```
#Macro  without argument
.MODEL SMALL
.STACK 100H
.DATA
 MSG DB "MACRO EXPANSION SUCCESSFUL!$"
.CODE
; Define a MACRO without arguments
DISPLAY_MSG MACRO
 MOV DX, OFFSET MSG
 MOV AH, 09H
 INT 21H
ENDM
MAIN PROC
 MOV AX, @DATA
 MOV DS, AX
 DISPLAY_MSG ; Macro Call 1
 DISPLAY_MSG ; Macro Call 2
 MOV AH, 4CH
 INT 21H
MAIN ENDP
END MAIN
#Lex
Code:
import re
def analyze_text(text):
 char_count = len(text)
 word_count = len(re.findall(r'\b\w+\b', text))
```

```
 sentence_count = len(re.findall(r'[.!?]', text))
 line_count = text.count('\n') + 1
 tab_count = text.count('\t')
 number_count = len(re.findall(r'\b\d+\b', text))
 space_count = text.count(' ')
 print(f"Character Count: {char_count}")
 print(f"Word Count: {word_count}")
 print(f"Sentence Count: {sentence_count}")
 print(f"Line Count: {line_count}")
 print(f"Tab Count: {tab_count}")
 print(f"Number Count: {number_count}")
 print(f"Space Count: {space_count}")
# Input from user
text = input("Enter your text:\n")
analyze_text(text)


#two pass
Code:
class AssemblerPass1:
 def __init__(self):
 self.symbol_table = {}
 self.literal_table = {}
 self.pool_table = []
 self.intermediate_code = []
 self.location_counter = 0
 self.literal_index = 1
 def process_line(self, label, opcode, operand):
```

```
 if opcode == "START":
 self.location_counter = int(operand)
 self.intermediate_code.append(f"(AD,01)
(C,{operand})")
 elif opcode == "END":
 self.intermediate_code.append("(AD,02)")
 elif opcode == "LTORG":
 for lit, addr in self.literal_table.items():
 if addr is None:
 self.literal_table[lit] =
self.location_counter
 self.intermediate_code.append(f"(DL,02)
(C,{lit})")
 self.location_counter += 1
 elif opcode == "DS":
 self.symbol_table[label] =
self.location_counter
 self.intermediate_code.append(f"(DL,01)
(C,{operand})")
 self.location_counter += int(operand)
 else:
 if label:
 self.symbol_table[label] =
self.location_counter
if "=" in operand:
 lit = operand.strip("=")
 self.literal_table[lit] = None
```

```
 operand = f"(L,{self.literal_index})"
 self.literal_index += 1
 self.intermediate_code.append(f"(IS,XX)
{operand}")
 self.location_counter += 1
 def display_results(self):
 print("Symbol Table:")
 for symbol, address in
self.symbol_table.items():
 print(f"{symbol}: {address}")
 print("\nLiteral Table:")
 for literal, address in
self.literal_table.items():
 print(f"{literal}: {address}")
 print("\nIntermediate Code:")
 for line in self.intermediate_code:
 print(line)
# Input Assembly Code
assembly_code = [
 ("JOHN", "START", "200"),
 ("", "MOVER", "R1, ='3'"),
 ("", "MOVEM", "R1, X"),
 ("L1", "MOVER", "R2, ='2'"),
 ("", "LTORG", ""),
 ("X", "DS", "1"),
 ("", "END", "")
]
```

```
# Process Assembly Code
assembler = AssemblerPass1()
for label, opcode, operand in assembly_code:
 assembler.process_line(label, opcode, operand)
# Display Results
assembler.display_results()


#First and follow
from collections import defaultdict
class Grammar:
 def __init__(self, productions):
 self.productions = productions
 self.first = defaultdict(set)
 self.follow = defaultdict(set)
 self.non_terminals = set(productions.keys())
 self.compute_first()
 self.compute_follow()
 def compute_first(self):
 """ Compute FIRST sets for all non-terminals
"""
 for nt in self.non_terminals:
 self.first[nt] = self.get_first(nt)
 def get_first(self, symbol):
 """ Compute FIRST set of a symbol """
 if symbol not in self.non_terminals:
 return {symbol} # Terminal symbol FIRST
is itself
```

```
 first_set = set()
 for production in self.productions[symbol]:
 for char in production:
 first_sub = self.get_first(char)
 first_set.update(first_sub - {'ε'}) # Add
FIRST(char) except epsilon
 if 'ε' not in first_sub:
 break
 else:
 first_set.add('ε') # If all characters
produce epsilon, add ε
 return first_set
 def compute_follow(self):
 """ Compute FOLLOW sets for all non-
terminals """
 start_symbol = list(self.productions.keys())[0]
 self.follow[start_symbol].add('$') # Start
symbol gets '$'
for _ in range(len(self.non_terminals)):
 for nt, rules in self.productions.items():
 for rule in rules:
 for i, symbol in enumerate(rule):
 if symbol in self.non_terminals:

first_next = self.get_first(rule[i + 1]) if i + 1 <
len(rule) else {'ε'}
```

```
self.follow[symbol].update(first_next - {'ε'})
 if 'ε' in first_next or i + 1 ==
len(rule):


self.follow[symbol].update(self.follow[nt])
 def display(self):
 """ Display FIRST and FOLLOW sets """
 print("\nFIRST sets:")
 for nt, first_set in self.first.items():
 print(f"FIRST({nt}) = {first_set}")
 print("\nFOLLOW sets:")
 for nt, follow_set in self.follow.items():
 print(f"FOLLOW({nt}) = {follow_set}")
# Example Grammar
productions = {
 'E': ['TR'],
 'R': ['+TR', 'ε'],
 'T': ['FY'],
 'Y': ['*FY', 'ε'],
 'F': ['(E)', 'id']
}
grammar = Grammar(productions)
grammar.display()
```