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 the next statement from the 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 **Code: pass1.c**

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

int main()
{ char opcode[10], mnemonic[10], operand[10], label[10], code[10], address[10], sizeAddress[10],
Motlabel[10]; int locctr = 0, start, length, flag = 0, mntc =
   1, mdtp = 1; FILE *fp1, *fp2, *fp3, *fp4; fp1 =
   fopen("INPUT.txt", "r"); fp2 = fopen("MNT.txt", "w");
   fp3 = fopen("MDT.txt", "w"); fp4 =
   fopen("Copyfile.txt", "w"); fscanf(fp1, "%s %s %s",
   label, opcode, operand);

   while (strcmp(opcode, "END") != 0)
   { if (strcmp(opcode, "MACRO") == 0)
      { while (strcmp(opcode, "MEND") != 0)
         { fscanf(fp1, "%s %s %s", label, opcode, operand); if
           (strcmp(operand, "**") == 0 && strcmp(opcode, "MEND") != 0)
           { fprintf(fp2, "%d %s %d\n", mntc, opcode, mdtp);
              mntc++;
           }
           fprintf(fp3, "%d %s %s %s\n", mdtp, label, opcode, operand);
           mdtp++;
         }
      }
      fscanf(fp1, "%s %s %s", label, opcode, operand); if
      (strcmp(opcode, "MACRO") != 0) fprintf(fp4, "%s
      %s %s\n", label, opcode, operand);
   } return
   0;
}
```

INPUT.txt

```
** MACRO **
** MATH **
** ar 5,3
** sr 5,4
** MEND **
** MACRO **
** MUL **
** mr 5,3
** MEND **
pg1 start 0
** using *,15
** l 1,five
** MATH **
five dc H'5'
** MATH **
** MUL **
** END
```

Output:

MNT.txt

```
MATH
MUL
```

MDT.txt

```
MATH **
ar 5,3
sr 5,4
MEND **
MUL **
mr 5,3
MEND **
```

Copyfile.txt

```
pg1 start 0
** using *,15
** l 1,five
** MATH **
five dc H'5'
** MATH **
** MUL **
** END **
```

pass2.c

```
int main()
{ char opcode[10], mnemonic[10], operand[10], label[10], normal[10];
   FILE *fp1, *fp2, *fp3, *fp4; fp1 = fopen("Copyfile.txt", "r"); fp2 =
   fopen("Expanded Source.txt", "w"); fp3 = fopen("MNT.txt", "r");
   fp4 = fopen("MDT.txt", "r"); fscanf(fp1, "%s %s %s", label, opcode,
   operand);

   while (strcmp(opcode, "END") != 0)
   { fscanf(fp3, "%s", mnemonic); printf("%s",
      mnemonic); if (strcmp(opcode, "START") == 0)
      fprintf(fp2, "%s %s %s\n", label, opcode, operand);
      else
      { if (strcmp(label, "**") == 0 && strcmp(operand, "**") == 0)
         { fscanf(fp4, "%s %s", mnemonic, normal);
            printf("%s\n", mnemonic); while
            (strcmp(opcode, mnemonic) != 0)
            { fscanf(fp4, "%s %s", mnemonic, normal);
            }
            if (strcmp(opcode, mnemonic) == 0)
            {

               while (strcmp(label, "MEND") != 0)
               { fscanf(fp4, "%s %s", label, opcode); if (strcmp(opcode, mnemonic)
                  == 0 || strcmp(label, "MEND") == 0)
                  { continue;
                  }
                  fprintf(fp2, "%s %s\n", label, opcode);
               }
               rewind(fp4);
            }
         }
         else fprintf(fp2, "%s %s %s\n", label, opcode,
            operand);
      }
      rewind(fp3);
      fscanf(fp1, "%s %s %s", label, opcode, operand);
   }
   fprintf(fp2, "%s %s %s\n", label, opcode, operand);
   return 0;
}
```

Output

Expanded Source Code

```
pg1 START 0
** using *,15
** l 1,five
ar 5,3
sr 5,4
five dc H'5'
ar 5,3
sr 5,4
mr 5,3
** END **
```

Conclusion:

Post Lab Questions:

1. **What is meant by macro processor**?
2. **What are the features of macro processor?**