System Programming and Operating Systems Lab

ASSIGNMENT 9

Name: Shrirang Mhalgi
Roll No: 322008
Batch: B1

1 Date of Completion:

13/03/2019

2 Aim:

Write a Java program for pass-II of a two-pass macro-processor. The output of assignment-3 (MNT, MDT and file without any macro definitions) should be input for this assignment.

3 Objectives:

To implement basic Macroprocessor

4 Theory

Macro definition starts with the key word MACRO and end with the keyword MEND .i.e it is defined in the following manner MACRO //macro name //body of macro MEND

The process of replacement is called expanding the macro. In the above case the macro definition is not appeared in the expanded source code.

4.1 MACROPROCESSOR IMPLEMENTATION:

It involves four steps. They are 1. Recognize macro definitions Macro processor must recognize macro definitions identified by the MACRO and MEND pseudo operations. 2. Save the definitions Macro definitions should be saved in MACRO DEFINITION TABLE (MDT) 3. Recognize calls The processor must recognize macro calls that appear as operation mnemonics. 4. Expand calls and substitute arguments The processor must substitute for dummy argument corresponding arguments from a macro calls

4.1.1 Pass 1 database:

1. Input source
2. Output macro source deck copy for use by pass2
3. MDT used to store the body of the macro definitions.
4. The macro name table (MNT) used to store the names of defined macros.

5. The macro definition table counter (MDTC) used to indicate the next available entry in the MDT.
6. The macro name table counter(MNTC)used to indicate the next available entry in the MNT
7. The Argument List Array (ALA) used to substitute index markers for dummy arguments before storing a macro definition.

4.1.2 Pass 2 database:

1. The copy of the input macro source deck
2. Expanded source output
3. MDT created by pass 1
4. MNT created by pass 1
5. MDTP used to indicate the next line of text to be used during macro expansion.
6. The ALA used to substitute macro call arguments for the index markers in the stored macro definition.

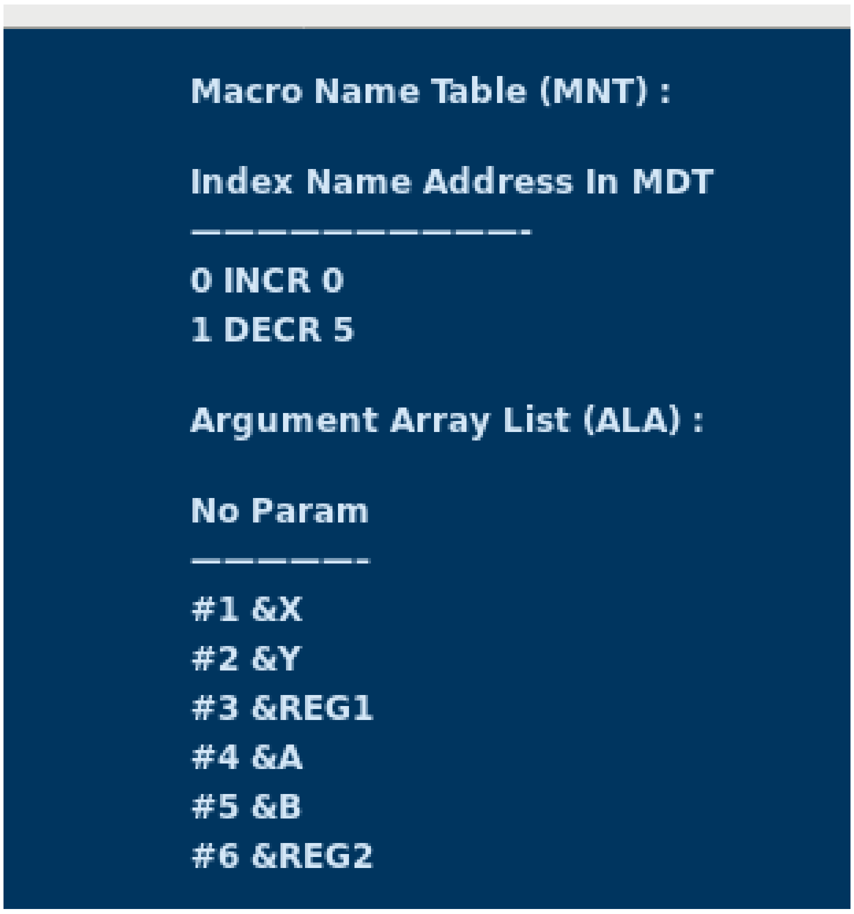
4.2 SPECIFICATION OF DATA BASE FORMAT:

The MNT and ALA format is described below

1. ARGUMENT LIST ARRAY:

   The ALA is used during both pass 1 and pass 2 with different functionalities. During pass 1 it is used to substitute index markers for dummy arguments before storing a macro definition. During pass 2 it is used to substitute macro call arguments for the index markers in the stored macro definition. During pass 2 upon encountering the call LOOP INCR DATA1,DATA2,DATA3 The macro call expander would prepare an argument list array

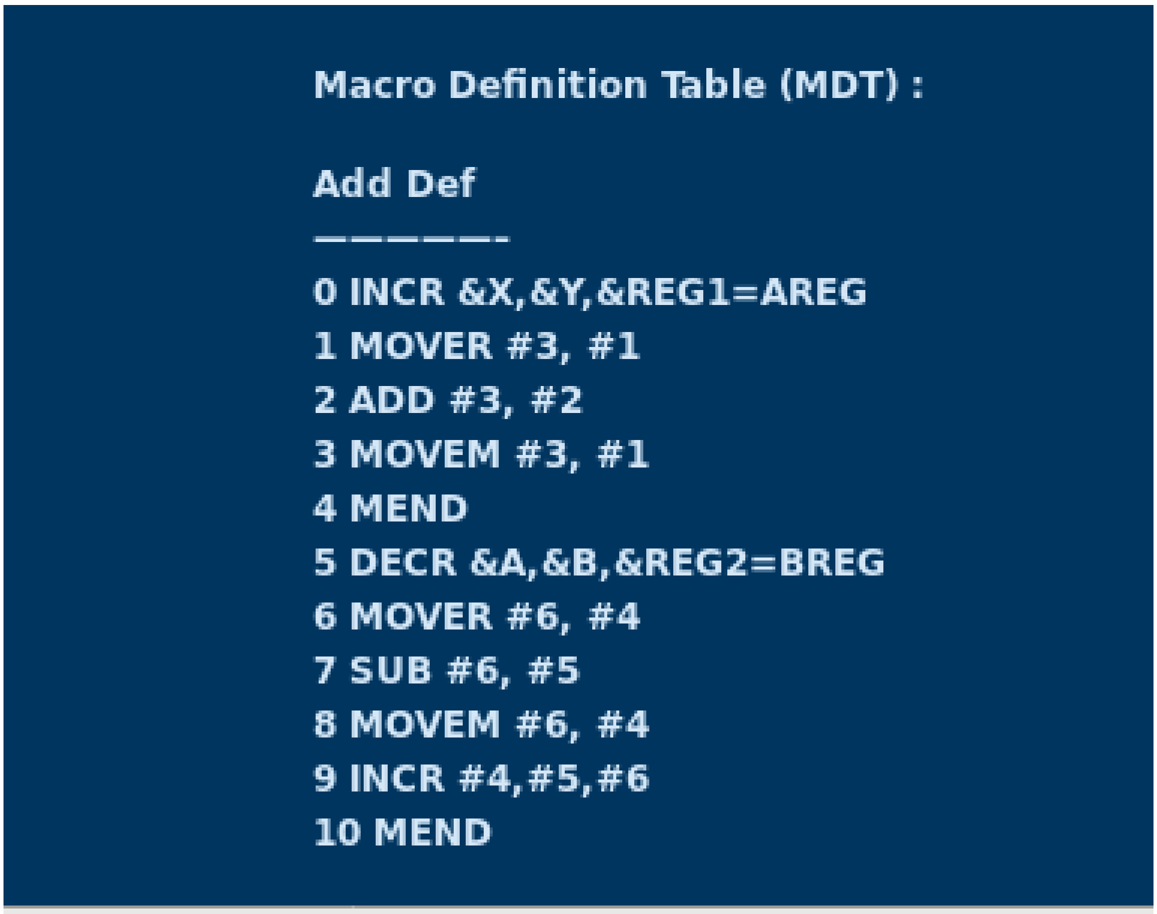
2. Index Arguments



```
Macro Name Table (MNT) :

Index Name Address In MDT
------------
0 INCR 0
1 DECR 5

Argument Array List (ALA) :

No Param
-------
#1 &X
#2 &Y
#3 &REG1
#4 &A
#5 &B
#6 &REG2
```

