Tugas Besar IF2124 Teori Bahasa Formal dan Automata: *Compiler* Bahasa Python

Anggota Kelompok "tumblr Nutrisari":

Aditya Prawira N 13520049

Felicia Sutandijo 13520050

Christopher Jeffrey 13520055

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2021

DAFTAR ISI

DAFTAR ISI....1
BAB 1....3
A. Finite Automata (FA)....3
B. Deterministic Finite Automata (DFA)....3
C. Context Free Grammar (CFG)....3
Parsing....4
Ambiguitas....5
D. Chomsky Normal Form (CNF)....6
Pembentukan Chomsky Normal Form....6
E. Cocke-Younger-Kasami (CYK)....6
F. Bahasa Python....8
BAB 2....10
A. Finite Automata....10
B. Context Free Grammar (CFG)....10
BAB 3....13
A. Spesifikasi Teknis Program....13
Struktur Data dan Fungsi atau Prosedur....13
Antarmuka....15
B. Screenshot....16
Test 1....16
Test 2....17
Test 3....18
Test 4....19
Test 5....19
Test 6....20
Test 7....20
Test 8....21
Test 9....21
Test 10....22
BAB 5....25
A. Kesimpulan....25
B. Saran....25
LAMPIRAN....26

A. Link Repository Github....26
B. Pembagian Tugas 26
REFERENSI 27

BAB 1
TEORI DASAR

A. Finite Automata (FA)

Finite automata atau finite state automata (automata berhingga state) merupakan suatu model dari suatu sistem yang menerima input dan output diskrit. FA merupakan mesin otomata dari bahas regular. Suatu FA memiliki state yang banyaknya berhingga dan dapat berpindah-pindah dari suatu state ke state lain. Perubahan state ini dinyatakan oleh fungsi transisi. Berbeda dengan CFG, FA tidak memiliki tempat penyimpanan, sehingga kemampuan 'mengingatnya' hanya bisa mengingat state yang terkini. Contoh FA antara lain elevator, text editor, analisa leksikal, dan protokol komunikasi jaringan. FA dapat dibagi menjadi dua jenis berdasarkan pada pendefinisian kemampuan berubah state-statenya, yaitu Deterministic Finite Automata (DFA) dan Non-deterministic Finite Automata (NFA), namun yang akan dibahas selanjutnya hanyalah DFA karena dalam pembuatan parser khususnya analisis kebenaran nama variable python, DFA-lah yang digunakan. Secara formal FA dapat dinyatakan sebagai berikut.

$$\mathbf{M = (Q, \sum, \delta, S, F)}$$

Di mana:
Q = himpunan state/kedudukan
$\sum$= himpunan simbol input/masukan/abjad
δ = fungsi transisi
S = state awal/kedudukan awal (initial state), S ϵ Q
F = himpunan state akhir, F $\cap$ Q (jumlah dapat lebih dari satu)

B. Deterministic Finite Automata (DFA)

Deterministic Finite Automata atau DFA merupakan sebuah jenis FA yang dapat menerima atau menolak sebuah deretan simbol, yang pada kasus ini merupakan nama variabel. Perbedaan DFA dengan NFA ada pada fungsi transisinya. Sebuah FA dikatakan sebagai deterministik bila pada setiap state, untuk setiap simbol masukan, ada tepat satu buah state resultan. Keadaan DFA sebelum dimasukkan simbol apapun selalu berada pada start state yang telah didefinisikan. Kemudian, setiap kali DFA dimasukkan sebuah simbol yang termasuk di dalam alfabet yang diterimanya, state akan berpindah dengan mengacu kepada fungsi transisi (δ). Setelah masukan berakhir, diterima atau tidaknya masukan ditentukan dari keadaan state saat selesai. Bila state saat selesai merupakan salah satu dari state akhir/final state, maka masukan diterima. Sebaliknya, bila state selesai bukan merupakan state akhir/final state, masukan ditolak.

C. Context Free Grammar (CFG)

Context free grammar (CFG) adalah *grammar* formal yang memiliki aturan produksi dalam bentuk $S \rightarrow \alpha$, dengan S adalah simbol nonterminal dan α adalah terminal dan bisa kosong. Secara definisi formal, CFG memiliki 4 komponen, yaitu terminal,

variabel/nonterminal, *start symbol*, dan aturan produksi. Terminal adalah simbol alfabet dari *language* yang didefinisikan. Variabel atau nonterminal adalah himpunan simbol-simbol yang merepresentasikan sebuah *language*. *Start symbol* adalah variabel yang mendefiniskan *language*. Definisi formal CFG dapat dituliskan sebagai berikut.

G = (V, T, P, S)

Di mana:
V = himpunan variabel
T = himpunan terminal
P = himpunan aturan produksi
S = start symbol

Seperti di definisi awal, aturan produksi memiliki bentuk umum "variabel → *string of variables and terminals*". Konvensi penamaan variable dan terminal adalah sebagai berikut,

- A, B, C,… merupakan variabel
- a, b, c,… merupakan terminal
- …, X, Y, Z merupakan terminal atau variabel
- …, w, x, y, z merupakan string dari terminal
- α, β, γ,… merupakan string dari terminal dan/atau variabel

Jika *grammar* G adalah sebuah CFG, maka L(G), bahasa dari G, adalah {w|S =>*w}. Contoh, G_{pal} = ({P}, {0,1}, A, P), dengan P adalah *start symbol*, A adalah produksi yang didefinisikan sebagai A = {P → ε|0|1|0P0|1P1}.

