COMPILER BAHASA PYTHON

LAPORAN TUGAS BESAR

Diajukan untuk Memenuhi Tugas Besar
IF2124 Teori Bahasa Formal dan Otomata

oleh

Apa Aja

DAMIANUS CLAIRVOYANCE DIVA P.	**13520035**
JEREMY S.O.N. SIMBOLON	**13520042**
HANSEL VALENTINO TANOTO	**13520046**

TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2021

DAFTAR ISI

BAB I
DASAR TEORI

1. *FINITE AUTOMATA* (FA)

Finite Automata merupakan salah satu *finite-state machine* (FSM). FSM adalah mesin abstrak yang terdiri atas himpunan state Q, himpunan input I, himpunan output Z, dan fungsi transisi delta. Fungsi transisi menerima *state* dan simbol input, lalu berpindah ke *state* baru.

Finite Automata menerima bahasa regular, yang secara formal didefinisikan dalam 5-tuple, yaitu sebagai berikut.

$$(Q, \Sigma, \delta, q0, F)$$

Q: himpunan *state*
Σ: himpunan simbol/alfabet input
δ: fungsi transisi
q0: *start state* pada Q
F: *final state* pada Q

Finite Automata terbagi menjadi *deterministic finite automata* (DFA) dan *nondeterministic finite automata* (NFA). Perbedannya terletak pada fungsi transisi. Pada NFA, satu simbol input pada suatu *state* bisa menuju ke lebih dari satu *state*, sehingga fungsi transisinya menghasilkan himpunan *state*, bukan satu *state* tertentu.

2. *CONTEXT-FREE GRAMMAR* (CFG)

CFG merupakan bahasa dengan tingkat yang lebih tinggi daripada *regular grammar* (yang ditangani oleh FA) pada tingkatan *grammar* Chomsky. Secara formal, CFG didefinisikan dalam 4-tuple, yaitu sebagai berikut.

$$G = (V, T, S, P)$$

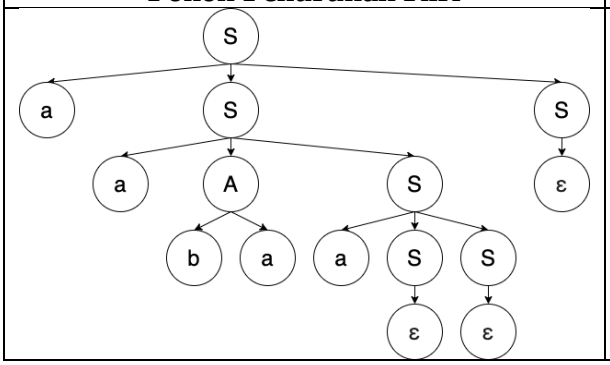
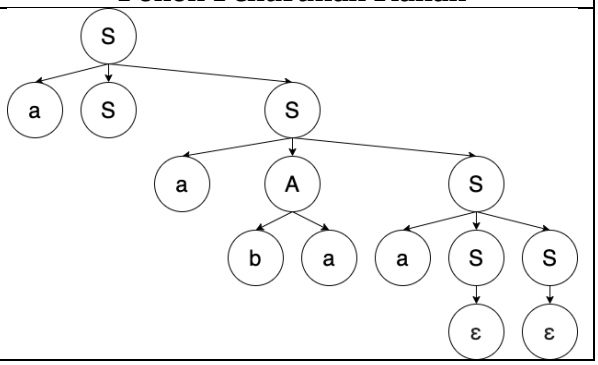
V: himpunan variabel atau simbol non-terminal
T: himpunan simbol terminal
S: *start symbol*
P: aturan produksi

Aturan produksi berbentuk $\alpha \rightarrow \beta$, dengan α dan β adalah string dalam V atau T, serta ruas kiri (*left-hand-side* [LHS]) merupakan variabel.

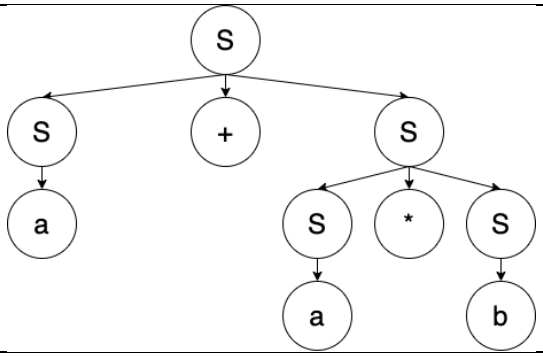
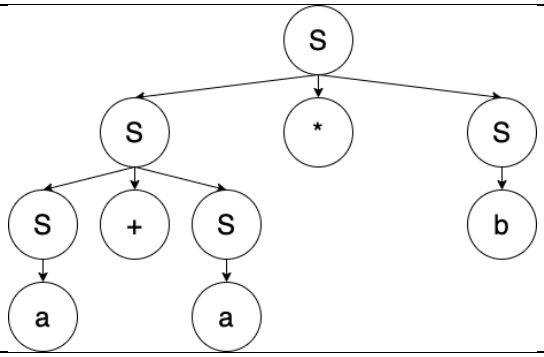
Dalam CFG, sebuah string dapat diperiksa keanggotaannya dalam suatu *grammar* dengan menelusuri aturan produksi. Sebagai contoh, misalkan terdapat CFG G = {V, T, S, P} dengan P: S → aAS | aSS | ε, A → SbA | ba, keanggotaan string aabaa dalam G dapat diverifikasi dengan langkah berikut.

S → aSS => S → aaAS => S → aabaS => S → aabaaSS => S → aabaaS => S → aabaa

Alur penurunan tersebut dapat divisualisasikan dengan pohon penurunan (atau *pohon parsing*). Pohon penurunan (d*erivation tree*) adalah pohon terurut (*ordered tree*) yang merepresentasikan informasi semantik dari string, dengan simbol non-terminal sebagai simpul, simbol terminal sebagai daun, dan *start symbol* sebagai akar. Terdapat dua submetode pohon penurunan kiri (*left derivation tree*) dan pohon penurunan kanan (r*ight derivation tree*). Pohon penurunan kiri ialah penerapan aturan produksi terhadap variabel paling kiri pada tiap langkah), sedangkan pohon penurunan kanan ialah penerapan aturan produksi terhadap variabel paling kanan pada tiap langkah. Kembali lagi pada contoh di atas, pohon penurunannya digambarkan sebagai berikut.

Pohon Penurunan Kiri	**Pohon Penurunan Kanan**
	

Dalam CFG pula, terdapat kasus ambiguitas, yakni bila terdapat dua atau lebih pohon penurunan untuk string omega yang sama. Sebagai contoh, misalkan terdapat CFG G = ({S}, {a, b, +, *}, P, S) dengan P: S → S + S | S * S | a | b, maka string a + a * b dapat diturunkan dengan dua pohon penurunan.

Alur Pertama	**Alur Kedua**
	
S → S + S S → a + S S → a + S * S S → a + a * S S → a + a * b	S → S * S S → S + S * S S → a + S * S S → a + a * S S → a + a * b

Maka, string a + a * b disebut string ambigu karena memiliki dua pohon penurunan, sedangkan *grammar* G disebut *grammar* ambigu karena menghasilkan sedikitnya satu string yang ambigu.

3. CHOMSKY *NORMAL FORM* (CNF)

Chomsky Normal Form (CNF) merupakan salah satu bentuk normal dari Context-Free Grammar (CFG). Tujuan dari konversi CFG ke CNF ialah menyederhanakan bentuk CFG dengan menghilangkan *useless* (atau *unreachable*) *production*, *unit production*, dan *null production*. Akan tetapi, CNF tetap memiliki produksi yang sama dengan CFG awal.

CNF merupakan CFG yang memenuhi syarat aturan produksi: ruas kanan (*right-hand-side* [RHS]) hanya boleh memiliki dua variabel saja atau satu terminal saja,

$$A \rightarrow a,$$
$$A \rightarrow BC,$$

dengan A, B, dan C adalah non-terminal (variabel) dan a adalah terminal.

Secara garis besar, algoritma konversi CFG ke CNF ialah sebagai berikut.

1. Tambahkan produksi S' → S bila terdapat *start symbol* S pada RHS suatu produksi.
2. Hapus *null productions* (produksi A → ε atau A → ... → ε, dengan A adalah non-terminal).
3. Hapus *unit productions* (produksi A → B, dengan A, B adalah non-terminal).
4. Ganti produksi A → $B_1...B_n$ dengan n > 2, dengan A → B_1C dan C → $B_2...B_n$. Ulangi sampai tidak ada produksi yang RHS-nya memiliki lebih dari dua variabel.
5. Ganti produksi A → aB dengan a adalah terminal dan A, B adalah non-terminal, dengan A → XB dan X → a. Ulangi sampai semua aturan produksi sudah dalam CNF.

Berikut diberikan contoh untuk memperlihatkan implementasi algoritma. Misalkan terdapat CFG dengan aturan produksi P

S → ASA | aB A → B | S B → b | ε

langkah konversi sesuai algoritmanya ialah sebagai berikut.

