, asegurando cálculos rápidos y eficientes.
2. **Eficiencia en el procesamiento:** La calculadora permitará futuras expansiones, como la inclusión de integrales múltiples o la resolución de ecuaciones diferenciales simples. Para ello, se expanden las reglas gramáticales.

5.2 Solución del caso

5.2.1 Análisis y descomposición

En el capítulo anterior, se presentaron los conceptos teóricos y las reglas matemáticas fundamentales para el cálculo de una integral indefinida en particular. Además, se definió la gramática libre de contexto que especifica las expresiones válidas para dicho lenguaje. De igual forma, se mencionó el objetivo de elaborar un conjunto más amplio de reglas y principios, una gramática capaz de soportar unidades lingüísticas de mayor complejidad.

Se define el alfabeto para el lenguaje sobre el cual se trabajará:

$$\Sigma = \left\{ \int, x, e, \pi, dx, +, -, \times, \hat{}, /, \sqrt{\ }, (,), \{1, 2, 3 \dots\}, \{\sin x, \cos x \dots\} \right\} \tag{3}$$

Sea L el lenguaje, entonces se sabe que $L \subseteq \Sigma^*$, lo cual significa que el lenguaje es un conjunto de cadenas formadas por combinaciones de los símbolos del alfabeto. Aunque el alfabeto especifica qué simbolos son válidos, la gramática define cómo esos símbolos se combinan para formar cadenas válidas dentro del lenguaje. Dicho de otra forma, las reglas gramáticales determinan que cadenas de Σ^* pertencen a L

Se define la gramática para el lenguaje L como una 4-tupla:

$$G_L = \{V, \Sigma, R, S\}$$

Donde $V = \{\langle \text{EXPR} \rangle, \langle \text{TERM} \rangle, \langle \text{FACT} \rangle, \langle \text{VAR} \rangle, \langle \text{FUNC} \rangle, \langle \text{NUM} \rangle \dots\}$ y Σ el alfabeto definido anteriormente, además se toma en cuenta que la variable de inicio es $S = \langle \text{INT\_IND} \rangle$ y que R

$$\langle \text{INT\_IND} \rangle \to \int (\langle \text{EXPR} \rangle)\, dx$$

$$\langle \text{EXPR} \rangle \to \langle \text{EXPR} \rangle + \langle \text{TERM} \rangle \,|\, \langle \text{EXPR} \rangle - \langle \text{TERM} \rangle \,|\, \langle \text{TERM} \rangle$$

$$\langle \text{TERM} \rangle \to \langle \text{TERM} \rangle \times \langle \text{FACT} \rangle \,|\, \langle \text{TERM} \rangle / \langle \text{FACT} \rangle \,|\, \langle \text{FACT} \rangle$$

$$\langle \text{FACT} \rangle \to \langle \text{NUM} \rangle \times \langle \text{VAR} \rangle \mid \langle \text{FUNC} \rangle \mid \langle \text{VAR} \rangle \mid (\langle \text{EXPR} \rangle) \mid \langle \text{NUM} \rangle \mid \langle \text{VAR} \rangle^{\langle \text{NUM} \rangle} \mid \sqrt{\langle \text{EXPR} \rangle}$$

$$\langle \text{FUNC} \rangle \to \sin(\langle \text{EXPR} \rangle) \mid \cos(\langle \text{EXPR} \rangle) \mid \tan(\langle \text{EXPR} \rangle) \mid \ln(\langle \text{EXPR} \rangle) \mid \langle \text{RACIONAL} \rangle$$

$$\langle \text{RACIONAL} \rangle \to \langle \text{EXPR} \rangle / \langle \text{EXPR} \rangle$$

$$\langle \text{VAR} \rangle \to x$$

$$\langle \text{NUM} \rangle \to \text{cualquier número real o constante}$$

A partir de las reglas anteriores, se pueden generar expresiones matemáticas válidas dentro del lenguaje L. La gramática definida genera cadenas válidas de integrales indefinidas, como se muestra en el siguiente ejemplo.

Se tiene la siguiente cadena:

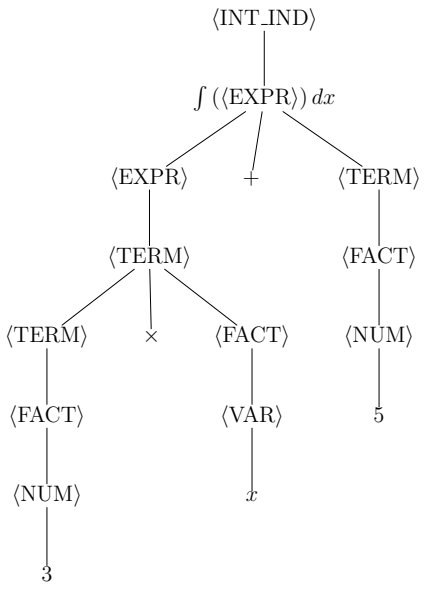
$$\int (3x + 5)\, dx \tag{4}$$

La cadena será válida si y solo si se puede derivar a partir de las reglas gramaticales anteriores. Se comenzará por identificar y separar las unidades de información básica (**tokens**) de la cadena.

Tokens	Lexemas
OP_INT	$\int$
DIFF	dx
OP_SUM	$+$
OP_MUL	$\times$
NUM	3
NUM	5
VAR	x

Table 1: Tokens

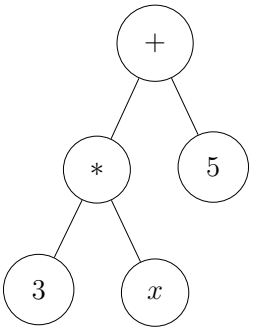
Su árbol de derivación.



De este modo, la cadena $\int (3x + 5)\, dx$ se genera de acuerdo con las reglas de la gramática.

Con esto, se verifica la validez de la cadena en el lenguaje L. Luego, se genera el árbol de expresión de la cadena con el objetivo de obtener una expresión válida para **Maxima**.

Se tiene el árbol de expresión para la cadena (4)



Se tiene la siguiente cadena:

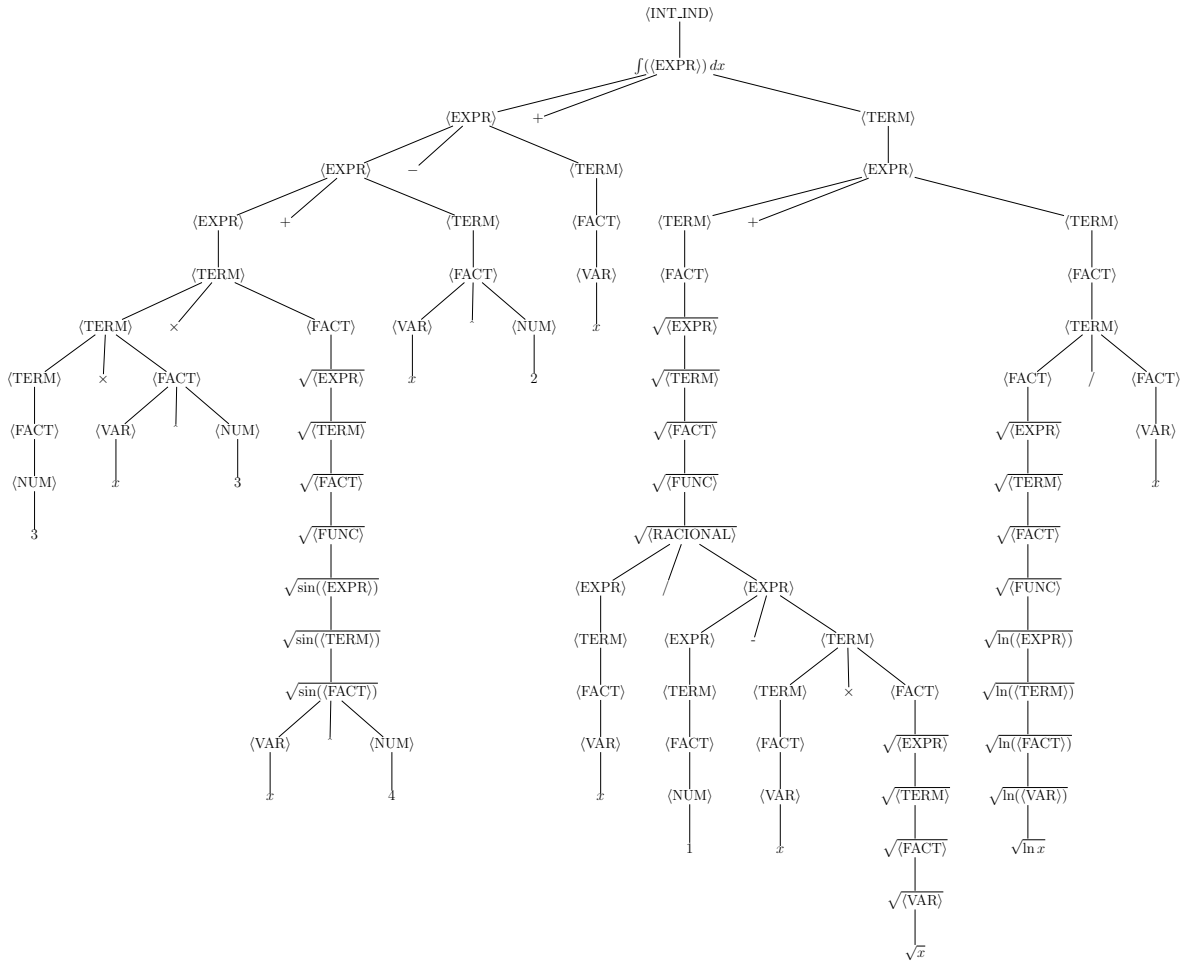
$$\int \left(3x^3 \sqrt{\cos x^4} + x^2 - x + \sqrt{\frac{x}{1 - x\sqrt{x}}} + \frac{\sqrt{\ln x}}{x} \right) dx \tag{5}$$

De forma análoga, la cadena será válida si y solo si se puede derivar a partir de las reglas gramaticales anteriores. Se comenzará por identificar y separar las unidades de información básica (**tokens**) de la cadena.

