Ahsanullah University of Science & Technology

Department of Computer Science & Engineering

CSE 4130
Formal Languages & Compilers Lab
Session: 02
Assignment: 02

Submitted By:

Name: Syed Sanzam
ID: 16.01.04.042
Lab Group: A2

Date of Submission: **August 1, 2019**

Question: Suppose, we have a C source program scanned and filtered as it was done in Session 1. We now take that modified file as input, and separate the lexemes first. We further recognize and mark the lexemes as different types of tokens like keywords, identifiers, operators, separators, parenthesis, numbers, etc.

Sample Input:

```
char c; int x1, x_2; float y1, y2; x1=5; x_2= 10; y1=2.5+x1*45; y2=100.o5-x_2/3; if(y1<=y2) c='y'; else c='n';
```

Source Code:

```
/*
** AUTHOR: Syed Sanzam
** TOPIC:  Categorize Lexemes and Modify Tokens
** DATE:   July 29, 2019
*/

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
#include <stdbool.h>
#define MAX 100

char errorMessage[MAX][MAX];
int errorCount = 0;

bool isKeyword(char buffer[])
{
    if(strlen(buffer) == 0) return false;
    char keywords[32][10] = {"auto","break","case","char","const","continue","default",
                             "do","double","else","enum","extern","float","for","goto",
                             "if","int","long","register","return","short","signed",
                             "sizeof","static","struct","switch","typedef","union",
                             "unsigned","void","volatile","while"};
    int i;
    bool flag = false;


    for(i = 0; i < 32; ++i)
    {
        if(strcmp(keywords[i], buffer) == 0)
        {
            flag = true;
            break;
        }
    }

    return flag;
}

bool isIdentifier(char *str)
```

```
{
    int len = strlen(str);
    int i = 0;

    while(i < len)
    {
        if(isalpha(str[i]) || str[i] == '_')
        {
            i++;
            while(isdigit(str[i]) || isalpha(str[i]) || str[i] == '_' )
            {
                i++;
            }

            break;
        }

        else
        {
            break;
        }
    }

    if(i == len)
        return true;
    else
        return false;
}

bool isNumber(char *str)
{
    int i = 0;
    while(i < strlen(str))
    {
        if(isdigit(str[i]))
        {
            i++;
        }

        else if(str[i] == '.')
        {
            i++;
            while(isdigit(str[i]))
                i++;
            break;
        }

        else
            break;
    }
    if(i == strlen(str))
        return true;
    else
        return false;

}

bool isSeperator(char* str)
      {
          char ch;
          int len = strlen(str);
```

```
103.            int i = 0;
104.            bool flag = false;
105.
106.            if(str[0] == '\0')
107.                return false;
108.
109.            for(i = 0; i < len; i++)
110.            {
111.                ch = str[i];
112.                if(ch == ';' || ch == ','  || ch == '\'' )
113.                {
114.                    flag = true;
115.                    break;
116.                }
117.            }
118.
119.            return flag;
120.        }
121.
122.        bool isOperator(char *str)
123.        {
124.            char ch = str[0];
125.            if(ch == '+' || ch == '-
    ' || ch == '*' || ch == '/' || ch == '>' || ch == '<' || ch == '!' || ch == '=')
126.                return true;
127.            return false;
128.        }
129.
130.        bool isParenthesis(char* str)
131.        {
132.            char ch;
133.            int len = strlen(str);
134.            int i = 0;
135.            bool flag = false;
136.
137.            if(str[0] == '\0')
138.                return false;
139.
140.            for(i = 0; i < len; i++)
141.            {
142.                ch = str[i];
143.                if(ch == '(' || ch == ')'  || ch == '{' || ch == '}' || ch == '[' || ch
    == ']' )
144.                {
145.                    flag = true;
146.                    break;
147.                }
148.            }
149.
150.            return flag;
151.        }
152.
153.
154.        void printErrorMessage()
155.        {
156.            for(int i = 0; i < errorCount; i++)
157.            {
158.                printf("%s",errorMessage[i]);
159.            }
160.        }
161.
```

```
162.        void print(char str[20])
163.        {
164.            if(str[0] != '\0')
165.            {
166.                if(isKeyword(str))
167.                {
168.                    printf("[kw ");
169.                    printf("%s",str);
170.                    printf("] ");
171.                }
172.                else if(isIdentifier(str))
173.                {
174.                    printf("[id ");
175.                    printf("%s",str);
176.                    printf("] ");
177.                }
178.
179.                else if(isNumber(str))
180.                {
181.                    printf("[num ");
182.                    printf("%s",str);
183.                    printf("] ");
