UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERIA

AUXILIA: GUILLERMO FRANCISCO MELLINI SALGUERO

PRACTICA 1

ANALIZADOR LEXICO

MANUAL TECNICO

PRESENTADO POR

DAVID ORLANDO FUENTES MORALES

REGISTRO ACADÉMICO 201709022

DPI 2998989540101

GUATEMALA, 19 de septiembre de 2024

TABLA DE CONTENIDO

INTRODUCCION

El presente documento describe el diseño del programa creado en fortran, en el cual se utilizaron diversas herramientas para su creación, se describirá cada parte del código, módulos, estados, interfaz gráfica, y la compilación entre fortran y Python usando tkinter para la interfaz gráfica, se puso el uso del conocimiento del método del árbol para generar un autómata finito determinista (AFD).

Propósito del documento

Proporcionar una guía del uso y explicación de la creación del analizador de mercado utilizando código fortran, Describir los métodos y técnicas de prueba utilizadas para la lectura de documentos y creación de archivos HTML y funcionalidad del código, integración de Python y fortran

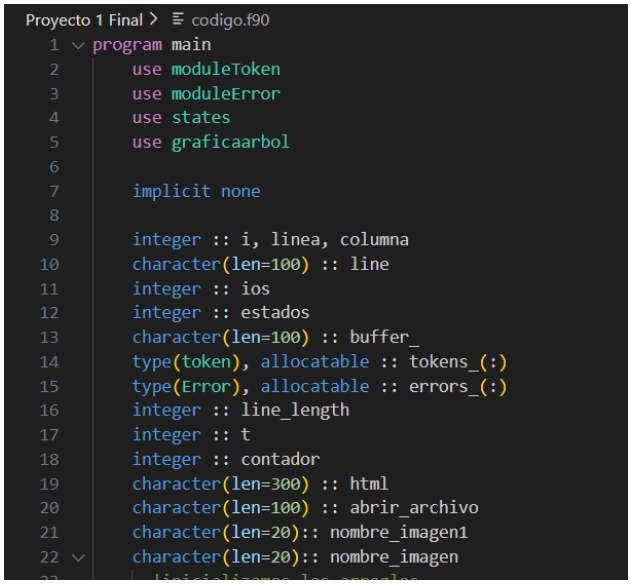
Descripción General

Este programa en Fortran procesa un archivo de texto línea por línea, aplicando un conjunto de estados definidos en módulos externos para analizar el contenido del archivo. El flujo del programa se estructura alrededor de la lectura de un archivo, procesamiento de tokens, y gestión de errores. Está diseñado para ejecutar un análisis de estados a través de diferentes funciones llamadas según el estado actual de procesamiento.

Código

Main

- Se comienza importando módulos creados para el funcionamiento del programa. Se declaran variables que se usaran para almacenar y rutas de archivos a utilizar por el usuario.



El programa utiliza varios módulos que contienen definiciones y funciones esenciales para su ejecución:

- **module Token**: Contiene la definición del tipo token y sus manipulaciones.
- **module Error**: Define el tipo Error y maneja los errores que ocurren durante el procesamiento.
- **states**: Contiene la lógica de los distintos estados (state0, state1, state2, etc.) que determinan cómo se procesa cada carácter del archivo.
- **graficaarbol**: se utiliza para generar representaciones gráficas de los resultados.

Declaración de Variables

Se utilizan diversas variables para controlar el flujo del programa:

- linea, columna: Enteros que controlan la posición actual en el archivo.
- line: Cadena de 100 caracteres para almacenar cada línea del archivo.
- ios: Variable entera para gestionar el estado de las operaciones de entrada/salida.
- estados: Variable entera que mantiene el estado actual del autómata o proceso de análisis.
- tokens_, errors_: Arrays de tipo token y Error respectivamente, para almacenar los tokens y errores encontrados durante el análisis.
- abrir_archivo, nombre_imagen, nombre_imagen1: Cadenas que almacenan el nombre del archivo que se va a procesar y los nombres de las imágenes para las gráficas.

- Se inicializan las variables

```
!inicializamos el estado
estados = 0
buffer_ = ""
ios = 0
line_length = 0
abrir_archivo = "Texto.txt"
nombre_imagen = "graficaultimo"
nombre_imagen1 = "graficaultimo1"
!abrir_archivo2 = "salidasss.txt"
!abrir_lectura ="salidasss2.txt"
```

- Se abre el archivo a utilizar con la función open, si la apertura del archivo es exitosa se utiliza el iostat=ios.

```
!abrir el archivo
open(unit=10, file=abrir_archivo, status="old", action="read", iostat=ios)
!leer el archivo
```

- El programa entra en un bucle do para leer línea por línea.

```
do
    i=1
    read(10, '(A)', iostat=ios) line
    if (ios /= 0) then
        exit
    end if
```

- Procesamiento de estados
  ara cada carácter en la línea, el programa llama a una función diferente que corresponde al estado actual.

```
!recorremos la linea
do while(i<=line_length)
    !print *, "linea: ", line(i:i), " estado: ", estados
    if (estados==0)then
        !estado 0
        call state0(line(i:i), buffer_,tokens_,errors_,linea,columna,estados,line_length==i)

    else if (estados==1)then
        !estado 1
        call state1(line(i:i), buffer_,tokens_,linea,columna,estados,i)
    else if (estados==2)then
        call state2(line(i:i), buffer_,tokens_,linea,columna,estados,i)
    else if (estados==3)then
        call state3(line(i:i), buffer_,tokens_,linea,columna,estados)

    end if
columna= columna + 1
  i= i+1
  end do
 linea = linea + 1
columna = 1
```

Generación de HTML tokens y errores

El programa crea un archivo HTML llamado tabla.html que contiene una tabla con todos los tokens generados durante el análisis. Este proceso incluye:

```
if(size(errors_)>0 .or. size(tokens_)>0 )then
    if(size(tokens_)>0)then
open(unit=10, file="tabla.html", status="replace", action="write", iostat=ios)
! Verificar si hubo error al abrir el archivo
if (ios /= 0) then
    print *, "Error al abrir el archivo HTML."
    stop
end if
```

- Creación de la estructura html.
  Tendrá como encabezado numero, lexema , tipo , fila y columna.

```
html = "<html>" // &
       "<head><title>Tabla de Datos</title></head>" // &
      ! "<body>" // &
       "<h1>TOKENS</h1>" // &
       "<table border=""1"">" // &
       !"<tr><th>No</th><th>leema</th><th>tipo</th><th>fila</th><th>calumna</th></tr>"
       !color de la tabla

       "<tr><th>No</th><th>lexema</th><th>tipo</th><th>fila</th><th>calumna</th></tr>"

write(10, '(A)') trim(html)
! Agregar las filas a la tabla

do t = 1, size(tokens_)
    contador = t
    html = "<tr><td>" // trim(adjustl(int_to_string(contador))) // "</td>" //&
    " <td>" // trim(tokens_(t)%lexema) // "</td>" // &
            "<td>" // trim(tokens_(t)%tipo) // "</td>" // &
            "<td>" // trim(adjustl(int_to_string((tokens_(t)%linea)/2))) // "</td>" // &
            "<td>" // trim(adjustl(int_to_string((tokens_(t)%columna)/2))) // "</td></tr>"
    write(10, '(A)') trim(html)
end do
! Cerrar la tabla y el documento HTML
html = "</table></body></html>"
write(10, '(A)') trim(html)
```

Se utiliza un do para la impresión de cada token leído.

- Cierre del archivo html

```
! Cerrar la tabla y el documento HTML
html = "</table></body></html>"
write(10, '(A)') trim(html)
! Cerrar el archivo
close(10)
!print *, "Archivo HTML creado correctamente."
end if
```

- Bloque condicional If

```
endif
if(size(errors_)==0)then
    call procesar_archivo(abrir_archivo, salida_archivo)
    end if
```

Verifica si la variable errors esta vacia, es decir si no hay errores registrados.
Si se cumple se llama el subrutine procesar_archivo el cual solicita dos variables o argumentos

- Inicio de subrutinas y funciones

```
contains

    function int_to_string(t) result(str)
        integer, intent(in) :: t
        character(len=100) :: str
        write(str, '(I0)') t
        str = adjustl(str)
    end function int_to_string

end program main
```

Función int_to_string: convierte un entero integer en una cadena de texto.
str: Una cadena de texto (de longitud máxima 100 caracteres) que contiene la representación del número entero.
End program main: finaliza el programa main.

Modul Error.

El módulo moduleError está diseñado para manejar errores en una aplicación. La estructura central es el tipo Error, que encapsula un mensaje de error, su tipo y su ubicación (línea y columna).

- El módulo contiene las definiciones relacionadas con el manejo de errores. También utiliza un módulo externo Utils, que debe contener funciones y subrutinas adicionales útiles.
- **implicit none:** Fuerza la declaración explícita de todas las variables para evitar errores de tipificació

```
module moduleError
    use Utils

    implicit none
    !definir el tipo de error
    type :: Error
        !integer :: id
        character(len=100) :: mensaje, tipo
        integer :: linea
        integer :: columna
    end type Error
contains
```

- Definición de tipo:
  Propósito: Define una estructura de datos llamada Error para almacenar información sobre un error.
  mensaje: Contiene un mensaje descriptivo del error.

tipo: Indica el tipo de error (puede ser, por ejemplo, un error sintáctico o de tipo lógico).
linea y columna: Indican la ubicación en el archivo o código donde se produjo el error.

- Subrutina Init Error

```
subroutine initError(mensaje,tipo,linea,columna,e)
    implicit none
    character(len=*), intent(in) :: mensaje
    character(len=*), intent(in) :: tipo
   ! integer, intent(in) :: tipo
    integer, intent(in) :: linea
    integer, intent(in) :: columna
    type(Error), intent(inout) :: e
    !limpiamos el error
    e%mensaje = ""
    !e%id = 0
    e%linea = 0
    e%columna = 0
    !asignamos los valores
    e%mensaje = trim(mensaje)
    e%linea = linea
    e%columna = columna
    e%tipo = trim(tipo)

end subroutine initError
```

Inicializa una variable del tipo Error con los datos que se le pasan como argumentos.
Entradas:
mensaje: Descripción del error.
tipo: Tipo de error.
linea: Línea donde ocurrió el error.
columna: Columna donde ocurrió el error.
e: Variable de tipo Error que será inicializada.
Funcionamiento:
Primero limpia los campos de la variable e asignando valores iniciales (vacíos o cero).
Luego, ajusta los valores recibidos a la variable e.