Su árbol de derivación.

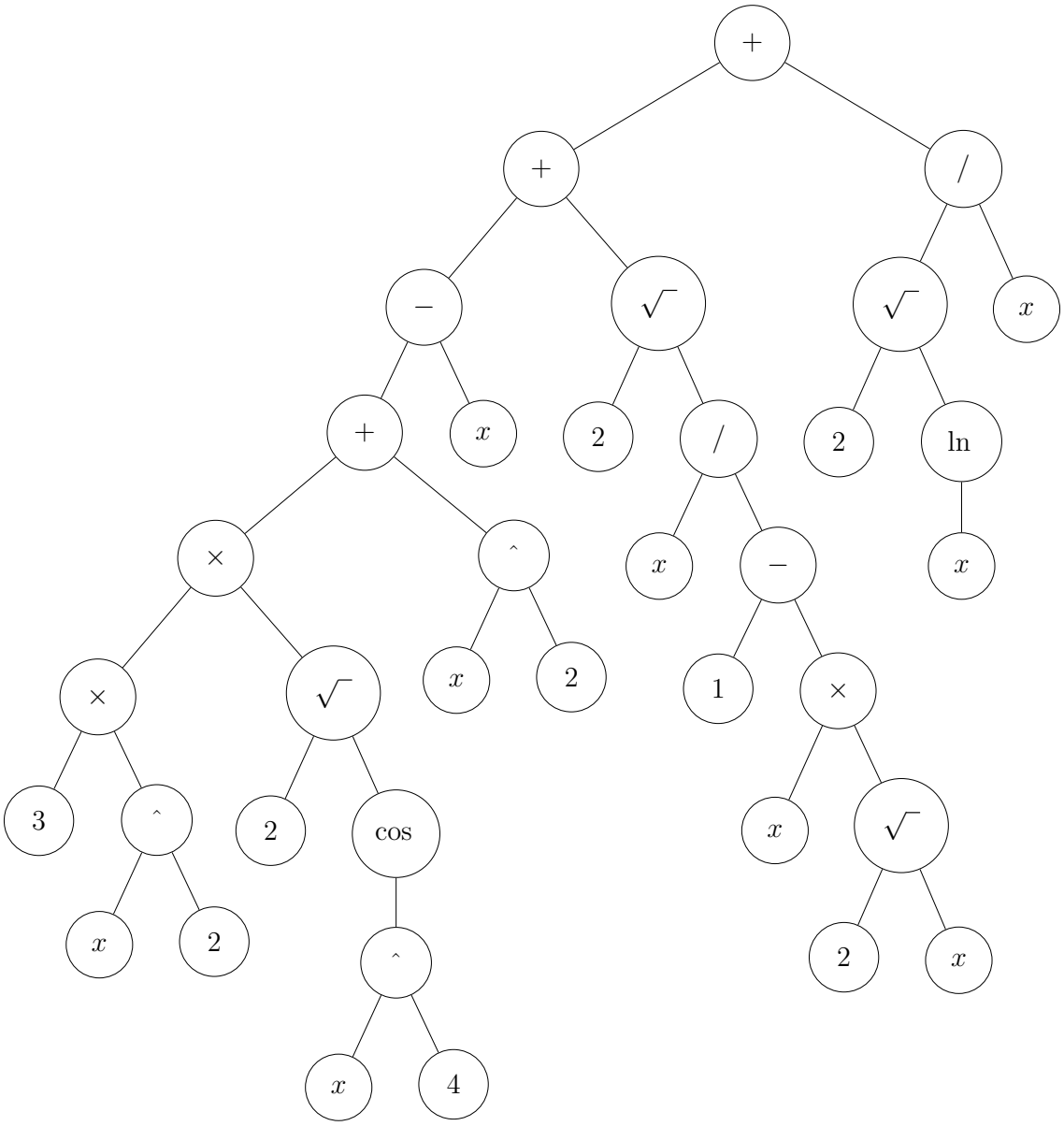
Tokens	Lexemas
OP_INT	$\int$
DIFF	dx
OP_SUM	$+$
OP_RES	$-$
OP_MUL	$\times$
OP_DIV	$/$
OP_POT	$\hat{}$
OP_RAIZ	$\sqrt{}$
FUN	sin
FUN	cos
FUN	ln
NUM	1
NUM	3
VAR	x

Table 2: Tokens



De este modo, la cadena $\int \left(3x^3\sqrt{\cos x^4} + x^2 - x + \sqrt{\frac{x}{1-x\sqrt{x}}} + \frac{\sqrt{\ln x}}{x}\right) dx$ se genera de acuerdo con las reglas de la gramática. Por lo tanto, (5) pertenece al lenguaje.

Se tiene el árbol de expresión para la cadena (5),



El árbol refleja cómo se deben agrupar los operadores y operandos. Se recorre el árbol **in-orden** y se añade el dato del nodo a una cadena de caracteres. Para (5).

3*xˆ2 * sqrt(cos(xˆ4)) + xˆ2 - x + sqrt(x/(x - 1 * sqrt(x))) + sqrt(lnx)/x

Ya mencionada en la página 14, **Maxima** es un sistema capaz de manejar expresiones simbólicas y numéricas —como la integración—. Este sistema será el encargado de realizar el cálculo de la integral; para ello, se le enviará la expresión generada a través del recorrido del árbol de expresión.

Sin embargo, dentro de **Maxima** existen diferentes tipos de funciones, cada una de ellas cumple con un cálculo en específico. Para el objetivo del trabajo, se hará uso de la función `integrate(expr, x);` que se utiliza para calcular la integral definida o indefinida de una expresión matemática. Donde `expr` es la expresión que se va a integrar y `x` es la variable con respecto a la cual se realizará la integración.

Sea `expr` una expresión —sin antes, verificar la validez de la cadena en el lenguaje L y convertir la expresión en una cadena—.

$$5 * x - \cos(3*x) + 1 \tag{6}$$

entonces, se tiene el comando.

```
integrate(5*x-cos(3*x)+1, x);
```

Maxima, retornará algo como:

```
-(sin(3*x) / 3) + (5*x^2) / 2 + x
```

O de la siguiente forma.

$$-\frac{\sin(3x)}{3} + \frac{5x^2}{2} + x + C$$

Se ejecuta el comando `integrate(5*x-cos(3*x)+e, x);` en **Maxima (command line)**.

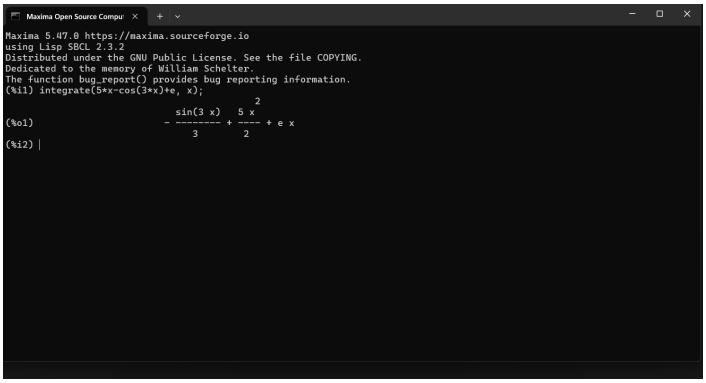


Figure 1: Ejecución de (6)

Integrando ambas ideas. se obtiene la identificación de tres etapas.

1. **Validación de una cadena ingresada en el lenguaje** L: Verificar que la expresión ingresada cumpla con las reglas gramaticales definidas. Asegurar que la estructura y los elementos de la cadena son correctos y reconocibles para el sistema.

