

MINOR C

MANUAL TECNICO



Universidad de San Carlos de Guatemala
JuanPablo Osuna de Leon
201503911



Requerimientos del Sistema:

- *Windows 10 x64 bits*
- *Python 3.8*
- *Graphviz*
- *ReportLab*
- *Ó Instalador proporcionado en la documentacion*

Creacion de Interfaz grafica (PyQt5):

Para la creación de la interfaz grafica se crea una clase llamada **UI_Augus**, esta será la encargada de todo el manejo de la interfaz grafica como las acciones, su método principal se llama **setuUI**, en el que se creara la ventana y todos los objetos que la ventana podrá mostrar.

```
class Ui_Augus(object):  
  
    def setupUi(self, Augus):  
        Augus.setObjectName("Augus")  
        Augus.resize(980, 816)  
        Augus.setStyleSheet('MainWindow{background-color: yellow; border: 1px solid black;}'  
        self.centralwidget = QtWidgets.QWidget(Augus)  
        self.centralwidget.setObjectName("centralwidget")  
        self.centralwidget.setStyleSheet("background-color: rgb(33,33,33);")  
        self.tabWidget = QtWidgets.QTabWidget(self.centralwidget)  
        self.tabWidget.setGeometry(QtCore.QRect(20, 20, 471, 541))  
        self.tabWidget.setObjectName("tabWidget")  
  
        self.textEditOuput = QtWidgets.QTextEdit(self.centralwidget)  
        self.textEditOuput.setGeometry(QtCore.QRect(20, 570, 950, 200))  
        self.textEditOuput.setObjectName("textEditOuput")  
> self.textEditOuput.setStyleSheet('background-color: rgb(33, 33, 33); ...  
        self.textEditOuput.setPlainText("OUTPUT:\n")  
  
        self.textEditConsole = QtWidgets.QTextEdit(self.centralwidget)  
        self.textEditConsole.setGeometry(QtCore.QRect(510, 20, 461, 541))  
        self.textEditConsole.setObjectName("textEditConsole")
```

El método nombrado **“retranslateUI”** será el encargado de asociar cada componente a su respectivo componente padre, además de setear el nombre visual al componente.

```
def retranslateUi(self, Augus):
    _translate = QtCore.QCoreApplication.translate
    Augus.setWindowTitle(_translate("Augus", "MinorC"))
    self.menuArchivo.setTitle(_translate("Augus", "Archivo"))
    self.menuEditar.setTitle(_translate("Augus", "Editar"))
    self.menuEjecutar.setTitle(_translate("Augus", "Ejecutar"))
    self.menuOpciones.setTitle(_translate("Augus", "Opciones"))
    self.menuReportes.setTitle(_translate("Augus", "Reportes"))
    self.menuAyuda.setTitle(_translate("Augus", "Ayuda"))
    self.actionNuevo.setText(_translate("Augus", "Nuevo"))
    self.actionReporteLexico.setText(_translate("Augus", "Reporte Lexico"))
    self.actionReporteSintactico.setText(_translate("Augus", "Reporte Sintactico"))
    self.actionReporteSemantico.setText(_translate("Augus", "Reporte Semantico"))
    self.actionReporteTS.setText(_translate("Augus", "Reporte Tabla de Simbolos"))
    self.actionReporteAST.setText(_translate("Augus", "Reporte AST ascendente"))
    self.actionReporteGramatical.setText(_translate("Augus", "Reporte Gramatical"))
```

Acciones de los menús:

Los menus tendrá asociado un método para su actuar, esto será por medio de un `actionListener` en este caso `“Triggered.connect”` en el cual se le asociará un método que contendrá las acciones de este.

```
#actions
self.actionNuevo.triggered.connect(lambda : self.fn_Nuevo())
self.actionAbrir.triggered.connect(lambda : self.fn_Abrir())
self.actionAscendente.triggered.connect(lambda : self.fn_Ejecutar_Ascendente())
self.actionDescendente.triggered.connect(lambda : self.fn_Ejecutar_Descendente())
self.actionDebuguer.triggered.connect(lambda : self.fn_Ejecutar_Debuguer())
self.actionGuardar.triggered.connect(lambda : self.fn_Guardar())
self.actionGuardar_Como.triggered.connect(lambda : self.fn_Guardar_Como())
self.actionCerrar.triggered.connect(lambda : self.fn_Cerrar())
self.actionSalir.triggered.connect(lambda : self.fn_Salir())
self.actionReporteLexico.triggered.connect(lambda : self.fn_repLexico())
self.actionReporteSintactico.triggered.connect(lambda : self.fn_repSintactico())
self.actionReporteSemantico.triggered.connect(lambda : self.fn_repSemantico())
self.actionReporteGramatical.triggered.connect(lambda : self.fn_repGramatical())
```

El método asociado contendrá todas las acciones del menú seleccionado.

```
def fn_Next(self): ...

def fn_Ejecutar_Debuguer(self): ...

def fn_Ejecutar_Ascendente(self):
    #try:
        #inicializacion
        traduction.augusTxt = 'main: \n'
        traduction.augusTxt += F'PL_: \n'    #parche
        traduction.augusTxt += F'PL_: \n'
        traduction.augusTxtCalls = ''
        traduction.augusTxtAuxVar = ''
        traduction.augusTxtAuxJUMPS = 'manejador: \n'
        #traduction.augusTxt = ''
        traduction.contadorT = 0
```

Llamada a análisis:

La llamada al análisis se hará por medio de la acción del menú Ejecutar, dentro del método se creará una instancia de la clase “grammarMinorC.py” en cual recibirá como parámetro una cadena, esta cadena será recuperada del editor de texto y será enviada como parámetro, la clase gramática retornara una lista de instrucciones o de errores léxicos y sintácticos dependiendo de cual fue el status del análisis.

```
content = self.tabWidget.currentWidget().findChild(QtWidgets.QTextEdit, "textEdit").toPlainText()
content += '\n'
content.encode('utf-8')
result = grammarAscMinorC.parse(content)
global instructionList
```


Creacion de Gramatica (class grammarMinorC.py):

Para la creación de la clase inicialmente debemos definir un método de llamada el cual se llamará parser, este será el encargado de crear una nueva instancia de las clases “Lexer” y de la clase “Parser”.

```
def parse(input):
    global input_, sintacticErroList, LexicalErrosList

    sintacticErroList[:] = []
    LexicalErrosList[:] = []

    input_ = input
    lexer = lex.lex()
    parser = yacc.yacc()
    instructions = parser.parse(input)
    print(str(instructions))
    lexer.lineno = 1
    parser.restart()
    if len(LexicalErrosList) > 0 or len(sintacticErroList) > 0: ...
    return instructions
```

El método parser será el encargado de reportar todos los errores sintácticos y léxicos que se encuentren en el análisis, para posteriormente retornar o la lista de errores o la lista de instrucciones, el cual contendrá el árbol de sintaxis abstracta.

Creación de la Gramática:

Esta se detallará en el documento de gramática, en esta sección solo se mostrarán las partes importantes para la creación y generación del árbol de sintaxis abstracta.

Para ello aremos uso de 2 clases principales las cuales serán Instrucciones y Expresiones, se deberán importar a la clase gramática.

```
from expresionsMinorC import *
from instructionsMinorC import *
```

Clase Expresión:

Esta clase abstracta contendrá objetos de tipo expresión, los cuales serán utilizados para encapsular los datos obtenidos en la gramática.

```
class Expretion:
    ''' this class represent an expresion'''

#####----- declaration Section
class DeclarationExp:
    ''' this class represent an numeric expresion'''

class SingleDeclaration(DeclarationExp):
    def __init__(self, id, val, line, column):
        self.id = id
        self.val = val
        self.line = line
        self.column = column

class Declaration_Array(DeclarationExp):
    def __init__(self, id, expression, line, column):
        self.id = id
        self.expression = expression
        self.line = line
        self.column = column

#####----- Numeric Section
class NumericExpression:
    ''' this class represent an numeric expresion'''

class BinaryExpression(NumericExpression):
    def __init__(self, op1, op2, operator, line, column):
        self.op1 = op1
        self.op2 = op2
        self.operator = operator
        self.line = line
```

