LEXICAL ANALYZER IMPLEMENTATION

TEXT FILE:

```
##
Define a class to represent a simple bank account
##
class BankAccount {
     restricted  StrChar name   # Account holder's name (instance variable)
     restricted  num balance    # Account balance (instance variable)

      # Constructor to initialize account holder's name and balance
     universal BankAccount(StrChar accName, num initialBalance = 0) {
         name = accName
         balance = initialBalance
      }

      # Method to deposit money
      universal void deposit(num amount) {
         when (amount > 0) {
            balance = balance + amount # Add deposit amount to balance
            display ("Deposited" + amount + ". New balance is " + balance + "." )
         } otherwise {
            display ("Deposit amount must be positive.")
         }
      }

      # Method to withdraw money
      universal void withdraw(num amount) {
         when (amount > 0 AND amount <= balance) {
            balance = balance - amount  # Subtract withdrawal amount from balance
            display ( "Withdraw " + amount + ". New balance is " + balance + ".")
         } otherwise {
            display ( "Invalid withdrawal amount." )
         }
      }

      # Method to display account balance
      universal void display_balance() {
         display ("Account holder: " + name + ", Balance: " + balance )
      }
}

# Main function to create a BankAccount object and perform operations
universal void objectCreate() {
    # Create a BankAccount object for John with an initial balance of 100
    BankAccount account=new BankAccount("John", 100)

    # Use a loop to perform multiple transactions
    num i=0
    while ( i < 3) {
```

```
        display ("\nTransaction " + i + 1 )

        # Deposit 50 into the account
        account.deposit(50)

        # Withdraw 30 from the account
        account.withdraw(30)

        # Display the current balance
        account.display_balance()

        i++
    }
}
```