1. Tambahkan produksi S' → S karena terdapat *start symbol* S pada RHS.

   S' → S S → ASA | aB A → B | S B → b | ε

2. Hapus *null productions* (B → epsilon dan A → ε).
   a) Hapus B → ε,

      S' → S S → ASA | aB | a A → B | S | ε B → b

   b) Hapus A → ε,

      S' → S S → ASA | aB | a | AS | SA | S
      A → B | S B → b

3. Hapus *unit productions* (S → S, S' → S, A → B, dan A → S).
   a) Hapus S → S,

      S' → S S → ASA | aB | a | AS | SA

A → B | S B → b

b) Hapus S' → S,
S' → ASA | aB | a | AS | SA S → ASA | aB | a | AS | SA
A → B | S B → b

c) Hapus A → B,
S' → ASA | aB | a | AS | SA S → ASA | aB | a | AS | SA
A → b | S B → b

d) Hapus A → S
S' → ASA | aB | a | AS | SA S → ASA | aB | a | AS | SA
A → b | ASA | aB | a | AS | SA B → b

4. Ubah produksi yang RHS-nya memiliki lebih dari dua variabel (S' → ASA, S → ASA, dan A → ASA) dengan simbol produksi baru.
S' → AC | aB | a | AS | SA S → AC | aB | a | AS | SA
A → b | AC | aB | a | AS | SA B → b
C → SA

5. Ubah produksi yang RHS-nya belum dalam CNF (S' → aB, S → aB, dan A → aB), sehingga diperoleh aturan produksi yang sudah dalam CNF.
S' → AC | DB | a | AS | SA S → AC | DB | a | AS | SA
A → b | AC | DB | a | AS | SA B → b
C → SA D → a

4. COCKE-YOUNGER-KASAMI (CYK)

Cocke-Younger-Kasami (CYK) merupakan salah satu algoritma *parsing* yang bertujuan untuk mengecek keanggotaan dari suatu string terhadap *Context-Free Language* (CFL) tertentu. Sesuai namanya, algoritma ini diciptakan oleh J. Cocke, DH. Younger, dan T. Kasami. Sebelum menggunakan algoritma ini *Context-Free Grammar* (CFG) harus dikonversi terlebih dahulu ke dalam format CNF (Chomsky Normal Form). Algoritma ini bekerja secara *bottom-up* yaitu dimulai dari bawah (daun dari *parse tree*) sampai ke atas (akar dari *parse tree*).

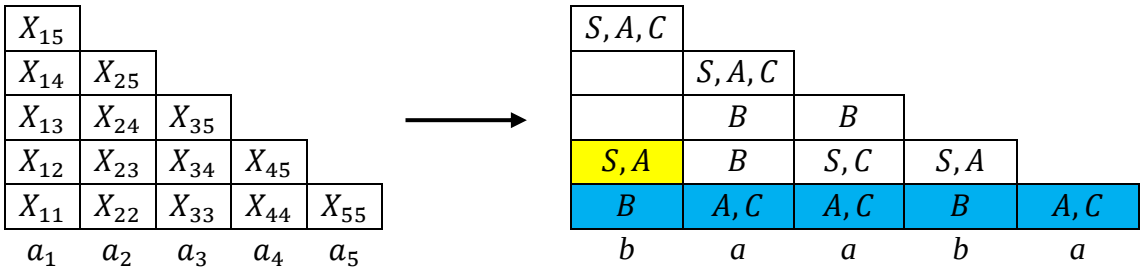
Untuk menggunakan algoritma ini, pertama-tama, akan dibuat *triangular array* berukuran $n \times n$, untuk *string* $w = a_1 a_2 \dots a_n$, lalu elemen array akan diisi dengan $X_{ij} = \{variables\ A \mid A =>* a_i..a_j\}$. Baris pada tabel menyatakan panjang dari substring, sedangkan kolom pada tabel menyatakan indeks karakter pertama dari substring. Basis dari *parse table* tersebut (baris paling bawah) dapat diperoleh dengan mencocokkan setiap *non-terminal symbol* pada string dengan *terminal symbol*-nya sesuai *production rule* pada CFG (yang sudah dalam format CNF). Selanjutnya, untuk baris diatasnya, akan dicari kombinasi *terminal symbol* baris di bawahnya lalu kembali dicocokkan dengan *production rule* yang ada.

Jika pada X_{1n} (bagian paling atas/*root*) terdapat *start symbol* (umumnya S), maka *string* tersebut merupakan bagian dari bahasa tersebut.

Misalkan terdapat CFG dengan aturan produksi (S adalah *Start Symbol*) seperti di bawah ini:

S → AB | BC
A → BA | a
B → CC | b
C → AB | a

Jika, string yang akan dites adalah $w = baaba$ maka akan terbentuk tabel/*array* sebagai berikut:



Baris yang berwarna biru pada tabel di atas merupakan basis yang diperoleh dengan mencocokkan karakter 'a' dan 'b' dengan *terminal symbol*-nya sesuai *production rule*. Sel yang berwarna kuning merupakan representasi dari *substring* dengan panjang 2 dan dimulai dengan 'b', yaitu "ba". Kombinasi *terminal symbol* yang mungkin adalah BA dan BC yang jika dicocokkan dengan *production rule*, diperoleh *terminal symbol* S dan A. Hal yang sama akan dilakukan untuk setiap sel sampai ke akar (baris paling atas). Karena terdapat *start symbol* S pada akar (*root*), maka *string* 'baaba' merupakan bagian dari CFG di atas.

5. BAHASA PYTHON

Python merupakan bahasa pemrograman tingkat tinggi yang bersifat interpretative dan multi-paradigma yang diciptakan oleh Guido van Rossum dan sudah diperkenalkan sejak tahun 1991. Python termasuk bahasa pemrograman yang mudah dipelajari karena *syntax* nya yang sederhana (ringkas) sehingga memiliki tingkat keterbacaan yang tinggi dan mudah dipahami. Bahasa Python sering digunakan untuk membuat berbagai program yang berkaitan dengan *Data Science*, *Machine Learning*, dan *Internet of Things*. Python memiliki pustaka (*library*) yang luas, komunitas yang besar saat ini, serta mendukung modularitas program sehingga dapat memudahkan proses pengembangan program dan proses *debugging*.

Salah satu hal yang perlu diperhatikan dalam pembuatan program parser bahasa Python adalah penamaan variabel, dimana variabel hanya dapat diawali dengan huruf dan *underscore* ('_'), hanya mengandung karakter alfa-numerik, dan bersifat *case-sesitive*. Berbeda dengan Bahasa pemrograman lain seperti C, pada Python tidak diperlukan deklarasi variabel berdasarkan tipenya melainkan tipe data variabel akan ditentukan langsung oleh nilai yang dimasukkan ke dalamnya. Hal yang perlu diperhatikan dalam bahasa Python adalah indentasi yang akan menyatakan blok program, tetapi dalam program Python Parser kali ini, indentasi akan

diabaikan. Beberapa tipe data dasar yang didukung oleh Python yaitu boolean (True dan False), number (int, float, complex), dan string (dengan “” atau ‘’). Penulisan comment di Python adalah dengan menggunakan ‘#’ untuk *single-line comment* dan tanda petik sebanyak tiga kali untuk menandakan awal dan akhir dari *multi-line comment*. Di Python terdapat beberapa kelompok operator yaitu operator aritmatika (*comparison*), logika (*logical*), identitas (*identity*), keanggotaan (*membership*), dan bitwise. Hal mengenai *syntax* yang perlu diperhatikan adalah tanda titik dua (‘:’) yang digunakan pada akhir *statement* def, if, elif, else, while, for, dan class yang fungsinya mirip dengan tanda kurung kurawal (‘{}’) pada bahasa C.

BAB II
ANALISIS PERSOALAN DAN DEKOMPOSISI

1. *FINITE AUTOMATA*

Implementasi *finite automata* (FA) dalam *compiler* bahasa Python ialah dalam pemeriksaan validitas nama variabel dan angka.

Untuk validitas nama variabel, sesuai dokumentasi Python dan *w3schools.com*, berikut merupakan ketentuan validitas variabel dalam bahasa Python.

- Variabel harus dimulai oleh karakter huruf (A-z) atau karakter *underscore* (‘_’).
- Variabel tidak dapat dimulai oleh angka.
- Variabel hanya mengandung karakter alfanumerik (A-z, 0-9) dan underscore (‘_’).
- Penamaan variabel bersifat *case-sensitive*, artinya `nama`, `Nama`, dan `NAMA` merupakan 3 variabel yang berbeda.

Contoh variabel valid: `varsaya, var_saya, _var_saya, varSaya, VARSAYA, varsaya2`.
Contoh variabel tidak valid: `2varsaya, var-saya, var saya`.

Dari definisi tersebut, didesain diagram *deterministic finite automata* (DFA) sebagai berikut.

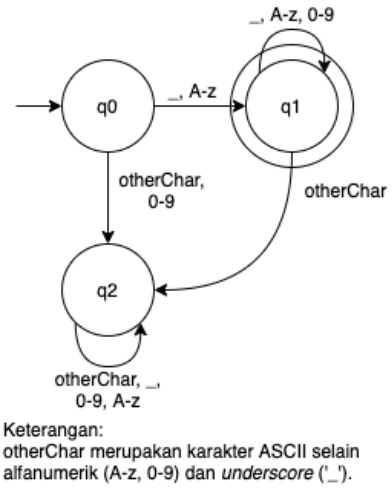


Diagram II.1.1. FA untuk Nama Variabel

Secara formal, DFA di atas didefinisikan dalam (Q, Σ, δ, q0, F) sebagai ({q0, q1, q2}, Σ, δ, q0, q1), dengan q2 sebagai *dead state* dan Σ merupakan himpunan karakter ASCII. Berikut merupakan penjelasan untuk tiap *state* DFA.

