LEXICAL ANALYZER DAN PARSER

Mata Kuliah Teori Bahasa Dan Automata

Disusun oleh:

1. Berlian Muhammad Galin Al Awienoor (1301204378)
2. Eric Nur Rahman (1301200010)
3. Kiki Dwi Prasetyo (1301204027)

Kelas IF-44-10
Kelompok 14

Fakultas Informatika
Universitas Telkom
Tahun 2021/2022

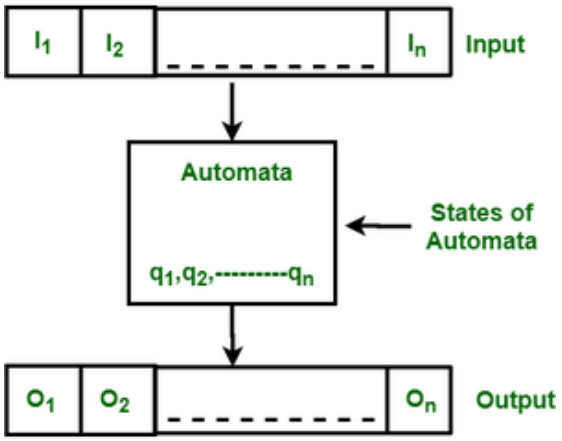
DAFTAR ISI

PENDAHULUAN

Seperti halnya pada tata bahasa regular, sebuah Tata Bahasa Bebas Konteks (Context Free Grammar atau CFG) adalah suatu cara yang menunjukkan bagaimana menghasilkan untai-untai dalam sebuah bahasa. Seperti kita ketahui, pada saat menurunkan suatu string, simbol-simbol variabel akan mewakili bagian-bagian yang belum yang belum diturunkan dari string tersebut. Pada tata bahasa regular, bagian yang belum terturunkan tersebut selalu terjadi pada suatu ujung, pada Context Free Grammar bisa terdapat lebih banyak bagian yang belum terturunkan itu dan bisa terjadi dimana saja. Ketika penurunan itu sudah lengkap, semua bagian yang belum terturunkan telah diganti oleh string - string (yang mungkin saja kosong) dari himpunan simbol terminal. Context Free Grammar menjadi dasar dalam pembentukan suatu parser/proses analisis sintaksis. Bagian sintaks dalam suatu kompilator kebanyakan didefinisikan dalam tata bahasa bebas konteks. Context free grammar sederhana pada laporan ini dibuat dengan representasi aturan atau sintaks kalimat dalam sebuah bahasa bahasa Batak**.**

A. Finite Automata

Finite Automata adalah mesin automata dari suatu bahasa regular. Mesin abstrak yang terdiri dari finite number of State. Salah satu sebagai initial state dan minimal satu accepted state. Mesin akan menerima input stream berupa symbol / alphabet yang datang secara sekuensial. Mesin akan berubah dari state satu ke state lain berdasarkan simbol input dan current state. Finite Automata memiliki jumlah state yang banyaknya berhingga dan dapat berpindah-pindah dari suate state ke state yang lainnya. Finite Automata dibagi menjadi Deterministic Finite Automata (DFA) dan Non-Deterministic Finite Automata (NFA). Finite Automata (FA) adalah mesin paling sederhana untuk mengenali pola. finite automata atau finite state machine adalah mesin abstrak yang memiliki lima elemen atau tupel. Ini memiliki satu set state bagian dan aturan untuk berpindah dari satu state ke state lain tetapi itu tergantung pada simbol input yang diterapkan. Pada dasarnya, ini adalah model abstrak dari komputer digital.



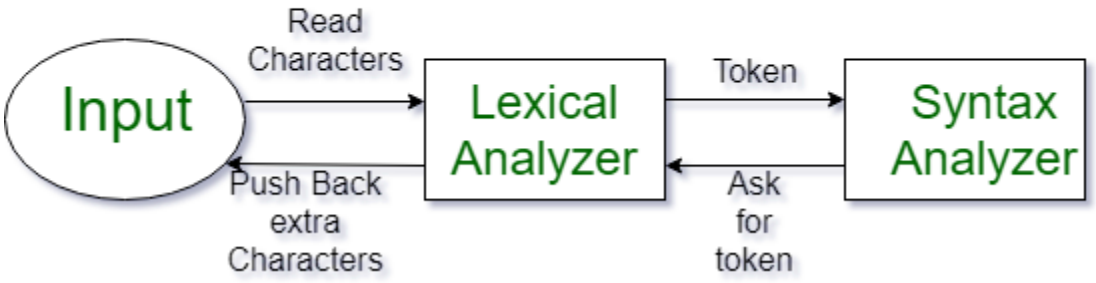
B. Context Free Grammar

Context Free Grammar (CFG) adalah tata bahasa yang mempunyai tujuan sama seperti halnya tata bahasa regular yaitu merupakan suatu cara untuk menunjukkan bagaimana menghasilkan suatu untai-untai dalam sebuah bahasa. CFG dapat disebut juga Tata Bahasa Bebas Konteks yang dapat diartikan sebagai sebuah tata bahasa dimana tidak terdapat pembatasan pada hasil produksinya. CFG adalah suatu notasi formal untuk menyatakan definisi rekursif dari suatu bahasa. Grammar terdiri atas satu atau lebih variabel yang mewakili kelas kelas untai. Terdapat aturan yang menyatakan bagaimana tiap untai dibangun. Aturan ini menghasilkan alfabet, untai lain yang telah diketahui, atau keduanya. Suatu Context Free Grammar dapat berbentuk sangat melebar, sangat menyempit, atau terjadi rekursif kiri, yang semuanya sering dinamakan bentuk tidak formal. CFG banyak digunakan untuk menentukan struktur bahasa pemrograman (contohnya parser).

C. Lexical Analysis

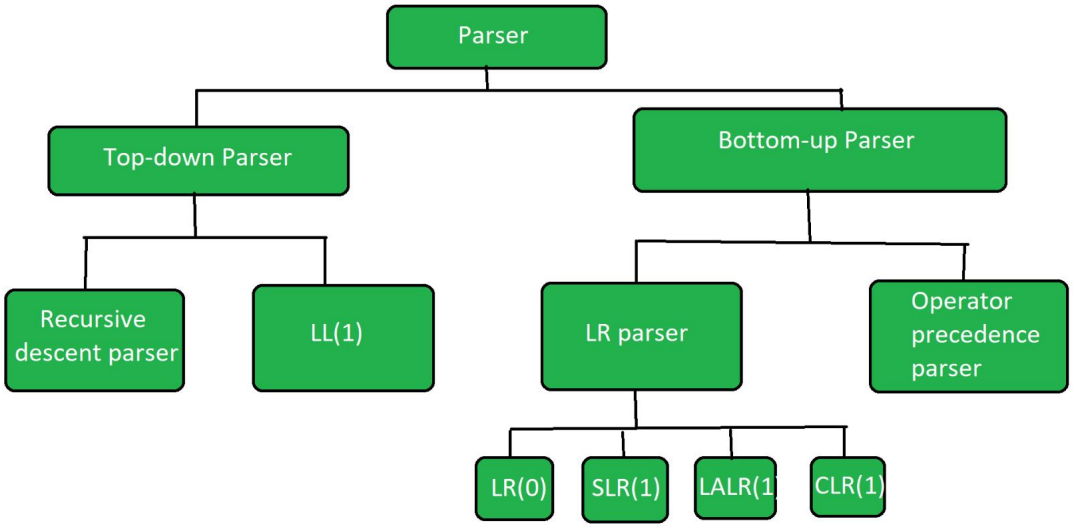
Lexical Analysis adalah sebuah rangkaian karakter yang dilakukan di tahapan pertama pada compiler. Lexical Analysis menerima masukan (input) serangkaian karakter dan menghasilkan deretan simbol yang masing - masing dinamakan token, proses parsing akan lebih mudah dilakukan bila inputnya sudah berupa token. Lexical Analysis dapat disebut sebagai scanner yang berarti dapat mengubah deretan karakter-karakter menjadi deretan token-token. Proses yang dilakukan pada tahapan ini adalah membaca program sumber karakter per karakter. Satu atau lebih (deretan) karakter - karakter ini dikelompokkan menjadi suatu kesatuan mengikuti pola kesatuan kelompok karakter (token) yang ditentukan dalam bahasa sumber dan disimpan dalam tabel simbol,

sedangkan karakter yang tidak mengikuti pola akan dilaporkan sebagai token tak dikenal atau tidak valid.



D. Parser

Parser adalah komponen compiler atau juru bahasa memecah data menjadi elemen yang lebih kecil untuk memudahkan terjemahan ke bahasa lain. Penguraian atau parsing adalah suatu cara memecah-mecah suatu rangkaian masukan dengan mengambil input dalam bentuk urutan token atau intruksi program dan menghasilkan struktur data dalam bentuk pohon uraian (parse) atau pohon sintaksis abstrak yang akan digunakan pada tahap kompilasi berikutnya yaitu analisis semantik. Parser bisa disebut fase kompiler yang mengambil string token sebagai input dan dengan bantuan tata bahasa yang ada, mengubahnya menjadi Representasi Menengah (IR) yang sesuai. Parser juga dikenal sebagai Syntax Analyzer.



CONTEXT FREE GRAMMAR

Untuk Context Free Grammar pada tugas besar Teori Bahasa dan Automata kali ini kami menggunakan bahasa Batak. Adapun bentuk validasi yang mempresentasikan bahasa Batak adalah sebagai berikut.

<S> ::= <SB> <VB> <OB>

<SB> ::= (ayah) amang | (ibu) inang | (saya) au | (kami) hami

<VB> ::= (makan) mangan | (memegang) maniop | (memotong) mamonggol

<OB> ::= (buah) boras | (ikan) dekke | (daging) sibuk

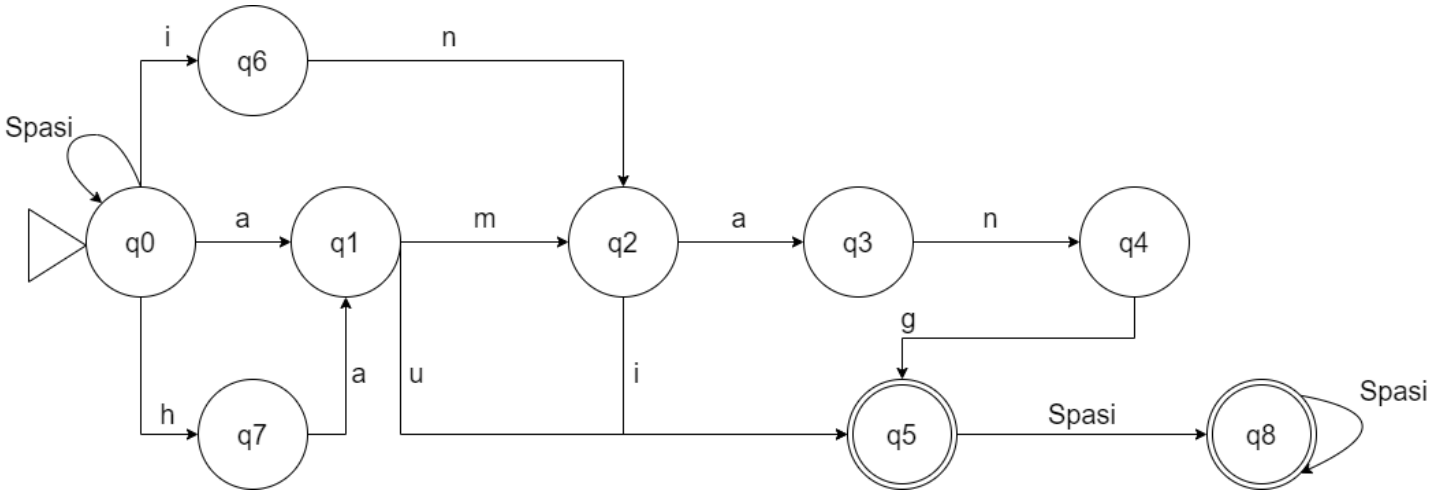
- **Simbol non-terminal:**
  S (Starting symbol), SB (Subject), VB (Verb), OB (Object)

- **Simbol terminal:**
  Amang, inang, au, hami, mangan, maniop, mamonggol, boras, dekke, sibuk

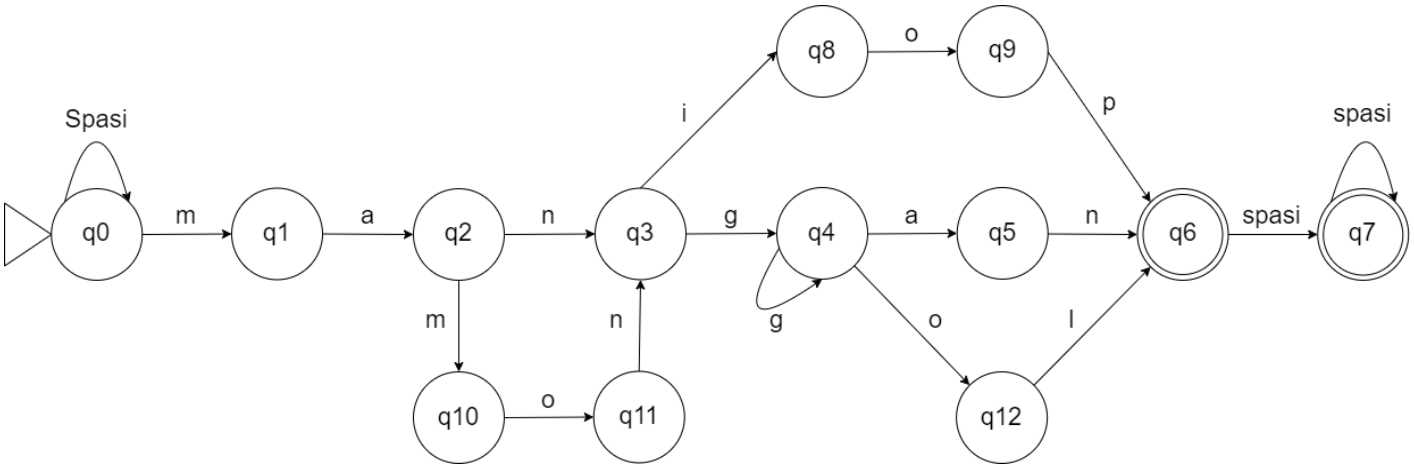
FINITE AUTOMATA

Setelah membuat aturan untuk grammar, langkah selanjutnya yang akan dilakukan adalah membangun Finite Automata (FA) sebagai rules dari state - state serta value yang akan digunakan pada program lexical analyzer yang akan dibuat. Adapun bentuk model Finite Automata (FA) yang dibuat yaitu:

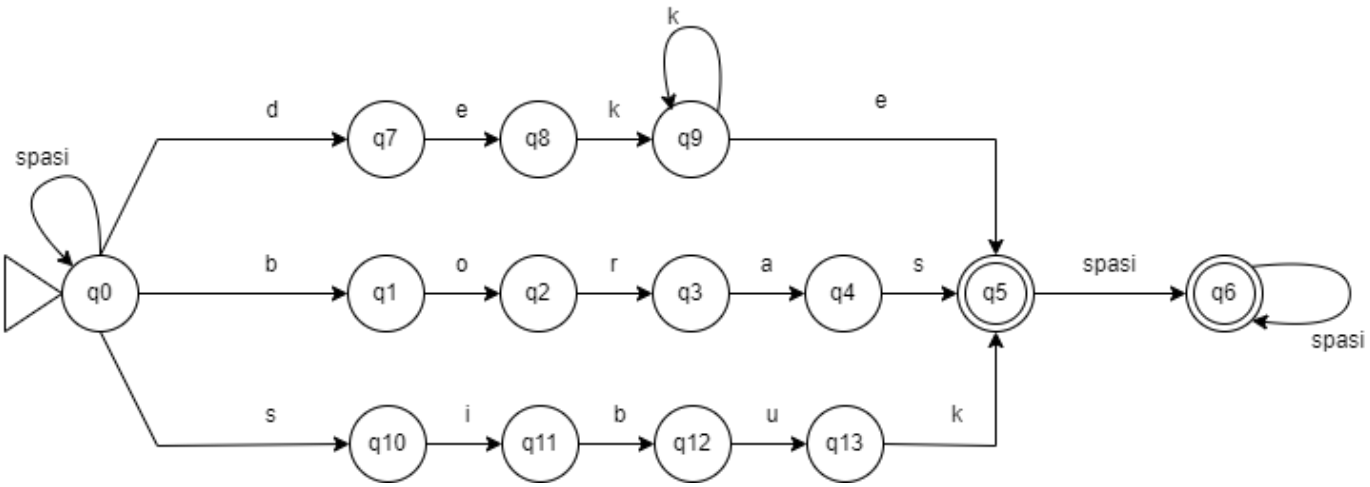
1. Subject



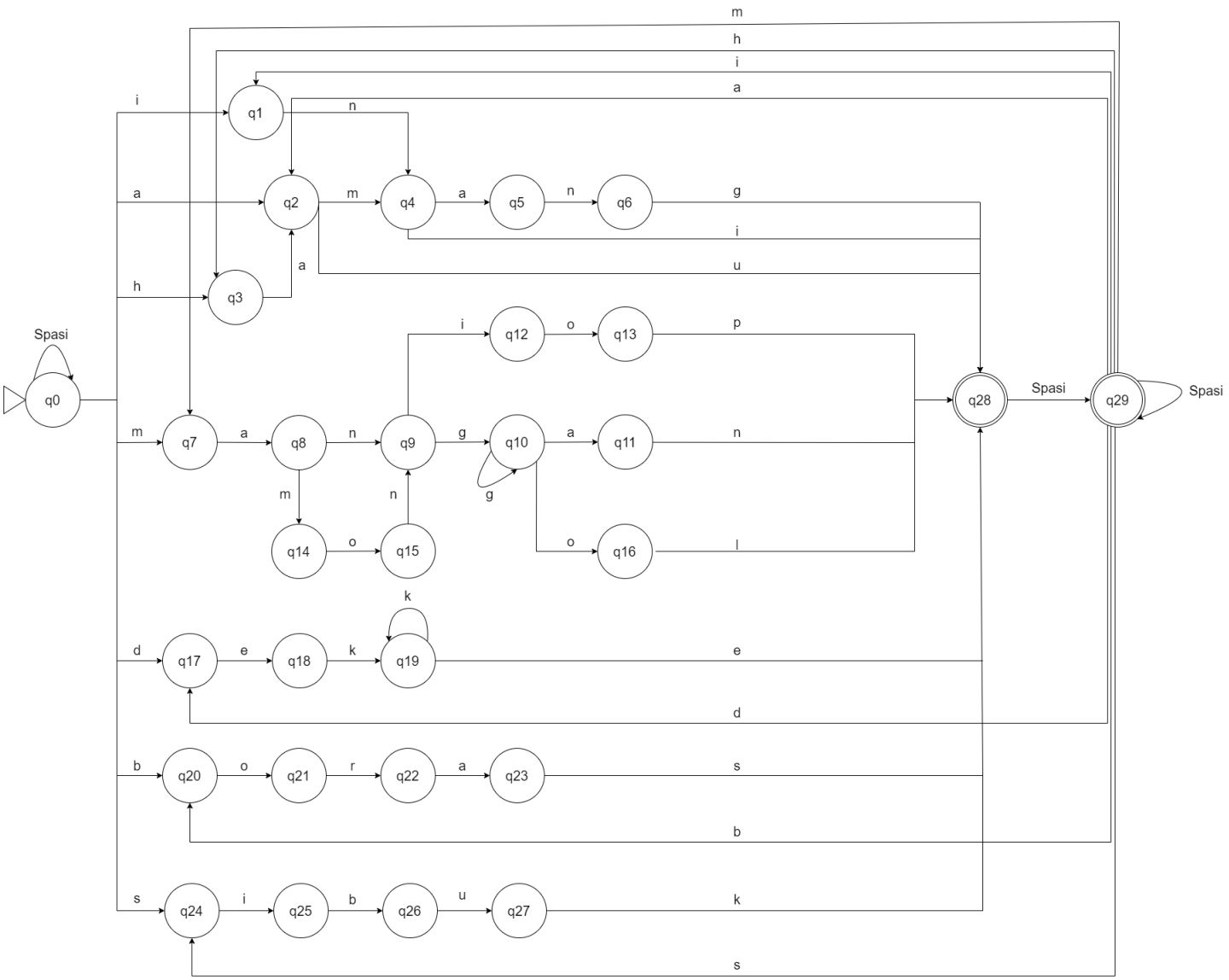
2. Verb



3. Object



4. Hasil Akhir Desain Finite Automata



PARSE TABLE (LL)

Setelah melakukan penyusunan Finite Automata (FA), yang akan dibangun selanjutnya adalah membangun atau membuat Parse Table (LL) yang akan digunakan sebagai pembangun program parse kalimat sebagai menentukan susunan aturan grammar yang telah ditentukan yaitu <subject> <verb> <object>.

Adapun susunan Parse Table (LL) dijabarkan sebagai berikut.

	amang	**inang**	**au**	**hami**	**mangan**	**maniop**	**mamonggol**	**boras**	**dekke**	**sibuk**	**EOS**
S	SB VB OB	SB VB OB	SB VB OB	SB VB OB	error	error	error	SB VB OB	SB VB OB	SB VB OB	error
SB	amang	inang	au	hami	error	error	error	error	error	error	error
VB	error	error	error	error	mangan	maniop	mamonggol	error	error	error	error
OB	error	error	error	error	error	error	error	boras	dekke	sibuk	error

IMPLEMENTASI PROGRAM

A. Lexical Analysis

Berikut adalah kode program Lexical Analysis dengan bahasa pemrograman Python, dimana kode program ini bertujuan untuk menguji kata masukan (inputan) yang telah ditentukan. Adapun kata yang akan diuji validitasnya dalam Lexical Analysis yaitu amang, inang, au, hami, mangan, maniop, mamonggol, boras, dekke dan sibuk. Untuk kata amang, inang, au dan hami merupakan kata untuk subjek (subject). Untuk kata mangan, maniop dan mamonggol merupakan kata untuk kata kerja (verb). Untuk kata boras, dekke dan sibuk merupakan kata untuk objek (object).

1. Inisialisasi semua state

```
def lexical(sentence):

    #Initialization || menginialisasi semua state
    alphabet_list = list(string.ascii_lowercase) #mengubah semua huruf ke lowercase
    state_list = ['q0', 'q1', 'q2', 'q3', 'q4', 'q5', 'q6', 'q7', 'q8', 'q9', 'q10', 'q11',
                  'q12', 'q13', 'q14', 'q15', 'q16', 'q17', 'q18', 'q19', 'q20', 'q21', 'q22',
                  'q23', 'q24', 'q25', 'q26', 'q27', 'q28', 'q29']

    transition_table = {}

    for state in state_list:
        for alphabet in alphabet_list:
            transition_table[(state, alphabet)] = "error"
            transition_table[(state, "#")]      = "error"
            transition_table[(state, " ")]      = "error"
```

2. Membuat Initial State dan Finish State

```
#Initial State (start)
transition_table["q0", " "]    = "q0"

#Finish State (FS)
transition_table[("q28", "#")] = "accept"
transition_table[("q28", " ")] = "q29"

transition_table[("q29", "#")] = "accept"
transition_table[("q29", " ")] = "q29"
```

3. Membuat State untuk Subject

```
#Subject
#String "amang"
transition_table[("q0", "a")]  = "q1"
transition_table[("q2", "m")]  = "q4"
transition_table[("q4", "a")]  = "q5"
transition_table[("q5", "n")]  = "q6"
transition_table[("q6", "g")]  = "q28"
transition_table[("q28", " ")] = "q29"
transition_table[("q29", "a")] = "q2"

#String "inang"
transition_table[("q0", "i")]  = "q1"
transition_table[("q1", "n")]  = "q4"
transition_table[("q4", "a")]  = "q5"
transition_table[("q5", "n")]  = "q6"
transition_table[("q6", "g")]  = "q28"
transition_table[("q28", " ")] = "q29"
transition_table[("q29", "i")] = "q1"

#String "hami"
transition_table[("q0", "h")]  = "q3"
transition_table[("q3", "a")]  = "q2"
transition_table[("q2", "m")]  = "q4"
transition_table[("q4", "i")]  = "q28"
transition_table[("q28", " ")] = "q29"
transition_table[("q29", "h")] = "q3"

#String "au"
transition_table[("q0", "a")]  = "q2"
transition_table[("q2", "u")]  = "q28"
transition_table[("q28", " ")] = "q29"
transition_table[("q29", "a")] = "q2"
```

