College Of Engineering Trivandrum

SYSTEM SOFTWARE LAB

Exercise 14: Two Pass Macro Processor

Author:
Shuhaib Ibrahim
Roll No: 60
TVE18CS061

November 28, 2020

Contents

1 Aim

Implement a two pass macro processor

2 Algorithm

```
step 1: Start
step 2: Define a structure namTab with
            - name //name of the macro
            - startLine //starting line of a macro in deftab
            - endLine //ending line of a macro in deftab
        as its members and declare an array variable macroName[] of the struct namTab
step 3: macroCount <- 0 //used as index to macroName array
step 4: defLine <- 0 //to store current line number of dtab2.txt

pass1() //function to do pass1
step 5: f1 <- Open the file input.txt in read mode
step 6: f2 <- Open the file ntab2 in write mode
step 7: f3 <- Open the file dtab2 in write mode
step 8: Read the current line of f1 into variable la, mne, opnd //label, mnemonic, operand
step 9: Repeat step 10 to 12 while mne is not equal to "END"
step 10: paramCount <- 0 // to store the count of parameters in macro definition
step 11: If mne is equal to "MACRO" ,then
        i)Repeat Steps ii to iv while mne is not equal to "MEND"
        ii)If mne is equal to "MACRO", then
            - macroName[macroCount].startLine <- defLine
            - Write la into f2
            - Write la and opnd into f3
            - p <- 0
            - i <- 0
            -Repeat the following while i is less than string length of opnd
                a) If opnd is not equal to ',' , then
                    - parameters[paramCount][p] <- opnd[i] //parameters is an array storing the
                                                            //current macro arguments
                    - p <- p+1
                b) Else, //end of 1 argument
                    - parameters[paramCount][p] <- '\0'
                    - p <- 0
                    - paramCount <- paramCount+1
                c) i <- i+1
            - parameters[paramCount][p] <- '\0'
            - defLine <- defLine+1
        iii) Else,
            - Write mne into f3
            - argPosition <- 0
            - argIndex <- 0
            - k <- 0
            - Repeat the following while k is less than string length of opnd
                a) If opnd[k] is equal to '&', then //argument
                    - Write '&' into f3
                    - k <- k+1
                    - arg[argIndex] <- '&' //arg is a string to store the current argument
                    - argIndex <- argIndex+1
                    - Repeat the following while opnd[k] is not equal to ',' and k is less
                    than strlen(opnd)
                        i)arg[argIndex] <- opnd[k]
                        ii)argIndex <- argindex+1
                        iii) k <- k+1
                    - arg[argIndex] <- '\0'
                    - i <- 0
                    - Repeat the following while i is less than paramCount
                        i) If parameters[i] is equal to arg, then
                            - Write i into f3
                            - break from this loop
                b) Else,
                    - Write opnd[k] into f3
                    - k <- k+1
            - Write "\n" into f3
            - defLine <- defLine+1
        iv) Read the current line of f1 into variable la, mne, opnd
        v) macroName[macroCount].endLine <- defLine
        vi) Write mne innto f3 // MEND
        vii) defLine <- defLine+1
        viii) macroCount <- macroCount+1
step 12: Read the current line of f1 into variable la, mne, opnd
step 13: Close the files f1, f2, and f3

End of pass1

step 14: Define a structure argTab with charater array name as its member
step 15: Declare an array args of the type struct argTab // to store the parameters

pass2()
step 16: macroFound <- 0 // change to 1 during macro expansion
step 17: outMacroDef <- 0 // To check whether current instruction is out of macro definition
step 18: f1 <- Open input.txt in read mode
step 19: f2 <- Open ntab2.txt in r mode
step 20: f3 <- Open dtab2.txt in r mode
step 21: Read the current line of f1 into variable la, mne, opnd
step 22: Repeat steps 23 to 29 while mne is not equal to "END"
step 23: If mne is equal to "MACRO", go to the line after this macro definition
step 24: Read the current line of f1 into variable la, mne, opnd
step 25: If mne is equal to "START", then outMacroDef <- 1
step 26: macroFound <- 0
step 27: i <- 0
step 28: Repeat the following while i is less than macroCount
        i) If macroName[i].name is equal to mne, then
            - Print la, mne and opnd
            - q <- 0
            - argCount <- 0
            - argIndex <- 0
            - Repeat the following while q is less than string length of opnd
                a) If opnd[q] is not equal to ','
                    - args[argCount].name[argIndex] <- opnd[q]
                    - argIndex <- argIndex+1
                    - q <- q+1
                b) Else,
                    - args[argCount].name[argIndex] <- '\0'
                    - argCount <- argCount+1
                    - argIndex <- 0
                    - q <- q+1
            - args[argCount].name[argIndex] <- '\0'
            - start <- macroName[i].startLine
            - end <- macroName[i].endLine
            - line <- 0
            - Repeat the following while line is not equal to start
```

```
                a) Read current line of f3 into variable currLine
                b) line <- line+1

            //macro expansion starts
            - Print  la
            - Read current line of f3 into currLine //To skip first line of macro definition
            - line <- line+1
            - Repeat the following while line is not equal to end
                a) If line is greater than start + 1, print - // in the place of label
                b) Read the current line of f3 into dmne and dopnd
                c) Print dmne
                d) currParamIndex <- 0 //to store the parameter index specified in the instruction
                                        //in deftab
                e)j <- 0
                f) Repeat the following while j is less than string length of dopnd
                    - currParamIndex <- 0
                    - If dopndp[j] is equal to '&', then
                        i) j <- j+1
                        ii)Repeat the following while dopnd[j] is not equal to ',' and j is less
                        than string length of dopnd
                            - currParam[currParamIndex]=dopnd[j]
                            - j <- j+1
                            - currParamIndex <- currParamIndex+1
                        iii) currParam[currParamIndex] <- '\0'
                        iv) icurrParam <- atoi(currParam)
                        v) Print args[icurrParam].name
                    - Else,
                        i) Print dopnd[j]
                        ii) j <- j+1
                g) macroFound <- 1
                h) Break from this loop
        ii) i <- i+1
step 29: If macroFound is equal to 0 and outMacroDef is equal to 1, then
        // current instruction neither contains a macro call nor it is inside a macro definition
        i) Print la, mne, and opnd
step 30: Close the files f1, f2, and f3

main()
step 31: Call the function pass1()
step 32: Display the contents of the file ntab2.txt
step 33: Display the contents of the file dtab2.txt
step 34: Call the function pass2()
step 35: Stop
```

