Department of CSE
SSN College of Engineering

Vishakan Subramanian - 18 5001 196 - Semester VI

22 January 2021

UCS 1602 - Compiler Design

Exercise 1: Lexical Analyser Using C

Aim:

To write a program using C to perform the basic functionalities of a **Lexical Analyser**.

Code:

```c
/*  C Program that performs a basic lexical analysis of a given string  */

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

int isOperator(char ch);
int isDelimiter(char ch);
int isValidIdentifier(char *str);
int isInteger(char *str);
char *subString(char *str, int start, int end);
int printOperator(char ch1, char ch2);
int lexicalParse(char *str);

int main(void){
    int status = 0;
    char str[100];

    printf("\n\t\t\tLexical Analyser Using C\n");
    printf("\n\t\tEnter a string to parse: ");
    scanf("%[^\n]", str);

    status = lexicalParse(str);

    if(status){
        printf("\n\n\t\tThe given expression is lexically valid.\n");
    }

    else{
        printf("\n\n\t\tThe given expression is lexically invalid.\n");
    }

    return 0;
}

int isOperator(char ch){
    //Checks if the character is a valid operator

    if (ch == '+' || ch == '-' || ch == '*' ||
        ch == '/' || ch == '>' || ch == '<' ||
        ch == '=' || ch == '%' || ch == '!' ){
            return 1;
        }

    return 0;
}
```

```
48
49 int isDelimiter(char ch){
50     //Checks if the character is a valid delimiter
51
52     if (ch == ' ' || ch == ';' || ch == '(' || ch == ')'
53         || ch == '{' || ch == '}' || ch == '=' || isOperator(ch) == 1){
54             return 1;
55         }
56
57     return 0;
58 }
59
60 int isValidIdentifier(char *str){
61     //Checks if the character is a valid identifier
62
63     if(isdigit(str[0]) > 0 || isDelimiter(str[0]) == 1){
64         //First character shouldn't be a digit or a special character
65         return 0;
66     }
67
68     return 1;
69 }
70
71 int isInteger(char *str){
72     //Checks if the string is a valid integer
73
74     int i = 0, len = strlen(str);
75
76     if(!len){
77         return 0;
78     }
79
80     for(i = 0; i < len; i++){
81         if(!isdigit(str[i])){
82             return 0;
83         }
84     }
85
86     return 1;
87 }
88
89 char *subString(char *str, int start, int end){
90     //Get a substring from the given string
91     int i = 0;
92     char *sub = (char *)malloc(sizeof(char) * (end - start + 2));
93
94     for(i = start; i <= end; i++){
95         sub[i - start] = str[i];
96     }
97
98     sub[end - start + 1] = '\0';
```

```
99
100     return sub;
101 }
102
103 int printOperator(char ch1, char ch2){
104     //Print the details of the parsed operator
105
106     switch(ch1){
107         case '+':
108             if(ch2 == '='){
109                 printf("\n\t\t'%c%c' is ADD/ASSIGNMENT operator.", ch1,
    ch2);
110             }
111             else if(ch2 == ' '){
112                 printf("\n\t\t'%c' is ADD operator.", ch1);
113             }
114             else{
115                 printf("\n\t\t'%c' is not a valid operator.", ch1);
116                 return 0;
117             }
118             break;
119
120
121         case '-':
122             if(ch2 == '='){
123                 printf("\n\t\t'%c%c' is SUBTRACT/ASSIGNMENT operator.",
    ch1, ch2);
124             }
125             else if(ch2 == ' '){
126                 printf("\n\t\t'%c' is SUBTRACT operator.", ch1);
127             }
128             else{
129                 printf("\n\t\t'%c' is not a valid operator.", ch1);
130                 return 0;
131             }
132             break;
133
134         case '*':
135             if(ch2 == '='){
136                 printf("\n\t\t'%c%c' is PRODUCT/ASSIGNMENT operator.", ch1
    , ch2);
137             }
138             else if(ch2 == ' '){
139                 printf("\n\t\t'%c' is PRODUCT operator.", ch1);
140             }
141             else{
142                 printf("\n\t\t'%c' is not a valid operator.", ch1);
143                 return 0;
144             }
145             break;
146
```

```
147         case '/':
148             if(ch2 == '='){
149                 printf("\n\t\t'%c%c' is DIVISION/ASSIGNMENT operator.",
    ch1, ch2);
150             }
151             else if(ch2 == ' '){
152                 printf("\n\t\t'%c' is DIVISION operator.", ch1);
153             }
154             else{
155                 printf("\n\t\t'%c' is not a valid operator.", ch1);
156                 return 0;
157             }
158             break;
159 
160         case '%':
161             if(ch2 == '='){
162                 printf("\n\t\t'%c%c' is MODULO/ASSIGNMENT operator.", ch1,
     ch2);
163             }
164             else if(ch2 == ' '){
165                 printf("\n\t\t'%c' is MODULO operator.", ch1);
166             }
167             else{
168                 printf("\n\t\t'%c' is not a valid operator.", ch1);
169                 return 0;
170             }
171             break;