3. MACRO DEFINITION TABLE FORMAT:

   The MDT is a table of text lines . Every line of each macro definition, except MACRO line is stored in MDT. The MEND is kept to indicate the end of the definition.



```
Macro Definition Table (MDT) :

Add Def
------
0 INCR &X,&Y,&REG1=AREG
1 MOVER #3, #1
2 ADD #3, #2
3 MOVEM #3, #1
4 MEND
5 DECR &A,&B,&REG2=BREG
6 MOVER #6, #4
7 SUB #6, #5
8 MOVEM #6, #4
9 INCR #4,#5,#6
10 MEND
```

4.2.1 ALGORITHM

4.2.2 PASS I MACRO DEFINITION:

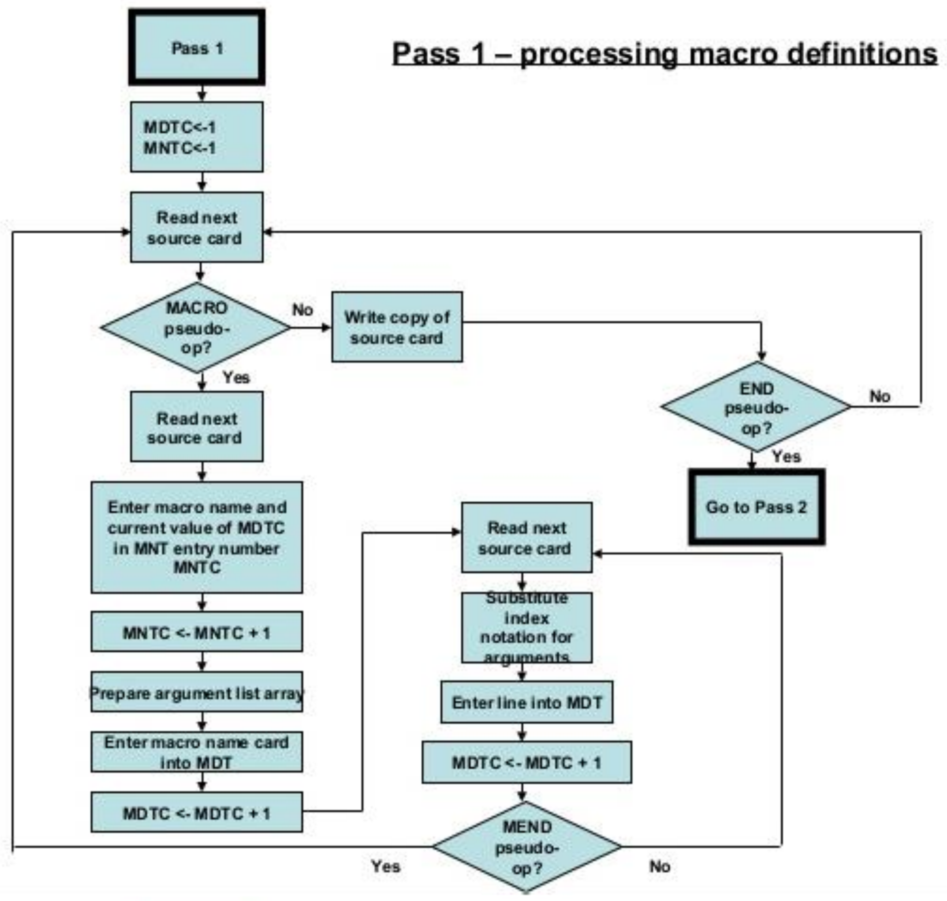
The algorithm tests for each input line. If it is a MACRO pseudo operation, the entire macro definition that follows is saved in the next available location in the macro definition table. The first line of the definition is the macro name line. The name is entered in to the macro name table (MNT), along with a pointer to the first location of the MDT entry of the definition. When the END pseudo op is encountered all of the macro definitions have been processed, so control transfers to pass 2. The entire algorithm is explained in pass 1 flowchart.