CODE:

```
import re

class Token:  # token class
    def __init__(self, value_part, class_part, line_number):
        self.value_part = value_part
        self.class_part = class_part
        self.line_number = line_number

    def __repr__(self):  # return values for printing
        return f"Token(value='{self.value_part}', type='{self.class_part}',
line={self.line_number})"

def Validate_string(temp):  # validate function
    A = r"[\\|'|\"]"  # can not occur without /
    B = r"[bntro]"  # can and can not occur with backslash /
    C = r"[@+.]"  # do not require a backslash
    D = r"[a-zA-Z\s+_]"
    char_const = rf"(\\{A}|\\{B}|{B}|{C}|{D})"
    # string and character RE
    strchar_pattern = rf"^\"({char_const})*\"$"
    # number RE include int and float
    number_pattern = r'^[0-9]+$|^[+-]?[0-9]*\.[0-9]+$'
    # identifier RE start with alphabet or underscore and end with alpha or digit
    identifier_pattern = r'[a-zA-Z]|^[a-zA-Z_][a-zA-Z0-9_]*[a-zA-Z0-9]$'
    # dic for operators
    operator_list = {
        "+": "PM",  # PM=Plus Minus
        "-": "PM",
        "*": "MDM",  # MDM=Multiply Divide Modulo
        "/": "MDM",
        "%": "MDM",
        "<": "ROP",  # ROP=Relational Operator
```

```
        ">": "ROP",
        "<=": "ROP",
        ">=": "ROP",
        "!=": "ROP",
        "==": "ROP",
        "++": "Inc_Dec",
        "--": "Inc_Dec",
        "=": "="
    }
    # dict for keywords
    keywords_list = {
        "class": "class",
        "universal": "AM",  # AM=Access Modifier
        "restricted": "AM",
        "void": "void",
        "ext": "extends",
        "ret": "return",
        "this": "this",
        "new": "new",
        "final": "final",
        "num": "DT",  # DT=Data Type
        "StrChar": "DT",
        "when": "when",
        "otherwise": "else",
        "input": "input",
        "display": "print",
        "while": "while",
        "brk": "break",
        "cont": "continue",
        "try": "try",
        "catch": "catch",
        "finally": "finally",
        "NOT": "NOT",
        "AND": "AND",
        "OR": "OR"
    }
    # dict for punctuators
    punctuator_list = {
        "{": "{",
        "}": "}",
        "(": "(",
        ")": ")",
        "[": "[",
        "]": "]",
        ".": ".",
        ",": ",",
        ";": ";"
    }
```

```
    # matching temp with all scenarios
    if re.match(strchar_pattern, temp):
        return "StrChar"
    elif re.match(number_pattern, temp):
        return "num"
    elif temp in operator_list:
        return operator_list.get(temp)
    elif temp in punctuator_list:
        return punctuator_list.get(temp)
    elif temp in keywords_list:
        return keywords_list.get(temp)
    elif re.match(identifier_pattern, temp):
        return "ID"
    else:
        return "Invalid Lexeme"

def break_word(file):
    temp = ""  # to store a complete word
    punct_array = [",", ".", "[", "]", "{", "}", "(", ")", ";"]
    opr_array = ["*", "/", "%"]
    check_opr_array = ["+", "-", "=", ">", "<", "!"]
    line_number = 1

    index = 0
    while index < len(file):  # iterate until the file ends
        char = file[index]  # reading file char by char

        # Handle spaces and new lines
        if char.isspace():
            if temp:
                cp = Validate_string(temp.strip())
                yield Token(temp.strip(), cp, line_number)
                temp = ""
            if char == "\n":
                line_number += 1
            index += 1
            continue

        # Handle comments
        if char == "#":
            if index + 1 < len(file) and file[index + 1] == "#":  # check for
multiline comment
                index += 2  # Skip the ##
                while index + 1 < len(file) and not (file[index] == "#" and
file[index + 1] == "#"):
                    if file[index] == "\n":
                        line_number += 1
                    index += 1
```

```
                if index + 1 < len(file) and file[index] == "#" and file[index + 1]
== "#":
                    index += 2  # Skip the ##
                continue
            else:
                while index < len(file) and file[index] != "\n":
                    index += 1
                continue

        # Handle operators
        if char in opr_array or char in check_opr_array:
            if temp:
                cp = Validate_string(temp.strip())
                yield Token(temp.strip(), cp, line_number)
                temp = ""
            if char == "+" and index + 1 < len(file) and file[index + 1] == "+":
                cp = Validate_string("++")
                yield Token("++", cp, line_number)  # increment operator
                index += 1
            elif char == "-" and index + 1 < len(file) and file[index + 1] == "-":
                cp = Validate_string("--")
                yield Token("--", cp, line_number)  # decrement operator
                index += 1
            elif char == "=" and index + 1 < len(file) and file[index + 1] == "=":
                cp = Validate_string("==")
                yield Token("==", cp , line_number)
                index += 1
            elif char == "<" and index + 1 < len(file) and file[index + 1] == "=":
                cp = Validate_string("<=")
                yield Token("<=", cp, line_number)
                index += 1
            elif char == ">" and index + 1 < len(file) and file[index + 1] == "=":
                cp = Validate_string(">=")
                yield Token(">=", cp, line_number)
                index += 1
            elif char == "!" and index + 1 < len(file) and file[index + 1] == "=":
                cp = Validate_string("!=")
                yield Token("!=", cp, line_number)
                index += 1
            else:
                yield Token(char, Validate_string(char), line_number)
        elif char in punct_array:
            if temp:
                cp = Validate_string(temp.strip())
                yield Token(temp.strip(), cp, line_number)
                temp = ""
            yield Token(char, char, line_number)
        # handle string or char
```

```
        elif char == "\"":
            if temp:
                cp = Validate_string(temp.strip())
                yield Token(temp.strip(), cp, line_number)
                temp = ""
            quote_type = char  # store "
            temp += char
            index += 1
            start_line = line_number
            while index < len(file):
                char = file[index]
                temp += char
                if char == "\\" and index + 1 < len(file):  # Handle escape
sequences
                    temp += file[index + 1]
                    index += 1
                elif char == quote_type:  # Found the closing quote
                    break
                if char == "\n":
                    line_number += 1
                index += 1
            cp = Validate_string(temp.strip())
            yield Token(temp.strip(), cp, start_line)
            temp = ""
        # not a break character
        else:
            temp += char

        index += 1  # increment index

    if temp:
        cp = Validate_string(temp.strip())
        yield Token(temp.strip(), cp, line_number)

# file reading
with open("D:\\6th sem\\compiler lab\\project\\file.txt", "r") as f:
    file = f.read()

# Tokenize and print tokens
for token in break_word(file):
    print(token)
```

