REPORT

for

Compiler

Version 0.3

Prepared by

Giovanni Guzmán
A01566171

CONTENTS II

REVISIONS II

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
0.3	Giovanni Guzmán	The scanner was implemented	09/04/21

1 Introduction

1.1 Summarry

TO DO: Write 1-2 paragraphs describing the purpose of this document as explained above.>
This report documents the Analysis, Design, Implementation, Testing, and Deployment of the compiler C--. The document will provide examples and design documents that will be useful to understand the whole project.

1.2 Notation

Regular expressions are widely used in the scanner of a compiler. Regular expressions (regex) represent strings of characters. It's precise and can represent the lexical rules of a language. State machines are also part of the lexical analysis. This machines represent and describe algorithms fot systems based on transitions and behaviour. Where transitions between the states determine how a particular machine will react. Lastly, a transition table will be implemented. This tables are two-dimentional arrays that are indexed by the states of the machine and by the input characters it can receive. Then it provides all the possible transitions a particular state machine has.
The model used.
For the analysis a class diagram was decided as the main design document of the project. This diagram provides with the interactions between components and the functionality of each.
Python 3 will be used to develop this compiler. It may not be the fastest or the most robust language but its ease of use will allow for all the resources to be working on putting together the compiler and not how to do it. The focus is more on the concepts and not on how to use the programming language. [1]

2 Analysis

2.1 Requirements

Develop a scanner for the C-- programming language, which is essentially a subset of the C language

2.1.1 Keywords

The Keywords of the language are:
else
if
int
return
void
while
input
output

**All keywords are reserved words and MUST be written in lowercase*

2.1.2 Special Symbols

Special symbols are the following:

+	*arithmetic addition operation*
-	*arithmetic subtraction operation*
*	*arithmetic multiplication operation*
/	*arithmetic division operation*
<	*logic operator less than*
<=	*logicoperator less or equal than*
>	*logic operator greater than*
>=	*logic operator greater or equal than*
==	*logic operator equal*
!=	*logic operator different*
=	*assignation*
;	*semicolon*
,	*coma*
(	*open parenthesis*
)	*close parenthesis*
[	*open square brackets*
]	*close square brackets*
{	*open curly brackets*
}	*close curly brackets*
/*	*open comment*
*/	*close comment*

2.1.3 Other Tokens

Other tokens are ID and NUM, their corresponding Regular Expressions definitions are as follows:

ID= letter+
NUM= digit+
letter= [a-zA-Z]
digit= [0-9]

Identifiers are letter sensitive, i.e., lower and uppercase letters are distinct

2.1.4 Other

White space consists of blanks, newlines, and tabs. White space is ignored, but it MUST be recognized. White space together withID's, NUM's, and keywords, are considered as delimiters.

2.1.5 Comments

Comments are C language style, i.e, they are enclosed by / ... */. Comments can be placed anywhere white space can appear, i,e., comments cannot be placed within tokens. Comments may include more than one line.*

To achieve all the requirements the compiler must need to receive a text file containing the desired code to scan. After receiving it the scanner needs to read one character at the time having in mind the lookahead problem. Using a previously built transition table the scanner will determine which state to transition to. When finding a delimiter the id or keyword will be added to an id table to be stored. The scanner must read but eliminate all blank spaces and comments.
If the code that was input to the compiler doesn't comply with all the specifications the compiler (taking the current state into consideration) must print the corresponding error.

3 Design

For the design, the automata, transition table and the class diagram were used to have a general picture of the components and how to create the scanner.

