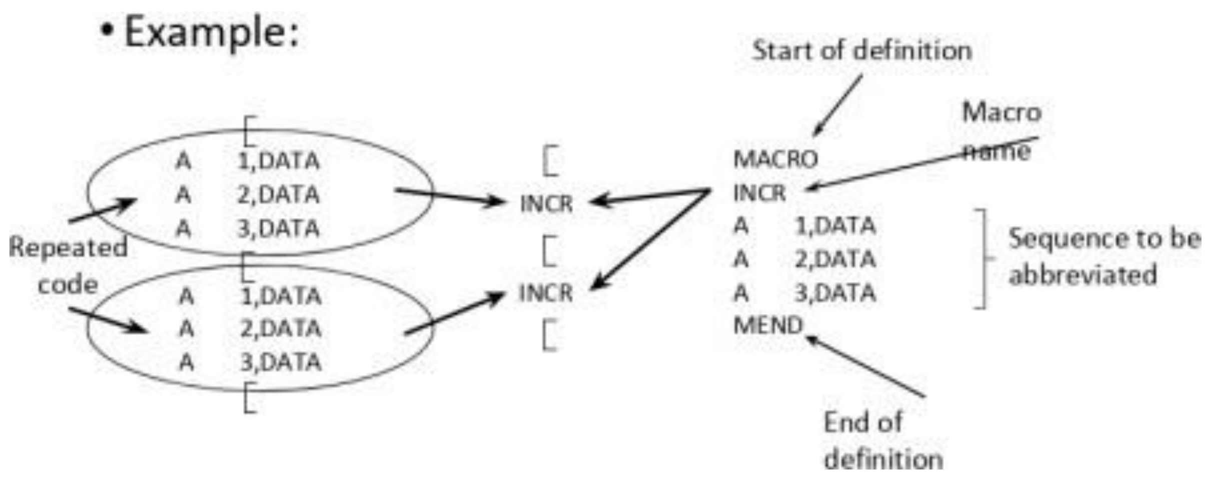
EXPERIMENT 3

CLASS: TE CMPN A **PID:182027**
NAME: REBECCA DIAS **ROLL NO. : 19**

AIM: To demonstrate a 2 pass macro processor.

THEORY:

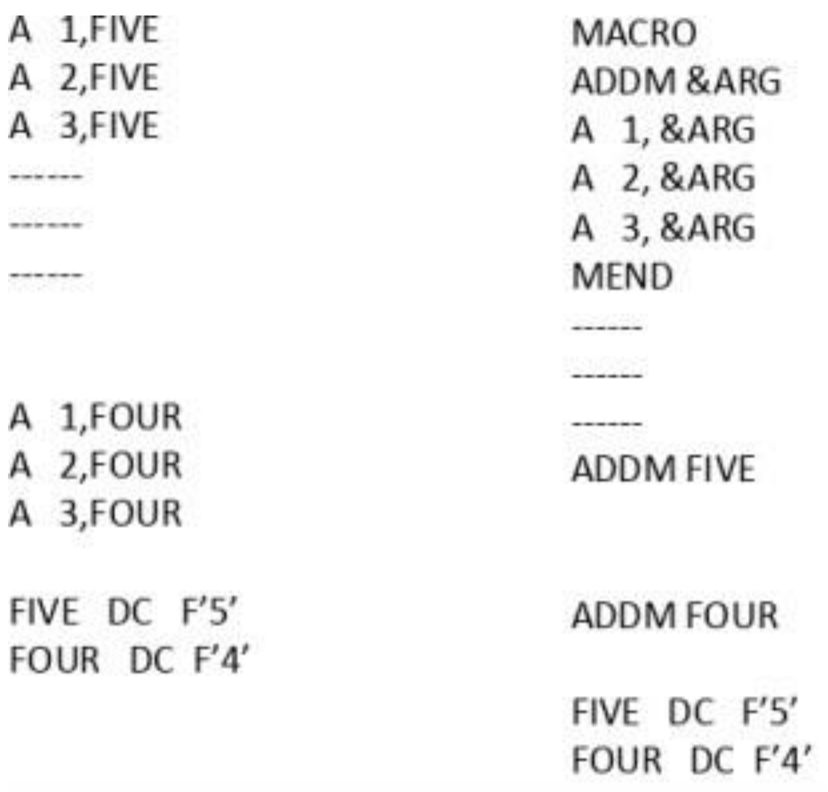
Macro instructions or Macros are single line abbreviations for groups of instructions. Single instruction is used to represent a block of code. For every occurrence of this one line macro instruction, the macro processing assembler will substitute the entire block.



Features of Macro Facility:

<u>Macro Instruction Arguments</u>

Macro calls replaces the call by a block of code. No flexibility to modify code that replaces the call. Extension for providing arguments or parameters in macro call. Macro instruction argument (dummy arguments) are used in definition. It is specified in the macro name line and distinguished by '&' Arguments that are not specified, are presumed blank by macro processor.



