Name : Upmanyu Jha

Rollno : 8875

Comps A Batch B

SPCC - Experiment No 3

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 the currently read source statement is pseudo-opcode MEND. If yes thengoto 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 code3. If macro name found then goto step 4 else goto step 11.

4. Retrieve the MDT index from MNT .

5. Set up argument list array

6. Retrieve MDT

7. Check if currently retrieved line is pseudo-opcode MEND, if yes goto step 1 elsegoto step 8

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

9. Write source statement directly into expanded source file

10. Check if pseudo-opcode END encountered, if yes goto step 11 else goto step 1

11. End of PASS II

Implementation Details

1.Read input file with Macros

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

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

Test Cases :

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

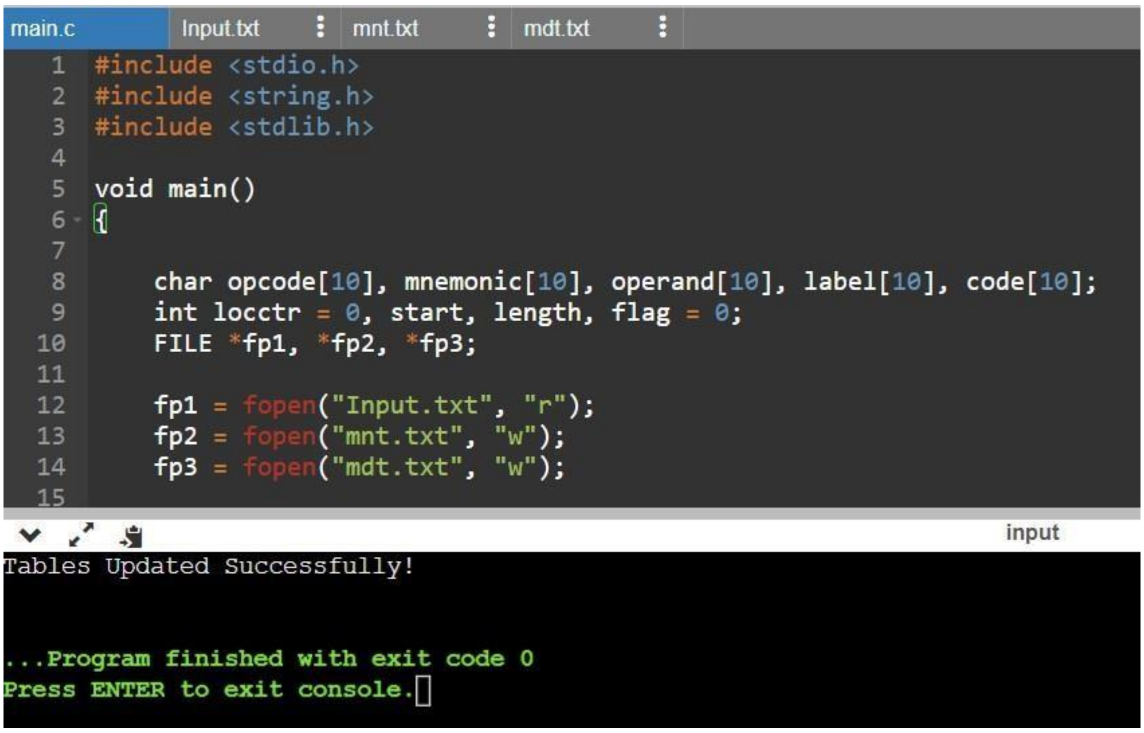
CODE + INPUT + OUTPUT :

pass1.c (code)

```
#include <stdio.h>

#include <string.h>

#include <stdlib.h>


void main()

{


    char opcode[10], mnemonic[10], operand[10], label[10],

    code[10]; int locctr = 0, start, length, flag = 0; FILE *fp1, *fp2,

    *fp3;


    fp1 = fopen("Input.txt", "r");

    fp2 = fopen("mnt.txt", "w"); fp3

    = fopen("mdt.txt", "w");


    fscanf(fp1, "%s%s%s", label, opcode, operand);


    int count = 1; while

    (strcmp(opcode, "Start") != 0)

    { if (strcmp(opcode, "MACRO") == 0)

        { fscanf(fp1, "%s%s%s", label, opcode, operand);

            fprintf(fp2, "%s %d \n", opcode, count); // //
            Storing Macroname along with
MDT pointer in MNT fscanf(fp1, "%s%s%s", label,
            opcode, operand);

            // Storing Macro Definition in MDT
            while (strcmp(opcode, "MEND") != 0)
            { count++; fprintf(fp3, "%s %s\n", opcode,
            operand); fscanf(fp1, "%s%s%s", label,
            opcode, operand); } count++; fprintf(fp3,
            "%s %s\n", opcode, "**");
        }

        fscanf(fp1, "%s%s%s", label, opcode, operand);
    } fprintf(fp2, "%s %s\n", "END",
    "**");

    printf("Tables Updated Successfully!\n");

    fclose(fp1); fclose(fp2);
    fclose(fp3);
}
```

Input.txt

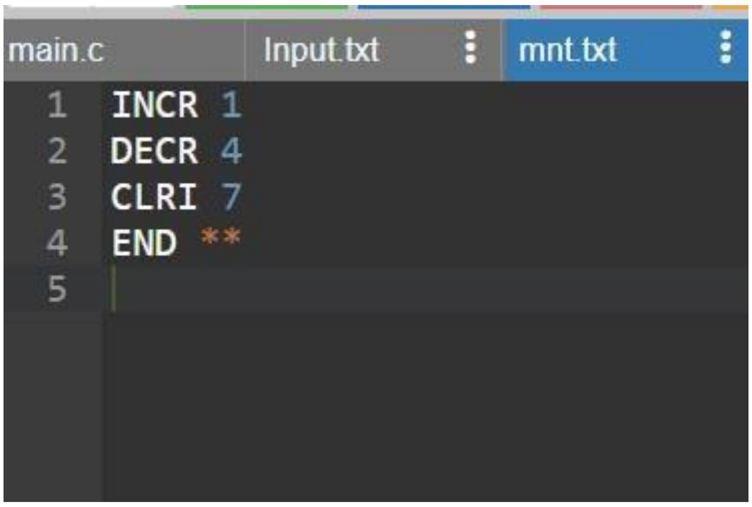
```
**  MACRO **
**  INCR **
**  AR 2,3
**  SR 1,2
**  MEND **
**  MACRO **
**  DECR **
**  AX 8,9
**  ST 5,6
**  MEND **
**  MACRO **
**  CLRI **
**  AK 8,9
**  SK 5,6
**  MEND **
PG1 Start 0
**  Using *,15
**  L 1,FIVE
**  A 1,FOUR
** INCR **
**  ST 1,TEMP
** INCR **
FIVE DC F'5'
FOUR DC F'4'
TEMP DS 1F ** DECR **
FOUR DC F'4'
** CLRI **
TEMP DS 1F
**  END **
```



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

void main()
{

    char opcode[10], mnemonic[10], operand[10], label[10], code[10];
    int locctr = 0, start, length, flag = 0;
    FILE *fp1, *fp2, *fp3;

    fp1 = fopen("Input.txt", "r");
    fp2 = fopen("mnt.txt", "w");
    fp3 = fopen("mdt.txt", "w");

```

```
Tables Updated Successfully!


...Program finished with exit code 0
Press ENTER to exit console.
```

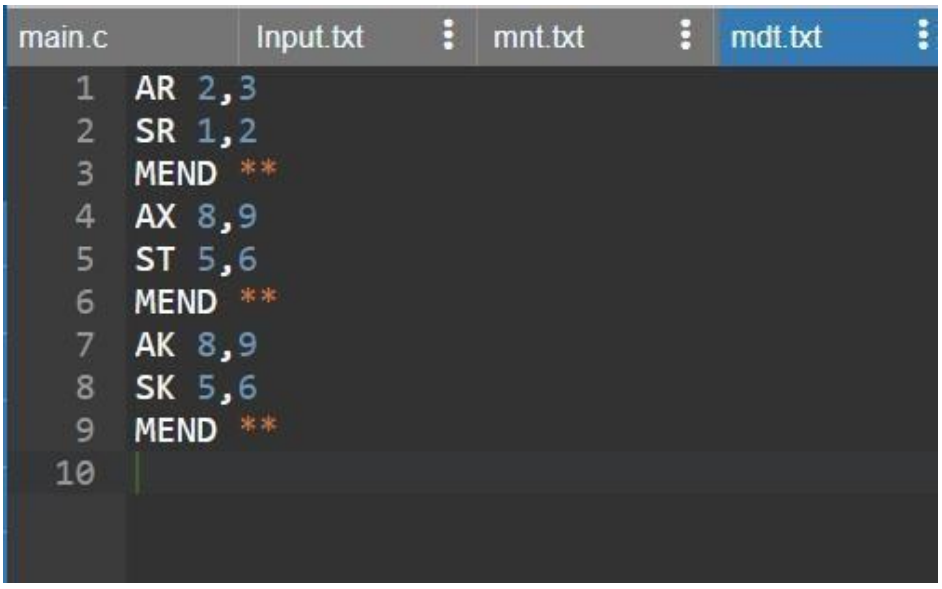
mnt.txt (output file)



```
INCR 1
DECR 4
CLRI 7
END **

```

mdt.txt

(output.txt)



```
AR 2,3
SR 1,2
MEND **
AX 8,9
ST 5,6
MEND **
AK 8,9
SK 5,6
MEND **
```

pass2.c (code)

```
#include <stdio.h>

#include <string.h>

#include <stdlib.h>


int checkIfPresent(char inputOpcode[]) // this function will return position of macroname in mnt if present

{ char macroname[10], mdtpointer[10];

   char label[10], opcode[10], operand[10];

   FILE *mnptr;


   mnptr = fopen("mnt.txt", "r");


   fscanf(mnptr, "%s%s", macroname, mdtpointer);

   int i = 0;

   while (strcmp(macroname, "END") != 0)

   { if (strcmp(inputOpcode, macroname) == 0)
```

```
      { return i; } fscanf(mnptr, "%s%s", macroname,
mdtpointer); i++; } return -1; }


void main()
{


   char opcode[10], mnemonic[10], operand[10], label[10], code[10],
macroname[10], opcodemdt[10], operandmdt[10];
   int mdtline[10]; int locctr = 0, start, length, flag
   = 0; FILE *fp1, *fp2, *fp3, *fp4;


   fp1 = fopen("Input.txt", "r");
   fp2 = fopen("mnt.txt", "r");
   fp3 = fopen("mdt.txt", "r");
   fp4 = fopen("expsrc.txt", "w");
   // Ignoring macro definition
   lines while (strcmp(opcode,
   "Start") != 0)
   { fscanf(fp1, "%s%s%s", label, opcode, operand); }
   fprintf(fp4, "%s %s %s\n", label, opcode,
   operand);


   while (strcmp(opcode, "END") != 0)
   { fscanf(fp1, "%s%s%s", label, opcode, operand);


      int x = checkIfPresent(opcode); // check if opcode present in mnt
      if (x != -1) // Replace Macroname with macro definition
```

```
    { fscanf(fp2, "%s%d", macroname, mdtline);
       // loop x number of times to reach the matched Macroname
       for (int i = 0; i < x; i++)
       { fscanf(fp2, "%s%d", macroname, mdtline);
       }
       // run mdtline number of times to reach the matched Macro Definition
       for (int i = 0; i < *mdtline; i++)
       { fscanf(fp3, "%s%s", opcodemdt, operandmdt);
       }
       while (strcmp(opcodemdt, "MEND") != 0)
       { fprintf(fp4, "%s %s %s\n", "**", opcodemdt, operandmdt); fscanf(fp3,
          "%s%s", opcodemdt, operandmdt);
       }
       rewind(fp3);
       rewind(fp2);
       continue;
    }

    // Print as it is if normal instruction fprintf(fp4,
    "%s %s %s\n", label, opcode, operand);
 }

printf("Expanded Source Code generated successfully!\n");
```

```
    fclose(fp1); fclose(fp2);
    fclose(fp3);
    fclose(fp4);
}
```

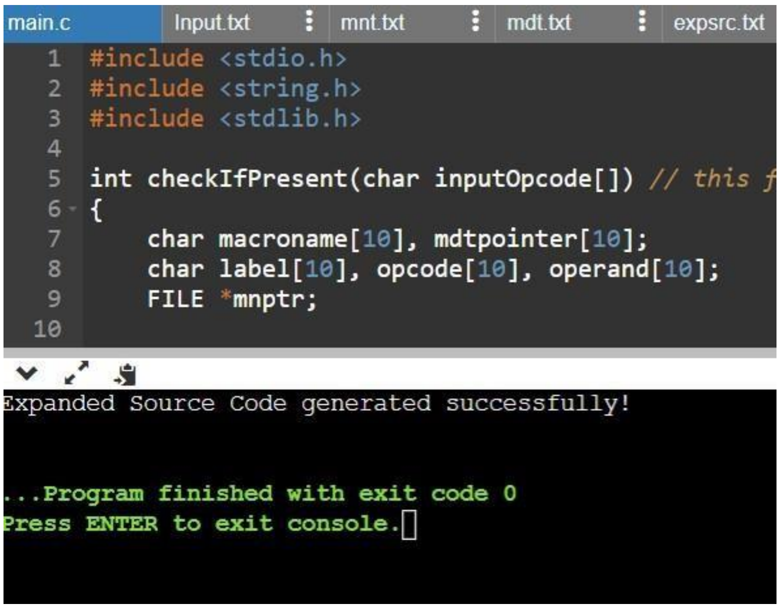
Input.txt (same input file)

mnt.txt (input file) INCR

```
1
DECR 4
CLRI 7 END
**
```

mdt.txt (input.txt)

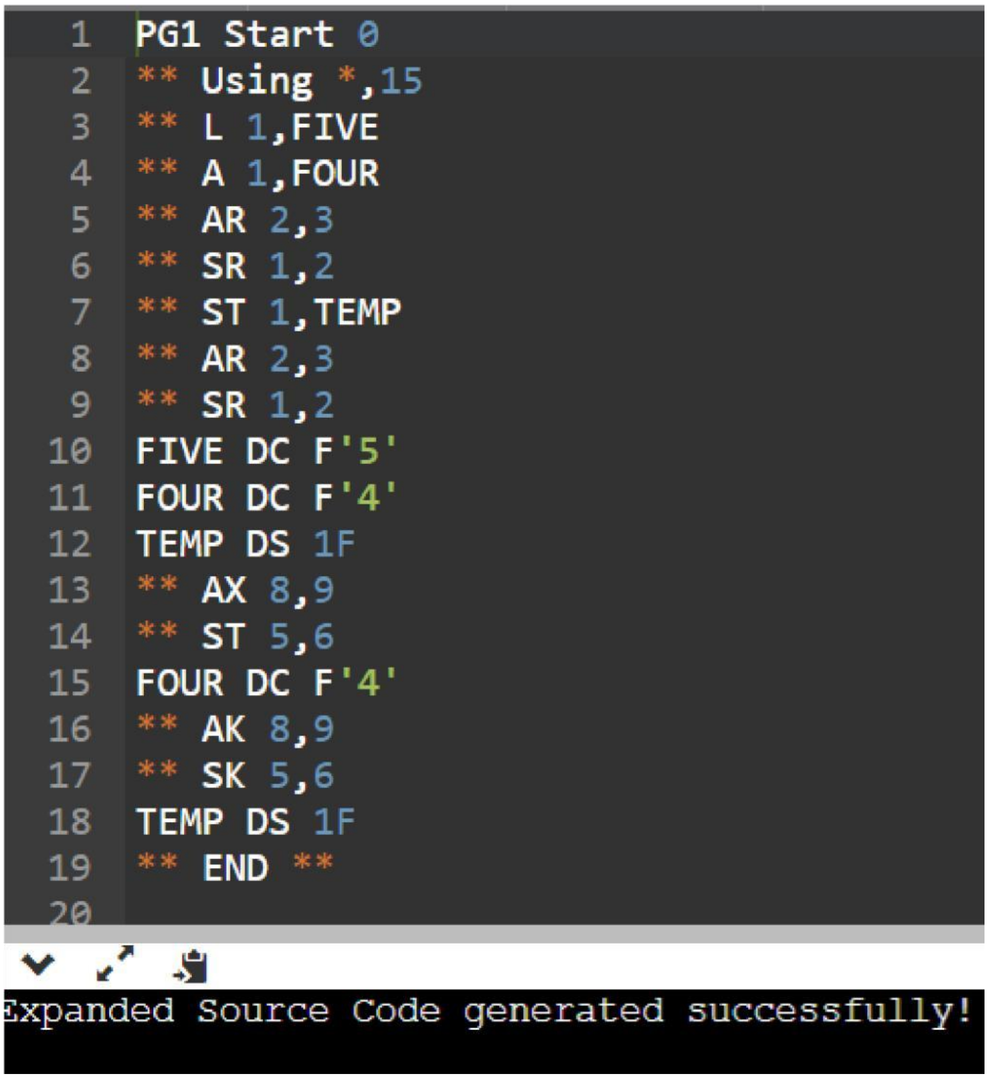
```
AR 2,3
SR 1,2
MEND **
AX 8,9
ST 5,6
MEND **
AK 8,9
SK 5,6
MEND **
```



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

int checkIfPresent(char inputOpcode[]) // this f
{
    char macroname[10], mdtpointer[10];
    char label[10], opcode[10], operand[10];
    FILE *mnptr;

```

```
Expanded Source Code generated successfully!


...Program finished with exit code 0
Press ENTER to exit console.
```

expsrc.txt (output file)



```
PG1 Start 0
** Using *,15
** L 1,FIVE
** A 1,FOUR
** AR 2,3
** SR 1,2
** ST 1,TEMP
** AR 2,3
** SR 1,2
FIVE DC F'5'
FOUR DC F'4'
TEMP DS 1F
** AX 8,9
** ST 5,6
FOUR DC F'4'
** AK 8,9
** SK 5,6
TEMP DS 1F
** END **

```

```
Expanded Source Code generated successfully!
```

Conclusion: Implemented two pass Macro Processor, test cases accomplished.

1. Learnt how to scan marco name and fetch macro definitions → pass1

pass1.txt (code1), input.txt (input), mnt.txt (output1- macro name), mdt.txt (output-macro definitions)

2. Learned to use output from pass1 code to use as inputs for pass2 to trace macro names and replace macro definitions followed by the rest of the input instructions.

pass2.txt (code1), input.txt (input1), mnt.txt (input2- macro name), mdt.txt (input3macro definitions), expsrc.txt (output-expanded source code)

Post Lab Questions:

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

→

a. A Macro instruction is the notational convenience for the programmer. For everyoccurrence of macro the whole macro body or macro block of statements gets expanded in the main source code. Thus Macro instructions make writing code more convenient.

b. Syntax of Macro definition :

```
MACRO      ← start of definition
– – – – –  ← name of Macro
           ← Body of the Macro
.

.

.
```

MEND ← End of the Macro

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

→ Features of a macro processor are:

1. Macro represents a group of commonly used statements in the source programming language.
2. Macro Processor replaces each macro instruction with the corresponding group of source language statements. This is known as the expansion of macros.
3. Using Macro instructions programmers can leave the mechanical details to be handled by the macro processor.
4. Macro Processor designs are not directly related to the computer architecture on which it runs.
5. Macro Processor involves definition, invocation, and expansion.