PROGRAMMING LANGUAGES
TPL PROJECT – REPORT

GROUP MEMBERS:

Member 1: Mustafa Cem ONAN
ID: 180315064

Member 2: Baran GEZENOĞLU
ID: 170315008

Member 3: Gülben EMİROĞLU
ID: 160315041

EXECUTIVE LECTURER:

Dr. Didem ABİDİN

P.S: Example code files are at the /src directory such as: “code1.txt”, “code2.txt” and “code3.txt”.

P.S: Lookup table, EBNF rules and the state diagrams and example code screenshots are **at the end of this report.**

P.S: Since state diagrams are way too long and complex it may not be fit in width into this word page. Thus, other than this, we included state diagrams original jpeg outputs to **/extras** directory in the .zip file. So you can examine state diagrams in full width from there also.

1) VERY IMPORTANT NOTE: COMPILING INSTRUCTIONS:

In order to compile this program properly, please follow the instructions as follows:

- In this project, the compiler is supposed to be the MinGW (GCC on Windows). Other compilers e.g. MSVC may cause some errors. Make sure you've downloaded and installed the latest version of MinGW on your Windows OS and do not forget to add MinGW compiler directory to your PATH.
- Because of its design, some data structures and variables are externally linked between project files. In order to link & compile entire project properly, make sure you've followed the command-line instructions shown as follows:

```
 Directory of C:\Users\Cem\Desktop\PL_PROJECT\test

25.05.2020  07:27    <DIR>          .
25.05.2020  07:27    <DIR>          ..
25.05.2020  07:27            95.884 executable_name.exe
23.05.2020  11:48               425 kod.txt
24.05.2020  00:14             9.576 lexer.c
24.05.2020  00:18             1.329 lexer.h
24.05.2020  04:52            81.015 parser.c
23.05.2020  12:32             2.000 parser.h
               6 File(s)        190.229 bytes
               2 Dir(s)  132.999.139.328 bytes free

C:\Users\Cem\Desktop\PL_PROJECT\test>gcc lexer.c parser.c -o executable_name

C:\Users\Cem\Desktop\PL_PROJECT\test>
```

- **OPTIONAL :** We had not any compiler warning during the tests. But even so you can add "-w" command to your command-line instructions before "-o" command if you wish to suppress possible compiler warnings nevertheless compiler warnings are not an obstacle to build & run a program properly.

2) PROJECT SUMMARY :

This project is about to operate Lexical Analysis and Syntax Analysis on a text file with the features explained in the assignment.

The files ,they are needed to run this project, are placed in the \src directory in the .zip file. These files can be listed as:

- lexer.c
- lexer.h
- parser.c (main driver)
- parser.h
- code1.txt, code2.txt, code3.txt

Please read instructions before try to build & run this project. (Section 1)

lexer.h and parser.h are the header files, they consist of some function prototypes and constant values.

lexer.c is the program that operates a character-by-character lexical analysis on the passed file. Generated lexemes and tokens are stored in a list and this is going to be passed to parser.c

parser.c is the main driver for operating syntax analysis on a lexeme list. In case of no syntax errors, parser.c is going to generate a working code which literally evaluates all the statements and print the results on the screen. In case of syntax errors exist on code file, parser is going to throw error message that explains what and where exactly is the error.

3) LEXER (LEXICAL ANALYSIS):

Lexer is the program that operates recognizing and categorizing lexemes and tokens by character-by-character reading from a text-file.

In fact, a lexer is a finite automata model. So on each character reading, it's needed to be setted next characters state and type.

```
enum char_t_types {
    HARF = 0,
    RAKAM = 1,
    BILINMEYEN = 99
};
```

In order to hold both next character and its enumerated type, a struct char_t has been implemented to handle these operations more organized.

```
typedef struct char_t {
    char ch;
    int type;
} char_t;
```

Lexemes are also stored as structs so they can be accessed and stored more organized. A lexeme struct has a name, length and a token type. Name is restricted with [MAX_LEX_SIZE] preprocessor definition. It's set to 100 characters by default but it can be adjusted easily.

```c
typedef struct lexeme_t {
    char lexeme[MAX_LEX_SIZE];
    int length;
    int token;
} lexeme_t;
```

Also it has a token type which determines the category of the lexeme. Token types are defined as enum types. So each category can be recognized easily with simple integer values. All the definitions can be found in lexer.h header file.

```c
enum token_types {
    TAMSAYI_SABIT = 10,
    ONDALIK_SABIT = 11,
    KARAKTER_SABIT = 12,
    MANTIKSAL_SABIT = 13,
    YAZI_SABIT = 15,
    ISIM = 20,
    ESITLE_OPERATOR = 30,
    ELEMAN_OPERATOR = 31,
    TOPLA_OPERATOR = 32,
    CIKAR_OPERATOR = 33,
    CARP_OPERATOR = 34,
    BOL_OPERATOR = 35,
    MANTIKSAL_DEGIL_OPERATOR = 36,
    SOL_PARANTEZ = 37,
    SAG_PARANTEZ = 38,
    BIRLESTIR_OPERATOR = 39,
    SOL_INDEX_OPERATOR = 40,
    SAG_INDEX_OPERATOR = 41,
    YAZDIR_OPERATOR = 42,
    KUCUKTUR_OPERATOR = 43,
    ESITTIR_OPERATOR = 44,
    BUYUKTUR_OPERATOR = 45,
    VIRGUL = 50,
    NOKTALI_VIRGUL = 51,
    KAPSAM_BASLANGICI = 52,
    KAPSAM_SONU = 53,
    EGER = 61,
    DEGILSE = 62,
    DONGU = 63,
    TAMSAYI_TIP = 70,
    ONDALIK_TIP = 71,
    KARAKTER_TIP = 72,
    MANTIKSAL_TIP = 73,
    DIZI_TIP = 74,
    YAZI_TIP = 75
};
```

Since we don't know how many lexemes are going to be analyzed; in this project, we'd rather store all the lexemes in a dynamically growing list (similar to Java's ArrayLists) and then after all the lexemes are stored, this list is going to passed into parser for syntax analysis.

```c
typedef struct list_lexeme_t {
    size_t size;
    size_t length;
    lexeme_t *data;
} list_lexeme_t;
```

Since C programming language doesn't have a dynamic arraylist structure, we have implemented a list structure and its functions, that can be created in a desired size, and once its maxed out, it automatically increases its size up to double.

```c
list_lexeme_t *list_lexeme_t_create(size_t s) {
    list_lexeme_t *list = malloc(sizeof *list);

    if (list == NULL)
        return NULL;

    list->data = (lexeme_t *) malloc( _Size: s * sizeof(lexeme_t));

    if (list->data == NULL) {
        free(list);
        return NULL;
    }

    list->size = s;
    list->length = 0;

    return list;
}
```

```c
void list_lexeme_t_add(list_lexeme_t *list, lexeme_t item) {
    if (list->length < list->size) {
        *(list->data + list->length) = item;
        list->length += 1;
    } else {
        list_lexeme_t_enlarge(list);
        list_lexeme_t_add(list, item);
    }
}

void list_lexeme_t_enlarge(list_lexeme_t *list) {
    size_t newSize = 2 * list->size;
    list->data = (lexeme_t *) realloc(list->data, _NewSize: 2 * newSize * sizeof(lexeme_t));
    list->size = newSize;
}
```

void run_lexer(char *file_name) function operates the main process of lexical analysis. First, it opens up the text file which consists of our TPL codes. And reads each character, up to reach the end of file. Character reading and tokenizing subroutines are defined as void read() and int lex() functions.

```c
void run_lexer(char *file_name) {
    fp = fopen(file_name, _Mode: "r");
    lex_list = list_lexeme_t_create( s: 50);
    if (fp != NULL) {
        read();
        do {
            lex();
        } while (next_lex.token != EOF);
    }
    else printf( _Format: "HATA: %s DOSYASI ACILAMADI!\n", file_name);

    fclose(fp);
}
```

void read() function, reads the next character from the text file and states its char_t_type as letter, digit or unknown.

```
void read() {
    next_ch.ch = fgetc(fp);

    if (next_ch.ch != EOF)
        if (isalpha(next_ch.ch))
            next_ch.type = HARF;
        else if (isdigit(next_ch.ch))
            next_ch.type = RAKAM;
        else
            next_ch.type = BILINMEYEN;
    else
        next_ch.type = EOF;
}
```

int lex() function checks next characters state if it is letter, digit or unknown. Whatever type of the next character, in order to get rid of whitespaces, void skip_space() function is invoked once on each call of int lex() function.

```
void skip_space() {
    while (isspace(next_ch.ch))
        read();
}
```

In case of next character is a letter, it checks if it is a first letter of a reserved word. If it is, then int lookup(char ch) function is going to invoked. After checking it from lookup table it's going to be determined the next lexeme is a reserved word or just an ordinary identifier. A letters successor character can be both letter or digit.

```
        case HARF:
            if (next_ch.ch == 'b' ||
                next_ch.ch == 'c' ||
                next_ch.ch == 'd' ||
                next_ch.ch == 'e' ||
                next_ch.ch == 'i' ||
                next_ch.ch == 'k' ||
                next_ch.ch == 'm' ||
                next_ch.ch == 'o' ||
                next_ch.ch == 't' ||
                next_ch.ch == 'y') {
                lookup(next_ch.ch);
                break;
            }
            add();
            read();
            while (next_ch.type == HARF || next_ch.type == RAKAM) {
                add();
                read();
            }
            next_lex.token = ISIM;
            break;
```

In case of next character is a digit, no successor character can be read except digit. It means that the next lexeme can be only integer literal or float literal.

```
case RAKAM:
    add();
    read();
    while (next_ch.type == RAKAM) {
        add();
        read();
    }
    if (next_ch.ch == '.') {
        add();
        read();
        size_t count = 0;
        while (next_ch.type == RAKAM) {
            add();
            read();
            ++count;
        }
        if (count == 0) {
            next_lex.token = EOF;
            break;
        } else next_lex.token = ONDALIK_SABIT;
        break;
    } else
        next_lex.token = TAMSAYI_SABIT;
    break;
```

In case of next character has an unknown type, that means it can be only defined on the lookup table. Because of this, in this case, only void lookup() function called. That state generally represents the special characters e.g seperators, semicolon, parantheses or brackets.

```
case BILINMEYEN:
    lookup(next_ch.ch);
    break;
```

```
case '(':
    add();
    read();
    next_lex.token = SOL_PARANTEZ;
    break;
case ')':
    add();
    read();
    next_lex.token = SAG_PARANTEZ;
    break;
case '[':
    add();
    read();
    next_lex.token = SOL_INDEX_OPERATOR;
    break;
case ']':
    add();
    read();
    next_lex.token = SAG_INDEX_OPERATOR;
    break;
case '{':
    add();
    read();
    next_lex.token = KAPSAM_BASLANGICI;
    break;
```

```
case ';':
    add();
    read();
    next_lex.token = NOKTALI_VIRGUL;
    break;
case '\'':
    read();
    if (isprint(next_ch.ch)) {
        add();
        read();
    } else {
        add();
        next_lex.token = EOF;
    }
    if (next_ch.ch == '\'')
        next_lex.token = KARAKTER_SABIT;
    else {
        add();
        next_lex.token = EOF;
    }
    read();
    break;

default:
    add();
    next_lex.token = EOF;
    break;
```

Finally, if lookup table does not contain the passed character/string or it's the end of file, next token is going to be set as EOF.

```
        default:
            add();
            next_lex.token = EOF;
            break;
```

```
case EOF:
    next_lex.token = EOF;
    strcpy(next_lex.lexeme, _Source: "DOSYA SONU");
    break;
```

After generating next token, next lexeme is going to be added to lex_list which is our lexeme arraylist that we've mentioned earlier.

```
list_lexeme_t_add(lex_list, next_lex);
return next_lex.token;
```

This process is going to be iterated until the lexer reaches the end of file. After this, since we don't want to lose our lexeme list, we did not release the allocated memory on lex_list on purpose. Later on this list is going to be accessed externally in parser and finally the parsing process has done, then it's going to be free'd at the end of parser's main function.

4) <u>OBJECTIVES EXPLAINED IN THE ASSIGNMENT</u>

Since parser contains long long numbers of line of code, we are not going to put screenshot of every single line. Instead of this, we are going to explain what's going on in backend by working code outputs.

All the forward declarations can be found in parser.h header file included with project files.

A. ARITHMETIC OPERATIONS DEFINED ON INT. AND FLOAT:

Arithmetic operations are handled by double arithmetic_operation(size_t *current_ptr) function. Since C does NOT support function overloading, there are some wrapper function with different names also.

Since arithmetic operations does NOT mean anything standalone, it would produce meaningful values only if it used in a complete statement.

```
topla(3,5);
```

```
PL_16_05_2020
C:\Users\Cem\Desktop\PL_16-05-2020\cmake-build-debug\PL_16_05_2020.exe
LUTFEN KODLARIN YAZILI OLDUGU DOSYAYI UZANTISIYLA BIRLIKTE GIRIN:
kod.txt
HATA: YAZILAN KODLAR BIR YARGI ICERMEMEKTEDIR.

Process finished with exit code 1
```

There are four arithmetic operators such as: topla(), cikar(), carp() and bol(). All operators are binary operators so they can have only 2 operands. But since the operands are allowed to be any kind of reference that refers to a numeric literal, it is allowed to call these 4 operators with integer or float literals, variables, array elements (explained later on), or recursively another arithmetic expressions. It means that, unlimited nested arithmetic expressions with precedence is allowed:

```
tamsayi t1;
tamsayi t2;
tamsayi t3;
esitle(t2, 10);
esitle (t3, 20);
esitle(t1, topla(bol(t3, cikar(t3, t2)), carp(topla(t3, 10), cikar(t2, 5))));
yazdir(t1);
```

```
PL_16_05_2020
C:\Users\Cem\Desktop\PL_16-05-2020\cmake-build-debug\PL_16_05_2020.exe
LUTFEN KODLARIN YAZILI OLDUGU DOSYAYI UZANTISIYLA BIRLIKTE GIRIN:
kod.txt
DEGISKEN TIPI: TAMSAYI, ISMI: t1, DEGERI: 152

Process finished with exit code 0
```

By our design preference, all arithmetic operators produce real numbers in order to get precised results. But in assignment process, it’s strictly type checked. If you try to assign a real number ,whose fractional part is 0, to an integer variable, it’s allowed because there are no data loss actually.

```
tamsayi t1;
esitle(t1, topla(-0.5, -0.5));
yazdir(t1);
```

```
PL_16_05_2020
C:\Users\Cem\Desktop\PL_16-05-2020\cmake-build-debug\PL_16_05_2020.exe
LUTFEN KODLARIN YAZILI OLDUGU DOSYAYI UZANTISIYLA BIRLIKTE GIRIN:
kod.txt
DEGISKEN TIPI: TAMSAYI, ISMI: t1, DEGERI: -1

Process finished with exit code 0
```

But in case of you try to assign a real number, whose fractional part differs from 0, to an integer variable, it’s NOT allowed. Because there would be data loss:

```
tamsayi t1;
esitle(t1, topla(-0.5, -0.3));
yazdir(t1);
```

```
PL_16_05_2020
C:\Users\Cem\Desktop\PL_16-05-2020\cmake-build-debug\PL_16_05_2020.exe
LUTFEN KODLARIN YAZILI OLDUGU DOSYAYI UZANTISIYLA BIRLIKTE GIRIN:
kod.txt
HATA: DARALAN TIP DONUSUMUNE (TAMSAYI <- ONDALIK) IZIN VERILMEMEKTEDIR.
HATA YERI: 12. LEXEME
Process finished with exit code 1
```

Besides, there are also some possible logic errors are handled. E.g. if the denominator equals to 0 in a division, an error message would be shown as well:

```
ondalik o;
esitle(o, bol(10, 0));
PL_16_05_2020
C:\Users\Cem\Desktop\PL_16-05-2020\cmake-build-debug\PL_16_05_2020.exe
LUTFEN KODLARIN YAZILI OLDUGU DOSYAYI UZANTISIYLA BIRLIKTE GIRIN:
kod.txt
HATA: BOLME ISLEMINDE PAYDA 0 OLAMAZ.
HATA YERI: 12. LEXEME
Process finished with exit code 1
```

All arithmetic operations can be used with float variables as well:

```
ondalik o;
tamsayi t;
esitle(t, 76);

yazdir(o);
esitle(o, topla(carp(22.6, cikar(t, 68.9)) , bol(carp(t, 0.082), 66.28)));
yazdir(o);
PL_16_05_2020
C:\Users\Cem\Desktop\PL_16-05-2020\cmake-build-debug\PL_16_05_2020.exe
LUTFEN KODLARIN YAZILI OLDUGU DOSYAYI UZANTISIYLA BIRLIKTE GIRIN:
kod.txt
DEGISKEN TIPI: ONDALIK, ISMI: o, DEGERI: 0.000000
DEGISKEN TIPI: ONDALIK, ISMI: o, DEGERI: 160.554025

Process finished with exit code 0
```

B. NOT OPERATION FOR BOOLEANS:

Boolean operations are handled by the function: char *do_bool_operation (size_t *current_ptr).

degil() function represents the boolean NOT operation in our language. It can take any kind of parameter has a reference type of boolean literal.

```
mantiksal m;
esitle(m, degil(yanlis));
yazdir(m);
esitle(m, degil(m));
yazdir(m);
```

```
PL_16_05_2020
C:\Users\Cem\Desktop\PL_16-05-2020\cmake-build-debug\PL_16_05_2020.exe
LUTFEN KODLARIN YAZILI OLDUGU DOSYAYI UZANTISIYLA BIRLIKTE GIRIN:
kod.txt
DEGISKEN TIPI: MANTIKSAL, ISMI: m, DEGERI: dogru
DEGISKEN TIPI: MANTIKSAL, ISMI: m, DEGERI: yanlis

Process finished with exit code 0
```

Also there are 3 extra functions they have a return type of boolean literal such as: buyuktur(), kucuktur() and esittir(). All of these functions takes whether integer and float literals and compare them. So they can be used standalone or can be combined with degil() function.

```
tamsayi t;
esitle(t, topla(10, 20));
mantiksal m;
esitle(m, degil(buyuktur(t, cikar(65.768, t))));
yazdir(m);
```

```
PL_16_05_2020
C:\Users\Cem\Desktop\PL_16-05-2020\cmake-build-debug\PL_16_05_2020.exe
LUTFEN KODLARIN YAZILI OLDUGU DOSYAYI UZANTISIYLA BIRLIKTE GIRIN:
kod.txt
DEGISKEN TIPI: MANTIKSAL, ISMI: m, DEGERI: dogru

Process finished with exit code 0
```

C. CATENATION OPERATION:

Catenation operations are handled by char *do_catenation (size_t *current_ptr) function.

