

Universidad San Carlos de Guatemala
Organización de Lenguajes y Compiladores 2

Manual Técnico de Proyecto 2

GoLight

Sergio Sebastian Sandoval Ruiz
202010298
21 de marzo del 2025

1. Descripción General

Este proyecto consiste en el desarrollo de un entorno de desarrollo integrado (IDE) para el lenguaje **GoLight**, utilizando tecnologías como **ANTLR**, **C#** y ejecución nativa en **Linux**. El sistema incluye un editor de texto personalizado, un intérprete del lenguaje y herramientas para análisis, visualización y reporte.

2. Tecnologías Utilizadas

Tecnología	Descripción
ANTLR	Generación de analizadores léxicos y sintácticos.
C#	Desarrollo del intérprete, análisis semántico, generación de reportes y consola.
Linux	Plataforma objetivo de ejecución.
Avalonia / ASP.NET	Frameworks propuestos para la interfaz de usuario. A elección del desarrollador.

Arquitectura del Sistema

3.1 Componentes

- **Editor de Texto (IDE):** Permite escribir y editar archivos **.glt**.
- **Intérprete en C#:** Procesa el código utilizando ANTLR y genera resultados.
-

- **Módulo de Reportes:** AST, tabla de símbolos y errores.
- **Consola de Salida:** Presenta mensajes, resultados y advertencias.

Estructura del Código

5.1 ANTLR

- Archivo de gramática: `GoLight.g4`

Comandos para generación:

`bash`

`antlr4 -Dlanguage=CSharp GoLight.g4 -o Generated`

-

5.2 Intérprete

- Implementado en C# usando `Visitor` o `Listener`.
- Divide el proceso en:
 - `Lexer`
 - `Parser`
 - `ASTVisitor` o `Interpreter`
 - `SemanticAnalyzer`

5.3 API (si se usa ASP.NET)

- Endpoint principal: `/interpretar`

- Entradas: contenido del archivo `.glt`
 - Salidas: JSON con reportes y resultados.
-

6. Requisitos de Ejecución

6.1 Requisitos del Sistema

- Sistema Operativo: Linux (Ubuntu 20.04+ recomendado)
- .NET SDK 7.0+
- Java (para generación con ANTLR)
- ANTLR 4.13+

6.2 Instalación

bash

```
sudo apt update
```

```
sudo apt install default-jre
```

```
dotnet new install Avalonia.Templates # Si se usa Avalonia
```

6.3 Compilación

bash

```
dotnet build
```

```
dotnet run
```

Funciones y Procedimientos

Gramatica definida

```
declaracion_funciones: 'func' ID '(' parametros? ')' '{' declaraciones* '}'  
;  
  
parametros: ID tipo ( ',' ID tipo)*;  
  
stmt: expr ';' ? # Expression  
      | 'fmt.Println(' expr (',' expr)* ')' ';' ? # FmtPrint  
      | '{' declaraciones* '}' # Bloque  
      | 'if' expr stmt ('else' stmt)? # If  
      | 'for' expr stmt # ForCond  
      | 'for' inicializacionesfor ';' expr ';' expr stmt # For  
      | 'break' ';' ? # Break  
      | 'continue' ';' ? # Continue  
      | 'return' expr? ';' ? # Return  
;  
  
inicializacionesfor : declaracion_variable | expr ;  
  
tipo: 'int' # TipoInt  
      | 'float64' # TipoFloat  
      | 'string' # TipoString  
      | 'bool' # TipoBool  
      | 'rune' # TipoRune  
;  
  
call: '(' atri? ')' # FunCall // | '.' ID #Get  
;  
  
atri: expr (',' expr)*;  
  
expr:  
      | '-' expr # Negacion  
      | '!' expr # Not  
      | expr call+ # CallExpr  
      | expr '[' expr ']' ('[' expr ']')? # GetSlices  
      | ID '++' # Incre  
      | ID '--' # Decre  
      | expr op = ('*' | '/' | '%') expr # MulDivMod  
      | expr op = ('+' | '-') expr # SumRes  
      | expr op = ('>' | '<' | '>=' | '<=') expr # Relacionales  
      | expr op = ('==' | '!=') expr # Igualdad  
      | 'slices.Index(' expr ',' expr')' # SliceIndex  
      | expr '&&' expr # And  
      | expr '||' expr # Or  
      | expr '=' expr # Asignacion  
      | BOOL ';' ? # Boolean  
      | FLOAT ';' ? # Float  
      | STRING ';' ? # String  
      | RUNE ';' ? # Rune  
      | INT ';' ? # Entero  
      | NIL ';' ? # Nulo  
      | 'new' ID '(' atri? ')' # Instancia  
      | ID # Id  
      | '(' expr ')' # Parentesis  
      ;
```

