

**LAPORAN TUGAS BESAR IF2124 TEORI BAHASA FORMAL  
DAN AUTOMATA APLIKASI ALGORITMA COCKE-  
YOUNGER-KASAMI PADA *COMPILER* PYTHON  
SEDERHANA**



Disusun oleh

Suryanto	13520059
Raden Haryosatyo Wisjnunandono	13520070
Farrel Ahmad	13520110

**TEKNIK INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG  
BANDUNG  
2021**

## **DAFTAR ISI**

<b>BAB I DASAR TEORI.....</b>	<b>3</b>
<b>BAB II ANALISIS PERSOALAN DAN DEKOMPOSISI .....</b>	<b>6</b>
<b>BAB III IMPLEMENTASI DAN PENGUJIAN.....</b>	<b>13</b>
<b>BAB IV KESIMPULAN DAN SARAN.....</b>	<b>20</b>
<b>REFERENSI.....</b>	<b>21</b>

# **BAB I**

## **DASAR TEORI**

### **1.1 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.

Definisi formal dari CFG dapat didefinisikan sebagai berikut.

$$\mathbf{G = (V, T, P, S)}$$

Dengan:

G = Context-Free Grammar

V = Simbol Non-Terminal

T = Simbol Terminal

P = Himpunan terbatas dari produksi

S = Simbol *start*

### **1.2 Chomsky Normal-Form**

Bentuk normal Chomsky / Chomsky Normal Form (CNF) merupakan salah satu bentuk normal yang sangat berguna untuk Context Free Grammar (CFG). Bentuk normal Chomsky dapat dibuat dari sebuah tata bahasa bebas konteks yang telah mengalami penyederhanaan yaitu penghilangan produksi useless, unit, dan  $\epsilon$ . Dengan kata lain, suatu tata bahasa bebas konteks dapat dibuat menjadi bentuk normal Chomsky dengan syarat tata bahasa bebas konteks tersebut:

- Tidak memiliki produksi useless
- Tidak memiliki produksi unit
- Tidak memiliki produksi  $\epsilon$

Bentuk normal Chomsky (Chomsky Normal Form, CNF) adalah Context Free Grammar (CFG) dengan setiap produksinya berbentuk :

$$\mathbf{A \rightarrow BC \text{ atau } A \rightarrow a.}$$

### 1.3 Algoritma Cocke-Younger-Kasami (CYK)

Algoritma Cocke-Younger-Kasami adalah salah satu algoritma parsing untuk memeriksa apakah suatu *string* dapat diterima oleh suatu language Context-Free Grammar. Algoritma ini membutuhkan sebuah grammar CFG yang telah dikonversi menjadi bentuk Chomsky Normal Form.

Berikut adalah contoh dari tabel CYK yang memeriksa apakah sebuah string = "baaba" dapat diterima oleh suatu CFG dalam bentuk CNF.

Aturan produksi dari CFG adalah :

- $S \rightarrow AB \mid BC$
- $A \rightarrow BA \mid a$
- $B \rightarrow CC \mid b$
- $C \rightarrow AB \mid a$

S,A,C				
-	S,A,C			
-	B	B		
S,A	B	S,C	S,A	
B	A,C	A,C	B	A,C
b	a	a	b	a

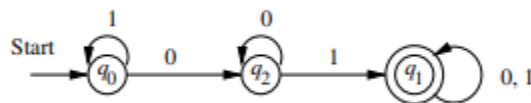
Dapat di lihat pada tabel CYK di atas, *start-symbol* S berada pada *cell* teratas yang menandakan bahwa *string* = "baaba" diterima oleh CFG.

### 1.4 Finite Automata (FA)

Finite Automata adalah mesin automata level 3 yang digunakan untuk mengenali suatu pola huruf. Secara definisi formalnya, automata adalah koleksi dari 5 tuple, yaitu  $A = (Q, \Sigma, \delta, q_0, F)$ . Penjelasan dari masing-masing tuple ini adalah

1.  $Q$  : himpunan (set) dari state-state yang ada
2.  $\Sigma$  : himpunan (set) simbol input
3.  $\delta$  : fungsi transisi
4.  $q_0$  : state awal
5.  $F$  : state akhir

Finite Automata memiliki dua state, yaitu state accepted dan rejected. Suatu input string diterima apabila berada pada accepted state. Finite automata dibagi menjadi dua, yaitu DFA (Deterministic Finite Automata) dan NFA (Nondeterministic Finite Automata). Perbedaan dari DFA dan NFA adalah NFA pada NFA, suatu simbol input dapat ke lebih dari satu state sedangkan DFA tidak.



