UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

ESCUELA DE SISTEMAS

ORGANIZACIÓN DE LENGUAJES COMPILADORES

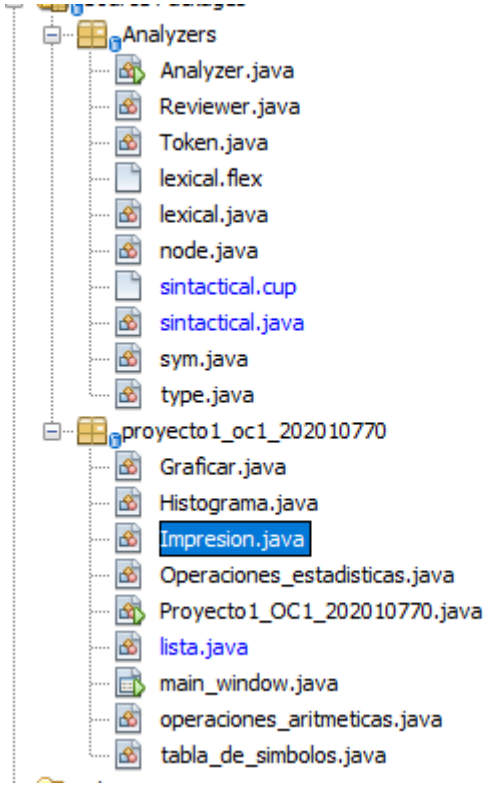
ING. MARIO BAUTISTA

PROYECTO 1

LUIS MARIANO MOREIRA GARCÍA

202010770

CLASES UTILIZADAS:



Analyzer: Se encarga de ejecutar los códigos de cup y de flex para poder hacer los archivos que ejecutaran nuestra gramática.

Reviewer: No se utilizo en el proyecto, su función era hacer un árbol sintactico.

Token: No se uso en el proyecto, se opto por utilizar una impresión simple para mostrar los tokens y errores.

Lexical.flex: Aquí es donde se encuentra nuestro archivo jflex, si se desea agregar o quitar palabras reservadas es en este archivo que se hace. Para ver las palabras reservadas consultar el manual de usuario.

Lexical.java, es uno de los dos archivos que genera Analyzer por parte del jflex.

Node: No se utiliza, es para crear los nodos del árbol reviewer.

Sintactical.cup, es aquí donde se hace la gramática y las instrucciones posteriores a ejecutar.

Sintactical.java es lo que genera Analyzer por parte de cup, no se debe

borrar por ninguna cuestión.

Sym y type, sin estas clases el código no funcionaria, nunca eliminar.

Graficar: Aquí es donde se encuentran las graficas.
Histograma: Ya no se utilizo, en su lugar para el histograma se encuentra en graficar.

Impresión: Sirve para imprimir.

Proyecto1_OC1_202010770: Main

Lista: Sirve para almacenar las listas, aquí recibe un string como variable y un array list de tabla de símbolos. Debido al poco tiempo restante es por esta misma razón que no se presenta en la misma tabla de símbolos.

Operaciones_estadisticas: Sirve para hacer todas las operaciones estadísticas (Todas devuelven string).

```
Media(<ARREGLO_DOUBLE>)
Mediana(<ARREGLO_DOUBLE>)
Moda(<ARREGLO_DOUBLE>)
Varianza(<ARREGLO_DOUBLE>)
Max(<ARREGLO_DOUBLE>)
Min(<ARREGLO_DOUBLE>)
```

Main_window: La ventana y sus funcionalidades.

