

Universidad Nacional De Colombia.

Taller 1: Manual técnico



Juan Manuel Torres Leon

Nicolás Cortés Gutiérrez

Gabriel Felipe Fonseca Guerrero

Adrian Alberto Diosa Benavides

Jorge Andres Mora Leon

Lenguajes de programación

Bogotá D.C enero 2025

1. Introducción

En este documento se describe en detalle el diseño e implementación técnica del analizador léxico desarrollado en Java (JFlex) para el dominio de pruebas tipo Friedman y procesamiento de rachas, según las especificaciones del taller.

2. Prototipos de programas (Literal a)

Fundamentos de lenguajes de programación

Lenguajes de propósito específico: Son lenguajes de programación diseñados para resolver problemas concretos dentro de un dominio.

Se usan cuando se requiere expresar una solución clara y concisa en un campo particular o tarea específica. Por ejemplo para:

Configuración de sistemas, Consultas de bases de datos, modelado matemático o financiero, diseño de interfaces gráficas, entre otros.

Dentro de las ventajas de los lenguajes de propósito específico está sus simplicidad y legibilidad, productividad, especialización y mantenibilidad

Paradigma Imperativo: El enfoque de este paradigma se basa en dar instrucciones especificando detalladamente cómo deben ser realizadas definiendo explícitamente el orden y pasos necesarios, donde se manipula el estado del programa, modificando variables y datos a medida que avanza la ejecución del programa.

Con características como:

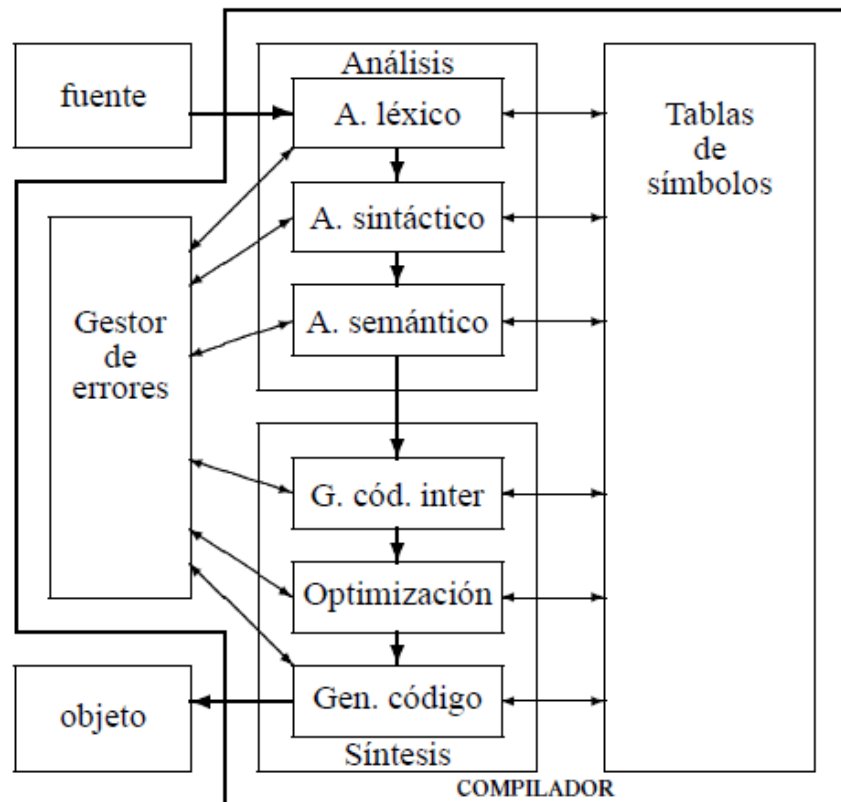
- Secuencia de instrucciones
- Uso de variables y asignaciones
- Estado mutable
- Control explícito del flujo
- Énfasis en el “cómo” hacer las cosas

Tipos Abstractos de Datos: Es una estructura lógica para almacenar y organizar datos. Por ejemplo: Una lista, una pila, una cola, un diccionario entre otros.

Análisis Léxico y Herramientas:

Compiladores y analizadores léxicos: El compilador está encargado de traducir el código fuente escrito de un lenguaje de alto nivel a lenguaje de máquina que es el que es ejecutado por la computadora. El compilador está compuesto de diferentes partes: El Analizador léxico, sintáctico, semántico, generador de código intermedio, optimización y generador código.

