Natalia Mariel Calderón Echeverría
202200007
Organización de Lenguajes y Compiladores 1
Primer semestre 2024

Manual de Tecnico

Requerimientos básicos:

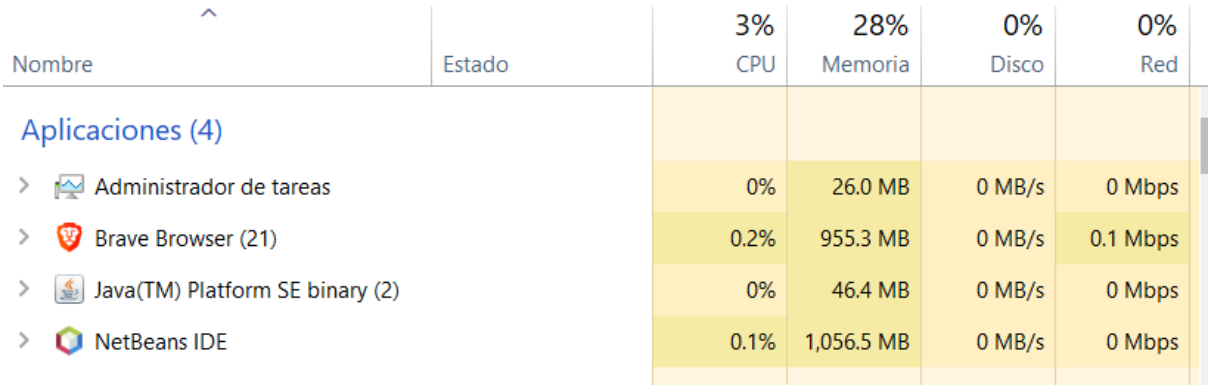
- **Lenguaje programado:** Java (JAVA JDK)
    - uso de html para reportes
- **IDE utilizada:** Netbeans 16
- **Librerías:**
    - **Java cup 11b runtime**
    - **Java cup 11b**

      Java cup es la librería encargada de la creación del analizador sintáctico en base a la gramática proporcionada.
    - **Jflex 1.9.1**

      Librería utilizada para la realización de análisis léxico, la obtención de los tokens.
    - **Jfreechart**

      Jfreechart es utilizada para la creación de los distintos gráficos propios del programa

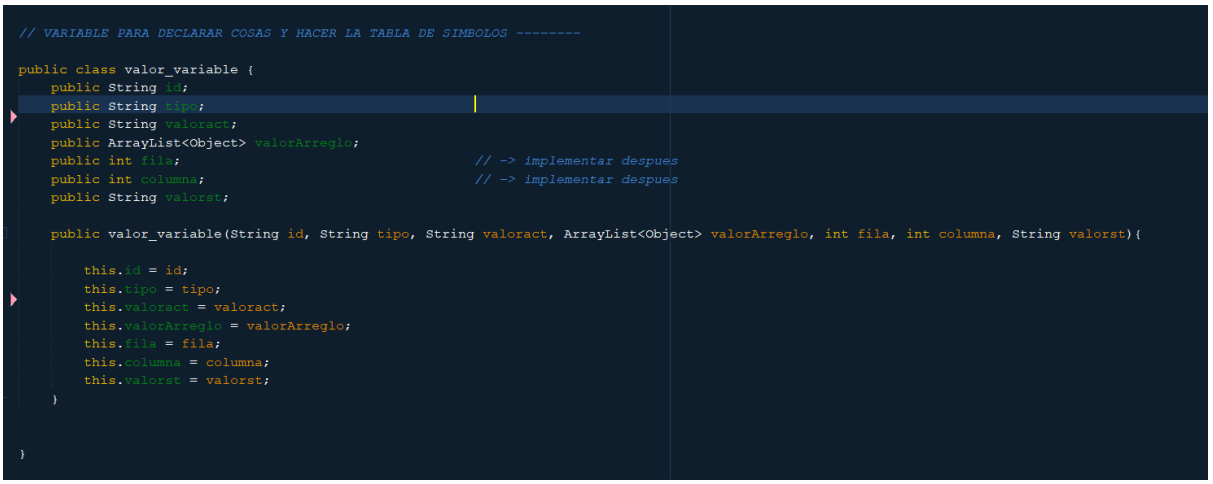
Requerimientos del equipo:



Nombre	Estado	3% CPU	28% Memoria	0% Disco	0% Red
Aplicaciones (4)					
Administrador de tareas		0%	26.0 MB	0 MB/s	0 Mbps
Brave Browser (21)		0.2%	955.3 MB	0 MB/s	0.1 Mbps
Java(TM) Platform SE binary (2)		0%	46.4 MB	0 MB/s	0 Mbps
NetBeans IDE		0.1%	1,056.5 MB	0 MB/s	0 Mbps