- Subrutina Add errors

```
subroutine addError(errors,mensaje,buffer_,tipo,linea,columna)
    implicit none
    type(Error), allocatable, intent(inout) :: errors(:)
    character(len=*), intent(inout) :: buffer_
    character(len=*), intent(in) :: mensaje
    character(len=*), intent(in) :: tipo
    integer, intent(in) :: linea
    integer, intent(in) :: columna
    type(Error):: e
    integer :: i
    !inicializamos el token
    call initError(mensaje,tipo,linea,columna,e)
```

```
subroutine addError(errors,mensaje,buffer_,tipo,linea,columna)
    implicit none
    type(Error), allocatable, intent(inout) :: errors(:)
    character(len=*), intent(inout) :: buffer_
    character(len=*), intent(in) :: mensaje
    character(len=*), intent(in) :: tipo
    integer, intent(in) :: linea
    integer, intent(in) :: columna
    type(Error):: e
    integer :: i
    !inicializamos el token
    call initError(mensaje,tipo,linea,columna,e)
    !agregamos el token al arreglo
    if(allocated(errors))then
        i = size(errors)
        call extendArray(errors)
    else
        allocate(errors(1))
        i =0

    end if
    errors(i+1) = e
    call clearBuffer(buffer_)

end subroutine addError
```

Agrega un nuevo error al arreglo errors.
Entradas:
errors: Arreglo de errores donde se agregará el nuevo error.
mensaje: Descripción del error.
buffer_: Un buffer temporal que se limpia después de agregar el error.
tipo: Tipo de error.
linea y columna: Ubicación del error en el archivo.
Funcionamiento:
Se inicializa un nuevo error llamando a initError.
Si el arreglo errors está asignado, su tamaño se extiende con la subrutina extendArray; de lo contrario, se inicializa el arreglo con un solo elemento.
El nuevo error se añade al arreglo, y luego se limpia el buffer temporal con call clearBuffer(buffer_).

- Subrutina extendendArray

```
    subroutine extendArray(errors)
        implicit none
        type(Error), allocatable, intent(inout) :: errors(:)
        type(Error), allocatable :: temp(:)
        integer :: i
        !extendemos el arreglo
        i = size(errors)
        allocate(temp(i+1))
        temp(1:i) = errors
        deallocate(errors)
        errors = temp

    end subroutine extendArray

end module moduleError
```

Extiende el tamaño del arreglo errors para permitir agregar un nuevo elemento.
Se crea un arreglo temporal temp con un tamaño mayor en una unidad al original.
Se copian los datos del arreglo original a temp, luego se libera el espacio ocupado por el arreglo original y se reasigna errors con el nuevo tamaño.

Modulo Token

Este módulo contiene las definiciones relacionadas con el manejo de tokens. Al igual que en el módulo anterior, se hace uso del módulo Utils (que debe contener utilidades adicionales), y la directiva implicit none asegura que todas las variables estén explícitamente declaradas.

- Tipo

```
module moduleToken
    use Utils
    implicit none
    !definir el tipo de token
    type :: token
        !integer :: id
        character(len=100) :: lexema, tipo
        integer :: linea
        integer :: columna
    end type token
contains
```

Define una estructura de datos llamados token que almacena la información.
Atributos :
Lexema: representa textual a un token
Tipo: el tipo de token.
Línea: posición en el archivo
Columna: posición en el archivo

- Subrutina inittken

```
subroutine initToken(buffer_,tipo,linea,columna,token_)
    implicit none
    character(len=*), intent(in) :: buffer_
    character(len=*), intent(in) :: tipo
   ! integer, intent(in) :: tipo
    integer, intent(in) :: linea
    integer, intent(in) :: columna
    type(token), intent(out) :: token_
    !limpiamos el token
    token_%lexema = ""
    !token_%id = 0
    token_%linea = 0
    token_%columna = 0
    !asignamos los valores
    token_%lexema = trim(buffer_)
    token_%linea = (linea)
    token_%columna = (columna)
    token_%tipo = trim(tipo)

end subroutine initToken
```

Aquí se inicializa la variable del tipo token con los valores que se pasan como argumentos.
Buffer: el lexema del token
Tipo: el tipo de token.
Linea: línea donde se encuentra el token
Columna: columna donde se encuentra el token.
Token_: la variable de tipo token que se inicializa.
Funcionamiento:
Primero, se limpian los atributos de la variable token_ (vacía el lexema, línea, columna, etc.).
Luego, se asignan los valores recibidos como parámetros al token_ (ajustando espacios si es necesario mediante la función(trim).
Esta subrutina se usa para crear o reinicializar un token con nuevos valores cuando se detecta en el código fuente.

- Subrutina Addtoken.

```
subroutine addToken(tokens_,buffer_,tipo,linea,columna)
    implicit none
    type(token), allocatable, intent(inout) :: tokens_(:)
    character(len=100), intent(inout) :: buffer_
    character(len=*), intent(in) :: tipo
    integer, intent(in) :: linea,columna
    type(token):: token_
    integer :: i
    !inicializamos el token
    call initToken(buffer_,tipo,linea,columna,token_)
    !agregamos el token al arreglo
    if(allocated(tokens_))then
        i = size(tokens_)
        call extendArrays(tokens_)
    else
        allocate(tokens_(1))
        i =0

    end if
    tokens_(i+1) = token_
    call clearBuffer(buffer_)
end subroutine addToken
```

Agrega un nuevo token al arreglo.
Entradas:
tokens_: Arreglo de tokens donde se almacenará el nuevo token.
buffer_: El lexema que se va a convertir en token.
tipo: Tipo del token.
linea y columna: Ubicación del token en el archivo.

Llama a la subrutina initToken para inicializar el token con los valores proporcionados.

Si el arreglo tokens_ está asignado, se incrementa su tamaño con la subrutina extendArrays; si no está asignado, se inicializa con un tamaño de 1.
Finalmente, el token es añadido al arreglo, y el buffer_ se limpia utilizando call clearBuffer(buffer_).

- Subrutina ExtendArrays

```
    subroutine extendArrays(tokens_)
        implicit none
        type(token), allocatable, intent(inout) :: tokens_(:)
        type(token), allocatable :: temp(:)
        integer :: i
        !extendemos el arreglo
        i = size(tokens_)
        allocate(temp(i+1))
        temp(1:i) = tokens_
        deallocate(tokens_)
        tokens_ = temp

    end subroutine extendArrays
end module moduleToken
```

Incrementa el tamaño del arreglo tokens_ para permitir agregar un nuevo elemento.
Entradas:
tokens_: El arreglo de tokens que debe ser extendido.
Funcionamiento:
Se crea un arreglo temporal temp con una longitud mayor en una unidad al arreglo original.
Se copian los datos del arreglo original en temp, luego se desasigna el arreglo original, y tokens_ es reasignado con el nuevo tamaño.
End module, cierra el modulo token

Modulo States

define un subprograma dentro de un módulo llamado states, que utiliza los módulos moduleToken, moduleError, y Utils para realizar operaciones relacionadas con el procesamiento de tokens y errores. La subrutina state0 parece ser parte de una máquina de estados finitos utilizada para analizar secuencias de caracteres.

```
module states
    use moduleToken
    use moduleError
    use Utils
    implicit none


    contains
    !stado 0
    subroutine state0(current_char,buffer_,tokens_,errors,linea,columna,estado,salto_linea)
    implicit none
    character(len=1), intent(in) :: current_char
    character(len=100), intent(inout) :: buffer_
    type(token), allocatable, intent(inout) :: tokens_(:)
    type(Error), allocatable, intent(inout) :: errors(:)
    integer, intent(inout) :: linea,columna
    integer, intent(inout) :: estado
    logical, intent(in) :: salto_linea
    !añadir el caracter al buffer
```

- Dependencias:
  Module token: para la gestión de tokens
  Module Error: para manejar errores
  Module utils: declara la variable.

  Entradas clave:
  current_char: Es el carácter que se está procesando en ese momento.
  buffer_: Almacena caracteres consecutivos para formar tokens.
  tokens_: Un arreglo dinámico donde se almacenan los tokens identificados.
  errors: Un arreglo dinámico donde se registran errores léxicos encontrados durante el análisis.
  linea y columna: Posicionan el carácter actual en el código fuente.
  estado: Representa el estado actual de la máquina de estados, que puede cambiar según el flujo de análisis.
  salto_linea: Indica si el carácter es un salto de línea, lo que podría afectar cómo se maneja el análisis.

- Subrutina addbuffer
- completa de la subrutina state0 en el módulo states, que procesa un carácter a la vez, verificando si el carácter corresponde a un token válido o si se debe registrar un error.

```
call addbuffer(current_char,buffer_,columna)
!verificar si es un caracter del estado 0
if(current_char == '=') then
    call addToken(tokens_,buffer_,'IGUAL',linea,columna)

else if(current_char == "(")then
    call addToken(tokens_,buffer_,"parentesis_abierto",linea,columna)

else if(current_char == ")")then
    call addToken(tokens_,buffer_,"parentesis_cerrado",linea,columna)

else if(current_char == "{")then
    call addToken(tokens_,buffer_,"llave_abierta",linea,columna)
!else if(current_char == "\n")then
    !call addToken(tokens_,buffer_,"llave_abierta",linea,columna)
else if(current_char == "}")then
    call addToken(tokens_,buffer_,"llave_cerrada",linea,columna)
else if(current_char == ":")then
    call addToken(tokens_,buffer_,"dos_puntos",linea,columna)
else if(current_char == "%")then
    call addToken(tokens_,buffer_,"porcentaje",linea,columna)
else if(current_char == ";")then
    call addToken(tokens_,buffer_,"punto_y_coma",linea,columna)
else if(isspecialChar(current_char))then
```

Verifica si el contenido que proviene del current char pertenece a elementos del estado inicial, si son elementos o tokens los agrega llamando al call addtoken, utilizando token, buffer, que tipo de carácter es, line y columna.

```
 !si no es ningun caracter valido
else
   ! print*, "caracter no valido", current_char

    call addError(errors, "Caracter no reconocido: " // current_char, buffer_, "LEXICO", linea, columna)
    call irastate(estado,0)
end if
!si es salto de linea
if(salto_linea)then
    call newLine(linea,columna)
    call clearBuffer(buffer_)
end if
```

Si se encuentra algún carácter no valido se agrega al add erros para su manejo y se cambia de estado.
If de salto de línea: cambiamos a una nueva línea new_Line.