2. **Generación de una cadena para Maxima a partir del árbol de expresión**: Se construye un árbol de expresión. Luego, se recorre el árbol **in-orden** para convertir la expresión en una cadena de texto que respete la notación requerida por **Maxima**.

3. **Enviar cadena a Máxima y procesar resultado**:

    1. Se envía la cadena generada al sistema Maxima mediante un archivo de script (`.mac`), que contiene las instrucciones necesarias para calcular la operación o resolver la expresión.
    2. **Maxima** procesa la cadena y genera un resultado en formato texto.
    3. El resultado obtenido es leído desde la salida estándar de Maxima y, posteriormente, se imprime en pantalla.

5.2.2 Diseño

1. **Validación de una cadena ingresada en el lenguaje L.**

```
    INICIO

// Definir los tipos de tokens
TIPO TokenType
    NUM         // Número (3, 42, 3.14)
    VAR         // Variable ("x")
    FUNC        // Función matemática ("sin", "cos", etc.)
    OPERATOR    // Operador (+, -, *, /)
    OPEN_PAREN  // Paréntesis de apertura ("(")
    CLOSE_PAREN// Paréntesis de cierre (")")
    UNKNOWN     // Carácter desconocido

TIPO Token
    type: TokenType
    value: STRING

// Clase para representar nodos del árbol de derivación
CLASE DerivationNode
    ATRIBUTOS:
        value: STRING
        children: LIST<DerivationNode>
    MÉTODOS:
        CONSTRUCTOR(value: STRING)
            INICIALIZAR value
            INICIALIZAR children COMO LISTA VACÍA

// Función para tokenizar la entrada
FUNC Tokenize(input: STRING) -> LIST<Token>
    tokens = LISTA VACÍA
    i = 0
    MIENTRAS i < LENGTH(input)
        char = input[i]

        // Ignorar espacios
        SI char ES espacio_blanco
            i += 1
            CONTINUAR

        // Detectar números
        SI char ES dígito
            número = ""
            MIENTRAS char ES dígito O char ES "."
                número += char
                i += 1
                char = input[i]
            AGREGAR Token(NUM, número) A tokens

        // Detectar funciones o variables
        SI char ES letra
            identificador = ""
            MIENTRAS char ES letra
                identificador += char
```

```
                i += 1
                char = input[i]
            SI identificador EN ["sin", "cos", "tan", "ln", "e"]
                AGREGAR Token(FUNC, identificador) A tokens
            SINO SI identificador == "x"
                AGREGAR Token(VAR, identificador) A tokens
            SINO
                AGREGAR Token(UNKNOWN, identificador) A tokens

        // Detectar operadores
        SI char EN ["+", "-", "*", "/"]
            AGREGAR Token(OPERATOR, char) A tokens
            i += 1

        // Detectar paréntesis
        SI char == "("
            AGREGAR Token(OPEN_PAREN, "(") A tokens
            i += 1

        SI char == ")"
            AGREGAR Token(CLOSE_PAREN, ")") A tokens
            i += 1

        // Caracter desconocido
        SINO
            AGREGAR Token(UNKNOWN, char) A tokens
            i += 1

    RETORNAR tokens

// Función principal de parseo
FUNC Parse(tokens: LIST<Token>) -> DerivationNode
    index = 0

    FUNC CrearNodo(value: STRING) -> DerivationNode
        RETORNAR NUEVO DerivationNode(value)

    FUNC ParseExpr() -> DerivationNode
        nodo = CrearNodo("<EXPR>")
        nodo_term = ParseTerm()
        SI nodo_term ES NULO, RETORNAR NULO
        AgregarHijo(nodo, nodo_term)

        MIENTRAS index < LENGTH(tokens) Y token_actual ES ["+", "-"]
            op = token_actual.value
            index += 1
            nodo_op = CrearNodo(op)
            nodo_term_next = ParseTerm()
            SI nodo_term_next ES NULO, RETORNAR NULO
            AgregarHijo(nodo, nodo_op)
            AgregarHijo(nodo, nodo_term_next)

        RETORNAR nodo

    FUNC ParseTerm() -> DerivationNode
        nodo = CrearNodo("<TERM>")
        nodo_factor = ParseFactor()
```

```
        SI nodo_factor ES NULO, RETORNAR NULO
        AgregarHijo(nodo, nodo_factor)

        MIENTRAS index < LENGTH(tokens) Y token_actual ES ["*", "/"]
            op = token_actual.value
            index += 1
            nodo_op = CrearNodo(op)
            nodo_factor_next = ParseFactor()
            SI nodo_factor_next ES NULO, RETORNAR NULO
            AgregarHijo(nodo, nodo_op)
            AgregarHijo(nodo, nodo_factor_next)

        RETORNAR nodo

    FUNC ParseFactor() -> DerivationNode
        nodo = CrearNodo("<FACT>")

        SI token_actual.type == NUM
            AgregarHijo(nodo, CrearNodo("NUM:" + token_actual.value))
            index += 1
            RETORNAR nodo

        SI token_actual.type == VAR
            AgregarHijo(nodo, CrearNodo("VAR:" + token_actual.value))
            index += 1
            RETORNAR nodo

        SI token_actual.type == FUNC
            func_name = token_actual.value
            index += 1
            SI token_actual.type == OPEN_PAREN
                index += 1
                nodo_expr = ParseExpr()
                SI nodo_expr ES NULO, RETORNAR NULO
                AgregarHijo(nodo, CrearNodo("FUNC:" + func_name))
                AgregarHijo(nodo, nodo_expr)
                SI token_actual.type == CLOSE_PAREN
                    index += 1
                SINO
                    RETORNAR NULO
            SINO
                RETORNAR NULO

        SI token_actual.type == OPEN_PAREN
            index += 1
            nodo_expr = ParseExpr()
            SI nodo_expr ES NULO, RETORNAR NULO
            AgregarHijo(nodo, nodo_expr)
            SI token_actual.type == CLOSE_PAREN
                index += 1
            SINO
                RETORNAR NULO
            RETORNAR nodo

        RETORNAR NULO

    RETORNAR ParseExpr()
```

```
// Función para validar expresiones
FUNC ValidateExpression(expression: STRING) -> BOOL
    tokens = Tokenize(expression)
    árbol = Parse(tokens)
    RETORNAR árbol != NULO

// Ejecución principal
MAIN
    entrada = "3 * sin(x)"
    SI ValidateExpression(entrada)
        IMPRIMIR "La expresión es válida."
    SINO
        IMPRIMIR "La expresión no es válida."

FIN
```

