PROYECTO
AUGUS Y MINOR C

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
Facultad de Ingeniería

MANUAL TECNICO

Luis Alfonso Ordoñez Carrillo
201603127

Proyecto No.2 Organización de lenguajes y compiladores 2

Contenido

Introducción

La finalidad de este manual es dar a conocer al lector la lógica con la que se desarrolló este software por medio de imágenes de las pantallas y explicaciones del codigo.

Augus es un lenguaje de programación, basado en PHP y en MIPS. Su principal funcionalidad es ser un lenguaje intermedio, ni de alto nivel como PHP ni de bajo nivel como el lenguaje ensamblador de MIPS.

Para manejar el flujo de control se proporciona la declaración de etiquetas, sin tener palabras reservadas para ese uso. Es decir, no hay ciclos for, while, ni do-while.

MinorC es un subconjunto del lenguaje C, creado con el fin de poner en práctica los conceptos del proceso de compilación, esto quiere decir que es un lenguaje de alto nivel, al cual nosotros vamos a poder tranformar al lenguaje Augus

Objetivos

General

Proporcionar, al administrador del sistema u otros desarrolladores, una guía sobre las clases y sus atributos para facilitar la manipulación y el control del software.

Específicos

- Orientar al lector sobre la funcionalidad de los principales procesos del sistema.
- Facilitar el entendimiento del código para que el lector no tenga complicaciones al momento de realizar cambios en el programa.
- Dar a conocer la estructura de las gramáticas utilizadas para realizar el análisis ascendente y descendente
- Orientar al lector sobre el uso de graphviz y la creación de reportes en Python
- Que el lector compreda como se hace la traducción del un lenguaje de alto nivel a C3D
- Dar a conocer las estructuras utilizadas para el almacenamiento de cada tipo de dato neceario para crear y ejecutar el AST de MinorC

Requisitos del sistema

Configuración mínima

- **Microsoft Windows Vista SP1 / Windows 7 Professional:**
  - **Procesador:** Intel Pentium III a 800 MHz o equivalente
  - **Memoria:** 512 MB
  - **Espacio en disco:** 750 MB de espacio libre en disco
- **Ubuntu 9.10:**
  - **Procesador:** Intel Pentium III a 800 MHz o equivalente
  - **Memoria:** 512 MB
  - **Espacio en disco:** 650 MB de espacio libre en disco
- **Macintosh OS X 10.7 Intel:**
  - **Procesador:** Intel Dual-Core
  - **Memoria:** 2 GB
  - **Espacio en disco:** 650 MB de espacio libre en disco

Configuración recomendada

- **Microsoft Windows 7 Professional / Windows 8 / Windows 8.2:**
  - **Procesador:** Intel Core i5 o equivalente
  - **Memoria:** 2 GB (32 bits), 4GB (64bits)
  - **Espacio en disco:** el espacio libre en disco de 1,5 GB
- **Ubuntu 15.04:**
  - **Procesador:** Intel Core i5 o equivalente
  - **Memoria:** 2 GB (32 bits), 4GB (64bits)
  - **Espacio en disco:** el espacio libre en disco de 1,5 GB
- **OS X 10.10 Intel:**
  - **Procesador:** Intel Dual-Core
  - **Memoria:** 4 GB
  - **Espacio en disco:** el espacio libre en disco de 1,5 GB

AUGUS

GRAMATICA ASCENDENTE

```
inicio : instrucciones

instrucciones : instrucciones instruccion
              | instruccion

instruccion : etiqueta_main
            | etiqueta_ID
            | asignacion
            | inst_goto
            | inst_if
            | inst_print
            | inst_exit
            | inst_unset

etiqueta_main : MAIN DOSPUNTOS

etiqueta_ID : ID DOSPUNTOS

asignacion : variables IGUAL operaciones PUNTOCOMA

asignacion : variables_array IGUAL operaciones PUNTOCOMA

inst_goto : GOTO ID PUNTOCOMA

inst_if : IF PARENTA operaciones PARENTC GOTO ID PUNTOCOMA

inst_print : PRINT PARENTA variables PARENTC PUNTOCOMA
           | PRINT PARENTA val PARENTC PUNTOCOMA

inst_unset : UNSET PARENTA variables PARENTC PUNTOCOMA

inst_exit : EXIT PUNTOCOMA

operaciones : val MAS val
            | val MENOS val
            | val POR val
            | val DIVICION val
            | val RESIDUO val
            | val AND1 val
            | val OR1 val
            | val XOR val
            | val AND2 val
            | val OR2 val
            | val XOR2 val
            | val SHIFTI val
            | val SHIFTD val
            | val IGUALIGUAL val
            | val DIFERENTE val
            | val MAYORIGUAL val
            | val MENORIGUAL val
            | val MAYOR val
            | val MENOR val
            | MENOS val
            | EXCLAMA val
            | NOT val
            | ABS PARENTA val PARENTC
            | READ PARENTA PARENTC
            | ARRAY PARENTA PARENTC
            | val

val : conversiones
    | ENTERO
    | DECIMAL
    | CADENA
    | variables
    | variables_array

variables : TEMPORALES
          | PARAMETROS
          | VALORES_DEVUELTOS
          | SIMULADO
          | PILA
          | PUNTERO_PILA

 variables_array : TEMPORALES indices
                 | PARAMETROS indices
                 | VALORES_DEVUELTOS indices
                 | SIMULADO indices
                 | PILA indices
                 | PUNTERO_PILA indices

indices : indices indice
        | indice

 indice : CORCHEA val CORCHEC

conversiones : PARENTA INT PARENTC val
             | PARENTA FLOAT PARENTC val
             | PARENTA CHAR PARENTC val
```