Operaciones_aritméticas: Se usa para hacer todas las operaciones solicitadas en el proyecto, todas devuelven string:

```
var:double:: suma <- SUM(5, 2) end;
var:double:: resta <- RES(3, 2) end;
var:double:: multi <- MUL(4, numero) end;  ! Funciona con variables
var:double:: division <- DIV(1, variable) end;
var:double:: modulo <- MOD(5, 4) end;
```

Tablas de símbolos: Es donde se almacenan los símbolos y se realiza el reporte de estos:

```
public class tabla_de_simbolos {
    public String tipo;
    public String variable;
    public Object dato;


   public tabla_de_simbolos(String tipo, String variable, Object dato){
       this.tipo = tipo;
       this.variable = variable;
       this.dato = dato;

   }
```

Describiendo algunos algoritmos:

Como se puede apreciar en la gramática:

```
<set_graficas> ::= <set_graficas> <nueva_lista_graficas>
                 | <nueva_lista_graficas>;
```

Tenemos una recursividad por la izquierda, esto es de vital importancia ya que nos ayuda a crear instrucciones por así decirlo “infinitas” como por ejemplo la asignación en la lista que tiene la siguiente forma:

```
arr:<TIPO>::@<ID> <- <LISTA_VALORES> end;

! Ejemplos
arr:double::@darray <- [1, 2, 3, 4, 5] end; ! Arreglo de tipo double
arr:char[]::@carray <- [“12”, “2”, “3”] end; ! Arreglo de tipo string
arr:double::@carray <- [numero, copia, 7] end; ! Puede usar variables
```

Esto nos ayuda a formar fácilmente cuestiones que a priori parecieran no tener un final como>

[“hola”,”como estas”, “yo bien”, “gracias a la recursividad”]

Esta es formada por la siguiente gramática:

```
<impresion_de_la_lista> ::= ARROBA VARIABLE
                          | CORCHETE_ABIERTO <mas_listas> CORCHETE_CERRADO;

<mas_listas> ::= <mas_listas> COMMA dato
               | dato;
```

La forma en la que recupero los datos es bastante interesante, todo lo voy metiendo al siguiente string en caso de que no sea una variable:

```
public String datos_en_lista = "";
public String operacion_estadistica =
```

Luego este string:

```
double numero = 0.0;
Scanner scanner = new Scanner(datos_en_lista);
while (scanner.hasNext()) {
    String numeroStr = scanner.next();
    numero = Double.parseDouble(numeroStr);
    for (proyectol_ocl_202010770.lista elemento : lista_de_elementos) {
        if(elemento.variable.equals("Tremdo")){
            elemento.agregarElemento("double","Tremdo", numero);
    }
```

Temdo se utiliza para almacenar listas temporales, cuando iba avanzando en el proyecto ya no se utilizo mas, lamentablemente esta parte no fue actualizada y ya es muy riesgoso realizar un cambio a algo que ya funciona.

Como mencione mas adelante se hizo uso de otra variable y estas son char_temp y double_temp, los cuales facilitaban el uso, donde podemos verlas es en:

Variable: char_temp

Tipo	Variable	Dato

Variable: double_temp

Tipo	Variable	Dato
double	char_temp	2
double	char_temp	2
double	char_temp	2
double	char_temp	5
double	char_temp	5
double	char_temp	7
double	char_temp	8

Estas van a ir variando conforme se avanza en el programa.

```
public ArrayList<proyectol_ocl_202010770.tabla_de_simbolos> tabla_simbolos = new ArrayList<>();
public ArrayList<proyectol_ocl_202010770.lista> lista_de_elementos = new ArrayList<>();
public proyectol_ocl_202010770.operaciones_aritmeticas operaciones = new operaciones_aritmeticas();
public proyectol_ocl_202010770.Impresion impresion = new Impresion();
public proyectol_ocl_202010770.tabla_de_simbolos reporte_simbolos = new proyectol_ocl_202010770.tabla_de_simbolos("hola", "hola", "hola");
public proyectol_ocl_202010770.lista reporte_listas = new proyectol_ocl_202010770.lista("hola");
public String actual = "";
public String datos_en_lista = "";
public String operacion_estadistica = "";
public String operacion_grafica ="";
proyectol_ocl_202010770.Graficar graficals = new proyectol_ocl_202010770.Graficar();
public String titulo_grafica = "";
public String titulo_x_grafica = "";
public String titulo_y_grafica = "";
public String variable_en_x = "";
public String variable_en_y = "";
public String titulo_label = "";
public String valor_de_pies = "";
```

