Al Imam Mohammad ibn Saud Islamic University

College of Computer and Information Sciences

Computer Science Department

Second Semester 1443 H – 2022 G

Compiler Project Report

Phase #1

By

Nasser Alkhuraiji (439011631)

Abdulilah Alqasem (439014750)

Mohammad Zouman (439012798)

Supervisor

Dr. GaneshKumar Perumal

April 6, 2022

Table of Contents

1 Regular Expressions (RE)

- **Delimiters** = r"[\,|\s|\(|\)|\[|\]|\{|\}]"
- **Operaters** = r"[*|-|/|+|%|=|<|>|^<=$|^>=$]"
- **Preemtive Types** = r"^boolean$|^byte$|^char$|^short$|^int$|^long$|^float$|^double$"
- **None Preemtive Types** = r"^string$|^array$|^class$"
- **Keywords** = r"^var$|^and$|^or$|^not$|^if$|^elif$|^else$|^for$|^to$|^step$|^while$|^fun$|^then$|^end$|^return$|^continue$|^break$|^print$"
- **Float Number** = r"[+-]?[0-9]+[.,][0-9]+"
- **Integer Number** = r"[+-]?[0-9]"
- **String** = r"[A-Za-z0-9_./\-]*"
- **Character** = r"'[0-9]'|'[a-zA-Z]'"

2 Context Free Grammar (CFG)

Integer:

S -> Integer

Sign -> +|-|e

Integer -> Sign,Digit,Intger | Sign,Digit

Digit -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Float: S -> Float

Sign -> +|-|e

Float -> Sign,Digit,Dot,Digit Float| Sign,Digit,Dot,Digit

Digit -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Dot -> .

Character: S -> Char

Char -> Quotation,(Digit | Character),Quotation,Char | Quotation,(Digit | Character),Quotation

Digit -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Character -> 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

Quotation -> '

String: S -> String

String -> Quotation,(Digit | Character),Quotation ,String | Quotation,(Digit | Character),Quotation

Digit -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Character -> 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

Quotation -> "

Identifier: S -> Line

Line-> Datatype Identifer Equal Type

Datatype -> boolean| byte | char | short | int | long | float | double | string | array

Identifer -> Character,Identifer | Character,Digit,Identifer | Character | Character,Digit

Digit -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Character -> 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

Equal -> =

Type -> Identifer | Integer | String | Char | Float | True | False

---

Operator: S -> Expression

Expression-> Expression Operator Expression

Expression-> (Expression)