184.                }
185.                else if(isSeperator(str))
186.                {
187.                    printf("[sep ");
188.                    printf("%s",str);
189.                    printf("] ");
190.                }
191.                else if(isParenthesis(str))
192.                {
193.                    printf("[par ");
194.                    printf("%s",str);
195.                    printf("] ");
196.                }
197.                else if(isOperator(str))
198.                {
199.
200.                }
201.                else
202.                {
203.                    printf("[unkn ");
204.                    printf("%s",str);
205.                    printf("] ");
206.                    char errorPrompt[100];
207.                    strcpy(errorPrompt,str);
208.                    strcat(errorPrompt, " Is invalid\n");
209.                    strcpy(errorMessage[errorCount],errorPrompt);
210.                    errorCount++;
211.                }
212.            }
213.
214.        }
215.
216.        int main()
217.        {
218.
219.            FILE *fpRead;
220.            fpRead = fopen("input.txt","r");
221.            //fpRead = fopen("testinput.txt","r");
222.
```

```
223.            char operators[] = "+-*/%=<>";
224.            char ch;
225.            char testC;
226.            char buffer[1000];
227.            char toks[1000];
228.
229.
230.            int test = 0;
231.            int j = 0;
232.            int i = 0;
233.            int k = 0;
234.
235.            while( (ch = fgetc(fpRead)) != EOF)
236.            {
237.                if ( !(ch == ' ' || ch == '+' || ch == '-' || ch == '*' ||
238.                ch == '/' || ch == ',' || ch == ';' || ch == '>' ||
239.                ch == '<' || ch == '=' || ch == '(' || ch == ')' ||
240.                ch == '[' || ch == ']' || ch == '{' || ch == '}' || ch == '\''))
241.                {
242.
243.                    buffer[j++] = ch;
244.                }
245.
246.                else
247.                {
248.
249.                    buffer[j] = '\0';
250.                    j = 0;
251.                    print(buffer);
252.                }
253.
254.                if ( (ch == '+' || ch == '-' || ch == '*' ||
255.                ch == '/' || ch == ',' || ch == ';' || ch == '>' ||
256.                ch == '<' || ch == '=' || ch == '(' || ch == ')' ||
257.                ch == '[' || ch == ']' || ch == '{' || ch == '}' || ch == '\''))
258.                {
259.                    toks[k++] = ch;
260.                    k = 0;
261.                    print(toks);
262.
263.                }
264.
265.                if(ch == '+' || ch == '-
    ' || ch == '=' || ch == '>' || ch == '<' || ch == '*' || ch == '/')
266.                {
267.                    bool alreadyPrinted = false;
268.                    for(i = 0; i < strlen(operators); ++i)
269.                    {
270.                        if(ch == operators[i])
271.                        {
272.                            testC = fgetc(fpRead);
273.                            for(test = 0; test < strlen(operators); test++)
274.                            {
275.                                if(testC == operators[test])
276.                                {
277.                                    printf("[op ");
278.                                    printf("%c%c",ch,testC);
279.                                    printf("] ");
280.                                    alreadyPrinted = true;
281.                                }
282.                            }
```

```
283.
284.                             if(!alreadyPrinted)
285.                               {
286.                                  printf("[op ");
287.                                  printf("%c",ch);
288.                                  printf("] ");
289.                                  ungetc(testC,fpRead);
290.                               }
291.                         }
292.                     }
293.
294.                 }
295.             }
296.
297.             printf("\n\n");
298.             if(errorCount == 0)
299.             {
300.                 printf("No errors found!\n");
301.             }
302.             else
303.             {
304.                 printf("!!Warning!!\n");
305.                 printErrorMessage();
306.                 printf("\n");
307.             }
308.             return 0;
309.         }
```

Output:

```
[kw char] [id c] [sep ;] [kw int] [id x1] [sep ,] [id x_2] [sep ;] [kw float] [id y1] [sep ,] [id y2] [sep ;] [id x1] [op =] [num 5] [sep ;] [id x_2] [op =] [num 10] [sep
;] [id y1] [op =] [num 2.5] [op +] [id x1] [op *] [num 45] [sep ;] [id y2] [op =] [unkn 100.o5] [op -] [id x_2] [op /] [num 3] [sep ;] [kw if] [par (] [id y1] [op <=] [id
y2] [par )] [id c] [op =] [sep '] [id y] [sep '] [sep ;] [kw else] [id c] [op =] [sep '] [id n] [sep '] [sep ;]

!!Warning!!
100.o5 Is invalid

[Finished in 0.5s]
```