The birlestir() function represents the catenation function in our language. This function can take whether a single character or a string as parameter. So basicly it catenates arguments and returns a new string. Catenation operator can be used nested as well :

```
karakter k;
esitle(k, '1');
yazi y1;
esitle(y1, " elma ");
yazi y2;
yazdir (y2);
esitle(y2, birlestir(y2, birlestir(birlestir(k, y1), birlestir('2', " armut"))));
yazdir(y2);
```

```
PL_16_05_2020
C:\Users\Cem\Desktop\PL_16-05-2020\cmake-build-debug\PL_16_05_2020.exe
LUTFEN KODLARIN YAZILI OLDUGU DOSYAYI UZANTISIYLA BIRLIKTE GIRIN:
kod.txt
DEGISKEN TIPI: YAZI, ISMI: y2, DEGERI:
DEGISKEN TIPI: YAZI, ISMI: y2, DEGERI: 1 elma 2 armut

Process finished with exit code 0
```

D. ARRAY DECLARATIONS:

Array declarations are handled by arr_t *define_arr(size_t *current_ptr) function.

We have introduced, primal operations for primitive data types so far. All the operations that we've mentioned above, can be used with arrays and array elements as well. Our TPL is a dynamic, flexible language. So instead of literals, you can always pass a variable or an array element or an appropriate expression in case of has the same reference type.

In this language, arrays can be declared for all 5 primitive data types such as: TAMSAYI SABIT, ONDALIK SABIT, MANTIKSAL SABIT, KARAKTER SABIT and YAZI SABIT. Arrays are fixed in size so once they allocated on heap they cannot be dynamically grow during the runtime as well as in C style languages.

Lets head over to how to declare an array:

```
tamsayi boyut;
esitle(boyut, 10);

dizi tamsayi td[cikar(boyut, 7)];
dizi ondalik od[boyut];
dizi karakter kd[2];
dizi mantiksal md[4];
dizi yazi yd[3];

yazdir(td);
yazdir(od);
yazdir(kd);
yazdir(md);
yazdir(yd);
```

```
C:\Users\Cem\Desktop\PL_16-05-2020\cmake-build-debug\PL_16_05_2020.exe
LUTFEN KODLARIN YAZILI OLDUGU DOSYAYI UZANTISIYLA BIRLIKTE GIRIN:
kod.txt
DIZI TIPI: TAMSAYI, ISMI: td
0. ELEMANIN DEGERI: 0
1. ELEMANIN DEGERI: 0
2. ELEMANIN DEGERI: 0
DIZI TIPI: ONDALIK, ISMI: od
0. ELEMANIN DEGERI: 0.000000
1. ELEMANIN DEGERI: 0.000000
2. ELEMANIN DEGERI: 0.000000
3. ELEMANIN DEGERI: 0.000000
4. ELEMANIN DEGERI: 0.000000
5. ELEMANIN DEGERI: 0.000000
6. ELEMANIN DEGERI: 0.000000
7. ELEMANIN DEGERI: 0.000000
8. ELEMANIN DEGERI: 0.000000
9. ELEMANIN DEGERI: 0.000000
```

```
DIZI TIPI: KARAKTER, ISMI: kd
0. ELEMANIN DEGERI:
1. ELEMANIN DEGERI:
DIZI TIPI: MANTIKSAL, ISMI: md
0. ELEMANIN DEGERI: yanlis
1. ELEMANIN DEGERI: yanlis
2. ELEMANIN DEGERI: yanlis
3. ELEMANIN DEGERI: yanlis
DIZI TIPI: YAZI, ISMI: yd
0. ELEMANIN DEGERI:
1. ELEMANIN DEGERI:
2. ELEMANIN DEGERI:

Process finished with exit code 0
```

So, as you can understand from previous code examples, in our language it's not needed to initialize variables or array elements. Likewise in C++ and many other modern programming languages, all the primitive types are going to be automatically initialized to their predefined values. This default values can be listed as:

TAMSAYI -> 0
ONDALIK -> 0.000000
MANTIKSAL -> yanlis
KARAKTER -> \0 (Null Character)
YAZI -> "" (Empty String)

Also as you can see, yazdir() function can be used for both variables and array names in order to see their types, names and current values.

E. ARRAY SUBSCRIPTIONS:

Array subscriptions are handled by var_t *get_arr_element (size_t *current_ptr) function.

In order to directly access array elements, we've implemented eleman() function. First argument has to be the array's name and the second argument has to be the index of desired element.

```
dizi tamsayi td[3];
yazdir(td);

esitle(eleman(td, topla(-5, bol(14,2))), topla(5, 3));
esitle(eleman(td, 0), 10);
yazdir(td);
```

```
C:\Users\Cem\Desktop\PL_16-05-2020\cmake-build-debug\PL_16_05_2020.exe
LUTFEN KODLARIN YAZILI OLDUGU DOSYAYI UZANTISIYLA BIRLIKTE GIRIN:
kod.txt
DIZI TIPI: TAMSAYI, ISMI: td
0. ELEMANIN DEGERI: 0
1. ELEMANIN DEGERI: 0
2. ELEMANIN DEGERI: 0
DIZI TIPI: TAMSAYI, ISMI: td
0. ELEMANIN DEGERI: 10
1. ELEMANIN DEGERI: 0
2. ELEMANIN DEGERI: 8

Process finished with exit code 0
```

We’have implemented array index boundary check, so when the user try to access a null element, program throws an error message.

```
dizi tamsayi td[10];

esitle(eleman(td, 20), -99);
```

```
PL_16_05_2020
C:\Users\Cem\Desktop\PL_16-05-2020\cmake-build-debug\PL_16_05_2020.exe
LUTFEN KODLARIN YAZILI OLDUGU DOSYAYI UZANTISIYLA BIRLIKTE GIRIN:
kod.txt
HATA: GIRMIS OLDUGUNUZ INDEKS ILGILI DIZININ SINIRLARININ DISINDA.
HATA YERI: 14. LEXEME
Process finished with exit code 1
```

Also character arrays can be defined and used as well :

```
tamsayi boyut;
esitle(boyut, 10);
dizi karakter k[boyut];

tamsayi i;
dongu(kucuktur(i, boyut)){
    esitle(eleman(k, i), 'A');
    esitle(i, topla(1, i));
}

yazdir(k);
```

```
C:\Users\Cem\Desktop\PL_16-05-2020\cmake-build-debug\PL_16_05_2020.exe
LUTFEN KODLARIN YAZILI OLDUGU DOSYAYI UZANTISIYLA BIRLIKTE GIRIN:
kod.txt
DIZI TIPI: KARAKTER, ISMI: k
0. ELEMANIN DEGERI: A
1. ELEMANIN DEGERI: A
2. ELEMANIN DEGERI: A
3. ELEMANIN DEGERI: A
4. ELEMANIN DEGERI: A
5. ELEMANIN DEGERI: A
6. ELEMANIN DEGERI: A
7. ELEMANIN DEGERI: A
8. ELEMANIN DEGERI: A
9. ELEMANIN DEGERI: A

Process finished with exit code 0
```

(P.S.: Loop structure will be explained later on)

All arithmetic operations also can be used with integer arrays :

```
dizi tamsayi td[6];
yazdir(td);

esitle(eleman(td, 0), 1);
esitle(eleman(td, 2), 2);
esitle(eleman(td, 4), 3);
esitle(eleman(td, 1), carp(eleman(td, 0), 5));
esitle(eleman(td, 3), bol(eleman(td, 2), 2));
esitle(eleman(td, 5), cikar(carp(eleman(td, 1), eleman(td, 3)), topla(eleman(td, 1), eleman(td, 3))));
yazdir(td);
```

```
C:\Users\Cem\Desktop\PL_16-05-2020\cmake-build-debug\PL_16_05_2020.exe
LUTFEN KODLARIN YAZILI OLDUGU DOSYAYI UZANTISIYLA BIRLIKTE GIRIN:
kod.txt
DIZI TIPI: TAMSAYI, ISMI: td
0. ELEMANIN DEGERI: 0
1. ELEMANIN DEGERI: 0
2. ELEMANIN DEGERI: 0
3. ELEMANIN DEGERI: 0
4. ELEMANIN DEGERI: 0
5. ELEMANIN DEGERI: 0
DIZI TIPI: TAMSAYI, ISMI: td
0. ELEMANIN DEGERI: 1
1. ELEMANIN DEGERI: 5
2. ELEMANIN DEGERI: 2
3. ELEMANIN DEGERI: 1
4. ELEMANIN DEGERI: 3
5. ELEMANIN DEGERI: -1

Process finished with exit code 0
```

Also, all arithmetic operations can be used with float arrays as well:

```
dizi ondalik od[5];
esitle(eleman(od, 0), 3.7);
esitle(eleman(od, 2), -8.2);
esitle(eleman(od, 4), 62.7);
esitle(eleman(od, 1), topla(carp(eleman(od,4),cikar(eleman(od,4),eleman(od,2))),bol(carp(eleman(od,4),5),10)));
esitle(eleman(od, 2), bol(eleman(od,1), topla(eleman(od,0),2)));
esitle(eleman(od, 4), carp(10, eleman(od, 2)));
yazdir(od);
```

```
C:\Users\Cem\Desktop\PL_16-05-2020\cmake-build-debug\PL_16_05_2020.exe
LUTFEN KODLARIN YAZILI OLDUGU DOSYAYI UZANTISIYLA BIRLIKTE GIRIN:
kod.txt
DIZI TIPI: ONDALIK, ISMI: od
0. ELEMANIN DEGERI: 3.700000
1. ELEMANIN DEGERI: 4476.780000
2. ELEMANIN DEGERI: 785.400000
3. ELEMANIN DEGERI: 0.000000
4. ELEMANIN DEGERI: 7854.000000

Process finished with exit code 0
```