172 
173         case '=':
174             if(ch2 == '='){
175                 printf("\n\t\t'%c%c' is EQUALITY operator.", ch1, ch2);
176             }
177             else if(ch2 == ' '){
178                 printf("\n\t\t'%c' is ASSIGNMENT operator", ch1);
179             }
180             else{
181                 printf("\n\t\t'%c' is not a valid operator.", ch1);
182                 return 0;
183             }
184             break;
185 
186         case '>':
187             if(ch2 == '='){
188                 printf("\n\t\t'%c%c' is GREATER THAN/EQUAL TO operator.",
    ch1, ch2);
189             }
190             else if(ch2 == ' '){
191                 printf("\n\t\t'%c' is GREATER THAN operator.", ch1);
192             }
193             else{
194                 printf("\n\t\t'%c%c' is not a valid operator.", ch1, ch2);
```

```
195                 return 0;
196             }
197             break;
198 
199         case '<':
200             if(ch2 == '='){
201                 printf("\n\t\t'%c%c' is LESSER THAN/EQUAL TO operator.",
    ch1, ch2);
202             }
203             else if(ch2 == ' '){
204                 printf("\n\t\t'%c' is LESSER THAN operator.", ch1);
205             }
206             else{
207                 printf("\n\t\t'%c%c' is not a valid operator.", ch1, ch2);
208                 return 0;
209             }
210             break;
211 
212         case '!':
213             printf("\n\t\t'%c' is a NOT operator.", ch1);
214             break;
215 
216         default:
217             printf("\n\t\t'%c' is a not a valid operator.", ch1);
218             return 0;
219     }
220 
221     return 1;
222 }
223 
224 int lexicalParse(char *str){
225     //Parse the given string to check for validity
226     int left = 0, right = 0, len = strlen(str), status = 1;
227 
228     while(right <= len && left <= right){
229         //While we are within the valid bounds of the string, check:
230 
231         if(isDelimiter(str[right]) == 0){
232             //If we do not encounter a delimiter, keep moving forward
233             //"right" points to the next character
234             right++;
235         }
236 
237         if(isDelimiter(str[right]) == 1 && left == right){
238             //If it is a delimiter, and we haven't parsed it yet
239 
240             if(isOperator(str[right]) == 1){
241                 //Check if the delimiter is an operator
242                 if((right + 1) <= len && isOperator(str[right + 1]) == 1){
243                     //Check if the next character is also an operator
244                     status = printOperator(str[right], str[right + 1]);
```

```
245                     right++;
246                 }
247 
248                 else{
249                     //Next character is not an operator
250                     status = printOperator(str[right], ' ');
251                 }
252 
253                 //printf("\n\t\t'%c' is an operator.", str[right]);
254             }
255 
256             right++;
257             left = right;
258         }
259 
260         else if(isDelimiter(str[right]) == 1 && left != right || (right == len && left != right)){
261             //We encountered a delimiter in the "right" position, but left != right, thus a chunk of
262             //unparsed characters exist between left and right
263 
264             //Make a substring of the unparsed characters
265             char *sub = subString(str, left, right - 1);
266 
267             if(isInteger(sub) == 1){
268                 //Check if substring is an integer
269                 printf("\n\t\t'%s' is an integer.", sub);
270             }
271             else if(isValidIdentifier(sub) == 1){
272                 //Check if substring is a valid identifier
273                 printf("\n\t\t'%s' is a valid identifier.", sub);
274             }
275             else if(isValidIdentifier(sub) == 0 && isDelimiter(str[right - 1]) == 0){
276                 //Otherwise, print that it is not a valid identifier
277                 status = 0;
278                 printf("\n\t\t'%s' is not a valid identifier.", sub);
279             }
280 
281             left = right;   //We have parsed the chunk, thus "left" = "right"
282         }
283 
284     }
285 
286     return status;
287 }
```

Output:

```
gcc Lex.c -o l
./l

            Lexical Analyser Using C

        Enter a string to parse: a + b = c

        'a' is a valid identifier.
        '+' is ADD operator.
        'b' is a valid identifier.
        '=' is ASSIGNMENT operator
        'c' is a valid identifier.

        The given expression is lexically valid.

gcc Lex.c -o l
./l

            Lexical Analyser Using C

        Enter a string to parse: a >! b == 2c

        'a' is a valid identifier.
        '>!' is not a valid operator.
        'b' is a valid identifier.
        '==' is EQUALITY operator.
        '2c' is not a valid identifier.

        The given expression is lexically invalid.
```