République Algérienne Démocratique et Populaire
Ministère de l'enseignement supérieur et de la recherche scientifique
Université KASDI Merbah - Ouargla
Faculté des Nouvelles Technologies de l'information et de la Communication
Département d'Informatique et de la Technologie de l'Information

# Project Report

# CREATING A COMPILER FOR A DEFINED LANGUAGE

**By safa Miloudi**

# **Student Information:**

Name: Safa Miloudi
Group: 5
Matriculation Number: 212139050226
Course: Compilation
Date: 7/12/2023

# 1.Introduction:

**The ambitious field of compiler construction plays a pivotal role in software development, acting as a bridge between high-level programming languages and machine code. where we aim to create a compiler for a unique language. This language presents distinctive features, such as identifier rules, constants, comments, and program structures. Our goal is to navigate through various compilation stages without relying on automatic generator tools, ensuring a hands-on and insightful experience.**

# 2.Project Scope:

**The language specification for this compiler project encompasses distinctive features, including rules for identifiers, constants, comments, and program structure. From the meticulous definition of identifiers, ensuring adherence to character limits and constraints, to handling complex numeric and real constants.**

# 3.Objectives:

**The primary goal of this project is to create a compiler that can robustly process source code written in the defined language. The key objectives include:**

## .Lexical Analysis:

**Develop a comprehensive lexical analyzer to break down the source code into meaningful tokens.Implement a finite automaton for identifying and categorizing entities such as identifiers, constants, operators, and separators.**

## .Syntax Analysis:

**Construct a syntax analyzer capable of recognizing the grammatical structure of the source code.**
**Develop parsing mechanisms to ensure adherence to the specified language rules.**

## .Semantic Analysis:

**Implement semantic analysis components to identify and report any inconsistencies or violations of language semantics.**
**Enforce type checking and ensure compatibility between expressions.**

## .Code Generation:

**Design and implement code generation modules to convert the analyzed source code into an intermediate representation.**
**Generate target code that adheres to the architecture and requirements of the specified language.**

## .User Interface:

**Develop an intuitive user interface using Tkinter to facilitate user interaction with the compiler.**
**Enable functionalities such as opening and saving source code files, providing a seamless user experience.**

# 4.Methodology:

**The project follows a structured approach, dividing the compilation process into distinct phases. Each phase addresses specific aspects of the language, ensuring a step-by-step progression towards the creation of a fully functional compiler.**

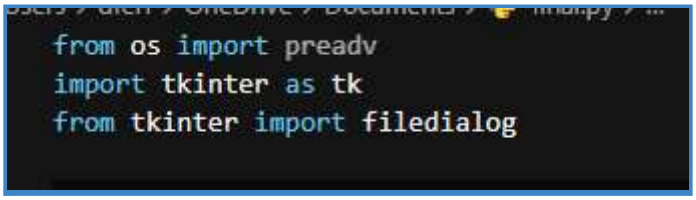
# 5.Tools and Technologies:

**Programming Language: Python**
**Graphical User Interface (GUI) Library: Tkinter**
**File Handling: os, tkinter.filedialog**

# 6.The workflow:

**This code appears to be a Python script that creates a simple graphical user interface (GUI) using the Tkinter library for a lexical analyzer or compiler tool. Let's break down the code and explain its various components:**

## 6.1.Import Statements:




**from os import preadv**

**.The os module in Python provides a way to interact with the operating system.**
**preadv is a function in the os module that is used to perform a pread system call.**
**.The pread system call reads data from a file descriptor into a buffer without modifying the file offset.**

**import tkinter as tk**

**.Imports the tkinter module, which is the standard GUI (Graphical User Interface) toolkit for Python.**
**.Renames the module to tk for convenience.**

**from tkinter import filedialog**

**.Imports the filedialog submodule from the tkinter module.**
**.filedialog provides dialogs for opening and saving files. In this code, it is used to create file dialogs for opening and saving files in the GUI application.**

# 6.2.The main application window:

## The code:

```
# Create the main application window
root = tk.Tk()  # Create the main window
root.title("Interface")  # Set the title of the window

menu = tk.Menu(root)  # Create a menu for the window
root.config(menu=menu)  # Configure the menu for the window

file_menu = tk.Menu(menu)  # Create a submenu for file-related options
menu.add_cascade(label="File",
                 menu=file_menu)  # Add the submenu to the main menu
file_menu.add_command(label="Open",
                      command=open_file)  # Add an option to open a file
file_menu.add_command(
    label="Save",
    command=save_file)  # Add an option to save the output to a file
file_menu.add_command(label="Remove", command=remove_text)  # Add an option to remove text
file_menu.add_separator()  # Add a separator in the menu
file_menu.add_command(label="Exit",command=root.quit)  # Add an option to exit the program

input_label = tk.Label(root,
                       text="Input:")  # Create a label for the input text box
input_label.pack()  # Display the input label

input_text = tk.Text(root, height=10, width=30)  # Create the input text box
input_text.pack()  # Display the input text box

output_label = tk.Label(
    root, text="Output:")  # Create a label for the output text box
output_label.pack()  # Display the output label

output_text = tk.Text(root, height=10, width=30)  # Create the output text box
output_text.pack()  # Display the output text box

process_input_button = tk.Button(
    root, text="OK",
    command=output)  # Create a button to process the input
process_input_button.pack()  # Display the process input button

root.mainloop()  # Start the main event loop for the application
```

**This code snippet creates a simple GUI (Graphical User Interface) application using the Tkinter library in Python:**

## Main Application Window:

**root = tk.Tk(): Creates the main window for the application.**
**root.title("Interface"): Sets the title of the window.**

## Menu Bar:

**menu = tk.Menu(root): Creates a menu for the window.**
**root.config(menu=menu): Configures the menu for the window.**

## File Menu (Submenu):

**file_menu = tk.Menu(menu): Creates a submenu for file-related options.**
**menu.add_cascade(label="File", menu=file_menu): Adds the submenu to the main menu.**

## File menu options:

**"Open":**
**Calls the open_file function.**
**"Save":**
**Calls the save_file function.**
**"Remove":**
**Calls the remove_text function.**
**Separator:**
**Adds a separator in the menu.**
**"Exit":**
**Calls root.quit to exit the program.**