Gambar 1 Contoh DFA yang menerima string yang memiliki substring 01



*Gambar 2 Contoh NFA yang menerima string berakhiran 01*

## BAB II

### ANALISIS PERSOALAN DAN DEKOMPOSISI

#### 2.1 Deskripsi Permasalahan

Pada tugas besar ini, penulis diminta untuk membuat sebuah *compiler* sederhana untuk bahasa pemrograman Python. Program yang dibangun dapat memeriksa kebenaran dari *statement*-statement dan sintaks-sintaks bawaan Python. Digunakan konsep CFG untuk mengevaluasi sintaks pada program yang ingin diujikan dan konsep FA untuk pengecekan variabelnya.

Dalam proses pembuatan program dari sebuah bahasa menjadi instruksi yang dapat dieksekusi oleh mesin, terdapat pemeriksaan sintaks atau kompilasi bahasa yang dibuat oleh programmer. Kompilasi ini bertujuan untuk memastikan instruksi yang dibuat oleh programmer mengikuti aturan yang sudah ditentukan oleh bahasa tersebut. Baik bahasa berjenis *interpreter* maupun *compiler*, keduanya pasti melakukan pemeriksaan sintaks. Perbedaannya terletak pada apa yang dilakukan setelah proses pemeriksaan (kompilasi/*compile*) tersebut selesai dilakukan.

Dibutuhkan *grammar* bahasa dan algoritma *parser* untuk melakukan kompilasi. Sudah sangat banyak *grammar* dan algoritma yang dikembangkan untuk menghasilkan *compiler* dengan performa yang tinggi. Terdapat CFG,  $CNF^-$ ,  $CNF^+$ , 2NF, 2LF, dll untuk *grammar* yang dapat digunakan, dan terdapat LL(0), LL(1), CYK, Earley's Algorithm, LALR, GLR, Shift-reduce, SLR, LR(1), dll untuk algoritma yang dapat digunakan untuk melakukan *parsing*.

False	class	<del>finally</del>	is	return
None	continue	for	<del>lambda</del>	<del>try</del>
True	def	from	<del>nonlocal</del>	while
and	<del>del</del>	<del>global</del>	not	with
as	elif	if	or	<del>yield</del>
<del>assert</del>	else	import	pass	<del>Async</del>
break	<del>except</del>	in	raise	<del>Await</del>

Tabel 2.1 daftar kata kunci bawaan Python yang diimplementasikan pada program

*Compiler* yang telah dibuat dapat membaca teks input melalui sebuah file eksternal dan setelahnya memberikan keluaran berdasarkan hasil pembacaan. Apabila *syntax* dari program dinilai benar, maka layar akan menampilkan pesan "Accepted!". Sebaliknya, pesan "Syntax Error!" beserta nomor *line* akan ditampilkan jika terdapat kesalahan sintaks pada file masukan.

## 2.2 Finite Automata

Dalam program ini kami menggunakan Finite Automata berjenis Nondeterministic Finite Automata (NFA). Nondeterministic Finite Automata ini direpresentasikan secara formal sebagai

$$A = (Q, \Sigma, \delta, q_0, F)$$

Dengan

$Q$  = himpunan terbatas dari state

$\Sigma$  = himpunan terbatas dari simbol masukan

$q_0$  = anggota himpunan  $Q$  yang merupakan start state

$F$  = subhimpunan dari  $Q$  yang merupakan final state

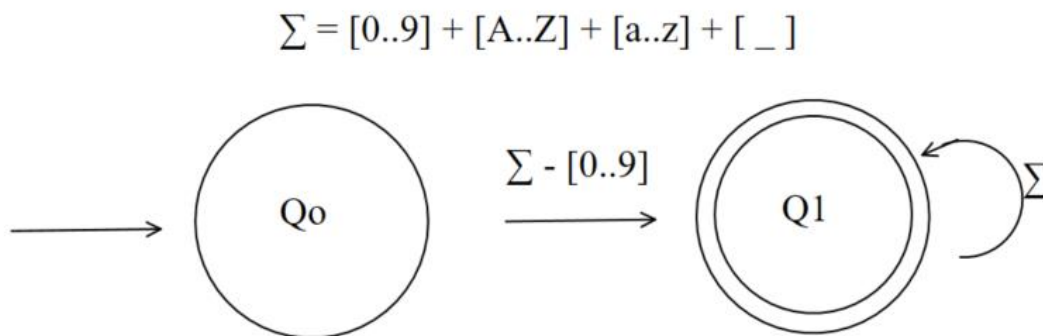
$\delta$  = fungsi transisi

Dalam program ini dibutuhkan Nondeterministic Finite Automata yang menerima variable yang penamaannya sesuai dengan kaidah penamaan variable pada bahasa Python, yakni NFA yang menerima masukan simbol alfa-numerik dan garis bawah (0-9, A-z, \_) dan menerima himpunan string yang berawalan huruf atau garis bawah (\_).

NFA yang terbentuk adalah didefinisikan sebagai berikut:

$$A = (\{Q_0, Q_1\}, \{A..z, 0..9, _\}, \delta, Q_0, \{Q_1\})$$

Dengan  $\delta$  direpresentasikan sebagai diagram transisi pada gambar berikut:



Gambar 3 NFA untuk proses variabel

## 2.3 Grammar CFG

S -> S S  
S -> ARBITRARY\_METHOD  
S -> COMMENT  
S -> FOR\_IMPORT  
S -> IFELSE  
S -> PASS  
S -> RAISE  
S -> PRINT\_METHOD  
S -> WHILE\_METHOD  
S -> FOR\_METHOD  
S -> OBJECT  
S -> ASSIGNMENT  
S -> CLASS  
S -> RETURN  
S -> BREAK  
S -> DEF  
S -> WITH  
S -> CONTINUE  
FLOAT\_METHOD -> CLOSE\_PAR  
FLOAT\_METHOD -> COMA METHOD\_2  
FLOAT\_METHOD -> PLUS NUMBERMET  
FLOAT\_METHOD -> MULTIPLY NUMBERMET  
FLOAT\_METHOD -> MINUS NUMBERMET  
FLOAT\_METHOD -> DIVIDE NUMBERMET  
FLOAT\_METHOD -> POWER NUMBERMET  
COMMENT -> TO\_COMMENT NEWLINE  
COMMENT -> TO\_COMMENT  
TO\_COMMENT -> 'COMMENT'  
FOR\_IMPORT -> 'FROM' A\_IMPORT  
FOR\_IMPORT -> B\_IMPORT  
A\_IMPORT -> IDENTIFIER B\_IMPORT  
B\_IMPORT -> 'IMPORT' C\_IMPORT  
C\_IMPORT -> IDENTIFIER D\_IMPORT  
C\_IMPORT -> IDENTIFIER  
D\_IMPORT -> 'AS' E\_IMPORT  
E\_IMPORT -> IDENTIFIER NEWLINE\_IMPORT  
E\_IMPORT -> IDENTIFIER  
NEWLINE\_IMPORT -> NEWLINE  
IDENTIFIER -> 'IDENTIFIER' DOT\_OPERATOR  
IDENTIFIER -> 'IDENTIFIER'  
IDENTIFIER -> NUMBER  
IDENTIFIER -> 'TRUE'  
IDENTIFIER -> 'FALSE'  
IDENTIFIER -> OBJECT  
DOT\_OPERATOR -> 'DOT\_OPERATOR'  
IDENTIFIER  
PRINT\_METHOD -> PRINT METHOD\_1  
METHOD\_1 -> OPEN\_PAR METHOD\_2  
METHOD\_2 -> OBJECT CLOSE\_PAR  
METHOD\_2 -> CLOSE\_PAR  
METHOD\_2 -> STRING METHOD\_STRING  
METHOD\_2 -> NUMBER  
METHOD\_2 -> EXPRESSION CLOSE\_PAR  
METHOD\_STRING -> CLOSE\_PAR  
METHOD\_STRING -> PLUS  
METHOD\_STRING\_PLUS  
METHOD\_STRING -> MULTIPLY  
MULTIPLY\_WITH\_INT  
METHOD\_STRING -> COMA METHOD\_2  
MULTIPLY\_WITH\_INT -> NUMBER  
METHOD\_STRING



METHOD\_STRING\_PLUS -> STRING  
 METHOD\_STRING  
 METHOD\_STRING\_PLUS -> IDENTIFIER  
 METHOD\_STRING  
 ID\_METHOD -> CLOSE\_PAR  
 ID\_METHOD -> COMA METHOD\_2  
 ID\_METHOD -> PLUS NUMBERMET  
 ID\_METHOD -> MULTIPLY M2  
 ID\_METHOD -> ID\_METHODNUS  
 NUMBERMET  
 ID\_METHOD -> DIVIDE NUMBERMET  
 ID\_METHOD -> POWER NUMBERMET  
 NUMBERMET -> NUMBER FLOAT\_METHOD  
 NUMBERMET -> NUMBER ID\_METHOD  
 NUMBERMET -> FLOAT FLOAT\_METHOD  
 NUMBERMET -> FLOAT ID\_METHOD  
 STRING -> 'STRING'  
 FLOAT -> 'FLOAT'  
 NUMBER -> 'NUMBER'  
 MULTIPLY -> 'MULTIPLY'  
 PRINT -> 'PRINT'  
 PLUS -> 'PLUS'  
 COMA -> 'COMA'  
 FUNC -> 'FUNC'  
 FOR\_METHOD -> FOR F1  
 F1 -> IDENTIFIER F2  
 F2 -> IN F3  
 F3 -> IDENTIFIER COLON  
 F3 -> ARBITRARY\_METHOD F5  
 F5 -> CLOSE\_PAR COLON  
 F5 -> CLOSE\_PAR  
 FOR -> 'FOR'  
 IN -> 'IN'