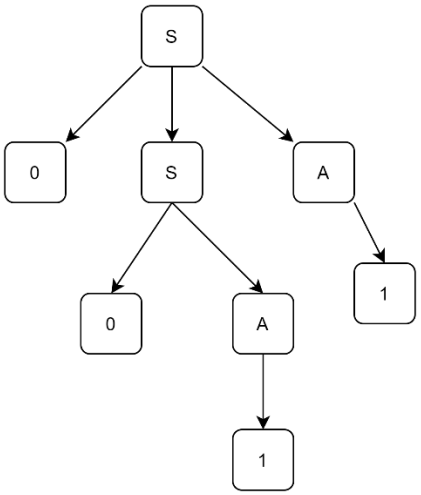
Parsing

Salah satu penggunaan CFG adalah dalam pembentukan parser/analisis suatu sintaks. Kebanyakan sintaks pada *compiler* atau *interpreter* didefinisikan dalam CFG. Ketika menganalisis suatu sintaks, dapat digunakan *derivation tree/parse tree* 'pohon penurunan' untuk mempermudah penurunan *grammar*. Dalam pohon penurunan, akar pohon dimulai dari *start symbol* kemudian dilanjutkan sesuai dengan aturan produksi hingga ujung dari pohon adalah terminal.

Contoh, terdapat G = ({A, B, S}, {0, 1}, P, S) dengan P sebagai berikut:

- S → 0SA | 0A
- A → 1

Parse tree atau pohon penurunan dari G jika masukkannya "0011" adalah

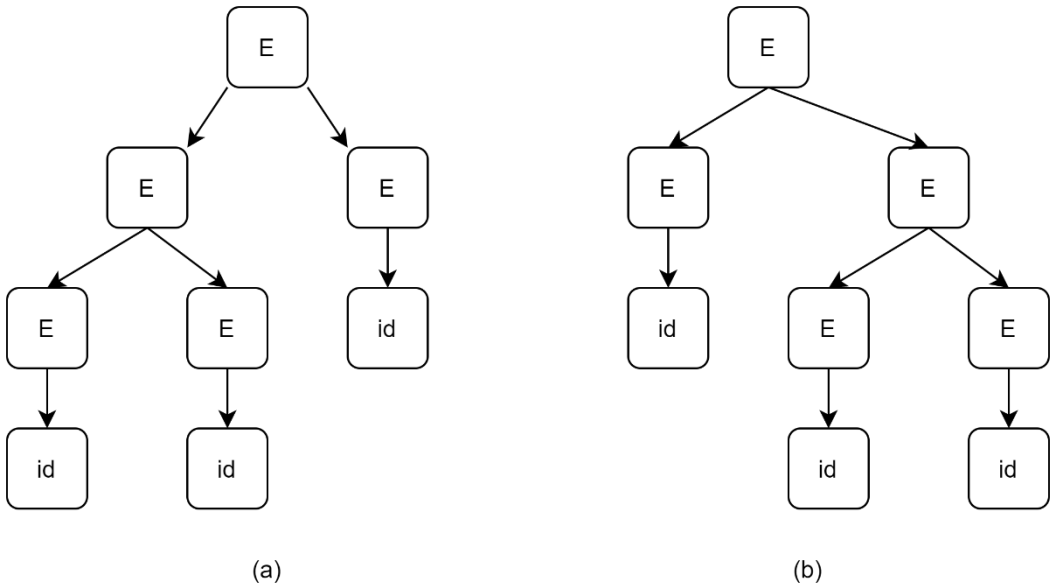


Gambar 1.1 Pohon penurunan pembacaan masukkan “0011”

Dalam menggambarkan pohon penurunan, terdapat dua cara, yaitu *left-most derivation* dan *right-most derivation*. *Left-most derivation* adalah cara yang menurunkan variabel paling kiri terlebih dahulu, sedangkan *right-most derivation* adalah cara yang menurunkan variabel paling kanan terlebih dahulu.

Ambiguitas

Apabila terdapat lebih dari satu pohon penurunan yang berbeda dari suatu masukan/string, maka bahasa tersebut ambigu. Sebagai contoh, terdapat G = ({E}, {id}, E → EE | id, E), jika dibuat pohon penurunan *left-most derivation* dan *right-most derivation*, akan didapatkan pohon seperti berikut



Gambar 1.2 (a) *left-most derivation tree* (b) *right-most derivation tree*

Terlihat bahwa terdapat dua pohon berbeda untuk mendapatkan string yang sama. Maka dari itu, CFG tersebut dikatakan memiliki sifat ambigu.

D. Chomsky Normal Form (CNF)

Chomsky normal form (CNF) merupakan salah satu bentuk CFG. CFG dikatakan memiliki bentuk CNF jika semua bentuk aturan produksinya berupa:

- $A \rightarrow BC$, atau
- $A \rightarrow a$, atau
- $S \rightarrow \varepsilon$

dengan A, B, dan C merupakan nonterminal, a merupakan terminal, S merupakan *start symbol*, dan ε merupakan simbol yang melambangkan string kosong. Artinya, CNF merupakan bentuk dari CFG yang aturan produksinya tidak memiliki produksi tak terpakai, produksi unit, dan produksi ε.

