Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Escuela de Ciencias y Sistemas

Organización de Lenguajes y Compiladores 2

Catedrático: M.Sc. Luis Espino

Auxiliares: Juan Carlos Maeda, Pavel Vásquez

**MinorC**

**Manual Técnico**

Kairi Itzmar Alexander Franco Medina

201222591

3 de julio de 2020

ÍNDICE

[INTRODUCCIÓN 2](#_Toc44622951)

[Objetivo general 2](#_Toc44622952)

[Objetivo específico 2](#_Toc44622953)

[RESPONSABLES 2](#_Toc44622954)

[CONTENIDO TÉCNICO 3](#_Toc44622955)

[Módulos 3](#_Toc44622956)

[Módulo Error 3](#_Toc44622957)

[Módulo Expresión 3](#_Toc44622958)

[Módulo Instruction 3](#_Toc44622959)

[Módulo SymsbolTable 4](#_Toc44622960)

[Módulo Translator 4](#_Toc44622961)

# INTRODUCCIÓN

MinorC es un subconjunto del lenguaje C, creado con el fin de poner en práctica los conceptos del proceso de compilación.

El siguiente manual le será útil principalmente a los programadores para poder darle mantenimiento a la aplicación o revisarla en caso de alguna falla.

## Objetivo general

Analizar y sintetizar un compilador para un lenguaje de programación MinorC, utilizando PLY.

## Objetivo específico

Construir un analizador léxico y sintáctico, implementando la traducción dirigida por la sintaxis y reglas semánticas con atributos sintetizados, que permita la traducción de un código de alto nivel (MinorC) a un código de tres direcciones con la sintaxis Augus.

# RESPONSABLES

Asignación del proyecto: M.Sc. Luis Espino

Programador: Kairi Itzmar Alexander Franco Medina

Encargados de revisión: Juan Carlos Maeda, Pavel Vásquez

# CONTENIDO TÉCNICO

A continuación, se presenta la explicación de los módulos que posee el traductor de MinorC, junto con sus clases y métodos.

### Módulos

### Módulo Error

* + Clase Error: almacena errores
  + Clase Error List: almacena listas de errores
    - Método And(self, error): agrega un objeto a la clase errr list
    - Método Clear(self): quita un objeto de la clase error list
    - Método Print(self): imprime un objeto de la clase error list

### Módulo Expresión

* Clase ArithmeticOperation(Enum): maneja operaciones aritméticas
* Clase RelationalOperation(Enum): maneja operaciones relacionales
* Clase NumericExpression: almacena expresiones numéricas
* Clase BinaryExpression(NumericExpression): almacena expresiones numéricas binarias
* Clase NegativeExpresion(NumericExpression): almacena expresiones numéricas negativas
* Clase Integer(NumericExpression): almacena expresiones numéricas con enteros
* Clase Float(NumericExpression): almacena expresiones numéricas con decimales
* Clase Strig(NumericExpression): almacena expresiones numéricas con cadenas
* Clase Character(NumericExpression): almacena expresiones numéricas con caracteres
* Clase Variable(NumericExpression): almacena expresiones numéricas con variables
* Clase Call(NumericExpression): llama a una función

### Módulo Instruction

* Clase Printf(Instruction): instrucción de imprimir
* Clase Scanf(Instruction): instrucción de escanear
* Clase Definition(Instruction): instrucción definir
* Clase Declaration(Instruction): instrucción declarar
* Clase Assignation(Instruction): instrucción asignar
* Clase Parameter(Instruction): instrucción parámetro
* Clase Function(Instruction): instrucción función
* Clase Iff(Instruction): instrucción si
* Clase Ifelse(Instruction): instrucción si además
* Clase Whilee(Instruction): instrucción mientras
* Clase Dowhile: instrucción hacer mientras
* Clase Forr(Instruction): instrucción por
* Clase Case(Instruction): instrucción caso
* Clase Switch(Instruction): instrucción cambiar
* Clase Enum(Instruction): instrucción enumerar
* Clase Struct(Instruction): instrucción estructurar
* Clase Union(Instruction): instrucción unir
* Clase Caste(Instruction): instrucción casto de tipos de dato
* Clase Return(): instrucción regresar
* Clase Breack(Instruction): instrucción salir del ciclo actual
* Clase Continue(Instruction): instrucción continuar
* Clase Goto(Instruction): instrucción ir a
* Clase Sizeof(Instruction): instrucción tamaño en bytes de tipo de dato primitivo
* Clase Label(Instruction): almacena etiquetas
* Clase PostOperation(Instruction): maneja post operaciones
* Clase PreOperation(Instruction): maneja pre operaciones
* Clase Unary Operation(Instruction): maneja operaciones unitarias
* Clase Arithmetic Operation(Instruction): maneja operaciones aritméticas
* Clase Shift Operation(Instruction): maneja operaciones desplazadas
* Clase Relational Operation(Instruction): maneja operaciones relacionales
* Clase Logical Operation(Instruction): maneja operaciones lógicas
* Clase Bitwise Operation(Instruction): maneja operaciones bit a bit
* Clase Ternary Operation(Instruction): maneja operaciones ternarias
* Clase Assign Operation(Instruction): maneja operaciones asignadas
* Clase ExpresionList(): almacena listas de expresiones

