UNIVERSIDAD MAYOR REAL Y PONTIFICIA DE SAN FRANCISCO XAVIER DE CHUQUISACA

FACULTAD: CIENCIAS Y TECNOLOGÍA CARRERA: CIENCIAS DE LA COMPUTACIÓN



***Reconocimiento de Ecuaciones***

Nombre: Univ. Navarro Balanza Igor Santiago

Docente: Ing. Velasquez Aroni Oswaldo Gerardo

Materia: Taller de Especialidad (SCH-130)

Sucre – Bolivia

Contenido

[CAPÍTULO I 1](#_Toc110469609)

[1. GENERALIDADES 1](#_Toc110469610)

[1.1. Introducción 1](#_Toc110469611)

[1.1.1. Título del proyecto 1](#_Toc110469612)

[1.1.2. Teoría de autómatas 1](#_Toc110469613)

[1.2. Antecedentes 3](#_Toc110469614)

[1.3. Objetivos 4](#_Toc110469615)

[1.3.1. Objetivo general 4](#_Toc110469616)

[1.3.2. Objetivos específicos 4](#_Toc110469617)

[CAPÍTULO II 4](#_Toc110469618)

[2. MARCO CONTEXTUAL 4](#_Toc110469619)

[2.1. Contexto 4](#_Toc110469620)

[2.2. Aplicaciones 4](#_Toc110469621)

[CAPÍTULO III 5](#_Toc110469622)

[3. MARCO TEÓRICO 5](#_Toc110469623)

[3.1. Algoritmos 5](#_Toc110469624)

[3.1.1. Autómata Finito Determinista AFD 5](#_Toc110469625)

[3.1.2. Autómata Finito No Determinista AFN 5](#_Toc110469626)

[3.1.3. Autómata Finito No Determinista con Transición Épsilon 6](#_Toc110469627)

[3.1.4. Minimización de autómatas finitos deterministas 6](#_Toc110469628)

[3.1.5. Analizador léxico 6](#_Toc110469629)

[3.1.6. Analizador sintáctico 7](#_Toc110469630)

[3.1.7. Analizador semántico 7](#_Toc110469631)

[3.1.8. Tipos de errores 7](#_Toc110469632)

[3.1.9. Estructura de pilas 8](#_Toc110469633)

[3.1.10. Recursos técnicos 8](#_Toc110469634)

[3.1.11. Recursos Tecnológicos 8](#_Toc110469635)

[CAPÍTULO IV 10](#_Toc110469636)

[4. INGENIERIA DEL PROYECTO 10](#_Toc110469637)

[4.1. Desarrollo del proyecto 10](#_Toc110469638)

[4.2. Capturas de pantalla del sistema 10](#_Toc110469639)

[CONCLUSIONES Y RECOMENDACIONES 15](#_Toc110469640)

[Conclusiones 15](#_Toc110469641)

[Recomendaciones 15](#_Toc110469642)

[Anexos 15](#_Toc110469643)

# CAPÍTULO I

## GENERALIDADES

## Introducción

### Título del proyecto

Sistema de reconocimiento de ecuaciones.

### Teoría de autómatas

La teoría de autómatas es una rama de la teoría de la computación que estudia las máquinas abstractas y los problemas que éstas son capaces de resolver.

Antes de que existieran las computadoras, en la década de los años treinta, A. Turing estudió una máquina abstracta que tenía todas las capacidades de las computadoras de hoy día, al menos en lo que respecta a lo que podían calcular. El objetivo de Turing era describir de forma precisa los límites entre lo que una máquina de cálculo podía y no podía hacer; estas conclusiones no sólo se aplican a las máquinas abstractas de Turing, sino a todas las máquinas reales actuales. (John E. Hopcroft Rajeev Motwani Jeffrey D. Ullman, Teoría de autómatas, 2007, 19).

Orientada a exponer los principios científicos que han dado soporte a la noción matemática de computación: la definición, análisis y diseño de máquinas abstractas.

Estas máquinas son capaces de reconocer lenguajes, que no son más que conjuntos de cadenas que cumplen cierta propiedad.

A finales de la década de los cincuenta, el lingüista N. Chomsky inició el estudio de las “gramáticas” formales. Estas gramáticas están estrechamente relacionadas con los autómatas abstractos y sirven actualmente como base de algunos importantes componentes de software, entre los que se incluyen componentes de los compiladores. (John E. Hopcroft Rajeev Motwani Jeffrey D. Ullman, Teoría de autómatas, 2007, 19).

Un autómata es un modelo matemático para una máquina de estado finito, dada una entrada de símbolos salta a través de una serie de estado de acuerdo a una función de transición. Esta función de transición le dice al autómata a que estado debe cambiar dados determinados estados y símbolos.

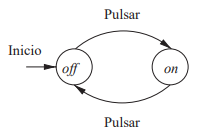
Los autómatas finitos constituyen un modelo útil para muchos tipos de hardware y software.

1. Software para diseñar y probar el comportamiento de circuitos digitales.
2. El “analizador léxico” de un compilador típico, es decir, el componente del compilador que separa el texto de entrada en unidades lógicas, tal como identificadores, palabras clave y signos de puntuación.
3. Software para explorar cuerpos de texto largos, como colecciones de páginas web o para determinar el número de apariciones de palabras, frases u otros patrones.
4. Software para verificar sistemas de todo tipo que tengan un número finito de estados diferentes, tales como protocolos de comunicaciones o protocolos para el intercambio seguro de información.

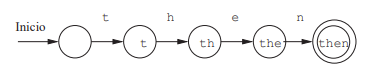
Existen muchos sistemas o componentes que pueden encontrarse siempre en uno de una serie de “estados” finitos. El propósito de un estado es el de recordar la parte relevante del historial del sistema. Puesto que sólo existe un número finito de estados, generalmente, no es posible recordar el historial completo, por lo que el sistema debe diseñarse cuidadosamente, con el fin de recordar lo que es importante y olvidar lo que no lo es. La ventaja de disponer de sólo un número finito de estados es que podemos implementar el sistema mediante un conjunto fijo de recursos. Por ejemplo, podríamos implementarlo por hardware como un circuito, o como una forma simple de programa que puede tomar decisiones consultando sólo una cantidad limitada de datos o utilizando la posición del propio código para tomar la decisión. (John E. Hopcroft Rajeev Motwani Jeffrey D. Ullman, Teoría de autómatas, 2007, 20)

Ejemplos

1. El autómata finito no trivial más simple es un interruptor de apagado/encendido (posiciones on/off). El dispositivo recuerda si está en el estado encendido (“on”) o en el estado apagado (“off”), y permite al usuario pulsar un botón cuyo efecto es diferente dependiendo del estado del interruptor. Es decir, si el interruptor está en el estado off, entonces al pulsar el botón cambia al estado on, y si el interruptor está en el estado on, al pulsar el mismo botón pasa al estado off.



1. En ocasiones, lo que recuerda un estado puede ser mucho más complejo que una elección entre las posiciones apagado/encendido(on/off). En la imagen se muestra otro autómata finito que podría formar parte de un analizador léxico. El trabajo de este autómata consiste en reconocer la palabra clave then, por lo que necesita cinco estados, representando cada uno de ellos la posición dentro de dicha palabra que se ha alcanzado hasta el momento. Estas posiciones se corresponden con los prefijos de la palabra, desde la cadena de caracteres vacía (es decir, cuando no contiene ningún carácter) hasta la palabra completa. Los cinco estados se designan mediante el correspondiente prefijo de then visto hasta el momento. Las entradas se corresponden con las letras. Podemos imaginar que el analizador léxico examina un carácter del programa que se está compilando en un determinado instante, y que el siguiente carácter que se va a examinar es la entrada al autómata. El estado inicial se corresponde con la cadena vacía y cada uno de los estados tiene una transición a la siguiente letra de la palabra then, al estado que corresponde al siguiente prefijo más largo. El estado denominado then se alcanza cuando la entrada está formada por todas las letras de dicho término. Puesto que el trabajo de este autómata es indicar el reconocimiento de la palabra then, podemos considerar dicho estado como el único estado de aceptación.



## Antecedentes

Los modelos abstractos de computación tienen su origen en los años 30, mucho antes de que existieran los ordenadores modernos, en el trabajo de los lógicos Church, Gödel, Kleene, Post y Turing.

Estos primeros trabajos han tenido una profunda influencia no solo en el desarrollo teórico de las Ciencias Computacionales, sino que muchos aspectos de la práctica de la computación que son ahora lugar común de los informáticos, fueron presagiados por ellos; incluyendo la existencia de ordenadores de propósito general, la posibilidad de interpretar programas, la dualidad entre software y hardware, y la representación de lenguajes por estructuras formales basados en reglas de producción.

El punto de partida de estos primeros trabajos fueron las cuestiones fundamentales que D. Hilbert formuló en 1928, durante el transcurso de un congreso internacional:

¿Son completas las matemáticas, en el sentido de que pueda probarse o no cada aseveración matemática?

¿Son las matemáticas consistentes, en el sentido de que no pueda probarse simultáneamente una aseveración y su negación?

¿Son las matemáticas decidibles, en el sentido de que exista un método definido que se pueda aplicar a cualquier aseveración matemática, y que determine si dicha aseveración es cierta?

Por desgracia para Hilbert, en la década de 1930 se produjeron una serie de investigaciones que mostraron que esto no era posible. Las primeras noticias en contra surgen en 1931 con K. Gödel y su Teorema de Incompletitud: «Todo sistema de primer orden consistente que contenga los teoremas de la aritmética y cuyo conjunto de axiomas sea recursivo no es completo.»

El siguiente paso importante lo constituye la aparición casi simultánea en 1936 de varias caracterizaciones independientes de la noción de calculabilidad efectiva, en los trabajos de Church, Kleene, Turing y Post. Los tres primeros mostraban problemas que eran efectivamente indecidibles; Church y Turing probaron además que el Entscheidungsproblem (problema de decisión) era un problema indecidible.

Cuando Turing conoció los trabajos de Church-Kleene, demostró que los conceptos de función λ-deﬁnible y función calculable por medio de una máquina de Turing coinciden. Naturalmente a la luz de esto la Tesis de Turing resulta ser equivalente a la de Church.

Posteriormente, se demostró la equivalencia entre lo que se podía calcular mediante una máquina de Turing y lo que se podía calcular mediante un sistema formal en general. A la vista de estos resultados, la Tesis de Church-Turing es aceptada como un axioma en la teoría de la computación, y ha servido como punto de partida en la investigación de los problemas que se pueden resolver mediante un algoritmo.

## Objetivos

### Objetivo general

Realizar una aplicación de cálculo de expresiones y verificación de sentencias de entrada.

### Objetivos específicos

* + 1. Revisar los aspectos teóricos relacionados a la aplicación del cálculo de expresiones.
    2. Revisar aspectos teóricos relacionados a la estructura de datos.
    3. Utilizar las estructuras de datos para el correcto funcionamiento del sistema.
    4. Analizar la teoría de autómatas.

# CAPÍTULO II

## MARCO CONTEXTUAL

## Contexto

Los autómatas finitos y los autómatas finitos no deterministas tienen como función la aceptación o no aceptación de la salida de acuerdo a los parámetros de entrada, lo que implica una gran aplicación en diferentes campos y no sólo en sistemas informáticos o programas de computación sino también en diferentes ramas como la robótica, industria, control de procesos y electrónica.

## Aplicaciones

Las principales aplicaciones para la teoría de autómatas son:

Búsquedas en la web y extracción de información de textos.

Bots de búsqueda.

Bots compradores automáticos.

Editores de texto para buscar y manipular texto.

Compiladores, que se encargan de traducir un programa escrito en algún lenguaje de programación a otro lenguaje generando un programa equivalente que la máquina será capaz de interpretar.

Equipos electrónicos que realizan procesos secuenciales a nivel industrial, cadenas de montaje.

Control de procesos

Tareas repetitivas

Asistentes de voz, reconocimiento de voz, ejecución de comandos, respuestas.

