

**LAPORAN TUGAS BESAR**  
**IF2124 TEORI BAHASA FORMAL DAN OTOMATA**  
***COMPILER BAHASA PYTHON***



DISUSUN OLEH:

**KELOMPOK 4 - K2**

**PUTRI NURHALIZA** **13520066**

**JOVA ANDRES RISKI SIRAIT** **13520072**

**ADELLINE KANIA SETIYAWAN** **13520084**

**TEKNIK INFORMATIKA**  
**INSTITUT TEKNOLOGI BANDUNG**

**2021**

## **DAFTAR ISI**

DAFTAR ISI.....	i
BAB I TEORI DASAR.....	1
A. Context-Free Grammar (CFG).....	1
B. Chomsky Normal Form (CNF) .....	1
C. Cocke-Younger Kasami (CYK).....	3
D. Bahasa Pemograman Python.....	4
BAB II HASIL CFG .....	5
BAB III IMPLEMENTASI DAN PENGUJIAN .....	8
A. SPESIFIKASI TEKNIS PROGRAM .....	8
B. PENGUJIAN.....	14
LAMPIRAN.....	24
A. Repository Github .....	24
B. Pembagian Tugas .....	24
DAFTAR PUSTAKA .....	25

## BAB I

### TEORI DASAR

#### A. Context-Free Grammar (CFG)

Dalam teori bahasa formal, Context—Free Grammar (bahasa bebas konteks) adalah tata bahasa formal yang aturan produksinya berbentuk  $A \rightarrow \alpha$  dengan  $A$  adalah simbol nonterminal dan  $\alpha$  adalah simbol terminal (bisa berbentuk string kosong atau disimbolkan dengan  $\epsilon$ ). Pada dasarnya, CFG memiliki tujuan yang sama dengan tata bahasa regular, yaitu untuk membentuk suatu aturan produksi yang dapat menggambarkan semua kemungkinan *string* dalam bahasa formal tertentu.

Secara formal, Context—Free Grammar didefinisikan dengan 4 *tuple*, yaitu

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

dengan  $V$  adalah himpunan terbatas variabel,  $T$  adalah himpunan terbatas terminal,  $P$  adalah himpunan terbatas dari produksi, dan  $S$  adalah simbol mulai.

Aturan produksi dari CFG tidak terdapat batasan terhadap hasil produksinya. Penulisan produksi dari CFG adalah pada ruas kiri hanya mengandung satu simbol nonterminal dan tidak ada batasan pada ruas sebelah kanan (hasil produksi). Contoh dari aturan produksi CFG adalah sebagai berikut:

$$S \rightarrow AB$$

$$S \rightarrow aabB$$

#### B. Chomsky Normal Form (CNF)

Chomsky Normal Form adalah salah satu bentuk dari Context—Free Grammar yang aturan produksinya lebih terbatas dari Context—Free Grammar. Bentuk dari Chomsky Normal Form merupakan penyederhanaan dari Context—Free Grammar yang memudahkan untuk memeriksa apakah suatu *string* merupakan bagian dari bahasa formal tertentu. Pada Chomsky Normal Form, hasil produksinya hanya antara 2 variabel dan satu terminal. Aturan produksinya adalah:

$$A \rightarrow a$$

atau

$$A \rightarrow BC$$

dengan  $A, B, C$  adalah variabel/nonterminal dan  $a$  adalah terminal. Chomsky Normal Form harus memenuhi syarat tata bahasa bebas konteks, yaitu tidak memiliki produksi *string* kosong atau  $\epsilon$ , tidak memiliki produksi *useless* dan tidak memiliki produksi unit.

Untuk mengubah suatu Context—Free Grammar menjadi bentuk Chomsky Normal—Form adalah sebagai berikut:

1. Apabila *start symbol*,  $S$ , terdapat pada beberapa bagian ruas kanan suatu produksi, buatlah *start symbol* baru,  $S'$ , dan produksi baru;
2. Hapus semua *null productions* dan *unit productions*;
3. Ganti setiap produksi  $A \rightarrow B_1 B_2 \dots B_n$ , untuk  $n > 2$ , dengan  $A \rightarrow B_1 C$ , dimana  $C \rightarrow B_2 \dots B_n$ . Ulangi hingga semua produksi hanya mengandung dua nonterminal/variabel pada ruas kanannya;
4. Apabila hasil produksinya berbentuk  $A \rightarrow aB$ , ganti produksi tersebut dengan  $A \rightarrow XB$  dan  $X \rightarrow a$ . Ulangi hingga semua produksi hanya dalam bentuk  $A \rightarrow a$  atau  $A \rightarrow BC$ .

**Contoh:**

Suatu CFG dengan aturan produksi:

$$P: S \rightarrow ASA \mid aB, A \rightarrow B \mid S, B \rightarrow b \mid \epsilon$$

diubah menjadi bentuk CNF-nya menjadi

$$P: S' \rightarrow AX \mid YB \mid a \mid AS \mid SA,$$

$$S \rightarrow AX \mid YB \mid a \mid AS \mid SA,$$

$$A \rightarrow b \mid AX \mid YB \mid a \mid AS \mid SA,$$

$$B \rightarrow b,$$

$$X \rightarrow SA,$$

$$Y \rightarrow a$$

### C. Cocke-Younger Kasami (CYK)

Cocke-Younger Kasami (CYK) adalah suatu algoritma *parsing* untuk menguji apakah suatu *string* merupakan bagian dari suatu bahasa formal tertentu untuk tata bahasa bebas konteks. Untuk menggunakan algoritma CYK ini, suatu tata bahasa harus sudah dalam bentuk Chomsky-Normal Form.

Algoritma Cocke—Younger Kasami memanfaatkan struktur data berupa tabel *array* dua dimensi dengan ukuran  $n \times n - 1 \times \dots \times 1$ , dimana  $n$  adalah jumlah kata dari suatu *string*.

x14			
x13	x23		
x12	x22	x32	
x11	x21	x31	x41

Algoritma CYK memanfaatkan hasil *parsing* dari *array* sebelumnya. Untuk menentukan apakah suatu *string* merupakan bagian dari suatu bahasa formal, akan dicek apakah pada baris paling atas, pada tabel misalnya adalah x14, terdapat *start symbol*. Apabila terdapat *start symbol*, maka *string* tersebut adalah bagian dari bahasa yang dicek, begitu juga sebaliknya.