4. Membuat State untuk Verb

```
#Verb
#String "mangan"
transition_table[("q0", "m")]  = "q7"
transition_table[("q7", "a")]  = "q8"
transition_table[("q8", "n")]  = "q9"
transition_table[("q9", "g")]  = "q10"
transition_table[("q10", "a")] = "q11"
transition_table[("q11", "n")] = "q28"
transition_table[("q28", " ")] = "q29"
transition_table[("q29", "m")] = "q7"

#String "maniop"
transition_table[("q0", "m")]  = "q7"
transition_table[("q7", "a")]  = "q8"
transition_table[("q8", "n")]  = "q9"
transition_table[("q9", "i")]  = "q12"
transition_table[("q12", "o")] = "q13"
transition_table[("q13", "p")] = "q28"
transition_table[("q28", " ")] = "q29"
transition_table[("q29", "m")] = "q7"

#String "mamonggol"
transition_table[("q0", "m")]  = "q7"
transition_table[("q7", "a")]  = "q8"
transition_table[("q8", "m")]  = "q14"
transition_table[("q14", "o")] = "q15"
transition_table[("q15", "n")] = "q9"
transition_table[("q9", "g")]  = "q10"
transition_table[("q10", "g")] = "q10"
transition_table[("q10", "o")] = "q16"
transition_table[("q16", "l")] = "q28"
transition_table[("q28", " ")] = "q29"
transition_table[("q29", "m")] = "q7"
```

5. Membuat State untuk Object

```
#Object
#String "dekke"
transition_table[("q0", "d")]  = "q17"
transition_table[("q17", "e")] = "q18"
transition_table[("q18", "k")] = "q19"
transition_table[("q19", "k")] = "q19"
transition_table[("q19", "e")] = "q28"
transition_table[("q28", " ")] = "q29"
transition_table[("q29", "d")] = "q17"

#String "boras"
transition_table[("q0", "b")]  = "q20"
transition_table[("q20", "o")] = "q21"
transition_table[("q21", "r")] = "q22"
transition_table[("q22", "a")] = "q23"
transition_table[("q23", "s")] = "q28"
transition_table[("q28", " ")] = "q29"
transition_table[("q29", "b")] = "q20"

#String "sibuk"
transition_table[("q0", "s")]  = "q24"
transition_table[("q24", "i")] = "q25"
transition_table[("q25", "b")] = "q26"
transition_table[("q26", "u")] = "q27"
transition_table[("q27", "k")] = "q28"
transition_table[("q28", " ")] = "q29"
transition_table[("q29", "s")] = "q24"
```

6. Lexical Analysis

```
#Lexical Analysis
idx_char = 0
state = 'q0'
current_token = ''
while state != 'accept':
    current_char = input_string[idx_char]
    current_token += current_char
    state = transition_table[(state, current_char)]
    if state=='q28':
        print('CURRENT TOKEN  : ', current_token, ', valid')
        current_token = ''
    if state =="error":
        print('ERROR')
        break
    idx_char = idx_char + 1

#Conclusion || state yang di accept
if state == "accept":
    print('SEMUA TOKEN YANG DIINPUT : ', sentence, ', valid')

return lexical
```

7. Main Program untuk Lexical Analysis

```
#Main Program Lexical
print("======== TERMINAL ========")
print("amang, inang, hami, au")
print("mangan, maniop, mamonggol")
print("boras, dekke, sibuk")
print("========================== \n ")
sentence = input("MASUKKAN INPUT : ",)
input_string = sentence.lower()+'#'
lexical(sentence)
```

8. Hasil Output Program
    a. Contoh Inputan yang Valid
    Contoh inputan yang dimasukan adalah semua kata dari terminal. Setelah program dijalankan, hasilnya adalah VALID. Karena kata-kata yang dimasukkan sesuai dengan yang ada pada daftar simbol terminal.

```
======== TERMINAL ========
amang, inang, hami, au
mangan, maniop, mamonggol
boras, dekke, sibuk
=========================

MASUKKAN INPUT : amang inang hami au mangan maniop mamonggol boras dekke sibuk
CURRENT TOKEN  :  amang , valid
CURRENT TOKEN  :   inang , valid
CURRENT TOKEN  :   hami , valid
CURRENT TOKEN  :   au , valid
CURRENT TOKEN  :   mangan , valid
CURRENT TOKEN  :   maniop , valid
CURRENT TOKEN  :   mamonggol , valid
CURRENT TOKEN  :   boras , valid
CURRENT TOKEN  :   dekke , valid
CURRENT TOKEN  :   sibuk , valid
SEMUA TOKEN YANG DIINPUT :  amang inang hami au mangan maniop mamonggol boras dekke sibuk , valid
```

    b. Contoh Inputan yang Tidak Valid
    Contoh kalimat inputan yang digunakan adalah ‘hami maniop sibu’. Setelah program dijalankan, hasilnya adalah ERROR. Karena ada kata ‘sibu’ yang dimasukkan tidak sesuai dengan yang ada pada daftar simbol terminal.

```
======== TERMINAL ========
amang, inang, hami, au
mangan, maniop, mamonggol
boras, dekke, sibuk
=========================

MASUKKAN INPUT : hami maniop sibu
CURRENT TOKEN  :  hami , valid
CURRENT TOKEN  :   maniop , valid
ERROR
```

B. Parser

Berikut adalah kode program Parser dengan bahasa pemrograman Python, dimana kode program ini bertujuan mengecek susunan inputan sudah memenuhi aturan grammar. Kami menggabungkan program lexical analyzer diatas dengan program parser dibawah ini menjadi satu program. Program ini hanya meminta satu kali inputan berupa kalimat sepanjang tiga kata sesuai grammar yang telah kami tetapkan dengan struktur SB - VB - OB dalam bahasa Batak untuk satu kali menjalankan program. Dibawah ini adalah hasil program gabungan antara Lexical Analysis dan Parser yang telah kami rancang.

1. Deklarasi Terminal dan Non-Terminal dan Inisialisasi Parse Table

```
#Fungsi Parser
def parser(sentence):
    print("                                      ")
    print("======================================")
    print("===           PROSES PARSER          ===")
    print("====================================== \n")

    tokens = sentence.lower().split()
    tokens.append('EOS')

    non_terminals = ['S', 'SB', 'VB', 'OB']
    terminals = ['amang', 'inang', 'au', 'hami', 'mangan', 'maniop', 'mamonggol', 'boras', 'dekke', 'sibuk']

    parse_table = {}
```

2. Membuat Parse Table untuk Starting Symbol

```
#Parse Table S
parse_table[('S', 'amang')]      = ['SB', 'VB', 'OB']
parse_table[('S', 'inang')]      = ['SB', 'VB', 'OB']
parse_table[('S', 'au')]         = ['SB', 'VB', 'OB']
parse_table[('S', 'hami')]       = ['SB', 'VB', 'OB']
parse_table[('S', 'mangan')]     = ['error']
parse_table[('S', 'maniop')]     = ['error']
parse_table[('S', 'mamonggol')]  = ['error']
parse_table[('S', 'boras')]      = ['SB', 'VB', 'OB']
parse_table[('S', 'dekke')]      = ['SB', 'VB', 'OB']
parse_table[('S', 'sibuk')]      = ['SB', 'VB', 'OB']
parse_table[('S', 'EOS')]        = ['error']
```

3. Membuat Parse Table untuk Subject

```
#Parse Table SB
parse_table[('SB', 'amang')]     = ['amang']
parse_table[('SB', 'inang')]     = ['inang']
parse_table[('SB', 'au')]        = ['au']
parse_table[('SB', 'hami')]      = ['hami']
parse_table[('SB', 'mangan')]    = ['error']
parse_table[('SB', 'maniop')]    = ['error']
parse_table[('SB', 'mamonggol')] = ['error']
parse_table[('SB', 'boras')]     = ['error']
parse_table[('SB', 'dekke')]     = ['error']
parse_table[('SB', 'sibuk')]     = ['error']
parse_table[('SB', 'EOS')]       = ['error']
```