3 Program Code

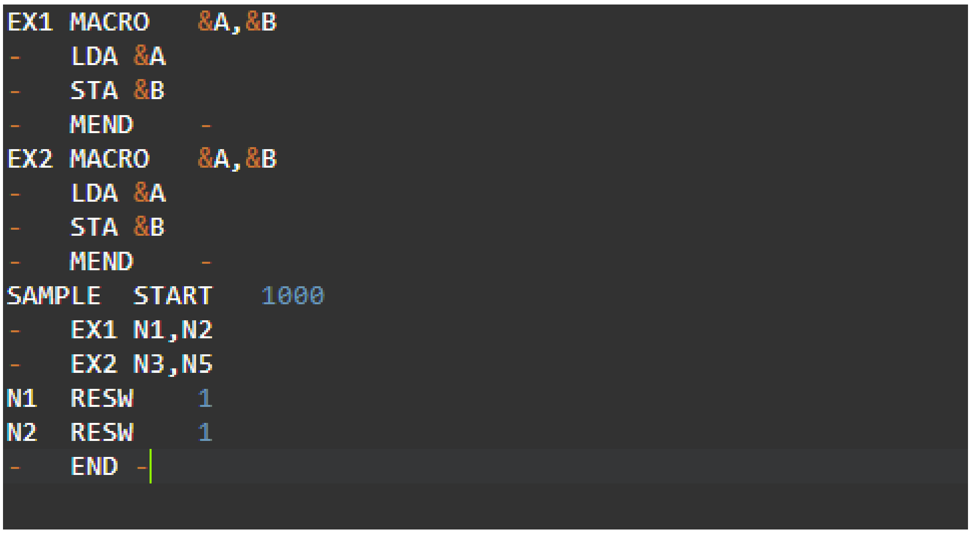
```
#include<stdio.h>

#include<conio.h>

#include<string.h>

#include<stdlib.h>
struct namTab{
    char name[20];
    int startLine;
    int endLine;
```

```
}macroName[20];

int macroCount=0;
int defLine=0; //line number in deftab.txt

void pass1()
{
    FILE * f1, * f2, * f3;
    char mne[20], opnd[20], la[20];
    f1 = fopen("input.txt", "r");
    f2 = fopen("ntab2.txt", "w+");
    f3 = fopen("dtab2.txt", "w+");

    char parameters[10][50];
    int paramCount=0;

    fscanf(f1, "%s%s%s", la, mne, opnd);
    while(strcmp(mne,"END")!=0)
    {
        paramCount=0;
        if (strcmp(mne, "MACRO") == 0)
        {
            while (strcmp(mne, "MEND") != 0)
            {
                if (strcmp(mne, "MACRO") == 0)
                {
                    strcpy(macroName[macroCount].name,la);
                    macroName[macroCount].startLine=defLine;

                    fprintf(f2, "%s\n", la);
                    fprintf(f3, "%s\t%s\n", la, opnd);

                    int p=0;
                    for(int i=0;i<strlen(opnd);i++)
                    {
                        if(opnd[i]!=',')
                        {
                            parameters[paramCount][p]=opnd[i];
                            p++;
                        }
                        else
                        {
                            parameters[paramCount][p]='\0';
                            p=0;
                            paramCount++;
                        }
                    }
                    parameters[paramCount][p]='\0'; //for the last parameter

                    defLine++;
                }
                else
                {
                    // fprintf(f3, "%s\t%s\n", mne, opnd);
                    fprintf(f3, "%s\t", mne);

                    int argPosition=0;
                    char arg[20];
                    int argIndex=0;
                    int k=0;
                    while(k<strlen(opnd))
                    {
                        if(opnd[k]==’&’)
                        {
                            fprintf(f3,"&");
                            k++;

                            arg[argIndex]=’&’;
                            argIndex++;
                            while(opnd[k]!=’,’ && k<strlen(opnd))
                            {
                                arg[argIndex]=opnd[k];
                                argIndex++;
                                k++;
                            }
                            arg[argIndex]=’\0’;

                            for(int i=0; i<=paramCount;i++)
                            {
                                if(strcmp(parameters[i],arg)==0)
                                {
                                    fprintf(f3,"%d",i);
                                    break;
                                }
                            }

                        }
                        else
                        {
                            fprintf(f3,"%c",opnd[k]);
                            k++;
                        }
                    }
                    fprintf(f3,"\n");
                    defLine++;

                }
                fscanf(f1, "%s%s%s", la, mne, opnd);
            }