Funcionalidades importantes del programa

Uno de los pilares de este programa es la correcta clasificación y almacenamiento de las variables creadas, tanto como para su correcta generación como para su correcto uso en el futuro. Es por eso que una de las clases más importantes se encuentra en la clase “valor_variable”



```
// VARIABLE PARA DECLARAR COSAS Y HACER LA TABLA DE SIMBOLOS --------

public class valor_variable {
    public String id;
    public String tipo;
    public String valoract;
    public ArrayList<Object> valorArreglo;
    public int fila;                                    // -> implementar despues
    public int columna;                                 // -> implementar despues
    public String valorst;

    public valor_variable(String id, String tipo, String valoract, ArrayList<Object> valorArreglo, int fila, int columna, String valorst){

        this.id = id;
        this.tipo = tipo;
        this.valoract = valoract;
        this.valorArreglo = valorArreglo;
        this.fila = fila;
        this.columna = columna;
        this.valorst = valorst;
    }


}
```

Como se puede apreciar en el extracto de código, esta clase genera un objeto que contiene dentro de sí mismo varios atributos indispensables que permiten tanto clasificar a la variable como llevar un correcto funcionamiento de ella.

Esta clase permite no solo generar la tabla de símbolos como se menciona al inicio si no que permite que se generen correctamente los arreglos y que sea posible llamarlos nuevamente. Es por eso que esta clase es el pilar de 2 funcionalidades primordiales para el resto del funcionamiento del programa, como lo es:

- Declaración de variables
- Llamar a las variables
- Trabajar y manipular dichas variables

Estas funcionalidades se encuentran plasmadas en el archivo cup del programa, de igual forma se proveerá una explicación breve sobre estas 2 funciones y cómo interactúan entre ellas dentro del programa.

- Declaración de variables
  A través de la creación de un arraylist de objetos tipo “valor_variable” es posible generar cada una de las distintas variables y clasificarlas de acuerdo a las características propias de cada uno.

```
variables::= VAR DOSPUNTOS tipo:tp DOSPUNTOS DOSPUNTOS PALABRA_I:teid GUION expresion:val_var FIN PUNTOYCOMA
            {:
                    if (tp == "double") {
                        valor_variable nueva_var = new valor_variable(teid.toString(), "double", val_var.toString(), null,  0,  0, val_var.to
                        tabla_simbolos.add(nueva_var);
                        lista.clear();
                        /*
                        for (valor_variable variable : tabla_simbolos) {
                            System.out.println("ID: " + variable.id);
                            System.out.println("valor: " + variable.valoract);
                            System.out.println("tipo: " + variable.tipo);
                        }*/

                        System.out.println("----------------------------------------");
                        nombre_arList = "nada";
                    } else if (tp== "String") {
                        valor_variable nueva_var = new valor_variable(teid.toString(), "cadena", val_var.toString(), null, 0, 0, val_var.toSt
                        tabla_simbolos.add(nueva_var);
                        lista.clear();

                        /*
                        for (valor_variable variable : tabla_simbolos) {
                            System.out.println("ID: " + variable.id);
                            System.out.println("valor: " + variable.valoract);
                            System.out.println("tipo: " + variable.tipo);
                        }*/

                        System.out.println("----------------------------------------");
                        nombre_arList = "nada";
                    } else {
                                System.out.println("ELEGIR UN VALOR VALIDO PARA CREAR LA VARIABLE");
                                nombre_arList = "nada";
```