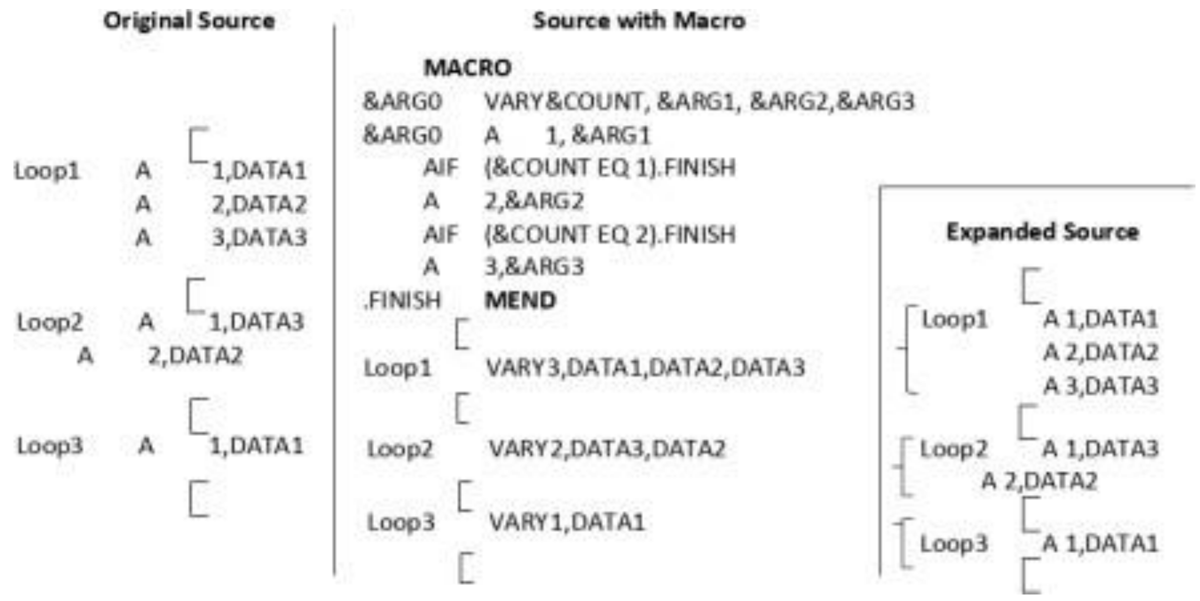
```
A  1,FIVE
A  2,FIVE
A  3,FIVE
------
------
------

A  1,FOUR
A  2,FOUR
A  3,FOUR

FIVE  DC  F'5'
FOUR  DC  F'4'
```

```
MACRO
ADDM &ARG
A  1, &ARG
A  2, &ARG
A  3, &ARG
MEND
------
------
------
ADDM FIVE

ADDM FOUR

FIVE  DC  F'5'
FOUR  DC  F'4'
```

<u>Conditional Macro Expansion</u>

AIF and AGO permit conditional reordering of the sequence of macro expansion. Machine instructions that appear in the expansion of a macro call can be selected based on condition.

AIF is used for conditional branching whereas AGO is used for unconditional branching.



Original Source

```
Loop1     A     1,DATA1
          A     2,DATA2
          A     3,DATA3

Loop2     A     1,DATA3
       A     2,DATA2

Loop3     A     1,DATA1
```

Source with Macro

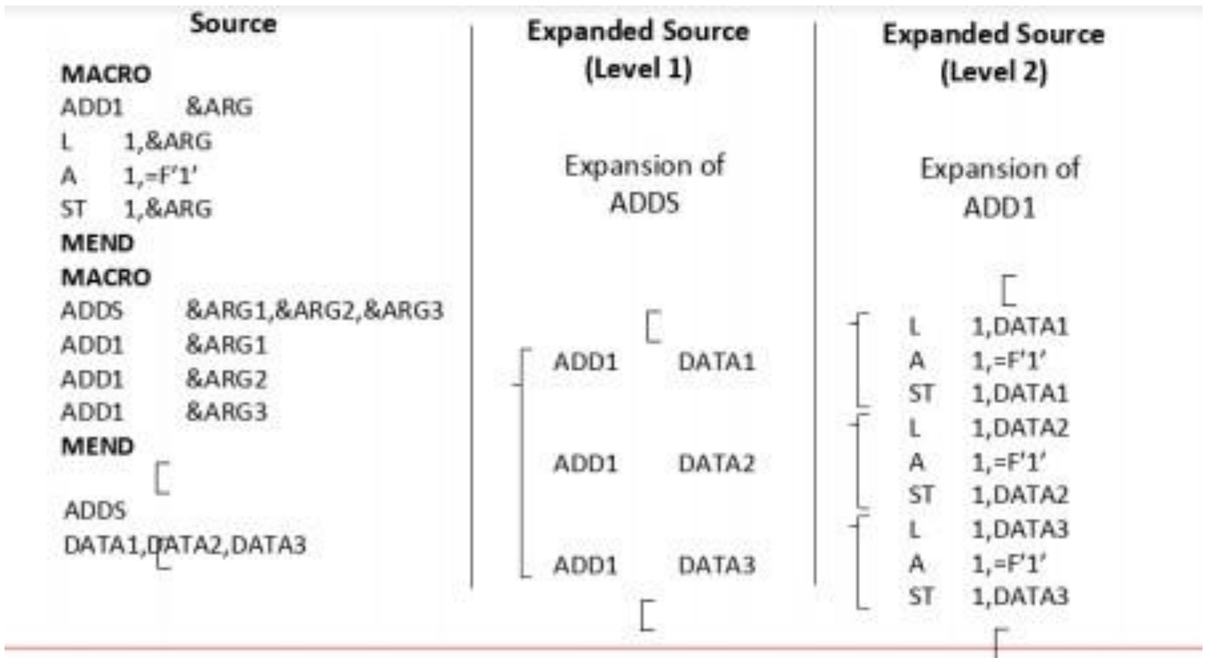
```
        MACRO
&ARG0      VARY&COUNT, &ARG1, &ARG2,&ARG3
&ARG0      A     1, &ARG1
      AIF  (&COUNT EQ 1).FINISH
      A    2,&ARG2
      AIF  (&COUNT EQ 2).FINISH
      A    3,&ARG3
.FINISH    MEND

Loop1      VARY3,DATA1,DATA2,DATA3

Loop2      VARY2,DATA3,DATA2

Loop3      VARY1,DATA1
```