GRAMATICA DESCENDENTE

```
inicio: instrucciones

instrucciones: instruccion instrucciones_p

instrucciones_p: instruccion instrucciones_p
               |3

instruccion: etiqueta_main
           | etiqueta_ID
           | asignacion
           | inst_goto
           | inst_if
           | inst_print
           | inst_exit
           | inst_unset

etiqueta_main: MAIN DOSPUNTOS

etiqueta_ID: ID DOSPUNTOS

asignacion: variables IGUAL operaciones PUNTOCOMA
          | variables_array IGUAL operaciones PUNTOCOMA

inst_goto : GOTO ID PUNTOCOMA

inst_if : IF PARENTA operaciones PARENTC GOTO ID PUNTOCOMA

inst_print : PRINT PARENTA variables PARENTC PUNTOCOMA
           | PRINT PARENTA val PARENTC PUNTOCOMA

inst_unset : UNSET PARENTA variables PARENTC PUNTOCOMA

inst_exit : EXIT PUNTOCOMA

operaciones : val operaciones_p

operaciones_p :  MAS val
                | MENOS val
                | POR val
                | DIVICION val
                | RESIDUO val
                | AND1 val
                | OR1 val
                | XOR val
                | AND2 val
                | OR2 val
                | XOR2 val
                | SHIFTI val
                | SHIFTD val
                | IGUALIGUAL val
                | DIFERENTE val
                | MAYORIGUAL val
                | MENORIGUAL val
                | MAYOR val
                | MENOR val

operaciones : MENOS val
            | EXCLAMA val
            | NOT val
            | ABS PARENTA val PARENTC
            | READ PARENTA PARENTC
            | ARRAY PARENTA PARENTC

operaciones : val

val: conversiones
   | ENTERO
   | DECIMAL
   | CADENA
   | variables
   | variables_array

variables : TEMPORALES
          | PARAMETROS
          | VALORES_DEVUELTOS
          | SIMULADO
          | PILA
          | PUNTERO_PILA

variables_array : TEMPORALES indices
                | PARAMETROS indices
                | VALORES_DEVUELTOS indices
                | SIMULADO indices
                | PILA indices
                | PUNTERO_PILA indices

indices : indice indices_P

indices_P : indice indices_P
          | 3

indice : CORCHEA val CORCHEC

conversiones : PARENTA INT PARENTC val
             | PARENTA FLOAT PARENTC val
             | PARENTA CHAR PARENTC val
```

Clase Main de programa

En la clase Main, es donde inicia la aplicación.

Primero se crea una clase llamda Ui_MainWindow que es la que va a recibir todos nuestros objetos de QWidgets que formaran parte de nuestra venta principal

Mas abajo también se declaran acciones para cada menú que necesitaremos para realizar diferentes acciones como ejecutar, debuggear o generar reportes

Tambien cuenta con otras funciones que nos permiten manejar los archivos y las pestañas.

```
class Ui_MainWindow(object):
    array_rutas = []
    array_editores = []
    array_nombre_tab = []
    nombre_archivo = ""
    ruta_archivo = ""
    Fuente = QtGui.QFont("Microsoft Sans Serif",10)
    Fuente2 = QtGui.QFont("Ebrima", 12)
    ruta_iconos = str(os.getcwd()) + "\Iconos"
    tabla_global = None
    indice = -1

    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.setWindowTitle("MI PROGRAMOTA!!")
        MyIcon = QtGui.QIcon(self.ruta_iconos + "\\favicon.png")
        MainWindow.setWindowIcon(MyIcon)
        MainWindow.resize(1355, 768)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")

        self.tbTab = QtWidgets.QTabWidget(self.centralwidget)
        self.tbTab.setGeometry(QtCore.QRect(20, 10, 770, 490))
        self.tbTab.setObjectName("tbTab")

        #Declaracion de la consola
        self.groupBox = QtWidgets.QGroupBox(self.centralwidget)
        self.groupBox.setGeometry(QtCore.QRect(20, 510, 770, 190))
        self.groupBox.setObjectName("groupBox")
        self.groupBox.setFont(self.Fuente)
        self.txtConsola = QtWidgets.QPlainTextEdit(self.groupBox)
        self.txtConsola.setGeometry(QtCore.QRect(10, 20, 750, 160))
        self.txtConsola.setObjectName("txtConsola")
        self.txtConsola.setFont(self.Fuente2)
        self.txtConsola.setStyleSheet("""QPlainTextEdit {background-color: #333; color: #00FF00;}""")
        self.txtConsola.setReadOnly(True)


        #Creacion del visor de imagenes
        self.visor = QtWidgets.QGraphicsView(self.centralwidget)
        self.visor.setGeometry(QtCore.QRect(795, 10, 555, 690))
        self.visor.setObjectName("qgVisor")

        MainWindow.setCentralWidget(self.centralwidget)

