Experiment No.3
Design & Implementation of Two Pass Macro Processor
Date of Performance: 03-02-2025
Date of Submission: 17-02-2025

Aim: Design & Implementation of Pass 1 of Two Pass Macro Processor.

Objective: To study and implement Pass 1 of two pass Macro Processor for IBM 360 Machine.

Theory:

Macro: A macro is a unit of specification for program generation through expansion. A macro instruction is a notational convenience for the programmer. It allows the programmer to write shorthand version of a program (module programming).

Macro Processor: The macro processor replaces each macro invocation with the corresponding sequence of statements.

You may design a two-pass macro processor.

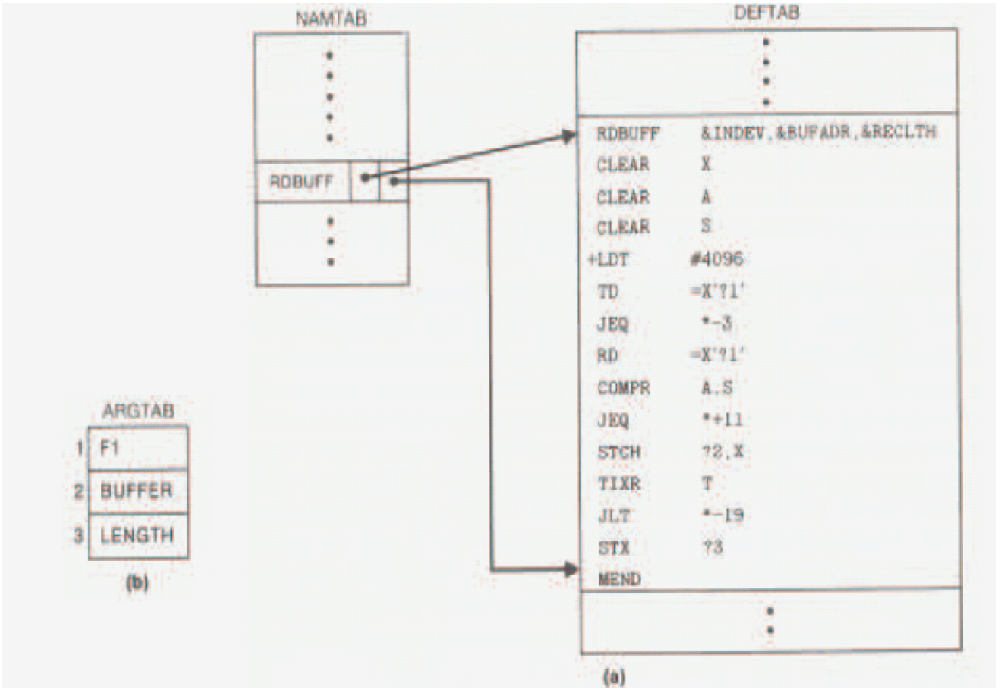
Pass 1: Process all macro definitions.

- Identifies macro definitions and calls in the program
- Determine the formal parameters & their values

Pass 1 Databases of Macro Processor

1. Copy of source program statements
2. Output macro source listings for use by Pass-2
3. MDT to store body of macro definition
4. MDTC used to points the next entry towards in MDT
5. NMT to store names of macro defines in the program
6. MNTC used to points towards next entry into MNT.
7. An array called Parameter List Array (PLA) to manage an index for formal parameters (arguments)

