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+

.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      ';'?                                                    # Expresion
    | '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 CompileVisitor para el manejo de cada definición, declaración, statement, etc de la gràmatica

```
using System.Globalization;
using System.Text;
using analyzer;
using Microsoft.VisualBasic;

6 references
public class CompilerVisitor : LanguageBaseVisitor<ValueWrapper>
{

    4 references
    public string output = "";
    51 references
    public Environment currentEnvironment;
    18 references
    public ValueWrapper defaultVoid = new VoidValue();



    1 reference
    public CompilerVisitor()
    {
        currentEnvironment = new Environment(null);
        Embeded.Generate(currentEnvironment);
    }

    3 references
    public override ValueWrapper VisitProgram(LanguageParser.ProgramContext context)
    {

        foreach (var dcl in context.declaraciones())
        {
            Visit(dcl);
        }
        return defaultVoid;

    }


    3 references
    public override ValueWrapper VisitDeclaracionVariableConValor(LanguageParser.DeclaracionVariableConValorContext context)
    {
        /*string texto = null;
        Console.WriteLine(texto + " -> null");*/
        string id = context.ID().GetText();
        string tipo = context.tipo().GetText();
        ValueWrapper value = Visit(context.expr());

        if (!TipoCoincide(tipo, value))
        {
            throw new Exception($"Error: No se puede asignar un valor de tipo {value.GetType()} a la variable '{id}' de tipo {tipo}.");
        }

        currentEnvironment.DeclaracionVariable(id, value, context.Start);
        return defaultVoid;
    }
```

```
    }

    3 references
    public override ValueWrapper VisitDeclaracionVariableSinValor(LanguageParser.DeclaracionVariableSinValorContext context)
    {
        string id = context.ID().GetText();
        string tipo = context.tipo().GetText();
        ValueWrapper value;
        switch (tipo) {
            case "int":
                currentEnvironment.DeclaracionVariable(id, new EnteroValue(0), context.Start);
                break;
            case "float64":
                //Console.WriteLine((double)0);
                currentEnvironment.DeclaracionVariable(id, new FloatValue(0.0f), context.Start);
                break;
            case "bool":
                currentEnvironment.DeclaracionVariable(id, new BooleanValue(false), context.Start);
                break;
            case "rune":
                currentEnvironment.DeclaracionVariable(id, new RuneValue(0), context.Start);
                break;
            case "string":
                currentEnvironment.DeclaracionVariable(id, new StringValue(""), context.Start);
                break;
            default:
                break;
        }
        return defaultVoid;
    }

    /*public override ValueWrapper VisitDeclaracionVarStruct(LanguageParser.DeclaracionVarStructContext context)
    {
        string id = context.ID().GetText();
        string tipo = context.tipo().GetText();
        ValueWrapper value;
        switch (tipo) {
            case "int":
                currentEnvironment.DeclaracionVariable(id, new EnteroValue(0), context.Start);
                break;
            case "float64":
                currentEnvironment.DeclaracionVariable(id, new FloatValue(0.0f), context.Start);
                break;
            case "bool":
                currentEnvironment.DeclaracionVariable(id, new BooleanValue(false), context.Start);
                break;