Manejo de Entorno

```
public class Environment
{
    6 references
    public Dictionary<string, ValueWrapper> variables = new Dictionary<string, ValueWrapper>();
    5 references
    private Environment? parent;

    5 references
    public Environment(Environment? parent)
    {
        this.parent = parent;
    }

    10 references
    public ValueWrapper GetVariable(string id, Antlr4.Runtime.IToken token)
    {
        if (variables.ContainsKey(id))
        {
            return variables[id];
        }

        if (parent != null)
        {
            return parent.GetVariable(id, token);
        }

        throw new ErrorSemantico("Variable " + id + " not found", token);
    }

    21 references
    public void DeclaracionVariable(string id, ValueWrapper value, Antlr4.Runtime.IToken? token)
    {
        if (variables.ContainsKey(id))
        {
            if (token != null) throw new ErrorSemantico("Variable " + id + " already declared", token);
        }
        else
        {
            variables[id] = value;
        }
    }

    6 references
    public ValueWrapper AsignacionVariable(string id, ValueWrapper value, Antlr4.Runtime.IToken? token)
    {
        if (variables.ContainsKey(id))
        {
            variables[id] = value;
            return value;
        }

        if (parent != null)
        {
            return parent.AsignacionVariable(id, value, token);
        }

        throw new ErrorSemantico("Variable " + id + " not found", token);
    }
}
```

Envuelto de variables para manejo personalizado

```
99+ references
public abstract record ValueWrapper;

81 references
public record EnteroValue(int Value) : ValueWrapper;
71 references
public record FloatValue(float Value) : ValueWrapper;
25 references
public record StringValue(string Value) : ValueWrapper;
65 references
public record BooleanValue(bool Value) : ValueWrapper;
25 references
public record RuneValue(uint Value) : ValueWrapper;

15 references
public record SliceValue(List<ValueWrapper> Elements) : ValueWrapper;
6 references
public record Slice2Value(List<List<ValueWrapper>> Elements) : ValueWrapper;

2 references
public record InstanceValue(Instance instance) : ValueWrapper;
4 references
public record VoidValue : ValueWrapper;
0 references
public record ObjectValue(Environment env) : ValueWrapper;
8 references
public record FunctionValue(Invocable invocable, string name) : ValueWrapper;
4 references
public record StructValue(LanguageStruct LanguageStruct) : ValueWrapper;
```

Definiciones de CompilerVisitor para el manejo de cada definición, declaración, statement, etc de la gràmatica

```
2 using System.Globalization;
3 using System.Text;
4 using analyzer;
5 using Microsoft.VisualBasic;
6
7 6 references
8 public class CompilerVisitor : LanguageBaseVisitor<ValueWrapper>
9 {
10     4 references
11     public string output = "";
12     5 references
13     public Environment currentEnvironment;
14     18 references
15     public ValueWrapper defaultVoid = new VoidValue();
16
17     1 reference
18     public CompilerVisitor()
19     {
20         currentEnvironment = new Environment(null);
21         Embedded.Generate(currentEnvironment);
22     }
23
24     3 references
25     public override ValueWrapper VisitProgram(LanguageParser.ProgramContext context)
26     {
27         foreach (var decl in context.declaraciones())
28         {
29             Visit(decl);
30         }
31         return defaultVoid;
32     }
33
34     3 references
35     public override ValueWrapper VisitDeclaracionVariableConValor(LanguageParser.DeclaracionVariableConValorContext context)
36     {
37         /*string texto = null;
38         Console.WriteLine(texto + " -> null");*/
39         string id = context.ID().GetText();
40         string tipo = context.tipo().GetText();
41         ValueWrapper value = Visit(context.expr());
42
43         if (!TipoCoincide(tipo, value))
44         {
45             throw new Exception($"Error: No se puede asignar un valor de tipo {value.GetType()} a la variable '{id}' de tipo {tipo}.");
46         }
47
48         currentEnvironment.DeclaracionVariable(id, value, context.Start);
49         return defaultVoid;
50     }
51
52     3 references
53     public override ValueWrapper VisitDeclaracionVariableSinValor(LanguageParser.DeclaracionVariableSinValorContext context)
54     {
55         string id = context.ID().GetText();
56         string tipo = context.tipo().GetText();
57         ValueWrapper value;
58         switch (tipo) {
59             case "int":
60                 currentEnvironment.DeclaracionVariable(id, new EnteroValue(0), context.Start);
61                 break;
62             case "float64":
63                 //Console.WriteLine((double)0);
64                 currentEnvironment.DeclaracionVariable(id, new FloatValue(0.0f), context.Start);
65                 break;
66             case "bool":
67                 currentEnvironment.DeclaracionVariable(id, new BooleanValue(false), context.Start);
68                 break;
69             case "rune":
70                 currentEnvironment.DeclaracionVariable(id, new RuneValue(0), context.Start);
71                 break;
72             case "string":
73                 currentEnvironment.DeclaracionVariable(id, new StringValue(""), context.Start);
74                 break;
75             default:
76                 break;
77         }
78         return defaultVoid;
79     }
80
81     /*public override ValueWrapper VisitDeclaracionVarStruct(LanguageParser.DeclaracionVarStructContext context)
82     {
83         string id = context.ID().GetText();
84         string tipo = context.tipo().GetText();
85         ValueWrapper value;
86         switch (tipo) {
87             case "int":
88                 currentEnvironment.DeclaracionVariable(id, new EnteroValue(0), context.Start);
89                 break;
90             case "float64":
91                 currentEnvironment.DeclaracionVariable(id, new FloatValue(0.0f), context.Start);
92                 break;
93             case "bool":
94                 currentEnvironment.DeclaracionVariable(id, new BooleanValue(false), context.Start);
95                 break;
96             case "rune":
97                 currentEnvironment.DeclaracionVariable(id, new RuneValue(0), context.Start);
98                 break;
99             case "string":
100                 currentEnvironment.DeclaracionVariable(id, new StringValue(""), context.Start);
101                 break;
102             default:
103                 break;
104         }
105     }
106     */
107 }
```



```

102     return defaultVoid;
103 }*/
104
105
106
107 3 references
108 public override ValueWrapper VisitDeclaracionImplicita(LanguageParser.DeclaracionImplicitaContext context)
109 {
110     string id = context.ID().GetText();
111     ValueWrapper value = Visit(context.expr());
112
113     switch (value)
114     {
115         case EnteroValue:
116             currentEnvironment.DeclaracionVariable(id,value, context.Start);
117             break;
118         case FloatValue:
119             currentEnvironment.DeclaracionVariable(id,value, context.Start);
120             break;
121         case BooleanValue:
122             currentEnvironment.DeclaracionVariable(id,value, context.Start);
123             break;
124         case StringValue:
125             currentEnvironment.DeclaracionVariable(id,value, context.Start);
126             break;
127         case RuneValue:
128             currentEnvironment.DeclaracionVariable(id,value, context.Start);
129             break;
130         default:
131             throw new ErrorSemantico("Error con Declaracion Implica", context.Start);
132     }
133
134     return defaultVoid;
135 }
136
137 3 references
138 public override ValueWrapper VisitDeclaracionSliceValores(LanguageParser.DeclaracionSliceValoresContext context)
139 {
140     string id = context.ID().GetText();
141     List<ValueWrapper> valores = new List<ValueWrapper>();
142     var tipo = context.tipo().GetText();
143
144     foreach(var expression in context.expr()){
145         ValueWrapper value = Visit(expression);
146         if(value is EnteroValue && tipo == "int" || value is FloatValue && tipo == "float64"
147            || value is BooleanValue && tipo == "bool" || value is RuneValue && tipo == "rune"
148            || value is StringValue && tipo == "string"){
149             valores.Add(value);
150         }
151         else
152         {
153             throw new ErrorSemantico("Asignacion de valores diferente al tipo de Slice1", context.Start);
154         }
155     }
156
157     currentEnvironment.DeclaracionVariable(id,new SliceValue(valores),context.Start);
158     return defaultVoid;
159 }

```

```

160
161 3 references
162 public override ValueWrapper VisitDeclaracion2SliceValores(LanguageParser.Declaracion2SliceValoresContext context)
163 {
164     string id = context.ID().GetText();
165     List<List<ValueWrapper>> valoresConjunto = new List<List<ValueWrapper>>();
166
167     var tipo = context.tipo().GetText();
168     //var valores2Slices = context.valores2Slices().valorUnitario();
169     foreach (var conjunto in context.valores2Slices().valorUnitario())
170     {
171         List<ValueWrapper> valoresElemento = new List<ValueWrapper>();
172         foreach(var elemento in conjunto.expr() )
173         {
174             ValueWrapper value = Visit(elemento);
175             if(value is EnteroValue && tipo == "int" || value is FloatValue && tipo == "float64"
176                || value is BooleanValue && tipo == "bool" || value is RuneValue && tipo == "rune"
177                || value is StringValue && tipo == "string" || value is SliceValue){
178                 valoresElemento.Add(value);
179             }
180             else
181             {
182                 throw new ErrorSemantico("Asignacion de valores diferente al tipo de Slice2", context.Start);
183             }
184         }
185
186         //ValueWrapper value2 = Visit(conjunto);
187         //Console.WriteLine(value2);
188         valoresConjunto.Add(valoresElemento);
189
190         //valoresConjunto.Add(valoresElemento);
191     }
192
193     currentEnvironment.DeclaracionVariable(id,new Slice2Value(valoresConjunto),context.Start);
194     return defaultVoid;
195 }
196
197 3 references
198 public override ValueWrapper VisitDeclaracionSliceSinValores(LanguageParser.DeclaracionSliceSinValoresContext context)
199 {
200     string id = context.ID().GetText();
201     List<ValueWrapper> valores = new List<ValueWrapper>();
202     //Console.WriteLine(valores.GetType());
203     currentEnvironment.DeclaracionVariable(id,new SliceValue(valores),context.Start);
204     return defaultVoid;
205 }
206
207 3 references
208 public override ValueWrapper VisitGetslices(LanguageParser.GetslicesContext context)
209 {
210     ValueWrapper value1 = Visit(context.expr(1));
211
212     if(value1 is not EnteroValue)
213     {
214         throw new ErrorSemantico("Indice 1 no es EnteroValue", context.Start);
215     }
216
217     int index1 = ((EnteroValue) value1).valor;