2. **Generación de una cadena para Maxima a partir del árbol de expresión.**

```
ENUM TipoNodo
    NUMERO
    OPERADOR
    FUNCION
    VARIABLE
FIN ENUM

ESTRUCTURA NodoExpresion
    CADENA dato
    TipoNodo tipo
    NodoExpresion* izquierdo
    NodoExpresion* derecho

    FUNCION CONSTRUCTOR(dato, tipo)
        this.dato = dato
        this.tipo = tipo
        this.izquierdo = NULO
        this.derecho = NULO
    FIN CONSTRUCTOR

    FUNCION DESTRUCTOR()
        ELIMINAR izquierdo
        ELIMINAR derecho
    FIN DESTRUCTOR
FIN ESTRUCTURA

FUNCION esOperador(caracter)
    DEVOLVER caracter es '+' O caracter es '-' O caracter es '*' O caracter es '/' O
    ↪ caracter es '^'
FIN FUNCION

FUNCION precedencia(operador)
    SI operador es '+' O operador es '-' ENTONCES
        DEVOLVER 1
    SINO SI operador es '*' O operador es '/' ENTONCES
        DEVOLVER 2
    SINO SI operador es '^' ENTONCES
        DEVOLVER 3
    SINO
        DEVOLVER 0
    FIN SI
FIN FUNCION

FUNCION esFuncion(cadena)
    DEVOLVER cadena es "sin" O cadena es "cos" O cadena es "tan" O cadena es "ln" O
    ↪ cadena es "sqrt"
FIN FUNCION

FUNCION esVariable(caracter)
    DEVOLVER caracter es una letra Y NO es un dígito
FIN FUNCION

FUNCION analizarExpresion(expresion, indice)
```

```
PILA valores
PILA operadores

MIENTRAS indice < longitud(expresion)
    SI el carácter en expresion[indice] es espacio
        INCREMENTAR indice
        CONTINUAR
    FIN SI

    SI el carácter en expresion[indice] es una letra
        INICIAR cadena vacía
        MIENTRAS el carácter en expresion[indice] es una letra
            añadir caracter a la cadena
            INCREMENTAR indice
        FIN MIENTRAS

        SI cadena es una función
            SI el siguiente carácter es '('
                INCREMENTAR indice
                nodoArg = analizarExpresion(expresion, indice)
                SI el carácter en expresion[indice] es ')'
                    INCREMENTAR indice
                    nodoFuncion = nuevo NodoExpresion(cadena, FUNCION)
                    nodoFuncion.derecho = nodoArg
                    agregar nodoFuncion a la pila de valores
                SINO
                    LANCER error "Paréntesis faltante después de la función."
                FIN SI
            SINO
                agregar nodo de variable a la pila de valores
            FIN SI
        SINO
            agregar nodo de variable a la pila de valores
        FIN SI
    SINO SI el carácter en expresion[indice] es un dígito O '.'
        INICIAR cadena vacía
        MIENTRAS el carácter en expresion[indice] es un dígito O '.'
            añadir caracter a la cadena
            INCREMENTAR indice
        FIN MIENTRAS
        agregar nodo de número a la pila de valores
    SINO SI el carácter en expresion[indice] es un operador
        MIENTRAS la pila de operadores NO est vaca Y la precedencia del operador
        ↪  superior es mayor o igual que la precedencia del operador actual
            POP operador
            POP nodo derecho
            POP nodo izquierdo
            crear nodo operador con los nodos izquierdo y derecho
            agregar a la pila de valores
        FIN MIENTRAS
        agregar operador actual a la pila de operadores
        INCREMENTAR indice
    SINO SI el carácter en expresion[indice] es '('
        INCREMENTAR indice
        agregar el resultado de analizarExpresion a la pila de valores
    SINO SI el carácter en expresion[indice] es ')'
        romper el ciclo
```

```
        SINO
            LANCER error "Carácter inesperado"
        FIN SI
    FIN MIENTRAS

    MIENTRAS la pila de operadores NO esté vacía
        POP operador
        POP nodo derecho
        POP nodo izquierdo
        crear nodo operador con los nodos izquierdo y derecho
        agregar a la pila de valores
    FIN MIENTRAS

    DEVOLVER nodo superior de la pila de valores
FIN FUNCION

FUNCION convertirAFormatoMaxima(nodo, esPadre)
    SI nodo es NULO
        DEVOLVER cadena vacía
    FIN SI

    SI nodo.tipo es FUNCION
        DEVOLVER cadena "funcion(nodo.derecho)"
    SINO SI nodo.tipo es OPERADOR
        izquierda = convertirAFormatoMaxima(nodo.izquierdo, verdadero)
        derecha = convertirAFormatoMaxima(nodo.derecho, verdadero)
        expresion = izquierda + " " + nodo.dato + " " + derecha
        SI esPadre
            DEVOLVER "(" + expresion + ")"
        SINO
            DEVOLVER expresion
    SINO SI nodo.tipo es NUMERO O VARIABLE
        DEVOLVER nodo.dato
    FIN SI
FIN FUNCION

FUNCION convertirExpresionAFormatoMaxima(expresion)
    iniciar indice en 0
    nodoRaiz = analizarExpresion(expresion, indice)
    SI indice NO es igual a la longitud de la expresión
        LANCER error "Carácter inesperado en la posición de la expresión"
    FIN SI
    resultado = convertirAFormatoMaxima(nodoRaiz)
    ELIMINAR nodoRaiz
    DEVOLVER resultado
FIN FUNCION

FUNCION cadena(expresion)
    convertirMaxima = convertirExpresionAFormatoMaxima(expresion)

    SI encontrar(convertirMaxima)
        DEVOLVER convertirMaxima
    SINO
        DEVOLVER cadena vacía
FIN FUNCION
```

3. **Enviar cadena a Máxima y procesar resultado.**

```
FUNCION escribirScriptMaxima(expresion)
    ABRIR archivo "script.mac" PARA escritura
    ESCRIBIR "display2d:false;" AL archivo
    ESCRIBIR "string(integrate(expresion, x));" AL archivo
    ESCRIBIR "quit();" AL archivo
    CERRAR archivo
FIN FUNCION

FUNCION ejecutarScriptMaxima()
    DEFINIR comandoMaxima COMO "\"C:\\maxima-5.47.0\\bin\\maxima.bat\" --very-quiet
    ↪  -b script.mac > output.txt"

    EJECUTAR comandoMaxima

    SI el comando falla (valor de retorno diferente de 0)
        IMPRIMIR "Error al ejecutar Maxima" EN consola
    FIN SI

    ABRIR archivo "output.txt" PARA lectura
    DEFINIR salida COMO cadena vacía
    LINEA_NUMERO = 0

    MIENTRAS HAYA más líneas en el archivo
        LEER línea desde el archivo
        INCREMENTAR LINEA_NUMERO
        SI LINEA_NUMERO ES 7
            ASIGNAR línea a salida
        FIN SI
    FIN MIENTRAS

    DEVOLVER salida
FIN FUNCION

FUNCION obtenerExpresion(expresion)
    DEFINIR cadena COMO llamar cadena(expresion)
    LLAMAR escribirScriptMaxima(cadena)

    DEVOLVER llamar ejecutarScriptMaxima()
FIN FUNCION
```

4. **Lista enlazada simple para el almacenamiento de resultados.**

```
    Clase NodeList:
    Atributos:
        link: Puntero al siguiente nodo
        expression: Cadena de texto que almacena la expresión

    Métodos:
        Constructor(link, expression):
            link = valor de link
            expression = valor de expression

        getterLink():
            Retornar link

        setLink(newLink):
            link = newLink

        getExpression():
            Retornar expression

Clase List:
    Atributos:
        head: Puntero al primer nodo de la lista (inicialmente nulo)

    Métodos:
        Constructor():
            head = nulo

        addNodeList(expression):
            Crear newNode como un nuevo NodeList con:
                link = nulo
                expression = expresión recibida
            Si head es nulo:
                head = newNode
            Sino:
                aux = head
                Mientras aux.getterLink() no sea nulo:
                    aux = aux.getterLink()
                aux.setLink(newNode) // Conectar el último nodo al nuevo nodo

        getLastNode():
            Si head es nulo:
                Retornar "Lista vacía"
            Sino:
                aux = head
                Mientras aux.getterLink() no sea nulo:
                    aux = aux.getterLink()
                Retornar aux.getExpression()
```

4. **Interfaz**:

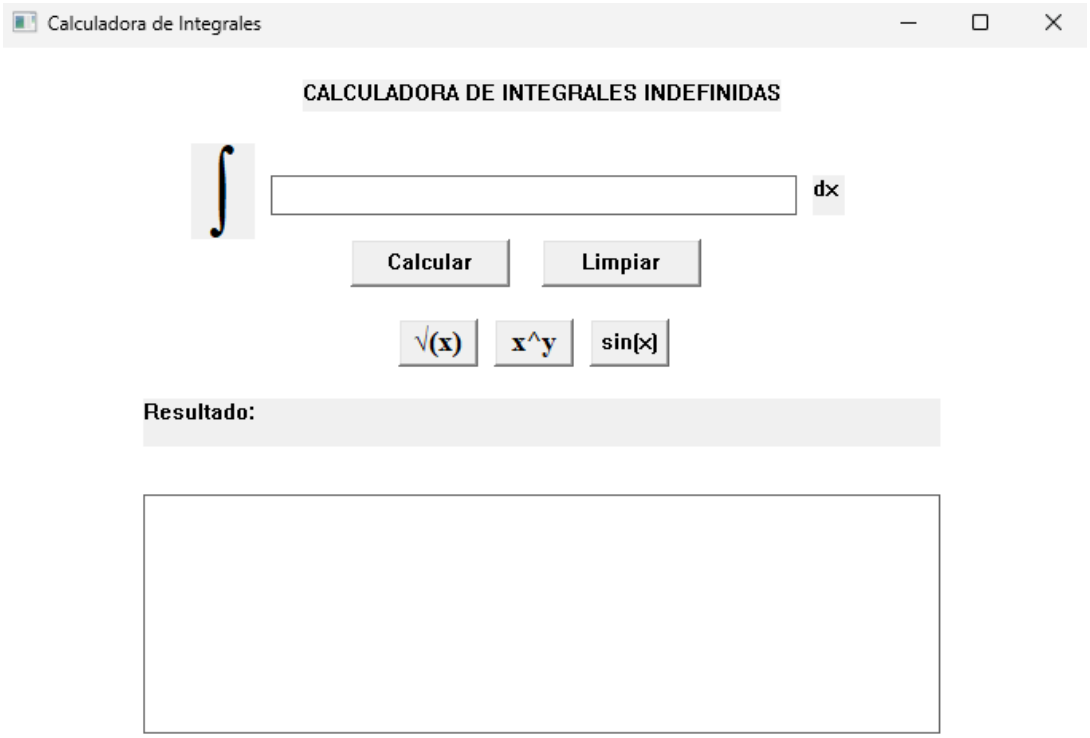


Figure 2: Interfaz de Calculadora

Para la implementación de la interfaz se utilizó la API de Windows (WinAPI). La interfaz es capaz de manejar eventos mediante el uso de mensajes del sistema operativo.

