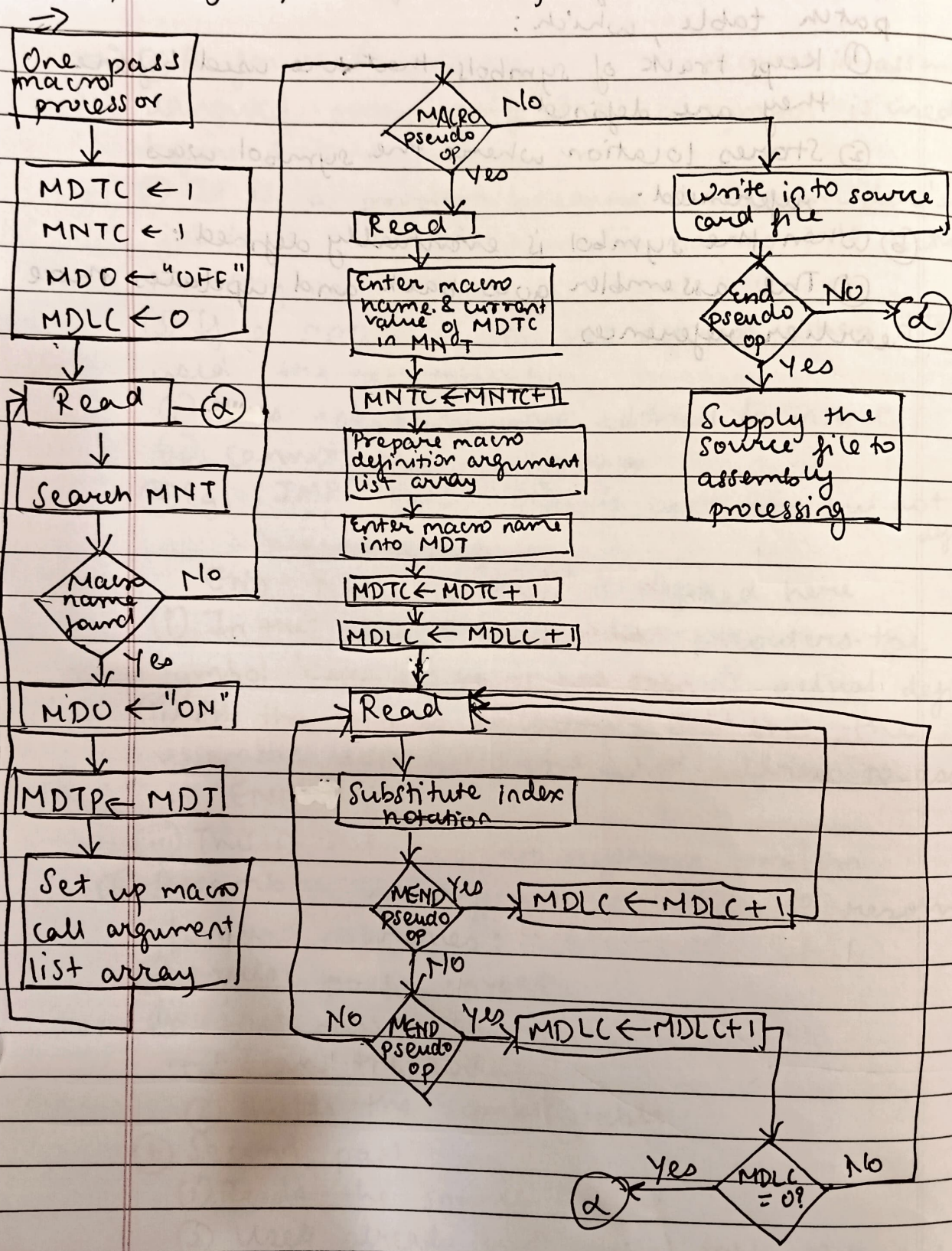
Module 3

Q) Single pass macro processor



(1) A one pass macroprocessor handles macro definition and expansion in a single pass for the source code.

(2) Steps:

(i) Scan the input line-by-line

(ii) If a MACRO directive is found:
   (a) Add macro name to MNT
   (b) Store macro body in MDT
   (c) Build ALA for formal parameters

(iii) If a macro call is found:
   (a) Search for macro in MNT
   (b) Retrieve body for MDT
   (c) Replace formal parameters using ALA with actual ones.
   (d) Insert expanded lines into the output.

(iv) If a normal line is found, then copy as it is to output.

(3) Key tables used in one pass macroprocessor

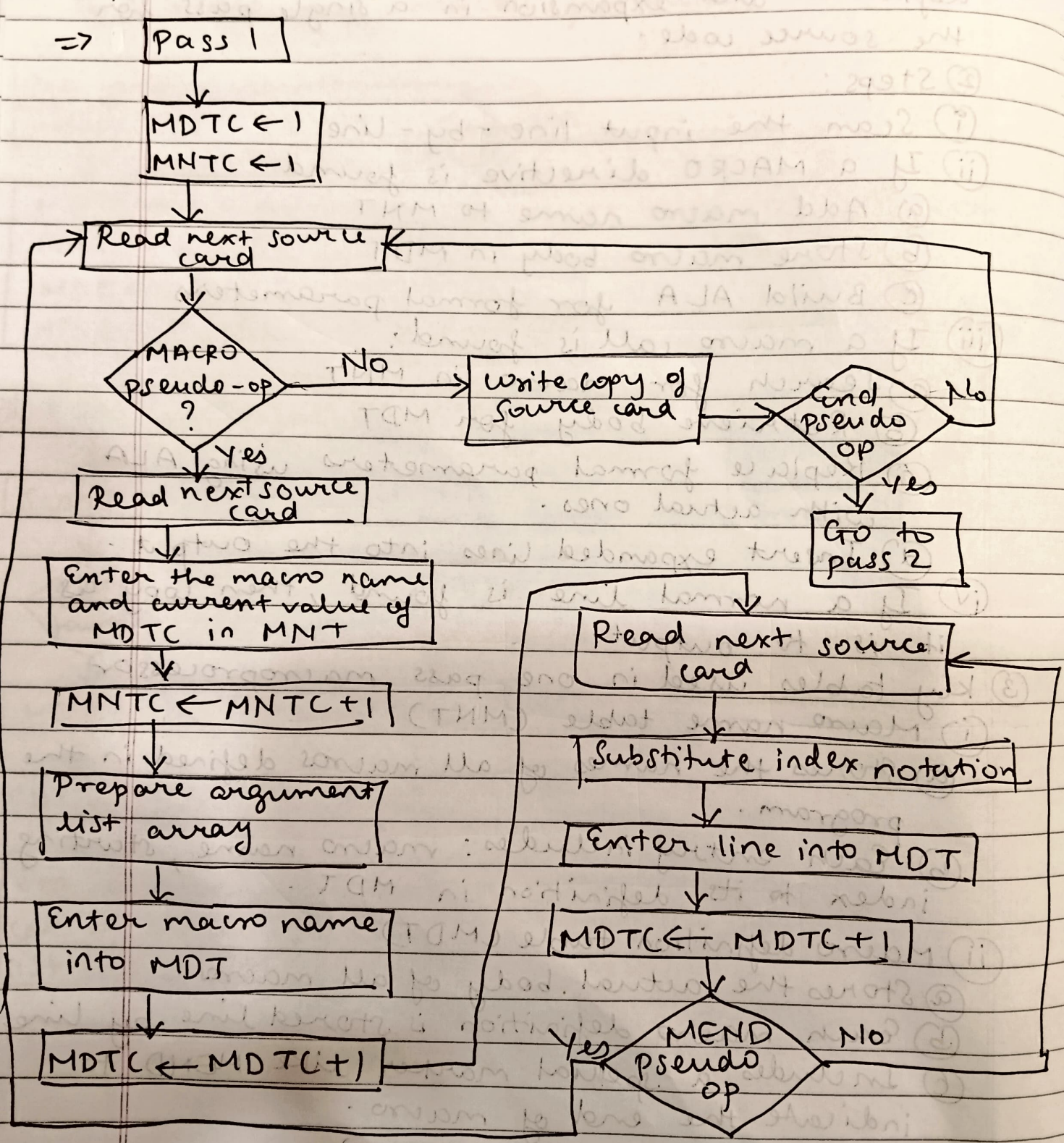
(i) Macro name table (MNT)
   (a) Stores the names of all macros defined in the program.
   (b) Each entry includes: macro name, starting index to its definition in MDT.

(ii) Macro definition table (MDT)
   (a) Stores the actual body of all macros.
   (b) Each macro definition is stored line by line
   (c) Includes a special marker like MEND to indicate the end of macro.

(iii) Argument list array (ALA)
   (a) Used to map formal parameters to actual parameters.
   (b) It handles parameter replacement during expansion.

Q) Pass 1 of two pass macro processor

=>



(1) The objective of pass I is to store macro definitions without generating actual code.

(2) Steps :

(i) Check each input line to identify the MACRO pseudo-op.

(ii) If a MACRO is detected :

(a) Save the macro name and its definition.

(b) Store the definition in MDT

(c) Store the macro name and its index to MDT entry in MNT

(d) Store macro parameters in ALA.

(iii) Continue until the MEND pseudo-op is encountered

(iv) Skip actual macro call during this phase.

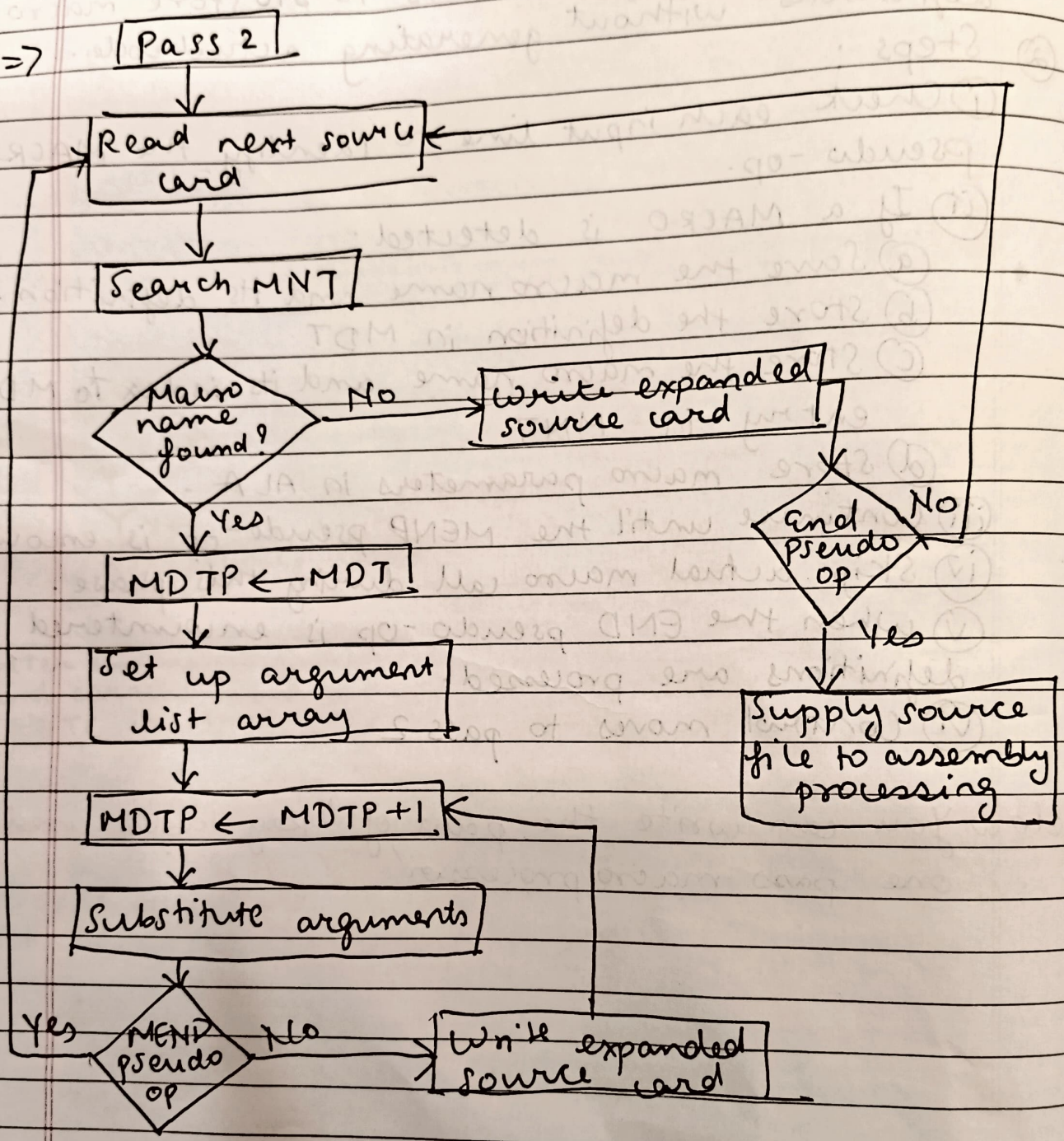
(v) When the END pseudo-op is encountered, all definitions are processed.