OUTPUT:

Token(value='class', type='class', line=4)
Token(value='BankAccount', type='ID', line=4)
Token(value='{', type='{', line=4)

Token(value='restricted', type='AM', line=5)
Token(value='StrChar', type='DT', line=5)
Token(value='name', type='ID', line=5)
Token(value='restricted', type='AM', line=6)
Token(value='num', type='DT', line=6)
Token(value='balance', type='ID', line=6)
Token(value='universal', type='AM', line=9)
Token(value='BankAccount', type='ID', line=9)
Token(value='(', type='(', line=9)
Token(value='StrChar', type='DT', line=9)
Token(value='accName', type='ID', line=9)
Token(value=',', type=',', line=9)
Token(value='num', type='DT', line=9)
Token(value='initialBalance', type='ID', line=9)
Token(value='=', type='=', line=9)
Token(value='0', type='num', line=9)
Token(value=')', type=')', line=9)
Token(value='{', type='{', line=9)
Token(value='name', type='ID', line=10)
Token(value='=', type='=', line=10)
Token(value='accName', type='ID', line=10)
Token(value='balance', type='ID', line=11)
Token(value='=', type='=', line=11)
Token(value='initialBalance', type='ID', line=11)
Token(value='}', type='}', line=12)
Token(value='universal', type='AM', line=15)
Token(value='void', type='void', line=15)
Token(value='deposit', type='ID', line=15)
Token(value='(', type='(', line=15)
Token(value='num', type='DT', line=15)
Token(value='amount', type='ID', line=15)
Token(value=')', type=')', line=15)
Token(value='{', type='{', line=15)
Token(value='when', type='when', line=16)
Token(value='(', type='(', line=16)
Token(value='amount', type='ID', line=16)
Token(value='>', type='ROP', line=16)
Token(value='0', type='num', line=16)
Token(value=')', type=')', line=16)
Token(value='{', type='{', line=16)
Token(value='balance', type='ID', line=17)
Token(value='=', type='=', line=17)
Token(value='balance', type='ID', line=17)
Token(value='+', type='PM', line=17)
Token(value='amount', type='ID', line=17)
Token(value='display', type='print', line=18)
Token(value='(', type='(', line=18)
Token(value="'Deposited'", type='StrChar', line=18)
Token(value='+', type='PM', line=18)
Token(value='amount', type='ID', line=18)
Token(value='+', type='PM', line=18)
Token(value="'. New balance is '", type='StrChar', line=18)
Token(value='+', type='PM', line=18)
Token(value='balance', type='ID', line=18)
Token(value='+', type='PM', line=18)

Token(value='".', type='StrChar', line=18)
Token(value=')', type=')', line=18)
Token(value='}', type='}', line=19)
Token(value='otherwise', type='else', line=19)
Token(value='{', type='{', line=19)
Token(value='display', type='print', line=20)
Token(value='(', type='(', line=20)
Token(value='"Deposit amount must be positive."', type='StrChar', line=20)
Token(value=')', type=')', line=20)
Token(value='}', type='}', line=21)
Token(value='}', type='}', line=22)
Token(value='universal', type='AM', line=25)
Token(value='void', type='void', line=25)
Token(value='withdraw', type='ID', line=25)
Token(value='(', type='(', line=25)
Token(value='num', type='DT', line=25)
Token(value='amount', type='ID', line=25)
Token(value=')', type=')', line=25)
Token(value='{', type='{', line=25)
Token(value='when', type='when', line=26)
Token(value='(', type='(', line=26)
Token(value='amount', type='ID', line=26)
Token(value='>', type='ROP', line=26)
Token(value='0', type='num', line=26)
Token(value='AND', type='AND', line=26)
Token(value='amount', type='ID', line=26)
Token(value='<=', type='ROP', line=26)
Token(value='balance', type='ID', line=26)
Token(value=')', type=')', line=26)
Token(value='{', type='{', line=26)
Token(value='balance', type='ID', line=27)
Token(value='=', type='=', line=27)
Token(value='balance', type='ID', line=27)
Token(value='-', type='PM', line=27)
Token(value='amount', type='ID', line=27)
Token(value='display', type='print', line=28)
Token(value='(', type='(', line=28)
Token(value='"Withdraw "', type='StrChar', line=28)
Token(value='+', type='PM', line=28)
Token(value='amount', type='ID', line=28)
Token(value='+', type='PM', line=28)
Token(value='". New balance is "', type='StrChar', line=28)
Token(value='+', type='PM', line=28)
Token(value='balance', type='ID', line=28)
Token(value='+', type='PM', line=28)
Token(value='".', type='StrChar', line=28)
Token(value=')', type=')', line=28)
Token(value='}', type='}', line=29)
Token(value='otherwise', type='else', line=29)
Token(value='{', type='{', line=29)
Token(value='display', type='print', line=30)
Token(value='(', type='(', line=30)
Token(value='"Invalid withdrawal amount."', type='StrChar', line=30)
Token(value=')', type=')', line=30)
Token(value='}', type='}', line=31)

