SS Lab Exam

Sahil Bondre: U18CO021

```
import sys
import re
from termcolor import colored
from tabulate import tabulate

if len(sys.argv) != 3:
   print("Usage: python index.py <file-name> <file-name>")
   exit(1)

file_name = sys.argv[1]
macro_file = sys.argv[2]

ssn_tab = []
evn_tab = []
pn_tab = []
mdt = []  # label, opcode, operands


ssn_tab2 = []
evn_tab2 = []
pn_tab2 = []
mdt2 = []  # label, opcode, operands

address = 200


with open(macro_file) as f:
   for num, line in enumerate(f, 1):
       tokens = line.split()
       is_label_def = not bool(re.match(r'\s', line))
       if num != 1 and num != 2 and is_label_def:
           label = tokens[0]
           if label[0] == '&' and label not in evn_tab:
               evn_tab.append(label)
           elif label[0] == '.' and label not in ssn_tab:
               ssn_tab.append(label)

with open(macro_file) as f:
   for num, line in enumerate(f, 1):
       address += 1
       tokens = line.split()
       is_label_def = not bool(re.match(r'\s', line))
       if num == 1:
```

```python
            print(line)
        elif num == 2:
            pn_tab = [x.split(',')[0].split('=')[0] for x in tokens[1:]]
            print(line)
        else:
            # fill mdt:
            row = ['', '', '']
            if is_label_def:

                label = tokens[0]
                tokens = tokens[1:]
                if label in evn_tab:
                    row[0] += f"(E, {evn_tab.index(label) + 1})"
                elif label in ssn_tab:
                    row[0] += f"(S, {ssn_tab.index(label) + 1})"

            row[1] = tokens[0]
            tokens = tokens[1:]

            res = ''
            for token in tokens:
                temp = token.split(',')[0]
                has_comma = (temp != token)
                token = temp
                if token in evn_tab:
                    res += f"(E, {evn_tab.index(token) + 1})"
                elif token in ssn_tab:
                    res += f"(S, {ssn_tab.index(token) + 1})"
                elif token in pn_tab:
                    res += f"(P, {pn_tab.index(token) + 1})"
                else:
                    res += token

                if has_comma:
                    res += ','

            row[2] = res
            row.insert(0, address)
            mdt2.append(row)

with open(file_name) as f:
    for num, line in enumerate(f, 1):
        tokens = line.split()
        is_label_def = not bool(re.match(r'\s', line))
        if num != 1 and num != 2 and is_label_def:
            label = tokens[0]
            if label[0] == '&' and label not in evn_tab:
                evn_tab.append(label)
            elif label[0] == '.' and label not in ssn_tab:
                ssn_tab.append(label)

with open(file_name) as f:
    for num, line in enumerate(f, 1):
        address += 1
        tokens = line.split()
        is_label_def = not bool(re.match(r'\s', line))
        if num == 1:
            print(line)
        elif num == 2:
            pn_tab = [x.split(',')[0].split('=')[0] for x in tokens[1:]]
            print(line)
        else:
            # fill mdt:
            row = ['', '', '']
            if is_label_def:

                label = tokens[0]
                tokens = tokens[1:]
                if label in evn_tab:
                    row[0] += f"(E, {evn_tab.index(label) + 1})"
                elif label in ssn_tab:
                    row[0] += f"(S, {ssn_tab.index(label) + 1})"

            row[1] = tokens[0]
            tokens = tokens[1:]

            res = ''
            for token in tokens:
                temp = token.split(',')[0]
                has_comma = (temp != token)
                token = temp
                if token in evn_tab:
                    res += f"(E, {evn_tab.index(token) + 1})"
                elif token in ssn_tab:
                    res += f"(S, {ssn_tab.index(token) + 1})"
                elif token in pn_tab:
                    res += f"(P, {pn_tab.index(token) + 1})"
                else:
                    res += token

                if has_comma:
                    res += ','

            row[2] = res
            row.insert(0, address)
            mdt.append(row)
```