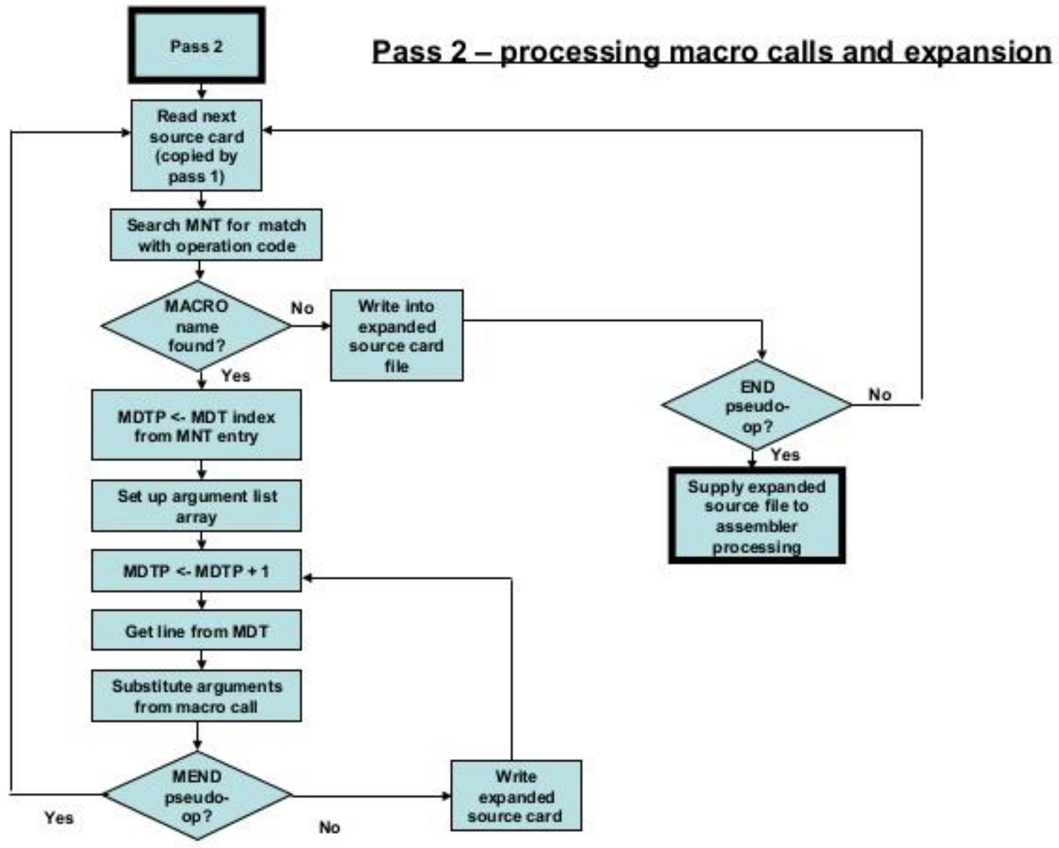
4.2.3 PASS 2 MACRO CALLS AND EXPANSION

When a macro call is found, the call processor sets a pointer, the MDTP, to the corresponding macro definitions stored in the MDT. The initial value of the MDTP is obtained from the MDT index field of the MNT entry. The macro expander prepares the ALA consisting of a table of dummy arguments indicates and corresponding arguments to the call.