Token(value='}', type='}', line=32)
Token(value='universal', type='AM', line=35)
Token(value='void', type='void', line=35)
Token(value='display_balance', type='ID', line=35)
Token(value='(', type='(', line=35)
Token(value=')', type=')', line=35)
Token(value='{', type='{', line=35)
Token(value='display', type='print', line=36)
Token(value='(', type='(', line=36)
Token(value='"Account holder: "', type='Invalid Lexeme', line=36)
Token(value='+', type='PM', line=36)
Token(value='name', type='ID', line=36)
Token(value='+', type='PM', line=36)
Token(value='", Balance: "', type='Invalid Lexeme', line=36)
Token(value='+', type='PM', line=36)
Token(value='balance', type='ID', line=36)
Token(value=')', type=')', line=36)
Token(value='}', type='}', line=37)
Token(value='}', type='}', line=38)
Token(value='universal', type='AM', line=41)
Token(value='void', type='void', line=41)
Token(value='objectCreate', type='ID', line=41)
Token(value='(', type='(', line=41)
Token(value=')', type=')', line=41)
Token(value='{', type='{', line=41)
Token(value='BankAccount', type='ID', line=43)
Token(value='account', type='ID', line=43)
Token(value='=', type='=', line=43)
Token(value='new', type='new', line=43)
Token(value='BankAccount', type='ID', line=43)
Token(value='(', type='(', line=43)
Token(value='"John"', type='StrChar', line=43)
Token(value=',', type=',', line=43)
Token(value='100', type='num', line=43)
Token(value=')', type=')', line=43)
Token(value='num', type='DT', line=46)
Token(value='i', type='ID', line=46)
Token(value='=', type='=', line=46)
Token(value='0', type='num', line=46)
Token(value='while', type='while', line=47)
Token(value='(', type='(', line=47)
Token(value='i', type='ID', line=47)
Token(value='<', type='ROP', line=47)
Token(value='3', type='num', line=47)
Token(value=')', type=')', line=47)
Token(value='{', type='{', line=47)
Token(value='display', type='print', line=48)
Token(value='(', type='(', line=48)
Token(value='"\nTransaction "', type='StrChar', line=48)
Token(value='+', type='PM', line=48)
Token(value='i', type='ID', line=48)
Token(value='+', type='PM', line=48)
Token(value='1', type='num', line=48)
Token(value=')', type=')', line=48)
Token(value='account', type='ID', line=51)

Token(value='.', type='.', line=51)
Token(value='deposit', type='ID', line=51)
Token(value='(', type='(', line=51)
Token(value='50', type='num', line=51)
Token(value=')', type=')', line=51)
Token(value='account', type='ID', line=54)
Token(value='.', type='.', line=54)
Token(value='withdraw', type='ID', line=54)
Token(value='(', type='(', line=54)
Token(value='30', type='num', line=54)
Token(value=')', type=')', line=54)
Token(value='account', type='ID', line=57)
Token(value='.', type='.', line=57)
Token(value='display_balance', type='ID', line=57)
Token(value='(', type='(', line=57)
Token(value=')', type=')', line=57)
Token(value='i', type='ID', line=59)
Token(value='++', type='Inc_Dec', line=59)
Token(value='}', type='}', line=60)
Token(value='}', type='}', line=61)