FACULTY OF ENGINEERING

COMPUTER ENGINEERING DEPARTMANT

2021-2022 SPRING

PROGRAMMING LANGUAGES

BUILD YOUR OWN PROGRAMMING LANGUAGE

BASE Programming Language

27.05.2022

190315060 -SENANUR KÖSE **190315078-AKİF TUNÇ**

200315092 -EMRE YILDIZ **190315010-BİRHAT TAŞ**

THE PURPOSE OF THE ASSIGNMENT:

Our aim in this assignment is to create a new and unique programming language by using the programming language creation features that we learned in the programming languages course.

DETAILS AND EXPLANATIONS OF CODES:

```
2   import sys
3
4   with open(sys.argv[1], 'r') as my_file:
5       inputContent = my_file.read()
6   fileIndex = 0
7   EOF = "EOF"
8   INVALID = "INVALID"
9   tokens = []
10  lexemes = []
11  tokens_lines = []
12  lexemes_lines = []
```

We imported sys for use sys.argv[]. After that we created two variables that named EOF and INVALID. We will use them later. Line 9 to line 12 we created 4 empty lists. Later we will append some variables to them.

```
14  # Character classes
15  LETTER = 0
16  DIGIT = 1
17  UNKNOWN = 99
```

In here we created 3 character classes.

LETTER = 0

DIGIT = 1

UNKNOWN = 99

They will check is the input letter or digit or unknown.

```
# Token codes
INT_LIT = 10
IDENT = 11
STRING = 12
ASSIGN_OP = 20
ADD_OP = 21
SUB_OP = 22
MULT_OP = 23
DIV_OP = 24
LEFT_PAREN = 25
RIGHT_PAREN = 26
IF_TOKEN = 41
THEN_TOKEN = 42
FLOAT_TOKEN = 46
COMMA = 48
SEMI_COLON = 49
COLON = 50
EQUALS_TO = 51
GREATER = 52
LESS = 53
PRINT = 54
SINGLE_QUOTE = 55
GOTO_TOKEN = 56
MOD_OP = 57
```

We have assigned a token variable for each token. Then we gave integer values to these token variables.

These values will be used in the later parts of our code to control token values and to act according to token values.

```
lexeme = ""
lexLen = 0
```

We created lexeme = empty string and lexLen = zero integer variables here. We will append lexleme to words and lexLen to 1.

So if lexleme becomes “xyz”

lexLen becomes 3.

```python
def lookupSymbol(character):
    if (character == "("):
        nextToken = LEFT_PAREN
    elif (character == ")"):
        nextToken = RIGHT_PAREN
    elif (character == ">"):
        nextToken = GREATER
    elif (character == "<"):
        nextToken = LESS
    elif (character == "-"):
        nextToken = SUB_OP
    elif (character == "*"):
        nextToken = MULT_OP
    elif (character == "/"):
        nextToken = DIV_OP
    elif (character == ","):
        nextToken = COMMA
    elif (character == ";"):
        nextToken = SEMI_COLON
    elif (character == ":"):
        nextToken = COLON
    elif (character == "'"):
        nextToken = SINGLE_QUOTE
    else:
        nextToken = INVALID

    return nextToken
```

In here we created a function that named lookupSymbol. This function takes a character value (string) inside of it. Then compares character and symbols that our program computes. If character and the symbol matches, nextToken variable takes the value of given token.

For example:

If our character value = “/”

It goes to => elif (character == “/”):

nextToken = DIV_OP

And DIV_OP token value is 24. So our nextToken takes 24 value in given example.

If the symbol corresponding to the character variable is not used in the programming language we created, nextToken becomes INVALID (“INVALID”).

At the end of this function we take the nextToken’s return value.

```
78  def getChar():
79      global inputContent
80      global fileIndex
81      if (fileIndex < len(inputContent)):
82          nextChar = inputContent[fileIndex]
83          fileIndex += 1
84          return nextChar
85      else:
86          return EOF
```

We have created the getChar() function here, which we will use a lot in the later parts of our code. In this function we don't give a value inside of ().

