HW4

WorkType	HW
Due Date	@April 10, 2023
Status	Not started

Design a recursive decent algorithm for the following Grammar that takes in a list of tokens and returns true if the token list is in the language:
30 POINT | TOKENIZE THE TERMINALS, WRITING GOOD CODE IN OBJECT ORIENTED FASHIONED THAT TAKES IN LIST OF TOKENS

🐱 GIT : PalProgLingistics/pl4330_hw4 at master · Palpeleno/PalProgLingistics (github.com)

token

```
tokens = [
    Token('IF', 'if'),
    Token('LPAREN', '('),
    Token('ID', 'x'),
    Token('GT', '>'),
    Token('INT_LIT','0'),
    Token('RPAREN', ')'),
    Token('LBRACE','{'),
    Token('ID', 'x'),
    Token('ASSIGN', '='),
    Token('ID', 'y'),
    Token('SEMICOLON', ';'),
    Token('RBRACE', '}')
]
```

10 points | <STMT> --> <IF_STMT> | <BLOCK> | <EXPR> | <WHILE_LOOP>

```
def stmt(self):
        if self.match('IF'):
            self.expect('LPAREN')
            self.bool_expr()
             self.expect('RPAREN')
            if self.match('LBRACE'):
                self.stmt_list()
                self.expect('RBRACE')
            else:
                self.stmt()
            if self.match('ELSE'):
                if self.match('LBRACE'):
                    self.stmt_list()
                    self.expect('RBRACE')
                else:
                    self.stmt()
        elif self.match('LBRACE'):
            self.stmt_list()
            self.expect('RBRACE')
        elif self.match('WHILE'):
            self.expect('LPAREN')
            self.bool_expr()
            self.expect('RPAREN')
            if self.match('LBRACE'):
                self.stmt_list()
                self.expect('RBRACE')
            else:
                self.stmt()
        else:
            self.expr()
```

5 points | <STMT_LIST> --> { <STMT> `;` }

```
def stmt_list(self):
         while self.tokens[self.pos].type != 'RBRACE':
            self.stmt()
            self.expect('SEMICOLON')
```

5 points | <WHILE_LOOP> --> `while` `(` <BOOL_EXPR> `)` (<STMT> `;` | <BLOCK>)

```
elif self.match('WHILE'):
            self.expect('LPAREN')
            self.bool_expr()
            self.expect('RPAREN')
            if self.match('LBRACE'):
                self.stmt_list()
                self.expect('RBRACE')
            else:
                self.stmt()
```

10 points | <IF_STMT> --> `if` `(` <BOOL_EXPR> `)` (<STMT> `;` | <BLOCK>) [`else` (<STMT> `;` | <BLOCK>)]

```
   if self.match('IF'):
            self.expect('LPAREN')
            self.bool_expr()
             self.expect('RPAREN')
            if self.match('LBRACE'):
                self.stmt_list()
                self.expect('RBRACE')
            else:
                self.stmt()
            if self.match('ELSE'):
                if self.match('LBRACE'):
                    self.stmt_list()
                    self.expect('RBRACE')
                else:
                    self.stmt()
        elif self.match('LBRACE'):
            self.stmt_list()
            self.expect('RBRACE')
```

10 points | <BLOCK> --> `{` <STMT_LIST> `}`

```
if self.match('LBRACE'):
                self.stmt_list()
                self.expect('RBRACE')
```

15 points |<EXPR> --> <TERM> {(`+`|`-`) <TERM>}

```
def expr(self):
        self.term()
        while self.match('PLUS') or self.match('MINUS'):
            self.term()
```

<TERM> --> <FACT> {(`*`|`/`|`%`) <FACT>}

```
def term(self):
        self.fact()
        while self.match('MULT') or self.match('DIV') or self.match('MOD'):
            self.fact()
```

15 points |<BOOL_EXPR> --> <BTERM> {(`>`|`<`|`>=`|`<=`) <BTERM>}

```
def bool_expr(self):
        self.bterm()
        while self.match('LT') or self.match('GT') or self.match('LE') or self.match('GE'):
            self.bterm()
```

<BTERM> --> <BAND> {(`==`|`!=`) <BAND>}

```
def bterm(self):
        self.band()
        while self.match('EQ') or self.match('NE'):
            self.band()
```

<BAND> --> <BOR> {`&&` <BOR>}

```
def band(self):
        self.bor()
        while self.match('AND'):
            self.bor()
```

<BOR> --> <EXPR> {`&&` <EXPR>}

```
    def bor(self):
        self.expr()
        while self.match('OR'):
            self.expr()
```

<FACT> --> ID | INT_LIT | FLOAT_LIT | `(` <EXPR> `)`

```
def fact(self):
        if self.match('ID') or self.match('INT_LIT') or self.match('FLOAT_LIT'):
            pass
        elif self.match('LPAREN'):
            self.expr()
            self.expect('RPAREN')
        else:
            raise Exception(f"Expected identifier, integer literal, or floating point literal, found '{self.tokens[self.pos].value}'")
```