```

```

int index1 = ((EnteroValue)value1).Value;

var valueId = context.expr(0);
string srtId = "none";
if(valueId is LanguageParser.IdContext id)
{
    srtId = id.ID().GetText();
}
ValueWrapper valueSlice = currentEnvironment.GetVariable(srtId, context.Start);
if (valueSlice is SliceValue slice)
{
    List<ValueWrapper> elementos = slice.Elements;
    ValueWrapper newValue = elementos[index1] switch
    {
        EnteroValue 1 => new EnteroValue(1.Value),
        FloatValue 1 => new FloatValue(1.Value),
        StringValue 1 => new StringValue(1.Value),
        BooleanValue 1 => new BooleanValue(1.Value),
        RuneValue 1 => new RuneValue(1.Value),
        SliceValue 1 => new SliceValue(1.Elements),
        _ => throw new ErrorSemantico("Erro Access Slice", context.Start)
    };
    return newValue;
} else if (valueSlice is Slice2Value slice2)
{
    if(context.expr().Length == 3)
    {
        ValueWrapper value2 = Visit(context.expr(2));
        if(value2 is not EnteroValue)
        {
            throw new ErrorSemantico("Indice 2 no es EnteroValue", context.Start);
        }
        int index2 = ((EnteroValue)value2).Value;
        List<List<ValueWrapper>> elementos = slice2.Elements;
        ValueWrapper newValue = elementos[index1][index2] switch
        {
            EnteroValue 1 => new EnteroValue(1.Value),
            FloatValue 1 => new FloatValue(1.Value),
            StringValue 1 => new StringValue(1.Value),
            BooleanValue 1 => new BooleanValue(1.Value),
            RuneValue 1 => new RuneValue(1.Value),
            SliceValue 1 => new SliceValue(1.Elements),
            _ => throw new ErrorSemantico("Erro Access Slice", context.Start)
        };
        return newValue;
    }
    if(context.expr().Length == 2)
    {
        int index2 = ((EnteroValue)value1).Value;
        List<List<ValueWrapper>> elementos = slice2.Elements;
        ValueWrapper newValue = elementos[index1] switch
        {

```

```

275 public override ValueWrapper VisitSliceIndex(LanguageParser.SliceIndexContext context)
276 {
277     var asignado = context.expr(0);
278     if (asignado is LanguageParser.IdContext idContext){
279         Console.WriteLine(asignado);
280         string id = idContext.ID().GetText();
281         ValueWrapper slice = currentEnvironment.GetVariable(id, context.Start);
282         ValueWrapper value = Visit(context.expr(1));
283         if (slice is SliceValue sliceVar){
284             List<ValueWrapper> elementos = sliceVar.Elements;
285             for(int i = 0; i < elementos.Count; i++)
286             {
287                 if (value == elementos[i])
288                 {
289                     return new EnteroValue(i);
290                 }
291             }
292         }
293         return new EnteroValue(-1);
294     }
295 }
296
297 //reference
298 private bool TipoCoincide(string tipoDeclarado, ValueWrapper valor)
299 {
300     return tipoDeclarado switch
301     {
302         "int" => valor is EnteroValue,
303         "float64" => valor is FloatValue,
304         "string" => valor is StringValue,
305         "bool" => valor is BooleanValue,
306         "rune" => valor is RuneValue,
307         _ => false
308     };
309 }
310
311 //3 references
312 public override ValueWrapper VisitIf(LanguageParser.IfContext context)
313 {
314     ValueWrapper cond = Visit(context.expr());
315     if (cond is not BooleanValue)
316     {
317         throw new Exception("Invalid condition");
318     }
319     if ((cond as BooleanValue).Value)
320     {
321         Visit(context.stmt(0));
322     }
323     else if (context.stmt().Length > 1)
324     {
325         Visit(context.stmt(1));
326     }
327 }