Expanded Source

```
Loop1     A 1,DATA1
          A 2,DATA2
          A 3,DATA3

Loop2     A 1,DATA3
       A 2,DATA2

Loop3     A 1,DATA1
```

Macro calls within Macros

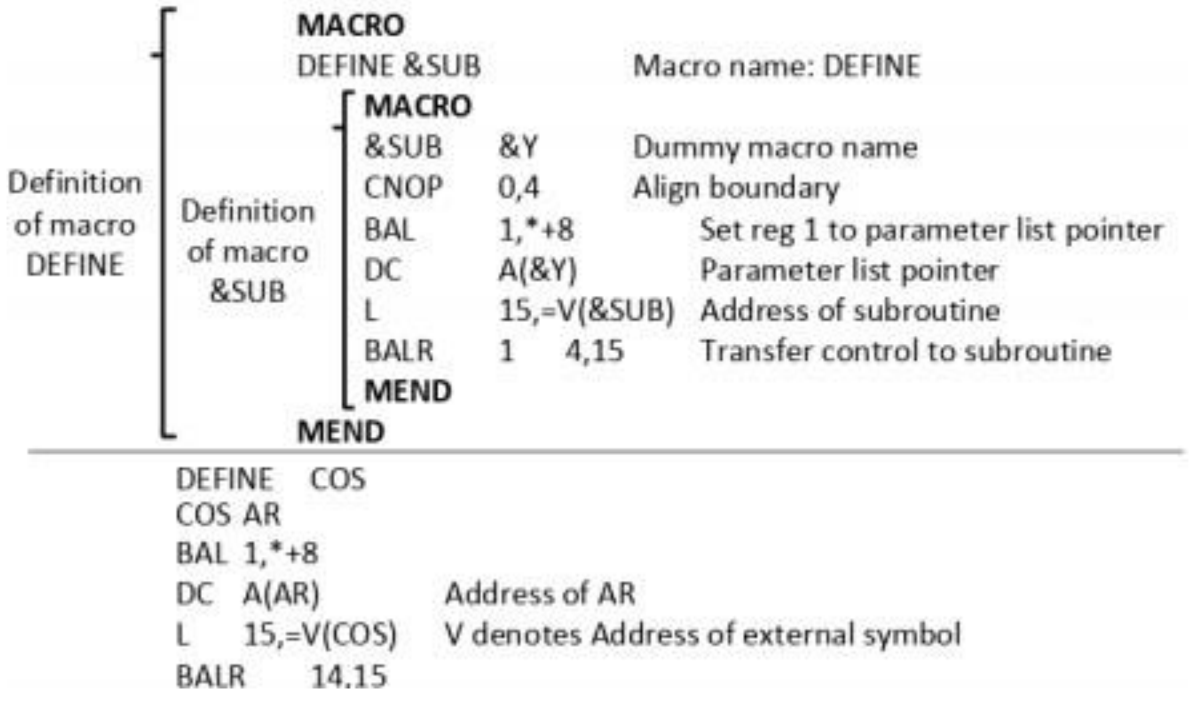
Also known as nested macro calls. A macro can be called within another macro. A macro can call itself (using AIF or AGO) so long as it doesn't go into an infinite loop. Macro calls within macros can have several levels.



```
Source                          Expanded Source        Expanded Source
                                (Level 1)              (Level 2)
MACRO
ADD1      &ARG
L     1,&ARG                    Expansion of           Expansion of
A     1,=F'1'                   ADDS                   ADD1
ST    1,&ARG
MEND
MACRO                                                  L     1,DATA1
ADDS      &ARG1,&ARG2,&ARG3                            A     1,=F'1'
ADD1      &ARG1                 ADD1      DATA1        ST    1,DATA1
ADD1      &ARG2                                        L     1,DATA2
ADD1      &ARG3                                        A     1,=F'1'
MEND                            ADD1      DATA2        ST    1,DATA2
                                                       L     1,DATA3
ADDS                                                   A     1,=F'1'
DATA1,DATA2,DATA3               ADD1      DATA3        ST    1,DATA3
```