            macroName[macroCount].endLine=defLine;

            fprintf(f3, "%s\n", mne);
            defLine++;

            macroCount++;
        }
        fscanf(f1, "%s%s%s", la, mne, opnd);

    }
    fclose(f1);
    fclose(f2);
    fclose(f3);

}
```

```
struct argTab
{
    char name[20];
} args[10];

void pass2()
{
    FILE * f1, * f2, * f3, *f4;
    char mne[20], opnd[20], la[20];

    int macroFound=0, outMacroDef=0;

    f1 = fopen("input.txt", "r");
    f2 = fopen("ntab2.txt", "r");
    f3 = fopen("dtab2.txt", "r");
    fscanf(f1, "%s%s%s", la, mne, opnd);
    while(strcmp(mne,"END")!=0)
    {
        if (strcmp(mne, "MACRO") == 0)
        {
            while (strcmp(mne, "MEND") != 0)
            {
                fscanf(f1, "%s%s%s", la, mne, opnd);
            }
        }
        fscanf(f1, "%s%s%s", la, mne, opnd);

        if(strcmp(mne,"START")==0)
            outMacroDef=1;
        macroFound=0;
        for(int i=0;i<macroCount;i++)
        {
            if(strcmp(macroName[i].name,mne)==0)
            {
                printf(".%s\t%s\t%s\n",la,mne,opnd); //comment line
                //storing the arguments in argtab
                int q=0;
                int argCount=0;
                int argIndex=0;
                while(q<strlen(opnd))
                {
                    if(opnd[q]!=',')
                    {
                        args[argCount].name[argIndex]=opnd[q];
                        argIndex++;
                        q++;
                    }
                    else
                    {
                        args[argCount].name[argIndex]='\0';
                        argCount++;
                        argIndex=0;
                        q++;
                    }
                }
                args[argCount].name[argIndex]='\0';

                int start=macroName[i].startLine;
        int end=macroName[i].endLine;
        int line=0;
        char currLine[100];
        char dmne[20], dopnd[20];

        while(line!=start)
        {
            fscanf(f3," %[^\n]", currLine);
            line++;
        }

        //macro expansion
        printf("%s\t",la);

        fscanf(f3," %[^\n]", currLine);
        line++;
        while(line!=end)
        {
            if(line>start+1)
                printf("-\t");
            fscanf(f3,"%s%s",dmne,dopnd);

            printf("%s\t",dmne);

            char currParam[20];
            int currParamIndex=0;
            int icurrParam;
            int j=0;
            while(j<strlen(dopnd))
            {
                currParamIndex=0;
                if(dopnd[j]==’&’)
                {
                    j++;
                    while(dopnd[j]!=’,’ && j<strlen(dopnd))
                    {
                        currParam[currParamIndex]=dopnd[j];
                        j++;
                        currParamIndex++;
                    }
                    currParam[currParamIndex]=’\0’;
                    icurrParam=atoi(currParam);
                    printf("%s",args[icurrParam].name);
                }
                else
                {
                    printf("%c",dopnd[j]);
                    j++;
                }
            }
            printf("\n");
            line++;
        }
        macroFound=1;
        break;
    }
}
if(!macroFound && outMacroDef)
```

