

ASSIGNMENT #02

ASSIGNMENT OF: Compiler Construction

COURSE: BSCS-01 (B).

SUBMIT TO: Ma’am Reeda.

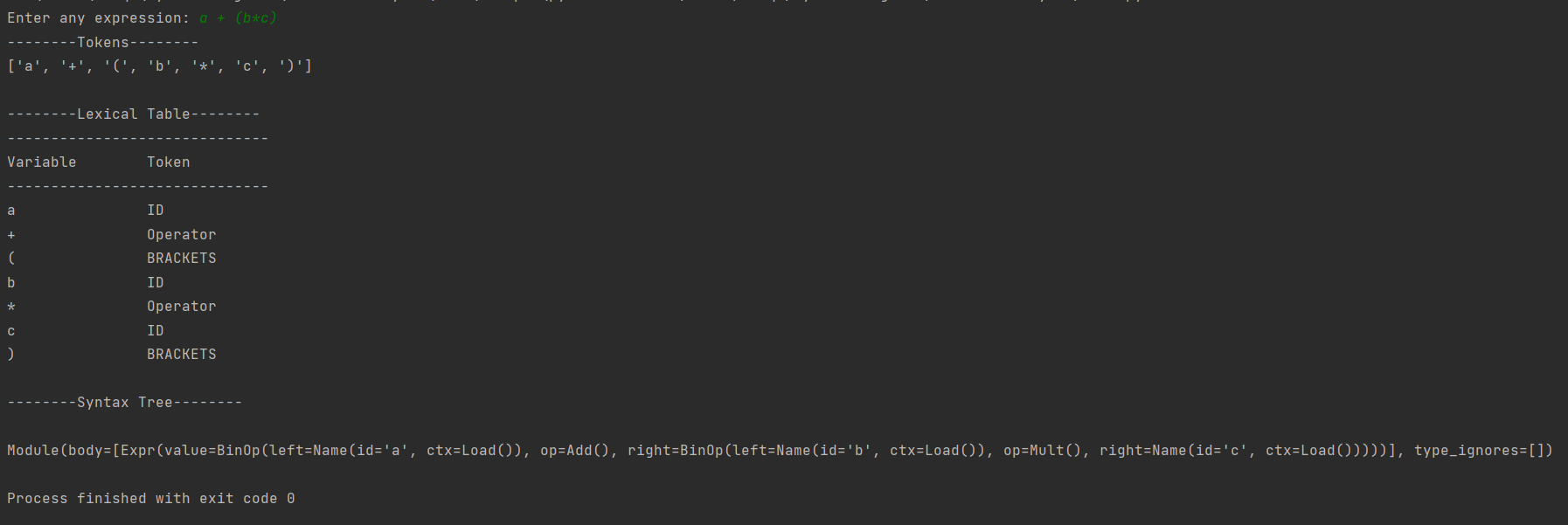
SYBMIT BY: Furqan Ali (200901073).



December 30, 2022

|  |
| --- |
| **CODE:**  import re import ast  # Regular expression for integers INTEGER\_REGEX = r"\b\d+\b" # Regular expression for float FLOAT\_REGEX = r"\b\d\*\.\d+\b" # Regular expression for operator or any non alphanumeric data type OPERATOR\_REGEX = r"\W" # Regular expression for alphabets ID\_REGEX = r"\b[a-zA-Z]\w\*\b" # Regular expression for special characters SPECIAL\_CHAR\_REGEX = r"\b[!@#$%&]\b"  # Module-1 def lexical\_analyzer(input\_string):  tokens = []  search=re.finditer(f"{INTEGER\_REGEX}|{FLOAT\_REGEX}|{ID\_REGEX}|{OPERATOR\_REGEX}|{SPECIAL\_CHAR\_REGEX}", input\_string)  for match in search:  if match.group(0).isdigit():  token\_type = "INTEGER"  elif "." in match.group(0):  token\_type = "FLOAT"  elif match.group(0).isalpha():  token\_type = "ID"  elif "+" in match.group(0) or "-" in match.group(0) or "\*" in match.group(0) or "/" in match.group(0) or "^" in match.group(0) or "=" in match.group(0):  token\_type = "Operator"  elif " " in match.group(0):  continue  elif "!" in match.group(0) or "@" in match.group(0) or "#" in match.group(0) or "$" in match.group(0)or "%" in match.group(0) or "&" in match.group(0):  token\_type="Special Character"  elif "(" in match.group(0) or ")" in match.group(0) or "{" in match.group(0) or "}" in match.group(0) or "[" in match.group(0) or"]" in match.group(0):  token\_type = "BRACKETS"  tokens.append((token\_type, match.group(0)))  return tokens  def display(tokens):  token1 = []  for token in tokens:  x = token[1]  token1.append(x)   print("--------Tokens--------")  print(token1)  print("\n--------Lexical Table--------")  print("-" \* 30)  print("Variable".ljust(15), "Token".ljust(15))  print("-" \* 30)  for token in tokens:  print(token[1].ljust(15), token[0].ljust(15))   return 0  def Syntax\_tree(input\_String):  print("\n--------Syntax Tree--------\n")  try:  Syntax\_tree1 = ast.parse(input\_string)  print(ast.dump(Syntax\_tree1))  except:  print("Syntac Error try again")   return 0  if \_\_name\_\_ == "\_\_main\_\_":  input\_string = input("Enter any expression: ")  tokens = lexical\_analyzer(input\_string)  display(tokens)  Syntax\_tree(input\_string) |

**OUTPUT:**

****

**Text

Description automatically generated**

**Text

Description automatically generated**

**Explanation:**

At first we have created **integer, float, id, operator** and **special character** regular expression which is going to help us in the tokenization process.

The **re.finditer()** function is used to iterate over all the matches in the input string. For each match, the **group(0)** method is used to extract the matched text. Then, a simple conditional statement is used to determine the type of the token based on the matched text.

Each token and its type is appended in a list which is later use in display function to print tokens and lexical table on the console.

To create a syntax tree the **AST (Abstract Syntax Tree)** library is used, the expression that was taken by the user is then parsed using the ast.parse function.

As **AST** does not provide us with visualization facilities we simply use the ast.dump function on the parsed expression to return a string representation of the syntax tree.

**GIT-HUB:**

* *FURQANALI111 - Overview* (no date) *GitHub*. Available at: https://github.com/Furqanali111 (Accessed: December 31, 2022).