Pembentukan Chomsky Normal Form

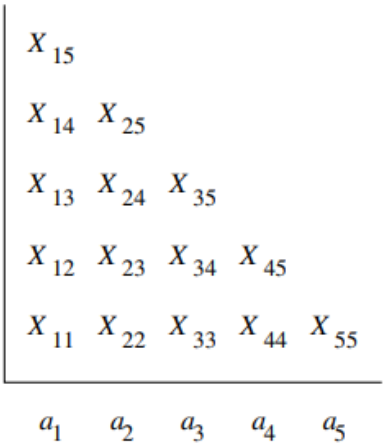
Langkah-langkah pembentukan CNF atau konversi CFG ke CNF adalah sebagai berikut:

1. Menghapus ε-*productions*
2. Menghapus *unit productions*
3. Menghapus *useless variables*, yaitu
   a. Variabel yang tidak menurunkan string terminal
   b. Variabel yang tidak bisa dicapai dari *start symbol*
4. Ubah ke bentuk
   a. $A \rightarrow BC$
   b. $A \rightarrow a$

E. Cocke-Younger-Kasami (CYK)

Algoritma Cocke-Younger-Kasami atau biasa disebut CYK *algorithm* adalah sebuah algoritma yang melakukan *parsing* dari bawah ke atas dan menggunakan pemrograman dinamis. Algoritma CYK hanya bisa melakukan *parsing* pada CFG dalam bentuk CNF. Tujuan dari algoritma CYK adalah mengetes apakah sebuah string dapat diterima atau tidak jika diberikan sebuah bahasa G.

Pada umumnya, algoritma CYK diilustrasikan dalam bentuk tabal. Misalkan terdapat string yang akan dicek dengan panjang 5, maka tabel akan berbentuk seperti berikut:



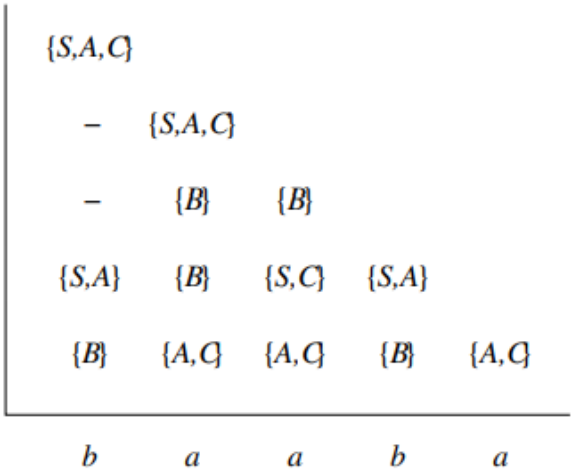
Gambar 1.3 Tabel CYK-Algorithm dengan panjang string 5

Pengisian tabel tersebut dilakukan dengan mengisi baris paling bawah terlebih dahulu, kemudian naik per baris. Baris paling atas dari tabel merupakan akar dari sebuah *parse tree*, sehingga saat akar mengandung *start symbol* string diterima.

Contoh, terdapat CFG dalam bentuk CNF yang didefinisikan sebagai G = ({S, A, B, C}, {a, b}, P, S) dengan P:

- S → AB | BC
- A → AB | a
- B → CC | b
- C → AB | a

Jika kita akan mengecek apakah string “baaba” merupakan bagian dari G, maka akan terbentuk tabel algoritma CYK sebagai berikut



Gambar 1.4 Tabel CYK-Algorithm untuk string “baaba”

Terlihat bahwa terdapat S, yaitu *start symbol* dari G, pada baris teratas tabel sehingga string “baaba” merupakan bagian dari G.

F. Bahasa Python

Python adalah salah satu bahasa pemrograman tingkat tinggi yang ditujukan untuk penggunaan umum. Bahasa Python ditujukan sebagai bahasa yang mudah dibaca, sehingga *formatting* dan sintaksnya kebanyakan menggunakan bahasa Inggris. Dalam membatasi suatu blok kode, bahasa Python memanfaatkan indentasi, berbeda dengan bahasa pemrograman lainnya yang pada umumnya menggunakan simbol semi-colon (;) untuk membatasi suatu blok kode.

Beberapa *statement* Python adalah:

- Pernyataan yang menetapkan nilai pada suatu variabel, menggunakan tanda "`=`"
- Pernyataan `if`, digunakan untuk memberi syarat pada suatu blok kode
- Pernyataan `elif`, singkatan dari else if dan digunakan setelah `if`
- Pernyataan `else`, digunakan setelah `if`
- Pernyataan `while`, akan mengeksekusi blok kode selama pernyataan bernilai benar
- Pernyataan `for`, akan mengiterasi objek yang dapat diiterasi hingga habis
- Pernyataan `try`, memperbolehkan pengecualian di dalam blok kode yang menempel untuk diatasi oleh klausa `except`, pernyataan `try` juga memastikan bahwa blok kode dalam pernyataan `finally` akan dijalankan
- Pernyataan `raise`, akan memunculkan error atau kesalahan dalam kode dan memunculkan ulang error atau kesalahan
- Pernyataan `class`, digunakan untuk mengeksekusi suatu blok kode dan menempelkan namanya pada suatu "kelas", digunakan dalam pemrograman berparadigma objek
- Pernyataan `def`, digunakan untuk mendefinisikan suatu fungsi atau metode
- Pernyataan `with`, akan menutup suatu blok kode dalam manajer konteks
- Pernyataan `break`, akan keluar dari sebuah loop
- Pernyataan `continue`, akan melewati sebuah iterasi dan melanjutkan ke iterasi selanjutnya
- Pernyataan `del`, akan menghapus sebuah variabel
- Pernyataan `pass`, digunakan untuk membentuk sebuah blok kode yang kosong
- Pernyataan `assert`, digunakan dalam proses *debugging* untuk mengecek kondisi yang berlaku
- Pernyataan `yield`, akan mengembalikan sebuah nilai dari fungsi *generator*
- Pernyataan `return`, digunakan untuk mengembalikan sebuah nilai dari fungsi
- Pernyataan `import`, digunakan untuk memasukkan modul file lain untuk digunakan pada program yang sedang dibuat

Kemudian beberapa operator pada Python adalah:

- Operator `<`, untuk mengecek sisi kiri lebih kecil dari sisi kanan
- Operator `>`, untuk mengecek sisi kiri lebih besar dari sisi kanan
- Operator `==`, untuk membandingkan dan mengecek apakah sisi kanan dan kiri bernilai sama