Macro Instruction defining Macros

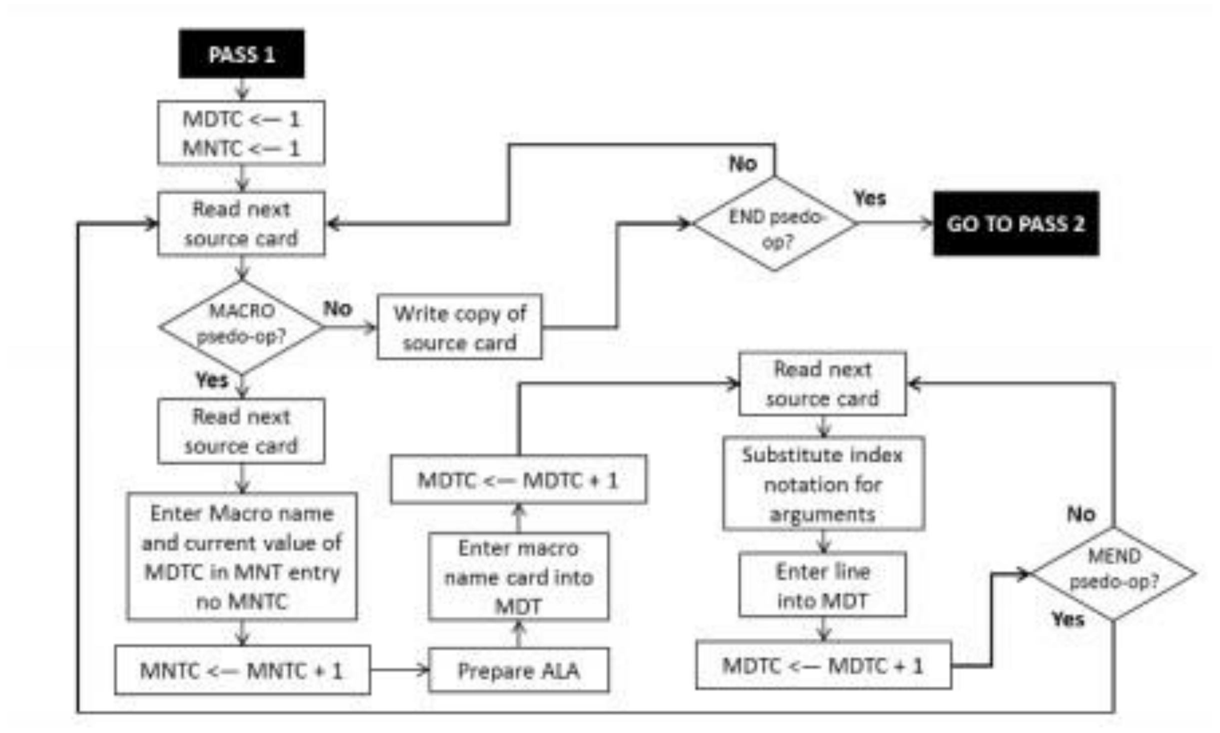
Macros can be defined within a macro. Inner macro definition is not defined until after the outer macro has been called. Group of macros can be defined for subroutine calls with some standardized calling sequence.