#### Contoh:

Akan dicek apakah suatu *string* “baaba” diterima pada suatu CFG yang didefinisikan sebagai berikut:

$$G = (\{S, A, B, C\}, \{a, b\}, P, S)$$

dan memiliki aturan produksi

$$P : S \rightarrow AB \mid BC,$$

$$A \rightarrow BA \mid a,$$

$$B \rightarrow CC \mid b,$$

$$C \rightarrow AB \mid a$$

Berdasarkan algoritma CYK, akan terbentuk tabel:

{S, A, C}				
{ $\phi$ }	{S, A, C}			
{ $\phi$ }	{B}	{B}		
{S, A}	{B}	{S, C}	{S, A}	
{B}	{A, C}	{A, C}	{B}	{A, C}

Karena *start symbol*, *S*, terdapat pada baris *cell* paling atas, maka *string* “baaba” diterima oleh CFG tersebut.

#### D. Bahasa Pemrograman Python

Python adalah salah satu Bahasa pemrograman tingkat tinggi yang sangat populer. Python memiliki sintaks yang sederhana dengan tingkat keterbacaan yang tinggi sehingga menunjang kinerja *developers*. Setiap instuksi di python menggunakan baris baru dan sangat bergantung pada indentasi untuk *code block*, tidak seperti kebanyakan Bahasa pemrograman yang menggunakan kurung kurawal. Program yang dibuat dengan Python dapat langsung dieksekusi karena python merupakan Bahasa yang interpretatif. Python dapat diperlakukan dengan berbagai paradigma pemrograman. Misal, pemrograman fungsional, pemrograman prosedural, pemrograman berorientasi objek, dan sebagainya.

Python memiliki *library* dan *framework* yang sangat luas dan komprehensif untuk berbagai keperluan. Tak hanya itu, python juga bersifat modular dan fleksibel sehingga mudah untuk dikembangkan bahkan diintegrasikan dengan Bahasa pemrograman lain. Beberapa pemanfaatan python saat ini adalah untuk pengembangan web (*server-side*), pengembangan perangkat lunak, skrip sistem, menangani *big data* yang mencakup perhitungan matematik yang kompleks, serta untuk *rapid prototyping*. Versi terbaru Python saat ini adalah 3.10.0 dan dapat bekerja pada berbagai sistem operasi seperti Windows, Mac, Linux, Raspberry Pi, dan lain lain.

## BAB II

### HASIL CFG

Implementasi CFG bertujuan untuk menghasilkan *production* yang kemudian disederhanakan menjadi Chomsky-Normal Form(CNF) agar dapat dijalankan pada algoritma CYK untuk mengevaluasi validitas sintaks kode dalam bahasa pemrograman python.

#### Variable/Non-Terminal Symbol

S	SLOOP	VAR	VAL	STRING	NUMBER	ARRAY	VARRAY	INDEX	OPARITH
OP	PRINT	VPRINT	FORHEAD	VRANGE	FUNCTION	VFUNCTION	FOR	IMPORT	VIMPORT
WITH	VWITH	OPCOMP	BOOLEAN	NONE	OPLOGIC	OPIDENTITY	WHILE	COMP	IF
VIF	IFLOOP	VIFL	ELIF	VELIF	VELIF	ELIFLOOP	ELSE	VELSE	VELSEL
ELSELOOP	IFRET	DEF	DEFV	VDEF	VRET	CLASS	ICLASS	VCLASS	BREAK
PASS	CONTINUE	RAISE	EXCP	COMPOPRT	INPUT				

#### Terminal Symbol

variable	if	print	continue	as	is	in	not
string	elif	def	break	=	(	)	'
number	else	class	pass	+	<	>	,
True	for	return	with	*	&	%	:
False	while	from	open	/	^	!	"
None	range	import	input	raise	.	-	

#### Productions

Rules	Result
<b>S</b>	S S   VAR = VAL   VAR += STRING   VAR OPARITH = NUMBER   INDEX   PRINT   FOR   IMPORT   WITH   WHILE   IF   DEF   CLASS   FUNCTION   PASS   RAISE
<b>SLOOP</b>	SLOOP SLOOP   VAR = VAL   VAR += STRING   VAR OPARITH = NUMBER   INDEX   PRINT   FOR   WITH   WHILE   IFLOOP   FUNCTION   PASS   RAISE   BREAK   CONTINUE
<b>VAR</b>	variable   VCLASS
<b>VAL</b>	VAR   STRING   NUMBER   ARRAY   FUNCTION   BOOLEAN   NONE   VCLASS   INPUT   INDEX
<b>STRING</b>	STRING + STRING   ' string '   " string "   ' string ' + ' string '   " string " + " string "   " string " + ' string '   ' string ' + " string "   ( STRING )
<b>NUMBER</b>	NUMBER OPARITH NUMBER   number   number OPARITH number   number * number   number // number   number > number   number < number   ( NUMBER )   number . number   . number   + number   - number
<b>VARRAY</b>	VAL   VARRAY , VARRAY   ARRAY