Específicamente, el analizador léxico es el encargado de tomar el código fuente y dividirlo en tokens, estos representan palabras reservadas, identificadores, operadores, etc. Revisa que las palabras estén bien escritas y hagan parte del alfabeto



JFlex: Es una herramienta generadora de analizadores léxicos para Java. Permite definir reglas léxicas, usando una especificación escrita de un .flex y genera automáticamente un analizador léxico en Java.

A continuación se presentan dos prototipos esquemáticos de programas escritos en el lenguaje de propósito específico, que podrán ser correctamente traducidos por el compilador futuro.

Prototipo 1:

```
Start;
AMatrix matrix [x][y];
for(int i=0;i<matrix.size){
Vector vec=[1,...,n];
vec=datstream();
matrix[i]=vec;
}
```

```
Datshow(matrix[num]);Datshow(matrix[num][num]);
int num1=matrix[num,num].RachaQ();//cantidad de rachas en la casilla i, j ;
int num2=matrix[num,num].RachaP(5);//promedio de rachas;
```

```
byte num3 = (matrix[num][num].COP);//Cuenta las pruebas de la casilla
```

```
int var = matrix.n*matrix.m
int var5 = matrix.block(num).NProbes();
int var6 = matrix.Trt(num).NProbes();
```

```
for(i=0;i<N;i++){
    var1=matrix.BlockSet(i)+var1;
}
```

```
Rules r;r.add(num>num,"A");
```

```
var1=ROrder(var1);//Reordena el arreglo concatenado.
var1=MCot(var1, r);//transforma los valores aplicando las reglas de
multicotomización
var3=Rcount(var1,k);//rachas de toda la matriz o sea, el arreglo anterior.
```

```
var1=matrix.BlockSet(1); Matrix rc[n][m];
for(i=0;ii<n;i++){
    for(j=0;j<m;j++){
        rc[i][j]=matrix.RachaQ(i,j); //arroja un número.
        rc[i][j]=matrix.RachaP(i,j); //arroja un número.
    }
}
```

```
DatShow(rc);
```

```
End;
```

3. Vocabulario Reconocido (Literal b). El analizador léxico identifica los siguientes elementos:

- Palabras reservadas:
 - si - para estructuras condicionales
 - entonces - parte de la estructura condicional
 - sino - alternativa condicional (else)
 - mientras - para bucles while
 - sumatoria - operación de suma acumulativa
 - multiplicatoria - operación de multiplicación acumulativa

raiz - operación matemática de raíz cuadrada

función para generar funciones

matrix: para general matrices

verdadero falso para valores booleanos

retornar para las funciones que retornan algo

var - Para declarar variables

friedman - para pruebas friedman

rachas - para ver patrones de rachas

mostar - para que aparezca en pantalla un mensaje

(ent) - para valores enteros

(dec) - para decimales

(car) - para caracteres

(pal) - para palabras

(Conc) - concatenación de números, vectores.

- Operadores: =, +, -, *, /.
- Delimitadores: (,), {, }, ;.
- Comentarios de línea: iniciados con //.
- Identificadores (nombres de variables y funciones).
- Literales numéricas (enteros y decimales).

4. Categorías léxicas (Literal c). Se definieron las siguientes categorías:

Palabras Reservadas (Keywords)

- si - para estructuras condicionales
- entonces - parte de la estructura condicional
- sino - alternativa condicional (else)
- mientras - para bucles while
- sumatoria - operación de suma acumulativa
- multiplicatoria - operación de multiplicación acumulativa
- raiz - operación matemática de raíz cuadrada
- función para generar funciones

- matrix: para general matrices
- verdadero falso para valores booleanos
- retornar para las funciones que retornan algo
- var - Para declarar variables
- friedman - para pruebas friedman
- rachas - para ver patrones de rachas
- mostrar - para que aparezca en pantalla un mensaje
- (ent) - para valores enteros
- (dec) - para decimales
- (car) - para caracteres
- (pal) - para palabras
- (Conc) - concatenación de números, vectores.

