TWO PASS MACRO PROCESSOR
System Software Lab (CS 331)

November 28, 2020

Sheen Xavier A
Roll No : 57
University Roll No : TVE18CS058
Department of Computer Science and Engineering
College of Engineering, Trivandrum

Contents

Implementation of Two Pass Macro Processor

1.1 Problem Statement

To develop a C program to implement a Two Pass Macro Processor.

1.2 Theory

A Macro Processor is used for identifying the macros and performing expansion pertaining to the corresponding macro.

The functions of a macro processor are :

- Recognize the macro definition
- Save macro definition
- Recognize the macro invocation
- Perform macro expansion

A Two Pass Macro processor is used to avoid the issue of forward referencing where a macro maybe sometimes defined in between the source program. In the first pass we identify all the macro definitions within the source program and store information regarding them in NAMTAB and DEFTAB. In the second pass, we identify all macro invocations within the source program and expand the macro with the positional parameters being replaced with provided arguments.

1.3 Algorithm

1.3.1 Pass 1

```
begin
  read first line from source program
  while OPCODE != 'END' do
    begin
      if OPCODE = 'MACRO' then
        begin
          enter macro name into NAMTAB
          enter macro prototype into DEFTAB
          LEVEL := 1
          while LEVEL > 0 do
            begin
              read next line from source program
              if this is not a comment line then
                begin
                  substitute positional notation for parameters
                  enter line into DEFTAB
                  if OPCODE = 'MACRO' then
                    LEVEL := LEVEL + 1
                  else if OPCODE = 'MEND' then
                    LEVEL := LEVEL - 1
                end {if not comment}
            end {while}
          store NAMTAB pointers to beginning and end of definition
        end {if}
      read next line from source program
    end {while}
end
```

1.3.2 Pass 2

```
begin
  read first line from source program
  while OPCODE != 'END' do
    begin
      if OPCODE = 'MACRO' then
        begin
```

```
7           LEVEL := 1
8           while LEVEL > 0 do
9             begin
10              read next line from source program
11              if this is not a comment line then
12                begin
13                  if OPCODE = 'MACRO' then
14                    LEVEL := LEVEL + 1
15                  else if OPCODE = 'MEND' then
16                    LEVEL := LEVEL - 1
17                end {if not comment}
18            end {while}
19        end {if}
20      else
21        begin
22            search NAMTAB for OPCODE
23            if found then
24              begin
25                get first line of macro definition (prototype) from
    DEFTAB
26                set up arguments from macro invocation in ARGTAB
27                write macro invocation to expanded file as a comment
28                while not end of macro definition do
29                  begin
30                    get next line of macro definition from DEFTAB
31                    substitute arguments from ARGTAB for positional
    notation
32                    write modified line to expanded file
33                  end {while}
34              end {if found}
35            else
36              write source line to expanded file
37        end {else}
38      read next line from source program
39    end {while}
40end
```