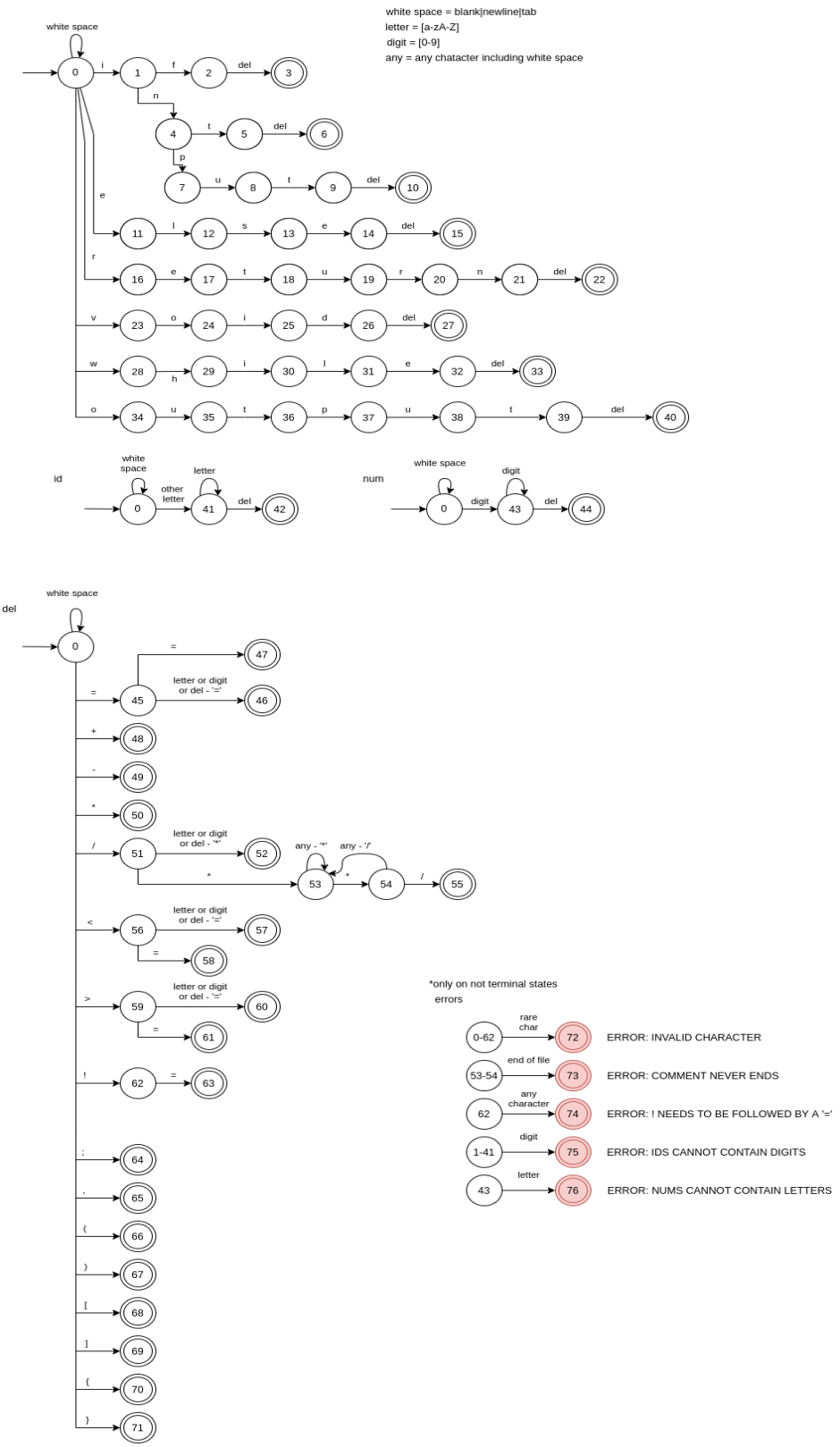

Figure 1: Automata

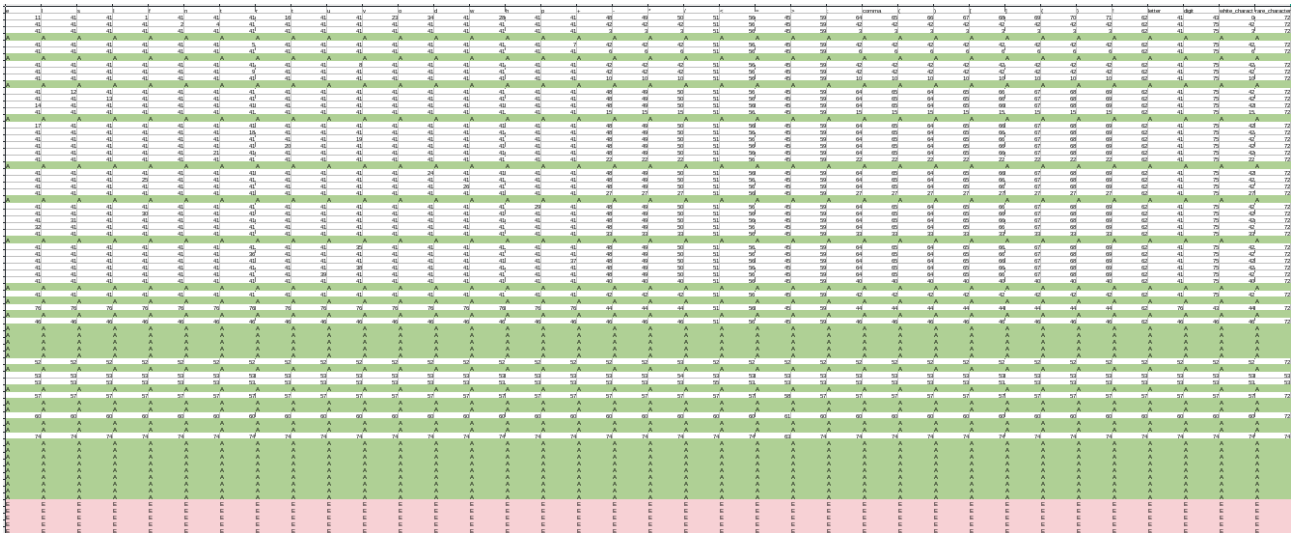

Figure 2: Transition Table

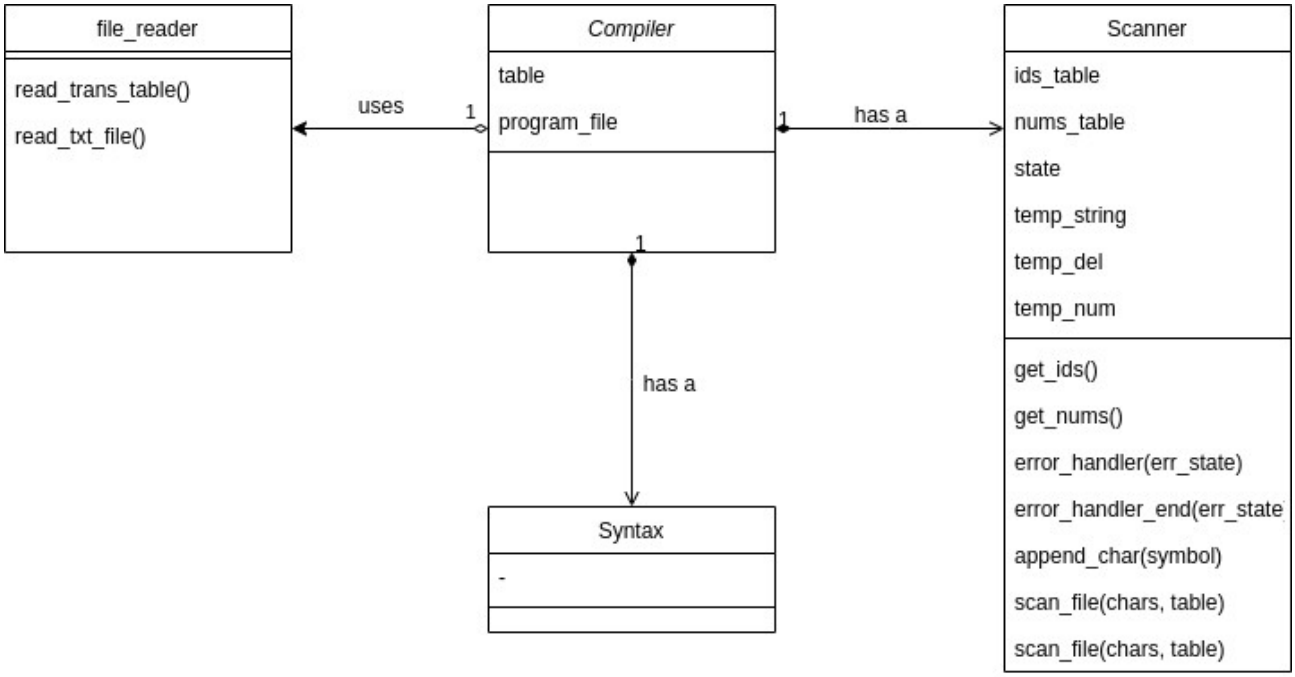

Figure 3: Class Diagram

4 Implementation

4.1 Code

main.py > main

```python
from utilities.file_reader import read_trans_table, read_txt_file
from scanner import Scanner

def main():
    table = read_trans_table()
    program_file = read_txt_file()
    scanner = Scanner()
    scanner.scan_file(program_file, table)
    i = scanner.get_ids()
    n = scanner.get_nums()

    print('Identifiers')
    for x in range(len(i)):
        print(x, '|', i[x])
    print('\n', 'Numbers')
    for x in range(len(n)):
        print(x, '|', n[x])


if __name__ == '__main__':
    main()
```

Figure 4: Code1

file_reader.py > ...

```python
import csv

def read_trans_table():
    table = []
    with open('tabla_transicion.csv', newline='') as csvfile:
        reader = csv.DictReader(csvfile)
        for row in reader:
            table.append(row)
        return table

def read_txt_file():
    symbol_list = []
    with open('program.txt') as text_file:
        lines = text_file.read()
        symbol_list = list(lines)
    return symbol_list
```

