

TUGAS BESAR IF 2124
TEORI BAHASA FORMAL DAN OTOMATA

Oleh

Afrizal Sebastian	13520120
Adzka Ahmadetya Zaidan	13520127
Mohamad Hilmi Rinaldi	13520149



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

DAFTAR ISI

DAFTAR ISI.....	2
BAB 1 TEORI DASAR.....	3
1.1. Finite Automata (FA).....	3
1.2. Context Free Grammar (CFG)	3
1.3. Syntax Python.....	4
BAB 2 HASIL CFG	6
2.1. Variables	6
2.2. Terminal Symbols	6
2.3. Productions	6
2.4. Start Symbol.....	7
2.5. Hasil CNF.....	7
BAB 3 IMPLEMENTASI DAN PENGUJIAN	12
3.1. Spesifikasi Program	12
3.1.1. CFG2CNF.py.....	12
3.1.2. helper.py	12
3.1.3. CYK.py	12
3.1.4. parserprogram.py	12
3.2. Uji Kasus	13
3.2.1. tc1acc.py.....	13
3.2.2. tc1rej.py	13
3.2.3. tc2acc.py.....	14
3.2.4. tc2rej.py	14
3.2.5. tc3acc.py.....	15
3.2.6. tc3rej.py	15
BAB 4 KESIMPULAN DAN SARAN.....	16
4.1. Kesimpulan	16
4.2. Saran.....	16
Link Repository Github	17
Pembagian Tugas.....	17

BAB 1

TEORI DASAR

1.1. Finite Automata (FA)

Finite Automata adalah mesin abstrak berupa sistem model matematika dengan masukan dan keluaran diskrit yang dapat mengenali bahasa dan dapat diimplementasikan secara nyata di mana sistem dapat berada di salah satu dari sejumlah berhingga konfigurasi internal yang disebut state.

Sebuah Finite Automata memiliki 5 komponen, yaitu :

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

dengan,

1. Q adalah himpunan berhingga state pada FA.
2. Σ adalah himpunan berhingga simbol.
3. δ adalah fungsi transisi yang merupakan fungsi yang menerima state dan simbol untuk berpindah state.
4. q_0 adalah state awal FA.
5. F adalah himpunan final state dengan F adalah bagian dari Q .

Cara penulisan Finite Automata dapat dilakukan dengan cara:

1. Diagram transisi (transition diagram), dengan berupa graf.
2. Tabel transisi (transition table), dengan menggunakan table dengan parameter state dan simbol.

1.2. Context Free Grammar (CFG)

Context Free Grammar merupakan tata bahasa formal yang setiap aturan produksinya dalam bentuk $\alpha \rightarrow \beta$. α merupakan pemproduksi dan β adalah hasil produksi yang tidak memiliki batasan.

Aturan produksi CFG :

$$\alpha \rightarrow \beta$$

Ruas kiri adalah sebuah simbol variabel atau non-terminal ($\alpha \in N$).

Ruas kanan dapat berupa terminal, variabel ataupun ϵ ($\beta \in (T \cup N)$).

Bentuk formal CFG :

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

V adalah daftar variabel produksi.

T adalah daftar terminal yang dipakai dalam CFG.

P adalah aturan produksi dari CFG.

S adalah *start variable* dari aturan produksi CFG.

Context Free Grammar ini menjadi dasar pembentuk suatu parser (proses analisis sintaks). Proses analisis ini terdapat dalam pembacaan string dalam bahasa sesuai CFG tertentu

yang mematuhi aturan produksi. Bagian sintaks dalam suatu *compiler* biasanya didefinisikan dalam CFG.

Dalam proses analisis sintaks, CFG yang sudah didefinisikan sebelumnya perlu disederhanakan dan dipastikan tidak ambigu di dalamnya. Setelah itu, CFG perlu dinormalisasi menjadi Chomsky Normal Form (CNF). CNF dapat dibentuk dari CFG yang telah mengalami penyederhanaan yaitu yang terdiri dari penghilangan produksi useless, unit, dan kosong.

Aturan produksi CNF :

$$\alpha \rightarrow \beta$$

$\alpha = 1$ Non-terminal

$\beta = 1$ Terminal atau 2 Non-terminal

Setelah terbentuk CNF, dalam implementasi parsing terdapat algoritma yang dapat membantu dalam prosesnya. Algoritma ini disebut dengan algoritma Cocke-Younger-Kasami (CYK). Algoritma CYK digunakan untuk menunjukkan apakah suatu string tertentu dapat diperoleh dari suatu tata bahasa.

Algoritma CYK dapat digambarkan dalam bentuk tabel seperti di bawah ini yang mengecek string dengan panjang string ($n = 5$).

	1	2	3	4	5
1	$V_{1,1}$	$V_{2,1}$	$V_{3,1}$	$V_{4,1}$	$V_{5,1}$
2	$V_{1,2}$	$V_{2,2}$	$V_{3,2}$	$V_{4,2}$	
3	$V_{1,3}$	$V_{2,3}$	$V_{3,3}$		
4	$V_{1,4}$	$V_{2,4}$			
5	$V_{1,5}$				

$V_{i,j}$: i merupakan kolom dan j merupakan baris

Syarat suatu string dapat diterima yaitu jika $V_{1,n}$ memuat simbol awal.

