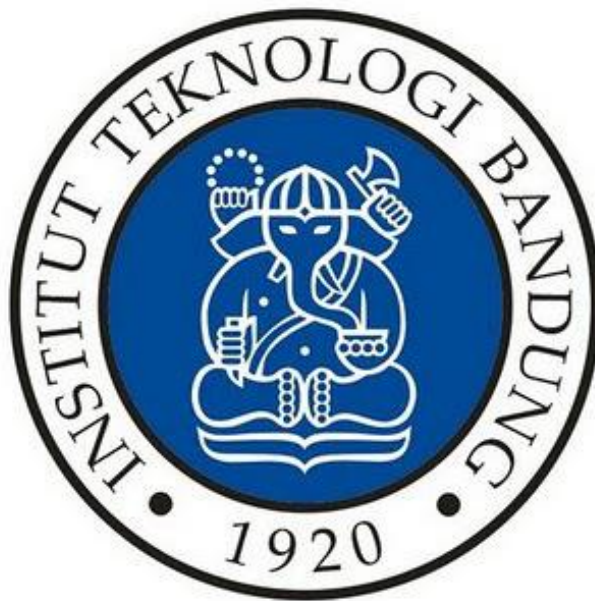


IF2124 Teori Bahasa Formal dan Otomata

Laporan Tugas Besar

Compiler Bahasa Python



Kelompok : compile error

Gede Sumerta Yoga - 13320021

Bryan Bernigen - 13520034

Ng Kyle - 13520040

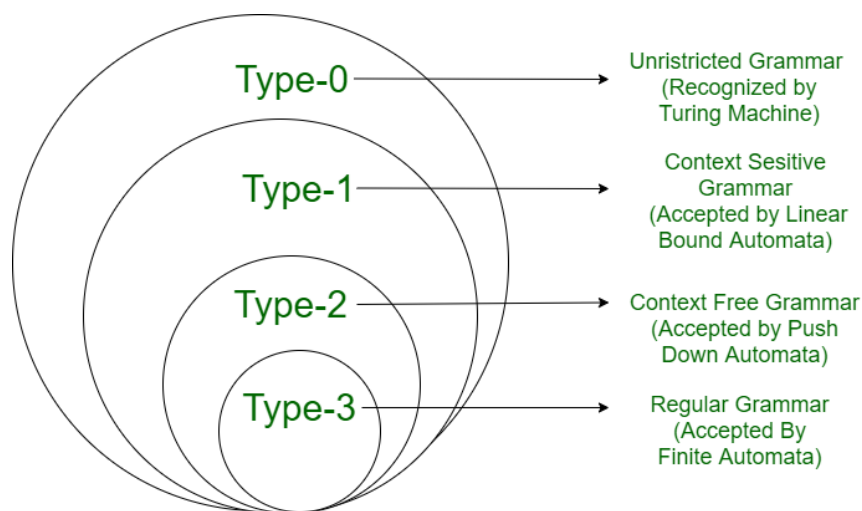
**PROGRAM STUDI INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2021**

I. Dasar Teori

1.1 Automata Theory

Teori Automata merupakan cabang ilmu yang mempelajari logika komputasi yang berkaitan dengan “mesin” / perangkat komputasi abstrak. Dalam pembahasan teori automata, dikenali automaton yaitu model abstrak dari mesin yang melalui kumpulan “state” yang berubah berdasarkan fungsi transisi yang ditentukan berdasarkan kumpulan masukan simbol, sehingga mesin dapat berubah “state”. Kumpulan masukan (input) yang merubah “state” mesin akan menjalankan mesin berdasarkan fungsi transisi dan pada akhirnya menghasilkan suatu kondisi “accept” (diterima) untuk “state” atau kondisi mesin yang telah ditentukan. Berdasarkan lingkup bahasa yang dapat, automaton dibagi beberapa jenis secara hierarkis kemampuan sebuah automaton memeriksa himpunan bahasa / ruang lingkup bahasa yang dapat diperiksa atau dijalankan automaton tersebut, disebut juga *Chomsky Hierarchy*. Berikut merupakan jenis automaton dan tipe bahasa yang dapat diterimanya :

Automaton Family	Type	Grammar (Language)
Finite Automata	Type-3	Regular Grammar
Push Down Automata	Type-2	Context Free Grammar
Linear Bound Automata	Type-1	Context Sensitive Grammar
Turing Machine	Type-0	Unrestricted Grammar



Gambar 1.1 Chomsky Hierarchy

Automaton secara formal didefinisikan terdapat 4 aspek utama yaitu :

1. Input, berupa word yaitu kumpulan alphabet (symbols) yang akan diperiksa automaton dan ditentukan input (word) diterima oleh automaton dengan aturan produksi (production rule) dan state yang ada, yaitu sebuah word terdapat pada language jika diterima automaton (accepting condition).

