LAPORAN TUGAS BESAR

“Compiler Bahasa Python”

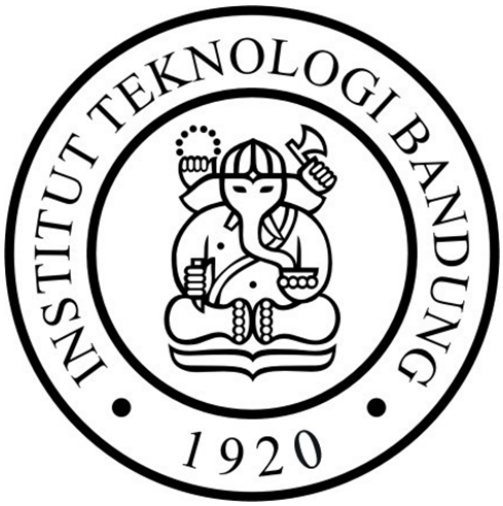
Laporan Ini Dibuat Untuk Memenuhi Tugas Perkuliahan

Mata Kuliah Teori Bahasa Formal dan Automata (IF2124)

KELAS 02

Dosen : Rizal Dwi Prayogo, S.Si., M.Si., M.Sc.

Dra. Harlili, M.Sc.



DISUSUN OLEH:

Kelompok 13

Anggota:

Michael Marcellus Herman Kahari (13520057)

Kristo Abdi Wiguna (13520058)

Jason Kanggara (13520080)

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

SEMESTER I TAHUN 2021-2022

Daftar Isi

Daftar Tabel

Daftar Gambar

BAB I
Deskripsi Masalah

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 sangat banyak *grammar* dan algoritma yang dikembangkan untuk menghasilkan *compiler* dengan performa yang tinggi. Terdapat CFG, CNF^{-e}, CNF^{+e}, 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*.

Pada tugas besar ini, implementasikanlah *compiler* untuk Python untuk *statement-statement* dan sintaks-sintaks bawaan Python. Gunakanlah konsep CFG untuk pengerjaan compiler yang mengevaluasi syntax program. Untuk nama variabel dalam program, gunakanlah FA.

Algoritma yang dipakai dibebaskan, namun tim asisten menyarankan menggunakan algoritma CYK (Cocke-Younger-Kasami). Algoritma CYK harus menggunakan *grammar* CNF (Chomsky Normal Form) sebagai *grammar* masukannya. Oleh karena itu, jika ingin menggunakan CYK buatlah terlebih dahulu *grammar* dalam CFG (Context Free Grammar), kemudian konversikan *grammar* CFG tersebut ke *grammar* CNF.

Berikut adalah daftar kata kunci bawaan Python yang harus terdaftar dalam *grammar* (yang dicoret tidak perlu diimplementasi).

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

BAB II

Teori Dasar

2.1. Finite Automata

Finite Automata (FA) memiliki definisi formal, yaitu FA merupakan list dari 5 komponen, yaitu kumpulan state, input , aturan perpindahan, state awal, dan state akhir. FA dapat didefinisikan sebagai berikut,

Definisi 2.1.1 Sebuah finite automata terdiri dari lima komponen (Q, Σ, δ, q0, F), di mana :

1. Q adalah himpunan set berhingga yang disebut dengan himpunan states.

2. Σ adalah himpunan berhingga alfabet dari simbol .

3. δ : Q × Σ adalah fungsi transisi, merupakan fungsi yang mengambil states dan alfabet input sebagai argumen dan menghasilkan sebuah state. Fungsi transisi sering dilambangkan dengan δ.

4. q0 Q adalah states awal.

5. F ⊆ Q adalah himpunan states akhir.

Dalam definisi 2.1.1, pada FA terdapat suatu F yang melambangkan himpunan state penerima. Jika suatu string membawa state FA dari suatu state inisial ke state lainnya dan berakhir pada state tersebut, maka dapat dikatakan bahwa string tersebut diterima sebagai anggota bahasa tersebut.