En el caso de los arreglos, la declaración y almacenamiento dentro de la tabla de símbolos es similar, con la diferencia de que se crea un elemento de la clase hashmap también en base a la declaración previa, esto con el objetivo de poder acceder a este mas facil y rapido, al momento de trabajar con arreglos en las operaciones estadísticas o en la generación de gráficas.

```
340             | ARREGLO DOSPUNTOS tipo:tp DOSPUNTOS DOSPUNTOS ARROBA PALABRA_I:teid GUION ABRIRCORCHETES lista_arreglo:at CERRARCORCHETES FIN PUNTOYCOMA
341                 {:
342                         //System.out.println("DECLARAR ARREGLOS");
343                         if (tp == "double") {
344                             ArrayList<Object> arrayVarToInsert = new ArrayList<>(arrayVar);
345                             valor_variable nuevo_arr = new valor_variable(teid.toString(), "double", "arreglo", arrayVarToInsert,  0,  0, "arreglo");
346                             tabla_simbolos.add(nuevo_arr);
347                             hashMap.setArrayListById(teid.toString(), arrayVarToInsert);
348
349                             lista.clear();
350                              /*
351                             System.out.println("ID: " + nuevo_arr.id);
352                             System.out.println("tipo: " + nuevo_arr.tipo);
353                             System.out.println("VALORES" );
354                             for (Object value : nuevo_arr.valorArreglo) {
355                                 System.out.println(value);
356                             }
357
358
359                             System.out.println("----------------------------------------");
360                             ArrayList<Object> retrievedArrayVar1 = hashMap.getArrayListById(teid.toString());
361                             System.out.println("-------" + teid.toString()+"-------------");
362                             System.out.println(retrievedArrayVar1);
363                             */
364                             arrayVar.clear();
365                             nombre_arList = "nada";
366                         } else if (tp== "String") {
367                             ArrayList<Object> arrayVarToInsert = new ArrayList<>(arrayVar);
368                             valor_variable nuevo_arr = new valor_variable(teid.toString(), "cadena", "arreglo", arrayVarToInsert, 0, 0, "arreglo");
369                             tabla_simbolos.add(nuevo_arr);
370                             hashMap.setArrayListById(teid.toString(), arrayVarToInsert);
```

```
369                             tabla_simbolos.add(nuevo_arr);
370                             hashMap.setArrayListById(teid.toString(), arrayVarToInsert);
371                             lista.clear();
372                             nombre_arList = "nada";
373                             /*
374                             System.out.println("ID: " + nuevo_arr.id);
375                             System.out.println("tipo: " + nuevo_arr.tipo);
376                             System.out.println("VALORES" );
377                             for (Object value : nuevo_arr.valorArreglo) {
378                                 System.out.println(value);
379                             }*/
380
381                             arrayVar.clear();
382
383                             /*
384                             System.out.println("----------------------------------------");
385                             System.out.println("----------------------------------------");
386                             ArrayList<Object> retrievedArrayVar1 = hashMap.getArrayListById(teid.toString());
387                             System.out.println("-------" + teid.toString()+"-------------");
388                             System.out.println(retrievedArrayVar1);*/
389
390                         } else {
391                                     System.out.println("ELEGIR UN VALOR VALIDO PARA CREAR LA VARIABLE");
392                         }
393                 :}
394     ;
395
```

- Llamar a variables

  Esta llamada a variables se realiza gracias a la distinción entre el token “cadena” y el token palabra_id, en caso de tratarse de una palabra_id, el programa procede a buscar dentro de la tabla de símbolos algun id que coincide con la palabra que se presenta.

```
titulo_str ::= CADENA_F:c
{:
                String titulo_strg= c;
                System.out.println("titulo histograma: " + c );
                RESULT =titulo_strg;
:}
            | PALABRA_I:dest
{:
                boolean variableExists = false;
                for (valor_variable variable : tabla_simbolos) {
                    if (variable.id.equalsIgnoreCase(dest)) {
                        variableExists = true;
                        //System.out.println("variable encontrada");
                        if (variable.tipo.equals("cadena")) {
                            //System.out.println("es una cadena");
                            dest= variable.valoract;
                            System.out.println("titulo histograma: " + dest );
                            String titulo_strg= dest;
                            RESULT =titulo_strg;
                        } else if (variable.tipo.equals("double")) {
                            System.out.println("LA VARIABLE " + dest + "NO ES DE TIPO CHAR -INGRESAR UNA VALIDA.");
                            acumulador.addToOutput("LA VARIABLE " + dest + "NO ES DE TIPO CHAR -INGRESAR UNA VALIDA.");
                        }
                        //RESULT = dest;
                        break; // No need to continue searching once found
                    }
                }
                if (!variableExists) {
                    System.out.println("Variable " + dest + " no existe en el programa.");
                    acumulador.addToOutput("Variable " + dest + " no existe en el programa.");
                }
:};
```

Esto es en el caso de hablar de variables simples, es decir, no son arreglos. Si nos referimos a arreglos, es el momento en donde se hace uso del hashmap generado anteriormente, en donde se obtiene los valores del arreglo a través de la clave, que en este caso corresponde al id.

```
  | EJEX DOSPUNTOS DOSPUNTOS CADENACHAR ABRIRCORCHETES CERRARCORCHETES IGUAL ref_grafoarr:tr FIN PUNTOYCOMA
 {:
            if (!nombre_arList.equals("nada")) {
                eje_x.clear();
                ArrayList<Object> array_line_str = hashMap.getArrayListById(nombre_arList.toString());
                eje_x.addAll(array_line_str);
                variable_g.clear();
                nombre_arList = "nada";
```

- Manipular dichas variables

Las variables declaradas son manipuladas al momento de realizar operaciones aritméticas o estadísticas con ellas. Es aca en donde el tipado previo dentro de la clase valor_variable se vuelve muy funcional, ya que permite únicamente trabajar con doubles, evitando asi cualquier error al tratar de usar una variable tipo String para hacer una suma, o utilizar un

valor para sacar un varianza. El tipado previo, permite seleccionar una variable que sí sea la correcta.