F. ASSIGNMENT AND CATENATION OF CHAR ARRAYS TO EACH OTHER :

Catenation and assignment operators can be used with array elements as well. In order to operate these instructions, it's just needed to pass array element into birlestir() and assignment() operators:

```
tamsayi boyut;
esitle(boyut, 6);
dizi karakter k1[6];
dizi karakter k2[6];
dizi yazi y[6];

tamsayi i;
dongu(kucuktur(i, boyut)){
    esitle(eleman(k1,i), 'a');
    esitle(i, topla(1,i));
}

esitle(i, 0);
dongu(kucuktur(i, boyut)){
    esitle(eleman(k2,i), 'b');
    esitle(i, topla(1,i));
}
```

```

yazdir(k1);
yazdir(k2);

esitle(i, 0);
dongu(kucuktur(i, boyut)){
    esitle(eleman(y,i), birlestir(eleman(k1, i) , eleman(k2, i)));
    esitle(i, topla(1,i));
}

yazdir(y);
```

(P.S.: Loop structure will be explained later on)

```
C:\Users\Cem\Desktop\PL_16-05-2020\cmake-build-debug\PL_16_05_2020.exe
LUTFEN KODLARIN YAZILI OLDUGU DOSYAYI UZANTISIYLA BIRLIKTE GIRIN:
kod.txt
DIZI TIPI: KARAKTER, ISMI: k1
0. ELEMANIN DEGERI: a
1. ELEMANIN DEGERI: a
2. ELEMANIN DEGERI: a
3. ELEMANIN DEGERI: a
4. ELEMANIN DEGERI: a
5. ELEMANIN DEGERI: a
DIZI TIPI: KARAKTER, ISMI: k2
0. ELEMANIN DEGERI: b
1. ELEMANIN DEGERI: b
2. ELEMANIN DEGERI: b
3. ELEMANIN DEGERI: b
4. ELEMANIN DEGERI: b
5. ELEMANIN DEGERI: b
DIZI TIPI: YAZI, ISMI: y
0. ELEMANIN DEGERI: ab
1. ELEMANIN DEGERI: ab
2. ELEMANIN DEGERI: ab
3. ELEMANIN DEGERI: ab
4. ELEMANIN DEGERI: ab
5. ELEMANIN DEGERI: ab

Process finished with exit code 0
```

G. IF AND WHILE STRUCTURES :

In this project we’have implemented a control flow statement, and also a loop statement as they exist in the C style languages.

If structure is handled by int do_if_if(size_t *current_ptr) and the while loop structure is handled by int do_if_while(size_t *current_ptr) functions.

If statement, as in C style languages, can be in only an if or both if and else form.

While statement is also implemented to be able to handle both situations such as: iteration number is known and unknown.

Both while and if structures must be succeeded by a pair of brackets {} in order to contain statements that they are going to be executed in case of the Boolean condition of the control flow statement/loop statement is true. Same thing is needed to operate properly if there exist a else (degilse) structure.

So despite C-style languages, bracketless if and while statements are not allowed.

Both if and while loop structures can be used nested.

```
tamsayi boyut;
esitle(boyut, 10);

dizi tamsayi t1[boyut];

tamsayi i;
esitle(i, 0);
dongu(kucuktur(i, boyut)){
    eger(esittir(i, 5)){
        esitle(eleman(t1, i), 99);
    }
    degilse{
        esitle(eleman(t1,i), i);
    }
    esitle(i, topla(i,1));
}
yazdir(t1);
```

```
C:\Users\Cem\Desktop\PL_16-05-2020\cmake-build-debug\PL_16_05_2020.exe
LUTFEN KODLARIN YAZILI OLDUGU DOSYAYI UZANTISIYLA BIRLIKTE GIRIN:
kod.txt
DIZI TIPI: TAMSAYI, ISMI: t1
0. ELEMANIN DEGERI: 0
1. ELEMANIN DEGERI: 1
2. ELEMANIN DEGERI: 2
3. ELEMANIN DEGERI: 3
4. ELEMANIN DEGERI: 4
5. ELEMANIN DEGERI: 99
6. ELEMANIN DEGERI: 6
7. ELEMANIN DEGERI: 7
8. ELEMANIN DEGERI: 8
9. ELEMANIN DEGERI: 9

Process finished with exit code 0
```

An example program, declares an integer array with the size of 10 and with a while loop, it assign current loop variable into current array element. If the loop variable (index) equals to 5, then it assigns 99 instead of 5.

H. SUMMARY:

No special keywords needed to start writing codes like start → statements → end. You can directly write your codes.

Every statement (except if and while) must have a semicolon at the end.

All the statements and expressions explained in the assignment has been implemented.

All expressions designed as functions in fact they are binary operators.

e.g. topla(3,5) → 3 + 5

All expressions can be used nested

e.g. topla(3, cikar(11, 6)) → 3 + (11 - 6)

If and while statements can be also used nested.

There are many wrapper and external functions exist in backend C code. We tried to explain everything with simple running code samples.

Except the language rules that we've mentioned above, all the syntax errors are checked. When you made a mistake program terminates itself and returns an error message which tells you what and where exactly is the error occurred.

Here are some syntax error examples :

```
karakter k

PL_16_05_2020
C:\Users\Cem\Desktop\PL_16-05-2020\cmake-build-debug\PL_16_05_2020.exe
LUTFEN KODLARIN YAZILI OLDUGU DOSYAYI UZANTISIYLA BIRLIKTE GIRIN:
kod.txt
HATA: YARGI BELIRTEN IFADELERIN SONUNA ";" KONULMASI GEREKLIDIR.
HATA YERI: 2. LEXEME
Process finished with exit code 1
```

```
qwklsdfjdl;

PL_16_05_2020
C:\Users\Cem\Desktop\PL_16-05-2020\cmake-build-debug\PL_16_05_2020.exe
LUTFEN KODLARIN YAZILI OLDUGU DOSYAYI UZANTISIYLA BIRLIKTE GIRIN:
kod.txt
HATA: YAZILAN KODLAR BIR YARGI ICERMEMEKTEDIR.
HATA YERI: 0. LEXEME
Process finished with exit code 1
```

```
tamsayi t;
esitle(t, topla 3,5));
```

```
PL_16_05_2020
C:\Users\Cem\Desktop\PL_16-05-2020\cmake-build-debug\PL_16_05_2020.exe
LUTFEN KODLARIN YAZILI OLDUGU DOSYAYI UZANTISIYLA BIRLIKTE GIRIN:
kod.txt
HATA: EKSIK "("
HATA YERI: 8. LEXEME
Process finished with exit code 1
```

```
tamsayi t;
mantiksal m;
esitle(t, m);
```

```
PL_16_05_2020
C:\Users\Cem\Desktop\PL_16-05-2020\cmake-build-debug\PL_16_05_2020.exe
LUTFEN KODLARIN YAZILI OLDUGU DOSYAYI UZANTISIYLA BIRLIKTE GIRIN:
kod.txt
HATA: ATANAN IFADEYLE ATANACAK DEGISKENIN TIPLERI AYNI OLMAK ZORUNDADIR.
HATA YERI: 10. LEXEME
Process finished with exit code 1
```

```
dizi tamsayi t[-3];
```

```
PL_16_05_2020
C:\Users\Cem\Desktop\PL_16-05-2020\cmake-build-debug\PL_16_05_2020.exe
LUTFEN KODLARIN YAZILI OLDUGU DOSYAYI UZANTISIYLA BIRLIKTE GIRIN:
kod.txt
HATA: DIZILERIN BOYUTU 1'DEN KUCUK OLAMAZ.
HATA YERI: 4. LEXEME
Process finished with exit code 0
```

```
tamsayi t;
esitle(t, topla(5, dogru));
```

```
PL_16_05_2020
C:\Users\Cem\Desktop\PL_16-05-2020\cmake-build-debug\PL_16_05_2020.exe
LUTFEN KODLARIN YAZILI OLDUGU DOSYAYI UZANTISIYLA BIRLIKTE GIRIN:
kod.txt
HATA: NUMERIK OLMAYAN TIPLER ILE ARITMETIK ISLEM YAPILAMAZ.
HATA YERI: 11. LEXEME
Process finished with exit code 1
```

5. TODO LIST:

A. **LOOKUP TABLE:**

TOKEN NAME	ENUM. PART	RESERVED WORD
ESITLE_OPERATOR	30	esitle
ELEMAN_OPERATOR	31	eleman
TOPLA_OPERATOR	32	topla
CIKAR_OPERATOR	33	cikar
CARP_OPERATOR	34	carp
BOL_OPERATOR	35	bol
MANTIKSAL_DEGIL_OPERATOR	36	degil
SOL_PARANTEZ	37	(
SAG_PARANTEZ	38	)
BIRLESTIR_OPERATOR	39	birlestir
SOL_INDEX_OPERATOR	40	[
SAG_INDEX_OPERATOR	41	]
YAZDIR_OPERATOR	42	yazdir
KUCUKTUR_OPERATOR	43	kucuktur
ESITTIR_OPERATOR	44	esittir
BUYUKTUR_OPERATOR	45	buyuktur
VIRGUL	50	,
NOKTALI_VIRGUL	51	;
KAPSAM_BASLANGICI	52	{
KAPSAM_SONU	53	}
EGER	61	eger
DEGILSE	62	degilse
DONGU	63	dongu
TAMSAYI_TIP	70	tamsayi
ONDALIK_TIP	71	ondalik
KARAKTER_TIP	72	karakter
MANTIKSAL_TIP	73	mantiksal
DIZI_TIP	74	dizi
YAZI_TIP	75	yazi