Nama *State*	**Sifat *State***	**Fungsi**	**Contoh *Accepted String***
q0	*start state*	Memulai pembacaan string.	(bukan *final state*)

q1	*final state*	Menangani nama variabel yang sejauh ini masih valid.	varsaya, var_saya, _var_saya, varSaya, VARSAYA, varsaya2
q2	*dead state*	Menangani nama variabel yang sudah tidak valid.	(bukan *final state*)

Untuk validitas bilangan (*integer*, *float*, dan eksponen dalam notasi saintifik), didesain diagram FA berikut.

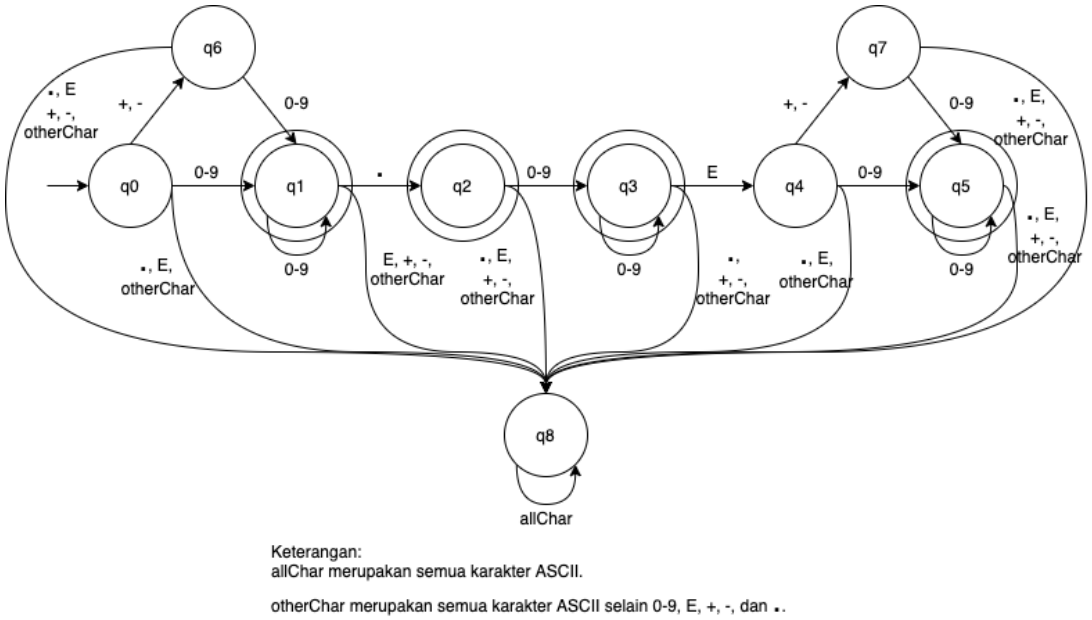


Diagram II. 1. 2. FA untuk Bilangan

Secara formal, FA didefinisikan dalam (Q, Σ, δ, q0, F) sebagai ({q0, q1, q2, q3, q4, q5, q6, q7}, Σ, δ, q0, {q1, q2, q3, q5}) dengan q8 sebagai *dead state* dan Σ merupakan himpunan karakter ASCII. Berikut merupakan penjelasan untuk tiap *state* FA.

Nama *State*	**Sifat *State***	**Fungsi**	**Contoh *Accepted String***
q0	*start state*	Memulai pembacaan string.	(bukan *final state*)
q1	*final state*	Menangani bilangan bulat.	0, +0, -0, +01, -0023, 000456, +2, -1, 3, +45, -567, 7890.
q2	*final state*	Menangani bilangan bulat dengan titik di akhir.	0., +0., -0., +01., -0023., 000456., +2., -1., 3., +45., -567., 7890..
q3	*final state*	Menangani bilangan bulat dengan desimal.	0.1, +0.023, -0.00456, +1.2, -23.45, 678.901.
q4	*state* biasa	Menangani bilangan bulat dengan desimal dan E di akhir.	(bukan *final state*)

q5	*final state*	Menangani bilangan bernotasi saintifik.	0.1E2, 0.034E-56, 0.007E+891, +0.023E+2, -0.00456E-2, +1.2E+2, -23.45E-2, 678.901E2.
q6	*state* biasa	Menangani + atau – pada awal string.	(bukan *final state*)
q7	*state* biasa	Menangani + atau – setelah bilangan bulat dengan desimal dan E.	(bukan *final state*)
q8	*dead state*	Menangani bilangan yang sudah tidak valid.	(bukan *final state*)

2. *CFG PRODUCTION*

Berikut adalah *Context-Free Grammar* yang kami gunakan dalam membangun program ini.

$$G = (V, T, P, S)$$

Variables (V):

```
S Assignment Input Bracket Type Term Arithmetic Operand Operator Number String Text Boolean Relation Print If Expression Condition Body Elif Else Exception For While List Element Slice Index Tuple Set Function Return Module From Class Comment Open File
```

Terminals (T):

```
+ - * / % = & ^ | ~ @ ! > < ( ) [ ] { } " ' : , . # variable number string str int float False None True and as break class continue def elif else for from if import in is not or pass raise return while with input len list open print range round set tuple
```

Productions (P):

```
S -> S S | Assignment | Print | If | For | While | Function | Module | Class | File | Comment;
Assignment -> variable = Input | variable = Open | variable = Boolean;
Input -> input Bracket | Type ( Input );
Bracket -> ( ) | ( Boolean ) | ( S );
Type -> str | int | float;
Term -> variable | variable Bracket | variable . variable Bracket | len Bracket | Arithmetic | String | Open | Term , Term;
Arithmetic -> round Bracket | Operand | Operand Operator Operand | Operator Operand | Arithmetic Operator Operand | Bracket | Type Bracket;
Operand -> variable | Number | String;
Operator -> + | - | * | / | * * | / / | % | & | ^ | | | ~ | < < | > > | @ | : =;
Number -> number | number . number;
String -> " Text " | ' Text ' | Text | String + String;
Text -> string | Text Text;
```

```
Boolean -> True | False | None | Term | List | variable List | not Boolean
| Boolean and Boolean | Boolean or Boolean | Boolean is Boolean | Boolean
in Boolean | Boolean Relation Boolean;
Relation -> > | < | > = | < = | = = | ! =;
Print -> print Bracket;
If -> if Expression | if Expression If | if Expression Elif | if Expression
Else | If raise Exception | If break | If pass | If continue;
Expression -> Condition : Body;
Condition -> ( Boolean ) | Boolean;
Body -> raise Exception | break | pass | continue | Return | S | S raise
Exception | S break | S pass | S continue | S Return | Body Body;
Elif -> elif Expression | Elif Elif | Elif Else;
Else -> else : Body;
Exception -> variable;
For -> for variable in Term : Body | for variable in range Bracket :
Body;
While -> while Expression;
List -> [ ] | [ Element ] | [ S ] | [ List ] | list Bracket | List
Operator List | Index;
Element -> Term | Element , Element | Slice;
Slice -> : | : : | Term : Term |  : Term | Term : | Term : Term : Term |
: Term : Term | Term : : Term | Term : Term : | Term : : | : Term : | :
: Term;
Index -> [ Element ] | [ Element ] Index;
Tuple -> ( ) | ( Element ) | ( S ) | tuple Bracket;
Set -> { } | { Element } | { S } | set Bracket | Set Operator Set;
Function -> def variable Bracket : S | def variable Bracket : Return |
def variable Bracket : S Return;
Return -> return Boolean | return Term;
Module -> import variable | import variable as variable | From import
variable | From import variable as variable;
From -> from variable;
Class -> class variable : Body;
Comment -> # Text | " " " Text " " " | ' ' ' Text ' ' ';
Open -> open Bracket;
File -> with Open as variable : Body
```

Start Symbol (S):
S

3. *CNF PRODUCTION*

Berikut adalah Context-Free Grammar yang kami gunakan dalam membangun program ini.

$$G = (V, T, P, S)$$

Variables (V):

```
S Z4 Assignment A1 B1 C1 Y4 Input X4 W4 D1 D2 Bracket E1 F1 Type Term V4
G1 G2 U4 T4 H1 S4 Arithmetic I1 J1 Operand Operator Number K1 R4 String
L1 Q4 M1 N1 Text Boolean P4 O4 O1 N4 P1 M4 Q1 L4 R1 S1 Relation K4 J4
Print I4 If T1 U1 V1 H4 W1 Expression X1 Condition Y1 Body Z1 G4 Elif F4
Else A11 Exception E4 For B11 B12 B13 B14 D4 C11 C12 C13 C14 C15 C4 While
B4 A4 List D11 E11 F11 Z3 G11 H11 Element I11 Slice J11 K11 L11 L12 L13
M11 M12 N11 N12 O11 O12 P11 Q11 R11 Index S11 T11 T12 Tuple U11 W11 Y3
X3 W3 Set X11 Y11 V3 Z11 U3 Function A21 A22 A23 B21 B22 B23 C21 C22 C23
C24 T3 Return S3 Module R3 D21 D22 E21 F21 F22 F23 Q3 From P3 Class G21
G22 O3 Comment H21 H22 H23 H24 H25 I21 I22 I23 I24 I25 N3 Open M3 File
J21 J22 J23 J24 S0
```