- Operator `!=`, untuk membandingkan dan mengecek apakah sisi kanan dan sisi kiri berbeda
- Operator `<=`, untuk mengecek sisi kiri lebih kecil atau sama dengan sisi kanan
- Operator `>=`, untuk mengecek sisi kiri lebih besar atau sama dengan sisi kanan
- Operator `is`, untuk mengecek apakah sisi kiri dan kanan merupakan objek yang sama
- Operator `*`, untuk mengalikan dua buah angka
- Operator `/`, untuk membagi dua buah angka
- Operator `+`, untuk menambahkan dua buah angka
- Operator `-`, untuk mengurangi dua buah angka
- Operator `%`, untuk mencari sisa bagi (modulo)
- Operator `**`, untuk mempangkatkan angka dengan suatu pangkat
- Operator `//`, untuk membagi angka dan membulatkan hasil ke bawah

Pada Python, beberapa pernyataan memerlukan tanda titik dua (:) untuk menunjukkan bahwa pernyataan berakhir. Beberapa pernyataan tersebut adalah if, else, while, for, try except, finally, class, def. Contoh penggunaannya pada Python adalah seperti berikut:

```
def func():
      if x == 3:
            continue
      elif x == 4:
            print(X is 4)
      else:
            return True
```

Selain itu, Python memiliki aturan penamaan variabel. Aturan tersebut adalah:

- Nama variabel harus diawali sebuah huruf atau karakter underscore (_)
- Nama variabel tidak bisa diawali angka
- Nama variabel hanya bisa mengandung (A-z, 0-9, _)
- Penggunaan huruf kapital pada nama variabel akan berpengaruh. Contoh, "`a = 4`" dan "`A = 3`" adalah dua variabel yang berbeda

Beberapa contoh penamaan variabel yang ilegal:

1. 12Variabel
2. vari-abel
3. variabel 1

Dalam Python, kita dapat menuliskan sebuah komentar dalam kode program dengan menggunakan simbol "#" atau mengawali dan mengakhiri dengan " ''' " (petik tunggal tiga kali) atau ' """ ' (petik ganda tiga kali).

BAB 2
HASIL

A. Finite Automata

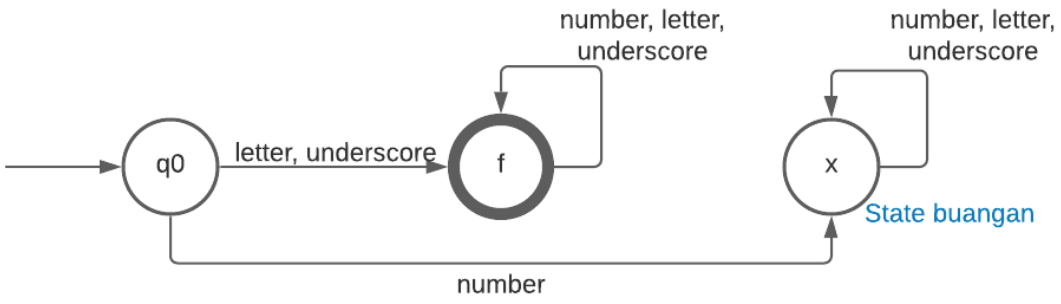
DFA yang digunakan dalam penentuan legalitas nama variabel adalah sebagai berikut:

A = ({q0, x, f}, {number, letter, underscore}, δ, q0, f)

Fungsi transisi (δ):
δ (q0, number) = x
δ (q0, letter) = f
δ (q0, underscore) = f
δ (f, number) = f
δ (f, letter) = f
δ (f, underscore) = f
δ (x, number) = x
δ (x, letter) = x
δ (x, underscore) = x

Di mana:
number = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
letter = {A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W X, Y, Z, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z}
underscore = {_}

DFA tersebut dapat digambarkan menggunakan diagram sebagai berikut.



B. Context Free Grammar (CFG)

Berikut adalah production rules CFG yang dipakai dalam pembuatan parser python.
S → S S | VAR = VARVAL | VAR += VARVAL | VAR -= VARVAL | VAR *= VARVAL | VAR /= VARVAL | IF S | PRINT | WHILE S| FOR S| DEF S RETURN | CLASS S | IMPORT | FROM | COMMENT | VAR INBRACKET | VAR = VAR INBRACKET | METHOD | WITH S | PASS | RAISE;

VARVAL → VAR | VAL | VARVAL OPERATOR VARVAL | INPUT | VARVAL , VARVAL | METHOD | NONE;

OPERATOR → + | - | * | / | %;

VAR → variable;

VAL → angka | VARVAL OPERATOR VARVAL | VARVAL ** VARVAL | VARVAL // VARVAL | (VARVAL) | BOOLEAN | ' STRING '| " STRING " | [VARVAL] | { VARVAL } |[ ] | { } | " " | ' ' ;

EVALUATOR → > | < | = = | ! = | < = | > = ;

STRING → string | STRING STRING;

BOOLEAN → True | False | BOOLEAN and BOOLEAN | BOOLEAN or BOOLEAN | not BOOLEAN | VARVAL is VARVAL | VARVAL EVALUATOR VARVAL;

COMMENT → TRIPLESINGLEQUOTEOPEN STRING TRIPLESINGLEQUOTECLOSE | TRIPLEDOUBLEQUOTEOPEN STRING TRIPLEDOUBLEQUOTECLOSE | # STRING;

TRIPLEDOUBLEQUOTEOPEN → " " " STRING | " " " ;

TRIPLEDOUBLEQUOTECLOSE → " " " | STRING " " " ;

TRIPLESINGLEQUOTEOPEN → ' ' ' STRING | ' ' ' ;

