**Pass 1 of Two pass assembler: (pass1.py)***def* pass1(*source\_code*):

    lines = source\_code.strip().split('\n')

    symbol\_table = {}

    location\_counter = 0

    intermediate\_code = []

    optab = ['LDA', 'STA', 'ADD', 'SUB', 'JMP', 'MUL', 'DIV', 'COMP']

    for i, line in enumerate(lines):

        parts = line.strip().split()

        if len(parts) == 3:

            label, opcode, operand = parts

        elif len(parts) == 2:

            label = ''

            opcode, operand = parts

        elif len(parts) == 1:

            label = ''

            opcode = parts[0]

            operand = ''

        else:

            continue

*# START directive*

        if opcode == 'START':

            location\_counter = *int*(operand)

            intermediate\_code.append((location\_counter, line))

            continue

*# Add label to symbol table*

        if label != '':

            symbol\_table[label] = location\_counter

        intermediate\_code.append((location\_counter, line))

*# Increase location counter*

        if opcode in optab:

            location\_counter += 3

        elif opcode == 'WORD':

            location\_counter += 3

        elif opcode == 'RESW':

            location\_counter += 3 \* *int*(operand)

        elif opcode == 'RESB':

            location\_counter += *int*(operand)

        elif opcode == 'BYTE':

            location\_counter += len(operand) - 3 *# example: C'EOF' → 3 bytes*

        elif opcode == 'END':

            break

    return symbol\_table, intermediate\_code

*# Example input*

code = """

START 1000

LOOP LDA NUM

    ADD ONE

    STA NUM

NUM WORD 5

ONE WORD 1

    END

"""

*# Run Pass 1*

symtab, intercode = pass1(code)

*#Output*

print("Symbol Table:")

for label, addr in symtab.items():

    print(*f*"{label} -> {addr}")

print("\nIntermediate Code:")

for addr, line in intercode:

    print(*f*"{addr} : {line}")

HOW TO RUN:  
1. open terminal

2. write:   
python –version (press enter)

Suppose if the version is python3..something then

3. write:

python3 pass1.py (press enter)

**MACRO without arguments: (macro.py)**

input\_code = """MOV R

JOHN

DCR R

AND R

JOHN

MUL 88

HALT"""

macro\_def = """MACRO

JOHN

ADD 30

SUB 25

OR R

MEND"""

macro\_name, macro\_body = macro\_def.split("\n")[1], macro\_def.split("\n")[2:-1]

output\_code = [macro\_body if line.strip() == macro\_name else line for line in input\_code.split("\n")]

expanded\_code = sum(([line] if isinstance(line, *str*) else line for line in output\_code), [])

mc, mi, ti = input\_code.count(macro\_name), len(macro\_body), len(expanded\_code)

ni = len(input\_code.split("\n")) - mc  *# Number of instructions excluding macro calls*

print("Expanded Source Code:\n", "\n".join(expanded\_code), *f*"\n\nStats:\nMacro Calls = {mc}\nInstructions in Macro = {mi}\nTotal Instructions = {ti}\nInstructions (excluding Macro calls) = {ni}")

HOW TO RUN:  
1. open terminal

2. write:   
python –version (press enter)

Suppose if the version is python3..something then

3. write:

python3 macro.py (press enter)

**First of CFG: (first.py)**

from collections import defaultdict

*# Static input for grammar productions*

productions = [

    "E -> TG",

    "T -> FG | #",

    "F -> (E) | i",

    "G -> +TG | #"

]

prods, first = defaultdict(*list*), {}

*# Parse static input and store productions*

for line in productions:

    line = line.replace(" ", "")

    if '->' in line:

        lhs, rhs = line.split("->")

        prods[lhs].extend(rhs.split('|'))

*# Function to compute FIRST set*

*def* find\_first(*symbol*):

    if symbol in first:

        return first[symbol]

    first[symbol] = result = *set*()

    print(*f*"Processing {symbol}, current FIRST = {first.get(symbol, *set*())}")  *# Debugging statement*

    for rule in prods.get(symbol, []):

        if rule == '#':

            result.add('#')  *# Epsilon*

        else:

            for char in rule:

                if not char.isupper():

                    result.add(char)  *# Terminal*

                    break

                sub = find\_first(char)

                result |= sub - {'#'}

                if '#' not in sub:

                    break

            else:

                result.add('#')

    first[symbol] = result

    print(*f*"Computed FIRST({symbol}) = {result}")  *# Debugging statement*

    return result

*# Compute FIRST sets for each non-terminal*

for nt in prods:

    find\_first(nt)

*# Print the computed FIRST sets*

print("\nFinal FIRST Sets:")

for key, value in sorted(first.items()):

    print(*f*"FIRST({key}) = {{ {' '.join(sorted(value))} }}")