Conducción automatizada.

Aplicación en videojuegos.

Aplicación de autómatas para el desarrollo de inteligencia artificial

Regulación y control de trafico

Telefonía, autentificación, detección de errores y estados de espera.

Control de ascensores

Sistemas embebidos, microondas, lavadoras.

# CAPÍTULO III

## MARCO TEÓRICO

## Algoritmos

### Autómata Finito Determinista AFD

Estos autómatas solo se limitarán a aceptar o no una determinada cadena recibida en la entrada, por lo tanto, la salida de los mismos solo tendrá dos valores posibles aceptar o no aceptar a la palabra de entrada.

Estos autómatas transitarán entre un conjunto finito de estados posibles, a medida que reciban sucesivamente los caracteres de entrada, en un instante determinado de tiempo el autómata solo podrá estar en uno y solo uno de los estados posibles.

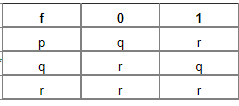
Una característica importante de este tipo de autómatas es el determinismo, lo cual significa que estando en un estado y recibiendo una entrada del exterior el autómata tendrá la posibilidad de transitar a uno y solo un estado del conjunto de estados posibles.

Con respecto al conjunto de estados ahora se pueden clasificar en tres tipos: Estado Inicial, que es pon donde comenzará la ejecución de la máquina; Estados finales o de aceptación que será un subconjunto del conjunto de estados por los que transitará la máquina, y si cuando se hayan terminado de procesar todos los símbolos de entrada y no reste ningún símbolo por leer, la máquina quede posicionada en uno de estos estados de aceptación, se concluirá que la cadena procesada será aceptada por el autómata. y Estados Intermedios, que tienen comportamiento idéntico a los definidos en las máquinas secuenciales.

Definición Formal

AFD = ( ∑ , Q, q0, F, f )

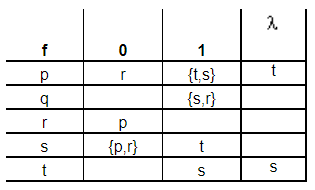
|  |  |
| --- | --- |
| ∑ | Alfabeto de símbolos de entrada. |
| Q | Conjunto finito de estados |
| q0 | q0  Q – estado inicial previsto |
| F | F   Q - es el conjunto de estado finales de aceptación. |
| f | Función de transición de estados definida como   f: Q x ∑  Q |



### Autómata Finito No Determinista AFN

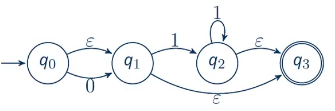
Este tipo de autómatas se diferencia de los AFD, básicamente en cómo puede constituirse la función de transición. Estas diferencias son las siguientes:

1. Para cada par estado entrada, el autómata puede tener la posibilidad de transitar a más de un estado posible.
2. Para algún par de estado entrada, el autómata puede no tener definido ninguna transición. Lo que significa que podrá realizar transición alguna.
3. Puede realizar transiciones de un estado a otro sin leer símbolo alguno de la entrada. A este tipo particular de transiciones se las denomina transiciones- 



### Autómata Finito No Determinista con Transición Épsilon

Un autómata finito no determinista con transición épsilon es un autómata que puede realizar una transición sin consumir el símbolo que actualmente se está leyendo.



### Minimización de autómatas finitos deterministas

Dos estados de un autómata finito determinista son estados equivalentes si al unirse en un solo estado, pueden reconocer el mismo lenguaje regular que si estuviesen separados. Esta unión de estados implica la unión tanto de sus transiciones de entrada como de salida. Si dos estados no son equivalentes, se dice que son estados distinguibles. Un estado final con un estado no-final nunca serán equivalentes.

Un AFD está minimizado, si todos sus estados son distinguibles y alcanzables. Un algoritmo de minimización de AFD es el siguiente:

1. Eliminar los estados inaccesibles del autómata.
2. Construir una tabla con todos los pares (p, q) de estados restantes.
3. Marcar en la tabla aquellas entradas donde un estado es final y el otro es no-final, es decir, aquellos pares de estados que son claramente distinguibles.
4. Para cada par (p, q) y cada símbolo a del alfabeto, tal que r = δ(p,a) y s = δ(q,a):

Si (r, s) ya ha sido marcado, entonces p y q también son distinguibles, por lo tanto, marcar la entrada (p, q).

