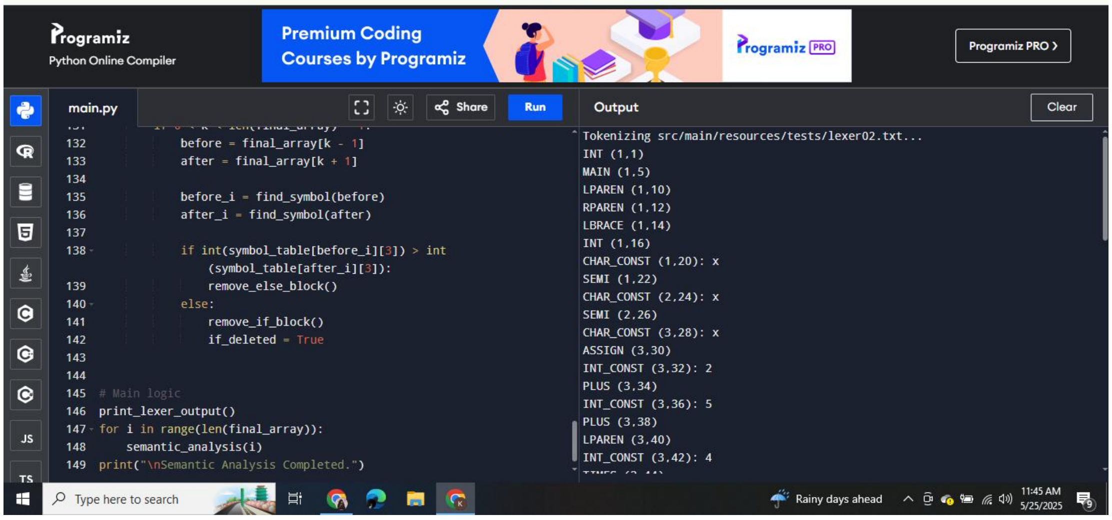
```
import re
symbol_table = [
["x", "id", "int", "0"],
["y", "id", "int", "0"],
["i", "id", "int", "0"],
["l", "id", "char", "0"]
]
final_array = [
"int", "main", "(", ")", "{",
"int", "x", ";",
"x", ";",
"x", "=", "2", "+", "5", "+", "(", "4", "*", "8", ")", "+", "l", "/", "9.0", ";",
"if", "(", "x", "+", "y", ")", "{",
"if", "(", "x", "!=", "4", ")", "{",
"x", "=", "6", ";",
"y", "=", "10", ";",
"i", "=", "11", ";",
"}", "}",
"}"
]
constants = []
variable_reg = re.compile(r"^[A-Za-z_][A-Za-z0-9]*$")
if_deleted = False
def print_lexer_output():
print("Tokenizing src/main/resources/tests/lexer02.txt...")
row, col = 1, 1
for token in final_array:
if token == "int":
print(f"INT ({row},{col})")
elif token == "main":
print(f"MAIN ({row},{col})")
elif token == "(":
print(f"LPAREN ({row},{col})")
elif token == ")":
print(f"RPAREN ({row},{col})")
elif token == "{":
print(f"LBRACE ({row},{col})")
elif token == "}":
print(f"RBRACE ({row},{col})")
elif token == ";":
print(f"SEMI ({row},{col})")
elif token == "=":
print(f"ASSIGN ({row},{col})")
elif token == "+":
print(f"PLUS ({row},{col})")
elif token == "-":
print(f"MINUS ({row},{col})")
elif token == "*":
print(f"TIMES ({row},{col})")
elif token == "/":
print(f"DIV ({row},{col})")
elif token == "!=":
print(f"NEQ ({row},{col})")
elif re.match(r"^[0-9]+$", 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
print(f"EOF ({row},{col})")
def find_symbol(name):
for i, entry in enumerate(symbol_table):
if entry[0] == name:
return i
return -1
def remove_else_block():
if "else" not in final_array:
return
start = final_array.index("else")
end = len(final_array) - 1
for i in range(len(final_array) - 1, start, -1):
if final_array[i] == "}":
end = i
break
del final_array[start:end + 1]
def remove_if_block():
if "if" not in final_array:
return
start = final_array.index("if")
end = final_array.index("}", start)
del final_array[start:end + 1]
def semantic_analysis(k):
global if_deleted
if k >= len(final_array):
return
if final_array[k] in ["+", "-"]:
if 0 < k < len(final_array) - 1:
if variable_reg.match(final_array[k - 1]) and variable_reg.match(final_array[k + 1]):
type_ = final_array[k - 4]
left = final_array[k - 3]
before = final_array[k - 1]
after = final_array[k + 1]
left_i = find_symbol(left)
before_i = find_symbol(before)
after_i = find_symbol(after)
if type_ == symbol_table[before_i][2] == symbol_table[after_i][2]:
ans = int(symbol_table[before_i][3]) + int(symbol_table[after_i][3])
constants.append(ans)
if symbol_table[left_i][2] == symbol_table[before_i][2] == symbol_table[after_i][2]:
ans = int(symbol_table[before_i][3]) + int(symbol_table[after_i][3])
if constants:
constants.pop()
constants.append(ans)
```

```
symbol_table[left_i][3] = str(ans)
if final_array[k] == ">":
if 0 < k < len(final_array) - 1:
before = final_array[k - 1]
after = final_array[k + 1]
before_i = find_symbol(before)
after_i = find_symbol(after)
if int(symbol_table[before_i][3]) > int(symbol_table[after_i][3]):
remove_else_block()
else:
remove_if_block()
if_deleted = True
# Main logic
print_lexer_output()
for i in range(len(final_array)):
semantic_analysis(i)
print("\nSemantic Analysis Completed.")
```

Output:



```
            before = final_array[k - 1]
            after = final_array[k + 1]

            before_i = find_symbol(before)
            after_i = find_symbol(after)

            if int(symbol_table[before_i][3]) > int
                (symbol_table[after_i][3]):
                remove_else_block()
            else:
                remove_if_block()
                if_deleted = True


# Main logic
print_lexer_output()
for i in range(len(final_array)):
    semantic_analysis(i)
print("\nSemantic Analysis Completed.")
```

```
Tokenizing src/main/resources/tests/lexer02.txt...
INT (1,1)
MAIN (1,5)
LPAREN (1,10)
RPAREN (1,12)
LBRACE (1,14)
INT (1,16)
CHAR_CONST (1,20): x
SEMI (1,22)
CHAR_CONST (2,24): x
SEMI (2,26)
CHAR_CONST (3,28): x
ASSIGN (3,30)
INT_CONST (3,32): 2
PLUS (3,34)
INT_CONST (3,36): 5
PLUS (3,38)
LPAREN (3,40)
INT_CONST (3,42): 4
```