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);
int isKeyword(char *str);
int isPreprocessorDirective(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;
        }

```

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

```
96          !strcmp(str, "switch") || !strcmp(str, "continue") || !strcmp(str,
     "return") ||
97          !strcmp(str, "case") || !strcmp(str, "default") || !strcmp(str, "
    void") ||
98          !strcmp(str, "int") || !strcmp(str, "char") || !strcmp(str, "bool"
    ) ||
99          !strcmp(str, "struct") || !strcmp(str, "goto") || !strcmp(str, "
    typedef") ||
100         !strcmp(str, "unsigned") || !strcmp(str, "long") || !strcmp(str, "
    short")){
101             return 1;
102         }
103
104     return 0;
105 }
106
107 int isPreprocessorDirective(char *str){
108     //Checks if the string is a valid preprocessor directive
109
110     if(str[0] == '#'){
111         //Basic check, works for header files, macros and const
    declarations
112         return 1;
113     }
114     return 0;
115 }
116
117 char *subString(char *str, int start, int end){
118     //Get a substring from the given string
119     int i = 0;
120     char *sub = (char *)malloc(sizeof(char) * (end - start + 2));
121
122     for(i = start; i <= end; i++){
123         sub[i - start] = str[i];
124     }
125
126     sub[end - start + 1] = '\0';
127
128     return sub;
129 }
130
131 int printOperator(char ch1, char ch2){
132     //Print the details of the parsed operator
133
134     switch(ch1){
135         case '+':
136             if(ch2 == '='){
137                 printf("\n\t\t'%c%c' is ADD/ASSIGNMENT operator.", ch1,
    ch2);
138             }
139             else if(ch2 == ' '){
```

```
140                 printf("\n\t\t'%c' is ADD operator.", ch1);
141             }
142             else{
143                 printf("\n\t\t'%c' is not a valid operator.", ch1);
144                 return 0;
145             }
146             break;
147 
148 
149         case '-':
150             if(ch2 == '='){
151                 printf("\n\t\t'%c%c' is SUBTRACT/ASSIGNMENT operator.",
    ch1, ch2);
152             }
153             else if(ch2 == ' '){
154                 printf("\n\t\t'%c' is SUBTRACT operator.", ch1);
155             }
156             else{
157                 printf("\n\t\t'%c' is not a valid operator.", ch1);
158                 return 0;
159             }
160             break;
161 
162         case '*':
163             if(ch2 == '='){
164                 printf("\n\t\t'%c%c' is PRODUCT/ASSIGNMENT operator.", ch1
    , ch2);
165             }
166             else if(ch2 == ' '){
167                 printf("\n\t\t'%c' is PRODUCT operator.", ch1);
168             }
169             else{
170                 printf("\n\t\t'%c' is not a valid operator.", ch1);
171                 return 0;
172             }
173             break;
174 
175         case '/':
176             if(ch2 == '='){
177                 printf("\n\t\t'%c%c' is DIVISION/ASSIGNMENT operator.",
    ch1, ch2);
178             }
179             else if(ch2 == ' '){
180                 printf("\n\t\t'%c' is DIVISION operator.", ch1);
181             }
182             else{
183                 printf("\n\t\t'%c' is not a valid operator.", ch1);
184                 return 0;
185             }
186             break;
187 
```