TRIPLESINGLEQUOTECLOSE → ' ' ' | STRING ' ' ' ;

INBRACKET → (METHOD) | (VARVAL) | ( ) | (STRING) ;

EXPRES → (BOOLEAN) | BOOLEAN | INBRACKET;

RAISE → raise METHOD ;

BREAK → break;

PASS → pass;

PRINT → print INBRACKET;

CONTINUE → continue;

ELSE → else : ;

IF → if EXPRES :| IF S ELIF S| IF S ELSE S| IF S RAISE ;

ELIF → elif EXPRES : | ELIF S ELIF S | ELIF S ELSE S;

TYPE → str | float | int | double;

INPUT → input INBRACKET | TYPE (INPUT);

RANGE → range INBRACKET;

FROM → from VARVAL IMPORT ;

WHILE → while EXPRES : ;

FOR → for VAR in STRING : | for VAR in RANGE : ;

RETURN → return BOOLEAN | return VARVAL;

CLASS → class VAR :;

NONE → None;

IMPORT → import VARVAL as VAR | import VARVAL;

METHOD → VAR INBRACKET | VAR INBRACKET . METHOD | VAR . METHOD ;

DEF → def VAR INBRACKET : ;

WITH → with S as VAR : ;

BAB 3
IMPLEMENTASI DAN PENGUJIAN

A. Spesifikasi Teknis Program

algoritma python parser kami bagi menjadi beberapa langkah besar.

a. Membuat CFG
b. Mengubah CFG menjadi CNF
c. Mengubah CNF menjadi python dictionary
d. Untuk tiap line program yang akan dicek,
   - Akan dipecah menurut sintax yang kita tentukan (kami sebut mengubahnya menjadi 'token')
   - Pada bagian awal dari CYK, dilakukan pengecekan validitas dari token untuk dapat dianggap sebagai variabel
   - Dilakukan pengecekan dengan algortima CYK
e. Hasil dari CYK akan mewakilli kevalidan dari tiap-tiap line, lalu seluruh line akan dievaluasi kevalidannya kembali.

Struktur Data dan Fungsi atau Prosedur

Untuk *langkah a* algoritma(membuat CFG), data kami simpan di dalam sebuah file txt. Pemilihan ini beralibikan keinginan untuk meniympan CFG dengan cara yang paling sederhana, untuk memudahkan kami dalam membuat CFG yang benar. File kami simpan dengan nama `cfg.txt`

Untuk *langkah b*, pengubahan CFG menjadi CNF dilakukan dengan menggunakan sebuah fungsi, yang membaca file cfg di `cfg.txt` lalu melakukan pengubahan menjadi CNF, dan menuliskannya dalam sebuah file txt kembali. File tersebut kami simpan dalam file cnf.txt. Alasan pemilihan menuliskan cnf.txt adalah memudahkan melakukan pendebugkan jika terjadi kesalahan pada algoritma *langkah b* atau algoritma di langkah-langkah selanjutnya yang memerlukan kita melakukan pengecekan secara manual terhadap CNF. Langkah ini dilakukan dengan menggunakan prosedur `CFGtoCNF` yang disimpan di dalam file `CFGtoCNF.py`.

Pada *langkah c*, CNF diubah menjadi dictionary python. Alasan pengubahan ini adalah, kami memerlukan suatu data yang lebih mudah untuk melakukan evaluasi berdasarkan CNF. Namun alasan tersebut masih membuka opsi penyimpanan data dengan cara selain dictionary, misalnya array. Alasan pemilihan dictionary adalah penggunaan kami. Hampir seluruh penggunaan CNF pada program kami adalah pencarian. Dictionary python memiliki kompleksitas O(1) dalam hal pencarian. Fakta tersebut merupakan pendorong utama kami menggunakan dictionary. Langkah ini diimplementasikan dalam fungsi `CNFtoCNFdict`. Fungsi tersebut mengembalikan (return) CNF yang ada di dalam `cnf.txt` dalam bentuk dictionary python. Fungsi `CNFtoCNFdict` disimpan di dalam file `CNFtoCNFdict.py`.

Dictionary pada python terdiri atas dua bagian, keys dan value. Untuk keys, kami isi dengan *sisi kanan dari CNF* dan untuk value, kami isi dengan *sisi kiri dari CNF*. Berikut contohnya,

cnf.txt	CNF dictionary
A -> BC ; E -> DE | BC | a;	{“BC”: [“A”,”E”], “DE” : [“E”], “a” :[“E”]}

Pada value di dictionary, kami menggunakan array karena program memungkinan adanya lebih dari satu buah nonterminal yang menghasilkan *dua non terminal atau satu terminal* yang sama. Dapat dilihat bahwa dari cnf.txt ke CNF dictionary seakan-akan *dibalik*. Keputusan ini dibuat karena penggunaan CNF yang kita lakukan adalah pencarian nonterminal *yang ditunjuk dua non terminal atau satu terminal.* Informasi yang kita miliki adalah dua non terminal atau satu terminal, jadi informasi tersebutlah yang dijadikan key agar kita bisa memanfaatkan keunggulan python dictionary,
Pada *langkah d bagian 1*, line akan diubah menjadi ‘token’ yaitu line yang sudah dipisah menurut sintaks yang sudah kami tentukan. Misalnya `x = 3` akan diubah menjadi `'x', '=' ,'3'`. Pengubahan ini dilakukan terlebih dahulu karena python memiliki aturan sintaks sendiri. Misalnya `'var1'` merupakan satu kesatuan, bukan dipisah menjadi `'var', '1'`, atau `'v', 'a', 'r', '1'`. Setelah dilakukan pemisahan (atau disebut diubah menjadi token), barulah akan diterapkan algoritma CYK. Penentuan sintaks ini bisa saja dilakukan di dalam CFG, namun kami memilih untuk sebisa mungkin tidak melakukan hal tersebut, karena akan menjadikan CFG relatif lebih rumit. Program ini diimplementasikan dengan fungsi `tokenizer`. Fungsi tersebut menerima sebuah string, lalu mengembalikan string tersebut dengan yang sudah dibagi berdasarkan sintaks yang kami inginkan, dalam sebuah array. Kami memilih array karena data yang diberikan memiliki jumlah jamak yang tidak pasti, dan tipe data array merupakan tipe yang relatif paling umum, dan sudah bisa memenuhi kriteria yang kami butuhkan. Fungsi `tokenizer` disimpan di dalam file `tokenizer.py`.

Token akan dicek validitasnya untuk menjadi variabel pada *langkah d bagian 2*. Pengecekan ini dilakukan dengan pola pikir FA (tepatnya DFA pada kasus ini) yang telah dijelaskan pada bagian hasil. Fungsi `isVarValid` menerima input berupa kata yang akan dievaluasi validitasnya menurut aturan variabel di python dengan cara membaginya menjadi per huruf, dan tiap huruf sebagai input FA yang akan menentukan state FA. Jika berhasil mencapai finalState berdasarkan FA yang kami tentukan, artinya kata tersebut valid dan fungsi akan mengembalikan True. Jika tidak, akan mengembalikan False. Agar lebih rapi, Aturan FA (seperti state, transition) dituliskan pada sebuah file eksternal bernama `dfa.txt`. Fungsi `isVarValid` dituliskan di dalam file `FA.py`, yang juga berisi fungsi `readDFA` untuk membaca dfa.txt menjadi tipe data yang ada di python.

Selanjutnya *langkah d bagian 3*. Disini tiap line yang sudah di ubah menjadi token akan di evaluasi berdasarkan CNF dictionary yang didapat pada *langkah c*. Untuk cara pengerjaan, tidak ada perubahan signifikan dari algoritma CYK umum. Algoritma CYK

akan membuat tabel (atau array dua dimensi, atau matriks), yang masing-masing selnya berisi set. Kami memilih set karena set akan mencegah terbentuknya data duplikat, dan kami tidak membutuhkan data duplikat. Seperti algoritma CYK yang seharusnya, akan dilakukan pengecekan tiap *level* hingga mencapai bagian akhir. Akan didapatkan satu buah set di akhir, yang akan berisi (atau juga kosong) nonterminal yang dapat membentuk line(dalam bentuk token) yang didapat sebagai input. Set tersebut akan menjadi return dari fungsi ini, yang kami implementasikan dengan fungsi `CYK` yang menerima dua buah argumen, line yang sudah diubah menjadi token dan CNF dictionary. Fungsi `CYK` disimpan di dalam file `CYK.py`.

Terakhir di *langkah e*. Pada bagian ini, set yang merupakan output dari masing-masing line yang telah di evaluasi dengan CYK akan di proses berdasarkan isi dari set tersebut, untuk ditentukan validitas dari kumpulan baris kode yang merupakan input sebenarnya(keseluruhan baris). Berdasarkan isi dari set, akan dilakukan manipulasi stack. Contohnya jika isi dari set adalah IF, maka akan dilakukan push ke dalam stack. Jika isi dari set adalah ELSE, akan dilakukan pop terhadap stack(dengan syarat tertentu, misalnya top of stack harus IF). Keuntungan dari pengaplikasian ini adalah kita dapat mengetahui jika tindakan ilegal terjadi, misalnya terdapat ELSE tanpa IF, atau RETURN tanpa DEF (penggunaan huruf besar disini dilakukan untuk menyesuaikan dengan kode program kami, dan melambangkan nonterminal yang menjadi isi dari set). Bagian ini sekaligus menjadi 'pintu masuk' dari parser python kami, yang diimplemnetasikan dalam file `parserprogram.py`. program ini menerima argumen *nama dari file yang akan di dilakukan parser*, dan dijalankan dengan menggunakan python di terminal. Contohnya `python .\parserprogram.py .\testcase.py`, dengan `testcase.py` terletak di dalam folder yang sama dengan parser program dan berisi kode python yang akan di cek validitasnya.

Antarmuka

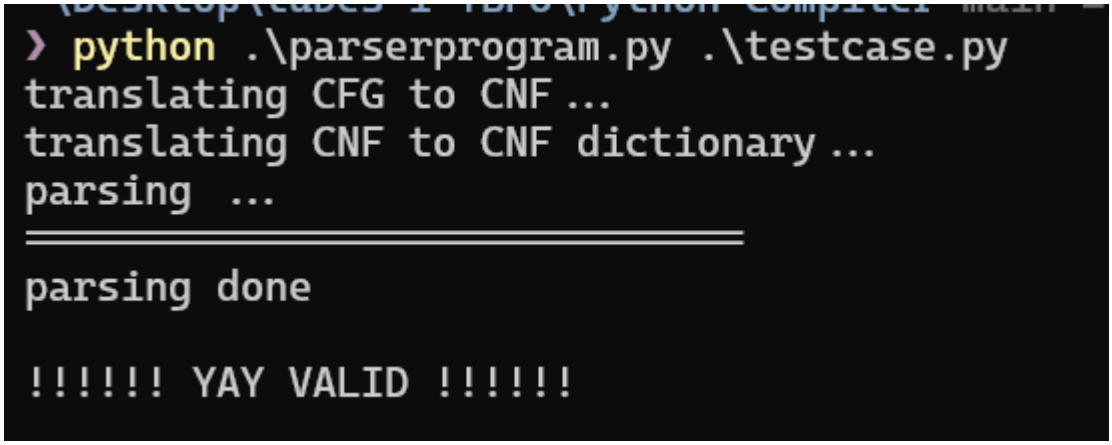
Berikut seluruh file program

```
parserprogram.py
cfg.txt
cnf.txt
dfa.txt
CFGtoCNF.py
CNFtoCNFdict.py
FA.py
CYK.py
tokenizer.py
```

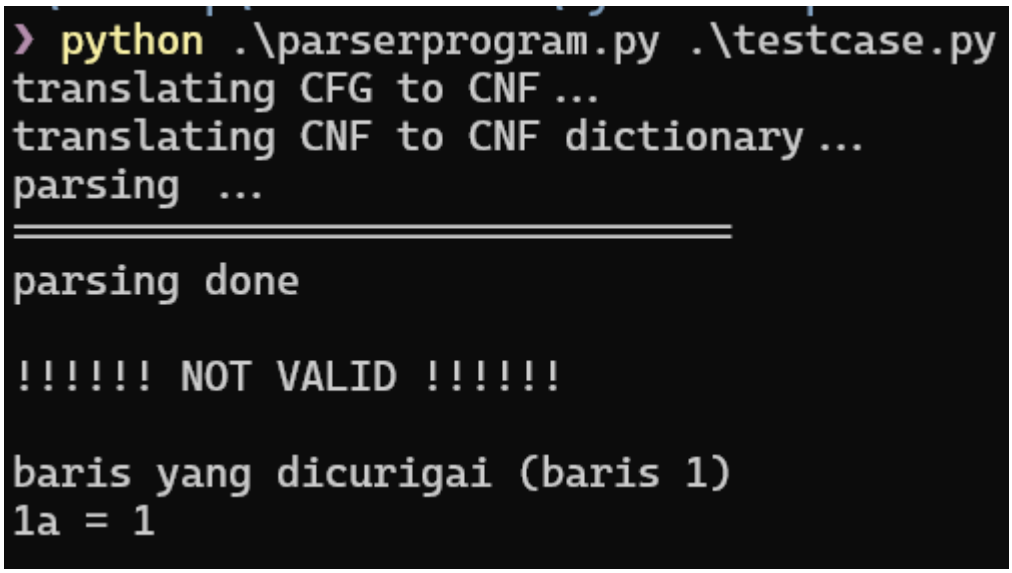
berikut cara menjalankan program
masuk ke directory yang sama dengan parserprogram.py. letakkan file yang berisi kode python yang akan dievaluasi validitasnya(misalkan bernama testcase.py). buka terminal dan jalankan kode berikut

```
python .\parserprogram.py .\testcase.py
```

jika program parser berjalan dengan baik dan testcase berisi kode yang valid, outputnya adalah sebagai berikut



Kode valid dapat dilihat dari output `!!!!!! YAY VALID !!!!!!`
Jika program parser berjalan dengan baik dan testcase berisi kode yang tidak valid, outputnya adalah sebagai berikut



Kode dapat diketahui tidak valid berdasarkan output `!!!!!! NOT VALID !!!!!!`, sedangkan output lainnya adalah informasi tambahan mengenai keberjalanan program parser dan kode yang tidak valid serta lokasinya.

B. Screenshot

Bagian ini berisi uji coba program terhadap beberapa file.

Test 1

Berikut isi file test 1

```python
# file yang digunakan untuk testing. berisi kode python yang  valid

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"
```

Berikut hasil program terhadap file test 1

```
❯ python .\parserprogram.py .\testcase\test1.py
translating CFG to CNF ...
translating CNF to CNF dictionary ...
parsing  ...
==========================
parsing done

!!!!!! YAY VALID !!!!!!
```

File yang terdapat pada test 1 sudah sesuai dengan kriteria python, sehingga mendapatkan penilaian valid menurut parser.

Test 2

Berikut isi file test 2

```python
# file yang digunakan untuk testing. berisi kode python yang tidak valid

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"
```

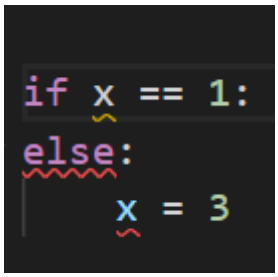
Berikut hasil program terhadap file test 2

```
> python .\parserprogram.py .\testcase\test2.py
translating CFG to CNF ...
translating CNF to CNF dictionary ...
parsing ...
================================
parsing done

!!!!!! NOT VALID !!!!!!

baris yang dicurigai (baris 7)
x + 2 = 3

KONDISI STACK
DEF (baris 3)
```

Pada testcase 2, terdapat line ‘x + 2 = 3’. Line ini tidak valid, dan program menyadarinya. Pada bagian lain dari program juga tedapat line yang tidak valid, contohnya tepat setelah line tersebut, terdapat sebuah if tanpa menggunakan ‘:’. Program membaca per line, dan berhenti ketika menemukan masalah. Sehingga line tersebut belum di evaluasi. Terdapat keterangan stack juga, untuk membantu memberikan detail lokasi.