- Subrutina Stado1
  parte del sistema de procesamiento de tokens que analiza una secuencia de caracteres para identificar números en particular. Este estado maneja la construcción de tokens numéricos y, cuando detecta que el siguiente carácter no es un número, genera un token de tipo numero y cambia el estado.

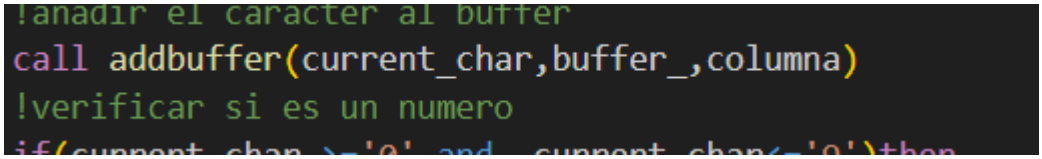
```
subroutine state1(current_char,buffer_,tokens_,linea,columna,estado,i)
    implicit none
    character(len=1), intent(in) :: current_char
    character(len=100), intent(inout) :: buffer_
    type(token), allocatable, intent(inout) :: tokens_(:)
    integer, intent(inout) :: linea,columna
    integer, intent(inout) :: estado
    integer, intent(inout) :: i
    !añadir el caracter al buffer
    call addbuffer(current_char,buffer_,columna)
    !verificar si es un numero
    if(current_char >='0'.and. current_char<='9')then
        !seguir en el estado 1
        call irastate(estado,1)
    else
        ! solo retroceder si es un caracter imprimeble
        if(isspecialChar(current_char).eqv. .false.)then
           call goBack(i,columna,buffer_)
        end if
        call addToken(tokens_,buffer_,"numero",linea,columna)
        !cambiar de estado
        call irastate(estado,0)
    end if


end subroutine state1
```

  Esta subrutina maneja el estado 1, que corresponde al procesamiento de números. Los números se acumulan en el buffer hasta que se encuentra un carácter que no es un número, momento en el cual se crea el token numérico y se regresa al estado inicial (estado 0).

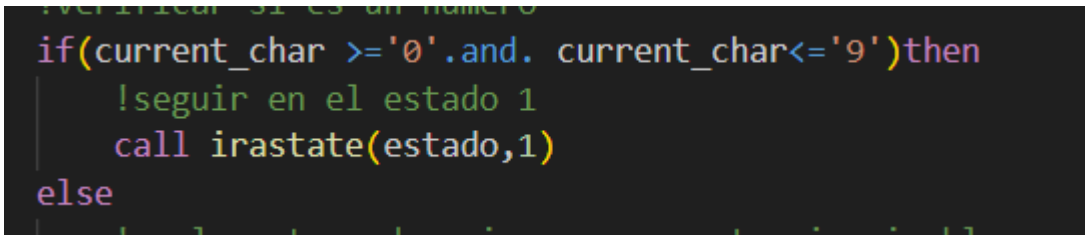
  Parámetros:
- **current_char**: Carácter actual a procesar.
- **buffer_**: Buffer temporal que contiene caracteres hasta formar un token (en este caso, un número).
- **tokens_**: Arreglo dinámico de tokens.
- **linea**: Número de línea actual.
- **columna**: Número de columna actual.
- **estado**: Estado actual de la máquina de estados (inicialmente 1 para números).

- **i**: Índice actual en la cadena de entrada.

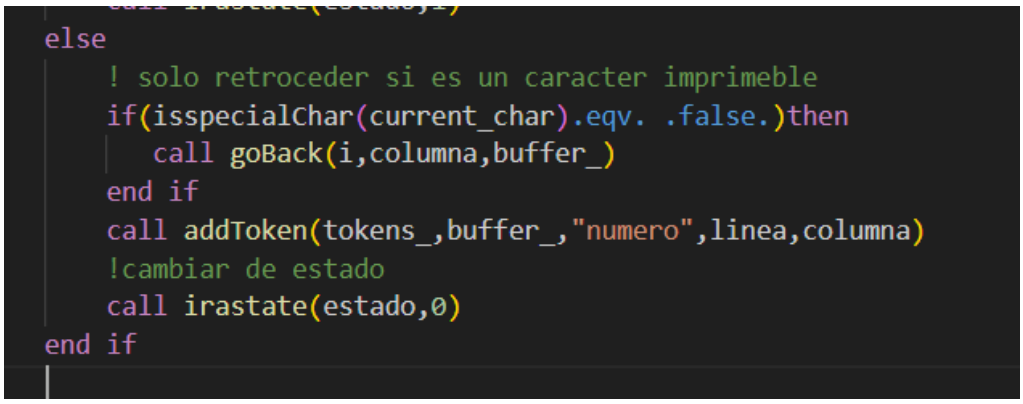
Call add buffer: añade el carácter actual al buffer para formar el token numérico.



Si el carácter es un número (0-9), el estado se mantiene en **estado 1**, lo que significa que se continúa acumulando caracteres para formar el número.



Si el carácter actual no es un número, el código primero verifica si no es un carácter especial usando la función isspecialChar(). Si no es un carácter especial, la función **goBack** se invoca para retroceder el índice (i) y la columna, volviendo a analizar el carácter anterior como parte del siguiente token.



- **Subrutina State 2**

  procesar secuencias de letras en la entrada para identificar si forman palabras reservadas o cadenas alfanuméricas. Esta subrutina maneja la creación de tokens que representan palabras reservadas o cadenas generales, y vuelve al estado 0 una vez que completa el procesamiento de un token.

Esta subrutina procesa caracteres alfabéticos y genera tokens de tipo Palabrareservada si el contenido del buffer coincide con una palabra reservada, o de tipo Cadena si no es una palabra reservada.

```
subroutine state2(current_char,buffer_,tokens_,linea,columna,estado,i)
    implicit none
    character(len=1), intent(in) :: current_char
    character(len=100), intent(inout) :: buffer_
    type(token), allocatable, intent(inout) :: tokens_(:)
    integer, intent(inout) :: linea,columna
    integer, intent(inout) :: estado
    integer, intent(inout) :: i
    character(len=100) :: Tokentype
```

current_char: Carácter actual a procesar.
buffer_: Buffer que acumula caracteres hasta formar un token (en este caso, una secuencia de letras).
tokens_: Arreglo dinámico de tokens.
linea: Número de línea actual.
columna: Número de columna actual.
estado: Estado actual de la máquina de estados (inicialmente 2 para letras).
i: Índice actual en la cadena de entrada.

- Se anañe el carácter actual del buffer

```
!anadir el caracter al buffer
call addbuffer(current_char,buffer_,columna)
!verificar si es una letra
if(current_char >='a'.and. current_char<='z' .or.&
    current_char >='A'.and. current_char<='Z')then
    !seguir en el estado 2
    call irastate(estado,2)
else
```

If: verifica si en el current char hay un carácter alfabético con minúsculas o mayúsculas, si es así se dirige al estado 2

- e verifica si el buffer contiene una palabra reservada. Si es así, se genera un token con el tipo **Palabra reservada**, y si no, se genera un token del tipo **Cadena**. Luego, se cambia el estado a 0 si es una cadena.

```
lse
   ! solo retroceder si es un caracter imprimeble
   if(isspecialChar(current_char).eqv. .false.)then
      call goBack(i,columna,buffer_)
   end if
   !verificar si es una palabra reservada
   if(esPalabraReservada(buffer_)=="Palabra reservada")then
      call addToken(tokens_,buffer_,"Palabra reservada",linea,columna)
   else
     call addToken(tokens_,buffer_,"Cadena",linea,columna)
```

- tokentype
  después, se llama de nuevo a esPalabraReservada(buffer_) y se asigna el resultado a Tokentype, para luego añadir de

```
            Tokentype = esPalabraReservada(buffer_)
            call addToken(tokens_,buffer_,Tokentype,linea,columna)
            !cambiar de estado
            call irastate(estado,0)
        end if
    end subroutine state2
    !estado 3
```

  nuevo el token con el valor de Tokentype.

- Subrutina estado 3
  maneja la detección de cadenas de texto que están delimitadas por comillas (").

```
    subroutine state3(current_char,buffer_,tokens_,linea,columna,estado)
       implicit none
       character(len=1), intent(in) :: current_char
         character(len=100), intent(inout) :: buffer_
            type(token), allocatable, intent(inout) :: tokens_(:)
            integer, intent(inout) :: linea,columna
            integer, intent(inout) :: estado
            !añadir el caracter al buffer
            call addbuffer(current_char,buffer_,columna)
            !verificar si es una comilla
            if(current_char == '"')then
                !añadir el token
                call addToken(tokens_,buffer_,"cadena",linea,columna)
                !cambiar de estado
                call irastate(estado,0)
            else
                !seguir en el estado 3
                call irastate(estado,3)
            end if
    end subroutine state3
```

se encarga de procesar las secuencias de caracteres entre comillas, considerándolas como una cadena de texto. La secuencia de caracteres se almacena en un buffer hasta que se encuentra la comilla de cierre ("), momento en el que se genera un token con el tipo "cadena".
Parámetros de entrada:
current_char (character(len=1)): El carácter actual que se está procesando.
buffer_ (character(len=100)): El buffer donde se almacenan temporalmente los caracteres que forman la cadena.
tokens_ (type(token), allocatable, intent(inout)): Arreglo dinámico que almacena los tokens que se van generando.
linea (integer, intent(inout)): El número de la línea actual en el archivo.
columna (integer, intent(inout)): El número de la columna actual en la línea.
estado (integer, intent(inout)): El estado actual del autómata que controla el análisis léxico.

