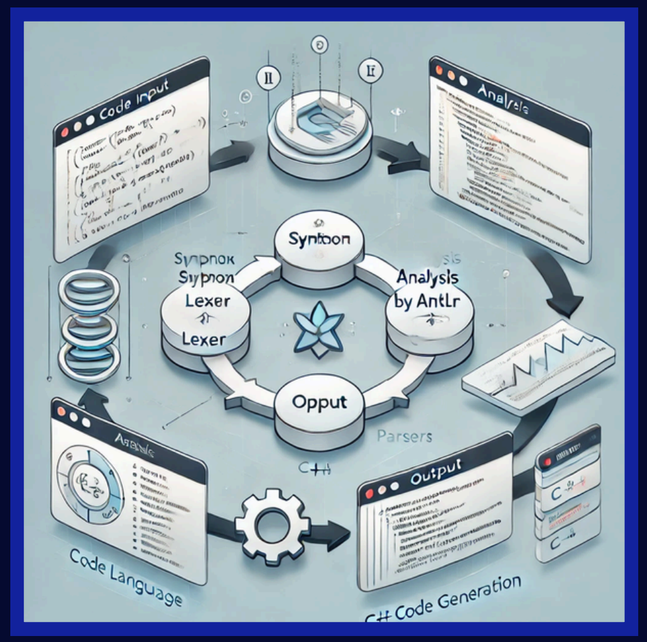
TRABAJO PARCIAL

- Alejandro Olaf López Flores
- Ian Joaquin Sanchez Alva
- Gabriel Alonso Reyna Alvarado
- Ibrahim Imanol Jordi Arquinigo Jacinto

PROBLEMA

- Desarrollo de un lenguaje propio para hispanohablantes.
- Uso de ANTLR para generar analizadores léxicos y sintácticos.
- Implementación del lenguaje en C++.
- Desafío: diseño de una gramática robusta y código eficiente.



MOTIVACIÓN

Debido al crecimiento de la industria de la tecnología,en especial de la inteligencia artificial. Un gran público quiere aprender sobre esta, en vista de busca de este conocimiento deciden aprender a programar.

Nuestro lenguaje basado en C++, busca lograr ayudar a este público novato a adentrarse dentro de este mundo con mayor facilidad

INVESTIGACIÓN

- Implementación exitosa del driver con ANTLR4 y C++.
- Uso de memoria y una buena gramática
- Aprendizaje de errores con ejemplos de códigos en C++.
- Modularización para separar componentes y aumentar independencia y control.
- Control del comportamiento de la gramática y manejo de errores.
- Colocación del DLL en windows/32 para facilitar la implementación en el proyecto.

FUNCIONALIDADES

FUNCIONALIDADES:

- MANEJO DE VARIABLES
- MANEJO DE IF Y ELSE EN ESPAÑOL
- MANEJO DE CÁLCULOS BÁSICOS DE UNA CALCULADORA CON OPERACIONES BÁSICA (SUMA, RESTA, DIVISIÓN Y MULTIPLICACIÓN).
- LECTURA DE VARIABLES
- SALIDA DE VARIABLES Y DE TEXTO (EJEMPLO: SALIDA(“HOLA”), SALIDA(A_VARIABLE))
- NÚMEROS NEGATIVOS
- FUNCIÓN MAIN PRINCIPAL

Lenguaje Latino

```
grammar Expr;

// Regla de inicio
program : (principal | io3tatement | statement)* EOF ;

// ---------- GRAMÁTICA DE STATEMENTS ----------

statement
    : varDeclaracion3tmt # VarDe3tmt
    | varAsignacion3tmt  # VarAs3tmt
    | 'retorno' exp ';' # Return3tmt
    | 'salida' '(' exp ')' ';' # Output3tmt
    | 'si' '(' exp ')' '{' statement* '}' ('sino' '{'
statement* '}')? # IfElse3tmt
    | 'mientras' '(' exp ')' '{' statement* '}' #
While3tmt
    ;


varDeclaracion3tmt
    : tipo ID '=' exp ';'
    ;

varAsignacion3tmt
    : ID '=' exp ';'
    ;

io3tatement
    : 'entrada' '(' ID ')' ';' # Input3tatement
    | 'salida' '(' exp ')' ';' # Output3tatement
    ;

tipo
    : 'entero'  # Integer
    | 'flotante' # Float
```

Parser

```
    | 'cadena' # String
    ;

exp
    : ID                            # ID
    | NUM                           # Number
    | STRING                        # StringLiteral
    | '(' exp ')'                   # Parenthesis
    | '-' exp                       # Negation
    | exp '*' exp                   # Multiplication
    | exp '/' exp                   # Division
    | exp '+' exp                   # Addition
    | exp '-' exp                   # Subtraction
    | exp '&&' exp                  # LogicalAnd
    | exp '||' exp                  # LogicalOr
    | exp '==' exp                  # EqualityCheck
    | exp '!=' exp                  # InequalityCheck
    | exp '>' exp                   # GreaterThan
    | exp '<' exp                   # LessThan
    ;

principal
    : 'entero' 'principal' '(' ')' '{' mainBody '}'
    ;

mainBody
    : statement* # StatementList
    ;

// ----------- REGLAS COMUNES -----------
ID      : [a-zA-Z_][a-zA-Z0-9_]* ;
NUM     : [-]?[0-9]+('.'[0-9]+)? ;  // Acepta tanto
enteros como flotantes
STRING  : '"' (~["\r\n])* '"' ;
WS      : [ \t\r\n]+ -> skip ;
COMMENT : '//' ~[\r\n]*->skip ;
```

LEXER

Driver.h

```
#ifndef DRIVER_H
#define DRIVER_H

#include "ExprBaseVisitor.h"
#include "llvm/IR/IRBuilder.h"
#include "llvm/IR/LLVMContext.h"
#include "llvm/IR/Module.h"
#include <string>
#include <map>

using namespace antlr4;
using namespace llvm;
using namespace std;

class Driver : public ExprBaseVisitor {
private:
    LLVMContext &C;
    unique_ptr<Module> M;
    unique_ptr<IRBuilder<>> builder;
    map<string, Value *> variables; // Mapa de variables
declaradas

    IntegerType *i32Ty;

    // Crear una constante entera
    Value *ConstI32(int val) {
        return ConstantInt::get(i32Ty, val);
    }

    // Crear una constante de cadena
    Value *ConstString(const string &str) {
        return builder->CreateGlobalString(str);
    }

    // Obtener o crear la función printf
    Function *getOrCreatePrintf() {
        Function *printfFn = M->getFunction("printf");
        if (!printfFn) {
            FunctionType *printfTy = FunctionType::get(
                Type::getInt32Ty(C),
                PointerType::get(Type::getInt8Ty(C), 0),
true);
            printfFn = Function::Create(printfTy,
Function::ExternalLinkage, "printf", M.get());
        }
        return printfFn;
    }

public:
    Driver(const string &sourceFileName, LLVMContext &C)
        : C(C), M(make_unique<Module>(sourceFileName,
C)), builder(make_unique<IRBuilder<>>(C)) {
        i32Ty = IntegerType::getInt32Ty(C);
    }

    Module *getModule() {
        return M.get();
    }

    // Función principal
    virtual std::any
visitPrincipal(ExprParser::PrincipalContext *ctx)
override {
        FunctionType *mainFnTy = FunctionType::get(i32Ty,
false);
        Function *mainFn = Function::Create(mainFnTy,
Function::ExternalLinkage, "language", M.get());
        BasicBlock *entryBB = BasicBlock::Create(C,
"language.0", mainFn);
        builder->SetInsertPoint(entryBB);

        visit(ctx->mainBody());

        builder->CreateRet(ConstI32(0));
        return nullptr;
    }

    // Declaración de variables
    virtual std::any
visitVarDeclaracionStmt(ExprParser::VarDeclaracionStmtCon
text *ctx) override {
        string varName = ctx->ID()->getText();
        Value *initVal = std::any_cast<Value
*>(visit(ctx->exp()));
        AllocaInst * alloc = builder->CreateAlloca(i32Ty,
nullptr, varName.c_str());
        variables[varName] = alloc;
        builder->CreateStore(initVal,
variables[varName]);
        return nullptr;
    }

    // Asignación de variables
    virtual std::any
visitVarAsignacionStmt(ExprParser::VarAsignacionStmtConte
xt *ctx) override {
        string varName = ctx->ID()->getText();
        Value *newVal = std::any_cast<Value
*>(visit(ctx->exp()));
        if (!variables.count(varName)) {
            cerr << "Error: Variable no declarada: " <<
varName << endl;
            exit(1);
        }
        builder->CreateStore(newVal, variables[varName]);
        return nullptr;
```