(vi) Control moves to pass 2

Below you can write the part of Key tables used in one pass macro processor,

Q) Pass 2 of two pass macroprocessor

=>



(1) The objective of pass 2 is to expand macro calls using the definitions stored using Pass 1

(2) Steps:

(i) Scan each input line again

(ii) If a macro call is encountered:

(a) Look up macro name in the MNT

(b) Retrieve macro body from MDT

(c) Use ALA to substitute actual arguments

(d) Expand the macro in the output code.

(iii) If a line is not a macro call, pass it unchanged to the output.

(iv) MEND line in the MDT terminates the expansion of the macro,

(v) When the END pseudo-op is encountered, the expanded source deck is transferred to the assembler for further processing.

(3) At this point, the assembler processes the expanded source code in the same way that it would process regular assembly code.

Below you can write the key tables used in one pass macroprocessor

Q) Parametrized macro

=> (1) Macro instruction argument are values or variables passed as input to a macro during its definition.

(2) They make macros flexible by allowing a single macro definition to behave differently based on the arguments provided.

(3) Eg:

Source:

```
INCR  &ARG
A     1,&ARG
A     2,&ARG
A     3,&ARG
MEND
....
....
INCR  DATA1
.....
INCR  DATA2
....
DATA1 DC  F'5'
DATA2 DC  F'10'
```

Expanded source:

```
A     1,DATA1
A     2,DATA1
A     3,DATA1

A     1,DATA2
A     2,DATA2
A     3,DATA2
```

(i) In the above example, the macro 'INCR' is defined using a parameter &ARG.
(ii) when INCR DATA1 is called, &ARG becomes DATA1.
(iii) Similarly, INCR DATA expands using DATA2

④ ways to specify arguments:

(a) Positional arguments.
  (i) Dummy arguments are matched based on the position of the arguments.
  (ii) The order of the parameters matters.
  (iii) Eg: INCR &ARG1, &ARG2, &ARG3

(b) Keyword arguments
  (i) Parameters are referenced by their names
  (ii) useful when some arguments may be optional.
  (iii) eg: INCR &ARG1=A, &ARG3=C, &ARG2=B