Añadir el carácter al buffer: Cada carácter leído se agrega al buffer mediante la llamada a la subrutina addbuffer,

```
!añadir el caracter al buffer
call addbuffer(current_char,buffer_,columna)
!verificar si es una comilla
if(current_char == '"')then
```

Verificación de comillas: Se comprueba si el carácter actual es una comilla ("). Si es así, se considera que la cadena ha terminado.

```
if(current_char == '"')then
    !añadir el token
    call addToken(tokens_,buffer_,"cadena",linea,columna)
    !cambiar de estado
    call irastate(estado,0)
else
    !seguir en el estado 3
```

Generación del token: Si se encuentra la comilla de cierre, el contenido del buffer se convierte en un token del tipo "cadena". Este token se añade al arreglo de tokens mediante la subrutina addToken.
Cambio de estado: Después de procesar la cadena completa, el estado del autómata se regresa a 0 para volver al estado inicial y continuar con el análisis de otros tipos de tokens.

```
        else
            !seguir en el estado 3
            call irastate(estado,3)
        end if
nd subroutine state3
```

Continuar en estado 3: Si no se ha encontrado la comilla de cierre, el autómata permanece en el estado 3 para seguir procesando los caracteres de la cadena.
End subroutine ; cierra la subrutine stado 3.

- Module Utils
  La subrutina addbuffer se encarga de añadir un carácter al final del buffer, que es una cadena de texto, y actualizar la posición de la columna en función de la longitud del buffer.

```
module Utils

contains
!añadir un caracter al buffer
    subroutine addbuffer(current_char,buffer_,columna)
    implicit none
    character(len=1), intent(in) :: current_char
    character(len=100), intent(inout) :: buffer_
    integer, intent(inout) :: columna
    !añadir el caracter al buffer
    buffer_ = trim(buffer_)//current_char
    columna = columna + 1

    end subroutine addbuffer
```

La subrutina recibe un carácter y lo añade al final de un buffer que almacena temporalmente una secuencia de caracteres. A su vez, incrementa el valor de la columna para reflejar la posición actual del carácter en el análisis.
Parámetros de entrada:
current_char (character(len=1)): El carácter que se va a agregar al buffer.
buffer_ (character(len=100)): El buffer donde se almacenan temporalmente los caracteres. Es modificado para incluir el nuevo carácter.

columna (integer, intent(inout)): La posición actual de la columna en la línea. Se incrementa con cada carácter añadido.
-concatenacion de caracteres:

```
buffer_ = trim(buffer_)//current_char
columna = columna + 1
```

-Aumento en el contador de columna:

```
buffer_ = trim(buffer_)//
columna = columna + 1

end subroutine addbuffer
```

- Función isspecialchar.
  La función isspecialChar determina si un carácter es considerado un carácter especial. En este caso, los caracteres especiales incluyen espacios en blanco, tabuladores, saltos de línea y otros caracteres de control.

```
function isspecialChar(current_char) result(special)
 implicit none
 character(len=1), intent(in) :: current_char
 logical :: special
 !verificar si es un caracter especial
 if(current_char == " " .or. current_char == "\t" .or. &
    current_char == "\r" .or. current_char== "\f" .or. &
    current_char == "\0" .or. current_char== "\s".or.current_char=="\n"&
    .or. current_char==""    ".or.current_char==char(9))then
    special = .true.
    !print*, "caracter especialjolasod", current_char
    else
  special = .false.
 end if
 end function isspecialChar
```

Esta función evalúa si el carácter de entrada es un carácter especial, como espacios en blanco, tabuladores, o saltos de línea. Retorna un valor lógico: true si el carácter es especial y false si no lo es.

- Subrutina ClearBuffer
  La subrutina **clearBuffer** limpia el contenido del buffer, es decir, reinicia el valor de la variable de tipo cadena (string) que almacena temporalmente datos como tokens o caracteres en el proceso de análisis léxico.

```
!limpiar el buffer
    subroutine clearBuffer(buffer)
    implicit none
    character(len=100), intent(inout) :: buffer
    buffer = ""
    end subroutine clearBuffer
```

La subrutina simplemente establece el valor del buffer a una cadena vacía (""), eliminando cualquier contenido que pueda tener.

- Subrutina Irastate

  La subrutina irastate cambia el valor del estado actual de una máquina de estados finitos, pasando del estado actual (estado) al siguiente estado (nex_estado). Es utilizada para controlar la transición entre diferentes estados.

```
subroutine irastate(estado,nex_estado)
    implicit none
    integer, intent(inout) :: estado
    integer, intent(in) :: nex_estado
    !cambiar de estado
    estado = nex_estado
    end subroutine irastate
```

  estado (integer, intent(inout)): Representa el estado actual de la máquina de estados. Se pasa como argumento de entrada/salida, lo que significa que se modifica dentro de la subrutina.
  nex_estado (integer, intent(in)): Representa el nuevo estado al cual se desea cambiar. Es un valor de entrada que indica el siguiente estado de la máquina.

- Subrutina newline

  La subrutina newLine se utiliza para gestionar los saltos de línea en un archivo o flujo de entrada. Cuando se encuentra un salto de línea (\n), la subrutina incrementa el número de línea y reinicia el contador de columnas a 1.

```
subroutine newLine(linea,columna)
    implicit none
    integer, intent(inout) :: linea,columna

    linea = linea + 1
    columna = 1
end subroutine newLine
```

- Funcion esPalabraReservada
  La función esPalabraReservada verifica si el contenido del buffer (una cadena de caracteres) coincide con una de las palabras clave predefinidas (como Grafica, Nombre, Continente, etc.) que se consideran reservadas.

```
function esPalabraReservada(buffer) result(reservada)
    implicit none
    character(len=*), intent(in) :: buffer
    character(len=100) :: reservada
    if (buffer=='Grafica'.or. buffer=='grafica'.or.buffer=='Nombre'.or.buffer=='nombre'.or.&
    buffer=='Continente'.or. buffer=='continente'.or. buffer=='pais'.or.buffer=='Pais'.or.&
    buffer=='poblacion'.or. buffer=='Poblacion'.or.buffer=='Saturacion'.or.buffer=='Bandera') then
    reservada = 'reservada'

    else if(buffer=='')then
        reservada='espacio vacio'
    else
        reservada='identificador'

    end if
    end function esPalabraReservada
```

  buffer (character(len=*), intent(in)): Es la cadena que contiene el texto a evaluar. Representa un potencial token que puede ser una palabra reservada, un identificador, o un espacio vacío.
  reservada (character(len=100)): Es la cadena que devuelve el tipo de token encontrado. Puede tener los siguientes valores:
  'reservada': Si el contenido del buffer es una palabra reservada.
  'espacio vacio': Si el buffer está vacío.
  'identificador': Si no es ni una palabra reservada ni un espacio vacío, entonces se considera un identificador.

  La primera condición (if) verifica si el buffer coincide con una lista predefinida de palabras reservadas, como 'Grafica', 'Nombre', 'Continente', etc. Esta comparación se realiza para ambas versiones, mayúsculas y minúsculas.

```
if (buffer=='Grafica'.or. buffer=='grafica'.or.buffer=='Nombre'.or.buffer=='nombre'.or.&
buffer=='Continente'.or. buffer=='continente'.or. buffer=='pais'.or.buffer=='Pais'.or.&
buffer=='poblacion'.or. buffer=='Poblacion'.or.buffer=='Saturacion'.or.buffer=='Bandera') then
reservada = 'reservada'

else if(buffer=='')then
    reservada='espacio vacio'
else
    reservada='identificador'

end if
```

- Subrutina Goback
  La subrutina goBack se utiliza para retroceder una posición en el índice de análisis y actualizar tanto el índice como la columna actual. Además, elimina el último carácter añadido al buffer para reflejar el retroceso en el flujo de análisis léxico.

```
subroutine goBack(i, columna, buffer_)
    implicit none
    integer, intent(inout) :: i
    integer, intent(inout) :: columna
    character(len=100), intent(inout) :: buffer_
    integer :: actual_length

    i = i - 1
    columna = columna - 1

    ! Obtén la longitud real del contenido de buffer_ sin los espacios en blanco finales
    actual_length = len_trim(buffer_)
    if (actual_length > 0) then
        buffer_ = buffer_(:actual_length-1)
    endif
end subroutine goBack
```

i (integer, intent(inout)): Representa el índice actual de la posición en el texto o cadena que se está analizando. Se decrementa en 1 al retroceder.
columna (integer, intent(inout)): Contador que rastrea la posición de la columna en el archivo o línea que se analiza. También se decrementa en 1 al retroceder.
buffer_ (character(len=100), intent(inout)): El buffer que almacena el texto que está siendo analizado o acumulado. Se elimina el último carácter del buffer como parte del retroceso.

```
actual_length = len_trim(buffer_)
if (actual_length > 0) then
    buffer_ = buffer_(:actual_length-1)
endif
```

len_trim(buffer_): Se utiliza para obtener la longitud del buffer sin los espacios en blanco finales. Esto permite evitar errores al manipular cadenas que podrían contener espacios adicionales no deseados.
Eliminación del último carácter: Si el buffer tiene longitud mayor que 0, se actualiza eliminando el último carácter con buffer_ = buffer_(:actual_length-1).