Debajo de **Resultado**, se guardarán resultados calculados previamente.

5.2.3 Implementación

1. **Validación de una cadena ingresada en el lenguaje L. (ValidarExpresiones.cpp)**

```cpp
#include <iostream>
#include <memory>
#include <string>
#include <vector>
#include <stdexcept>
#include <cctype>
using namespace std;

// Nodo del árbol de derivación
class Node
{
public:
    string value;                       // Valor del nodo
    vector<shared_ptr<Node>> children; // Hijos del nodo

    Node(const string &val) : value(val) {}
};

// Generar el árbol
class Parser
{
private:
    string expr; // Expresión a analizar
    size_t pos;  // Posición actual en la expresión

    void skipSpaces()
    {
        while (pos < expr.size() && isspace(expr[pos]))
            ++pos;
    }

    void throwError(const string &message) const
    {
        throw runtime_error("Error en la posicin " + to_string(pos) + ": " +
        ↪  message);
    }

    // Construir árbol para las reglas de la gramática
    shared_ptr<Node> parseExpr()
    {
        auto node = parseTerm();

        skipSpaces();
        while (pos < expr.size() && (expr[pos] == '+' || expr[pos] == '-'))
        {
            char op = expr[pos++];
            skipSpaces();
            auto right = parseTerm();
            auto parent = make_shared<Node>(string(1, op)); //  crear un nuevo nodo
            ↪  en el rbol de derivacin que represente un operador matemtico
            parent->children.push_back(node);
```

```
            parent->children.push_back(right);
            node = parent;
        }
        return node;
    }

    shared_ptr<Node> parseTerm()
    {
        auto node = parseFactor();

        skipSpaces();
        while (pos < expr.size() && (expr[pos] == '*' || expr[pos] == '/'))
        {
            char op = expr[pos++];
            skipSpaces();
            auto right = parseFactor();
            auto parent = make_shared<Node>(string(1, op));
            parent->children.push_back(node);
            parent->children.push_back(right);
            node = parent;
        }
        return node;
    }

    shared_ptr<Node> parseFactor()
    {
        skipSpaces();
        if (pos >= expr.size())
            throwError("Falta un factor");

        // Manejar signos unarios (- o +)
        if (expr[pos] == '-')
        {
            ++pos;
            skipSpaces();
            auto child = parseFactor();
            auto parent = make_shared<Node>("-");
            parent->children.push_back(child);
            return parent;
        }

        if (isdigit(expr[pos]) || expr[pos] == '.')
        { // Números
            size_t start = pos;
            while (pos < expr.size() && (isdigit(expr[pos]) || expr[pos] == '.'))
                ++pos;
            return make_shared<Node>(expr.substr(start, pos - start));
        }

        if (expr[pos] == '(')
        {          // Paréntesis
            ++pos; // Consumir '('
            auto node = parseExpr();
            skipSpaces();
            if (pos >= expr.size() || expr[pos] != ')')
                throwError("Falta el paréntesis de cierre");
            ++pos; // Consumir ')'
```

```
            return node;
        }

        if (expr[pos] == 'x')
        { // Variables
            ++pos;
            return make_shared<Node>("x");
        }

        if (expr.substr(pos, 3) == "sin" || expr.substr(pos, 3) == "cos" ||
        ↪  expr.substr(pos, 3) == "tan")
        {
            return parseFunction();
        }

        if (expr.substr(pos, 2) == "ln")
        {
            return parseFunction();
        }

        // Símbolo raíz
        if (expr.substr(pos, 2) == "" || expr.substr(pos, 4) == "sqrt")
        { // Raíz cuadrada
            string rootSymbol = (expr.substr(pos, 2) == "") ? "" : "sqrt";
            pos += (rootSymbol == "") ? 2 : 4; // Consumir '' o 'sqrt'
            skipSpaces();
            if (pos >= expr.size() || expr[pos] != '(')
            {
                throwError("Falta el paréntesis de apertura después de la raíz");
            }
            ++pos; // Consumir '('
            auto child = parseExpr();
            skipSpaces();
            if (pos >= expr.size() || expr[pos] != ')')
            {
                throwError("Falta el paréntesis de cierre para la raíz");
            }
            ++pos; // Consumir ')'
            auto parent = make_shared<Node>(rootSymbol);
            parent->children.push_back(child);
            return parent;
        }

        throwError("Factor desconocido");
        return nullptr; // Nunca se alcanza
    }

    shared_ptr<Node> parseFunction()
    {
        skipSpaces();
        string func;
        if (expr.substr(pos, 3) == "sin" || expr.substr(pos, 3) == "cos" ||
        ↪  expr.substr(pos, 3) == "tan")
        {
            func = expr.substr(pos, 3);
            pos += 3;
        }
```

```
        else if (expr.substr(pos, 2) == "ln")
        {
            func = expr.substr(pos, 2);
            pos += 2;
        }
        else
        {
            throwError("Función desconocida");
        }

        skipSpaces();
        if (pos >= expr.size() || expr[pos] != '(')
        {
            throwError("Falta el paréntesis de apertura en la función");
        }
        ++pos; // Consumir '('
        auto child = parseExpr();
        skipSpaces();
        if (pos >= expr.size() || expr[pos] != ')')
        {
            throwError("Falta el paréntesis de cierre en la función");
        }
        ++pos; // Consumir ')'

        auto parent = make_shared<Node>(func);
        parent->children.push_back(child);
        return parent;
    }

public:
    Parser(const string &expression) : expr(expression), pos(0) {}

    shared_ptr<Node> parse()
    {
        auto root = parseExpr();
        skipSpaces();
        if (pos < expr.size())
        {
            throwError("Expresión inválida: caracteres adicionales encontrados");
        }
        return root;
    }
};

bool findOut(const string &input)
{
    try
    {
        Parser parser(input);
        parser.parse();
        return true;
    }
    catch (...)
    {
        return false;
    }
}
```