1.3. Syntax Python

Python merupakan bahasa pemrograman tingkat tinggi yang mudah dibaca oleh manusia, serta menggunakan kata kunci bahasa Inggris tanpa banyak menggunakan tanda baca dibandingkan bahasa pemrograman lainnya. Python lebih banyak menggunakan indentasi dibanding kurung kurawal atau tanda baca lainnya. Namun, pada tugas besar yang kami kerjakan, kami tidak diminta dan tidak mengimplementasikan pengecekan indentasi pada kode Python yang akan kami cek.

Pernyataan dan *control flow* Python meliputi: *if*, *for*, *while*, *raise*, *class*, *def*, *with*, *break*, *continue*, *pass*, *import*, dan *print*. *For* dan *while* pada Python merupakan pernyataan untuk looping suatu algoritma. *If*, bersama *elif*, dan *else* digunakan untuk mengeksekusi blok kode secara kondisional. *Raise* digunakan untuk memanggil suatu pengecualian yang tertangkap. *Class* mengeksekusi blok kode dan biasa digunakan untuk pemrograman berorientasi objek.

Def digunakan untuk mendefinisikan suatu fungsi atau prosedur, untuk fungsi, biasanya ditambah pernyataan *return*, untuk mengembalikan suatu nilai. *Break* digunakan untuk memberhentikan suatu *loop*, *continue* digunakan untuk melewati satu iterasi, dan *pass* digunakan untuk membuat blok kode kosong. *Import* digunakan untuk memanggil suatu *library* Python atau untuk mengimpor suatu modul berisi fungsi dan variabel. *Print* digunakan untuk menuliskan sesuatu pada layar.

BAB 2

HASIL CFG

Berikut merupakan Context Free Grammar yang telah kami buat untuk *compiler* Bahasa Python:

$G = (V, T, P, S)$, dengan V : *variables*, T : *terminal symbols*, P : *productions*, dan S : *start symbol*.

2.1. Variables

S V VAR VAL CBRACKET OP VAL TYPE INPUT COMPARE COMPAREB BOOLEAN
EXPRESSION IF ELIF ELSE TEXT STRING COMMENT PRINT IMPORT RAISE
LOOPEND WHILE FOR RETURN DEF CLASS METHOD

2.2. Terminal Symbols

+ - * / % ! = > < () ' " : , if elif else and or def return
while for in range break pass continue variable string number
True False class is none not print input str float int double
import as from with open raise len

2.3. Productions

$S \rightarrow S S \mid \text{VAR} = V \mid \text{VAR} = \text{BOOLEAN} \mid \text{VAR} + = V \mid \text{VAR} - = V \mid \text{VAR} * = V \mid \text{VAR} / = V \mid \text{IMPORT} \mid \text{VAR CBRACKET} \mid \text{VAR (VAL)} \mid \text{COMMENT} \mid \text{PRINT} \mid \text{IF} \mid \text{WHILE} \mid \text{FOR} \mid \text{DEF} \mid \text{CLASS};$
 $V \rightarrow \text{VAR} \mid \text{VAL} \mid V \text{ OP } V \mid \text{INPUT} \mid V , V \mid \text{none} \mid \text{METHOD};$
 $\text{VAR} \rightarrow \text{VAR} , \text{VAR} \mid \text{variable};$
 $\text{CBRACKET} \rightarrow (\text{VAL}) \mid () \mid (\text{VAR});$
 $\text{OP} \rightarrow + \mid - \mid * \mid / \mid \%;$
 $\text{VAL} \rightarrow \text{number} \mid (V) \mid V \text{ OP } V \mid V * * V \mid V / / V \mid \text{BOOLEAN} \mid \text{STRING} \mid \text{VAL} , \text{VAL};$
 $\text{TYPE} \rightarrow \text{int} \mid \text{str} \mid \text{float} \mid \text{double};$
 $\text{INPUT} \rightarrow \text{input CBRACKET} \mid \text{TYPE (INPUT)};$
 $\text{COMPARE} \rightarrow = = \mid ! = \mid > = \mid < = \mid > \mid < \mid \text{is};$
 $\text{COMPAREB} \rightarrow \text{and} \mid \text{or};$
 $\text{BOOLEAN} \rightarrow \text{True} \mid \text{False} \mid \text{VAR} \mid V \text{ COMPARE } V \mid \text{BOOLEAN COMPAREB} \mid \text{not BOOLEAN};$
 $\text{EXPRESSION} \rightarrow (\text{BOOLEAN}) : S \mid \text{BOOLEAN} : S \mid (\text{BOOLEAN}) : \text{RETURN} \mid \text{BOOLEAN} : \text{RETURN};$
 $\text{IF} \rightarrow \text{if EXPRESSION} \mid \text{IF ELIF} \mid \text{IF ELSE} \mid \text{IF RAISE};$
 $\text{ELIF} \rightarrow \text{elif EXPRESSION} \mid \text{ELIF ELIF} \mid \text{ELIF ELSE};$
 $\text{ELSE} \rightarrow \text{else} : S \mid \text{else} : \text{RETURN};$
 $\text{TEXT} \rightarrow \text{VAR} \mid \text{TEXT TEXT} \mid \text{OP} \mid \text{COMPARE} \mid \text{COMPAREB} \mid \text{not} \mid \text{LOOPEND} \mid ;;$
 $\text{STRING} \rightarrow " \text{string} " \mid ' \text{string} ' \mid " \text{TEXT} " \mid ' \text{TEXT} ' \mid \text{STRING} + \text{STRING} \mid \text{STRING} * \text{number} \mid \text{STRING} * (\text{VAL}) \mid ' ' \mid " ";$