Figure 5: Code2

```
import re

class Scanner:

    def __init__(self):
        self.ids_table = []
        self.nums_table = []
        self.state = 0
        self.temp_string = ''
        self.temp_del = ''
        self.temp_num = ''

    def get_ids(self):
        return self.ids_table
    def get_nums(self):
        return self.nums_table

    def error_handler(self, err_state):
        msg = 'ERROR: '
        switcher={
            72: msg + 'INVALID CHARACTER',
            73: msg + 'COMMENT NEVER ENDS',
            74: msg + '! NEEDS TO BE FOLLOWED BY A =',
            75: msg + 'IDS CANNOT CONTAIN DIGITS',
            76: msg + 'NUMS CANNOT CONTAIN LETTERS',
            77: msg + 'REACHED END OF FILE PREMATURELY'
        }
        return switcher.get(err_state, "UNKNOWN ERROR")
    def error_handler_end(self, err_state):
        msg = 'ERROR: '
        switcher={
            53: msg + 'COMMENT NEVER ENDS',
            54: msg + 'COMMENT NEVER ENDS',
        }
```

Figure 6: Code3

```
def scan_file(self, chars, T):
    for symbol in chars:

        # Check to see if we are in a non terminal state
        if(T[self.state]['e'] != 'E' and T[self.state]['e'] != 'A'):
            self.append_char(symbol)
            ts = symbol

            # Depending on the symbol we transform it so the dictionary can recognice it
            if(re.search('e|l|s|i|f|n|t|r|t|u|v|o|d|w|h|p|\+|\-|\*|\/|<|=|>|\;|\(|\)|\[|\]|\{|\}|\!', symbol)):
                ts = symbol
            else:
                if(re.search('\,', symbol)):
                    ts = 'comma'
                elif(re.search('\d', symbol)):
                    ts = 'digit'
                elif(re.search('\s', symbol)):
                    ts = 'white_character'
                elif(re.search('[a-zA-Z]', symbol)):
                    ts = 'letter'
                else:
                    ts = 'rare_character'

            # Go to the next state
            self.state = int(T[self.state][ts])

        # Handle the accepting state
        if(T[self.state]['e'] == 'A'):
            if (self.temp_string != ''):
                self.ids_table.append(self.temp_string)
                self.temp_string = ''
            if (self.temp_num != ''):
                self.nums_table.append(self.temp_num)
                self.temp_num = ''
            self.temp_del = ''

            self.append_char(symbol)

            self.state = 0
```

Figure 7: code4

```
    # ERROR WAS DETECTED
    elif(T[self.state]['e'] == 'E'):
        msg = self.error_handler(self.state)
        print(msg)
        exit()

# Check the last state when the file has been read
# to check for errors at the end
if(self.state != 0):
    self.error_handler_end(self.state)
```

Figure 8: Code5

4.2 Tests

The tests used for the compiler are shown in Figure 4. This tests where manually checked after each new functionality added. The downside Is the great amount of time needed to do the tests but manually checking them each times gives certainty of the results

ID	Desc	Input	Output
1	The compiler reads the csv transition table correctly.	transition_table.csv	ReadDictionary
2	The transition table is divided into rows. Each row is a dictionary	ReadDictionary	Array of Dict of the same length as the transition table
3	The compiler reads the text file of the code correctly.	program.txt	ReadDictionary
4	The program file is divided into individual characters and added to a list	ReadDictionary	Array of single characters of the same length as the transition table
5	The scanner transitions into the correct state with each next character	character, state	new state
6	The scanner recognices an accepted state. It appends the new string of characters into its corresponding list	character, state	State = 0 List appended with new string of characters
7	The scanner recognices an error state. It displays the corresponding error message and ends the execution	character, state	error message Exit program

Figure 9: Tests

5 References

1. R. Castelló, Class Lecture, Topic: “Chapter 2 –Lexical Analysis.”TC3048, School of Engineering and Science, ITESM, Chihuahua, Chih, April, 2020.
2. Data Flair, *2 Simple Ways to Implement Python Switch Case Statement* [blog on-line] ; available from https://data-flair.training/blogs/python-switch-case/; Internet; accessed 8 April, 2021
3. Python Development Team, *Python 3.9.4 Documentation* [Language Documentation on-line]; available from https://docs.python.org/; Internet; accessed 8 April, 2021
4. GhostofGoes, Gitignore emplate For Python Projects [Repository on-line]; available from https://gist.github.com/GhostofGoes/94580e76cd251972b15b4821c8a06f59; Internet; accessed 9 April, 2021