```

```

327     }
328
329     return defaultVoid;
330 }
331
332
333 3 references
334 public override ValueWrapper VisitExpression(LanguageParser.ExpressionContext context)
335 {
336     return Visit(context.expr());
337 }
338
339 3 references
340 public override ValueWrapper VisitFmtPrint(LanguageParser.FmtPrintContext context)
341 {
342     foreach (var expr in context.expr()){
343         ValueWrapper value = Visit(expr);
344         output += value switch
345         {
346             EnteroValue a => a.Value.ToString(),
347             FloatValue b => b.Value.ToString("G7", CultureInfo.InvariantCulture),
348             StringValue c => c.Value,
349             BooleanValue d => d.Value.ToString(),
350             RuneValue e => e.Value.ToString(),
351             FunctionValue f => "<function>" + f.name.ToString(),
352             SliceValue s => "<slice> [" + string.Join(",", s.Elements.Select(e =>
353                 e switch {
354                     EnteroValue a => a.value.ToString(),
355                     FloatValue b => b.Value.ToString(CultureInfo.InvariantCulture),
356                     StringValue c => $"\"{c.value}\"",
357                     BooleanValue d => d.Value.ToString(),
358                     RuneValue r => r.Value.ToString(),
359                     FunctionValue f => "<function>" + f.name,
360                     VoidValue => "void",
361                     SliceValue => "<slice>", // prevenir recursión infinita
362                     _ => "[unknown]"
363                 }) + "]",
364             Slice2Value s2 => "<slice2> [" + string.Join(",", s2.Elements.Select(innerList =>
365                 "[" + string.Join(",", innerList.Select(e =>
366                     e switch {
367                         EnteroValue a => a.Value.ToString(),
368                         FloatValue b => b.Value.ToString(CultureInfo.InvariantCulture),
369                         StringValue c => $"\"{c.value}\"",
370                         BooleanValue d => d.Value.ToString(),
371                         RuneValue r => r.Value.ToString(),
372                         FunctionValue f => "<function>" + f.name,
373                         VoidValue => "void",
374                         SliceValue => "<slice>", // prevenir recursión infinita
375                         Slice2Value => "<slice2>", // prevenir recursión infinita
376                         _ => "[unknown]"
377                     }) + "]"
378                 )) + "]"
379             VoidValue v => "Trying to print a Void Value",
380             _ => $"[Tipo desconocido: {value.GetType().Name}]"
381         }

```

```

382     };
383     output += " ";
384 }
385 output += "\n";
386 //ValueWrapper value = Visit(context.expr());
387
388
389 return defaultVoid;
390 }
391
392 3 references
393 public override ValueWrapper VisitCallExpr(LanguageParser.CallExprContext context)
394 {
395     ValueWrapper callee = Visit(context.expr());
396
397     foreach (var call in context.call()){
398     {
399         if (call is LanguageParser.FuncCallContext funcCall){
400
401             if (callee is FunctionValue functionValue)
402             {
403                 callee = VisitCall(functionValue.invocable, funcCall.atri());
404             }
405             else
406             {
407                 throw new ErrorSemantico("Invalid function call", context.Start);
408             }
409         }
410         /*else if (call is LanguageParser.GetContext propertyAccess)
411         {
412             if (callee is InstanceValue instanceValue)
413             {
414                 callee = instanceValue.Instance.Get(propertyAccess.ID().GetText(), propertyAccess.Start);
415             }
416             else
417             {
418                 throw new ErrorSemantico("Propiedad Invalida a la que se quiere acceder", context.Start);
419             }
420         }*/
421     }
422
423     return callee;
424 }
425
426 1 reference
427 public ValueWrapper VisitCall(Invocable invocable, LanguageParser.AtriContext context)
428 {
429     List<ValueWrapper> arguments = new List<ValueWrapper>();
430
431     if (context != null)
432     {
433         foreach (var expr in context.expr())
434         {
435             arguments.Add(Visit(expr));
436         }

```

```

437     }
438
439     // if (context != null && arguments.Count != invocable.Arity())
440     // {
441     //     throw new SemanticError("Invalid number of arguments", context.Start);
442     // }
443
444     return invocable.Invoke(arguments, this);
445
446 }
447
448
449
450
451 3 references
452 public override ValueWrapper VisitAsignacion(LanguageParser.AsignacionContext context)
453 {
454     /*string id = context.ID().GetText();
455     ValueWrapper value = Visit(context.expr());
456     return currentEnvironment.AsignacionVariable(id, value);*/
457     var asignado = context.expr(0);
458     ValueWrapper value = Visit(context.expr(1));
459     //Console.WriteLine(asignado.GetType());
460     if (asignado is LanguageParser.IDContext idContext) {
461
462         string id = idContext.ID().GetText();
463         //Console.WriteLine(id);
464         Console.WriteLine(value.GetType());
465         Console.WriteLine(currentEnvironment.GetVariable(id, context.Start).GetType());
466         if (value.GetType() != currentEnvironment.GetVariable(id, context.Start).GetType())
467         {
468             throw new ErrorSemantico("Diferente tipo de valor a asignar a la variable", context.Start);
469         }
470         currentEnvironment.AsignacionVariable(id, value, context.Start);
471     }
472     /*else if (asignado is LanguageParser.CallExprContext calleContext)
473     {
474
475         ValueWrapper callee = Visit(calleContext.expr());
476         for (int i = 0; i < calleContext.call().Length; i++)
477         {
478             var call = calleContext.call(i);
479             if (i == calleContext.call().Length - 1)
480             {
481                 if (call is LanguageParser.GetTextContext propertyAccess)
482                 {
483                     if (callee is InstanceValue instanceValue)
484                     {
485                         instanceValue.instance.Set(propertyAccess.ID().GetText(), value);
486                     }
487                     else
488                     {
489                         throw new ErrorSemantico("Propiedad Invalida a la que se quiere acceder", context.Start);
490                     }
491                 }
492             }
493         }
494     }*/

