**Symbol , literal , intermediate , pool table**# Defining some constants to be used throughout the code

ad = ['START', 'END'] # 'START' and 'END' are used as labels for the start and end points of the program

IS = ['ADD', 'MOVER', 'SUB', 'MOVEM', 'MULT', 'PRINT', 'DC', 'DS'] # List of imperative statements

DL = ['DS', 'DC'] # 'DS' (Define Storage) and 'DC' (Define Constant) are data statements

SYMB = ['NEXT', 'LOOP', 'LOAD'] # List of symbols used in the code

reg = ['AREG', 'BREG', 'CREG', 'DREG'] # Register names (AREG, BREG, etc.)

# Dictionaries to store symbol, literal, and pole data

sym\_dict = {} # Dictionary to store symbols and their corresponding addresses

lit\_dict = {} # Dictionary to store literals and their corresponding addresses

pole\_dict = {} # New Pole Table (to store unresolved references for symbols and literals)

# Function to check if any of the tokens are symbols

def is\_symbol(x):

for i in x:

if (i in SYMB): # If the token is in the SYMB list (symbol list)

sym\_dict[i] = (-1) # Add it to the sym\_dict with an initial address of -1

pole\_dict[i] = "UNRESOLVED" # Add it to the pole\_dict with status 'UNRESOLVED'

# Function to check if any of the tokens are literals

def is\_literal(x):

for i in x:

if (i[0] == "="): # If the token starts with '=', it's a literal

lit\_dict[i] = (-1) # Add it to the lit\_dict with an initial address of -1

pole\_dict[i] = "UNRESOLVED" # Add the literal to the pole\_dict with status 'UNRESOLVED'

# Function to initialize and increment the location counter (LC)

def inc\_lc(x):

global lc # Declaring lc (location counter) as a global variable

if (x[0] == ad[0]): # Check if the first token is 'START' (ad[0])

lc = int(x[1]) # Initialize LC to the value provided after 'START'

print("location\_counter", lc) # Print the initial LC value

i = x[0] # Store the first token in variable 'i'

if (i in IS): # If the first token is an imperative statement (from IS list)

lc = lc + 1 # Increment LC by 1

for i in x: # Loop through all the tokens in the line

if (i in sym\_dict): # If the token is a symbol (found in sym\_dict)

sym\_dict[i] = lc # Assign the current LC value as the address of the symbol

pole\_dict[i] = lc # Update the pole\_dict with the resolved address

lc = lc + 1 # Increment LC by 1

if ((i == "DC") or (i == "DS")): # If the token is a Data Statement (DC or DS)

if (i == "DS"): # If it's DS (Define Storage)

id = x.index(i) # Find the index of 'DS' in the list

lc = lc + (int(x[id + 1])) - 1 # Increment LC by the value following DS (size)

sym\_dict[x[id - 1]] = lc # Store the symbol's address

pole\_dict[x[id - 1]] = lc # Update the pole\_dict with the resolved address

lc = lc + 1 # Increment LC by 1

else: # If it's DC (Define Constant)

id = x.index(i) # Find the index of 'DC' in the list

sym\_dict[x[id - 1]] = lc # Store the symbol's address

pole\_dict[x[id - 1]] = lc # Update the pole\_dict with the resolved address

lc = lc + 1 # Increment LC by 1

if (i == ad[1]): # If the first token is 'END' (ad[1])

for i in lit\_dict: # Loop through all literals

lit\_dict[i] = lc # Assign the current LC value as the address of the literal

pole\_dict[i] = lc # Update the pole\_dict with the resolved address

lc = lc + 1 # Increment LC by 1

# Function to generate the Literal Table

def gen\_lit\_table(lit\_dict):

with open('lit.txt', 'w') as f2: # Open the file 'lit.txt' for writing

f2.write("\tLiteral Table\n") # Write table header

f2.write("Name Address\n") # Write column headers for name and address

for i in lit\_dict: # Loop through the literal dictionary

f2.write(i + " ") # Write the literal name

f2.write(str(lit\_dict[i]) + "\n") # Write the address of the literal

# Function to generate the Symbol Table

def sym\_gen():

with open('sym.txt', 'w') as f1: # Open the file 'sym.txt' for writing

f1.write("\tSymbol Table \n") # Write table header

f1.write("Name Address\n") # Write column headers for name and address

for i in sym\_dict: # Loop through the symbol dictionary

f1.write(i + " ") # Write the symbol name

f1.write(str(sym\_dict[i]) + "\n") # Write the address of the symbol

# Function to generate the Pole Table

def gen\_pole\_table(pole\_dict):

with open('pole.txt', 'w') as f4: # Open the file 'pole.txt' for writing

f4.write("\tPole Table\n") # Write table header

f4.write("Name Status\n") # Write column headers for name and status

for i in pole\_dict: # Loop through the pole dictionary

f4.write(i + " ") # Write the pole name

f4.write(str(pole\_dict[i]) + "\n") # Write the status or address of the pole (unresolved or resolved)

# Function to generate Intermediate Code

def gen\_interm\_code(x):

for i in x: # Loop through all tokens in the line

if (i in ad): # If the token is 'START' or 'END' (from ad list)

if (x[0] == ad[0]): # If it's 'START'

f3.write("AD 1 " + " C " + x[1]) # Write the corresponding intermediate code

else: # If it's 'END'

f3.write("AD 02 ")

if (i in IS): # If the token is an imperative statement (from IS list)

if (i == 'DS') or (i == 'DC'): # If it's DC or DS

f3.write("DL " + str(DL.index(i)) + " C " + x[x.index(i) + 1]) # Write corresponding intermediate code

else:

f3.write("IS " + str(IS.index(i) + 1) + " ") # Write intermediate code for imperative statement

if (i in lit\_dict): # If the token is a literal (from lit\_dict)

f3.write(" L " + str(list(lit\_dict).index(i) + 1) + " ") # Write the index of the literal in the intermediate code

if ((i in sym\_dict)): # If the token is a symbol (from sym\_dict)

f3.write(i + " ") # Write the symbol name

if ((i in reg)): # If the token is a register (from reg list)

f3.write(" " + i + "") # Write the register name

# Execution starts here

file1 = open("INPUT.txt", 'r') # Open the input file 'INPUT.txt' for reading

lines = file1.readlines() # Read all lines from the input file

f3 = open("inter.txt", 'a') # Open the output file 'inter.txt' for appending

for line in lines:

x = line.split() # Split each line into tokens

if (len(x) > 0): # If there are any tokens in the line

is\_symbol(x) # Check if any tokens are symbols

is\_literal(x) # Check if any tokens are literals

inc\_lc(x) # Increment the location counter based on the tokens

gen\_interm\_code(x) # Generate intermediate code

sym\_gen() # Generate the symbol table

gen\_lit\_table(lit\_dict) # Generate the literal table

gen\_pole\_table(pole\_dict) # Generate the pole table