```
188         case '%':
189             if(ch2 == '='){
190                 printf("\n\t\t'%c%c' is MODULO/ASSIGNMENT operator.", ch1,
     ch2);
191             }
192             else if(ch2 == ' '){
193                 printf("\n\t\t'%c' is MODULO operator.", ch1);
194             }
195             else{
196                 printf("\n\t\t'%c' is not a valid operator.", ch1);
197                 return 0;
198             }
199             break;
200 
201         case '=':
202             if(ch2 == '='){
203                 printf("\n\t\t'%c%c' is EQUALITY operator.", ch1, ch2);
204             }
205             else if(ch2 == ' '){
206                 printf("\n\t\t'%c' is ASSIGNMENT operator", ch1);
207             }
208             else{
209                 printf("\n\t\t'%c' is not a valid operator.", ch1);
210                 return 0;
211             }
212             break;
213 
214         case '>':
215             if(ch2 == '='){
216                 printf("\n\t\t'%c%c' is GREATER THAN/EQUAL TO operator.",
    ch1, ch2);
217             }
218             else if(ch2 == ' '){
219                 printf("\n\t\t'%c' is GREATER THAN operator.", ch1);
220             }
221             else{
222                 printf("\n\t\t'%c%c' is not a valid operator.", ch1, ch2);
223                 return 0;
224             }
225             break;
226 
227         case '<':
228             if(ch2 == '='){
229                 printf("\n\t\t'%c%c' is LESSER THAN/EQUAL TO operator.",
    ch1, ch2);
230             }
231             else if(ch2 == ' '){
232                 printf("\n\t\t'%c' is LESSER THAN operator.", ch1);
233             }
234             else{
235                 printf("\n\t\t'%c%c' is not a valid operator.", ch1, ch2);
```

```
                return 0;
            }
            break;

        case '!':
            printf("\n\t\t'%c' is a NOT operator.", ch1);
            break;

        default:
            printf("\n\t\t'%c' is a not a valid operator.", ch1);
            return 0;
    }

    return 1;
}

int lexicalParse(char *str){
    //Parse the given string to check for validity
    int left = 0, right = 0, len = strlen(str), status = 1;

    while(right <= len && left <= right){
        //While we are within the valid bounds of the string, check:

        if(isDelimiter(str[right]) == 0){
            //If we do not encounter a delimiter, keep moving forward
            //"right" points to the next character
            right++;
        }

        if(isDelimiter(str[right]) == 1 && left == right){
            //If it is a delimiter, and we haven't parsed it yet

            if(isOperator(str[right]) == 1){
                //Check if the delimiter is an operator
                if((right + 1) <= len && isOperator(str[right + 1]) == 1){
                    //Check if the next character is also an operator
                    status = printOperator(str[right], str[right + 1]);
                    right++;
                }

                else{
                    //Next character is not an operator
                    status = printOperator(str[right], ' ');
                }

                //printf("\n\t\t'%c' is an operator.", str[right]);
            }

            right++;
            left = right;
        }
```

```
287
288         else if(isDelimiter(str[right]) == 1 && left != right || (right ==
     len && left != right)){
289             //We encountered a delimiter in the "right" position, but left
     != right, thus a chunk of
290             //unparsed characters exist between left and right
291
292             //Make a substring of the unparsed characters
293             char *sub = subString(str, left, right - 1);
294
295             if(isPreprocessorDirective(sub) == 1){
296                 //Check if substring is preprocessor directive
297                 printf("\n\t\t'%s' is a valid preprocessor directive.",
    sub);
298             }
299             else if(isValidIdentifier(sub) == 1){
300                 //Check if substring is a valid identifier
301                 printf("\n\t\t'%s' is a valid identifier.", sub);
302             }
303             else if(isInteger(sub) == 1){
304                 //Check if substring is an integer
305                 printf("\n\t\t'%s' is an integer.", sub);
306             }
307             else if(isKeyword(sub) == 1){
308                 //Check if substring is a keyword
309                 printf("\n\t\t'%s' is a valid keyword.", sub);
310             }
311             else if(isValidIdentifier(sub) == 0 && isDelimiter(str[right -
     1]) == 0){
312                 //Otherwise, print that it is not a valid identifier
313                 status = 0;
314                 printf("\n\t\t'%s' is not a valid identifier.", sub);
315             }
316
317             left = right;   //We have parsed the chunk, thus "left" = "
    right"
318         }
319
320     }
321
322     return status;
323 }
```

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.
```