Lexical Analyzer Report

Nilay YÜCEL[1], **Zeynep KAYA**[2]
[1]Computer Engineering, Engineering Faculty, Dokuz Eylul University, İzmir, Turkey, 2017510082
[2]Computer Engineering, Engineering Faculty, Dokuz Eylül University,Bursa, Turkey, 2017510048

Assignment Date: May 2021 • **Report Date:** June 2021

Abstract: This report is a LaTeXreport prepared for the detailed problems and solutions of the term "Lexical Analyzer" project given within the scope of Dokuz Eylül University Computer Engineering Department CME 3202 Concepts of Programming Language course. The aim of the term project is to make Lexical Analysis of a given programming language.

Key words: Lexical Analysis, C, Compiler, Token, Dev C++, Algorithm Development

1. Introduction

The "Lexical Analyzer" project, which was given as a term paper, requested the correct compile of the input in the given file, adhering to the rules in the C programming language family. Lexical analysis is the process of converting the given expression into a token array. The token mentioned here is a meaningful string of one or more characters.

There are certain requests and limitations within the scope of this project. These will be explained in detail in the third chapter. Based on this requirement list and using the C programming language, a compiler was created. This compiler reads the input given in the .ceng file and analyzes the expression according to the limitations in the written code. As a result of the analysis, the compilation information is recorded in a file with a .lex extension.

2. Requirement

The main purpose of the Lexical Analyzer for Ceng++ project is to compile the language of the given program and determine its tokens. A programming token is the basic component of source code. Characters are categorized as classes of tokens that describe their functions (constants, identifiers, operators, reserved words, and separators) in accordance with the rules of the programming language.[1] The requested token is shown in table 1.

Each of the above-mentioned requirements refers to a token. Performing the file reading operation, identifying the tokens and printing the results to the file is the main requirement of the project.

Table 1. Requirements of the project.

Tokens	
Identifiers	It is the process of detecting identifiers and performing length checks while defining variables.
Integer Constants	It is the process of determining the variables specified as Integer and performing length checks.
Operators	It is the process of determining the valid operators to be used.
Brackets	It is the process of checking the bracket types and numbers of be used.
String Constants	It is the process of detecting constants between double quotes.
Keywords	It is the process of determining identifiers used in other programming languages which serving a special function. (like: break,case,char...)
End of Line	It is the process of detecting the end of line sign that indicates the end of an operation in the written piece of code.
Comments	It is the process of detecting the expressions left as comments in the written code.

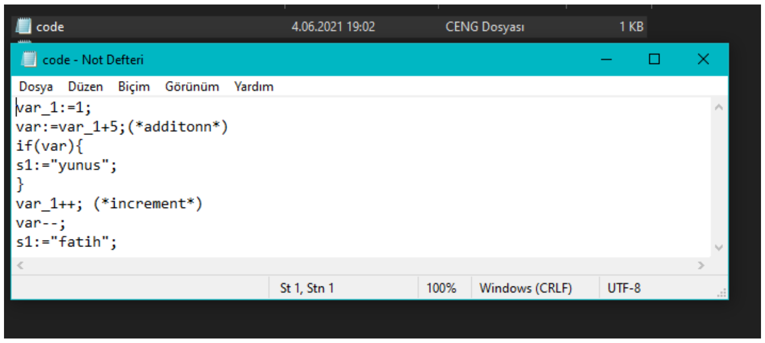
3. Environment

Lexical Analyzer for Ceng++ is a project developed in the C programming language. C is a general-purpose, procedural computer programming language supporting structured programming, lexical variable scope, and recursion, with a static type system.[3] Dev-C++ is preferred as IDE. Dev-C++ is a free full-featured integrated development environment (IDE) distributed under the GNU General Public License for programming in C and C++.[4] [https://en.wikipedia.org/wiki/Dev-C] It has been preferred because of the hardware needs of the software team, because it has a simple interface and is used only for the compile of languages from the C family. Fast and reliable results were obtained, and debugging could be done successfully and clearly. One of the negative aspects is that it gives a meaningless error when trying to compile again without closing the terminal window that is opened after the code is compiled.However, other than that no problem was encountered and the project was successfully completed in this IDE.

4. Progress of Project

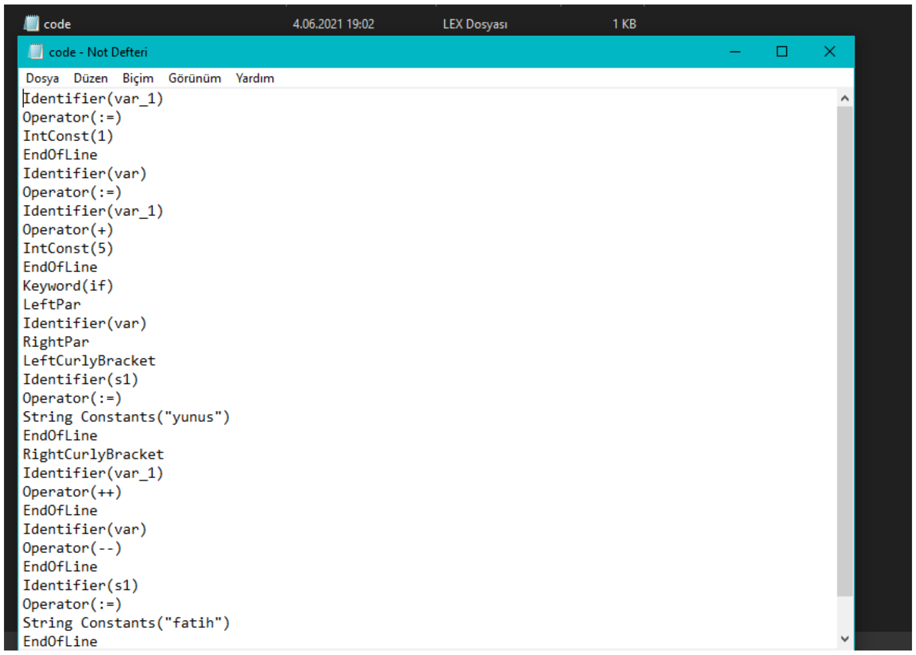
In this project, we were asked to read the expression in a .ceng file and convert it into lexically meaningful expressions. This converted expression was written to a file with a .lex extension. So the first problem to be solved is to determine the appropriate tokens according to the needs. Appropriate controls were decided according to the tokens determined. Possible errors were considered. Parts to prevent these errors have been added to the code.

In the algorithm used in the project, the lab document was referenced. For this reason, no major problems were encountered. But since too many extra controls were added, detailed tests were done on all of them. Problems encountered during the tests and their solutions will be added below. Finally, the project, which passed the tests successfully, was delivered and a presentation was made.



```
var_1:=1;
var:=var_1+5;(*additonn*)
if(var){
s1:="yunus";
}
var_1++; (*increment*)
var--;
s1:="fatih";
```

Figure 1. Input File, .ceng



```
Identifier(var_1)
Operator(:=)
IntConst(1)
EndOfLine
Identifier(var)
Operator(:=)
Identifier(var_1)
Operator(+)
IntConst(5)
EndOfLine
Keyword(if)
LeftPar
Identifier(var)
RightPar
LeftCurlyBracket
Identifier(s1)
Operator(:=)
String Constants("yunus")
EndOfLine
RightCurlyBracket
Identifier(var_1)
Operator(++)
EndOfLine
Identifier(var)
Operator(--)
EndOfLine
Identifier(s1)
Operator(:=)
String Constants("fatih")
EndOfLine
```

Figure 2. Output File, .lex

4.1. Problems Encountered

In this section, mistakes that were not taken into account when planning at the beginning of the project and due to forgotten checks will be discussed. How these problems are solved will be explained in the **Problems Solutions** section.

These errors are:

- Parenthesis check: It has been noticed that the braces used for comment lines are also taken into account when checking whether the parenthesis are opened or closed correctly. This was supposed to be prevented because statements written inside the comment should be ignored.

- Binary operator check(:= , ++ ,–): Any other two operators should not be next to each other except the binary enclosed in parentheses. If it does, an error message should be given. A check had to be made to adjust this situation.

- End of File Control: The project should see and indicate this error when the conditions such as comment line, end of line, parentheses that should be finished before the program ends are not finished. For this, in addition to the line control, it is necessary to check the file in general.

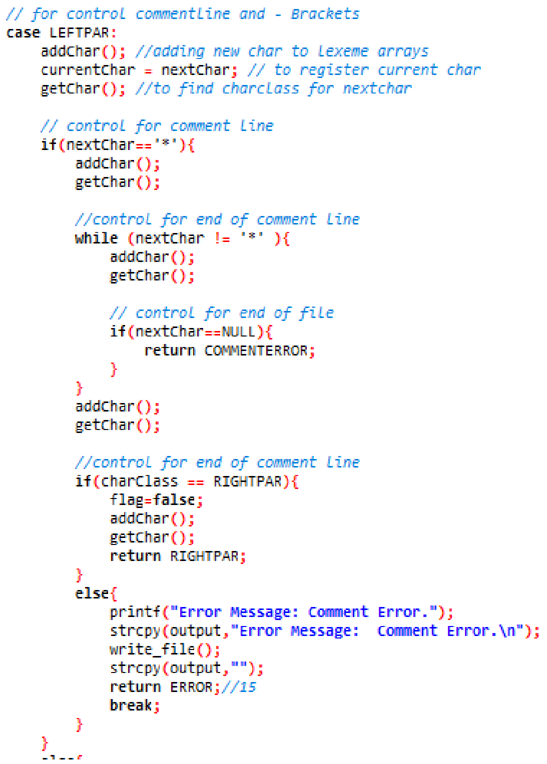
4.2. Problems Solutions

The problems encountered during the project were mentioned above. In this section, the solutions found for these errors will be discussed.

First of all, the instructor of the course was interviewed for situations that were not understood in the project paper. It was learned that no additional control was required other than what was clearly requested on the paper.

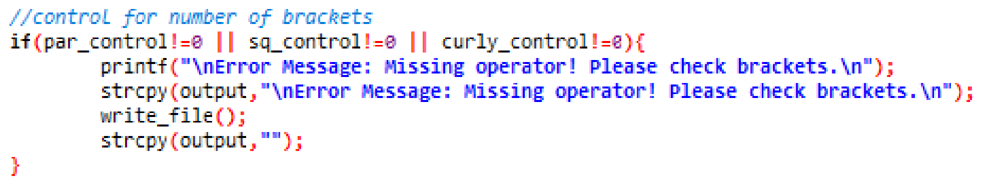
There were two issues encountered after starting the code. Both of these issues are related to binary operators ((*, *), :=, ++, −), which must be used side by side. A common algorithm was found to solve the problem. With this algorithm, currentChar(4) is started to be held in a variable before we go to look at what nextChar(2) is. The current and next char pairs were checked before starting the operator and parentheses checks. If an operator pair that does not match these is seen, an error message is printed.

The problem specified as end-of-file control, on the other hand, is solved with variables that specify the numbers and positions of the expression used when we come to the end of the file, except for the controls in the loop. The cause of the error can be found easily by checking them at the end of the file.



```
// for control commentline and - Brackets
case LEFTPAR:
    addChar(); //adding new char to lexeme arrays
    currentChar = nextChar; // to register current char
    getChar(); //to find charclass for nextchar

    // control for comment line
    if(nextChar=='*'){
        addChar();
        getChar();

        //control for end of comment line
        while (nextChar != '*' ){
            addChar();
            getChar();

            // control for end of file
            if(nextChar==NULL){
                return COMMENTERROR;
            }
        }
        addChar();
        getChar();

        //control for end of comment line
        if(charClass == RIGHTPAR){
            flag=false;
            addChar();
            getChar();
            return RIGHTPAR;
        }
        else{
            printf("Error Message: Comment Error.");
            strcpy(output,"Error Message:  Comment Error.\n");
            write_file();
            strcpy(output,"");
            return ERROR;//15
            break;
        }
    }
```

Figure 3. Comment Lines Control



```
//control for number of brackets
if(par_control!=0 || sq_control!=0 || curly_control!=0){
        printf("\nError Message: Missing operator! Please check brackets.\n");
        strcpy(output,"\nError Message: Missing operator! Please check brackets.\n");
        write_file();
        strcpy(output,"");
}
```

Figure 4. End of Line, Brackets Control

4.3. Functions

In order to make the code written within the requirements of the project understandable and clear, a total of 9 functions were created. Each function performs a specific operation. Functions and their operations are represented in Table-2 below.

Table 2. Functions of the project.

Functions	
void addChar() (1)	Find and register the desired token.
char getNextChar() (2)	Finding out what happened in the next character
void getChar() (3)	Finding out which token is the next character. (charClass)
int lex() (4)	Current character operations.
void remove_spaces(char input)* (5)	Removing spaces in input
void case_sensitive() (6)	Making all characters smaller
void read_file() (7)	Reading from input file with ".ceng" extension
void write_file() (8)	Writing to output file with ".lex" extension
int main() (9)	Main functions

5. Conclusion

The purpose of this project is to make a lexical analysis of the entries in the given file. In computer science, lexical analysis, lexing or tokenization is the process of converting a sequence of characters into a sequence of tokens.[5]

It was expected that the expressions in the .ceng file given in the homework document would be interpreted in a way that the computer could understand, and would be written to another file with the .lex extension, by converting them into tokens. Token conversion transactions were made within the framework of certain limitations and needs. Details such as the tokens used during this process, keywords, the platform on which the project was written, the problems experienced and their solutions are mentioned above.

References

[1] “Definition of Token”,whatis.techtarget.com, 2021. (Web Page) Available at : 23/06/2021 https://whatis.techtarget.com/definition/token

[2] Dokuz Eylül University, Engineering Faculty, Department of Computer Engineering, Lesson of CME 3202 Concepts of the Programming Languages, Document of Programming Languages Project 1: Lexical Analyzer for Ceng++

[3] “Definition of C Programming Language”, wikipedia.org, 2021. (Web Page) Available at : 23/06/2021 https://en.wikipedia.org/wiki/C_(programming_language)

[4] “Definition of Dev-C++”, wikipedia.org, 2021. (Web Page) Available at : 23/06/2021 https://en.wikipedia.org/wiki/Dev-C

[5] “Lexical Analysis”, wikipedia.org, 2021. (Web Page) Available at : 23/06/2021 https://en.wikipedia.org/wiki/Lexical_analysis