## Input Text Box:

**input_text = tk.Text(root, height=10, width=50): Creates a multiline text box for input.**

## Output Text Box:

**output_text = tk.Text(root, height=10, width=50): Creates a multiline text box for output.**

## Labels:

**input_label and output_label: Labels for the input and output text boxes.**

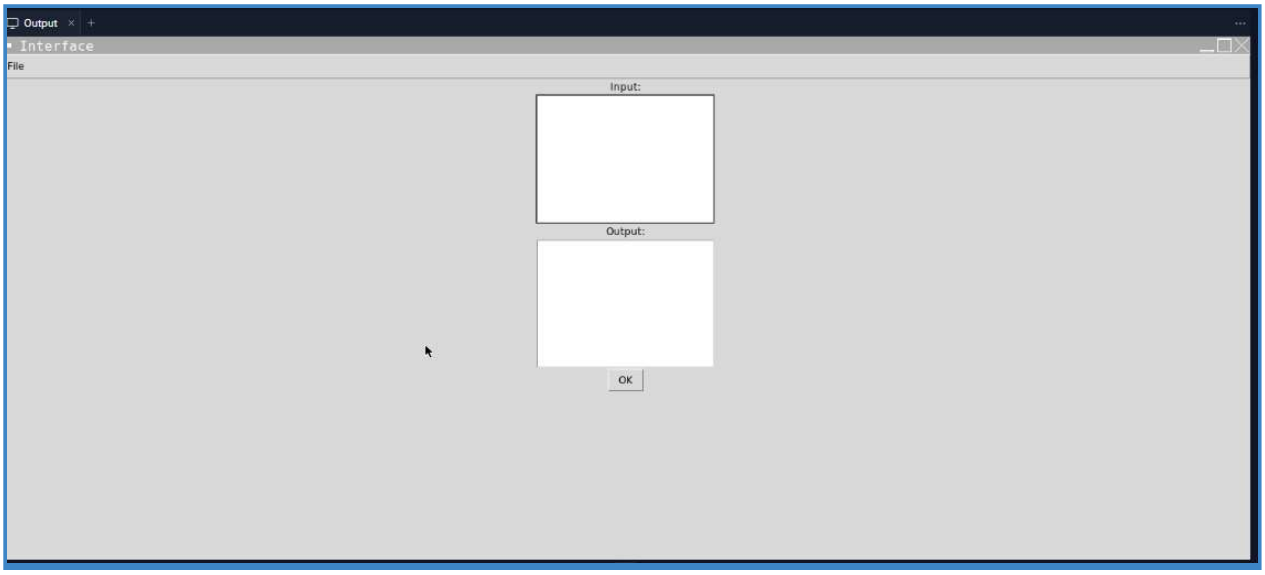
## Process Input Button:

**process_input_button = tk.Button(root, text="OK", command=output): Creates a button labeled "OK" that calls the output function when clicked.**

## Main Event Loop:

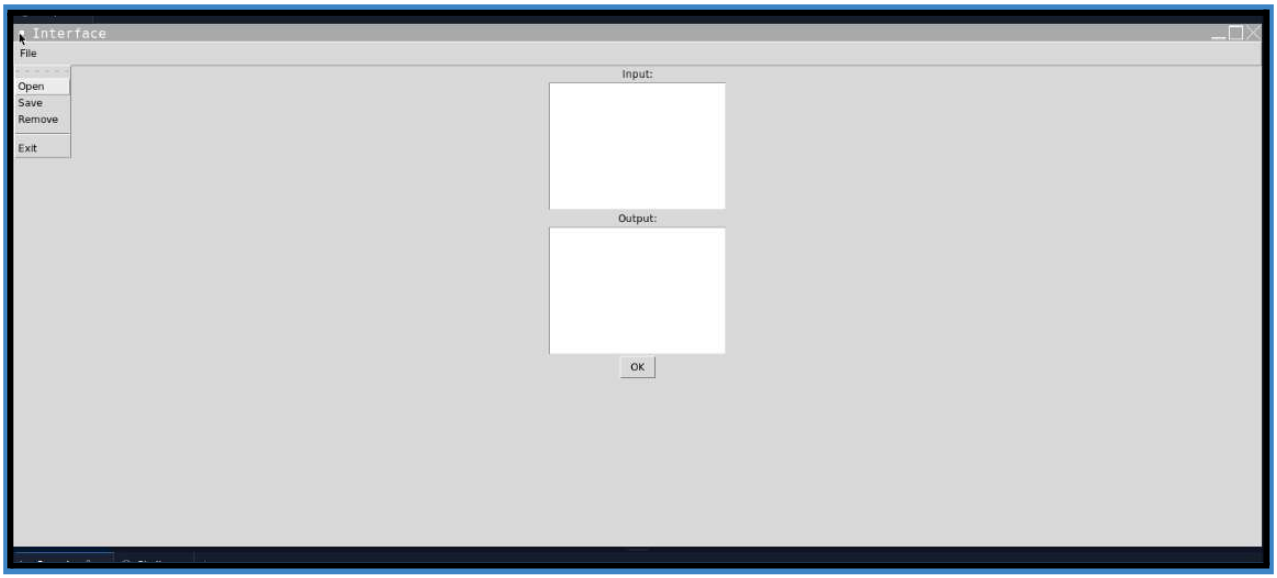
**root.mainloop(): Starts the main event loop for the application, allowing it to respond to user interactions.**

**The result:**




## 6.3.File handling fonctions:

**Menu:**




- **open:**

## The code:open_file():

```
# Function to open a file and display its content in the "Input" text box
def open_file():
  file_path = filedialog.askopenfilename()
  if file_path:
    with open(file_path, 'r') as file:  # Open the selected file in read mode
      content = file.read()  # Read the content of the file
      input_text.delete(1.0, tk.END)  # Clear the input text box
      input_text.insert(tk.END,
                        content)  # Insert the content into the input text box
```

**This function includes a Tkinter GUI with a Text widget named input_text for displaying and editing text. The function allows the user to open a file and view its content in the application:**

**Import Statements:**

**from tkinter import filedialog, tk: Import the necessary modules from Tkinter. filedialog is used to display a file dialog for file selection, and tk is used to reference the Tkinter module.**

**Function Definition:**

**def open_file():: Define a function named open_file that will be responsible for opening a file and displaying its content.**

**Selecting a File:**

**file_path = filedialog.askopenfilename(): Use filedialog.askopenfilename() to prompt the user to select a file. The selected file's path is stored in the file_path variable.**

**Checking File Selection:**

**if file_path:: Check if a file was selected by ensuring that file_path is not an empty string (i.e., the user didn't cancel the file dialog).**

**Reading File Content:**

**with open(file_path, 'r') as file:: Open the selected file in read mode ('r'). The with statement ensures that the file is properly closed after reading its content.**

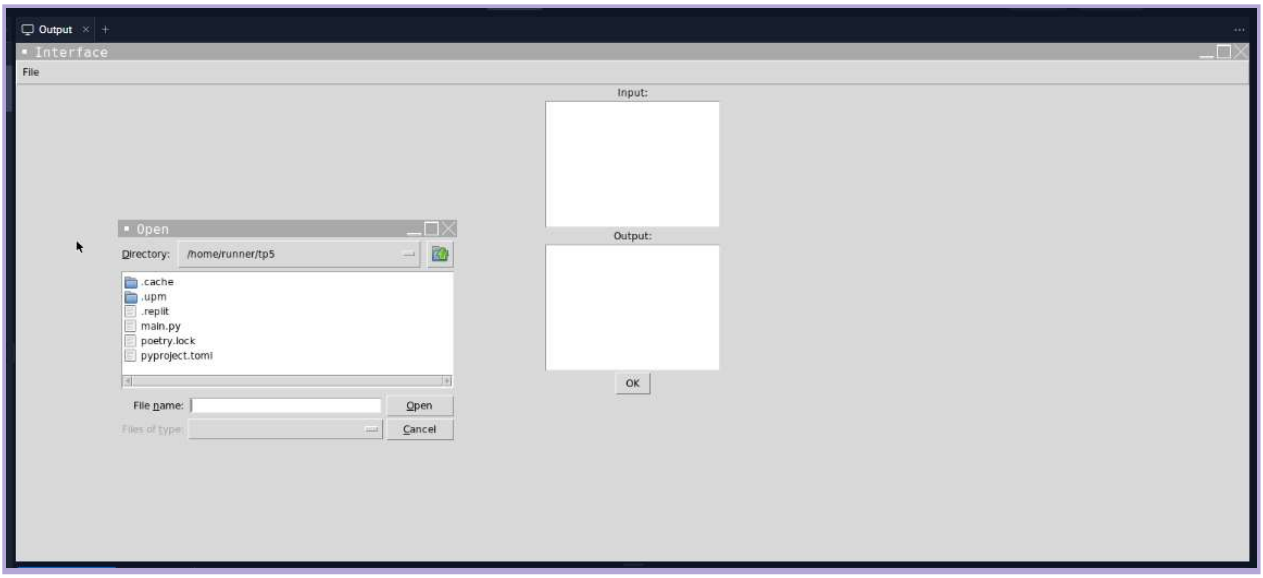
**content = file.read(): Read the content of the file and store it in the content variable.**