2. **Generación de una cadena para Maxima a partir del árbol de expresión. (ArbolDeExpresion.cpp)**

```
#include
↪  "C:\Users\51993\Desktop\C++POO\poo\ValidarExpresiones\ValidarExpresiones.cpp"
#include <iostream>
#include <string>
#include <stack>
#include <cctype>
#include <sstream>
#include <stdexcept>

using namespace std;

// Se define los tipos de nodos
enum NodeType
{
    NUMBER,
    OPERATOR,
    FUNCTION,
    VARIABLE
};

// Estructura de nodo para el árbol de expresión
struct ExpressionNode
{
    string data;
    NodeType type;
    ExpressionNode* left;
    ExpressionNode* right;

    ExpressionNode(const string& d, NodeType t) : data(d), type(t), left(nullptr),
    ↪  right(nullptr) {}
    ~ExpressionNode()
    {
        delete left;
        delete right;
    }
};

// Verificar si un carácter es un operador
bool isOperator(char c)
{
    return c == '+' || c == '-' || c == '*' || c == '/' || c == '^';
}

// FObtener precedencia de los operadores
int precedence(char op)
{
    if (op == '+' || op == '-')
        return 1;
    if (op == '*' || op == '/')
        return 2;
    if (op == '^')
        return 3;
    return 0;
}
```

```
// Verificar si una cadena es una función matemática
bool isFunction(const string& str)
{
    return str == "sin" || str == "cos" || str == "tan" || str == "ln" || str ==
    ↪  "sqrt";
}

//  verificar si un carácter es una letra (una variable)
bool isVariable(char c)
{
    return isalpha(c) && !isdigit(c); // solo letras, excluyendo números
}

// Crear un árbol de expresión a partir de una cadena
ExpressionNode* parseExpression(const string& expression, size_t& i)
{
    stack<ExpressionNode*> values;
    stack<char> operators;

    while (i < expression.size())
    {
        if (isspace(expression[i]))
        {
            i++;
            continue;
        }

        if (isalpha(expression[i]))
        {
            string token;
            while (i < expression.size() && isalpha(expression[i]))
            {
                token += expression[i++];
            }

            if (isFunction(token))
            {
                if (i < expression.size() && expression[i] == '(')
                {
                    i++; // Saltar '('
                    ExpressionNode* arg = parseExpression(expression, i);
                    if (i >= expression.size() || expression[i] != ')')
                    {
                        throw runtime_error("Falta un parntesis de cierre para la
                        ↪  funcin.");
                    }
                    i++; // Saltar ')'
                    ExpressionNode* funcNode = new ExpressionNode(token, FUNCTION);
                    funcNode->right = arg;
                    values.push(funcNode);
                }
                else
                {
                    throw runtime_error("Parntesis esperado despus de la funcin: " +
                    ↪  token);
                }
```

```
            }
            else
            {
                values.push(new ExpressionNode(token, VARIABLE));
            }
        }
        else if (isdigit(expression[i]) || expression[i] == '.')
        {
            string num;
            while (i < expression.size() && (isdigit(expression[i]) || expression[i]
            ↪  == '.'))
            {
                num += expression[i++];
            }

            // Verificamos si es seguido de una variable o parntesis, lo que indica
            ↪  multiplicacin implcita
            if (i < expression.size() && (isVariable(expression[i]) || expression[i]
            ↪  == '('))
            {
                string var_or_paren(1, expression[i]);
                i++;                                        // Saltamos el carcter que
                ↪  es una variable o parntesis
                string num_var = num + "*" + var_or_paren; // Multiplicacin implcita
                values.push(new ExpressionNode(num_var, NUMBER));
            }
            else
            {
                values.push(new ExpressionNode(num, NUMBER));
            }
        }
        else if (isOperator(expression[i]))
        {
            while (!operators.empty() && precedence(operators.top()) >=
            ↪  precedence(expression[i]))
            {
                char op = operators.top();
                operators.pop();
                ExpressionNode* right = values.top();
                values.pop();
                ExpressionNode* left = values.top();
                values.pop();
                ExpressionNode* opNode = new ExpressionNode(string(1, op), OPERATOR);
                opNode->left = left;
                opNode->right = right;
                values.push(opNode);
            }
            operators.push(expression[i]);
            i++;
        }
        else if (expression[i] == '(')
        {
            i++; // Saltar '('
            values.push(parseExpression(expression, i));
        }
        else if (expression[i] == ')')
        {
```

```
            break;
        }
        else
        {
            throw runtime_error("Carácter inesperado: " + string(1, expression[i]));
        }
    }

    while (!operators.empty())
    {
        char op = operators.top();
        operators.pop();
        ExpressionNode* right = values.top();
        values.pop();
        ExpressionNode* left = values.top();
        values.pop();
        ExpressionNode* opNode = new ExpressionNode(string(1, op), OPERATOR);
        opNode->left = left;
        opNode->right = right;
        values.push(opNode);
    }

    return values.empty() ? nullptr : values.top();
}

// Convertir el árbol de expresión a formato Maxima
string toMaximaString(ExpressionNode* node, bool isParent = false)
{
    if (!node)
        return "";

    if (node->type == FUNCTION)
    {
        return node->data + "(" + toMaximaString(node->right, false) + ")";
    }
    else if (node->type == OPERATOR)
    {
        string left = toMaximaString(node->left, true);
        string right = toMaximaString(node->right, true);

        string expr = left + " " + node->data + " " + right;
        if (isParent)
        {
            return "(" + expr + ")";
        }
        else
        {
            return expr;
        }
    }
    else if (node->type == NUMBER || node->type == VARIABLE)
    {
        return node->data;
    }

    return "";
}
```

```
// Convertir una expresión infija (2x + 1) al formato de Maxima
string convertToMaximaFormat(const string& expression)
{
    size_t i = 0;
    ExpressionNode* root = parseExpression(expression, i);
    if (i != expression.size())
    {
        throw runtime_error("Error al analizar la expresin. Carcter inesperado en la
        ↪  posicin " + to_string(i));
    }
    string result = toMaximaString(root);
    delete root;
    return result;
}

string chain(const string& expression)
{
    string convertToMaxima = convertToMaximaFormat(expression);

    if (findOut(convertToMaxima))
    {
        return convertToMaxima;
    }

    return "";
}
```

3. Enviar cadena a Máxima y procesar resultado. (Maxima.cpp)

```cpp
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <string>
#include "C:\Users\51993\Desktop\C++POO\poo\ArbolDeExpresion\ArbolDeExpresion.cpp"

void writeMaximaScript(const std::string &expression)
{
    std::ofstream file("script.mac");
    file << "display2d:false;\n";                              // Evita salida gráfica
    file << "string(integrate(" << expression << ", x));\n"; // Genera salida como
    ↪  texto simple
    file << "quit();\n";                                       // Termina Maxima
    file.close();
}

std::string runMaximaScript()
{
    const char *maximaCommand = "\"C:\\maxima-5.47.0\\bin\\maxima.bat\" --very-quiet
    ↪  -b script.mac > output.txt";

    // Ejecutar Maxima redirigiendo la salida al archivo
    int result = std::system(maximaCommand);
    if (result != 0)
    {
        std::cerr << "Error al ejecutar Maxima. Cdigo de salida: " << result <<
        ↪  std::endl;
    }

    std::ifstream resultFile("output.txt");
    std::string line, output;
    int lineNumber = 0;

    // Leer el resultado del archivo
    while (std::getline(resultFile, line))
    {
        lineNumber++;
        if (lineNumber == 7)
        {
            output = line; // Devolver la línea 5
        }
    }

    return output;
}

// Obtener la expresión para Maxima
std::string getExpression(const std::string &expression)
{
    std::string str = chain(expression);
    writeMaximaScript(str);

    return runMaximaScript();
}
```

4. **Lista enlazada simple para el almacenamiento de resultados. (ListaDeIntegrales.cpp)**

```cpp
#include <iostream>
#include <string>
#include <vector>

class NodeList
{
    NodeList* link;
    std::string expression;

public:
    NodeList(NodeList* link, std::string expression)
    {
        this->link = link;
        this->expression = expression;
    }

    NodeList* getterLink() { return link; }

    void setLink(NodeList* newLink) { link = newLink; }

    std::string getExpression() { return expression; }
};

class List
{
    NodeList* head;

public:
    List() : head(nullptr) {}

    void addNodeList(std::string expression)
    {
        NodeList* newNode = new NodeList(nullptr, expression);
        if (head == nullptr)
        {
            head = newNode;
        }
        else
        {
            NodeList* aux = head;
            while (aux->getterLink() != nullptr)
            {
                aux = aux->getterLink();
            }
            aux->setLink(newNode); // Conectar el último nodo al nuevo nodo
        }
    }

    // Obtener último nodo
    std::string getLastNode()
    {
        if (head == nullptr) // Si la lista está vacía
            return "Lista vacía";
```

```
        NodeList* aux = head;
        while (aux->getterLink() != nullptr)
        {
            aux = aux->getterLink();
        }

        return aux->getExpression();
    }
};
```