COMMENT → " " " TEXT " " " | ' ' ' TEXT ' ' ' ;
 PRINT → print CBRACKET | print (STRING) | print (VAR) | print
 (VAL) ;
 IMPORT → import VAR as VAR | import VAR | from VAR import VAR |
 from VAR import VAR as VAR ;
 RAISE → raise CBRACKET ;
 LOOPEND → continue | break | pass ;
 WHILE → while EXPRESSION | WHILE LOOPEND ;
 FOR → for VAR in STRING : | for VAR in range (VAL) : | for VAR
 in VAR : | FOR S | FOR LOOPEND ;
 RETURN → return BOOLEAN | return V | return | return STRING ;
 DEF → def VAR CBRACKET : S | def VAR (VAL) : S | DEF RETURN |
 def VAR CBRACKET : RETURN ;
 CLASS → class VAR : S ;
 METHOD → len CBRACKET | with open CBRACKET as VAR

2.4. Start Symbol

S

2.5. Hasil CNF

S → S S | VAR A1 | VAR B1 | VAR C1 | VAR D1 | VAR E1 | VAR F1 |
 VAR CBRACKET | VAR G1 | L3 N11 | L3 VAR | J3 O11 | J3 P11 | P3
 I11 | N3 J11 | M3 CBRACKET | M3 K11 | M3 L11 | M3 M11 | S3
 EXPRESSION | IF ELIF | IF ELSE | IF LOOPEND | IF RAISE | H3
 EXPRESSION | G3 Q11 | G3 R11 | G3 S11 | D3 T11 | D3 U11 | DEF
 RETURN | D3 W11 | C3 X11
 Z3 → =
 A1 → Z3 V
 B1 → Z3 BOOLEAN
 C1 → OP C2
 C2 → Z3 V
 D1 → OP D2
 D2 → Z3 V
 E1 → OP E2
 E2 → Z3 V
 F1 → OP F2
 F2 → Z3 V
 Y3 → (
 X3 →)
 G1 → Y3 G2
 G2 → VAL X3
 V → V H1 | V I1 | none | variable | number | Y3 L1 | V M1 | V N1
 | V O1 | VAL P1 | U3 CBRACKET | TYPE Q1 | B3 CBRACKET | A3 Y11 |
 True | False | V R1 | BOOLEAN S1 | TEXT BOOLEAN | variable | P3
 B11 | N3 C11 | P3 D11 | N3 E11 | STRING F11 | STRING G11 | STRING
 H11 | N3 N3 | P3 P3

H1 → OP V
 W3 → , | TEXT RETURN
 I1 → W3 V
 VAR → variable
 CBRACKET → Y3 J1 | Y3 X3 | Y3 K1
 J1 → VAL X3
 K1 → VAR X3
 OP → + | - | * | / | %
 VAL → number | Y3 L1 | V M1 | V N1 | V O1 | VAL P1 | True | False
 | V R1 | BOOLEAN S1 | TEXT BOOLEAN | P3 B11 | N3 C11 | P3 D11 |
 N3 E11 | STRING F11 | STRING G11 | STRING H11 | N3 N3 | P3 P3 |
 variable
 L1 → V X3
 M1 → OP V
 N1 → OP N2
 N2 → OP V
 O1 → OP O2
 O2 → OP V
 P1 → W3 VAL
 TYPE → int | str | float | double
 U3 → input
 INPUT → U3 CBRACKET | TYPE Q1
 Q1 → Y3 Q2
 Q2 → INPUT X3
 COMPARE → Z3 Z3 | T3 Z3 | COMPARE Z3 | COMPARE Z3 | > | < | is
 T3 → ! | TEXT S
 COMPAREB → and | or
 BOOLEAN → True | False | V R1 | BOOLEAN S1 | TEXT BOOLEAN |
 variable
 R1 → COMPARE V
 S1 → COMPAREB BOOLEAN
 EXPRESSION → Y3 T1 | BOOLEAN U1 | Y3 W1 | BOOLEAN X1
 T1 → BOOLEAN T2
 T2 → X3 T3
 U1 → TEXT S
 W1 → BOOLEAN W2
 W2 → X3 W3
 X1 → TEXT RETURN
 S3 → if
 IF → S3 EXPRESSION | IF ELIF | IF ELSE | IF LOOPEND | IF RAISE
 R3 → elif
 ELIF → R3 EXPRESSION | ELIF ELIF | ELIF ELSE | ELIF LOOPEND
 Q3 → else
 ELSE → Q3 Y1 | Q3 Z1 | Q3 A11
 Y1 → TEXT S
 Z1 → TEXT RETURN