            case "rune":
                currentEnvironment.DeclaracionVariable(id, new RuneValue(0), context.Start);
                break;
            case "string":
                currentEnvironment.DeclaracionVariable(id, new StringValue(""), context.Start);
                break;
            default:
                break;
        }
```

```
        return defaultVoid;
    }*/


    3 references
    public override ValueWrapper VisitDeclaracionImplicita(LanguageParser.DeclaracionImplicitaContext context)
{
    string id = context.ID().GetText();
    ValueWrapper value = Visit(context.expr());

    switch (value)
    {
        case EnteroValue:
            currentEnvironment.DeclaracionVariable(id,value, context.Start);
            break;
        case FloatValue:
            currentEnvironment.DeclaracionVariable(id,value, context.Start);
            break;
        case BooleanValue:
            currentEnvironment.DeclaracionVariable(id,value, context.Start);
            break;
        case StringValue:
            currentEnvironment.DeclaracionVariable(id,value, context.Start);
            break;
        case RuneValue:
            currentEnvironment.DeclaracionVariable(id,value, context.Start);
            break;
        default:
            throw new ErrorSemantico("Error con Declaracion Implica", context.Start);
    }

    return defaultVoid;
}
    3 references
    public override ValueWrapper VisitDeclaracionSliceValores(LanguageParser.DeclaracionSliceValoresContext context)
    {
        string id = context.ID().GetText();
        List<ValueWrapper> valores = new List<ValueWrapper>();
        var tipo = context.tipo().GetText();

        foreach(var expresion in context.expr()){
            ValueWrapper value = Visit(expresion);
            if(value is EnteroValue && tipo == "int" || value is FloatValue && tipo == "float64"
            || value is BooleanValue && tipo == "bool" || value is RuneValue && tipo == "rune"
            || value is StringValue && tipo == "string"){
                valores.Add(value);
            } else
            {
                throw new ErrorSemantico("Asignacion de valores diferente al tipo de Slice1", context.Start);
            }
        }
        currentEnvironment.DeclaracionVariable(id,new SliceValue(valores),context.Start);
        return defaultVoid;
    }
    3 references
    public override ValueWrapper VisitDeclaracion2SliceValores(LanguageParser.Declaracion2SliceValoresContext context)
    {
        string id = context.ID().GetText();
        List<List<ValueWrapper>> valoresConjunto = new List<List<ValueWrapper>>();

        var tipo = context.tipo().GetText();
        //var valores2Slices = context.valores2Slices().valorUnitario();
        foreach (var conjunto in context.valores2Slices().valorUnitario())
        {
            List<ValueWrapper> valoresElemento = new List<ValueWrapper>();
            foreach(var elemento in conjunto.expr() )
            {
                ValueWrapper value = Visit(elemento);
                if(value is EnteroValue && tipo == "int" || value is FloatValue && tipo == "float64"
                || value is BooleanValue && tipo == "bool" || value is RuneValue && tipo == "rune"
                || value is StringValue && tipo == "String" || value is SliceValue){
                    valoresElemento.Add(value);
                } else
                {
                    throw new ErrorSemantico("Asignacion de valores diferente al tipo de Slice2", context.Start);
                }
            }

            //ValueWrapper value2 = Visit(conjunto);
            //Console.WriteLine(value2);
