

Manual técnico

En este proyecto se creo un interprete para un lenguaje parecido a c, el cual cuenta con las siguientes funciones y características:

Declaración de variables

Acá se pueden declarar variables las cuales serán guardadas en un entorno de la función que se este manejando o si es en un ámbito global se guardaran en su dicha tabla de símbolos, la forma para declarar una variable es la siguiente.

```
declaracionv: tipo
| tipo ids IGUAL
| ids IGUAL
```

El cual puede venir con un su respectivo tipo su id o lista de ids, ya que se pueden declarar múltiples variables en una misma línea, y opcionalmente su valor que contendrán si no se le asigna valor se le colocarán unos valores por defecto

```
1  export default class Declaracion extends Instruccion {
2    private id: string[];
3    private valor: Instruccion | undefined;
4
5    constructor(tipo: Tipo, linea: number, columna: number, id: string[], valor?: Instruccion) {
6      super(tipo, linea, columna);
7      this.id = id;
8      this.valor = valor;
9    }
10
11    interpretar(ArbolS: ArbolS, tabla: TablaSimbolos) {
12      if (this.valor) {
13        let valorFinal = this.valor.interpretar(ArbolS, tabla);
14        if (valorFinal instanceof Errores) return valorFinal;
15
16        if (this.valor.Tipo.getTipo() !== this.Tipo.getTipo()) {
17          ArbolS.createAndAddError(ArbolS, 'Semantico', 'El tipo de dato no es igual', this.Linea, this.Columna);
18          return new Errores('Semantico', 'El tipo de dato no es igual', this.Linea, this.Columna);
19        }
20
21        for (let ide of this.id) {
22          if (!tabla.setVariable(new Simbolo(this.Tipo, ide, valorFinal))) {
23            ArbolS.createAndAddError(ArbolS, 'Semantico', 'La variable ${ide} ya existe', this.Linea, this.Columna);
24            return new Errores('Semantico', 'La variable ${ide} ya existe', this.Linea, this.Columna);
25          }
26        }
27      }
28      else {
29        for (let ide of this.id) {
30          if (!tabla.setVariable(new Simbolo(this.Tipo, ide, this.valorDefecto(this.Tipo.getTipo())))) {
31            ArbolS.createAndAddError(ArbolS, 'Semantico', 'La variable ${ide} ya existe', this.Linea, this.Columna);
32            return new Errores('Semantico', 'La variable ${ide} ya existe', this.Linea, this.Columna);
33          }
34        }
35      }
36    }
37  }
```

Como podemos observar en esta parte del código se hacen múltiples verificaciones para comprobar la lógica dentro del código que se esta ingresando, tales como verificar que el tipo

de dato y el valor que se le esta asignando son del mismo tipo o de que no se asignen variables repetidas dentro de un mismo ámbito

The screenshot shows the CompiScript IDE with a C++ program and its execution output. The code defines a 'Declaracion' method that prints variable declarations and their values, and an 'Ambitos' method that manages variable scopes. The console output shows the program's execution, including variable declarations, scope management, and arithmetic operations.

```

1 int num1 = 1;
2 int punteo = 0;
3
4 void Declaracion() {
5     /* SALIDA ESPERADA:
6     ***** Metodo Declaracion *****
7     Voy a ganar Compiladores 1 :0
8     *****
9     */
10    cout << ("***** Metodo Declaracion *****") << endl;
11    int num1, num2, num3, num4 = 1;
12    Std::String str1 = "Voy a ganar Compiladores";
13    Std::String str2 = "Voy a ganar Compiladores";
14    Std::String str3 = "Voy a ganar Compiladores";
15    Std::String str4 = "Voy a ganar Compiladores";
16    Double db1 = 0.0;
17    Double db2 = 0.0;
18    Double db3 = 0.0;
19    Double db4 = 0.0;
20    char chr1 = 's';
21    char chr2 = 's';
22    char chr3 = 's';
23    char chr4 = 's';
24    //Asi se modifica la asignaci?n
25    if (db1 == db4) {
26        cout << (str1 + chr2 + " " + num3 + " :0") << endl;
27        punteo = punteo + 4;
28        cout << ("Declaraci?n correcta") << endl;
29        cout << ("Haz sumado 4 puntos") << endl;
30    } else {
31        cout << ("Problemas en el metodo declaracion :(") << endl;
32        cout << ("Perdiste 6 pts :(") << endl;
33    }
34    cout << ("*****") << endl;
35    cout << ("Punteo = " + punteo) << endl;
36    cout << ("*****") << endl;
37 }
38
39 void Ambitos() {
40     //Ambito Local
41     std::string ambi1 = "Desde ambito2";
42     cout << ("*****Ambitos 2*****") << endl;
43     if (ambi1 == "Desde ambito2") {
44         cout << (ambi1) << endl;
45         punteo = punteo + 8;
46     }
47 }
  