Terminals (T):

```
+ - * / % = & ^ | ~ @ ! > < ( ) [ ] { } " ' : , . # variable number string
str int float False None True and as break class continue def elif else
for from if import in is not or pass raise return while with input len
list open print range round set tuple
```

```
S -> S S | Exception A1 | Exception B1 | Exception C1 | J4 Bracket | I4
Expression | I4 T1 | I4 U1 | I4 V1 | If W1 | If Body | If Body | If Body
| E4 B11 | E4 C11 | C4 Expression | U3 A21 | U3 B21 | U3 C21 | S3 Exception
| S3 D21 | From E21 | From F21 | P3 G21 | M3 J21 | O3 Text | R4 H21 | Q4
I21
Z4 -> =
Assignment -> Exception A1 | Exception B1 | Exception C1
A1 -> Z4 Input
B1 -> Z4 Open
C1 -> Z4 Boolean
Y4 -> input
Input -> Y4 Bracket | Type D1
X4 -> (
W4 -> )
D1 -> X4 D2
D2 -> Input W4
Bracket -> X4 W4 | X4 E1 | X4 F1
E1 -> Boolean W4
F1 -> S W4
Type -> str | int | float
Term -> variable | Exception G1 | Exception Bracket | U4 Bracket | Term
H1 | S4 Bracket | Operand I1 | Operator Operand | Arithmetic J1 | Type
```

```
Bracket | R4 L1 | Q4 M1 | String N1 | N3 Bracket | variable | number |
Number K1 | R4 L1 | Q4 M1 | String N1 | X4 W4 | X4 E1 | X4 F1 | string |
Text Text | string | Text Text
V4 -> .
G1 -> V4 G2
G2 -> Exception Bracket
U4 -> len
T4 -> ,
H1 -> T4 Term
S4 -> round
Arithmetic -> S4 Bracket | Operand I1 | Operator Operand | Arithmetic J1
| Type Bracket | variable | X4 W4 | X4 E1 | X4 F1 | number | Number K1 |
R4 L1 | Q4 M1 | String N1 | string | Text Text
I1 -> Operator Operand
J1 -> Operator Operand
Operand -> variable | number | Number K1 | R4 L1 | Q4 M1 | String N1 |
string | Text Text
Operator -> + | - | * | / | Operator Operator | Operator Operator | % |
& | ^ | | | ~ | Relation Relation | Relation Relation | @ | Slice Z4
Number -> number | Number K1
K1 -> V4 Number
R4 -> "
String -> R4 L1 | Q4 M1 | String N1 | string | Text Text
L1 -> Text R4
Q4 -> '
M1 -> Text Q4
N1 -> Operator String
Text -> string | Text Text
Boolean -> True | False | None | Exception List | P4 Boolean | Boolean
O1 | Boolean P1 | Boolean Q1 | Boolean R1 | Boolean S1 | variable |
Exception G1 | Exception Bracket | U4 Bracket | Term H1 | B4 A4 | B4 D11
| B4 E11 | B4 F11 |  G11 | List H11 | S4 Bracket | Operand I1 | Operator
Operand | Arithmetic J1 | Type Bracket | variable | X4 W4 | X4 E1 | X4
F1 | R4 L1 | Q4 M1 | String N1 | string | Text Text | N3 Bracket | B4 S11
| B4 T11 | number | Number K1 | R4 L1 | Q4 M1 | String N1 | string | Text
Text
P4 -> not
O4 -> and
O1 -> O4 Boolean
N4 -> or
P1 -> N4 Boolean
M4 -> is
Q1 -> M4 Boolean
L4 -> in
R1 -> L4 Boolean
S1 -> Relation Boolean
Relation -> > | < | Relation Z4 | Relation Z4 | Z4 Z4 | K4 Z4
K4 -> !
J4 -> print
```

```
Print -> J4 Bracket
I4 -> if
If -> I4 Expression | I4 T1 | I4 U1 | I4 V1 | If W1 | If Body | If Body
| If Body
T1 -> Expression If
U1 -> Expression Elif
V1 -> Expression Else
H4 -> raise
W1 -> H4 Exception
Expression -> Condition X1
X1 -> Slice Body
Condition -> X4 Y1 | True | False | None | Exception List | P4 Boolean |
Boolean O1 | Boolean P1 | Boolean Q1 | Boolean R1 | Boolean S1 | variable
| Exception G1 | Exception Bracket | U4 Bracket | Term H1 | S4 Bracket |
Operand I1 | Operator Operand | Arithmetic J1 | Type Bracket | R4 L1 |
Q4 M1 | String N1 | N3 Bracket | B4 A4 | B4 D11 | B4 E11 | B4 F11 |  G11
| List H11 | B4 S11 | B4 T11 | variable | number | Number K1 | R4 L1 |
Q4 M1 | String N1 | string | Text Text | X4 W4 | X4 E1 | X4 F1 | string
| Text Text
Y1 -> Boolean W4
Body -> H4 Exception | break | pass | continue | S Z1 | S Body | S Body
| S Body | S Return | Body Body | T3 Boolean | T3 Term | S S | Exception
A1 | Exception B1 | Exception C1 | J4 Bracket | I4 Expression | I4 T1 |
I4 U1 | I4 V1 | If W1 | If Body | If Body | If Body | E4 B11 | E4 C11 |
C4 Expression | U3 A21 | U3 B21 | U3 C21 | S3 Exception | S3 D21 | From
E21 | From F21 | P3 G21 | M3 J21 | O3 Text | R4 H21 | Q4 I21
Z1 -> H4 Exception
G4 -> elif
Elif -> G4 Expression | Elif Elif | Elif Else
F4 -> else
Else -> F4 A11
A11 -> Slice Body
Exception -> variable
E4 -> for
For -> E4 B11 | E4 C11
B11 -> Exception B12
B12 -> L4 B13
B13 -> Term B14
B14 -> Slice Body
D4 -> range
C11 -> Exception C12
C12 -> L4 C13
C13 -> D4 C14
C14 -> Bracket C15
C15 -> Slice Body
C4 -> while
While -> C4 Expression
B4 -> [
A4 -> ]
```

```
List -> B4 A4 | B4 D11 | B4 E11 | B4 F11 |  G11 | List H11 | B4 S11 | B4
T11
D11 -> Element A4
E11 -> S A4
F11 -> List A4
Z3 -> list
G11 -> Z3 Bracket
H11 -> Operator List
Element -> Element I11 | variable | Exception G1 | Exception Bracket |
U4 Bracket | Term H1 | B4 A4 | B4 D11 | B4 E11 | B4 F11 |  G11 | List H11
| : | Slice Slice | Term J11 |  K11 | Term Slice | Term L11 | Slice M11
| Term N11 | Term O11 | Term P11 | Slice Q11 | Slice R11 | S4 Bracket |
Operand I1 | Operator Operand | Arithmetic J1 | Type Bracket | variable
| X4 W4 | X4 E1 | X4 F1 | R4 L1 | Q4 M1 | String N1 | string | Text Text
| N3 Bracket | B4 S11 | B4 T11 | number | Number K1 | R4 L1 | Q4 M1 |
String N1 | string | Text Text
I11 -> T4 Element
Slice -> : | Slice Slice | Term J11 |  K11 | Term Slice | Term L11 |
Slice M11 | Term N11 | Term O11 | Term P11 | Slice Q11 | Slice R11
J11 -> Slice Term
K11 -> Slice Term
L11 -> Slice L12
L12 -> Term L13
L13 -> Slice Term
M11 -> Term M12
M12 -> Slice Term
N11 -> Slice N12
N12 -> Slice Term
O11 -> Slice O12
O12 -> Term Slice
P11 -> Slice Slice
Q11 -> Term Slice
R11 -> Slice Term
Index -> B4 S11 | B4 T11
S11 -> Element A4
T11 -> Element T12
T12 -> A4 Index
Tuple -> X4 W4 | X4 U11 | X4 W11 | Y3 Bracket
U11 -> Element W4
W11 -> S W4
Y3 -> tuple
X3 -> {
W3 -> }
Set -> X3 W3 | X3 X11 | X3 Y11 | V3 Bracket | Set Z11
X11 -> Element W3
Y11 -> S W3
V3 -> set
Z11 -> Operator Set
U3 -> def
```

```
Function -> U3 A21 | U3 B21 | U3 C21
A21 -> Exception A22
A22 -> Bracket A23
A23 -> Slice S
B21 -> Exception B22
B22 -> Bracket B23
B23 -> Slice Return
C21 -> Exception C22
C22 -> Bracket C23
C23 -> Slice C24
C24 -> S Return
T3 -> return
Return -> T3 Boolean | T3 Term
S3 -> import
Module -> S3 Exception | S3 D21 | From E21 | From F21
R3 -> as
D21 -> Exception D22
D22 -> R3 Exception
E21 -> S3 Exception
F21 -> S3 F22
F22 -> Exception F23
F23 -> R3 Exception
Q3 -> from
From -> Q3 Exception
P3 -> class
Class -> P3 G21
G21 -> Exception G22
G22 -> Slice Body
O3 -> #
Comment -> O3 Text | R4 H21 | Q4 I21
H21 -> R4 H22
H22 -> R4 H23
H23 -> Text H24
H24 -> R4 H25
H25 -> R4 R4
I21 -> Q4 I22
I22 -> Q4 I23
I23 -> Text I24
I24 -> Q4 I25
I25 -> Q4 Q4
N3 -> open
Open -> N3 Bracket
M3 -> with
File -> M3 J21
J21 -> Open J22
J22 -> R3 J23
J23 -> Exception J24
J24 -> Slice Body
```

```
S0 -> S S | Exception A1 | Exception B1 | Exception C1 | J4 Bracket | I4 Expression | I4 T1 | I4 U1 | I4 V1 | If W1 | If Body | If Body | If Body | E4 B11 | E4 C11 | C4 Expression | U3 A21 | U3 B21 | U3 C21 | S3 Exception | S3 D21 | From E21 | From F21 | P3 G21 | M3 J21 | O3 Text | R4 H21 | Q4 I21
```

Start Symbol (S):
S

BAB III
IMPLEMENTASI DAN PENGUJIAN

1. SPESIFIKASI PROGRAM

A. *File* cfg_cnf.py

File cfg_cnf.py berisi fungsi-fungsi untuk melakukan konversi *Context-Free Grammar* menjadi *Chomsky Normal Form*. Kode program pada *file* ini diadaptasi dari program serupa buatan Adel Massimo Ramadan yang dapat ditemukan pada pranala https://github.com/adelmassimo/CFG2CNF. Berikut ini adalah fungsi yang terdapat pada *file* ini.