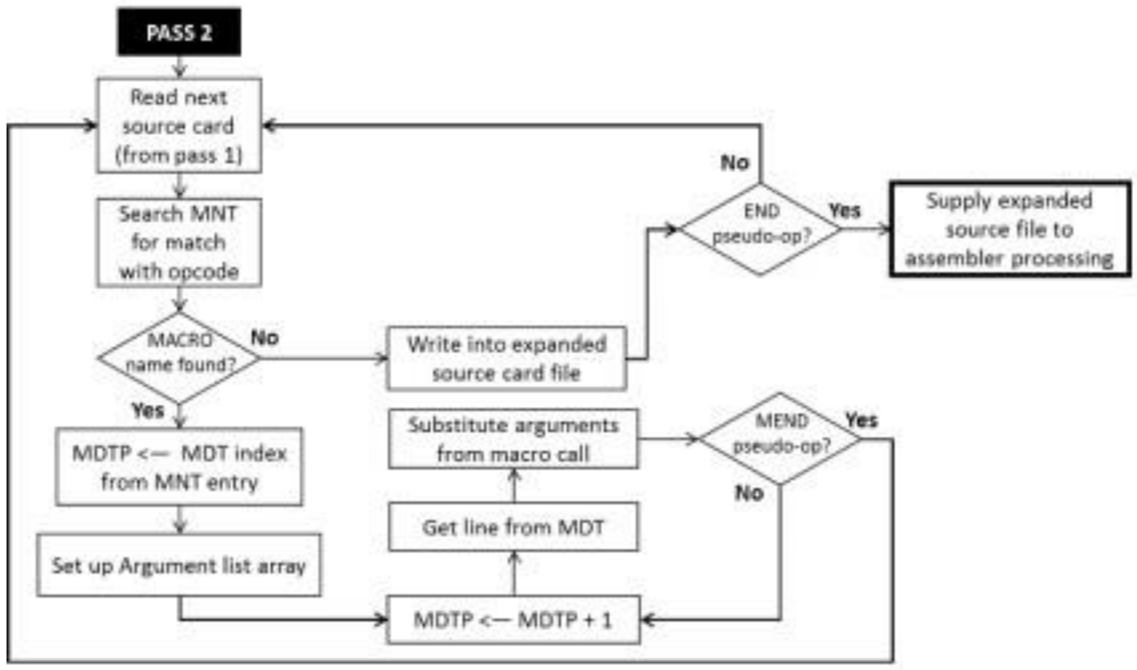
```
                           MACRO
                           DEFINE &SUB             Macro name: DEFINE
                              MACRO
                              &SUB     &Y          Dummy macro name
Definition                    CNOP     0,4         Align boundary
of macro     Definition       BAL      1,*+8           Set reg 1 to parameter list pointer
DEFINE       of macro         DC       A(&Y)           Parameter list pointer
             &SUB             L        15,=V(&SUB)     Address of subroutine
                              BALR     1    4,15       Transfer control to subroutine
                              MEND
                           MEND
                    DEFINE   COS
                    COS AR
                    BAL 1,*+8
                    DC   A(AR)          Address of AR
                    L    15,=V(COS)     V denotes Address of external symbol
                    BALR      14,15
```

Flowchart of a 2 pass macro processor:

Pass 1 –



Pass 2 –

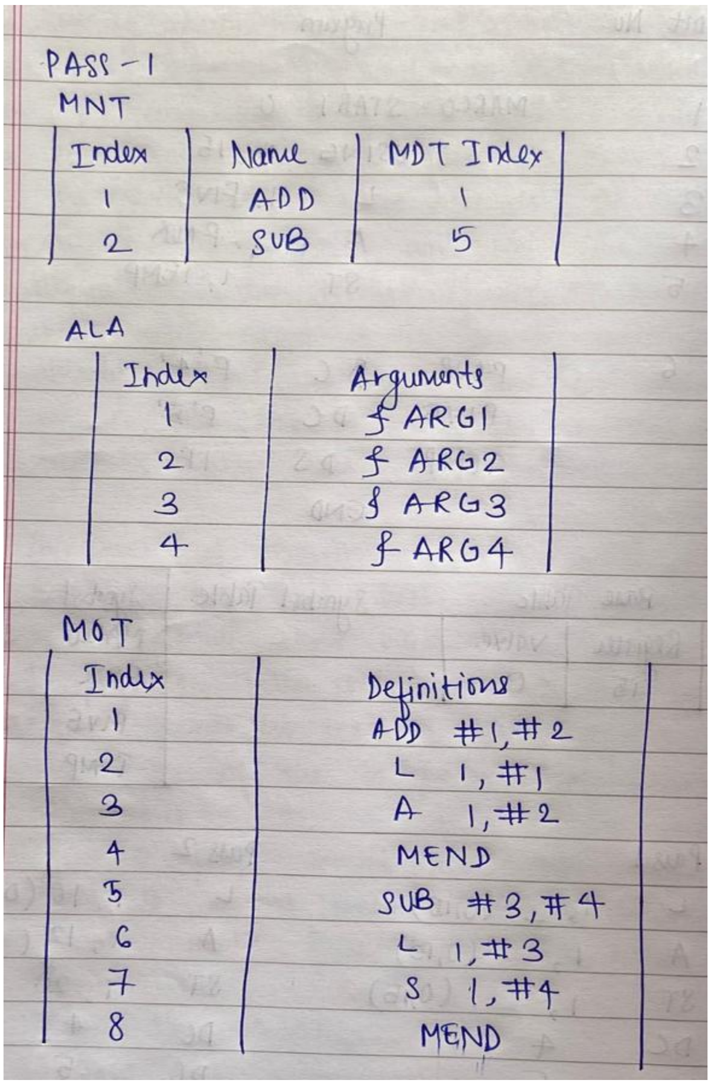


PASS 2
Read next source card (from pass 1)
Search MNT for match with opcode
MACRO name found?
No
Yes
Write into expanded source card file
END pseudo-op?
No
Yes
Supply expanded source file to assembler processing
MDTP <— MDT index from MNT entry
Set up Argument list array
MDTP <— MDTP + 1
Get line from MDT
Substitute arguments from macro call
MEND pseudo-op?
Yes
No