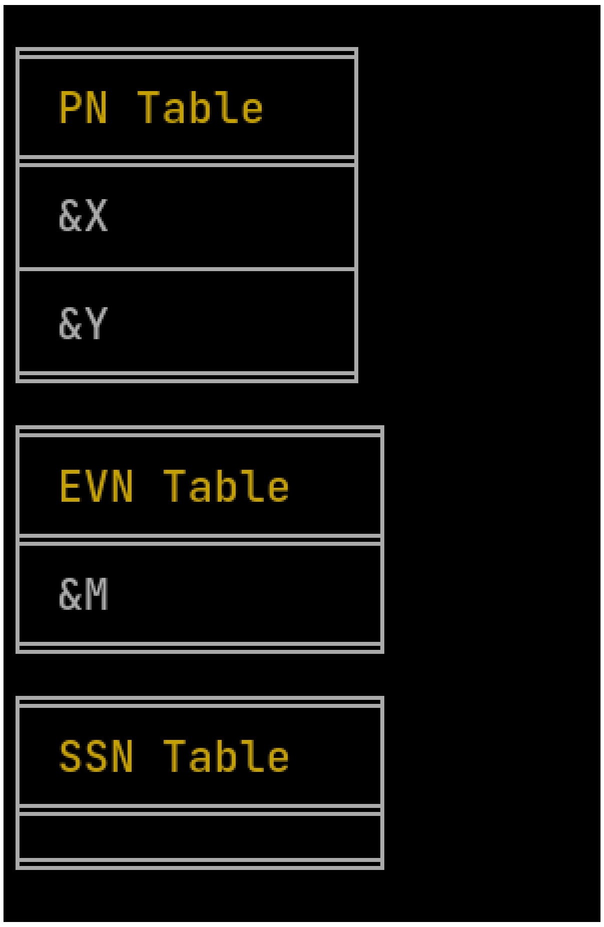
```
print(tabulate([[x] for x in pn_tab], headers=[
     colored("PN Table", color="yellow")], tablefmt="fancy_grid"))

print(tabulate([[x] for x in evn_tab], headers=[
   colored("EVN Table", color="yellow")], tablefmt="fancy_grid"))

print(tabulate([[x] for x in ssn_tab], headers=[
   colored("SSN Table", color="yellow")], tablefmt="fancy_grid"))

print()
print(colored("Macro Definition Table Initial", attrs=["bold"],
color="blue"))
print(tabulate(mdt, headers=[colored("Addr", color="yellow"),
     colored("Label", color="yellow"), colored("Opcode", color="yellow"),
colored("Operands", color="yellow")], tablefmt="fancy_grid"))

print(colored("Macro Definition Table Nested", attrs=["bold"],
color="blue"))
print(tabulate(mdt2, headers=[colored("Addr", color="yellow"),
     colored("Label", color="yellow"), colored("Opcode", color="yellow"),
colored("Operands", color="yellow")], tablefmt="fancy_grid"))


address = mdt[0][0]
for i in range(len(mdt)):
   x = i
   if(mdt[i][2] == "ADDNUM"):
       mdt.pop(i)
       for r in mdt2:
           mdt.insert(x, r)
           x += 1

address = mdt[0][0]
for i in range(len(mdt)):
   mdt[i][0] = address
   address += 1


print(colored("Macro Definition Table Combined", attrs=["bold"],
color="blue"))
print(tabulate(mdt, headers=[colored("Addr", color="yellow"),
     colored("Label", color="yellow"), colored("Opcode", color="yellow"),
colored("Operands", color="yellow")], tablefmt="fancy_grid"))
```

file.asm

```
      MACRO
      ADDTEN &X, &Y
      MOVER AREG, X
      MOVER BREG, Y
      LCL &M
&M    SET 0
      ADDNUM AREG, BREG
      INCR AREG
      INCR BREG
&M    SET &M + 1
      AIF ( &M NE 10 ) .MORE
      MEND
```