**Updating the Text Box:**

**input_text.delete(1.0, tk.END): Clear the content of the "Input" text box. input_text is assumed to be a reference to a Tkinter Text widget.**

**input_text.insert(tk.END, content): Insert the content of the file into the "Input" text box at the end (tk.END) of its current content.**

## The result:




## Functionality:

**Opens a file dialog to select and open a file.**
**GUI Element Interaction: Invoked when the user wants to open a file.**

- **Save:**

## The code:save_file():

```
# Function to save the content of the "Output" text box to a text file
def save_file():
  file_path = filedialog.asksaveasfilename(
      defaultextension=".txt")  # Open the file dialog for saving
  if file_path:
    content = output_text.get(1.0,
                              tk.END)  # Get the content of the output text box
    with open(file_path, 'w') as file:  # Open the file in write mode
      file.write(content)  # Write the content to the file
```

**The purpose of this function is to save the content of the "Output" text box to a text file. Let's break down the code and explain each part:**

### Import Statements:

**The code assumes that the necessary modules from Tkinter (filedialog and tk) have been imported elsewhere in the program.**

**Function Definition:**

**def save_file():**

**Define a function named save_file that will be responsible for saving the content of the "Output" text box to a text file.**

**Opening the File Dialog for Saving:**

**file_path = filedialog.asksaveasfilename(defaultextension=".txt"): Use filedialog.asksaveasfilename to prompt the user to select a file path for saving.**
**The defaultextension=".txt" specifies that the default file extension should be ".txt."**

**Checking File Selection:**

**if file_path::**
**Check if a file path was selected by ensuring that file_path is not an empty string (i.e., the user didn't cancel the file dialog).**

**Getting Content from Text Box:**

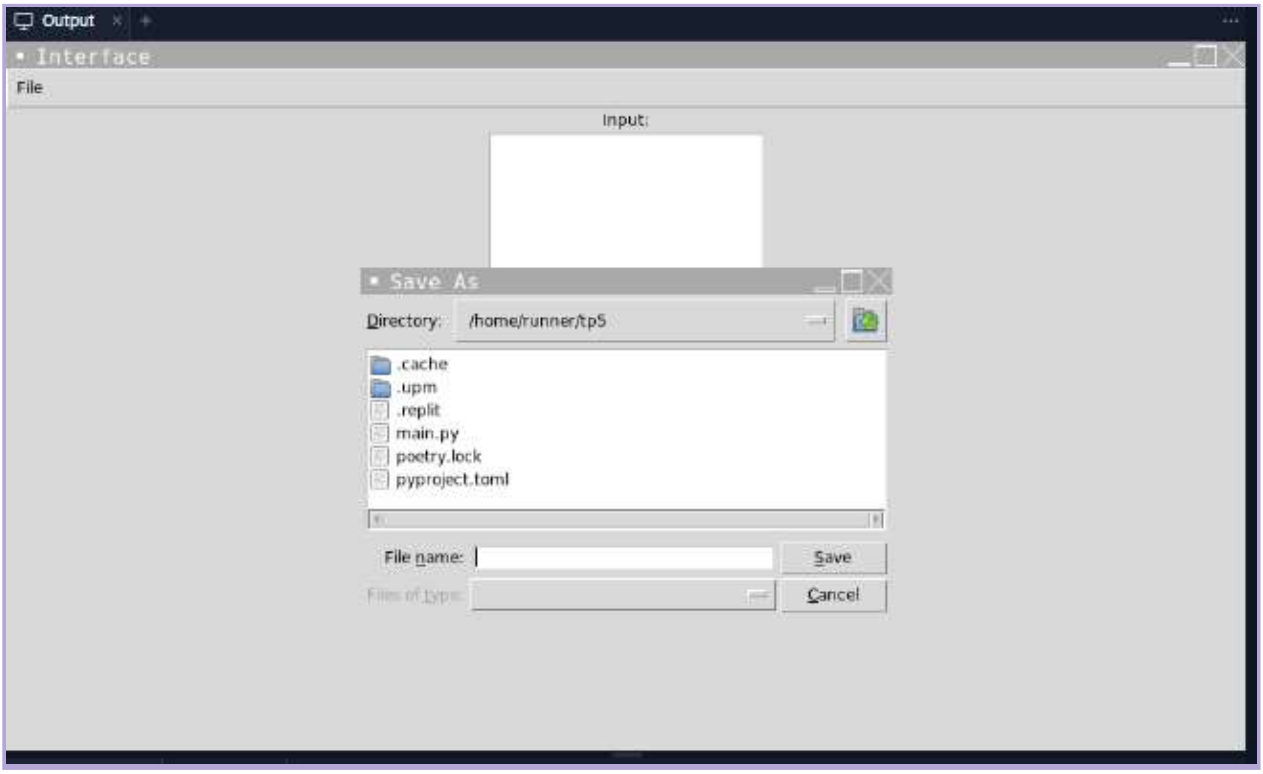
**content = output_text.get(1.0, tk.END):**
**Get the content of the "Output" text box using the get method. output_text is assumed to be a reference to a Tkinter Text widget.**

**Writing Content to File:**

**with open(file_path, 'w') as file:**
**Open the selected file in write mode ('w'). The with statement ensures that the file is properly closed after writing its content.**
**file.write(content):**
**Write the content of the "Output" text box to the file.**

**This function allows the user to save the content of the "Output" text box to a text file by selecting a file path through a file dialog.**

## The result:




## Functionality:

**Opens a file dialog to save the content of the "Output" text box to a text file.**
**GUI Element Interaction: Invoked when the user wants to save the output to a file.**

- ## Remove:

## The code:*remove_text():*

```
# Function to remove text from both the "Input" and "Output" text boxes
def remove_text():
  input_text.delete(1.0, tk.END)  # Clear the content of the input text box
  output_text.delete(1.0, tk.END)  # Clear the content of the output text box
```

**The purpose of this function is to clear the content of both the "Input" and "Output" text boxes. Let's break down the code and explain each part:**

**Function Definition:**
**def remove_text():: Defines a function named remove_text that is responsible for removing or clearing the text content from both the "Input" and "Output" text boxes.**