ARRAY	[ ]   [ VARRAY ]   ( )   ( VARRAY )
INDEX	VAR [ NUMBER ]   VAR [ : ]   VAR [ : NUMBER ]   VAR [ NUMBER : ]   VAR [ NUMBER : NUMBER ]
OPARITH	+   -   *   /   %
OPCOMP	<   >   !
OP	OPARITH   &   ^   OPCOMP
VPRINT	VAL   VPRINT , VPRINT
PRINT	print ( VPRINT )   print ( )
INPUT	input ( VAL )   ( INPUT )   input ( )
FORHEAD	for VAR in range ( VRANGE )   for VAR in VAL   for VAR , VAR in VAL
VRANGE	VAL , VAL   VAL   VAL , VAL , VAL
FUNCTION	VAR ( VFUNCTION )   FUNCTION ( FUNCTION )   VAR ( )
VFUNCTION	VFUNCTION , VFUNCTION   VAL   VAR = VAL;
FOR	FORHEAD : SLOOP
BOOLEAN	True   False
NONE	None
OPLOGIC	and   or
OPIDENTITY	is   is not   in
IMPORT	from VAR import VIMPORT   from VAR import *   import VIMPORT   import VAR as VAR   from VAR import VAR as VAR
VIMPORT	VAR   VIMPORT , VIMPORT
VWITH	VAL   VAL , VAL   VAL , VAL , VAL   VAL , VAL , VAL , VAL   VAL , VAL , VAL , VAL , VAL   VAL   VAL , VAL , VAL , VAL , VAL , VAL   VAL , VAL , VAL , VAL , VAL , VAL , VAL   VAL , VAL , VAL , VAL , VAL , VAL , VAL , VAL
WITH	with open ( VWITH ) : S   with open ( VWITH ) as VAR : S
WHILE	while COMP : SLOOP
COMP	BOOLEAN   VAR   ( COMP )   COMP OPLOGIC COMP   COMPOPRT OPCOMP COMPOPRT   COMPOPRT OPCOMP = COMPOPRT   COMPOPRT = = COMPOPRT   not VAR   not COMP   COMPOPRT OPIDENTITY COMPOPRT
COMPOPRT	COMPOPRT OPARITH COMPOPRT   COMPOPRT OPCOMP COMPOPRT   COMPOPRT OPCOMP = COMPOPRT   VAL   VAL OPARITH VAL   VAL * * VAL   VAL // VAL   VAL > > VAL   VAL < < VAL   VAL OPCOMP VAL   VAL OPCOMP = VAL   ( COMPOPRT )
RAISE	raise VAR ( STRING )   raise ( STRING )   raise ( )
BREAK	break
PASS	pass
CONTINUE	continue
VIF	if COMP : S   if COMP : IFRET
VELIF	elif COMP : S   elif COMP : IFRET
VELSE	else : S   else : IFRET
IF	VIF   VIF ELIF   VIF ELSE   VIF RAISE   VIF RAISE ELIF   VIF RAISE ELSE   VIF PASS   VIF PASS ELIF   VIF PASS ELSE
ELIF	VELIF   VELIF ELIF   VELIF ELSE   VELIF RAISE   VELIF RAISE ELIF   VELIF RAISE ELSE   VELIF PASS   VELIF PASS ELIF   VELIF PASS ELSE
ELSE	VELSE   VELSE RAISE   VELSE PASS
VIFL	if COMP : SLOOP   if COMP : IFRET
VELIFL	elif COMP : SLOOP   elif COMP : IFRET
VELSEL	else : SLOOP   else : IFRET
IFLOOP	VIFL   VIFL ELIFLOOP   VIFL ELSELOOP   VIFL RAISE   VIFL RAISE ELIFLOOP   VIFL RAISE ELSELOOP   VIFL PASS   VIFL PASS ELIFLOOP   VIFL PASS ELSELOOP   VIFL



	CONTINUE   VIFL CONTINUE ELIFLOOP   VIFL CONTINUE ELSELOOP   VIFL BREAK   VIFL BREAK ELIFLOOP   VIFL BREAK ELSELOOP
<b>ELIFLOOP</b>	VELIFL   VELIFL ELIFLOOP   VELIFL ELSELOOP   VELIFL RAISE   VELIFL RAISE ELIFLOOP   VELIFL RAISE ELSELOOP   VELIFL PASS   VELIFL PASS ELIFLOOP   VELIFL PASS ELSELOOP   VELIFL CONTINUE   VELIFL CONTINUE ELIFLOOP   VELIFL CONTINUE ELSELOOP   VELIFL BREAK   VELIFL BREAK ELIFLOOP   VELIFL BREAK ELSELOOP
<b>ELSELOOP</b>	VELSEL   VELSEL RAISE   VELSEL PASS   VELSEL CONTINUE   VELSEL BREAK   else : BREAK
<b>IFRET</b>	S return VRET   return VRET   S return   return
<b>DEF</b>	DEFV : S return VRET   DEFV : S return   DEFV : S
<b>DEFV</b>	def VAR ( VDEF )   def VAR ( )
<b>VDEF</b>	VDEF , VDEF   variable   VAR = VAL
<b>VRET</b>	VRET , VRET   variable   COMP   ( VRET )   NUMBER   BOOLEAN   STRING   NONE
<b>ICLASS</b>	VAR   ICLASS , ICLASS   VAR = VAR;
<b>CLASS</b>	class VAR : S   class VAR ( ICLASS ) : S
<b>VCLASS</b>	VCLASS . VCLASS   VCLASS . FUNCTION   variable   FUNCTION

## BAB III

### IMPLEMENTASI DAN PENGUJIAN

#### A. SPESIFIKASI TEKNIS PROGRAM

##### 1. Evaluasi Nama Variabel dengan Finite Automata

Pada bahasa pemrograman python, terdapat aturan untuk nama variable yang digunakan, beberapa aturan tersebut adalah sebagai berikut:

- a. Harus dimulai dengan huruf *alphabet* atau karakter *underscore*.
- b. Tidak boleh dimulai dengan angka.
- c. Hanya boleh mengandung karakter *alpha-numeric* dan *underscore* (A-z, 0-9, dan \_).
- d. Nama variable bersifat *case-sensitive*.
- e. Tidak boleh nama yang merupakan *keywords* pada python.

Untuk mengevaluasi setiap nama variable yang ada, pada program ditambahkan class FA, dan juga model *Deterministic Finite Automata*. Class FA menerima *constructor* berupa variable yang akan dievaluasi, dan mempunyai dua variable lainnya yaitu *currentState*, untuk menyimpan state setiap pengecekan karakter, dan juga *accept*, yang menyimpan informasi state yang diterima oleh *Finite Automata*. Berikut adalah *method* yang ada pada class FA:

No	Method	Tujuan
1	readSymbol	Menelusuri tiap karakter pada string yang diberikan, dan mengubah state berdasarkan <i>currentState</i> dan <i>input</i> . Mirip dengan $\delta$ .

Model DFA yang digunakan terdiri dari 3 state, yaitu *start*, *dead*, dan *final*. Model DFA tersebut diimplementasikan dengan menggunakan dictionary dengan aturan sebagai berikut:

upper = [A-Z] ; lower = [a-z] ; number [0-9] ; underscore = [ \_ ] ;

anotherAscii = [{printable ascii character(32-127) kecuali *alphanum*, *underscore*, *whitespaces*, dan tidak termasuk *extended ascii*}]

No	Fungsi Transisi	State Akhir
1	$\delta(\text{start}, [\text{upper}, \text{lower}, \text{underscore}])$	final
2	$\delta(\text{start}, [\text{number}, \text{anotherAscii}])$	dead
3	$\delta(\text{dead}, [\text{upper}, \text{lower}, \text{underscore}, \text{number}, \text{anotherAscii}])$	dead
4	$\delta(\text{final}, [\text{upper}, \text{lower}, \text{underscore}, \text{number}])$	final
5	$\delta(\text{final}, [\text{anotherAscii}])$	dead

## 2. Konversi Context Free Grammar ke Chomsky Normal Form

Karena algoritma *cocke younger kasami* membutuhkan grammar berbentuk *Chomsky Normal Form* untuk bisa berjalan, maka *grammar* yang telah dibuat harus dikonversi terlebih dahulu ke CNF. Untuk konversi ini, kami menggunakan algoritma CFG to CNF yang diambil dari <https://github.com/adelmassimo/CFG2CNF>. Algoritma ini ditulis dalam Bahasa pemrograman python dan terdiri dari dua file, yaitu CFGtoCNF.py dan helper.py. Program ini membutuhkan masukan berupa file txt dari grammar CFG yang telah dibuat, dan menghasilkan keluaran berupa file txt yang berisi hasil konversinya ke bentuk CNF.

Berikut adalah penjelasan setiap method pada program konversi grammar:

### CFGtoCNF.py

No	Fungsi	Tujuan
1	isUnitary(rule, variables) $\rightarrow$ bool	Fungsi untuk mengecek apakah rules memiliki production berupa satu variable.
2	isSimple(rule) $\rightarrow$ bool	Fungsi untuk mengecek apakah rules menghasilkan satu produksi terminal.
3	START(productions, variables) $\rightarrow$ list	Fungsi untuk menambahkan simbol S0 sebagai start symbol kedalam rules.
4	TERM(productions, variables) $\rightarrow$ list	Fungsi untuk menghapus produksi simbol terminal dan variable yang bergabung, menjadi satu terminal dan dua variable.

5	BIN(productions, variables) → list	Fungsi untuk mengeliminasi aturan produksi yang berupa lebih dari dua variable.
6	DEL(productions) → set	Fungsi untuk menghapus aturan produksi yang bukan merupakan simbol terminal.
7	unit_routine(rules, variables) → list	Fungsi untuk mengecek apakah suatu aturan produksi merupakan unitary, dan mengecek apakah aturan tersebut bisa diganti.
8	UNIT(productions, variables) → list	Mengeliminasi unit production dari suatu aturan produksi.

### helper.py

No	Fungsi/Prosedur	Tujuan
1	union(lst1, lst2) → list	Fungsi untuk menggabungkan dua buah list, jika ada duplikasi maka akan dieliminasi.
2	loadModel(modelPath) → (list, list, list)	Fungsi untuk memuat file cfg dan mengelompokkannya menjadi terminal, variable, dan productions.
3	cleanProduction(expression) → list	Fungsi untuk membersihkan ekspresi grammar dari whitespace, dan mengelompokkannya menjadi bagian rules dan hasil produksi.
4	cleanAlphabet(expression) → list	Fungsi untuk memisahkan setiap rules yang ada.
5	seekAndDestroy(target, production) → (list, list)	Fungsi untuk menghapus sebuah non-terminal production dari sebuah aturan produksi.
6	setupDict(productions, variables, terms) → dictionary	Fungsi untuk menginisiasi penggunaan dictionary untuk menyimpan aturan produksi, agar berupa key, value.

7	rewrite(target, productions) → list	Fungsi untuk mengolah ulang aturan produksi yang telah ada, dan mengeliminasi berdasarkan target yang diberikan.
8	dict2set(dictionary) → list	Fungsi untuk mengubah dictionary menjadi bentuk list.
9	pprintRules(rules)	Prosedur untuk mencetak semua aturan produksi yang telah dibentuk.
10	prettyForm(rules) → string	Fungsi untuk mengubah semua bentuk aturan produksi menjadi string agar bisa ditulis ke dalam file output.

### 3. Program Utama

Program compiler bahasa pemrograman python yang telah dibuat terdiri dari satu class yang terdiri atas beberapa variables dan method untuk mengevaluasi kebenaran sintaks dari Bahasa pemrograman python yang diberikan. Program utama ini merupakan file **cyk.py**. Class CykParser merupakan class utama dari program yang membutuhkan dua buah konstruktor pada inisiasinya, yaitu file txt dari grammar CNF dan file txt yang berisi bahasa pemrograman python yang akan diuji. Class ini memiliki beberapa variable yang menyimpan hasil pengolahan data dari setiap method yang ada di class ini.

Berikut merupakan penjelasan setiap variabel pada class CykParser:

No	Variabel	Tujuan
1	cnfPath	Menyimpan <i>path to file</i> dari grammar cnf yang dibutuhkan program.
2	testFile	Menyimpan <i>path to file</i> dari bahasa pemrograman yang akan diuji.
3	chomskyGrammar: dictionary	Menyimpan grammar CNF yang telah diolah dari file txt grammar CNF.

4	validString: bool	Menyimpan informasi apakah semua string yang ada di dalam bahasa yang diuji memiliki format yang valid.
5	inputText: list	Menyimpan bahasa python yang sudah dibersihkan dan dibentuk menjadi list yang bisa diolah oleh algoritma cyk.
6	cykTable: list of set	Menyimpan table hasil algoritma cyk dari inputText yang diberikan.
7	contents: str	Menyimpan hasil pembacaan file txt bahasa python yang diberikan.
8	err_line: int	Menyimpan letak string yang memiliki format yang salah.

Selain variabel, class CykParser juga memiliki beberapa method untuk mengolah bahasa Python yang diberikan, berikut adalah penjelasannya:

No	Method	Tujuan
1	loadGrammar	Membaca file grammar CNF yang diberikan dan mengolah setiap aturan produksi agar bisa dimasukkan ke dalam dictionary chomskyGrammar.
2	string_analyzer	Melakukan validasi setiap string yang ada, kemudian menggantinya menjadi string sederhana untuk memudahkan evaluasi dengan algoritma cyk. Method ini juga memvalidasi <i>comment</i> dan membersihkannya dari contents yang akan dievaluasi.
3	readInputFile	Melakukan formatting, yaitu pemisahan setiap ekspresi dalam bahasa yang diberikan berdasarkan <i>delimiters</i> agar menjadi sebuah

		simbol terminal yang valid, serta melakukan pembersihan contents dari string kosong untuk memudahkan proses validasi.
4	insertTable	Metode penunjang untuk memudahkan insert rules yang memenuhi ke dalam tabel cyk.
5	makeCYKTable	Method utama yang mengandung algoritma cyk yang dimaksud. Melakukan validasi variabel, angka, dan string, serta mengevaluasi setiap ekspresi bahasa python dengan algoritma cyk.
6	printCYKTable	Mencetak hasil dari algoritma cyk dalam bentuk tabel.
7	result	Mencetak hasil keseluruhan dari program yang telah berjalan, yaitu apakah bahasa diterima oleh grammar CNF yang telah dibuat atau tidak.
8	validate	Method yang untuk mengatur proses jalannya algoritma, dengan memanggil setiap method yang sudah dijelaskan di atas, berdasarkan kondisi yang memenuhi. Method ini merupakan satu-satunya method public yang bisa dipanggil ketika membuat object CykParser baru.

## B. PENGUJIAN

### 1. Input dan Output

Berikut ini adalah *code-code* yang akan di-*compile* oleh *compiler* bahasa Python yang telah kami buat:

a.

```
tess = (input(hahaha))

tess2 = str(input())

aa = a + ((5 * c) % f) / 10

print(aa)

print('hahahahaha')

print("$5lhyr*&^+ _/")
```

Output yang didapatkan:

```
Path to test file: input.txt
[{'COMPOPRT', 'VRET', 'VIMPORT', 'COMP', 'VRANGE', 'VCLASS', 'X4', 'VAR', 'VFUNCTION', 'VDEF', 'VAL', 'NUMBER'}, {'T4'}, {'T4'}, {'O4'}, {'T4'}, {'COMPOPRT', 'VRET', 'VIMPORT', 'COMP', 'VRANGE', 'VCLASS', 'X4', 'VAR', 'VFUNCTION', 'VDEF', 'VAL', 'NUMBER'}, {'S4'}, {'S4'}, {'COMPOPRT', 'VRET', 'VIMPORT', 'COMP', 'VRANGE', 'VCLASS', 'X4', 'VAR', 'VFUNCTION', 'VDEF', 'VAL', 'NUMBER'}, {'T4'}, {'O4'}, {'T4'}, {'S4'}, {'S4'}, {'COMPOPRT', 'VRET', 'VIMPORT', 'COMP', 'VRANGE', 'VCLASS', 'X4', 'VAR', 'VFUNCTION', 'VDEF', 'VAL', 'NUMBER'}, {'T4'}, {'COMPOPRT', 'VRET', 'VIMPORT', 'COMP', 'VRANGE', 'VCLASS', 'X4', 'VAR', 'VFUNCTION', 'VDEF', 'VAL', 'NUMBER'}, {'OP', 'OPARITH'}, {'T4'}, {'T4'}, {'COMPOPRT', 'VRET', 'VIMPORT', 'COMP', 'VRANGE', 'VCLASS', 'X4', 'VAR', 'VFUNCTION', 'VDEF', 'VAL', 'NUMBER'}, {'S4'}, {'OP', 'OPARITH'}, {'COMPOPRT', 'VRET', 'VIMPORT', 'COMP', 'VRANGE', 'VCLASS', 'X4', 'VAR', 'VFUNCTION', 'VDEF', 'VAL', 'NUMBER'}, {'S4'}, {'OP', 'OPARITH'}, {'COMPOPRT', 'VRET', 'VIMPORT', 'COMP', 'VRANGE', 'VCLASS', 'X4', 'VAR', 'VFUNCTION', 'VDEF', 'VAL', 'NUMBER'}, {'T4'}, {'COMPOPRT', 'VRET', 'VIMPORT', 'COMP', 'VRANGE', 'VCLASS', 'X4', 'VAR', 'VFUNCTION', 'VDEF', 'VAL', 'NUMBER'}, {'T4'}, {'U4'}, {'S4'}, {'R4'}, {'T4'}, {'U4'}, {'X4'}, {'U4'}, {'S4'}]
```

...

```
[{'S0', 'S'}, set(), set(), set(), set(), set(), set()]
[set(), set(), set(), set(), set(), set()]
[set(), set(), set(), set(), set()]
[set(), set(), set(), set()]
[set(), set(), set()]
[set(), set()]
[{'S0', 'S'}]
Accepted
```

Dari percobaan tersebut, didapatkan hasil “Accepted”. Potongan *code* tersebut sudah sesuai dengan syntax pada bahasa Python. Pada percobaan ini, *production* yang digunakan adalah INPUT, VAR, VAL, PRINT, COMPOPRT dan STRING.



b.

```
tess2 = str(input(hahha))

aa = a + ((5 * c) % f) 10

print(aa)

print('hahahahahaha')

print("$51hry*&^+ _/")
```

Output yang didapatkan:

```
Path to test file: input.txt
[{'VFUNCTION', 'VIMPORT', 'NUMBER', 'VRANGE', 'VAL', 'COMPOPT', 'X4', 'VDEF', 'VRET', 'E', 'VAL', 'COMPOPT', 'X4', 'VDEF', 'VRET', 'VCLASS', 'VAR', 'COMP', {'T4'}, {'Q4'}}, {'VDEF', 'VRET', 'VCLASS', 'VAR', 'COMP', {'S4'}, {'VFUNCTION', 'VIMPORT', 'NUMBER', {'Z4'}, {'VFUNCTION', 'VIMPORT', 'NUMBER', 'VRANGE', 'VAL', 'COMPOPT', 'X4', 'VDEF', 'VRET', 'VCLASS', 'VAR', 'COMP', {'S4'}, {'OP', 'OPARITH', {'VFUNCTION', 'VIMPORT', 'VRET', 'VCLASS', 'VAR', 'COMP', {'S4'}, {'OP', 'OPARITH', {'VFUNCTION', 'VIMPORT', 'COMP', {'S4'}, {'VFUNCTION', 'NUMBER', 'VRANGE', 'VAL', 'COMPOPT', 'X4', 'VRET', 'T', 'X4', 'VDEF', 'VRET', 'VCLASS', 'VAR', 'COMP', {'S4'}, {'R4'}, {'T4'}, {'U4'}, {'set(), {'L31', 'A1', 'T12', 'C2', 'U12'}, set(), set(), set(), set(), {'D21', 'A15', 'T12', 'C2', 'U12'}, set(), set(), set(), set(), set(), {'VFUNCTION', 'M1', 'NUMBER', 'Y11'}, {'D21', 'A15', 'E12', 'N31', 'I33', 'X2', 'S1', 'Q11', 'U2'}, set(), {'VFUNCTION', 'A22', 'O2', 'P2', 'Y11'}, {'D21', 'A15', 'E12', 'N31', 'I33', 'X2', 'S1', 'Q11', 'U1', 'Q11', 'U2'}, set(), set(), set(), set(), {'I5', 'K5', 'G1'}, set(), set(), set()]}
...
[set(), set(), set(), set(), set(), set()]
[set(), set(), set(), set(), set()]
[set(), set(), set(), set()]
[set(), set(), set()]
[set(), set()]
[set()]
Syntax error!
```

Dari percobaan tersebut, hasil yang didapatkan adalah “Syntax Error”. Hal ini karena potongan *code* tersebut masih terdapat kesalahan secara syntax pada bagian yang telah diberi *highlight* warna kuning. Pada *line* ke-1, seharusnya terdapat satu tanda tutup kurung lagi dan pada *line* ke-2 seharusnya terdapat operator aritmatik yang menghubungkan dengan angka 10.