A11 → TEXT LOOPEND
 TEXT → TEXT TEXT | not | : | variable | + | - | * | / | % | Z3 Z3
 | T3 Z3 | COMPARE Z3 | COMPARE Z3 | > | < | is | and | or |
 continue | break | pass
 P3 → "
 O3 → string
 STRING → P3 B11 | N3 C11 | P3 D11 | N3 E11 | STRING F11 | STRING
 G11 | STRING H11 | N3 N3 | P3 P3
 B11 → O3 P3
 N3 → '
 C11 → O3 N3
 D11 → TEXT P3
 E11 → TEXT N3
 F11 → OP STRING
 G11 → OP VAL
 H11 → OP H12
 H12 → Y3 H13
 H13 → VAL X3
 COMMENT → P3 I11 | N3 J11
 I11 → P3 I12
 I12 → P3 I13
 I13 → TEXT I14
 I14 → P3 I15
 I15 → P3 P3
 J11 → N3 J12
 J12 → N3 J13
 J13 → TEXT J14
 J14 → N3 J15
 J15 → N3 N3
 M3 → print
 PRINT → M3 CBRACKET | M3 K11 | M3 L11 | M3 M11
 K11 → Y3 K12
 K12 → STRING X3
 L11 → Y3 L12
 L12 → VAR X3
 M11 → Y3 M12
 M12 → VAL X3
 L3 → import
 K3 → as
 IMPORT → L3 N11 | L3 VAR | J3 O11 | J3 P11
 N11 → VAR N12
 N12 → K3 VAR
 J3 → from
 O11 → VAR O12
 O12 → L3 VAR
 P11 → VAR P12

P12 → L3 P13
 P13 → VAR P14
 P14 → K3 VAR
 I3 → raise
 RAISE → I3 CBRACKET
 LOOPEND → continue | break | pass
 H3 → while
 WHILE → H3 EXPRESSION
 G3 → for
 F3 → in
 FOR → G3 Q11 | G3 R11 | G3 S11
 Q11 → VAR Q12
 Q12 → F3 Q13
 Q13 → STRING Q14
 Q14 → TEXT S
 E3 → range
 R11 → VAR R12
 R12 → F3 R13
 R13 → E3 R14
 R14 → Y3 R15
 R15 → VAL R16
 R16 → X3 R17
 R17 → TEXT S
 S11 → VAR S12
 S12 → F3 S13
 S13 → VAR S14
 S14 → TEXT S
 RETURN → RETURN BOOLEAN | RETURN V | return | RETURN STRING
 D3 → def
 DEF → D3 T11 | D3 U11 | DEF RETURN | D3 W11
 T11 → VAR T12
 T12 → CBRACKET T13
 T13 → TEXT S
 U11 → VAR U12
 U12 → Y3 U13
 U13 → VAL U14
 U14 → X3 U15
 U15 → TEXT S
 W11 → VAR W12
 W12 → CBRACKET W13
 W13 → TEXT RETURN
 C3 → class
 CLASS → C3 X11
 X11 → VAR X12
 X12 → TEXT S
 B3 → len

METHOD → B3 CBRACKET | A3 Y11
 A3 → with
 Z2 → open
 Y11 → Z2 Y12
 Y12 → CBRACKET Y13
 Y13 → K3 VAR
 S0 → S S | VAR A1 | VAR B1 | VAR C1 | VAR D1 | VAR E1 | VAR F1 |
 VAR CBRACKET | VAR G1 | L3 N11 | L3 VAR | J3 O11 | J3 P11 | P3
 I11 | N3 J11 | M3 CBRACKET | M3 K11 | M3 L11 | M3 M11 | S3
 EXPRESSION | IF ELIF | IF ELSE | IF LOOPEND | IF RAISE | H3
 EXPRESSION | G3 Q11 | G3 R11 | G3 S11 | D3 T11 | D3 U11 | DEF
 RETURN | D3 W11 | C3 X11

BAB 3

IMPLEMENTASI DAN PENGUJIAN

3.1. Spesifikasi Program

Struktur program yang kami buat dalam pembuatan compiler python ini terdiri dari 4 file utama yaitu CFG2CNF.py, helper.py, CYK.py, dan parserprogram.py. Program utama untuk menjalankan compiler ini terdapat pada file parserprogram.py.

3.1.1. CFG2CNF.py

File CFG2CNF.py berisi algoritma untuk mengubah text file `cfg_model.txt` yang berisi CFG yang telah dibuat untuk sintaks Python menjadi text file `cnf_model.txt` yang berisi CNF untuk sintaks Python. File tersebut mengeksekusi langkah-langkah yang diperlukan untuk mengubah suatu CFG menjadi CNF. Algoritma konversi secara umum dilakukan dengan menambahkan aturan $S0 \rightarrow S$. Setelah itu, program menghapus aturan yang menghasilkan terminal dan variabel, kemudian menggantikannya dengan aturan yang menghasilkan 2 variabel dan menambahkan suatu variabel yang menghasilkan terminal yang dihapus. Sebagai contoh, program menggantikan $A \rightarrow Bc$ dengan $A \rightarrow BZ$, dan menambahkan $Z \rightarrow c$. Setelah itu, program akan menghapus aturan non-unitary, yaitu aturan yang menghasilkan lebih dari satu simbol terminal.

3.1.2. helper.py

File helper.py di-*import* pada file CFG2CNF.py untuk digunakan fungsi yang ada didalamnya pada algoritma konversi CFG menjadi CNF. Utamanya, helper.py digunakan untuk membersihkan *production* pada CFG2CNF.py.