Driver.h

```
    }

    // Manejo de identificadores (variables)
    virtual std::any visitID(ExprParser::IDContext *ctx)
override {
        string varName = ctx->getText();
        if (!variables.count(varName)) {
            cerr << "Error: Variable no declarada: " <<
varName << endl;
            exit(1);
        }
        AllocaInst *varPtr =
cast<AllocaInst>(variables[varName]);

        Value *varValue = builder->CreateLoad(i32Ty,
varPtr, varName.c_str());

        return varValue;
    }

    virtual std::any
visitParenthesis(ExprParser::ParenthesisContext *ctx)
override {
        return std::any_cast<Value *>(visit(ctx->exp()));
    }

    // Operaciones aritméticas
    virtual std::any
visitAddition(ExprParser::AdditionContext *ctx) override
{
        Value *lhs = std::any_cast<Value
*>(visit(ctx->exp(0)));
        Value *rhs = std::any_cast<Value
*>(visit(ctx->exp(1)));
        return builder->CreateAdd(lhs, rhs, "addtmp");
    }
```

```
    virtual std::any
visitSubtraction(ExprParser::SubtractionContext *ctx)
override {
        Value *lhs = std::any_cast<Value
*>(visit(ctx->exp(0)));
        Value *rhs = std::any_cast<Value
*>(visit(ctx->exp(1)));
        return builder->CreateSub(lhs, rhs, "subtmp");
    }

    virtual std::any
visitMultiplication(ExprParser::MultiplicationContext
*ctx) override {
        Value *lhs = std::any_cast<Value
*>(visit(ctx->exp(0)));
        Value *rhs = std::any_cast<Value
*>(visit(ctx->exp(1)));
        return builder->CreateMul(lhs, rhs, "multmp");
    }

    virtual std::any
visitDivision(ExprParser::DivisionContext *ctx) override
{
        Value *lhs = std::any_cast<Value
*>(visit(ctx->exp(0)));
        Value *rhs = std::any_cast<Value
*>(visit(ctx->exp(1)));
        return builder->CreateSDiv(lhs, rhs, "divtmp");
    }

    // Salida de valores
    virtual std::any
visitOutputStmt(ExprParser::OutputStmtContext *ctx)
override {
        Value *output = std::any_cast<Value
*>(visit(ctx->exp()));
```

```
        Function *printfFn = getOrCreatePrintf();
        Value *formatStr;

        if (output->getType()->isIntegerTy()) {
            formatStr = ConstString("%d\n");
        } else if (output->getType()->isPointerTy()) {
            formatStr = ConstString("%s\n");
        } else {
            cerr << "Error: Tipo no soportado en salida."
<< endl;
            exit(1);
        }

        builder->CreateCall(printfFn, {formatStr,
output});
        return nullptr;
    }

// Comparaciones lógicas
    virtual std::any
visitGreaterThan(ExprParser::GreaterThanContext *ctx)
override {
        Value *lhs = std::any_cast<Value
*>(visit(ctx->exp(0)));
        Value *rhs = std::any_cast<Value
*>(visit(ctx->exp(1)));
        return builder->CreateICmpSGT(lhs, rhs, "gttmp");
// Mayor que
    }

    virtual std::any
visitLessThan(ExprParser::LessThanContext *ctx) override
{
        Value *lhs = std::any_cast<Value
*>(visit(ctx->exp(0)));
        Value *rhs = std::any_cast<Value
*>(visit(ctx->exp(1)));
```