```

```

1 -----CALIFICACION ARCHIVO 1-----
2 Valor: 35 pts
3 -----
4 Muy bien, prioridad de variable local correcta
5 Haz sumado 8 puntos
6 Punteo = 8
7 ***** Metodo Declaracion *****
8 Voy a ganar Compiladores 1 :0
9 Declaraci?n correcta
10 Haz sumado 4 puntos
11 -----
12 Punteo = 14
13 *****
14 *****Ambitos 2*****
15 Desde ambito2
16 Punteo = 22
17 *****Aritmeticas*****
18
19 Hola COMPI
20 El valor de num1 = 52.1
21 El valor de n3 = 70
22 Operaciones Aritmeticas 1: valor esperado: a:42 b:0
23 a: 42
24 b: 0
25 c: -19
26 d: 256
27 Operaciones aritmeticas 1 bien :0
28 Operaciones Aritmeticas 2: valor esperado:-20 41
29 resultado:
30 -20
31 41
32 Operaciones aritmeticas 2 bien :0
33 Punteo = 41
34 *****Logicas1*****
35 Bien primera condici?n:
36 Bien segunda condici?n:
37 *****Logicas2*****
38 Not y And Correctos
39 Nota y Ors correctos
40 *****Logicas3*****
41 NAMUS Correctos
42 WORS correctos
43 *****
44
  
```

El proyecto tambi?n cuenta con un IDE donde se pueden manejar m?ltiples pesta?as y un control sobre los errores, lo que se imprima en consola, la tabla de s?mbolos, cargar archivos, y el reporte del ?rbol de sintaxis generado por la entrada a evaluar, este ?rbol se realizo de la siguiente manera

Se declaro una clase contador, del que utilizaremos el contador para los nodos con una misma instancia para as? controlar cual seria el nodo anterior al que se le tiene que conectar

The screenshot shows a C++ code snippet defining a static class 'Contador'. The class has a static instance variable, a private constructor, and public static methods 'getInstance()' and 'get()' to manage a global counter.

```

1 export default class Contador {
2     private static instance: Contador;
3     private contador: number;
4     private constructor() {
5         this.contador = 0;
6     }
7     public static getInstance(): Contador {
8         if (!Contador.instance) {
9             Contador.instance = new Contador();
10        }
11        return Contador.instance;
12    }
13
14    get(){
15        this.contador++;
16        return this.contador;
17    }
18 }
19 //Clase para tener un contador global, una unica instancia de la clase, la cual servira
20 //para tener un orden en los nodos y los links que se creen en el grafo
  
```

Luego se manda a llamar a la instancia de la clase dentro del método de cada instrucción llamado BuildAst

```
1 buildAst(anterior: string): string {
2     let contador = Contador.getInstance();
3     let nodoCaso = `n${contador.get()}`;
4     let nodoCase = `n${contador.get()}`;
5     let nodoExpresion = `n${contador.get()}`;
6     let nodoDosPuntos = `n${contador.get()}`;
7     let nodoCodigos = `n${contador.get()}`;
8     let resultado = `${nodoCaso}[label="Caso"]\n`;
9     resultado += `${nodoCase}[label="CASE"]\n`;
10    resultado += `${nodoCaso} → ${nodoCase}\n`;
11    resultado += `${nodoExpresion}[label="Expresion"]\n`;
12    resultado += `this.expresion.buildAst(nodoExpresion);
13    resultado += `${nodoCaso} → ${nodoExpresion}\n`;
14    resultado += `${nodoDosPuntos}[label=""]\n`;
15    resultado += `${nodoCaso} → ${nodoDosPuntos}\n`;
16    resultado += `${nodoCodigos}[label="Instrucciones"]\n`;
17    for (let instruccion of this.instrucciones) {
18        resultado += instruccion.buildAst(nodoCodigos);
19    }
20    resultado += `${nodoCaso} → ${nodoCodigos}\n`;
21    resultado += `${anterior} → ${nodoCaso}\n`;
22    return resultado;
23 }
24 }
```