Here we use the inputContent (inputContent = my_file.read()) that we assigned in the 5th line (with using global). The global feature ensures that the changes made within the function are reflected in the entire code.

Also we used fileIndex(value = 0) here.

Then we compare the fileIndex value with the length of the inputContent. Then the function performs operations according to the comparison values.

İf fileIndex >= len(inputContent) the function returns EOF (End of file).

```python
def getNonBlank():
    char = getChar()
    while (char.isspace()):
        char = getChar()
    return char
```

This function uses getChar() func inside and makes char variable = getChar() function's return value. After that If the char variable is " ", it assigns the next letter to the char variable and ignores the space letter. And it checks space letter by using .isspace() method.

End of the function it returns char variable.

```python
def getCharClass(char):
    if char.isalpha():
        charClass = LETTER
    elif char.isdigit():
        charClass = DIGIT
    else:
        charClass = UNKNOWN
    return charClass
```

This function checks whether our char variable is a letter, a number or an unknown value.

It uses .isalpha() method for check if it's letter.

It uses .isdigit() method for check if it's digit.

If it's not digit or letter => CharClass = UNKNOWN

```python
def lex(char):
    lexeme = ""
    charClass = getCharClass(char)
    global fileIndex

    if (charClass == LETTER):
        lexeme += char
        nextChar = getChar()
        while(nextChar != EOF and nextChar != " " and
        (getCharClass(nextChar) == LETTER or
        getCharClass(nextChar) == DIGIT)):
            lexeme += nextChar
            nextChar = getChar()

        # Check for keywords

        if (lexeme == "IF"):
            nextToken = IF_TOKEN
        elif (lexeme == "THEN"):
            nextToken = THEN_TOKEN
        elif (lexeme == "SET"):
            nextToken = ASSIGN_OP
        elif (lexeme == "integer"):
            nextToken = INT_LIT
        elif (lexeme == "float"):
            nextToken = FLOAT_TOKEN
        elif (lexeme == "string"):
            nextToken = STRING
        elif (lexeme == "GOTO"):
            nextToken = GOTO_TOKEN
        elif (lexeme == "ADD"):
            nextToken = ADD_OP
        elif (lexeme == "SUB"):
            nextToken = SUB_OP
        elif (lexeme == "MULT"):
            nextToken = MULT_OP
        elif (lexeme == "DIV"):
        elif (lexeme == "DIV"):
            nextToken = DIV_OP
        elif (lexeme == "MOD"):
            nextToken = MOD_OP
        elif (lexeme == "EQ"):
            nextToken = EQUALS_TO
        elif (lexeme == "GRE"):
            nextToken = GREATER
        elif (lexeme == "LESS"):
            nextToken = LESS
        elif (lexeme == "PRINT"):
            nextToken = PRINT
        else:
            nextToken = IDENT

        if nextChar != " " and nextChar != EOF:
            fileIndex -= 1

    elif (charClass == DIGIT):
        lexeme += char
        nextChar = getChar()
        while((nextChar != EOF) and (nextChar != " ")
        and (getCharClass(nextChar) == DIGIT)):
            lexeme += nextChar
            nextChar = getChar()
        nextToken = INT_LIT
        if nextChar != " " and nextChar != EOF:
            fileIndex -= 1

    elif (charClass == UNKNOWN):
        token = lookupSymbol(char)
        lexeme += char
        nextToken = token

    tokens.append(nextToken)
    lexemes.append(lexeme)
    return nextToken
```

This function does many things. It takes char variable inside of () and uses getCharClass function here.

Then it checks if the charClass variable is letter. If charClass == LETTER:

We append char value to lexeme and if this lexeme becomes a suitible word, nextToken takes the matched token value. Else nextToken = IDENT.

After that it checks if the charClass variable is digit. If charClass == DIGIT:

Lexeme += nextChar

nextToken = INT_LIT

Else charClass is equals to UNKNOWN.