### Módulo SymsbolTable

* Clase Table: almacena tablas
  + Método IsSymbolInTable(self id): indica si una variable está en la tabla
  + Método Add(self, symbol): agrega un objeto a la clase table
  + Método Get(self, id): obtiene un objeto de la clase table
  + Método Update(self, id, newValue): actualiza un objeto de la clase table
  + Método Remove(self,id): remueve un objeto de la clase table
  + Método Print(self): imprime un objeto de la clase table
* Clase Symbol: almacena simbolos
  + Método Print(self): imprime un objeto de la clase symbol

### Módulo Translator

* Método Create\_l(): crea una variable temporal l
* Método Create\_t(): crea una variable temporal t
* Método Create\_a(): crea una variable temporal a
* Método Create\_v(): crea una variable temporal v
* Método Translate(instrucctions): instrucción de traducir
* Método TranslateBlock(b): traduce un objeto de la clase block
* Método TranslateFunction(f): traduce un objeto de la clase function
* Método TranslateStruct(s): traduce un objeto de la clase struct
* Método TranslateUnion(u): traduce un objeto de la clase union
* Método TranslateEnum(e): traduce un objeto de la clase enum
* Método TranslateVarDec(v): traduce un objeto de la clase vardec
* Método TranslatePrintf(i): traduce un objeto de la clase printf
* Método Weave(list1, list2): entrelaza las listas 1 y 2
* Método TranslateScanf(i): traduce un objeto de la clase scanif
* Método TranslateGoto(i): traduce un objeto de la clase goto
* Método TranslateLabel(i): traduce un objeto de la clase label
* Método TranslateBreak(i): traduce un objeto de la clase break
* Método TranslateContinue(i): traduce un objeto de la clase continue
* Método TranslateReturn(i): traduce un objeto de la clase return
* Método TranslateVariable(i): traduce un objeto de la clase variable
* Método TranslateIf(i): traduce un objeto de la clase if
* Método TranslateIfElse(i): traduce un objeto de la clase ifelse
* Método TranslateWhile(i): traduce un objeto de la clase while
* Método TranslateDoWhile(i): traduce un objeto de la clase dowile
* Método TranslateFor(i): traduce un objeto de la clase for
* Método TranslateSwitch(i): traduce un objeto de la clase switch
* Método TranslateExpressionList(e): traduce un objeto de la clase expressionlist
* Método TranslateAssignOperation(exp): traduce un objeto de la clase assignoperation
* Método TranslateTernaryOperation(exp): traduce un objeto de la clase ternaryoperation
* Método TranslateLogicalOperation(exp): traduce un objeto de la clase logicaloperation
* Método TranslateBitwiseOperation(exp): traduce un objeto de la clase bitwiseoperation
* Método TranslateRelationalOperation(exp): traduce un objeto de la clase relationaloperation
* Método TranslateShiftOperation(exp): traduce un objeto de la clase shiftoperation
* Método TranslateArithmeticalOperation(exp): traduce un objeto de la clase arithmeticaloperation
* Método TranslateUnaryOperation(exp): traduce un objeto de la clase unaryoperation
* Método TranslatePreOperation(exp): traduce un objeto de la clase preoperation
* Método TranslatePostOperation(exp): traduce un objeto de la clase postoperation
* Método TranslateSizeof(exp): traduce un objeto de la clase sizeof
* Método TranslateCast(exp): traduce un objeto de la clase cast
* Método TranslateInteger(exp): traduce un objeto de la clase integer
* Método TranslateString(exp): traduce un objeto de la clase string
* Método TranslateCharacter(exp): traduce un objeto de la clase character
* Método TranslateFloat(exp): traduce un objeto de la clase float
* Método TranslateCall(exp): traduce un objeto de la clase call
* Método TranslateFunctionCaller(f, parameters, callLabel, returnLabel): traduce un objeto de la clase functioncaller
* Método ReplaceVariables(funtion, parameters): remplaza un objeto de la clase variables
* Método IsPrimitive(i): indica si una variable es de un tipo primitivo
* Método SemanticErrorReport(): reporta si existe un error semántico