Variables importantes

1. Aquí se van almacenando las variables que no son listas.
2. Aquí se almacenan las variables de las listas
3. Se utiliza para realizar las operaciones aritméticas
4. Para las impresiones
5. Se utiliza para hacer los htmls de las variables (Tablas de simbolos).
6. Se utiliza para hacer el reporte de las listas
7. Se utiliza para ver dato actual.
8. Se utiliza para ver que operación estadística se va a utilizar
9. Se utiliza para ver que grafica se va a realizar
10. Se utiliza para graficar
11. Se utiliza para graficar
12. El resto se utiliza para formar las graficas.

¿Como se realizan las graficas?

```
<set_graficas> ::= <set_graficas> <nueva_lista_graficas>
                 | <nueva_lista_graficas>;
```

Se manda a llamar este set de graficas que a su vez contiene:

```
<nueva_lista_graficas> ::= <asignacion_titulo>
                         | <eje_x>
                         | <eje_y>
                         | <titulo_x>
                         | <titulo_y>
                         | <label_pie>
                         | <valores_pie>
```

Cada uno de los resultados de los valores hace que las variables que se comentan en el punto 12 se limpien y siempre agarren el ultimo valor asignado, se puede ver que se pueden poner infinitas cantidad de opciones, si bien esto es posible para todas las graficas los parámetros de estas solo buscan las variables necesarias y así evitamos que el usuario cometa un error.

Varianza:

```
public String varianza(ArrayList<tabla_de_simbolos> lista) {


    double media = Double.parseDouble(media(lista));

    double sumaCuadradosDiferencias = 0;
    for (tabla_de_simbolos elemento : lista) {

        String dato_string = elemento.dato.toString();
        double dato = Double.parseDouble(dato_string);

        sumaCuadradosDiferencias += Math.pow(dato - media, 2);
    }


    double varianza = sumaCuadradosDiferencias / lista.size();


    String resultado = Double.toString(varianza);

    return resultado;
}
```

El siguiente algoritmo calcula la varianza de una lista de datos representados por una lista de objetos tabla_de_simbolos. Primero calcula la media de los datos en la lista. Luego, itera sobre

cada elemento de la lista, calcula la diferencia entre el dato y la media, eleva al cuadrado esta diferencia y la agrega a una suma acumulativa. Después, divide esta suma por el número de elementos en la lista para obtener la varianza. Finalmente, devuelve la varianza como una cadena de texto.

Este algoritmo utiliza operaciones matemáticas básicas y operaciones de conversión de datos para calcular la varianza de una lista de datos numéricos.

Valor máximo

```
public String valor_maximo(ArrayList<tabla_de_simbolos> lista) {
    double maximo = 0;
    for (tabla_de_simbolos elemento : lista) {
        String dato_string = elemento.dato.toString();
        double dato = Double.parseDouble(dato_string);

        if (dato > maximo) {
            maximo = dato;
        }
    }
    String resultado = Double.toString(maximo);

    return resultado;
}
```

La función valor_maximo recibe una lista de objetos tabla_de_simbolos y devuelve el valor máximo encontrado en los datos almacenados en dicha lista.

Inicialmente, establece una variable maximo en 0. Luego, itera sobre cada elemento en la lista y convierte el dato asociado en cada objeto tabla_de_simbolos a un número de punto flotante. Si el dato es mayor que el valor actual de maximo, actualiza el valor de maximo con el valor del dato. Al finalizar la iteración, devuelve el valor máximo como una cadena de texto.

Este algoritmo busca el valor máximo en una lista de datos numéricos y devuelve ese valor como una cadena de texto. Para el mínimo es lo mismo solo que al revés.