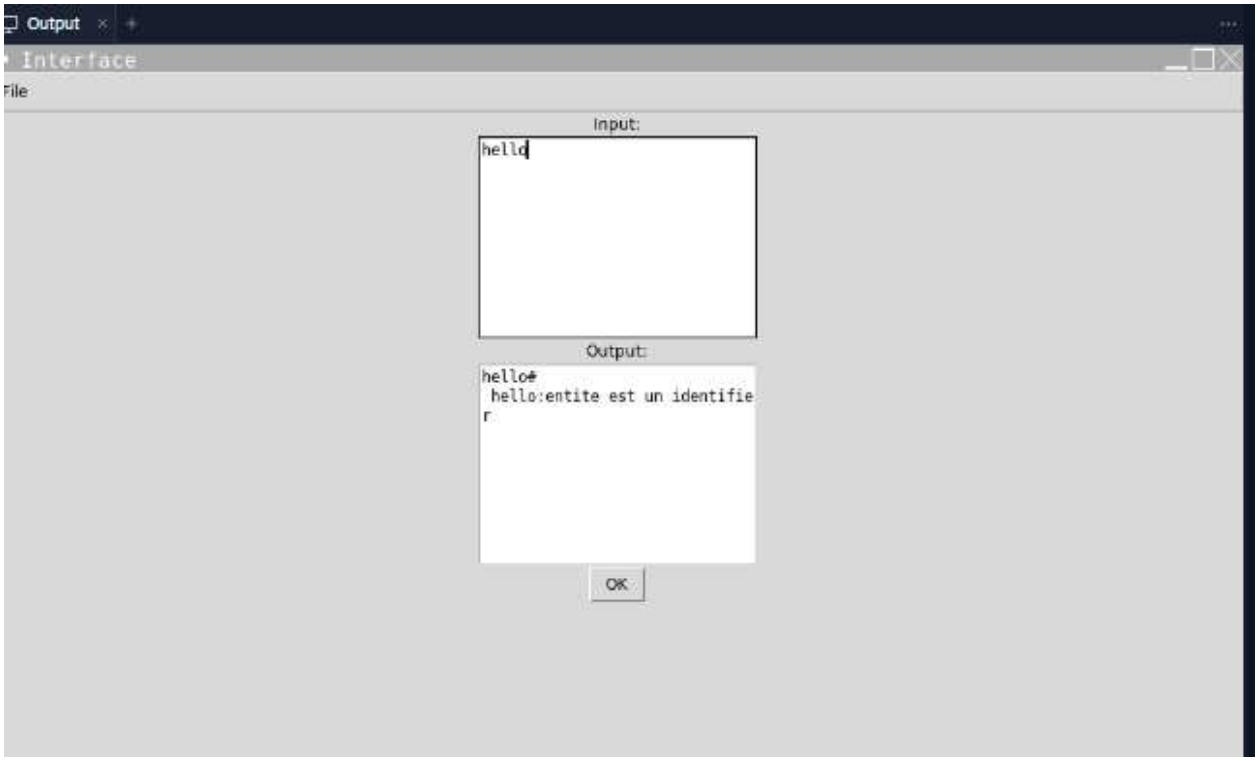
**Clearing "Input" Text Box:**
**input_text.delete(1.0, tk.END): The delete method of the Tkinter Text widget (input_text) is used to remove text. The parameters 1.0 and tk.END specify the range of text to be**

**deleted. In this case, it starts from the beginning of the text box (1.0) and deletes everything until the end (tk.END).**

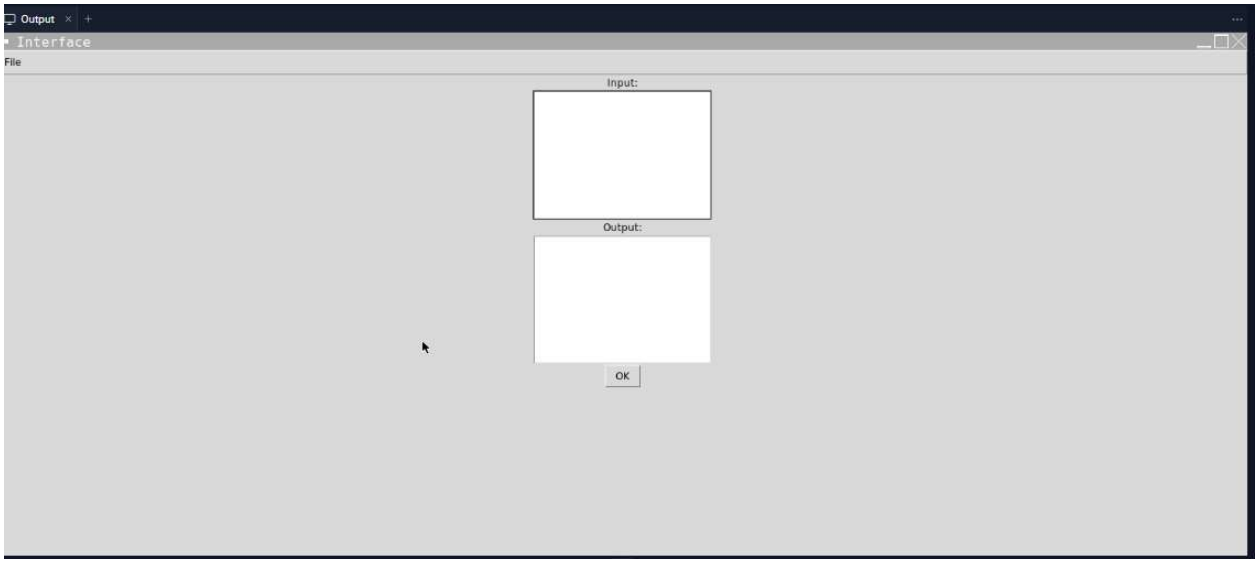
**Clearing "Output" Text Box:**
**output_text.delete(1.0, tk.END): Similar to the "Input" text box, the delete method is used to clear the content of the "Output" text box (output_text).**

## The result:

**Before using remove:**



**After using remove:**




***Functionality:***

**Clears the content of both the "Input" and "Output" text boxes.GUI Element Interaction: Invoked when the user wants to clear the text boxes.**

# 6.4. Text Processing Functions:

## length(word):

**The code:**

```
def length(word):
    counter = 0  # Initialize a counter variable to keep track of the number of characters

    # Iterate through each character in the input word
    for char in word:
        counter += 1  # Increment the counter for each character in the word

    return counter  # Return the final count, representing the length of the word
```

**This function provides a simple and manual way to calculate the length of a string without using built-in functions like len(). It is equivalent to using len(word) to get the length of the string. The function can be called with a string as an argument, and it will return the number of characters in that string:**

**Function Definition:**

**def length(word)::**
**Defines a function named length that takes a single parameter word, which is expected to be a string.**

**Counter Initialization:**

**counter = 0:**
**Initializes a variable counter to zero. This variable will be used to count the number of characters in the given string.**

**Iteration through Characters:**

**for char in word::**
**Initiates a for loop that iterates through each character (char) in the input string word.**

**counter += 1:**
**Inside the loop, for each character encountered, the counter variable is incremented by 1. This effectively counts the number of characters in the string.**

**Return Statement:**

**return counter:**
**After the loop has iterated through all characters in the input string, the function returns the final value of the counter variable. This value represents the length of the input string.**

## Functionality:

**Calculates the length of a given word (string).**

## Usage:

**Used in the type checking function.**

## carecter_majuscule(carecter):

## The code:

**The purpose of this function is to determine whether a given character (carecter) is an uppercase letter. It uses the ASCII values of characters to check if the input character falls within the range of uppercase letters ('A' to 'Z'):**

```
def carecter_majuscule(carecter):
    # Check if the character is in the range of uppercase letters ('A' to 'Z')
    if carecter >= 'A' and carecter <= 'Z':
        # If true, return True
        return True
    # If not in the range, return False
    return False
```