- Module Archivo_mod
  Este modulo se utiliza para la modificación de los datos de entrada.

```
module archivo_mod
    use graficaarbol
    implicit none
contains
```

La subrutina procesar_archivo que estás implementando abre un archivo de entrada y otro de salida, y tiene como propósito procesar el contenido del archivo de entrada, posiblemente relacionado con el manejo de datos de países, gráficos, y otros atributos, y escribir los resultados en el archivo de salida.

```
subroutine procesar_archivo(archivo_entrada, archivo_salida)
    character(len=*), intent(in) :: archivo_entrada, archivo_salida
    character(len=256) :: linea
    integer :: ios
    character(len=256) :: nombre, continente, pais, poblacion, saturacion, bandera
    logical :: pais_abierto

    pais_abierto = .false.

    ! Abrir el archivo de entrada
    open(unit=10, file=archivo_entrada, status='old', action='read', iostat=ios)
    if (ios /= 0) then
        print *, 'Error al abrir el archivo de entrada'
        stop
    end if
```

archivo_entrada y archivo_salida: Son los nombres de los archivos de entrada y salida pasados como parámetros.
linea: Una cadena de caracteres que almacenará cada línea leída del archivo de entrada.
nombre, continente, pais, poblacion, saturacion, bandera: Estas cadenas de caracteres almacenan los diferentes atributos que se encontrarán en el archivo de entrada (como el nombre de un país, su población, etc.).
pais_abierto: Una variable lógica (boolean) que indica si actualmente se está procesando un bloque de datos de un país. Inicialmente está en false, lo que significa que no se está procesando un bloque país.
El archivo de entrada se abre en la unidad 10 para lectura (status='old'), lo que indica que el archivo debe existir.
Si ocurre un error al abrir el archivo de entrada (indicado por el valor diferente de 0 en ios), se imprime un mensaje de error y se detiene el programa.

```
    ! Leer y escribir el archivo línea por línea
    do
        read(10, '(A)', iostat=ios) linea
        if (ios /= 0) exit

        ! Procesar la línea para extraer datos y escribir en el archivo de salida
    if (index(linea, 'Grafica:{') > 0) then
        write(20, '(A)') trim(linea)
        ! Leer el nombre de la gráfica
        read(10, '(A)', iostat=ios) linea
        call extraer_valor(linea, nombre)
        write(20, '(A)') 'Nombre: ' // trim(nombre) // ''
    else if (index(linea, 'Continente:{') > 0) then
        write(20, '(A)') trim(linea)
        ! Leer el nombre del continente
        read(10, '(A)', iostat=ios) linea
        call extraer_valor(linea, continente)
        write(20, '(A)') '    Nombre: ' // trim(continente) // ';'
```

Se abre el archivo de salida en la unidad 20 con la opción status='replace' para que el archivo se reemplace si ya existe. Si ocurre un error al abrirlo, se imprime un mensaje de error y se detiene el programa.
Leer y procesar línea por línea:

Se lee cada línea del archivo de entrada con el formato (A) que indica que se trata de una cadena de caracteres. Si hay un error de entrada o si se llega al final del archivo (cuando ios es diferente de 0), se sale del bucle.
Condiciones para procesar bloques específicos:

Grafica: Si la línea contiene la palabra clave Grafica:{, se escribe en el archivo de salida y se lee la siguiente línea para extraer el nombre de la gráfica.

Continente: Si la línea contiene Continente:{, se escribe la línea en el archivo de salida y se lee la siguiente línea para extraer el nombre del continente.

Pais: Si la línea contiene Pais:{, indica que se ha abierto un bloque de país. Se inicializan las variables del país (pais, poblacion, saturacion, bandera) para almacenarlas cuando se encuentren.

```
else if (index(linea, 'Pais:{') > 0) then
    write(20, '(A)') trim(linea)
    pais_abierto = .true.
    ! Inicializar variables del país
    pais = ''
    poblacion = ''
    saturacion = ''
    bandera = ''
else if (index(linea, 'Nombre:') > 0 .and. pais_abierto) then
    call extraer_valor(linea, pais)
else if (index(linea, 'Poblacion:') > 0 .and. pais_abierto) then
    call extraer_valor(linea, poblacion)
else if (index(linea, 'Saturacion:') > 0 .and. pais_abierto) then
    call extraer_valor(linea, saturacion)
else if (index(linea, 'Bandera:') > 0 .and. pais_abierto) then
    call extraer_valor(linea, bandera)
else if (index(linea, '}') > 0 .and. pais_abierto) then
    ! Escribir los datos del país en el orden correcto
    !write(20, '(A)') '    Pais:{'
```

Cuando se encuentra la llave de cierre (}) en el bloque de país, el código escribe los datos del país en el archivo de salida y cierra el bloque de país. Luego, el valor de pais_abierto se pone en false para indicar que ya no se está procesando un país.

```
        else
            write(20, '(A)') trim(linea)
            end if
        end do

        ! Cerrar los archivos
        close(10)
        close(20)
    call graficaarbo(archivo_salida)
```

Después de procesar el archivo de salida, se llama a la subrutina graficaarbo, que probablemente genera una gráfica o algún tipo de procesamiento adicional basado en el archivo de salida.

La subrutina extraer_valor que incluyes dentro del módulo archivo_mod tiene la función de extraer el valor asociado a una etiqueta (como "Nombre", "Poblacion", etc.)

```
        subroutine extraer_valor(linea, valor)
            character(len=*), intent(in) :: linea
            character(len=256), intent(out) :: valor
            integer :: pos

            pos = index(linea, ':') + 1
            valor = adjustl(trim(linea(pos:)))
        end subroutine extraer_valor

    end subroutine procesar_archivo

end module archivo_mod
```

linea (entrada): es una cadena que contiene una línea completa del archivo, la cual sigue el formato Etiqueta: Valor.
valor (salida): es una cadena en la que se almacenará el valor extraído de la línea.
pos = index(linea, ':') + 1:
La función index busca el primer carácter: en la línea.
valor = adjustl(trim(linea(pos:))):
trim elimina cualquier espacio en blanco adicional al final de la cadena.
adjustl ajusta el contenido de la cadena a la izquierda, es decir, elimina espacios en blanco al principio del valor, si los hubiera.

- Module Grafica
  Este modulo su finalidad es realizar la grafica de árbol segun el contenido de los archivos analizados, utilizando grapihviz.

```
module graficaarbol
    implicit none
    contains
    subroutine graficaarbo(entrada_archivo)

            implicit none
            character(len=200) :: linea
            character(len=50) :: nombre_grafica, continente, pais, pais_min_saturacion
            character(len=50) :: pais_menor_global, continente_menor_global
            character(len=100) :: bandera, bandera_min_saturacion, bandera_menor_global
            integer :: poblacion, poblacion_menor_global, poblacion_min_saturacion, end_of_file, i, num_paises
            character(len=10) :: saturacion
            logical :: leyendo_continente, leyendo_pais, leyendo_grafica
            real :: suma_saturacion, promedio_saturacion, min_saturacion
            real :: min_saturacion_global, promedio_saturacion_global
            integer :: saturacion_val, promedio_saturacion_entero
            integer :: salida_unit
            integer :: ios
            character(len=100) :: dot_filename, command
            character(len=20) :: cadens
            character(len=20) :: cadena
            character(len=20) :: entrada_archivo
            character(len=20) :: nombre_imagens
            dot_filename="graficaultima3.dot"
            !entrada_archivo="Entrada2.ORG"
            salida_unit = 30
```

Se declaran las variables que se utilizaran para este modulo y la subrutina graficaarbol

Aquí se describe un fragmento de código que lee un archivo de entrada línea por línea, buscando datos específicos para generar un archivo de salida y crear una estructura en formato DOT (usado comúnmente en Graphviz para generar gráficos).

```
! Abrir archivo de salida
open(unit=20, file="SalidaDavid.txt", status="replace", action="write")
!crear archivo .dot
open(unit=11, file=dot_filename, status="replace",iostat=ios)
if (ios /= 0) then
    print *, "Error al abrir el archivo."
    stop
end if
open(unit=10, file=entrada_archivo, status="old", action="read")

! Leer el archivo línea por línea
do
   read(10, '(A)', iostat=end_of_file) linea
   if (end_of_file /= 0) exit

   ! Buscar el inicio de la gráfica
   if (index(linea, 'Grafica:{') /= 0) then
      leyendo_grafica = .true.
   end if

   ! Detectar y leer el nombre de la gráfica
   if (index(linea, 'Nombre:') /= 0 .and. leyendo_grafica) then
      call extraer_valor(linea, nombre_grafica)
      write(11, '(A)') 'digraph G {'
      write(11, '(A)') '  "' // trim(nombre_grafica) // '" [shape=box];'

      write(salida_unit, *) "Nombre de la gráfica: ", nombre_grafica
      leyendo_grafica = .false.
   end if
```

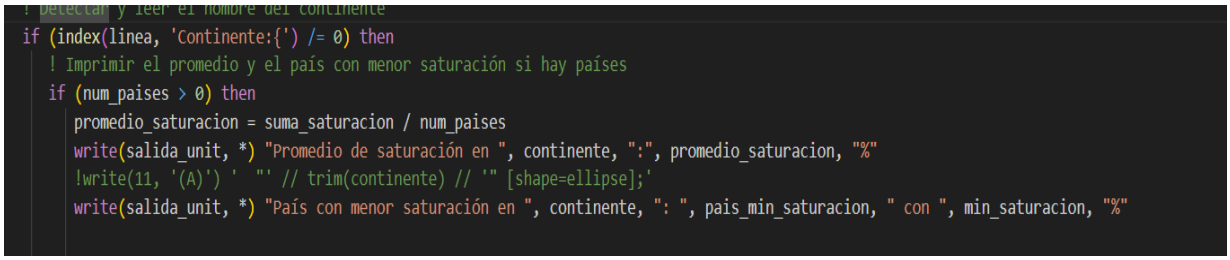
Archivo de salida textual: Se abre el archivo SalidaDavid.txt en el unit 20 con la opción "replace", lo que significa que si el archivo existe, será reemplazado.
Archivo DOT: Se abre el archivo dot_filename, que contiene el nombre graficaultima3.dot, en el unit 11. Este archivo es donde se escribirá la estructura gráfica que luego puede ser interpretada por herramientas como Graphviz.
Archivo de entrada: Se abre el archivo de entrada en el unit 10, que contiene los datos estructurados en bloques (gráfica, continente, país, etc.).

- Generación del documento dot

```
! Detectar y leer el nombre de la gráfica
if (index(linea, 'Nombre:') /= 0 .and. leyendo_grafica) then
   call extraer_valor(linea, nombre_grafica)
   write(11, '(A)') 'digraph G {'
   write(11, '(A)') '  "' // trim(nombre_grafica) // '" [shape=box];'

   write(salida_unit, *) "Nombre de la gráfica: ", nombre_grafica
   leyendo_grafica = .false.
end if
```

- Detección del bloque 'Continente:{':

Si se encuentra la línea 'Continente:{', el código realiza varias operaciones antes de comenzar a procesar un nuevo continente.