Driver.h

```
        return builder->CreateICmpSLT(lhs, rhs, "lttmp");
// Menor que
    }

    virtual std::any
visitEqualityCheck(ExprParser::EqualityCheckContext *ctx)
override {
        Value *lhs = std::any_cast<Value
*>(visit(ctx->exp(0)));
        Value *rhs = std::any_cast<Value
*>(visit(ctx->exp(1)));
        return builder->CreateICmpEQ(lhs, rhs, "eqtmp");
// Igualdad
    }

    virtual std::any
visitInequalityCheck(ExprParser::InequalityCheckContext
*ctx) override {
        Value *lhs = std::any_cast<Value
*>(visit(ctx->exp(0)));
        Value *rhs = std::any_cast<Value
*>(visit(ctx->exp(1)));
        return builder->CreateICmpNE(lhs, rhs, "netmp");
// Desigualdad
    }
// Declaración de condiciones (if-else)
    virtual std::any
visitIfElseStmt(ExprParser::IfElseStmtContext *ctx)
override {
        Value *cond = std::any_cast<Value
*>(visit(ctx->exp()));
        cond = builder->CreateICmpNE(cond, ConstI32(0),
"ifcond");

        Function *fn =
builder->GetInsertBlock()->getParent();
        BasicBlock *thenBB = BasicBlock::Create(C,
"then", fn);
        BasicBlock *elseBB = BasicBlock::Create(C,
"else");
        BasicBlock *mergeBB = BasicBlock::Create(C,
"merge");

        builder->CreateCondBr(cond, thenBB, elseBB);

        builder->SetInsertPoint(thenBB);
        visit(ctx->statement(0));
        builder->CreateBr(mergeBB);

        elseBB->insertInto(fn);
        builder->SetInsertPoint(elseBB);

        if (ctx->statement(1)) {
            visit(ctx->statement(1));
        }
        builder->CreateBr(mergeBB);

        mergeBB->insertInto(fn);
        builder->SetInsertPoint(mergeBB);
        return nullptr;
    }

    // Manejo de números
    virtual std::any
visitNumber(ExprParser::NumberContext *ctx) override {
        return ConstI32(stoi(ctx->getText()));
    }

    // Manejo de cadenas literales
    virtual std::any
visitStringLiteral(ExprParser::StringLiteralContext *ctx)
override {
        string str = ctx->STRING()->getText();
        str = str.substr(1, str.length() - 2); // Remover
comillas
        return ConstString(str);
    }
};

#endif // DRIVER_H
```

Main.cpp

```cpp
#include <iostream>
#include <fstream>
#include <memory>

#include "Driver.h"
#include "ExprLexer.h"
#include "ExprParser.h"
#include "llvm/IR/LLVMContext.h"
#include "llvm/IR/Verifier.h"
#include "llvm/Support/raw_ostream.h"

using namespace antlr4;
using namespace llvm;
using namespace std;

void addMainFunction(Module *M) {
    LLVMContext &C = M->getContext();
    FunctionType *mainTy =
FunctionType::get(Type::getInt32Ty(C), {}, false);
    Function *mainFn = Function::Create(mainTy,
Function::ExternalLinkage, "main", M);
    BasicBlock *mainBB = BasicBlock::Create(C, "entry",
mainFn);
    IRBuilder<> builder(mainBB);
    // Llamar a la función "language"
    Function *languageFn = M->getFunction("language");
    if (!languageFn) {
        errs() << "Error: 'language' function not found
in module.\n";
        return;
    }
    builder.CreateCall(languageFn);

builder.CreateRet(ConstantInt::get(Type::getInt32Ty(C),
0));
}

int main(int argc, const char *argv[]) {
    // Leer archivo de entrada o stdin
    ifstream inputFile;
    string sourceFileName = "stdin";
    if (argc > 1) {
        sourceFileName = argv[1];
        inputFile.open(sourceFileName);
        if (!inputFile.is_open()) {
            cerr << "Error: Cannot open file " <<
sourceFileName << "\n";
            return 1;
        }
    }
    istream &inputStream = (argc > 1) ? inputFile : cin;

    ANTLRInputStream input(inputStream);
    ExprLexer lexer(&input);
    CommonTokenStream tokens(&lexer);
    ExprParser parser(&tokens);

    // Contexto LLVM
    LLVMContext context;
    Driver driver(sourceFileName, context);
    // Generar IR
    try {
        unique_ptr<Module> module(driver.getModule());
        driver.visit(parser.program());

        addMainFunction(module.get());

        // Imprimir IR generado
        module->print(outs(), nullptr);

        // Verificar módulo
        if (verifyModule(*module, &errs())) {
            errs() << "Error: module verification
failed.\n";
            return 1;
        }
    } catch (const exception &e) {
        cerr << "Error during code generation: " <<
e.what() << "\n";
        return 1;
    }

    return 0;
}
```

```
entero principal() {
    entero a = 9;
    entero b = 11 + (5 * 2 / (2*1));

    a = 5 + b;
    salida("a es igual: ");
    salida(a);
```

```
    salida("b es igual: ");
    salida(b -  -1);
    si (10 > 1){
        salida("10 es mayor a 1");
    } sino {
        salida("10 no es mayor a 1");
    }
}
```