LEN -> 'LEN'  
 RANGE -> 'RANGE'  
 COLON -> 'COLON'  
 IFELSE -> IF IF2  
 IFELSE -> IF4  
 IFELSE -> IF7  
 IFELSE -> IF IF97  
 IF97 -> OBJECT IF0  
 IF -> 'IF'  
 ELSE -> 'ELSE'  
 ELIF -> 'ELIF'  
 ELIFTOK -> 'ELIFTOK'  
 CONTENT -> S  
 EXPRESSION -> OBJECT  
 EXPRESSION -> OBJECT A  
 EXPRESSION -> EXPRESSION A  
 A -> COMPARE\_OP EXPRESSION  
 B -> BINARY\_OP EXPRESSION  
 B -> ARITHMETIC\_OP EXPRESSION  
 EXPRESSION -> OPEN\_PAR C  
 C -> EXPRESSION CLOSE\_PAR  
 NOT -> 'NOT'  
 WHILE\_METHOD -> WHILE W1  
 W1 -> IDENTIFIER W4  
 W1 -> EXPRESSION W4  
 W1 -> OPEN\_PAR W2  
 W2 -> IDENTIFIER W3  
 W2 -> EXPRESSION W3  
 W3 -> CLOSE\_PAR W4  
 W4 -> COLON NEWLINE  
 W4 -> COLON  
 WHILE -> 'WHILE'

NEWLINE -> 'NEWLINE'  
 DEF -> DEFWORD DEF1  
 DEF1 -> IDENTIFIER DEF2  
 DEF1 -> IDENTIFIER COLON  
 DEF2 -> DEF4  
 DEF2 -> OPEN\_PAR DEF3  
 DEF3 -> DEF4  
 DEF3 -> IDENTIFIER DEF4  
 DEF3 -> STRING DEF4  
 EXPRESSION -> NOT EXPRESSION  
 EXPRESSION -> NOT EXPRESSION  
 EXPRESSION -> EXPRESSION B  
 DEF3 -> NUMBER DEF4  
 DEF3 -> IDENTIFIER  
 DEF3 -> STRING  
 DEF3 -> NUMBER  
 DEF3 -> DEF3 DEFINBETWEEN  
 DEFINBETWEEN -> COMA DEF3  
 DEF4 -> CLOSE\_PAR COLON  
 DEFWORD -> 'DEF'  
 WITH -> WITHWORD WITH0  
 WITH0 -> IDENTIFIER WITH1  
 WITH1 -> OPEN\_PAR WITH2  
 WITH2 -> IDENTIFIER WITH3  
 WITH2 -> STRING WITH3  
 WITH2 -> NUMBER WITH3  
 WITH3 -> CLOSE\_PAR WITH4  
 WITH3 -> COMA WITH2  
 WITH4 -> COLON  
 WITH4 -> 'AS' WITH5  
 WITH5 -> IDENTIFIER COLON  
 WITHWORD -> 'WITH'

COMPARE\_OP -> 'EQUALS'  
 COMPARE\_OP ->  
 'GREATER\_OR\_EQUAL\_THAN'  
 COMPARE\_OP -> 'LESS\_OR\_EQUAL\_THAN'  
 COMPARE\_OP -> 'GREATER\_THAN'  
 COMPARE\_OP -> 'LESS\_THAN'  
 COMPARE\_OP -> 'NOT\_EQUAL'  
 BINARY\_OP -> 'AND'  
 BINARY\_OP -> 'OR'  
 CLASS -> CLASSWORD CLASS1  
 CLASS1 -> IDENTIFIER COLON  
 CLASS1 -> IDENTIFIER DEF2  
 CLASSWORD -> 'CLASS'  
 OBJECT -> 'STRING'  
 OBJECT -> NUMBER  
 OBJECT -> 'IDENTIFIER'  
 OBJECT -> 'FALSE'  
 OBJECT -> 'NONE'  
 OBJECT -> 'TRUE'  
 OBJECT -> OBJECT OBJ1  
 OBJECT -> STRING OBJ1  
 OBJECT -> OBJECT OBJ3  
 OBJECT -> OBJECT OBJ0  
 OBJECT -> OBJECT OBJ0  
 OBJECT -> OBJECT OBJIN  
 OBJIN -> IN OBJECT  
 OBJ0 -> ARRAY  
 OBJ0 -> ARRAY OBJ1  
 OBJ0 -> ARRAY OBJ3  
 OBJ1 -> 'DOT\_OPERATOR' OBJ2  
 OBJ2 -> OBJECT  
 OBJ2 -> 'IDENTIFIER' ARRAY

OBJ2 -> OBJECT OBJ3  
 IFNEXT -> CONTENT  
 IF4 -> ELIFTOK IF99  
 IF99 -> ELIF IF5  
 IF5 -> EXPRESSION IF6  
 IF5 -> IDENTIFIER IF0  
 IF6 -> COLON IFNL  
 OBJ3 -> OPEN\_PAR OBJ4  
 OBJ3 -> OPEN\_PAR OBJCLOSE\_PAR  
 OBJCLOSE\_PAR -> CLOSE\_PAR  
 OBJ4 -> IDENTIFIER OBJ5  
 OBJ4 -> STRING OBJ5  
 OBJ4 -> NUMBER OBJ5  
 OBJ4 -> OBJECT OBJ5  
 OBJ4 -> EXPRESSION OBJ5  
 OBJ4 -> OBJ5  
 OBJ5 -> ARITHMETIC\_OP AOP1  
 AOP1 -> OBJ4  
 OBJ5 -> CLOSE\_PAR  
 OBJ5 -> CLOSE\_PAR OBJ1  
 OBJ2 -> OBJ1  
 ASSIGNMENT -> OBJECT ASS1  
 ASS1 -> ASSIGNTO OBJECT  
 ASS1 -> ASSIGNTO NUMBER  
 ASS1 -> ASSIGNTO STRING  
 ASS1 -> ASSIGNTO ARRAY  
 ASS1 -> ASSIGNTO EXPRESSION  
 ARRAY -> 'LSB' ARR1  
 ARR1 -> 'RSB'  
 ARR1 -> OBJECT 'RSB'  
 ARR1 -> STRING 'RSB'  
 ARR1 -> NUMBER 'RSB'

ARR1 -> 'IDENTIFIER' ARR2  
 ARR1 -> 'IDENTIFIER' 'RSB'  
 ARR2 -> FOR\_METHOD 'RSB'  
 ASSIGNTO -> 'ASSIGNS'  
 ARITHMETIC\_OP -> 'MINUS'  
 ARITHMETIC\_OP -> 'PLUS'  
 ARITHMETIC\_OP -> 'MULTIPLY'  
 ARITHMETIC\_OP -> 'DIVIDE'  
 ARITHMETIC\_OP -> 'POWER'  
 ARITHMETIC\_OP -> 'MOD'  
 RETURN -> 'RETURN' OBJECT  
 PASS -> 'PASS'  
 RAISE -> 'RAISE' OBJECT  
 BREAK -> 'BREAK'  
 CONTINUE -> 'CONTINUE'  
 ARBITRARY\_METHOD -> IDENTIFIER ARM1  
 ARM1 -> OPEN\_PAR ARM2  
 ARM2 -> ARBITRARY\_METHOD CLOSE\_PAR  
 ARM2 -> ARM1 CLOSE\_PAR  
 ARM2 -> OBJECT CLOSE\_PAR  
 ARM2 -> EXPRESSION CLOSE\_PAR  
 OPEN\_PAR -> 'OPEN\_PAR'  
 CLOSE\_PAR -> 'CLOSE\_PAR'  
 MULTILINE -> "TRIPLEQUOTE"  
 IF0 -> IN IF1  
 IF1 -> OBJECT COLON  
 IF2 -> EXPRESSION IF3  
 IF3 -> COLON IFNL  
 IF3 -> COLON  
 IFNL -> NEWLINE IFNEXT  
 IF6 -> COLON  
 IF7 -> ELIFTOK IF98

IF98 -> ELSE IF8

IF8 -> COLON IF9

IF8 -> COLON

IF9 -> NEWLINE IF10

IF9 -> IF10

IF10 -> CONTENT

## BAB III

### IMPLEMENTASI DAN PENGUJIAN

#### 3.1 Implementasi Program

##### 3.1.1 File lexer.py

File lexer.py berisi class-class yang berfungsi untuk mentokenisasi text masukan dari pengguna. Class-class yang berada dalam file lexer.py ini kemudian akan dipanggil dalam fungsi utama program yaitu main.py. Berikut ialah keterangan mengenai class-class yang terdapat dalam file lexer.py beserta penjelasan mengenai method yang kami gunakan.

Class Token	Penjelasan
def __init__(self, type, value, position) #Konstruktor	Konstruktor menerima parameter type, value, dan position dari Objek Token.
def __str__(self):	Method ini berfungsi untuk menampilkan Objek Token sebagai string supaya memudahkan debugging.

Tabel 3.1 Daftar *Method* pada Class Token

Class Lexer	Penjelasan
def __init__(self, rules, skip_whitespace): #Konstruktor	Konstruktor Objek Lexer dengan menerima parameter rules yang didefinisikan di file token_expressions.py. Selain itu juga menginisialisasi array holder untuk kemudian dipetakan.
input	Menerima input text dari pengguna yang akan ditokenisasi
token	Melakukan tokenisasi dari input yang telah dimasukkan pengguna. Method mengembalikan jenis token dari isi file yang diberikan pengguna.
tokens	Mengembalikan token hasil tokenisasi di method token serta melakukan terminasi bila input text pengguna ada yang tidak sesuai dengan aturan lexer yang terdapat pada file token_expressions.py

Tabel 3.2 Daftar *Method* pada Class Lexer

Class LexerError	Penjelasan
def __init__(self, pos)	Konstruktor yang menerima argument posisi. Class ini akan dipanggil jika terjadi kegagalan dalam Class Lexer.

Tabel 3.3 Daftar *Method* pada Class LexerError

### 3.1.2 File token\_expressions.py

File ini berisi rules-rules dari token yang nantinya akan digunakan lexer untuk membaca file masukan dari user.

### 3.1.3 File cekVar.py

File ini berisi fungsi fa yang berperan sebagai validator untuk variable yang ada dalam file pengguna sesuai dengan kaidah penamaan variable dalam bahasa Python. Variabel yang diterima ialah yang berawalan huruf atau garis bawah (\_). Selain itu, Variabel juga hanya terdiri dari karakter alfa-numerik dan garis bawah (A-z, 0-9, \_).

### 3.1.4 File CFGtoCNF.py

File ini berisi fungsi-fungsi yang dibutuhkan untuk mengubah *Context-Free Grammar* yang dibuat dalam file grammar.txt menjadi *Chomsky Normal Form*/Bentuk Normal Chomsky. File ini menerima masukan file .txt yang berisi grammar dari pengguna (yang dalam program kami ialah grammar.txt) dan setelah file ini dijalankan kemudian akan mengembalikan file .txt (yang dalam program kami adalah CNF.txt) yang berisikan Bentuk Normal Chomsky dari file grammar pengguna

Fungsi dan Prosedur	Penjelasan
readGrammar	Menerima file grammar sebagai parameter dan kemudian membuka serta membaca file tersebut. Fungsi mengembalikan array yang berisi elemen dari file grammar yang sudah displit menjadi sisi kanan tanda (->) dan sisi kiri tanda (->).
addRule	Menerima sebuah map yang berisi rule tentang grammar kemudian mengembalikan map tersebut beserta tambahan-tambahan rule yang ingin diberikan.
convertGrammar	Menerima parameter berupa Context-Free-Grammar dari pengguna dan mengonversi CFG itu dalam yang mulanya berbentuk: S -> NP VP NP -> Det ADV N dan seterusnya menjadi Bentuk Normal Chomsky dari CFG tersebut. Beberapa symbol non terminal baru dibuat supaya CFG tersebut sesuai dengan kaidah CNF. Simbol non terminal tersebut dinamai seperti simbol yang mereka ganti dengan indeks yang ditambahkan.

Tabel 3.4 Daftar Fungsi pada File CFG2CNF.py

### 3.1.5 File cyk.py

File ini berisi fungsi-fungsi dan prosedur yang merupakan implementasi dari algoritma CYK pada program compiler ini. Penulis memutuskan untuk tidak menggunakan prosedur `printTable` pada main program karena besarnya ukuran dari tabel yang dihasilkan. Namun, prosedur ini cukup penting sebagai salah satu *tool* untuk melakukan *debugging* pada program.

Fungsi dan Prosedur	Penjelasan
MafOfCNF (filename)	Menerima file grammar CNF hasil konversi dari grammar CFG dan mengembalikan Dictionary CNFmap. Selanjutnya, tiap aturan produksi dipecah menjadi dua, produksi kiri dan kanan. Nantinya simbol pada produksi sebelah kiri akan dipetakan ke produksi di sebelah kanan.
CYK(sentence, CNFMap)	Menerima sentence berupa line of code perbaris dari program yang diujikan yang dihasilkan oleh lexer. Selanjutnya, tiap string akan diperiksa dengan CNFMap untuk kemudian dimasukkan ke dalam tabel cyk itu sendiri.
printTable(table)	Prosedur ini menerima tabel cyk dan menampilkan ke layar.
cekValid(table)	Fungsi ini mengembalikan data bertipe boolean. Data True akan dikembalikan apabila tabel kosong (pada line tersebut hanya terdiri dari spasi) atau <i>start-symbol</i> 'S' berada pada <i>cell</i> akar table.

Tabel 3.5 Daftar Fungsi pada File cyk.py

### 3.1.6 File main.py

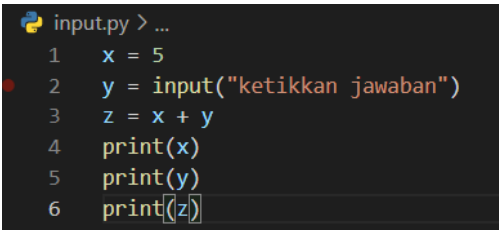
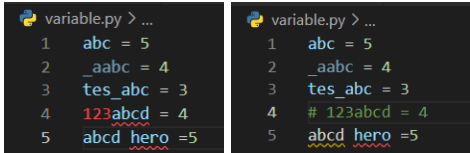
Sesuai dengan namanya, file ini berisi main program yang memanfaatkan semua fungsi dan prosedur dari file-file program di atas. Secara garis besar, program akan meminta user untuk memasukkan file yang ingin diujikan. Kemudian, program akan membaca file grammar CFG dan mengkonversinya menjadi grammar CNF. Setelahnya, program akan melakukan *parsing* dengan CYK secara bertahap baris per baris. Program akan berhenti apabila program lolos *compile* atau terdapat kesalahan sintaks pada baris tertentu.

No	Proses	Penjelasan
1	Pembacaan input file	Program dapat dijalankan dengan mengeksekusi main.py dengan <i>command line argumen</i> file yang ingin diujikan atau tanpa argumen. Apabila file yang dimasukkan user tidak ditemukan, maka program akan terus meminta user untuk memasukkan file yang sesuai.
2	Konversi grammar CFG ke dalam bentuk CNF	Program akan melakukan proses konversi file grammar CFG menjadi CNF. Proses ini dilakukan di main program untuk terus memperbarui file CNF yang digunakan.
3	Konversi file input dengan lexer	File input akan dikonversi menjadi token-token oleh fungsi-fungsi pada file lexer.py. Pada <i>state</i> ini

		juga dilakukan penyimpanan token-token tersebut ke dalam sebuah list per baris yang nantinya akan di-parse dengan algoritma CYK.
4	Proses <i>parsing</i>	Pada <i>state</i> ini, program akan melakukan <i>parsing</i> pada tiap baris Terdapat beberapa variabel untuk mendukung program seperti, if_count, multiline, dan errorFound. ErrorFound bernilai <i>default</i> False dan akan menjadi True apabila terdapat kesalahan sintaks pada file masukan (dicek dengan algoritma cyk). Nantinya, program akan mengembalikan pesan berhasil atau gagal berdasarkan nilai terakhir pada errorFound di akhir program.

Tabel 3.6 Daftar Fungsi pada File main.py

### 3.2 Pengujian Program

File Uji	Hasil dan Analisis
<p>Pengujian input dan output sederhana</p> 	 <p>Program menampilkan pesan “Accepted!” karena program benar secara kaidah bahasa Python.</p>
<p>Pengujian Penamaan variabel</p> 	 <p>Kesalahan pada line 4 terjadi karena penamaan variabel tidak boleh diawali dengan angka, sedangkan kesalahan pada line 5 (setelah line 4 dijadikan komentar) disebabkan karena nama variabel tidak boleh mengandung spasi.</p>



### Pengujian pada percabangan

```
def do_something(x):
    '''This is a sample multiline comment
    ...

    if x == 0:
        return 0
    elif x+4 == 1:
        if True:
            return 3
        else:
            return 2
    elif x == 32:
        return 4
    else:
        return "DODODOD"'''
```

```
BFO\TBFO-PYTHON_SYNTAX_CHECKER>python main.py inputACC.py
Accepted!
Tidak ditemukan kesalahan syntax pada file
```

Program menampilkan pesan “Accepted!” karena program benar secara kaidah bahasa Python.

### Pengujian pada percabangan

```
inputReject.py > do_something
1 def do_something(x):
2     ''' This is a sample multiline comment
3     ...
4     x + 2 = 3
5     if x == 0 + 1:
6         return 0
7     elif x + 4 == 1:
8         else:
9             return 2
10    elif x == 32:
11        return 4
12    else:
13        return "Doodoo"
```

```
Syntax Error!
Terdapat kesalahan syntax pada line 4
~~~~~
def do_something(x):
    ''' This is a sample multiline comment
    ...
    x + 2 = 3
    if x == 0 + 1:
        return 0
    elif x + 4 == 1:
        else:
            return 2
    elif x == 32:
        return 4
    else:
        return "Doodoo"
~~~~~
```

Pada saat meng-*compile* file program uji, program mengembalikan pesan “Syntax Error!”. Hal tersebut karena terdapat kesalahan pada percabangan dimana terdapat ‘elif’ setelah ‘else’.

### Pengujian pada import, def, dan class

```
inputImport.py > ...
1 from flask import Flask
2 #Ini adalah
3
4 def sebuahFungsi (x):
5     x = 4
6     y = x +4
7
8 class bebas(object):
9     print("hellp")
```

```
Ketikkan nama File yang ingin dicek: inputImport.py
~~~~~
Accepted!
Tidak ditemukan kesalahan syntax pada file
~~~~~
```

Program berhasil di-*compile* karena tidak ada syntax yang salah pada program.

### Pengujian pada for loop dan while loop

```
Ketikkan nama File yang ingin dicek: inputLoop.py
~~~~~
Accepted!
Tidak ditemukan kesalahan syntax pada file
~~~~~
```

Program berhasil di-*compile* karena tidak ada syntax yang salah pada program.

<pre> inputLoop.py &gt; ... 1  ''' 2  ini komentar 3  ''' 4  #ini juga komentar 5  for i in range (5): 6      print ("s") 7 8  i = 0 9  while (i!= 5): 10     print("loop while") </pre>	
<p>Pengujian pada komentar</p> <pre> inputKomen.py 1  ''' 2  FILE ini hanya untuk komentar 3  FILE ini hanya untuk komentar 4  FILE ini hanya untuk komentar 5  FILE ini hanya untuk komentar 6  FILE ini hanya untuk komentar 7  ''' 8 9  #FILE INI HANYA UNTUK KOMENTAR 10 #FILE INI HANYA UNTUK KOMENTAR 11 #FILE INI HANYA UNTUK KOMENTAR 12 #FILE INI HANYA UNTUK KOMENTAR 13 #FILE INI HANYA UNTUK KOMENTAR </pre>	<pre> Accepted! Tidak ditemukan kesalahan syntax pada file </pre> <p>Program berhasil di-<i>compile</i> karena tidak ada syntax yang salah pada program.</p>
<p>Pengujian pada multiline comment</p> <pre> inputErrorMultiLine.py 1  while (True): 2      """ 3      hallo ini adalah pengujian 4      komentar multiline yang tidak 5      ditutup sehingga nantinya akan 6      menghasilkan kesalahan syntax 7    8    9      if (True): 10         print("hello") </pre>	<pre> Syntax Error! Terdapat kesalahan syntax pada line 2 while (True):     """     hallo ini adalah pengujian     komentar multiline yang tidak     ditutup sehingga nantinya akan     menghasilkan kesalahan syntax       if (True):         print("hello") </pre> <p>Program menampilkan pesan “Syntax Error!” dan menandakan <i>line</i> yang menjadi penyebabnya. Pada kasus ini, line 2 menjadi masalah karena multiline comment tidak ditutup.</p>
<p>Pengujian secara menyeluruh</p>	<pre> Accepted! Tidak ditemukan kesalahan syntax pada file </pre> <p>Program berhasil di-<i>compile</i> karena tidak ada syntax yang salah pada program.</p>

```
inputAll.py > ...
1  from flask import flask as flask
2
3  '''
4  Intinya ini buat tes seluruhnya
5  '''
6
7  def fungsi(x):
8      for i in range (10):
9          if (i==0):
10             print(">")
11          elif (i==1):
12             print("<")
13          elif (i==2):
14             print("==")
15          else:
16             print("tes")
17      return 0
18
19  class misal(object):
20      i = 0
21      a = 5
22      a_2 = 4
23      while (i < 5):
24          x = a +a_2
25          #ini komentar
26
```

## **BAB IV**

### **KESIMPULAN DAN SARAN**

#### **5.1 Kesimpulan**

Program yang dibuat oleh penulis dengan memanfaatkan algoritma CYK ini telah berhasil memeriksa *syntax* Python dalam skala kecil. Pemeriksaan untuk *syntax* yang lebih jauh seperti *syntactic sugar*, indentasi, karakter-karakter di luar ASCII, *Decorator*, belum dapat diperiksa dengan compiler yang telah diprogram oleh penulis. Di luar itu, terdapat pula beberapa bug pada program ini yang diperkirakan disebabkan oleh grammar CFG yang digunakan belum mencakup seluruh grammar yang digunakan oleh Python itu sendiri.

#### **5.1 Saran**

Berikut adalah saran dari penulis apabila pembaca ingin membuat program serupa

1. Melakukan eksplorasi lebih dalam terkait grammar CFG yang digunakan oleh Python.
2. Melakukan pengujian yang lebih menyeluruh untuk *case-case* yang ada untuk menghindari adanya *edge case* yang tidak ter-cover.

## REFERENSI

- [1] Bendersky, Eli. 2010. *A Generic Regex-Based Lexer/Tokenizer Tool*. <https://gist.github.com/eliben/5797351>. Diakses pada 19 November 2021
- [2] Eisele, Robert. 2021. *cyk algorithm*. <https://www.xarg.org/tools/cyk-algorithm/>. Diakses pada 21 November 2021.
- [3] GeeksforGeeks. 2020. *cyk algorithm for context free grammar*. <https://www.geeksforgeeks.org/cyk-algorithm-for-context-free-grammar/>. Diakses pada 21 November 2021.
- [4] Hopcroft, John E., Rajeev Motwani, and Jeffrey D. Ullman. "Introduction to automata theory, languages, and computation." *Acm Sigact News* 32.1 (2001)
- [5] Luthfi, Inas. 2008. "APLIKASI PROGRAM DINAMIS DALAM ALGORITMA COCKEYOUNGER-KASAMI (CYK)". <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2007-2008/Makalah2008/MakalahIF2251-2008-079.pdf>. Diakses pada 24 November 2021
- [6] Pythonmembers.club. 2018 . *Building A Lexer in Python – a Tutorial*. <https://medium.com/@pythonmembers.club/building-a-lexer-in-python-a-tutorial-3b6de161fe84>. Diakses pada 20 November 2021