Expression-> Integer | Float

Operator -> *|-|/|+|%|=|<|>|<=|>=

3 Code

```
import re
import sys
from tkinter import *

delimiters = [';', '.', ',', '(', ')', '[', ']', '{', '}', ' ', '\n']
re_delimiters = r"[\,|\s|\(|\)|\[|\]|\{|\}]"
re_operaters = r"[*|-|/|+|%|=|<|>|^<=$|^>=$]"
re_preemtive_types =
r"^boolean$|^byte$|^char$|^short$|^int$|^long$|^float$|^double$"
re_non_preemtive_types = r"^string$|^array$|^class$"
re_keywords =
r"^var$|^and$|^or$|^not$|^if$|^elif$|^else$|^for$|^to$|^step$|^while$|^fun$|^then
$|^end$|^return$|^continue$|^break$|^print$"

re_float_number = r"[+-]?[0-9]+[.,][0-9]+"
re_integer_number = r"[+-]?[0-9]"
re_string = r"[A-Za-z0-9_./\-]*"
re_char = r"'[0-9]'|'[a-zA-Z]'"


def ifOperator(word):
    if re.match(re_operaters, word):
        return True
    return False


def ifPreemtiveType(word):
    if re.match(re_preemtive_types, word, re.IGNORECASE):
        return True
    return False


def ifNonPreemtiveType(word):
    if re.match(re_non_preemtive_types, word, re.IGNORECASE):
        return True
    return False


def ifKeyword(word):
    if re.match(re_keywords, word, re.IGNORECASE):
        return True
    return False
```

```python
def ifFloat(word):
    if re.match(re_float_number, word):
        return True
    return False


def ifInteger(word):
    if re.match(re_integer_number, word):
        # if re.search(r"[a-zA-Z]|\W[0-9]\W",word):
        # return False
        return True
    return False


def ifString(word):
    if re.match(re_string, word):
        return True
    return False


def ifChar(word):
    if re.match(re_char, word):
        return True
    return False


def ifDelimiter(word):
    if re.match(re_delimiters, word):
        return True
    return False


def ifEndStatement(word):
    if ';' in word:
        return True
    return False


def dfa(word, index_of_word, line_number):
    global flag_datatype
    global flag_identifier
    global flag_keyword
    if index_of_word == 0:
        if ifKeyword(word):
```

```
            tokens.append(['KEYWORD', word])
            tokens_list.append((word, line_number, 'KEYWORD'))
            flag_keyword = True
            return
        if ifPreemtiveType(word):
            tokens.append(['DATATYPE', word])
            tokens_list.append((word, line_number, 'DATATYPE'))
            return
        if ifNonPreemtiveType(word):
            tokens.append(['DATATYPE', word])
            tokens_list.append((word, line_number, 'DATATYPE'))
            return
        flag_datatype = True
        errors_list.append(
            ["ERORR at line #{}: TYPO[DATATYPE, KEYWORD]. [{}]".format(i+1,
word)])
        return
        # sys.exit("ERORR at line #{}: TYPO. [{}]".format(i+1, word))

    # identify identifiers.
    # if the last token was a datatype and was a typo then the next token must be
an IDENTIFIER.
    if flag_datatype == True:
        if re.match("[a-zA-Z]([a-zA-Z]|[0-9])*", word):
            tokens.append(['IDENTIFIER', word])
            tokens_list.append((word, line_number, 'IDENTIFIER'))
            flag_datatype = False
            return
        else:
            flag_identifier = True
            errors_list.append(
                ["ERORR at line #{}: INVALID IDENTIFIER NAME[IDENTIFIER].
[{}]".format(i+1, word)])
            return
            # sys.exit("ERORR at line #{}: INVALID IDENTIFIER NAME.
[{}]".format(i+1, word))

    if flag_identifier == True:
        if ifOperator(word) == True:
            tokens.append(['OPERATOR', word])
            tokens_list.append((word, line_number, 'OPERATOR'))
            flag_identifier == False
            return

    if tokens[len(tokens) - 1][0] == 'DATATYPE':
```

```
        if re.match("[a-z]|[A-Z]", word):
            tokens.append(['IDENTIFIER', word])
            tokens_list.append((word, line_number, 'IDENTIFIER'))
            return
        else:
            flag_identifier = True
            errors_list.append(
                ["ERORR at line #{}: INVALID IDENTIFIER NAME[IDENTIFIER].
[{}]".format(i+1, word)])
            return
            # sys.exit("ERORR at line #{}: INVALID IDENTIFIER NAME.
[{}]".format(i+1, word))

    if flag_keyword == True:
        if ifDelimiter(word) == True:
            tokens.append(['DELIMITER', word])
            tokens_list.append((word, line_number, 'DELIMITER'))
            flag_keyword == False
            return
        # keyword -> identifier.
        if re.match("[a-z]|[A-Z]", word):
            tokens.append(['IDENTIFIER', word])
            tokens_list.append((word, line_number, 'IDENTIFIER'))
            flag_keyword == False
            return

    # DELIMITER.
    if ifDelimiter(word) == True:
        tokens.append(['DELIMITER', word])
        tokens_list.append((word, line_number, 'DELIMITER'))
        return

    # identify END STATEMENTS.
    if ifEndStatement(word) == True:
        tokens.append(['END STATEMENT', word])
        tokens_list.append((word, line_number, 'END STATEMENT'))
        return

    # identify operators.
    if ifOperator(word) == True:
        tokens.append(['OPERATOR', word])
        tokens_list.append((word, line_number, 'OPERATOR'))
        return

    # identify FLOAT.
```

```
    if ifFloat(word):
        tokens.append(["FLOAT", word])
        tokens_list.append((word, line_number, 'FLOAT'))
        return

 # identify integer.
    if ifInteger(word):
        tokens.append(["INTEGER", word])
        tokens_list.append((word, line_number, 'INTEGER'))
        return

    # identify Character.
    if ifChar(word):
        tokens.append(["CHARACTER", word])
        tokens_list.append((word, line_number, 'CHARACTER'))
        return

 # identify STRING
    if ifString(word):
        tokens.append(["STRING", word])
        tokens_list.append((word, line_number, 'STRING'))
        return

    errors_list.append(
        ["ERORR at line #{}: ILLEGAL CHARACTER. [{}]".format(i+1, word)])
    return True


def writeFile():
    file = open('Editor.txt', 'w+')
    file.write(text.get('1.0', 'end') + '\n')
    file.close()
    gui.destroy()


gui = Tk()
gui.title("Pseudocode Editor - [Lexical Analyzer]")
gui.geometry("1000x750+250+25")

text = Text(gui, wrap=WORD, font=('Courier 15 bold'))
text.pack(side=LEFT, expand=True, fill=BOTH)
text.place(x=10, y=10, width=980, height=680)

button = Button(gui)
button.config(text='Write To File', command=writeFile)
```

