## Integrantes:

* Logotipo

  Descripción generada automáticamenteAriel Humberto Valle Escoto
* Eduardo Quetzal Delgado Pimentel
* Ricardo David López Arellano

Materia: Traductores de lenguajes II

2023b

## Introducción:

Un analizador sintáctico descendente verifica si una cadena de tokens pertenece al lenguaje definido por una gramática, siguiendo una derivación por la izquierda del símbolo inicial.

Hay diferentes tipos de analizadores sintácticos descendentes, como el por descenso recursivo, el predictivo y el con retroceso, que usan distintos métodos para elegir la regla de producción adecuada en cada paso.

El código de tres direcciones es un tipo de código intermedio que usan los compiladores para optimizar el rendimiento y el uso de recursos de los programas. Cada instrucción de este código tiene como máximo tres operandos: un destino y dos fuentes. Por ejemplo, t1 := t2 + t3. El código de tres direcciones facilita la evaluación de expresiones aritméticas, el control del flujo, la asignación de registros y la detección de subexpresiones comunes. Hay distintas formas de representar el código de tres direcciones, como los cuartetos, los tercetos y los tercetos indirectos, que tienen sus ventajas e inconvenientes.

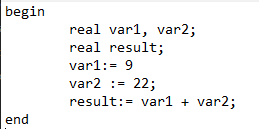
## Desarrollo:

Para comenzar, la gramática mostrada a la derecha se implementó en el código fuente por medio del analizador léxico. Se modificó la práctica 5 usando esta nueva gramática previamente corregida en la práctica 3.

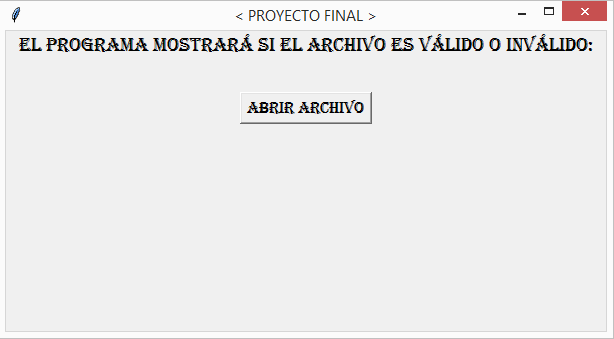
El objetivo de esta primera parte del proyecto es verificar que la gramática introducida sea válida, y, de no serlo, muestre el error.

La segunda parte del proyecto requiere que se imprima el código de 3 direcciones generado por el analizador sintáctico descendente predictivo, además de la función de la primera parte.

1. Primero, para probar la primera parte del programa, se ejecutará el siguiente archivo de entrada con un error en la 4ta línea (falta un punto y coma al final):



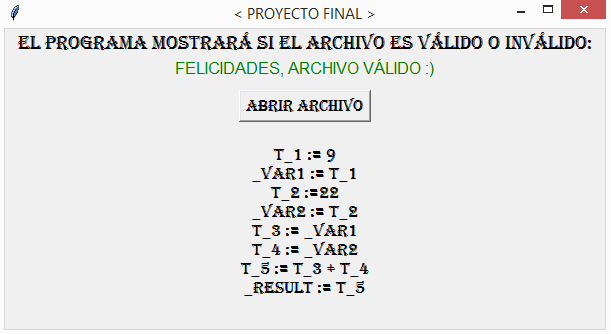
En el programa se mostrará así:



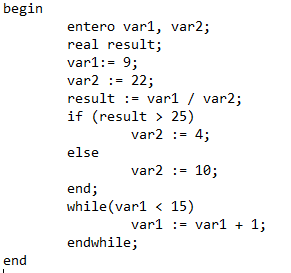
Y como el archivo le falta el “;” mostrará error:



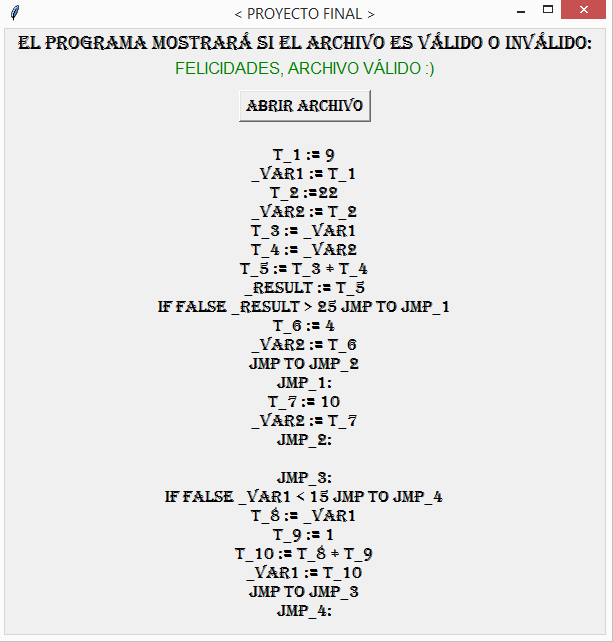
Ya modificando el “;” en el .txt mostrará lo siguiente el programa:



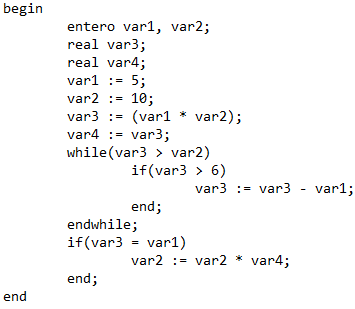
1. Ahora este es el siguiente ejemplo:



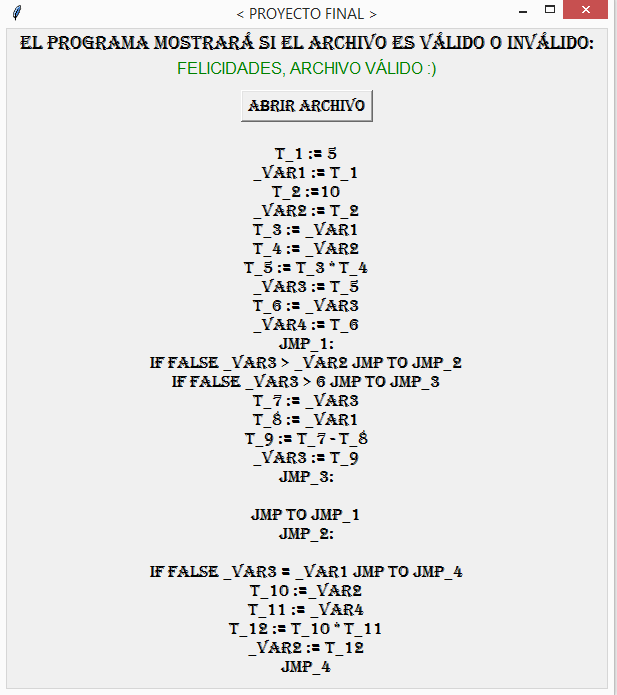
El programa mostrará así el resultado:



1. Ahora será con el siguiente algoritmo que es más complejo:



El programa mostrará así el resultado:



## Código:

Main.py

import ply.yacc as yacc

from lexer import tokens

import tkinter as tk

from tkinter import filedialog, font

import re

import tkinter as tk

from tkinter import font

from PIL import Image, ImageTk

import networkx as nx

import matplotlib.pyplot as plt

# Reglas de la gramática

def p\_expression(p):

    '''expression : expression PLUS term

                  | expression MINUS term

                  | term'''

    if len(p) == 2:

        p[0] = p[1]

    else:

        if p[2] == '+':

            p[0] = p[1] + p[3]

        elif p[2] == '-':

            p[0] = p[1] - p[3]

def p\_term(p):

    '''term : term TIMES factor

            | term DIVIDE factor

            | factor'''

    if len(p) == 2:

        p[0] = p[1]

    else:

        if p[2] == '\*':

            p[0] = p[1] \* p[3]

        elif p[2] == '/':

            p[0] = p[1] / p[3]

def p\_factor(p):

    '''factor : NUMBER

              | LPAREN expression RPAREN'''

    if len(p) == 2:

        p[0] = p[1]

    else:

        p[0] = p[2]

# Regla para manejar errores de sintaxis

def p\_error(p):

    print("ARCHIVO INVÁLIDO :(")

    raise SyntaxError("Error de sintaxis en la entrada")

# Construir el analizador

parser = yacc.yacc()

# Crear la interfaz gráfica

ventana = tk.Tk()

ventana.title("< PROYECTO FINAL >")

# Agregar un subtítulo

subtitulo\_label = tk.Label(ventana, text="El programa mostrará si el archivo es válido o inválido:", font=("Algerian", 14))

subtitulo\_label.pack()

# Variable para almacenar el resultado y enlazarla con la etiqueta

result\_var = tk.StringVar()

# Cambia el tamaño de la ventana

ventana.geometry("600x300")  # Cambia el ancho y alto

# Crear un Label para mostrar el resultado

resultado\_label = tk.Label(ventana, textvariable=result\_var, font=("Helvetica", 12))

resultado\_label.pack()

# Función para analizar el archivo fuente

def parse\_file(file\_path):

    try:

        with open(file\_path, 'r') as file:

            source\_code = file.read()

            parser.parse(source\_code)

            result\_var.set("FELICIDADES, ARCHIVO VÁLIDO :)")

            resultado\_label.config(fg="green")  # Cambia el color del texto a verde para indicar un resultado válido

    except SyntaxError as e:

        result\_var.set("LO SENTIMOS, ARCHIVO INVÁLIDO :(")

        resultado\_label.config(fg="red")  # Cambia el color del texto a rojo para indicar un resultado inválido

# Función para abrir un cuadro de diálogo y obtener la ruta del archivo

def open\_file\_dialog():

    file\_path = filedialog.askopenfilename()

    if file\_path:

        parse\_file(file\_path)

# Botón para abrir el cuadro de diálogo

open\_button = tk.Button(ventana, text="Abrir Archivo", command=open\_file\_dialog, font=("Algerian", 12))

open\_button.pack(pady=10)