4. Membuat Parse Table untuk Verb

```
#Parse Table VB
parse_table[('VB', 'amang')]     = ['error']
parse_table[('VB', 'inang')]     = ['error']
parse_table[('VB', 'au')]        = ['error']
parse_table[('VB', 'hami')]      = ['error']
parse_table[('VB', 'mangan')]    = ['mangan']
parse_table[('VB', 'maniop')]    = ['maniop']
parse_table[('VB', 'mamonggol')] = ['mamonggol']
parse_table[('VB', 'boras')]     = ['error']
parse_table[('VB', 'dekke')]     = ['error']
parse_table[('VB', 'sibuk')]     = ['error']
parse_table[('VB', 'EOS')]       = ['error']
```

5. Membuat Parse Table untuk Object

```
#Parse Table OB
parse_table[('OB', 'amang')]     = ['error']
parse_table[('OB', 'inang')]     = ['error']
parse_table[('OB', 'au')]        = ['error']
parse_table[('OB', 'hami')]      = ['error']
parse_table[('OB', 'mangan')]    = ['error']
parse_table[('OB', 'maniop')]    = ['error']
parse_table[('OB', 'mamonggol')] = ['error']
parse_table[('OB', 'boras')]     = ['boras']
parse_table[('OB', 'dekke')]     = ['dekke']
parse_table[('OB', 'sibuk')]     = ['sibuk']
parse_table[('OB', 'EOS')]       = ['error']
```

6. Inisialisasi Stack

```
stack = []
stack.append('#')
stack.append('S')

index_token = 0
symbol = tokens[index_token]
```

7. Proses Parser

```
#Parser
while(len(stack) > 0):
    top = stack[ len(stack) - 1 ]
    print('TOP    : ', top)
    print('SYMBOL : ', symbol)
    if top in terminals:
        print('TOP ADALAH SYMBOL TERMINAL')
        if top == symbol:
            stack.pop()
            index_token = index_token + 1
            symbol = tokens[index_token]
            if symbol == "EOS":
                stack.pop()
                print('ISI STACK : ', stack)
        else:
            print('ERROR')
            break;
    elif top in non_terminals:
        print('TOP ADALAH SYMBOL NON-TERMINAL')
        if parse_table[(top, symbol)][0] != 'error':
            stack.pop()
            symbol_to_be_pushed = parse_table[(top, symbol)]
            for i in range(len(symbol_to_be_pushed)-1, -1, -1):
                stack.append(symbol_to_be_pushed[i])
        else:
            print('ERROR')
            break;
    else:
        print('ERROR')
        break;
    print('ISI STACK : ', stack)
    print()
print()
```

8. Hasil Akhir dari Proses Parser

```
#Conclusion || Hasil Akhir dari Proses Parser
if symbol == 'EOS' and len(stack) == 0:
    print('Inputan String ', '"', sentence, '"', 'Diterima, Sesuai Grammar')
else:
    print('ERROR, Inputan String:', '"', sentence, '"', ', Tidak Diterima, Tidak Sesuai Grammar')

return parser
```

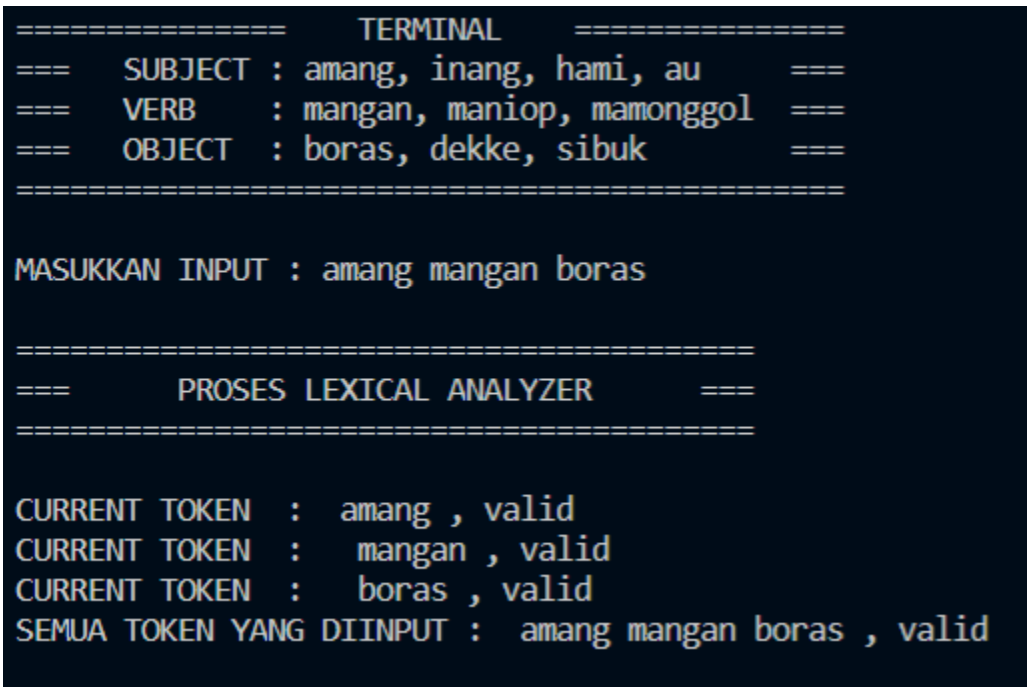
9. Main Program untuk Parser

```
#Main Program
print("===============    TERMINAL    ===============")
print("===   SUBJECT : amang, inang, hami, au     ===")
print("===   VERB    : mangan, maniop, mamonggol  ===")
print("===   OBJECT  : boras, dekke, sibuk        ===")
print("============================================== \n ")
sentence = input("MASUKKAN INPUT : ",)
input_string = sentence.lower()+'#'

if lexical(sentence):
    parser(sentence)
else:
    print("\n=========================================")
    print("===        LEXICAL TIDAK VALID        ===")
    print("===    TIDAK AKAN DILAKUKAN PARSER    ===")
    print("=========================================")
```

10. Hasil Output Program

a. Contoh Inputan yang Valid

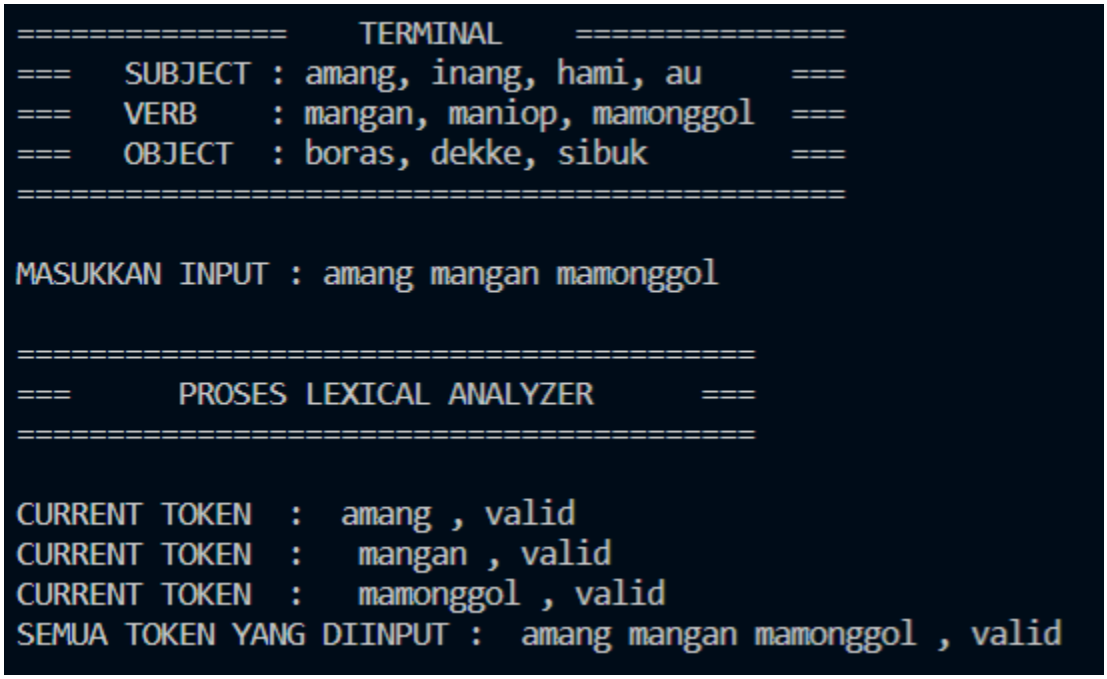
Contoh inputan yang dimasukkan sesuai dengan grammar yang telah ditentukan, yaitu membentuk pola ‘S - V - O’. Maka setelah program dijalankan hasilnya adalah Diterima.



```
=======================================
===           PROSES PARSER           ===
=======================================

TOP    :  S
SYMBOL :  amang
TOP ADALAH SYMBOL NON-TERMINAL
ISI STACK :  ['#', 'OB', 'VB', 'SB']

TOP    :  SB
SYMBOL :  amang
TOP ADALAH SYMBOL NON-TERMINAL
ISI STACK :  ['#', 'OB', 'VB', 'amang']

TOP    :  amang
SYMBOL :  amang
TOP ADALAH SYMBOL TERMINAL
ISI STACK :  ['#', 'OB', 'VB']

TOP    :  VB
SYMBOL :  mangan
TOP ADALAH SYMBOL NON-TERMINAL
ISI STACK :  ['#', 'OB', 'mangan']

TOP    :  mangan
SYMBOL :  mangan
TOP ADALAH SYMBOL TERMINAL
ISI STACK :  ['#', 'OB']

TOP    :  OB
SYMBOL :  boras
TOP ADALAH SYMBOL NON-TERMINAL
ISI STACK :  ['#', 'boras']

TOP    :  boras
SYMBOL :  boras
TOP ADALAH SYMBOL TERMINAL
ISI STACK :  []
ISI STACK :  []


Inputan String  " amang mangan boras " Diterima, Sesuai Grammar
```