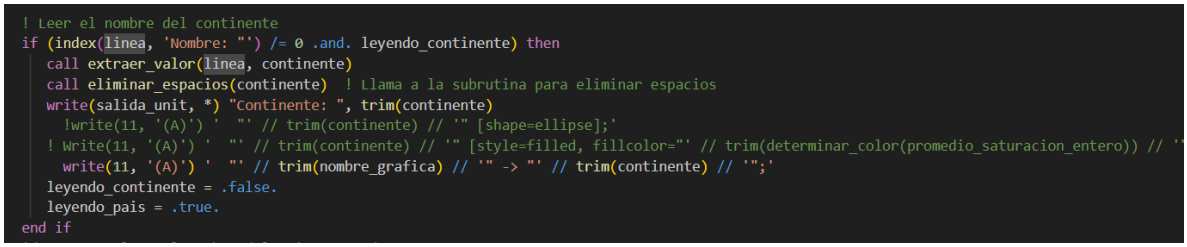


Promedio de saturación: Si se han procesado países (es decir, num_paises > 0), se calcula el promedio de saturación dividiendo la suma total de saturación por el número de países:

Detección del nombre del continente:

El bloque de código está diseñado para detectar líneas que contienen el texto 'Nombre: "' cuando se está dentro de un bloque relacionado con un continente (controlado por la variable lógica leyendo_continente).Si se encuentra este patrón en la línea, se ejecutan varias acciones.Extracción del valor del nombre del continente:

Llama a la subrutina extraer_valor para obtener el nombre del continente desde la línea. Esta subrutina toma el valor después de los dos puntos (:) en la línea y lo almacena en la variable continente:



Llama a la subrutina extraer_valor para obtener el nombre del continente desde la línea. Esta subrutina toma el valor después de los dos puntos (:) en la línea y lo almacena en la variable continente

Después de extraer el valor, se llama a la subrutina eliminar_espacios para eliminar cualquier espacio adicional o innecesario en la cadena continente

La subrutina eliminar_espacios probablemente elimina los espacios en blanco extra antes y después del nombre del continente.

Generación de relaciones gráficas (archivo DOT):

Se escribe una relación entre el nombre de la gráfica y el continente en el archivo DOT. Este archivo se utiliza para generar representaciones gráficas de la información, en este caso, se está creando una relación tipo flecha desde la gráfica (nombre_grafica) al continente

write(11, '(A)') ' "' // trim(nombre_grafica) // '" -> "' // trim(continente) // '";'

```
if (index(linea, 'Pais:{') /= 0) then
 read(10, '(A)', iostat=end_of_file) linea
 call extraer_valor(linea, pais)
 write(salida_unit, *) "   País: ", pais


 ! Leer población
 read(10, '(A)', iostat=end_of_file) linea
 call limpiar_linea(linea)
 call extraer_entero(linea, poblacion)
 write(salida_unit, *) "      Población: ", poblacion

 ! Leer saturación
 read(10, '(A)', iostat=end_of_file) linea
 call limpiar_linea(linea)
 call extraer_saturacion(linea, saturacion)
 write(salida_unit, *) "      Saturación: ", saturacion, "%"
```

La sentencia if la utilizaremos para tomar e identificar el contenido del bloque país

Lee línea por línea encontrando los valores de población y de saturación. En cada bloque se reinicia y limpia el contenido de la línea.

```
! Convertir saturación a entero y acumular
read(saturacion, *) saturacion_val
suma_saturacion = suma_saturacion + saturacion_val
num_paises = num_paises + 1

! Determinar el color según la saturación
select case (saturacion_val)
    case (0:15)
        write(salida_unit, *) "White"
    case (16:30)
        write(salida_unit, *) "Blue"
      case (31:45)
          write(salida_unit, *) "green"
      case (46:60)
          write(salida_unit, *) "yellow"
    case (61:75)
        write(salida_unit, *) "orange"
    case (76:100)
        write(salida_unit, *) "red"
    case default
        write(salida_unit, *) "black"
end select
```

Según la saturación se devolverá un color designado, el cual lo obtenemos según el rango utilizando select case(saturación). Cada país obtendrá un color según su saturación.

```
! Leer bandera
read(10, '(A)', iostat=end_of_file) linea
call limpiar_linea(linea)
call extraer_valor(linea, bandera)
write(salida_unit, *) "      Bandera: ", bandera

! Actualizar el país con la menor saturación en el continente
if (saturacion_val < min_saturacion) then
   min_saturacion = saturacion_val
   pais_min_saturacion = pais
   poblacion_min_saturacion = poblacion
   bandera_min_saturacion = bandera   ! Ahora la bandera está correctamente asignada
end if
write(11, '(A)') '  "' // trim(pais) // '" [style=filled, fillcolor="' // trim(determinar_color(saturacion_val)) // '"];'
write(cadena, '(I0)')  saturacion_val
write(11, '(A)') '  "' // trim(pais) // '" [shape=record, label="' // trim(pais) // '|' // trim(cadena) // '"];'
  write(11, '(A)') '  "' // trim(continente) // '" -> "' // trim(pais) // '";'
end if
```

Lectura de la línea al reconocer la bandera, se extrae el valor de bandera.

If saturación se utiliza para obtener la saturación para el país con menor valor en saturación.

```
   end if
   write(11, '(A)') '  "' // trim(pais) // '" [style=filled, fillcolor="' // trim(determinar_color(saturacion_val)) // '"];'
   write(cadena, '(I0)')  saturacion_val
   write(11, '(A)') '  "' // trim(pais) // '" [shape=record, label="' // trim(pais) // '|' // trim(cadena) // '"];'
     write(11, '(A)') '  "' // trim(continente) // '" -> "' // trim(pais) // '";'
end if
```

Aquí se escribe los resultados y de color para cada país para la generación del documento dot.

```
! Calcular y mostrar promedio para el último continente
if (num_paises > 0) then
   promedio_saturacion = suma_saturacion / num_paises
   write(salida_unit, *) "Promedio de saturación en ", continente, ":", promedio_saturacion, "%"
   write(salida_unit, *) "País con menor saturación en ", continente, ": ", pais_min_saturacion, " con ", min_saturacion, "%"

   ! Convertir el promedio de saturación a entero para determinar el color
   promedio_saturacion_entero = nint(promedio_saturacion)
```

Esta línea verifica que haya al menos un país en el continente antes de proceder con el cálculo. Si num_paises es mayor que 0, se calcula el promedio; de lo contrario, no tiene sentido realizar los cálculos.

```
   write(salida_unit, *) "Color del promedio de saturación en ", continente, ": ", determinar_color(promedio_saturacion_entero)
   write(cadens, '(I0)')  promedio_saturacion_entero
write(11, '(A)') '  "' // trim(continente) // '" [shape=record, label="' // trim(continente) // '|' // trim(cadens) // '"];'
Write(11, '(A)') '  "' // trim(continente) // '" [style=filled, fillcolor="' // trim(determinar_color(promedio_saturacion_entero)) // '"];'
```

Continuación para seguir formando el documento dot.

```
  ! Imprimir país con menor saturación global
! write(20, *) "----------------------------------------"
  write(20, *) "Pais con menor saturacion global: ", pais_menor_global
  print*, "Pais con menor saturacion global: ", pais_menor_global
  write(20, *) "Continente: ", continente_menor_global
  print*, "Continente: ", continente_menor_global
  write(20, *) "Poblacion: ", poblacion_menor_global
  print*, "Poblacion: ", poblacion_menor_global
  write(20, *) "Bandera: ", bandera_menor_global
  print*, "Bandera: ", bandera_menor_global
  write(20, *) "Saturacion: ", min_saturacion_global, "%"
  print*, "Saturacion: ", min_saturacion_global
 ! write(20, *) "----------------------------------------"

  close(10)
  write(11, '(A)') '}'
  close(11)
  close(20)
  !print*,'archivo generado',dot_filename
  !preparar el comando para generar la imagen
  command = 'dot -Tpng ' // trim(dot_filename) // ' -o graficoultimo.png'
  !ejecutar el comando
  ios=system(command)
```

Estos resultados nos servirán para la salida de la conexión con phyton, los prints.

Los write escriben sobre documentos de salida.

- Comannd:

```
  !preparar el comando para generar la imagen
  command = 'dot -Tpng ' // trim(dot_filename) // ' -o graficoultimo.png'
  !ejecutar el comando
  ios=system(command)
  if (ios /= 0) then
      print *, "Error al ejecutar el comando."
      stop
  end if
  !print*, "Archivo DOT generado exitosamente: graficoultimo.dot"
  !close(salida_unit)
```

Comannd utilizado para la generacion del dot y la imagen con .png.
Si ios es distinto a 0 no se a podido ejecutar el comando.

- Subrutina extraer valor

se utiliza para extraer un valor contenido entre comillas (") dentro de una línea de texto. Es útil cuando se quiere obtener información específica que está delimitada por comillas dobles, como nombres de países, continentes o gráficos.

```
! Subrutina para extraer valores entre comillas
subroutine extraer_valor(linea, valor)
    character(len=*), intent(in) :: linea
    character(len=*), intent(out) :: valor
    integer :: inicio, fin

    inicio = index(linea, '"') + 1
    fin = index(linea(inicio+1:), '"') + inicio
    valor = linea(inicio:fin-1)
end subroutine extraer_valor
```

- Subrutina extraer entero

```
! Subrutina para extraer enteros
subroutine extraer_entero(linea, valor)
    character(len=*), intent(in) :: linea
    integer, intent(out) :: valor
    integer :: pos
    character(len=50) :: sub_linea

    pos = index(linea, ':') + 1
    sub_linea = linea(pos:)  ! Extraemos la subcadena que contiene el número
    call limpiar_linea(sub_linea)  ! Limpiar la línea por si tiene caracteres extraños
    read(sub_linea, *) valor  ! Leer el valor entero desde la subcadena
end subroutine extraer_entero
```

está diseñada para extraer un valor entero que aparece después de dos puntos (:) en una línea de texto. Después de localizar los dos puntos, la subrutina limpia la subcadena restante para asegurarse de que solo contiene el valor entero, y luego lo convierte a un número.

linea: cadena de entrada donde se buscará el valor entero.

valor: variable de salida donde se almacenará el valor entero extraído.

pos: posición del carácter de interés (en este caso, los dos puntos).

sub_linea: subcadena que contendrá la parte de la línea después de los dos puntos.