            valoresConjunto.Add(valoresElemento);

            //valoresConjunto.Add(valoresElemento);
        }
        currentEnvironment.DeclaracionVariable(id,new Slice2Value(valoresConjunto),context.Start);
        return defaultVoid;
    }
    3 references
    public override ValueWrapper VisitDeclaracionSliceSinValores(LanguageParser.DeclaracionSliceSinValoresContext context)
    {
        string id = context.ID().GetText();
        List<ValueWrapper> valores = new List<ValueWrapper>();
        //Console.WriteLine(valores.GetType());
        currentEnvironment.DeclaracionVariable(id,new SliceValue(valores),context.Start);
        return defaultVoid;
    }

    3 references
    public override ValueWrapper VisitGetSlices(LanguageParser.GetSlicesContext context)
    {
        ValueWrapper value1 = Visit(context.expr(1));


        if(value1 is not EnteroValue)
        {
            throw new ErrorSemantico("Indice 1 no es EnteroValue", context.Start);
        }

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

```
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 l => new EnteroValue(l.Value),
        FloatValue l => new FloatValue(l.Value),
        StringValue l => new StringValue(l.Value),
        BooleanValue l => new BooleanValue(l.Value),
        RuneValue l => new RuneValue(l.Value),
        SliceValue l => new SliceValue(l.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 l => new EnteroValue(l.Value),
            FloatValue l => new FloatValue(l.Value),
            StringValue l => new StringValue(l.Value),
            BooleanValue l => new BooleanValue(l.Value),
            RuneValue l => new RuneValue(l.Value),
            SliceValue l => new SliceValue(l.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
        {
```

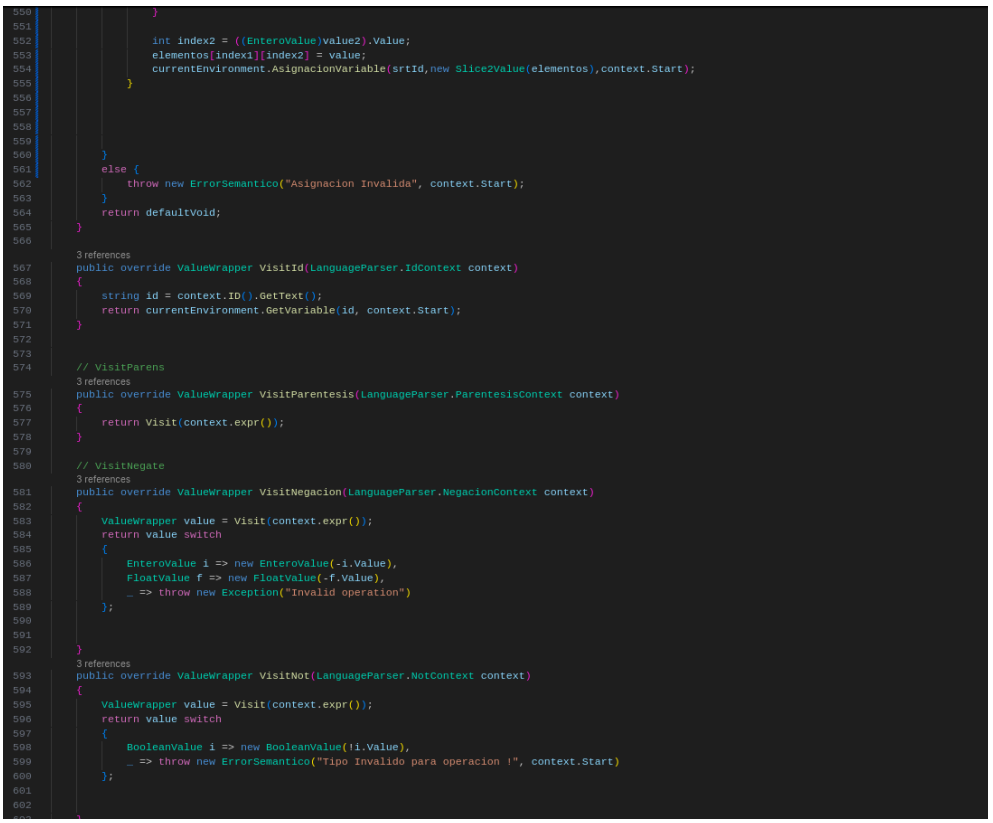
```
275     public override ValueWrapper VisitSliceIndex(LanguageParser.SliceIndexContext context)
276     {
277         var asignado = context.expr(0);
278         if (asignado is LanguageParser.IdContext idContext){
279             Console.WriteLine(asignado);
280             string id = idContext.ID().GetText();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

```
        }

        return defaultVoid;
    }


    3 references
    public override ValueWrapper VisitExpresion(LanguageParser.ExpresionContext context)
    {
        return Visit(context.expr());
    }

    3 references
    public override ValueWrapper VisitFmtPrint(LanguageParser.FmtPrintContext context)
    {

        foreach (var expr in context.expr()){
            ValueWrapper value = Visit(expr);
            output += value switch
                {
                    EnteroValue a => a.Value.ToString(),
                    FloatValue b => b.Value.ToString("G7", CultureInfo.InvariantCulture),
                    StringValue c => c.Value,
                    BooleanValue d => d.Value.ToString(),
                    RuneValue e => e.Value.ToString(),
                    FunctionValue f => "<function>" + f.name.ToString(),
                    SliceValue s => "<slice> [" + string.Join(",", s.Elements.Select(e =>
                        e switch {
                            EnteroValue a => a.Value.ToString(),
                            FloatValue b => b.Value.ToString(CultureInfo.InvariantCulture),
                            StringValue c => $"\"{c.Value}\"",
                            BooleanValue d => d.Value.ToString(),
                            RuneValue r => r.Value.ToString(),
                            FunctionValue f => "<function>" + f.name,
                            VoidValue => "void",