```

Clase Editor de código

Esta es una clase que se diseñó específicamente para poder colocar numero al editor de texto y que este mismo se vaya actualizando con forme se va escribiendo, también cuenta con la funcionalidad de subrayar la línea en la que estamos trabajando con un ligero color celeste.

Esta clase hereda todos los métodos de un QPlaintexEdit normal, lo que nos permite usar los métodos y funciones propios del mismo

```
from PyQt5.QtCore import Qt, QRect, QSize
from PyQt5.QtWidgets import QWidget, QPlainTextEdit, QTextEdit
from PyQt5.QtGui import QColor, QPainter, QTextFormat


class QLineNumberArea(QWidget):
    def __init__(self, editor):
        super().__init__(editor)
        self.codeEditor = editor

    def sizeHint(self):
        return QSize(self.editor.lineNumberAreaWidth(), 0)

    def paintEvent(self, event):
        self.codeEditor.lineNumberAreaPaintEvent(event)


class QCodeEditor(QPlainTextEdit):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.lineNumberArea = QLineNumberArea(self)
        self.blockCountChanged.connect(self.updateLineNumberAreaWidth)
        self.updateRequest.connect(self.updateLineNumberArea)
        self.cursorPositionChanged.connect(self.highlightCurrentLine)
        self.updateLineNumberAreaWidth(0)

    def lineNumberAreaWidth(self):
        digits = 1
        max_value = max(1, self.blockCount())
        while max_value >= 10:
            max_value /= 10
            digits += 1
        space = 3 + self.fontMetrics().width('9') * digits
        return space

    def updateLineNumberAreaWidth(self, _):
        self.setViewportMargins(self.lineNumberAreaWidth(), 0, 0, 0)

    def updateLineNumberArea(self, rect, dy):
        if dy:
            self.lineNumberArea.scroll(0, dy)
        else:
            self.lineNumberArea.update(0, rect.y(), self.lineNumberArea.width(), rect.height())
        if rect.contains(self.viewport().rect()):
            self.updateLineNumberAreaWidth(0)
```

Clase Valores_Funciones

Dentro de este documento se encuentran todas las estructuras de las variables que se manejan int, double, strings, también se guardan las operaciones binarias y unarias.

También se crean varios enum para ayudarnos a hacer mas fácil el análisis de que tipo de operación es cada uno y que tipo de datos hay en cada operación

Cada clase también, guarda el no de fila y columna donde fue encontrado, esto con fines de hacer mas fácil el análisis de los errores semánticos

```
class Tipo_Variable(Enum):
    TEMPORALES = 1
    PARAMETROS = 2
    DEVUELTOS = 3
    PILA = 4
    SIMULADOR = 5
    PUNTERO_PILA = 6

class Operacion_Convercion(Enum):
    TO_INT = 1
    TO_STR = 2
    TO_CHAR = 3


class Val_Numerico:
    '''print("clase val_num")'''

class Operacion_Binaria():
    def __init__(self, val1, val2, operacion, fila, columna):
        self.val1 = val1
        self.val2 = val2
        self.operacion = operacion
        self.fila = fila
        self.columna = columna

class Numerico_Negativo(Val_Numerico):
    def __init__(self, val, fila, columna):
        self.val = val
        self.fila = fila
        self.columna = columna

class Numerico_Absoluto(Val_Numerico):
    def __init__(self, val, fila, columna):
        self.val = val
        self.fila = fila
        self.columna = columna

#PRIMITIVOS
class Numerico_Entero(Val_Numerico):
    def __init__(self, val, tipo, fila, columna):
        self.val = val
        self.tipo = tipo
        self.fila = fila
        self.columna = columna
```