```

```

492     }
493
494 }
495
496 }
497
498 if (AsignarSlice.expr().Length == 2)
499 {
500     var valueId = AsignarSlice.expr(0);
501     string srtId = "none";
502     if (valueId is LanguageParser.IDContext id)
503     {
504         srtId = id.ID().GetText();
505     }
506     ValueWrapper valueSlice = currentEnvironment.GetVariable(srtId, context.Start);
507     if (valueSlice is not SliceValue slice)
508     {
509         throw new ErrorSemantico("Tratando de Entrar a una Variable !slice", context.Start);
510     }
511     List<ValueWrapper> elementos = slice.Elements;
512     ValueWrapper value1 = Visit(AsignarSlice.expr(1));
513     if (value1 is not EnteroValue)
514     {
515         throw new ErrorSemantico("Indice 1 no es EnteroValue", context.Start);
516     }
517
518     int index1 = ((EnteroValue)value1).Value;
519
520     elementos[index1] = value;
521     currentEnvironment.AsignacionVariable(srtId, new SliceValue(elementos), context.Start);
522 }
523
524 if (AsignarSlice.expr().Length == 3)
525 {
526     var valueId = AsignarSlice.expr(0);
527     string srtId = "none";
528     if (valueId is LanguageParser.IDContext id)
529     {
530         srtId = id.ID().GetText();
531     }
532     ValueWrapper valueSlice = currentEnvironment.GetVariable(srtId, context.Start);
533     if (valueSlice is not Slice2Value slice)
534     {
535         throw new ErrorSemantico("Tratando de Entrar a una Variable !2Slice", context.Start);
536     }
537
538     List<List<ValueWrapper>> elementos = slice.Elements;
539     ValueWrapper value1 = Visit(AsignarSlice.expr(1));
540     ValueWrapper value2 = Visit(AsignarSlice.expr(2));
541     if (value1 is not EnteroValue)
542     {
543         throw new ErrorSemantico("Indice 1 no es EnteroValue", context.Start);
544     }
545
546     int index1 = ((EnteroValue)value1).Value;
547     if (value2 is not EnteroValue)
548     {
549         throw new ErrorSemantico("Indice 1 no es EnteroValue", context.Start);
550     }
551 }

```

```

550     }
551
552     int index2 = ((EnteroValue)value2).Value;
553     elementos[index1][index2] = value;
554     currentEnvironment.AsignacionVariable(srtId,new Slice2Value(elementos),context.Start);
555 }
556
557
558
559
560
561     }
562     else {
563         throw new ErrorSemantico("Asignacion Invalida", context.Start);
564     }
565     return defaultVoid;
566 }
567
568 3 references
569 public override ValueWrapper VisitId(LanguageParser.IdContext context)
570 {
571     string id = context.ID().GetText();
572     return currentEnvironment.GetVariable(id, context.Start);
573 }
574
575 // VisitParens
576 3 references
577 public override ValueWrapper VisitParentesis(LanguageParser.ParentesisContext context)
578 {
579     return Visit(context.expr());
580 }
581
582 // VisitNegate
583 3 references
584 public override ValueWrapper VisitNegacion(LanguageParser.NegacionContext context)
585 {
586     ValueWrapper value = Visit(context.expr());
587     return value switch
588     {
589         EnteroValue i => new EnteroValue(-i.Value),
590         FloatValue f => new FloatValue(-f.Value),
591         _ => throw new Exception("Invalid operation")
592     };
593 }
594
595 3 references
596 public override ValueWrapper VisitNot(LanguageParser.NotContext context)
597 {
598     ValueWrapper value = Visit(context.expr());
599     return value switch
600     {
601         BooleanValue i => new BooleanValue(!i.Value),
602         _ => throw new ErrorSemantico("Tipo Invalido para operacion !", context.Start)
603     };
604 }
605
606

```

Manejo de Instancias de funciones y struct

```

1 public class Instance {
2
3     2 references
4     private LanguageStruct languageStruct;
5     4 references
6     private Dictionary<string, ValueWrapper> Propiedades;
7
8     1 reference
9     public Instance (LanguageStruct languageStruct){
10         this.languageStruct = languageStruct;
11         Propiedades = new Dictionary<string, ValueWrapper>();
12     }
13
14     0 references
15     public ValueWrapper Get(string name, Antlr4.Runtime.IToken token){
16         if(Propiedades.ContainsKey(name)){
17             return Propiedades[name];
18         }
19         var metodo = languageStruct.GetMetodo(name);
20         if(metodo != null){
21             return new FunctionValue(metodo.Bind(this),name);
22         }
23         throw new ErrorSemantico("Propiedad "+ name + " no existente", token);
24     }
25
26     2 references
27     public void Set(string name, ValueWrapper value){
28         Propiedades[name] = value;
29     }
30 }