b. Contoh Inputan yang Tidak Valid

Contoh inputan yang dimasukkan tidak sesuai dengan grammar yang telah ditentukan, yaitu membentuk pola ‘S - V - V’. Maka setelah program dijalankan hasilnya adalah Tidak Diterima.

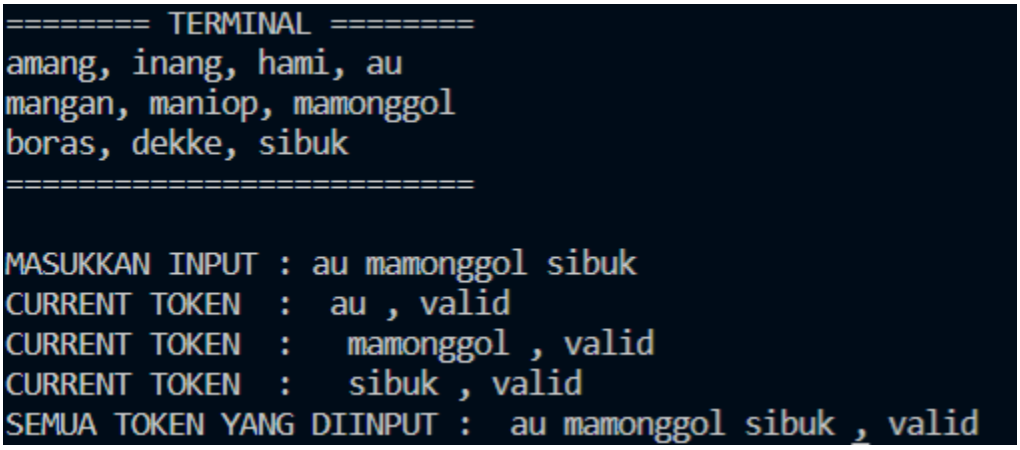


```
========================================
===           PROSES PARSER            ===
========================================

TOP    :  S
SYMBOL :  amang
TOP ADALAH SYMBOL NON-TERMINAL
ISI STACK :  ['#', 'OB', 'VB', 'SB']

TOP    :  SB
SYMBOL :  amang
TOP ADALAH SYMBOL NON-TERMINAL
ISI STACK :  ['#', 'OB', 'VB', 'amang']

TOP    :  amang
SYMBOL :  amang
TOP ADALAH SYMBOL TERMINAL
ISI STACK :  ['#', 'OB', 'VB']

TOP    :  VB
SYMBOL :  mangan
TOP ADALAH SYMBOL NON-TERMINAL
ISI STACK :  ['#', 'OB', 'mangan']

TOP    :  mangan
SYMBOL :  mangan
TOP ADALAH SYMBOL TERMINAL
ISI STACK :  ['#', 'OB']

TOP    :  OB
SYMBOL :  mamonggol
TOP ADALAH SYMBOL NON-TERMINAL
ERROR

ERROR, Inputan String: " amang mangan mamonggol " , Tidak Diterima, Tidak Sesuai Grammar
```

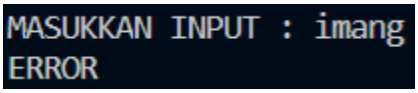
PENGUJIAN PROGRAM

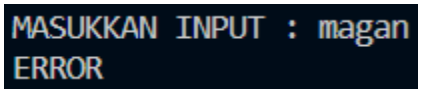
Setelah membuat kode program untuk Lexical Analysis dan Parser, selanjutnya kami akan melakukan pengujian program tersebut sesuai dengan perintah yang diberikan pada tugas besar Teori Bahasa dan Automata. Berikut hasil pengujian kode program:

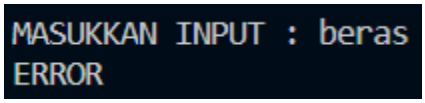
1. Kata uji untuk lexical analyzer berupa 3 kata yang valid dan 3 kata yang tidak valid berdasarkan daftar simbol terminal.
    - 3 Kata (inputan) yang sesuai Grammar



    - 3 Kata (inputan) yang tidak sesuai Grammar

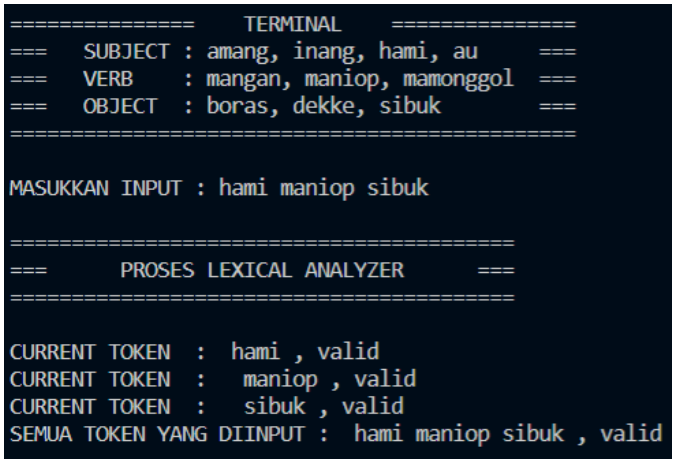




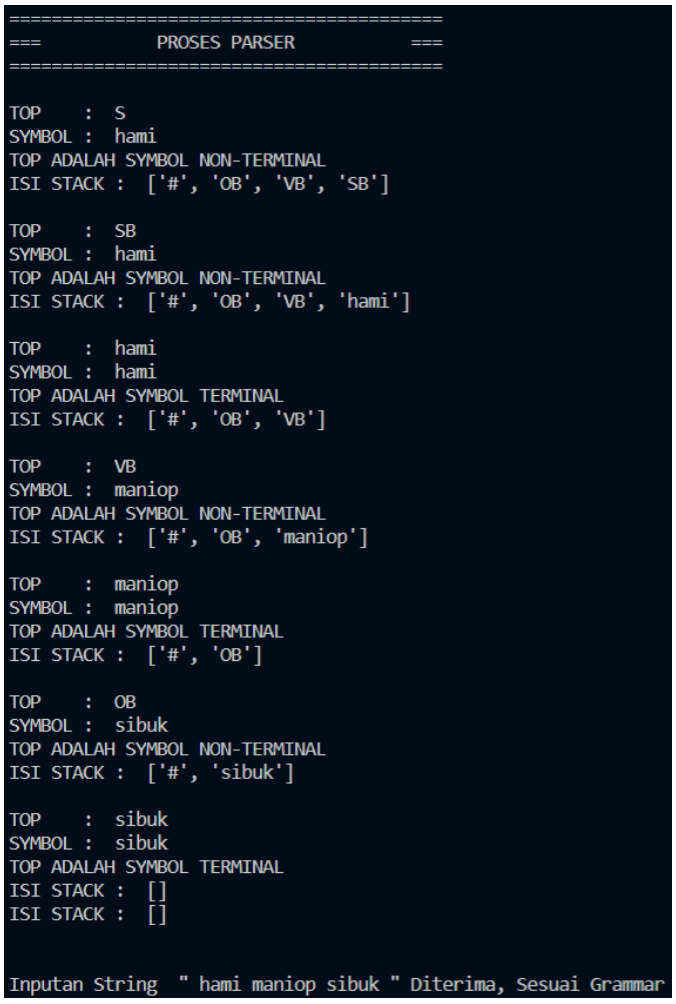


2. Kalimat uji untuk parser dengan panjang antara 2-5 kata, terdiri dari 3 kalimat yang sesuai dengan Grammar (diterima) dan 3 kalimat yang tidak sesuai dengan Grammar (tidak diterima).