At the end of the function we append nextToken to tokens list and lexeme to lexemes list and return nextToken.

```python
def match_set_code(program, i):
    match program[i]:
        case ((20, _), (25, _), (11, x), (48, _), (10, y), (26, _)):
            globals()[x] = int(y)
        case ((20, _), (25, _), (11, x), (48, _), (11, y), (26, _)):
            globals()[x] = globals()[y]
        case ((20, _), (11, x), (25, _), (21, _), (10, y), (48, _), (10, z), (26, _)):
            globals()[x] = int(y) + int(z)
        case ((20, _), (11, x), (25, _), (22, _), (10, y), (48, _), (10, z), (26, _)):
            globals()[x] = int(y) - int(z)
        case ((20, _), (11, x), (25, _), (23, _), (10, y), (48, _), (10, z), (26, _)):
            globals()[x] = int(y) * int(z)
        case ((20, _), (11, x), (25, _), (24, _), (10, y), (48, _), (10, z), (26, _)):
            globals()[x] = int(y) / int(z)
        case ((20, _), (11, x), (25, _), (21, _), (11, y), (48, _), (11, z), (26, _)):
            globals()[x] = globals()[y] + globals()[z]
        case ((20, _), (11, x), (25, _), (21, _), (11, y), (48, _), (10, z), (26, _)):
            globals()[x] = globals()[y] + int(z)
        case ((20, _), (11, x), (25, _), (22, _), (11, y), (48, _), (11, z), (26, _)):
            globals()[x] = globals()[y] - globals()[z]
        case ((20, _), (11, x), (25, _), (22, _), (11, y), (48, _), (10, z), (26, _)):
            globals()[x] = globals()[y] - int(z)
        case ((20, _), (11, x), (25, _), (23, _), (11, y), (48, _), (11, z), (26, _)):
            globals()[x] = globals()[y] * globals()[z]
        case ((20, _), (11, x), (25, _), (23, _), (11, y), (48, _), (10, z), (26, _)):
            globals()[x] = globals()[y] * int(z)
        case ((20, _), (11, x), (25, _), (24, _), (11, y), (48, _), (11, z), (26, _)):
            globals()[x] = globals()[y] / globals()[z]
        case ((20, _), (11, x), (25, _), (24, _), (11, y), (48, _), (10, z), (26, _)):
            globals()[x] = globals()[y] / int(z)
```

We used Python’s Match-case features here.

This function makes operations on integers and globals()[x], globals()[y], globals()[z].

```python
def match_int_code(program, i):
    match program[i]:
        case ((10, _), (11, x)):
            globals()[x] = 0
        case ((10, _), (11, x), (48, _), (10, y)):
            globals()[x] = int(y)
        case ((10, _), (25, _), (11, x), (48, _), (10, y), (26, _)):
            globals()[x] = int(y)

```

The match_int_code function assigns an integer value to globals()[x].

```python
def match_string_code(program, i):
    match program[i]:
        case ((12, _), (11, x)):
            globals()[x] = ''
        case ((12, _), (11, x), (48, _), (55, _), (11, y), (55, _)):
            globals()[x] = str(y)
        case ((12, _), (25, _), (11, x), (48, _), (55, _), (10, y), (55, _), (26, _)):
            globals()[x] = str(y)
```

The match_string_code function assigns an string value to globals()[x].

```python
def match_add_code(program, i):
    match program[i]:
        case ((21, _), (25, _), (10, x), (48, _), (10, y), (26, _)):
            int(x) + int(y)
        case ((21, _), (25, _), (11, x), (48, _), (10, y), (26, _)):
            globals()[x] = globals()[x] + int(y)
        case ((21, _), (25, _), (11, x), (48, _), (11, y), (26, _)):
            globals()[x] = globals()[x] + globals()[y]
        case ((21, _), (11, x), (48, _), (10, y)):
            globals()[x] = globals()[x] + int(y)
        case ((22, _), (25, _), (11, x), (48, _), (11, y), (26, _)):
            globals()[x] = globals()[x] - globals()[y]
        case ((22, _), (11, x), (48, _), (10, y)):
            globals()[x] = globals()[x] - int(y)
        case ((23, _), (25, _), (11, x), (48, _), (11, y), (26, _)):
            globals()[x] = globals()[x] * globals()[y]
        case ((23, _), (11, x), (48, _), (10, y)):
            globals()[x] = globals()[x] * int(y)
        case ((24, _), (25, _), (11, x), (48, _), (11, y), (26, _)):
            globals()[x] = globals()[x] / globals()[y]
        case ((24, _), (11, x), (48, _), (10, y)):
            globals()[x] = globals()[x] / int(y)
```

This function allows us to operate on value.

For Example:

globals()[x] = globals()[x] + int(y) / apple = 5, banana = 10

apple = apple + 15

apple = 20

banana = apple + banana

banana = 15

```python
def match_if_code(program, i):
    if(program[i][-2][0] == 56):
        match program[i]:
            case ((41, _), (11, x), (_, _), (11, y), (42, _), (56, _), (11, z)):
                match program[i][2][0]:
                    case 51:
                        if(globals()[x] == globals()[y]):
                            match_goto_code(program, i)
                    case 52:
                        if(globals()[x] > globals()[y]):
                            match_goto_code(program, i)
                    case 53:
                        if(globals()[x] < globals()[y]):
                            match_goto_code(program, i)
    elif (program[i][2][0] == 57):
        match_mod_code(program, i)

    else:
        match program[i][:5]:
            case ((41, _), (11, x), (_, _), (11, y), (42, _)):
                def fun():
                    num = 5 - len(program[i])
                    a_list = []
                    a_list.append(program[i][num:])
                    match_operator(a_list, 0)

                match program[i][2][0]:
                    case 51:
                        if(globals()[x] == globals()[y]):
                            fun()
                    case 52:
                        if(globals()[x] > globals()[y]):
                            fun()
                    case 53:
                        if(globals()[x] < globals()[y]):
                            fun()
```

This function allows us to use the if property in our code.

```python
def match_print_code(program, i):
    match program[i]:
        case ((54, _), (11, x)):
            print(globals()[x])
        case ((54, _), (55, _), (11, x), (55, _)):
            print(str(x))
        case ((54, _), (25, _), (21, _), (25, _), (11, x), (48, _), (11, y), (26, _), (26, _)):
            print(globals()[x] + globals()[y])
        case ((54, _), (25, _), (22, _), (25, _), (11, x), (48, _), (11, y), (26, _), (26, _)):
            print(globals()[x] - globals()[y])
        case ((54, _), (25, _), (23, _), (25, _), (11, x), (48, _), (11, y), (26, _), (26, _)):
            print(globals()[x] * globals()[y])
        case ((54, _), (25, _), (24, _), (25, _), (11, x), (48, _), (11, y), (26, _), (26, _)):
            print(globals()[x] / globals()[y])
```

match_print_code function allows us to use print property in our code.

It allows us top print a str(x) value

And it gives us the option to operate and print between x,y values.

For example:

Print(x / y) (x = 10, y = 5)

Output: 2

```python
def match_goto_pos(program, i):
    globals()[program[i][0][1]] = i


def match_goto_code(program, i):
    match program[i]:
        case ((56, _), (11, _)):
            for index in range(globals()[program[i][1][1]], i):
                print(index+1)
                match_operator(program, index)
        case (_, _, _, _, _, (56, _), (11, x)):
            for index in range(globals()[x], i+1):
                match_operator(program, index)
```

This function is using for goto operation in our own programming language.

```python
def match_mod_code(program, i):
    match program[i]:
        case((41, _), (11, x), (57, _), (11, y), (51, _), (11, z), (42, _), (54, _), (11, k)):
            if((globals()[x] % globals()[y]) == globals()[z]):
                print(globals()[k])
```

match_mod_code function is calculating mode of values.

```python
def match_operator(program, i):
    match program[i][0][0]:
        case 10:
            match_int_code(program, i)
        case 11:
            match_goto_pos(program, i)
        case 20:
            match_set_code(program, i)
        case 21:
            match_add_code(program, i)
        case 54:
            match_print_code(program, i)
        case 12:
            match_string_code(program, i)
        case 41:
            match_if_code(program, i)
        case 56:
            match_goto_code(program, i)
        case _:
            raise TypeError("not a operator we support")
```

This is our program's last match function. It checks operator type by token values.