De lo contrario, colocar (p, q) en una lista asociada a la entrada (r, s).

1. Agrupar los pares de estados no marcados.

### Analizador léxico

Lee la secuencia de caracteres de izquierda a derecha del programa fuente y agrupa la secuencia de caracteres en unidades con significado propio.

Las palabras clave identificadores, operadores, constantes numéricas, signos de puntuación como separadores de sentencia, llaves, parestesias, etcétera. son diversas clasificaciones de componentes léxicos.

Componentes léxicos

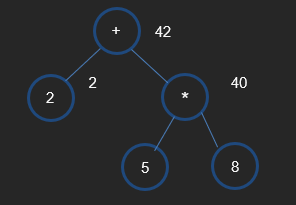
* Palabras reservadas: if, while, do, else, for.
* Identificadores: variables, funciones, tipos definidos por el usuario, etiquetas.
* Operadores: =, +, -, \*, /, ^, %, sin, cos, tan.
* Símbolos especiales: ;, ( ), { }.
* Constantes numéricas que representan valores enteros y flotantes.

### Analizador sintáctico

Determina si la secuencia de componentes léxicos sigue la sintaxis del lenguaje y obtiene la estructura jerárquica del programa en forma de árbol, donde los nodos son las construcciones de alto nivel del lenguaje.

Se determina las relaciones estructurales entre los componentes léxicos esto es semejante a realizar el análisis gramatical sobre una fase en el lenguaje natural. La estructura sintáctica se define mediante las gramáticas independientes del contexto.

2+5\*8



### Analizador semántico

El analizador semántico se encarga de que los tipos que intervienen en las expresiones sean compatibles o que los parámetros reales de una función sean coherentes con los parámetros formales

Funciones principales

Identificar cada tipo de instrucción y sus componentes

Completar la Tabla de Símbolos

Realizar distintas comprobaciones y validaciones:

Comprobaciones de tipos.

Comprobaciones del flujo de control.

Comprobaciones de unicidad.

Comprobaciones de emparejamiento.

### Tipos de errores

Errores léxicos

Cuando un lexema no corresponde a ningún componente léxico definido para el lenguaje.

Ej: 2+5\*8a o cuando no se cierran adecuadamente los paréntesis.

Error sintáctico

Ocurren cuando se intenta introducir una sentencia que no va de acuerdo con las reglas de escritura del lenguaje o gramática.

Error semántico

Un error semántico se produce cuando la sintaxis del código es correcta pero la semántica o significado no es el que se pretendía. La construcción obedece las reglas del lenguaje y por ello el compilador o intérprete no detectan errores semánticos. Un erro semántico puede hacer que el programa termine de forma anormal, con o sin un mensaje de error.

### Estructura de pilas

Una Pila es una colección ordenada de elementos en la que se pueden insertar y eliminar por un extremo llamado tope.

Por tal razón se conoce como una estructura de datos LIFO (last-in, first-out)

Algunas aplicaciones

* Apilar libros, productos.
* Problema de torres de Hanoi.
* Leer una secuencia de caracteres desde teclado e imprimirlos al revés.
* Expresión matemática que contenga paréntesis anidados correctamente.
* Convertir expresiones de notación infija a prefija, a postfija y viceversa.

### Recursos técnicos

Se desarrollará el proyecto en lenguaje Java

### Recursos Tecnológicos

Notación Postfija

El programa se encargará de traducir la expresión que digita el usuario (notación infija) en una expresión más simple (notación postfija), esta forma de escribir una expresión matemática tiene la ventaja de que se evalúa de forma lineal por lo que es más sencillo para una computadora "entenderlo''.

Lo que hace el lenguaje es colocar primero los números con los que va a operar y luego escribe la operación, por ejemplo: a + b se escribiría " a b +''.