3.1.3. CYK.py

File CYK.py merupakan subprogram yang akan digunakan pada parserprogram.py untuk menunjukkan apakah suatu string valid atau tidak. File ini berisi prosedur-prosedur yang berkaitan dengan algoritma Cocke-Younger-Kasami (CYK). Prosedur yang terdapat di dalam file ini terdiri dari prosedur `load_cnf(file_name)` yang menerima input dari file `cnf` yang sudah direalisasi sebelumnya dan akan digunakan sebagai map pada prosedur CYK. Lalu prosedur yang kedua yaitu `cyk(codeToken)` yang menerima input code token yang sudah diproses sebelumnya dan akan dicek menggunakan algoritma CYK dan akan mengembalikan sebuah tabel CYK.

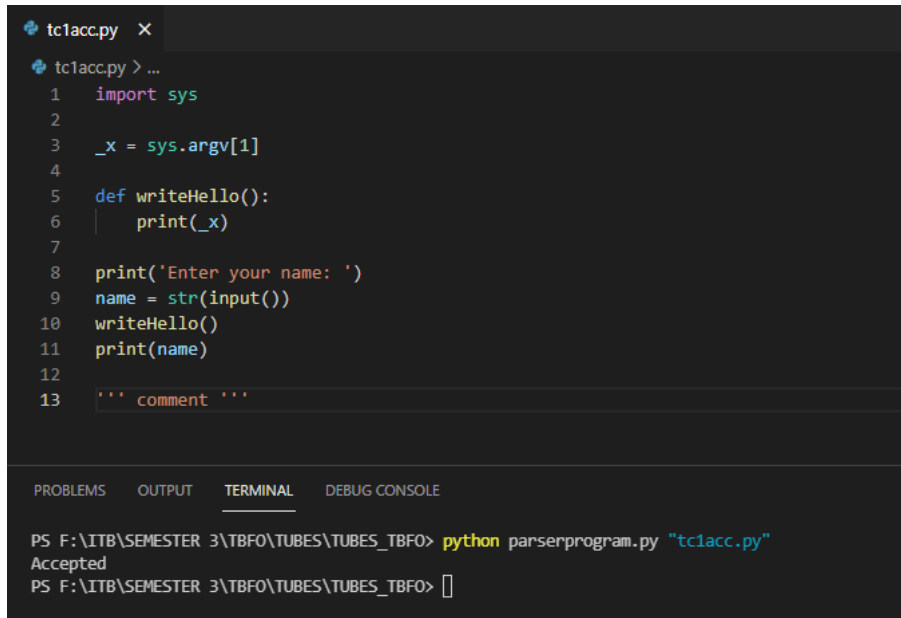
3.1.4. parserprogram.py

File parserprogram.py merupakan program utama yang didalamnya terdapat Main program dan fungsi `makeTokenInput(filename)`. Fungsi `makeTokenInput(filename)` menerima sebuah nama file yang akan dilakukan evaluasi syntax dan mengembalikan baris-baris kode yang ada di file tersebut dan telah menjadi token. Pada saat program dijalankan program akan meminta nama file, pada Main program akan dilakukan `loadCNF` untuk mengambil CNF yang akan digunakan untuk CYK, setelah itu mengubah potongan kode pada file menjadi

token dengan fungsi `makeTokenInput`, hasil dari fungsi tersebut akan dievaluasi dengan CYK agar mengetahui potongan kode tersebut diterima (“*Accepted*”) atau tidak (“*Syntax error*”).

3.2. Uji Kasus

3.2.1. tc1acc.py



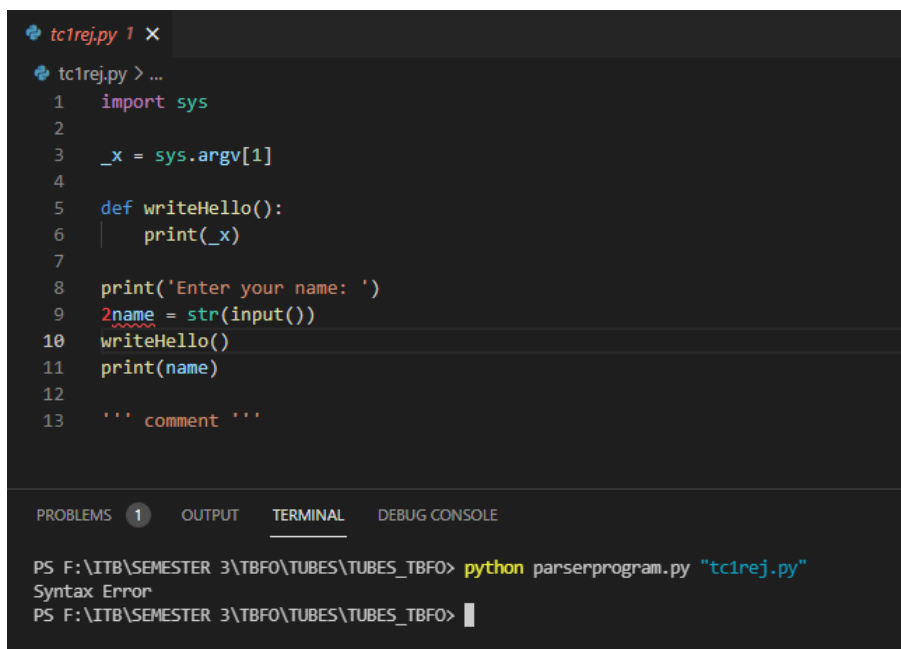
```
tc1acc.py X
tc1acc.py > ...
1  import sys
2
3  _x = sys.argv[1]
4
5  def writeHello():
6      print(_x)
7
8  print('Enter your name: ')
9  name = str(input())
10 writeHello()
11 print(name)
12
13 ''' comment '''

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

PS F:\ITB\SEMESTER 3\TBFO\TUBES\TUBES_TBFO> python parserprogram.py "tc1acc.py"
Accepted
PS F:\ITB\SEMESTER 3\TBFO\TUBES\TUBES_TBFO> 
```

