İçindekiler

[Introduction 2](#_Toc165667546)

[Requirements 3](#_Toc165667547)

[1)Identifiers 3](#_Toc165667548)

[2)Integer Constants 5](#_Toc165667549)

[3)Operators 6](#_Toc165667550)

[4)Brackets 11](#_Toc165667551)

[5)String Constants 12](#_Toc165667552)

[6)Keywords 13](#_Toc165667553)

[7)End Of Line 13](#_Toc165667554)

[8)Comma 13](#_Toc165667555)

[9)Comments 14](#_Toc165667556)

[Main Method 15](#_Toc165667557)

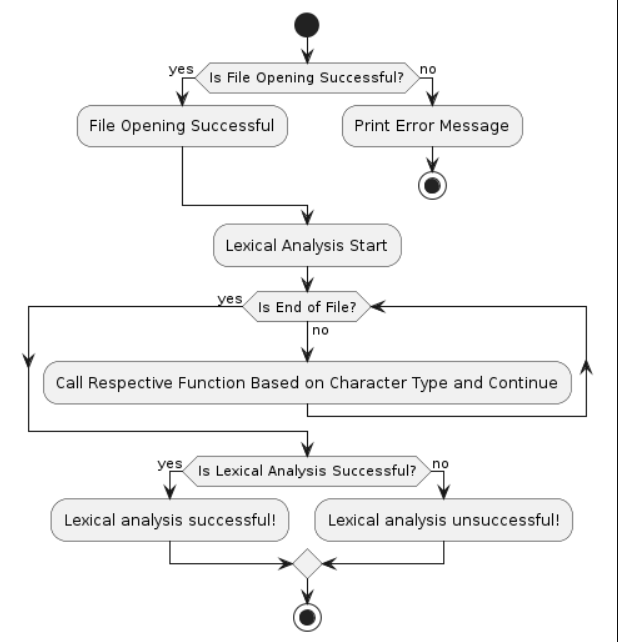
[Examples and Outputs 17](#_Toc165667558)

# Introduction and Flowchart

STAR Programming Language, a small scripting language designed for basic arithmetic operations. One of the primary objectives of the STAR language is to perform fundamental arithmetic calculations. The language supports only two data types: integers and strings. Integers are constrained to positive values and default to zero when forced to take a negative value. Assignments in STAR are limited to arithmetic expressions with only two operands, without support for complex expressions.

This report is prepared to elucidate the definition, features, and applications of the STAR language. It covers the syntax, data types, assignment operations, and limitations of the language. Furthermore, it highlights the primary objectives and usage scenarios of the language.

Beginning with a general overview of the STAR Programming Language, we will delve into its features and constraints in detail.



# Requirements

## 1)Identifiers

metin, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

This function will read a string of characters and separate the identifiers to indicate them. inputFile and outputFile files and get the ch start character.

char \*word = (char \*) malloc(sizeof(char) \* (10 + 1));

Memory is allocated for a character string called word. This string covers the maximum length of the identifier (10 characters) plus 11 characters to include a null character.

if (word == NULL) {

        printf("Memory allocation failed! (at line %zu)\n", lineNo);

        closeFilesAndExit(inputFile, outputFile, 1);

    }

It is checked whether the memory allocation failed. If memory cannot be allocated for the word, an error message is printed and the program is terminated by closing the files.

size\_t size = 0;

A variable named Size is defined and assigned a value of 0. This variable tracks the position of the read characters in the sequence.

while (ch != EOF && (isalnum(ch) || ch == '\_')) { //isalnum -harf veya rakam olmasını kontrol eder.

A cycle is started. This loop continues if the ch character is not the end of file (EOF) and the character is a letter, number (isalnum() works well), or underscore (\_).

if (size > 9) {

            printf("Identifier can't be longer than 10 characters! (at line %zu)\n", lineNo);

            closeFilesAndExit(inputFile, outputFile, 1);

        }

If the identifier is longer than 10 characters, an error message is printed and the program is terminated.

word[size++] = tolower(ch);

The ch character is converted to lowercase and assigned to the Word array and incremented to you.

ch = fgetc(inputFile);

The next character is read from the inputFile file and assigned to ch.

word[size] = '\0';

When the reading process is completed, a null character is added to the end of the word string. (The null character ('\0') is used to indicate the end of a C-style (null-terminated) string. Strings of characters are stored sequentially in memory, and the null character is used to indicate where the end of the string is.)

if (checkIfKeyword(word)) fprintf(outputFile, "Keyword(%s)\n", word);

If the word is a keyword, it is written to the outputFile file as "Keyword(word)". (check if keyword method)

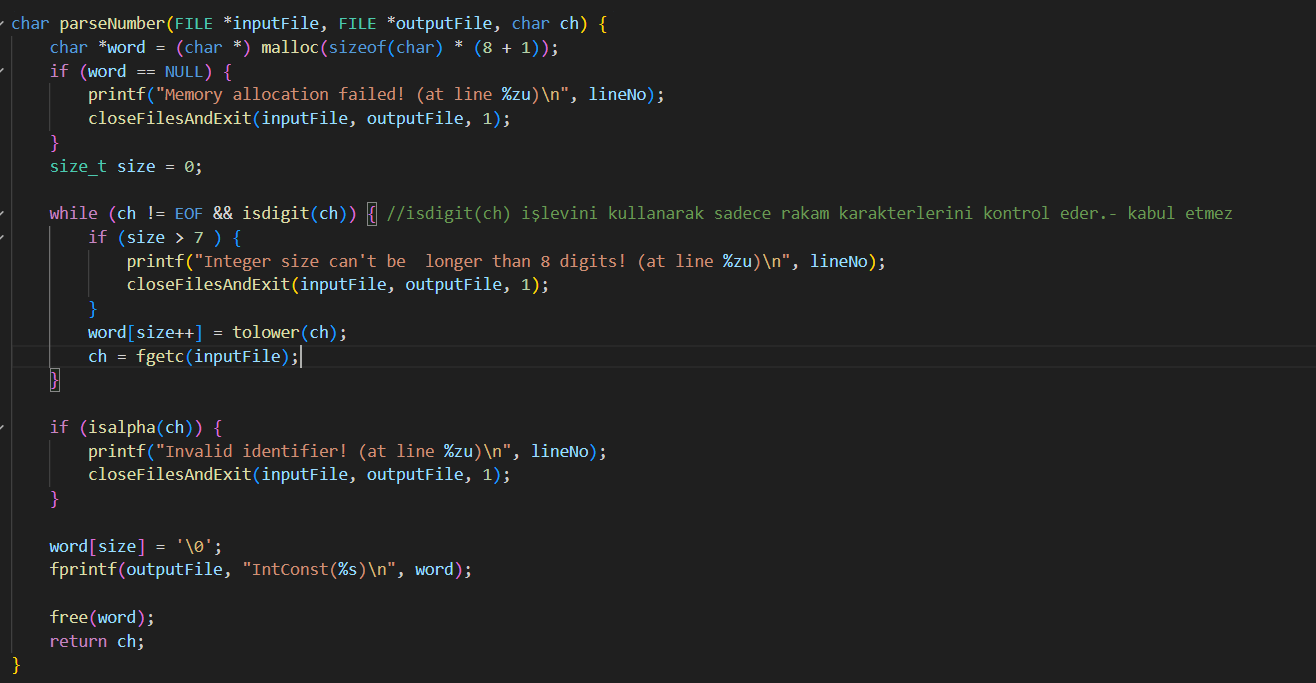
 else fprintf(outputFile, "Identifier(%s)\n", word);

If the word is not a keyword, it is written to the outputFile as "Identifier(word)".

free(word);

Once the memory is allocated, the memory allocated for the residual word is released.

## 2)Integer Constants



This function reads a character from a file, finds an integer constant, and writes this constant and its type (IntConst) to the output file. While the function checks whether the characters are a numeric constant, it gives an error message and closes the program when it encounters invalid characters.

char \*word = (char \*) malloc(sizeof(char) \* (8 + 1));

Memory is allocated for a character array called word. This array has the maximum length of the defined number (8 characters) and space for 9 characters to insert a null character.

if (word == NULL) {

        printf("Memory allocation failed! (at line %zu)\n", lineNo);

        closeFilesAndExit(inputFile, outputFile, 1);

    }

If memory allocation fails, it is checked. If memory allocation fails, the error message is printed and the program is terminated.

size\_t size = 0;

while (ch != EOF && isdigit(ch)) { //isdigit(ch) işlevini kullanarak sadece rakam karakterlerini kontrol eder.- kabul etmez

        if (size > 7 ) {

            printf("Integer size can't be  longer than 8 digits! (at line %zu)\n", lineNo);

            closeFilesAndExit(inputFile, outputFile, 1);

        }

        word[size++] = tolower(ch);

        ch = fgetc(inputFile);

    }

A variable named size is defined and assigned the value 0. This variable keeps track of the position of read characters in the array. A cycle is started. This loop continues as long as the ch character is not the end of file (EOF) and the character is a digit. If the number is longer than 8 characters, an error message is printed and the program is terminated. The number read is converted to lowercase, assigned to the word array and incremented to size. The next character is read from inputFile.

if (isalpha(ch)) {

        printf("Invalid identifier! (at line %zu)\n", lineNo);

        closeFilesAndExit(inputFile, outputFile, 1);

    }

If a letter is read, it is an invalid identifier and an error message is printed and the program is terminated.

word[size] = '\0';

    fprintf(outputFile, "IntConst(%s)\n", word);

    free(word);

    return ch;

When the reading process is completed, a null character (\0) is added to the end of the word string. The found number is written to the outputFile file in "IntConst(number)" format. The memory allocated for the word array is released as it will no longer use the memory block.

## 3)Operators

int checkIfOperator(char ch) {  //operatorleri tanımlar

    return ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == ':';

}

Operators are defined.

char parseOperator(FILE \*inputFile, FILE \*outputFile, char ch) {

    char \*operatorString = (char \*) malloc(sizeof(char) \* (2 + 1)); //malloc, "memory allocation,bellekte yer ayırdık.

    if (operatorString == NULL) {

        printf("Memory allocation failed! (at line %zu)\n", lineNo);

        closeFilesAndExit(inputFile, outputFile, 1);

    }

Memory is allocated for a character array called operatorString. This array has space for the maximum length of operators (2 characters) and 3 characters to insert a null character. If memory allocation fails, it is checked. If memory allocation fails, the error message is printed and the program is terminated.

char first = ch, next = '\0'; // İlk karakteri first değişkenine atar ve operatorString dizisinin ilk elemanına kaydeder.

    operatorString[0] = first;  //

Two character variables named first and next are defined. first, the first character passed to the function, is assigned to ch. The next variable is defined temporarily. The first variable is assigned to the first element of the operatorString array.

if (first == ':') {

        ch = fgetc(inputFile);

        if (ch != '=') { //sıradaki karakter = mı bakılır değilse hata mesajı verir

            printf("Invalid operator ':' ! (at line %zu)\n", lineNo);

            closeFilesAndExit(inputFile, outputFile, 1);

        } else {

            operatorString[1] = ch;  // = ise sıraya eklenir

            operatorString[2] = '\0'; //sona null kondu

        }

        next = fgetc(inputFile); //sıradaki ch ile devam edildi

        if (checkIfOperator(next)) { //next operator değil ise hata verir

            printf("Invalid operator ' %c%c ' ! (at line %zu)\n", first, next, lineNo);

            closeFilesAndExit(inputFile, outputFile, 1);

        }

    }

If the first character is : a conditional block is initialized. The next character is read from the inputFile file. If the next character = not, the error message is printed and the program is terminated. Otherwise, the second element of the operatorString array is assigned the = character and null is appended to the end.

The next character is read from the inputFile file. If the next character is an operator, the error message is printed and the program is terminated.

else if (first == '/') {

        next = fgetc(inputFile);

        if (next == '\*') { //yorum satırı bulundu

            ch = fgetc(inputFile);

            while (ch != EOF) { // dosya sonuna kadar devam eder

                if (ch == '\n') lineNo++; //satır sayısını arttırır

                if (ch == '\*') { // \* işaretini görünce yorum satırının bitmesi gerekir

                    ch = fgetc(inputFile);

                    if (ch == '/') { // \* dan sonra / gelirse yorum satırı biter

                        ch = fgetc(inputFile);

                        return ch;

                    }

                } else ch = fgetc(inputFile);

            }

            printf("End of file before terminating comment! (at line %zu)\n", lineNo); //yorum satırı kapanmadan dosya bitti

            closeFilesAndExit(inputFile, outputFile, 1);

        } else if (checkIfOperator(next)) {

            printf("Invalid operator ' %c%c ' ! (at line %zu)\n", first, next, lineNo);

            closeFilesAndExit(inputFile, outputFile, 1);

        } else

            operatorString[1] = '\0';

    }

If the first character is /, a conditional block is initialized./ operator is a division operator or sign that starts the comment line. Reads the second character (next) from the file. If the second character is \*, a condition block is initialized. This indicates that a comment line has started. Because the comment line begins, it reads the next character (ch) from the file. A while loop is started and continues until the end of the file. If the character being read is a line break character (\n), increments the line number. This is used to keep track of the number of lines within the comment line. If the character read is \*, a condition block is initialized. This status is checked so that the comment line ends. Reads the next character (ch) from the file. If the character after the \* character is /, a conditional block is initialized. This status indicates that the comment line has ended. Because the comment line ends, it reads the next character (ch) from the file. Because the comment line ends, it returns the next character to the calling code block. This allows you to continue the process where the comment line ends. An error message is printed indicating that the file has ended before the comment line is terminated. This means that the expected comment line is not closed and requires the program to be terminated. If the second character is an operator, a conditional block is initialized. This indicates an invalid combination of operators and requires the program to be terminated. If the second character is not an operator, the second character of the operatorString array is assigned with '\0' (null). This indicates that the comment line operator is a single-character operator with /.

else if (first == '+') {

        ch = fgetc(inputFile);

        if (ch == '+') { // ++ bulundu

            operatorString[1] = ch;

            operatorString[2] = '\0';

            next = fgetc(inputFile);

            if (checkIfOperator(next)) {

                printf("Invalid operator ' %c%c ' ! (at line %zu)\n", first, next, lineNo);

                closeFilesAndExit(inputFile, outputFile, 1);

            }

        } else {

            operatorString[1] = '\0';

            next = ch;

            if (checkIfOperator(next)) {

                printf("Invalid operator ' %c%c ' ! (at line %zu)\n", first, next, lineNo);

                closeFilesAndExit(inputFile, outputFile, 1);

            }

        }

    }

Reads the next character (ch) from the file. If the next character is '+', it initializes a condition block, indicating the presence of the increment operator. The second character ('+') is then assigned to the second element of the operatorString array. Subsequently, the third element of the operatorString array is assigned the null character ('\0'), signaling the termination of the operatorString array.

Then, it reads the next character from the file. This step is crucial to check the character following the boost operator. If the next character is an operator, it initializes a block of conditions, marking an invalid combination of operators, necessitating termination of the program. If the next character is not an operator, an 'else' block is initialized.

Within the 'else' block, if the next character is not an operator, it assigns the second element of the operatorString array ('+') with the null character ('\0'), indicating that the increment operator is single-character. The subsequent character is then assigned to 'ch', as the increment operator is a single character.

If the next character is an operator, it initializes a block of conditions, indicating an invalid combination of operators, and requiring the program to be terminated.

else if (first == '-') {

        ch = fgetc(inputFile);

        if (ch == '-') { // -- bulundu

            operatorString[1] = ch;

            operatorString[2] = '\0';

            next = fgetc(inputFile);

            if (checkIfOperator(next)) {

                printf("Invalid operator ' %c%c ' ! (at line %zu)\n", first, next, lineNo);

                closeFilesAndExit(inputFile, outputFile, 1);

            }

        } else if (isdigit(ch) != 0) { // -li ifade girişi engellendi

            printf("Negative values are not allowed! (at line %zu)\n", lineNo);

            closeFilesAndExit(inputFile, outputFile, 1);

        } else {

            next = ch;

            if (checkIfOperator(next)) {

                printf("Invalid operator ' %c%c ' ! (at line %zu)\n", first, next, lineNo);

                closeFilesAndExit(inputFile, outputFile, 1);

            }

            operatorString[1] = '\0';

        }

    }

If the next character is '-', it initializes a block of conditions, indicating the presence of the reduction operator. The second character ('-') is then assigned to the second element of the operatorString array. Subsequently, the third element of the operatorString array is assigned the null character ('\0'), indicating the termination of the operatorString array.

Then, it reads the next character from the file. After reading the subsequent character of the reduction operator, it checks if the next character is an operator. If it is, it initializes a block of conditions, indicating an invalid combination of operators, necessitating the termination of the program.

If the next character is a number, it initializes an 'else if' block. This indicates that negative values are invalid and requires the program to be terminated. The isdigit() function is used for this check. It prints an error message to prevent the use of negative values.

If the next character is neither an operator nor a number, it initializes an 'else' block. The next character is then assigned to 'ch', as the reduction operator is a single character.

If the next character is an operator, it initializes a block of conditions, indicating an invalid combination of operators, and requiring the program to be terminated.

else {

        next = fgetc(inputFile);

        if (checkIfOperator(next)) {

            printf("Invalid operator ' %c%c ' ! (at line %zu)\n", first, next, lineNo);

            closeFilesAndExit(inputFile, outputFile, 1);

        }

        operatorString[1] = '\0';

    }

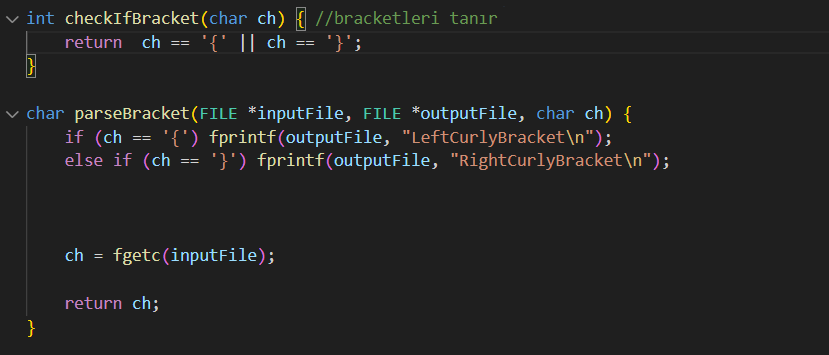
    fprintf(outputFile, "Operator(%s)\n", operatorString);

    free(operatorString);

    return next;

If none of the previous conditions are met, that is, if the operator is a single character, an else block is started. The next character (next) is read from the file. This step is necessary for processing single-character operators. If the next character is an operator, a condition block is initialized. This indicates an invalid operator combination and requires the program to be terminated. An error message is printed when there is an invalid operator combination. In case of error, the files are closed and the program is terminated. If the next character is not an operator, the second element of the operatorString array is marked with the null character (\0). It prints the type and value of the operator to the outputFile file. The memory space containing the operator's information is now freed.

## 4)Brackets



## checkIfBracket(): This function checks whether the character it receives as a parameter is a bracket (or curly brace). The function uses a logical expression to determine whether the given character is a curly brace: ch == '{' || ch == '}'. If the character is a curly brace, the function returns 1; otherwise it returns 0. parseBracket(): This function is used to parse and process curly brackets ({}). First, it is checked whether the incoming character is a curly bracket. If the character is {, "LeftCurlyBracket" is written to the outputFile file. If the character is }, it is written to the outputFile file. "RightCurlyBracket" is written. Then, the next character is read with the fgetc(inputFile) operation. Finally, the next character (ch) is returned to the calling code block. This allows the next character to be read.

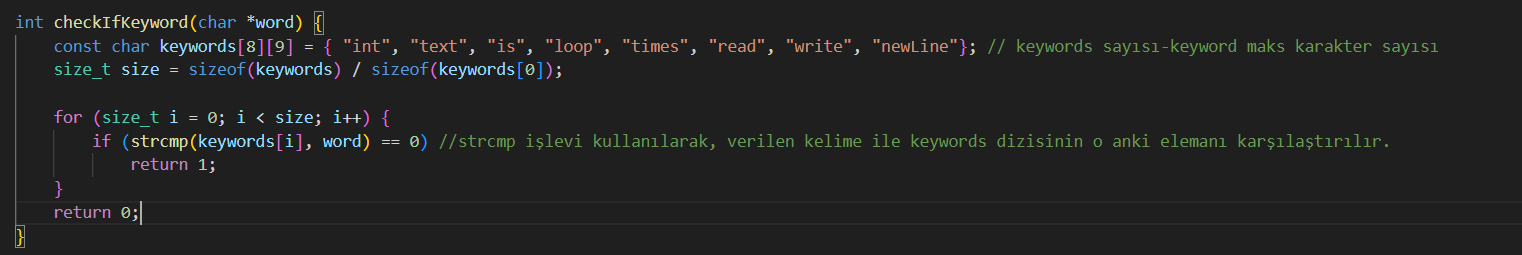
## 5)String Constants

metin, ekran görüntüsü içeren bir resim

Açıklama otomatik olarak oluşturuldu

First, a character is read from inputFile and assigned to the ch variable. This may be the last character before the function is called, or it may be the next character from a previous function call. Memory is allocated for a string constant. A maximum string constant of 256 characters is allowed. A place is reserved for the last character. If stringConstant is equal to NULL, memory allocation has failed and the program is closed with an error message. A variable named size is defined and assigned a value of 0. This variable tracks the position of the read characters in the array. A loop is started. This loop continues until it reaches the end of the file or the double quotation mark character (""). If the character read is a newline character (\n), the lineNo value is incremented by one. The read character is added to the stringConstant array and incremented by one. The next character is read with fgetc(inputFile) and assigned to the ch variable. A null character (\0) is added to the end of the string constant. This is used in the C language to indicate the end of strings. If the end of the file is reached before the loop ends with a double quote, an error message is printed and the program is closed. The next character is read from the file using fgetc(inputFile) and assigned to the ch variable. The read string constant, "stringConstant (...)" format is written to the outputFile file. Memory that is no longer used is freed.

## 6)Keywords



A 2D character array named keywords is created. Each keyword is an array element. A total of 8 keywords and a maximum of 8 characters per word are allowed. (The last character is reserved for the null character.) A variable named size is created and the number of elements of the keywords array is assigned to this variable. To do this, the total memory size of the array is divided by the memory size of an array element. A loop is started. This loop checks all elements of the keywords array. Using the strcmp function, the word string is compared with the word at index i of the keywords array. If the two words are equal (that is, strcmp returns 0), this word is considered a keyword and the function returns 1. If the loop terminates and no matches are found, the function returns 0. This indicates that the checked word is not a keyword.

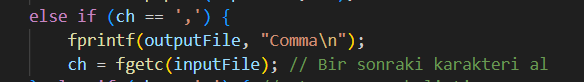
## 7)End Of Line

metin, ekran görüntüsü, yazı tipi içeren bir resim

Açıklama otomatik olarak oluşturuldu

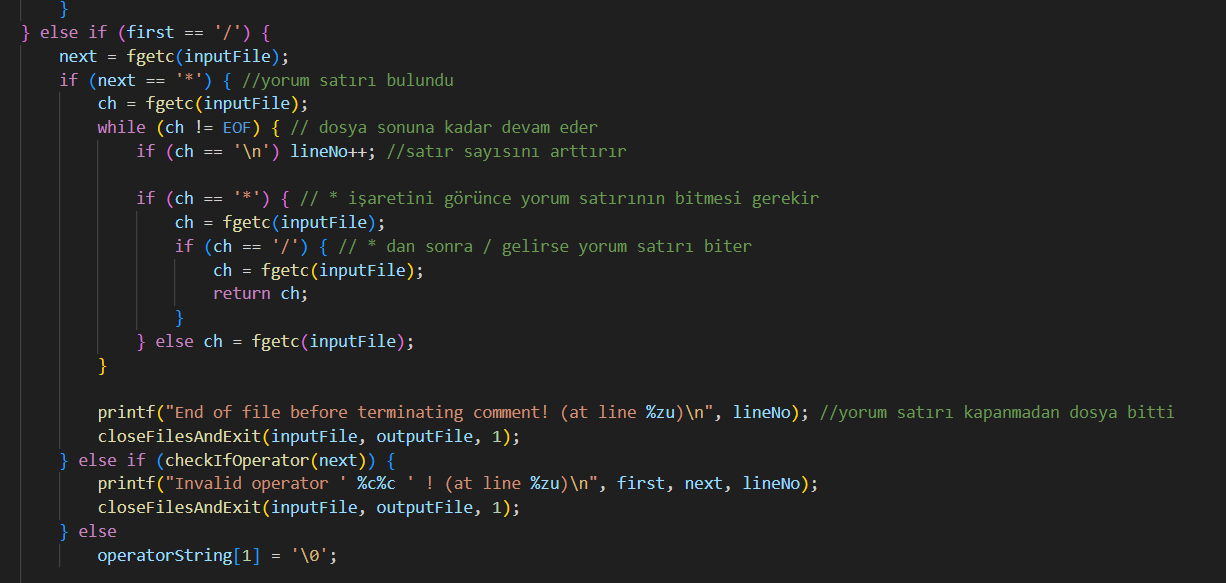
## If the character in the input file is equal to a period sign, that is, if this indicates a line end, we perform the following steps: The text "EndOfLine" is written to the output file. This represents a line break. It reads the next character from inputFile and assigns it to the ch variable. This allows the next character to be read and advances processing to the next character.

## 8)Comma



## If the character in the input file equals a comma, "comma" is printed.

## 9)Comments



First, the if (first == '/') condition checks that the current character is /. If so, the next character is expected to be read using fgetc(inputFile). If the next character is \*, we know that it is the beginning of the comment line. In this case, we look for the comment line starting with /\* with the nested loop. We read any character between /\* and \*/ within the comment line. If at any time we see an end-of-line \n character, we keep track of which line we are on by incrementing the lineNo variable. If the character read is \*, we check if the next character is /. If the / character is found, the comment line ends and we exit the loop and read the next character. If the next character is not \* and is not an operator, we understand that the comment line is incomplete and print an error message and terminate the program. If the next character is an operator, we specify that the comment line should not end with the operator and we terminate the program by printing an error message. If no condition is met, we indicate that the comment line continues with the next character and complete the operation by terminating the operatorString[1] element with '\0'.

# Main Method

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

    char \*fileName = "code.sta";

    FILE \*inputFile = fopen(fileName, "r");

     if (inputFile == NULL) {

        printf("Can't open the input file!\n");

        return 1;

    }

    FILE \*outputFile = fopen("code.lex", "w"); //yeni bir output fileı oluşturur

    if (outputFile == NULL) {

        printf("Can't create the output file!\n");

        fclose(inputFile);

        return 1;

    }

    char ch = fgetc(inputFile);

   while (ch != EOF) {

        if (isalpha(ch)) //alfabeden bir harf mi kontrol edildi

            ch = parseWord(inputFile, outputFile, ch);

        else if (isdigit(ch))

            ch = parseNumber(inputFile, outputFile, ch);

        else if (checkIfOperator(ch))

            ch = parseOperator(inputFile, outputFile, ch);

        else if (ch == '"')

            ch = parseStringConstant(inputFile, outputFile, ch);

        else if (checkIfBracket(ch))

            ch = parseBracket(inputFile, outputFile, ch);

        else if (checkIfSpace(ch))

            ch = skipSpace(inputFile, ch);

        else if (ch == ',') {

            fprintf(outputFile, "Comma\n");

            ch = fgetc(inputFile); // Bir sonraki karakteri al

        } else if (ch == '.') { //satır sonunu belirtir

            fprintf(outputFile, "EndOfLine\n");

            ch = fgetc(inputFile);

        } else {

            printf("Invalid character ' %c ' ! (at line %zu)\n", ch, lineNo);

            closeFilesAndExit(inputFile, outputFile, ch);

        }

    }

    closeFilesAndExit(inputFile, outputFile, 0);

}

A file named code.sta is read. If the file cannot be opened successfully, an error message is printed and the program terminates with 1. An output file named code.lex is created to write the lexic analysis results. If the output file cannot be created, an error message is printed, the input file is closed, and the program terminates with 1. One character (ch) is read. A loop is started until the character reaches the end of the file. At each loop step: If the character is a letter (checked with isalpha(ch)), the parseWord function is called and the next case of the character (ch) is returned. If the character is a number (checked with isdigit(ch)), the parseNumber function is called and the next state of the character (ch) is returned. If the character is an operator (checked with checkIfOperator(ch)), the parseOperator function is called and the character's next state (ch) is returned. If the character is a double quote (checked with ch == '"'), the parseStringConstant function is called and the next state (ch) of the character is returned. If the character is a bracket (checked with checkIfBracket(ch)), the parseBracket function is called and the next state of the character (ch) is returned. If the character is a space (checked with checkIfSpace(ch)), the skipSpace function is called and the next state of the character (ch) is returned. If the character is a comma (checked with ch == ','), "Comma" is written to the output file, the next character is taken (ch = fgetc(inputFile)) and the loop continues. If the character is a period (checked with ch == '.'), "EndOfLine" is written to the output file, the next character is taken (ch = fgetc(inputFile)) and the loop continues. If none of the above conditions are met, it is considered an invalid character, an error message is printed, and the program is terminated with the closeFilesAndExit function. When the loop finishes, the input and output files are closed and the program terminates successfully (closeFilesAndExit(inputFile, outputFile, 0)).

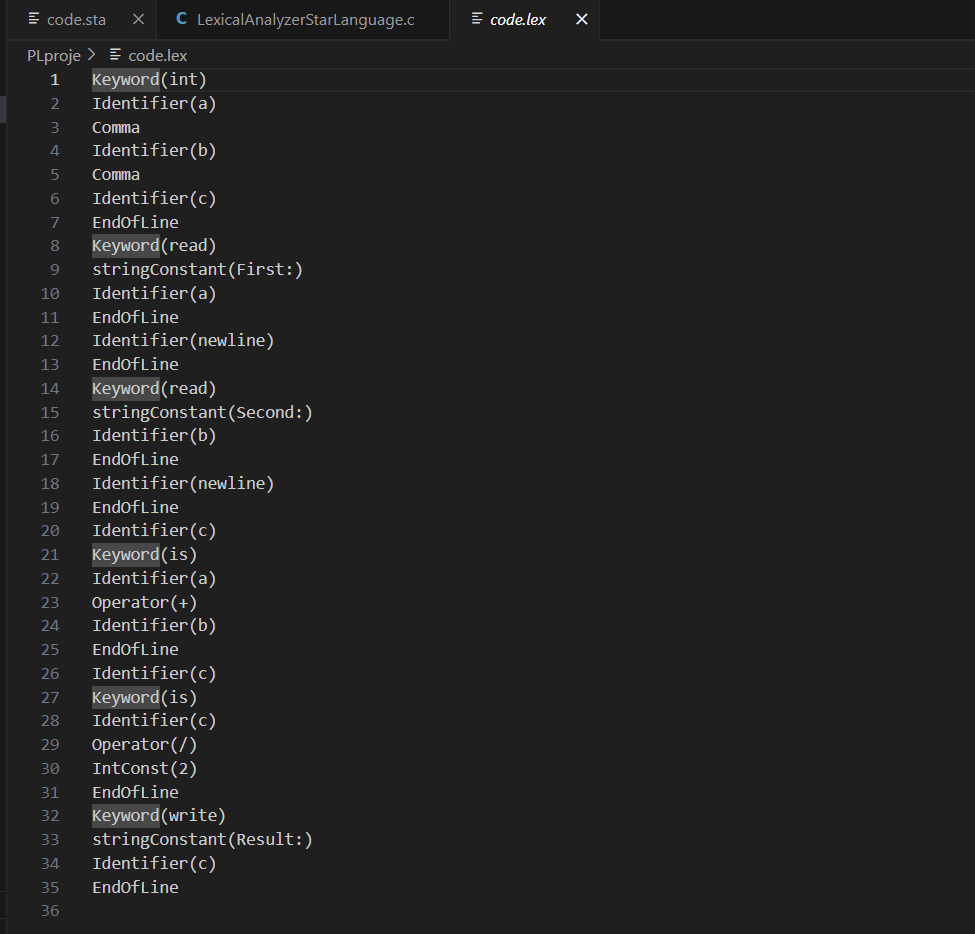
# Examples and Outputs

Code.sta

metin, ekran görüntüsü, ekran, görüntüleme, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Code.lex



code.sta with invalid inputs

1)

metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

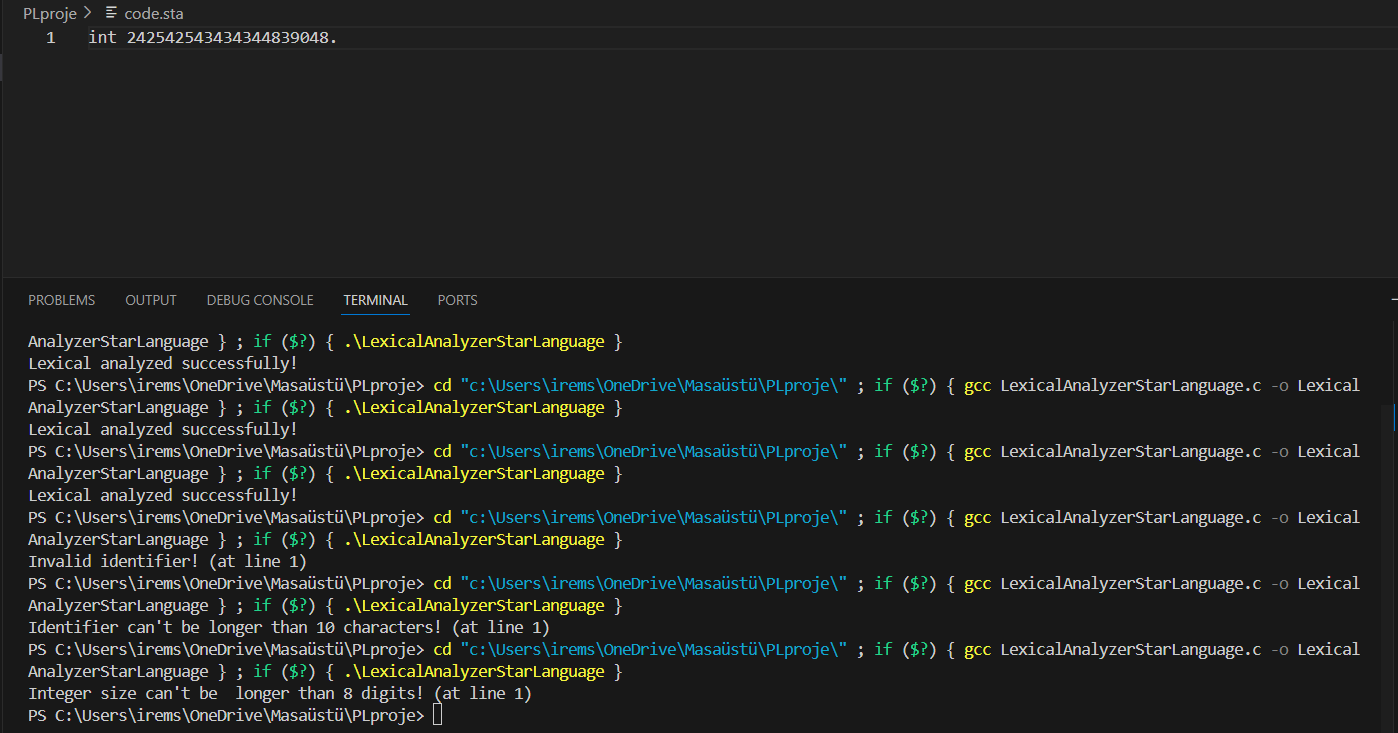
Açıklama otomatik olarak oluşturuldu

Identifiers start with an alphabetic character (a letter)

2) metin, ekran görüntüsü, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

Identifier can't be longer than 10 characters!.

3) 

Integer size can't be longer than 8 digits!

4) metin, ekran görüntüsü, multimedya yazılımı, yazılım içeren bir resim

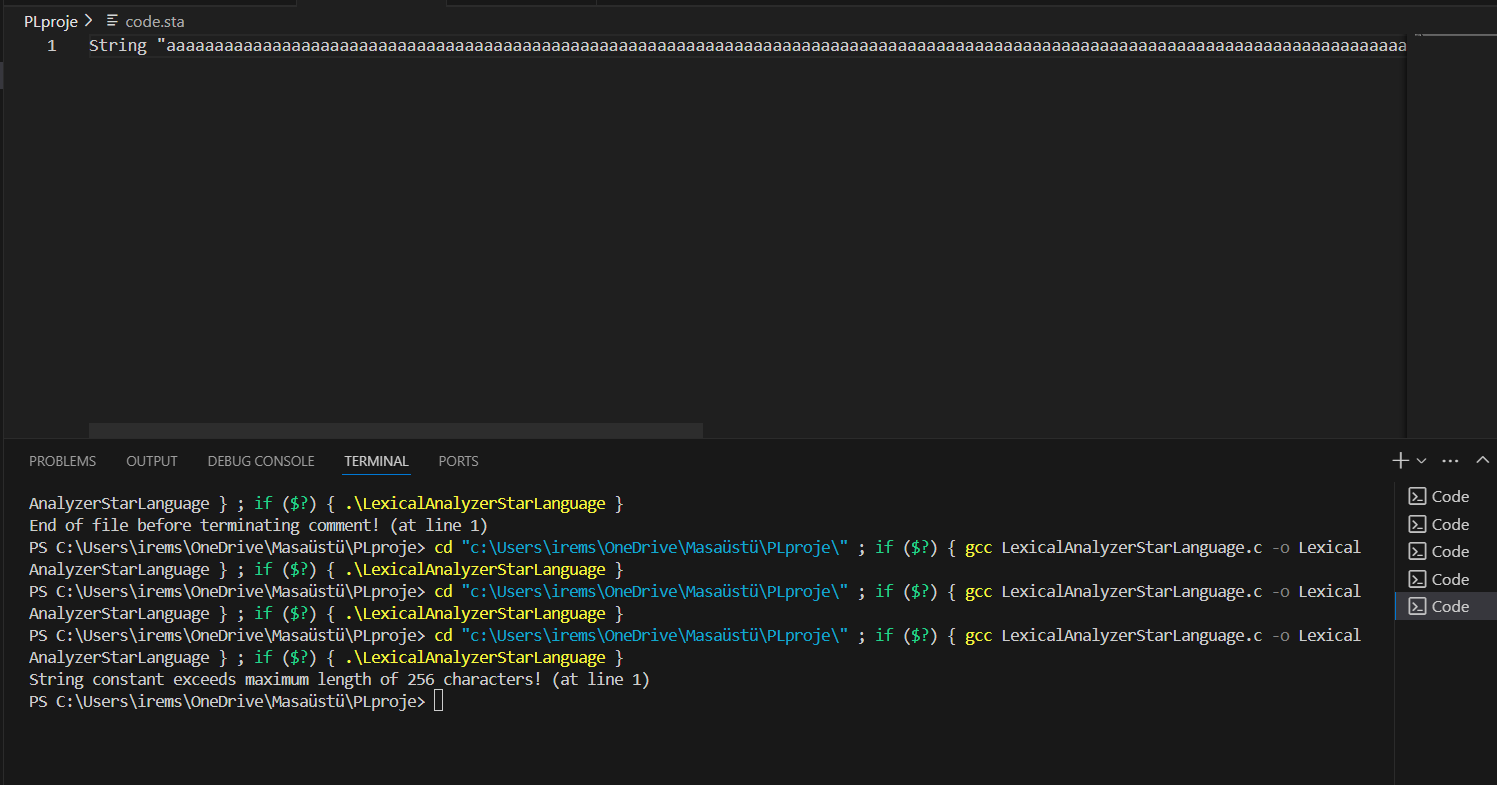
Açıklama otomatik olarak oluşturuldu

End of file before terminating string constant!

5) metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Açıklama otomatik olarak oluşturuldu

End of file before terminating comment!!

6) 

String constant exceeds 256 characters!!