**Condition Check:**
**checks if the ASCII value of the input character is greater than or equal to the ASCII value of 'A' and less than or equal to the ASCII value of 'Z'. This condition ensures that the character is in the range of uppercase letters.**
**Return True:**
**If the condition is true (i.e., the character is an uppercase letter), the function returns True.**
**Return False:**
**If the character is not in the range of uppercase letters, the function returns False.**

## Functionality:

**The carecter_majuscule function checks if the input character is an uppercase letter by comparing it to the range of uppercase letters ('A' to 'Z'). It returns True if the character is uppercase and False otherwise.**

## Usage:

**This function is used in the type checking function to verify the case of characters, which is relevant for certain token types.**

## Key(word):

## The code:

```
def Key(word):
    # Initialize an empty string to store the word without spaces
    word_no_space = ""

    # Iterate through each character in the word
    for char in word:
        # If the character is not a space, append it to the word_no_space
        if char != " ":
            word_no_space += char
    # Check if the word (without spaces) matches any keyword
    if word_no_space == "BEGIN" or word_no_space == "END" or word_no_space == "PROGRAMME" or word_no_space == "INT" or \
            word_no_space == "REAL" or word_no_space == "CHAR" or word_no_space == "STRING" or word_no_space == "OR" or \
            word_no_space == "AND" or word_no_space == "IF" or word_no_space == "ELSE" or word_no_space == "BOOLEAN" or \
            word_no_space == "FIN." or word_no_space == "DO" or word_no_space == "EXECUTE" or word_no_space == "VAR" or \
            word_no_space == "FOR" or word_no_space == "OTHERWISE" or word_no_space == "CONST" or word_no_space == "IDF" \
            or word_no_space == "WHILE":
        # If it matches, return True
        return True
    # If it doesn't match, return False
    return False
```

## Functionality:

**The Key function checks if the input word is a keyword by comparing it to a predefined list of keywords. It returns True if the word is a keyword and False otherwise.**

## Usage:

**This function is used in the type checking function to identify if an entity is a keyword or not.**

## removing_spaces_comments(input_text):

## The code:

```
def removing_spaces_comments(input_text):
  # Initialize variables
  processed_text = input_text
  input_text = ''  # The resulting text without spaces and comments
  space = " "  # Variable for a single space
  counter_commet = 1  # Counter to toggle between comment mode and normal mode
  temp_part = ''  # Temporary variable for a part of the input text
  previous_was_space = False  # Flag to track if the previous character was a space
  previous_was_newline = False  # Flag to track if the previous character was a newline
  # Iterate through each character in the processed text
  for char in processed_text:
      if counter_commet == 1:  # In normal mode
          if char != '%' and char != ' ' and char != '\n':
              # Append character to the input_text if it's not a comment character or space
              input_text += char
              previous_was_space = False
              previous_was_newline = False
          elif char == '%' or char == ' ' or char == '\n':
              if char == '%':
                  counter_commet *= -1  # Toggle between the two modes (comment mode or normal mode)
              else:
                  # Add a single space if not already added and the previous character was not a newline
                  if not previous_was_space and not previous_was_newline:
                      input_text += space
                  previous_was_space = True if char == ' ' else False
                  previous_was_newline = True if char == '\n' else False
      else:  # In comment mode
          if char != '%' and char != ' ' and char != '\n':
              input_text += ''  # Ignore characters in comment mode
              previous_was_space = False
              previous_was_newline = False
          elif char == '%' or char == ' ' or char == '\n':
              if char == '%':
                  counter_commet *= -1  # Toggle between the two modes (comment mode or normal mode)
              else:
                  # Add a single space if not already added and the previous character was not a newline
                  if not previous_was_space and not previous_was_newline:
                      input_text += space
                  previous_was_space = True if char == ' ' else False
                  previous_was_newline = True if char == '\n' else False
  return input_text
```

**This function, named removing_spaces_comments, processes a given input text by removing spaces and handling comments within a programming-like context. Let's break down the code and understand each part:**

**Initialization:**

**Processed_text:**

**A copy of the input text that will be processed.**

**input_text:**

**The resulting text without spaces and comments.**

**Space:**

**A variable representing a single space.**

**counter_comment:**

**A counter to toggle between comment mode and normal mode.**

**Temp_part:**
**A temporary variable for a part of the input text.**
**previous_was_space:**
**A flag to track if the previous character was a space.**
**previous_was_newline:**
**A flag to track if the previous character was a newline.**
**Purpose of Variables:**
**processed_text:**
**Holds a copy of the input text for processing.**
**input_text:**
**Stores the resulting text without spaces and comments.**
**space:**
**Represents a single space.**
**Counter_comment:**
**Keeps track of whether the code is in comment or normal mode.**
**Temp_part:**
**Temporarily stores a part of the input text.**
**previous_was_space:**
**A flag indicating whether the previous character was a space.**
**previous_was_newline:**
**A flag indicating whether the previous character was a newline.**

**Iteration:**
**Iterate through each character in the processed text.**

**Normal Mode:**

**If the character is not '%' and not a space or newline, append it to input_text.**
**If it is '%' or space or newline, handle accordingly, including toggling between comment and normal modes and adding spaces if needed.**

**Comment Mode:**

**If the code is in comment mode.**
**Ignore characters that are not '%' or space or newline.**
**Handle '%' or space or newline by toggling modes and adding spaces if needed.**

**Return Result:**
**Return the processed text without spaces and comments.**

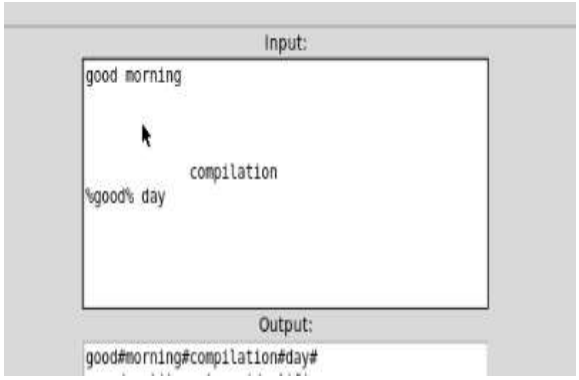
## Functionality:

**The function processes the input text character by character, removing unnecessary spaces and comments. It uses a counter to toggle between normal mode and comment mode, ensuring that spaces are appropriately handled.**

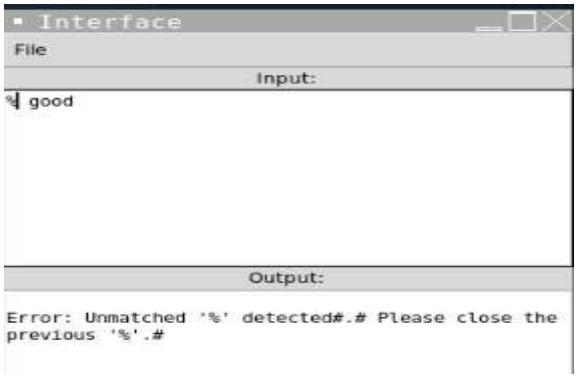
## Returns:

**A processed version of the input text.**

**In case1:**




**In case2:**




## Usage:

**Prepares the text for further processing.**

## separation_text(text):

## The code:

```
def separation_text(txt):
  # Initialize variables
  text_processed = txt
  txt = ""
  output = ""

  # Define separators to separate different components in the text
  separator = set('+-*/=(){}[]<>,;.')
  # Define operators that need special handling
  operations = set('+-*/=')
  # Define numeric characters
  number = set('0123456789')
  # Define alphabetic characters
  alpha = set('abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ')
  signals = set('+-')
  alphanumeric = alpha | number
  equality = set('<>!=:')
  punctuation = set(';,')


  if length(text_processed) >= 1:
      text_processed = text_processed + "#"
```

```
    i = 0
    while i < length(text_processed):
        if i == length(text_processed) - 1:
            break
        elif text_processed[i] == ' ':
            i += 1
        if text_processed[i] in alphanumeric and text_processed[i + 1] in alphanumeric or text_processed[i + 1] ==
' ' \
                or text_processed[i] == '_' and text_processed[i + 1] in alphanumeric \
                or i == 0 and text_processed[i] in signals and text_processed[i + 1] in number \
                or text_processed[i] in signals and text_processed[i + 1] == '.' and text_processed[
            i + 2] in number and i == 0 \
                or text_processed[i] in equality and text_processed[i + 1] == '=' \
                or text_processed[i - 1] in equality and text_processed[i] == '=' and text_processed[i + 1] == '='
and not \
        text_processed[i + 2] in number and not text_processed[i + 2] == '.' \
                or text_processed[i] in number and text_processed[i + 1] == '.' \
                or text_processed[i] == '.' and text_processed[i + 1] in number \
                or text_processed[i - 1] in operations and text_processed[i] in signals and text_processed[i + 1] in
number \
                or text_processed[i - 1] in alphanumeric and text_processed[i] == '+' and text_processed[i + 1] ==
'+' and not \
        text_processed[i + 2] in number and not text_processed[i + 2] == '.' \
                or text_processed[i - 1] in alphanumeric and text_processed[i] == '-' and text_processed[i + 1] == '-
' and not \
        text_processed[i + 2] in number and not text_processed[i + 2] == '.' \
                or text_processed[i - 1] in punctuation and text_processed[i] in signals and text_processed[i + 1]
in number \
                or text_processed[i - 1] in operations and text_processed[i] in signals and text_processed[i + 1] ==
'.' \
                or text_processed[i - 1] in signals and text_processed[i] in signals and text_processed[i + 1] ==
'.' \
                or text_processed[i - 1] == '=' and text_processed[i] in signals and text_processed[i + 1] == '.':
            txt += text_processed[i]

        elif i == i and text_processed[i - 1] in signals and text_processed[i] in signals and text_processed[
            i + 1] in number:
            txt += "#" + text_processed[i]
```

```
        elif i == i and text_processed[i - 1] in signals and text_processed[i] in signals and text_processed[
            i + 1] in number:
            txt += "#" + text_processed[i]
        elif text_processed[i] == '#':
            txt += text_processed[i]
        else:
            txt += text_processed[i] + "#"
        i += 1

    # Remove trailing '#' characters
    while txt[-1] == '#':
        txt = txt[:-1]
    txt += "#"
    i = 0

    # Add '#' to the end
    return txt
```

**Initialization of Sets:**

**Sets like separator, operations, number, alpha, signals, alphanumeric, equality, and punctuation are defined to categorize different types of characters.**

**Variable Initialization:**

**Variables like text_processed and txt are initialized to store the input and processed output, respectively.**
**The index variable i is set to 0.**

**Ensuring Input Length:**

**Checks if the input text has at least one character. If not, it appends a '#' to the text.**

**Main Loop:**

**Iterates through each character in the text_processed using a while loop.**

**Character Handling:**

**Checks various conditions to determine the type of character and how to handle it.**
**Manages cases for alphanumeric characters, separators, operators, and specific character combinations.**

**Buffer and Output Management:**

**Uses the txt variable to manage the output and insert '#' symbols between characters as needed.**
**Handles cases where consecutive signals precede a number.**

**Trailing '#' Removal and Finalization:**

**Removes trailing '#' characters from the output.**
**Adds a '#' to the end of the output.**

**Return:**

**Returns the final processed output.**

## Functionality:

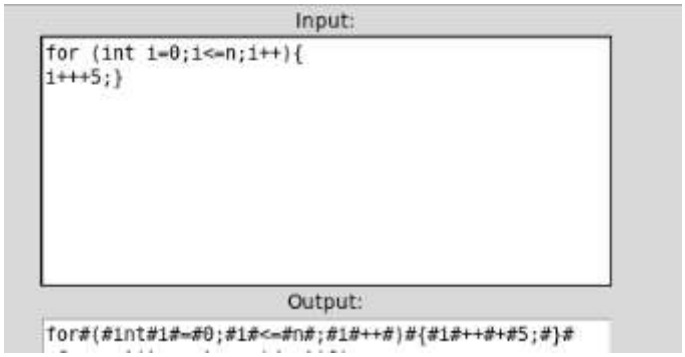
**Separates text into tokens using predefined separators.**

## Usage:

**Part of the text processing workflow.**

## Returns:

**A modified version of the input text with separators.**




## type_transition(carecter):

## The code:

```python
def type_transition(carecter):
    # Define sets of characters for different types
    number = set('0123456789')
    alpha = set('abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ')

    # Check the type of the input character and return a corresponding code
    if carecter in number:  # If the character is a digit
        return 0
    elif carecter == '+':  # If the character is '+'
        return 1
    elif carecter == '-':  # If the character is '-'
        return 2
    elif carecter == '=':  # If the character is '='
        return 3
    elif carecter == '.':  # If the character is '.'
        return 4
    elif carecter == '_':  # If the character is '_'
        return 5
    elif carecter == '<' or carecter == '>':  # If the character is '<' or '>'
        return 6
    elif carecter in ('(', ')', ';', '.'):  # If the character is '(', ')', ';', or '.'
        return 7
    elif carecter in ('*', '/'):  # If the character is '*' or '/'
        return 8
    elif carecter == ':':  # If the character is ':'
        return 9
    elif carecter in alpha:  # If the character is an alphabetic character
        return 10
    elif carecter == '!':  # If the character is '!'
        return 11
    else:
        return -1  # Return -1 for characters that don't match any type
```

**The type_transition function is designed to categorize a given character into different types and return a corresponding code. Let's break down the key elements of this function:**

### Character Sets:

**number:**
**A set containing numeric characters.**
**Alpha:**
**A set containing alphabetic characters.**

**Type Checking:**

**The function checks the type of the input character and returns a corresponding code based on its category**

**.**

**Type Codes:**

**0:**
**If the character is a digit.**
**1:**
**If the character is '+'.**
**2:**
**If the character is '-'.**
**3:**
**If the character is '='.**
**4:**
**If the character is '.'.**
**5:**
**If the character is '_'.**
**6:**
**If the character is '<' or '>'.**
**7:**
**If the character is '(', ')', ';', or '.'.**
**8:**
**If the character is '*' or '/'.**
**9:**
**If the character is ':'.**
**10:**
**If the character is an alphabetic character.**
**11:**
**If the character is '!'.**
**-1:**
**If the character doesn't match any of the specified types.**

**The function provides a quick and simple way to determine the type of a given character, which can be useful in lexical analysis or similar tasks where character categorization is required.**