### Gramática

decls -> decl decls

|

decl -> func decl

| func proto

| struct decl

| union decl

| enum decl

| var decl

| typedef decl

func proto -> func NAME L\_PAR types R\_PAR SEMICOLON

| func NAME L\_PAR params R\_PAR SEMICOLON

| func NAME L\_PAR R\_PAR SEMICOLON

func decl -> func NAME L\_PAR params R\_PAR block

| func NAME L\_PAR NAME list R\_PAR struct def block

| func NAME L\_PAR R\_PAR block

params -> param COMMA params

| param

param -> const type NAME

| type NAME

types -> type COMMA types

| type

NAME list -> NAME COMMA NAME list

| NAME

func NAME -> type NAME

| NAME

typedef decl -> typedef type NAME SEMICOLON

struct decl -> struct NAME L\_BRACKET struct def R\_BRACKET SEMICOLON

union decl -> union NAME L\_BRACKET struct def R\_BRACKET SEMICOLON

struct def -> var decl struct def

| var decl

var decl -> mod type var var list SEMICOLON

| type var var list SEMICOLON

| mod var var list SEMICOLON

var -> NAME array

| NAME array ASSIGN op if

array -> L\_BRACKET expr R\_BRACKET

| L\_BRACKET R\_BRACKET

|

var list -> COMMA var item var list

|

var item -> pointers var

mod -> extern

| static

| register

| auto

| volatile

| const

enum decl -> enum NAME L\_BRACKET enum def R\_BRACKET SEMICOLON

enum def -> enum val COMMA enum def

| enum val

enum val -> NAME

| NAME ASSIGN octliteral

| NAME ASSIGN hexliteral

| NAME ASSIGN decliteral

type -> base pointers

base -> sign scalar

| struct NAME

| struct L\_BRACKET struct def R\_BRACKET

| union NAME

| union L\_BRACKET struct def R\_BRACKET

| enum NAME

sign -> signed

| unsigned

|

scalar -> char

| int

| short

| float

| double

| void

pointers -> MULTIPLY pointers

|

stm -> var decl

| NAME COLON

| if L\_PAR expr R\_PAR stm

| if L\_PAR expr R\_PAR then stm else stm

| while L\_PAR expr R\_PAR stm

| for L\_PAR arg SEMICOLON arg SEMICOLON arg R\_PAR stm

| normal stm

| printf (STRING, params)

| scanf (STRING , param)

then stm -> if L\_PAR expr R\_PAR then stm else then stm

| while L\_PAR expr R\_PAR then stm

| for L\_PAR arg SEMICOLON arg SEMICOLON arg R\_PAR then stm

| normal stm

normal stm -> do stm while L\_PAR expr R\_PAR

| switch L\_PAR expr R\_PAR L\_BRACKET case stms R\_BRACKET

| block

| expr SEMICOLON

| goto NAME SEMICOLON

| break SEMICOLON

| continue SEMICOLON

| return expr SEMICOLON

| SEMICOLON

arg -> expr

|

case stms -> case value COLON stm list case stms

| default COLON stm list

|

block -> L\_BRACKET stm list R\_BRACKET

stm list -> stm stm list

|

expr -> expr COMMA op assign

| op assign

op assign -> op if ASSIGN op assign

| op if APLUS op assign

| op if AMINUS op assign

| op if AMULTIPLY op assign

| op if ADIVIDE op assign

| op if AREMAINDER op assign

| op if AXOR op assign

| op if AAND op assign

| op if AOR op assign

| op if AR\_SHIFT op assign

| op if AL\_SHIFT op assign

| op if

op if -> op or QUESTION op if COLON op if

| op or

op or -> op or OR op and

| op and

op and -> op and AND op binor

| op binor

op binor -> op binor OR\_B op binxor

| op binxor

op binxor -> op binxor XOR\_B op binand

| op binand

op binand -> op binand AND\_B op equate

| op equate

op equate -> op equate EQUAL op compare

| op equate NOT\_EQUAL op compare

| op compare

op compare -> op compare LESS op shift

| op compare GREATER op shift

| op compare LESS\_EQUAL op shift

| op compare GREATER\_EQUAL op shift

| op shift

op shift -> op shift L\_SHIFT op add

| op shift R\_SHIFT op add

| op add

op add -> op add PLUS op mult

| op add MINUS op mult

| op mult

op mult -> op mult MULTIPLY op unary

| op mult DIVIDE op unary

| op mult REMAINDER op unary

| op unary

op unary -> NOT op unary

| NOT\_B op unary

| MINUS op unary

| MULTIPLY op unary

| AND\_B op unary

| PLUS PLUS op unary

| MINUS MINUS op unary

| op pointer PLUSPLUS

| op pointer MINUS MINUS

| L\_PAR type R\_PAR op unary

| sizeof L\_PAR type R\_PAR

| sizeof L\_PAR NAME pointers R\_PAR

| op pointer

op pointer -> op pointer DOT value

| op pointer ACCESS value

| op pointer L\_BRACKET expr R\_BRACKET

| value

value -> decliteral

| stringliteral

| charliteral

| floatliteral

| NAME L\_PAR expr R\_PAR

| NAME L\_PAR R\_PAR

| NAME

| L\_PAR expr R\_PAR