```

Definición de la habilidad invocable de una función

```
1 0 references
2 public interface Invocable
3 {
4     4 references
5     int Arity();
6     6 references
7     ValueWrapper Invoke(List<ValueWrapper> args, CompilerVisitor visitor);
8 }
```

Manejo de Errores

```
4 26 references
5 public class ErrorSemantico : Exception
6 {
7     2 references
8     private string desc;
9     3 references
10    private Antlr4.Runtime.IToken token;
11
12    25 references
13    public ErrorSemantico(string desc, Antlr4.Runtime.IToken token)
14    {
15        this.desc = desc;
16        this.token = token;
17    }
18
19    0 references
20    public override string Message
21    {
22        get
23        {
24            return desc + " en línea " + token.Line + ", columna " + token.Column;
25        }
26    }
27 }
28
29 0 references
30 public class LexicalErrorListener : BaseErrorListener, IAntlrErrorListener<int>
31 {
32     0 references
33    public void SyntaxError(TextWriter output, IRecognizer recognizer, int offendingSymbol, int line, int charPositionInLine, string msg, RecognitionException e)
34    {
35        30
36        Console.WriteLine($"Error léxico en línea {line}, columna {charPositionInLine}: {msg}");
37        throw new ParseCanceledException($"Error léxico en línea {line}:{charPositionInLine} - {msg}");
38    }
39 }
40
41 0 references
42 public class SyntaxErrorListener : BaseErrorListener
43 {
44     0 references
45    public override void SyntaxError(TextWriter output, IRecognizer recognizer, IToken offendingSymbol, int line, int charPositionInLine, string msg, RecognitionException e)
46    {
47        40
48        Console.WriteLine($"Error sintáctico en línea {line}, columna {charPositionInLine}: {msg}");
49        throw new ParseCanceledException($"Error sintáctico en línea {line}:{charPositionInLine} - {msg}");
50    }
51 }
```

Definición de propiedades y funciones de Funciones Foráneas

```
7 private LanguageParser.Declaracion_FuncionesContext context;
8
9 5 references
10 public FuncionForeana(Environment closure, LanguageParser.Declaracion_FuncionesContext context){
11     this.closure = closure;
12     this.context = context;
13 }
14
15 2 references
16 public int Arity(){
17     if (context.parametros() == null){
18         return 0;
19     }
20     return context.parametros().ID().Length;
21 }
22
23 3 references
24 public ValueWrapper Invoke(List<ValueWrapper> args, CompilerVisitor visitor){
25     var nuevoEnv = new Environment(closure);
26     var lastEnvBeforeCall = visitor.currentEnvironment;
27     visitor.currentEnvironment = nuevoEnv;
28
29     if (context.parametros() != null){
30         for (int i = 0; i < context.parametros().ID().Length; i++){
31             nuevoEnv.DeclaracionVariables(context.parametros().ID(i).GetText(), args[i], null);
32             //Console.WriteLine(args[i].GetType());
33             //Console.WriteLine(context.parametros().ID(i).GetText() + "-->" + context.parametros().tipo(i).GetText());
34         }
35     }
36     try{
37         foreach (var stmt in context.declaraciones()){
38             visitor.Visit(stmt);
39         }
40     } catch(ReturnEx e){
41         visitor.currentEnvironment = lastEnvBeforeCall;
42         return e.Value;
43     }
44
45     visitor.currentEnvironment = lastEnvBeforeCall;
46     return visitor.defaultVoid;
47 }
48
49 //Cambiar el this tratando de mandar por parametros el nombre asignado
50 2 references
51 public FuncionForeana Bind(Instance instance){
52     var EnvOculto = new Environment(closure);
53     EnvOculto.DeclaracionVariable("this", new InstanceValue(instance), null);
54     return new FuncionForeana(EnvOculto, context);
55 }
56 }
```

Manejo de FrontEnd reactivo responsivo

```
1 'use client';
2 import { Editor } from "@monaco-editor/react";
3 import { useState } from "react";
4
5
6
7 const API_URL = 'http://localhost:5020';
8
9 export default function Home() {
10
11     const [entrada, setEntrada] = useState<string>('');
12     const [salida, setSalida] = useState<string>('');
13
14
15     const handleExecute = () => {
16         fetch(`${API_URL}/compile`, {
17             method: 'POST',
18             headers: {
19                 'Content-Type': 'application/json',
20             },
21             body: JSON.stringify({ entrada }),
22         })
23         .then((response) => response.json())
24         .then(data => {
25             setSalida(data.result);
26         })
27         .catch(error => {
28             console.error('Error:', error);
29         });
30     };
31
32     const handleCrearArchivo = () => {
33         setEntrada(''); // Limpiar el editor
34         setSalida(''); // Limpiar la consola
35     };
36
37     const handleOpenFile = (event: React.ChangeEvent<HTMLInputElement>) => {
38         const file = event.target.files?.[0]; // Asegurar que hay un archivo seleccionado
39         if (!file) return;
40
41         const reader = new FileReader();
42         reader.onload = (e) => {
43             setEntrada(e.target?.result as string); // Convertir resultado a string
44         };
45         reader.readAsText(file);
46     };
47
48
49
50     return (
51         <div className="flex flex-col justify-center min-h-screen w-3/4 mx-auto gap-4">
52             <h1 className="text-8xl font-bold"><center>GoLight</center></h1>
53
54             <div className="flex gap-4">
55                 <button className="px-4 py-2 bg-blue-500 text-white rounded-lg font-bold text-2xl" onClick={handleCrearArchivo}> Crear Archivo </button>
56                 <label className="px-4 py-2 bg-blue-500 text-white rounded-lg font-bold text-2xl cursor-pointer">