Clase Métodos

Dentro de esta clase se encuentran varias estructuras que nos ayudaran a la hora de ejecutar todas las instrucciones, generadas por el árbol. Estas también nos ayudaran con la generación del AST

```python
class Metodos:
    '''clase  abstracta'''

class Asignacion(Metodos):
    def __init__(self, variable, valor, fila, columna):
        self.variable = variable
        self.valor = valor
        self.fila = fila
        self.columna = columna

class Imprimir(Metodos):
    def __init__(self, mensaje, fila, columna):
        self.mensaje = mensaje
        self.fila = fila
        self.columna = columna

class If_Goto(Metodos):
    def __init__(self, op_logica, goto_etiqueta, fila, columna):
        self.op_logica = op_logica
        self.goto_etiqueta = goto_etiqueta
        self.fila = fila
        self.columna = columna

class Goto(Metodos):
    def __init__(self, etiqueta, fila, columna):
        self. etiqueta = etiqueta
        self.fila = fila
        self.columna = columna

class Read(Metodos):
    def __init__(self, fila, columna):
        '''Esta clase es del read'''
        self.fila = fila
        self.columna = columna

class Unset(Metodos):
    def __init__(self, variable, fila, columna):
        self.variable = variable
        self.fila = fila
        self.columna = columna

class Exit(Metodos):
    def __init__(self, fila, columna):
        ''''Esta clase es del exit'''
        self.fila = fila
        self.columna = columna
```

Tabla de Simbolos

En el documento de clases de simbolos podemos encontrar la clase símbolo que es el nodo en donde vamos a estar almacenando todos los valores que sean validos de la ejecucion y vamos a tener una clase llamada tabla_simbolos que tiene sus métodos de get y set para hacer un poco mas el guardado y el sacado de los simbolos de la misma

```python
class Simbolo():
    def __init__(self, id, tipo, valor, dimension, declarada, referencias):
        self.id = id
        self.tipo = tipo
        self.valor = valor
        self.dimension = dimension
        self. declarada = declarada
        self.referencias = referencias

class Tabla_Simbolos():
    def __init__(self, simbolos = {}):
        self.simbolos = simbolos

    def add_simbolo(self, simbolo):
        self.simbolos[simbolo.id] = simbolo

    def get_simbolo(self, id):
        if not id in self.simbolos:
            print("no existe el simbolo")
            return Simbolo(None,None,None,None,None,None)
        return self.simbolos[id]
        #imprimir error

    def update_simbolo(self, simbolo):
        if simbolo.id in self.simbolos:
            self.simbolos[simbolo.id] = simbolo

            #imprimir error

    def existe_simbolo(self, simbolo):
        if simbolo.id in self.simbolos:
            return True
        return False

    def existe_id(self, id):
        if id in self.simbolos:
            return True
        return False

    def clear(self):
        self.simbolos.clear()

    def get_all(self):
        return self.simbolos

    def delete_simbolo(self, id):
        if id in self.simbolos:
            texto = self.simbolos.pop(id)
```

Clase Errores

En esta tenemos una lista llamada Lista_errores que se declara como vacia y con visibilidad publica, lo que nos facilitara la inserción de los errores donde sea que los necesitemos. Tambien tenemos la clase Error que es el nodo donde se guarda lo necesario para hacer el reporte de errores luego

```
1  Lista_errores = []
2
3  class Error():
4      def __init__(self, token, tipo, desc, fila, columna):
5          self.token = token
6          self.tipo = tipo
7          self.desc = desc
8          self.fila = fila
9          self.columna = columna
```