Pada test case ini ingin dilakukan pengecekan terhadap hal-hal dasar seperti nama variabel, penggunaan `print`, `input`, `import`, dan fungsi. Berdasarkan gambar diatas potongan kode tersebut diterima atau tidak ada yang salah.

3.2.2. tc1rej.py



```
tc1rej.py 1 X
tc1rej.py > ...
1  import sys
2
3  _x = sys.argv[1]
4
5  def writeHello():
6      print(_x)
7
8  print('Enter your name: ')
9  2name = str(input())
10 writeHello()
11 print(name)
12
13 ''' comment '''

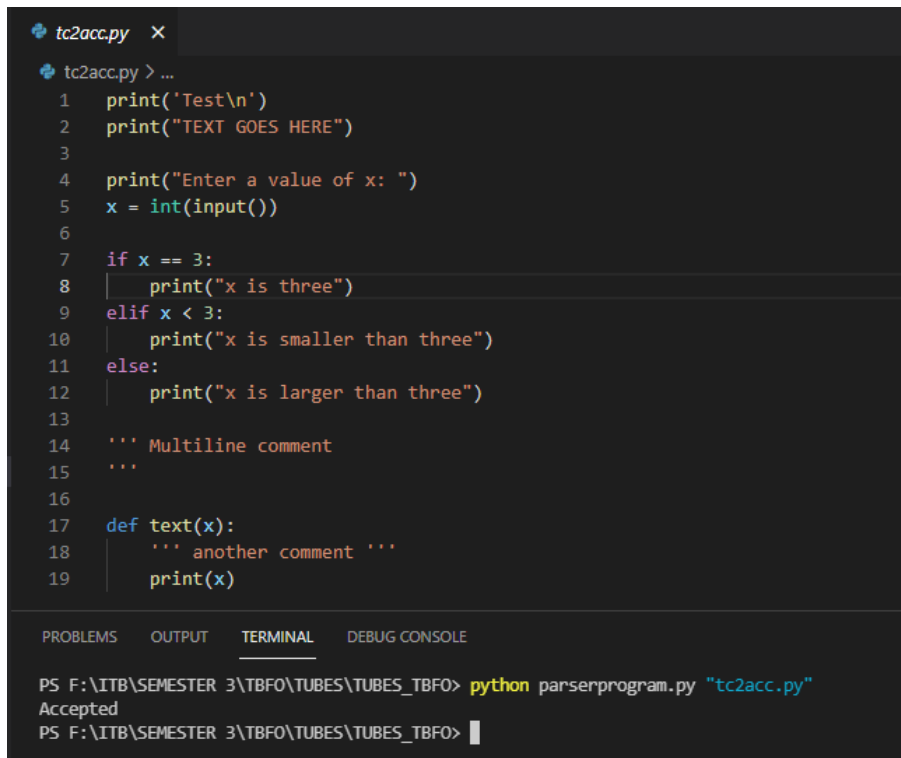
PROBLEMS 1 OUTPUT TERMINAL DEBUG CONSOLE

PS F:\ITB\SEMESTER 3\TBFO\TUBES\TUBES_TBFO> python parserprogram.py "tc1rej.py"
Syntax Error
PS F:\ITB\SEMESTER 3\TBFO\TUBES\TUBES_TBFO> 
```

Pada test case ini lebih menekankan pengecekan terhadap variabel, pada python variabel tidak dapat dimulai dengan angka dan dari gambar diatas terdapat variabel

yang dimulai dengan angka (line 9), sehingga program menampilkan pesan “Syntax Error”

