

Practical No: 02

Design suitable data structures and implement Pass-I and Pass-II of a two-pass macro-processor. The output of Pass-I (MNT, MDT and intermediate code file without any macro definitions) should be input for Pass-II.

Source Program :

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

#define MAX_MNT 10
#define MAX_MDT 100

// Define the structures first
struct MNT {
    char name[20];
    int mdt_index;
};

struct MDT {
    char line[100];
};

// Then declare the arrays
struct MNT mnt[MAX_MNT];
struct MDT mdt[MAX_MDT];

int mntc = 0, mdtc = 0;

// Function to check if line is macro call
int isMacroCall(char *line) {
    char name[20];
    sscanf(line, "%s", name);
    for (int i = 0; i < mntc; i++) {
        if (strcmp(mnt[i].name, name) == 0)
            return i; // Return index in MNT
    }
    return -1;
}
```

```

// ---- PASS-I ----
void pass1() {
    FILE *fin = fopen("MACIN.txt", "r");
    FILE *fint = fopen("intermediate.txt", "w");
    char line[100], macro_name[20];

    int in_macro = 0;

    while (fgets(line, sizeof(line), fin)) {
        char *tok = strtok(line, "\n");

        if (strcmp(tok, "MACRO") == 0) {
            in_macro = 1;
            continue;
        }

        if (in_macro) {
            if (strstr(tok, "MEND")) {
                strcpy(mdt[mdtc++].line, "MEND");
                in_macro = 0;
                continue;
            }

            if (mntc == 0 || strcmp(mnt[mntc - 1].name, macro_name) != 0) {
                // First line: macro prototype
                sscanf(tok, "%s", macro_name);
                strcpy(mnt[mntc].name, macro_name);
                mnt[mntc++].mdt_index = mdhc;
            }

            strcpy(mdt[mdtc++].line, tok);
        } else {
            fprintf(fint, "%s\n", tok);
        }
    }

    fclose(fin);
    fclose(fint);

    // Write MNT and MDT to files
    FILE *fmnt = fopen("mnt.txt", "w");
    for (int i = 0; i < mntc; i++)
        fprintf(fmnt, "%s %d\n", mnt[i].name, mnt[i].mdt_index);
    fclose(fmnt);

    FILE *fmdt = fopen("mdt.txt", "w");

```

```

for (int i = 0; i < md़tc; i++)
    fprintf(fmdt, "%s\n", mdt[i].line);
    fclose(fmdt);
}

// ---- PASS-II ----
void pass2() {
    FILE *fint = fopen("intermediate.txt", "r");
    FILE *fout = fopen("output.txt", "w");
    char line[100], macro_call[20], args[50], *actual[10], temp[100];
    char *formal[10];
    int mnt_index, i;

    while (fgets(line, sizeof(line), fint)) {
        strcpy(temp, line);
        mnt_index = isMacroCall(temp);

        if (mnt_index != -1) {
            // Macro call detected
            sscanf(line, "%s %s", macro_call, args);

            int arg_count = 0;
            char *tok = strtok(args, ",");
            while (tok) {
                actual[arg_count++] = strdup(tok);
                tok = strtok(NULL, ",");
            }

            int start = mnt[mnt_index].mdt_index;
            strcpy(temp, mdt[start].line);
            int i = 1, fcount = 0;

            // Extract formal arguments from mdt
            tok = strtok(temp, " ");
            while ((tok = strtok(NULL, ", ")) != NULL) {
                formal[fcount++] = strdup(tok);
            }

            for (int j = start + 1; strcmp(mdt[j].line, "MEND") != 0; j++)
                // Loop through the macro body from MDT until you hit MEND
                // REPLACE EACH ARGUMENT WITH ACTUAL
        }
    }
}

```

```

{
    strcpy(temp, mdt[j].line);
    for (i = 0; i < fcount; i++) {
        char *pos = strstr(temp, formal[i]);
        if (pos) {
            char expanded[100] = "", before[100] = "";
            strncpy(before, temp, pos - temp);
            before[pos - temp] = '\0';
            sprintf(expanded, "%s%s%s", before, actual[i], pos + strlen(formal[i]));
            strcpy(temp, expanded);
        }
    }
    fprintf(fout, "%s\n", temp);
}
} else {
    fprintf(fout, "%s", line);
}
}

fclose(fint);
fclose(fout);
}

int main() {
    pass1();
    pass2();
    printf("Macro expansion complete. See output.txt\n");
    return 0;
}

```

Source Program with Macro :

MACRO
INCR &ARG1,&ARG2
LDA &ARG1
ADD &ARG2
STA &ARG1
MEND
START
INCR A,B
END

Macroprocessor Output:

PASS-I OUTPUT :

1. Macro Name Table file :

INCR 0

2. MacroDEFinition Table File:

INCR &ARG1,&ARG2
LDA &ARG1
ADD &ARG2
STA &ARG1
MEND

3. Intermediate File :

START
INCR A,B
END

PASS-II Output

1. Output File :

START
LDA A
ADD B
STA A
END

- `fint` : Input file pointer for intermediate file
- `fout` : Output file pointer for final code
- `line` : Stores one line read from the intermediate file
- `macro_call` : Macro name
- `args` : Actual arguments in a macro call
- `actual[10]` : Array of pointers to store actual parameters
- `formal[10]` : Array of pointers to store formal parameters