## 2. Percabangan (If, Elif, Else)

Berikut ini adalah *code-code* yang akan di-*compile* oleh *compiler* bahasa Python yang telah kami buat:

a.

```
if hh == 3:

    print(hh)

elif (hh % 3) != 0:

    hh = 10 * 5

else :

    print(pp)

if ((hh - 5) + 3) > 2:

    print(pp)

else:

    print(p)
```

Output yang didapatkan:

```
Path to test file: input.txt
[{'D4'}, {'VCLASS', 'VAR', 'VRANGE', 'VIMPORT', 'NUMBER', 'VRET', 'VAL', 'COMP', 'VDEF', 'CO
T', 'VAL', 'COMPOPRT', 'VFUNCTION', 'X4'}, {'M4'}, {'R4'}, {'T4'}, {'VCLASS', 'VAR', 'VRANGE
UNCTION', 'X4'}, {'S4'}, {'C4'}, {'T4'}, {'VCLASS', 'VAR', 'VRANGE', 'VIMPORT', 'NUMBER', 'V
', 'OP'}, {'VRANGE', 'NUMBER', 'VRET', 'VAL', 'COMPOPRT', 'VFUNCTION', 'X4'}, {'S4'}, {'OP',
FUNCTION', 'X4'}, {'M4'}, {'VCLASS', 'VAR', 'VRANGE', 'VIMPORT', 'NUMBER', 'VRET', 'VAL', 'C
ER', 'VRET', 'VAL', 'COMPOPRT', 'VFUNCTION', 'X4'}, {'OPARITH', 'OP'}, {'VRANGE', 'NUMBER',
{'T4'}, {'VCLASS', 'VAR', 'VRANGE', 'VIMPORT', 'NUMBER', 'VRET', 'VAL', 'COMP', 'VDEF', 'CO
', 'VAR', 'VRANGE', 'VIMPORT', 'NUMBER', 'VRET', 'VAL', 'COMP', 'VDEF', 'COMPOPRT', 'VFUNCTION
POPRT', 'VFUNCTION', 'X4'}, {'S4'}, {'OPARITH', 'OP'}, {'VRANGE', 'NUMBER', 'VRET', 'VAL', '
UMBER', 'VRET', 'VAL', 'COMPOPRT', 'VFUNCTION', 'X4'}, {'M4'}, {'R4'}, {'T4'}, {'VCLASS', 'V
MPOPRT', 'VFUNCTION', 'X4'}, {'S4'}, {'B4'}, {'M4'}, {'R4'}, {'T4'}, {'VCLASS', 'VAR', 'VRAN
VFUNCTION', 'X4'}, {'S4'}]

...

[{'S0', 'J23', 'L23', 'S23', 'S', 'IF'}, set(), set(), set(), set(), set(), {'S23', 'J23'}]
[set(), set(), set(), set(), set(), {'J22', 'S22'}]
[set(), set(), set(), set(), set()]
[set(), set(), set(), set()]
[set(), set(), set()]
[set(), {'J21', 'S21'}]
[{'S0', 'J23', 'L23', 'S23', 'S', 'IF'}]
```

Dari percobaan tersebut, didapatkan hasil “Accepted”. Potongan *code* tersebut sudah sesuai dengan syntax pada bahasa Python. Pada percobaan ini, *production* yang digunakan adalah IF, ELIF, ELSE, VAR, VAL, PRINT, COMPOPRT dan STRING.

b.

```
if (hh % 3) != 0:

    hh = 10 * 5

else :

    print(pp)

elif ((hh - 5) + 3) == 2:

    print(pp)
```

Output yang didapatkan:

```
Path to test file: input.txt
[{'D4'}, {'T4'}, {'VCLASS', 'VDEF', 'VRANGE', 'X4', 'VAR', 'VFUNCTION', 'VAL', 'COMPOPRT', 'GE', 'VFUNCTION', 'VAL', 'COMPOPRT', 'X4', 'NUMBER'}, {'S4'}, {'OPCOMP', 'OP'}, {'Z4'}, {'VR', {'VCLASS', 'VDEF', 'VRANGE', 'X4', 'VAR', 'VFUNCTION', 'VAL', 'COMPOPRT', 'VIMPORT', 'COMP', 'COMPOPRT', 'X4', 'NUMBER'}, {'OPARITH', 'OP'}, {'VRET', 'VRANGE', 'VFUNCTION', 'VAL', 'COMPOPRT', 'VIMPORT', 'COMP', 'VRET', 'NUMBER'}, {'OPARITH', 'OP'}, {'VRET', 'VRANGE', 'VFUNCTION', 'VAL', 'COMPOPRT', 'VIMPORT', 'COMP', 'VRET', 'NUMBER'}, {'OPARITH', 'OP'}, {'S4'}, {'OPARITH', 'OP'}, {'VRET', 'VRANGE', 'VFUNCTION', 'VAL', 'COMPOPRT', 'X4', 'NUMBER', 'COMPOPRT', 'X4', 'NUMBER'}, {'M4'}, {'R4'}, {'T4'}, {'VCLASS', 'VDEF', 'VRANGE', 'X4', 'VAR', 'VFUNCTION', 'VAL', 'COMPOPRT', 'X4', 'NUMBER'}, {'S4'}]
```

...

```
[set(), set(), set(), set(), set(), set()]
[set(), set(), set(), set(), set()]
[set(), set(), set(), set()]
[set(), set(), set()]
[set(), set()]
[{'S23', 'J23'}]
Syntax error!
```

Dari percobaan tersebut, hasil yang didapatkan adalah “Syntax Error”. Hal ini karena potongan *code* tersebut masih terdapat kesalahan secara syntax pada bagian yang telah diberi *highlight* warna kuning. Pada *line* ke-5, seharusnya *elif* tidak boleh ada apabila *if* yang mendahuluinya belum ada.

