Experiment 07 : Two Pass Macro Processor

Learning Objective: Student should be able to Apply Two-pass Macro Processor.
Tools: Jdk1.8,Turbo C/C++, Python, Notepad++

Theory:

Macro processor is a program which is responsible for processing the macro.
There are four basic tasks/ functions that any macro instruction processor must perform.

1. **Recognize macro definition:**
   A macro instruction processor must recognize macro definitions identified by the MACRO and MEND pseudo-ops.
2. **Save the definitions:**
   The processor must store the macro instruction definitions, which it will need for expanding macro calls.
3. **Recognize calls:**
   The processor must recognize macro calls that appear as operation mnemonics. This suggests that macro names be handled as a type of op-code.
4. **Expand calls and substitute arguments:**
   The processor must substitute for dummy or macro definition arguments the corresponding arguments from a macro call; the resulting symbolic (in this case, assembly language) text is then substituted for the macro call. This text, of course, may contain additional macro definitions or calls.

Two Pass Macro processor:

The macro processor algorithm will two passes over the input text, searching for the macro definition & then for macro calls.

Data bases required for Pass1 & Pass2 Macro processor:

The following data bases are used by the two passes of the macro processor:

Pass 1 data bases:

1. The input macro source deck
2. The output macro source deck copy for use by pass 2
3. The Macro Definition Table (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

Pass 2 data bases:

1. The copy of the input macro source deck
2. The output expanded source deck to be used as input to the assembler
3. The Macro Definition Table (MDT), created by pass 1
4. The Macro Name Table (MNT), created by pass 1
5. The Macro Definition Table Pointer (MDTP), used to indicate the next line of text to be used during macro expansion
6. The Argument List Array (ALA), used to substitute macro call arguments for the index markers in the stored macro definition

Format of Databases:

1) Argument List Array:

- The Argument List Array (ALA) is used during both pass 1 and pass 2.

During pass 1, in order to simplify later argument replacement during macro expansion, dummy arguments in the macro definition are replaced with positional indicators when the definition is stored: The ith dummy argument on the macro name card 'is represented in the body of the macro by the index marker symbol #.

Where # is a symbol reserved for the use of the macro processor (i.e., not available to the programmers).

- These symbols are used in conjunction with the argument list prepared before expansion of a macro call. The symbolic dummy arguments are retained on the macro name card to enable the macro processor to handle argument replacement by name rather than by position.
- During pass 2 it is necessary to substitute macro call arguments for the index markers stored in the macro definition.

Argument List Array:

Index	8 bytes per entry
0	"bbbbbbbb" (all blank)
2	"DATA3bbb"
3	"DATA2bbb"
4	"DATA1bbb"

2) Macro Definition Table:

- The Macro Definition Table (MDT) is a table of text lines.
- Every line of each macro definition, except the MACRO line, is stored in the MDT. (The MACRO line is useless during macro expansion.)
- The MEND is kept to indicate the end of the definition; and the macro name line is retained to facilitate keyword argument replacement.

Macro Definition Table

8O bytes per entry

Index		Card	
.			.
.			.
15	&LAB	INCR	&ARG1,&AAG2,&AAG3
16	#0	A	1, #1
17		A	2, #2
18		A	3,#3
19		MEND	
.			.
.			.

2) The Macro Name Table (MNT) :

- MNT serves a function very similar to that of the assembler's Machine-Op Table (MOT) and Pseudo-Op Table(POT).
- Each MNT entry consists of a character string (the macro name) and a pointer (index) to the entry in the MDT that corresponds to the beginning of the macro definition.

Index	8 Bytes	4 Bytes
.	.	.
.	.	.
3	“INCRbbbb”	15
.	.	.
.	.	.

ALGORITHM

PASS I-MACRO DEFINITION: The algorithm for pass-l tests each input line. If-it is a MACRO pseudo-op:

1) The entire macro definition that follows is saved in the next available locations in the Macro Definition Table (MDT).
2) The first line of the definition is the macro name line. The name is entered into the Macro Name Table (MNT), along with a pointer to the first location of the MDT entry of the definition.
3) When the END pseudo-op is encountered, all of the macro definitions have been processed so control transfers to pass 2 in order to process macro calls.