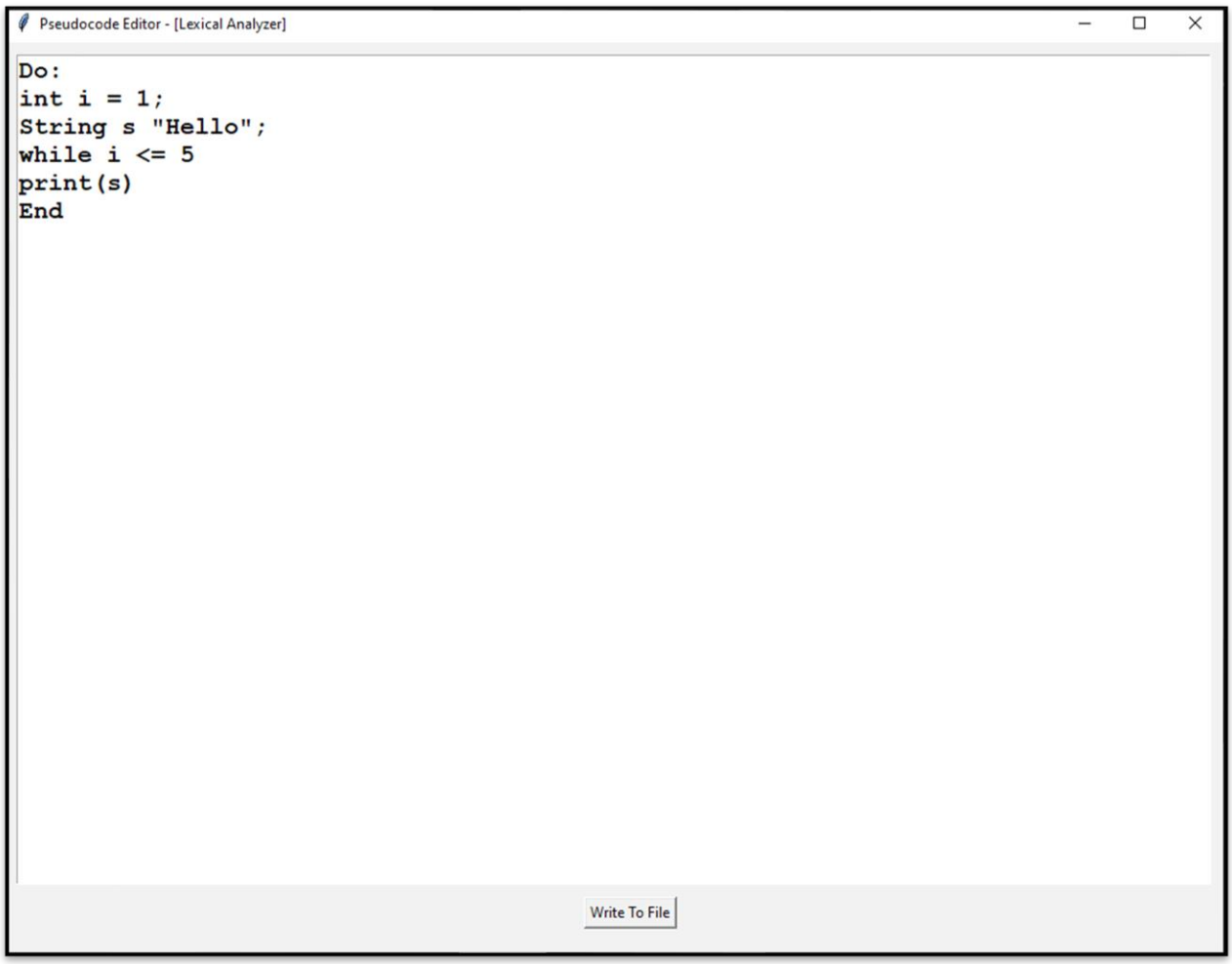
```
button.place(x=475, y=700)

gui.mainloop()

f = open('Editor.txt', 'r')
contents = f.readlines()
f.close()

global flag_identifier
flag_identifier = False
global flag_datatype
flag_datatype = False
global flag_keyword
flag_keyword = False

end_at_line = len(contents)

for i in range(len(contents)):
    if contents[i] != "":
        if "Do:" not in contents[i]:
            sys.exit("ERROR: Must start with 'Do:'")
        else:
            break

errors_list = []
tokens = []
tokens_list = []
counter = 0

for i in range(len(contents)):
    count = 0
    content_at_line = contents[i]
    temp_line = list(content_at_line)
    new_line = []
    string = ''
    for singchar in temp_line:
        if singchar in delimiters:
            if not string == '':
                new_line.append(string)
            new_line.append(singchar)
            string = ''
        else:
            string = string + singchar
    temp = " ".join(new_line)
    contents[i] = temp
```

```
flag_end = False
for i in range(len(contents)):
    if flag_end == True and contents[i] != "\n":  # كلام عقب ال END
        sys.exit("ERROR: END IS NOT THE LAST.")
    if flag_end == True and contents[i] == "\n":  # سطور فاضية عقب ال END
        continue
    if "End" in contents[i]:
        flag_end = True

if flag_end == False:
    sys.exit("ERROR: NO END KEYWORD.")

for i in range(end_at_line):
    if i == 0:
        continue
    contents_at_line = contents[i].split()
    for word in contents_at_line:
        #print("THE WORD= ", word, contents_at_line.index(word), i+1)
        if word == "End":
            continue
        dfa(word, contents_at_line.index(word), i+1)

    print('--> Line #{}:'.format(i+1), end=' ')
    print(tokens[counter:])
    counter = len(tokens)

print("PROGRAM FINISHED...")


class TableForTokens:
    def __init__(self, root):
        # code for creating table
        for i in range(1):
            for j in range(3):
                self.e = Entry(root, width=20, fg='white',
                               bg='#131E3A', font=('Arial', 16, 'bold'))
                self.e.grid(row=i, column=j)
                self.e.insert(END, token_table_headers[j])
        for i in range(token_table_total_rows):
            for j in range(token_table_total_columns):
                self.e = Entry(root, width=20, fg='white',
                               bg='#95C8D8', font=('Arial', 16, 'bold'))
                self.e.grid(row=i+1, column=j)
                self.e.insert(END, tokens_list[i][j])
```

```python
class TableForErrors:
    def __init__(self, root):
        # code for creating table
        for i in range(1):
            for j in range(1):
                self.e = Entry(root, width=70, fg='white',
                               bg='#131E3A', font=('Arial', 16, 'bold'))
                self.e.grid(row=i, column=j)
                self.e.insert(END, error_table_headers[j])
        for i in range(error_table_total_rows):
            for j in range(error_table_total_columns):
                self.e = Entry(root, width=70, fg='white',
                               bg='#95C8D8', font=('Arial', 16, 'bold'))
                self.e.grid(row=i+1, column=j)
                self.e.insert(END, errors_list[i][j])


token_table_headers = ['Token', 'Line Number', 'Type']
token_table_total_rows = len(tokens_list)
token_table_total_columns = len(tokens_list[0])

root = Tk()
root.title("Tokens Table")
root.geometry("+350+180")
table = TableForTokens(root)

if len(errors_list) != 0:
    error_table_headers = ['Error']
    error_table_total_rows = len(errors_list)
    error_table_total_columns = len(errors_list[0])

    root = Tk()
    root.title("Tokens Error Table")
    root.geometry("+350+0")
    table = TableForErrors(root)

root.mainloop()
```