Reading proceeds from the MDT, as each successive line is read, the values from the argument list are substituted for dummy arguments indicates in the macro definition. Reading of the MEND line in the MDT terminates expansion of the macro and scanning continues from the input file. When the END op is encountered the expanded source is transferred to the assembler for further processing.

FLOWCHART


Pass 1 – processing macro definitions
Pass 1
MDTC<-1
MNTC<-1
Read next source card
MACRO pseudo-op?
No
Write copy of source card
Yes
Read next source card
Enter macro name and current value of MDTC in MNT entry number MNTC
MNTC <- MNTC + 1
Prepare argument list array
Enter macro name card into MDT
MDTC <- MDTC + 1
Read next source card
Substitute index notation for arguments
Enter line into MDT
MDTC <- MDTC + 1
MEND pseudo-op?
Yes
No
END pseudo-op?
No
Yes
Go to Pass 2


Pass 2 – processing macro calls and expansion
Pass 2
Read next source card (copied by pass 1)
Search MNT for match with operation code
MACRO name found?
No
Write into expanded source card file
Yes
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
Write expanded source card
END pseudo-op?
No
Yes
Supply expanded source file to assembler processing

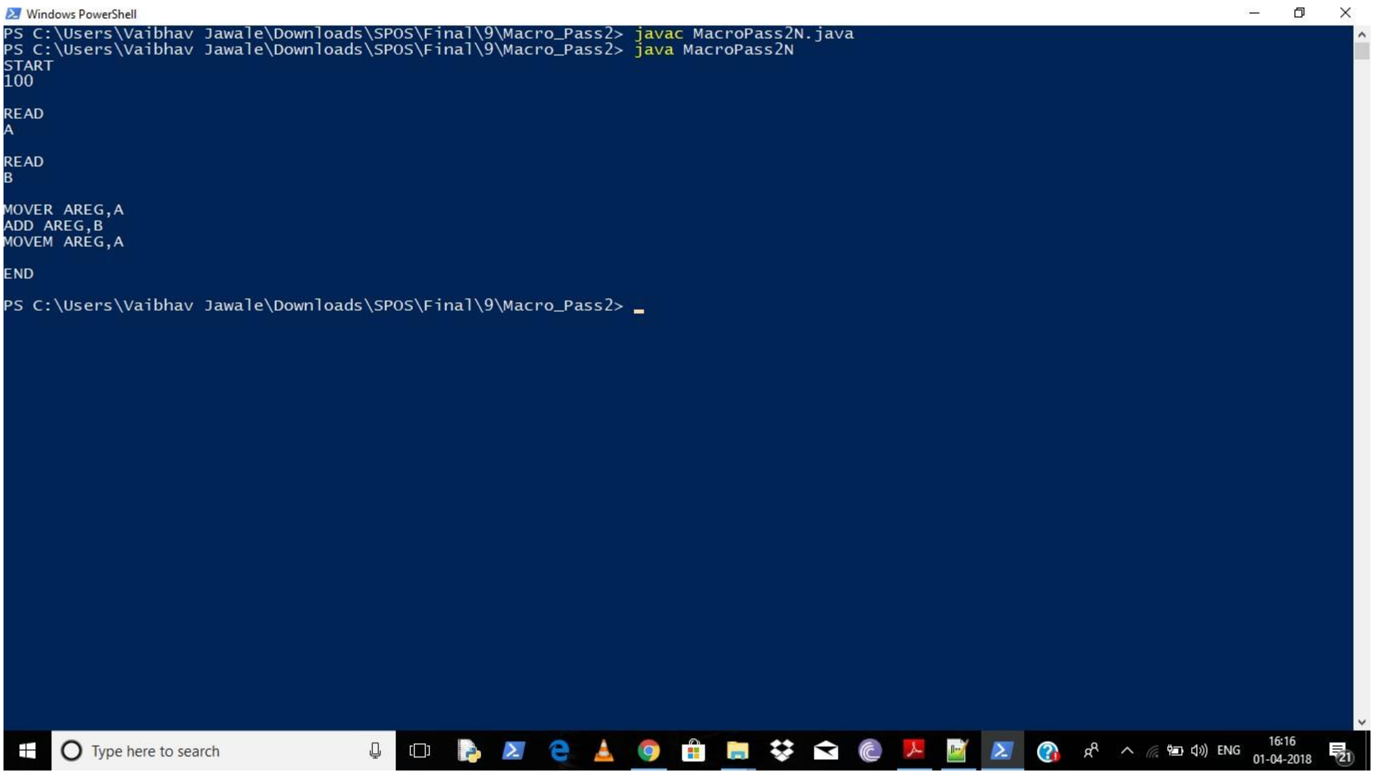
5 Code

```
import java.io.*;
import java.lang.String;
import java.util.Scanner;

public class MacroPass2N {
public static void main(String []args) throws Exception
{
File f=new File("program.txt");
Scanner s=new Scanner(f);

String[] tokens,macro_args;
boolean is_macro=false;
while(s.hasNextLine())
{
tokens=s.nextLine().split(" ");

for(int i=0;i<tokens.length;i++)
{
is_macro=check_if_macro(tokens[i]);if(
is_macro==true)
{
macro_args=tokens[i+1].split(",");
expand(tokens[i],macro_args);
//System.out.println(tokens[i]+" "+macro_args[0]+" "+macro_args[1]);


}
else
{
if(!tokens[i].contains(","))
System.out.println(tokens[i]);