                            SliceValue => "<slice>", // prevenir recursión infinita
                            _ => "[unknown]"
                        }
                    )) + "]",
                    Slice2Value s2 => "<slice2> [" + string.Join(",", s2.Elements.Select(innerList =>
                        "[" + string.Join(",", innerList.Select(e =>
                            e switch {
                                EnteroValue a => a.Value.ToString(),
                                FloatValue b => b.Value.ToString(CultureInfo.InvariantCulture),
                                StringValue c => $"\"{c.Value}\"",
                                BooleanValue d => d.Value.ToString(),
                                RuneValue r => r.Value.ToString(),
                                FunctionValue f => "<function>" + f.name,
                                VoidValue => "void",
                                SliceValue => "<slice>", // prevenir recursión infinita
                                Slice2Value => "<slice2>", // prevenir recursión infinita
                                _ => "[unknown]"
                            }
                        )) + "]"
                    )) + "]",
                    VoidValue v => "Trying to print a Void Value",
                    _ => $"[Tipo desconocido: {value.GetType().Name}]"
```

```
                };
                output += " ";
            }
            output += "\n";
        //ValueWrapper value = Visit(context.expr());


        return defaultVoid;
    }

    3 references
    public override ValueWrapper VisitCallExpr(LanguageParser.CallExprContext context)
    {
        ValueWrapper callee = Visit(context.expr());


        foreach (var call in context.call())
        {
            if (call is LanguageParser.FunCallContext funCall){

                if (callee is FunctionValue functionValue)
                {
                    callee = VisitCall(functionValue.invocable, funCall.atri());
                }
                else
                {
                    throw new ErrorSemantico("Invalid function call", context.Start);
                }
            }
            /*else if (call is LanguageParser.GetContext propertyAccess)
            {
                if (callee is InstanceValue instanceValue)
                {
                    callee = instanceValue.instance.Get(propertyAccess.ID().GetText(), propertyAccess.Start);
                }
                else
                {
                    throw new ErrorSemantico("Propiedad Invalida a la que se quiere acceder", context.Start);
                }
            }*/
        }

        return callee;
    }

    1 reference
    public ValueWrapper VisitCall(Invocable invocable, LanguageParser.AtriContext context)
    {

        List<ValueWrapper> arguments = new List<ValueWrapper>();

        if (context != null)
        {
            foreach (var expr in context.expr())
            {
                arguments.Add(Visit(expr));
            }
```

```
        }

        // if (context != null && arguments.Count != invocable.Arity())
        // {
        //     throw new SemanticError("Invalid number of arguments", context.Start);
        // }

        return invocable.Invoke(arguments, this);

    }



    3 references
    public override ValueWrapper VisitAsignacion(LanguageParser.AsignacionContext context)
    {
        /*string id = context.ID().GetText();
        ValueWrapper value = Visit(context.expr());
        return currentEnvironment.AsignacionVariable(id, value);*/
        var asignado = context.expr(0);
        ValueWrapper value = Visit(context.expr(1));
        //Console.WriteLine(asignado.GetType());
        if (asignado is LanguageParser.IdContext idContext){

            string id = idContext.ID().GetText();
            /*Console.WriteLine(id);
            Console.WriteLine(value.GetType());
            Console.WriteLine(currentEnvironment.GetVariable(id,context.Start).GetType());*/
            if (value.GetType() != currentEnvironment.GetVariable(id,context.Start).GetType())
            {
                throw new ErrorSemantico("Diferente tipo de valor a asignar a la variable", context.Start);
            }
            currentEnvironment.AsignacionVariable(id, value, context.Start);

        } /*else if (asignado is LanguageParser.CallExprContext calleContext)
        {

            ValueWrapper callee = Visit(calleContext.expr());
            for( int i = 0; i < calleContext.call().Length; i++)
            {
                var call = calleContext.call(i);
                if (i == calleContext.call().Length - 1)
                {

                    if (call is LanguageParser.GetContext propertyAccess)
                    {
                        if (callee is InstanceValue instanceValue)
                        {
                            instanceValue.instance.Set(propertyAccess.ID().GetText(), value);
                        }
                        else
                        {
                            throw new ErrorSemantico("Propiedad Invalida a la que se quiere acceder", context.Start);
                        }
                    }
                }
```

```
                }
            } */else if(asignado is LanguageParser.GetSlicesContext AsignarSlice)
            {
                if (AsignarSlice.expr().Length == 2)
                {
                    var valueId = AsignarSlice.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 not SliceValue slice)
                    {
                        throw new ErrorSemantico("Tratando de Entrar a una Variable !Slice", context.Start);
                    }

                    List<ValueWrapper> elementos = slice.Elements;
                    ValueWrapper value1 = Visit(AsignarSlice.expr(1));

                    if(value1 is not EnteroValue)
                    {
                        throw new ErrorSemantico("Indice 1 no es EnteroValue", context.Start);
                    }

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

                    elementos[index1] = value;
                    currentEnvironment.AsignacionVariable(srtId,new SliceValue(elementos),context.Start);
                }
                if (AsignarSlice.expr().Length == 3)
                {
                    var valueId = AsignarSlice.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 not Slice2Value slice)
                    {
                        throw new ErrorSemantico("Tratando de Entrar a una Variable !2Slice", context.Start);
                    }

                    List<List<ValueWrapper>> elementos = slice.Elements;
                    ValueWrapper value1 = Visit(AsignarSlice.expr(1));
                    ValueWrapper value2 = Visit(AsignarSlice.expr(2));
                    if(value1 is not EnteroValue)
                    {
                        throw new ErrorSemantico("Indice 1 no es EnteroValue", context.Start);
                    }

                    int index1 = ((EnteroValue)value1).Value;
                    if(value2 is not EnteroValue)
                    {
                        throw new ErrorSemantico("Indice 1 no es EnteroValue", context.Start);
```



```
                }

                int index2 = ((EnteroValue)value2).Value;