## Functionality:

**Returns the type of a given character.**

## Returns:

**An integer representing the type of the character.**

## Usage:

**Used in the state transition function.**

**type_entite(Ec, input):**

**The code:**

```
def type_entite(Ec, input):
    i = ''  # Initialize an empty string for the result
    entite = ''  # Initialize an empty string to store the entity

    # Iterate through each character in the input
    for char in input:
        if char != ' ':
            entite += char  # Build the entity by appending non-space characters

    # Check the entity type based on the code (Ec)
    if Ec == 1:
        # Check conditions for an identifier
        if length(entite) < 7:
            # Check if the identifier is uppercase
            if carecter_majuscule(entite):
                # Check if the identifier is a keyword
                if Key(entite):
                    i += " " + entite + ":entite correct est un identifier \n"
                else:
                    i += " " + entite + ":entite incorrect\n "
            else:
                i += " " + entite + ":entite correct est un identifier \n"
        else:
            i += " " + entite + ":entite incorrect\n "
    elif Ec == 4:
        try:
            # Check conditions for a constant
            if length(entite) < 8 and int(entite) < 1333635:
                i += " " + entite + ": entite est un constante \n"
            else:
                i += " " + entite + ":entite incorrect\n "
        except ValueError:
            i += " " + entite + ":entite incorrect\n "
    elif Ec == 5:
        i += " " + entite + ":entite est un opérateur - \n"
    elif Ec == 6:
        i += " " + entite + ":entite est un opérateur + \n"
    elif Ec == 7:
```

```
elif Ec == 7:
    # Check conditions for a real number
    if length(entite) < 10:
        i += " " + entite + ":entite est un nombre real \n"
    else:
        i += " " + entite + ":entite incorrect\n "
elif Ec == 8:
    # Check conditions for a real number
    if length(entite) < 10:
        i += " " + entite + ":entite est un nombre real \n"
    else:
        i += " " + entite + ":entite incorrect\n "
elif Ec == 12:
    i += " " + entite + ":entite est un opérateur + \n"
elif Ec == 13:
    i += " " + entite + ":entite est un séparateur  \n"
elif Ec == 14:
    i += " " + entite + ":entite est un opérateur \n"
elif Ec == 15:
    i += " " + entite + ":entite est un opérateur \n"
elif Ec == 16:
    i += " " + entite + ":entite est un séparateur  \n"
elif Ec == 17:
    i += " " + entite + ":entite est un opérateur - \n"
elif Ec == 18:
    i += " " + entite + ":entite est un séparateur  \n"
else:
    i += " " + entite + ":entite incorrect\n "

output_text.insert(tk.END, i + "\n")  # Insert the result into the output text
```

**The function type_entite is intended to categorize and display the type of an entity based on a given entity code (Ec) and an input string. Here's a step-by-step breakdown:**

**Input Parameters:**

**Ec:**
**An integer representing the type code for the entity.**
**Input:**
**A string containing the input from which entities are extracted.**

**Variables Initialization:**

**I:**
**An empty string used to accumulate the result or output.**
**Entite:**
**An empty string used to store the current entity being processed.**

**Entity Type Checking:**

**The type of the entity is determined based on the provided Ec code.**
**The code contains multiple branches (if-elif statements) to check different conditions for identifiers, constants, operators, real numbers, and separators.**

**Output String Construction:**

**The variable i is updated with the result message based on the type of entity and whether it is correct or incorrect.**

**Output Insertion:**

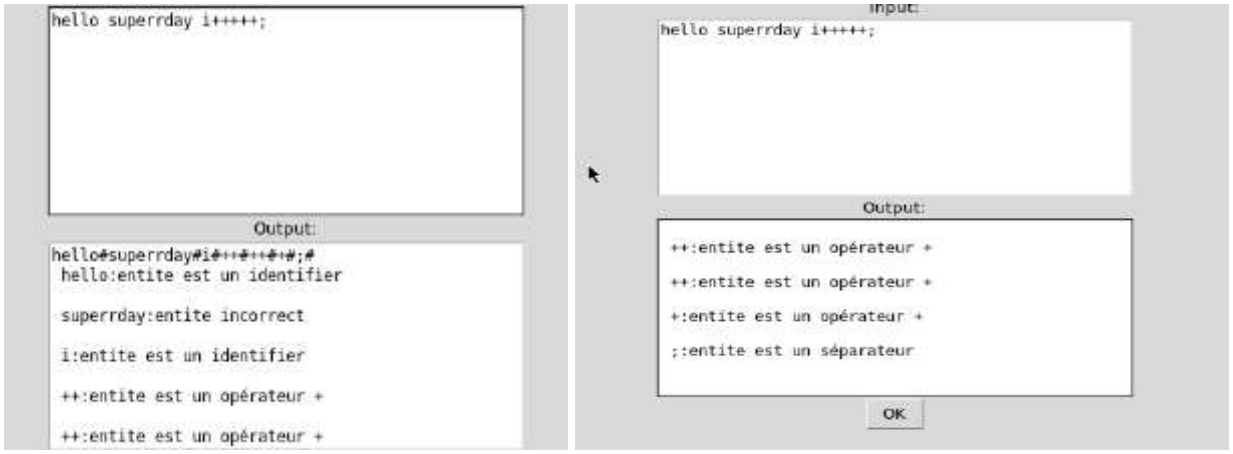
**The result (i) is inserted into the output_text (assuming it's a Tkinter text widget) with a newline character.**
**This function appears to analyze an entity (entite) based on a given code (Ec) and provides feedback on its type and correctness. The feedback messages are then inserted into an output text widget. Note: It relies on external functions like carecter_majuscule and Key, which are assumed to be defined elsewhere.**

## Functionality:

**Classifies an entity based on its type (identifier, constant, operator, etc.).**

## GUI Interaction:

**Insert the classification result into the "Output" text box.**




## Usage:

**Part of the token classification workflow.**

## automate():

## The code:

```
def automate():
    #0,#1,#2,#3,#4,#5,#6,#7,#8,#9,#10,#11,#12
    matrice = [
        [4, 6, 5, 13, 9, -1, 13, 16, 15, 18, 1, -1],  # 0
        [1, -1, -1, -1, -1, 2, -1, -1, -1, -1, 1, -1],  # 1
        [1, -1, -1, -1, -1, 3, -1, -1, -1, -1, 1, -1],  # 2
        [-1, -1, -1, -1, -1, 2, -1, -1, -1, -1, -1, -1],  # 3
        [4, -1, -1, -1, 7, -1, -1, -1, -1, -1, -1, -1],  # 4
        [4, -1, 17, -1, 9, -1, -1, -1, -1, -1, -1, -1],  # 5
        [4, 12, -1, -1, 9, -1, -1, -1, -1, -1, -1, -1],  # 6
        [8, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1],  # 7
        [8, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1],  # 8
        [8, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1],  # 9
        [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1],  # 10
        [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1],  # 11
        [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1],  # 12
        [-1, -1, -1, 14, -1, -1, -1, -1, -1, -1, -1, -1],  # 13
        [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1],  # 14
        [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1],  # 15
        [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1],  # 16
        [-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1],  # 17
        [-1, -1, -1, 14, -1, -1, -1, -1, -1, -1, -1, -1],  # 18
    ]
    seperator_text = process_input_return()
    indice = 0
    ts = ''
    Ec = 0
    for indice in range(length(seperator_text)):
      print(Ec)
      tc = seperator_text[indice]
      if tc != '#':
        ts = ts + seperator_text[indice]
        if Ec != -1 and type_transition(tc) != -1:
          Ec = matrice[Ec][type_transition(tc)]
        # else :ec=-1
      else:
        type_entite(Ec, ts)
        ts = ''
        Ec = 0
```