2. States, yaitu kumpulan kondisi (state) automaton, state dapat berupa finite state, stack, tape, dan lainnya berdasarkan definisi dari automaton bersangkutan.
3. Transition function, atau production rule yaitu aturan-aturan yang mendefinisikan berjalannya automaton berdasarkan input symbol dan state dari suatu mesin automaton yang mendefinisikan automaton itu.
4. Acceptance condition, yaitu keadaan ketika automaton menerima sebuah word (input), dapat berupa final state maupun keadaan yang didefinisikan lainnya dari automaton (halt, empty stack, dll).

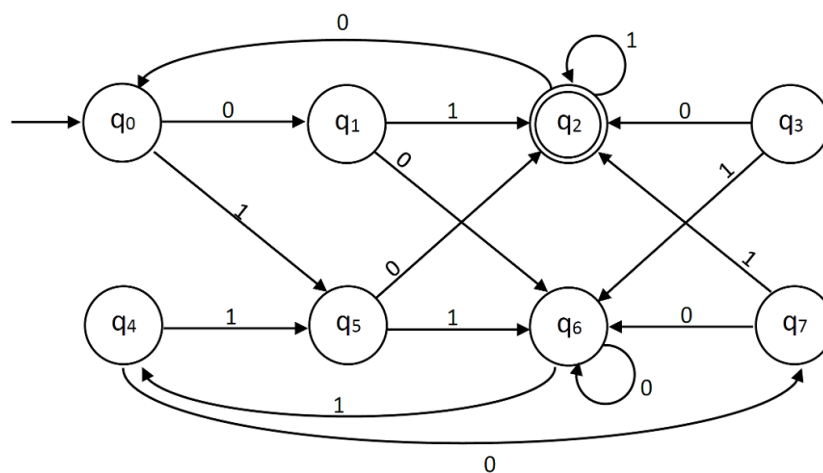
1.2 Finite Automata (FA)

Finite State Machine atau Finite Automata merupakan mesin yang dapat memeriksa bahasa dalam regular language. Finite Automata didefinisikan dalam bentuk 5 tuple

$(Q, \Sigma, \delta, q_0, F)$ yaitu :

1. Q : kumpulan state FA yang finite.
2. Σ : kumpulan symbol input yang finite.
3. δ : transition function, yaitu fungsi yang memetakan dari suatu state pada mesin dan suatu input symbol ke state lain pada mesin FA.
4. q_0 : State awal dari mesin, merupakan state pada Q .
5. F : kumpulan state akhir yang diterima oleh FA.

Secara visualisasi, FA dapat digambarkan sebagai graf berarah dengan setiap simpul merupakan state dari FA (yaitu state pada Q) dan sisi berarah merupakan representasi transtition function FA yang diberikan label simbol input.



Gambar 1.2 Transition Diagram

FA sendiri dibagi menjadi 2 jenis berdasarkan jenis transition function FA :

1. DFA (Deterministic Finite Automata), yaitu FA dengan transition function dari FA untuk setiap state q pada Q ($q \in Q$) dan setiap alphabet pada Σ ($\alpha \in \Sigma$), terdapat tepat satu transition function yang memetakan q ke state lain dengan input symbol α .
2. NFA (Nondeterministic Finite Automata), yaitu FA dengan transition function yang didefinisikan untuk setiap q pada Q ($q \in Q$) terdapat 0 atau lebih transition function dengan input symbol α pada Σ yang memetakan q ke state lain pada Q .

1.3 Context Free Grammar (CFG)

Context Free Grammar merupakan automata yang termasuk pada type-2 yaitu dapat memeriksa Context Free Language. Secara formal CFG didefinisikan sebagai 4 tuple yaitu:

1. V : Kumpulan variable / non-terminal dari CFG.
2. T : Kumpulan terminal dari CFG.
3. P : Production rule dari CFG, yaitu aturan produksi yang memetakan sebuah variable pada ruas kiri aturan produksi menjadi sekuens dari variabel dan/atau terminal.
4. S : Start variabel dari CFG, yaitu variabel pertama dari penurunan aturan produksi.

Suatu word diterima oleh CFG ketika semua symbol pada word dapat diturunkan dari S dari 1 atau lebih langkah production rule. CFG berekuivalensi dengan automata type-2 lainnya seperti Push Down Automata yang dapat menerima Context Free Language. Penurunan word dari start variable dapat dikerjakan secara visual dengan parse tree maupun algoritma lainnya baik secara top-down maupun bottom-up. Sesuai dengan hierarkis automata, CFG dapat memeriksa regular language seperti halnya Finite Automata.