- Operaciones aritméticas
  Una vez llamada la variable esta es devuelta como resultado de esta producción, por lo que es posible operarlas sin problemas de manera recursiva.

```
228             |  SUMA ABRIRPARENTESIS aritmetica:izq COMA aritmetica:der CERRARPARENTESIS
229                {:
230                             double result_s = (Double) izq + (Double) der;
231                             RESULT = result_s;
232                 :}
233             |  RESTA ABRIRPARENTESIS aritmetica:izq COMA aritmetica:der CERRARPARENTESIS
234                {:
235                             double result_s = (Double) izq - (Double) der;
236                             RESULT = result_s;
237                 :}
238             |  MULTIPLICACION ABRIRPARENTESIS aritmetica:izq COMA aritmetica:der CERRARPARENTESIS
239                {:
240                             double result_s = (Double) izq * (Double) der;
241                             RESULT = result_s;
242                 :}
243             |  DIVISION ABRIRPARENTESIS aritmetica:izq COMA aritmetica:der CERRARPARENTESIS
244                {:
245                             double result_s = (Double) izq / (Double) der;
246                             RESULT = result_s;
247                 :}
248             |  MODULO ABRIRPARENTESIS aritmetica:izq COMA aritmetica:der CERRARPARENTESIS
249                {:
250                             double result_s = (Double) izq % (Double) der;
251                             RESULT = result_s;
252                 :}
```

- Operaciones Estadísticas
  Del mismo modo que se recupera la información de los arreglos, haciendo uso de los hashmaps, se recupera la informacion para dichos arreglos.

```
269             | MODA ABRIRPARENTESIS estadistica_arreglo:hashnom CERRARPARENTESIS
270                {:
271                             ArrayList<Object> retrievedArrayS =hashVarianza.getArrayListById(hashnom.toString());
272                             hashVarianza.calculateMedian(retrievedArrayS);
273                             double total = hashVarianza.calculateMode(retrievedArrayS);
274                             //System.out.println(total);
275                             RESULT = total;
276                 :}
277             |  VARIANZA ABRIRPARENTESIS estadistica_arreglo:hashnom CERRARPARENTESIS
278                {:
279                             ArrayList<Object> retrievedArrayS =hashVarianza.getArrayListById(hashnom.toString());
280                             hashVarianza.calculateMedian(retrievedArrayS);
281                             double total = hashVarianza.calculateVariance(retrievedArrayS);
282                             //System.out.println(total);
283                             RESULT = total;
284                 :}
285             |  MAX ABRIRPARENTESIS estadistica_arreglo:hashnom CERRARPARENTESIS
286                {:
287                             ArrayList<Object> retrievedArrayS =hashVarianza.getArrayListById(hashnom.toString());
288                             hashVarianza.calculateMedian(retrievedArrayS);
289                             double total = hashVarianza.calculateMax(retrievedArrayS);
290                             //System.out.println(total);
291                             RESULT = total;
292                 :}
293             |  MIN ABRIRPARENTESIS estadistica_arreglo:hashnom CERRARPARENTESIS
294                {:
295                             ArrayList<Object> retrievedArrayS =hashVarianza.getArrayListById(hashnom.toString());
296                             hashVarianza.calculateMedian(retrievedArrayS);
297                             double total = hashVarianza.calculateMin(retrievedArrayS);
298                             //System.out.println(total);
299                             RESULT = total;
300                 :}
301
302     ;
```

Para un mejor cálculo y una gramática más ordenada, las funciones propias para obtener los resultados a las operaciones estadísticas se encuentran dentro de la clase IdArrayListHashMap, y para utilizar estas funciones únicamente las llamamos y mandamos como parámetro el arreglo recuperado dentro del mismo hashmap.

Por ejemplo, para calcular el valor máximo y el valor mínimo únicamente se recorre dicho arreglo y se va comparando el valor de referencia. A través de esta simple funcionalidad es posible calcular el valor máximo y el valor mínimo, de la misma forma se realizan los demás cálculos estadísticos, es decir únicamente se aplica el concepto de lo que es dicho cálculo y a través de loop y operaciones aritméticas es posible llegar al resultado deseado.

```java
// MAX
public double calculateMax(ArrayList<Object> arrayList) {
    double max = toDouble( obj: arrayList.get( index: 0));
    for (Object element : arrayList) {
        if (toDouble( obj: element) > max) {
            max = toDouble( obj: element);
        }
    }
    return max;
}

// MIN
public double calculateMin(ArrayList<Object> arrayList) {
    double min = toDouble( obj: arrayList.get( index: 0));
    for (Object element : arrayList) {
        if (toDouble( obj: element) < min) {
            min = toDouble( obj: element);
        }
    }
    return min;
}
```