                elementos[index1][index2] = value;
                currentEnvironment.AsignacionVariable(srtId,new Slice2Value(elementos),context.Start);
            }



        }
        else {
            throw new ErrorSemantico("Asignacion Invalida", context.Start);
        }
        return defaultVoid;
    }

    3 references
    public override ValueWrapper VisitId(LanguageParser.IdContext context)
    {
        string id = context.ID().GetText();
        return currentEnvironment.GetVariable(id, context.Start);
    }


    // VisitParens
    3 references
    public override ValueWrapper VisitParentesis(LanguageParser.ParentesisContext context)
    {
        return Visit(context.expr());
    }

    // VisitNegate
    3 references
    public override ValueWrapper VisitNegacion(LanguageParser.NegacionContext context)
    {
        ValueWrapper value = Visit(context.expr());
        return value switch
        {
            EnteroValue i => new EnteroValue(-i.Value),
            FloatValue f => new FloatValue(-f.Value),
            _ => throw new Exception("Invalid operation")
        };


    }
    3 references
    public override ValueWrapper VisitNot(LanguageParser.NotContext context)
    {
        ValueWrapper value = Visit(context.expr());
        return value switch
        {
            BooleanValue i => new BooleanValue(!i.Value),
            _ => throw new ErrorSemantico("Tipo Invalido para operacion !", context.Start)
        };


    }
```

Manejo de Instancias de funciones y struct

```
4 references
public class Instance {

    2 references
    private LanguageStruct languageStruct;
    4 references
    private Dictionary<string, ValueWrapper> Propiedades;

    1 reference
    public Instance (LanguageStruct languageStruct){
        this.languageStruct = languageStruct;
        Propiedades = new Dictionary<string, ValueWrapper>();


    }

    0 references
    public ValueWrapper Get(string name, Antlr4.Runtime.IToken token){
        if(Propiedades.ContainsKey(name)){
            return Propiedades[name];
        }
        var metodo = languageStruct.GetMetodo(name);
        if(metodo != null){
            return new FunctionValue(metodo.Bind(this),name);
        }
        throw new ErrorSemantico("Propiedad "+ name + " no existente", token);
    }

    2 references
    public void Set(string name, ValueWrapper value){
        Propiedades[name] = value;
    }

}
```

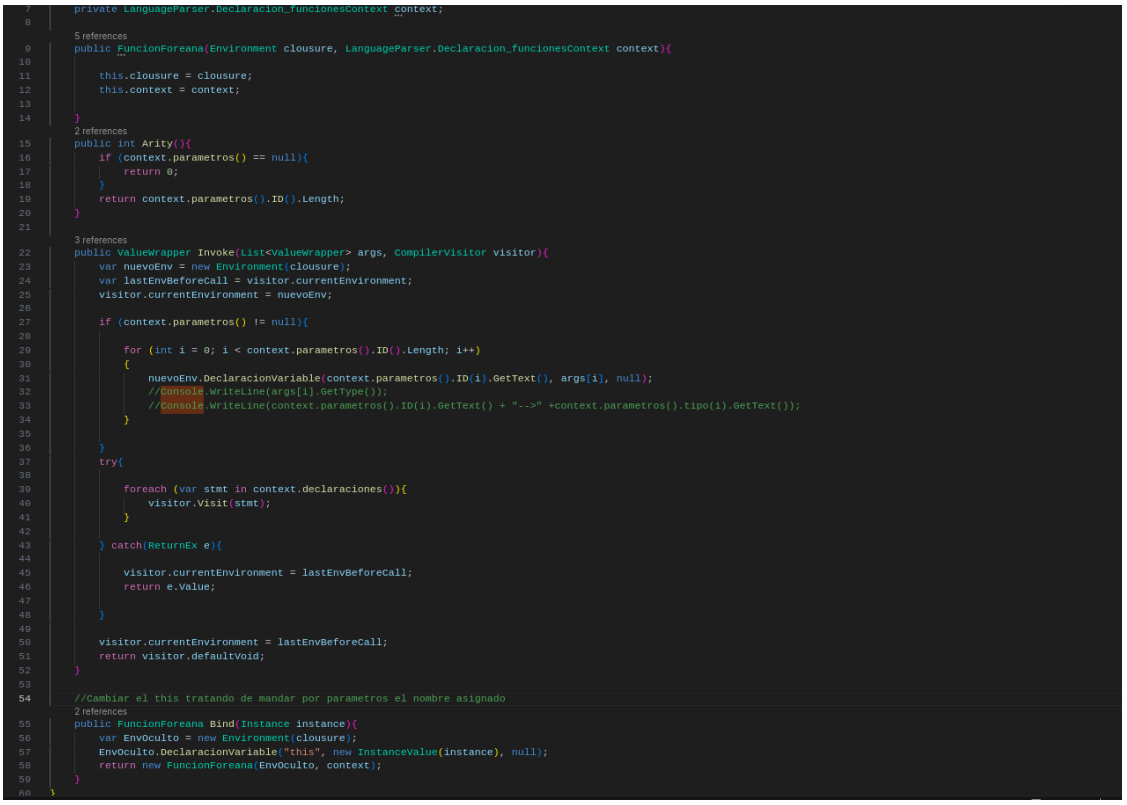
Definición de la habilidad invocable de una función

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

Manejo de Errores

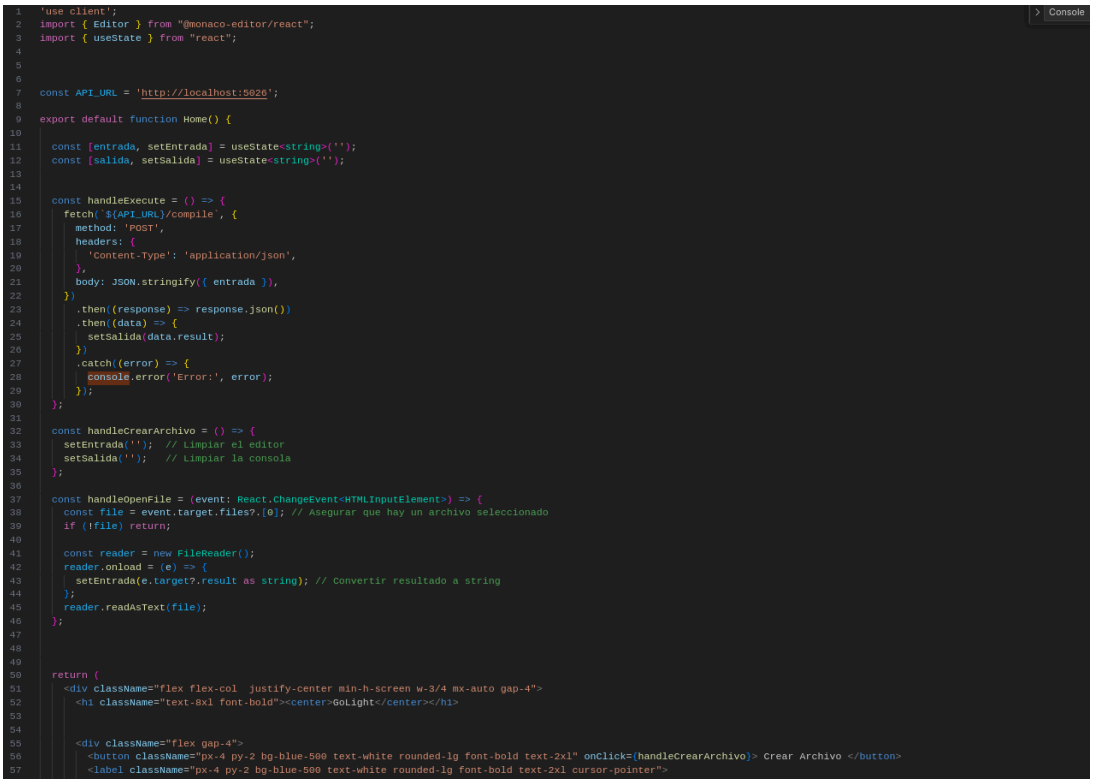
```
26 references
public class ErrorSemantico : Exception
{

    2 references
    private string desc;
    3 references
    private Antlr4.Runtime.IToken token;

    25 references
    public ErrorSemantico(string desc, Antlr4.Runtime.IToken token)
    {
        this.desc = desc;
        this.token = token;
    }

    0 references
    public override string Message
    {
        get
        {
            return desc + " en línea " + token.Line + ", columna " + token.Column;
        }
    }
}


0 references
public class LexicalErrorListener : BaseErrorListener, IAntlrErrorListener<int>
{
    0 references
    public void SyntaxError(TextWriter output, IRecognizer recognizer, int offendingSymbol, int line, int charPositionInLine, string msg, RecognitionException e)
    {
        Console.WriteLine($"Error léxico en línea {line}, columna {charPositionInLine}: {msg}");
        throw new ParseCanceledException($"Error léxico en línea {line}:{charPositionInLine} - {msg}");
    }

}