Prioridad

1. Las multiplicaciones, las divisiones, las raíces, las potencias, las funciones trigonométricas tienen prioridad sobre la suma y la resta.
2. Si se presenta un paréntesis, se deben realizar primero las operaciones dentro de éste. Si hay un paréntesis dentro de otro tiene prioridad el paréntesis interno.

Evaluación en notación infija y postfija

Tomemos un ejemplo sencillo como 4 + 23, esta expresión se escribe en las notaciones anteriores de la siguiente manera:

Notación infija: 4 + 2 ^ 3

Notación postfija: 4 2 3 ^ +

Para evaluar 4 + 23 en notación infija primero elevar 2 al cubo y luego sumarle 4; se nota que esto no está en forma lineal, es decir, se debe ir primero a la segunda parte de la expresión, evaluarla y luego sumarle al resultado el cuatro.

Es muy complejo hacer que un programa evalué de esta forma. Si la expresión "4 + 23'' se traduce a la forma "4 2 3 ^ +'', entonces se convierte en una expresión más sencilla; para evaluar esta expresión el algoritmo es leer el texto de izquierda a derecha, si se encuentra un número lo apila y las operaciones se irán realizando conforme aparezcan.

Así, en el ejemplo "4 2 3 ^ +'' se toma el 4 y se mete en una pila de números, el segundo valor que se toma el 2, luego se toma el 3 y se mete nuevamente en la pila de números, es decir, tenemos una pila de números como sigue.

|  |
| --- |
| 3 |
| 2 |
| 4 |

Ahora sigue una potencia, por lo que se toman los dos últimos números de la pila (recuerde que tomamos los dos de arriba: 2 y 3) y se realiza la potencia 23 = 8, así el 2 y el 3 se sustituyen en la pila por 8, se obtiene la pila

|  |
| --- |
| 4 |
| 8 |

Por último, sigue un signo de suma, éste también se realiza con dos números, por lo que se toman el 4 y el 8, el resultado es 4 + 8 = 12 y en la pila queda un 12; como ya se acabó la expresión entonces el resultado es 12.

Aunque a primera vista esta forma de evaluar parece más compleja, para un programa no lo es, ya que se siguen los pasos en forma lineal (la expresión se lee de izquierda a derecha), el algoritmo estaría compuesto por tres reglas:

1. Si lo que sigue en la expresión es un número, se agrega a la pila de salida.
2. Si sigue una operación que ocupa dos números (como la suma y la resta) se sacan los dos últimos números de la pila, se realiza la operación y se introduce el resultado en la pila.
3. Si es una función que ocupa un solo número (como seno o coseno) entonces se saca un número de la pila, se evalúa y se guarda el resultado.

De esta manera se observa que evaluar una expresión en notación postfija es más sencillo que hacerlo en notación infija, por lo que lo primero que se hará en el programa es "traducir'' la expresión que digite el usuario a dicha notación.

