**Fr. Conceicao Rodrigues College of Engineering**
**Department of Computer Engineering**

**Academic Term : Jan-May 2024 - 25**

**Class : T.E. (Computer - A)**
**Subject Name : System Programming and Compiler Construction**
**Subject Code : (CPC601)**

| | |
|---|---|
| **Practical No:** | 8 |
| **Title:** | program to implement 2-pass macro |
| **Date of Performance:** | 8/04/2025 |
| **Date of Submission:** | 25/04/2025 |
| **Roll No:** | 9913 |
| **Name of the Student:** | Mark Lopes |

**Evaluation:**

| Sr. No | Rubric | Grade |
|---|---|---|
| 1 | Time Line (2) | |
| 2 | Output(3) | |
| 3 | Code optimization (2) | |
| 4 | Postlab (3) | |

**Signature of the Teacher :**

**Experiment No 8**

**Aim:** Write a program to implement two pass Macro Processor.

**Learning Objective:** To understand how the pre-processor replaces all the macros in the program by its real definition prior to the compilation process of the program.

**Algorithm:**

Pass1:

1. Set the MDTC (Macro Definition Table Counter) to 1.

2. Set MNTC (Macro Name Table counter) to 1.

3. Read next statement from source program.

4. If this source statement is pseudo-opcode MACRO (start of macro definition)

5. Read next statement from source program (macro name line)

6. Enter Macro name found in step 5 in name field of MNT ( macro name table)

7. Increment MNTC by 1.

8. Prepare ALA

9. Enter macro name into MDT at index MDTC

10.Increment MDTC by 1.

11. Read source statement from source program

12. Create and substitute index notation for arguments in the source statement if any.

13. Enter this line into MDT

14. Increment MDTC by 1.

15. Check if currently read source statement is pseudo-opcode MEND. If yes then goto step 3 else goto step 11.

16. Write source program statement as it is in the file

17.Check if pseudo-opcode END is encountered. If yes goto step 18 else goto step 19

18. Goto Pass2

19. Go to step 3

20. End of PASS1.

Pass2:

1. Read next statement from source program

2. Search in MNT for match with operation code

3. If macro name found then goto step 4 else goto step 11.

4. Retrieve MDT index from MNT and store it in MDTP.

5. Set up argument list array

6. Increment MDTP by one.

7. Retrieve line pointer by MDTP from MDT

8. Substitute index notation by actual parameter from ALA if any.

9. Check if currently retrieved line is pseuodo-opcode MEND, if yes goto step 1 else goto step 10

10. Write statement formed in step 8 to expanded source file and goto step 6

11. Write source statement directly into expanded source file

12. Check if pseudo-opcode END encountered, if yes goto step 13 else goto step 1

13. End of PASS II

**Implementation Details**

1.Read input file with Macros

2.Display output of Pass1 as the output file, MDT, MNT, and ALA tables.

3.Display output of pass2 as the expanded source file, MDT, MNT and ALA tables.

**Test Cases :**

1. Call macro whose definition is not present
2. Define macro without MEND

**Conclusion**:

**Post Lab Questions:**

1. **What is meant by macro processor**?

   A macro processor is a tool that processes macros in a program. Macros are short names that represent longer code sequences. The macro processor replaces these macros with their full definitions before the program is compiled. This helps reduce repetition and makes the code easier to write and manage. It is commonly used in languages like Assembly and C.

2. **What are the features of macro processor?**

   Macro Definition and Expansion
   A macro processor allows defining code once and using it many times. It expands the macro into full code before compilation.

   Parameter Substitution
   Macros can take arguments, making them dynamic. The processor replaces these parameters with actual values during expansion.

   Code Reusability
   Common code can be reused through macros, reducing repetition. This saves time and keeps the program concise.

Improves Readability

Using macros with clear names makes the code easier to read. It hides complex or lengthy code behind simple identifiers.

Pre-processing Capability

Macro processing is done before the actual compilation. It prepares the code by expanding all macros beforehand.

CODE:-

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

#define MAX_LINE_LENGTH 100
#define MAX_LINES 50
#define MAX_ARGS 10

typedef struct {
    int macro_no;
    char macro_name[50];
    int definition_index;
} MNTEntry;

typedef struct {
    int arg_index;
    char arg_name[50];
    char macro_name[50];
```

```
} ALAEntry;


void definition_table(char definition[MAX_LINES][MAX_LINE_LENGTH],
const char *filename, int *num_lines)

{

    FILE *file = fopen(filename,"r");

    if (!file) {

        printf("Error opening file.\n");

        return;

    }


    char line[MAX_LINE_LENGTH];

    int index = 0;


    printf("Index\t| Definition\n");

    while (fgets(line, MAX_LINE_LENGTH, file) != NULL && index <
MAX_LINES) {

        line[strcspn(line, "\n")] = '\0';

        strcpy(definition[index], line);

        if (index > 0)

            printf("%d\t| %s\n", index + 1, definition[index]);

        index++;

    }


    *num_lines = index;

    fclose(file);
```

```
}


void parse_macro(const char *line, char *macro_name, char
args[MAX_ARGS][50], int *arg_count) {
    *arg_count = 0;

    char temp[MAX_LINE_LENGTH];
    strcpy(temp, line);

    char *token = strtok(temp, " ,");  // Tokenize using space or comma

    // First token is label (optional), second is macro name
    if (token != NULL) {
        token = strtok(NULL, " ,");  // Get macro name (second token)
        if (token != NULL) {
            strcpy(macro_name, token);  // Macro name (INCR1)

            // Get arguments
            token = strtok(NULL, " ,");
            while (token != NULL && *arg_count < MAX_ARGS) {
                strcpy(args[(*arg_count)++], token);
                token = strtok(NULL, " ,");
            }
        }
    }
```

```
}



void build_mnt_ala(char definition[MAX_LINES][MAX_LINE_LENGTH], int
num_lines)

{

    MNTEntry mnt[MAX_LINES];

    ALAEntry ala[MAX_LINES];

    int mnt_count = 0, ala_count = 0;


    for (int i = 0; i < num_lines; i++) {

        if (strcmp(definition[i], "Macro") == 0 && i + 1 < num_lines) {

            char macro_line[MAX_LINE_LENGTH];

            strcpy(macro_line, definition[i + 1]);


            char macro_name[50];

            char args[MAX_ARGS][50];

            int arg_count = 0;


            parse_macro(macro_line, macro_name, args, &arg_count);


            // Add to MNT

            mnt[mnt_count].macro_no = mnt_count + 1;

            strcpy(mnt[mnt_count].macro_name, macro_name);

            mnt[mnt_count].definition_index = i + 2;
```

```
            mnt_count++;


            // Add to ALA
            for (int j = 0; j < arg_count; j++) {
                ala[ala_count].arg_index = ala_count + 1;
                strcpy(ala[ala_count].arg_name, args[j]);
                strcpy(ala[ala_count].macro_name, macro_name);
                ala_count++;
            }
        }
    }


    printf("\n=== MNT Table ===\n");
    printf("No.\tMacro Name\tDefinition Index\n");
    for (int i = 0; i < mnt_count; i++) {
        printf("%d\t%s\t\t%d\n", mnt[i].macro_no, mnt[i].macro_name,
mnt[i].definition_index);
    }


    printf("\n=== ALA Table ===\n");
    printf("Index\tArgument\tMacro Name\n");
    for (int i = 0; i < ala_count; i++) {
        printf("%d\t%s\t\t%s\n", ala[i].arg_index, ala[i].arg_name,
ala[i].macro_name);
    }
}
```

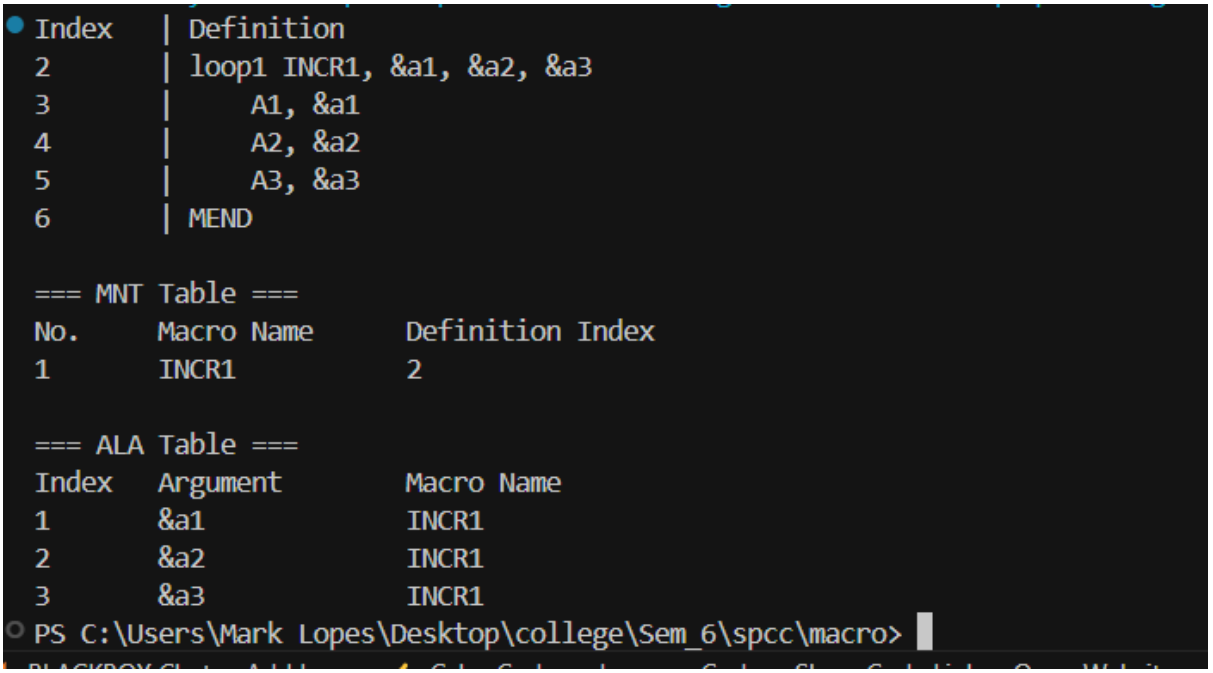
```c
int main()
{
    char definition[MAX_LINES][MAX_LINE_LENGTH];
    int num_lines = 0;

    definition_table(definition, "input.txt", &num_lines);
    build_mnt_ala(definition, num_lines);

    return 0;
}
```

input.txt

```
Macro
loop1 INCR1, &a1, &a2, &a3
   A1, &a1
   A2, &a2
   A3, &a3
MEND
```



```
Index   | Definition
2       | loop1 INCR1, &a1, &a2, &a3
3       |     A1, &a1
4       |     A2, &a2
5       |     A3, &a3
6       | MEND

=== MNT Table ===
No.     Macro Name      Definition Index
1       INCR1           2

=== ALA Table ===
Index   Argument        Macro Name
1       &a1             INCR1
2       &a2             INCR1
3       &a3             INCR1
PS C:\Users\Mark Lopes\Desktop\college\Sem_6\spcc\macro>
```