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:

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

int check_keyword(char *token){
    int res = 1;
```

```
    FILE *fp;
    fp = fopen("Keywords.txt", "r");
    if(fp == NULL){
        printf("\nRead Error");
        return 0;
    }
    else{
        char *key = (char*)calloc(100, sizeof(char));
        char ch = fgetc(fp);
        while(ch != EOF){
            if(ch == '\n'){
                res = strcmp(token, key);
                strcpy(key, "");
                if(res == 0)
                    break;
            }
            else{
                strncat(key, &ch, 1);
            }
            ch = fgetc(fp);
        }
    }
    fclose(fp);
    return !res;
}

int check_operator(char *token){
    int res = 0;
    FILE *fp;
    fp = fopen("ArithmeticOp.txt", "r");
    if(fp == NULL){
        printf("\nRead Error");
        return 0;
    }
    else{
        char *key = (char*)calloc(100, sizeof(char));
        char ch = fgetc(fp);
        while(ch != EOF){
            if(ch == '\n'){
                res = strcmp(token, key);
                strcpy(key, "");
                if(res == 0){
                    res++;
                    fclose(fp);
                    return res;
```

```
                }
            }
            else{
                strncat(key, &ch, 1);
            }
            ch = fgetc(fp);
        }
    }
    fclose(fp);
    fp = fopen("RelationalOp.txt", "r");
    if(fp == NULL){
        printf("\nRead Error");
        return 0;
    }
    else{
        char *key = (char*)calloc(100, sizeof(char));
        char ch = fgetc(fp);
        while(ch != EOF){
            if(ch == '\n'){
                res = strcmp(token, key);
                strcpy(key, "");
                if(res == 0){
                    res+=2;
                    fclose(fp);
                    return res;
                }
            }
            else{
                strncat(key, &ch, 1);
            }
            ch = fgetc(fp);
        }
    }
    fclose(fp);
    fp = fopen("LogicalOp.txt", "r");
    if(fp == NULL){
        printf("\nRead Error");
        return 0;
    }
    else{
        char *key = (char*)calloc(100, sizeof(char));
        char ch = fgetc(fp);
        while(ch != EOF){
            if(ch == '\n'){
                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             strcat(lex, "MC ");
160             break;
161         }
162 
163         if (t[0] == '/'){
164             if (t[1] == '/')
165             {
166                 strcat(lex, "SC ");
167                 break;
168             }
169             else if (t[1] == '*')
170             {
171                 strcat(lex, "MC ");
172                 break;
173             }
174         }
175         int kw = check_keyword(t); //Check if keyword
176         int op = check_operator(t);//Check if operator
177 
178         if(op == 1){
179             strcat(lex, "ARITHOP ");
180         }
181         else if(op == 2){
182             strcat(lex, "RELOP ");
183         }
184         else if(op == 3){
185             strcat(lex, "LOGICALOP ");
186         }
```

```
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(
    token, "for") == 0) || (strcmp(token, "while") == 0)){
200                     strcat(lex, "KW SP ");
201                     token = strtok(NULL, "(");
202                     for(int k = 0;token[k];k++){
203                         if(isalpha(token[k])){
204                             strcat(lex, "ID ");
205                             while(isalpha(token[k]) ||
    isdigit(token[k]) || token[k] == '_')
206                                 k++;
207                             k--;
208                         }
209                         else if(token[k] == '=')
210                             strcat(lex, "ASSIGN ");
211                         else if(check_separator(token[k]))
212                             strcat(lex, "SP ");
213                         else if(isdigit(token[k])){
214                             strcat(lex, "NUMCONSTANT ");
215                             while(isdigit(token[k]))
216                                 k++;
217                             k--;
218                         }
219                         else if(token[k] == '\''){
220                             strcat(lex, "CHARCONSTANT ");
221                             k++;
222                             while(token[k] != '\'')
223                                 k++;
224                         }
225                         else if(token[k] == '\"'){
226                             strcat(lex, "STRINGCONSTANT ");
227                             k++;
228                             while(token[k] != '\"')
229                                 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                         else if(check_separator(token[i]))
287                             strcat(lex, "SP ");
288                         else if(isdigit(token[i])){
289                             strcat(lex, "NUMCONSTANT ");
290                             while(isdigit(token[i]))
291                                 i++;
292                             i--;
293                         }
294                         else{
295                             char *c = (char*)calloc(10,
    sizeof(char));
296                             strncpy(c, &token[i], 1);
297                             char next = token[i+1];
298                             if(next == '=' || next == '|' ||
    next == '&'){
299                                 strncat(c, &token[++i], 1);
300                             }
301                             else if(check_operator(&next)>0)
302                                 i++;
303                             else;
304
305                             int check = check_operator(c);
306                             if(check == 1)
307                                 strcat(lex, "ARITHOP ");
308                             else if(check == 2)
309                                 strcat(lex, "RELOP ");
310                             else if(check == 3)
311                                 strcat(lex, "LOGICALOP ");
312                             else
313                                 strcat(lex, "INV ");
314
```

```
                        }
                    }
                }
            }
        }
    }
    return lex;

}

int line_count(char *file){
    FILE *fp;
    int count = 0;
    fp = fopen(file, "r");

    if (fp == NULL){
        return 0;
    }
    for(char c = getc(fp); c != EOF; c = getc(fp))
        if (c == '\n')
            count = count + 1;
    fclose(fp);
    return count;
}

int main(){
    FILE *fp;
    char ch;
    char *filename = (char*)calloc(100, sizeof(char));
    char *content = (char*)calloc(100, sizeof(char));
    char *copy = (char*)calloc(100, sizeof(char));
    char *lex = (char*)calloc(200, sizeof(char));

    /*Single line
    scanf(" %[^\n]", content);
    strcpy(copy, content);
    strcpy(lex, lexer(copy));

    printf("Ip: %s\n", content);
    printf("Op: %s\n", lex);
    */
    //File
    printf("\nEnter file name:");scanf(" %[^\n]", filename);

```

```
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:

```
/*Multiline
comment*/
main()
{
  int a=10,b=20;
  if(a!=b)
     printf(  a is greater );
   else
     printf(  b is greater );
}
add()
{
  int a = 10;
}
//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
```

Learning Outcomes:

- Understood the basic working of a Lexical Analyser.
- Learnt to parse a program for detection and identification of tokens.
- Learnt to match regular expressions.