# Iniciar el bucle de eventos de la interfaz gráfica

ventana.mainloop()

lexer.py

import ply.lex as lex

# Lista de tokens

tokens = (

    'NUMBER',

    'PLUS',

    'MINUS',

    'TIMES',

    'DIVIDE',

    'LPAREN',

    'RPAREN',

)

# Expresiones regulares para tokens simples

t\_PLUS = r'\+'

t\_MINUS = r'-'

t\_TIMES = r'\\*'

t\_DIVIDE = r'/'

t\_LPAREN = r'\('

t\_RPAREN = r'\)'

# Token para números

def t\_NUMBER(t):

    r'\d+'

    t.value = int(t.value)

    return t

# Ignorar caracteres espacio en blanco

t\_ignore = ' \t'

# Manejo de errores

def t\_error(t):

    #print("Carácter ilegal:", t.value[0])

    t.lexer.skip(1)

# Construir el analizador léxico

lexer = lex.lex()

# Ejemplo de uso

if \_\_name\_\_ == '\_\_main\_\_':

    lexer.input("3 + 4 \* 5")

    while True:

        tok = lexer.token()

        if not tok:

            break

        print(tok)

parsetab.py

# parsetab.py

# This file is automatically generated. Do not edit.

# pylint: disable=W,C,R

\_tabversion = '3.10'

\_lr\_method = 'LALR'

\_lr\_signature = 'DIVIDE LPAREN MINUS NUMBER PLUS RPAREN TIMESexpression : expression PLUS term\n                  | expression MINUS term\n                  | termterm : term TIMES factor\n            | term DIVIDE factor\n            | factorfactor : NUMBER\n              | LPAREN expression RPAREN'

\_lr\_action\_items = {'NUMBER':([0,5,6,7,8,9,],[4,4,4,4,4,4,]),'LPAREN':([0,5,6,7,8,9,],[5,5,5,5,5,5,]),'$end':([1,2,3,4,11,12,13,14,15,],[0,-3,-6,-7,-1,-2,-4,-5,-8,]),'PLUS':([1,2,3,4,10,11,12,13,14,15,],[6,-3,-6,-7,6,-1,-2,-4,-5,-8,]),'MINUS':([1,2,3,4,10,11,12,13,14,15,],[7,-3,-6,-7,7,-1,-2,-4,-5,-8,]),'RPAREN':([2,3,4,10,11,12,13,14,15,],[-3,-6,-7,15,-1,-2,-4,-5,-8,]),'TIMES':([2,3,4,11,12,13,14,15,],[8,-6,-7,8,8,-4,-5,-8,]),'DIVIDE':([2,3,4,11,12,13,14,15,],[9,-6,-7,9,9,-4,-5,-8,]),}

\_lr\_action = {}

for \_k, \_v in \_lr\_action\_items.items():

   for \_x,\_y in zip(\_v[0],\_v[1]):

      if not \_x in \_lr\_action:  \_lr\_action[\_x] = {}

      \_lr\_action[\_x][\_k] = \_y

del \_lr\_action\_items

\_lr\_goto\_items = {'expression':([0,5,],[1,10,]),'term':([0,5,6,7,],[2,2,11,12,]),'factor':([0,5,6,7,8,9,],[3,3,3,3,13,14,]),}

\_lr\_goto = {}

for \_k, \_v in \_lr\_goto\_items.items():

   for \_x, \_y in zip(\_v[0], \_v[1]):

       if not \_x in \_lr\_goto: \_lr\_goto[\_x] = {}

       \_lr\_goto[\_x][\_k] = \_y

del \_lr\_goto\_items

\_lr\_productions = [

  ("S' -> expression","S'",1,None,None,None),

  ('expression -> expression PLUS term','expression',3,'p\_expression','proyecto.py',6),

  ('expression -> expression MINUS term','expression',3,'p\_expression','proyecto.py',7),

  ('expression -> term','expression',1,'p\_expression','proyecto.py',8),

  ('term -> term TIMES factor','term',3,'p\_term','proyecto.py',18),

  ('term -> term DIVIDE factor','term',3,'p\_term','proyecto.py',19),

  ('term -> factor','term',1,'p\_term','proyecto.py',20),

  ('factor -> NUMBER','factor',1,'p\_factor','proyecto.py',30),

  ('factor -> LPAREN expression RPAREN','factor',3,'p\_factor','proyecto.py',31),

]

Conclusiones:

En conclusión, lo que aprendimos al terminar de realizar el proyecto de traductores II, es la importancia de todo lo que venimos viendo en el semestre, ya que desde el código de tres direcciones, hasta los analizadores sintácticos, es importante aprender de ellos, ya que estos nos han mejorado para ser mejores programadores y a su vez también nos ayudó a mejorar el cómo trabajamos en equipo y lo bueno de este proyecto fue que integramos mucho de lo visto durante el semestre que se siente como armar un rompecabezas, que acabamos de construir ya que por fin vemos todas las piezas juntas y en funcionamiento.

Para terminar esperemos que como equipo este proyecto nos haya enseñado las bases de los compiladores para así luego si se necesitara pues aprender más de ellos, pero con unas buenas bases para hacer de ese aprendizaje algo más rápido.