- 3 Kalimat (inputan) yang sesuai Grammar



```
===============    TERMINAL    ===============
===   SUBJECT : amang, inang, hami, au     ===
===   VERB    : mangan, maniop, mamonggol  ===
===   OBJECT  : boras, dekke, sibuk        ===
=============================================

MASUKKAN INPUT : hami maniop sibuk

=======================================
===      PROSES LEXICAL ANALYZER      ===
=======================================

CURRENT TOKEN  :  hami , valid
CURRENT TOKEN  :   maniop , valid
CURRENT TOKEN  :   sibuk , valid
SEMUA TOKEN YANG DIINPUT :  hami maniop sibuk , valid
```



```
=====================================
===          PROSES PARSER          ===
=====================================

TOP    :  S
SYMBOL :  hami
TOP ADALAH SYMBOL NON-TERMINAL
ISI STACK :  ['#', 'OB', 'VB', 'SB']

TOP    :  SB
SYMBOL :  hami
TOP ADALAH SYMBOL NON-TERMINAL
ISI STACK :  ['#', 'OB', 'VB', 'hami']

TOP    :  hami
SYMBOL :  hami
TOP ADALAH SYMBOL TERMINAL
ISI STACK :  ['#', 'OB', 'VB']

TOP    :  VB
SYMBOL :  maniop
TOP ADALAH SYMBOL NON-TERMINAL
ISI STACK :  ['#', 'OB', 'maniop']

TOP    :  maniop
SYMBOL :  maniop
TOP ADALAH SYMBOL TERMINAL
ISI STACK :  ['#', 'OB']

TOP    :  OB
SYMBOL :  sibuk
TOP ADALAH SYMBOL NON-TERMINAL
ISI STACK :  ['#', 'sibuk']

TOP    :  sibuk
SYMBOL :  sibuk
TOP ADALAH SYMBOL TERMINAL
ISI STACK :  []
ISI STACK :  []


Inputan String  " hami maniop sibuk " Diterima, Sesuai Grammar
```

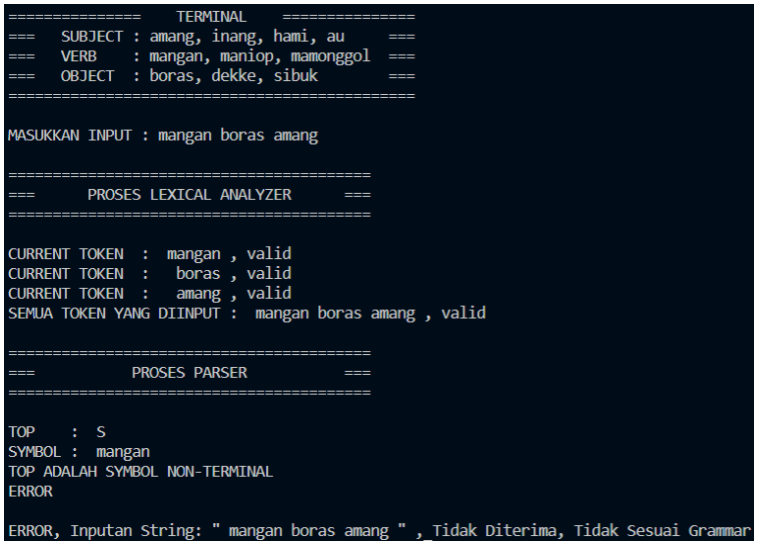
```
===============    TERMINAL    ===============
===   SUBJECT : amang, inang, hami, au     ===
===   VERB    : mangan, maniop, mamonggol  ===
===   OBJECT  : boras, dekke, sibuk        ===
=============================================

MASUKKAN INPUT : amang mangan boras

=======================================
===      PROSES LEXICAL ANALYZER      ===
=======================================

CURRENT TOKEN  :  amang , valid
CURRENT TOKEN  :   mangan , valid
CURRENT TOKEN  :   boras , valid
SEMUA TOKEN YANG DIINPUT :  amang mangan boras , valid
```

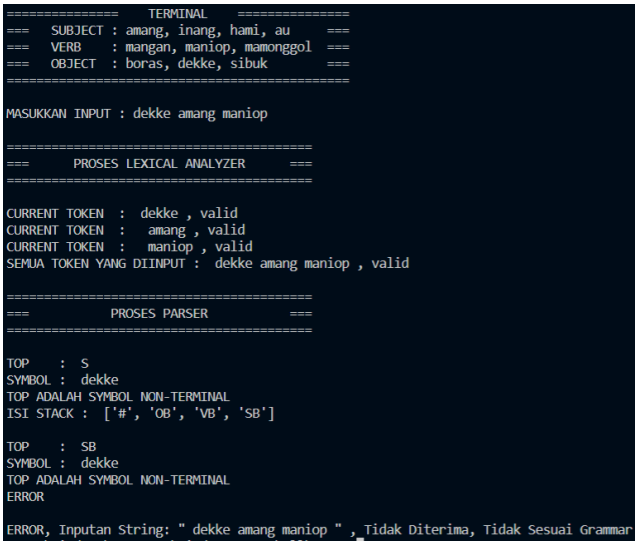
```
===============    TERMINAL    ===============
===   SUBJECT : amang, inang, hami, au     ===
===   VERB    : mangan, maniop, mamonggol  ===
===   OBJECT  : boras, dekke, sibuk        ===
=============================================

MASUKKAN INPUT : au maniop dekke

=======================================
===      PROSES LEXICAL ANALYZER      ===
=======================================

CURRENT TOKEN  :  au , valid
CURRENT TOKEN  :   maniop , valid
CURRENT TOKEN  :   dekke , valid
SEMUA TOKEN YANG DIINPUT :  au maniop dekke , valid
```