4 Tests

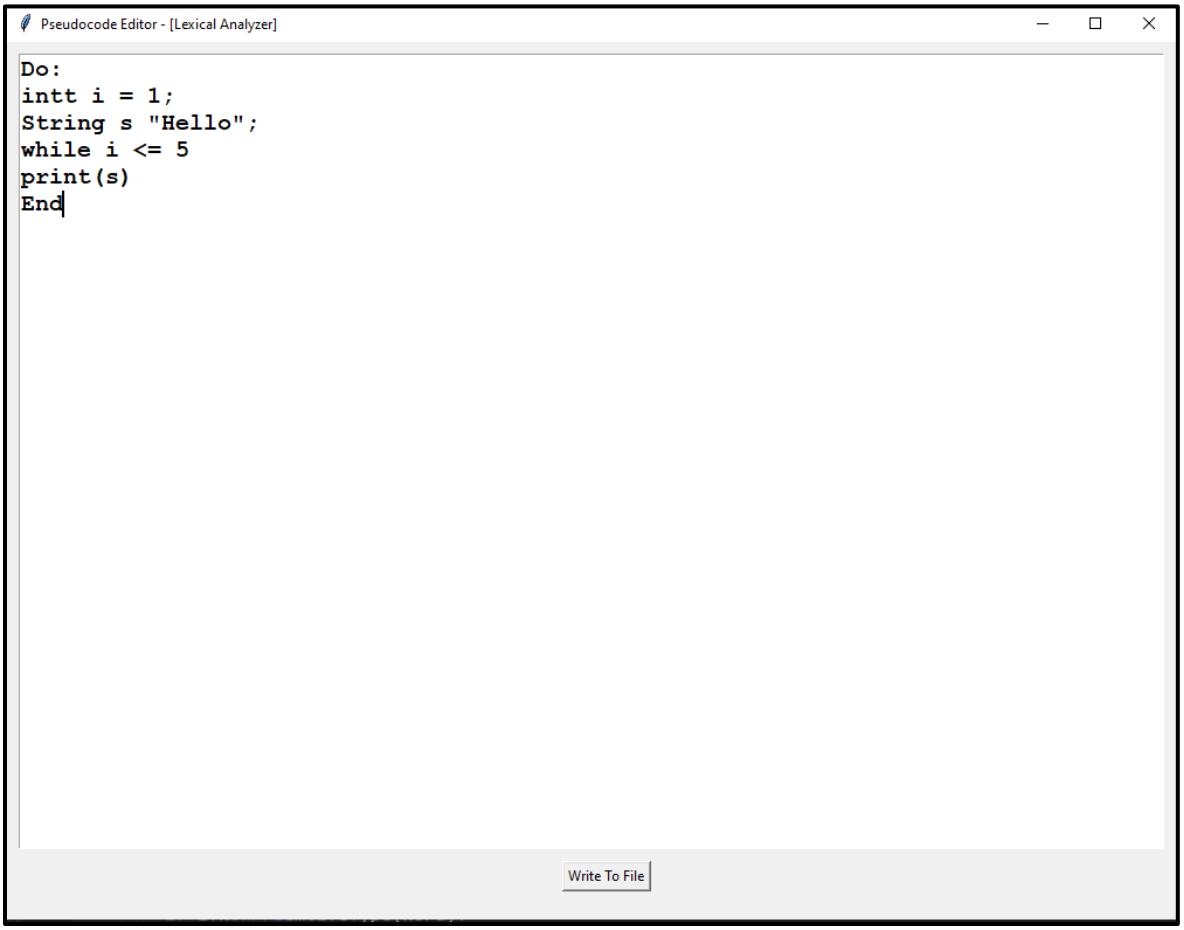
Test #1: Correct no error code.