**The automate function implements a finite automaton that processes input characters, recognizes entities, and determines their types based on transitions between states. It uses a transition matrix (matrice), entity recognition (type_entite), and transition checking (type_transition) functions. The finite automaton is designed to handle input strings with '#' as separators:**

## Transition Matrix (matrice):

**A 2D matrix representing the transitions between states in the finite automaton.**
**Rows correspond to states, and columns correspond to transitions based on input characters.**
**Negative values indicate an invalid transition.**

**Input Processing:**

**seperator_text is assigned the value returned by the process_input_return() function. This function is expected to provide a string containing input characters with '#' as separators.**

**Variables Initialization:**

**Indice:**
**A variable used as an index in the loop.**
**Ts:**
**An empty string used to accumulate characters forming an entity.**
**Ec:**
**A variable representing the current state in the finite automaton.**

**Main Loop:**

**The function iterates through each character in seperator_text.**

**Finite Automaton Transition:**

**If the current character (tc) is not '#', it is appended to the ts string.**
**The function checks if the transition is valid using the type_transition function.**
**If both Ec and type_transition(tc) are not -1, the finite automaton transitions to the next state based on the matrix (matrice).**

**Entity Recognition and Type Checking:**

**When '#' is encountered, the type_entite function is called with the current state (Ec) and the accumulated entity (ts).**
**The type_entite function determines the type of the entity and prints the result.**

**Reset Variables:**

**After processing each entity, ts is reset to an empty string, and Ec is reset to 0.**

## Functionality:
**Implements a finite state automaton for token classification.**

## GUI Interaction:
**Inserts the token classifications into the "Output" text box.**

## Usage:
**Drives the token classification process.**

## process_input_return():

## The code:

```
def process_input_return():
    text = input_text.get("1.0",
                          "end-1c")  # Get the content of the input text box
    processed_text = removing_spaces_comments(text)
    processed_text = separation_text(processed_text)
    return processed_text
```

**The process_input_return function retrieves the content of an input text box, processes it by removing spaces and comments, and further processes it using the separation_text function. The final processed text is then returned from the function. This function plays a role in preparing user input for subsequent processing in a larger program.**

### Input Retrieval:

**text is assigned the content of the input text box using input_text.get("1.0", "end-1c").
input_text.get() retrieves the text content from the input text box in a Tkinter GUI application.
"1.0" specifies that the content should be retrieved from the first character of the first line.
"end-1c" specifies that the content should be retrieved until the character before the last character.**

### Text Processing:

**processed_text is assigned the result of calling two functions:
removing_spaces_comments and separation_text.
The removing_spaces_comments function is expected to remove spaces and comments from the input text.
The separation_text function, as explained in a previous response, inserts '#' symbols to separate different components in the text.**

### Return Value:

**The processed text (processed_text) is returned from the function.**

## Functionality:

**Processes the input text for token classification and returns the result.**

## Returns:

**The processed text.**

## Usage:

**Part of the token classification workflow.**

## process_input_affichage():

## The code:

```
def process_input_affichage():
  text = input_text.get("1.0",
                        "end-1c")  # Get the content of the input text box
  processed_text = removing_spaces_comments(text)
  processed_text = separation_text(processed_text)
  output_text.insert(tk.END, processed_text + "\n")
```

**The process_input_affichage function retrieves the content of an input text box, processes it by removing spaces and comments, and further processes it using the separation_text function. The final processed text is then inserted into an output text box in a Tkinter GUI application. This function facilitates the interactive processing and display of text in the GUI:**

**Input Retrieval:**

**text is assigned the content of the input text box using input_text.get("1.0", "end-1c").**
**input_text.get() retrieves the text content from the input text box in a Tkinter GUI application.**
**"1.0" specifies that the content should be retrieved from the first character of the first line.**
**"end-1c" specifies that the content should be retrieved until the character before the last character.**

**Text Processing:**

**processed_text is assigned the result of calling two functions: removing_spaces_comments and separation_text.**
**The removing_spaces_comments function is expected to remove spaces and comments from the input text.**
**The separation_text function, as explained in a previous response, inserts '#' symbols to separate different components in the text.**

**Output Insertion:**

**The processed text (processed_text) is inserted into the output text box using output_text.insert(tk.END, processed_text + "\n").**
**output_text.insert() is a,. Tkinter method for inserting text into a text widget.**
**tk.END specifies that the text should be inserted at the end of the text widget.**
**processed_text + "\n" appends a newline character to the processed text before inserting it.**

## Functionality:

**Processes the input text for display in the "Output" text box.**

## GUI Interaction:

**Insert the processed text into the "Output" text box.**

## Usage:

**Prepares the text for display.**

## output():

## The code:

```
def output():
  output_text.delete("1.0", "end")
  process_input_affichage()
  automate()
```

**The output function is responsible for clearing the output text widget, processing the input text using process_input_affichage, and potentially triggering further automation or processing through the automate function. This function is likely associated with a GUI application for interactive input processing and output display:**

### Clear the Output Text Widget:

**output_text.delete("1.0", "end") is used to delete the content in the output text widget.**
**"1.0" specifies the starting position (row 1, column 0) from where the deletion begins.**
**"end" specifies the ending position, deleting everything until the end of the text in the widget.**

### Process Input and Display Result:

**process_input_affichage() is called to process the input text and display the result in the output text widget.**
**This involves retrieving the input text, removing spaces and comments, and inserting the processed text into the output text widget.**

### Automate:

**automate() is called after processing the input text.**
**The purpose of this call depends on the definition of the automate function, which is not provided in the given code snippet. Presumably, it performs some additional processing or automation based on the processed input.**

### Functionality:

**Clears the "Output" text box and triggers the processing of input text.**

### GUI Interaction:

**Invoked when the user clicks the "OK" button.**

### Usage:

**Initiates the overall processing workflow.**

# 6.5. GUI Setup Functions:

**GUI Setup and Main Loop**

### root = tk.Tk():

**Creates the main window.**

### menu, file_menu:
**Create menus for file-related options.**

### input_label, input_text:
**Create labels and text boxes for input.**

### output_label, output_text:
**Create labels and text boxes for output.**

### process_input_button:
**Create a button for processing input.**

### root.mainloop():
**Starts the main event loop for the GUI.**

## 6.6.Overall Workflow:

**1.User interacts with the GUI by opening, editing, and saving files.**
**2.When the user clicks the "OK" button, the input text is processed:**

- **Spaces and comments are removed.**
- **Text is tokenized using predefined separators.**
- **Tokens are classified using a finite state automaton.**
- **Classification results are displayed in the "Output" text box.**

# 7.Conclusion:

**In conclusion, the project successfully aimed to create a compiler for a defined language, covering various aspects of compiler construction. The use of Python and Tkinter facilitated the development of an interactive and user-friendly GUI. The implemented functionalities, from file handling to text processing and token classification, demonstrated a holistic approach to compiler development. However, it's important to note that the provided code may have some issues or incomplete parts that need further refinement for a fully functional compiler. Overall, the project provided valuable insights into the complexities of compiler construction and language processing.**