}
}

System.out.println();

}

s.close();

}

static void expand(String macro_name,String[] macro_args)
throws FileNotFoundException
{
File f1=new File("contents.txt");
```

```
Scanner s 1=new Scanner ( f 1 ) ;

String [ ]  tokens_arr ;
while (! s1 . nextLine ( ) . contains ("MDT: " ) )
s1 . nextLine ( ) ;
while (! s1 . nextLine ( ) . contains (macro_name ) )
continue ;

String  def ="";
while ( ! def . c o n ta i n s ("MEND" ))
{
def=s1 . nextLine ( ) ;

def=  def . replace ("#0", macro_args [ 0 ] ) ;
def=def . replace ("#1", macro_args [ 1 ] ) ;
if (! def . contains ("MEND"))
System . out . println ( def ) ;
}

}


static  boolean  check_if_macro ( String token ) throws FileNotFoundException
{

File  f1=new File (" contents . txt ");
Scanner  s1=new Scanner ( f1 ) ;

String [ ]  tokens_arr ;

while (! s1 . nextLine ( ) . contains ("MNT: " ) )
s1 . nextLine ( ) ;
while ( s 1 . hasNextLine ( ) )
{
tokens_arr=s 1 . nextLine ( ) . split ("   " ) ;
for ( int  i =0; i<tokens_arr . length ; i++)
{
if ( tokens_arr [ i ] . e q u a l s ( token ) )
{
s 1 . c l o s e ( ) ;

return true ;
}
}

}

s 1 . c l o s e ( ) ;
```

```
return false;

}

}
```

6 Output



Windows PowerShell

```
PS C:\Users\Vaibhav Jawale\Downloads\SPOS\Final\9\Macro_Pass2> javac MacroPass2N.java
PS C:\Users\Vaibhav Jawale\Downloads\SPOS\Final\9\Macro_Pass2> java MacroPass2N
START
100

READ
A

READ
B

MOVER AREG,A
ADD AREG,B
MOVEM AREG,A

END

PS C:\Users\Vaibhav Jawale\Downloads\SPOS\Final\9\Macro_Pass2> _
```

7 Conclusion

Hence we have studied that- Macro processor algorithm will make two systematic scans, or passes over the inputs text, searching first for macro definition and saving them, and then for macro calls as macros cannot be expanded until they are defined unlike symbols in assemblers. Hence the first pass handles the definition and the second pass handles the macro calls or expansions.