Minor C

Gramatica

```
inicio : instrucciones

instrucciones : instrucciones instruccion
              | instruccion

instruccion : metodos
            | funciones
            | structs
            | declaracion PUNTOCOMA

metodos : VOID ID PARENTA PARENTC bloque_sentencias
        | VOID ID PARENTA lista_param PARENTC bloque_sentenc

funciones : tipo ID PARENTA PARENTC bloque_sentencias
          | INT MAIN PARENTA PARENTC bloque_sentencias
          | tipo ID PARENTA lista_param PARENTC bloque_sente

structs : STRUCT ID LLAVEA declaracion LLAVEC PUNTOCOMA

lista_param : lista_param COMA param
            | param

param : tipo ID

bloque_sentencias : LLAVEA lista_sentencias LLAVEC
                  | LLAVEA LLAVEC

lista_sentencias : lista_sentencias sentencia
                 | sentencia
                 |

sentencia : declaracion PUNTOCOMA
          | asignacion PUNTOCOMA
          | fun_if
          | fun_switch
          | fun_for
          | fun_while
          | fun_do_while
          | print
          | fun_return PUNTOCOMA
          | fun_break PUNTOCOMA
          | incre_decre PUNTOCOMA
          | fun_continue PUNTOCOMA
          | fun_goto PUNTOCOMA
          | fun_label
          | llamadas_fun

declaracion : tipo lista_declaracion
```

```
lista_declaracion : lista_declaracion COMA bloque_declara
                  | bloque_declara

bloque_declara : declaraConVal
               | declaraSinVal

declaraConVal : tipo_ID IGUAL operaciones
              | ID CORCHEA CORCHEC IGUAL CADENA
              | tipo_ID IGUAL SCANF PARENTA PARENTC
              | tipo_ID IGUAL LLAVEA lista_filas LLAVEC
              | tipo_ID IGUAL LLAVEA lista_val LLAVEC

declaraSinVal : tipo_ID

tipo : INT
     | CHAR
     | DOUBLE
     | FLOAT

tipo_ID : ID
        | ID dimension

dimension : dimension CORCHEA val CORCHEC
          | CORCHEA val CORCHEC
          | CORCHEA CORCHEC

asignacion : lista_asignacion

lista_asignacion : lista_asignacion COMA bloque_asignacion
                 | bloque_asignacion

bloque_asignacion : tipo_ID tipo_asignacion operaciones
                  | tipo_ID IGUAL SCANF PARENTA PARENTC
                  | tipo_ID tipo_asignacion LLAVEA lista_filas LLAVEC
                  | tipo_ID tipo_asignacion LLAVEA lista_val LLAVEC

tipo_asignacion : IGUAL
                | MASIGUAL
                | MENOSIGUAL
                | PORIGUAL
                | DIVIIGUAL
                | RESIGUAL
                | IZQIGUAL
                | DERIGUAL
                | ANDIGUAL
                | ORIGUAL
                | XORIGUAL
```

```
lista_filas : lista_filas COMA fila
            | fila

fila : LLAVEA lista_columna LLAVEC

lista_columna : lista_columna COMA columna
              | columna

columna : operaciones

lista_val : lista_val COMA operaciones
          | operaciones

fun_if : IF PARENTA operaciones PARENTC bloque_sentencias
       | IF PARENTA operaciones PARENTC bloque_sentencias ELSE fun_if
       | IF PARENTA operaciones PARENTC bloque_sentencias ELSE bloque_sentencias

fun_switch : SWITCH PARENTA operaciones PARENTC LLAVEA list_switch default LLAVEC
           | SWITCH PARENTA operaciones PARENTC LLAVEA list_switch LLAVEC
           | SWITCH PARENTA operaciones PARENTC LLAVEA default LLAVEC
           | SWITCH PARENTA operaciones PARENTC LLAVEA LLAVEC

list_switch : list_switch cont_switch
            |  cont_switch

cont_switch : CASE val DOSPUNTOS lista_sentencias
            | CASE val DOSPUNTOS

default : DEFAULT DOSPUNTOS lista_sentencias
        | DEFAULT DOSPUNTOS


fun_for : FOR PARENTA declaracion PUNTOCOMA operaciones PUNTOCOMA incre_decre PARENTC bloque_sentencias
        | FOR PARENTA asignacion PUNTOCOMA operaciones PUNTOCOMA incre_decre PARENTC bloque_sentencias

fun_while : WHILE PARENTA operaciones PARENTC bloque_sentencias

fun_do_while : DO bloque_sentencias WHILE PARENTA operaciones PARENTC PUNTOCOMA

fun_return : RETURN operaciones
           | RETURN incre_decre
           | RETURN

fun_break : BREAK
```

```
incre_decre : INCREMENTO val
            | DECREMENTO val
            | val INCREMENTO
            | val DECREMENTO

print : PRINTF PARENTA lista_val PARENTC PUNTOCOMA

fun_continue : CONTINUE

fun_label : ID DOSPUNTOS

fun_goto : GOTO ID

llamadas_fun : ID PARENTA PARENTC PUNTOCOMA
             | ID PARENTA lista_val PARENTC PUNTOCOMA

operaciones : operaciones MAS operaciones
            | operaciones MENOS operaciones
            | operaciones POR operaciones
            | operaciones DIVICION operaciones
            | operaciones RESIDUO operaciones
            | operaciones AND1 operaciones
            | operaciones OR1 operaciones
            |  operaciones IGUALIGUAL operaciones
            | operaciones DIFERENTE operaciones
            | operaciones MAYORIGUAL operaciones
            | operaciones MENORIGUAL operaciones
            | operaciones MAYOR operaciones
            | operaciones MENOR operaciones
            | operaciones XOR operaciones
            | operaciones AND2 operaciones
            | operaciones OR2 operaciones
            | operaciones SHIFTI operaciones
            | operaciones SHIFTD operaciones
            | MENOS operaciones
            | EXCLAMA operaciones
            | NOT operaciones
            | AND2 operaciones
            | operaciones TERNARIO operaciones DOSPUNTOS operaciones
            | incre_decre
            | val
            | PARENTA operaciones PARENTC
```

```
186   val : ENTERO
187       | DECIMAL
188       | CADENA
189       | CHAR
190       | ID
191       | variables_array
192
193   variables_array : ID indices
194
195   indices : indices indice
196           | indice
197
198   indice : CORCHEA val CORCHEC
```

Clases de Minor C

Minor C se utiliza en la misma interfaz que Augus, ya que Minor C es una herramienta que nos ayudara a pasar de código de alto nivel a código de 3 direcciones, para hacer esto Minor C cuenta con varias clases

Clase Traducir

La clase traducir es la encargada de guardar la raíz y las instrucciones generadas por el análisis lexico y sintactico y por medio de otras clase que sirven para guardar algunos métodos y funciones heredados.