3.2.3. tc2acc.py

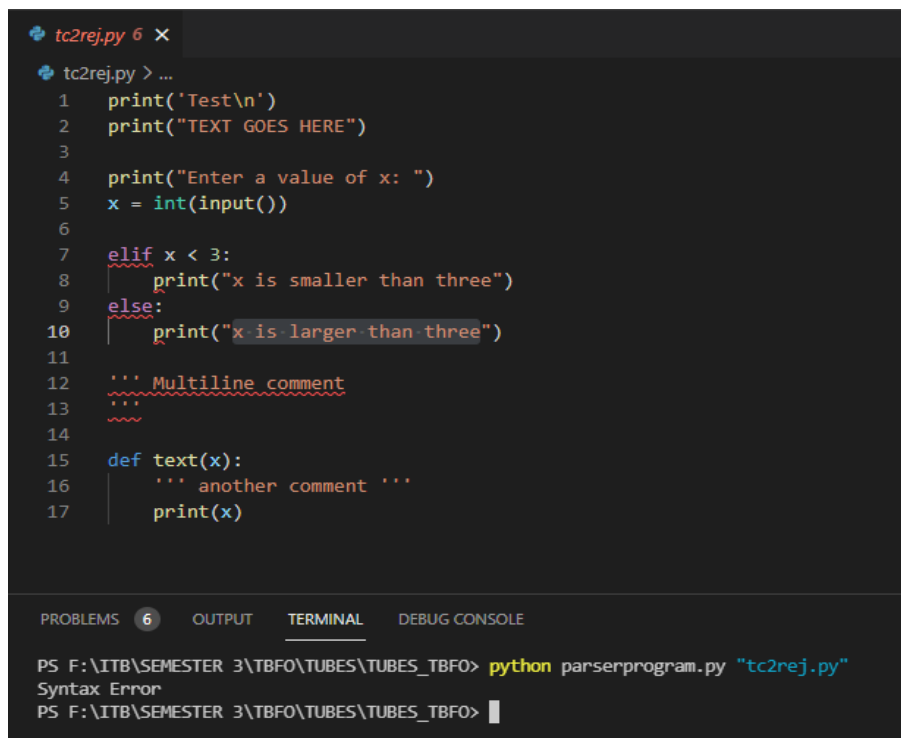


```
tc2acc.py X
tc2acc.py > ...
1  print('Test\n')
2  print("TEXT GOES HERE")
3
4  print("Enter a value of x: ")
5  x = int(input())
6
7  if x == 3:
8      print("x is three")
9  elif x < 3:
10     print("x is smaller than three")
11 else:
12     print("x is larger than three")
13
14 ''' Multiline comment
15 '''
16
17 def text(x):
18     ''' another comment '''
19     print(x)

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE
PS F:\ITB\SEMESTER 3\TBFO\TUBES\TUBES_TBFO> python parserprogram.py "tc2acc.py"
Accepted
PS F:\ITB\SEMESTER 3\TBFO\TUBES\TUBES_TBFO> |
```

Pada test case ini menekankan percabangan (control flow) pada python dan dari gambar di atas potongan kode tersebut diterima.

3.2.4. tc2rej.py

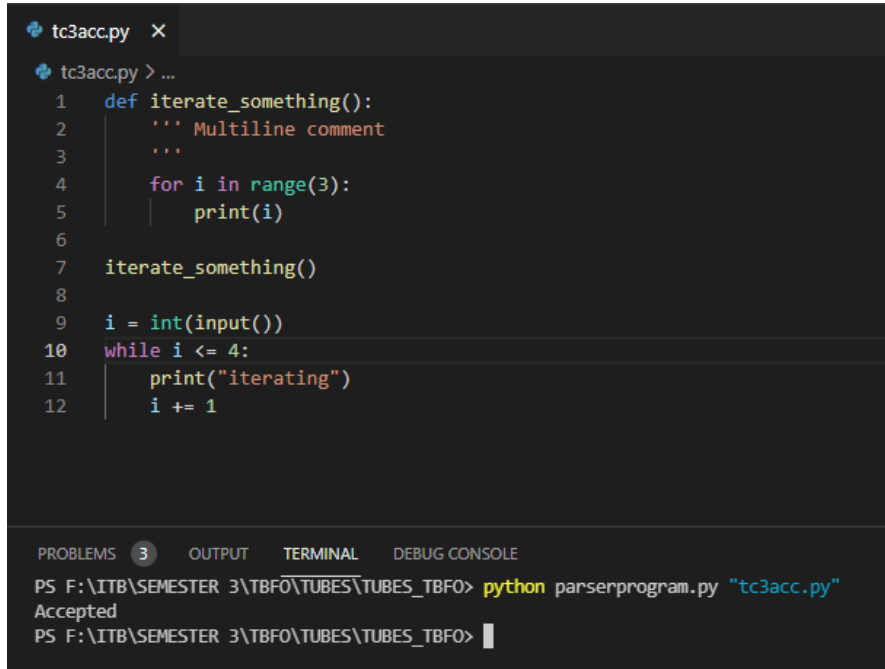


```
tc2rej.py 6 X
tc2rej.py > ...
1  print('Test\n')
2  print("TEXT GOES HERE")
3
4  print("Enter a value of x: ")
5  x = int(input())
6
7  elif x < 3:
8      print("x is smaller than three")
9  else:
10     print("x is larger than three")
11
12 ''' Multiline comment
13 '''
14
15 def text(x):
16     ''' another comment '''
17     print(x)

PROBLEMS 6 OUTPUT TERMINAL DEBUG CONSOLE
PS F:\ITB\SEMESTER 3\TBFO\TUBES\TUBES_TBFO> python parserprogram.py "tc2rej.py"
Syntax Error
PS F:\ITB\SEMESTER 3\TBFO\TUBES\TUBES_TBFO> |
```

Pada test case ini ingin dilakukan pengecekan terhadap percabangan (control flow), dari gambar diatas potongan kode salah karena pada python percabangan harus dimulai dengan “if”, sehingga program akan menampilkan pesan “Syntax Error”.

