Chapter 4 Macro Processors

-- Basic Macro Processor Functions

Introduction

- A macro instruction (macro) is a notational convenience for the programmer
  - It allows the programmer to write shorthand version of a program (module programming)
- The macro processor replaces each macro instruction with the corresponding group of source language statements (*expanding*)
  - Normally, it performs no analysis of the text it handles.
  - It does not concern the meaning of the involved statements during macro expansion.
- The design of a macro processor generally is *machine independent!*

Basic macro processor functions

- Two new assembler directives are used in macro definition
  - **MACRO:** identify the beginning of a macro definition
  - **MEND:** identify the end of a macro definition
- Prototype for the macro
  - Each parameter begins with ‘&’

```
name   MACRO   parameters
             :
           body
             :
       MEND
```

  - Body: the statements that will be generated as the expansion of the macro.

Macro expansion

Source

```
M1      MACRO    &D1, &D2
        STA      &D1
        STB      &D2
        MEND
   .
M1 DATA1, DATA2
   .
M1 DATA4, DATA3
```

Expanded source

```
   .
   .
   .
        STA     DATA1
        STB     DATA2
   .
        STA     DATA4
        STB     DATA3
   .
```

Example of macro definition Figure 4.1, pp. 178

```
COPY       START     0              COPY FILE FROM INPUT TO OUTPUT
RDBUFF     MACRO     &INDEV, &BUFADR &RECLTH
.
.          MACRO TO READ RECORD INTO BUFFER
.
           CLEAR      X             CLEAR LOOP COUNTER
           CLEAR      A
           CLEAR      S
          +LDT       #4096          SET MAXINUM RECORD LENTH
           TD         =X’&INDEV’    TEST INPUT DEVICE
           JEQ        *-3           LOOP UNTIL READY
           RD         =X’&INDEV’    READ CHARACTER INTO REG A
           COMPR       A , S        TEST FOR END OF RECORD
           JEQ         *+11         EXIT LOOP IF EOR
           STCH       &BUFADR, X    STORE CHARACTER IN BUFFER
           TIXR        T            LOOP UNLESS MAXIMUN LENGTH
           JLT         *-19         HAS BEEN RECARD
           STX        &RECLTH       SAVE RECORD LENGTH
           MEND
```

Macro invocation

- A macro invocation statement (a macro call) gives the name of the macro instruction being invoked and the arguments to be used in expanding the macro.
  - `macro_name   p1, p2, …`
- Difference between macro call and procedure call
  - Macro call: statements of the macro body are expanded each time the macro is invoked.
  - Procedure call: statements of the subroutine appear only one, regardless of how many times the subroutine is called.
- Question
  - How does a programmer decide to use macro calls or procedure calls?
    - From the viewpoint of a programmer
    - From the viewpoint of the CPU

Exchange the values of two variables

```
void exchange(int a, int b) {
  int temp;
  temp = a;
  a = b;
  b = temp;
  }

main() {
  int i=1, j=3;
  printf("BEFORE - %d %d\n", i, j);
  exchange(i, j);
  printf("AFTER - %d %d\n", i, j);
}
```

What’s the result?

Pass by Reference

```
void exchange(int *p1, int *p2) {
  int temp;
  temp = *p1;
  *p1 = *p2;
  *p2 = temp;
  }

main() {
  int i=1, j=3;
  printf("BEFORE - %d %d\n", i, j);
  exchange(&i, &j);
  printf("AFTER - %d %d\n", i, j);
}
```

12 Lines of Assembly Code

```
. Subroutine EXCH
EXCH     LDA     @P1
         STA      TEMP
         LDA     @P2
         STA     @P1
         LDA      TEMP
         STA     @P2
         RSUB
P1       RESW      1
P2       RESW      1
TEMP     RESW      1
```

```
MAIN     LDA     #1
         STA      I
         LDA     #3
         STA      J
. Call a subroutine
         LDA     #I
         STA      P1
         LDA     #J
         STA      P2
         JSUB     EXCH
I        RESW     1
J        RESW     1
         END      MAIN
```

Swap two variables by macro

```
#define swap(i,j) { int temp; temp=i; i=j; j=temp; }
```

```
main() {
  int i=1, j=3;
  printf("BEFORE - %d %d\n", i, j);
  swap(i,j);
  printf("AFTER - %d %d\n", i, j);
}
```

6 Lines of Assembly Code

```
MAIN     LDA     #1
         STA      I
         LDA     #3
         STA      J
. Invoke a macro
         LDA      I
         STA      TEMP
         LDA      J
         STA      I
         LDA      TEMP
         STA      J
I        RESW     1
J        RESW     1
TEMP     RESW     1
         END      MAIN
```

Macro expansion

- Each macro invocation statement will be expanded into the statements that form the body of the macro.
- Arguments from the macro invocation are substituted for the parameters in the macro prototype (according to their positions).
  - In the definition of macro: parameter
  - In the macro invocation: argument
- Comment lines within the macro body will be deleted.
- Macro invocation statement itself has been included as a comment line.
- The label on the macro invocation statement has been retained as a label on the first statement generated in the macro expansion.
  - We can use a macro instruction in exactly the same way as an assembler language mnemonic.

Example of macro invocation Figure 4.1, pp. 178

```
.                          MAIN PROGRAM
.
        FIRST    STL       RETADR             SAVE RETURN ADDRESS
        CLOOP    RDBUFF    F1,BUFFER,LENGTH   READ RECORD INTO BUFFER
                 LDA       LENGTH             TEST FOR END OF FILE
                 COMP      #0
                 JEQ       ENDFIL             EXIT IF EOF FOUND
                 WRBUFF    05,BUFFER,LENGTH   WRITE OUTPUT RECORD
                 J         CLOOP              LOOP
        ENDFIL   WRBUFF    05,EOF,THREE       INSERT EOF MARKER
                 J          @RETADR
        EOF      BYTE      C’EOF’
        THREE    WORD      3
        RETADR   RESW      1
        LENGTH   RESW      1                  LENGTH OF RECORD
        BUFFER   RESB      4096               4096-BYTE BUFFER AREA
                 END       FIRST
```

Example of macro expansion Figure 4.2, pp. 179

```
5       COPY     START    0                  COPY FILE FROM INPUT TO OUTPUT
180     FIRST    STL      RETADR             SAVE RETURN ADDRESS
190     .CLOOP   RDBUFF   F1,BUFFER,LENGTH   READ RECORD INTO BUFFER
190a    CLOOP    CLEAR    X                  CLEAR LOOP COUNTER
190b             CLEAR    A
190c             CLEAR    S
190d            +LDT     #4096               SET MAXIMUN RECORD LENGTH
190e             TD      =X’F1’              TEST INPUT DEVICE
190f             JEQ      *-3                LOOP UNTIL READY
190g             RD      =X’F1’              TEST FOR END OF RECORD
190h             COMPR    A, S               TEST FOR END OF RECORD
190i             JEQ      *+11               EXIT LOOP IF EOR
190j             STCH     BUFFER, X          STORE CHARACTER IN BUFFER
190k             TIXR     T                  LOOP UNLESS MAXIMUN LENGTH
190l             JLT      *-19               HAS BEEN REACHED
190M             STX      LENGTH             SAVE RECORD LENGTH
```

Example of macro expansion Figure 4.2, pp. 179

```
195               LDA     LENGTH               TEST FOR END OF FILE
200               COMP   #0
205               JEQ     ENDFIL               EXIT IF EOF FOUND
210               WRBUFF  05,BUFFER,LENGTH     WRITE OUTPUT RECORD
210a              CLEAR   X                    CLEAR LOOP COUNTER
210b              LDT     LENGTH
210c              LDCH    BUFFER,X             GET CHARACTER FROM BUFFER
210d              TD     =X’05’                TEST OUTPUT DEVICE
210e              JEQ     *-3                  LOOP UNTIL READY
210f              WD     =X’05’                WRITE CHARACTER
210g              TIXR    T                    LOOP UNTIL ALL CHARACTERS
210h              JLT     *-14                 HAVE BEEN WRITTEN
215               J       CLOOP                LOOP
220     .ENDFIL   WRBUFF  05,EOF,THREE         INSERT EOF MARKER
```

Example of macro expansion
Figure 4.2, pp. 179

```
220a   ENDFIL   CLEAR   X          CLEAR LOOP COUNTER
220b            LDT     THREE
220c            LDCH    EOF,X      GET CHARACTER FROM BUFFER
220d            TD     =X’05’      TEST OUTPUT DEVICE
220e            JEQ     *-3        LOOP UNTIL READY
220f            WD     =X’05’      WRITE CHARACTER
220g            TIXR    T          LOOP UNTIL ALL CHARACTERS
220h            JLT     *-14       HAVE BEEN WRITTEN
225             J      @RETADR
230    EOF      BYTE    C’EOF’
235    THREE    WORD    3
240    RETADR   RESW    1
245    LENGTH   RESW    1
250    BUFFER   RESB    4096       4096-BYTE BUFFER AREA
255             END     FIRST
```

No label in the macro body

- Problem of the label in the body of macro:
  - If the same macro is expanded multiple times at different places in the program …
  - There will be *duplicate labels*, which will be treated as errors by the assembler.
- Solutions:
  - Do not use labels in the body of macro.
  - Explicitly use PC-relative addressing instead.
    - `Ex, in RDBUFF and WRBUFF macros,`

      ```
      JEQ    *+11
      JLT    *-14
      ```
    - `It is inconvenient and error-prone.`
  - The way of avoiding such error-prone method will be discussed in Section 4.2.2

Two-pass macro processor

- You may design a two-pass macro processor
  - Pass 1:
    - Process all macro definitions
  - Pass 2:
    - Expand all macro invocation statements
- However, one-pass may be enough
  - Because all macros would have to be defined during the first pass before any macro invocations were expanded.
    - The definition of a macro must appear before any statements that invoke that macro.
  - Moreover, the body of one macro can contain definitions of other macros.

Example of recursive macro definition Figure 4.3, pp.182

- MACROS (for SIC)
  - Contains the definitions of RDBUFF and WRBUFF written in SIC instructions.

```
1          MACROS   MACOR              {Defines SIC standard version macros}
2          RDBUFF   MACRO              &INDEV,&BUFADR,&RECLTH
                    .
                    .                  {SIC standard version}
                    .
3                   MEND               {End of RDBUFF}
4          WRBUFF   MACRO              &OUTDEV,&BUFADR,&RECLTH
                    .
                    .                  {SIC standard version}
5                   MEND               {End of WRBUFF}
                    .
                    .
                    .
6                   MEND               {End of MACROS}
```

Example of recursive macro definition Figure 4.3, pp.182

- MACROX (for SIC/XE)
  - Contains the definitions of RDBUFF and WRBUFF written in SIC/XE instructions.

```
1    MACROX   MACRO          {Defines SIC/XE macros}
2    RDBUFF   MACRO          &INDEV,&BUFADR,&RECLTH
              .
              .              {SIC/XE version}
              .
3             MEND           {End of RDBUFF}
4    WRBUFF   MACRO          &OUTDEV,&BUFADR,&RECLTH
              .
              .              {SIC/XE version}
              .
5             MEND           {End of WRBUFF}
              .
              .
6             MEND           {End of MACROX}
```

Example of macro definitions

- A program that is to be run on SIC system could invoke MACROS whereas a program to be run on SIC/XE can invoke MACROX.
- However, defining MACROS or MACROX does not define RDBUFF and WRBUFF. These definitions are processed only when an invocation of MACROS or MACROX is expanded.

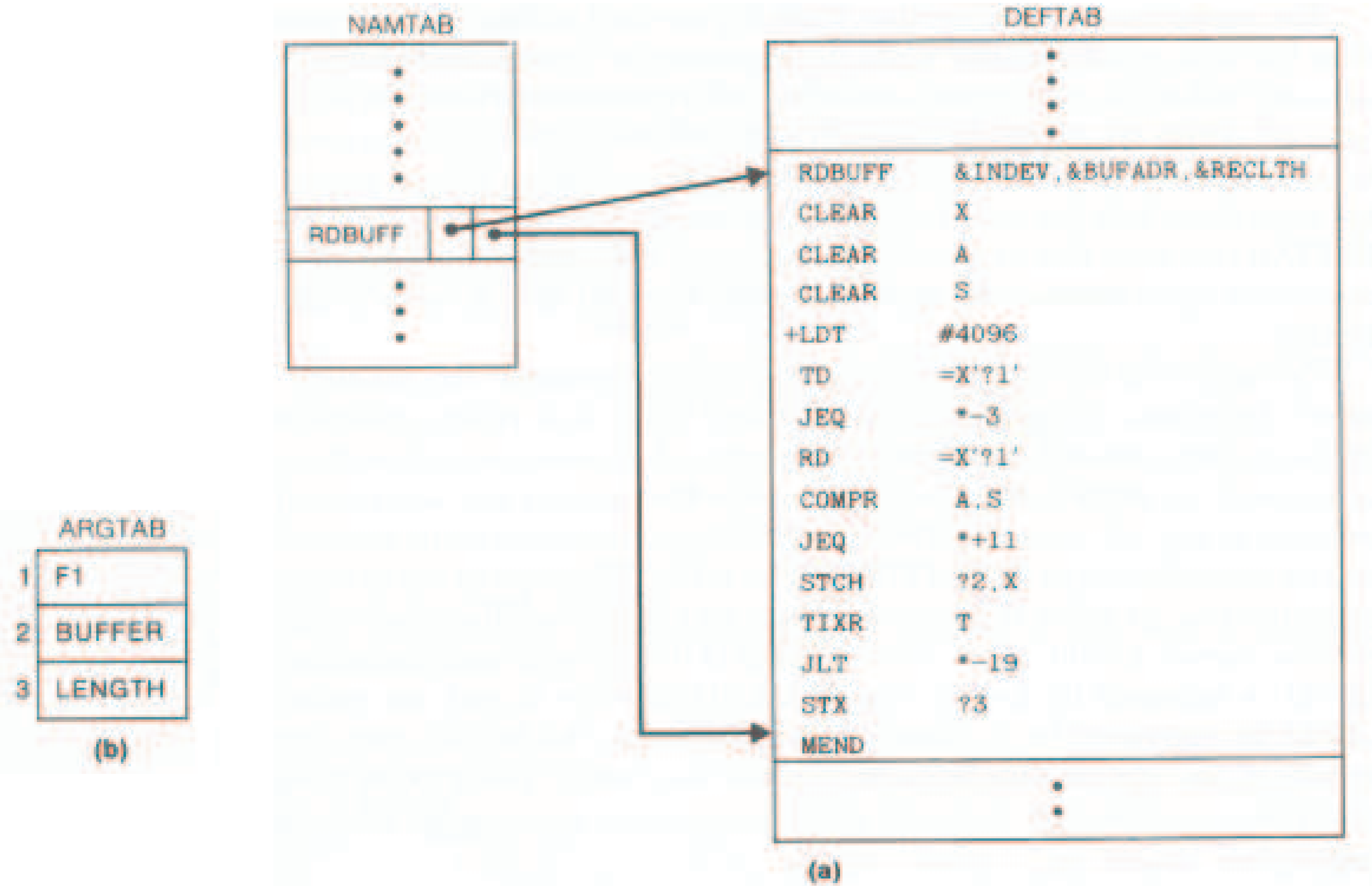
One-pass macro processor

- A one-pass macro processor that alternate between *macro definition* and *macro expansion* in a recursive way is able to handle recursive macro definition.
- Restriction
  - The definition of a macro must appear in the source program before any statements that invoke that macro.
  - This restriction does not create any real inconvenience.

Data structures for one-pass macro processor

- DEFTAB (definition table)
  - Stores the macro definition including *macro prototype* and *macro body*
  - Comment lines are omitted.
  - References to the macro instruction parameters are converted to a positional notation for efficiency in substituting arguments.
- NAMTAB
  - Stores macro names
  - Serves as an index to DEFTAB
    - `Pointers to the beginning and the end of the macro definition (DEFTAB)`
- ARGTAB
  - Stores the arguments of macro invocation according to their positions in the argument list
  - As the macro is expanded, arguments from ARGTAB are substituted for the corresponding parameters in the macro body.

Data structures



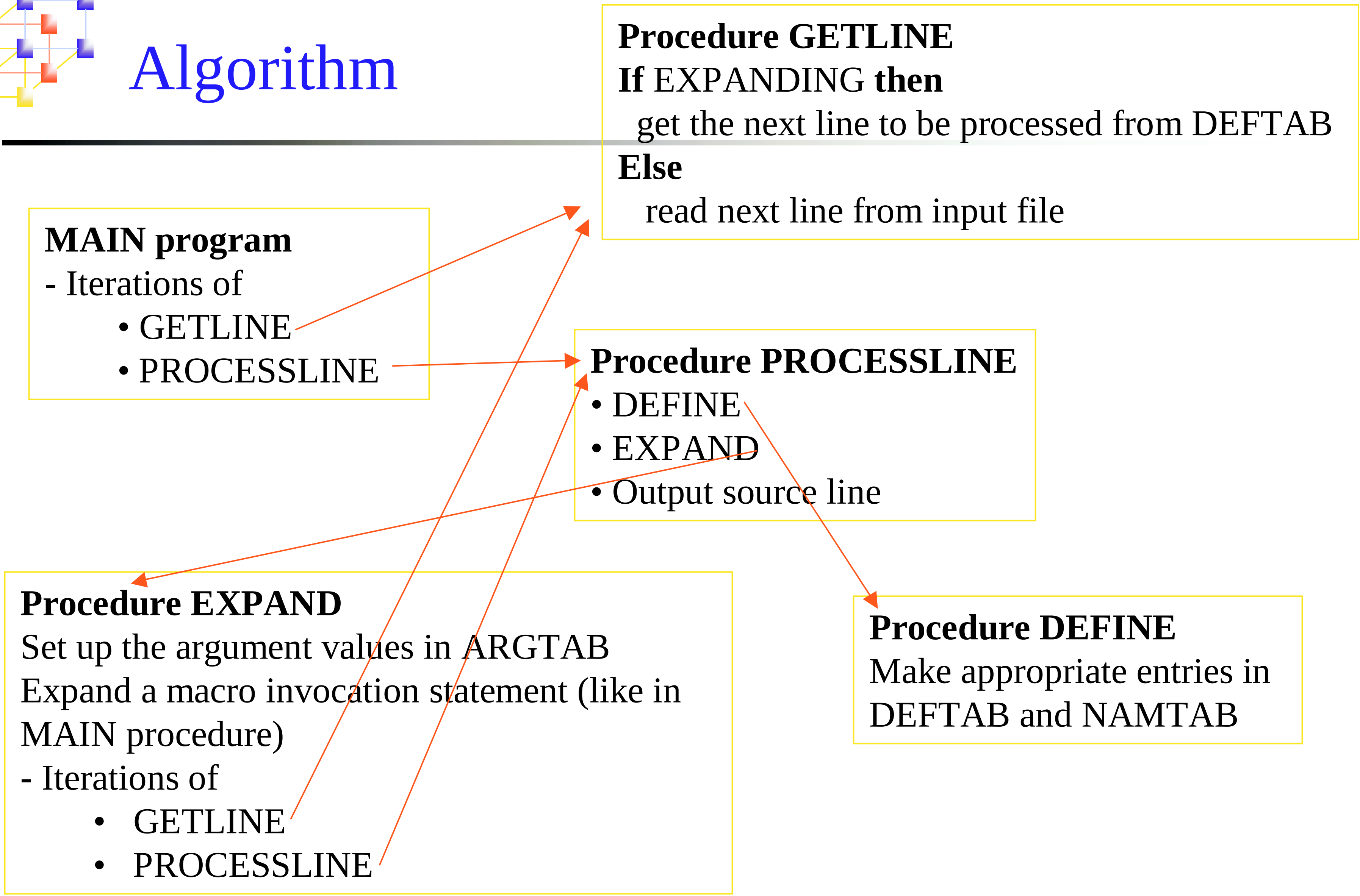
```
RDBUFF     &INDEV,&BUFADR,&RECLTH
CLEAR      X
CLEAR      A
CLEAR      S
+LDT      #4096
TD        =X'?1'
JEQ        *-3
RD        =X'?1'
COMPR      A,S
JEQ        *+11
STCH       ?2,X
TIXR       T
JLT        *-19
STX        ?3
MEND
```

(a)

	ARGTAB
1	F1
2	BUFFER
3	LENGTH

(b)

Algorithm



MAIN program

- Iterations of
  - GETLINE
  - PROCESSLINE

Procedure GETLINE

If EXPANDING **then**

get the next line to be processed from DEFTAB

Else

read next line from input file

Procedure PROCESSLINE

- DEFINE
- EXPAND
- Output source line

Procedure EXPAND

Set up the argument values in ARGTAB

Expand a macro invocation statement (like in MAIN procedure)

- Iterations of
  - GETLINE
  - PROCESSLINE

Procedure DEFINE

Make appropriate entries in DEFTAB and NAMTAB

Algorithm
Figure 4.5, pp. 184

```
begin {macro processor}
    EXPANDINF := FALSE
    while OPCODE ≠ 'END' do
        begin
            GETLINE
            PROCESSLINE
        end {while}
end {macro processor}
```

```
Procedure PROCESSLINE
    begin
      search MAMTAB for OPCODE
      if found then
            EXPAND
      else if OPCODE = 'MACRO' then
            DEFINE
      else write source line to expanded file
    end {PRCOESSOR}
```

Algorithm
Figure 4.5, pp. 185

```
Procedure DEFINE
    begin
        enter macro name into NAMTAB
        enter macro prototype into DEFTAB
        LEVEL   :- 1
        while LEVEL > do
            begin
                GETLINE
                if this is not a comment line then
                  begin
                    substitute positional notation for parameters
                    enter line into DEFTAB
                    if OPCODE = ‘MACRO’ then
                        LEVEL := LEVEL +1
                    else if OPCODE = ‘MEND’ then
                        LEVEL := LEVEL – 1
                  end {if not comment}
            end {while}
         store in NAMTAB pointers to beginning and end of definition
    end {DEFINE}
```

Algorithm
Figure 4.5, pp. 185

```
Procedure EXPAND
    begin
        EXPANDING := TRUE
        get first line of macro definition {prototype} from DEFTAB
        set up arguments from macro invocation in ARGTAB
        while macro invocation to expanded file as a comment
        while not end of macro definition do
            begin
                GETLINE
                PROCESSLINE
            end {while}
        EXPANDING := FALSE
    end {EXPAND}

Procedure GETLINE
    begin
        if EXPANDING then
            begin
                get next line of macro definition from DEFTAB
                substitute arguments from ARGTAB for positional notation
            end {if}
        else
            read next line from input file
    end {GETLINE}
```

Handling nested macro definition

- In DEFINE procedure
  - When a macro definition is being entered into DEFTAB, the normal approach is to continue until an MEND directive is reached.
  - This would not work for nested macro definition because the first MEND encountered in the inner macro will terminate the whole macro definition process.
  - To solve this problem, a counter LEVEL is used to keep track of the level of macro definitions.
    - Increase LEVEL by 1 each time a MACRO directive is read.
    - Decrease LEVEL by 1 each time a MEND directive is read.
    - A MEND terminates the whole macro definition process when LEVEL reaches 0.
    - This process is very much like matching left and right parentheses when scanning an arithmetic expression.

Comparison of Macro Processors Design

- One-pass algorithm
  - Every macro must be defined before it is called
  - One-pass processor can alternate between macro definition and macro expansion
  - Nested macro definitions are allowed but nested calls are not
- Two-pass algorithm
  - Pass1: Recognize macro definitions
  - Pass2: Recognize macro calls
  - Nested macro definitions are not allowed