Esta clase cuenta con el método de verificar tipos que sirve solo para verificar que los datos generados, por alguna expresión se regrese el tipo indicado.

```
class traducir():
    def __init__(self, instrucciones):
        self.instrucciones = instrucciones
        self.raiz = ambito(None)

    def inizializar_tablas(self):
        '''aqui va la traduccion'''
        limpiar()
        for instr in self.instrucciones:
            resultado = instr.agregar_Tabla(self.raiz, "global")

            if resultado == False:
                return False


    def verificar_tipos(self):
        for instr in self.instrucciones:
            resultado = instr.verificar_tipo(self.raiz)

            if resultado == False:
                return False

        return self

    def comenzar_traduccion(self):
        cod_augus = ""
        codigo_main = "main:\n"
        nueva_etiqueta =  new_etiqueta()
        set_salida(nueva_etiqueta)
        #atributo = Atributos()
        for instr in self.instrucciones:
            aux = instr.generar_C3D()
            if isinstance(instr,clase_main):
                #temp = codigo_main
                codigo_main += aux[0]
            elif isinstance(instr, Declaracion):
                codigo_main += aux[0]
            else:
                cod_augus += aux[0]
        retornos = get_etiquetal()
        salida_salida = new_etiqueta()
        codigo_main += "goto " + salida_salida + ";\n\n"
```

Clase variables

Esta clase es una instancia de nuestra propia clase abstracra, es ta es la encargada de verificar tipos, con su verificar tipo y de generar el AST del analizador correspondiente

```
class variables(abst):
    def __init__(self, id, fila, columna):
        self.id = id
        self.fila = fila
        self. columna = columna
        self.entorno = None

    def verificar_tipo(self, actual):
        simbolo = actual.get_simbol(self.id)

        if simbolo != False:
            self.entorno = actual
            return simbolo.tipo
        print("ERROR: NO EXISTE LA VARIABLE " + str(self.id))
        Err = Error("Variable", "Semantico", "No existe la variable", self.fila,
                    self.columna)
        Lista_errores.append(Err)
        return False

    def generar_C3D(self, ambt = None):
        simbolo = self.entorno.get_simbol(self.id)
        return ["", simbolo.var_aug]

    def get_tipo(self,ambt = None):
        simbolo = self.entorno.get_simbol(self.id)
        return simbolo.tipo

    def generar_AST(self, dot, nombre):
        nombre_hijo = str(self.id) + "_" + str(new_nombre())
        dot.edge(nombre, nombre_hijo)
        dot.node(nombre_hijo, self.id)

```

Clase Declaracion

Esta clase es una instancia de nuestra propia clase abstracta, esta es la encargada de verificar tipos, con su verificar tipo y la creación de las variables y el asignamiento de cada una, dentro de estas también tenemos

```
class Declaracion(abst):
    def __init__(self, tipo, lista, fila, columna):
        self.tipo = tipo
        self.lista = lista
        self.fila = fila
        self.columna = columna
        self.entorno = None

    def agregar_Tabla(self, actual, ambito_actual):
        for inst in self.lista:
            if isinstance(inst[0], variables):
                if not actual.exite_aqui(str(inst[0].id)):
                    sim = Simbolo(inst[0].id, self.tipo, "variable", ambito_actual, No
                    add_sim_report(sim)
                    actual.agregar_simbolo(sim)
                else:
                    print("la variable ya existe y no se puede vovler a declarar")
                    Err = Error("Declaracion", "Semantico", "La variable ya existe y n
                                self.fila, self.columna)
                    Lista_errores.append(Err)
                    return False
        return True

    def verificar_tipo(self, ambito):
        for inst in self.lista:
            if isinstance(inst[0], variables):
                simbolos = ambito.get_simbol(inst[0].id)

                if inst[1] != None:#verifica que la variable tenga valor o no
                    resultado = inst[1].verificar_tipo(ambito)

                    if resultado == False:
                        return False

                    if simbolos.tipo == Tipo_dato.ENTERO:
                        if resultado == Tipo_dato.CADENA:
                            print("Error no se puede asignar un valor")
                            Err = Error("Declaracion", "Semantico",
                                        "No se puede asignar un valor string a uno de
                                        self.columna)
                            Lista_errores.append(Err)
                            return False
```

Clase Asignacion

Esta clase es una instancia de nuestraclase abstracta, esta es la encargada de verificar tipos, con su verificar tipo y la creación de las variables y el asignamiento de cada una, dentro de estas también tenemos, esta también maneja el tipo de asignación que se quiere ya que pude ser =, +=, -=, etc

```
class Asignacion(abst):
    def __init__(self,lista, fila, columna):
        self.lista = lista
        self.fila = fila
        self.columna = columna
        self.entorno = None


    def verificar_tipo(self, ambito):
        for inst in self.lista:
            if isinstance(inst[0], variables):
                simbolos = ambito.get_simbol(inst[0].id)
                resultado = inst[1].verificar_tipo(ambito)
                if resultado == False or simbolos == False:
                    return False

                if simbolos.tipo == Tipo_dato.ENTERO:
                    if resultado == Tipo_dato.CADENA:
                        print("Error no se puede asignar un valor")
                        Err = Error("Asignacion", "Semantico", "No se puede asignar un valor string a uno de tipo entero", self.fila, self.columna)
                        Lista_errores.append(Err)
                        return False
                elif simbolos.tipo == Tipo_dato.DECIMAL:
                    if resultado == Tipo_dato.CADENA:
                        print("Error no se puede asignar un valor")
                        Err = Error("Asignacion", "Semantico",
                                    "No se puede asignar un valor string a uno de tipo decimal", self.fila, self.columna)
                        Lista_errores.append(Err)
                        return False
                elif simbolos.tipo == Tipo_dato.CARACTER:
                    if resultado == Tipo_dato.CADENA:
                        print("Error no se puede asignar un valor")
                        Err = Error("Asignacion", "Semantico",
                                    "No se puede asignar un valor string a uno de tipo caracter", self.fila, self.columna)
                        Lista_errores.append(Err)
                        return False
                elif simbolos.tipo == Tipo_dato.CADENA:
                    if resultado != Tipo_dato.CADENA and resultado != Tipo_dato.CARACTER:
                        print("Error no se puede asignar un valor")
                        Err = Error("Asignacion", "Semantico",
                                    "Solo se puede asignar un valor string o caracter a uno de tipo string", self.fila, self.columna)
                        Lista_errores.append(Err)
                        return False

        self.entorno = ambito
```

Clases de Metodos

Estas clases representan a los métodos que se pueden realizar en c, esta clase también es una clase que hereda de la clase abstracta y la utilizamos para manejar el flujo del código cuando se realizan llamadas a las funciones, esto se puede gracias a que se emula la stack y el heap

Este consta 3 funciones que sirven para ir creando todas la variables y llenar la tabla de simbolos, el siguente verifica los tipos de datos y por ultimo el generador de C3D

```
class clase_metodos(abst):
    def __init__(self, id, parametros, instrucciones, fila, columna):
        self.id = id
        self.parametros = parametros
        self.instrucciones = instrucciones
        self.fila = fila
        self.columna = columna
        self.entorno = None
        self.variables_param = None

    def agregar_Tabla(self, actual, ambito_actual):
        entorno_temp = ambito(actual)
        list_temp = []
        if self.parametros != None:
            for param in self.parametros:
                temp = new_param()
                param.append(temp)
                list_temp.append(param[0])

        dev = str(new_dev())

        sim1 = Simbolo(self.id, "Void", "Metodo", ambito_actual, self.parametros, 0, dev, self.fila, self.columna)
        actual.agregar_funcion(self.id, sim1)
        sim2 = Simbolo(self.id, "Void", "Metodo", ambito_actual, list_temp, 0, dev, self.fila, self.columna)
        add_sim_report(sim2)

        if self.parametros != None:
            for param in self.parametros:
                if not entorno_temp.exite_aqui(str(param[0])):
                    sim = Simbolo(param[0], param[1], "variable", ambito_actual + "_" + str(self.id), None, 0, param[2], self.fila,
                                  self.columna)
                    add_sim_report(sim)
                    entorno_temp.agregar_simbolo(sim)
                else:
                    print("la variable ya existe y no se puede vovler a declarar")
                    Err = Error("Metodos", "Semantico", "La variable ya existe y no se puede volver a declarar",
                                self.fila, self.columna)
                    Lista_errores.append(Err)
                    return False

        for inst in self.instrucciones:
            resultado = inst.agregar_Tabla(entorno_temp, ambito_actual + "_" + str(self.id))
            if resultado is False:
                Err = Error("Metodos", "Semantico", "Algo ha ocurrido en el cuerpo del metodo",
                            self.fila, self.columna)
                Lista_errores.append(Err)
                return False
```