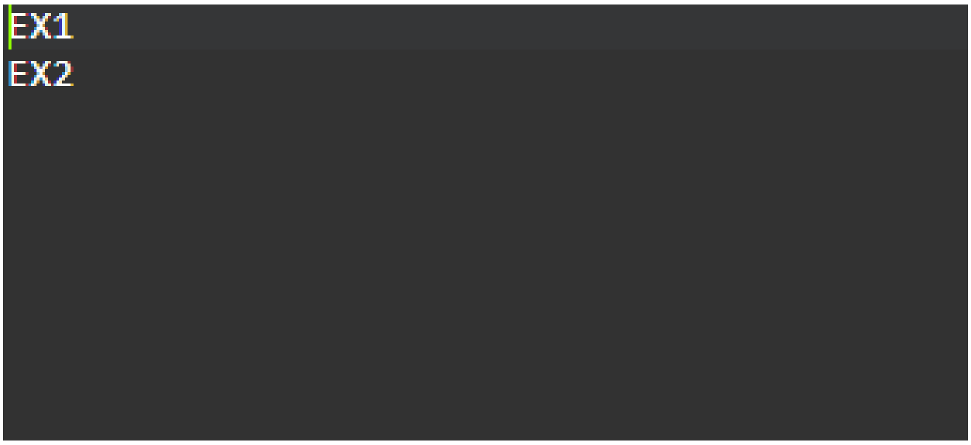
```
        {
            printf("%s\t%s\t%s\n",la,mne,opnd);
        }

    }
    fclose(f1);
    fclose(f2);
    fclose(f3);
}

void main()
{
    pass1();

    char str[100];

    FILE *f1, *f2;
    f1 = fopen("ntab2.txt", "r");
    f2 = fopen("dtab2.txt", "r");

    printf("NAMTAB : \n---------------------\n");
    while(fgets(str,100,f1))
    {
        printf("%s",str);
    }
    printf("---------------------\n\n");

    printf("DEFTAB : \n---------------------\n");
    while(fgets(str,100,f2))
    {
        printf("%s",str);
    }
    printf("---------------------\n\n");

    printf("After macro expansion\n---------------------\n");
    pass2();
}
```

4 Input Files and Output



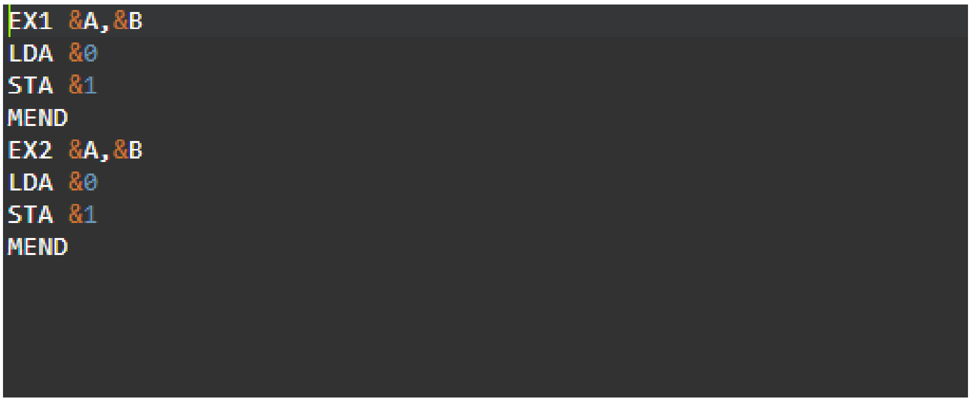
```
EX1 MACRO    &A,&B
-    LDA &A
-    STA &B
-    MEND     -
EX2 MACRO    &A,&B
-    LDA &A
-    STA &B
-    MEND     -
SAMPLE  START    1000
-    EX1 N1,N2
-    EX2 N3,N5
N1   RESW     1
N2   RESW     1
-    END -
```