Example:

source.txt

```
ABC START
MACRO
ADD &ARG1 , &ARG2
L 1  , &ARG1
A 1 , &ARG2
MEND
MACRO
SUB &ARG3 , &ARG4
L 1  , &ARG3
S 1 , &ARG4
MEND
ADD DATA1 , DATA2
SUB DATA1 , DATA2
DATA1  DC  F'9'
DATA2  DC  F'5'
END
```



PASS - I

MNT

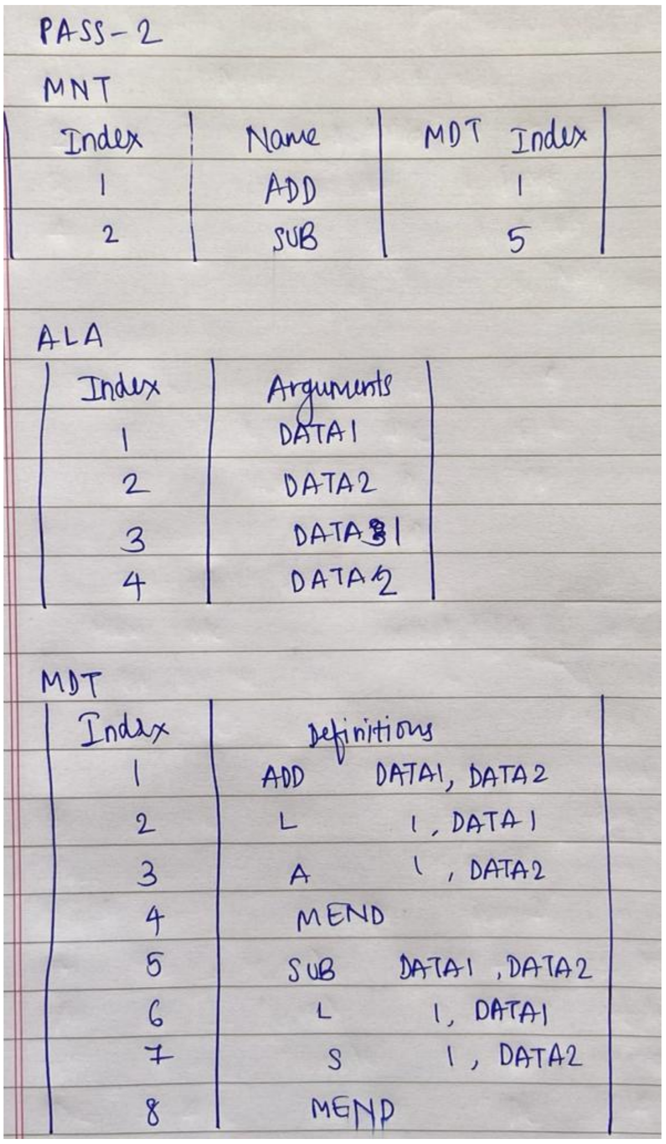
Index	Name	MDT Index
1	ADD	1
2	SUB	5

ALA

Index	Arguments
1	&ARG1
2	&ARG2
3	&ARG3
4	&ARG4

MDT

Index	Definitions
1	ADD #1,#2
2	L 1,#1
3	A 1,#2
4	MEND
5	SUB #3,#4
6	L 1,#3
7	S 1,#4
8	MEND



PASS-2

MNT

Index	Name	MDT Index
1	ADD	1
2	SUB	5

ALA

Index	Arguments
1	DATA1
2	DATA2
3	DATA~~3~~1
4	DATA~~4~~2

MDT

Index	Definitions
1	ADD DATA1, DATA2
2	L 1, DATA1
3	A 1, DATA2
4	MEND
5	SUB DATA1 ,DATA2
6	L 1, DATA1
7	S 1, DATA2
8	MEND

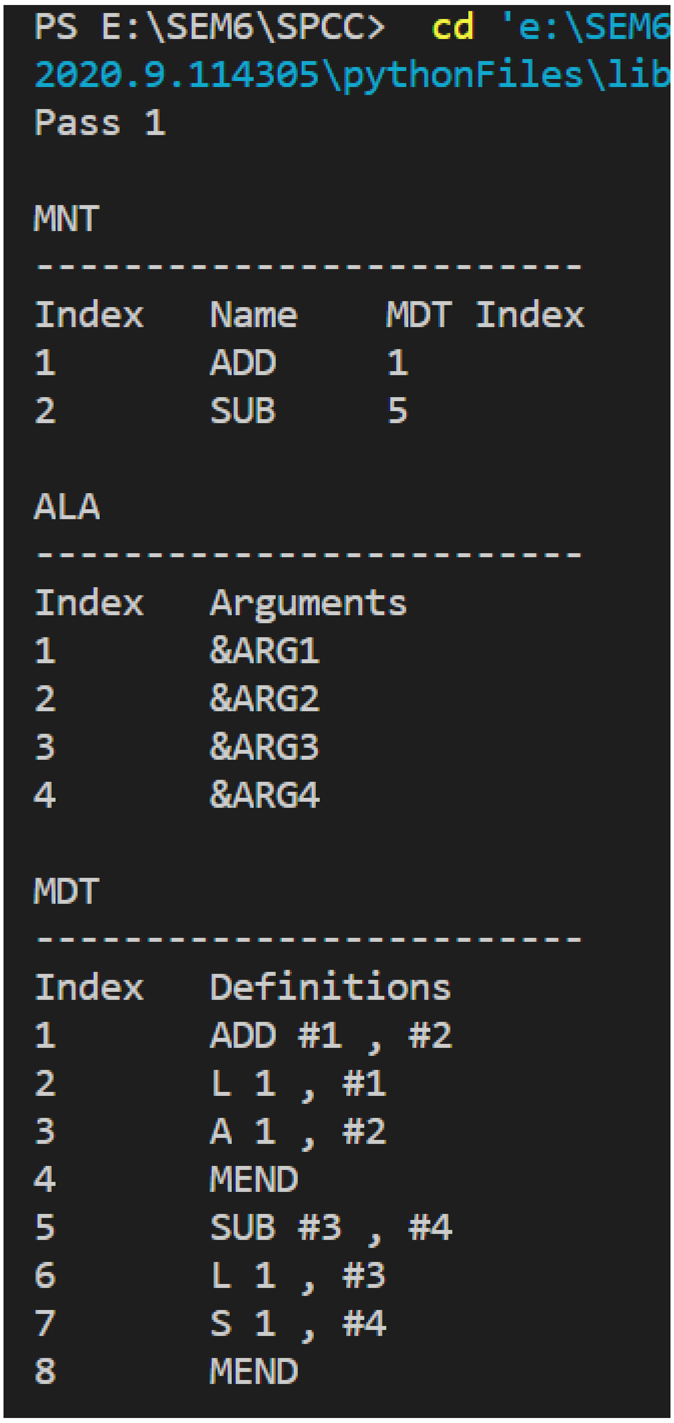
IMPLEMENTATION:

```
with open("source.txt","r") as fi:
    content=fi.readlines()

words=[]
for line in content:
    words.append(line.strip().split(" "))

MNT=[]
ALA={}
MDT=[]
MNTC=0
MDTC=0
```