1.4 Source Code

```
//Name    : twopassmacro.c
//Desc    : Program which performs performs two pass macro processing on
    provided input.
//Input   : The Program which contains the program file is provided as input
    .
//Output : It displays the contents of NAMTAB and DEFTAB after Pass 1 and
    the contents
//          of output file which contains the macro processed code.
//Author : Sheen Xavier A
//Date    : 28 / 11 / 2020

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

//Data structure for NAMTAB
typedef struct{
    char name[100];
    char ARGSTAB[10][100];
    int argsCount;
    int startPosition, endPosition;
    int flag;
}namtype;

//Data structure for DEFTAB
typedef struct{
    char line[100];
}deftype;

//Function to seperate lines into different sections based on seperating
    character
int lineSeperator(char *line, char seperator, char section[10][100])
{
    int i = 0, j = 0, sectionNumber = 0;
    while(line[i] != '\0')
    {
            if(line[i] == seperator)
            {
                section[sectionNumber][j] = '\0';
```

```
36                  j = 0;
37                  sectionNumber++;
38              }
39              else
40              {
41                  section[sectionNumber][j++] = line[i];
42              }
43              i++;
44      }
45      section[sectionNumber][j] = '\0';
46      return sectionNumber + 1;
47  }
48
49  //Function used for extracting the argument in the definition
50  int extractArg(char line[], char arg[])
51  {
52      int i;
53      for(i = 0 ; line[i] != '\0' ; i++)
54      {
55          if(line[i] == '\'' || line[i] == ',' || line[i] == ' ')
56          {
57              arg[i] = '\0';
58              break;
59          }
60          else
61              arg[i] =  line[i];
62      }
63      return i;
64  }
65
66  //Function which generates the positional notation form for macro
67  void convertLineToPN(char line[], char args[10][100], int argsCount, char
        outputLine[100])
68  {
69      int i, j = 0, length = 0, k;
70      char arg[100];
71      for(i = 0; line[i] != '\0'; )
72      {
73          if(line[i] == '&')
74          {
75              outputLine[j++] = '?';
76              length = extractArg(line + i, arg);
```

```
77              for(k = 0 ; k < argsCount ; k++)
78              {
79                  if(!(strcmp(args[k], arg)))
80                      break;
81              }
82              outputLine[j++] = 49 + k;
83              i += length;
84          }
85          else
86              outputLine[j++] = line[i++];
87      }
88      outputLine[j] = '\0';
89  }
90
91  //Function for extracting argument number based on postional notation
92  int extractArgIndex(char line[], char arg[], int * length)
93  {
94      int i;
95      for(i = 0 ; line[i] != '\0' ; i++)
96      {
97          if(line[i] == '\'' || line[i] == ',' || line[i] == ' ')
98          {
99              arg[i] = '\0';
100             break;
101         }
102         else
103             arg[i] =  line[i];
104     }
105     *length = i;
106     return atoi(arg) - 1;
107 }
108
109 //Function for substituting arguments instead of positional parameters
110 void convertPNToAN(char line[], char args[10][100], int argsCount, char
        outputLine[100])
111 {
112     int i, j = 0, length = 0, argPos, k = 0;
113     char arg[100];
114     for(i = 0; line[i] != '\0'; )
115     {
116         if(line[i] == '?')
117         {
```

```
118             i++;
119             argPos = extractArgIndex(line + i, arg, &length);
120             k = 0;
121             while(args[argPos][k] != '\0')
122                 outputLine[j++] = args[argPos][k++];
123             i += length;
124         }
125         else
126             outputLine[j++] = line[i++];
127     }
128     outputLine[j] = '\0';
129 }
130 
131 //Hash function for NAMTAB hash table implementation
132 int hash(char *label)
133 {
134     int i = 0;
135     int hashValue = 0;
136     while(label[i] != '\0')
137     {
138         hashValue += (i + 1) * (label[i] - 65 + 1);
139         i++;
140     }
141     return hashValue;
142 }
143 
144 int main()
145 {
146     FILE *ptr;
147     FILE *ptrW;
148     int i, j, expanding = 0, hashValue, defTabIndex = 0, level, argCount,
    length;
149     char fname[100], line[100], section[10][100], positionalNotation[100],
    ARGTAB[10][100], argumentNotation[100];
150     //Initializing the OPTAB and SYMTAB
151     namtype NAMTAB[2000];
152     deftype DEFTAB[2000];
153 
154     //Initializing the NAMTAB
155     for(i = 0 ; i < 2000 ; i++)
156         NAMTAB[i].flag = 0;
157 
```

```
158     //Reading the name of the file which contains the input code
159     printf("FILENAME : ");
160     scanf(" %[^\n]", fname);
161     ptr = fopen(fname, "r");
162 
163     fscanf(ptr, " %[^\n]", line);
164     lineSeperator(line, ' ', section);
165 
166     //Pass 1 : Defining the Macros and Populating the NAMTAB and DEFTAB
167     while(strcmp("END", section[1]))
168     {
169         if(!(strcmp("MACRO", section[1])))
170         {
171             hashValue = hash(section[0]);
172             strcpy(NAMTAB[hashValue].name, section[0]);
173             NAMTAB[hashValue].startPosition = defTabIndex;
174             NAMTAB[hashValue].flag = 1;
175             sprintf(line, "%s", section[0]);
176             length = strlen(section[0]);
177             sprintf(line + length, " %s", section[2]);
178             strcpy(DEFTAB[defTabIndex++].line, line);
179             NAMTAB[hashValue].argsCount = lineSeperator(section[2], ',',
    NAMTAB[hashValue].ARGSTAB);
180             level = 1;
181             while(level > 0)
182             {
183                 fscanf(ptr, " %[^\n]", line);
184                 lineSeperator(line, ' ', section);
185                 if(line[0] != '.')
186                 {
187                     convertLineToPN(line, NAMTAB[hashValue].ARGSTAB, NAMTAB
    [hashValue].argsCount, positionalNotation);
188                     strcpy(DEFTAB[defTabIndex++].line, positionalNotation +
     2);
189                     if(!(strcmp(section[1], "MACRO")))
190                         level++;
191                     else if(!(strcmp(section[1], "MEND")))
192                         level--;
193                 }
194             }
195             NAMTAB[hashValue].endPosition = defTabIndex - 1;
196         }
```

```
197         fscanf(ptr, " %[^\n]", line);
198         lineSeperator(line, ' ', section);
199     }
200     fclose(ptr);
201 
202     //Pass 2 : Expanding the macro invocations in the program
203     ptr = fopen(fname, "r");
204     ptrW = fopen("output.txt", "w");
205 
206     fscanf(ptr, " %[^\n]", line);
207     lineSeperator(line, ' ', section);
208 
209     while(strcmp("END", section[1]))
210     {
211         if(!(strcmp("MACRO", section[1])))
212         {
213             level = 1;
214             while(level > 0)
215             {
216                 fscanf(ptr, " %[^\n]", line);
217                 lineSeperator(line, ' ', section);
218                 if(!(strcmp(section[1], "MACRO")))
219                     level++;
220                 else if(!(strcmp(section[1], "MEND")))
221                     level--;
222             }
223         }
224         else
225         {
226             hashValue = hash(section[1]);
227             if(NAMTAB[hashValue].flag)
228             {
229                 fprintf(ptrW, ".%s\n", line);
230                 lineSeperator(section[2], ',', ARGTAB);
231                 for(j = NAMTAB[hashValue].startPosition + 1 ; j < NAMTAB[
    hashValue].endPosition ; j++ )
232                 {
233                     convertPNToAN(DEFTAB[j].line, ARGTAB, NAMTAB[hashValue
    ].argsCount, argumentNotation);
234                     lineSeperator(argumentNotation, ' ', section);
235                     fprintf(ptrW, "-\t%s\t%s\n", section[0], section[1]);
236                 }
```

```
237             }
238             else
239                 fprintf(ptrW, "%s\t%s\t%s\n", section[0], section[1],
    section[2]);
240         }
241         fscanf(ptr, " %[^\n]", line);
242         lineSeperator(line, ' ', section);
243     }
244     fprintf(ptrW, "%s\t%s\t%s\n", section[0], section[1], section[2]);
245     fclose(ptrW);
246     fclose(ptr);
247 
248     printf("******************\n");
249     printf("CONTENTS OF NAMTAB\n");
250     printf("******************\n");
251     for(i = 0 ; i < 2000 ; i++)
252     {
253         if(NAMTAB[i].flag)
254         {
255             printf("NAME: %s\n", NAMTAB[i].name);
256             printf("ARGUMENTS: ");
257             for(j = 0 ; j < NAMTAB[i].argsCount - 1 ; j++)
258                 printf("%s, ", NAMTAB[i].ARGSTAB[j] + 1);
259             printf("%s\n", NAMTAB[i].ARGSTAB[NAMTAB[i].argsCount - 1] + 1);
260             printf("MACRO LENGTH: %d\n", NAMTAB[i].endPosition - NAMTAB[i].
    startPosition + 1);
261             printf("******************\n");
262         }
263     }
264     printf("******************\n");
265     printf("CONTENTS OF DEFTAB\n");
266     printf("******************\n");
267     for (i = 0; i < defTabIndex; i++)
268         printf("%s\n", DEFTAB[i].line);
269 
270     ptr = fopen("output.txt", "r");
271     printf("****************************************\n");
272     printf("THE CONTENTS OF MACRO PROCESSED PROGRAM\n");
273     printf("****************************************\n");
274     while(fscanf(ptr, " %[^\n]", line) != EOF)
275         printf("%s\n", line);
276     printf("****************************************\n");
```

```
277     fclose(ptr);
278     return 0;
279 }
```

