Mini-projet

Analyseur lexicale

Étape 1 : Analyse Lexicale en Python

1.1 Définition des Tokens

Nous définirons les motifs des tokens en utilisant des expressions régulières. Voici les tokens définis selon la grammaire donnée :

- Délimiteurs de programme : **@PROG**, **PROG@**
- Délimiteurs de déclaration : **@DECL**, **DECL@**
- Délimiteurs de corps : **@CORPS**, **CORPS@**
- Mots-clés : **ECRIRE**, **FOR**, **ENTIER**, **REEL**, **CARACTERE**, **TABLEAU**, **CHAINE**
- Opérateurs : **\+\+**, **:=**, **ADD**, **SOUS**, **MULT**, **<**, **<=**, **>**, **>=**, **==**, **<>**
- Identifiants : **%\d+**
- Nombres : **\d+** (entiers), **\d+\.\d+** (réels)
- Chaines de caractères : **\"[^\"]*\"**
- Symboles de ponctuation : **\(**, **\)**, **\[**, **\]**, **,**, **;**

1.2 Code de l'Analyseur Lexical :

```
import re

import sys


class Token:

  def __init__(self, lexeme, token):

    self.lexeme = lexeme

    self.token = token


  def __str__(self):

    return f"<{self.token}, {self.lexeme}>"
```

```
def tokenize(input_text):
    token_patterns = {
        "PROG_START": r"@PROG",
        "PROG_END": r"PROG@",
        "DECL_START": r"@DECL",
        "DECL_END": r"DECL@",
        "CORPS_START": r"@CORPS",
        "CORPS_END": r"CORPS@",
        "ECRIRE": r"ECRIRE",
        "FOR": r"FOR",
        "ENTIER": r"ENTIER",
        "REEL": r"REEL",
        "CARACTERE": r"CARACTERE",
        "TABLEAU": r"TABLEAU",
        "CHAINE": r"CHAINE",
        "INCREMENT": r"\+\+",
        "AFFECT": r":=",
        "ADD": r"ADD",
        "SOUS": r"SOUS",
        "MULT": r"MULT",
        "OPREL": r"<=|>=|<>|==|>|<",
        "IDENT": r"%\d+",
        "NBREEL": r"\d+\.\d+",
        "NBENTIER": r"\d+",
        "STRING": r"\"[^\"]*\"",
        "LPAREN": r"\(",
        "RPAREN": r"\)",
        "LBRACKET": r"\[",
        "RBRACKET": r"\]",
        "COMMA": r",",
```

```
    "SEMICOLON": r";",
  }

  patterns = '|'.join(f"(?P<{name}>{pattern})" for name, pattern in token_patterns.items())
  compiled_pattern = re.compile(patterns)

  tokens = []
  for match in compiled_pattern.finditer(input_text):
    for name, _ in token_patterns.items():
      if match.group(name):
        tokens.append(Token(match.group(name), name))
        break
  return tokens

def main():
  if len(sys.argv) != 2:
    print("Usage: python LexicalAnalyzer.py <source-file>")
    sys.exit(1)

  source_file_path = sys.argv[1]
  print(f"Reading source file from: {source_file_path}")

  try:
    with open(source_file_path, 'r', encoding='utf-8') as file:
      code = file.read()
  except FileNotFoundError:
    print(f"Source file not found: {source_file_path}")
    sys.exit(1)
  except IOError as e:
    print(f"Error reading source file: {e}")
    sys.exit(1)
```

```
    print("Source file read successfully. Starting tokenization...")

    tokens = tokenize(code)
    for token in tokens:
        print(token)

    try:
        with open("output.lex", 'w', encoding='utf-8') as file:
            for token in tokens:
                file.write(str(token) + '\n')
    except IOError as e:
        print(f"Error writing to output file: {e}")

    print("Tokenization completed. Tokens written to output.lex.")

if __name__ == "__main__":
    main()
```

Explication du Code

1. **Importation des Modules :**
   - **re**: pour les expressions régulières.
   - **sys**: pour les arguments de la ligne de commande.
2. **Classe Token :**
   - Représente un token avec deux attributs : **lexeme** et **token**.
   - Méthode **__str__** redéfinie pour afficher le token sous la forme **"<type, lexeme>"**.
3. **Fonction tokenize :**
   - Définit les motifs des tokens en utilisant des expressions régulières.
   - Compile les motifs en une expression régulière combinée.
   - Utilise cette expression régulière pour trouver et identifier les tokens dans le texte d'entrée.
   - Retourne une liste de tokens.

4. **Fonction main :**
    - Vérifie les arguments de la ligne de commande pour s'assurer qu'un seul argument est fourni (le chemin du fichier source).
    - Lit le fichier source et gère les exceptions éventuelles.
    - Utilise la fonction **tokenize** pour identifier les tokens.
    - Affiche les tokens et les écrit dans un fichier de sortie **output.lex**.

Étape 2 : Document de Conception

1. **Structure de Données :**
    - La classe **Token** pour représenter les tokens.
    - Un dictionnaire pour stocker les motifs des tokens et leurs expressions régulières.
2. **Algorithme :**
    - Utilisation d'expressions régulières pour définir les motifs de tokenisation.
    - Lecture du fichier source en entier pour le tokeniser.
    - Utilisation de la fonction **finditer** de **re** pour trouver toutes les occurrences des motifs dans le texte.
    - Stockage des tokens dans une liste et écriture dans un fichier de sortie.

Étape 3 : Code Source, Code Exécutable et Jeux de Tests

1. **Code Source :**
    - Le script Python fourni ci-dessus est le code source de l'analyseur lexical.
2. **Code Exécutable :**
    - Pour exécuter le script, utilisez la commande suivante :

      python LexicalAnalyzer.py <source-file>

3.Jeux de Tests :

- Créez des fichiers **source.new** avec du code conforme à la grammaire du langage NEW pour tester l'analyseur lexical.
- Par exemple :
- @PROG
- @DECL %1:ENTIER, %2:REEL, %3:CARACTERE DECL@

- @CORPS ECRIRE("Hello, World!"); %1:=10; %2:=20.5; %3++ CORPS@
- PROG@