NAMA : THEODORE GABBELAMBOK SITUMORANG
NPM : 51421476
KELAS : 4IA27

Teknik Kompilasi
Pertemuan 3

INPUT

```
# Lexer: Mengubah input string menjadi daftar token
class Lexer:
    def __init__(self, input_text):
        self.input_text = input_text
        self.pos = 0

    def get_next_token(self):
        if self.pos >= len(self.input_text):
            return None
        current_char = self.input_text[self.pos]

        # Mengabaikan spasi
        if current_char.isspace():
            self.pos += 1
            return self.get_next_token()

        # Memeriksa apakah karakter adalah digit
        if current_char.isdigit():
            number = ''
            while self.pos < len(self.input_text) and
self.input_text[self.pos].isdigit():
                number += self.input_text[self.pos]
                self.pos += 1
            return ('NUMBER', int(number))

        if current_char == '+':
            self.pos += 1
            return ('PLUS', '+')

        if current_char == '*':
            self.pos += 1
            return ('MULTIPLY', '*')

        # Jika karakter tidak dikenali, lempar error
```

```
        raise ValueError(f"Unknown character: {current_char}")

# Parser: Membangun AST dari token yang dihasilkan lexer
class Parser:
    def __init__(self, lexer):
        self.lexer = lexer
        self.current_token = self.lexer.get_next_token()

    def eat(self, token_type):
        if self.current_token is not None and self.current_token[0] ==
token_type:
            self.current_token = self.lexer.get_next_token()
        else:
            raise ValueError(f"Unexpected token: {self.current_token}")

    def factor(self):
        # factor: NUMBER
        token = self.current_token
        self.eat('NUMBER')
        return ('NUMBER', token[1])

    def term(self):
        # term: factor ((MULTIPLY factor)*)
        node = self.factor()
        while self.current_token is not None and self.current_token[0] ==
'MULTIPLY':
            self.eat('MULTIPLY')
            node = ('MULTIPLY', node, self.factor())
        return node

    def expr(self):
        # expr: term ((PLUS term)*)
        node = self.term()
        while self.current_token is not None and self.current_token[0] ==
'PLUS':
            self.eat('PLUS')
            node = ('PLUS', node, self.term())
        return node

    def parse(self):
        return self.expr()

# Analisis Semantik: Mengecek tipe dan validitas operasi
def semantic_analysis(node):
    if node[0] == 'NUMBER':
```

```
        return 'int'
    elif node[0] in ('PLUS', 'MULTIPLY'):
        left_type = semantic_analysis(node[1])
        right_type = semantic_analysis(node[2])
        if left_type == right_type == 'int':
            return 'int'
        else:
            raise ValueError("Semantic Error: Type mismatch")
    else:
        raise ValueError(f"Unknown node in semantic analysis: {node}")

# Pembangkit Kode Antara dengan Instruksi Mesin
def generate_intermediate_code_with_machine_instructions(node, temp_count=[0]):
    if node[0] == 'NUMBER':
        return str(node[1])

    elif node[0] in ('PLUS', 'MULTIPLY'):
        left_code =
generate_intermediate_code_with_machine_instructions(node[1], temp_count)
        right_code =
generate_intermediate_code_with_machine_instructions(node[2], temp_count)
        temp = f't{temp_count[0]}'
        temp_count[0] += 1

        if node[0] == 'PLUS':
            print(f"{temp} = {left_code} + {right_code}")
            print(f"LOAD {left_code}")
            print(f"ADD {right_code}")
            print(f"STORE {temp}")

        elif node[0] == 'MULTIPLY':
            print(f"{temp} = {left_code} * {right_code}")
            print(f"LOAD {left_code}")
            print(f"MUL {right_code}")
            print(f"STORE {temp}")

        return temp

# Fungsi untuk mengevaluasi AST
def evaluate(node):
    if node[0] == 'NUMBER':
        return node[1]
    elif node[0] == 'PLUS':
        return evaluate(node[1]) + evaluate(node[2])
    elif node[0] == 'MULTIPLY':
```

```
        return evaluate(node[1]) * evaluate(node[2])
    else:
        raise ValueError(f"Unknown node: {node}")

# Contoh penggunaan parser sederhana
def main():
    input_text_1 = "5 + 1 * 4 + 2"
    lexer1 = Lexer(input_text_1)
    parser1 = Parser(lexer1)
    ast1 = parser1.parse()

    # Analisis Semantik
    semantic_type1 = semantic_analysis(ast1)
    print(f"Input 1 - Semantic Type: {semantic_type1}")

    # Pembangkit Kode Antara dengan Instruksi Mesin
    print("Input 1 - Generated Intermediate Code with Machine Instructions:")
    generate_intermediate_code_with_machine_instructions(ast1)

    # Evaluasi
    result1 = evaluate(ast1)
    print("Input 1 - Hasil Evaluasi:", result1)

    print("\n")  # Memisahkan hasil input 1 dan input 2

    input_text_2 = "1 * 4 + 7 * 6"
    lexer2 = Lexer(input_text_2)
    parser2 = Parser(lexer2)
    ast2 = parser2.parse()

    # Analisis Semantik
    semantic_type2 = semantic_analysis(ast2)
    print(f"Input 2 - Semantic Type: {semantic_type2}")

    # Pembangkit Kode Antara dengan Instruksi Mesin
    print("Input 2 - Generated Intermediate Code with Machine Instructions:")
    generate_intermediate_code_with_machine_instructions(ast2)

    # Evaluasi
    result2 = evaluate(ast2)
    print("Input 2 - Hasil Evaluasi:", result2)

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

OUTPUT

```
Input 1 - Semantic Type: int
Input 1 - Generated Intermediate Code with Machine Instructions:
t0 = 1 * 4
LOAD 1
MUL 4
STORE t0
t1 = 5 + t0
LOAD 5
ADD t0
STORE t1
t2 = t1 + 2
LOAD t1
ADD 2
STORE t2
Input 1 - Hasil Evaluasi: 11
```

```
Input 2 - Semantic Type: int
Input 2 - Generated Intermediate Code with Machine Instructions:
t3 = 1 * 4
LOAD 1
MUL 4
STORE t3
t4 = 7 * 6
LOAD 7
MUL 6
STORE t4
t5 = t3 + t4
LOAD t3
ADD t4
STORE t5
Input 2 - Hasil Evaluasi: 46
PS C:\Users\lenovo>
```