Se hace uso de esta instancia por medio de el reconocimiento de la gramática, por ejemplo, en la siguiente imagen se hace uso de la expresión BinaryExpresion, el cual hace referencia a una operación binaria en nuestra gramática.

```
f p_expresion(t):
    '''EXPRESION : EXPRESION MAS EXPRESION
    | EXPRESION MENOS EXPRESION
    | EXPRESION POR EXPRESION
    | EXPRESION DIV EXPRESION
    | EXPRESION MODULO EXPRESION
    | EXPRESION IGUALQUE EXPRESION
    | EXPRESION DIFERENTE EXPRESION
    | EXPRESION MENORQUE EXPRESION
    | EXPRESION MAYORQUE EXPRESION
    | EXPRESION MENORIGUAL EXPRESION
    | EXPRESION MAYORIGUAL EXPRESION
    | EXPRESION NOTBIT EXPRESION
    | EXPRESION ANDBIT EXPRESION
    | EXPRESION XORBIT EXPRESION
    | EXPRESION ORBIT EXPRESION
    | EXPRESION OR EXPRESION
    | EXPRESION AND EXPRESION
    | EXPRESION SHIFTIZQ EXPRESION
    | EXPRESION SHIFTER EXPRESION
    | PARIZQ EXPRESION PARDER
    | C_INT EXPRESION
    | C_FLOAT EXPRESION
    | C_CHAR EXPRESION
    | SIZEOF PARIZQ EXPRESION PARDER
    | NOTLOGICA EXPRESION
    | MENOS EXPRESION %prec Umenos
    | NOTBIT EXPRESION
    | ANDBIT EXPRESION %prec Uandbit
    | LLAMADA_FUNCION
    | SCANF PARIZQ PARDER
    | INCRE_DECRE'''
    # EXPRESION TERNARIO EXPRESION DOSPUNTOS EXPRESION'''

global grammarList
if len(t) == 4:
    #aritméticos
    if t[2] == '+': t[0] = BinaryExpression(t[1],t[3],Aritmetics.MAS, t.lineno(2), t.lexpos(2))
```

Este valor se sintetiza por medio de la posición t[0],

Clase Instrucciones:

Esta clase será la encargada de encapsular instrucciones dentro de nuestra gramática, entiéndase instrucciones como “printf”, “declaración”, etc. Son instrucciones que nuestro compilador deberá realizar.

```
class Instruction:
    '''this is an abstract class'''

class Printf_(Instruction):
    '''print statment, recieve a string'''

    def __init__(self, expressions, line, column):
        self.expressions = expressions
        self.line = line
        self.column = column

class Declaration(Instruction): ...

class FunctionDeclaration(Instruction): ...

class Unset(Instruction): ...

class Exit(Instruction): ...

class If(Instruction): ...

class IfElse(Instruction): ...

class Else(Instruction): ...
```

Cada instrucción recibirá los parámetros mínimos necesarios para su funcionamiento, por ejemplo, en el caso del “printf” este recibirá solo una expresión, que esta a su vez pudiera ser una Expresión Binaria como se definió y sintetizo anteriormente.

Estas instrucciones se adjuntarán en una lista que simulara ser nuestro Árbol de sintaxis Abstracto.

```
def p_listaInstrucciones(t):
    '''INSTRUCCIONES_GLOBALES : INSTRUCCIONES_GLOBALES DECLARACION_GLOBAL
    | DECLARACION_GLOBAL'''


    global grammarList
    if len(t) == 3:
        t[1].append(t[2])
        t[0] = t[1]
        grammarList.append(g.nodeGramatical('INSTRUCCIONES_GLOBALES -> INSTI
    else:
        t[0] = [t[1]]
        grammarList.append(g.nodeGramatical('INSTRUCCIONES_GLOBALES -> DECLI
```