Contoh dari Production Rule CFG :

$$S \rightarrow A1B$$

$$A \rightarrow 0A \mid \epsilon$$

$$B \rightarrow 0B \mid 1B \mid \epsilon$$

1.4 Chomsky Normal Form (CNF)

Sebuah *context-free grammar* dikatakan dalam CNF jika memenuhi *production rule* sebagai berikut:

$$A \rightarrow BC,$$

$$A \rightarrow a,$$

$$S \rightarrow \epsilon$$

dengan A , B , dan C adalah non-terminal, a adalah terminal, S adalah simbol start, dan ϵ menandakan string kosong. Selain itu CNF juga harus memenuhi syarat berupa:

- Tidak memiliki *useless variable*
- Tidak memiliki *ϵ -production*
- Tidak memiliki *unit production*

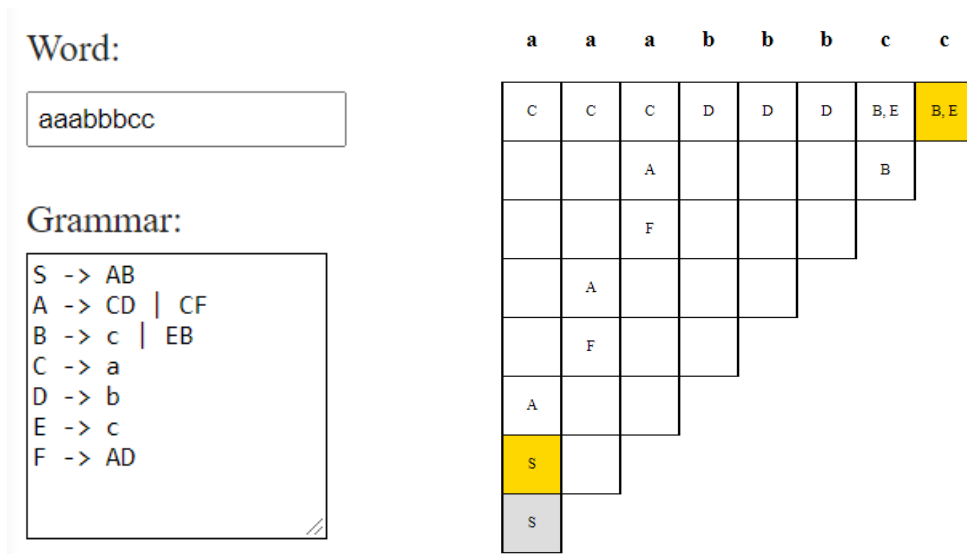
Setiap *grammar* dalam CNF adalah CFG dan sebaliknya setiap CFG dapat ditransformasikan ke bentuk yang ekuivalen dengan CNF.

1.5 CYK

Algoritma Cocke-Younger-Kasami (CYK) adalah sebuah algoritma *parsing* yang sangat efisien untuk *context-free grammar* (CFG). Algoritma ini ideal untuk memutuskan apakah sebuah kalimat sesuai dengan *context-free grammar* yang telah diubah dalam bentuk Chomsky *normal form* (CNF).

Keunggulan dari algoritma CYK ini adalah efisiensinya yang tinggi dalam situasi tertentu. Dengan menggunakan Big O notation, kasus terburuk akan berjalan pada $O(|G| \cdot n^3)$ dengan n adalah panjang dari string yang diuraikan dan $|G|$ adalah ukuran dari CNF. Ini menjadikannya salah satu algoritma penguraian paling efisien dalam kasus terburuknya.

Walaupun, algoritma lain memiliki waktu rata-rata berjalan lebih baik dalam suatu skenario lain.