Ejemplo

Generación del IR code (ej2.ll):

- Se genera el código con el comando: `build/prog ej2.expr > ej2.ll`
- Y luego este comando: `lli ej2.ll # para poder ver los resultados abajo`

```
; ModuleID = 'ej2.expr'
source_filename = "ej2.expr"

@0 = private unnamed_addr constant [13 x i8] c"a es igual: \00", align 1
@1 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@2 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
@3 = private unnamed_addr constant [13 x i8] c"b es igual: \00", align 1
@4 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@5 = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
@6 = private unnamed_addr constant [16 x i8] c"10 es mayor a 1\00", align 1
@7 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1
@8 = private unnamed_addr constant [19 x i8] c"10 no es mayor a 1\00", align 1
@9 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1

define i32 @language() {
language.0:
  %a = alloca i32, align 4
  store i32 9, ptr %a, align 4
  %b = alloca i32, align 4
  store i32 16, ptr %b, align 4
  %b1 = load i32, ptr %b, align 4
  %addtmp = add i32 5, %b1
  store i32 %addtmp, ptr %a, align 4
  %0 = call i32 (ptr, ...) @printf(ptr @1, ptr @0)
  %a2 = load i32, ptr %a, align 4
  %1 = call i32 (ptr, ...) @printf(ptr @2, i32 %a2)
  %2 = call i32 (ptr, ...) @printf(ptr @4, ptr @3)
  %b3 = load i32, ptr %b, align 4
  %subtmp = sub i32 %b3, -1
  %3 = call i32 (ptr, ...) @printf(ptr @5, i32 %subtmp)
  br i1 true, label %then, label %else

then:                                             ; preds = %language.0
  %4 = call i32 (ptr, ...) @printf(ptr @7, ptr @6)
  br label %merge

else:                                             ; preds = %language.0
  %5 = call i32 (ptr, ...) @printf(ptr @9, ptr @8)
  br label %merge

merge:                                            ; preds = %else, %then
  ret i32 0
}

declare i32 @printf(ptr, ...)

define i32 @main() {
entry:
  %0 = call i32 @language()
  ret i32 0
}
```

Ejemplo con interpretador:

```
[root@cefa7bdcd1f0 prueba2]# lli ej2.ll
a es igual:
21
b es igual:
17
10 es mayor a 1
```

CONCLUSIONES

Aprendido ANTLR4 para la creacion de un proyecto en c++ que puede interactuar y reconocer expresiones de este famoso lenguaje.

Uso de llvm para construir un generador de codigo IR

Mas modularizacion en la gramatica y evitar repeticiones innecesarias

Agregar mas lenguaje de español nativo.

Deteccion de errores dentro de el driver, detectar errors de usuario

GRACIAS