Una vez reconocidas todas las instrucciones y encapsuladas, esta se retornará a la clase guiMynorC.py para que ella haga uso de estas y continúe con el siguiente paso del ciclo de vida de la ejecución.

```
def parse(input):
    global input_, sintacticErroList, LexicalErrosList

    sintacticErroList[:] = []
    LexicalErrosList[:] = []


    input_ = input
    lexer = lex.lex()
    parser = yacc.yacc()
    instructions = parser.parse(input)
    print(str(instructions))
    lexer.lineno = 1
    parser.restart()
    if len(LexicalErrosList) > 0 or len(sintacticErroList) > 0: ...
    return instructions
```



Una vez terminado el análisis (1), se procede al siguiente paso que se trata de la traducción de las instrucciones a código Intermedio para que pueda ser interpretado por Augus.

Para ello debemos importar la clase Traduction.py

```
import execute
import traduction
import grammarAscMinorC
import grammar
import grammarDesc
import re
import reportGenerator as rg
import SymbolTable as TS
```



```
content = self.tabWidget.currentWidget().findChild(QtWidgets.QTextEdit,"textEdit").toPlainText
content += '\n'
content.encode('utf-8')
result = grammarAscMinorC.parse(content) 1
global instructionsList
instructionsList = result[:]

# si exists errores lexicos o sintacticos traera una lista vacia
global data, dataS, dataSema, dataTs
if len(result) == 0: ...
else:
    Augus.setStyleSheet('QMainWindow{background-color: green; border: 1px solid black;}')
    #si no existen errores mandamos a traducir el ast y pintamos la barra de color verde
    augus = traduction.execute(result, self.textEditConsole) 2
    self.textEditConsole.setPlainText("CODIGO 3D:\n")
    self.textEditConsole.append(augus)
    print(f"texto august:\n{augus}")
    print("traduccion completa")
```

Traduccion:

Esta clase será la encargada de realizar una traducción a un lenguaje que pueda ser interpretado por nuestro interprete "Augus".

Para ello la clase "guiMinorC.py" mandara como parámetro al método principal "execute".

Esta clase hará las inicializaciones necesarias para la clase, para posteriormente llamar al método process para ejecutar las instrucciones encapsuladas en la gramática.

```
def execute(input, textEdit):
    global tsGeneral, ambitoGeneral, ambitoGeneral, tableGlobal, contadorParams,
    tsGeneral = mcTS.SymbolTable()
    tableGlobal.clear()
    tsFunciones = {}
    tsStruct = {}
    tsFunciones = fTS.functionsTable()      #tabla de funciones
    tsStruct = sTS.structTable()            #tabla de struct
    contadorParams = 0
    contadorT = 0
    contadorEtiquetas = 0
    contadorEtiquetasAux = 0
    contadorCalls = 0
    arrayTables.append(tableGlobal)

    augusTxtAuxJUMPS = ''
    augusTxtAuxJUMPS += 'manejador:\n'
    augusTxtAuxJUMPS += '$s5 = $s0[$ra];\n'
    augusTxtAuxJUMPS += '$ra = $ra - 1;\n'
    process(input, tableGlobal)
    print(f"tsGlobal: {str(tableGlobal)}")
    print("ts General: ")
    for i in tsGeneral.symbols: ...

    return augusTxt
```

Método Process:

Este método será el encargado de analizar las instrucciones encapsuladas en nuestro análisis.

Para ello se harán 2 pasadas, la 1ra pasada constara en el reconocimiento de declaraciones globales, métodos y declaración de struct's.

```
def process(instructions,ts):
    #try:
        global augusTxt, augusTxtCalls, augusTxtAuxVar, contadorParams, augusTxtAuxJUMPS
        #PRIMERA PASADA
        #capturando las funciones, metodos y variables globales y struct
        contadorParams = 0
        i = 0
        while i < len(instructions):
            #isinstance verificar tipos
            b = instructions[i]
            if isinstance(b, Declaration):...
            elif isinstance(b, FunctionDeclaration):...
            elif isinstance(b, DeclarationStruct):...
            i += 1

        #capturo instrucciones del main
        augusTxt += '$ra = -1; #apuntador de pila de llamadas\n'
        augusTxt += '$s0 = array(); #pila de llamadas\n'
        #en mi etiqueta manejador siempre restara tope de pila
        i = 0
        while i < len(instructions):...
```