- Subrutina saturación

```
subroutine extraer_saturacion(linea, valor)
    character(len=*), intent(in) :: linea
    character(len=*), intent(out) :: valor
    integer :: pos

    pos = index(linea, ':') + 1
    valor = adjustl(linea(pos:))  ! Extraer el valor de saturación como texto
    ! Eliminar el símbolo % si está presente
    if (index(valor, '%') > 0) then
        valor = trim(adjustl(valor(1:index(valor, '%')-1)))
    end if
end subroutine extraer_saturacion
```

  tiene como objetivo extraer un valor de saturación desde una línea de texto que sigue el formato Saturacion: X%, donde X es un número y puede tener un símbolo de porcentaje (%) al final. La subrutina ajusta la cadena para eliminar los espacios y, si encuentra el símbolo %, lo elimina.

pos: busca la posición de los dos puntos (:) usando la función index, y luego suma 1 para apuntar al primer carácter después de los dos puntos, donde se espera que comience el valor de saturación.

valor: se extrae la parte de la línea que está después de los dos puntos (:). La función adjustl ajusta el valor para que elimine cualquier espacio en blanco a la izquierda de la cadena.

Eliminación del símbolo %:

Se utiliza index para buscar si hay un símbolo de porcentaje (%) en la subcadena valor.

Si se encuentra el símbolo %, se extrae la parte de la cadena antes del símbolo, eliminando también los espacios sobrantes con trim y adjustl.

- Subrutina limpiar línea

```
subroutine limpiar_linea(linea)
    character(len=*), intent(inout) :: linea
    integer :: i

    do i = 1, len_trim(linea)
        if (linea(i:i) == ';') linea(i:i) = ' '  ! Reemplazar punto y coma por espacio
    end do
    linea = adjustl(linea)  ! Ajustar espacios
end subroutine limpiar_linea
```

  tiene como objetivo limpiar una línea de texto al reemplazar ciertos caracteres y ajustar el formato de la cadena. En este caso, reemplaza los puntos y comas (;) por espacios y elimina los espacios en blanco a la izquierda de la cadena.

- Subrutina eliminar espacios

```
subroutine eliminar_espacios(linea)
    character(len=*), intent(inout) :: linea
    character(len=200) :: sin_espacios
    integer :: i, j

    j = 1
    do i = 1, len(linea)
        if (linea(i:i) /= ' ') then
            sin_espacios(j:j) = linea(i:i)
            j = j + 1
        end if
    end do
    sin_espacios(j:) = ''  ! Termina la cadena
    linea = sin_espacios  ! Actualiza la línea original
end subroutine eliminar_espacios
```

tiene como objetivo eliminar todos los espacios de una línea de texto, dejando solo los caracteres significativos.
Se inicializa j en 1, que se utilizará para rastrear la posición actual en la cadena sin_espacios.
Utiliza un bucle do que itera sobre cada carácter de linea usando el índice i desde 1 hasta la longitud de la línea (len(linea)).
Dentro del bucle, comprueba si el carácter actual (linea(i:i)) no es un espacio (' '). Si no lo es, copia ese carácter a la posición actual de sin_espacios (sin_espacios(j:j)) y luego incrementa j.

- Función determinar color

```
function determinar_color(saturacion_val) result(color)
    integer, intent(in) :: saturacion_val
    character(len=20) :: color

    select case (saturacion_val)
        case (0:15)
            color = "white"
        case (16:30)
            color = "blue"
        case (31:45)
```

```
            color = "green"
        case (46:60)
            color = "yellow"
        case (61:75)
            color = "orange"
        case (76:100)
            color = "red"
        case default
            color = "No definido"
    end select
end function determinar_color
```

La función determinar_color proporciona una forma eficiente de categorizar valores de saturación en colores. Esto puede ser útil en visualizaciones, como en gráficos donde diferentes rangos de saturación se representan con distintos colores, facilitando la interpretación visual de los datos.

- Subrutina lectura.

```
subroutine lectura(archivo)
    implicit none
    character(len=100) :: archivo
    character(len=200) :: lineas
    integer :: ios
    open(unit=40, file=archivo, status="old", action="read", iostat=ios)
    if (ios /= 0) then
        print *, "Error al abrir el archivo."
        stop
    end if
    do
        read(40, '(A)', iostat=ios) lineas
        if (ios /= 0) exit
        print *, lineas
    end do
    close(40)
end subroutine lectura
```

está diseñada para leer un archivo línea por línea y mostrar su contenido en la salida estándar. A continuación, se presenta una explicación detallada del código.
Se utiliza un bucle do para leer el archivo línea por línea.
read(40, '(A)', iostat=ios) lineas: Lee una línea del archivo y la almacena en lineas. La especificación '(A)' indica que se trata de una lectura de caracteres.
Si ios no es 0 (indica un error o el final del archivo), se sale del bucle.
Si la lectura es exitosa, se imprime la línea leída.

- Interfaz grafica

```
from tkinter import *
import tkinter as tk
from tkinter import ttk
from tkinter import scrolledtext
from tkinter import messagebox
from tkinter import PhotoImage
from PIL import Image, ImageTk
import subprocess
```

Se importan todos los elementos a usar para nuestra interfaz grafica.
Importando todo el paquete. También dando algunas etiquetas en la importación de los paquetes tkinter.

```
ventana = Tk()
archivo_abierto = None
considencia=False
ventana.title("Programa de Interfaz Grafica")
ventana.geometry("1200x900")
ventana.config(bg="lightblue")
frame=ttk.Frame(ventana)
frame.pack(fill="both", expand=True)
Scrollbar=ttk.Scrollbar(frame, orient="vertical")
Scrollbar.pack(side="right", fill="y")
etq1 = Label(ventana, text="ANALIZADAR DE MERCADO")
etq1.pack()
etq1.place(x=450, y=10)
etq5=Label(ventana, text="hollaaaaaaaaaaaaaaaaaaaaaaaaaaa.")
etq5.pack()
#ubicar el label en la ventana
etq5.place(x=450, y=1500)
```

Crear la Ventana Principal:

- Tk() inicializa una nueva ventana de la aplicación.

Variables Iniciales:

- archivo_abierto: Se inicializa como None, probablemente para más tarde almacenar el archivo que se abrirá o se procesará en la aplicación.
- considencia: Inicializada como False, puede ser utilizada como un indicador de estado en la aplicación

Título de la Ventana:

- title(...) establece el texto que aparecerá en la barra de título de la ventana.

Tamaño de la Ventana:

- geometry(...) define las dimensiones iniciales de la ventana en píxeles (1200 de ancho y 900 de alto).

Creación de un Frame:

- ttk.Frame(...) crea un marco (frame) dentro de la ventana. Los frames son útiles para organizar los widgets.
- pack(...) es un método de gestión de geometría que coloca el frame en la ventana, llenándolo en ambas direcciones (fill="both") y expandiéndolo para usar el espacio disponible (expand=True).
- Barra de Desplazamiento:
- ttk.Scrollbar(...) crea una barra de desplazamiento vertical dentro del frame.
- La barra se coloca a la derecha del frame y se ajusta para llenar la altura (fill="y").

- Def abrirá().

```python
from tkinter import filedialog
def abrira():
    archivo_ = filedialog.askopenfilename(
        defaultextension=".ORG",
        filetypes=[("Archivos ORG", "*.ORG"), ("Todos los archivos", "*.*")])
    if archivo_:
        with open(archivo_, "r") as archivo:
            contenido = archivo.read()
        text_area.delete(1.0, tk.END)  # Limpiar el área de texto
        text_area.insert(tk.END, contenido)  # Insertar el contenido en el área de texto
        ventana.title(f"Editor - {archivo_}")  # Mostrar el nombre del archivo en la ventana
        global archivo_actual

        archivo_actual = archivo_  # Actualizar el archivo actual
```

Importación de filedialog:
filedialog es un módulo de Tkinter que permite abrir diálogos para seleccionar archivos. Esto facilita la interacción del usuario con el sistema de archivos.
Definición de la Función abrira:
Se define una función llamada abrira. Esta función se invocará cuando se necesite abrir un archivo.
Abrir un Diálogo de Selección de Archivos:
askopenfilename() abre un diálogo que permite al usuario seleccionar un archivo.
defaultextension=".ORG" establece que, si el usuario no especifica una extensión, el archivo será guardado como .ORG.
filetypes es una lista que especifica los tipos de archivos que se mostrarán en el diálogo. Aquí, se permiten archivos con extensión .ORG y todos los archivos (*.*).

Verificación del Archivo Seleccionado:

Se verifica si el usuario ha seleccionado un archivo. Si no se selecciona nada, archivo_ será una cadena vacía y la condición será False.

Actualizar la Variable Global:
Se declara archivo_actual como una variable global. Esto permite que su valor se mantenga accesible en otras partes del programa.
archivo_actual se actualiza con el camino del archivo que se acaba de abrir.

- Def limpiar documento

```
def limpiar_documento():
    archivo = 'Errores.html'  # Especifica la ruta de tu archivo predeterminado
    try:
        with open(archivo, 'w') as file:
            file.truncate(0)
        #print(f"El archivo {archivo} ha sido limpiado.")
    except Exception as e:
        print(f"Error al limpiar el archivo: {e}")

        # aqui es el arreglo de que hice de los guardar
```

Se utiliza para limpiar el archivo html de errores generados.

- Def guardar archivo

```
def guardar_archivo():
    global archivo_abierto
    if archivo_abierto:
        with open(archivo_abierto, 'w') as file:
            contenido = text_area.get(1.0, tk.END)
            file.write(contenido)
            messagebox.showinfo("Guardar como", f"Archivo guardado como: {archivo_abierto}")
    elif text_area.get(1.0, tk.END) == "\n":  # Si el área de texto está vacía
        messagebox.showwarning("Advertencia", "No hay contenido para guardar")
    else:
        guardar_como()
```

Se utiliza para gurader el archivo o los datos que tenemos en nuestro tex área.
Si no esta guardado se abre la función guardar como.