Tokens Table

Token	Line Number	Type
int	2	DATATYPE
i	2	IDENTIFIER
=	2	OPERATOR
1	2	INTEGER
;	2	END STATEMENT
String	3	DATATYPE
s	3	IDENTIFIER
"Hello"	3	STRING
;	3	END STATEMENT
while	4	KEYWORD
i	4	IDENTIFIER
<=	4	OPERATOR
5	4	INTEGER
print	5	KEYWORD
(	5	DELIMITER
s	5	IDENTIFIER
)	5	DELIMITER

Test #2: Incorrect code with type error. ("intt")



Tokens Error Table

Error
ERORR at line #2: TYPO[DATATYPE, KEYWORD]. [intt]

Tokens Table

Token	Line Number	Type
i	2	IDENTIFIER
=	2	OPERATOR
1	2	INTEGER
;	2	END STATEMENT
String	3	DATATYPE
s	3	IDENTIFIER
"Hello"	3	STRING
;	3	END STATEMENT
while	4	KEYWORD
i	4	IDENTIFIER
<=	4	OPERATOR
5	4	INTEGER
print	5	KEYWORD
(	5	DELIMITER
s	5	IDENTIFIER
)	5	DELIMITER

References

[1] "medium," [Online]. Available: https://medium.com/factory-mind/regex-tutorial-a-simple-cheatsheet-by-examples-649dc1c3f285.

[2] "docs.python.org," [Online]. Available: https://docs.python.org/3/reference/grammar.html.

[3] "regexlab," [Online]. Available: http://www.regexlab.com/en/workshop.htm?pat=[bcd][bcd]&txt=abc123.

[4] "geeksforgeeks," [Online]. Available: https://www.geeksforgeeks.org/program-to-construct-dfa-for-regular-expression-c-a-b/.

[5] "programiz," [Online]. Available: https://www.programiz.com/python-programming/regex.

[6] "regexr," [Online]. Available: https://regexr.com/.

[7] "medium," [Online]. Available: https://medium.com/@mikhail.barash.mikbar/grammars-for-programming-languages-fae3a72a22c6.

[8] "regex101," [Online]. Available: https://regex101.com/.

[9] "maheshjangid," [Online]. Available: https://maheshjangid.files.wordpress.com/2011/07/1-4.pdf.

[10] "stanford," [Online]. Available: https://web.stanford.edu/class/archive/cs/cs103/cs103.1142/lectures/17/Small17.pdf.

[11] "stackoverflow," [Online]. Available: https://stackoverflow.com/.