HOW TO RUN:  
1. open terminal

2. write:   
python –version (press enter)

Suppose if the version is python3..something then

3. write:

python3 first.py (press enter)

**Follow of CFG: (follow.py)**from collections import defaultdict

*# Static input for CFG rules*

productions = [

    "E -> TG",

    "T -> FG | #",

    "F -> (E) | i",

    "G -> +TG | #"

]

*# Parse productions into a dictionary*

rules = defaultdict(*list*)

for line in productions:

    lhs, rhs = line.replace(" ", "").split("->")

    rules[lhs].extend(rhs.split('|'))

*# Initialize FIRST and FOLLOW sets*

first = defaultdict(*set*)

follow = defaultdict(*set*)

start\_symbol = *list*(rules.keys())[0]  *# Start symbol (first non-terminal)*

*# Helper function to compute FIRST set*

*def* compute\_first(*symbol*):

    if symbol in first:

        return first[symbol]

    if not symbol.isupper():  *# Terminal*

        return {symbol}

    for rule in rules[symbol]:

        for char in rule:

            char\_first = compute\_first(char)

            first[symbol].update(char\_first - {'#'})

            if '#' not in char\_first:

                break

        else:

            first[symbol].add('#')

    return first[symbol]

*# Compute FIRST sets for all non-terminals*

for non\_terminal in rules:

    compute\_first(non\_terminal)

*# Helper function to compute FOLLOW set*

*def* compute\_follow(*non\_terminal*):

    if not follow[non\_terminal]:  *# Check if FOLLOW already computed*

        for lhs, rhs\_list in rules.items():

            for rhs in rhs\_list:

                if non\_terminal in rhs:

                    idx = rhs.index(non\_terminal)

                    remainder = rhs[idx + 1:]

                    for char in remainder:

                        follow[non\_terminal].update(compute\_first(char) - {'#'})

                        if '#' not in compute\_first(char):

                            break

                    else:

                        if non\_terminal != lhs:  *# Prevent self-dependency*

                            follow[non\_terminal].update(compute\_follow(lhs))

        if non\_terminal == start\_symbol:

            follow[non\_terminal].add('$')  *# Add end-of-input symbol*

    return follow[non\_terminal]

*# Compute FOLLOW sets for all non-terminals*

for non\_terminal in rules:

    compute\_follow(non\_terminal)

*# Print FIRST sets*

print("FIRST Sets:")

for key, value in sorted(first.items()):

    print(*f*"FIRST({key}) = {{ {' '.join(sorted(value))} }}")

*# Print FOLLOW sets*

print("\nFOLLOW Sets:")

for key, value in sorted(follow.items()):

    print(*f*"FOLLOW({key}) = {{ {' '.join(sorted(value))} }}")

HOW TO RUN:  
1. open terminal

2. write:   
python –version (press enter)

Suppose if the version is python3..something then

3. write:

python3 follow.py (press enter)

**Lex program: (count.l)**

%{

#include <stdio.h>

int lines = 0, words = 0, sentences = 0, chars = 0;

int tabs = 0, spaces = 0, numbers = 0;

%}

%%

[\n] { lines++; chars++; }

[\t] { tabs++; chars++; }

[" "] { spaces++; chars++; }

[0-9]+ { numbers++; words++; chars += yyleng; }

[.?!] { sentences++; chars++; }

[a-zA-Z]+ { words++; chars += yyleng; }

. { chars++; }

%%

int main()

{

printf("Enter the input (Ctrl+D to end on Linux/macOS, Ctrl+Z on Windows):\n");

yylex();

printf("\n--- Analysis Report ---\n");

printf("Lines : %d\n", lines);

printf("Words : %d\n", words);

printf("Characters: %d\n", chars);

printf("Spaces : %d\n", spaces);

printf("Tabs : %d\n", tabs);

printf("Sentences : %d\n", sentences);

printf("Numbers : %d\n", numbers);

return 0;

}

int yywrap() {

return 1;

}

HOW TO RUN:

1. open terminal

2. write:  
flex count.l (press enter) \_\_\_ \_\_ => lex count.l (press enter)

cc lex.yy.c (press enter) | | |\_\_| => gcc lex.yy.c (press enter)

./a.out (press enter) |\_\_\_| | \ => ./a.out (press enter)

3. Enter inputs:

Enter the input (Ctrl+D to end on Linux/macOS, Ctrl+Z on Windows):

Hello,world! This is Lex.

1234 5678

How are you?

This is a tab line

4. Final output:

--- Analysis Report ---

Lines : 4

Words : 15

Characters: 71

Spaces : 13

Tabs : 0

Sentences : 3

Numbers : 2

abhixv@abhixv:~/Desktop$