Sementara itu, menurut Hopcroft et al. (2017), definisi dari FA dapat dinyatakan sebagai berikut,

Definisi 2.1.2 Suatu finite automata M = (Q,Σ,δ,q0,F) akan menerima sebuah string w jika kumpulan states r0r1 · · · rn dalam Q memenuhi tiga kondisi :

1. r0 = q0.

2. δ(ri , wi+1) = ri+1 untuk i = 0, · · · , n – 1.

3. rn F.

dengan w = w1w2...wn adalah string masing-masing wi adalah anggota alphabet Σ.

Berdasarkan definisi 2.1.2, kondisi pertama menunjukkan bahwa FA dimulai dari *start state.* Pada kondisi yang kedua dinyatakan bahwa FA akan berpindah dari satu state ke state lainnya berdasarkan fungsi transisi. Kondisi yang ketiga menyatakan bahwa FA akan menerima string sebagai anggota bahasa tersebut apabila string itu berakhir di *final state.* Dapat dinyatakan bahwa M mengenali bahasa A jika A = {w | M menerima w}.

Dalam definisi 2.1.1, pada suatu finite automata terdapat F yaitu himpunan state penerima. Apabila suatu string membawa state FA dari state inisial ke salah satu state dalam F dan berakhir pada state tersebut maka string tersebut diterima sebagai anggota bahasa tersebut.

Bahasa reguler dapat dikenalkan dari FA dan menurut definisi formalnya dapat didefinisikan sebagai berikut,

Definisi 2.1.3 FA sebagai pengenal bahasa reguler secara formal dapat didefinisikan :

1. Suatu M = (Q,Σ,δ,q0,F) merupakan suatu FA. Suatu string x dikatakan diterima oleh M jika δ(q0, x) F . Jika suatu string tidak diterima, maka string itu dikatakan ditolak oleh M .

*2. Suatu bahasa yang diterima atau diterima oleh M adalah himpunan : L(M) = {x Σ * |x diterima M }*

dengan L adalah bahasa pada Σ dan L diterima oleh M jika dan hanya jika L=L(M).

Suatu bahasa L pada σ adalah bahasa reguler jika dan hanya jika terdapat suatu finite automata yang mengenal L. Untuk setiap M , sembarang FA, terdapat suatu ekspresi reguler yang berkaitan dengan L(M) dan untuk ekspresi reguler tersebut terdapat suatu finite automata yang mengenali bahasa tersebut.

2.2. Context-Free Grammar

Context Free Grammar (CFG) adalah suatu tata bahasa yang memiliki tujuan sama seperti tata bahasa regular yaitu suatu tatanan untuk menunjukkan bagaimana menghasilkan suatu untai-untai dalam sebuah bahasa.

CFG dapat didefinisikan sebagai berikut,

Grammar = (S, T, P, V), dengan

S = Start symbol

T = Himpunan terbatas terminal

P = Himpunan terbatas produksi

V = Himpunan terbatas variabel

Context Free Grammar adalah suatu tata bahasa dimana tidak ada pembatasan pada hasil produksi. Contoh pada aturan produksi :

$$\alpha \rightarrow \beta$$

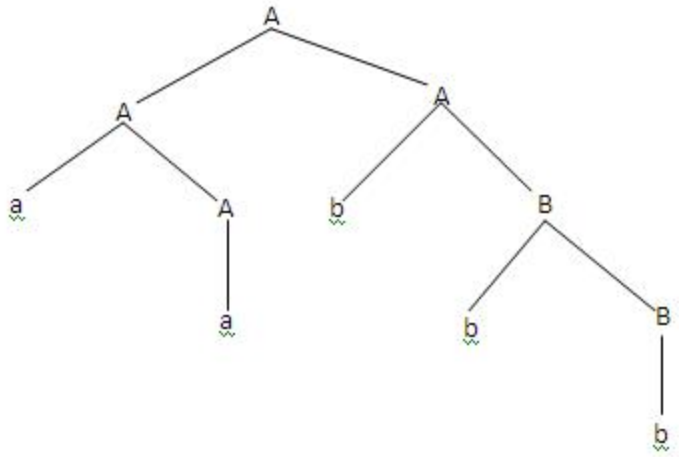
batasannya hanyalah ruas kiri (α) adalah sebuah simbol variabel. Sedangkan contoh aturan produksi yang termasuk CFG adalah seperti di bawah :

- $B \rightarrow CDeFg$
- $D \rightarrow BcDe$

Context Free Grammar menjadi dasar dalam pembentukan suatu parser/proses analisis sintaksis. Bagian sintaks dalam suatu kompilator kebanyakan didefinisikan dalam tata bahasa bebas konteks. Pohon penurunan (*derivation tree*/*parse tree*) berguna untuk menggambarkan simbol-simbol variabel menjadi simbol-simbol terminal setiap simbol variabel akan diturunkan menjadi terminal sampai tidak ada yang belum tergantikan. Contoh, terdapat CFG dengan aturan produksi sebagai berikut dengan simbol awal S :

- $S \rightarrow AB$
- $A \rightarrow aA \mid a$
- $B \rightarrow bB \mid b$

Maka jika ingin dicari gambar *pohon penurunan* dengan string : ‘aabbb’ hasilnya adalah seperti di bawah :



Gambar 2.1 Parse Tree

Proses penurunan / parsing bisa dilakukan dengan cara sebagai berikut :

- Penurunan terkiri (*leftmost derivation*): simbol variabel terkiri yang diperluas terlebih dahulu.
- Penurunan terkanan (*rightmost derivation*) : simbol variabel terkanan yang diperluas terlebih dahulu.

Misal : Grammar sbb :

- S → aAS | a
- A → SbA | ba

Untuk memperoleh string ‘aabbaa’ dari grammar diatas dilakukan dengan cara :

- Penurunan terkiri: S => aAS => aSbAS => aabAS => aabbaS => aabbaa
- Penurunan terkanan : S => aAS => aAa => aSbAa => aAbbaa => aabbaa

Ambiguitas terjadi bila terdapat lebih dari satu pohon penurunan yang berbeda untuk memperoleh suatu string. Misalkan terdapat tata bahasa sebagai berikut :

- $S \rightarrow A \mid B$
- $A \rightarrow a$
- $B \rightarrow a$

Untuk memperoleh untai ‘a’ bisa terdapat dua cara penurunan sebagai berikut :

- S => A => a
- S => B => a

Sebuah string yang mempunyai lebih dari satu pohon sintaks disebut *string ambigu* (*ambiguous*). Grammar yang menghasilkan paling sedikit sebuah string ambigu disebut *grammar ambigu*.

2.3. 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 ε. Dengan kata lain, suatu tata bahasa bebas konteks dapat dibuat menjadi bentuk normal Chomsky dengan syarat tata bahasa bebas konteks tersebu t:

- Tidak memiliki produksi useless
- Tidak memiliki produksi unit
- Tidak memiliki produksi ε

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

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

2.4. Cocke-Younger Kasami

Cocke-Younger-Kasami (CYK) Algorithm adalah salah satu algoritma parsing untuk Context-Free Grammar yang sudah dikonversi ke bentuk Chomsky Normal Form. Tujuan algoritma CYK ini adalah untuk membership testing, atau menunjukkan apakah suatu string dapat diterima atau bagian dari language CFG tersebut. CYK merupakan salah satu algoritma paling efektif dan memiliki pendekatan *dynamic programming*.

Proses parsing untai dengan algoritma CYK memanfaatkan struktur data sebuah array dua dimensi dan merupakan aplikasi Pemrograman Dinamis (Program Dinamis) karena proses parsing memanfaatkan hasil parsing sebelumnya untuk memutuskan apakah proses yang sedang berlangsung dapat diterima maupun tidak.

2.5. Bahasa Pemrograman Python

Python adalah bahasa pemrograman interpretatif multiguna dengan filosofi perancangan yang berfokus pada tingkat keterbacaan kode. Python diklaim sebagai bahasa yang menggabungkan kapabilitas, kemampuan, dengan sintaksis kode yang sangat jelas, dan dilengkapi dengan fungsionalitas pustaka standar yang besar serta komprehensif. Python juga didukung oleh komunitas yang besar.

Python mendukung multi paradigma pemrograman, utamanya; namun tidak dibatasi pada pemrograman berorientasi objek, pemrograman imperatif, dan pemrograman fungsional. Salah satu fitur yang tersedia pada python adalah sebagai bahasa pemrograman dinamis yang dilengkapi dengan manajemen memori otomatis. Seperti halnya pada bahasa pemrograman dinamis lainnya, python umumnya digunakan sebagai bahasa skrip meski pada praktiknya penggunaan bahasa ini lebih luas mencakup konteks pemanfaatan yang umumnya tidak dilakukan dengan menggunakan bahasa skrip. Python dapat digunakan untuk berbagai keperluan pengembangan perangkat lunak dan dapat berjalan di berbagai platform sistem operasi. Beberapa fitur yang dimiliki Python adalah:

- memiliki kepustakaan yang luas
- memiliki tata bahasa yang jernih dan mudah dipelajari

- memiliki aturan layout *source code* yang memudahkan pengecekan, pembacaan kembali dan penulisan ulang kode sumber
- *object-oriented*
- bersifat modular, mudah dikembangkan dengan menciptakan modul-modul baru; modul-modul tersebut dapat dibangun dengan bahasa Python maupun C/C++
- memiliki fasilitas pengumpulan sampah otomatis
- memiliki banyak fasilitas pendukung sehingga mudah digunakan

Dalam bahasa Python, perlu diperhatikan juga bagaimana penulisan syntax yang benar selama pembuatan program agar tidak mendapatkan syntax error saat program dijalankan. Contohnya adalah indentasi yang benar, penulis method yang benar, penempatan operator yang benar, dan masih banyak lagi. Berikut beberapa contoh implementasi program python yang akan memunculkan syntax error:

```
--- Tidak menggunakan def sebelum myfunction
myfunction(x,y):
     return x + y

--- Tidak ada if statement
else:
     print(“Hello World”)

--- Tidak ada colon setelah if statement
if x > 10
     print(“x diatas 10”)

--- Penulisan else statement yang salah
if x > 10:
     print(“x diatas 10”)
esle:
     print(“x dibawah 10”)
```

```
--- Tidak terdapat indentasi setelah if statement
if x > 10:
print(“x diatas 10”)
```

BAB III

Hasil FA dan CFG

3.1. Produksi CFG

Terminal (T) dan Non Terminal / Variabel (V)

Terminal CFG yang kami buat terdiri dari kata kunci, operator, dan tanda baca. Terminal merupakan karakter yang membentuk string yang dihasilkan kemudian akan diperiksa menggunakan algoritma CYK apakah merupakan bagian dari bahasa tertentu. Non terminal CFG dibentuk merupakan set bahasa Python.

NON-TERMINAL -> TERMINAL

IF -> a

ELIF -> b

ELSE -> c

FOR -> d

IN -> e

WHILE -> f

PASS -> h

CLASS -> j

DEF -> k

NAME -> x

NUMBER_CTN -> y

ALL -> *

AS -> m

IMPORT -> n

FROM -> o

RAISE -> p

AND -> q

OR -> r

NOT -> s

IS -> t

TRUE -> u

FALSE -> v

NONE -> w

QUOTE -> ‘

DOUBLEQUOTE -> “

COLON -> :

COMMA -> ,

DOT -> .

PLUS -> +

MINUS -> -

MULTIPLY -> *

SQR_BKT_OPN -> [

SQR_BKT_CLS ->]

CRL_BKT_OPN -> {

CRL_BKT_CLS -> }

NRM_BKT_OPN -> (

NRM_BKT_CLS ->)

Start Symbol : S

S yaitu start symbol grammar yang memproses kalimat - kalimat yang dibuat dalam Python.

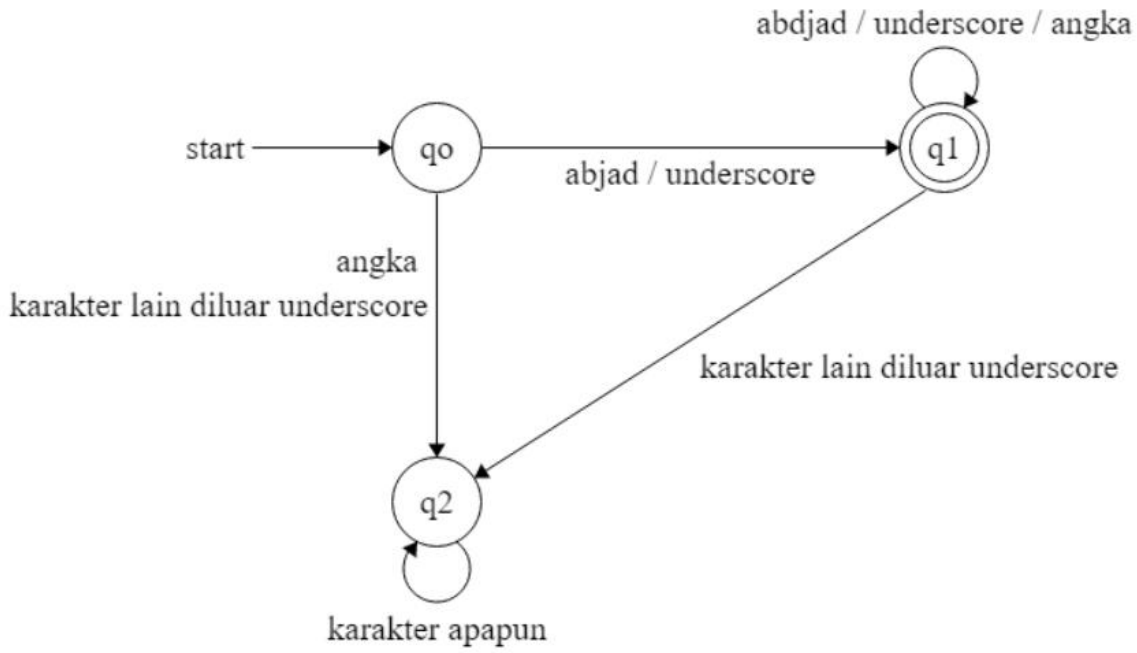
3.2. CNF (Chomsky Normal Form)

CNF adalah CFG telah disederhanakan dan disesuaikan dengan aturan dari CNF yaitu tidak adanya useless production, unit production, dan epsilon production. Produksi CNF hanya bisa menghasilkan dua variabel dan atau menghasilkan terminal. Konversi dilakukan oleh script dimulai dengan langkah binarization untuk mempermudah iterasi saat pengecekan aturan untuk pengeliminasian. Jumlah aturan produksi serta jumlah variabel dalam CNF akan lebih banyak dari pada CFG.

Berikut hasil CNF kelompok kami pada link berikut ini.

https://docs.google.com/document/d/1hgjIeehfZAY_-En8ye-MSoZCHcFOqHQIVP3v9N4zirQ/edit?usp=sharing

3.3. Hasil FA



Gambar 3.1 Hasil Finite Automata

Untuk mengecek tiap expression apakah valid atau tidak, digunakan Finite Automation untuk mengeceknya. Finite Automation akan mengecek apakah expression tersebut diawali dengan huruf abjad atau underscore. Jika diawali dengan abjad atau underscore, expression tersebut dapat dianggap valid selama proses menerima input karakternya, inputan tersebut bukanlah karakter lain di luar underscore seperti “!” atau “<”. Akan tetapi, jika input diawali dengan angka atau karakter lain di luar underscore, maka state akan berpindah ke q2 dimana q2 merupakan death state.

Contoh dari hasil pengecekan state yang dilakukan adalah sebagai berikut:

1. _123abc = Valid
2. Abc_123 = Valid
3. Abc% = Invalid
4. 432_1 = Invalid

BAB IV

Implementasi dan Pengujian

Program kami memiliki tiga program utama yaitu grammarconverter.py, cyk_parser.py, dan lexer.py. File grammarconverter.py melakukan konversi CFG menjadi CNF yang hasilnya disimpan di dalam file cnf.txt. File cyk_parser.py mencocokkan bahasa masukan file yang diuji dengan CNF yang telah didapat dari grammarconverter.py. File lexer.py yang digunakan untuk membaca masukan file yang menggunakan bahasa tertentu dan kemudian dijadikan token untuk diperiksa.

Pada tugas besar ini, kami mendeklarasikan tiga buah file utama Python untuk compiler yang kami buat, yang terdiri dari

1. Grammarconverter.py

   Kami memanfaatkan beberapa library untuk program ini, yaitu:

   - copy
   - string

   Tabel 1. Deskripsi fungsi / prosedur pada program grammarconverter.py

Fungsi / Prosedur	Deskripsi
`def isNonTerminal(elem)`	Fungsi untuk mengecek elem dan mengembalikan true jika elem adalah non terminal atau variabel
`def removeUnit(CFG)`	Fungsi ini menerima sebuah dictionary CFG yang masih utuh lalu menghapus semua unit production yang ada pada grammar dan mengembalikan CFG yang telah direduksi
`def getCFGFile(filename)`	Fungsi ini menerima path filename lalu memproses file dengan membaca isi file dan memasukkan ke dalam sebuah dictionary CFG yang akan direturn.
`def convertToCNF(CFG)`	Fungsi ini menerima dictionary CFG lalu mengonversi CFG menjadi Chomsky Normal Form dan mengembalikan dictionary yang diterima yang telah

	diupdate atau dikonversi
`def display(GRAMMAR_DICT)`	Fungsi ini menerima dictionary grammar untuk menampilkan pada layar grammar yang ada
`def CNFtoFile(CNF)`	Fungsi ini menerima dictionary CNF untuk membuat sebuah file di sistem OS untuk write hasil cnf.txt
`def getCNF(filename)`	Fungsi ini adalah fungsi utama yang menerima path dan mengembalikan hasil CNF

Tabel 4.1 Fungsi/Prosedur dan Deskripsi Program Grammar Converter

2. cyk_parser.py

Tabel 2. Deskripsi fungsi / prosedur pada program cyk_parser.py

Fungsi / Prosedur	Deskripsi
`def CYKParser(input, CNF, src)`	Fungsi CYKParser menerima tiga parameter, yaitu input yang berupa string dari file yang akan diperiksa, CNF yang merupakan hasil konversi dari CFG menjadi CNF, dan src yang merupakan file keseluruhan yang tidak di proses. Di dalamnya terdapat algoritma CYK serta bagian untuk mengeluarkan hasil setelah melakukan pemeriksaan apakah diterima atau terdapat syntax error.

Tabel 4.2 Fungsi/Prosedur dan Deskripsi Program CYK Parser

3. main.py

Kami memanfaatkan beberapa library bawaan maupun buatan kami untuk program ini, yaitu:

- sys
- cyk_parser
- grammarconverter

Tabel 5. Deskripsi fungsi / prosedur pada program main.py

Fungsi / Prosedur	Deskripsi
`def checkFiniteAutomata(inp)`	Fungsi ini akan menerima input berupa teks yang didapatkan dari file berekstensi .py. Setiap teks akan dipisah masing-masing menjadi line yang berbeda. Setiap line akan dicek melalui if else yang berbeda dan akan mereturn hasil berupa x jika expression merupakan string yang valid, y jika expression adalah angka yang valid, dan R jika expression merupakan expression yang invalid. Komentar dan isi di dalam data bertipe string (“”) akan diabaikan.
`def readFile(path)`	Fungsi ini berguna untuk membaca file dari sebuah path dan mengembalikan isinya.
`def header()`	Fungsi ini menampilkan header text art untuk menandai awal program

Tabel 4.3 Fungsi/Prosedur dan Deskripsi Program Main

DATA TEST

1. Test inputAcc.py

inputAcc.py

```
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 "DooDoo"
```

Gambar 4.1 Test inputAcc.py

Result compiler :

Gambar 4.2 Hasil Test inputAcc.py

2. Test inputReject.py

inputReject.py

```
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"
```

Gambar 4.3 Test inputReject.py

Result compiler :

```
D:\Tubes\TBFO (main -> origin)
$ python main.py

 ____  _  _____ _   _  ___  _   _    ____ ___  __  __ ____ ___ _     _____ ____
|  _ \| |_   _| | | |/ _ \| \ | |  / ___/ _ \|  \/  |  _ \_ _| |   | ____|  _ \
| |_) | | | | | |_| | | | |  \| | | |  | | | | |\/| | |_) | || |   |  _| | |_) |
|  __/| |_| | |  _  | |_| | |\  | | |__| |_| | |  | |  __/| || |___| |___|  _ <
|_|   |_____|_|_| |_|\___/|_| \_|  \____\___/|_|  |_|_|  |___|_____|_____|_| \_\

                                BY MICHAEL, KRISTO, AND JASON

Masukkan nama file: inputReject.py
Compiling inputReject.py...

Menunggu hasil compile...


%%%%%%%%%%%%%%%%%%%%%%%%% RESULT %%%%%%%%%%%%%%%%%%%%%%%%%
x + 2 = 3
Syntax Error
Terjadi kesalahan ekspresi pada line 4

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Gambar 4.4 Hasil Test inputAcc.py

3. Test testAcc1.py

testAcc1.py

```python
def foo(namaorang):
    print("Halo, " + namaorang + ". Semangat tubesnya!")

while (True) :
    nama = input("Masukkan nama Anda : ")
    if (nama != ""):
        foo(nama)
    else :
        break
```

Gambar 4.5 Test testAcc1.py

Result compiler :

```
D:\Tubes\TBFO (main -> origin)
$ python main.py
PYTHON COMPILER
BY MICHAEL, KRISTO, AND JASON

Masukkan nama file: testAcc1.py
Compiling testAcc1.py...

Menunggu hasil compile...

%%%%%%%%%%%%%%%%%%%%%%%%%% RESULT %%%%%%%%%%%%%%%%%%%%%%%%%%
Accepted. No errors detected.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Gambar 4.6 Hasil Test testAcc1.py

4. Test testReject1.py

testReject1.py

```
import numpy as num
importasd pandas as pd

for i in range():
    # ini comment
    while :
        if 1 > 0 :
            break;
        elif x < x:
            pass
        else :
            print(ini print')
```

Gambar 4.7 Test testReject1.py

Result compiler :

```
D:\Tubes\TBFO (main -> origin)
$ python main.py

PYTHON COMPILER
BY MICHAEL, KRISTO, AND JASON

Masukkan nama file: testReject1.py
Compiling testReject1.py...

Menunggu hasil compile...


%%%%%%%%%%%%%%%%%%%%%%%%%% RESULT %%%%%%%%%%%%%%%%%%%%%%%%%%
importasd pandas as pd
Syntax Error
Terjadi kesalahan ekspresi pada line 2

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Gambar 4.8 Hasil Test testReject1.py

5. Test testAcc2.py

testAcc2.py

```python
import math

def addition(x,y):
    return x+y

a = 5
b = 3
result = addition(a,b)

print(result)
```

Gambar 4.9 Test testAcc2.py

Result compiler :

Gambar 4.10 Hasil Test testAcc2.py

6. Test testReject2.py

testReject2.py

```
suatu_angka = 5

if x == 5:
  print("x adalah lima")
elif x > 5
  print("x diatas lima")
  else:
    print(Adalah sesuatu')
elsee:
  print("x dibawah lima")
  for i range(5):
    print(i)
```

Gambar 4.11 Test testReject2.py

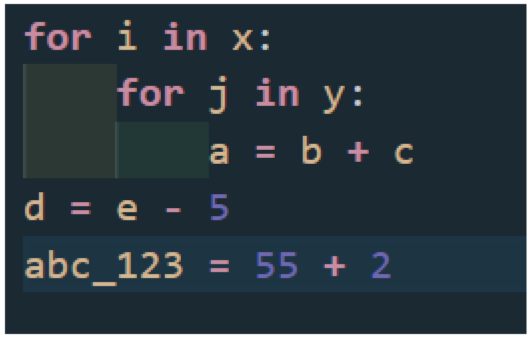
Result compiler :

```
D:\Tubes\TBFO (main -> origin)
$ python main.py

 ____  _  _____ _   _  ___  _   _    ____ ___  __  __ ____ ___ _     _____ ____
|  _ \| |_   _| | | |/ _ \| \ | |  / ___/ _ \|  \/  |  _ \_ _| |   | ____|  _ \
| |_) | | | | | |_| | | | |  \| | | |  | | | | |\/| | |_) | || |   |  _| | |_) |
|  __/| |_| | |  _  | |_| | |\  | | |__| |_| | |  | |  __/| || |___| |___|  _ <
|_|   |_____|_|_| |_|\___/|_| \_|  \____\___/|_|  |_|_|  |___|_____|_____|_| \_\

                    BY MICHAEL, KRISTO, AND JASON

Masukkan nama file: testReject2.py
Compiling testReject2.py...

Menunggu hasil compile...


%%%%%%%%%%%%%%%%%%%%%%%%% RESULT %%%%%%%%%%%%%%%%%%%%%%%%%
elsee:
Syntax Error
Terjadi kesalahan ekspresi pada line 9

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Gambar 4.12 Hasil Test testReject2.py

7. Test testAcc3.py

testAcc3.py



Gambar 4.13 Test testAcc3.py

Result compiler :

Gambar 4.14 Hasil Test testAcc3.py

8. Test testReject3.py

testReject3.py

```
for i in (x):
    for j in (y):
        a = b + c + 99
1____ = 55 + 2
```

Gambar 4.15 Test testReject3.py

Result compiler :

```
D:\Tubes\TBFO (main -> origin)
$ python main.py

PYTHON COMPILER
BY MICHAEL, KRISTO, AND JASON

Masukkan nama file: testReject3.py
Compiling testReject3.py...

Menunggu hasil compile...


%%%%%%%%%%%%%%%%%%%%%%%%%% RESULT %%%%%%%%%%%%%%%%%%%%%%%%%%
1____ = 55 + 2
Syntax Error
Terjadi kesalahan ekspresi pada line 4

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Gambar 4.16 Hasil Test testReject3.py

BAB V

Lampiran dan Pembagian Tugas

5.1. Lampiran

https://github.com/pandora-1/TBFO

5.2. Pembagian Tugas

Tabel 1. Pembagian Tugas

No.	Program yang dihandle	Penanggung jawab
1.	Grammar, Grammar Converter, Laporan	Kristo
2.	CYK Parser, Laporan	Jason
3.	Finite Automata, Laporan	Michael

Daftar Referensi

- Cohen, Shay. 2018. Chart Parsing: the CYK Algorithm, https://www.inf.ed.ac.uk/teaching/courses/inf2a/slides2018/inf2a_L20_slides.pdf, diakses pada 17 November 2021.
- Dean, Naufal. https://github.com/naufal-dean/Context-Free-Grammar-Python-Compiler, diakses pada 15 November 2021.
- Dubey, Aayush. A program in Python to demonstrate Finite Automata, https://www.codespeedy.com/a-program-in-python-to-demonstrate-finite-automata/, diakses pada 23 November 2021.
- Kurniawan, Ridian. 2011. Context Free Grammar (CFG), https://fairuzelsaid.wordpress.com/2011/06/16/tbo-context-free-grammar-cfg/, diakses pada 19 November 2021.
- Simon, Samuel. 2008. *Dynamic Programming Algorithm To Determine How Context-Free Grammar Can Generate A String*. Bandung: Institut Teknologi Bandung.