

# Department of Computer Science and Engineering

S.G.Shivanirudh , 185001146, Semester VI

1 February 2021

---

## UCS1602 - Compiler Design

---

### Exercise 1: Lexical Analyser using C

#### Objective:

Develop a scanner that will recognize all the above specified tokens. Test your program for all specified tokens. Example input and output specification is given below.

#### Code:

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<string.h>
4 #include<ctype.h>
5
6 int check_keyword(char *token){
7     int res = 1;
```

```

8 FILE *fp;
9 fp = fopen("Keywords.txt", "r");
10 if(fp == NULL){
11     printf("\nRead Error");
12     return 0;
13 }
14 else{
15     char *key = (char*)calloc(100, sizeof(char));
16     char ch = fgetc(fp);
17     while(ch != EOF){
18         if(ch == '\n'){
19             res = strcmp(token, key);
20             strcpy(key, "");
21             if(res == 0)
22                 break;
23         }
24         else{
25             strncat(key, &ch, 1);
26         }
27         ch = fgetc(fp);
28     }
29 }
30 fclose(fp);
31 return !res;
32 }
33
34 int check_operator(char *token){
35     int res = 0;
36     FILE *fp;
37     fp = fopen("ArithmeticOp.txt", "r");
38     if(fp == NULL){
39         printf("\nRead Error");
40         return 0;
41     }
42     else{
43         char *key = (char*)calloc(100, sizeof(char));
44         char ch = fgetc(fp);
45         while(ch != EOF){
46             if(ch == '\n'){
47                 res = strcmp(token, key);
48                 strcpy(key, "");
49                 if(res == 0){
50                     res++;
51                     fclose(fp);
52                     return res;

```

```

53         }
54     }
55     else{
56         strncat(key, &ch, 1);
57     }
58     ch = fgetc(fp);
59 }
60 }
61 fclose(fp);
62 fp = fopen("RelationalOp.txt", "r");
63 if(fp == NULL){
64     printf("\nRead Error");
65     return 0;
66 }
67 else{
68     char *key = (char*)calloc(100, sizeof(char));
69     char ch = fgetc(fp);
70     while(ch != EOF){
71         if(ch == '\n'){
72             res = strcmp(token, key);
73             strcpy(key, "");
74             if(res == 0){
75                 res+=2;
76                 fclose(fp);
77                 return res;
78             }
79         }
80         else{
81             strncat(key, &ch, 1);
82         }
83         ch = fgetc(fp);
84     }
85 }
86 fclose(fp);
87 fp = fopen("LogicalOp.txt", "r");
88 if(fp == NULL){
89     printf("\nRead Error");
90     return 0;
91 }
92 else{
93     char *key = (char*)calloc(100, sizeof(char));
94     char ch = fgetc(fp);
95     while(ch != EOF){
96         if(ch == '\n'){
97             res = strcmp(token, key);

```

```

98         strcpy(key, "");
99         if(res == 0){
100             res+=3;
101             fclose(fp);
102             return res;
103         }
104     }
105     else{
106         strncat(key, &ch, 1);
107     }
108     ch = fgetc(fp);
109 }
110 }
111 fclose(fp);
112 }
113
114 int check_separator(char token){
115     int res = 0;
116     FILE *fp;
117     fp = fopen("Separators.txt", "r");
118     if(fp == NULL){
119         printf("\nRead Error");
120         return 0;
121     }
122     else{
123         char ch = fgetc(fp);
124         while(ch != EOF){
125             if(ch == token){
126                 res = 1;
127                 break;
128             }
129             ch = fgetc(fp);
130         }
131     }
132     fclose(fp);
133     return res;
134 }
135
136 char* lexer(char *content){
137     char *lex = (char*)calloc(100, sizeof(char));
138     char *tok = strtok(content, " ");
139
140     int ctr = 0;
141     char *token_list[100];
142

```