### 3. Loop (For, While)

Berikut ini adalah *code-code* yang akan di-*compile* oleh *compiler* bahasa Python yang telah kami buat:

a.

```
for i in range(0, 8, 2):  
  
    for j in (pp):  
  
        while (p + 3 < 5):  
  
            while haha and (hh - (pp % 7) == ijk) or (not pp):  
  
                print('hahhahaha')
```

Output yang didapatkan:

```
Path to test file: input.txt  
[{'P4'}, {'COMPOprt', 'VCLASS', 'VAR', 'VIMPORT', 'COMP', 'VAL', 'VDEF', 'X4', 'VFUNCTION',  
'COMPOprt', 'VAL', 'X4', 'VFUNCTION', 'NUMBER', 'VRANGE'}, {'N4'}, {'VRET', 'COMPOprt', 'VAL',  
'VAL', 'X4', 'VFUNCTION', 'NUMBER', 'VRANGE'}, {'S4'}, {'M4'}, {'P4'}, {'COMPOprt', 'VCLAS  
'NUMBER', 'VRANGE'}, {'OIDENTITY'}, {'T4'}, {'COMPOprt', 'VCLASS', 'VAR', 'VIMPORT', 'COM  
4'}, {'M4'}, {'F4'}, {'T4'}, {'COMPOprt', 'VCLASS', 'VAR', 'VIMPORT', 'COMP', 'VAL', 'VDEF',  
VRET', 'COMPOprt', 'VAL', 'X4', 'VFUNCTION', 'NUMBER', 'VRANGE'}, {'OPCOMP', 'OP'}, {'VRET',  
M4'}, {'F4'}, {'COMPOprt', 'VCLASS', 'VAR', 'VIMPORT', 'COMP', 'VAL', 'VDEF', 'X4', 'VFUNCTION',  
CLASS', 'VAR', 'VIMPORT', 'COMP', 'VAL', 'VDEF', 'X4', 'VFUNCTION', 'VRET', 'NUMBER', 'VRANG  
'COMP', 'VAL', 'VDEF', 'X4', 'VFUNCTION', 'VRET', 'NUMBER', 'VRANGE'}, {'OPARITH', 'OP'},  
'S4'}, {'Z4'}, {'Z4'}, {'COMPOprt', 'VCLASS', 'VAR', 'VIMPORT', 'COMP', 'VAL', 'VDEF', 'X4',  
, {'L4'}, {'COMPOprt', 'VCLASS', 'VAR', 'VIMPORT', 'COMP', 'VAL', 'VDEF', 'X4', 'VFUNCTION',  
'X4'}, {'U4'}, {'S4'}]
```

...

```
[set(), set(), set(), set(), set(), set(), set())  
[set(), set(), set(), set(), set(), set())  
[set(), set(), set(), set(), set())  
[set(), set(), set(), set())  
[set(), set(), set())  
[set(), set())  
[{'FOR', 'S', 'S0'}]  
Accepted
```

Dari percobaan tersebut, didapatkan hasil “Accepted”. Potongan *code* tersebut sudah sesuai dengan syntax pada bahasa Python. Pada percobaan ini, *production* yang digunakan adalah FOR, WHILE, VAR, VAL, PRINT, COMPOprt dan STRING.

b.

```
for i in (0, 8, 2):  
  
    for j in (pp):  
  
        while (p + 3 < 5):  
  
            while haha  
  
                print('hahhahaha')
```

Output yang didapatkan:

```
Path to test file: input.txt
[{'P4'}, {'COMPOprt', 'VFUNCTION', 'VAR', 'VIMPORT', 'COMP', 'X4', 'VCLASS', 'VDEF', 'VAL', 'NCTION', 'X4', 'VAL', 'VRANGE', 'NUMBER', 'VRET'}, {'N4'}, {'COMPOprt', 'VFUNCTION', 'X4', 'X4', 'VAL', 'VRANGE', 'NUMBER', 'VRET'}, {'S4'}, {'M4'}, {'P4'}, {'COMPOprt', 'VFUNCTION', 'BER', 'VRET'}, {'OPIDENTITY'}, {'T4'}, {'COMPOprt', 'VFUNCTION', 'VAR', 'VIMPORT', 'COMP', '4'}, {'F4'}, {'T4'}, {'COMPOprt', 'VFUNCTION', 'VAR', 'VIMPORT', 'COMP', 'X4', 'VCLASS', 'VD', 'VFUNCTION', 'X4', 'VAL', 'VRANGE', 'NUMBER', 'VRET'}, {'OP', 'OPCOMP'}, {'COMPOprt', 'VF', 'F4'}, {'COMPOprt', 'VFUNCTION', 'VAR', 'VIMPORT', 'COMP', 'X4', 'VCLASS', 'VDEF', 'VAL', 'VR'}]]
```

...

```
[set(), set(), set(), set(), set(), set()]
[set(), set(), set(), set(), set()]
[set(), set(), set(), set()]
[set(), set(), set()]
[set(), set()]
[set()]
Syntax error!
```

Dari percobaan tersebut, hasil yang didapatkan adalah “Syntax Error”. Hal ini karena potongan *code* tersebut masih terdapat kesalahan secara syntax pada bagian yang telah diberi *highlight* warna kuning. Pada *line* ke-1, seharusnya sebelum tanda kurung buka terdapat *range* dan pada *line* ke-4 harus ada ‘.’ setelah variabel *haha*.

#### 4. Import, Class, Function

Berikut ini adalah *code-code* yang akan di-*compile* oleh *compiler* bahasa Python yang telah kami buat:

a.

```
from hello.py import hello.rb, yeye, zeze

import snake as eel

class object(class1, class2):

    def __init__(self):

        self.me = "HELLO"

    def printMe():

        if self.me == "cape":

            print()

        elif self.me != None:

            print("Hello, my name is...?")

        else:

            return
```