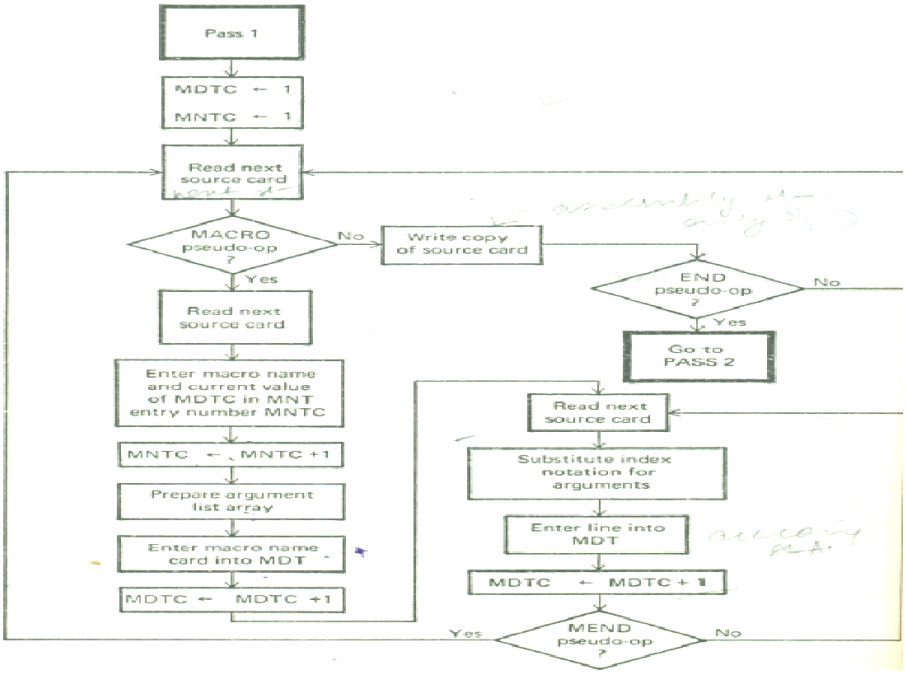
PASS2-MACRO CALLS AND EXPANSION:

The algorithm for pass 2 tests the operation mnemonic of each input line to see if it is a name in the MNT. When a call is found:-

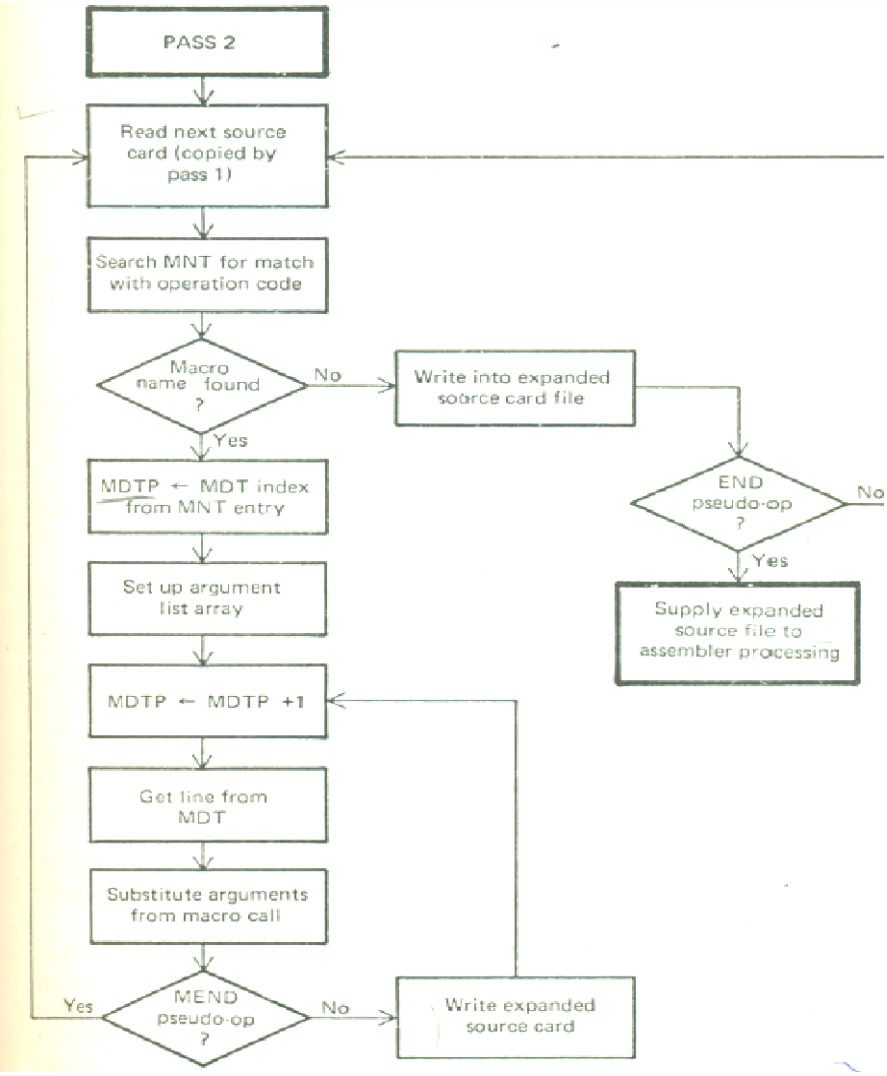
1) The macro processor sets a pointer, the Macro Definition Table Pointer (MDTP) , to the corresponding macro definition stored in the MDT. The initial value of the MDTP is obtained from the "MDT index" field of the MNT entry.
2) The macro expander prepares the Argument List Array(ALA) consisting of a table of dummy argument indices and corresponding arguments to the call.
3) Reading proceeds from the MDT; as each successive line is read, the values from the argument list are substituted for dummy argument indices in the macro definition.
4) Reading of the MEND line in the MDT terminates expansion of the macro, and scanning continues from the input file.
5) When the END pseudo-op is encountered, the expanded source deck is transferred to the assembler for further processing.

Flowchart of Pass1 & Pass2 Macro processor:

Pass 1: processing macro definitions



Pass2 : processing macro calls and expansion



Application: To design two-pass macroprocessor.

Design:

```
class MacroProcessor:
    def __init__(self):
        self.macros = {}

    def define_macro(self, name, definition):
        self.macros[name] = definition

    def expand_macros(self, input_text):
        lines = input_text.split('\n')
        expanded_lines = []
        for line in lines:
            if line.startswith('%macro'):
                macro_def = line.split()[1:]
                macro_name = macro_def[0]
                macro_body = '\n'.join(macro_def [1:])
                self.define_macro(macro_name, macro_body)
            else:
                expanded_lines.append(self.expand_line(line))
        return '\n'.join(expanded_lines)
    def expand_line(self, line):
        tokens = line.split()
        expanded_line = []
        for token in tokens:
            if token in self.macros:
                expanded_line.append(self.macros[token])
            else:
                expanded_line.append(token)
        return ' '.join(expanded_line)

def main():
    processor = MacroProcessor()
```

```
    # First pass
    with open('input_file.txt', 'r') as file:
        input_text = file.read()
        processed_text = processor.expand_macros(input_text)
    # Second pass
    with open('output_file.txt', 'w') as file:
        file.write(processed_text)

if __name__ == "__main__":
    main()
```

OUTPUT:

Input_file.txt

```
%macro add_two_numbers
LD R1, ARG1
ADD R1, ARG2
ST R1, RESULT
%endmacro
%macro multiply_two_numbers
LD R2, ARG1
LD R3, ARG2
MUL R4, R2, R3
ST R4, RESULT
%endmacro
main:
; Add two numbers
add_two_numbers
; Multiply two numbers
multiply_two_numbers
```

Output_file.txt

```
LD R1, ARG1
ADD R1, ARG2
ST R1, RESULT
%endmacro
LD R2, ARG1
LD R3, ARG2
MUL R4, R2, R3
ST R4, RESULT
%endmacro
main:
; Add two numbers
; Multiply two number
```

Result and Discussion: A macro instruction simplifies programming by expanding a group of statements wherever it appears in the code. It allows formal parameters and replaces each macro with corresponding source language statements. Macro processors handle mechanical details, independent of the computer architecture.

Learning Outcomes: The student should have the ability to

LO1: Describe the different database formats of two-pass Macro processor with the help of examples.

LO2: Design two-pass Macro processor.

LO3: Develop two-pass Macro processor.

LO4: Illustrate the working of two-pass Macro-processor.

Conclusion:

For Faculty Use

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				