```

```
49 export default function Home() {
50
51   return (
52     <div className="flex flex-col justify-center min-h-screen w-3/4 mx-auto gap-4">
53       <h1 className="text-8xl font-bold"><center>GoLight</center></h1>
54
55       <div className="flex gap-4">
56         <button className="px-4 py-2 bg-blue-500 text-white rounded-lg font-bold text-2xl" onClick={handleCrearArchivo}> Crear Archivo </button>
57         <label className="px-4 py-2 bg-blue-500 text-white rounded-lg font-bold text-2xl cursor-pointer">
58           <center>Abrir Archivo</center>
59         <input type="file" accept=".txt,.go" className="hidden" onChange={handleOpenFile} />
60       </div>
61       <div>
62         <button className="px-4 py-2 bg-blue-500 text-white rounded-lg font-bold text-2xl" > Guardar Archivo </button>
63         <button className="px-4 py-2 bg-green-500 text-white rounded-lg font-bold text-2xl ml-175" onClick={handleExecute}> Ejecutar </button>
64         <button className="px-4 py-2 bg-purple-500 text-white rounded-lg font-bold text-2xl ml-85"> Reporte AST </button>
65         <button className="px-4 py-2 bg-yellow-500 text-white rounded-lg font-bold text-2xl"> Reporte Errores </button>
66         <button className="px-4 py-2 bg-blue-500 text-white rounded-lg font-bold text-2xl"> Tabla de Símbolos </button>
67       </div>
68
69       <div className="flex gap-6 w-full h-[75vh]">
70         <div className="w-[60%] h-full border rounded-lg">
71           <div>
72             <div>
73               <div>
74                 <div>
75                   <div>
76                     <div>
77                       <div>
78                         <div>
79                           <div>
80                             <div>
81                               <div>
82                                 <div>
83                                   <div>
84                                     <div>
85                                       <div>
86                                         <div>
87                                           <div>
88                                             <div>
89                                               <div>
90                                                 <div>
91                                                 </div>
92                                               </div>
93                                             </div>
94                                           </div>
95                                         </div>
96                                       </div>
97                                     </div>
98                                   </div>
99                                 </div>
100                               </div>
101                             </div>
102                           </div>
103                         </div>
104                       </div>
105                     </div>
106                   </div>
107                 </div>
108               </div>
109             </div>
110           </div>
111         <div>
112           <div>
113             <div>
114               <div>
115                 <div>
116                   <div>
117                     <div>
118                       <div>
119                         <div>
120                           <div>
121                             <div>
122                               <div>
123                               </div>
124                             </div>
125                           </div>
126                         </div>
127                       </div>
128                     </div>
129                   </div>
130                 </div>
131               </div>
132             </div>
133           </div>
134         </div>
135       </div>
136     </div>
137   );
138 }
139 }
```

Definición de Responsable de ejecución Compile.cs

```
9 using Antlr4.Runtime.Tree;
10 using Microsoft.AspNetCore.Mvc;
11 using Microsoft.Extensions.Logging;
12
13 namespace api.Controllers
14 {
15   [Route("[controller]")]
16   3 references
17   public class Compile : Controller
18   {
19     1 reference
20     private readonly ILogger<Compile> _logger;
21
22     0 references
23     public Compile(ILogger<Compile> logger)
24     {
25       _logger = logger;
26     }
27
28     1 reference
29     public class CompileRequest
30     {
31       [Required]
32       1 reference
33       public required string entrada { get; set; }
34     }
35
36     // POST /compile
37     [HttpPost]
38     0 references
39     public IActionResult Post([FromBody] CompileRequest request)
40     {
41       if (!ModelState.IsValid)
42       {
43         return BadRequest(new { error = "Invalid request" });
44       }
45
46       var inputStream = new AntlrInputStream(request.entrada);
47       var lexer = new LanguageLexer(inputStream);
48       var tokens = new CommonTokenStream(lexer);
49       var parser = new LanguageParser(tokens);
50
51       var tree = parser.program();
52
53       var visitor = new CompilerVisitor();
54       visitor.Visit(tree);
55
56       return Ok(new { result = visitor.output });
57     }
58   }
59 }
```