```

143     for(int i = 0; i < 100; i++){
144         token_list[i] = (char*)calloc(100, sizeof(char));
145     }
146
147     while(tok){
148         strcpy(token_list[ctr], tok);
149         ctr++;
150         tok = strtok(NULL, " ");
151     }
152
153     for(int j = 0; j < ctr; j++){
154
155         char *t = (char*)calloc(100, sizeof(char));
156         strcpy(t, token_list[j]);
157
158         if(t[strlen(t) - 1] == '/' && t[strlen(t) - 2] == '*')
159     ){
160         strcat(lex, "MC ");
161         break;
162     }
163
164     if (t[0] == '/') {
165         if (t[1] == '/')
166         {
167             strcat(lex, "SC ");
168             break;
169         }
170         else if (t[1] == '*')
171         {
172             strcat(lex, "MC ");
173             break;
174         }
175     }
176     int kw = check_keyword(t); //Check if keyword
177     int op = check_operator(t); //Check if operator
178
179     if(op == 1){
180         strcat(lex, "ARITHOP ");
181     }
182     else if(op == 2){
183         strcat(lex, "RELOP ");
184     }
185     else if(op == 3){
186         strcat(lex, "LOGICALOP ");
187     }

```

```

187     else if(kw == 1){
188         strcat(lex, "KW ");
189     }
190     else if(strcmp(t, "=") == 0){
191         strcat(lex, "ASSIGN ");
192     }
193     else{
194         char *cp = (char *)calloc(100, sizeof(char));
195         strcpy(cp, t);
196         char *token = strtok(t, "(");
197         int func = check_keyword(token);
198         if(func == 1){
199             if((strcmp(token, "if") == 0) || (strcmp(
200 token, "for") == 0) || (strcmp(token, "while") == 0)){
201                 strcat(lex, "KW SP ");
202                 token = strtok(NULL, "(");
203                 for(int k = 0; token[k]; k++){
204                     if(isalpha(token[k])){
205                         strcat(lex, "ID ");
206                         while(isalpha(token[k]) ||
207 isdigit(token[k]) || token[k] == '_')
208                             k++;
209                         k--;
210                     }
211                     else if(token[k] == '=')
212                         strcat(lex, "ASSIGN ");
213                     else if(check_separator(token[k]))
214                         strcat(lex, "SP ");
215                     else if(isdigit(token[k])){
216                         strcat(lex, "NUMCONSTANT ");
217                         while(isdigit(token[k]))
218                             k++;
219                         k--;
220                     }
221                     else if(token[k] == '\\\'')
222                         strcat(lex, "CHARCONSTANT ");
223                     k++;
224                     while(token[k] != '\\\'')
225                         k++;
226                     else if(token[k] == '\\\"')
227                         strcat(lex, "STRINGCONSTANT ");
228                     k++;
229                     while(token[k] != '\\\"')
230                         k++;

```

```

230     }
231     else{
232         char *c = (char*)calloc(10,
sizeof(char));
233         strncpy(c, &token[k], 1);
234         char next = token[k+1];
235         if(next == '=' || next == '|' ||
next == '&'){
236             strncat(c, &token[++k], 1);
237         }
238         else if(check_operator(&next)>0){
239             k++;
240         }
241         else;
242
243         int check = check_operator(c);
244         if(check == 1)
245             strcat(lex, "ARITHOP ");
246         else if(check == 2)
247             strcat(lex, "RELOP ");
248         else if(check == 3)
249             strcat(lex, "LOGICALOP ");
250         else
251             strcat(lex, "INV ");
252     }
253 }
254 }
255 else{
256     strcat(lex, "FC ");
257     if(strcmp(token, "main")){
258         int flag = 0;
259         while(!flag && token_list[j]){
260             t = token_list[j];
261             for(int k = 0; token[k];k++){
262                 if(token[k] == ';')
263                     flag = 1;
264             }
265             j++;
266         }
267         j--;
268     }
269 }
270 }
271 else{
272     if (strcmp(token, cp) != 0)

```