1.5 Output

```
sheenxavi004@Beta-Station:~/Desktop/System Software Lab$ cat input.txt
EX1 MACRO &A,&B
- LDA &A
- STA &B
- MEND -
SAMPLE START 1000
- EX1 N1,N2
EX2 MACRO &C,&A,&B
- TD X'&C'
- LDA &B
- STA &A
- MEND -
- EX2 F1,DEST,SRC
N1 RESW 1
N2 RESW 1
- END -
sheenxavi004@Beta-Station:~/Desktop/System Software Lab$ cc twopassmacro.c
sheenxavi004@Beta-Station:~/Desktop/System Software Lab$ ./a.out
FILENAME : input.txt
******************
CONTENTS OF NAMTAB
******************
NAME: EX1
ARGUMENTS: A, B
MACRO LENGTH: 4
******************
NAME: EX2
ARGUMENTS: C, A, B
MACRO LENGTH: 5
******************
******************
CONTENTS OF DEFTAB
******************
EX1 &A,&B
LDA ?1
```

```
STA ?2
MEND -
EX2 &C,&A,&B
TD X'?1'
LDA ?3
STA ?2
MEND -
**************************************
THE CONTENTS OF MACRO PROCESSED PROGRAM
**************************************
SAMPLE  START   1000
.- EX1 N1,N2
-       LDA     N1
-       STA     N2
.- EX2 F1,DEST,SRC
-       TD      X'F1'
-       LDA     SRC
-       STA     DEST
N1      RESW    1
N2      RESW    1
-       END     -
**************************************
sheenxavi004@Beta-Station:~/Desktop/System Software Lab$ cat output.txt
SAMPLE  START   1000
.- EX1 N1,N2
-       LDA     N1
-       STA     N2
.- EX2 F1,DEST,SRC
-       TD      X'F1'
-       LDA     SRC
-       STA     DEST
N1      RESW    1
N2      RESW    1
-       END     -
```

1.6 Result

The program was done as per the algorithm and the output was verified. It was also tested against various test cases.