Data structure

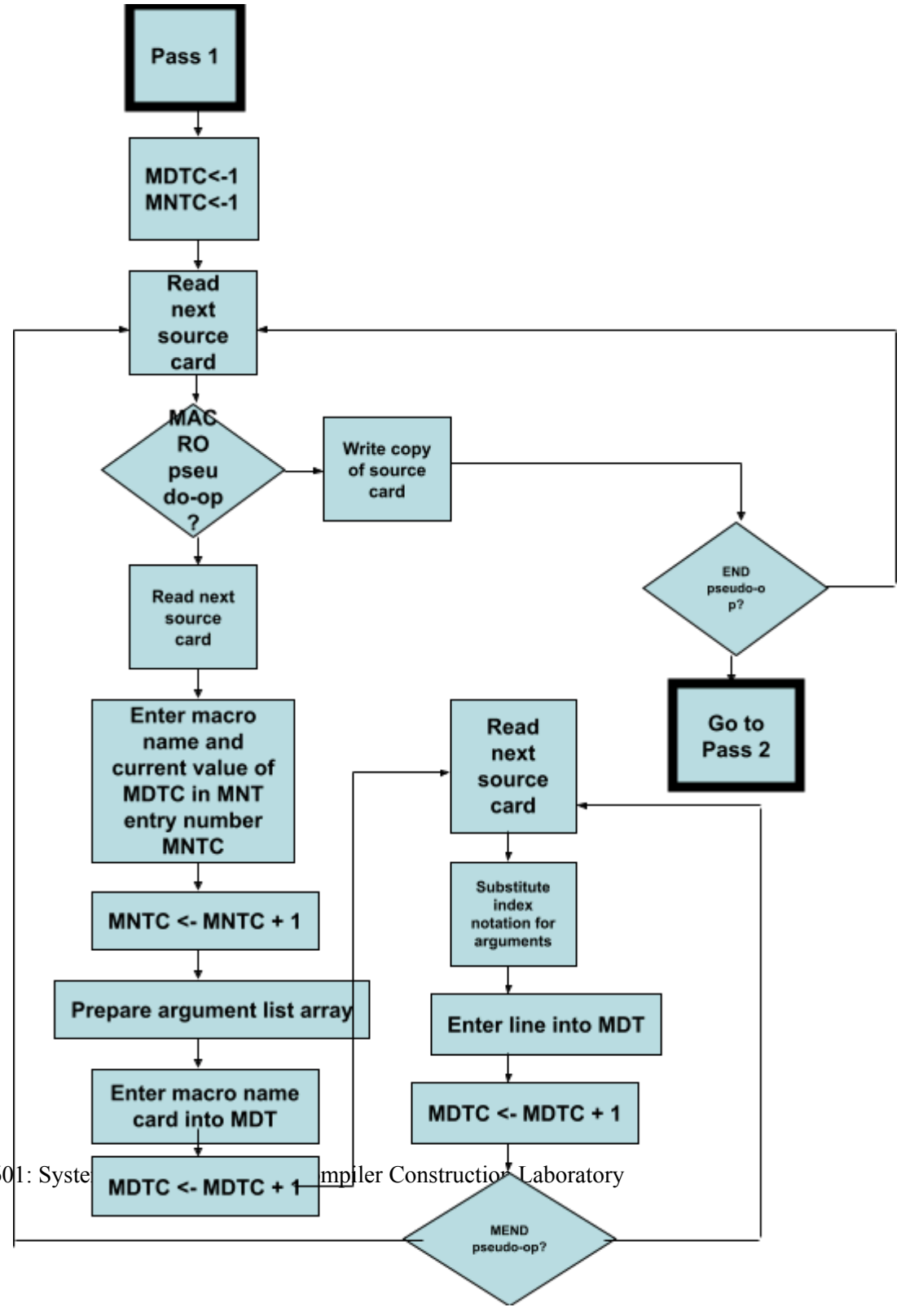


PASS-I of Macro Processor- Processing Macro Definitions

1. Initialize MDTC and MNTC
2. Read the next source statement of the program.
3. If the statement contains MACRO pseudo-op. go to step 6.
4. Output the instruction of the statement.
5. If the statement contains END pseudo-op, go to PASS-II else go to step 2
6. Read the next source statement of the program.
7. Make an entry of the macro name and MTDC in MNT at location MNTC and increment MNTC by 1.
8. Prepare the parameter (arguments) list array.
9. Enter macro name into MDT and increment MTDC by 1.
10. Read the next card and substitute index for the parameters (arguments).
11. Enter line into MDT and increment MDT by 1.

12. If MEND pseudo-op is found, go to step 2 else go to step 10

Pass 1 flowchart:



Code:

```
#include <stdio.h>
#include <string.h>

#define MAX_MACRO 10
#define MAX_MDT 100
#define MAX_ALA 10
#define MAX_LINES 100
#define MAX_LENGTH 100

typedef struct {
    char name[20];
    int index;
} MNTEntry;

typedef struct {
    char opcode[20];
    char operand[50];
} MDTEntry;

typedef struct {
    char param[20];
    char index[5];
} ALAEntry;

MNTEntry mnt[MAX_MACRO];
```

```
MDTEntry mdt[MAX_MDT];
ALAEntry ala[MAX_ALA];

int mntc = 0, mdtc = 0, alac = 0;

void add_to_ala(char *param) {
    sprintf(ala[alac].index, "#%d", alac);
    strcpy(ala[alac].param, param[0] == '&' ? param + 1 : param); // Remove '&' before storing
    alac++;
}

void process_macro_definition(char lines[MAX_LINES][MAX_LENGTH], int start, int end)
{
    char macro_name[20], params[50], param_list[50] = "";
    char *token;
    sscanf(lines[start], "MACRO %s %49[^\n]", macro_name, params);

    // Add to MNT
    strcpy(mnt[mntc].name, macro_name);
    mnt[mntc].index = mdtc;
    mntc++;

    // Process ALA
    alac = 0;
    token = strtok(params, ", ");
    while (token) {
        add_to_ala(token);
```

```
        strcat(param_list, token);
        token = strtok(NULL, ", ");
        if (token) strcat(param_list, ", ");
    }

    // Store macro header in MDT with all params
    strcpy(mdt[mdtc].opcode, macro_name);
    strcpy(mdt[mdtc].operand, param_list);
    mdtc++;

    // Process macro body
    for (int i = start + 1; i < end; i++) {
        char opcode[20], operand[50];
        sscanf(lines[i], "%s %[^\n]", opcode, operand);

        // Replace parameters with indexes
        char final_operand[50] = "";
        token = strtok(operand, ", ");
        while (token) {
            int found = 0;
            for (int j = 0; j < alac; j++) {
                if (strcmp(token, ala[j].param) == 0 || (token[0] == '&' && strcmp(token + 1,
ala[j].param) == 0)) {
                    strcat(final_operand, ala[j].index);
                    found = 1;
                    break;
                }
```

```
            }
            if (!found) strcat(final_operand, token);
            token = strtok(NULL, ", ");
            if (token) strcat(final_operand, ", ");
        }

        // Store in MDT
        strcpy(mdt[mdtc].opcode, opcode);
        strcpy(mdt[mdtc].operand, final_operand);
        mdtc++;
    }

    // Store MEND in MDT
    strcpy(mdt[mdtc].opcode, "MEND");
    strcpy(mdt[mdtc].operand, "");
    mdtc++;
}

void print_tables() {
    printf("\nMacro Name Table (MNT):\n");
    printf("--------------------------------\n");
    printf("Index   Macro Name   MDT Index\n");
    printf("--------------------------------\n");
    for (int i = 0; i < mntc; i++) {
        printf("%d       %-10s   %d\n", i, mnt[i].name, mnt[i].index);
    }
```

```
    printf("\n\nArgument List Array (ALA):\n");
    printf("---------------------------\n");
    printf("Index   Parameter\n");
    printf("---------------------------\n");
    for (int i = 0; i < alac; i++) {
        printf("%-7s  %s\n", ala[i].index, ala[i].param);
    }

    printf("\n\nMacro Definition Table (MDT):\n");
    printf("----------------------------------------\n");
    printf("Index   Opcode   Operands\n");
    printf("----------------------------------------\n");
    for (int i = 0; i < mdtc; i++) {
        printf("%-7d %-8s %s\n", i, mdt[i].opcode, mdt[i].operand);
    }
}

void print_intermediate_code(char lines[MAX_LINES][MAX_LENGTH], int total_lines, int
macro_end) {
    printf("\n\nIntermediate Code (without macro definition):\n");
    printf("-----------------------------------------------\n");
    for (int i = macro_end + 1; i < total_lines; i++) {
        printf("%s\n", lines[i]);
    }
}
```

```
int main() {
    FILE *file = fopen("input.asm", "r");
    if (!file) {
        printf("Error opening file.\n");
        return 1;
    }
    char lines[MAX_LINES][MAX_LENGTH];
    int line_count = 0, macro_end = -1;

    while (fgets(lines[line_count], MAX_LENGTH, file) && line_count < MAX_LINES) {
        lines[line_count][strcspn(lines[line_count], "\n")] = 0;
        if (strstr(lines[line_count], "MEND")) {
            macro_end = line_count;
        }
        line_count++;
    }
    fclose(file);
    if (macro_end != -1) {
        process_macro_definition(lines, 0, macro_end);
    }
    print_tables();
    print_intermediate_code(lines, line_count, macro_end);
    return 0;
}
```

Output:

```
Macro Name Table (MNT):
--------------------------------
Index   Macro Name   MDT Index
--------------------------------
0       INCR         0


Argument List Array (ALA):
----------------------------
Index   Parameter
----------------------------
#0       X
#1       Y
#2       Z


Macro Definition Table (MDT):
--------------------------------------
Index   Opcode   Operands
--------------------------------------
0       INCR     &X, &Y, &Z
1       MOVER    #2, #0
2       MOVER    #2, #1
3       MOVER    #2, #0
4       MEND
```

```
Intermediate Code (without macro definition):
-----------------------------------------------
START 100
READ N1
READ N2
INCR N1,N2,AREG
STOP
```

Conclusion:

→ 1. Working of Program by Providing Input Source Program :

Pass 1 of the Two Pass Macro Processor successfully processes the input source program by identifying and storing macro definitions. When a macro definition such as INCR is encountered in the source code, it is recorded in structured tables. The processor builds the Macro Name Table (MNT), Macro Definition Table (MDT), and Argument List Array (ALA) efficiently. This phase ensures that all macro-related information is stored without generating final code. The source program is parsed line by line, and macro headers and body content are segregated, preparing essential data for expansion during Pass 2. This validates the program's correct functionality.

→2. Databases Generated During Processing of Macro Definition :

During the processing of macro definitions in Pass 1, key databases such as the Macro Name Table (MNT), Argument List Array (ALA), and Macro Definition Table (MDT) are generated. The MNT holds macro names with corresponding MDT indices, allowing quick lookup during macro expansion. The ALA maps formal parameters to positional indices, helping in parameter replacement. The MDT stores the entire macro body with placeholders for parameters. These structured tables facilitate efficient macro expansion during Pass 2 by providing all necessary references. The well-organized database generation confirms the correctness and robustness of the macro processor's first pass design.