- `def is_unitary(rule, variables)`
  Fungsi ini berfungsi untuk memeriksa apakah `rule` masukan merupakan *unit production* serta apakah variabel pada `rule` terdapat pada `variables`.
- `def is_simple(rule)`
  Fungsi ini berfungsi untuk memeriksa apakah `rule` masukan merupakan *unit production* serta apakah variabel pada `rule` terdapat pada daftar variabel di `variable`.
- `def start(productions, variables)`
  Fungsi ini berfungsi untuk menambahkan aturan `S0 -> S` ke daftar aturan produksi `productions` yang telah ada di program.
- `def delete_mixed_terms(productions, variables)`
  Fungsi ini berfungsi untuk mengubah semua aturan produksi pada `productions` ke dalam *Chomsky Normal Form*.
- `def delete_nonunitary(productions, variables)`
  Fungsi ini berfungsi untuk menghilangkan semua aturan produksi pada `productions` yang memiliki lebih dari dua terminal atau variabel di sisi sebelah kanan produksi.
- `def delete_nonterminal(productions)`
  Fungsi ini berfungsi untuk menghilangkan semua aturan produksi pada `productions` yang memiliki ε pada aturan produksinya.
- `def unit_routine(rules, variables)`
  Fungsi ini berfungsi sebagai fungsi pembantu untuk menghilangkan *unit productions* pada `rules`. Fungsi ini menggabungkan fungsi-fungsi unit menjadi satu buah fungsi.
- `def delete_unit(productions, variables)`
  Fungsi ini berfungsi untuk menghilangkan *unit productions* pada `productions`. Proses ini dilakukan dengan cara melakukan iterasi pengubahan aturan hingga diperoleh daftar aturan yang tidak memiliki *unit productions*.

B. *File* cnf_helper.py

File cnf_helper.py berisi prosedur dan fungsi yang berperan sebagai fungsi pembantu dalam proses konversi *Context-Free Grammar* menjadi *Chomsky Normal Form*. Kode program pada *file* ini diadaptasi dari program serupa buatan Adel Massimo Ramadan yang dapat ditemukan pada pranala https://github.com/adelmassimo/CFG2CNF. Berikut ini adalah prosedur dan fungsi yang terdapat pada *file* ini.

- `def union(lst1, lst2)`
  Fungsi ini berfungsi untuk menggabungkan dua buah list masukan fungsi.
- `def load_model(model_path)`
  Fungsi ini membaca *file* uji, memisahkannya menjadi `terminals`, `variables`, dan `productions`, dan mengembalikan 3-tuple yang terdiri atas hasil sanitasi tiap list.
- `def clean_productions(expression)`
  Fungsi ini berfungsi untuk melakukan sanitasi pada aturan produksi dan mengubahnya ke dalam bentuk list yang siap diolah di tahap selanjutnya.
- `def clean_alphabet(expression)`
  Fungsi ini berfungsi untuk melakukan sanitasi pada terminal dan variabel serta mengubahnya ke dalam bentuk list yang siap diolah di tahap selanjutnya.
- `def seek_and_destroy(target, productions)`
  Fungsi ini berfungsi untuk memisahkan `productions` yang tidak mengandung `target` dari `productions` yang mengandung `target`.
- `def setup_dict(productions, variables, terms)`
  Fungsi ini berfungsi untuk membuat *dictionary* yang berisikan aturan dengan satu terminal.
- `def rewrite(target, production)`
  Fungsi ini berfungsi untuk menuliskan ulang kombinasi dari produksi masukan agar dapat diproses lebih lanjut.
- `def dict_to_set(dictionary)`
  Fungsi ini berfungsi untuk mengonversi *dictionary* menjadi *set*.
- `def print_rules(rules)`
  Prosedur ini berfungsi untuk menuliskan aturan produksi ke layar.
- `def pretty_form(rules)`
  Fungsi ini berfungsi untuk mengubah aturan produksi yang berbentuk list of list menjadi bentuk notasi CFG yaitu `var -> term`.

C. *File* token_machine.py

File token_machine.py berisi fungsi untuk melakukan tokenisasi terhadap *input file* yang akan di-*parsing*. Berikut ini adalah fungsi yang terdapat pada *file* ini.

- `def token(file_name)`
  Fungsi ini berfungsi untuk mengubah *string* hasil pembacaan *file input* file_name menjadi token-token yang akan digunakan saat menerapkan algoritma CYK. Misalnya *statement* **`if (x == 1): print(x)`** akan ditokenisasi menjadi **`['if', 'x', '=', '=', '1', ':', 'print', '(', 'x', ')']`**.
- `def ignoreComment(Str)`
  Fungsi ini berfungsi untuk menghapus komentar baik *multi-line* maupun *single-line* yang ada pada variabel *input* Str. Variabel Str merupakan representasi *string* dari kode program yang di-*parsing*.
- `def findError(file_name,error)`
  Fungsi ini berfungsi untuk mencari lokasi baris suatu variabel pada program yang di-*parsing* yang sudah teridentifikasi *invalid* menurut *Finite Automata*.

D. *File* fa.py

File fa.py berisi fungsi yang merepresentasikan *finite automata* untuk melakukan validasi terhadap nama variabel dan angka. Berikut ini adalah fungsi yang terdapat pada *file* ini. Penjelasan lebih lanjut telah disampaikan pada Bab II.

- `def validVarName(varName)`
  Fungsi ini menerima string berupa nama variabel yang akan divalidasi dan menghasilkan tipe boolean, true bila nama variabel valid dan false bila tidak valid.
- `def allVarNameValid(varNames)`
  Fungsi ini menerima array berisi kumpulan string berupa nama variabel yang akan divalidasi dan menghasilkan tipe boolean, true bila seluruh nama variabel valid dan false bila terdapat nama variabel tidak valid dalam array tersebut.
- `def validNum(input)`
  Fungsi ini menerima string berupa angka yang akan divalidasi dan menghasilkan booelan, true jika angka sesuai notasi saintifik dan false bila tidak sesuai.

E. *File* cyk.py

File cyk.py berisi fungsi untuk mengimplementasikan algoritma Cocke-Younger-Kasami. Berikut ini adalah fungsi yang terdapat pada *file* ini.

- `def cnf_to_dictionary(cnf_file)`
  Fungsi ini berfungsi untuk mengonversi *production rule dalam format* CNF dari bentuk *text file* (.txt) menjadi struktur data *dictionary* di Python. *Key* dari *dictionary* berupa *right hand side* (RHS) dari production rule, sedangkan *value*-nya berupa *left hand side* (LHS) dari *production rule*.
- `def cyk(tkn,dictionary)`
  Fungsi ini merupakan fungsi utama dalam modul ini yang digunakan untuk menerapkan algoritma CYK pada token sesuai dengan *rule* yang terdapat pada *dictionary*.
- `def cykBase(tkn,dictionary)`
  Fungsi ini digunakan untuk menghasilkan basis (baris terbawah) pada *parsing table* CYK.
- `def displayParseTable(parse_table)`
  Fungsi ini digunakan untuk menampilkan isi dari parse table ke terminal.

F. File main.py

File main.py merupakan program utama dalam Python *Parser* ini. *File* ini akan meng-*import* file token_machine.py, fa.py, dan cyk.py yang akan digunakan dalam melakukan *parsing*. Program akan melakukan tokenisasi terhadap isi *input file source code* Python dan membaca *grammar* yang terdapat pada *file* cnf.txt. Kemudian akan dilakukan *replacement* token-token variabel dan bilangan yang ada dengan *string* ‘variable’ dan ‘number’ untuk memudahkan *patern matching* saat diterapkan algoritma CYK. Token-token tersebut kemudian akan diolah menggunakan algoritma CYK untuk menentukan validitas *syntax*. Jika dari hasil parsing diperoleh simbol non-terminal S, maka *source code* lolos kompilasi dan ditampilkan pesan “Accepted” sedangkan jika tidak akan ditampilkan pesan “Syntax Error”.