nested.asm

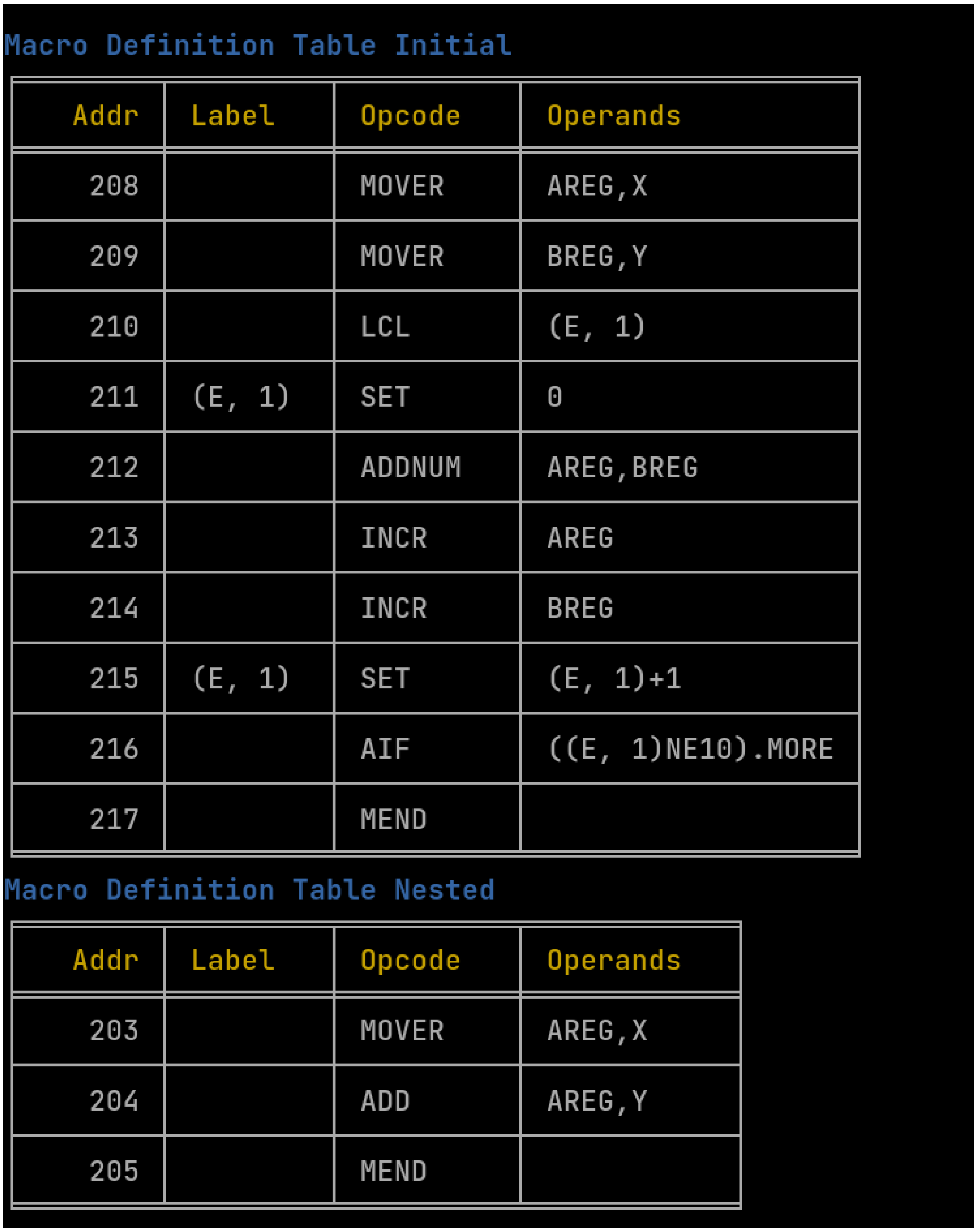
```
      MACRO
      ADDNUM &X, &Y
      MOVER AREG, X
      ADD AREG, Y
      MEND
```