Figure 1: input.txt - conatins object program



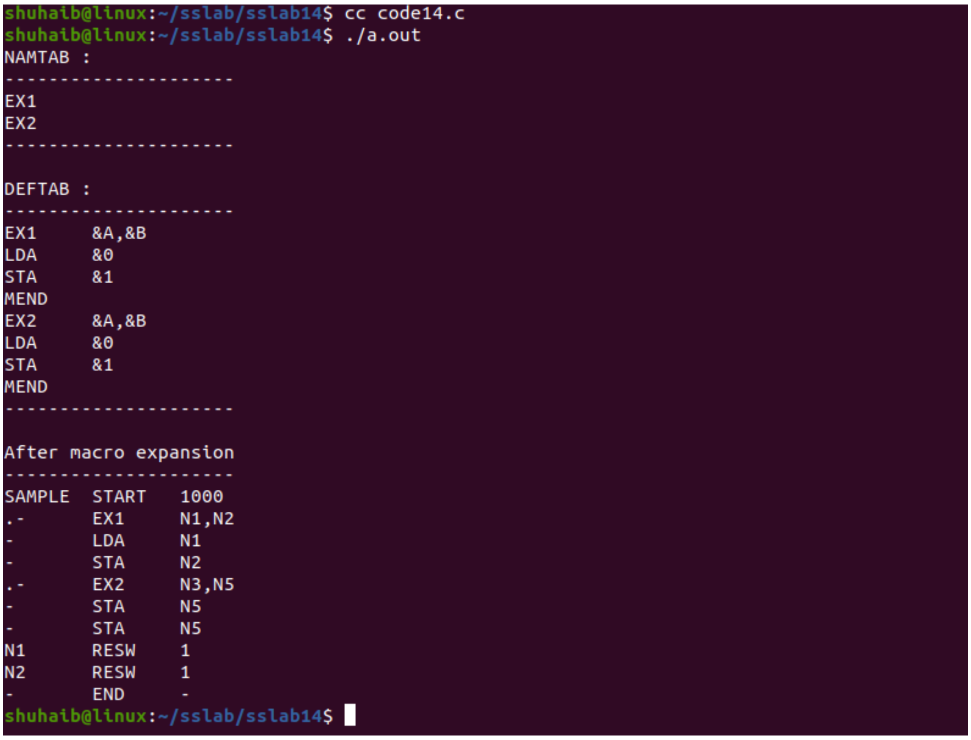
```
EX1
EX2
```

Figure 2: ntab2.txt - ouput from pass1



```
EX1 &A,&B
LDA &0
STA &1
MEND
EX2 &A,&B
LDA &0
STA &1
MEND
```

Figure 3: dtab2.txt - ouput from pass1



```
shuhaib@linux:~/sslab/sslab14$ cc code14.c
shuhaib@linux:~/sslab/sslab14$ ./a.out
NAMTAB :
---------------------
EX1
EX2
---------------------

DEFTAB :
---------------------
EX1     &A,&B
LDA     &0
STA     &1
MEND
EX2     &A,&B
LDA     &0
STA     &1
MEND
---------------------

After macro expansion
---------------------
SAMPLE  START   1000
.-      EX1     N1,N2
-       LDA     N1
-       STA     N2
.-      EX2     N3,N5
-       STA     N5
-       STA     N5
N1      RESW    1
N2      RESW    1
-       END     -
shuhaib@linux:~/sslab/sslab14$ 
```

Figure 4: Output

5 Result

Program to Implement a two pass macro processor was successfully implemented and output was obtained using C programming language