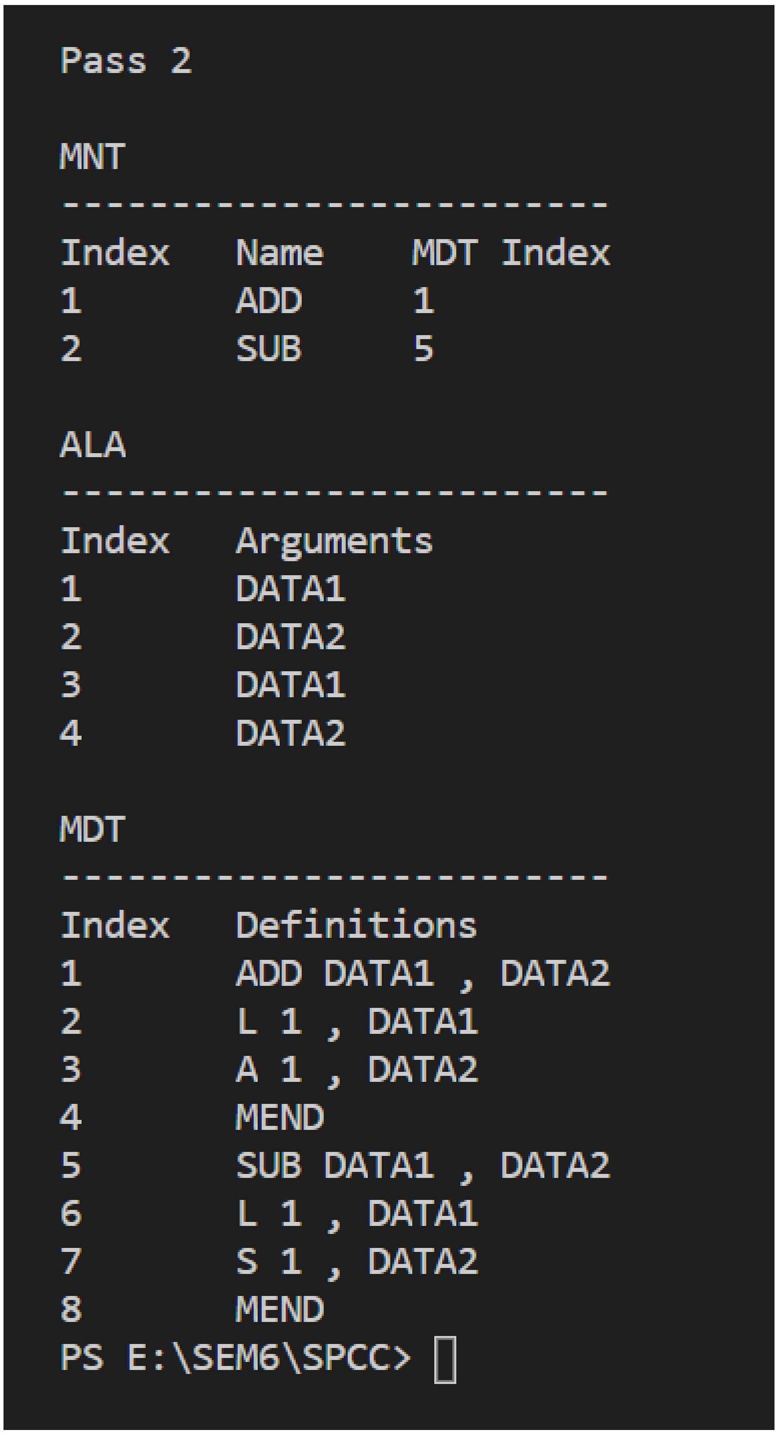
```python
ALAC=0
global_tracker={}
num=25

for i in range(len(words)):
    if (words[i][0]=="MACRO"):
        j=1
        arg_list=[]
        while (words[i+j][0]!="MEND"):
            if (j==1):
                MNTC+=1
                MDTC+=1
                MNT.append((MNTC,words[i+j][0],MDTC))
                formatted_line=words[i+j][0]+" "
                for word in words[i+j]:
                    if ("&" in word):
                        ALAC+=1
                        ALA[word]=ALAC
                        formatted_line+="#"+str(ALA[word])+" , "
                        arg_list.append(word)
                global_tracker[words[i+j][0]]=arg_list
                formatted_line=formatted_line[:len(formatted_line)-3]
                MDT.append([MDTC,formatted_line])
            else:
                formatted_line=""
                for word in words[i+j]:
                    if (word not in arg_list):
                        formatted_line+=word+" "
                    else:
                        formatted_line+="#"+str(ALA[word])
                MDTC+=1
                MDT.append([MDTC,formatted_line])
            j+=1
        MDTC+=1
        MDT.append([MDTC,"MEND"])

#Pass 1
print("Pass 1")
print("\nMNT")
print("-"*num)
print("Index\tName\tMDT Index")
for values in MNT:
    print(f"{values[0]}\t{values[1]}\t{values[2]}")

print("\nALA")
print("-"*num)
print("Index\tArguments")
```

```
for key,value in ALA.items():
    print(f"{value}\t{key}")

print("\nMDT")
print("-"*num)
print("Index\tDefinitions")
for values in MDT:
    print(f"{values[0]}\t{values[1]}")

gen_strings=[]
for macro_name,arguments in global_tracker.items():
    formatted_line=macro_name+" "
    for arg in arguments:
        formatted_line+=arg+" , "
    formatted_line=formatted_line[:len(formatted_line)-3]
    gen_strings.append(formatted_line.split(" "))

mapper={}
for x in gen_strings:
    for word in words:
        if (word[0]==x[0] and word!=x):
            for i in range(len(word)):
                if (word[i]!=x[i]):
                    mapper[ALA[x[i]]]=word[i]

for i in range(len(MDT)):
    _,m_content=MDT[i]
    for key,value in mapper.items():
        if ("#"+str(key) in m_content):
            m_content=m_content.replace("#"+str(key),mapper[key])
    MDT[i][1]=m_content

#Pass 2
print("\nPass 2")
print("\nMNT")
print("-"*num)
print("Index\tName\tMDT Index")
for values in MNT:
    print(f"{values[0]}\t{values[1]}\t{values[2]}")

print("\nALA")
print("-"*num)
print("Index\tArguments")
for key,value in mapper.items():
    print(f"{key}\t{value}")

print("\nMDT")
print("-"*num)
print("Index\tDefinitions")
for values in MDT:
    print(f"{values[0]}\t{values[1]}")
```

OUTPUT:



```
PS E:\SEM6\SPCC>  cd 'e:\SEM6
2020.9.114305\pythonFiles\lib
Pass 1

MNT
-------------------------
Index   Name    MDT Index
1       ADD     1
2       SUB     5

ALA
-------------------------
Index   Arguments
1       &ARG1
2       &ARG2
3       &ARG3
4       &ARG4

MDT
-------------------------
Index   Definitions
1       ADD #1 , #2
2       L 1 , #1
3       A 1 , #2
4       MEND
5       SUB #3 , #4
6       L 1 , #3
7       S 1 , #4
8       MEND
```



```
Pass 2

MNT
-------------------------
Index   Name    MDT Index
1       ADD     1
2       SUB     5

ALA
-------------------------
Index   Arguments
1       DATA1
2       DATA2
3       DATA1
4       DATA2

MDT
-------------------------
Index   Definitions
1       ADD DATA1 , DATA2
2       L 1 , DATA1
3       A 1 , DATA2
4       MEND
5       SUB DATA1 , DATA2
6       L 1 , DATA1
7       S 1 , DATA2
8       MEND
PS E:\SEM6\SPCC> ▯
```

CONCLUSION:

The working of a two pass macro processor is demonstrated. The output of the program was cross checked with actual