# CAPÍTULO IV

## INGENIERIA DEL PROYECTO

## Desarrollo del proyecto

Lenguaje de programación

* Java

Entorno de desarrollo

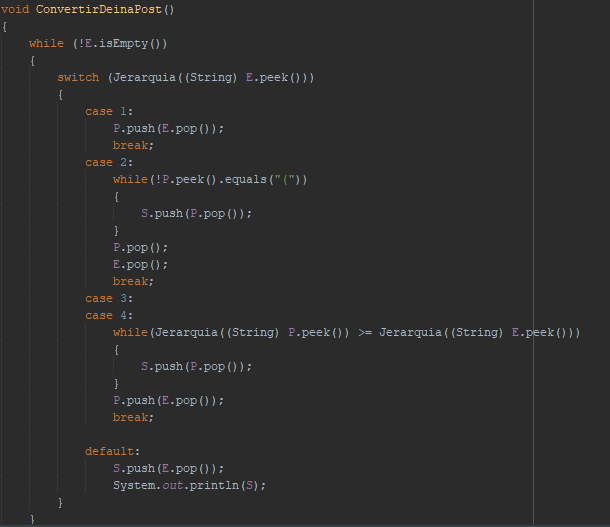
* Netbeans

Recursos técnicos

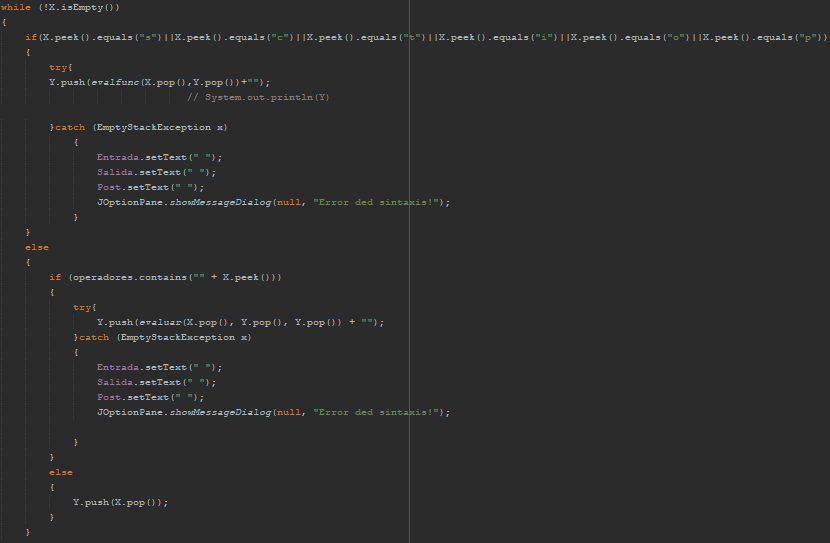
* Implementación de teoría de autómatas

## Capturas de pantalla del sistema

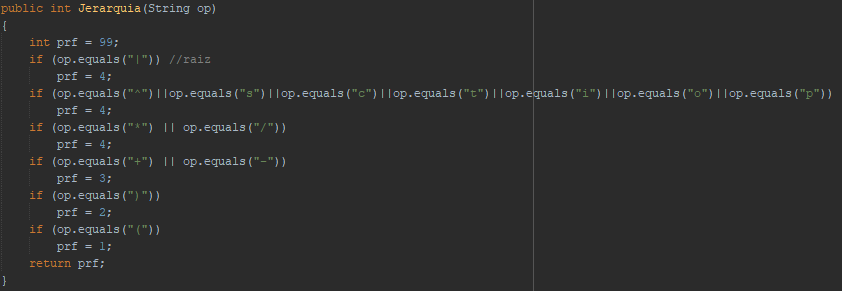
Función para convertir notación de infija a postfija



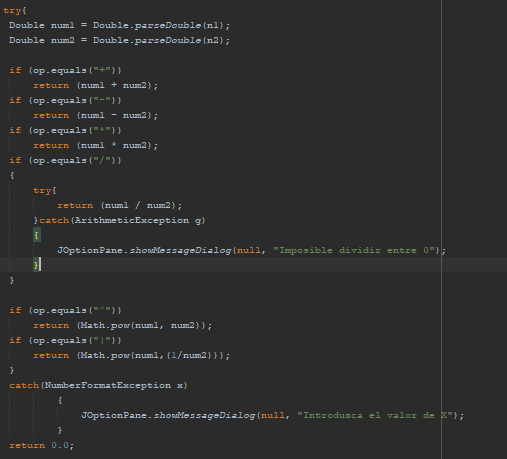
Función para calcular la expresión



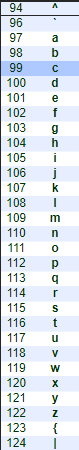
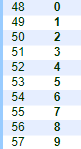
Analizador sintáctico