Cara untuk menjalankan program Python Parser adalah sebagai berikut:

1) Ubah *directory* terminal ke folder src
2) Jalankan program utama (main.py) dengan mengetikkan **`py main.py`** di terminal
3) Ketikkan **`py <path_file>`** untuk melakukan *parsing* dengan <path_file> adalah *path* dari *file* yang akan di-*parsing*
4) Untuk menampilkan token dan parse table, ketik **`token`**

Selain itu, program juga dapat dijalankan dengan *Command Line Argument* yaitu dengan mengetikkan **`py main.py <path_file>`** pada *directory* src di terminal

2. PENGUJIAN PROGRAM

A. Kasus 1 – Input/Output dan Operasi Dasar

Source Code: **`case1.py`**

```
PythonParser > test > case1.py > ...
x = 2
y = int(input())
z = input()
# ini komentar
a = x ** y / 2 - 4
b = z + 'blablabla'
print(a, '--> ini a')
print(b)
print("ini string")
'''
ini komentar juga
'''
```

Hasil *parsing*: **Accepted**

```
PS D:\Personal\OneDrive - Institut Teknologi Bandung\Documents\Programming\GitHub\PythonParser\src> py main.py
Python Parser
Type "help" for more information.
>>> py ../test/case1.py
Accepted

>>> token
Token:
['x', '=', '2', 'y', '=', 'int', '(', 'input', '(', ')', ')', 'z', '=', 'input', '(', ')', 'a', '=', 'x', '*', '*', 'y', '/', '2', '-', '4', 'b', '=', 'z', '+', '
string', 'print', '(', 'a', ',', 'string', ')', 'print', '(', 'b', ')', 'print', '(', 'string', ')']
```

```
Parsing Table:
[['S', 'Body', 'S0']]
[[], []]
[[], [], []]
[[], [], [], ['S', 'Body', 'S0']]
[['S', 'Body', 'S0'], [], [], [], []]
[[], [], [], [], [], []]
[[], [], [], [], [], [], []]
[[], [], [], ['S', 'Body', 'S0'], [], [], [], []]
[['S', 'Body', 'S0'], [], [], [], [], [], [], [], []]
[[], [], [], [], [], [], [], [], [], []]
[[], [], [], [], [], [], [], [], [], [], []]
[[], [], [], ['S', 'Body', 'S0'], [], [], [], [], [], [], [], ['S', 'Body', 'S0']]
[[], [], [], [], [], [], [], [], [], [], [], [], []]
[[], [], [], [], [], [], [], [], [], [], [], [], [], []]
[['S', 'Body', 'S0'], [], [], [], [], [], [], [], [], [], [], [], [], [], []]
[[], [], [], [], [], [], [], [], [], [], [], ['S', 'Body', 'S0'], [], [], [], []]
[['S', 'Body', 'S0'], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], ['S', 'Body', 'S0']]
[[], [], [], ['S', 'Body', 'S0'], [], [], [], [], [], [], [], [], [], [], [], [], [], []]
[[], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], []]
[['S', 'Body', 'S0'], [], [], ['S', 'Body', 'S0'], [], [], [], [], [], [], [], ['S', 'Body', 'S0'], [], [], [], [], [], [], [], []]
[[], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], ['S', 'Body', 'S0'], [], [], [], []]
[['S', 'Body', 'S0'], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], []]
[[], [], [], ['S', 'Body', 'S0'], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], []]
[['S', 'Body', 'S0'], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], []]
[[], [], [], ['S', 'Body', 'S0'], [], [], [], [], [], [], [], [], [], [], [], [], ['S', 'Body', 'S0'], [], [], [], [], [], [], [], []]
[[], [], [], [], [], [], [], [], [], [], [], ['S', 'Body', 'S0'], [], [], [], [], [], [], [], [], [], [], [], [], [], []]
[['S', 'Body', 'S0'], [], [], ['S', 'Body', 'S0'], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], ['S', 'Body', 'S0']]
[[], [], [], [], [], [], [], [], [], [], [], ['S', 'Body', 'S0'], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], []]
[[], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], []]
[['S', 'Body', 'S0'], [], [], ['S', 'Body', 'S0'], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], []]
[[], [], [], [], [], [], [], [], [], [], [], ['S', 'Body', 'S0'], [], [], [], [], ['S', 'Body', 'S0'], [], [], [], [], [], [], [], [], [], ['S', 'Body', 'S0'], []
, [], [], []]
[[], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], ['S', 'Body', 'S0']]
[[], [], [], ['S', 'Body', 'S0'], [], [], [], [], [], [], [], ['S', 'Body', 'S0'], [], [], [], [], ['S', 'Body', 'S0'], [], [], [], [], [], [], [], [], [], [], []
```

Hasil *parsing* dari *source code* di atas adalah "*Accepted*" karena bagian akar dari *parse table* mengandung *start symbol* S.

B. Kasus 2 – *Loop* dan *Conditional*

Source code: **`case2.py`**

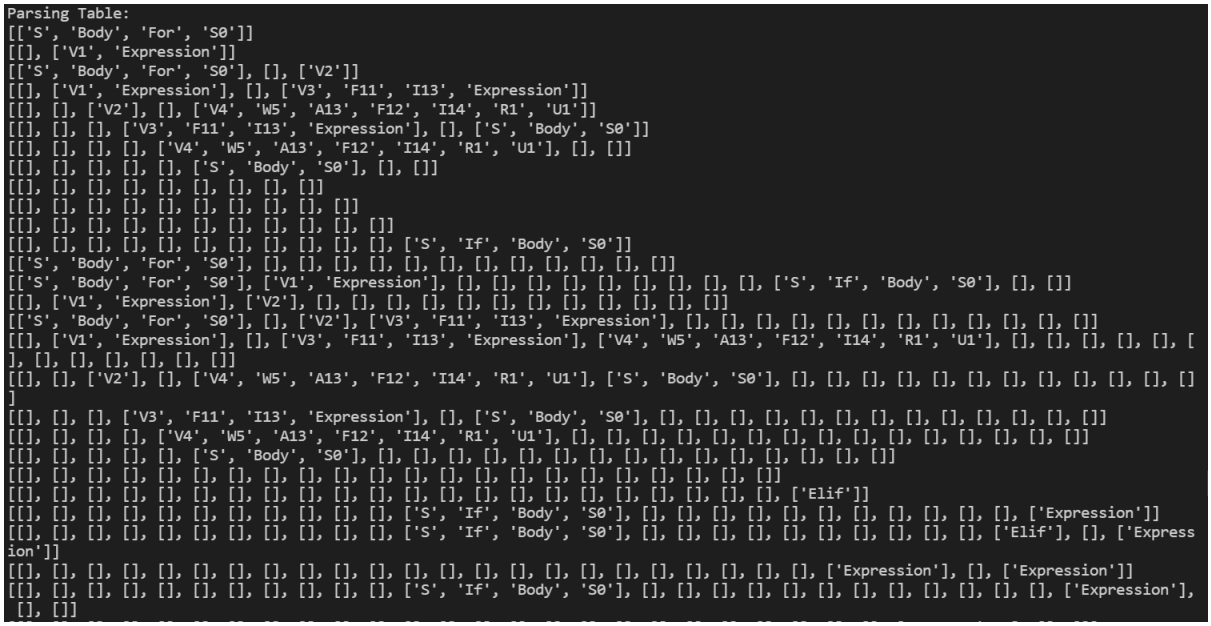
PythonParser > test > case2.py > ...

```
1   for x in 'something':
2       print("hallo", x)
3       if (yes and flag):
4           print(2)
5       elif x + 1 >= 4:
6           if not flag:
7               print("tiga")
8           else:
9               pass
10      else:
11          if True:
12              a = 2
13              x = a // 4
14              break
15      for j in range (5,0,-1):
16          y = 2 * j
17          continue
18  while (x < 100):
19      h = x % 2
```

Hasil *parsing*: **Accepted**

```
Python Parser
Type "help" for more information.
>>> py ../test/case2.py
Accepted

>>> token
Token:
['for', 'x', 'in', 'string', ':', 'print', '(', 'string', ',', 'x', ')', 'if', '(', 'yes', 'and', 'flag', ')', ':', 'print', '(', '2', ')', 'elif', 'x', '+', '1', '>', '=', '4', ':', 'if', 'not', 'flag', ':', 'print', '(', 'string', ')', 'else', ':', 'pass', 'else', ':', 'if', 'True', ':', 'a', '=', '2', 'x', '=', 'a', '/', '/', '4', 'break', 'for', 'j', 'in', 'range', '(', '5', ',', '0', ',', '-', '1', ')', ':', 'y', '=', '2', '*', 'j', 'continue', 'while', '(', 'x', '<', '100', ')', ':', 'h', '=', 'x', '%', '2']
```



dst… (karena terlalu panjang, hanya ditampilkan sebagian parse table)

Hasil *parsing* dari *source code* di atas adalah "*Accepted*" karena bagian akar dari *parse table* mengandung *start symbol* S.