0 references
public class SyntaxErrorListener : BaseErrorListener
{
    0 references
    public override void SyntaxError(TextWriter output, IRecognizer recognizer, IToken offendingSymbol, int line, int charPositionInLine, string msg, RecognitionException e)
    {
        Console.WriteLine($"Error sintáctico en línea {line}, columna {charPositionInLine}: {msg}");
        throw new ParseCanceledException($"Error sintáctico en línea {line}:{charPositionInLine} - {msg}");
    }
}
```

Definición de propiedades y funciones de Funciones Foráneas



```
    private LanguageParser.Declaracion_funcionesContext context;

    5 references
    public FuncionForeana(Environment clousure, LanguageParser.Declaracion_funcionesContext context){

        this.clousure = clousure;
        this.context = context;

    }
    2 references
    public int Arity(){
        if (context.parametros() == null){
            return 0;
        }
        return context.parametros().ID().Length;
    }

    3 references
    public ValueWrapper Invoke(List<ValueWrapper> args, CompilerVisitor visitor){
        var nuevoEnv = new Environment(clousure);
        var lastEnvBeforeCall = visitor.currentEnvironment;
        visitor.currentEnvironment = nuevoEnv;

        if (context.parametros() != null){

            for (int i = 0; i < context.parametros().ID().Length; i++)
            {
                nuevoEnv.DeclaracionVariable(context.parametros().ID(i).GetText(), args[i], null);
                //Console.WriteLine(args[i].GetType());
                //Console.WriteLine(context.parametros().ID(i).GetText() + "-->" +context.parametros().tipo(i).GetText());
            }

        }
        try{

            foreach (var stmt in context.declaraciones()){
                visitor.Visit(stmt);
            }

        } catch(ReturnEx e){

            visitor.currentEnvironment = lastEnvBeforeCall;
            return e.Value;

        }

        visitor.currentEnvironment = lastEnvBeforeCall;
        return visitor.defaultVoid;
    }

    //Cambiar el this tratando de mandar por parametros el nombre asignado
    2 references
    public FuncionForeana Bind(Instance instance){
        var EnvOculto = new Environment(clousure);
        EnvOculto.DeclaracionVariable("this", new InstanceValue(instance), null);
        return new FuncionForeana(EnvOculto, context);
    }
}
```

Manejo de FronteEnd reactivo responsivo



```
'use client';
import { Editor } from "@monaco-editor/react";
import { useState } from "react";



const API_URL = 'http://localhost:5026';

export default function Home() {

  const [entrada, setEntrada] = useState<string>('');
  const [salida, setSalida] = useState<string>('');


  const handleExecute = () => {
    fetch(`${API_URL}/compile`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({ entrada }),
    })
      .then((response) => response.json())
      .then((data) => {
        setSalida(data.result);
      })
      .catch((error) => {
        console.error('Error:', error);
      });
  };

  const handleCrearArchivo = () => {
    setEntrada('');  // Limpiar el editor
    setSalida('');   // Limpiar la consola
  };

  const handleOpenFile = (event: React.ChangeEvent<HTMLInputElement>) => {
    const file = event.target.files?.[0]; // Asegurar que hay un archivo seleccionado
    if (!file) return;

    const reader = new FileReader();
    reader.onload = (e) => {
      setEntrada(e.target?.result as string); // Convertir resultado a string
    };
    reader.readAsText(file);
  };



  return (
    <div className="flex flex-col  justify-center min-h-screen w-3/4 mx-auto gap-4">