If case _:

Raise TypeError

It throws error if token values does not match.

```python
def merge(list1, list2):

    merged_list = tuple(zip(list1, list2))
    return merged_list
```

Merge function takes list1 and list2 lists inside of it. Then zip's list1 and list2, convert's this zip to a tüple and returns it.

```python
def first_list_partition(list, x):
    return list[:list.index(x)]


def second_list_partition(list, x):
    return list[list.index(x)+1:]
```

These functions partitates list values with using .index() method. Then returns them.

```python
def getLines_addAnother(list, x, list_to_add):
    if(x not in list):
        return

    list_to_add.append(first_list_partition(list, x))
    getLines_addAnother(second_list_partition(list, x), x, list_to_add)
```

getLines_addAnouther append's parted list values to list_to_add. And it uses itself again.

First of appends first_list_partition. Then appends second_list_partition.

```python
def main():
    nextChar = getNonBlank()
    if (nextChar == EOF):
        print("File is empty")
        return

    while nextChar != EOF:
        nextToken = lex(nextChar)
        if (nextToken == INVALID):
            break
        nextChar = getNonBlank()

    getLines_addAnother(lexemes, ';', lexemes_lines)
    getLines_addAnother(tokens, SEMI_COLON, tokens_lines)
    main_list = merge_lists_toTuple(tokens_lines, lexemes_lines)
    for i in range(len(main_list)):
        match_operator(main_list, i)
```

This is the programs main function. Everything that we created runs in here.

INPUT 1:

Testing input like identifying, setting a value and expressions with them.

test.base

```
SET (kalde, 10);
integer apple;
integer banana, 25;
integer (carrot, 35);
ADD (banana, carrot);
ADD apple, 10;
SET melon (MULT 3, 15);
PRINT melon;
PRINT apple;
integer salt, 10;
PRINT (ADD (apple, banana));
PRINT (SUB (carrot, banana));
PRINT (MULT (kalde, apple));
PRINT 'HelloWorld';
string word, 'kelime';
PRINT word;
SET watermelon (SUB 20, 15);
IF apple LESS carrot THEN PRINT carrot;
IF apple EQ salt THEN PRINT (ADD (banana, kalde));
```

OUTPUT 1:

C:\WINDOWS\py.exe

```
45
10
70
-25
100
HelloWorld
kelime
35
70
```

INPUT 2:

A program that prints range of numbers(1 to 100) with a loop.

loop.base

```
integer number, 0;
integer finall, 100;
LOOP;
ADD (number, 1);
PRINT number;
IF number LESS finall THEN GOTO LOOP;
```

OUTPUT:

C:\WINDOWS\py.exe

```
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

Starts from 1

INPUT 3:

A program that outputs range of dividable numbers to 2.

dividable2.base

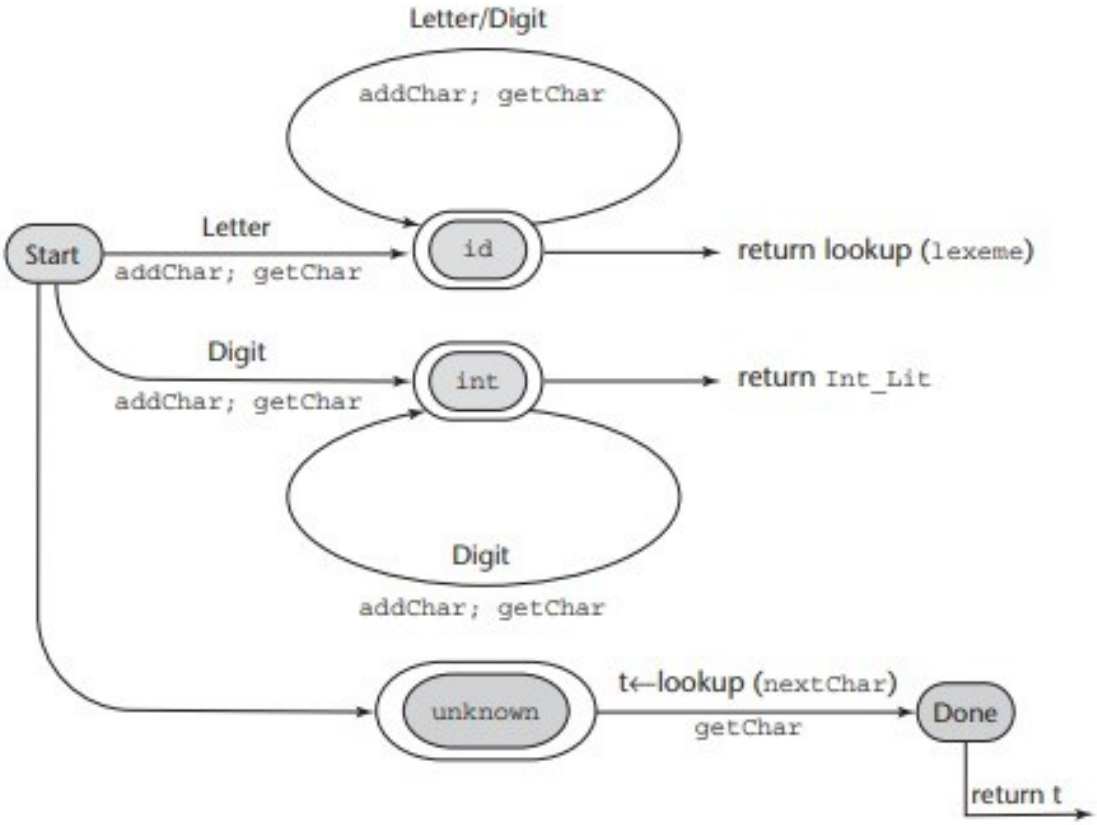
```
integer number, 0;
integer finall, 10;
integer divident, 2;
integer zero;
LOOP;
ADD (number, 1);
IF number MOD divident EQ zero THEN PRINT number;
IF number LESS finall THEN GOTO LOOP;
```

OUTPUT:

C:\WINDOWS\py.exe

```
2
4
6
8
10
```

STATE DIAGRAM:



“NAME” = BASE (Birhat Akif Sena Emre)

“AUTHORS” = Emre Yıldız, Senanur Köse, Akif Tunç, Birhat Taş

“VERSION” = 2022, ORIGINAL

“Start Symbol” = <Lines>

ID = {Letter}[x]

STRING = “{STRING CHARS}” {Letter} {Digit} {Symbol}

<Lines> = Integer <stmt> NewLine <Lines>

| Integer <stmt> NewLine

<Statements> = <Statement> ':' <Statements>

| <Statement>

<stmt> = CLOSE “ ; ”

<expr> = STEP INTEGER

| GOTO <expr>

| GOSUB <expr>

| IF <expr> then <stmt>

<Compare Expr> = <Add Expr> '=' <Compare Expr>

| <Add Expr> '>' <Compare Expr>

| <Add Expr> '>=' <Compare Expr>

| <Add Expr> '<' <Compare Expr>

| <Add Expr> '<=' <Compare Expr>

…

<Add_expr> => <Mult_expr> ‘ + ‘ <Add_expr>

<Statements> ::= <Statement> ':' <Statements>

| <Statement>

| <Add Exp>

<Constant> = Integer

| String

Conclusion: We have implemented a basic programming language with our team. Although we understand that the process is difficult, we have seen that is is not impossible. If we have an idea that solves a problem, we've gained the background knowledge in this project here to build a programming language in the future.

We built our compiler with python version 3.10. Because this version has a pleasant feature that named ‘pattern matching’. We used this feature based on the explanations of our textbook.

In case you do not have python 3.10 installed on your computer, we have uploaded a video of the codes to the link below.

https://cbuedu-my.sharepoint.com/:v:/g/personal/200315092_ogr_cbu_edu_tr/EYRvF8UGmJpFiriJGjUjzh0B6Fz4yysf-YQV5s3lRmFVBg?e=cvRBLP

And also we collabrate in a github address for this project.

https://github.com/dev-emre-yildiz/BASE/