B. EBNF RULES OF TPL

<program> → {<statement()> }

<statement()> → <do_if_define()> <NOKTALI_VIRGUL> |
<do_if_define_arr()> <NOKTALI_VIRGUL> | <do_if_assignment()>
<NOKTALI_VIRGUL> | <print()> <NOKTALI_VIRGUL> | <do_if_if()> |
<do_if_while()>

<do_if_define()> → <define()>

<define()> → (< TAMSAYI_TIP> | <ONDALIK_TIP> | <MANTIKSAL_TIP>
| <KARAKTER_TIP> | <YAZI_TIP>) <ISIM>

<do_if_define_arr()> → <define_arr()>

<define_arr()> → <DIZI_TIP> (< TAMSAYI_TIP> | <ONDALIK_TIP> |
<MANTIKSAL_TIP> | <KARAKTER_TIP> | <YAZI_TIP>) <ISIM>
<SOL_INDEX_OPERATOR> (<TAMSAYI_SABIT> | <get_arr_element()> |
<if_arithmetic_operation()>) <SAG_INDEX_OPERATOR>

<if_arithmetic_operation()> → <arithmetic_operation()>

<arithmetic_operation()> → (<TOPLA_OPERATOR> |
<CIKAR_OPERATOR> | <CARP_OPERATOR> | <BOL_OPERATOR>)
<SOL_PARANTEZ> <left_operand> <VIRGUL> <right_operand>
<SAG_PARANTEZ>

<left_operand> → <TAMSAYI_SABIT> | <ONDALIK_SABIT> | <ISIM> |
<get_arr_element()> | <if_arithmetic_operation()>

<right_operand> → <TAMSAYI_SABIT> | <ONDALIK_SABIT> | <ISIM> | <get_arr_element()> | <if_arithmetic_operation()>

<get_arr_element()> → <ELEMAN_OPERATOR> <SOL_PARANTEZ> <ISIM> <VIRGUL> (<TAMSAYI_SABIT> | <ISIM> | <get_arr_element()> | <if_arithmetic_operation()>) <SAG_PARANTEZ>

<do_if_assignment()> → <do_assignment()>

<do_assignment()> → <ESITLE_OPERATOR> <SOL_PARANTEZ> (<ISIM> | <get_arr_element()>) <VIRGUL> (<if_bool_operation()> | <if_catenation()> | <get_arr_element()> | <ISIM> | <if_arithmetic_operation()> | <ONDALIK_SABIT> | <TAMSAYI_SABIT> | <KARAKTER_SABIT> | <YAZI_SABIT> | <MANTIKSAL_SABIT>) <SAG_PARANTEZ>

<if_bool_operation()> → <do_bool_operation()>

<do_bool_operation()> → <is_comparison()> | (<MANTIKSAL_DEGIL_OPERATOR> <SOL_PARANTEZ> (<ISIM> | <get_arr_element()> | <MANTIKSAL_SABIT> | <is_comparison()>)<SAG_PARANTEZ>)

<is_comparison()> → <is_less()> | <is_equal()> | <is_greater()>

<is_less()> → <KUCUKTUR_OPERATOR> <SOL_PARANTEZ> <left_operand(2)> | < VIRGUL> <right_operand(2)><SAG_PARANTEZ>

<is_equal()> → <KUCUKTUR_OPERATOR> <SOL_PARANTEZ> <left_operand(2)> | < VIRGUL> <right_operand(2)><SAG_PARANTEZ>

<is_greater()> → <KUCUKTUR_OPERATOR> <SOL_PARANTEZ> <left_operand(2)> | < VIRGUL> <right_operand(2)><SAG_PARANTEZ>

<left_operand(2)> → <TAMSAYI_SABIT> | <ONDALIK_SABIT> | <ISIM> | <get_arr_element()> | <if_arithmetic_operation()>

<right_operand(2)> → <TAMSAYI_SABIT> | <ONDALIK_SABIT> | <ISIM> | <get_arr_element()> | <if_arithmetic_operation()>

<if_catenation()> → <do_catenation()>

<do_catenation()> → <BIRLESTIR_OPERATOR> <SOL_PARANTEZ> <*left_operand> <VIRGUL> <*right_operand> <SAG_PARANTEZ>

<*left_operand> → (<KARAKTER_SABIT> | <YAZI_SABIT> | <ISIM> | <get_arr_element()> | <do_catenation()>)

<*right_operand> → (<KARAKTER_SABIT> | <YAZI_SABIT> | <ISIM> | <get_arr_element()> | <do_catenation()>)

<print()> → <YAZDIR_OPERATOR> <SOL_PARANTEZ> <ISIM> <SAG_PARANTEZ>

<do_if_if()> → <EGER> <SOL_PARANTEZ> (<MANTIKSAL_SABIT> | <ISIM> | <get_arr_element()> | <if_bool_operation>) <SAG_PARANTEZ> <KAPSAM_BASLANGICI> {<statement()>} <KAPSAM_SONU> {<DEGILSE> <KAPSAM_BASLANGICI> {<statement()> } <KAPSAM_SONU>}

<do_if_while()> → <DONGU> <SOL_PARANTEZ> (<MANTIKSAL_SABIT> | <ISIM> | <get_arr_element()> |

<if_bool_operation>) <SAG_PARANTEZ> <KAPSAM_BASLANGICI> {<statement()>} <KAPSAM_SONU>

<TAMSAYI_SABIT> → [-]{RAKAM}

<ONDALIK_SABIT> → [-]{RAKAM}.{RAKAM}

<KARAKTER_SABIT> → ‘<HARF>‘

<MANTIKSAL_SABIT> → dogru | yanlis

<YAZI_SABIT> → “{HARF}“

<ISIM> → <HARF> {(<HARF> | <RAKAM>)}

<ESITLE_OPERATOR> → esitle

<TOPLA_OPERATOR> → topla

<CIKAR_OPERATOR> → cikar

<CARP_OPERATOR> → carp

<BOL_OPERATOR> → bol

<MANTIKSAL_DEGIL_OPERATOR> → degil

<SOL_PARANTEZ> → (// PARANTHESIS IS NOT EBNF SIGN

<SAG_PARANTEZ> →) //PARANTHESIS IS NOT EBNF SIGN

<BIRLESTIR_OPERATOR> → birlestir

<SOL_INDEX_OPERATOR> → [// BRACKET IS NOT EBNF SIGN

<SAG_INDEX_OPERATOR> →] // BRACKET IS NOT EBNF SIGN

<KUCUKTUR_OPERATOR> → kucuktur

<ESITTIR_OPERATOR> → esittir

<BUYUKTUR_OPERATOR> → buyuktur

<VIRGUL> → , // COMMA IS NOT EBNF SIGN

<NOKTALI_VIRGUL> → ; // SEMICOLON IS NOT EBNF SIGN

<KAPSAM_BASLANGICI> → { // BRACE IS NOT EBNF SIGN

<KAPSAM_SONU> → } // BRACE IS NOT EBNF SIGN

<EGER> → eger

<DEGILSE> → degilse

<DONGU> → dongu

<TAMSAYI_TIP> → tamsayi

<ONDALIK_TIP> → ondalik

<KARAKTER_TIP> → karakter

<MANTIKSAL_TIP> → mantiksal

<DIZI_TIP> → dizi

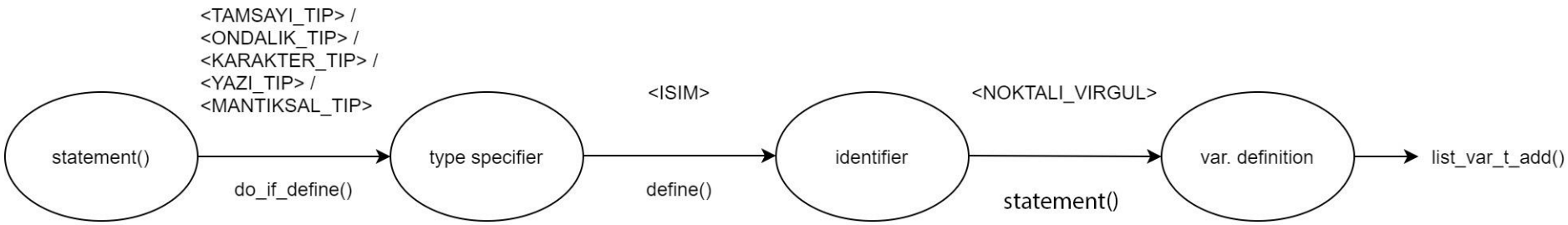
<YAZI_TIP> → yazi

<HARF> → (a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z)

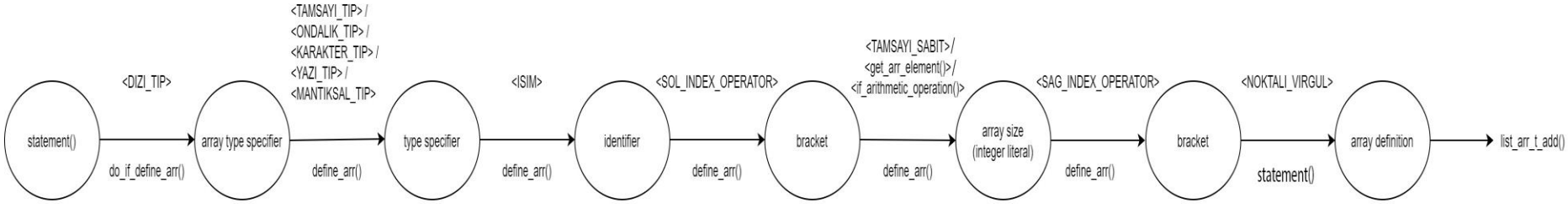
<RAKAM> → (0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9)

<BILINMEYEN> → (space | ! | “ | # | $ | % | & | ‘ | (|) | * | + | , | - | . | / | : | ; | < | = | > | ? | @ | [| \ |] | ^| _ | ` | { | | | } | ~) // THIRD FROM LAST VERTICAL BAR AND (< , > : =) SIGNS ARE NOT EBNF SIGNS

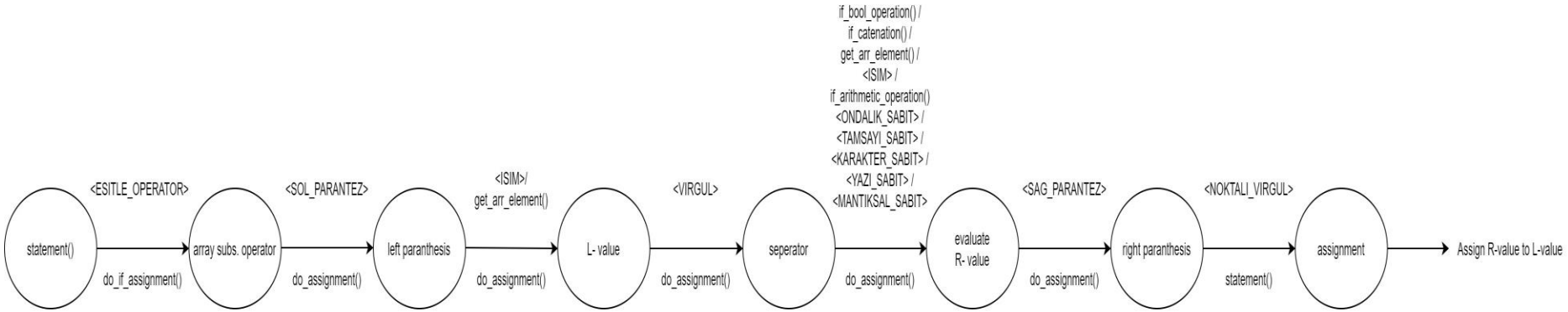
C.STATE DIAGRAMS:



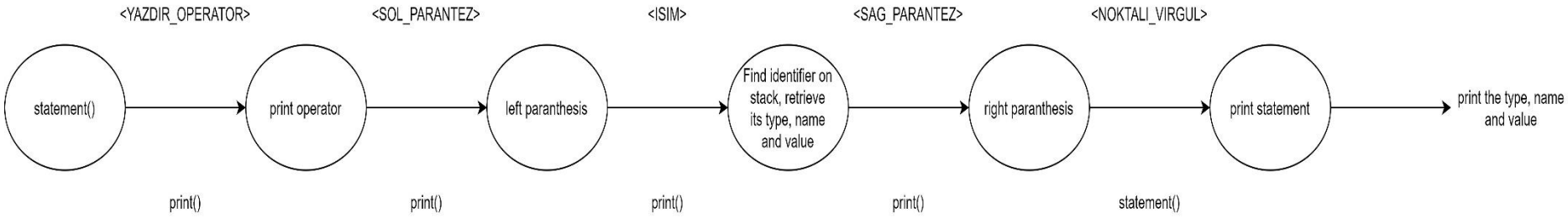
STATE DIAGRAM: VARIABLE DEFINITION



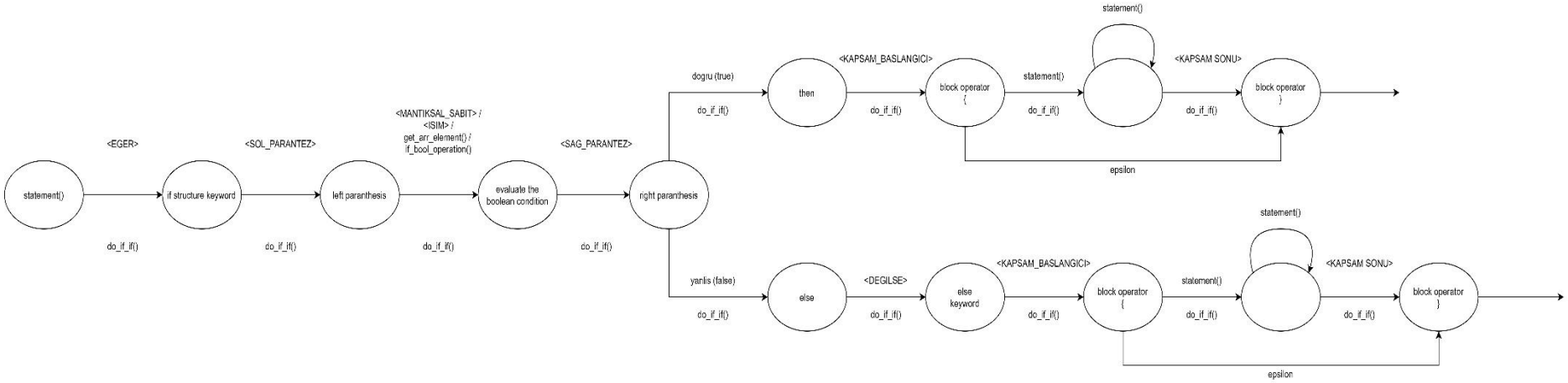
STATE DIAGRAM: ARRAY DEFINITION



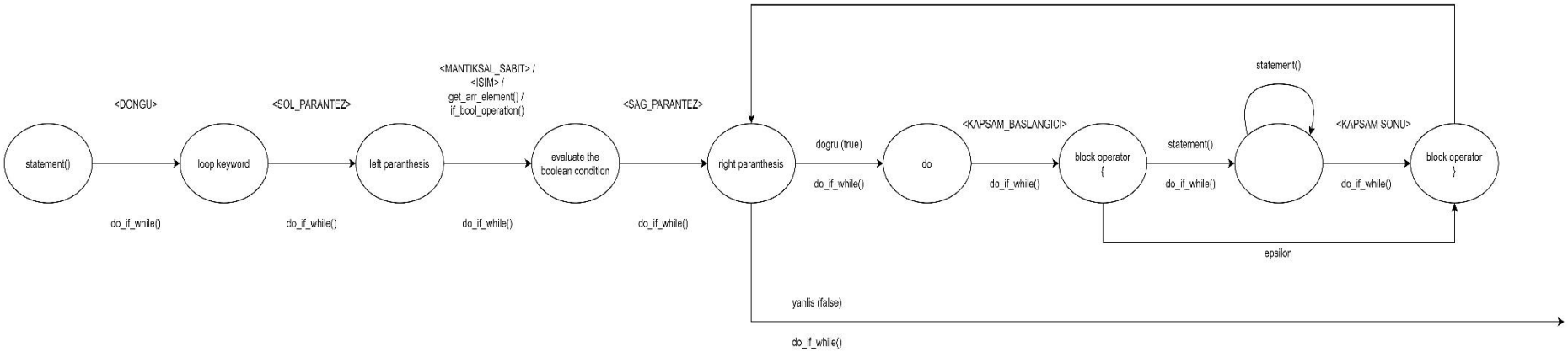
STATE DIAGRAM: ASSIGNMENT STATEMENT



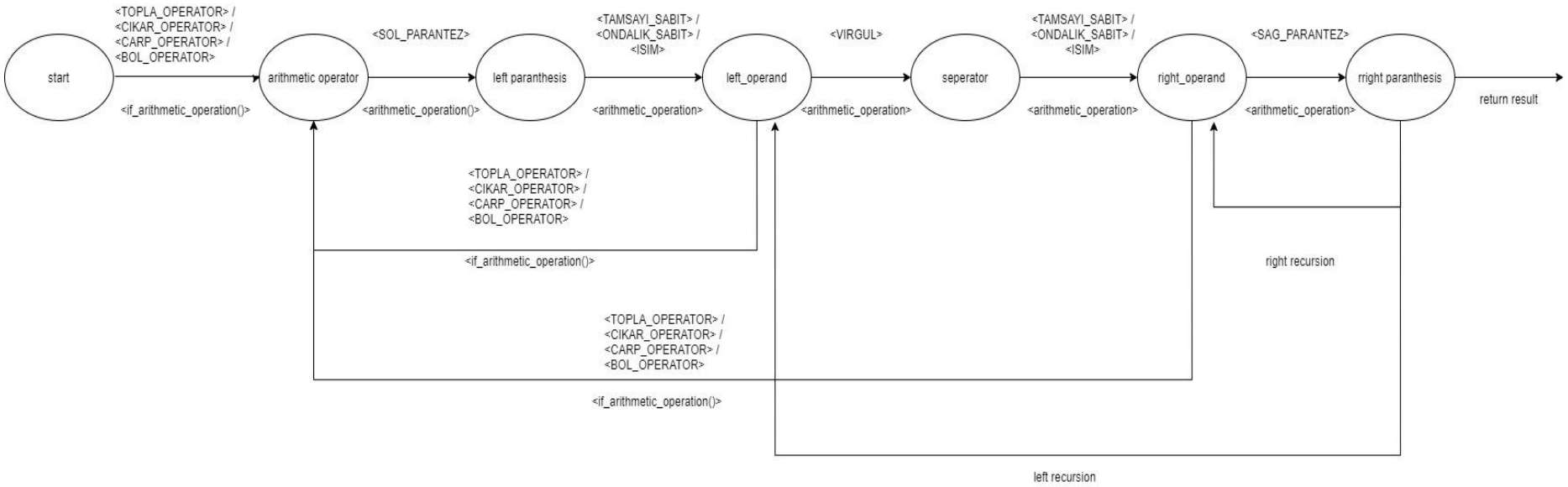
STATE DIAGRAM: print() Statement



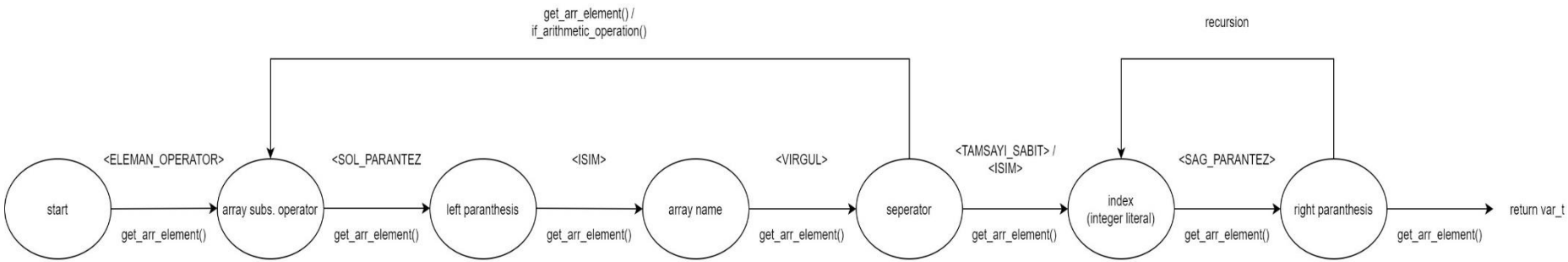
STATE DIAGRAM: IF STATEMENT



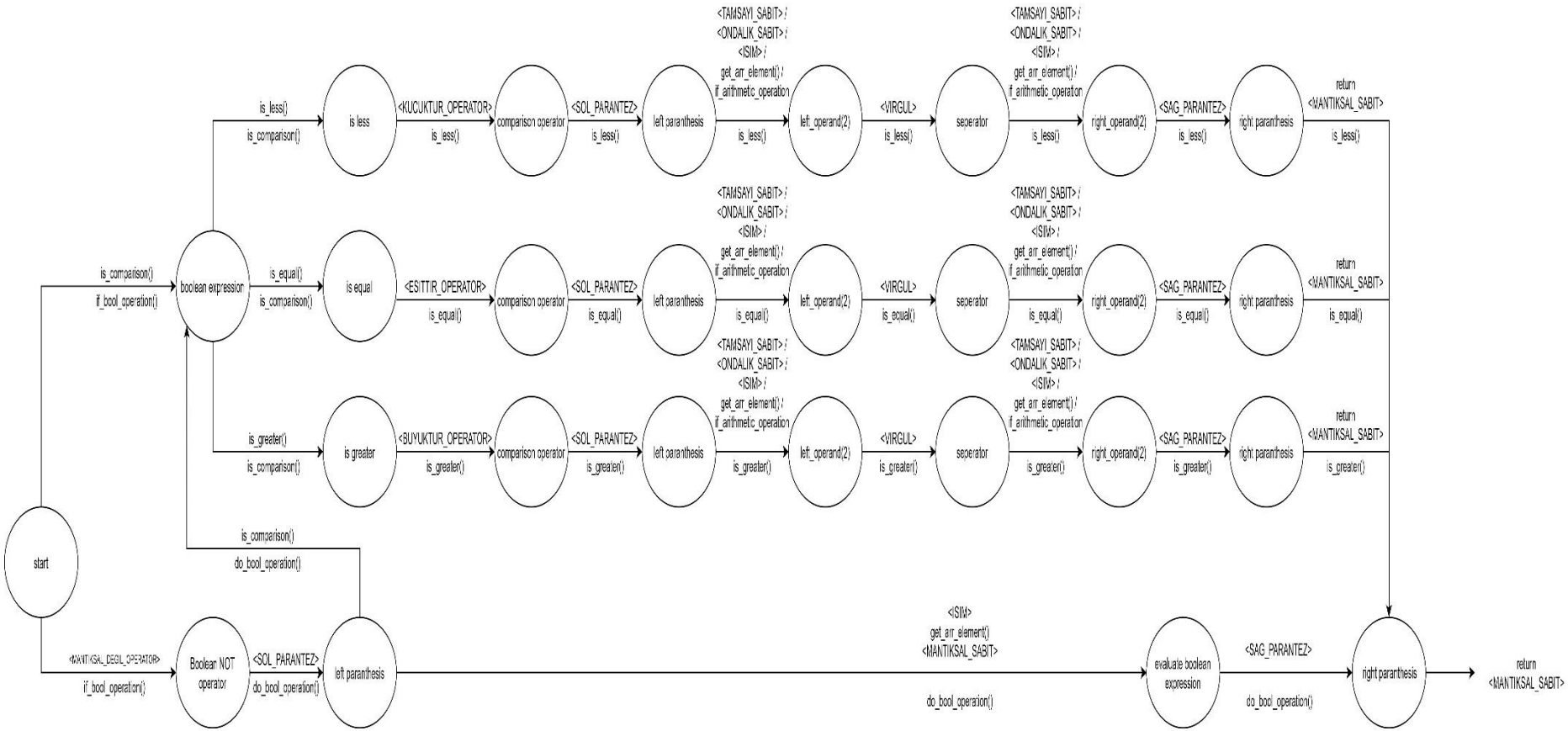
STATE DIAGRAM: WHILE LOOP



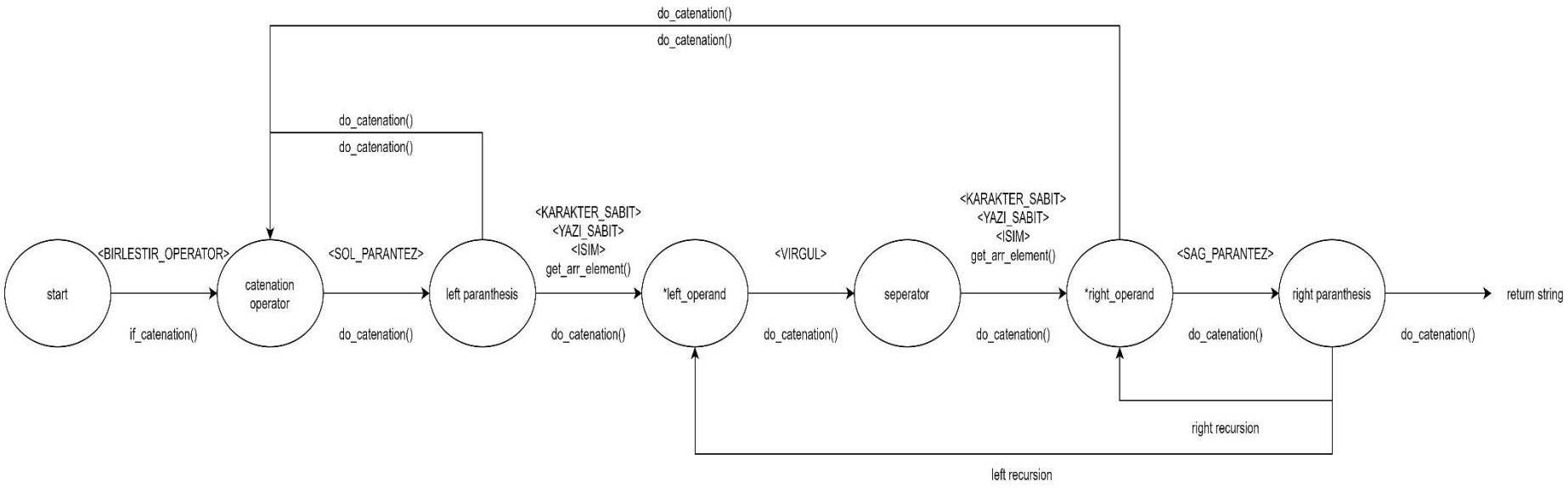
STATE DIAGRAM: ARITHMETIC OPERATION



STATE DIAGRAM: ARRAY SUBSCRIPTION



STATE DIAGRAM : BOOLEAN OPERATIONS



STATE DIAGRAM: CATENATION OPERATION

D.CODE EXAMPLES:

D.1 CODE1.TXT:

You can find the code1.txt in the /src directory in the .zip file. Basicly, it creates an array with the size of 20 (it can be set by changing variable whose name is boyut). And it assigns fibonacci sequence elements into this array. After this, it prints entire array.