- Def guardar como

```
def guardar_como():
    global archivo_abierto
    archivo_abierto = filedialog.asksaveasfilename(defaultextension=".txt", filetypes=[("Archivos de texto", "*.txt")])

    if archivo_abierto:
        with open(archivo_abierto, 'w') as file:
            contenido = text_area.get(1.0, tk.END)
            file.write(contenido)
```

Abre un filedialog para que el usuario pueda guardar los datos como el desee.

- Def guardar y analizar.

```
def guardar_y_analizar():
    texto=text_area.get(1.0,END)
    with open("texto.txt", "w") as archivo:
        archivo.write(texto)
    subprocess.run(["./lexer"])
    resultado = subprocess.run(["./lexer.exe"], capture_output=True, text=True)
```

Obtener el Contenido del Área de Texto:
text_area.get(1.0, END) obtiene todo el texto del área de texto, desde el principio (1.0) hasta el final (END), y lo almacena en la variable texto.
Guardar el Contenido en un Archivo:
with open("texto.txt", "w") abre (o crea si no existe) un archivo llamado texto.txt en modo escritura ("w").
archivo.write(texto) escribe el contenido de la variable texto en el archivo. Cuando se sale del bloque with, el archivo se cierra automáticamente.
Ejecutar un Programa Externo:
subprocess.run(["./lexer"]) intenta ejecutar un programa llamado lexer. Este podría ser un analizador de texto o un compilador, pero no se están capturando sus salidas.
Ejecutar un Programa y Capturar su Salida:
subprocess.run(["./lexer.exe"], capture_output=True, text=True) ejecuta un programa llamado lexer.exe. Esta línea captura la salida estándar (stdout) y la salida de error (stderr) del programa.
capture_output=True indica que se desea capturar la salida.
text=True convierte las salidas capturadas de bytes a cadenas de texto.

- For widget.

```
for widget in frame2.winfo_children():
    widget.destroy()
for linea in lineas:
    if 'Bandera:' in linea:
        ruta_imagen = linea.split(":")[1].strip()
        try:
            img=Image.open(ruta_imagen)
            tamaño_img = (100,100)
            img.thumbnail(tamaño_img)
            img. img = img.resize(tamaño_img)
            img_tk = ImageTk.PhotoImage(img)
            etq3=tk.Label(frame2, image=img_tk)
            etq3.image = img_tk
            etq3.pack()
            imprimir_imagen()

        except Exception as e:
            etq3_error = tk.Label(frame2, text="Error al cargar la imagen")
            etq3_error.pack()
    elif 'Errores' in linea:
        imprimir_imagen1()
        #limpiar label
        for widget in frame2.winfo_children():
            widget.destroy()
        etq3=tk.Label(frame2, text=linea)
        etq3.pack()
```

Limpiar el Frame:
Esta línea itera sobre todos los widgets (elementos de la interfaz) dentro del frame2 y los destruye. Esto se utiliza para limpiar el frame antes de agregar nuevos elementos, asegurando que no se acumulen imágenes o textos anteriores.
Iterar sobre las Líneas:
Se itera sobre una lista de lineas. Esta lista probablemente contiene información sobre países, imágenes y errores que se desea mostrar en el frame2.
Buscar la Bandera:
Se verifica si la cadena 'Bandera:' está presente en la línea actual. Si es así, se extrae la ruta de la imagen usando split(":"), lo que divide la línea en dos partes en base a los dos puntos :. La segunda parte (índice [1]) es la ruta de la imagen, que se limpia de espacios en blanco con strip().
Cargar y Ajustar la Imagen:
Se intenta abrir la imagen en la ruta especificada utilizando Image.open(ruta_imagen). Luego, se define un tamaño de imagen deseado de 100x100 píxeles.
img.thumbnail(tamaño_img) redimensiona la imagen para que quepa dentro de las dimensiones especificadas, manteniendo su relación de aspecto.
Sin embargo, hay un error en la siguiente línea: img.img = img.resize(tamaño_img). Esta línea no es necesaria y debería ser eliminada porque img ya está siendo ajustada con el método thumbnail.
Mostrar Mensajes de Error:
Si la línea actual contiene la palabra 'Errores', se llama a la función imprimir_imagen1(). Esta función no está definida en el fragmento, pero se asume que maneja la visualización de errores de alguna manera.
Luego, se limpia el frame2 (similar al inicio) y se muestra la línea de error en un nuevo Label.

- Def imprimir imagen

```
def imprimir_imagen():
    #limpiar area de texto
    text_area2.config(state=NORMAL)
    text_area2.delete(1.0, END)
    text_area2.config(state=DISABLED)


    arcchivo_imagen = "graficoultimo.png"

    imagen = Image.open(arcchivo_imagen)
    imagen = imagen.resize((1100, 350))
    imagen_tk = ImageTk.PhotoImage(imagen)
    text_area2.delete(1.0, END)
    text_area2.config(state=NORMAL)
    text_area2.image_create(END, image=imagen_tk)
    text_area2.config(state=DISABLED)
    text_area2.image = imagen_tk
```

La función imprimir_imagen() se encarga de cargar una imagen desde un archivo, redimensionarla, y mostrarla en un área de texto de la interfaz gráfica. Asegura que el área de texto se limpie antes de mostrar la nueva imagen y que el usuario no pueda editar el contenido.
Carga la imagen creada por el documento dot.

- Def imprimir imagen1

```
def imprimir_imagen1():
    #limpiar area de texto
    text_area2.config(state=NORMAL)
    text_area2.delete(1.0, END)
    text_area2.config(state=DISABLED)


    arcchivo_imagen = "e (variable) arcchivo_imagen: Literal['error.png']
    imagen = Image.open(arcchivo_imagen)
    imagen = imagen.resize((300, 300))
    imagen_tk = ImageTk.PhotoImage(imagen)
    text_area2.delete(1.0, END)
    text_area2.config(state=NORMAL)
    text_area2.image_create(END, image=imagen_tk)
    text_area2.config(state=DISABLED)
    text_area2.image = imagen_tk
```

Es la misma estructura de imprimir imagen anterior, solo que esta tiene una imagen diferente que se usara cuando el documento tenga errores léxicos y se mostrara como una advertencia de errores.

- Def funciones.

```
def funcione():
    #limpiar_area_imagen()
    limpiar_documento()
    limpiar_area_imagen()
    guardar_y_analizar()
```

esta función llama en una sola a cada una de las funciones que hemos ido creando.

- Def limpiar área de imagen.

```
def limpiar_area_imagen():
    text_area2.config(state=NORMAL)
    text_area2.delete(1.0, 'end')
    text_area2.config(state=DISABLED)
```

Esta función limpia el área de texto 2 donde se imprimen imágenes.

- Barra manu

```
#barra de menu
barraMenu = Menu(ventana)
ventana.config(menu=barraMenu)
#menu desplegable
menuArchivo = Menu(barraMenu, tearoff=False)
menuacerca = Menu(barraMenu, tearoff=False)
barraMenu.add_cascade(label="Archivo", menu=menuArchivo)
barraMenu.add_cascade(label="Acerca de", menu=menuacerca)
```

Se crea una barra menú que contendrá dos botones desplegables Archivos y acerca de.

- Def salir

```
#salir del programa
def salir():
    ventana.quit()
    ventana.destroy()
#acerca de
def acercade():
    messagebox.showinfo("Acerca de", "Analizador de Mercado\nVersion 1.0\nDesarrollado por: \nDavid Orlando Fuentes Morales\nCARNE: 201709022")
```

Esta función termina con el proceso del programa y lo cierra.

- Opciones del menú

```
#opciones del menu
menuArchivo.add_command(label="Abrir", command=abrira)
menuArchivo.add_command(label="Guardar", command=guardar_archivo)
menuArchivo.add_command(label="Guardar Como", command=guardar_como)
menuArchivo.add_command(label="Salir", command=salir)
menuArchivo.add_command(label="Limpiar", command=limpiar)
menuacerca.add_command(label="Acerca de",command=acercade)
```

Se escriben los nombres y los botones que tendrá la barra menú. Cada uno se le define un command para que al momento de seleccionarlo funcione y se realice la función creada.

- Text area2

```
#area de texto para codigo de salida bloqueado
text_area2 = Text(ventana, width=140, height=25)
text_area2.place(x=50, y=400)
text_area2.config(state=DISABLED)
```

Se define el tamaño y la posición del área en la ventana principal y el disable es para que el usuario no pueda editar

- Frame botones.

```
#frame para los botones
frame = Frame(ventana)
frame.place(x=600, y=300)

#botones dentro del frame
btn1 = Button(frame, text="Analizar", command=funcione)
#btn1= Button(frame, text="Analizar", command=imprimir_imagen)
btn1.grid(row=0, column=0)
btn2 = Button(frame, text="Limpiar", command=limpiar)
btn2.grid(row=1, column=0)
btn3 = Button(frame, text="Salir", command=salir)
btn3.grid(row=2, column=0)
```

Se crea un frame que contrndra los botones que nos ayudaran con las funcionalidades del programa, a cada uno se le coloca su command que será la función que se ejecutara al momento de seleccionarlo.

- Frame 2

```
#frame para resultados nombre e imagen
frame2 = Frame(ventana, width=400, height=200, bg="lightyellow
frame2.pack_propagate(False)
frame2.place(x=600, y=70)

#etiquetas para los resultados
etq2 = Label(frame2, text="Resultado:")
etq2.pack()
etq3 = Label(frame2, text=".")
etq3.place(x=10, y=30)
etq4 = Label(frame2, text=".")
etq4.place(x=10, y=50)
etq5 = Label(frame2, text=".")
ventana.mainloop()
```

Se crea un frame 2 que contendrá los resultados de nuestro programa el cual se imprimirán el país y sus datos.

Se define la posición del frame2 en la ventana principal y su color.

Ventana. Mainloop es la terminación de nuestra ventana de código en phyton.