Por ejemplo, en la declaración, es acá donde empezara la traducción a código intermedio, en una cadena global llamada "AugusTxt" se ira concatenando el condigo intermedio.

```
def process(instructions,ts):
    #try:
        global augusTxt, augusTxtCalls, augusTxtAuxVar, contadorParams, augusTxtAuxJUMPS
    def Declaration(b, ts, type_):
        global augusTxt, contadorT, arrayTables, tsGeneral, ambitoGeneral, contadorGenereal
        for i in b:
            if isinstance(i, SingleDeclaration):
                if isinstance(i.val, IncreDecre_Post):...
                elif isinstance(i.val, IncreDecre_Pre):...
                elif isinstance(i.val, IdentifierArray):...
                else:
                    if isinstance(i.val, ReferenceBit):
                        res = valueExpression(i.val, ts)
                        augusTxt += '$t'+ str(contadorT)
                        augusTxt += f' = &{{str(res)}};\n'
                        arrayTables.pop()
                        ts.setdefault(i.id, f'${str(contadorT)}')
                        arrayTables.append(ts)
                        #tabla de simbolos mc
                        sym = mcTS.Symbol(f'{{i.id}}_{{contadorGenereal}}', type_, ambitoGeneral, f'&{{str(res)}}')
                        contadorGenereal += 1
                        tsGeneral.add(sym)
                    else:...
```

Ejemplo de traducción:



La traducción que se muestra en la imagen, es la traducción de una declaración global, en el código intermedio "\$s0" simulara una pila esta para las llamadas de funciones haciendo uso del apuntador "\$ra" que inicialmente estará con valor de -1.

```

else:
    Augus.setStyleSheet('QMainWindow{background-color: green; border: 1px solid black;}')
    #si no existen errores mandamos a traducir el ast y pintamos la barra de color verde
    augus = traduction.execute(result, self.textEditConsole)
    self.textEditConsole.setPlainText("CODIGO 3D:\n")
    self.textEditConsole.append(augus)
    print(f"texto augus:\n{augus}")
    print("traduccion completa")

    #mando a ejecutar augus
    result = grammar.parse(augus)
    #instructionsList = result[:]
    execute.execute(result, self.textEditOutput)
    #VALIDACION DE ERRORES SEMANTICOS

```

MinorC

Archivo Editar Ejecutar Opciones Reportes Ayuda

C:/Users/juarp/Desktop/Archivos de Prueba-20200628/basico.mc

```
int var1 = 1;
int punteo = 0;

void Declaracion()
{
    printf("===== Metodo Declaracion =====");
    int n4 = 2;
    char str4[] = "Voy a ganar Compiladore";
    double db4 = 0.0;
    double db1 = db4;
    char chr4 = 's';
    if (db1 == db4){
        printf(" %s%c %d :D\n", str4, chr4, n4);
        punteo = punteo + 5;
    }else {
        printf("Problemas en el metodo declaracion");
    }
    printf("=====");
}

void Aritmeticas(){

    printf("=====Aritmeticas=====");
}
```

CODIGO 3D:

```
main:
PL_:
PL_:
$t0 = 1;
$t1 = 0;
$ra = -1; #apuntador de pila de llamadas
$s0 = array(); #pila de llamadas
$t225 = 0;
print(" -----
CALIFICACION----- ");
print("\n");
$t226 = $t225 != 0 ;
if($t226) goto il57;
goto il58;
il57:
print(" No ");
print(" se ");
print(" toma ");
print(" con ");
print(" prioridad ");
print(" la ");
print(" variable ");
print(" local ");
print(" ante ");
print(" la ");
print(" global ");
```

OUTPUT:

```
-----CALIFICACION-----
===== Metodo Declaracion =====
Voy a ganar Compiladores2 :D
=====
=====Aritmeticas=====
El valor de n1 = 52.1
El valor de n3 = 70.0
Operaciones Aritmeticas 1: valor esperado:
a)62
```