Acá se construye el árbol sintáctico de manera que se crea un nodo por cada instrucción ingresada de la que se conectan los nodos de los terminales y no terminales que componen dicha expresión

Y de esta manera se obtiene un archivo .dot parecido al siguiente

```
1 digraph G {
2     n0[label="Inicio"]
3     nCodigo[label="Codigo"]
4     n0 → nCodigo
5     n25[label="Instruccion"]
6     nCodigo → n25
7     n26[label="Declaracion"]
8     n27[label="0"]
9     n26 → n27
10    n28[label="Ids"]
11    n26 → n28
12    n29[label="addafsdffasdf"]
13    n28 → n29
14    n25 → n26
15
16 }
```

En el cual se observa como correctamente crea las conexiones según el contador de la clase creada.

```

1 import TablaSimbolos from '../SimboloC/TablaSimbolos';
2 import Tipo from '../SimboloC/Tipo';
3 import Arboles from '../SimboloC/Arboles';
4 //Esta clase nos servira para crear el arbol con los nodos de instruccion que contendran
5 //a su vez otros nodos de instruccion o nodos de expresion para leer e interpretar ese arbol
6 export abstract class Instruccion {
7     public Tipo: Tipo;
8     public Linea: number;
9     public Columna: number;
10
11     constructor(tipo: Tipo, linea: number, columna: number){
12         this.Tipo = tipo;
13         this.Linea = linea;
14         this.Columna = columna;
15     }
16
17     abstract interpretar(Arboles: Arboles, tabla: TablaSimbolos): any;
18
19     abstract buildAst(anterior: string): string;
20
21 }
22

```

Para cada funcionalidad de este proyecto se utilizó esta clase abstracta, en el que se aplico lo siguiente que es el patrón interprete

El patrón intérprete es un patrón de diseño de software que se utiliza para definir una gramática para un lenguaje y proporcionar un intérprete que interpreta las sentencias de ese lenguaje. Consiste en dividir un problema en una representación recursiva de la gramática del lenguaje y luego implementar un intérprete que pueda procesar estas sentencias según la gramática definida.

Esto es lo que la clase Instrucción se encarga de hacer, ya que tiene su constructor, y los métodos que interpretarán y harán las funciones del programa.

```

1 ;
2 expression : Casteos
3             | NUMERO
4             | DECIMAL
5             | CADENA
6             | TRUE
7             | FALSE
8             | ID
9             | PARENTESISISI expression PARENTESISID
10            | operacion
11            | CARACTER
12            | ternaryOp
13            | operacionRelacional
14            | accesoVector
15            | funcToLower
16            | funcToUpper
17            | funcionRound
18            | funcionLength
19            | funcionTypeOf
20            | funcionToString
21            | Llamada
22
23
24 ;

```

Para mi gramática se a utilizado esta producción de expresión, la cual cuenta con los distintos tipos de datos o valores que pueden ser utilizados para, asignar datos a una variable, guardar en un arreglo, etc, ya que en si este cubre las variantes de lo que puede ser ingresado por el usuario, un ejemplo seria el siguiente

```
1 void metodo1(){
2     //llamada del metodo
3     figura1(10);
4 }
5 void figura1(int n){
6     std::string cadenaFigura = "";
7     double i;
8     i=-3*n/2;
9     //iniciando dibujo
10    while(i<=n){
11        cadenaFigura = "";
12        double j;
13        j=-3*n/2;
14        while(j<=3*n){
15            double absolutoi;
16            absolutoi = i;
17            double absolutoj;
18            absolutoj = j;
19            if(i < 0)
20            {
21                absolutoi = i * -1;
22            }
23            if(j < 0)
24            {
25                absolutoj = j * -1;
26            }
27            if((absolutoi + absolutoj < n)
28                || ((-n / 2 - i) * (-n / 2 - i) + (n / 2 - j) * (n / 2 - j) <= n * n / 2)
29                || ((-n / 2 - i) * (-n / 2 - i) + (-n / 2 - j) * (-n / 2 - j) <= n * n / 2)) {
30                cadenaFigura = cadenaFigura + "* ";
31            }
32            else
33            {
34                cadenaFigura = cadenaFigura + ". ";
35            }
36            j=j+1;
37        }
38        cout << cadenaFigura << endl;
39        i=i+1;
40    }
41    cout << "Si la figura es un corazón, tas bien :3"<< endl;
42 }
43
44 execute metodo1();
```

En este método podemos observar la expresión que es lo que este contenido dentro del if, dentro de las impresiones etc.