Clase Primitivo

En esta clase se guardan todos los datos básicos que se manejan en Minor C. Esta también cumple con la función de hacer los casteos implícitos necesarios para que la respuesta sea la correcta y de devolver también el valor al cual representa

```
5   class Primitivo(abst):
6       def __init__(self, valor, tipo, fila, columna):
7           self.valor = valor
8           self.tipo = tipo
9           self.fila = fila
10          self.columna = columna
11
12
13      def verificar_tipo(self, ambito_actual):
14          if self.tipo == None:
15              return False
16          return self.tipo
17
18      def generar_C3D(self, tipo_A = None):
19
20          if tipo_A == "print" or not isinstance(tipo_A, Tipo_dato):
21              if self.tipo == Tipo_dato.CARACTER:
22                  return ["", "\'" + str(self.valor) + "\'"]
23              elif self.tipo == Tipo_dato.CADENA:
24                  return ["", "\"" + str(self.valor) + "\""]
25              else:
26                  return["", self.valor]
27
28          if self.tipo == Tipo_dato.ENTERO:
29              if tipo_A == Tipo_dato.DECIMAL:
30                  val1 = new_temp()
31                  val2 = new_temp()
32                  aug = str(val1) + " = " + str(self.valor) + ";\n"
33                  aug += str(val2) + " = (float)" + str(val1) + ";\n"
34                  return [aug, val2]
35              elif tipo_A == Tipo_dato.CADENA:
36                  val1 = new_temp()
37                  val2 = new_temp()
38                  aug = str(val1) + " = " + str(self.valor) + ";\n"
39                  aug += str(val2) + " = (char)" + str(val1) + ";\n"
40                  return [aug, val2]
41              else:
42                  return ["", self.valor]
```

Clase Main

Esta clase se encarga de almacenar todas las instrucciones que el analizador obtiene del la función, esta clase sirve para poder llevar un mejor control sobre el flujo de programa, esta también cuenta con los 3 metodos de llenar tabal de simbolos, verificar y generar 3CD. Este también cuenta con un método para crear el AST

```
class clase_main(abst):
    def __init__(self, instrucciones, fila, columna):
        self.instrucciones = instrucciones
        self.fila = fila
        self.columna = columna
        self.entorno = None

    def agregar_Tabla(self, actual, ambito_actual):
        sim = Simbolo("main", Tipo_dato.ENTERO, "Funcion", ambito_actual, None, 0, "main", self.fila, self.columna)
        add_sim_report(sim)
        entorno_temp = ambito(actual)
        for inst in self.instrucciones:
            resultado = inst.agregar_Tabla(entorno_temp, ambito_actual + str('_main'))
            if resultado is False:
                Err = Error("Main", "Semantico", "No se han podido declarar o asignar algunas variables",
                            self.fila, self.columna)
                Lista_errores.append(Err)
                return False
        self.entorno = entorno_temp
        return True

    def verificar_tipo(self, ambito):
        for inst in self.instrucciones:
            resultado = inst.verificar_tipo(self.entorno)
            if resultado is False:
                Err = Error("Main", "Semantico", "Algo ha ocurrido en el main",
                            self.fila, self.columna)
                Lista_errores.append(Err)
                return False
        return True


    def generar_C3D(self):
        augus = ""

        augus += "$s0 = array();\n"
        augus += "$s1 = array();\n"
        augus += "$sp = 0;\n"
        augus += "$ra = -1;\n"


        for instr in self.instrucciones:
            resultado = instr.generar_C3D()
            augus += resultado[0]
        return [augus, ""]
```

Reportes

Reporte de errores

```
class Reporte_Errores:
    def crear_reporte(self, errores):
        dot = Digraph(comment="Reporte de Errores",
                      format="png",
                      node_attr={'shape': 'plaintext'},
                      graph_attr={'rankdir': 'LR'})
        contenido = "<<table border = '1'>\n<tr><td>Token</td><td>Tipo</td><td>Descripcion</td><td>Fila</td><td>Columna</td></tr>\n"

        for error in errores:
            cont = str(error.token)
            if cont.find('&') != -1:
                cont = str(error.token).replace('&', "&#38;")
            elif cont.find('<') != -1:
                cont = str(error.token).replace('<', "&#60;")
            elif cont.find('>') != -1:
                cont = str(error.token).replace('>', "&#62;")
            else:
                cont = str(error.token)
            contenido += "<tr><td>" + cont + "</td><td>" + str(error.tipo) + "</td><td>" + str(error.desc) + "</td><td>" + str(error.fila) + "</td><td>" + str(error.columna) + "</td></tr>\n"

        contenido += "</table>>"
        dot.node("A", contenido)
        dot.node("T", "Reporte de Errores")

        try:
            dot.render("Reportes/Reporte_Error", view=False)
        except:
            print("El reporte no se pudo gener")

    def __init__(self):
        '''clase del reporte simbolos'''
```

Reporte AST Minor C

```
from graphviz import Digraph
from AST import *
from Traduccion.Valores import *


class AST_MiniC():
    contador = 0

    def crear_reporte(self, lista):
        dot = Digraph(comment="Reporte AST descendente",
                      format="png",
                      node_attr={'shape': 'box'})

        nombre_padre = str(new_nombre())
        dot.node(nombre_padre, "Inicio")

        for inst in lista:
            inst.generar_AST(dot, nombre_padre)

        try:
            dot.render("Reportes/Reporte_AST_C", view=False)
        except:
            print("El reporte no se pudo gener")

    def __init__(self):
        '''clase del reporte simbolos'''

```