```
import re

symbol_table = [
    ["x", "id", "int", "0"],
    ["y", "id", "int", "0"],
    ["i", "id", "int", "0"],
    ["l", "id", "char", "0"]
]

final_array = [
    "int", "main", "(", ")", "{",
    "int", "x", ";",
    "x", ";",
    "x", "=", "2", "+", "5", "+", "(", "4", "*", "8", ")", "+", "l", "/", "9", ";",
    "if", "(", "x", "+", "y", ")", "{",
    "if", "(", "x", "!=", "4", ")", "{",
    "x", "=", "6", ";",
    "y", "=", "10", ";",
    "i", "=", "11", ";",
    "}", "}", "}"
]

variable_reg = re.compile(r"^[A-Za-z_][A-Za-z0-9_]*$")

def print_lexer_output():
    print("Tokenizing...")
    row, col = 1, 1
    for token in final_array:
        if token == "int": print(f"INT ({row},{col})")
        elif token == "main": print(f"MAIN ({row},{col})")
        elif token == "(": print(f"LPAREN ({row},{col})")
        elif token == ")": print(f"RPAREN ({row},{col})")
        elif token == "{": print(f"LBRACE ({row},{col})")
        elif token == "}": print(f"RBRACE ({row},{col})")
        elif token == ";": print(f"SEMI ({row},{col})")
        elif token == "=": print(f"ASSIGN ({row},{col})")
        elif token == "+": print(f"PLUS ({row},{col})")
        elif token == "-": print(f"MINUS ({row},{col})")
        elif token == "*": print(f"TIMES ({row},{col})")
        elif token == "/": print(f"DIV ({row},{col})")
        elif token == "!=": print(f"NEQ ({row},{col})")
        elif re.match(r"^[0-9]+$", token): print(f"INT_CONST ({row},{col}): {token}")
        elif re.match(r"^[0-9]+\.[0-9]+$", token): print(f"FLOAT_CONST ({row},{col}):
{token}")
        elif re.match(r"^[a-zA-Z]$", token): print(f"CHAR_CONST ({row},{col}): {token}")
        elif variable_reg.match(token): print(f"ID ({row},{col}): {token}")
        else: print(f"UNKNOWN ({row},{col}): {token}")
        col += len(token) + 1
        if token == ";":
            row += 1
            col = 1
    print(f"EOF ({row},{col})")

def syntax_directed_translation(index):
    var_name = final_array[index]
```

```
    assign_index = index + 1
    i = assign_index + 1
    expr = []
    while i < len(final_array) and final_array[i] != ";":
        expr.append(final_array[i])
        i += 1
    result = evaluate_expression(expr)
    symbol_index = find_symbol(var_name)
    if symbol_index != -1:
        symbol_table[symbol_index][3] = str(result)
        print(f"Updated {var_name} = {result}")

def evaluate_expression(tokens):
    postfix = infix_to_postfix(tokens)
    return evaluate_postfix(postfix)

def infix_to_postfix(tokens):
    output = []
    stack = []
    precedence = {"+": 1, "-": 1, "*": 2, "/": 2}
    for token in tokens:
        if re.match(r"^[0-9]+$", token) or variable_reg.match(token):
            output.append(token)
        elif token == "(":
            stack.append(token)
        elif token == ")":
            while stack and stack[-1] != "(":
                output.append(stack.pop())
            stack.pop()
        elif token in precedence:
            while stack and stack[-1] in precedence and precedence[token] <= precedence[stack[-
1]]:
                output.append(stack.pop())
            stack.append(token)
    while stack:
        output.append(stack.pop())
    return output

def evaluate_postfix(postfix):
    stack = []
    for token in postfix:
        if re.match(r"^[0-9]+$", token):
            stack.append(int(token))
        elif variable_reg.match(token):
            idx = find_symbol(token)
            if idx != -1:
                stack.append(int(symbol_table[idx][3]))
        else:
            b = stack.pop()
            a = stack.pop()
            if token == "+": stack.append(a + b)
            elif token == "-": stack.append(a - b)
            elif token == "*": stack.append(a * b)
            elif token == "/": stack.append(a // b)
    return stack.pop()
```

```
def find_symbol(name):
    for i, entry in enumerate(symbol_table):
        if entry[0] == name:
            return i
    return -1

# Main logic
print_lexer_output()
for i in range(len(final_array)):
    if final_array[i] == "=":
        syntax_directed_translation(i - 1)

print("\nFinal Symbol Table:")
for entry in symbol_table:
    print(f"Name: {entry[0]}, Type: {entry[2]}, Value: {entry[3]}")

print("\nSemantic Analysis with SDT Completed.")
```

Output: