MANİSA
CELAL BAYAR
ÜNİVERSİTESİ

FACULTY OF ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

2022-2023 SPRING

PROGRAMMING LANGUAGES ASSIGNMENT REPORT

NAME: ENES

SURNAME: DOĞAN

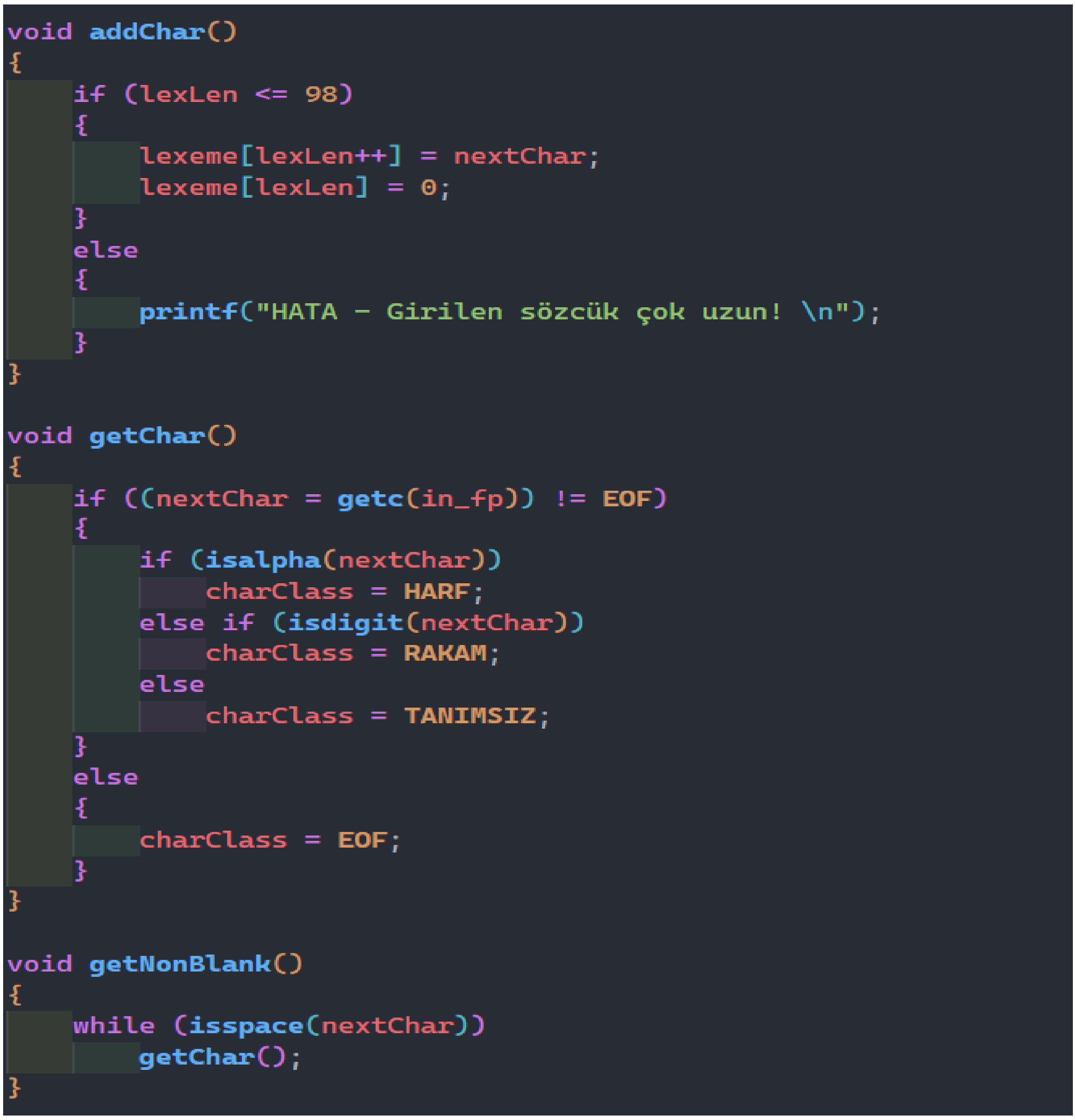
STUDENT NUMBER: 200315030

NAME: EMRE

SURNAME: BİÇER

STUDENT NUMBER: 200315081

addChar() – getChar() – getNonBlank()



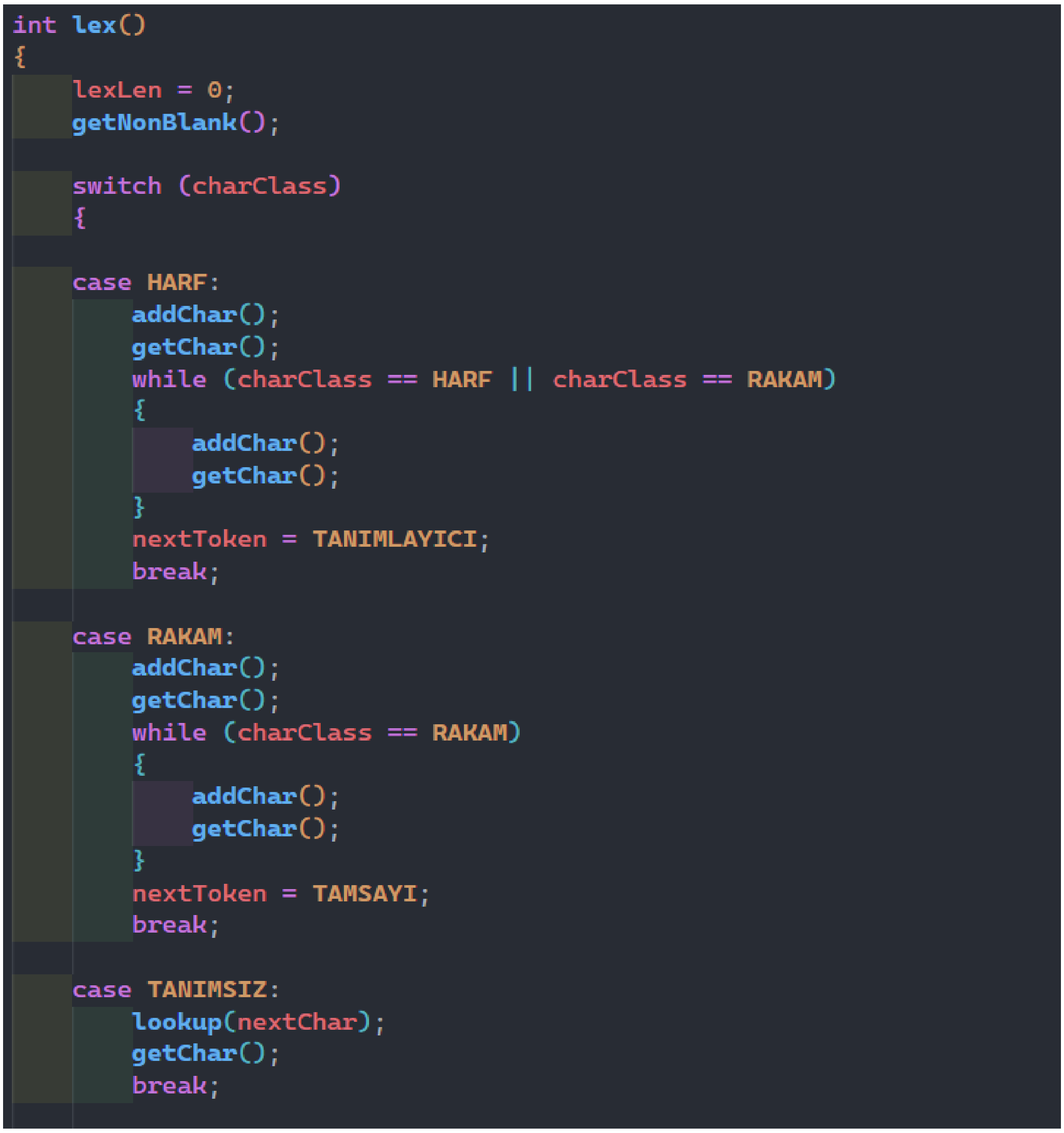
```c
void addChar()
{
    if (lexLen <= 98)
    {
        lexeme[lexLen++] = nextChar;
        lexeme[lexLen] = 0;
    }
    else
    {
        printf("HATA - Girilen sözcük çok uzun! \n");
    }
}

void getChar()
{
    if ((nextChar = getc(in_fp)) != EOF)
    {
        if (isalpha(nextChar))
            charClass = HARF;
        else if (isdigit(nextChar))
            charClass = RAKAM;
        else
            charClass = TANIMSIZ;
    }
    else
    {
        charClass = EOF;
    }
}

void getNonBlank()
{
    while (isspace(nextChar))
        getChar();
}
```

The addChar() function is used to add characters to the lexeme array in the language we have created. It keeps track of the length of the inserted character by checking the lexLen variable. If lexLen is less than or equal to 98, it adds characters to the lexeme array. Otherwise, the message “HATA - Girilen sözcük çok uzun!” returns the message. This function is also called in the lookup() method and is used in the expression of tokens.

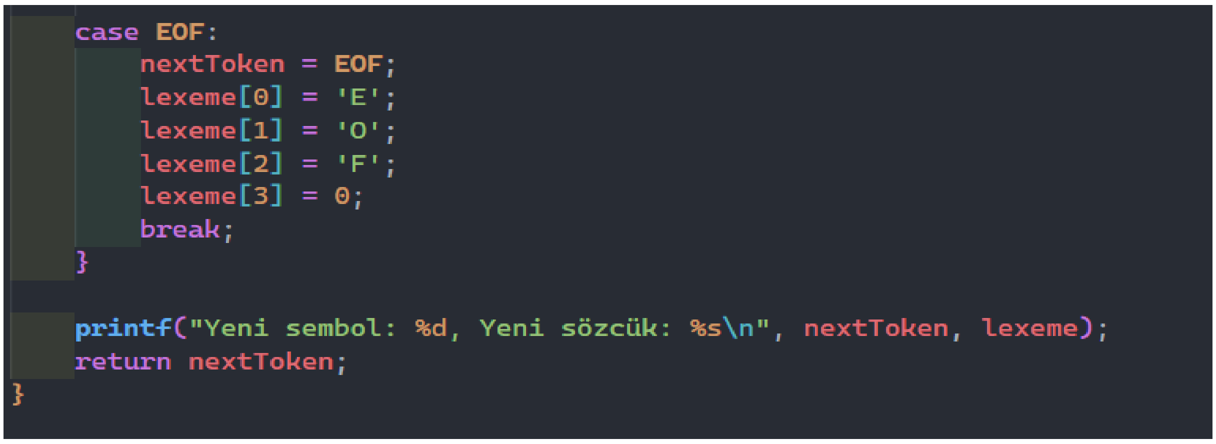
The getChar() function basically assigns the next character to the nextChar variable, but if the end of the file is not reached in the reading operation, it applies this operation. If the value in nextChar is a letter, it assigns a HARF to the variable charClass. If nextChar is a number, then it assigns a RAKAM value to the variable charClass. Otherwise, it assigns the value TANIMSIZ to the charClass variable. If the end of the file is reached, it assigns the value EOF to the charClass variable.

The getNonBlank() function skips whitespace characters and includes the next character. This function checks for space characters using the isspace() function. It also gets the next character by calling the getChar() function.

lex()



```c
int lex()
{
    lexLen = 0;
    getNonBlank();

    switch (charClass)
    {

    case HARF:
        addChar();
        getChar();
        while (charClass == HARF || charClass == RAKAM)
        {
            addChar();
            getChar();
        }
        nextToken = TANIMLAYICI;
        break;

    case RAKAM:
        addChar();
        getChar();
        while (charClass == RAKAM)
        {
            addChar();
            getChar();
        }
        nextToken = TAMSAYI;
        break;

    case TANIMSIZ:
        lookup(nextChar);
        getChar();
        break;

```

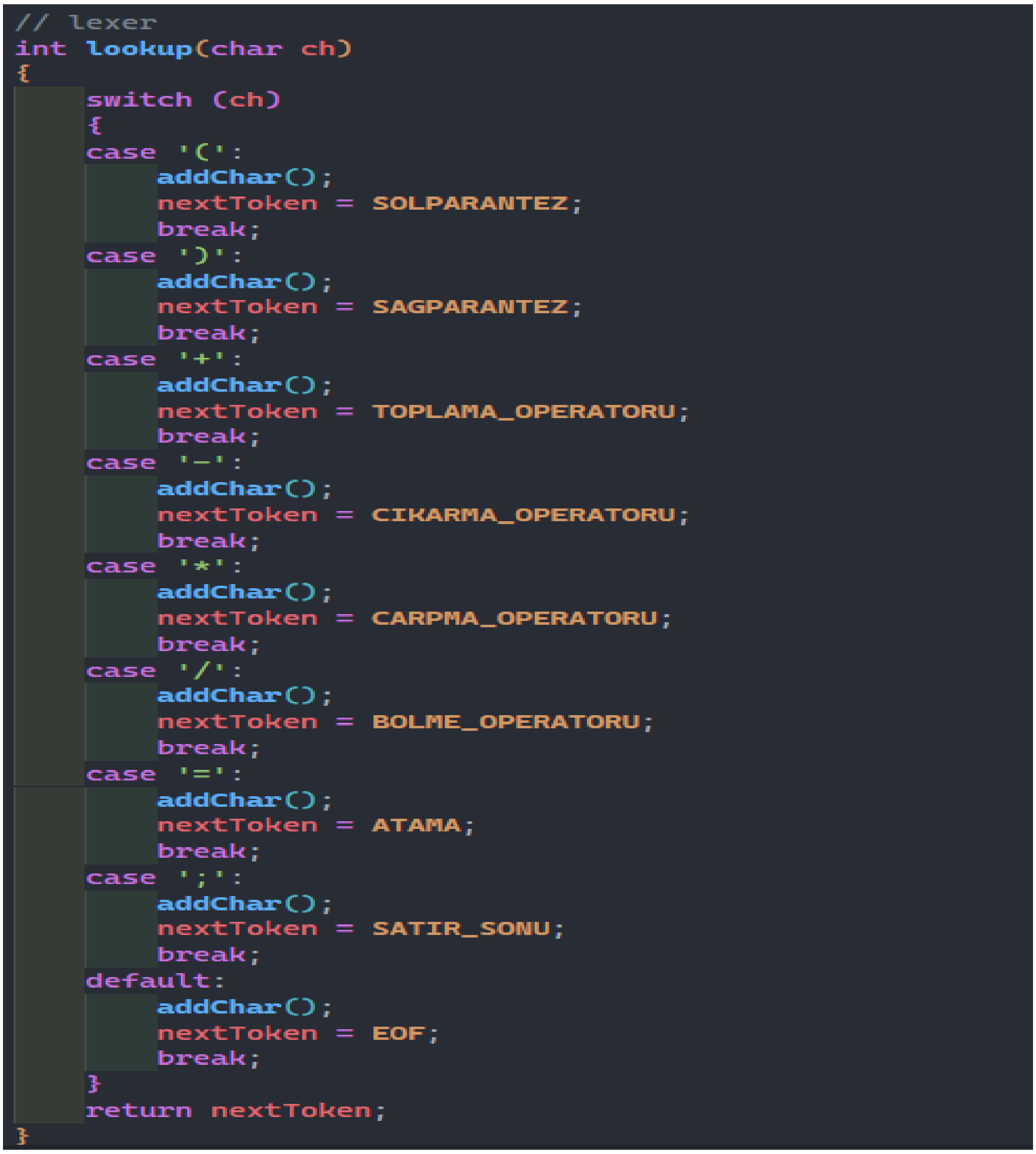


```c
    case EOF:
        nextToken = EOF;
        lexeme[0] = 'E';
        lexeme[1] = 'O';
        lexeme[2] = 'F';
        lexeme[3] = 0;
        break;
    }

    printf("Yeni sembol: %d, Yeni sözcük: %s\n", nextToken, lexeme);
    return nextToken;
}
```

The lex() function performs many tasks as a single function. This function determines the token of the next symbol and assigns it to the nextToken variable. It also resets the lexLen variable and calls getNonBlank() to assign whitespace characters. The lexLen() function recognizes the symbol as a switch-case structure relative to charClass and sets the nextToken.

If the value of nextToken() is LETTER, it indicates an identifier symbol and sets nextToken as Identifier. If the value of nextToken() is NUMBER, it indicates an integer symbol and sets the nextToken to an INTEGATE. If the value of nextToken() is UNDEFINED, it indicates that the symbol is an operator or one of the other symbols. If nextToken() is EOF, it indicates the end of the file and sets the nextToken() result to EOF. Finally, it prints nextToken and lexeme and returns nextToken().

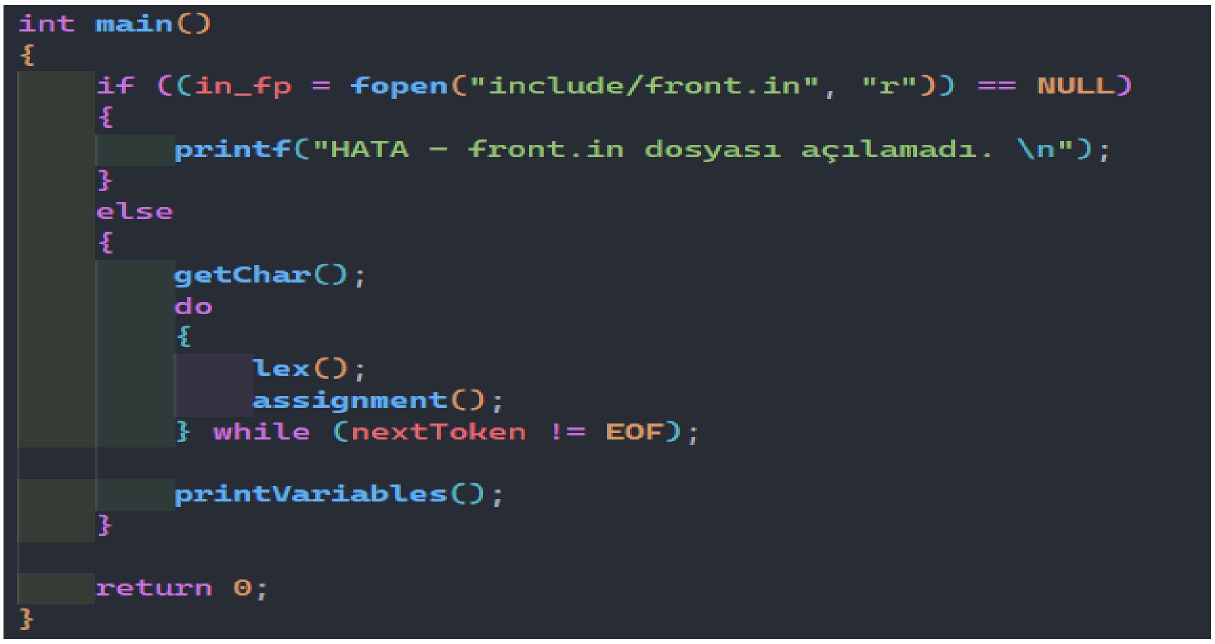
lookup()



```
// lexer
int lookup(char ch)
{
    switch (ch)
    {
    case '(':
        addChar();
        nextToken = SOLPARANTEZ;
        break;
    case ')':
        addChar();
        nextToken = SAGPARANTEZ;
        break;
    case '+':
        addChar();
        nextToken = TOPLAMA_OPERATORU;
        break;
    case '-':
        addChar();
        nextToken = CIKARMA_OPERATORU;
        break;
    case '*':
        addChar();
        nextToken = CARPMA_OPERATORU;
        break;
    case '/':
        addChar();
        nextToken = BOLME_OPERATORU;
        break;
    case '=':
        addChar();
        nextToken = ATAMA;
        break;
    case ';':
        addChar();
        nextToken = SATIR_SONU;
        break;
    default:
        addChar();
        nextToken = EOF;
        break;
    }
    return nextToken;
}
```

This function recognizes the symbol according to the given character and sets the nextToken accordingly. The lookup function takes a char type parameter and checks the character that comes with this parameter. The lookup function also determines the symbol corresponding to the character using the switch-case structure and returns the nextToken.

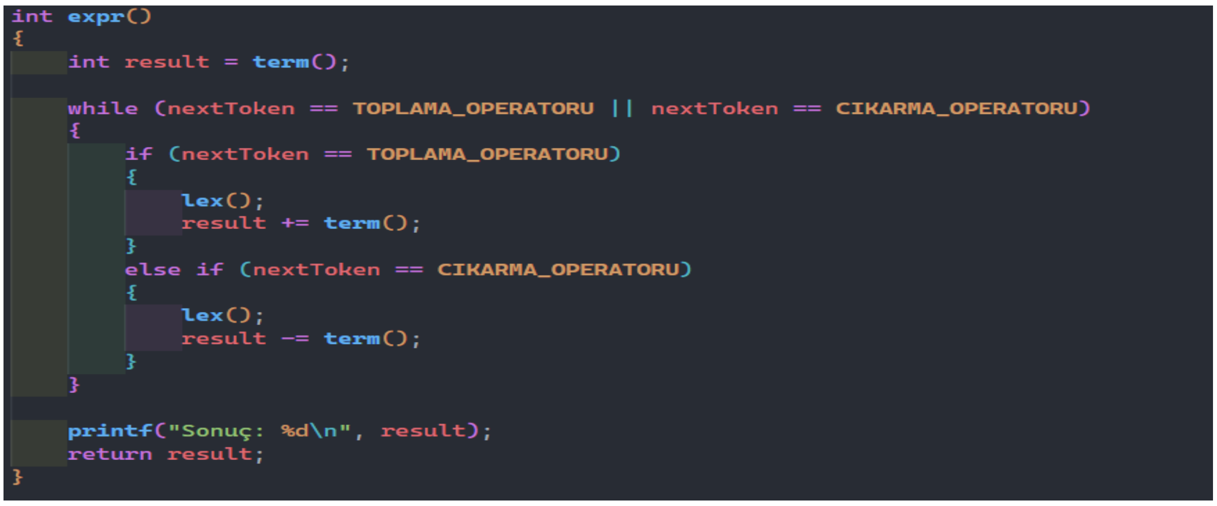
main()



```c
int main()
{
    if ((in_fp = fopen("include/front.in", "r")) == NULL)
    {
        printf("HATA - front.in dosyası açılamadı. \n");
    }
    else
    {
        getChar();
        do
        {
            lex();
            assignment();
        } while (nextToken != EOF);

        printVariables();
    }

    return 0;
}
```

This function represents the main function of the program. Opens the originally specified input file and assigns it to the in_fp pointer. It gets the next character from here by calling the getChar() function. It processes all the tokens by calling the lex() function inside the loop and continues the loop until the last token is EOF. The main() function calls the expr() function to analyze the expression and print the result to the screen. Finally, it closes the input file and terminates the program.

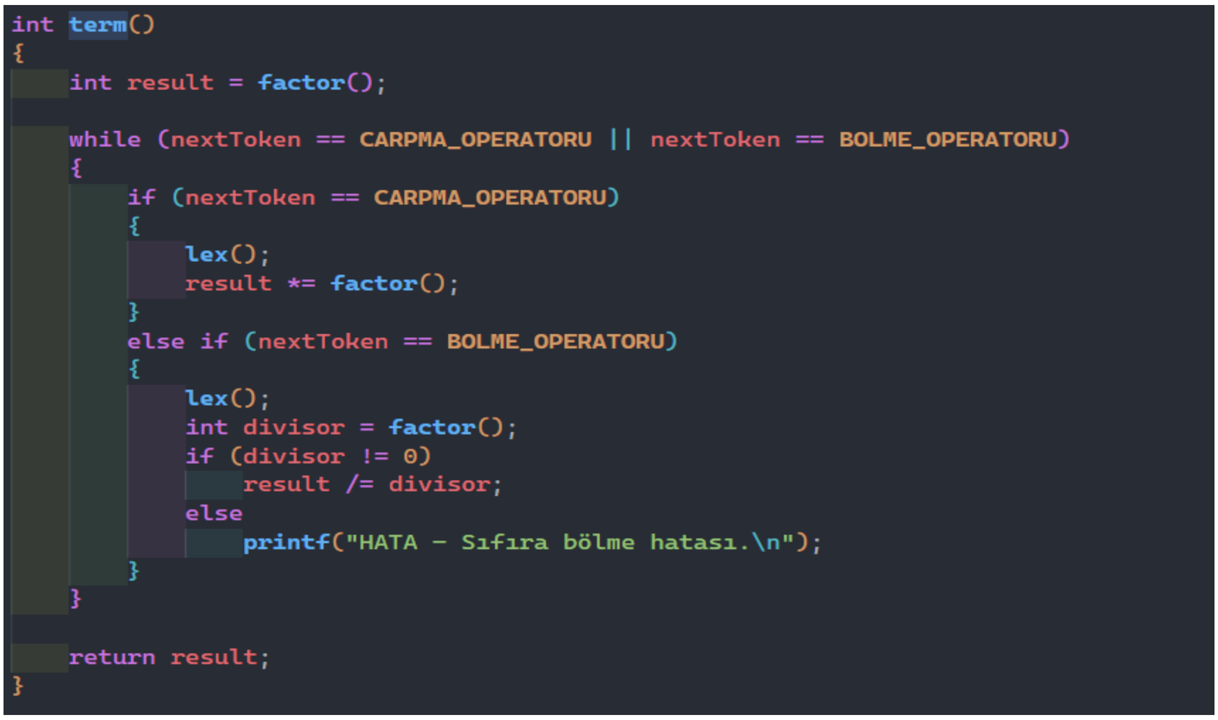
expr()



```c
int expr()
{
    int result = term();

    while (nextToken == TOPLAMA_OPERATORU || nextToken == CIKARMA_OPERATORU)
    {
        if (nextToken == TOPLAMA_OPERATORU)
        {
            lex();
            result += term();
        }
        else if (nextToken == CIKARMA_OPERATORU)
        {
            lex();
            result -= term();
        }
    }

    printf("Sonuç: %d\n", result);
    return result;
}
```

The expr() function generally splits and processes an expression. This function breaks a term by calling the term() function. If the result of nextToken is an addition or subtraction, it performs these operations sequentially. In other cases, it prints the error message to the screen.

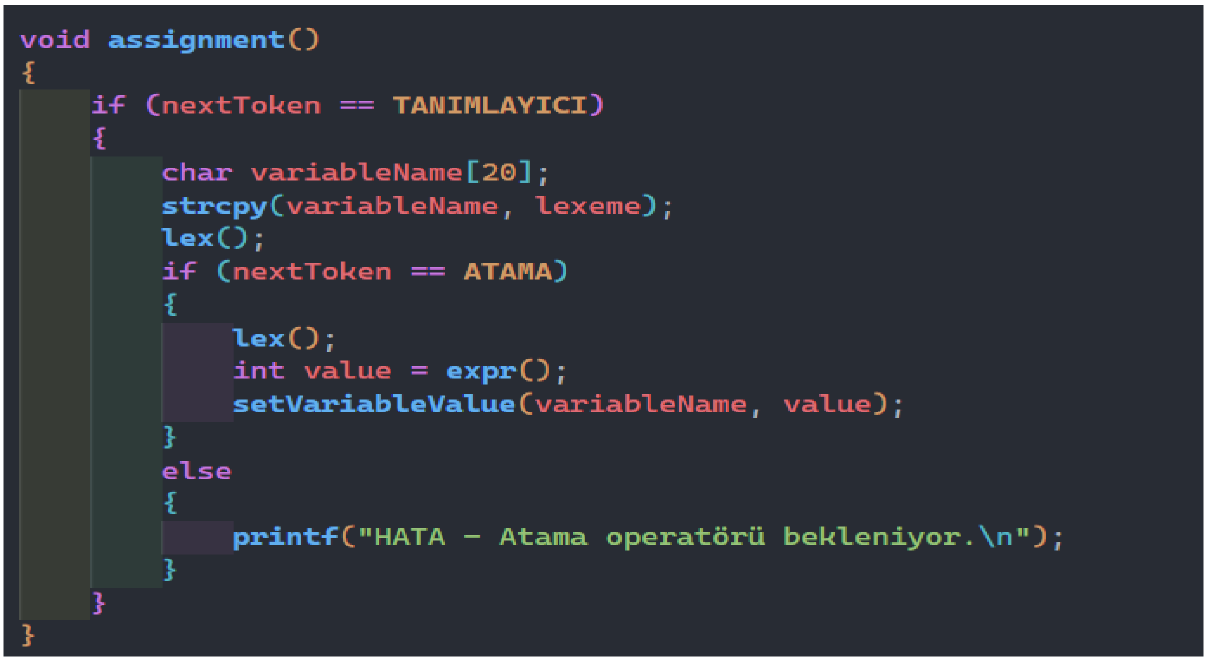
term()



```c
int term()
{
    int result = factor();

    while (nextToken == CARPMA_OPERATORU || nextToken == BOLME_OPERATORU)
    {
        if (nextToken == CARPMA_OPERATORU)
        {
            lex();
            result *= factor();
        }
        else if (nextToken == BOLME_OPERATORU)
        {
            lex();
            int divisor = factor();
            if (divisor != 0)
                result /= divisor;
            else
                printf("HATA - Sıfıra bölme hatası.\n");
        }
    }

    return result;
}
```

This function breaks down a term and processes it. Unlike the expr() function, it breaks down a factor by calling the factor() function. If nextToken is a multiplication or division, it performs these operations sequentially. In other cases, it prints an error message on the screen.

assignment()



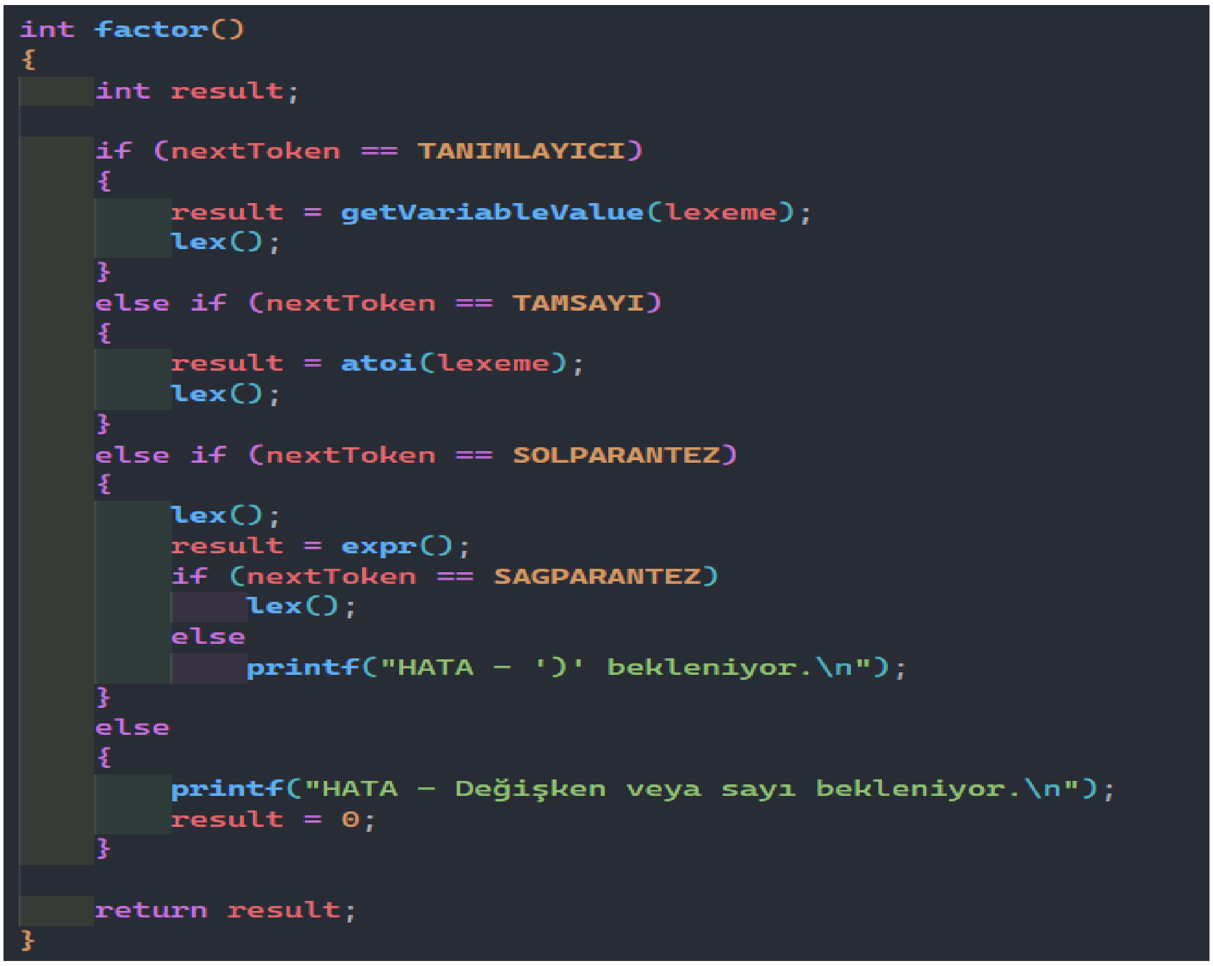
```c
void assignment()
{
    if (nextToken == TANIMLAYICI)
    {
        char variableName[20];
        strcpy(variableName, lexeme);
        lex();
        if (nextToken == ATAMA)
        {
            lex();
            int value = expr();
            setVariableValue(variableName, value);
        }
        else
        {
            printf("HATA - Atama operatörü bekleniyor.\n");
        }
    }
}
```

This function is the function that allows us to assign a value to a variable that we have previously determined. The ATAMA token is used. If the assignment has not been completed, the message " HATA – Atama operatörü bekleniyor." prints.

isVariable(), getVariableIndex(), addVariable(), getVariableValue(), setVariableValue(), printVariables()

These functions are auxiliary functions that have functions such as returning the value, adding a new value, changing the value, printing the value on the screen when a value is assigned.

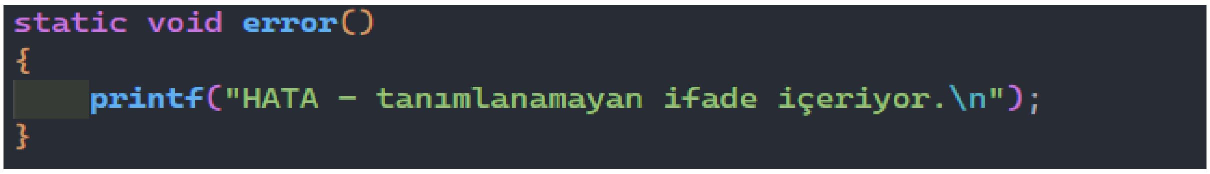
factor()



```c
int factor()
{
    int result;

    if (nextToken == TANIMLAYICI)
    {
        result = getVariableValue(lexeme);
        lex();
    }
    else if (nextToken == TAMSAYI)
    {
        result = atoi(lexeme);
        lex();
    }
    else if (nextToken == SOLPARANTEZ)
    {
        lex();
        result = expr();
        if (nextToken == SAGPARANTEZ)
            lex();
        else
            printf("HATA - ')' bekleniyor.\n");
    }
    else
    {
        printf("HATA - Değişken veya sayı bekleniyor.\n");
        result = 0;
    }

    return result;
}
```

The main function of the factor() function breaks down and processes a factor. If nextToken is an identifier or an integer, it prints the nextToken and lexeme on the screen. nextToken SOLPARANTEZ calls the expr() function and processes nested statements. nextToken SAGPARANTEZ terminates the transaction. In other cases, it prints an error message on the screen.

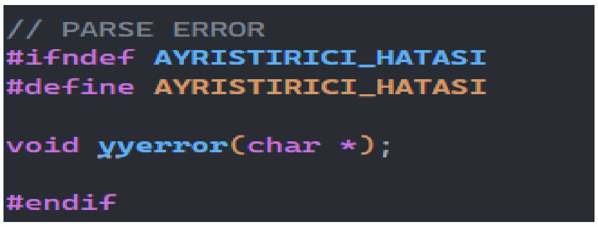
error()



```
static void error()
{
    printf("HATA - tanımlanamayan ifade içeriyor.\n");
}
```

This function is a function that is called when an error occurs. When called, "HATA - Geçersiz karakter!" prints the message.

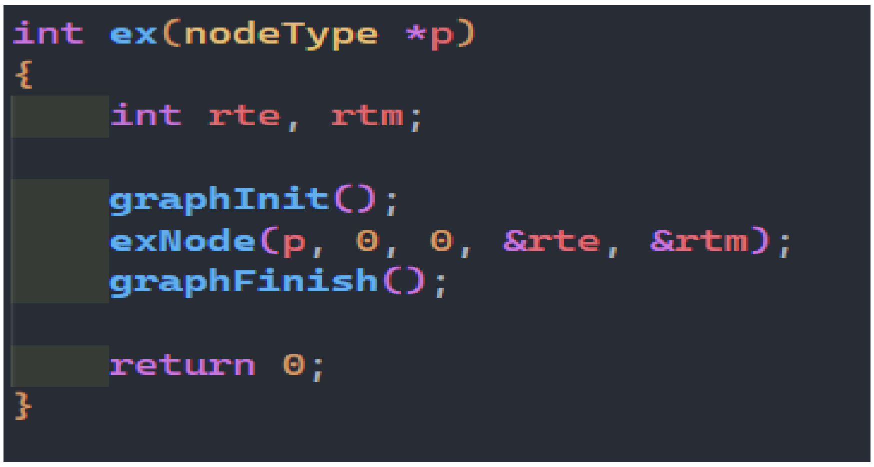
yyerror()



```
// PARSE ERROR
#ifndef AYRISTIRICI_HATASI
#define AYRISTIRICI_HATASI

void yyerror(char *);

#endif
```

1This function has a function that reports parsing errors. If the analysis results do not correspond to the file, they are not defined in this file or should be found elsewhere.

ex()



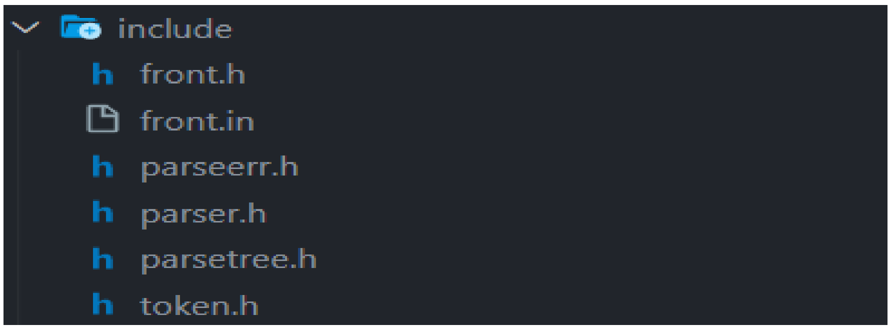
```
int ex(nodeType *p)
{
    int rte, rtm;

    graphInit();
    exNode(p, 0, 0, &rte, &rtm);
    graphFinish();

    return 0;
}
```

This function has a construct that is used to visualize the fragmented expression. Draws the shredded tree and prints it on the screen.

graphInit(), exNode(), graphDrawBox(), graphBox(), graphDrawArrow(), graphFinish()

These functions are helper functions used to visualize the fragmented expression.

include Folder



include
front.h
front.in
parseerr.h
parser.h
parsetree.h
token.h

· The front.h file does the symbol definitions and declaration of the lex function.

· The front.in file provides our Ebedy language we wrote runs the code in this file.

· The parseer.h file declares the yyerror function.

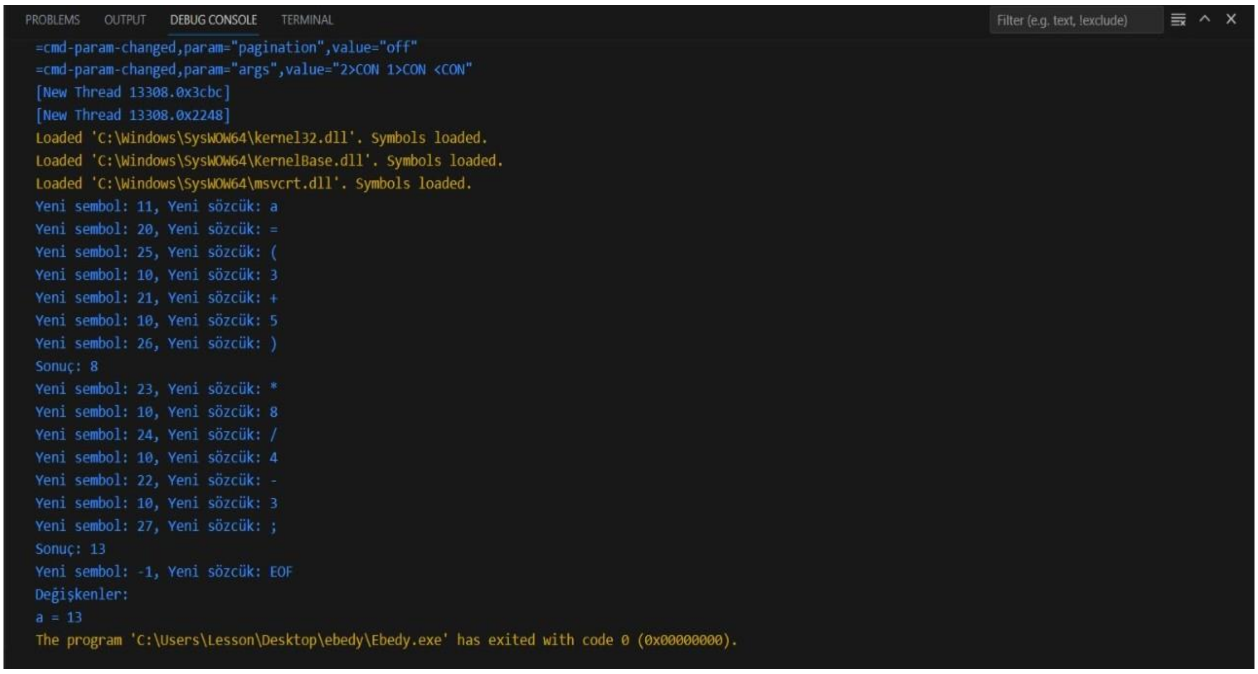
· The parser.h file defines data structures and function declarations, as well as a structure called node. The node structure contains a union data structure used to store different types of data. It contains a composite data structure member named opr and a character member named operator.

· The parsetree.h file declares an ex() function with symbol definitions and header file parser.h included.

· The token.h file includes the stdlib.h header file and defines the TOKEN_H symbol. Next, a structure named token_T is defined and contains the declaration of the init_token function of type token_T *, which is a pointer of type token_T. The token_T Structure in token.h consists of two members, type and value. The type member represents the token types (TOKEN_ID, TOKEN_EQUALS, etc.) defined in the enum type. The value member represents a string (char *) value. The init_token structure in this file takes the type and value parameters and returns a value of type token_T *, which is a pointer of type token_T. The function allocates space in memory for a structure of type token_T (using the malloc function), assigns the type and value parameters to the structure, and returns the generated structure pointer.

Outputs

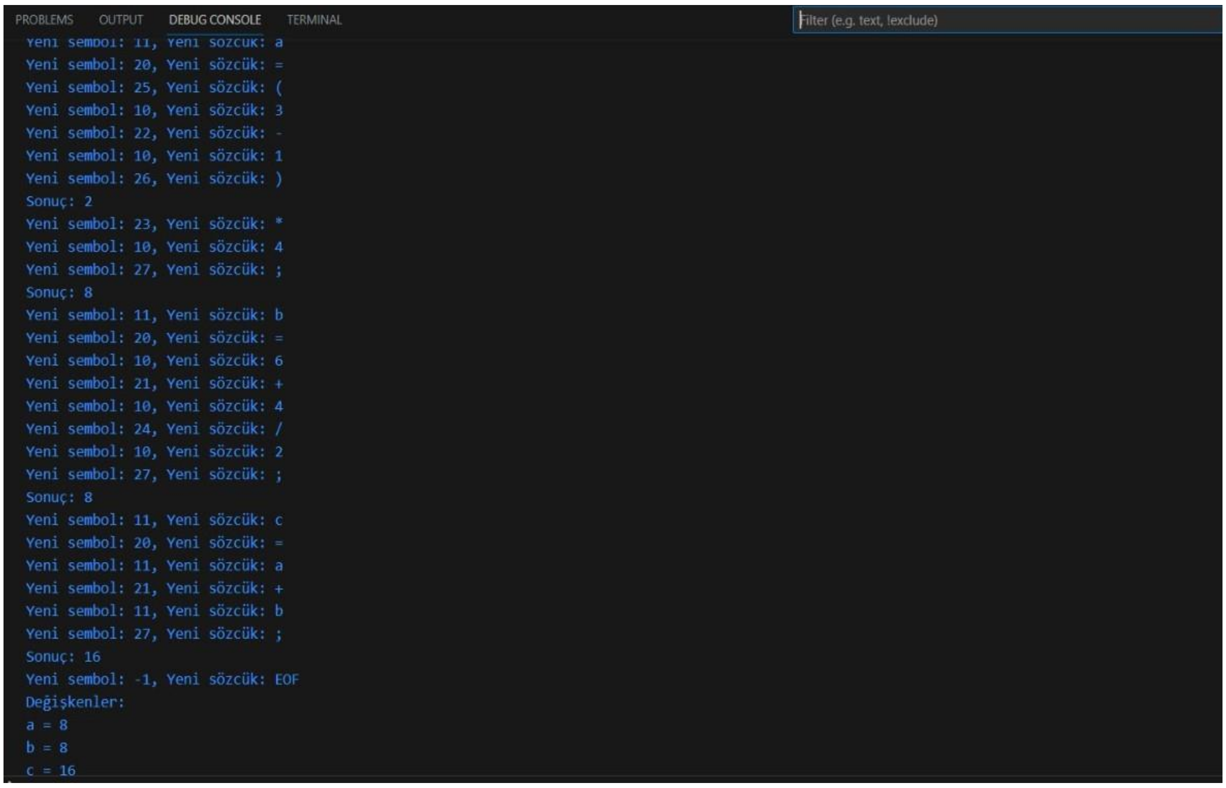
· example1.txt



```
=cmd-param-changed,param="pagination",value="off"
=cmd-param-changed,param="args",value="2>CON 1>CON <CON"
[New Thread 13308.0x3cbc]
[New Thread 13308.0x2248]
Loaded 'C:\Windows\SysWOW64\kernel32.dll'. Symbols loaded.
Loaded 'C:\Windows\SysWOW64\KernelBase.dll'. Symbols loaded.
Loaded 'C:\Windows\SysWOW64\msvcrt.dll'. Symbols loaded.
Yeni sembol: 11, Yeni sözcük: a
Yeni sembol: 20, Yeni sözcük: =
Yeni sembol: 25, Yeni sözcük: (
Yeni sembol: 10, Yeni sözcük: 3
Yeni sembol: 21, Yeni sözcük: +
Yeni sembol: 10, Yeni sözcük: 5
Yeni sembol: 26, Yeni sözcük: )
Sonuç: 8
Yeni sembol: 23, Yeni sözcük: *
Yeni sembol: 10, Yeni sözcük: 8
Yeni sembol: 24, Yeni sözcük: /
Yeni sembol: 10, Yeni sözcük: 4
Yeni sembol: 22, Yeni sözcük: -
Yeni sembol: 10, Yeni sözcük: 3
Yeni sembol: 27, Yeni sözcük: ;
Sonuç: 13
Yeni sembol: -1, Yeni sözcük: EOF
Değişkenler:
a = 13
The program 'C:\Users\Lesson\Desktop\ebedy\Ebedy.exe' has exited with code 0 (0x00000000).
```

We assigned the value of the expression (3+5) * 8 / 4 – 3 to the variable a that we created in our first example. When we compile our language, it printed the value of a as 13.

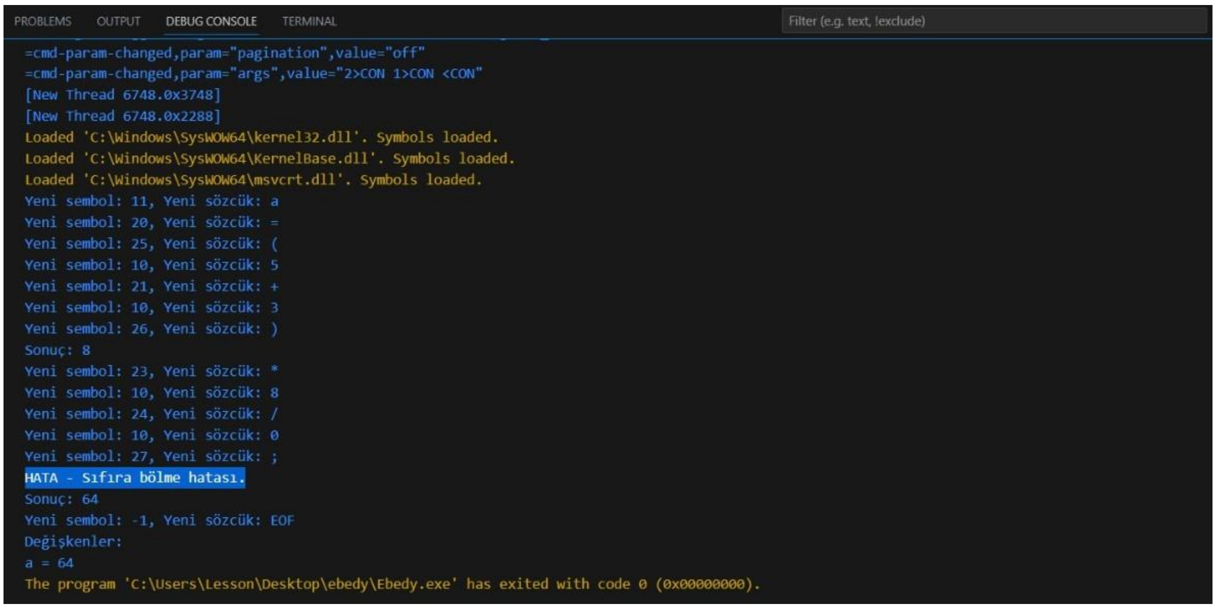
· example2.txt



```
Yeni sembol: 11, Yeni sözcük: a
Yeni sembol: 20, Yeni sözcük: =
Yeni sembol: 25, Yeni sözcük: (
Yeni sembol: 10, Yeni sözcük: 3
Yeni sembol: 22, Yeni sözcük: -
Yeni sembol: 10, Yeni sözcük: 1
Yeni sembol: 26, Yeni sözcük: )
Sonuç: 2
Yeni sembol: 23, Yeni sözcük: *
Yeni sembol: 10, Yeni sözcük: 4
Yeni sembol: 27, Yeni sözcük: ;
Sonuç: 8
Yeni sembol: 11, Yeni sözcük: b
Yeni sembol: 20, Yeni sözcük: =
Yeni sembol: 10, Yeni sözcük: 6
Yeni sembol: 21, Yeni sözcük: +
Yeni sembol: 10, Yeni sözcük: 4
Yeni sembol: 24, Yeni sözcük: /
Yeni sembol: 10, Yeni sözcük: 2
Yeni sembol: 27, Yeni sözcük: ;
Sonuç: 8
Yeni sembol: 11, Yeni sözcük: c
Yeni sembol: 20, Yeni sözcük: =
Yeni sembol: 11, Yeni sözcük: a
Yeni sembol: 21, Yeni sözcük: +
Yeni sembol: 11, Yeni sözcük: b
Yeni sembol: 27, Yeni sözcük: ;
Sonuç: 16
Yeni sembol: -1, Yeni sözcük: EOF
Değişkenler:
a = 8
b = 8
c = 16
```

We made assignments to the variables a and b that we created in the second example, and equated the sum of these two variables to variable c. When we compile our program, it printed a = 8, b = 8, and c = 16.

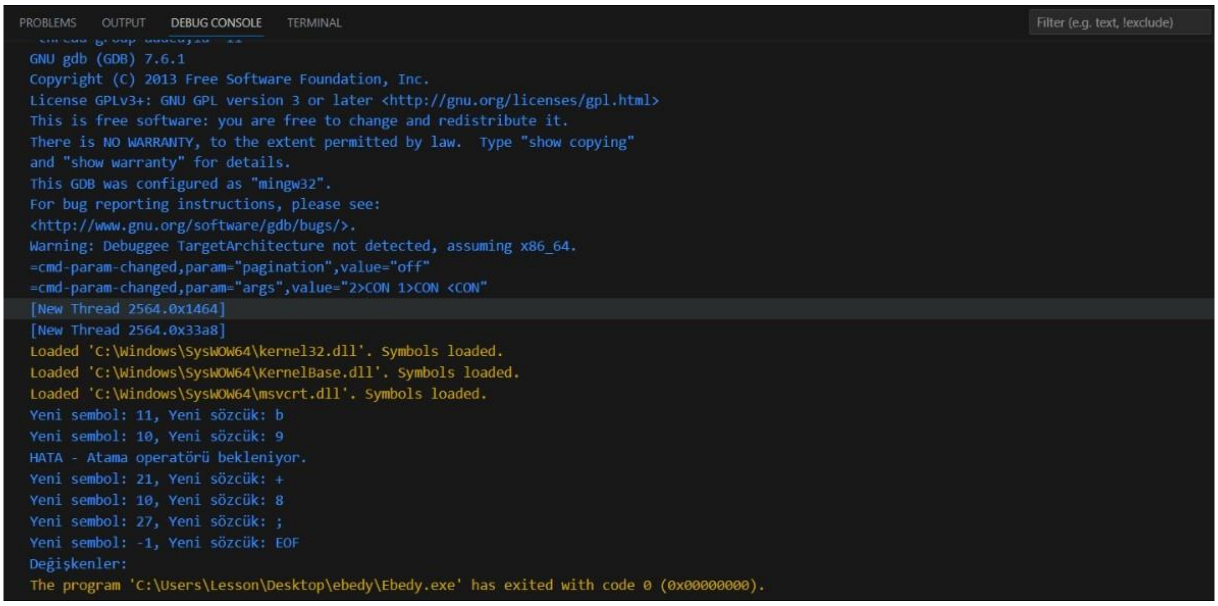
· example3.txt



```
=cmd-param-changed,param="pagination",value="off"
=cmd-param-changed,param="args",value="2>CON 1>CON <CON"
[New Thread 6748.0x3748]
[New Thread 6748.0x2288]
Loaded 'C:\Windows\SysWOW64\kernel32.dll'. Symbols loaded.
Loaded 'C:\Windows\SysWOW64\KernelBase.dll'. Symbols loaded.
Loaded 'C:\Windows\SysWOW64\msvcrt.dll'. Symbols loaded.
Yeni sembol: 11, Yeni sözcük: a
Yeni sembol: 20, Yeni sözcük: =
Yeni sembol: 25, Yeni sözcük: (
Yeni sembol: 10, Yeni sözcük: 5
Yeni sembol: 21, Yeni sözcük: +
Yeni sembol: 10, Yeni sözcük: 3
Yeni sembol: 26, Yeni sözcük: )
Sonuç: 8
Yeni sembol: 23, Yeni sözcük: *
Yeni sembol: 10, Yeni sözcük: 8
Yeni sembol: 24, Yeni sözcük: /
Yeni sembol: 10, Yeni sözcük: 0
Yeni sembol: 27, Yeni sözcük: ;
HATA - Sıfıra bölme hatası.
Sonuç: 64
Yeni sembol: -1, Yeni sözcük: EOF
Değişkenler:
a = 64
The program 'C:\Users\Lesson\Desktop\ebedy\Ebedy.exe' has exited with code 0 (0x00000000).
```

In the third example, we assigned the result of (5+3)*8/0 to the variable a, but since an expression cannot be divided by 0, so " HATA – Sıfıra bölme hatası." prints.

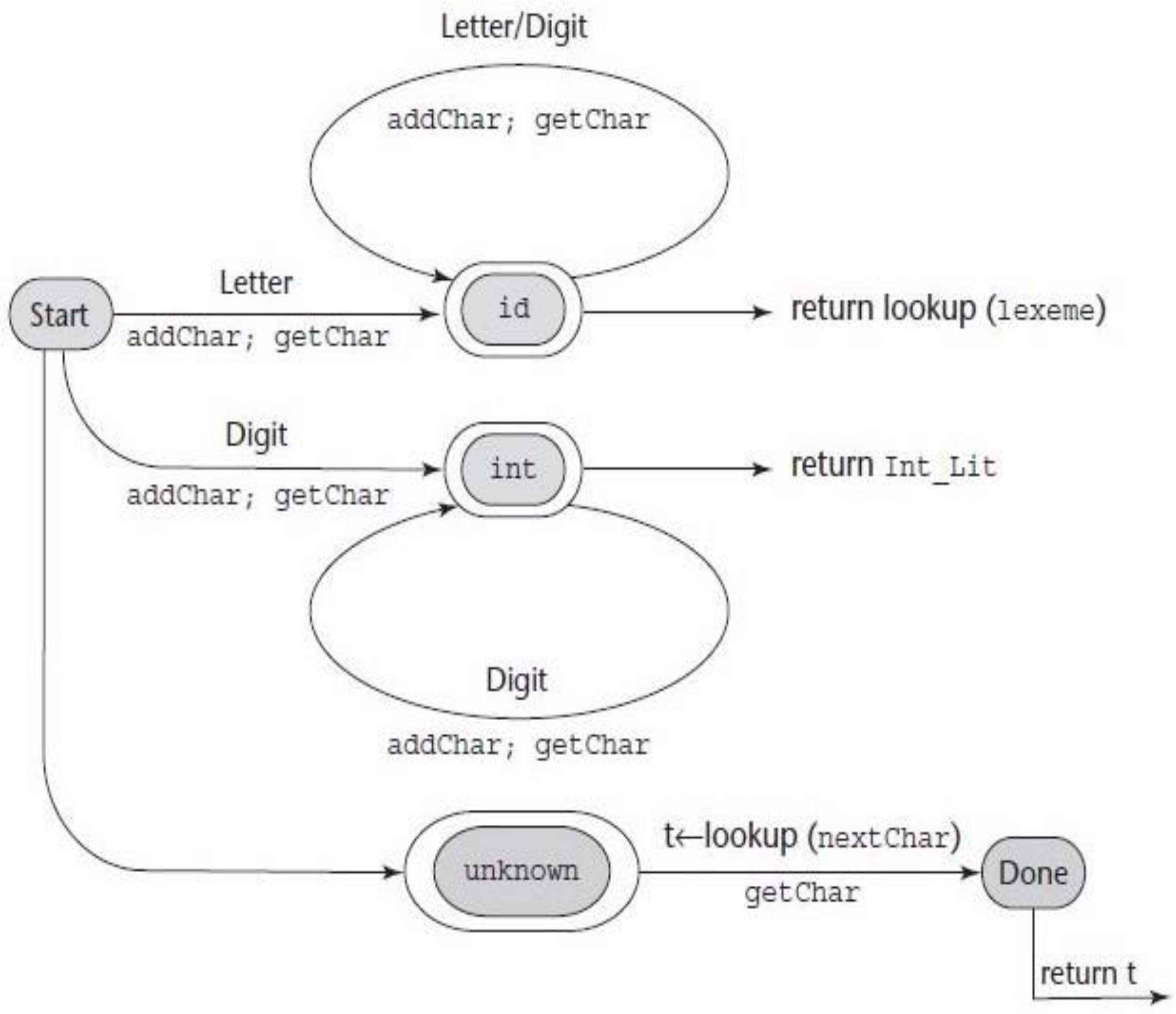
· example4.txt



```
GNU gdb (GDB) 7.6.1
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "mingw32".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Warning: Debuggee TargetArchitecture not detected, assuming x86_64.
=cmd-param-changed,param="pagination",value="off"
=cmd-param-changed,param="args",value="2>CON 1>CON <CON"
[New Thread 2564.0x1464]
[New Thread 2564.0x33a8]
Loaded 'C:\Windows\SysWOW64\kernel32.dll'. Symbols loaded.
Loaded 'C:\Windows\SysWOW64\KernelBase.dll'. Symbols loaded.
Loaded 'C:\Windows\SysWOW64\msvcrt.dll'. Symbols loaded.
Yeni sembol: 11, Yeni sözcük: b
Yeni sembol: 10, Yeni sözcük: 9
HATA - Atama operatörü bekleniyor.
Yeni sembol: 21, Yeni sözcük: +
Yeni sembol: 10, Yeni sözcük: 8
Yeni sembol: 27, Yeni sözcük: ;
Yeni sembol: -1, Yeni sözcük: EOF
Değişkenler:
The program 'C:\Users\Lesson\Desktop\ebedy\Ebedy.exe' has exited with code 0 (0x00000000).
```

In our last example, we wrote b 9+8; in front.in file and wanted to observe the result. Since the assignment to the variable b was attempted without using the assignment operator, it gave us an error in this part as well.

State Diagram



State diagram is a modelling tool used in programming languages to visually represent the behaviour of state-based systems. We made our language functional by using tokens in Ebedy, which we wrote in C, and by defining functions in C. The tokens we used in the state diagram of this language and the relationship between tokens and functions are shown in the state diagram by using addChar(), lookup(), getChar() functions. What it returns after each function is also included in the state diagram.

Sources

https://dev.to/jasonsbarr/how-to-create-your-own-programming-language-2642

https://www.w3schools.com/c/

https://www.cs.umd.edu/class/spring2017/cmsc430/slides/03-lexing-parsing.pdf