PN Table
&X
&Y

EVN Table
&M

SSN Table

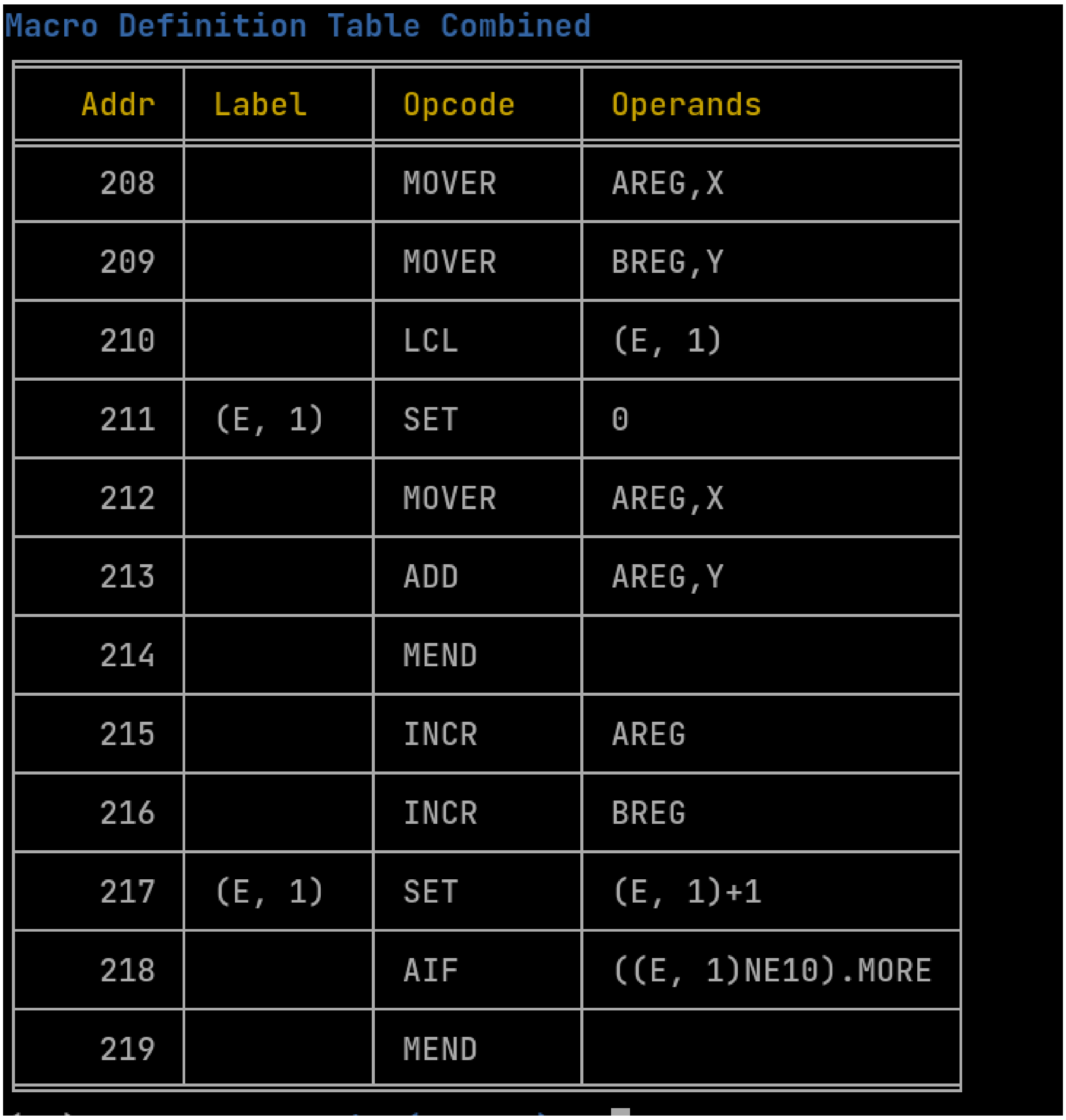


Macro Definition Table Initial

Addr	Label	Opcode	Operands
208		MOVER	AREG,X
209		MOVER	BREG,Y
210		LCL	(E, 1)
211	(E, 1)	SET	0
212		ADDNUM	AREG,BREG
213		INCR	AREG
214		INCR	BREG
215	(E, 1)	SET	(E, 1)+1
216		AIF	((E, 1)NE10).MORE
217		MEND	