Analizador semántico

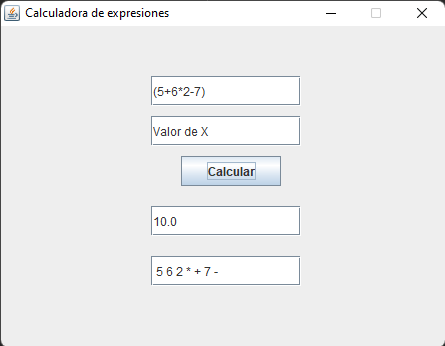


Valores utilizados de código ASCII

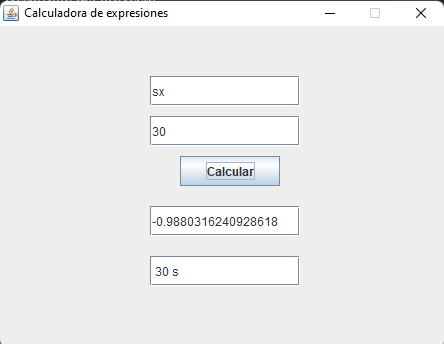


Resultados

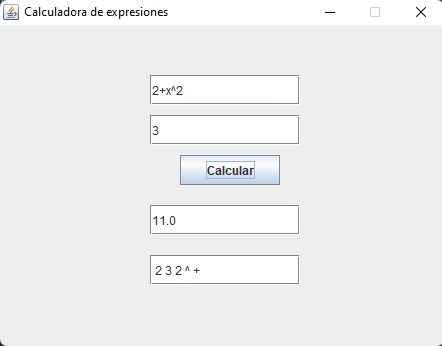
Operación simple de prueba



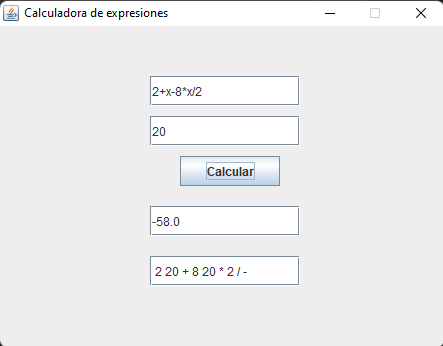
Operación simple con constante



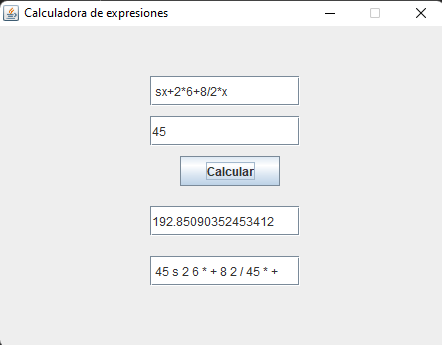
Operación simple con constante



Operación compleja con constante



Operación compleja con constante



# CONCLUSIONES Y RECOMENDACIONES

## Conclusiones

Hoy en día los autómatas son un avance tecnológico de gran escala, ya que facilitan el trabajo físico como en las industrias, implementando autómatas para envasado, control de procesos, etc. La rama de aplicación de los autómatas es muy extensa ya que son muy moldeables de acuerdo a lo necesitado, sus principales ventajas es que poseen gran variedad de modelos, ahorran tiempo en la elaboración de proyectos, su costo es bajo a la hora de implementarlo y el mantenimiento es económico y reducido tiempo. En la actualidad los autómatas son usados en la industria automotriz en el área de ensamblado de piezas, en el control de los ascensores, en el control de tráfico por medio de operaciones con semáforos, principalmente en las telecomunicaciones y en el ruteo. Los autómatas son utilizados como analogías por físicos debido a su asertividad y a su precisión de la realidad, ayudan a entender algoritmos cuando se habla de redes neuronales artificiales o autómatas celulares.

Es posible la aplicación de teoría de autómatas a una calculadora de modo que se valide una sentencia de entrada y esta pueda ser o no validada por el autómata de modo que se pueda seguir el proceso del programar para realizar el cálculo respectivo de la operación matemática introducida.

## Recomendaciones

Se recomienda indagar más en la teoría autómatas ya que potencialmente el campo de aplicación es muy grande, en especial en ámbitos como la inteligencia artificial, con la que se pueden automatizar muchos trabajos de maneras más eficientes aplicándolas en redes neuronales.

# Anexos

Enlace de github del proyecto

https://github.com/santinaba/Calculadora