Identificadores

- Reglas:
 - Primer carácter: letra (a-z, A-Z) o guión bajo _
 - Caracteres subsiguientes: letras, dígitos (0-9) o _
 - Sensible a mayúsculas/minúsculas (case-sensitive)

Ejemplos válidos: contador, _temp, var1, MAX_VALUE

Literales

- Números enteros: 123, -45, 0
- Números decimales: 3.14, -0.001, 2.0
- Cadenas de texto: "hola", 'mundo'
- Booleanos: verdadero, falso
- notación científica: e+/-
- saltos de línea \n

Operadores

- Aritméticos:
 - + (suma)
 - - (resta)
 - * (multiplicación)
 - / (división)
 - ^ (potenciación - o XOR si es para operaciones lógicas)
 - % (módulo)
- Comparación:
 - == (igualdad)
 - != (desigualdad)
 - <, >, <=, >= (relacionales)

- Lógicos :
 - && (AND)
 - || (OR)
 - ! (NOT)

Delimitadores

- [] - para arrays/indexación
- () - para agrupación de expresiones/parámetros
- { } - para bloques de código
- , - separador de elementos
- ; - terminador de sentencias

Comentarios

- // comentarios de una línea

5. Patrones y expresiones regulares (Literal d) Los patrones definidos en **Lexer.flex** son:

Palabras Reservadas (Keywords)

Patrón:

(si|entonces|sino|mientras|para|hacer|var|función|matriz|secuencia|ent|dec|car|pal|bool|verdadero|falso|friedman_test|contar_rachas|estadistico_rachas|sumatoria|multiplicatoria|raiz|potencia|mostrar|leer|retornar|romper|continuar)

Extensión Regular:

PALABRA_RESERVADA = si | entonces | sino | mientras | para | hacer |

var | función | matriz | secuencia |

ent | dec | car | pal | bool |

verdadero | falso |

friedman_test | contar_rachas | estadistico_rachas |

sumatoria | multiplicatoria | raiz | potencia |

mostrar | leer | retornar | romper | continuar

Identificadores

Patrón: [a-zA-Z_][a-zA-Z0-9_]*

Extensión Regular:

LETRA = [a-zA-Z]

DÍGITO = [0-9]

IDENTIFICADOR = (LETRA | _)(LETRA | DÍGITO | _)*

Literales

- Números enteros: 123, -45, 0

Patrón: [+]?[0-9]+

Extensión Regular:

SIGNO = [+]

DÍGITO = [0-9]

ENTERO = SIGNO? DÍGITO+

- Números decimales: 3.14, -0.001, 2.0

Patrón: [+]?([0-9]*\.[0-9]+|[eE][+]?[0-9]+)?|[0-9]+[eE][+]?[0-9]+)

Extensión Regular:

PARTE_ENTERA = DÍGITO*

PARTE_DECIMAL = DÍGITO+

EXPONENTE = [eE] SIGNO? DÍGITO+

DECIMAL = SIGNO? ((PARTE_ENTERA . PARTE_DECIMAL EXPONENTE?) | (DÍGITO+ EXPONENTE))

Operadores

- Aritméticos:

Patrón: `[+\\-*/^%]`

- Comparación:

Patrón: `(=|!=|<=|>=|<|>)`

- Lógicos :

Patrón: `(&&|\\||!)`

5. Delimitadores

Patrón: `[(){}\\[\\],;]`

Extensión Regular:

DELIMITADOR = `(|) | { | } | [|] | , | ;`

Comentarios

Patrón: `//.*$`

Extensión Regular:

COMENTARIO_LÍNEA = `// [^\\n]*`

6. Analizador léxico con FLEX (Literal e). El analizador se generó con JFlex a partir de **Lexer.flex**. La estructura del proyecto es:

ANALIZADORLEXICO/

```
|— src/codigo/
|   |— Lexer.flex           // especificación FLEX
|   |— Lexer.java          // clase generada automáticamente mediante Principal.java
|   |— Tokens.java         // enumeración de tokens
|   |— FrmPrincipal.java   // interfaz gráfica de usuario
|   |— Principal.java      // clase para regenerar el lexer
```

Generación: Ejecutar `Principal.main()` cambiando la dirección por la de la carpeta local con el programa o usar `jflex Lexer.flex` desde la línea de comandos.

Integración: `FrmPrincipal` lee la entrada del usuario, escribe a `archivo.txt`, invoca `Lexer.yylex()` y muestra los tokens o errores en pantalla.