parser.c | code1.txt

```
tamsayi boyut;
esitle(boyut, bol(topla(carp(6,2), topla(36,12)) , cikar(17, 14)));
esitle(boyut, cikar(carp(boyut, topla(0.9, 0.45)) , bol(topla(boyut,1), 3)));

dizi tamsayi fibo[topla(bol(boyut,2), carp(boyut, 0.5))];

tamsayi toplam;
tamsayi s1;
tamsayi s2;
esitle(s2, 1);

tamsayi i;
dongu(kucuktur(i, boyut)){
    esitle(eleman(fibo, i), s1);
    esitle(toplam, topla(s1, s2));
    esitle(s1, s2);
    esitle(s2, toplam);
    esitle(i, topla(i,1));
}

yazdir(fibo);
```

```
C:\Users\Cem\Desktop\PL_16-05-2020\cmake-build-debug\PL_16_05_2020.exe
LUTFEN KODLARIN YAZILI OLDUGU DOSYAYI UZANTISIYLA BIRLIKTE GIRIN:
code1.txt
DIZI TIPI: TAMSAYI, ISMI: fibo
0. ELEMANIN DEGERI: 0
1. ELEMANIN DEGERI: 1
2. ELEMANIN DEGERI: 1
3. ELEMANIN DEGERI: 2
4. ELEMANIN DEGERI: 3
5. ELEMANIN DEGERI: 5
6. ELEMANIN DEGERI: 8
7. ELEMANIN DEGERI: 13
8. ELEMANIN DEGERI: 21
9. ELEMANIN DEGERI: 34
10. ELEMANIN DEGERI: 55
11. ELEMANIN DEGERI: 89
12. ELEMANIN DEGERI: 144
13. ELEMANIN DEGERI: 233
14. ELEMANIN DEGERI: 377
15. ELEMANIN DEGERI: 610
16. ELEMANIN DEGERI: 987
17. ELEMANIN DEGERI: 1597
18. ELEMANIN DEGERI: 2584
19. ELEMANIN DEGERI: 4181
CALISTIRMA BASARILI. HATAYA RASTLANMADI.
CIKMAK ICIN BIR TUSA BASIN...


Process finished with exit code 0
```

D.2 CODE2.TXT:

In this example, we are going to create a boolean array whose size of 50. (size can be adjusted by the variable whose name is boyut.) Every i'th element represents the corresponding decimal number from i = 0 up to the i = 49. And program defines if the i is a prime number or not. If it's a prime number, program states the i'th element of the boolean array to true(dogru) else, i'th element of the boolean array is going to be set to false(yanlis) and prints the numbers to the screen So you can find the prime numbers up to desired number.

parser.c | code2.txt

```
tamsayi boyut;
esitle(boyut, 50);
dizi mantiksal asalsaDogruDegilseYanlis[boyut];

tamsayi i;
esitle(i, 3);
tamsayi j;
tamsayi k;
tamsayi asalMi;

esitle(eleman(asalsaDogruDegilseYanlis, 2), dogru);

dongu(kucuktur(i, boyut)){
    esitle(j,2);
    esitle(k,2);
    esitle(asalMi, 1);

    dongu(kucuktur(j, i)){
        esitle(k, j);
        dongu(kucuktur(k, i)){
            eger(esittir(carp(j,k),i)){
                esitle(asalMi, 0);
            }
            esitle(k, topla(k,1));
        }
        esitle(j, topla(j,1));
    }
    eger(esittir(asalMi, 1)){
        esitle(eleman(asalsaDogruDegilseYanlis, i), dogru);
    }
    esitle(i, topla(i,1));
}


yazdir(asalsaDogruDegilseYanlis);
```

```
C:\Users\Cem\Desktop\PL_16-05-2020\cmake-build-debug\PL_16_05_2020.exe
LUTFEN KODLARIN YAZILI OLDUGU DOSYAYI UZANTISIYLA BIRLIKTE GIRIN:
code2.txt
DIZI TIPI: MANTIKSAL, ISMI: asalsaDogruDegilseYanlis
0. ELEMANIN DEGERI: yanlis
1. ELEMANIN DEGERI: yanlis
2. ELEMANIN DEGERI: dogru
3. ELEMANIN DEGERI: dogru
4. ELEMANIN DEGERI: yanlis
5. ELEMANIN DEGERI: dogru
6. ELEMANIN DEGERI: yanlis
7. ELEMANIN DEGERI: dogru
8. ELEMANIN DEGERI: yanlis
9. ELEMANIN DEGERI: yanlis
10. ELEMANIN DEGERI: yanlis
11. ELEMANIN DEGERI: dogru
12. ELEMANIN DEGERI: yanlis
13. ELEMANIN DEGERI: dogru
14. ELEMANIN DEGERI: yanlis
15. ELEMANIN DEGERI: yanlis
16. ELEMANIN DEGERI: yanlis
17. ELEMANIN DEGERI: dogru
18. ELEMANIN DEGERI: yanlis
19. ELEMANIN DEGERI: dogru
20. ELEMANIN DEGERI: yanlis
21. ELEMANIN DEGERI: yanlis
22. ELEMANIN DEGERI: yanlis
23. ELEMANIN DEGERI: dogru
```

```
24. ELEMANIN DEGERI: yanlis
25. ELEMANIN DEGERI: yanlis
26. ELEMANIN DEGERI: yanlis
27. ELEMANIN DEGERI: yanlis
28. ELEMANIN DEGERI: yanlis
29. ELEMANIN DEGERI: dogru
30. ELEMANIN DEGERI: yanlis
31. ELEMANIN DEGERI: dogru
32. ELEMANIN DEGERI: yanlis
33. ELEMANIN DEGERI: yanlis
34. ELEMANIN DEGERI: yanlis
35. ELEMANIN DEGERI: yanlis
36. ELEMANIN DEGERI: yanlis
37. ELEMANIN DEGERI: dogru
38. ELEMANIN DEGERI: yanlis
39. ELEMANIN DEGERI: yanlis
40. ELEMANIN DEGERI: yanlis
41. ELEMANIN DEGERI: dogru
42. ELEMANIN DEGERI: yanlis
43. ELEMANIN DEGERI: dogru
44. ELEMANIN DEGERI: yanlis
45. ELEMANIN DEGERI: yanlis
46. ELEMANIN DEGERI: yanlis
47. ELEMANIN DEGERI: dogru
48. ELEMANIN DEGERI: yanlis
49. ELEMANIN DEGERI: yanlis
CALISTIRMA BASARILI. HATAYA RASTLANMADI.
CIKMAK ICIN BIR TUSA BASIN...


Process finished with exit code 0
```

D.3 CODE3.TXT:

In this example, we are going to create a character array, which actually represents a shuffled string. And also there is integer array which holds the actual orders of letters. What we are going to do is, ordering and catenating this shuffled letters and printing the meaningful result.

parser.c | code3.txt

```
tamsayi kelimeUzunlugu;
esitle(kelimeUzunlugu, 22);
dizi karakter karisik[kelimeUzunlugu];
dizi tamsayi sifreSirasi[kelimeUzunlugu];

esitle(eleman(karisik, 0), 'A');
esitle(eleman(karisik, 1), 'L');
esitle(eleman(karisik, 2), 'E');
esitle(eleman(karisik, 3), 'C');
esitle(eleman(karisik, 4), 'Y');
esitle(eleman(karisik, 5), 'A');
esitle(eleman(karisik, 6), 'B');
esitle(eleman(karisik, 7), 'L');
esitle(eleman(karisik, 8), 'N');
esitle(eleman(karisik, 9), 'U');
esitle(eleman(karisik, 10), 'R');
esitle(eleman(karisik, 11), 'A');
esitle(eleman(karisik, 12), 'R');
esitle(eleman(karisik, 13), 'E');
esitle(eleman(karisik, 14), 'V');
esitle(eleman(karisik, 15), 'I');
esitle(eleman(karisik, 16), 'E');
esitle(eleman(karisik, 17), 'T');
esitle(eleman(karisik, 18), 'I');
esitle(eleman(karisik, 19), 'S');
esitle(eleman(karisik, 20), 'I');
esitle(eleman(karisik, 21), 'S');
```

parser.c | code3.txt

```
esitle(eleman(sifreSirasi, 0), 3);
esitle(eleman(sifreSirasi, 1), 13);
esitle(eleman(sifreSirasi, 2), 7);
esitle(eleman(sifreSirasi, 3), 0);
esitle(eleman(sifreSirasi, 4), 1);
esitle(eleman(sifreSirasi, 5), 6);
esitle(eleman(sifreSirasi, 6), 11);
esitle(eleman(sifreSirasi, 7), 4);
esitle(eleman(sifreSirasi, 8), 5);
esitle(eleman(sifreSirasi, 9), 10);
esitle(eleman(sifreSirasi, 10), 9);
esitle(eleman(sifreSirasi, 11), 8);
esitle(eleman(sifreSirasi, 12), 18);
esitle(eleman(sifreSirasi, 13), 14);
esitle(eleman(sifreSirasi, 14), 2);
esitle(eleman(sifreSirasi, 15), 12);
esitle(eleman(sifreSirasi, 16), 21);
esitle(eleman(sifreSirasi, 17), 15);
esitle(eleman(sifreSirasi, 18), 17);
esitle(eleman(sifreSirasi, 19), 16);
esitle(eleman(sifreSirasi, 20), 19);
esitle(eleman(sifreSirasi, 21), 20);

yazi sirali;
tamsayi i;
dongu(kucuktur(i, kelimeUzunlugu)){
    esitle(sirali, birlestir(sirali, eleman(karisik, eleman(sifreSirasi, i))));
    esitle(i, topla(i,1));
}
yazdir(sirali);
```

```
C:\Users\Cem\Desktop\PL_16-05-2020\cmake-build-debug\PL_16_05_2020.exe
LUTFEN KODLARIN YAZILI OLDUGU DOSYAYI UZANTISIYLA BIRLIKTE GIRIN:
code3.txt
DEGISKEN TIPI: YAZI, ISMI: sirali, DEGERI: CELALBAYARUNIVERSITESI
CALISTIRMA BASARILI. HATAYA RASTLANMADI.
CIKMAK ICIN BIR TUSA BASIN...


Process finished with exit code 0
```