

**EGE UNIVERSITY**

**FACULTY OF ENGINEERING**

**COMPUTER ENGINEERING DEPARTMENT**

**PROGRAMMING LANGUAGES**

**2020–2021 SPRING SEMESTER**

**PROJECT-1 REPORT**

**(LEXICAL ANALYZER)**

**DELIVERY DATE**

07/06/2021

**PREPARED BY**

05180000056, Emre BALKAYA

05180000091, Kutay AVCI

**EXPLANATION OF CODE**

Lexical analysis is the first phase of a compiler. It takes the modified source code from language preprocessors that are written in the form of sentences. The lexical analyzer breaks these syntaxes into a series of tokens, by removing any whitespace or comments in the source code.

If the lexical analyzer finds a token invalid, it generates an error. The lexical analyzer works closely with the syntax analyzer. It reads character streams from the source code, checks for legal tokens, and passes the data to the syntax analyzer when it demands.

The lexical analyzer for the BigAdd language works from the command line with the command LA, and takes the script file’s name as the only attribute. After that the lexical analyzer reads the script that have “.ba” extension. The lexical analyzer reads the script and put it in char array. Then it separates the script into tokens by space character. Later, the lexical analyzer detect tokens where it belongs by rules of BigAdd language and write the tokens with their category on the file with “.lx” extension. If there is an error with tokens, writes it on screen.

## 

**SOURCE CODE**

#include <stdbool.h>

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

/\* run this program using the console pauser or add your own getch, system("pause") or input loop \*/

bool isValidSeperator(char ch) {

if (ch == ' ' || ch == ',' || ch == '.' || ch == '{' || ch == '[' || ch == ']' )

{

return (true);

}

return (false);

}

// Returns 'true' if the string is a VALID IDENTIFIER.

bool isvalidIdentifier(char\* str){

if(isalpha(str[0])){

int i;

for(i;i<strlen(str);i++)

{

if(!isalnum(str[i]))

{

return (false);

}

}

return (true);

}

return (false);

}

bool isValidKeyword(char\* str) {

if (!strcmp(str, "move") || !strcmp(str, "to") || !strcmp(str, "add") || !strcmp(str, "sub") || !strcmp(str, "out") || !strcmp(str, "loop") || !strcmp(str, "int")

|| !strcmp(str, "times") || !strcmp(str, "newline") || !strcmp(str, "from"))

{

return (true);

}

return (false);

}

bool isValidInteger(char\* str) {

int i, len = strlen(str);

if (len == 0)

return (false);

for (i = 0; i < len; i++) {

if (str[i] != '0' && str[i] != '1' && str[i] != '2'&& str[i] != '3' && str[i] != '4' && str[i] != '5'

&& str[i] != '6' && str[i] != '7' && str[i] != '8' && str[i] != '9' || (str[i] == '-' && i > 0))

return (false);

}

return (true);

}

char\* subString(char\* str, int left, int right) {

int i;

char\* subStr = (char\*)malloc( sizeof(char) \* (right - left + 2));

for (i = left; i <= right; i++)

subStr[i - left] = str[i];

subStr[right - left + 1] = '\0';

return (subStr);

}

void detectTokens(char\* str, char\* filename) {

FILE \*laFile;

laFile = fopen(filename,"w");

if(laFile == NULL)

{

printf("File could not be found");

}

else

{

int left = 0, right = 0;

int length = strlen(str);

int closeindex = 0;

while (right <= length && left <= right) {

if (isValidSeperator(str[right]) == false)

right++;

if (isValidSeperator(str[right]) == true && left == right) {

if(str[right] == '.')

fprintf(laFile, "%s" , "EndOfLine\n");

else if(str[right] == ',')

fprintf(laFile, "%s" , "Seperator\n");

else if(str[right] == '{')

{

left = right;

right++;

while(str[right] != '}'){

if(right<strlen(str)-1)

right++;

if(right >= strlen(str)-1)

break;

}

if(str[right] != '}'){

printf("%s" , "CommentError: Comment line has not finished \n");

}

}

else if(str[right] == '[')

{

fprintf(laFile, "%s" , "OpenBlock\n");

left = right;

right++;

while(str[right] != ']'){

if(right<strlen(str)-1)

right++;

if(right >= strlen(str)-1)

break;

}

if(str[right] != ']'){

printf("%s" , "BlockError: There is no CloseBlock \n");

}

if(str[right] == ']'){

closeindex = right;

}

right = left;

}

else if(str[right] == ']'){

if(right == closeindex){

fprintf(laFile, "%s" , "CloseBlock\n");

}

else{

printf("%s" , "BlockError: There is no OpenBlock \n");

}

}

right++;

left = right;

}

else if (isValidSeperator(str[right]) == true && left != right || (right == length && left != right)) {

char\* subStr = subString(str, left, right - 1);

if (isValidKeyword(subStr) == true)

fprintf(laFile, "Keyword : %s\n", subStr);

else if (isValidInteger(subStr) == true && strlen(subStr)<=5)

fprintf(laFile, "Integer : %s\n", subStr);

else if (isValidInteger(subStr) == true && strlen(subStr)>5)

printf("Integer Error : An integer can be as big as 100 decimal digits -> %s\n", subStr);

else if (isvalidIdentifier(subStr) == true

&& isValidSeperator(str[right - 1]) == false && strlen(subStr)<=20)

fprintf(laFile, "Identifier : %s\n", subStr);

else if (isvalidIdentifier(subStr) == true

&& isValidSeperator(str[right - 1]) == false && strlen(subStr)>20)

printf("Identifier Error : Variable names have a maximum length of 20 characters -> %s\n", subStr);

else if (isvalidIdentifier(subStr) == false

&& isValidSeperator(str[right - 1]) == false && subStr[0] == '"' && subStr[strlen(subStr)-1] == '"')

fprintf(laFile, "StringConstant : %s\n", subStr);

else if (isvalidIdentifier(subStr) == false

&& isValidSeperator(str[right - 1]) == false)

printf("Identifier Error : Unknown character -> %s\n", subStr);

left = right;

}

}

}

fclose(laFile);

return;

}

int main(int argc, char \*argv[]) {

char input[50];

char fileName[50] = "";

char baFile[30] = "";

char lxFile[30] = "";

start:

printf("\nType any command: ");

gets(input);

char \* token = strtok(input, " ");

if( strcmp(token, "la"))

{

printf("\nInvalid command -> %s\n", token );

goto start;

}

int i = 0;

while( token != NULL ) {

token = strtok(NULL, " ");

i++;

if(i==1){

strcat(fileName,token);

}

}

strcat(baFile,fileName);

strcat(baFile,".ba");

strcat(lxFile,fileName);

strcat(lxFile,".lx");

FILE \*file;

file = fopen(baFile,"r");

char str[99999];

char s[99999];

char d[99999];

if(file == NULL)

{

printf("File could not be found");

}

else

{

int count = 0;

while(!feof(file)){

fscanf(file,"%s",d);

count++;

}

rewind(file);

int counter = 0;

while(!feof(file)){

fscanf(file,"%s",s);

strcat(s," ");

strcat(str,s);

counter++;

if(counter == (count-1))

{

break;

}

}

detectTokens(str, lxFile);

}

fclose(file);

return 0;

}

**SCREENSHOTS**

BA File:

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

Console:

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu

LX File:

tablo içeren bir resim

Açıklama otomatik olarak oluşturuldu

**LIMITATIONS**

* The command must start with “la”.
* “la” command’s first parameter must be file name.

For example: la myscript