5. **Interfaz**:

```
#include <windows.h>
#include <string>
#include <sstream>
#include <wctype.h>
#include "C:\Users\51993\Desktop\C++POO\poo\Maxima\Maxima.cpp"
#include
↪  "C:\Users\51993\Desktop\C++POO\poo\ListaDeIntegrales\ListaDeIntegrales.cpp" //
↪  Lista de integrales

// Prototipos
LRESULT CALLBACK ProcedimientoVentana(HWND hwnd, UINT msg, WPARAM wParam, LPARAM
↪  lParam);
std::string ResolverIntegral(const std::string &input);

HINSTANCE hInstanciaGlobal; // Instancia global

// Identificadores
#define ID_INPUT 101
#define ID_OUTPUT 102
#define ID_CALCULAR 103
#define ID_RAIZ 104
#define ID_POTENCIA 105
#define ID_PI 106
#define ID_LIMPIAR 107
#define ID_RESULTADOS 1001

// Crear Lista
List resultList;

// Función principal
int main()
{
    return WinMain(GetModuleHandle(NULL), NULL, GetCommandLineA(), SW_SHOWDEFAULT);
}

// Aagregar un resultado
void addResultToList(HWND hwnd, const std::wstring &result)
{
    // Obtener el control LISTBOX
    HWND hresultadosPrevios = GetDlgItem(hwnd, ID_RESULTADOS);
    if (hresultadosPrevios == NULL)
    {
        MessageBox(hwnd, L"No se encontr el control de la lista", L"Error",
        ↪  MB_ICONERROR);
        return;
    }

    int count = SendMessage(hresultadosPrevios, LB_GETCOUNT, 0, 0);
    for (int i = 0; i < count; i++)
    {
        wchar_t buffer[256];
        SendMessage(hresultadosPrevios, LB_GETTEXT, i, (LPARAM)buffer);
        if (result == buffer)
        {
            MessageBox(hwnd, L"El resultado ya existe en la lista.", L"Advertencia",
            ↪  MB_ICONWARNING);
```

```
            return; // Salir sin agregar el resultado
        }
    }

    // Agregar el resultado a la lista
    SendMessage(hresultadosPrevios, LB_ADDSTRING, 0, (LPARAM)result.c_str());
}

int WINAPI WinMain(HINSTANCE hInstanciaActual, HINSTANCE hInstanciaPrevia, LPSTR
↪  lpCmdLinea, int nCmdShow)
{
    HWND ventana;
    MSG mensaje;
    WNDCLASSEX clase;

    clase.cbSize = sizeof(WNDCLASSEX);
    clase.style = CS_HREDRAW | CS_VREDRAW;
    clase.lpfnWndProc = ProcedimientoVentana;
    clase.hInstance = hInstanciaActual;
    clase.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    clase.hIconSm = LoadIcon(NULL, IDI_APPLICATION);
    clase.hCursor = LoadCursor(NULL, IDC_ARROW);
    clase.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
    clase.lpszClassName = L"CalculadoraIntegrales";
    clase.lpszMenuName = NULL;
    clase.cbClsExtra = 0;
    clase.cbWndExtra = 0;

    if (!RegisterClassEx(&clase))
    {
        MessageBox(NULL, L"No se pudo registrar la clase de ventana", L"Error",
        ↪  MB_ICONERROR);
        return EXIT_FAILURE;
    }

    ventana = CreateWindowEx(
        0,
        L"CalculadoraIntegrales",
        L"Calculadora de Integrales",
        WS_OVERLAPPEDWINDOW ^ WS_THICKFRAME, // Ventana sin maximizar
        400, 80, 700, 530,                   // Posición y tamaño
        HWND_DESKTOP,
        NULL,
        hInstanciaActual,
        NULL);

    // Crear un texto estático
    HWND hStaticText = CreateWindowEx(
        0, L"STATIC", L"CALCULADORA DE INTEGRALES INDEFINIDAS", // Texto del control
        WS_CHILD | WS_VISIBLE | SS_LEFT,                    // Estilo del control
        190, 20, 300, 20,                                   // Posicin y tamao del
        ↪  texto
        ventana, NULL, hInstanciaActual, NULL);             // Ventana padre, sin
        ↪  men

    ShowWindow(ventana, nCmdShow);
    UpdateWindow(ventana);
```

```
    // Bucle de mensajes
    while (GetMessage(&mensaje, NULL, 0, 0))
    {
        TranslateMessage(&mensaje);
        DispatchMessage(&mensaje);
    }

    return mensaje.wParam;
}

LRESULT CALLBACK ProcedimientoVentana(HWND hwnd, UINT msg, WPARAM wParam, LPARAM
↪  lParam)
{
    static HWND hIntegralSimbolo, hInput, hDxSimbolo, hOutput, hBoton, hRaizBoton,
    ↪  hPotenciaBoton, hPiBoton, hLimpiarBoton, hAgregarResultado,
    ↪  hResultadosPrevios;
    static HFONT hFuenteGrande;
    bool flag = false;     // Bandera
    std::string resultado; // Almacenar resultado

    switch (msg)
    {
    case WM_CREATE:
    {
        hFuenteGrande = CreateFont(
            65,                        // Altura de la fuente
            0,                         // Ancho
            0, 0,
            FW_BOLD,
            FALSE,                     // Cursiva
            FALSE,                     // Subrayado
            FALSE,
            DEFAULT_CHARSET,
            OUT_DEFAULT_PRECIS,
            CLIP_DEFAULT_PRECIS,
            DEFAULT_QUALITY,           // Calidad
            DEFAULT_PITCH | FF_SWISS,
            L"Times New Roman");       // Nombre de la fuente

        HFONT hFuenteRaiz = CreateFont(
            20,                        // Altura de la fuente
            0,                         // Ancho
            0, 0,
            FW_BOLD,
            FALSE,                     // Cursiva
            FALSE,                     // Subrayado
            FALSE,
            DEFAULT_CHARSET,
            OUT_DEFAULT_PRECIS,
            CLIP_DEFAULT_PRECIS,
            DEFAULT_QUALITY,           // Calidad
            DEFAULT_PITCH | FF_SWISS,
            L"Times New Roman");       // Nombre de la fuente

        // Crear el símbolo de la integral grande
        hIntegralSimbolo = CreateWindowEx(
```

```
        0, L"STATIC", L"\u222b",
        WS_CHILD | WS_VISIBLE | SS_CENTER | SS_NOPREFIX,
        120, 60, 40, 60,
        hwnd, NULL, hInstanciaGlobal, NULL);

    // Aplicar la fuente personalizada
    SendMessage(hIntegralSimbolo, WM_SETFONT, (WPARAM)hFuenteGrande, TRUE);

    // Crear cuadro de texto para ingresar la función
    hInput = CreateWindowEx(
        0, L"EDIT", L"",
        WS_CHILD | WS_VISIBLE | WS_BORDER | ES_AUTOHSCROLL,
        170, 80, 330, 25,
        hwnd, (HMENU)ID_INPUT, hInstanciaGlobal, NULL);

    // Crear "dx"
    hDxSimbolo = CreateWindowEx(
        0, L"STATIC", L"dx",
        WS_CHILD | WS_VISIBLE,
        510, 80, 20, 25,
        hwnd, NULL, hInstanciaGlobal, NULL);

    // Crear botón para calcular la integral
    hBoton = CreateWindowEx(0, L"BUTTON", L"Calcular", WS_CHILD | WS_VISIBLE |
    ↪  BS_PUSHBUTTON,
                            220, 120, 100, 30, hwnd, (HMENU)ID_CALCULAR,
                            ↪  hInstanciaGlobal, NULL);

    // Crear botón para la raíz
    hRaizBoton = CreateWindowEx(
        0, L"BUTTON", L"\u221a(x)", // Símbolo de la raíz cuadrada
        WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
        250, 170, 50, 30,
        hwnd, (HMENU)ID_RAIZ, hInstanciaGlobal, NULL);

    // Aplicar la fuente personalizada
    SendMessage(hRaizBoton, WM_SETFONT, (WPARAM)hFuenteRaiz, TRUE);

    // Crear área para el resultado
    hOutput = CreateWindowEx(
        0, L"STATIC", L"Resultado:",
        WS_CHILD | WS_VISIBLE,
        90, 220, 500, 30,
        hwnd, (HMENU)ID_OUTPUT, hInstanciaGlobal, NULL);

    // Crear botón (simbolo de potencia)
    hPotenciaBoton = CreateWindowEx(
        0, L"BUTTON", L"x^y", // Símbolo de la raíz cuadrada
        WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
        310, 170, 50, 30,
        hwnd, (HMENU)ID_POTENCIA, hInstanciaGlobal, NULL);

    // Aplicar la fuente personalizada
    SendMessage(hPotenciaBoton, WM_SETFONT, (WPARAM)hFuenteRaiz, TRUE);

    // Crear botón (simbolo sin(x))
    hPotenciaBoton = CreateWindowEx(
```

```
            0, L"BUTTON", L"sin(x)",
            WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
            370, 170, 50, 30,
            hwnd, (HMENU)ID_PI, hInstanciaGlobal, NULL);

        // Crear botón de "Limpiar"
        hLimpiarBoton = CreateWindowEx(
            0, L"BUTTON", L"Limpiar",                          // Texto del botón
            WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,             // Estilo del botón
            340, 120, 100, 30,                                 // Posición y tamaño
            hwnd, (HMENU)ID_LIMPIAR, hInstanciaGlobal, NULL); // Identificador de
            ↪  control

        // Crear cuadro para almacenar resultados
        hResultadosPrevios = CreateWindowEx(
            0, L"LISTBOX", NULL,
            WS_CHILD | WS_VISIBLE | WS_BORDER | LBS_NOTIFY | LBS_NOINTEGRALHEIGHT |
            ↪  LBS_MULTIPLESEL,
            90, 280, 500, 150,
            hwnd, (HMENU)ID_RESULTADOS, hInstanciaGlobal, NULL);
        break;
    }

    case WM_COMMAND:
    {
        if (LOWORD(wParam) == ID_CALCULAR)
        {
            wchar_t buffer[256];
            GetWindowText(hInput, buffer, sizeof(buffer) / sizeof(buffer[0])); //
            ↪  Leer entrada del usuario
            std::wstring input(buffer);

            // Resolver integral
            resultado = ResolverIntegral(std::string(input.begin(), input.end()));
            ↪  // Resolver Integral
            addResultToList(hwnd, std::wstring(resultado.begin(), resultado.end()));
            ↪  // Agregar Integral
            flag = true;

            // Mostrar el resultado
            std::wstring textoResultado = L"Resultado: " +
            ↪  std::wstring(resultado.begin(), resultado.end());
            SetWindowText(hOutput, textoResultado.c_str());
        }
        // Responder a los botones
        else if (LOWORD(wParam) == ID_RAIZ)
        {
            // Obtener el texto actual
            wchar_t buffer[256];
            GetWindowText(hInput, buffer, sizeof(buffer) / sizeof(buffer[0]));
            std::wstring input(buffer);

            // Agregar "sqrt()" al final
            if (input.empty() || input.back() != ')')
            {
                input += L"\u221a(x)";
            }
```

```
            else
            {
                input += L"\u221a(x)";
            }

            SetWindowText(hInput, input.c_str()); // Mostrar el nuevo texto
        }
        // Responder al botón de potenciación
        else if (LOWORD(wParam) == ID_POTENCIA)
        {
            // Obtener el texto actual del input
            wchar_t buffer[256];
            GetWindowText(hInput, buffer, sizeof(buffer) / sizeof(buffer[0]));
            std::wstring input(buffer);

            if (!input.empty())
            {
                input += L"^";
            }

            SetWindowText(hInput, input.c_str()); // Mostrar el nuevo texto
        }
        // Responder a los botones (sin(x))
        else if (LOWORD(wParam) == ID_PI)
        {
            // Obtener el texto
            wchar_t buffer[256];
            GetWindowText(hInput, buffer, sizeof(buffer) / sizeof(buffer[0]));
            std::wstring input(buffer);
            input += L"sin(x)";

            SetWindowText(hInput, input.c_str()); // Mostrar el nuevo texto
        }
        else if (LOWORD(wParam) == ID_LIMPIAR)
        {
            SetWindowText(hInput, L"");
        }
        break;
    }

    case WM_DESTROY:
        DeleteObject(hFuenteGrande); // Liberar recursos
        PostQuitMessage(0);
        break;

    default:
        return DefWindowProc(hwnd, msg, wParam, lParam);
    }
    return 0;
}

// Función para resolver integrales
std::string ResolverIntegral(const std::string &input)
{
    return getExpression(input);
}
```