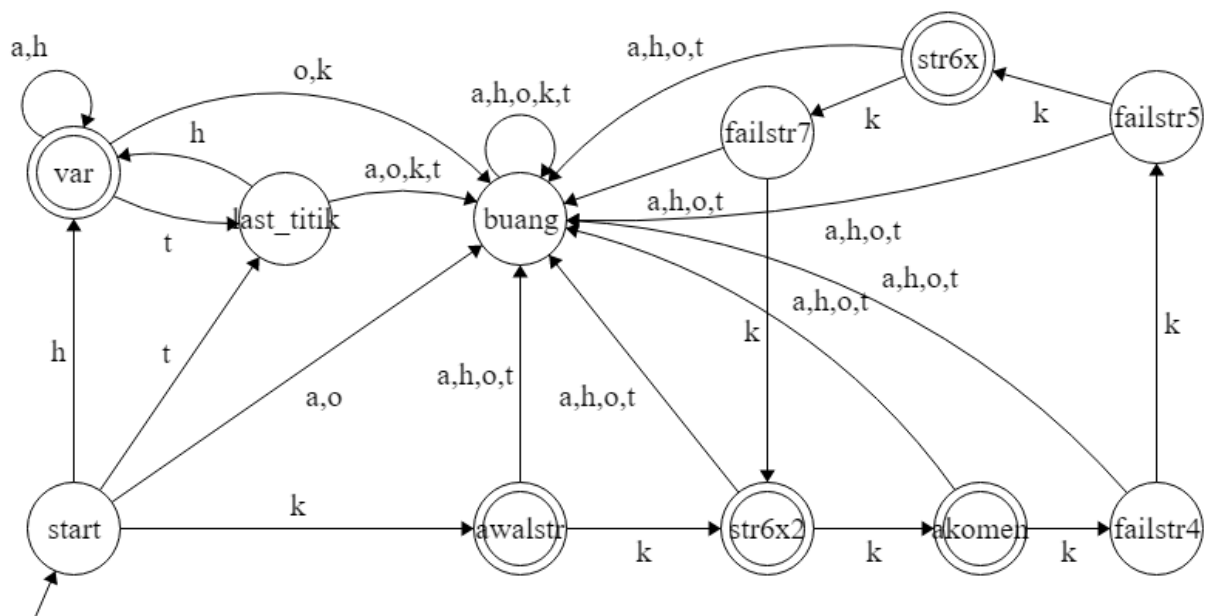


Gambar 1.3 Contoh Word, CNF, dan CYK Table

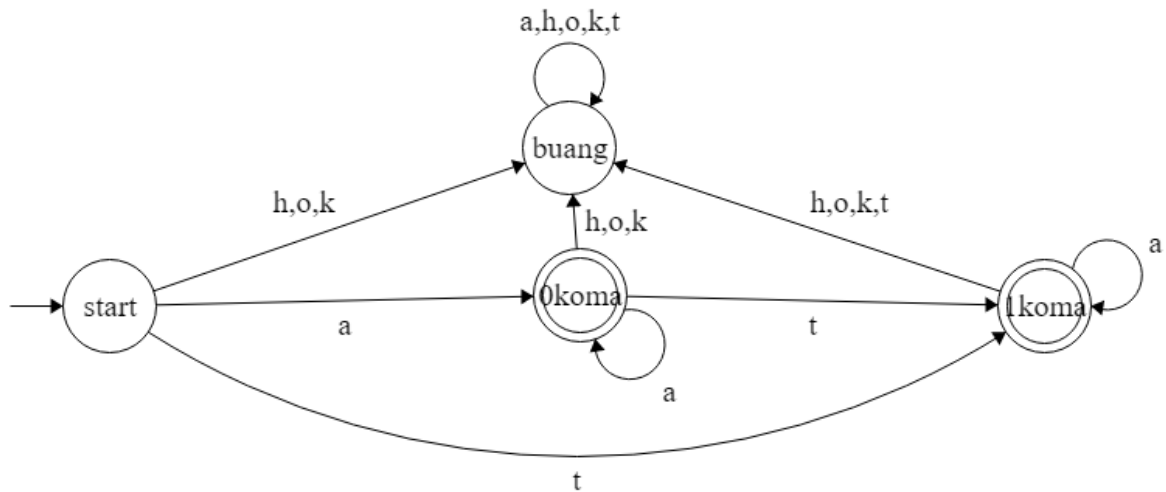
II. FA dan CFG

2.1 FA

FA yang dibuat merupakan DFA. DFA yang digunakan terdapat 2 buah DFA trans_table dan DFA trans_angka, berikut merupakan masing-masing transition diagram DFA.



Gambar 2.1 trans_table



Gambar 2.2 trans_angka

Kedua DFA memiliki Input Symbols yang sama. Secara formal kedua DFA didefinisikan dengan 5 tuple sebagai berikut:

1. DFA trans_table = ({start, var, last_titik, buang, awalstr, str6x2, akomen, failstr4, failstr5, str6x, failstr7}, {a,h,o,k,t}, start, {var, awalstr, str6x2, akomen, str6x})
2. DFA trans_angka = ({start, 0koma, 1koma, buang}, {a,h,o,k,t}, start, {0koma, 1koma})

Berikut merupakan penjelasan representasi tiap symbol :

1. a : Kumpulan token angka
2. h : Kumpulan token alphabet (a-z , A-Z)
3. k : Kumpulan tanda kutip (“ dan ‘)
4. t : Karakter titik (.)
5. o : Karakter lainnya

Berikut merupakan representasi (pemaknaan) tiap state pada masing-masing DFA:

1. DFA trans_table
 - a. start : Start state
 - b. var : State merepresentasikan kata berupa variable yang valid
 - c. last_titik : State merepresentasikan kata berakhiran titik
 - d. awalstr : State merepresentasikan kata berupa awalan string (“ atau ‘)
 - e. str6x2 : State merepresentasikan banyak kutip sebanyak 6x+2
 - f. akomen : State merepresentasikan komen multiline
 - g. failstr4 : State merepresentasikan banyak kutip sebanyak 6x+4
 - h. failstr5 : State merepresentasikan banyak kutip sebanyak 6x+5
 - i. str6x : State merepresentasikan banyak kutip sebanyak 6x
 - j. failstr7 : State merepresentasikan banyak kutip 6x+1 (x > 0)
 - k. buang : State buangan

DFA trans_table digunakan untuk memeriksa kebenaran penamaan variable atau kebenaran kumpul kutip.

2. DFA trans_angka
 - a. start : Start state

- b. Okoma : State merepresentasikan kumpulan angka tanpa koma
- c. 1koma : State merepresentasikan kumpulan angka dengan 1 koma
- d. buang : State buangan

DFA trans_angka digunakan untuk memeriksa sebuah kata merupakan angka atau bukan.

2.2 CFG

Secara garis besar, CFG kami adalah sebagai berikut:

HURUFKAPITAL \rightarrow State

huruf kecil/simbol \rightarrow terminal

STARTSTATE \rightarrow . S | . SS

S \rightarrow ASSIGNSTATE | CONDITIONALSTATE | CLASSSTATE | DEFSTATE |

FORSTATE | FUNCSTATE | IFSTATE | NOTSTATE | RANGESTATE | WHILESTATE | WITHSTATE | VAR | NUM

SS \rightarrow S SS | S

ASSIGNSTATE \rightarrow VAR = CONDITIONALSTATE |

CONDITIONALSTATE \rightarrow (VAR OPERATOR VAR) | VAR OPERATOR VAR

CLASSSTATE \rightarrow class VAR :

DEFSTATE \rightarrow def VAR :

FORSTATE \rightarrow for VAR in VAR | for VAR in NUM | for VAR in RANGESTATE

FUNCSTATE \rightarrow VAR (CONDITIONALSTATE)

IFSTATE \rightarrow if CONDITIONALSTATE : SS ELIFSTATE ELSESTATE |
if () : SS ELIFSTATE ELSESTATE

ELIFSTATE \rightarrow elif (CONDITIONALSTATE) : SS ELIFSTATE |
elif () : SS ELIFSTATE | ϵ

ELSESTATE \rightarrow else: SS | ϵ

IMPORTSTATE \rightarrow FROM VAR import VAR AS VAR

FROM \rightarrow from | ϵ

AS \rightarrow as | ϵ

NOTSTATE \rightarrow not CONDITIONALSTATE | (not CONDITIONALSTATE)

RANGESTATE \rightarrow range (CONDITIONALSTATE) | range ()

WHILESTATE \rightarrow while (CONDITIONALSTATE) :

VAR \rightarrow seluruh string/karakter yang ada pada file yang dibaca. VAR juga bisa menghasilkan list

([_,_]) dan dictionary {__:__}.

VAR akan dibagi lagi menjadi beberapa jenis yakni:

VARONLY \rightarrow hanya mengandung karakter/string, contoh : a | abc | abjad

VARFUNC → mengandung fungsi, contoh : abc()

VARTITIK → mengandung VAR.VAR, contoh : numpy.linalg

Kombinasi dari VARONLY, VARFUNC, dan VARTITIK akan digunakan untuk berbagai kasus yang berbeda seperti pada bagian IMPORTSTATE, maka jenis VAR yang dipakai adalah VARTITIK. sedangkan untuk bagian class, jenis VAR yang digunakan adalah VARFUNC.

NUMBER → seluruh angka yang ada pada file yang dibaca

OPERATOR → seluruh simbol operasi aritmatika / logika / bit yang ada pada file yang dibaca

OPERATOR akan dibagi lagi menjadi:

OPERATORARITMATIKA → + | - | * | / | dan lain-lain

OPERATORLOGIKA → == | > | < | >= | <= | dan lain-lain

OPERATORASSIGNMENT → += | /= | *= | dan lain-lain

OPERATORBITWISE → >> | << | ~ | dan lain-lain

CFG tersebut merupakan **gambaran singkat** dari CFG yang kami implementasikan pada main.py. CFG yang kami implementasikan cukup panjang dan banyak handle kasus khusus pada setiap state sehingga **jika kami tulis secara terperinci**, maka CFG tersebut akan sangat **panjang dan sulit dibaca**.

III. Implementasi

3.1 Struktur Data

Dua file utama yang menyusun program kami adalah main.py dan tokenreader.py. tokenreader.py berfungsi untuk mem-*parse* file python menjadi token-token yang digabung dalam sebuah list. tokenreader.py juga berfungsi untuk mengecek apakah nama variabel sudah benar atau belum. setelah file input di-*parse*, token-token tersebut akan di cek kebenarannya dengan menggunakan algoritma CYK (Cocke-Younger-Kasami). algoritma tersebut akan mengecek apakah token-token yang dimasukkan sesuai dengan bahasa (CNF) yang kita buat di main.py. jika bahasa tersebut cocok, maka program akan menampilkan “*Accepted*”. Sedangkan jika token-token yang dimasukkan tidak berada pada bahasa yang kita buat, maka program akan menampilkan “*Syntax Error*”.

3.2 Fungsi dan Prosedur

Pada main.py hanya ada satu fungsi, yaitu cykParse() yang berfungsi untuk melakukan algoritma CYK pada rule yang telah dibuat dengan masukan yang diterima.

Pada tokenreader.py terdapat dua fungsi, yaitu:

1. dfa() berfungsi mengembalikan suatu state dari DFA yang telah dibuat terhadap masukan yang diberikan.
2. readtokens() berfungsi untuk membaca sebuah masukan file menjadi token-token yang akan dimasukkan ke algoritma CYK.

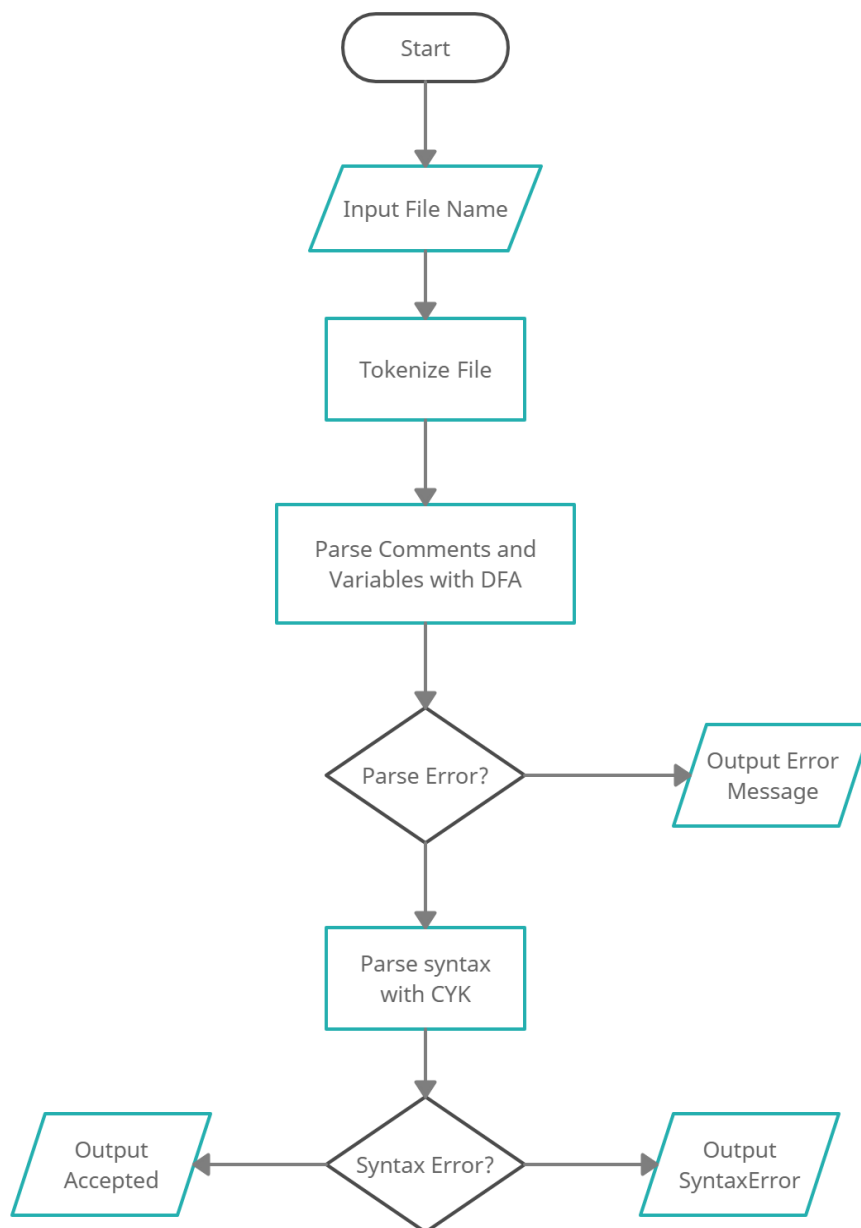
3.3 Antarmuka

Pada tugas besar ini digunakan Antarmuka yang berbasis CLI (*Command Line Interface*). Program dapat dijalankan pada terminal/*command prompt* yang sudah menunjuk folder yang memuat “main.py”, tanpa tanda kutip. Masukkan input “python main.py” dan akan diminta memasukkan nama file program yang akan diperiksa. Pastikan file yang diperiksa berada satu folder dengan program utama. Setelah itu akan keluar hasil apakah terdapat *syntax error* pada program atau *accepted*.

```
D:\Sumerta Yoga\ITB\Semester 3\TBF0\Tubes\New folder\TBF001-20021>python main.py
Masukkan nama file yang akan dites: input.py
Syntax Error
```

Gambar 3.1 Tampilan program pada cmd

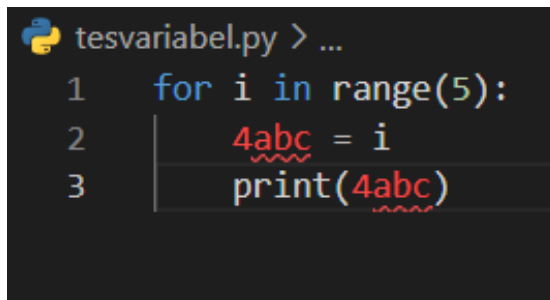
3.4 Flow Program



Gambar 3.2 Flowchart Program

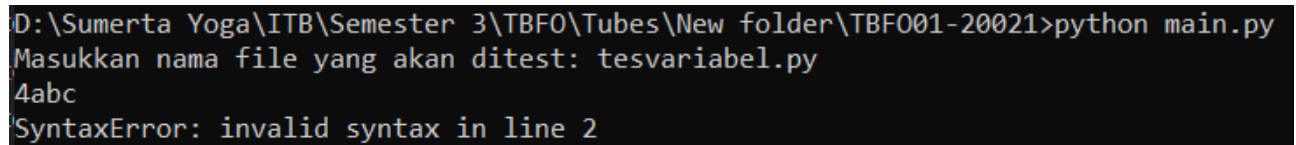
IV. Pengujian / Program Testing

4.1 tesvariabel.py



```
tesvariabel.py > ...
1  for i in range(5):
2      4abc = i
3      print(4abc)
```

Gambar 4.1 tesvariabel.py

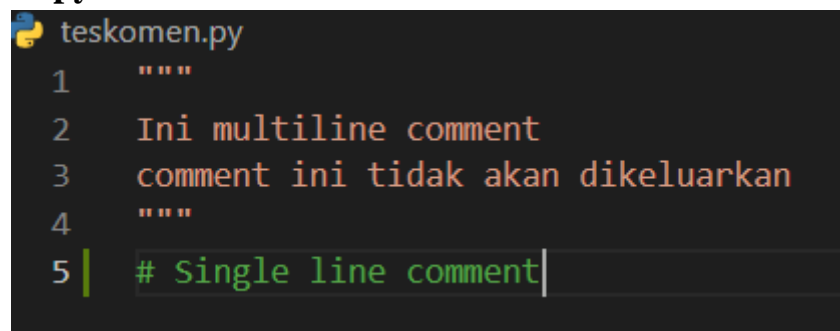


```
D:\Sumerta Yoga\ITB\Semester 3\TBF0\Tubes\New folder\TBF001-20021>python main.py
Masukkan nama file yang akan dites: tesvariabel.py
4abc
SyntaxError: invalid syntax in line 2
```

Gambar 4.2 Output pengecekan tesvariabel.py

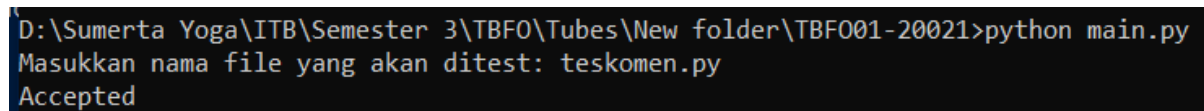
Kode program pada tesvariabel.py memiliki kesalahan sintaks pada line ke-2, yaitu kesalahan nama variabel. Terdapat beberapa syarat penamaan variabel pada python dan yang tidak terpenuhi adalah variabel tidak boleh diawali oleh angka.

4.2 teskomen.py



```
teskomen.py
1  """
2  Ini multiline comment
3  comment ini tidak akan dikeluarkan
4  """
5  # Single line comment
```

Gambar 4.3 teskomen.py



```
D:\Sumerta Yoga\ITB\Semester 3\TBF0\Tubes\New folder\TBF001-20021>python main.py
Masukkan nama file yang akan dites: teskomen.py
Accepted
```

Gambar 4.4 Output pengecekan teskomen.py

Kode program pada teskomen.py berhasil di-*compile* karena hanya berisi komentar *singeline* dan *multiline*.

4.3 tesif.py

```
angka_1= int(input('masukkan satu angka untuk ratusan ='))
angka_2= int(input('masukkan satu angka untuk puluhan='))
angka_3= int(input('masukkan satu angka untuk satuan='))

#menentukan apakah bilangan tersebut membesar atau tidak
if((angka_2>angka_1) and (angka_3>angka_2) and (angka_3>angka_1)):
    print('bilangan tersebut digit membesar')

elif ((angka_3<angka_2) and (angka_2<angka_1) and (angka_3<angka_1)):
    print('bilangan tersebut digit mengecil')

else:print('bilangan tersebut tidak beraturan')
```

Gambar 4.5 tesif.py

```
D:\Sumerta Yoga\ITB\Semester 3\TBF0\Tubes\New folder\TBF001-20021>python main.py
Masukkan nama file yang akan dites: tesif.py
Accepted
```

Gambar 4.6 Output pengecekan tesif.py

Kode program pada tesif.py berhasil di-*compile* karena sesuai dengan aturan *if-elif-else* yang berlaku pada python.

4.4 tesfor.py

```
tesfor.py > ...
1 kata = "abcd"
2 | for i in kata
3 |     print(i)
```

Gambar 4.7 tesfor.py

```
D:\Sumerta Yoga\ITB\Semester 3\TBF0\Tubes\New folder\TBF001-20021>python main.py
Masukkan nama file yang akan dites: tesfor.py
Syntax Error
```

Gambar 4.8 Output pengecekan tesfor.py

Kode program tesfor.py mengalami *syntax error* pada baris ke-2. Kesalahannya adalah kurangnya tanda titik dua pada *for statement*.

4.5 teswhile.py

```
teswhile.py > ...  
1   i = 1  
2   while(i < 5):  
3       if (i % 2 == 0):  
4           continue  
5       i += 1
```

Gambar 4.9 teswhile.py

```
D:\Sumerta Yoga\ITB\Semester 3\TBF0\Tubes\New folder\TBF001-20021>python main.py  
Masukkan nama file yang akan dites: teswhile.py  
Accepted
```

Gambar 4.10 Output pengecekan teswhile.py

Kode program teswhile.py berhasil di-*compile* karena sudah memenuhi syarat while statement pada python. Selain itu continue juga dipanggil pada loop sehingga tidak menyalahi aturan sintaks python.

4.6 tesdef.py

```
def 4f(x):                                #deklarasi fungsi  
    y = x**2 - 2*x + 5  
    return y  
  
A = int(input("Masukkan nilai A: "))  
B = int(input("Masukkan nilai B: "))  
  
for x in range(A, B+1):  
    print("f(" + str(x) + ") = " + str(f(x)))
```

Gambar 4.11 tesdef.py

```
D:\Sumerta Yoga\ITB\Semester 3\TBF0\Tubes\New folder\TBF001-20021>python main.py  
Masukkan nama file yang akan dites: tesdef.py  
4f  
SyntaxError: invalid syntax in line 1
```

Gambar 4.12 Output pemeriksaan tesdef.py

Kode program tesdef.py gagal di-*compile* karena terdapat kesalahan sintaks. Kesalahan ada pada baris ke-1 yaitu pada penamaan nama fungsi yang mirip seperti variabel, yaitu tidak boleh diawali dengan angka.

4.7 tesdict.py

```
tesdict.py > [E] var_check
1  var_check = {
2      0:{'a' : 2, 'h' : 1, 'o' : 2 },
3      1:{'a' : 1, 'h' : 1, 'o' : 2 },
4      2:{'a' : 2, 'h' : 2, 'o' : 2 },
5  }
6
```

Gambar 4.13 tesdict.py

```
D:\Sumerta Yoga\ITB\Semester 3\TBF0\Tubes\New folder\TBF001-20021>python main.py
Masukkan nama file yang akan dites: tesdict.py
Accepted
```

Gambar 4.14 Output pemeriksaan tesdict.py

Kode program tesdict.py berhasil di-*compile* karena sudah sesuai dengan aturan *dictionary* pada python.

4.8 tesstring.py

```
tesstring.py > ...
1  if a == b :
2      c = "stringsalah
3      "
```

Gambar 4.15 tesstring.py

```
D:\Sumerta Yoga\ITB\Semester 3\TBF0\Tubes\New folder\TBF001-20021>python main.py
Masukkan nama file yang akan dites: tesstring.py
SyntaxError: EOL while scanning string literal in line 2
```

Gambar 4.16 Output pemeriksaan tesstring.py

Kode program tesstring.py menghasilkan syntax error karena tanda petik dua yang tidak berada satu baris dengan pasangannya. Pada output muncul jenis kesalahannya dan urutan barisnya.

4.9 teserror.py

```
teserror.py > ...
1  for i in range(1.100):
2      if a > b : ""
3
4
5      print("aduh")
```

Gambar 4.17 teserror.py

```
D:\Sumerta Yoga\ITB\Semester 3\TBFO\Tubes\New folder\TBFO01-20021>python main.py
Masukkan nama file yang akan dites: teserror.py
SyntaxError: EOF while scanning triple-quoted string literal in line 5
```

Gambar 4.18 Output pengecekan teserror.py

Kode program teserror.py menghasilkan syntax error karena tanda petik tiga yang tidak memiliki pasangan sampai akhir program. Pada output muncul jenis kesalahannya dan urutan barisnya.

V. Link Repository

Berikut link repository kami:

<https://github.com/bryanbernigen/TBFO01-20021.git>

VI. Pembagian Tugas

No.	NIM	Nama	Tugas
1.	13520021	Gede Sumerta Yoga	Grammar
2.	13520034	Bryan Bernigen	Grammar
3.	13520040	Ng Kyle	Parser

VII. Daftar Referensi

<https://cs.stanford.edu/people/eroberts/courses/soco/projects/2004-05/automata-theory/basics.html>

<https://www.irif.fr/~jep/PDF/MPRI/MPRI.pdf>

<https://www.xarg.org/tools/cyk-algorithm/>

<https://www.quora.com/Why-are-compilers-so-hard-to-write#:~:text=They%20are%20hard%20to%20write,hands%20Don%2C%20iterative%20learning.>