C. Kasus 3 – Fungsi/Prosedur, Import, Class

Source code: **`case3.py`**

PythonParser > test > case3.py > kelas > kelas2

```
1   from cyk import cykBase as c
2
3   def func(b):
4       print(b)
5       if b:
6           result = 'a'
7       else:
8           result = 'b'
9       return result
10
11  class kelas:
12      def f1():
13          a = 1
14          b = 7 * a
15      class kelas2:
16          x = 'test'
```

Hasil *parsing*: **Accepted**

```
>>> py ../test/case3.py
Accepted

>>> token
Token:
['from', 'cyk', 'import', 'cykBase', 'as', 'c', 'def', 'func', '(', 'b', ')', ':', 'print', '(', 'b', ')', 'if', 'b', ':', 'result', '=
', 'string', 'else', ':', 'result', '=', 'string', 'return', 'result', 'class', 'kelas', ':', 'def', 'f1', '(', ')', ':', 'a', '=', '1'
, 'b', '=', '7', '*', 'a', 'class', 'kelas2', ':', 'x', '=', 'string']
```

```
Parsing Table:
[['S', 'Body', 'S0']]
[[], []]
[[], [], []]
[[], [], [], []]
[[], [], [], [], []]
[[], [], [], [], [], []]
[['S', 'Body', 'S0'], [], [], [], [], [], ['S', 'Body', 'Function', 'S0']]
[[], [], [], [], [], [], [], ['E1', 'A11', 'S1']]
[['S', 'Body', 'S0'], [], [], [], [], [], [], [], ['W4', 'A12', 'Expression']]
[[], [], [], [], [], [], [], [], [], []]
[[], [], [], [], [], [], [], [], [], [], []]
[['S', 'Body', 'S0'], [], [], [], [], [], [], [], [], [], [], ['V4', 'W5', 'A13', 'F12', 'I14', 'R1', 'U1']]
[[], [], [], [], [], [], ['S', 'Body', 'Function', 'S0'], [], [], [], [], [], ['S', 'Body', 'S0']]
[[], [], [], [], [], [], [], ['E1', 'A11', 'S1'], [], [], [], [], [], []]
[[], [], [], [], [], [], [], ['S', 'Body', 'Function', 'S0'], [], ['W4', 'A12', 'Expression'], [], [], [], [], [], []]
[[], [], [], [], [], [], [], [], ['E1', 'A11', 'S1'], [], [], [], [], [], [], [], []]
[[], [], [], [], [], [], [], [], [], ['W4', 'A12', 'Expression'], [], [], [], [], [], [], [], ['S', 'If', 'Body', 'S0']]
[[], [], [], [], [], [], [], ['S', 'Body', 'Function', 'S0'], [], [], [], [], ['V4', 'W5', 'A13', 'F12', 'I14', 'R1', 'U1'], [], [], [], []
, [], []]
[[], [], [], [], [], [], [], [], ['E1', 'A11', 'S1'], [], [], [], [], ['S', 'Body', 'S0'], [], [], [], [], [], []]
[[], [], [], [], [], [], [], [], [], ['W4', 'A12', 'Expression'], [], [], ['V4', 'W5', 'A13', 'F12', 'I14', 'R1', 'U1'], [], [], [], [], []
, [], [], []]
[[], [], [], [], [], [], [], [], [], [], [], [], [], ['S', 'Body', 'S0'], [], [], [], [], [], [], [], []]
[[], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], []]
[['S', 'Body', 'S0', 'C14'], [], [], [], [], [], [], [], [], [], [], [], ['V4', 'W5', 'A13', 'F12', 'I14', 'R1', 'U1'], [], [], [], [], ['S
', 'If', 'Body', 'S0'], [], [], [], [], [], ['Else']]
[[], [], [], [], [], [], [], [], [], [], [], [], [], ['S', 'Body', 'S0'], [], [], [], [], [], [], [], [], [], [], ['R1', 'U1']]
[['S', 'Body', 'S0'], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], ['S', 'If', 'Body', 'S0'], [], [], [], [], [], [], []
, ['Body']]
```

dst… (karena terlalu panjang, hanya ditampilkan sebagian parse table)

```
[['From'], [], ['D11'], [], ['E13'], [], [], [], [], ['E1', 'S1'], [], [], [], [], ['E1', 'S1'], [], [], [], [], [], ['C1'], [], [], []
, [], ['C1'], [], ['Body', 'Return'], [], [], [], [], [], [], [], ['Bracket', 'Term', 'Arithmetic', 'Boolean', 'Condition', 'Element'], [],
 [], [], ['C1'], [], [], ['C1'], [], ['Term', 'Arithmetic', 'G1', 'H1', 'Boolean', 'Condition', 'Element'], [], [], [], [], [], ['C1']]
[['V3'], ['Term', 'Term', 'Arithmetic', 'Operand', 'Boolean', 'Boolean', 'Condition', 'Condition', 'Element', 'Element'], ['X3'], ['Ter
m', 'Term', 'Arithmetic', 'Operand', 'Boolean', 'Boolean', 'Condition', 'Condition', 'Element', 'Element'], ['W3'], ['Term', 'Term', 'A
rithmetic', 'Operand', 'Boolean', 'Boolean', 'Condition', 'Condition', 'Element', 'Element'], ['Z3'], ['Term', 'Term', 'Arithmetic', 'O
perand', 'Boolean', 'Boolean', 'Condition', 'Condition', 'Element', 'Element'], ['X4'], ['Term', 'Term', 'Arithmetic', 'Operand', 'Bool
ean', 'Boolean', 'Condition', 'Condition', 'Element', 'Element'], ['W4'], ['T4'], ['K4'], ['X4'], ['Term', 'Term', 'Arithmetic', 'Opera
nd', 'Boolean', 'Boolean', 'Condition', 'Condition', 'Element', 'Element'], ['W4'], ['J4'], ['Term', 'Term', 'Arithmetic', 'Operand', '
Boolean', 'Boolean', 'Condition', 'Condition', 'Element', 'Element'], ['T4'], ['Term', 'Term', 'Arithmetic', 'Operand', 'Boolean', 'Boo
lean', 'Condition', 'Condition', 'Element', 'Element'], ['Z4'], ['Term', 'Term', 'Arithmetic', 'Operand', 'String', 'Text', 'Boolean',
'Boolean', 'Condition', 'Condition', 'Element', 'Element'], ['F4'], ['T4'], ['Term', 'Term', 'Arithmetic', 'Operand', 'Boolean', 'Boole
an', 'Condition', 'Condition', 'Element', 'Element'], ['Z4'], ['Term', 'Term', 'Arithmetic', 'Operand', 'String', 'Text', 'Boolean', 'B
oolean', 'Condition', 'Condition', 'Element', 'Element'], ['Y3'], ['Term', 'Term', 'Arithmetic', 'Operand', 'Boolean', 'Boolean', 'Cond
ition', 'Condition', 'Element', 'Element'], ['U3'], ['Term', 'Term', 'Arithmetic', 'Operand', 'Boolean', 'Boolean', 'Condition', 'Condi
tion', 'Element', 'Element'], ['T4'], ['Z3'], ['Term', 'Term', 'Arithmetic', 'Operand', 'Boolean', 'Boolean', 'Condition', 'Condition',
 'Element', 'Element'], ['X4'], ['W4'], ['T4'], ['Term', 'Term', 'Arithmetic', 'Operand', 'Boolean', 'Boolean', 'Condition', 'Condition
', 'Element', 'Element'], ['Z4'], ['Term', 'Arithmetic', 'Operand', 'Boolean', 'Condition', 'Element'], ['Term', 'Term', 'Arithmetic',
'Operand', 'Boolean', 'Boolean', 'Condition', 'Condition', 'Element', 'Element'], ['Z4'], ['Term', 'Arithmetic', 'Operand', 'Boolean',
'Condition', 'Element'], ['Operator'], ['Term', 'Term', 'Arithmetic', 'Operand', 'Boolean', 'Boolean', 'Condition', 'Condition', 'Eleme
nt', 'Element'], ['U3'], ['Term', 'Term', 'Arithmetic', 'Operand', 'Boolean', 'Boolean', 'Condition', 'Condition', 'Element', 'Element'
], ['T4'], ['Term', 'Term', 'Arithmetic', 'Operand', 'Boolean', 'Boolean', 'Condition', 'Condition', 'Element', 'Element'], ['Z4'], ['T
erm', 'Term', 'Arithmetic', 'Operand', 'String', 'Text', 'Boolean', 'Boolean', 'Condition', 'Condition', 'Element', 'Element']]

>>>
```

Hasil *parsing* dari *source code* di atas adalah "*Accepted*" karena bagian akar dari *parse table* mengandung *start symbol* S.

D. Case 4 – Rejected (Variable)

Source code: **`case4.py`**

PythonParser > test > case4.py > ...

```
1  a = 2
2  9a = a % 9
3  _var9 = 9a + 2
4  print(_var9)
```

Hasil *parsing*: **Syntax Error**

```
>>> py ../test/case4.py
  9a = a % 9
  ^ invalid variable in line 2
Syntax Error

>>> token
Token:
['a', '=', '2', '9a', '=', 'a', '%', '9', '_var9', '=', '9a', '+', '2', 'print', '(', '_var9', ')']
Parsing Table:
[[]]
[[], []]
[[], [], []]
[[], [], [], []]
[[], [], [], [], []]
[[], [], [], [], [], []]
[[], [], [], [], [], [], []]
[[], [], [], [], [], [], [], []]
[[], [], [], [], [], [], [], [], []]
[[], [], [], [], [], [], [], [], [], []]
[[], [], [], [], [], [], [], [], [], [], []]
[[], [], [], [], [], [], [], [], [], [], [], []]
[[], [], [], [], ['C1'], [], [], [], [], [], [], [], ['S', 'Print', 'Body', 'S0']]
[['S', 'Assignment', 'Body', 'S0'], [], [], [], [], ['Term', 'Arithmetic', 'Boolean', 'Condition', 'Element'], [], [], [], [],
[], [], [], [], ['Bracket', 'Term', 'Arithmetic', 'Boolean', 'Condition', 'Element']]
[[], ['C1'], [], [], ['C1'], [], ['Term', 'Arithmetic', 'G1', 'H1', 'Boolean', 'Condition', 'Element'], [], [], [], [], ['Term'
, 'Arithmetic', 'G1', 'H1', 'Boolean', 'Condition', 'Element'], [], [], [], ['E1', 'S1']]
[['Term', 'Term', 'Arithmetic', 'Operand', 'Boolean', 'Boolean', 'Condition', 'Condition', 'Element', 'Element'], ['Z4'], ['Ter
m', 'Arithmetic', 'Operand', 'Boolean', 'Condition', 'Element'], ['H4'], ['Z4'], ['Term', 'Term', 'Arithmetic', 'Operand', 'Boo
lean', 'Boolean', 'Condition', 'Condition', 'Element', 'Element'], ['Operator'], ['Term', 'Arithmetic', 'Operand', 'Boolean', '
Condition', 'Element'], ['Term', 'Term', 'Arithmetic', 'Operand', 'Boolean', 'Boolean', 'Condition', 'Condition', 'Element', 'E
lement'], ['Z4'], ['H4'], ['Operator'], ['Term', 'Arithmetic', 'Operand', 'Boolean', 'Condition', 'Element'], ['K4'], ['X4'], [
'Term', 'Term', 'Arithmetic', 'Operand', 'Boolean', 'Boolean', 'Condition', 'Condition', 'Element', 'Element'], ['W4']]

>>>
```

Hasil *parsing* dari *source code* di atas adalah "*Syntax Error*" karena saat dicek dengan FA, variabel *invalid* dan bagian akar dari *parse table* tidak mengandung *start symbol* S. Untuk *error* yang diakibatkan penamaan variabel, akan ditampilkan lokasi kesalahannya.