Macro Definition Table Nested

Addr	Label	Opcode	Operands
203		MOVER	AREG,X
204		ADD	AREG,Y
205		MEND	



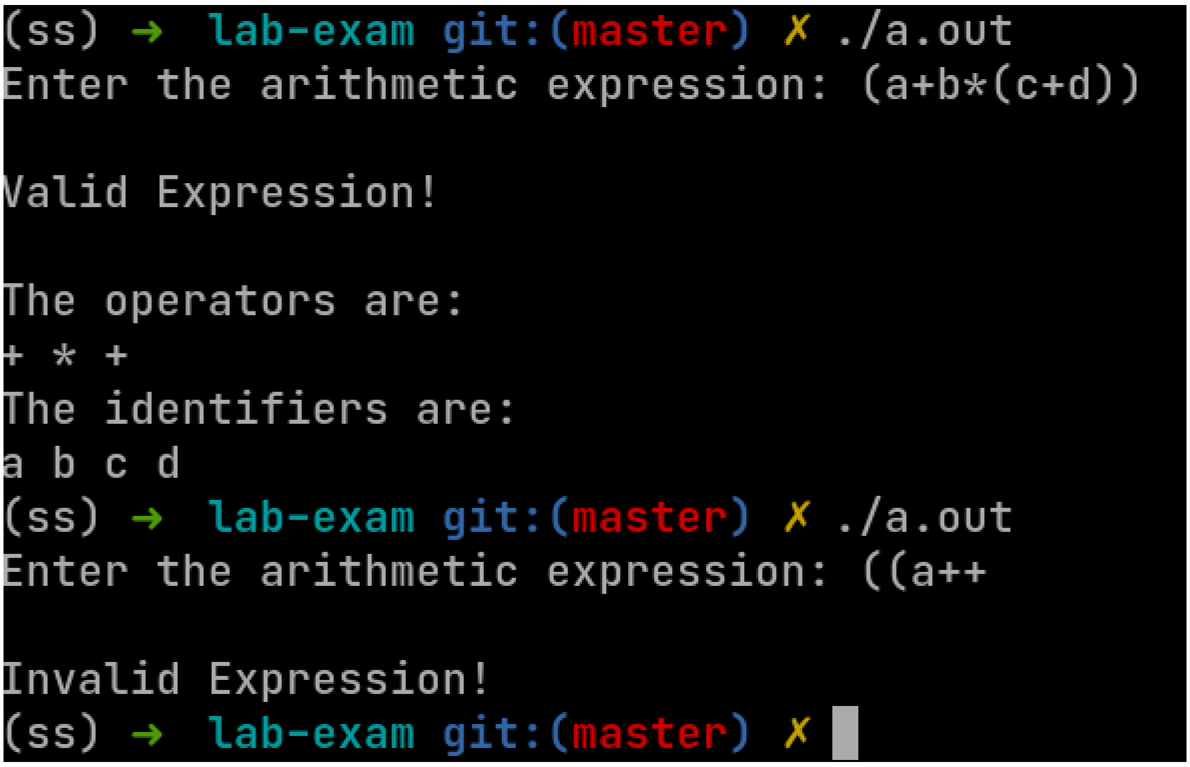
```
Macro Definition Table Combined
```

Addr	Label	Opcode	Operands
208		MOVER	AREG,X
209		MOVER	BREG,Y
210		LCL	(E, 1)
211	(E, 1)	SET	0
212		MOVER	AREG,X
213		ADD	AREG,Y
214		MEND	
215		INCR	AREG
216		INCR	BREG
217	(E, 1)	SET	(E, 1)+1
218		AIF	((E, 1)NE10).MORE
219		MEND	

Q2:

```
%{
#include <stdio.h>
#include <string.h>
int op = 0, oprs = 0, valid = 1, top = -1, l = 0, j = 0;
char operands[10][10], operators[10][10], stack[100];
%}
%%

"+"|"-"|"*"|"/" {
 op++;
 strcpy(operators[l], yytext);
 l++;
}

[0-9]+|[a-zA-Z][a-zA-Z0-9_]* {
 oprs++;
 strcpy(operands[j], yytext);
 j++;
}

"(" {
 top++;
 stack[top] = '(';
}

")" {
 if (stack[top] != '(') {
   valid = 0;
 } else if (oprs > 0 && (oprs - op) != 1) {
   valid = 0;
 } else {
   top--;
   oprs = 1;
   op = 0;
 }
}

"{" {
 top++;
 stack[top] = '{';
}

"}" {
 if (stack[top] != '{') {
   valid = 0;
 } else if (oprs > 0 && (oprs - op) != 1){
   valid=0;
 } else {
   top--;
   oprs = 1;
   op=0;
 }
}

"[" {
 top++;
 stack[top] = '[';
}
```

```
"]" {
 if (stack[top] != '[') {
   valid = 0;
 } else if (oprs > 0 && (oprs - op) != 1){
   valid=0;
 } else {
   top--;
   oprs=1;
   op=0;
 }

}

"\n" return 0;
%%


int yywrap(){ return 1; }
int main(){
 int k;
 printf("Enter the arithmetic expression: ");
 yylex();

 if (valid == 1 && top == -1) {
   printf("\nValid Expression!\n");
       printf("\nThe operators are: \n");
       for(int i = 0; i < l; ++i)
           printf("%s ", operators[i]);
       printf("\nThe identifiers are: \n");
       for(int i = 0; i < j; ++i)
           printf("%s ", operands[i]);
       printf("\n");
 }
 else
   printf("\nInvalid Expression!\n");

 return 0;
}
```



```
(ss) ➜  lab-exam git:(master) ✗ ./a.out
Enter the arithmetic expression: (a+b*(c+d))

Valid Expression!

The operators are:
+ * +
The identifiers are:
a b c d
(ss) ➜  lab-exam git:(master) ✗ ./a.out
Enter the arithmetic expression: ((a++

Invalid Expression!
(ss) ➜  lab-exam git:(master) ✗
```