```

273     {
274         strcat(lex, "FC ");
275     }
276     else{
277         for(int i = 0; token[i]; i++){
278             if(isalpha(token[i])){
279                 strcat(lex, "ID ");
280                 while(isalpha(token[i]) ||
isdigit(token[i]) || token[i] == '_' )
281                     i++;
282                 i--;
283             }
284             else if(token[i] == '='){
285                 strcat(lex, "ASSIGN ");
286             }
287             else if(check_separator(token[i])){
288                 strcat(lex, "SP ");
289             }
290             else if(isdigit(token[i])){
291                 strcat(lex, "NUMCONSTANT ");
292                 while(isdigit(token[i]))
293                     i++;
294                 i--;
295             }
296             else{
297                 char *c = (char*)calloc(10,
sizeof(char));
298                 strncpy(c, &token[i], 1);
299                 char next = token[i+1];
300                 if(next == '=' || next == '|' ||
next == '&'){
301                     strncat(c, &token[++i], 1);
302                 }
303                 else if(check_operator(&next)>0){
304                     i++;
305                 }
306                 else{
307                     int check = check_operator(c);
308                     if(check == 1)
309                         strcat(lex, "ARITHOP ");
310                     else if(check == 2)
311                         strcat(lex, "RELOP ");
312                     else if(check == 3)
313                         strcat(lex, "LOGICALOP ");
314                     else
315                         strcat(lex, "INV ");

```



```

315         }
316     }
317 }
318 }
319 }
320 }
321 return lex;
322
323 }
324
325 int line_count(char *file){
326     FILE *fp;
327     int count = 0;
328     fp = fopen(file, "r");
329
330     if (fp == NULL){
331         return 0;
332     }
333     for(char c = getc(fp); c != EOF; c = getc(fp))
334         if (c == '\n')
335             count = count + 1;
336     fclose(fp);
337     return count;
338 }
339
340 int main(){
341     FILE *fp;
342     char ch;
343     char *filename = (char*)calloc(100, sizeof(char));
344     char *content = (char*)calloc(100, sizeof(char));
345     char *copy = (char*)calloc(100, sizeof(char));
346     char *lex = (char*)calloc(200, sizeof(char));
347
348     /*Single line
349     scanf("%[^\\n]", content);
350     strcpy(copy, content);
351     strcpy(lex, lexer(copy));
352
353     printf("Ip: %s\\n", content);
354     printf("Op: %s\\n", lex);
355     */
356     //File
357     printf("\\nEnter file name:");scanf("%[^\\n]", filename);
358

```

```

359     printf("
\n");
360     printf("FC: Function call\n");
361     printf("KW: Keyword\n");
362     printf("ID:identifier");
363     printf("RELOP: Relational operator\n");
364     printf("LOGICALOP: Logical Operator\n");
365     printf("ARITHOP: Arithmetic Operator\n");
366     printf("SP:Separator\n");
367     printf("
\n");

368
369     fp = fopen(filename, "r");
370     fscanf(fp, " %[^\n]", content);
371     int c = 0;
372     while(c < line_count(filename)){
373         strcpy(copy, content);
374         strcpy(lex, lexer(copy));
375         printf("%s\n", lex);
376         fscanf(fp, " %[^\n]", content);
377         c++;
378     }
379     fclose(fp);
380
381
382 }

```

## Input file:

```
1  /*Multiline
2  comment*/
3  main()
4  {
5      int a=10,b=20;
6      if(a!=b)
7          printf(  a  is  greater  );
8      else
9          printf(  b  is  greater  );
10 }
11 add()
12 {
13     int a = 10;
14 }
15 //Single line comment
```

## Output:

```
Enter file name:file.c
-----
FC: Function call
KW: Keyword
ID:identifierRELOP: Relational operator
LOGICALOP: Logical Operator
ARITHOP: Arithmetic Operator
SP:Separator
-----
MC
MC
FC
SP
KW ID ASSIGN NUMCONSTANT SP ID ASSIGN NUMCONSTANT SP
KW SP ID RELOP ID SP
FC
KW
FC
SP
FC
SP
KW ID ASSIGN NUMCONSTANT SP
SP
```

---