3.2.5. tc3acc.py

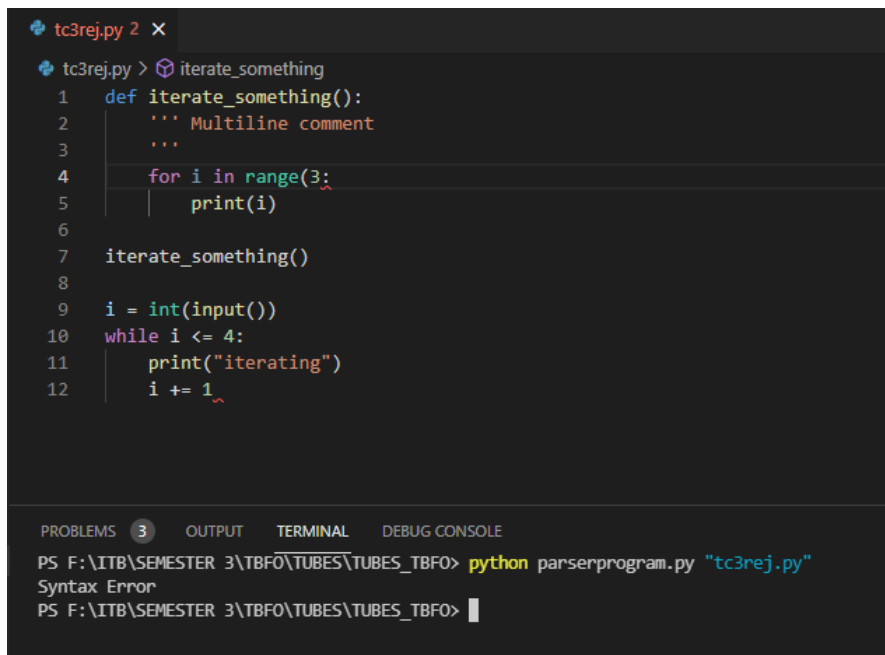


```
tc3acc.py X
tc3acc.py > ...
1  def iterate_something():
2      ''' Multiline comment
3      ...
4      for i in range(3):
5          print(i)
6
7      iterate_something()
8
9      i = int(input())
10 while i <= 4:
11     print("iterating")
12     i += 1

PROBLEMS 3 OUTPUT TERMINAL DEBUG CONSOLE
PS F:\ITB\SEMESTER 3\TBFO\TUBES\TUBES_TBFO> python parserprogram.py "tc3acc.py"
Accepted
PS F:\ITB\SEMESTER 3\TBFO\TUBES\TUBES_TBFO> |
```

Pada test case ini, ingin dilakukan pengecekan terhadap loop dan dari gambar diatas potongan kode tersebut dapat diterima.

3.2.6. tc3rej.py



```
tc3rej.py 2 X
tc3rej.py > iterate_something
1  def iterate_something():
2      ''' Multiline comment
3      ...
4      for i in range(3:
5          print(i)
6
7      iterate_something()
8
9      i = int(input())
10 while i <= 4:
11     print("iterating")
12     i += 1

PROBLEMS 3 OUTPUT TERMINAL DEBUG CONSOLE
PS F:\ITB\SEMESTER 3\TBFO\TUBES\TUBES_TBFO> python parserprogram.py "tc3rej.py"
Syntax Error
PS F:\ITB\SEMESTER 3\TBFO\TUBES\TUBES_TBFO> |
```

Pada test case ini, loop “for” pada python harus memiliki kurung penutup, sehingga dari gambar diatas potongan kode tersebut salah sehingga program memberikan pesan “Syntax Error”.

BAB 4

KESIMPULAN DAN SARAN

4.1.Kesimpulan

Program parserprogram.py yang telah kami buat untuk tugas besar ini dapat menentukan kebenaran suatu sintaks program sesuai dengan Bahasa Python dalam skala yang belum besar. Hal tersebut karena CFG yang kami buat belum memuat seluruh kasus bentukan sintaks Python. Selain itu, terdapat beberapa kasus di mana test case yang seharusnya benar mengembalikan “Syntax Error”, walaupun sudah dimuat pada CFG secara teori. Hal tersebut mungkin dikarenakan hasil konversi CFG menjadi CNF yang belum sesuai.

4.2.Saran

Setelah menjalani proses pembuatan tugas besar ini, berikut merupakan saran yang dapat kami berikan:

1. Pelajari dan dalami konsep yang akan digunakan dari jauh-jauh hari.
2. Memperbanyak test *case* serta testing untuk mendapatkan CFG yang lebih sesuai.
3. Lakukan *testing* dari hal-hal yang simpel, kemudian perbesar *test case* secara perlahan.
4. Mencari algoritma yang sekiranya paling sesuai sebelum membuat program.

Link Repository Github

<https://github.com/mhilmirinaldi/Compiler-Python>

Pembagian Tugas

No.	NIM - Nama	Pembagian Tugas
1.	13520120 – Afrizal Sebastian	Pembuatan parserprogram.py, CFG, dan pengetesan
2.	13520127 – Adzka Ahmadetya Zaidan	Pembuatan CFG, CNF, Test Case, laporan, dan pengetesan.
3.	13520149 – Mohamad Hilmi Rinaldi	Pembuatan CYK, Test Case, dan laporan.