Output yang didapatkan:

```
C:\Users\jvsit\PyProject\codes\T080-121\code\adec\py\cyk.py 4.txt
[{'L4'}, {'VRANGE', 'VCLASS', 'COMP', 'VRET', 'VAR', 'VDEF', 'VFUNTION', 'COMPOPRT', 'VAL', 'ICLASS',
E', 'VCLASS', 'COMP', 'VRET', 'VAR', 'VDEF', 'VFUNTION', 'COMPOPRT', 'VAL', 'ICLASS', 'X4', 'NUMBER',
OMP', 'VRET', 'VAR', 'VDEF', 'VFUNTION', 'COMPOPRT', 'VAL', 'ICLASS', 'X4', 'NUMBER', 'VIMPORT'}, {'Y4
AR', 'VDEF', 'VFUNTION', 'COMPOPRT', 'VAL', 'ICLASS', 'X4', 'NUMBER', 'VIMPORT'}, {'04'}, {'VRANGE', '
UNCTION', 'COMPOPRT', 'VAL', 'ICLASS', 'X4', 'NUMBER', 'VIMPORT'}, {'04'}, {'VRANGE', 'VCLASS', 'COMP',
PRT', 'VAL', 'ICLASS', 'X4', 'NUMBER', 'VIMPORT'}, {'K4'}, {'VRANGE', 'VCLASS', 'COMP', 'VRET', 'VAR',
LASS', 'X4', 'NUMBER', 'VIMPORT'}, {'J4'}, {'VRANGE', 'VCLASS', 'COMP', 'VRET', 'VAR', 'VDEF', 'VFUNTION
MBER', 'VIMPORT'}, {'A4'}, {'VRANGE', 'VCLASS', 'COMP', 'VRET', 'VAR', 'VDEF', 'VFUNTION', 'COMPOPRT',
}, {'T4'}, {'VRANGE', 'VCLASS', 'COMP', 'VRET', 'VAR', 'VDEF', 'VFUNTION', 'COMPOPRT', 'VAL', 'ICLASS',
NGE', 'VCLASS', 'COMP', 'VRET', 'VAR', 'VDEF', 'VFUNTION', 'COMPOPRT', 'VAL', 'ICLASS', 'X4', 'NUMBER',
NGE', 'VCLASS', 'COMP', 'VRET', 'VAR', 'VDEF', 'VFUNTION', 'COMPOPRT', 'VAL', 'ICLASS', 'X4', 'NUMBER',
'COMP', 'VRET', 'VAR', 'VDEF', 'VFUNTION', 'COMPOPRT', 'VAL', 'ICLASS', 'X4', 'NUMBER', 'VIMPORT'}, {'S
```

...

```
[set(), set(), set(), set(), {'S', 'S0', 'G23', 'P23'}, set(), set(), set()]  
[set(), set(), set(), set(), set(), set(), set()]  
[set(), set(), set(), set(), set(), set()]  
[set(), set(), set(), set(), {'S0', 'S', 'G23', 'P23', 'I23'}]  
[{'S', 'S0', 'G23', 'P23'}, set(), set(), set()]  
[set(), set(), set()]  
[set(), set()]  
[{'S0', 'S', 'G23', 'P23', 'I23'}]  
Accepted
```

Dari percobaan tersebut, didapatkan hasil “Accepted”. Potongan *code* tersebut sudah sesuai dengan syntax pada bahasa Python. Pada percobaan ini, *production* yang digunakan adalah IMPORT, CLASS, DEF, IF, ELIF, dan ELSE.

b.

```
import halo()  
  
class object:  
    def nonFactor:  
        print()
```

Output yang didapatkan:

```
[set(), set(), set(), set(), set(), set(), set(), set()]  
[set(), set(), set(), set(), set(), set(), set(), set()]  
[set(), set(), set(), set(), set(), set()]  
[set(), set(), set(), set(), set()]  
[set(), set(), set(), set()]  
[set(), set(), set()]  
[set(), set()]  
[set()]  
Syntax error!
```

Dari percobaan tersebut, hasil yang didapatkan adalah “Syntax Error”. Hal ini karena potongan *code* tersebut masih terdapat kesalahan secara syntax pada bagian yang telah diberi *highlight* warna kuning. Pada *line* ke-1, import tidak dapat diikuti oleh *parentheses*, karena menandakan function, cukup dengan menuliskan nama yang akan diimport saja. Pada *line* ke-3, penulisan function seharusnya diikuti dengan *parentheses*, dan juga argument didalamnya jika ada.

## 5. Kode Lengkap

Berikut ini adalah *code-code* yang akan di-*compile* oleh *compiler* bahasa Python yang telah kami buat, yang mengandung grammar yang cukup lengkap:

a.

```
'''
This is multiline comment

Obviously

''' """ This is another
:) """

# this is oneline comment

from pp.lol import haha, hihi, huhu

import ppp.svdd

class Panda:

    def __init__ (self, haha='haha', hhh, dttd=5) :

        haha = input("What is your nameeee?")

        print(haha)

        if p == -5.22:

            a = b.c.d(woy, woy, woy)

            haha = ppppn(4, lllln(p, 3))

        elif p != 5:

            print('haha >> 5 + 5 <<|')

            print("me + you = ? '5 + hello'")

            pass
```



```
elif (p + 7.2) + .15 - hh >= k:

    pq = False

    a = [[5], [5, [5, (5, 5, [5])]]]

    print(a[:5], b[:])

else:

    hhh += 10

    raise IDontKnowError("HAHAHA")

    return (pp, (gdgd == 3), gg)

def iterateMePlease(self):

    while a>0:

        for i in range(1,9,2):

            me.call('62822..').get()

            a = me.reply('Hello').getCredential

panda = Panda(hhh='meh')

panda.iterateMePlease()
```

Output yang didapatkan:

```
...
...
[set(), set(), set(), set(), {'S0', 'S'}, set(), set(), set(), set(), set()]
[set(), set(), set(), set(), set(), set(), set(), set(), set()]
[set(), set(), set(), set(), set(), set(), set(), set()]
[set(), set(), set(), set(), set(), set(), set()]
[{'S0', 'S'}, set(), set(), set(), set(), set()]
[set(), set(), set(), set(), {'S0', 'S'}]
[set(), set(), set(), set()]
[set(), set(), set()]
[set(), set()]
[{'S0', 'S'}]
Accepted
```

Karena merupakan code python yang valid secara sintaks, maka kode tersebut diterima (Accepted). Pada percobaan ini, *production* yang digunakan antara lain, FOR, WHILE, IMPORT, CLASS, DEF, IF, RAISE.

## LAMPIRAN

### A. Repository Github

<https://github.com/adellinekania/Tubes-TBFO-Compiler-Python.git>

### B. Pembagian Tugas

<i>Nama / NIM</i>	<i>Tugas</i>
Putri Nurhaliza / 13520066	Context Free Grammar, laporan bab I dan II
Jova Andres Riski Sirait / 13520072	Context Free Grammar, program utama, laporan bab III
Adelline Kania Setiyawan / 13520084	Context Free Grammar, method CYK Table, laporan bab I dan III

## **DAFTAR PUSTAKA**

- [1] <https://text-id.123dok.com/document/6zkexnpmz-algoritma-cocke-younger-kasami-cyk-analisis-perbandingan-algoritma-left-corner-parsing-dan-algoritma-cyk-cocke-younger-kasami-untuk-memeriksa-pola-kalimat-baku-bahasa-indonesia.html>. Diakses pada 21 November 2021.
- [2] <https://www.gatevidyalay.com/cyk-cyk-algorithm/>. Diakses pada 21 November 2021.
- [3] <https://github.com/adelmassimo/CFG2CNF>. Diakses pada 16 November 2021.