

TUGAS BESAR
COMPILER BAHASA PYTHON

LAPORAN

**Diajukan sebagai salah satu tugas mata kuliah IF2124 Teori Bahasa Formal dan
Automata pada Semester I
Tahun Akademik 2021-2022**

Oleh

Fachry Dennis Heraldi	13520139
Zayd Muhammad Kawakibi Zuhri	13520144
Vito Ghifari	13520153



PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
2021

DAFTAR ISI

TEORI DASAR.....	3
Finite Automata (FA).....	3
Context Free Grammar (CFG).....	4
Chomsky Normal Form (CNF).....	5
Algoritma Cocke-Younger-Kasami (CYK).....	6
Kompilasi dalam Bahasa Pemrograman Python.....	7
HASIL	9
Hasil Analisis Finite Automata (FA)	9
Hasil Analisis Context-Free Grammar (CFG)	9
Hasil Analisis Chomsky Normal Form (CNF)	13
IMPLEMENTASI DAN PENGUJIAN.....	14
Spesifikasi Teknis Program	14
Screenshot Pengujian Kasus	17
REPOSITORY GITHUB	23
PEMBAGIAN TUGAS.....	24
KESIMPULAN, SARAN, DAN REFLEKSI.....	25
REFERENSI.....	27

TEORI DASAR

Finite Automata (FA)

Finite automata adalah mesin abstrak berupa sistem model matematika dengan masukan dan keluaran diskrit yang dapat mengenali bahasa paling sederhana (bahasa reguler). Di bidang ilmu komputer, Finite Automata sangat erat kaitannya dengan proses kompilasi program. Salah satu komponen dalam proses kompilasi adalah analisis leksikal pada bahasa pemrograman, yaitu proses penerjemahan bahasa pemrograman tingkat tinggi ke tingkat rendah agar bahasa tersebut dikenali oleh mesin. Definisi formal dari FA adalah sebagai berikut.

Sebuah finite automata dinyatakan dalam 5 tupel atau $M=(Q,\Sigma,\delta,q_0,F)$, dimana:

- Q adalah himpunan state/kedudukan
- Σ adalah himpunan simbol input/masukan/alfabet
- δ adalah fungsi transisi, fungsi yang mengambil states dan alfabet input sebagai argumen dan menghasilkan sebuah state ($Q \times \Sigma$)
- q_0 adalah state awal ($q_0 \in Q$)
- F adalah himpunan state akhir ($F \subseteq Q$)

Finite automata dimulai dari state awal (q_0), kemudian finite automata akan berpindah dari satu state ke state yang lain berdasarkan fungsi transisinya (δ), dan finite automata dikatakan menerima suatu inputan yang diberikan apabila transisi berakhir pada state akhir (F). Dapat dinyatakan bahwa M mengenali bahasa A jika $A = \{w \mid M \text{ menerima } w\}$.

Finite automata dapat dinyatakan dalam dua cara. Cara yang pertama adalah dengan menggunakan diagram transisi, yaitu berupa graf yang simpulnya merepresentasikan state dan sisinya merepresentasikan transisi dari state ke state lain berdasarkan inputan simbol. Cara yang kedua adalah dengan menggunakan tabel transisi, yaitu daftar berbentuk tabel untuk fungsi transisi yang menyatakan hubungan antara himpunan states dengan alfabet input.

Berdasarkan kemampuan mengubah statenya, FA dikelompokkan kedalam dua jenis, yaitu DFA (Deterministic Finite Automata) dan NFA (Non-Deterministic Finite Automata). Pada DFA, dari suatu state hanya ada tepat satu state berikutnya untuk setiap simbol masukan yang diterima. Sedangkan pada NFA, aturan transisi state lebih longgar, dari suatu state bisa terdapat 0, 1, atau lebih transisi dengan label input yang sama. NFA juga dapat melakukan perubahan state secara spontan tanpa input (transisi kosong) yang dikenal dengan ϵ (epsilon).

Kedua DFA dan NFA sama-sama menerima inputan yang berakhir pada state akhir, namun pada NFA mungkin saja suatu inputan dapat berakhir pada state yang bukan state akhir, tetapi jika dilakukan tracing dan inputan tersebut dapat berakhir pada state akhir juga, maka inputan tersebut diterima oleh NFA.

Context Free Grammar (CFG)

Context-free grammar (CFG) digunakan untuk menyatakan Context-free language (CFL). CFG adalah sebuah himpunan dari aturan rekursif yang digunakan untuk menghasilkan pola dari suatu string. CFG menghasilkan CFL menggunakan himpunan variabel dan aturan produksi yang didefinisikan secara rekursif. Setiap aturan produksi adalah dalam bentuk $A \rightarrow B$ di mana A adalah pemproduksi, dan B adalah hasil produksi. Batasannya hanyalah ruas kiri adalah sebuah simbol variabel. Dan pada ruas kanan bisa berupa terminal, simbol, variable ataupun ϵ . Secara formal CFG didefinisikan sebagai berikut.

Sebuah context-free grammar dapat dinyatakan dalam 4 tupel $G = (V, \Sigma, R, S)$, dimana:

- V adalah himpunan variabel berhingga (non-terminal);
- Σ adalah himpunan simbol terminal berhingga ;
- R adalah himpunan dari aturan produksi yang memetakan sebuah variabel ke suatu string s ($s \in (V \cup \Sigma)^*$);
- S adalah simbol awal.

Context Free Grammar (CFG) menjadi dasar dalam pembentukan suatu parser/proses analisis sintaksis. Bagian sintaks dalam suatu kompilator kebanyakan di definisikan dalam tata bahasa bebas konteks. Context-free grammar dapat dimodelkan sebagai suatu pohon penurunan (parse tree). Pohon penurunan (derivation tree/parse tree) berguna untuk menggambarkan simbol-simbol variabel menjadi simbol-simbol terminal setiap simbol variabel akan di turunkan menjadi terminal sampai tidak ada yang belum tergantikan. Pada pohon penurunan, node merepresentasikan simbol, edge merepresentasikan aturan produksi, dan leave merepresentasikan simbol terminal sebagai hasil akhir.

Proses penurunan / parsing bisa dilakukan dengan cara sebagai berikut :

- Penurunan terkiri (leftmost derivation): simbol variabel paling kiri yang di perluas terlebih dahulu.

- Penurunan terkanan (rightmost derivation) : simbol variabel paling kanan yang diperluas terlebih dahulu.

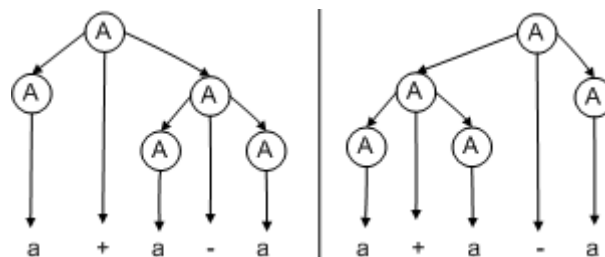
Misal : Grammar sebagai berikut :

$$A \rightarrow A+A \mid A-A \mid a$$

Untuk memperoleh string 'a+a-a' dari grammar diatas dilakukan dengan cara :

- Penurunan ter kiri: $A \Rightarrow A-A \Rightarrow A+A-A \Rightarrow a+a-a$
- Penurunan terkanan : $A \Rightarrow A+A \Rightarrow A+A-A \Rightarrow a+a-a$

Pohon penurunan dari grammar tersebut.



Karena grammar tersebut dapat diimplementasikan oleh lebih dari satu pohon penurunan dan diperoleh string yang sama, grammar tersebut mengandung ambiguitas. Grammar yang menghasilkan paling sedikit sebuah string ambigu disebut grammar ambigu.

Chomsky Normal Form (CNF)

CFG perlu disederhankan dengan tujuan untuk melakukan pembatasan sehingga tidak menghasilkan pohon penurunan yang memiliki kerumitan yang tak perlu atau aturan produksi tak berarti. Penyederhanaan CFG dapat dilakukan dengan mencari bentuk normalnya yang dikenal dengan istilah CNF (Chomsky Normal Form). CNF dapat dibuat dari sebuah tata bahasa bebas konteks yang telah mengalami penyederhanaan yaitu penghilangan produksi useless, unit, dan ϵ .

Bentuk normal Chomsky (Chomsky Normal Form, CNF) adalah Context Free Grammar (CFG) dengan setiap produksinya berbentuk : $\alpha \rightarrow \beta$ dimana α adalah 1 non-terminal dan β adalah 1 terminal atau 2 non-terminal.

Langkah-langkah pembentukan CNF secara umum sebagai berikut.

- Biarkan aturan produksi yang sudah dalam bentuk normal Chomsky

- Lakukan penggantian aturan produksi yang ruas kanannya memuat simbol terminal dan panjang ruas kanan > 1
- Lakukan penggantian aturan produksi yang ruas kanannya memuat > 2 simbol variabel
- Penggantian-penggantian tersebut bisa dilakukan berkali-kali sampai akhirnya semua aturan produksi dalam bentuk normal Chomsky
- Selama dilakukan penggantian, kemungkinan kita akan memperoleh aturan-aturan produksi baru, dan juga memunculkan simbol-simbol variabel baru

Algoritma Cocke-Younger-Kasami (CYK)

Algoritma Cocke-Younger-Kasami (CYK) merupakan algoritma parsing dan keanggotaan (membership) untuk Context-free grammar. Algoritma ini dinamakan CYK karena diciptakan oleh tiga orang programmer, J. Cocke, DH. Younger, dan T. Kasami. Syarat dari penggunaan algoritma ini adalah suatu grammar harus dalam CNF. Tujuan dari penggunaan algoritma ini untuk menunjukkan apakah suatu string dapat diperoleh dari grammar. Algoritma CYK dapat diilustrasikan dalam bentuk tabel.

Berikut ini adalah contoh apakah kalimat “she eats a fish with a fork” termasuk ke dalam grammar

$S \rightarrow NP VP$
 $VP \rightarrow VP PP$
 $VP \rightarrow V NP$
 $VP \rightarrow \text{eats}$
 $PP \rightarrow P NP$
 $NP \rightarrow \text{Det } N$
 $NP \rightarrow \text{she}$
 $V \rightarrow \text{eats}$
 $P \rightarrow \text{with}$
 $N \rightarrow \text{fish}$
 $N \rightarrow \text{fork}$
 $\text{Det} \rightarrow \text{a}$

Kalimat she eats a fish with a fork dianalisis menggunakan tabel CYK sebagai berikut.

CYK table

S						
	VP					
S						
	VP			PP		
S		NP			NP	
NP	V, VP	Det.	N	P	Det	N
she	eats	a	fish	with	a	fork

Dapat terlihat pada tabel tersebut, puncak tabel atau dapat disebut sebagai akar mengandung simbol awal (S). Dengan demikian kalimat “she eats a fish with a fork” diterima dan merupakan bagian dari grammar language tersebut.

Kompilasi dalam Bahasa Pemrograman Python

Python adalah bahasa interpreter tingkat tinggi (high-level), dan juga general-purpose. Python diciptakan oleh Guido van Rossum dan dirilis pertama kali pada tahun 1991. Filosofi desain pemrograman Python mengutamakan code readability dengan penggunaan whitespace-nya. Python adalah bahasa multiparadigma karena mengimplementasi paradigma fungsional, imperatif, berorientasi objek, dan reflektif.

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 banyak grammar dan algoritma yang dikembangkan untuk menghasilkan compiler dengan performa yang tinggi. Salah satu grammar yang dapat digunakan adalah CFG, yang dapat disederhanakan dalam bentuk formalnya, yaitu CNF. Sementara itu, CNF diaplikasikan untuk algoritma parser, yaitu algoritma CYK.

Apabila diterapkan algoritma CYK dalam melakukan parsing pada sintaks Python, tentu diperlukan CNF sebagai grammar-nya. Pada CNF terdapat kata kunci bawaan pada bahasa Python yang diimplementasikan sebagai simbol terminal. Terdapat 35 kata kunci yang terdapat dalam bahasa Python yang ditunjukkan pada tabel berikut.

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

Pada tabel di atas kata kunci yang ditulis dengan warna merah merupakan kata kunci yang tidak perlu diimplementasikan dalam Tugas Besar ini. Selain kata kunci di atas, beberapa *special character* diimplementasikan juga sebagai simbol terminal. *Special character* tersebut ditunjukkan pada tabel berikut.

(+	/	^	>	“	[
)	-	=	%	‘	!]
.	*	&	<	:	#	

Sintaks kata kunci dan *special character* di atas merupakan hal yang perlu diperhatikan selama pembuatan CFG dalam Tugas Besar ini. Sementara itu, pada nama variabel digunakan FA untuk mengevaluasi validasi dari suatu variabel. Dalam bahasa Python, nama variabel terdiri karakter alfa-numerik dan garis bawah (A-z, 0-9, dan _). Nama variabel tidak dapat dimulai oleh angka dan nama variabel bersifat *case-sensitive* (Huruf besar dan huruf kecil dibedakan walaupun membentuk kata yang sama).

HASIL

Hasil Analisis Finite Automata (FA)

Pada tugas besar ini, finite automata (FA) digunakan untuk mengevaluasi nama variabel pada program. Di dalam bahasa Python, variabel yang valid adalah variabel yang dibentuk oleh karakter lowercase a sampai z, karakter uppercase A sampai Z, angka 0-9, dan garis bawah ‘_’ dengan syarat nama variabel tidak boleh dimulai oleh angka. Berdasarkan syarat tersebut, dapat dikonstruksi FA dengan jenis NFA sebagai berikut.

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

Himpunan state $Q = \{q_0, q_1\}$

Himpunan simbol input $\Sigma = \{[a-z], [A-Z], [0-9], _ \}$

Tabel fungsi transisi (δ)

	[a-z]	[A-Z]	[0-9]	_
$\rightarrow q_0$	q_1	q_1	\emptyset	q_1
$*q_1$	q_1	q_1	q_1	q_1

State awal (Initial State) $q_0 = q_0$

State akhir (Final state) $F = \{q_1\}$

atau dapat dinyatakan juga bahwa $w = \{ w \mid M \text{ menerima } w \text{ yang merupakan string berawalan } a-z, A-Z, \text{ atau } _ \text{ kemudian diikuti oleh } a-z, A-Z, _, \text{ atau } 0-9 \}$.

Hasil Analisis Context-Free Grammar (CFG)

Pada tugas besar ini, Context-Free Grammar (CFG) digunakan untuk mengevaluasi sintaksis program secara keseluruhan. Deskripsi CFG yang dibuat adalah sebagai berikut.

$$G = (V, \Sigma, R, S)$$

Tabel Simbol Non-Terminal/Variabel (V)

S	COMMENT	FOR	LIST	RETURN
S1	CONTINUE	FUNCTION	LISTELMT	STRCONTENT
SFUNCTION	DEF	IF	METHOD	STRING
SLOOP	ELIF	IFFUNCTION	MULSTR	VAL
ARGUMENT	ELIFFUNCTION	IFLOOP	NL	VARIABLE
ASSIGNMENT	ELIFLOOP	IMPORT	NONE	WHILE
BOOLEAN	ELSE	IMPORTAS	OPERATOR	WITH
BREAK	ELSEFUNCTION	IN	ONELINER	
BRCON	ELSELOOP	INSTANCE	PASS	
CLASS	EXPRESSION	IS	RAISE	

Tabel Simbol Terminal (Σ)

false	else	or	.	<
none	for	pass	+	>
true	from	raise	-	'
and	if	return	*	:
as	import	strcontent	**	"
break	in	variable	/	!
class	is	while	=	#
continue	newline	with	&	[
def	not	(^]
elif	number)	%	

Aturan Produksi (R)

S -> S newline S | NL | S1

S1 -> S1 newline S1 | IMPORT | IMPORTAS | CLASS | DEF | FUNCTION | FUNCTION NL | STRING | COMMENT | RAISE | WHILE | FOR | VARIABLE ASSIGNMENT VARIABLE NL | VARIABLE ASSIGNMENT EXPRESSION NL | IF | VARIABLE | VARIABLE ASSIGNMENT VARIABLE | VARIABLE ASSIGNMENT EXPRESSION | INSTANCE | METHOD | IS | PASS | LIST | ONELINER

SFUNCTION -> S1 | SFUNCTION newline SFUNCTION | RETURN | IFFUNCTION | newline SFUNCTION

SLOOP -> S1 | SLOOP newline SLOOP | BRCON | IFLOOP | newline SLOOP

ARGUMENT -> VARIABLE | INSTANCE | METHOD | FUNCTION | VARIABLE , ARGUMENT | INSTANCE , ARGUMENT | METHOD , ARGUMENT | FUNCTION , ARGUMENT

ASSIGNMENT -> + = | - = | * = | / = | * * = | / / = | & = | ^ = | % = | =

BOOLEAN -> true | false

BREAK -> break NL | break

BRCON -> BREAK | CONTINUE

CLASS -> class VARIABLE : newline S1 | class VARIABLE (INSTANCE) : newline S1

COMMENT -> # STRCONTENT NL | " " " MULSTR " " " NL | ' ' ' MULSTR ' ' ' NL | " " " MULSTR " " " | ' ' ' MULSTR ' ' ' NL

CONTINUE -> continue NL | continue

DEF -> def FUNCTION : newline SFUNCTION | def FUNCTION : newline S1 RETURN

ELIF -> elif VARIABLE : newline S1 | elif (VARIABLE) : newline S1 | ELIF ELIF | ELIF ELSE | ELIF IN

ELIFFUNCTION -> elif VARIABLE : S RETURN | elif (VARIABLE) : S RETURN | elif VARIABLE : SFUNCTION | elif (VARIABLE) : SFUNCTION | elif VARIABLE : NL RETURN | elif (VARIABLE) : NL RETURN | ELIFFUNCTION ELSEFUNCTION | ELIF ELSEFUNCTION | ELIF ELIFFUNCTION | ELIF | ELIF IFFUNCTION | ELIFFUNCTION IF | ELIFFUNCTION IFFUNCTION

ELIFLOOP -> elif VARIABLE : S BRCON | elif (VARIABLE) : S BRCON | elif VARIABLE : SLOOP | elif (VARIABLE) : SLOOP | elif VARIABLE : NL BRCON | elif (VARIABLE) : NL BRCON | ELIFLOOP ELSELOOP | ELIF ELSELOOP | ELIF ELIFLOOP | ELIF | ELIF IFLOOP | ELIFLOOP IF | ELIFLOOP IFLOOP

ELSE -> else : newline S1

ELSEFUNCTION -> else : S RETURN | else : NL RETURN | else : SFUNCTION

ELSELOOP -> else : S BRCON | else : NL BRCON | else : SLOOP

EXPRESSION -> VARIABLE OPERATOR VARIABLE | VARIABLE OPERATOR EXPRESSION

FOR -> for VARIABLE in FUNCTION : newline S1 | for VARIABLE in METHOD : newline S1 | for VARIABLE in VARIABLE : newline S1 | FOR FOR | FOR IN | FOR BREAK | FOR CONTINUE | for VARIABLE in FUNCTION : newline SLOOP | for VARIABLE in METHOD : newline SLOOP | for VARIABLE in VARIABLE : newline SLOOP

FUNCTION -> VARIABLE (ARGUMENT) | VARIABLE () | VARIABLE (FUNCTION)

IF -> if VARIABLE : newline S1 | if (VARIABLE) : newline S1 | IF ELIF | IF ELSE | IF IN | IF IS | if VARIABLE : newline IF | if (VARIABLE) : newline IF | IF IF

IFFUNCTION -> IF | if VARIABLE : S RETURN | if (VARIABLE) : S RETURN | if VARIABLE : NL RETURN | if (VARIABLE) : NL RETURN | if VARIABLE : SFUNCTION | if (VARIABLE) : SFUNCTION | IFFUNCTION IFFUNCTION | IFFUNCTION ELIFFUNCTION | IFFUNCTION ELSE | IF IFFUNCTION | IF ELIFFUNCTION | IF ELSEFUNCTION | IFFUNCTION IF | IFFUNCTION ELIF | IFFUNCTION ELSE

IFLOOP -> IF | if VARIABLE : S BRCON | if (VARIABLE) : S BRCON | if VARIABLE : NL BRCON | if (VARIABLE) : NL BRCON | if VARIABLE : SLOOP | if (VARIABLE) : SLOOP | IFLOOP IFLOOP | IFLOOP ELIFLOOP | IFLOOP ELSE | IF IFLOOP | IF ELIFLOOP | IF ELSELOOP | IFLOOP IF | IFLOOP ELIF | IFLOOP ELSE

IMPORT -> from INSTANCE import VARIABLE NL | import VARIABLE NL

IMPORTAS -> from INSTANCE import VARIABLE as VARIABLE NL | import INSTANCE as VARIABLE NL

IN -> VARIABLE in VARIABLE | VARIABLE not in VARIABLE

INSTANCE -> variable | variable . INSTANCE

IS -> VARIABLE is VARIABLE | VARIABLE is not VARIABLE

LIST -> [] | [LISTELMT] | [LISTELMT ,]

LISTELMT -> LISTELMT , LISTELMT | VARIABLE | EXPRESSION

METHOD -> variable . METHOD | variable . FUNCTION

MULSTR -> newline | newline STRCONTENT | STRCONTENT newline | STRCONTENT newline MULSTR

NL -> newline | newline S

NONE -> none

OPERATOR -> + | - | * | / | % | * * | / / | & | ^ | and | or | not | > | < | > = | < = | = = | ! =

ONELINER -> VARIABLE if VARIABLE else VARIABLE | variable ASSIGNMENT VARIABLE if VARIABLE else VARIABLE | variable ASSIGNMENT VARIABLE if VARIABLE else variable ASSIGNMENT VARIABLE | VARIABLE if VARIABLE else variable ASSIGNMENT VARIABLE

PASS -> pass NL | pass

RAISE -> raise VARIABLE NL | raise FUNCTION NL

RETURN -> return | return VARIABLE NL | return VARIABLE | newline RETURN | RETURN RETURN

STRCONTENT -> strcontent | STRCONTENT STRCONTENT | VARIABLE | VAL

STRING -> ' STRCONTENT ' | " STRCONTENT " | ' ' | " "

VAL -> number | number OPERATOR VAL | NONE | (VAL) | FUNCTION | METHOD | ONELINER

VARIABLE -> variable | variable OPERATOR VARIABLE | variable OPERATOR VAL | (VARIABLE) | BOOLEAN | STRING | VAL | INSTANCE | VARIABLE OPERATOR VARIABLE | IN | IS | LIST

WHILE -> while VARIABLE : NL S | while (VARIABLE) : NL S | WHILE IS : NL S | WHILE IN : NL S | while VARIABLE: NL BREAK | while (VARIABLE) : NL BREAK | while VARIABLE : NL SLOOP | while (VARIABLE) : NL SLOOP

WITH -> with FUNCTION as VARIABLE : newline S1

Simbol Awal (S) = S

Hasil Analisis Chomsky Normal Form (CNF)

Sebelum diolah kedalam algoritma CYK, CFG perlu disederhanakan terlebih dahulu ke dalam bentuk formalnya. Untuk menyederhanakan CFG ke dalam bentuk CNF, kelompok kami menggunakan converter CFG to CNF pada pranala <https://cyberzhg.github.io/toolbox/cfg2cnf>

IMPLEMENTASI DAN PENGUJIAN

Spesifikasi Teknis Program

cfg.txt dan cnf.txt

Pada file cfg.txt terdapat informasi Terminals, Variables, dan Productions. Terminals merujuk kepada himpunan simbol terminal yang ditulis dalam huruf lowercase, Variables merujuk kepada himpunan simbol non-terminal/variabel yang ditulis dalam huruf uppercase, dan Productions merujuk kepada aturan produksi yang berlaku. Adapun penulisan yang diterapkan untuk menyatakan satu aturan produksi adalah variabel terletak paling kiri kemudian dipisahkan oleh “ -> ” dan sebelah kanan dari “ -> ” adalah hasil produksi. Aturan produksi yang memiliki lebih dari satu kemungkinan hasil produksi dipisahkan oleh “ | ”. Tiap satu aturan produksi dipisahkan oleh garis baru. Tujuan penulisan ini agar dapat disesuaikan dengan aturan input CFG yang diterima oleh converter CFG to CNF. Setelah proses konversi berhasil, hasil konversi disimpan pada file cnf.txt dan siap untuk diakses pada file cyk.py.

nfa.py

Untuk mengimplementasikan NFA sebagai validator nama variabel, dibentuklah program dengan nama file nfa.py. Struktur program dari nfa.py adalah sebagai berikut.

Struktur	Penjelasan
<code>(variable) nfa : dict [int, dict[int, str]]</code>	State dari NFA didefinisikan menggunakan dictionary yang tujuannya seperti fungsi transisi. Key dari dictionary menyatakan state, value dari dictionary adalah sebuah dictionary yang key-nya menyatakan state dan valuenya menyatakan inputan simbol.
<code>def accepts(transitions, initial, accepting, s)</code>	Fungsi accepts menerima argumen transitions (dictionary nfa), initial (state awal = 0), accepting (state akhir = 1), dan s (string nama variabel). Fungsi akan mengembalikan nilai boolean True jika nama variabel valid dan False jika nama variabel tidak valid.

tokenizer.py

Sebelum diolah dengan CYK, akan dilakukan proses tokenize sintaks pada program Python. Tokenizer bertugas untuk memecah dan mengevaluasi jenis sintaks yang terdapat pada program kemudian disimpan hasilnya dalam suatu list. Struktur program dari tokenizer.py adalah sebagai berikut.

Struktur	Penjelasan
<code>def filter_unnecessary_tokens(tokens: list) -> list</code>	Fungsi <code>filter_unnecessary_tokens</code> menerima argumen list token kemudian melakukan proses cleaning dengan menghapus string kosong dan whitespace pada list token. Hasil cleaning dikembalikan dalam bentuk list.
<code>def file_tokenizer(file_name: str) -> list</code>	Fungsi <code>file_tokenizer</code> menerima nama file program yang akan di-tokenize. Fungsi akan melakukan proses pembacaan tiap sintaks pada program dan dipecah sebagai token, token tersebut disimpan dalam list, dan akan diolah oleh fungsi <code>filter_unnecessary_tokens</code> untuk dilakukan cleaning. Fungsi akan mengembalikan hasil tokenize yang sudah siap untuk diproses pada algoritma CYK.

cyk.py

File program yang telah berhasil di-tokenize siap untuk diproses oleh algoritma CYK. File hasil tokenize akan diperlakukan sebagai suatu string inputan dengan file CNF sebagai referensi grammar yang akan menentukan string inputan diterima oleh language atau tidak. String inputan diterima oleh language menandakan bahwa program tersebut sintaksnya sudah sesuai dan lulus proses compile.

Struktur	Penjelasan
<code>def get_cnf(file_path)</code>	Fungsi <code>get_cnf</code> menerima argumen path dari suatu file <code>cnf.txt</code> berada, kemudian dilakukan proses parsing dari <code>cnf.txt</code> dan hasilnya

	disimpan dalam bentuk dictionary. Fungsi ini akan mengembalikan dictionary tersebut untuk digunakan dalam algoritma CYK.
<code>def clean_tokenized(tokenized)</code>	Fungsi <code>clean_tokenized</code> menerima argumen list yang berisi token dari program yang telah berhasil di-tokenize, kemudian akan dilakukan proses cleaning penulisan token yang menjadi terminal agar sesuai dengan CNF yang telah dibuat. Fungsi akan mengembalikan list yang sudah clean dan sesuai dengan CNF.
<code>def cyk_parse(raw_tokenized, tokenized, grammar)</code>	Fungsi <code>cyk_parse</code> menerima argumen <code>tokenized</code> (list dari token program) dan <code>grammar</code> (dictionary dari CNF) kemudian akan dilakukan proses pengisian tabel sesuai dengan algoritma CYK. Fungsi akan mengembalikan nilai boolean <code>True</code> jika terdapat simbol awal 'S' pada puncak tabel dan mengembalikan nilai <code>False</code> jika sebaliknya. Fungsi ini juga mengembalikan angka line yang mengandung kesalahan syntax dengan menelusuri tabel dan juga <code>raw_tokenized</code> . Fungsi juga menampilkan persentase progress algoritma dan elapsed time.

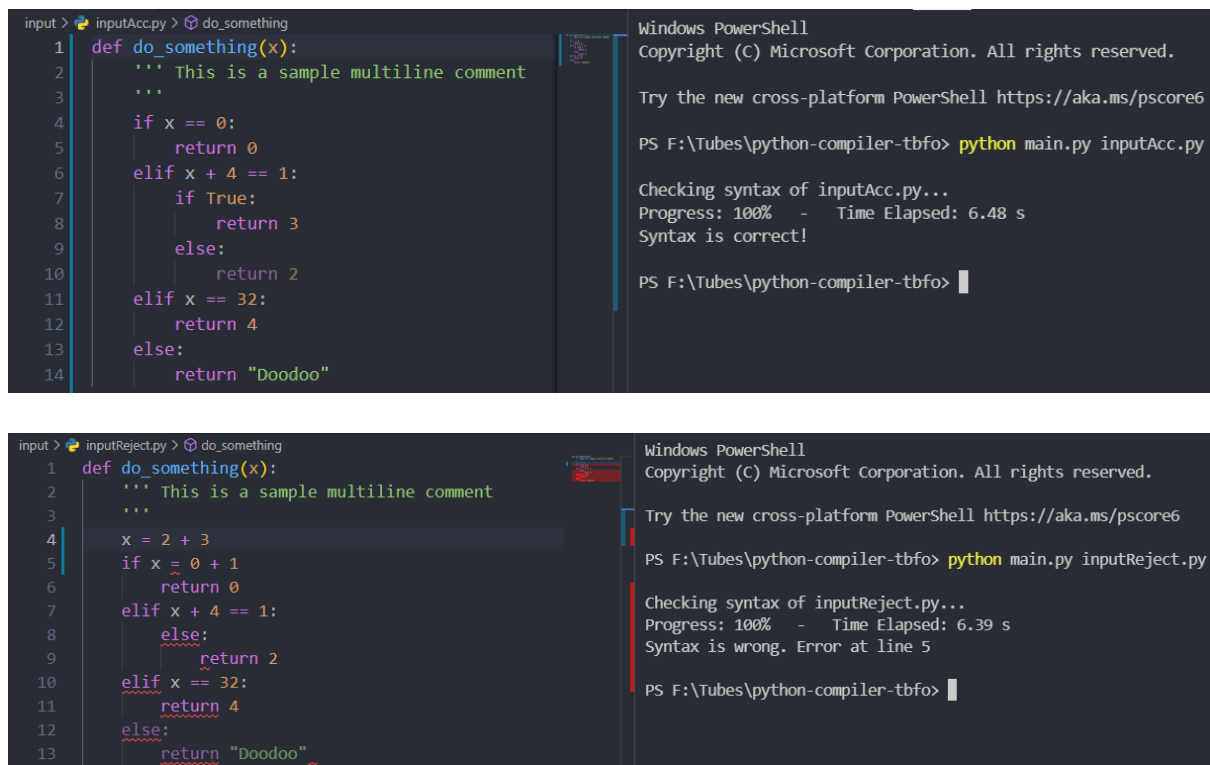
main.py

File `main.py` merupakan file driver sebagai program utama dari keseluruhan program yang telah dibuat. Pada `main.py` terdapat argument parser agar program dapat dieksekusi melalui command line dengan menambahkan argumen file yang ingin dievaluasi.

Screenshot Pengujian Kasus

Berikut ini adalah *screenshot* yang menunjukkan kode program Python yang dievaluasi beserta hasil evaluasi sintaksnya. Pengujian kasus dapat menggambarkan kebenaran program compiler bahasa Python yang telah kelompok kami buat. Kami mengambil beberapa kasus uji yang dapat mewakili seluruh kata kunci dalam bahasa Python yang perlu dievaluasi sesuai dengan spesifikasi tugas ini.

Kasus Uji 1



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

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS F:\Tubes\python-compiler-tbfo> python main.py inputAcc.py

Checking syntax of inputAcc.py...
Progress: 100% - Time Elapsed: 6.48 s
Syntax is correct!

PS F:\Tubes\python-compiler-tbfo>

input > 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"

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS F:\Tubes\python-compiler-tbfo> python main.py inputReject.py

Checking syntax of inputReject.py...
Progress: 100% - Time Elapsed: 6.39 s
Syntax is wrong. Error at line 5

PS F:\Tubes\python-compiler-tbfo>
```

Kasus uji diatas menguji sintaks fungsi (def) sederhana dan sintaks percabangan (if-else). *Screenshot* di atas menunjukkan pada sintaks def dan if-else yang benar program akan menampilkan pesan “Syntax is correct” sedangkan pada sintaks yang salah program akan menampilkan pesan “Syntax is wrong” dan lokasi baris yang salah yaitu terletak pada baris ke 5. “if x = 0 + 1” merupakan sintaks yang salah karena setelah kata kunci if, seharusnya diikuti oleh ekspresi logika yang bernilai True atau False.

Kasus Uji 2

```
input > test.py > guacamole
1 import numpy as np
2 from scipy import inv
3
4 def guacamole(x):
5     # testing comments
6     if x == 0:
7         x += 1
8     elif x + 4 == 1:
9         if True:
10             foo()
11         else:
12             x = y / 2
13     elif x == 32:
14         func(foo)
15     else:
16         print("guacamole")

PS C:\Users\vitos\Files\Kuliah\Sem3\tbfo\tubes\python-compiler-tbfo> python
main.py test.py

Checking syntax of test.py...
Progress: 100% - Time Elapsed: 8.8 s
Syntax is correct!

PS C:\Users\vitos\Files\Kuliah\Sem3\tbfo\tubes\python-compiler-tbfo> █
```

```
ishmash.py 1, U test.py 4, M X
input > test.py > guacamole
1 import numpy as np
2 from scipy import inv
3
4 def guacamole(x):
5     # testing comments
6     if x == 0:
7         x += 1
8     elif x + 4 == 1:
9         if True:
10             foo()
11         else:
12             x = y / 2
13     elif x == 32:
14         func(
15     else:
16         print("guacamole")

PROBLEMS 5 OUTPUT TERMINAL DEBUG CONSOLE

PS C:\Users\vitos\Files\Kuliah\Sem3\tbfo\tub
main.py test.py

Checking syntax of test.py...
Progress: 100% - Time Elapsed: 8.85 s
Syntax is correct!

PS C:\Users\vitos\Files\Kuliah\Sem3\tbfo\tub
main.py test.py

Checking syntax of test.py...
Progress: 100% - Time Elapsed: 8.08 s
Syntax is wrong. Error at line 14

PS C:\Users\vitos\Files\Kuliah\Sem3\tbfo\tub
```

Kasus uji diatas menampilkan bahwa penulisan fungsi yang tidak tepat akan memberikan pesan “Syntax is wrong”. penulisan fungsi “func(“ kurang tepat karena seharusnya fungsi ditulis dengan lengkap menjadi “func(foo)”.

Kasus Uji 3

```
input > oop.py > Car > __init__
1 class Vehicle:
2     def __init__(self, color, doors, tires, vtype):
3         self.color = color
4         self.doors = doors
5         self.tires = tires
6         self.vtype = vtype
7     def brake(self):
8         print("Braking!")
9
10 class Car(Vehicle):
11     def __init__(self, color, doors, tires, vtype):
12         pass

PS C:\Users\vitos\Files\Kuliah\Sem3\tbfo\tubes\
python-compiler-tbfo> python main.py oop.py

Checking syntax of oop.py...
Progress: 100% - Time Elapsed: 9.33 s
Syntax is correct!

PS C:\Users\vitos\Files\Kuliah\Sem3\tbfo\tubes\
python-compiler-tbfo> █
```

```

input > oop.py > Vehicle > __init__
1 class Vehicle:
2     def __init__(self, color, doors, tires, vtype):
3         self.color = color
4         self.doors = doors
5         self.tires = tires
6         self.vtype = vtype
7     def brake(self):
8         print("Braking!")
9
10 class Car(Vehicle):
11     def __init__(self, color, doors, tires, vtype):
12         pass
    
```

```

PS C:\Users\vit0s\Files\Kuliah\Sem3\tbfo\tubes\python-compiler-tbfo> python main.py oop.py

Checking syntax of oop.py...
Progress: 100% - Time Elapsed: 8.88 s
Syntax is wrong. Error at line 6

PS C:\Users\vit0s\Files\Kuliah\Sem3\tbfo\tubes\python-compiler-tbfo>
    
```

```

input > oop.py
1 class :
2     def __init__(self, color, doors, tires, vtype):
3         self.color = color
4         self.doors = doors
5         self.tires = tires
6         self.vtype = vtype
7     def brake(self):
8         print("Braking!")
9
10 class Car(Vehicle):
11     def __init__(self, color, doors, tires, vtype):
12         pass
    
```

```

PS C:\Users\vit0s\Files\Kuliah\Sem3\tbfo\tubes\python-compiler-tbfo> python main.py oop.py

Checking syntax of oop.py...
Progress: 100% - Time Elapsed: 8.9 s
Syntax is wrong. Error at line 1

PS C:\Users\vit0s\Files\Kuliah\Sem3\tbfo\tubes\python-compiler-tbfo>
    
```

Pada kasus 3 ini, ditampilkan contoh program Python yang menggunakan OOP (*Object-Oriented Programming*). Pada gambar pertama, ditunjukkan kasus yang berhasil. Pada gambar kedua, ada satu masalah yang terdapat pada baris ke-6, yaitu kesalahan dalam melakukan *assign* pada atribut kelas. Sementara itu, pada gambar ketiga, kesalahan berada di definisi kelas (kelas tidak mempunyai nama).

Kasus Uji 4

```

input > lists.py > ...
1 a = 5
2 li = [1, 2, 3, 4, 5, 6]
3 if a in li:
4     print("a is in the list")
5
6 while 10 not in li:
7     li.append(a)
8     a += 1
    
```

```

PS C:\Users\vit0s\Files\Kuliah\Sem3\tbfo\tubes\python-compiler-tbfo> python main.py lists.py

Checking syntax of lists.py...
Progress: 100% - Time Elapsed: 2.96 s
Syntax is correct!

PS C:\Users\vit0s\Files\Kuliah\Sem3\tbfo\tubes\python-compiler-tbfo>
    
```

```

input > lists.py > ...
1 a = 5
2 li = [1, 2, 3, 4, 5, 6
3 if a in li:
4     print("a is in the list")
5
6 while 10 not in li:
7     li.append(a)
8     a += 1
    
```

```

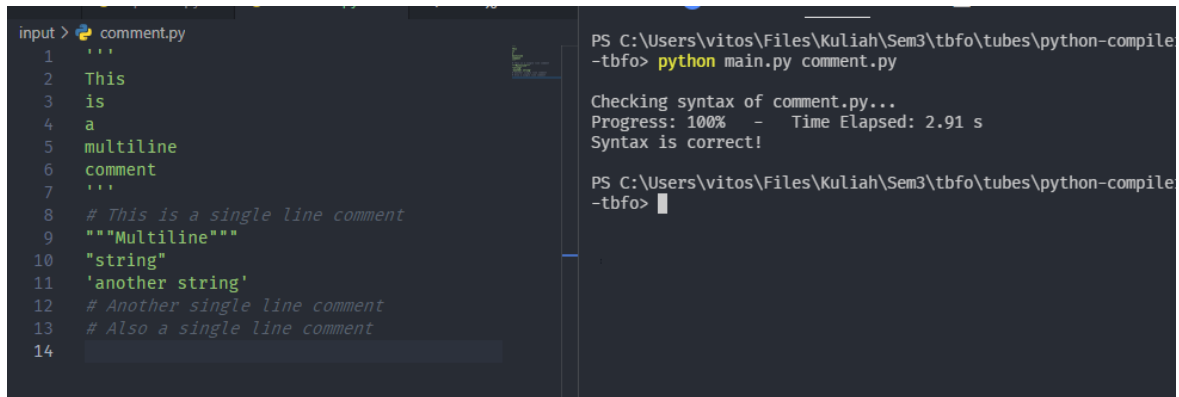
PS C:\Users\vit0s\Files\Kuliah\Sem3\tbfo\tubes\python-compiler-tbfo> python main.py lists.py

Checking syntax of lists.py...
Progress: 100% - Time Elapsed: 2.78 s
Syntax is wrong. Error at line 2

PS C:\Users\vit0s\Files\Kuliah\Sem3\tbfo\tubes\python-compiler-tbfo>
    
```

Kasus keempat merupakan *test case compiler* terhadap list dan *in*. Gambar pertama adalah contoh penggunaan list yang benar pada program Python. Sementara itu, deklarasi *list* pada gambar kedua tidak mempunyai *closing bracket* (]).

Kasus Uji 5

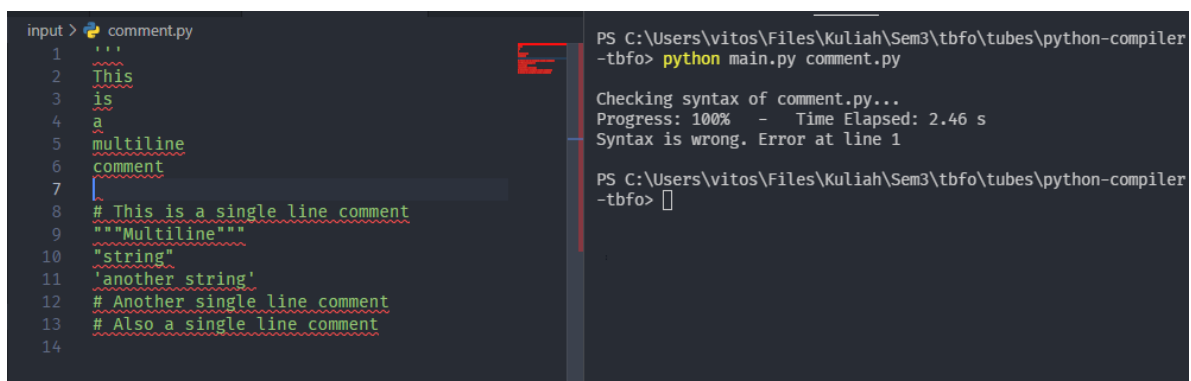


```
input > comment.py
1 '''
2 This
3 is
4 a
5 multiline
6 comment
7 '''
8 # This is a single line comment
9 """Multiline"""
10 "string"
11 'another string'
12 # Another single line comment
13 # Also a single line comment
14

PS C:\Users\vitos\Files\Kuliah\Sem3\tbfo\tubes\python-compile
-tbfo> python main.py comment.py

Checking syntax of comment.py...
Progress: 100% - Time Elapsed: 2.91 s
Syntax is correct!

PS C:\Users\vitos\Files\Kuliah\Sem3\tbfo\tubes\python-compile
-tbfo>
```



```
input > comment.py
1 '''
2 This
3 is
4 a
5 multiline
6 comment
7 [
8 # This is a single line comment
9 """Multiline"""
10 "string"
11 'another string'
12 # Another single line comment
13 # Also a single line comment
14

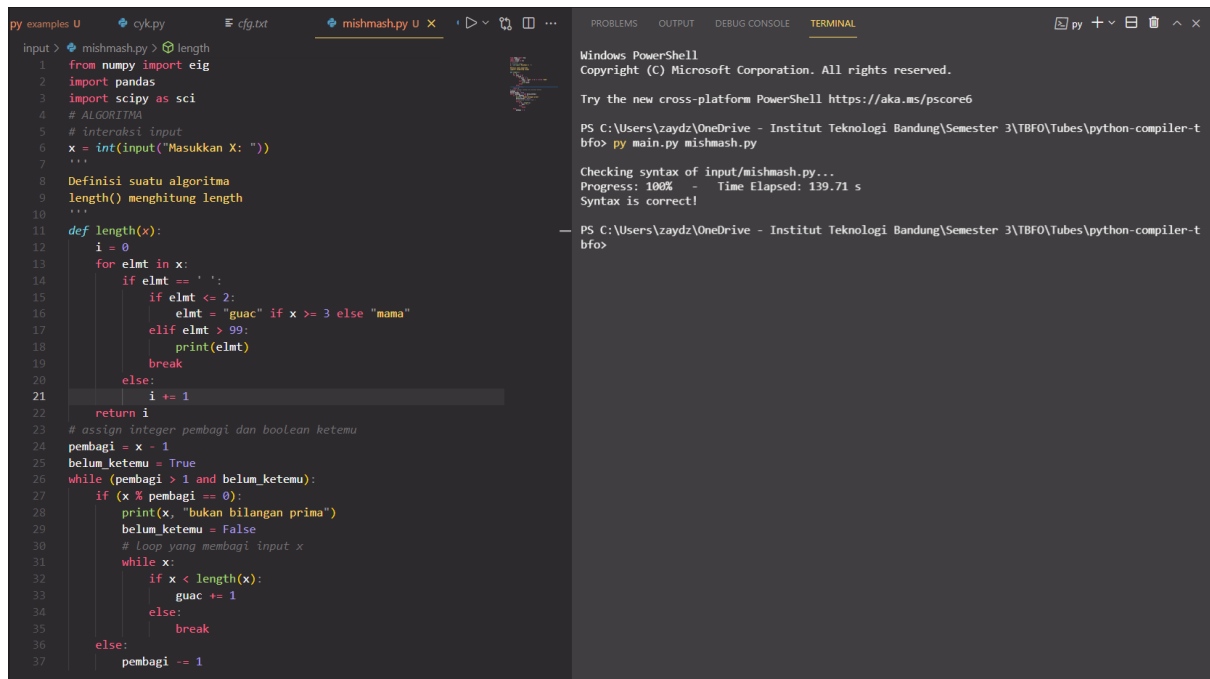
PS C:\Users\vitos\Files\Kuliah\Sem3\tbfo\tubes\python-compiler
-tbfo> python main.py comment.py

Checking syntax of comment.py...
Progress: 100% - Time Elapsed: 2.46 s
Syntax is wrong. Error at line 1

PS C:\Users\vitos\Files\Kuliah\Sem3\tbfo\tubes\python-compiler
-tbfo>
```

Kasus ini menguji kemampuan untuk mengecek syntax comment dan string. Terdapat 2 macam comment dalam python, multiline dan oneline. String juga dapat dinyatakan dengan “ atau ‘. Semua dapat dihandle dalam compiler kami. Dalam screenshot kedua multiline comment pertama tidak dilengkapi “” sebagai penutup, sehingga syntax tidak sesuai. Indikator error menunjuk ke line satu karena di sana lah mulainya multiline comment.

Kasus Uji 6



```
py examples U  cyk.py  cfg.txt  mishmash.py U x  ...  PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

input > mishmash.py > length
1 from numpy import eig
2 import pandas
3 import scipy as sci
4 # ALGORITMA
5 # interaksi input
6 x = int(input("Masukkan X: "))
7 ...
8 Definisi suatu algoritma
9 length() menghitung length
10 ...
11 def length(x):
12     i = 0
13     for elmt in x:
14         if elmt == ' ':
15             if elmt <= 2:
16                 elmt = "guac" if x >= 3 else "mama"
17             elif elmt > 99:
18                 print(elmt)
19                 break
20         else:
21             i += 1
22     return i
23 # assign integer pembagi dan boolean ketemu
24 pembagi = x - 1
25 belum_ketemu = True
26 while (pembagi > 1 and belum_ketemu):
27     if (x % pembagi == 0):
28         print(x, "bukan bilangan prima")
29         belum_ketemu = False
30         # Loop yang membagi input x
31         while x:
32             if x < length(x):
33                 guac += 1
34             else:
35                 break
36     else:
37         pembagi -= 1

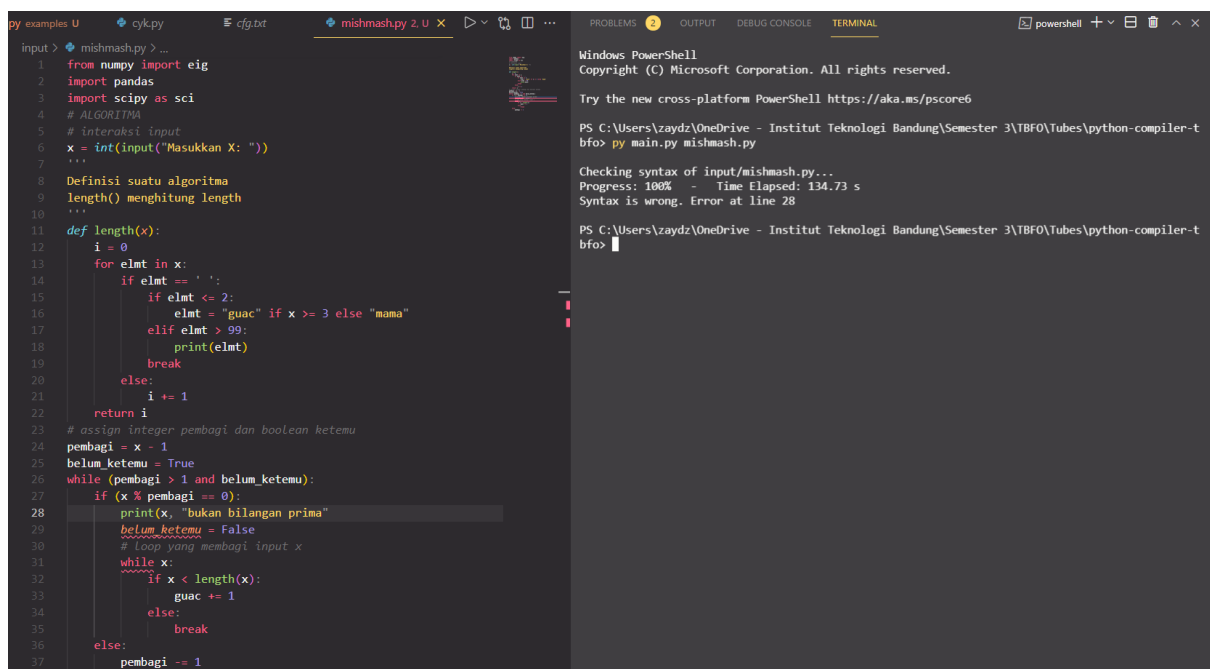
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\zaydz\OneDrive - Institut Teknologi Bandung\Semester 3\TBFO\Tubes\python-compiler-tbfo> py main.py mishmash.py

Checking syntax of input/mishmash.py...
Progress: 100% - Time Elapsed: 139.71 s
Syntax is correct!

PS C:\Users\zaydz\OneDrive - Institut Teknologi Bandung\Semester 3\TBFO\Tubes\python-compiler-tbfo>
```



```
py examples U  cyk.py  cfg.txt  mishmash.py 2 U x  ...  PROBLEMS 2  OUTPUT  DEBUG CONSOLE  TERMINAL

input > mishmash.py > ...
1 from numpy import eig
2 import pandas
3 import scipy as sci
4 # ALGORITMA
5 # interaksi input
6 x = int(input("Masukkan X: "))
7 ...
8 Definisi suatu algoritma
9 length() menghitung length
10 ...
11 def length(x):
12     i = 0
13     for elmt in x:
14         if elmt == ' ':
15             if elmt <= 2:
16                 elmt = "guac" if x >= 3 else "mama"
17             elif elmt > 99:
18                 print(elmt)
19                 break
20         else:
21             i += 1
22     return i
23 # assign integer pembagi dan boolean ketemu
24 pembagi = x - 1
25 belum_ketemu = True
26 while (pembagi > 1 and belum_ketemu):
27     if (x % pembagi == 0):
28         print(x, "bukan bilangan prima")
29         belum_ketemu = False
30         # Loop yang membagi input x
31         while x:
32             if x < length(x):
33                 guac += 1
34             else:
35                 break
36     else:
37         pembagi -= 1

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\zaydz\OneDrive - Institut Teknologi Bandung\Semester 3\TBFO\Tubes\python-compiler-tbfo> py main.py mishmash.py

Checking syntax of input/mishmash.py...
Progress: 100% - Time Elapsed: 134.73 s
Syntax is wrong. Error at line 28

PS C:\Users\zaydz\OneDrive - Institut Teknologi Bandung\Semester 3\TBFO\Tubes\python-compiler-tbfo>
```

Kasus ini menguji banyak bentuk syntax sekaligus. Terdapat banyak fungsi, loop, dan conditional yang nested, yang cukup sulit untuk dicek syntaxnya. Namun, kasus ini pun dapat di-handle. Kesalahan syntax fungsi tidak lengkap dalam screenshot 2 mellihatkan bahwa dalam banyak nested juga dapat dideteksi kesalahan syntaxnya.

REPOSITORY GITHUB

Link repository GitHub

<https://github.com/zaydzuhi/python-compiler-tbfo>

PEMBAGIAN TUGAS

Nama	NIM	Tugas
Fachry Dennis Heraldi	13520139	Membuat FA untuk validator variabel, menambah production rule pada CFG, menginisiasi laporan
Zayd Muhammad Kawakibi Zuhri	13520144	Membuat reader CNF, membuat algoritma CYK untuk parsing, menambah production rule pada CFG
Vito Ghifari	13520153	Membuat tokenizer, Menginisiasi Terminal, Variable, dan Productions utama pada CFG

KESIMPULAN, SARAN, DAN REFLEKSI

Kesimpulan

Berdasarkan hasil, implementasi, dan pengujian program yang telah dilakukan, kelompok kami mendapatkan kesimpulan sebagai berikut.

1. Telah dibuat suatu program compiler bahasa Python dengan menggunakan konsep Context-Free Grammar (CFG), Chomsky Normal Form (CNF), algoritma Cocke-Young-Kasami (CYK), dan Finite Automata (FA).
2. Sintaks kode dalam bahasa Python dapat dievaluasi kebenarannya menggunakan algoritma CYK dengan grammar CFG yang kemudian disederhanakan dalam bentuk formalnya menjadi CNF, namun tidak semua bentuk sintaks dapat dievaluasi menggunakan cara ini.

Saran

Saran yang dapat kelompok kami berikan sebagai *room of improvement* adalah sebagai berikut.

1. Lebih banyak menggali referensi terkait compiler bahasa Python
2. Alokasikan dan efektifkan waktu sebaik mungkin dalam pengerjaan
3. Lakukan debugging program langkah demi langkah sehingga berbagai macam kasus uji dapat ditangani oleh program.

Refleksi

Dalam proses pengerjaan tugas besar ini tentu kami menemukan beberapa kendala, namun dengan kendala tersebut kami banyak belajar dalam mencari solusi untuk menghadapinya. Adapun hal-hal yang menjadi perhatian bagi kami sebagai berikut.

1. Dibutuhkan waktu yang banyak untuk memahami spesifikasi tugas. Terutama di bagian CFG karena banyak kemungkinan aturan produksi yang perlu ditambahkan seiring dilakukannya debugging pada program.
2. Dengan tugas besar ini, kami lebih memahami bagaimana CFG, CNF, algoritma CYK, dan FA bekerja. Semakin banyak debugging yang dilakukan pada CFG, semakin paham bagaimana cara membaca dan mengkonstruksi suatu CFG.

3. Spesifikasi tugas yang bertambah dari *sheets* QnA sehingga diperlukan revisi pada bagian-bagian tertentu di program.

REFERENSI

https://docs.google.com/document/d/1Fd8wLOP_GzJ66atpw1yK1_S1dLCFQcKFTgnePFHq17Y/edit

<http://library.binus.ac.id/eColls/eThesisdoc/Bab2/2011-2-00004-MTIF%20Bab2001.pdf>

<https://youtube.com/playlist?list=PL32QSDw4KKwjYZEnwlc34riJzWIZE7-ux>

<https://brilliant.org/wiki/context-free-grammars/>

https://en.wikipedia.org/wiki/CYK_algorithm

<https://www.geeksforgeeks.org/cyk-algorithm-for-context-free-grammar/>

<https://web.cs.ucdavis.edu/~rogaway/classes/120/winter12/CYK.pdf>

<https://cyberzhg.github.io/toolbox/cfg2cnf>