5.2.4 Pruebas

Para la expresión (6) de la página 18:

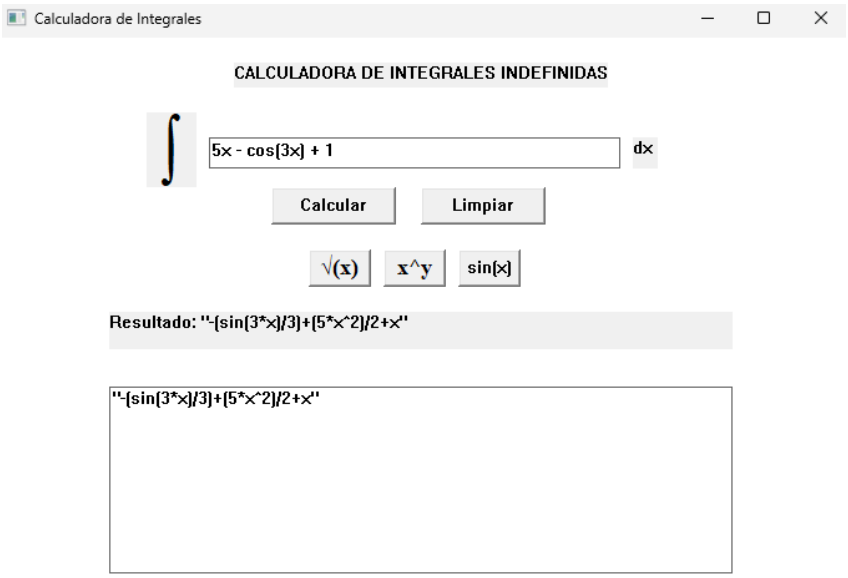


Figure 3: Enter Caption

Se ingresan n expresiones para garantizar el funcionamiento de la calculadora de integrales indefinidas.

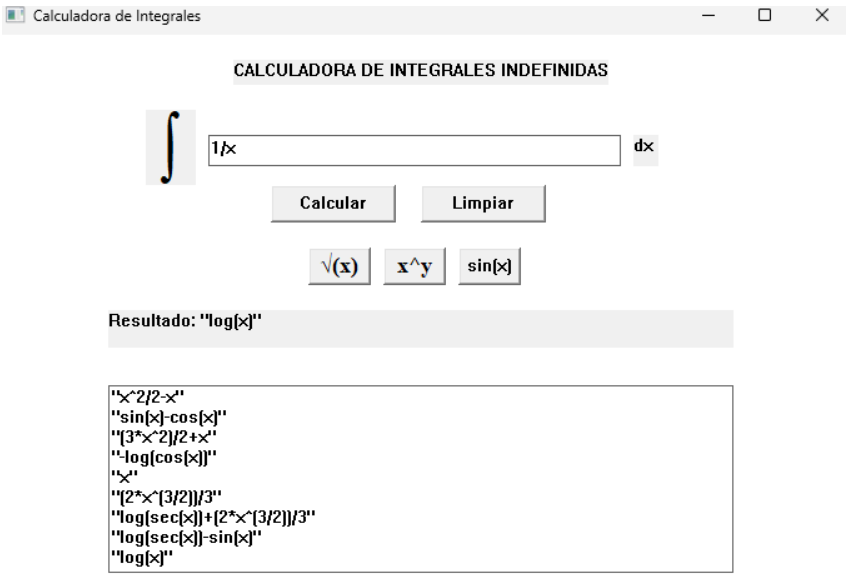


Figure 4: Enter Caption

Sea el caso en el cual una expresión x sea ingresada dos veces.

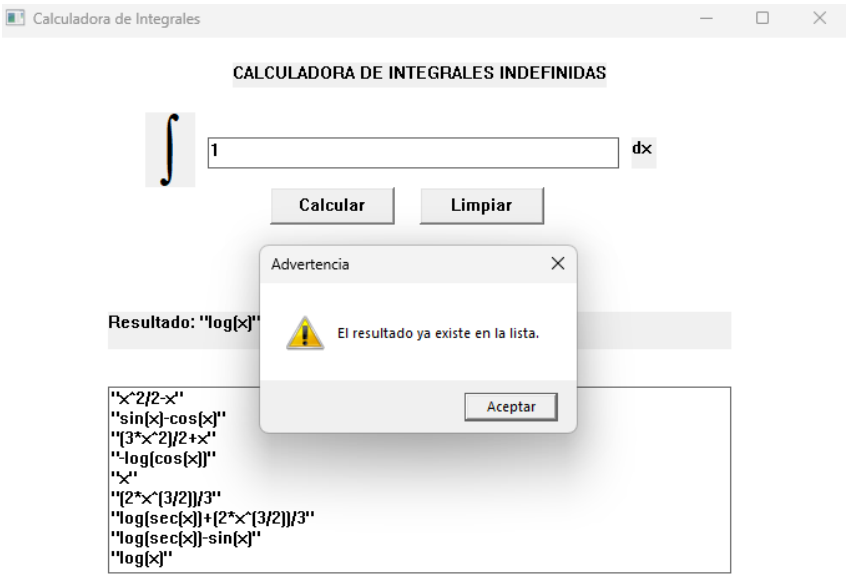


Figure 5: Enter Caption

6 Conclusiones y recomendaciones

En conclusión, la implementación de una calculadora de integrales indefinidas con memoria demuestra ser una herramienta educativa efectiva para reducir la curva de aprendizaje de los estudiantes en cursos de cálculo de una y varias variables. Su diseño e integración en el ámbito académico facilita la comprensión y resolución de problemas matemáticos, abordando directamente las deficiencias en la formación matemática de muchos estudiantes.

La calculadora automatiza cálculos que pueden llegar a ser complejos, también valida la corrección de las expresiones ingresadas a través del diseño de una **Gramática Libre de Contexto** y genera cadenas compatibles con sistemas como Maxima a través del recorrido **in-orden** de un **árbol de expresión**. Considero que el desarrollo de esta calculadora fomenta un aprendizaje más autónomo y eficaz, promoviendo el desarrollo de competencias técnicas esenciales para el desempeño en ingeniería y ciencias aplicadas.

Recomendaciones

- **Ampliación del Lenguaje**: Se recomienda expandir las capacidades de la calculadora para incluir otras funciones avanzadas, haciendo uso de reglas gramaticales más generalizadas.

- **Mejora en la interfaz gráfica**: Una interfaz más amigable y visualmente atractiva podría mejorar la experiencia del usuario, especialmente para estudiantes que inician en el uso de herramientas tecnológicas.

- **Correción de Errores**: La calculadora no reconoce ciertos símbolos del lenguaje actual, así como reglas gramáticales.

- **Capacitación docente**: Incluir sesiones de capacitación para docentes en el uso de la herramienta, promoviendo su adopción como recurso en los cursos de cálculo.

7 Referencias

Cultura Científica (2010). El problema del recorrido del caballo en el tablero de ajedrez.

https://books.google.es/books?hl=es&lr=&id=qXoVzD23DBsC&oi=fnd&pg=PA1&dq=C%2B%2B+estructuras+de+datos&ots=7X-BKXcSZ0&sig=6ILde4d9yHQVhCe5dCYGhToQ4VA

González, A. E. C. Apuntes para el curso de "Estructuras de datos en C/C++".

https://www.academia.edu/download/37329388/ESTRUCTURA_DE_DATOS__2.pdf

Retana, J. Á. G. (2013). La problemática de la enseñanza y el aprendizaje del cálculo para ingeniería. Revista Educación, 37(1), 29-42.

https://www.redalyc.org/pdf/440/44028564002.pdf

Sáez Vacas, F. (1970). La teoría de la computación en las encrucijada de las teorías de autómatas y de la información. Revista de telecomunicación, 99.

https://oa.upm.es/id/eprint/22430