E. Case 5 – Rejected

Source code: **`case5.py`**

PythonParser > test > case5.py > func1

```
1  def func1():
2      flag = True
3      if not flag:
4          print('flag')
5          else:
6              flag = True
7      else:
8          print(flag)
```

Hasil parsing: **Syntax Error**

```
Python Parser
Type "help" for more information.
>>> py ../test/case5.py
Syntax Error

>>> token
Token:
['def', 'func1', '(', ')', ':', 'flag', '=', 'True', 'if', 'not', 'flag', ':', 'print', '(', 'string', ')', 'else', ':', 'flag', '=', '
True', 'else', ':', 'print', '(', 'flag', ')']
```

```
Parsing Table:
[[]]
[[], []]
[[], [], []]
[[], [], [], []]
[[], [], [], [], []]
[[], [], [], [], [], []]
[[], [], [], [], [], [], []]
[[], [], [], [], [], [], [], []]
[[], [], [], [], [], [], [], [], []]
[[], [], [], [], [], [], [], [], [], []]
[[], [], [], [], [], [], [], [], [], [], []]
[[], [], [], [], [], [], [], [], [], [], [], []]
[[], [], [], [], [], [], [], [], [], [], [], [], []]
[[], [], [], [], [], [], [], [], [], [], [], [], [], []]
[[], [], [], [], [], [], [], [], [], [], [], [], [], [], []]
[[], [], [], [], [], [], [], [], [], [], [], [], [], [], [], []]
[[], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], []]
[[], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], []]
[[], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], [], []]
[[], [], [], [], [], [], [], [], ['S', 'If', 'Body', 'S0'], [], [], [], [], [], [], [], [], [], [], []]
[[], [], [], [], [], [], [], [], [], ['Expression'], [], [], [], [], [], [], [], [], [], [], []]
[[], [], [], [], [], [], [], [], [], [], ['V3', 'F11', 'I13', 'Expression'], [], [], [], [], [], [], [], [], [], [], ['Else']]
[[], [], [], [], [], [], [], [], [], [], [], ['V4', 'W5', 'A13', 'F12', 'I14', 'R1', 'U1'], [], [], [], [], [], [], [], [], [], [], ['V
4', 'W5', 'A13', 'F12', 'I14', 'R1', 'U1']]
[[], [], [], [], [], [], [], [], [], [], [], [], ['S', 'Print', 'Body', 'S0'], [], [], [], [], [], [], [], [], [], [], ['S', 'Print', '
Body', 'S0']]
[[], [], [], [], [], [], [], [], [], [], [], [], [], ['Bracket', 'Term', 'Arithmetic', 'Boolean', 'Condition', 'Element'], [], [], [],
[], [], [], [], [], [], [], ['Bracket', 'Term', 'Arithmetic', 'Boolean', 'Condition', 'Element']]
[[], [], ['Bracket', 'Term', 'Arithmetic', 'Boolean', 'Condition', 'Element'], [], [], [], [], [], [], ['Boolean', 'Condition'], [], []
, [], [], ['E1', 'S1'], [], [], [], [], [], [], [], [], [], [], ['E1', 'S1']]
[['Z3'], ['Term', 'Term', 'Arithmetic', 'Operand', 'Boolean', 'Boolean', 'Condition', 'Condition', 'Element', 'Element'], ['X4'], ['W4'
], ['T4'], ['Term', 'Term', 'Arithmetic', 'Operand', 'Boolean', 'Boolean', 'Condition', 'Condition', 'Element', 'Element'], ['Z4'], ['B
oolean', 'Condition'], ['J4'], ['Q4'], ['Term', 'Term', 'Arithmetic', 'Operand', 'Boolean', 'Boolean', 'Condition', 'Condition', 'Eleme
nt', 'Element'], ['T4'], ['K4'], ['X4'], ['Term', 'Term', 'Arithmetic', 'Operand', 'String', 'Text', 'Boolean', 'Boolean', 'Condition',
 'Condition', 'Element', 'Element'], ['W4'], ['F4'], ['T4'], ['Term', 'Term', 'Arithmetic', 'Operand', 'Boolean', 'Boolean', 'Condition
', 'Condition', 'Element', 'Element'], ['Z4'], ['Boolean', 'Condition'], ['F4'], ['T4'], ['K4'], ['X4'], ['Term', 'Term', 'Arithmetic',
 'Operand', 'Boolean', 'Boolean', 'Condition', 'Condition', 'Element', 'Element'], ['W4']]

>>>
```

Hasil *parsing* dari *source code* di atas adalah "*Syntax Error*" karena bagian akar dari *parse table* tidak mengandung *start symbol* S. Ini terjadi karena terdapat *statement el*se tanpa didahului s*tatement if* (*line* 7).

BAB IV
PEMBAGIAN TUGAS DAN *LINK REPOSITORY*

1. *LINK REPOSITORY* GITHUB

Link Repository: https://github.com/tastytypist/PythonParser

2. PEMBAGIAN TUGAS

NIM	Nama	Tugas
13520035	Damianus Clairvoyance Diva P.	Perancangan dan implementasi *Finite Automata*. Laporan Bab 1.1, 1.2, 1.3, 2.1, 3.1.D, 5.2
13520042	Jeremy S.O.N. Simbolon	Perancangan CFG dan konversinya ke CNF beserta implementasinya, Laporan Bab 2.2, 2.3, 3.1.A, 3.1.B.
13520046	Hansel Valentino Tanoto	Perancangan program utama, tokenisasi masukan, dan algoritma CYK beserta implementasinya, Laporan Bab 1.4, 1.5, 3.1.C, 3.1.E, 3.1.F, 3.2, 5.1.

BAB V
PENUTUP

1. KESIMPULAN

Tugas Besar dengan judul “Compiler Bahasa Python” telah kelompok kami selesaikan dengan baik. Program kami telah dapat melakukan evaluasi *syntax* sesuai ketentuan pada spesifikasi Tugas Besar. Program kami dapat menangani kasus-kasus umum yang diberikan dengan baik meskipun masih banyak kasus yang belum dapat ditangani program kami karena CFG yang kami gunakan masih sederhana dan pastinya belum mencakup keseluruhan *syntax* dalam bahasa Python. Program kami juga sudah dapat menunjukkan lokasi kesalahan khusus untuk penaman variabel yang tidak valid sedangkan lokasi untuk jenis kesalahan lainnya belum dapat ditunjukkan.

2. SARAN

Dalam pengerjaan tugas besar ini, terdapat beberapa kekurangan yang sebaiknya tidak dilakukan oleh pembaca yang ingin mereplikasi pembuatan program ini.

1. Menyiapkan *test case* dengan berbagai macam ekspresi terlebih dulu untuk memastikan kebenaran program.
2. Memantapkan pemahaman terhadap CFG, CNF, CYK, dan FA sebelum memulai pengerjaan program untuk mempermudah.

Selain itu, terdapat pula beberapa saran pengembangan untuk program kelompok kami, yakni sebagai berikut.

1. Menambah dan mengembangkan CFG sehingga mencakup *test case* yang lebih luas.
2. Mengembangkan fitur untuk mengidentifikasi lokasi kesalahan pada kode program dan jenis kesalahannya.

REFERENSI

Hopcroft, J. E., Motwani, R., Ullman, J. D. (2006). *Introduction to Automata Theory, Language, and Computation* (3rd ed). Pearson Education, Inc.

List of Keywords in Python. (2021, November 15). Programiz. https://www.programiz.com/python-programming/keyword-list.

Neso Academy. (2021, November 10). *Theory of Computation & Automata Theory*. *Playlists* [YouTube channel]. YouTube. https://www.youtube.com/playlist?list=PLBlnK6fEyqRgp46KUv4ZY69yXmpwKOIev.

Python Software Foundation. (2021, November 16). *Python 3.9.7 documentation*. https://docs.python.org/release/3.9.7/.

Ramadan, A. M. (2021, November 16). *CFG2CNF*. https://github.com/adelmassimo/CFG2CNF.

Stanford University. (n.d.). *Basics of Automata Theory*. https://cs.stanford.edu/people/eroberts/courses/soco/projects/2004-05/automata-theory/basics.html.