Diagrama de clases Resumido:

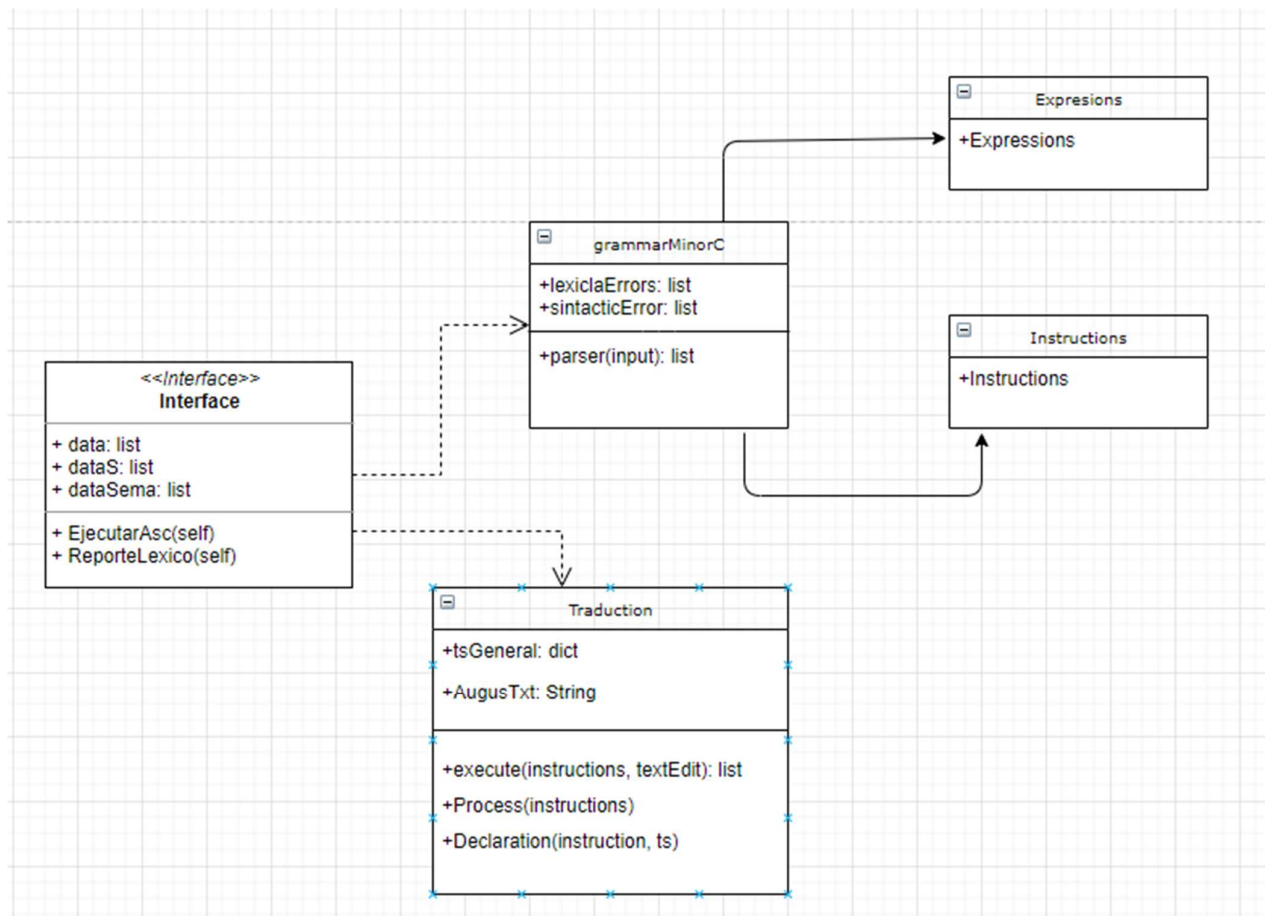


Diagrama de Flujo de ciclo de vida de ejecución:

