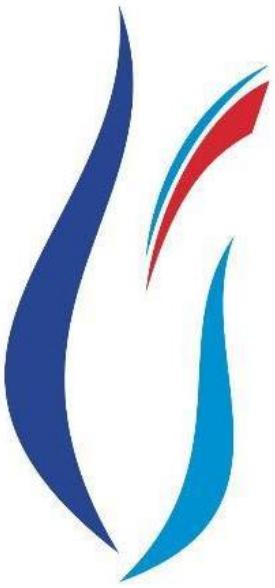


MANISA
CELAL BAYAR
UNIVERSITY

FACULTY OF ENGINEERING
COMPUTER ENGINEERING DEPARTMENT
2023-2024 SPRING
PROGRAMMING LANGUAGES
PROJECT

Furkan BULUT 210316011
Binnur SÖZTUTAR 210316084
Barış KOÇ 210316042
Evening Class

INTRODUCTİON

In this project, we designed a special programming language that includes Turkish keywords and structures. The main purpose of this language is to show how programming language concepts can be adapted to specific linguistic conditions. This report summarizes the language's grammar in BNF format, includes state diagrams of important language constructs, and provides sample code examples. In addition, the source code for word analysis and parsing was changed to adapt to Turkish letters and keywords, providing syntax and semantic information appropriate to the designed language. C was used to implement it.

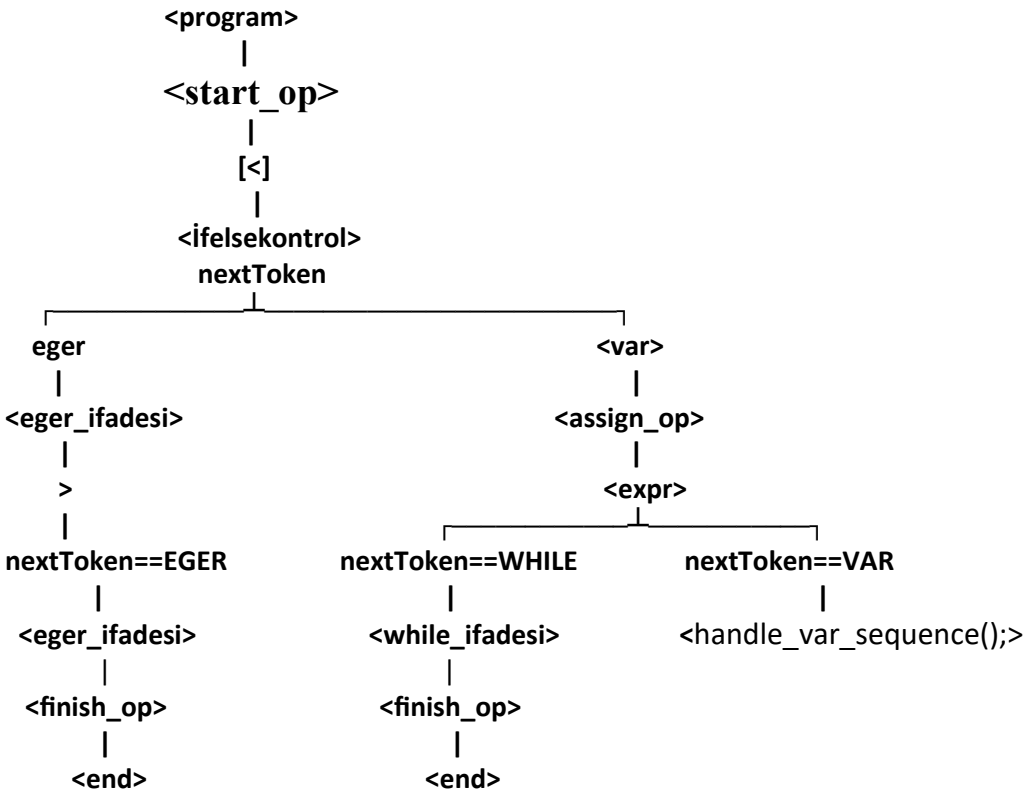
Properties of Languages :

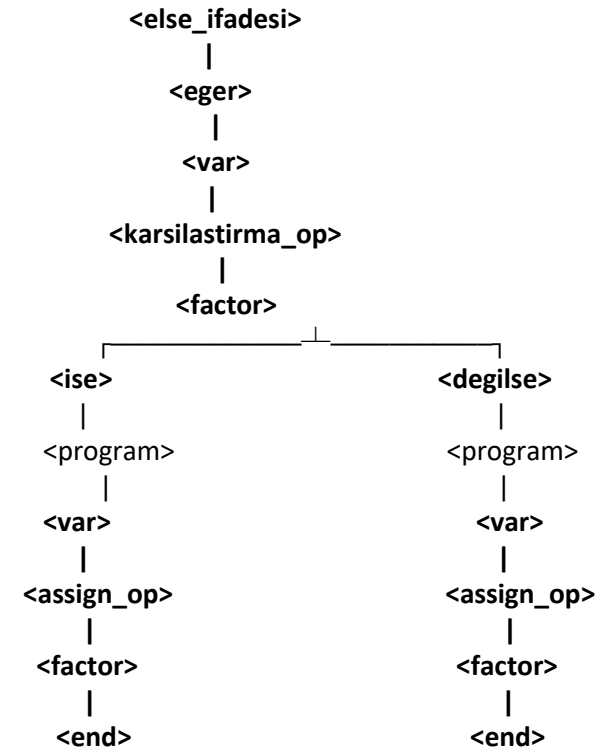
BNF Structure of Language :

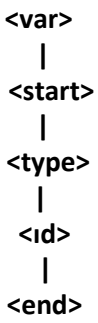
<program> -> <start_op> <else_ifadesi> <finish_op> | <start_op> <var> <assign_op> <expr> <finish_op> | <start_op> <while_ifadesi> <finish_op>
<else_ifadesi> -> <eger> <var> <karsilastirma_op> <factor><ise> <var> <assign_op> <factor> <degilse> <var> <assign_op> <factor>
<start_op> -> <
<finish_op> -> >
<var> -> <type> <id>
<type> -> int
<id> -> <letter> <id2>
<id2> -> <letter> <id2> | <digit> <id2> | ε
<expr> -> <term> - <expr> | <term> + <expr> | <term>
<term> -> <factor> / <term> | <factor> * <term> | <factor>
<factor> -> <digit> <factor> | <digit>
<digit> -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<letter> -> a | b | c | ç | d | e | f | g | ? | h | ? | i | j | k | l | m | n | o | ö | p | r | s | ? | t | u | ü | v | y | z
<assign_op> -> =
<ise>-> ise
<degilse> -> degilse
<eger> -> eger
<karsilastirma_op> -> < | > | <= | >= | == | !=
<while_ifadesi> -> <while_op> <var> <karsilastirma_op> <factor> <do_op> <program>
<while_op> -> while
<do_op> -> do

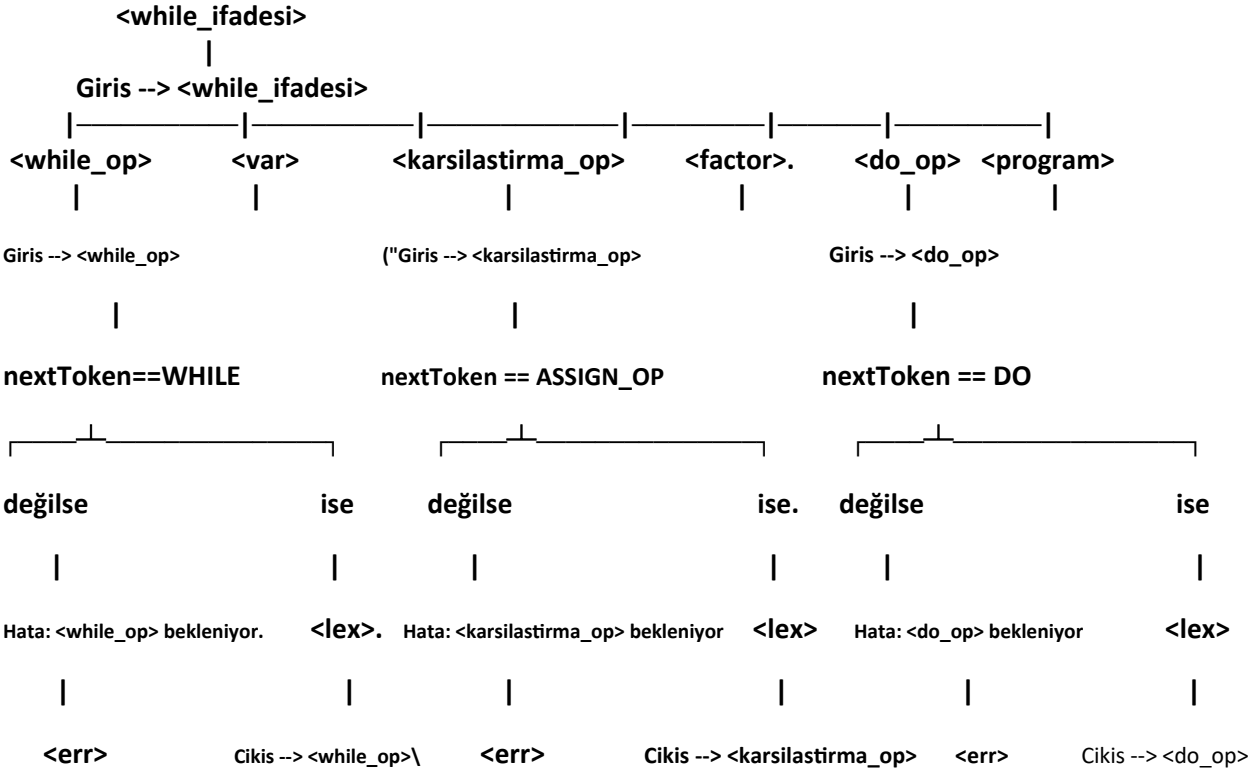
STATE DIAGRAM

STATE DIAGRAM




<else_ifadesi>
<eger>
<var>
<karsilastirma_op>
<factor>
<ise>
<program>
<var>
<assign_op>
<factor>
<end>
<degilse>
<program>
<var>
<assign_op>
<factor>
<end>


<var>
<start>
<type>
<ıd>
<end>


<while_ifadesi>
Giris --> <while_ifadesi>
<while_op>
<var>
<karsilastirma_op>
<factor>.
<do_op>
<program>
Giris --> <while_op>
("Giris --> <karsilastirma_op>
Giris --> <do_op>
nextToken==WHILE
nextToken == ASSIGN_OP
nextToken == DO
değilse
ise
değilse
ise.
değilse
ise
Hata: <while_op> bekleniyor.
<lex>.
Hata: <karsilastirma_op> bekleniyor
<lex>
Hata: <do_op> bekleniyor
<lex>
<err>
Cikis --> <while_op>\
<err>
Cikis --> <karsilastirma_op>
<err>
Cikis --> <do_op>

<LEFT_PAREN>: This node represents a left parenthesis symbol "(".

<RIGHT_PAREN>: This node represents a right parenthesis symbol ")".

<ADD_OP>: This node represents an addition operator symbol "+".

<SUB_OP>: This node represents a subtraction operator symbol "-".

<MULT_OP>: This node represents a multiplication operator symbol "*".

<DIV_OP>: This node represents a division operator symbol "/".

<ASSIGN_OP>: This node represents to assign a value to a variable. operator symbol "=".

<START_OP>: This node represent to marks the beginning of a block or a program segment. Operator symbol " < "

<FINISH_OP>: This node represents a marks the end of a block or a program segment assigned operator symbol " > ".

<**karsilastirma_op**> = < | > | <= | >= | == | !=
<**eof**>: This node represents the end of the input sequence, denoted by the symbol "EOF
<**digit**> : 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<**letter**> : a | b | c | ç | d | e | f | g | ? | h | ? | i | j | k | l | m | n | o | ö | p | r | s | ? | t | u | ü | v | y | z

CODE BLOCKS AND FUNCTİONS

*****main Function:**
main(): It is the entry point of the program. It opens the input file and performs token operations by calling the lex() and program() . setlocale(LC_ALL, "Turkish"); : ets the language and character set settings of the program to Turkish. This ensures correct processing of Turkish characters. The file is opened, but if not, an error message is received. Here do { lex(); program(); } while (nextToken != EOF);: lex and program functions are called and repeated until the end of the input file. The end of the file is reached and the program ends.

Code Blocks of main Function:

```
/* Main driver */
int main() {
   setlocale(LC_ALL, "Turkish");
        if ((in_fp = fopen("ex4.txt", "r")) == NULL)

                printf("Dosya Okuma Hatas?. \n");
        else {
                getChar();
                do {
                        lex();
                                program();
                } while (nextToken != EOF);
        }


                                                }
```

*****Fonction of 'program' :**
'start_op' is called and the start of the program is processed. The nested while_statement and if_statement are called. The program is terminated with 'finish_op' .

Code Blocks of program Function:

```
/* Custom functions */
/* <program> -> <start_op> <var> <assign_op> <expr> <finish_op> | <starp_op>
<while_ifadesi> <finish_op> | <star_op> <ifelse> <finish_op>  */
// Main program
void program() {
        printf("Program Baslatiliyor...\n");
        start_op();
        ifelsekontrol();
        // Function to handle repetitive VAR checks
        void handle_var_sequence() {
                while (nextToken == VAR) {
```

```
                        var();
                        assign_op();
                        expr();
                        if (nextToken == WHILE) {
                                while_ifadesi();
                                return;
                        }
                        if(nextToken == EGER) {
                                eger_ifadesi();
                                return;
                        }
                        if (nextToken == VAR) {
                                handle_var_sequence();
                        }
                }
        }
        if (nextToken == WHILE) {
                while_ifadesi();
        } else if (nextToken == VAR) {
                handle_var_sequence();
        }
        while (nextToken != FINISH_OP) {
                ifelsekontrol();
                handle_var_sequence();
        }
        finish_op();
}
```

*****Fonction of 'var' :**
'type' and 'id' are called with this function.

Code Blocks of var Function:

```
/* <var> -> <type> <id> */
void var() {
        printf("Giris --> <var>\n");
        type();
        id();
        printf("Cikis --> <var>\n");
}
```

Fonction of ‘factor’ :
‘digit ‘is called and if nextToken is ‘INT_LIT’, ‘lex ‘is called.

Code Blocks of factor Function:

```
/* <factor> -> <digit> <factor> | <digit> */
void factor() {
	printf("Giris --> <factor>\n");
	digit();
	if (nextToken == INT_LIT) {
		lex();
	}
	printf("Cikis --> <factor>\n");
}
```

Fonction of ‘karsilastirma_op’ :
If 'nextToken' is 'ASSIGN_OP' then 'lex' is called, otherwise it gives an error message.

Code Blocks of ‘karsilastirma_op’ Function:

```
/* <karsilastirma_op> -> < | > | <= | >= | == | != */
void karsilastirma_op() {
	printf("Giris --> <karsilastirma_op>\n");
	if (nextToken == ASSIGN_OP) {
		lex();
	}
	else {
		printf("Hata: <karsilastirma_op> bekleniyor.\n");
		err();
	}
	printf("Cikis --> <karsilastirma_op>\n");
```

Fonction of ‘while_ifadesi’ :

Code Blocks of ‘‘while_ifadesi’ Function:

```
/* <while_ifadesi> -> <while_op> <var> <karsilastirma_op> <factor> <do_op> <program> */
void while_ifadesi() {
	printf("Giris --> <while_ifadesi>\n");
	while_op();
	var();
	karsilastirma_op();
	karsilastirma_op();
	factor();
```

```
        do_op();
        program();
        printf("Cikis --> <while_ifadesi>\n");
}
```

Fonction of ‘while_op’ :
If 'nextToken' is 'WHILE' then 'lex' is called, otherwise it gives an error message
Code Blocks of ‘‘while_op’ Function:

```
/* <while_op> -> while */
void while_op() {
        printf("Giris --> <while_op>\n");
        if (nextToken == WHILE) {
                lex();
        } else {
                printf("Hata: <while_op> bekleniyor.\n");
                err();
        }
        printf("Cikis --> <while_op>\n");
}
```

Fonction of ‘do_op’ :
If 'nextToken' is 'DO' then 'lex' is called, otherwise it gives an error message.

Code Blocks of ‘do_op’ Function:

```
/* <do_op> -> do */
void do_op() {
        printf("Giris --> <do_op>\n");
        if (nextToken == DO) {
                lex();
        } else {
                printf("Hata: <do_op> bekleniyor.\n");
                err();
        }
        printf("Cikis --> <do_op>\n");
}
```

Fonction of ‘degilse’:
The function checks if the variable 'nextToken' is 'NOT'.
If 'nextToken' is 'NOT', the next token is processed by calling the 'lex' function.
If 'nextToken' is 'NOT', an error message is printed and the 'err' function is called.

Code Blocks of ‘degilse’ Fonction:

```
/* <degilse> -> degilse */
void degilse() {
	printf("Giris --> <degilse>\n");
	if (nextToken == DEGILSE) {
		lex();
	}
	else {
		printf("Hata: <degilse> bekleniyor.\n");
		err();
	}
	printf("Cikis --> <degilse>\n");
}
```

Fonction of ‘ise’:
This function checks for the 'if' keyword. The function checks if the variable 'nextToken' is 'ISE'. If 'nextToken' is 'ISE', the next token is processed by calling the 'lex' function. If 'nextToken' is not 'ISE', an error message is printed and the 'err' function is called.

Code Blocks of ‘ise’ Fonction:

```
void ise() {
	printf("Giris --> <ise>\n");
	if (nextToken == ISE) {
		lex();
	}
	else {
		printf("Hata: <ise> bekleniyor.\n");
		err();
	}
	printf("Cikis --> <degilse>\n");
}
```

Fonction of ‘ifelsekontrol’ :

Code Blocks of ‘ifelsekontrol’ Fonction:

```
void ifelsekontrol() {
	if (nextToken == EGER) {
		eger_ifadesi();
	} else if (nextToken == WHILE) {
		while_ifadesi();
	}
}
```

Fonction of ‘isCorrectDataType’ :

This function checks whether a given lexeme is the correct data type. If the lexeme is 'number', the function returns 1.
Otherwise, the function returns 0.

Code Blocks of ‘isCorrectDataType’ Fonction:

```
correct data type func for if else and program
/* correct data type func for if else and program */
int isCorrectDataType(char* lexeme) {
	if (strcmp(lexeme, "sayi") == 0) {
		return 1;
	}
	else {
		return 0;
	}
}
```

Fonction of ‘err’ :

This function is called when an error occurs. The function prints an error message and terminates the program

Code Blocks of ‘err’ Fonction:

```
/* Error function */
	void err() {
	printf("Hata fonksiyonu calisti. \n");
	exit(0);
}


/* End of custom functions */
/* custome funcs for if else */
```

Fonction of ‘eger_ifadesi’ :

Code Blocks of ‘eger_ifadesi’ Fonction:

```
/* <eger_ifadesi> -> <eger> <var><karsilastirma_op> <factor> <dogruluk_kontrol> <degilse>
<dogruluk_kontrol> <finish_op> */
void eger_ifadesi() {
        printf("Giris --> <eger_ifadesi>\n");
        lex();
        var();
        karsilastirma_op();
        karsilastirma_op();
        factor();
        ise();
        program();
        degilse();
        program();
        printf("Cikis --> <eger_ifadesi>\n");
}
```

These codes are written to recognize and process the "if statement" structure. The `eger_ifadesi` function processes each part of this statement one by one and invokes the rest of the program to execute the if-else logic. Additionally, the `err` function is called in case of errors, terminating the program.

Fonction of ‘start_op’ :

This function is used to recognize and process the < operator. The function checks whether the variable nextToken is START_OP (representing the < operator in this case). If nextToken is 'START_OP', the next token is processed by calling the l'ex()' function. If 'nextToken' is not 'START_OP', an error message is printed and the program is terminated by calling the 'err' function**.**

Code Blocks of ‘start_op’:

```
/* <start_op> -> < */
void start_op() {
        printf("Giris --> <start_op>\n");
        if (nextToken == START_OP) {
                lex();
        }
        else {
                printf("Hata: <start_op> bekleniyor.\n");
                err();
        }
        printf("Cikis --> <start_op>\n");
}
```

Fonction of ‘finish_op’ :
This function is used to recognize and process the > operator. The function checks if the nextToken variable is 'FINISH_OP'. If 'nextToken' is 'FINISH_OP', the next token is processed by calling the 'lex' function. If nextToken 'FINISH_OP' is not present, an error message is printed and the program is terminated by calling the 'err' function. The nextToken value is printed at the beginning of the function for error setting purposes.

Code Blocks of ‘finish_op’:

```
/* <finish_op> -> > */
void finish_op() {
	printf("Giris --> <finish_op>\n");
	printf("nextToken: %d\n", nextToken);
	if (nextToken == FINISH_OP) {
		lex();
	}
	else {
		printf("Hata: <finish_op> bekleniyor.\n");
		err();
	}
	printf("Cikis --> <finish_op>\n");
}
```

Fonction of ‘getNonBlank’ :
Finds the first non-space character, skipping the space characters.

Code Blocks of ‘getNonBlank’ Fonction:

```
/* getNonBlank - a function to call getChar until it
 returns a non-whitespace character */
void getNonBlank() {
	while (isspace(nextChar))
		getChar();
}
```

Fonction of 'lex' :
This function is used as a simple lexical analyzer and generates tokens for arithmetic expressions. When any token is read, the token type is assigned to the nextToken variable and the function returns this token. If the character is a letter, it can be an identifier. A lexeme consisting of letters and numbers is defined. If the character is a number, it can be a numeric value. By parsing the digits the lexeme is identified and nextToken is set to INT_LIT. Unknown characters and operators are handled with the lookup function. Parses arithmetic expressions by identifying tokens and returns the appropriate token types. When the end of the file is reached, it prints the message that the program is finished.

Code Blocks of 'lex' Foction:

```
/* lex - a simple lexical analyzer for arithmetic
 expressions */
int lex() {
	lexLen = 0;
	getNonBlank();
	switch (charClass) {
		/* Parse identifiers */
	case LETTER:
		addChar();
		getChar();
		while (charClass == LETTER || charClass == DIGIT) {
			addChar();
			getChar();
		}

		/* If the read lexeme is a variable, it assigns the value VAR to the nextToken
		If the read lexeme is a identifier, it assigns the value IDENT to the nextToken
		(The isEqual function is defined below the code block) */
		if (isCorrectDataType(lexeme)) {
			nextToken = VAR;
		}
		else {
			nextToken = IDENT;
		}
		//break;
		if (strcmp(lexeme, "eger") == 0)
			nextToken = EGER;
		else if (strcmp(lexeme, "ise") == 0)
			nextToken = ISE;
		else if (strcmp(lexeme, "degilse") == 0)
			nextToken = DEGILSE;
		else if (strcmp(lexeme, "e?ittir") == 0)
			nextToken = KARSILASTIRMA_DURUMU;
		else if (strcmp(lexeme, "b?y?kt?r") == 0)
			nextToken = KARSILASTIRMA_DURUMU;
```

```
                else if (strcmp(lexeme, "k???kt?r") == 0)
                        nextToken = KARSILASTIRMA_DURUMU;
                else if (strcmp(lexeme, "surece") == 0)
                        nextToken = WHILE;
                else if (strcmp(lexeme, "yap") == 0)
                        nextToken = DO;

                else {

                }
                break;
                /* Parse integer literals */
        case DIGIT:
                addChar();
                getChar();
                while (charClass == DIGIT) {
                        addChar();
                        getChar();
                }
                nextToken = INT_LIT;
                break;
                /* Parentheses and operators */
        case UNKNOWN:
                lookup(nextChar);
                getChar();
                break;
                /* EOF */
        case EOF:
                nextToken = EOF;
                lexeme[0] = 'E';
                lexeme[1] = 'O';
                lexeme[2] = 'F';
                lexeme[3] = 0;
                printf("Program bitti.\n");
                break;
        }       /* End of switch */

        printf("Sonraki sozcuk: %d, sonraki sozcukbirim %s\n",
                nextToken, lexeme);
        return nextToken;
} /* End of function lex */
```

Fonction of ‘type’:
Recognizes and processes the variable type int.
Code Blocks of. ‘type’ Fonction:

```
/* <type> -> int */
void type() {
	printf("Giris --> <type>\n");
	if (nextToken == VAR) {
		lex();
	}
	else {
		printf("Hata: <type> bekleniyor.\n");
		err();
	}
	printf("Cikis --> <type>\n");
}
```

Fonction of ‘id’:
Recognizes an identifier and calls the necessary functions to process it.
Code Blocks of. ‘id’ Fonction:

```
/* <id> -> <letter> <id2> */
void id() {
	printf("Giris --> <id>\n");
	letter();
	id2();
	printf("Cikis --> <id>\n");
}
```

Fonction of ‘id2’:
Processes the rest of the identifier and checks whether it is a letter or a number.
Code Blocks of. ‘id2’ Fonction:

```
/* <id2> -> <letter> <id2> | <digit> <id2> | ? */
void id2() {
	printf("Giris --> <id2>\n");
	if (nextToken == IDENT) {
		lex();
	}
	else if (nextToken == INT_LIT) {
		lex();
	}
	else {
		printf("Gecersiz karakter. \n");
		return;
	}
	printf("Cikis --> <id2>\n");
```

```
}
```

Fonction of ‘expr:
Recognizes an expression and processes it according to addition or subtraction operators.

Code Blocks of. ‘expr’ Fonction:

```
/* <expr> -> <term> - <expr> | <term> + <expr> | <term> */
void expr() {
	printf("Giris --> <expr>\n");
	term();
	if (nextToken == SUB_OP) {
		lex();
		expr();
	}
	else if (nextToken == ADD_OP) {
		lex();
		expr();
	}
	printf("Cikis --> <expr>\n");
}
```

Fonction of ‘term’:
Recognizes a term and processes it according to multiplication or division operators.
Code Blocks of. ‘term’ Fonction:

```
/* <term> -> <factor> / <term> | <factor> * <term> | <factor> */
void term() {
	printf("Giros --> <term>\n");
	factor();
	if (nextToken == DIV_OP) {
		lex();
		term();
	}
	else if (nextToken == MULT_OP) {
		lex();
		term();
	}
	printf("Cikis --> <term>\n");
}
```

Fonction of ‘digit’ :
This function is used to recognize and process a digit.
Code Blocks of. ‘digit’ Fonction:

```
/* <digit> -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 */
void digit() {
	printf("Giris --> <digit>\n");
	if (nextToken == INT_LIT) {
		lex();
	}
	printf("Cikis --> <digit>\n");
}
```

Fonction of ‘letter’ :
This function is used to recognize and process a letter.
Code Blocks of. ‘letter’ Fonction:

```
/* <letter> -> a | b | c | ? | d | e | f | g | ? | h | ? | i | j | k | l | m | n | o | ? | p | r | s | ? | t | u | ? | v | y | z
*/
void letter() {
	printf("Giris --> <letter>\n");
	if (nextToken == IDENT) {
		lex();
	}
	printf("Cikis --> <letter>\n");
}
```

Fonction of ‘assıgn_op ‘:
This function defines and processes an assignment operator ('='). If it is not an assignment operator, it prints an error message and terminates the program.
Code Blocks of. ‘assıgn_op’ Fonction:

```
/* <assign_op> -> = */
void assign_op() {
	printf("Giris --> <assign_op>\n");
	if (nextToken == ASSIGN_OP) {
		lex();
	}
	else {
```

```
            printf("Hata     : <assign_op> bekleniyor.\n");
            err();
        }
        printf("Cikis --> <assign_op>\n");
}
```

Fonction of ‘lookup’ :
This function is used to recognize operators and parentheses and return appropriate token values. It takes a character as input and determines which token this character corresponds to.
Code Blocks of. ‘lookup’ Fonction:

```
/* Open the input data file and process its contents
 lookup - a function to lookup operators and parentheses
 and return the token */
int lookup(char ch) {
        switch (ch) {

        case '(':
            addChar();
            nextToken = LEFT_PAREN;
            break;

        case ')':
            addChar();
            nextToken = RIGHT_PAREN;
            break;

        case '+':
            addChar();
            nextToken = ADD_OP;
            break;

        case '-':
            addChar();
            nextToken = SUB_OP;
            break;

        case '*':
            addChar();
            nextToken = MULT_OP;
            break;

        case '/':
            addChar();
            nextToken = DIV_OP;
            break;
```

```
            /* Assigment operator assigned to "=" */
        case '=':
            addChar();
            nextToken = ASSIGN_OP;
            break;

            /* Start operator assigned to "<" */
        case '{':
            addChar();
            nextToken = START_OP;
            break;

            /* Finish operator assigned to ">" */
        case '}':
            addChar();
            nextToken = FINISH_OP;
            break;
        default:
                    addChar();
                    if (strcmp(lexeme, "surece") == 0)
                        nextToken = WHILE;
                    else if (strcmp(lexeme, "yap") == 0)
                        nextToken = DO;
                    else
                        nextToken = EOF;
                    break;
        }
        return nextToken;
}
```

Fonction of ‘addChar’ :
This function adds the character 'nextChar' to the lexeme array. If lexLen does not exceed 98 characters, 'nextChar' is added and the lexeme is terminated with null. Otherwise, an error message is printed.
Code Blocks of. ‘addChar’ Fonction:

```
/* addChar - a function to add nextChar to lexeme */
void addChar() {
        if (lexLen <= 98) {
            lexeme[lexLen++] = nextChar;
            lexeme[lexLen] = 0;
        }
```

```
        else
                printf("Hata. \n");
}

/* getChar - a function to get the next character of
 input and determine its character class */
```

Fonction of ‘getChar’ :
This function takes the next character from the input and determines the class of the character. The character class can be letter (LETTER), digit (DIGIT), unknown (UNKNOWN), or end-of-file (EOF).
Code Blocks of. ‘getChar’ Fonction:

```
void getChar() {
        if ((nextChar = getc(in_fp)) != EOF) {
                if (isalpha(nextChar))
                        charClass = LETTER;
                else if (isdigit(nextChar))
                        charClass = DIGIT;

                else charClass = UNKNOWN;
        }
        else
                charClass = EOF;
}
```

Example Inputs and Outputs Of The Language:

ERROR EXAMPLES

INPUT: { sayi bar 35 + 2 }
OUTPUT:

File - Unnamed

```
 1 C:\Users\FurkanBulut\Documents\GitHub\Turcanalyze\
   main.exe
 2 Sonraki sozcuk: 30, sonraki sozcukbirim {
 3 Program Baslatiliyor...
 4 Giris --> <start_op>
 5 Sonraki sozcuk: 12, sonraki sozcukbirim sayi
 6 Cikis --> <start_op>
 7 Giris --> <var>
 8 Giris --> <type>
 9 Sonraki sozcuk: 11, sonraki sozcukbirim bar
10 Cikis --> <type>
11 Giris --> <id>
12 Giris --> <letter>
13 Sonraki sozcuk: 10, sonraki sozcukbirim 35
14 Cikis --> <letter>
15 Giris --> <id2>
16 Sonraki sozcuk: 21, sonraki sozcukbirim +
17 Cikis --> <id2>
18 Cikis --> <id>
19 Cikis --> <var>
20 Giris --> <assign_op>
21 Hata    : <assign_op> bekleniyor.
22 Hata fonksiyonu calisti.
23
24 Process finished with exit code 0
25
```

```
INPUT: {
   eger sayi 5 == 5 ise { surece  sayi == 3 yap { sayi furkan = 5 }}
   degilse {
   sayi deneme = 10
   }
   eger sayi == 5 ise { surece sayi == 3 do { sayi furkan = 5 }}
     }
```

OUTPUT:

File - Unnamed

```
 1 C:\Users\FurkanBulut\Documents\GitHub\Turcanalyze\
   main.exe
 2 Sonraki sozcuk: 30, sonraki sozcukbirim {
 3 Program Baslatiliyor...
 4 Giris --> <start_op>
 5 Sonraki sozcuk: 40, sonraki sozcukbirim eger
 6 Cikis --> <start_op>
 7 Giris --> <eger_ifadesi>
 8 Sonraki sozcuk: 12, sonraki sozcukbirim sayi
 9 Giris --> <var>
10 Giris --> <type>
11 Sonraki sozcuk: 10, sonraki sozcukbirim 5
12 Cikis --> <type>
13 Giris --> <id>
14 Giris --> <letter>
15 Cikis --> <letter>
16 Giris --> <id2>
17 Sonraki sozcuk: 20, sonraki sozcukbirim =
18 Cikis --> <id2>
19 Cikis --> <id>
20 Cikis --> <var>
21 Giris --> <karsilastirma_op>
22 Sonraki sozcuk: 20, sonraki sozcukbirim =
23 Cikis --> <karsilastirma_op>
24 Giris --> <karsilastirma_op>
25 Sonraki sozcuk: 10, sonraki sozcukbirim 5
26 Cikis --> <karsilastirma_op>
27 Giris --> <factor>
28 Giris --> <digit>
29 Sonraki sozcuk: 51, sonraki sozcukbirim ise
30 Cikis --> <digit>
31 Cikis --> <factor>
32 Giris --> <ise>
33 Sonraki sozcuk: 30, sonraki sozcukbirim {
34 Cikis --> <degilse>
35 Program Baslatiliyor...
36 Giris --> <start_op>
37 Sonraki sozcuk: 52, sonraki sozcukbirim surece
38 Cikis --> <start_op>
39 Giris --> <while_ifadesi>
40 Giris --> <while_op>
```

File - Unnamed

```
41 Sonraki sozcuk: 12, sonraki sozcukbirim sayi
42 Cikis --> <while_op>
43 Giris --> <var>
44 Giris --> <type>
45 Sonraki sozcuk: 20, sonraki sozcukbirim =
46 Cikis --> <type>
47 Giris --> <id>
48 Giris --> <letter>
49 Cikis --> <letter>
50 Giris --> <id2>
51 Gecersiz karakter.
52 Cikis --> <id>
53 Cikis --> <var>
54 Giris --> <karsilastirma_op>
55 Sonraki sozcuk: 20, sonraki sozcukbirim =
56 Cikis --> <karsilastirma_op>
57 Giris --> <karsilastirma_op>
58 Sonraki sozcuk: 10, sonraki sozcukbirim 3
59 Cikis --> <karsilastirma_op>
60 Giris --> <factor>
61 Giris --> <digit>
62 Sonraki sozcuk: 53, sonraki sozcukbirim yap
63 Cikis --> <digit>
64 Cikis --> <factor>
65 Giris --> <do_op>
66 Sonraki sozcuk: 30, sonraki sozcukbirim {
67 Cikis --> <do_op>
68 Program Baslatiliyor...
69 Giris --> <start_op>
70 Sonraki sozcuk: 12, sonraki sozcukbirim sayi
71 Cikis --> <start_op>
72 Giris --> <var>
73 Giris --> <type>
74 Sonraki sozcuk: 11, sonraki sozcukbirim furkan
75 Cikis --> <type>
76 Giris --> <id>
77 Giris --> <letter>
78 Sonraki sozcuk: 20, sonraki sozcukbirim =
79 Cikis --> <letter>
80 Giris --> <id2>
81 Gecersiz karakter.
```

File - Unnamed

```
 82 Cikis --> <id>
 83 Cikis --> <var>
 84 Giris --> <assign_op>
 85 Sonraki sozcuk: 10, sonraki sozcukbirim 5
 86 Cikis --> <assign_op>
 87 Giris --> <expr>
 88 Giros --> <term>
 89 Giris --> <factor>
 90 Giris --> <digit>
 91 Sonraki sozcuk: 31, sonraki sozcukbirim }
 92 Cikis --> <digit>
 93 Cikis --> <factor>
 94 Cikis --> <term>
 95 Cikis --> <expr>
 96 Giris --> <finish_op>
 97 nextToken: 31
 98 Sonraki sozcuk: 31, sonraki sozcukbirim }
 99 Cikis --> <finish_op>
100 Cikis --> <while_ifadesi>
101 Giris --> <finish_op>
102 nextToken: 31
103 Sonraki sozcuk: 45, sonraki sozcukbirim degilse
104 Cikis --> <finish_op>
105 Giris --> <degilse>
106 Sonraki sozcuk: 30, sonraki sozcukbirim {
107 Cikis --> <degilse>
108 Program Baslatiliyor...
109 Giris --> <start_op>
110 Sonraki sozcuk: 12, sonraki sozcukbirim sayi
111 Cikis --> <start_op>
112 Giris --> <var>
113 Giris --> <type>
114 Sonraki sozcuk: 11, sonraki sozcukbirim deneme
115 Cikis --> <type>
116 Giris --> <id>
117 Giris --> <letter>
118 Sonraki sozcuk: 20, sonraki sozcukbirim =
119 Cikis --> <letter>
120 Giris --> <id2>
121 Gecersiz karakter.
122 Cikis --> <id>
```

File - Unnamed

```
123 Cikis --> <var>
124 Giris --> <assign_op>
125 Sonraki sozcuk: 10, sonraki sozcukbirim 10
126 Cikis --> <assign_op>
127 Giris --> <expr>
128 Giros --> <term>
129 Giris --> <factor>
130 Giris --> <digit>
131 Sonraki sozcuk: 31, sonraki sozcukbirim }
132 Cikis --> <digit>
133 Cikis --> <factor>
134 Cikis --> <term>
135 Cikis --> <expr>
136 Giris --> <finish_op>
137 nextToken: 31
138 Sonraki sozcuk: 40, sonraki sozcukbirim eger
139 Cikis --> <finish_op>
140 Cikis --> <eger_ifadesi>
141 Giris --> <eger_ifadesi>
142 Sonraki sozcuk: 12, sonraki sozcukbirim sayi
143 Giris --> <var>
144 Giris --> <type>
145 Sonraki sozcuk: 20, sonraki sozcukbirim =
146 Cikis --> <type>
147 Giris --> <id>
148 Giris --> <letter>
149 Cikis --> <letter>
150 Giris --> <id2>
151 Gecersiz karakter.
152 Cikis --> <id>
153 Cikis --> <var>
154 Giris --> <karsilastirma_op>
155 Sonraki sozcuk: 20, sonraki sozcukbirim =
156 Cikis --> <karsilastirma_op>
157 Giris --> <karsilastirma_op>
158 Sonraki sozcuk: 10, sonraki sozcukbirim 5
159 Cikis --> <karsilastirma_op>
160 Giris --> <factor>
161 Giris --> <digit>
162 Sonraki sozcuk: 51, sonraki sozcukbirim ise
163 Cikis --> <digit>
```

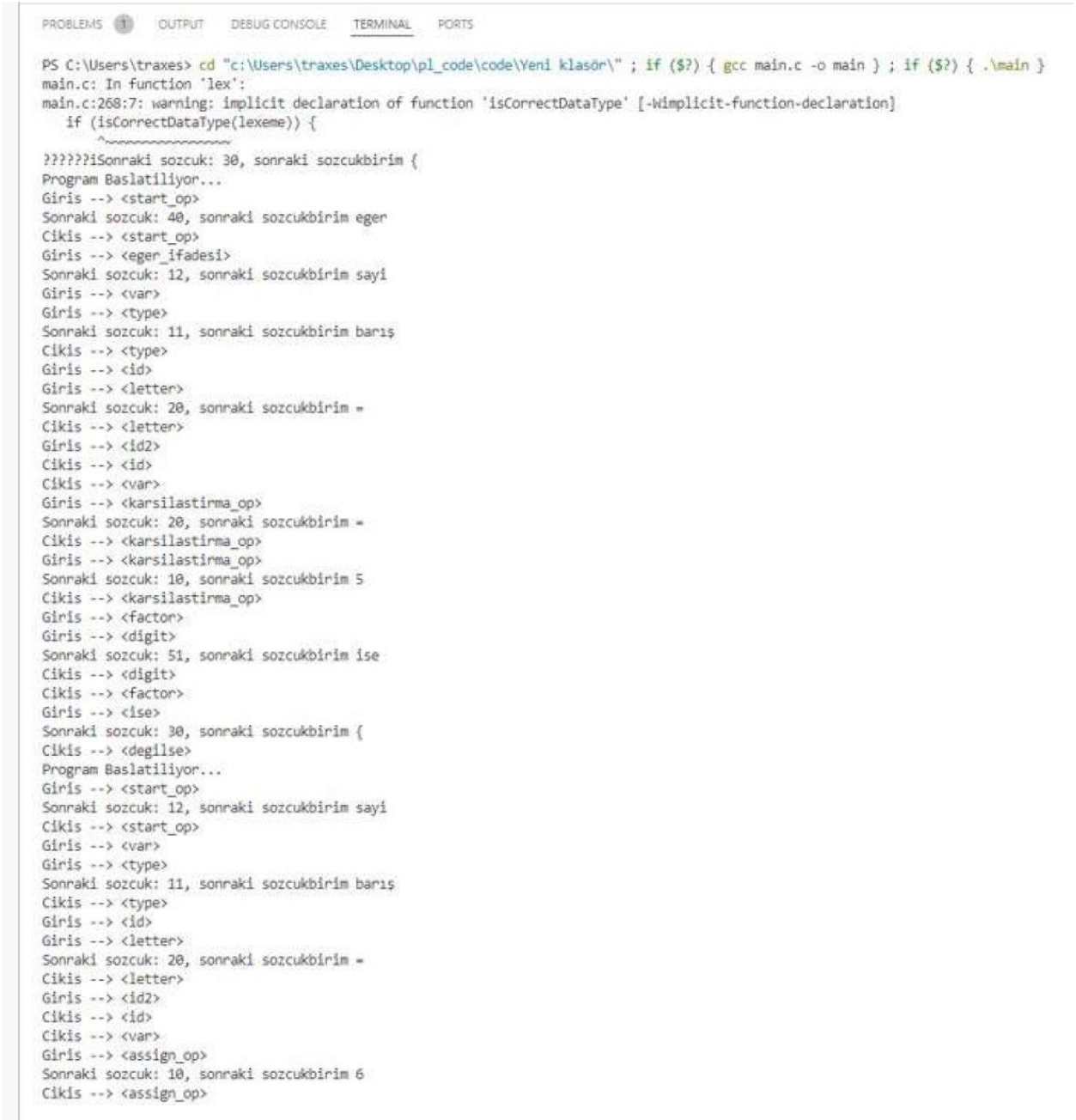
```
164 Cikis --> <factor>
165 Giris --> <ise>
166 Sonraki sozcuk: 30, sonraki sozcukbirim {
167 Cikis --> <degilse>
168 Program Baslatiliyor...
169 Giris --> <start_op>
170 Sonraki sozcuk: 52, sonraki sozcukbirim surece
171 Cikis --> <start_op>
172 Giris --> <while_ifadesi>
173 Giris --> <while_op>
174 Sonraki sozcuk: 12, sonraki sozcukbirim sayi
175 Cikis --> <while_op>
176 Giris --> <var>
177 Giris --> <type>
178 Sonraki sozcuk: 20, sonraki sozcukbirim =
179 Cikis --> <type>
180 Giris --> <id>
181 Giris --> <letter>
182 Cikis --> <letter>
183 Giris --> <id2>
184 Gecersiz karakter.
185 Cikis --> <id>
186 Cikis --> <var>
187 Giris --> <karsilastirma_op>
188 Sonraki sozcuk: 20, sonraki sozcukbirim =
189 Cikis --> <karsilastirma_op>
190 Giris --> <karsilastirma_op>
191 Sonraki sozcuk: 10, sonraki sozcukbirim 3
192 Cikis --> <karsilastirma_op>
193 Giris --> <factor>
194 Giris --> <digit>
195 Sonraki sozcuk: 11, sonraki sozcukbirim do
196 Cikis --> <digit>
197 Cikis --> <factor>
198 Giris --> <do_op>
199 Hata: <do_op> bekleniyor.
200 Hata fonksiyonu calisti.
201
202 Process finished with exit code 0
203
```

EXAMPLES THAT DO NOT CAUSE ERRORS

INPUT:

{eger sayi bar″¸ == 5 ise { sayi bar″¸ = 6 } degilse { sayi bar″¸ = 4 } sayi ˆ□ ,i¸ = 88}

OUTPUT:



```
PROBLEMS 1   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

PS C:\Users\traxes> cd "c:\Users\traxes\Desktop\pl_code\code\Yeni klasör\" ; if ($?) { gcc main.c -o main } ; if ($?) { .\main }
main.c: In function 'lex':
main.c:268:7: warning: implicit declaration of function 'isCorrectDataType' [-Wimplicit-function-declaration]
   if (isCorrectDataType(lexeme)) {
       ^~~~~~~~~~~~~~~~~
??????iSonraki sozcuk: 30, sonraki sozcukbirim {
Program Baslatiliyor...
Giris --> <start_op>
Sonraki sozcuk: 40, sonraki sozcukbirim eger
Cikis --> <start_op>
Giris --> <eger_ifadesi>
Sonraki sozcuk: 12, sonraki sozcukbirim sayi
Giris --> <var>
Giris --> <type>
Sonraki sozcuk: 11, sonraki sozcukbirim barış
Cikis --> <type>
Giris --> <id>
Giris --> <letter>
Sonraki sozcuk: 20, sonraki sozcukbirim =
Cikis --> <letter>
Giris --> <id2>
Cikis --> <id>
Cikis --> <var>
Giris --> <karsilastirma_op>
Sonraki sozcuk: 20, sonraki sozcukbirim =
Cikis --> <karsilastirma_op>
Giris --> <karsilastirma_op>
Sonraki sozcuk: 10, sonraki sozcukbirim 5
Cikis --> <karsilastirma_op>
Giris --> <factor>
Giris --> <digit>
Sonraki sozcuk: 51, sonraki sozcukbirim ise
Cikis --> <digit>
Cikis --> <factor>
Giris --> <ise>
Sonraki sozcuk: 30, sonraki sozcukbirim {
Cikis --> <degilse>
Program Baslatiliyor...
Giris --> <start_op>
Sonraki sozcuk: 12, sonraki sozcukbirim sayi
Cikis --> <start_op>
Giris --> <var>
Giris --> <type>
Sonraki sozcuk: 11, sonraki sozcukbirim barış
Cikis --> <type>
Giris --> <id>
Giris --> <letter>
Sonraki sozcuk: 20, sonraki sozcukbirim =
Cikis --> <letter>
Giris --> <id2>
Cikis --> <id>
Cikis --> <var>
Giris --> <assign_op>
Sonraki sozcuk: 10, sonraki sozcukbirim 6
Cikis --> <assign_op>
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Sonraki sozcuk: 20, sonraki sozcukbirim =
Cikis --> <letter>
Giris --> <id2>
Cikis --> <id>
Cikis --> <var>
Giris --> <assign_op>
Sonraki sozcuk: 10, sonraki sozcukbirim 4
Cikis --> <assign_op>
Giris --> <expr>
Giros --> <term>
Giris --> <factor>
Giris --> <digit>
Sonraki sozcuk: 31, sonraki sozcukbirim }
Cikis --> <digit>
Cikis --> <factor>
Cikis --> <term>
Cikis --> <expr>
Giris --> <finish_op>
nextToken: 31
Sonraki sozcuk: 12, sonraki sozcukbirim sayi
Cikis --> <finish_op>
Cikis --> <eger_ifadesi>
Giris --> <var>
Giris --> <type>
Sonraki sozcuk: 11, sonraki sozcukbirim öğüiş
Cikis --> <type>
Giris --> <id>
Giris --> <letter>
Sonraki sozcuk: 20, sonraki sozcukbirim =
Cikis --> <letter>
Giris --> <id2>
Cikis --> <id>
Cikis --> <var>
Giris --> <assign_op>
Sonraki sozcuk: 10, sonraki sozcukbirim 88
Cikis --> <assign_op>
Giris --> <expr>
Giros --> <term>
Giris --> <factor>
Giris --> <digit>
Sonraki sozcuk: 31, sonraki sozcukbirim }
Cikis --> <digit>
Cikis --> <factor>
Cikis --> <term>
Cikis --> <expr>
Giris --> <finish_op>
nextToken: 31
Program bitti.
Sonraki sozcuk: -1, sonraki sozcukbirim EOF
Cikis --> <finish_op>
PS C:\Users\traxes\Desktop\pl_code\code\Yeni klasör>
```

INPUT:

```
{
    sayi x = 3
    surece sayi == 3 yap
    {
        eger sayi x == 5 ise { sayi x = 6 } degilse { sayi x = 4 }
        sayi x = 88
        surece sayi x == 5 yap
        {
            sayi deneme  = 96
            eger sayi x == 5 ise { sayi x = 6 } degilse { sayi x = 55 }
        }

        eger sayi x == 5 ise { sayi x = 6 } degilse { sayi x = 59599 }

    }
        eger sayi x == 5 ise { sayi x = 6 } degilse { sayi x = 59599 }
        sayi x = 5
    }
```

OUTPUT:

File - Unnamed

```
 1 C:\Users\FurkanBulut\Documents\GitHub\Turcanalyze\
   main.exe
 2 Sonraki sozcuk: 30, sonraki sozcukbirim {
 3 Program Baslatiliyor...
 4 Giris --> <start_op>
 5 Sonraki sozcuk: 12, sonraki sozcukbirim sayi
 6 Cikis --> <start_op>
 7 Giris --> <var>
 8 Giris --> <type>
 9 Sonraki sozcuk: 11, sonraki sozcukbirim x
10 Cikis --> <type>
11 Giris --> <id>
12 Giris --> <letter>
13 Sonraki sozcuk: 20, sonraki sozcukbirim =
14 Cikis --> <letter>
15 Giris --> <id2>
16 Gecersiz karakter.
17 Cikis --> <id>
18 Cikis --> <var>
19 Giris --> <assign_op>
20 Sonraki sozcuk: 10, sonraki sozcukbirim 3
21 Cikis --> <assign_op>
22 Giris --> <expr>
23 Giros --> <term>
24 Giris --> <factor>
25 Giris --> <digit>
26 Sonraki sozcuk: 52, sonraki sozcukbirim surece
27 Cikis --> <digit>
28 Cikis --> <factor>
29 Cikis --> <term>
30 Cikis --> <expr>
31 Giris --> <while_ifadesi>
32 Giris --> <while_op>
33 Sonraki sozcuk: 12, sonraki sozcukbirim sayi
34 Cikis --> <while_op>
35 Giris --> <var>
36 Giris --> <type>
37 Sonraki sozcuk: 20, sonraki sozcukbirim =
38 Cikis --> <type>
39 Giris --> <id>
40 Giris --> <letter>
```

```
41 Cikis --> <letter>
42 Giris --> <id2>
43 Gecersiz karakter.
44 Cikis --> <id>
45 Cikis --> <var>
46 Giris --> <karsilastirma_op>
47 Sonraki sozcuk: 20, sonraki sozcukbirim =
48 Cikis --> <karsilastirma_op>
49 Giris --> <karsilastirma_op>
50 Sonraki sozcuk: 10, sonraki sozcukbirim 3
51 Cikis --> <karsilastirma_op>
52 Giris --> <factor>
53 Giris --> <digit>
54 Sonraki sozcuk: 53, sonraki sozcukbirim yap
55 Cikis --> <digit>
56 Cikis --> <factor>
57 Giris --> <do_op>
58 Sonraki sozcuk: 30, sonraki sozcukbirim {
59 Cikis --> <do_op>
60 Program Baslatiliyor...
61 Giris --> <start_op>
62 Sonraki sozcuk: 40, sonraki sozcukbirim eger
63 Cikis --> <start_op>
64 Giris --> <eger_ifadesi>
65 Sonraki sozcuk: 12, sonraki sozcukbirim sayi
66 Giris --> <var>
67 Giris --> <type>
68 Sonraki sozcuk: 11, sonraki sozcukbirim x
69 Cikis --> <type>
70 Giris --> <id>
71 Giris --> <letter>
72 Sonraki sozcuk: 20, sonraki sozcukbirim =
73 Cikis --> <letter>
74 Giris --> <id2>
75 Gecersiz karakter.
76 Cikis --> <id>
77 Cikis --> <var>
78 Giris --> <karsilastirma_op>
79 Sonraki sozcuk: 20, sonraki sozcukbirim =
80 Cikis --> <karsilastirma_op>
81 Giris --> <karsilastirma_op>
```

```
 82 Sonraki sozcuk: 10, sonraki sozcukbirim 5
 83 Cikis --> <karsilastirma_op>
 84 Giris --> <factor>
 85 Giris --> <digit>
 86 Sonraki sozcuk: 51, sonraki sozcukbirim ise
 87 Cikis --> <digit>
 88 Cikis --> <factor>
 89 Giris --> <ise>
 90 Sonraki sozcuk: 30, sonraki sozcukbirim {
 91 Cikis --> <degilse>
 92 Program Baslatiliyor...
 93 Giris --> <start_op>
 94 Sonraki sozcuk: 12, sonraki sozcukbirim sayi
 95 Cikis --> <start_op>
 96 Giris --> <var>
 97 Giris --> <type>
 98 Sonraki sozcuk: 11, sonraki sozcukbirim x
 99 Cikis --> <type>
100 Giris --> <id>
101 Giris --> <letter>
102 Sonraki sozcuk: 20, sonraki sozcukbirim =
103 Cikis --> <letter>
104 Giris --> <id2>
105 Gecersiz karakter.
106 Cikis --> <id>
107 Cikis --> <var>
108 Giris --> <assign_op>
109 Sonraki sozcuk: 10, sonraki sozcukbirim 6
110 Cikis --> <assign_op>
111 Giris --> <expr>
112 Giros --> <term>
113 Giris --> <factor>
114 Giris --> <digit>
115 Sonraki sozcuk: 31, sonraki sozcukbirim }
116 Cikis --> <digit>
117 Cikis --> <factor>
118 Cikis --> <term>
119 Cikis --> <expr>
120 Giris --> <finish_op>
121 nextToken: 31
122 Sonraki sozcuk: 45, sonraki sozcukbirim degilse
```

```
123 Cikis --> <finish_op>
124 Giris --> <degilse>
125 Sonraki sozcuk: 30, sonraki sozcukbirim {
126 Cikis --> <degilse>
127 Program Baslatiliyor...
128 Giris --> <start_op>
129 Sonraki sozcuk: 12, sonraki sozcukbirim sayi
130 Cikis --> <start_op>
131 Giris --> <var>
132 Giris --> <type>
133 Sonraki sozcuk: 11, sonraki sozcukbirim x
134 Cikis --> <type>
135 Giris --> <id>
136 Giris --> <letter>
137 Sonraki sozcuk: 20, sonraki sozcukbirim =
138 Cikis --> <letter>
139 Giris --> <id2>
140 Gecersiz karakter.
141 Cikis --> <id>
142 Cikis --> <var>
143 Giris --> <assign_op>
144 Sonraki sozcuk: 10, sonraki sozcukbirim 4
145 Cikis --> <assign_op>
146 Giris --> <expr>
147 Giros --> <term>
148 Giris --> <factor>
149 Giris --> <digit>
150 Sonraki sozcuk: 31, sonraki sozcukbirim }
151 Cikis --> <digit>
152 Cikis --> <factor>
153 Cikis --> <term>
154 Cikis --> <expr>
155 Giris --> <finish_op>
156 nextToken: 31
157 Sonraki sozcuk: 12, sonraki sozcukbirim sayi
158 Cikis --> <finish_op>
159 Cikis --> <eger_ifadesi>
160 Giris --> <var>
161 Giris --> <type>
162 Sonraki sozcuk: 11, sonraki sozcukbirim x
163 Cikis --> <type>
```

```
164 Giris --> <id>
165 Giris --> <letter>
166 Sonraki sozcuk: 20, sonraki sozcukbirim =
167 Cikis --> <letter>
168 Giris --> <id2>
169 Gecersiz karakter.
170 Cikis --> <id>
171 Cikis --> <var>
172 Giris --> <assign_op>
173 Sonraki sozcuk: 10, sonraki sozcukbirim 88
174 Cikis --> <assign_op>
175 Giris --> <expr>
176 Giros --> <term>
177 Giris --> <factor>
178 Giris --> <digit>
179 Sonraki sozcuk: 52, sonraki sozcukbirim surece
180 Cikis --> <digit>
181 Cikis --> <factor>
182 Cikis --> <term>
183 Cikis --> <expr>
184 Giris --> <while_ifadesi>
185 Giris --> <while_op>
186 Sonraki sozcuk: 12, sonraki sozcukbirim sayi
187 Cikis --> <while_op>
188 Giris --> <var>
189 Giris --> <type>
190 Sonraki sozcuk: 11, sonraki sozcukbirim x
191 Cikis --> <type>
192 Giris --> <id>
193 Giris --> <letter>
194 Sonraki sozcuk: 20, sonraki sozcukbirim =
195 Cikis --> <letter>
196 Giris --> <id2>
197 Gecersiz karakter.
198 Cikis --> <id>
199 Cikis --> <var>
200 Giris --> <karsilastirma_op>
201 Sonraki sozcuk: 20, sonraki sozcukbirim =
202 Cikis --> <karsilastirma_op>
203 Giris --> <karsilastirma_op>
204 Sonraki sozcuk: 10, sonraki sozcukbirim 5
```

```
205 Cikis --> <karsilastirma_op>
206 Giris --> <factor>
207 Giris --> <digit>
208 Sonraki sozcuk: 53, sonraki sozcukbirim yap
209 Cikis --> <digit>
210 Cikis --> <factor>
211 Giris --> <do_op>
212 Sonraki sozcuk: 30, sonraki sozcukbirim {
213 Cikis --> <do_op>
214 Program Baslatiliyor...
215 Giris --> <start_op>
216 Sonraki sozcuk: 12, sonraki sozcukbirim sayi
217 Cikis --> <start_op>
218 Giris --> <var>
219 Giris --> <type>
220 Sonraki sozcuk: 11, sonraki sozcukbirim deneme
221 Cikis --> <type>
222 Giris --> <id>
223 Giris --> <letter>
224 Sonraki sozcuk: 20, sonraki sozcukbirim =
225 Cikis --> <letter>
226 Giris --> <id2>
227 Gecersiz karakter.
228 Cikis --> <id>
229 Cikis --> <var>
230 Giris --> <assign_op>
231 Sonraki sozcuk: 10, sonraki sozcukbirim 96
232 Cikis --> <assign_op>
233 Giris --> <expr>
234 Giros --> <term>
235 Giris --> <factor>
236 Giris --> <digit>
237 Sonraki sozcuk: 40, sonraki sozcukbirim eger
238 Cikis --> <digit>
239 Cikis --> <factor>
240 Cikis --> <term>
241 Cikis --> <expr>
242 Giris --> <eger_ifadesi>
243 Sonraki sozcuk: 12, sonraki sozcukbirim sayi
244 Giris --> <var>
245 Giris --> <type>
```

```
246 Sonraki sozcuk: 11, sonraki sozcukbirim x
247 Cikis --> <type>
248 Giris --> <id>
249 Giris --> <letter>
250 Sonraki sozcuk: 20, sonraki sozcukbirim =
251 Cikis --> <letter>
252 Giris --> <id2>
253 Gecersiz karakter.
254 Cikis --> <id>
255 Cikis --> <var>
256 Giris --> <karsilastirma_op>
257 Sonraki sozcuk: 20, sonraki sozcukbirim =
258 Cikis --> <karsilastirma_op>
259 Giris --> <karsilastirma_op>
260 Sonraki sozcuk: 10, sonraki sozcukbirim 5
261 Cikis --> <karsilastirma_op>
262 Giris --> <factor>
263 Giris --> <digit>
264 Sonraki sozcuk: 51, sonraki sozcukbirim ise
265 Cikis --> <digit>
266 Cikis --> <factor>
267 Giris --> <ise>
268 Sonraki sozcuk: 30, sonraki sozcukbirim {
269 Cikis --> <degilse>
270 Program Baslatiliyor...
271 Giris --> <start_op>
272 Sonraki sozcuk: 12, sonraki sozcukbirim sayi
273 Cikis --> <start_op>
274 Giris --> <var>
275 Giris --> <type>
276 Sonraki sozcuk: 11, sonraki sozcukbirim x
277 Cikis --> <type>
278 Giris --> <id>
279 Giris --> <letter>
280 Sonraki sozcuk: 20, sonraki sozcukbirim =
281 Cikis --> <letter>
282 Giris --> <id2>
283 Gecersiz karakter.
284 Cikis --> <id>
285 Cikis --> <var>
286 Giris --> <assign_op>
```

```
287 Sonraki sozcuk: 10, sonraki sozcukbirim 6
288 Cikis --> <assign_op>
289 Giris --> <expr>
290 Giros --> <term>
291 Giris --> <factor>
292 Giris --> <digit>
293 Sonraki sozcuk: 31, sonraki sozcukbirim }
294 Cikis --> <digit>
295 Cikis --> <factor>
296 Cikis --> <term>
297 Cikis --> <expr>
298 Giris --> <finish_op>
299 nextToken: 31
300 Sonraki sozcuk: 45, sonraki sozcukbirim degilse
301 Cikis --> <finish_op>
302 Giris --> <degilse>
303 Sonraki sozcuk: 30, sonraki sozcukbirim {
304 Cikis --> <degilse>
305 Program Baslatiliyor...
306 Giris --> <start_op>
307 Sonraki sozcuk: 12, sonraki sozcukbirim sayi
308 Cikis --> <start_op>
309 Giris --> <var>
310 Giris --> <type>
311 Sonraki sozcuk: 11, sonraki sozcukbirim x
312 Cikis --> <type>
313 Giris --> <id>
314 Giris --> <letter>
315 Sonraki sozcuk: 20, sonraki sozcukbirim =
316 Cikis --> <letter>
317 Giris --> <id2>
318 Gecersiz karakter.
319 Cikis --> <id>
320 Cikis --> <var>
321 Giris --> <assign_op>
322 Sonraki sozcuk: 10, sonraki sozcukbirim 55
323 Cikis --> <assign_op>
324 Giris --> <expr>
325 Giros --> <term>
326 Giris --> <factor>
327 Giris --> <digit>
```

```
328 Sonraki sozcuk: 31, sonraki sozcukbirim }
329 Cikis --> <digit>
330 Cikis --> <factor>
331 Cikis --> <term>
332 Cikis --> <expr>
333 Giris --> <finish_op>
334 nextToken: 31
335 Sonraki sozcuk: 31, sonraki sozcukbirim }
336 Cikis --> <finish_op>
337 Cikis --> <eger_ifadesi>
338 Giris --> <finish_op>
339 nextToken: 31
340 Sonraki sozcuk: 40, sonraki sozcukbirim eger
341 Cikis --> <finish_op>
342 Cikis --> <while_ifadesi>
343 Giris --> <eger_ifadesi>
344 Sonraki sozcuk: 12, sonraki sozcukbirim sayi
345 Giris --> <var>
346 Giris --> <type>
347 Sonraki sozcuk: 11, sonraki sozcukbirim x
348 Cikis --> <type>
349 Giris --> <id>
350 Giris --> <letter>
351 Sonraki sozcuk: 20, sonraki sozcukbirim =
352 Cikis --> <letter>
353 Giris --> <id2>
354 Gecersiz karakter.
355 Cikis --> <id>
356 Cikis --> <var>
357 Giris --> <karsilastirma_op>
358 Sonraki sozcuk: 20, sonraki sozcukbirim =
359 Cikis --> <karsilastirma_op>
360 Giris --> <karsilastirma_op>
361 Sonraki sozcuk: 10, sonraki sozcukbirim 5
362 Cikis --> <karsilastirma_op>
363 Giris --> <factor>
364 Giris --> <digit>
365 Sonraki sozcuk: 51, sonraki sozcukbirim ise
366 Cikis --> <digit>
367 Cikis --> <factor>
368 Giris --> <ise>
```

File - Unnamed

```
369 Sonraki sozcuk: 30, sonraki sozcukbirim {
370 Cikis --> <degilse>
371 Program Baslatiliyor...
372 Giris --> <start_op>
373 Sonraki sozcuk: 12, sonraki sozcukbirim sayi
374 Cikis --> <start_op>
375 Giris --> <var>
376 Giris --> <type>
377 Sonraki sozcuk: 11, sonraki sozcukbirim x
378 Cikis --> <type>
379 Giris --> <id>
380 Giris --> <letter>
381 Sonraki sozcuk: 20, sonraki sozcukbirim =
382 Cikis --> <letter>
383 Giris --> <id2>
384 Gecersiz karakter.
385 Cikis --> <id>
386 Cikis --> <var>
387 Giris --> <assign_op>
388 Sonraki sozcuk: 10, sonraki sozcukbirim 6
389 Cikis --> <assign_op>
390 Giris --> <expr>
391 Giros --> <term>
392 Giris --> <factor>
393 Giris --> <digit>
394 Sonraki sozcuk: 31, sonraki sozcukbirim }
395 Cikis --> <digit>
396 Cikis --> <factor>
397 Cikis --> <term>
398 Cikis --> <expr>
399 Giris --> <finish_op>
400 nextToken: 31
401 Sonraki sozcuk: 45, sonraki sozcukbirim degilse
402 Cikis --> <finish_op>
403 Giris --> <degilse>
404 Sonraki sozcuk: 30, sonraki sozcukbirim {
405 Cikis --> <degilse>
406 Program Baslatiliyor...
407 Giris --> <start_op>
408 Sonraki sozcuk: 12, sonraki sozcukbirim sayi
409 Cikis --> <start_op>
```

```
410 Giris --> <var>
411 Giris --> <type>
412 Sonraki sozcuk: 11, sonraki sozcukbirim x
413 Cikis --> <type>
414 Giris --> <id>
415 Giris --> <letter>
416 Sonraki sozcuk: 20, sonraki sozcukbirim =
417 Cikis --> <letter>
418 Giris --> <id2>
419 Gecersiz karakter.
420 Cikis --> <id>
421 Cikis --> <var>
422 Giris --> <assign_op>
423 Sonraki sozcuk: 10, sonraki sozcukbirim 59599
424 Cikis --> <assign_op>
425 Giris --> <expr>
426 Giros --> <term>
427 Giris --> <factor>
428 Giris --> <digit>
429 Sonraki sozcuk: 31, sonraki sozcukbirim }
430 Cikis --> <digit>
431 Cikis --> <factor>
432 Cikis --> <term>
433 Cikis --> <expr>
434 Giris --> <finish_op>
435 nextToken: 31
436 Sonraki sozcuk: 31, sonraki sozcukbirim }
437 Cikis --> <finish_op>
438 Cikis --> <eger_ifadesi>
439 Giris --> <finish_op>
440 nextToken: 31
441 Sonraki sozcuk: 40, sonraki sozcukbirim eger
442 Cikis --> <finish_op>
443 Cikis --> <while_ifadesi>
444 Giris --> <eger_ifadesi>
445 Sonraki sozcuk: 12, sonraki sozcukbirim sayi
446 Giris --> <var>
447 Giris --> <type>
448 Sonraki sozcuk: 11, sonraki sozcukbirim x
449 Cikis --> <type>
450 Giris --> <id>
```

File - Unnamed

```
451 Giris --> <letter>
452 Sonraki sozcuk: 20, sonraki sozcukbirim =
453 Cikis --> <letter>
454 Giris --> <id2>
455 Gecersiz karakter.
456 Cikis --> <id>
457 Cikis --> <var>
458 Giris --> <karsilastirma_op>
459 Sonraki sozcuk: 20, sonraki sozcukbirim =
460 Cikis --> <karsilastirma_op>
461 Giris --> <karsilastirma_op>
462 Sonraki sozcuk: 10, sonraki sozcukbirim 5
463 Cikis --> <karsilastirma_op>
464 Giris --> <factor>
465 Giris --> <digit>
466 Sonraki sozcuk: 51, sonraki sozcukbirim ise
467 Cikis --> <digit>
468 Cikis --> <factor>
469 Giris --> <ise>
470 Sonraki sozcuk: 30, sonraki sozcukbirim {
471 Cikis --> <degilse>
472 Program Baslatiliyor...
473 Giris --> <start_op>
474 Sonraki sozcuk: 12, sonraki sozcukbirim sayi
475 Cikis --> <start_op>
476 Giris --> <var>
477 Giris --> <type>
478 Sonraki sozcuk: 11, sonraki sozcukbirim x
479 Cikis --> <type>
480 Giris --> <id>
481 Giris --> <letter>
482 Sonraki sozcuk: 20, sonraki sozcukbirim =
483 Cikis --> <letter>
484 Giris --> <id2>
485 Gecersiz karakter.
486 Cikis --> <id>
487 Cikis --> <var>
488 Giris --> <assign_op>
489 Sonraki sozcuk: 10, sonraki sozcukbirim 6
490 Cikis --> <assign_op>
491 Giris --> <expr>
```

```
492 Giros --> <term>
493 Giris --> <factor>
494 Giris --> <digit>
495 Sonraki sozcuk: 31, sonraki sozcukbirim }
496 Cikis --> <digit>
497 Cikis --> <factor>
498 Cikis --> <term>
499 Cikis --> <expr>
500 Giris --> <finish_op>
501 nextToken: 31
502 Sonraki sozcuk: 45, sonraki sozcukbirim degilse
503 Cikis --> <finish_op>
504 Giris --> <degilse>
505 Sonraki sozcuk: 30, sonraki sozcukbirim {
506 Cikis --> <degilse>
507 Program Baslatiliyor...
508 Giris --> <start_op>
509 Sonraki sozcuk: 12, sonraki sozcukbirim sayi
510 Cikis --> <start_op>
511 Giris --> <var>
512 Giris --> <type>
513 Sonraki sozcuk: 11, sonraki sozcukbirim x
514 Cikis --> <type>
515 Giris --> <id>
516 Giris --> <letter>
517 Sonraki sozcuk: 20, sonraki sozcukbirim =
518 Cikis --> <letter>
519 Giris --> <id2>
520 Gecersiz karakter.
521 Cikis --> <id>
522 Cikis --> <var>
523 Giris --> <assign_op>
524 Sonraki sozcuk: 10, sonraki sozcukbirim 59599
525 Cikis --> <assign_op>
526 Giris --> <expr>
527 Giros --> <term>
528 Giris --> <factor>
529 Giris --> <digit>
530 Sonraki sozcuk: 31, sonraki sozcukbirim }
531 Cikis --> <digit>
532 Cikis --> <factor>
```

```
533 Cikis --> <term>
534 Cikis --> <expr>
535 Giris --> <finish_op>
536 nextToken: 31
537 Sonraki sozcuk: 12, sonraki sozcukbirim sayi
538 Cikis --> <finish_op>
539 Cikis --> <eger_ifadesi>
540 Giris --> <var>
541 Giris --> <type>
542 Sonraki sozcuk: 11, sonraki sozcukbirim x
543 Cikis --> <type>
544 Giris --> <id>
545 Giris --> <letter>
546 Sonraki sozcuk: 20, sonraki sozcukbirim =
547 Cikis --> <letter>
548 Giris --> <id2>
549 Gecersiz karakter.
550 Cikis --> <id>
551 Cikis --> <var>
552 Giris --> <assign_op>
553 Sonraki sozcuk: 10, sonraki sozcukbirim 5
554 Cikis --> <assign_op>
555 Giris --> <expr>
556 Giros --> <term>
557 Giris --> <factor>
558 Giris --> <digit>
559 Sonraki sozcuk: 31, sonraki sozcukbirim }
560 Cikis --> <digit>
561 Cikis --> <factor>
562 Cikis --> <term>
563 Cikis --> <expr>
564 Giris --> <finish_op>
565 nextToken: 31
566 Program bitti.
567 Sonraki sozcuk: -1, sonraki sozcukbirim EOF
568 Cikis --> <finish_op>
569
570 Process finished with exit code 0
571
```