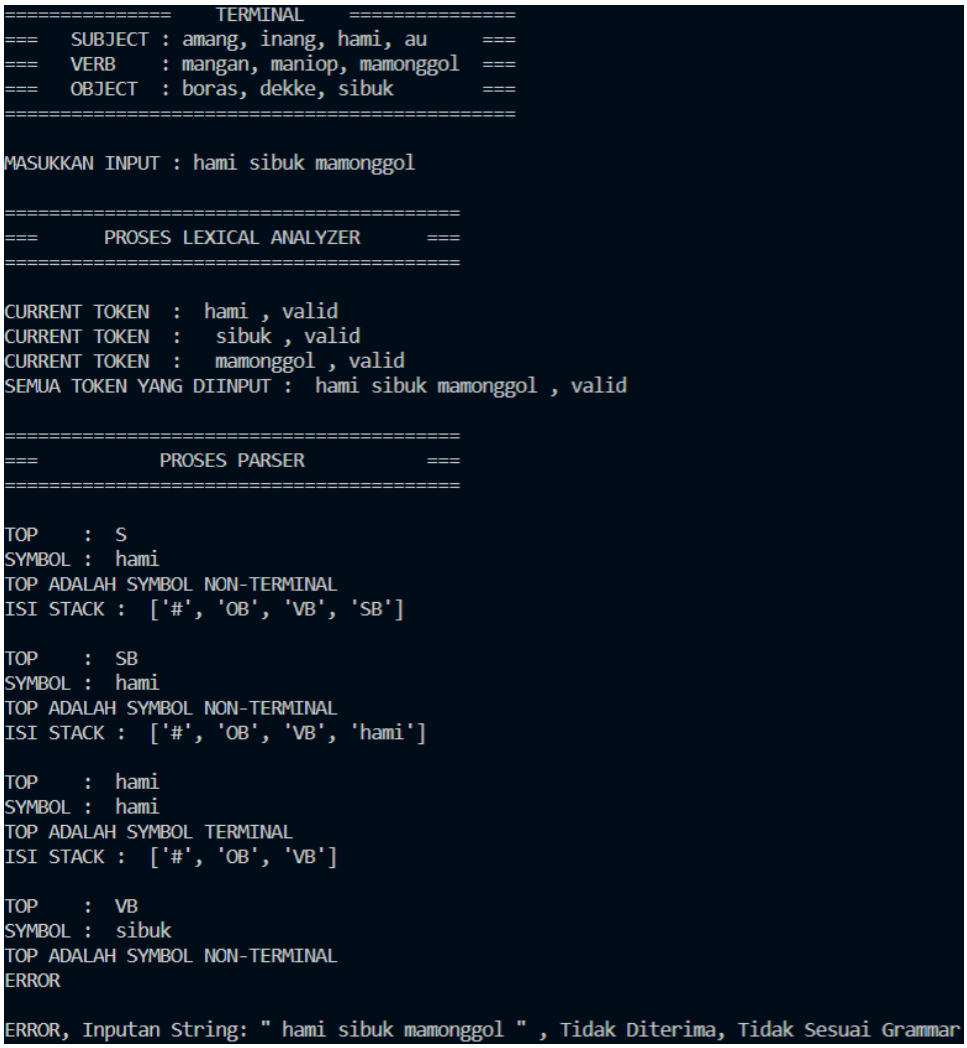
```
=====================================
===          PROSES PARSER          ===
=====================================

TOP    :  S
SYMBOL :  amang
TOP ADALAH SYMBOL NON-TERMINAL
ISI STACK :  ['#', 'OB', 'VB', 'SB']

TOP    :  SB
SYMBOL :  amang
TOP ADALAH SYMBOL NON-TERMINAL
ISI STACK :  ['#', 'OB', 'VB', 'amang']

TOP    :  amang
SYMBOL :  amang
TOP ADALAH SYMBOL TERMINAL
ISI STACK :  ['#', 'OB', 'VB']

TOP    :  VB
SYMBOL :  mangan
TOP ADALAH SYMBOL NON-TERMINAL
ISI STACK :  ['#', 'OB', 'mangan']

TOP    :  mangan
SYMBOL :  mangan
TOP ADALAH SYMBOL TERMINAL
ISI STACK :  ['#', 'OB']

TOP    :  OB
SYMBOL :  boras
TOP ADALAH SYMBOL NON-TERMINAL
ISI STACK :  ['#', 'boras']

TOP    :  boras
SYMBOL :  boras
TOP ADALAH SYMBOL TERMINAL
ISI STACK :  []
ISI STACK :  []


Inputan String  " amang mangan boras " Diterima, Sesuai Grammar
```

```
=====================================
===          PROSES PARSER          ===
=====================================

TOP    :  S
SYMBOL :  au
TOP ADALAH SYMBOL NON-TERMINAL
ISI STACK :  ['#', 'OB', 'VB', 'SB']

TOP    :  SB
SYMBOL :  au
TOP ADALAH SYMBOL NON-TERMINAL
ISI STACK :  ['#', 'OB', 'VB', 'au']

TOP    :  au
SYMBOL :  au
TOP ADALAH SYMBOL TERMINAL
ISI STACK :  ['#', 'OB', 'VB']

TOP    :  VB
SYMBOL :  maniop
TOP ADALAH SYMBOL NON-TERMINAL
ISI STACK :  ['#', 'OB', 'maniop']

TOP    :  maniop
SYMBOL :  maniop
TOP ADALAH SYMBOL TERMINAL
ISI STACK :  ['#', 'OB']

TOP    :  OB
SYMBOL :  dekke
TOP ADALAH SYMBOL NON-TERMINAL
ISI STACK :  ['#', 'dekke']

TOP    :  dekke
SYMBOL :  dekke
TOP ADALAH SYMBOL TERMINAL
ISI STACK :  []
ISI STACK :  []


Inputan String  " au maniop dekke " Diterima, Sesuai Grammar
```

- 3 Kalimat (inputan) yang tidak sesuai Grammar







PENUTUP

A. Kesimpulan

Setelah melakukan Lexical Analysis dan Parser, dimana Lexical Analysis untuk tahap awal dalam penerimaan inputan untuk dicek validitasnya dan dapat dilanjutkan ke proses Parser, yang mana proses untuk mengecek susunan inputan tersebut sudah memenuhi aturan grammar. Pada parser, data atau informasi yang dipecah menjadi bagian - bagian komponen sehingga sintaksnya dapat dianalisis, dikategorikan, serta dipahami. Tujuan parser adalah untuk melibatkan pencarian pohon parse (Parse Tree) untuk menemukan derivasi paling kiri dari input stream dengan menggunakan ekspansi top - down dan melibatkan penulisan ulang input kembali ke simbol awal. Cara kerja parser salah satunya adalah Lexical Analysis yang pada tahap awal digunakan untuk menghasilkan token dari aliran karakter string input, yang dipecah menjadi komponen kecil untuk membentuk ekspresi yang bermakna.

Dari hasil pengerjaan tugas besar yang telah kami lakukan secara keseluruhan, kami dapat menentukan bahwa menggunakan grammar (tata bahasa) bahasa Batak dengan struktur SB - VB - OB (subjek - predikat - objek) membuat program berjalan dengan baik, oleh karena itu jika struktur bahasa tersebut terbalik, tertukar ataupun tidak sesuai, maka akan mempengaruhi ketidaksesuaian grammar.

B. Petunjuk Running Kode Program

Berikut langkah - langkah untuk menjalankan kode program yang telah kami buat :

1. Buka file kode program “Kelompok 14 - Lexical dan Parser.py” yang sudah kelompok kami buat dalam bahasa pemrograman python dengan aplikasi seperti Visual Studio Code, dll.
2. Atau buka link google collab berikut : https://colab.research.google.com/drive/1UkfAWf1wq7WZNQS1gAzuoUG9ANqtIFjH
3. Run program
4. Inputkan masukan (string) dengan kata yang sudah kami sediakan di terminal
5. Lalu tekan enter untuk melanjutkan
6. Akan muncul hasil atau keluaran (output) valid atau tidak validnya kata (string) yang telah diinputkan

C. Lampiran

Dokumen Laporan :
https://docs.google.com/document/d/1uXpE0gmxg8eMoMRkn0_-uS8YrtHbz1taVIVL8gn8Jkw/edit?usp=sharing

Kode Program Lexical :
https://colab.research.google.com/drive/1UlrCSDgxLevEsRV2TJQr9sxghxfpdq7_?usp=sharing

Kode Program Parser :
https://colab.research.google.com/drive/1vxNfGPuhI7JuDmV9NzQoIImYU4344oxk

Kode Program Penggabungan Lexical dan Parser :
https://colab.research.google.com/drive/1UkfAWf1wq7WZNQS1gAzuoUG9ANqtIFjH

D. Daftar Pustaka

"Classification of Context Free Grammars." *GeeksforGeeks*, 21 Nov. 2019, www.geeksforgeeks.org/classification-of-context-free-grammars/?ref=lbp.

"Introduction of Lexical Analysis." *GeeksforGeeks*, 28 June 2021, www.geeksforgeeks.org/introduction-of-lexical-analysis/?ref=lbp.

Aho, A. V., and J. D. Ullman. "Translations on a Context Free Grammar." *Information and Control*, vol. 19, no. 5, 1971, pp. 439–75. *Crossref*, https://doi.org/10.1016/s0019-9958(71)90706-6.

PERRIN, Dominique. "Finite Automata." *Formal Models and Semantics*, 1990, pp. 1–57. *Crossref*, https://doi.org/10.1016/b978-0-444-88074-1.50006-8.