      <h1 className="text-8xl font-bold"><center>GoLight</center></h1>


      <div className="flex gap-4">
        <button className="px-4 py-2 bg-blue-500 text-white rounded-lg font-bold text-2xl" onClick={handleCrearArchivo}> Crear Archivo </button>
        <label className="px-4 py-2 bg-blue-500 text-white rounded-lg font-bold text-2xl cursor-pointer">
```

```
export default function Home() {

  return (
    <div className="flex flex-col  justify-center min-h-screen w-3/4 mx-auto gap-4">
      <h1 className="text-8xl font-bold"><center>GoLight</center></h1>


      <div className="flex gap-4">
        <button className="px-4 py-2 bg-blue-500 text-white rounded-lg font-bold text-2xl" onClick={handleCrearArchivo}> Crear Archivo </button>
        <label className="px-4 py-2 bg-blue-500 text-white rounded-lg font-bold text-2xl cursor-pointer">
        <center>Abrir Archivo</center>
        <input type="file" accept=".txt,.go" className="hidden" onChange={handleOpenFile} />
      </label>
        <button className="px-4 py-2 bg-blue-500 text-white rounded-lg font-bold text-2xl" > Guardar Archivo </button>
        <button className="px-4 py-2 bg-green-500 text-white rounded-lg font-bold text-2xl ml-175" onClick={handleExecute}> Ejecutar </button>
        <button className="px-4 py-2 bg-purple-500 text-white rounded-lg font-bold text-2xl ml-85"> Reporte AST </button>
        <button className="px-4 py-2 bg-yellow-500 text-white rounded-lg font-bold text-2xl"> Reporte Errores </button>
        <button className="px-4 py-2 bg-blue-500 text-white rounded-lg font-bold text-2xl"> Tabla de Simbolos </button>


      </div>

      <div className="flex gap-6 w-full h-[75vh]">
      <div className="w-[60%] h-full border rounded-lg">
            <Editor
            height="75vh"
            //width="100vh"
            defaultLanguage="go"
            className="w-full"
            options={{ fontSize: 20 }}
            value={entrada}
            onChange={(value) => setEntrada(value || '')}
            />
            </div>
            <div className="w-[40%] h-full border rounded-lg p-4 bg-gray-100 overflow-auto">
            <h2 className="text-lg font-bold">Consola:</h2>
            <pre className="whitespace-pre-wrap text-3xl">{salida || "No output yet"}</pre>
            </div>

          </div>
    </div>


  );
}
```

Definición de Responsable de ejecución Compie.cs

```
using Antlr4.Runtime.Tree;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;

namespace api.Controllers
{
    [Route("[controller]")]
    3 references
    public class Compile : Controller
    {
        1 reference
        private readonly ILogger<Compile> _logger;

        0 references
        public Compile(ILogger<Compile> logger)
        {
            _logger = logger;
        }

        1 reference
        public class CompileRequest
        {
            [Required]
            1 reference
            public required string entrada { get; set; }
        }

        // POST /compile
        [HttpPost]
        0 references
        public IActionResult Post([FromBody] CompileRequest request)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(new { error = "Invalid request" });
            }

            var inputStream = new AntlrInputStream(request.entrada);
            var lexer = new LanguageLexer(inputStream);
            var tokens = new CommonTokenStream(lexer);
            var parser = new LanguageParser(tokens);

            var tree = parser.program();

            var visitor = new CompilerVisitor();
            visitor.Visit(tree);

            return Ok(new { result = visitor.output });


        }

    }
}
```