Test 3

Berikut isi file test 3



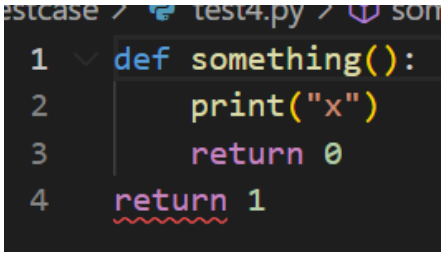
Berikut hasil program terhadap file test 3

```
> python .\parserprogram.py .\testcase\test3.py
translating CFG to CNF ...
translating CNF to CNF dictionary ...
parsing ...
================================
parsing done

!!!!!! NOT VALID !!!!!!

baris yang dicurigai (baris 3)
else:
```

Program tidak valid karena terdapat sebuah if yang tidak diberikan sebuah statement setelahnya. Perbaikannya bisa diberikan statement seperti ‘x = 3’ atau ‘pass’

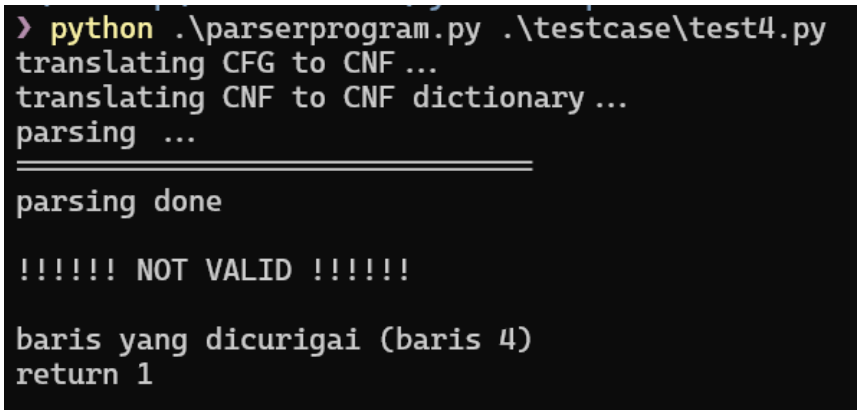
Test 4

Berikut isi file test 4



```
1  def something():
2      print("x")
3      return 0
4  return 1
```

Berikut hasil program terhadap file test 4

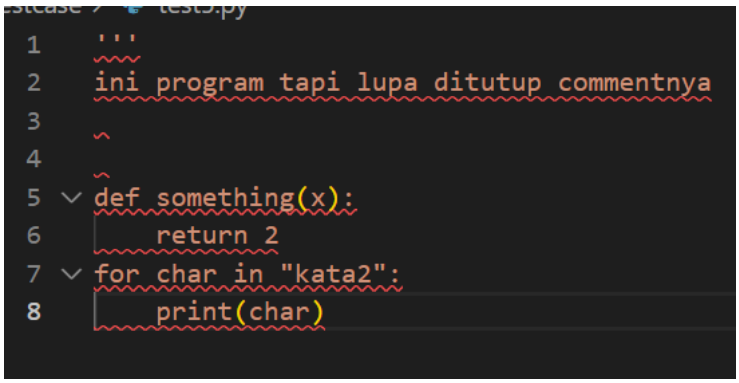


```
❯ python .\parserprogram.py .\testcase\test4.py
translating CFG to CNF ...
translating CNF to CNF dictionary ...
parsing  ...
==============================
parsing done

!!!!!! NOT VALID !!!!!!

baris yang dicurigai (baris 4)
return 1
```

Pada test case tersebut, test 4 tidak valid karena terdapat sebuah return tanpa fungsi atau prosedur yang ‘membungkusnya’.

Test 5

Berikut isi file test 5



```
1  '''
2  ini program tapi lupa ditutup commentnya
3  ^
4  ^
5  def something(x):
6      return 2
7  for char in "kata2":
8      print(char)
```

Berikut hasil program terhadap file test 5

```
❯ python .\parserprogram.py .\testcase\test5.py
translating CFG to CNF ...
translating CNF to CNF dictionary ...
parsing  ...
================================
parsing done

!!!!!! NOT VALID !!!!!!

stack tidak kosong.
ada  TRIPLESINGLEQUOTEOPEN tanpa penutup
```

Program tidak valid karena terdapat sebuah comment yang dibuka, namun tidak ditutup

Test 6

Berikut isi file test 6

```
1
2  if 3 == 4:
3      4var = 3
4  else:
5      x = 1
```

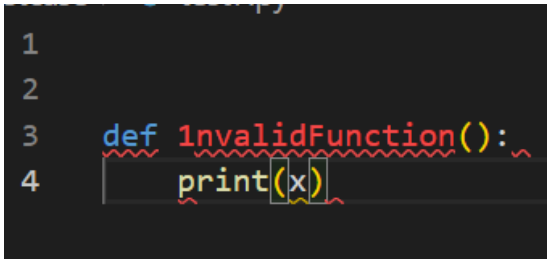
Berikut hasil program terhadap file test 6

```
❯ python .\parserprogram.py .\testcase\test6.py
translating CFG to CNF ...
translating CNF to CNF dictionary ...
parsing  ...
================================
parsing done

!!!!!! NOT VALID !!!!!!

baris yang dicurigai (baris 3)
4var = 3

KONDISI STACK
IF (baris 2)
```

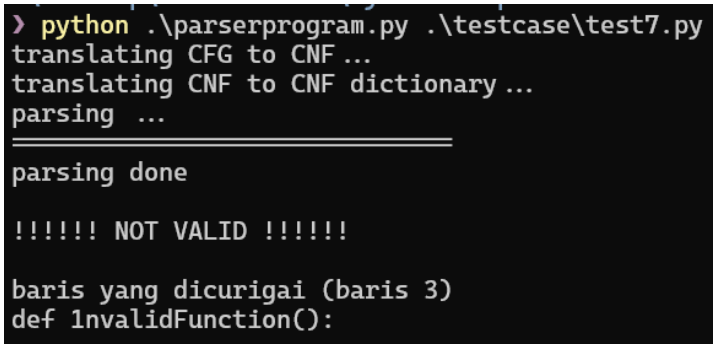
Program tidak valid karena terdapat kesalahan penamaan variabel. Variabel di python tidak boleh diawali dengan angka, sedangkan variabel pada program adalah ‘4var’, dan cukup jelas bahwa karakter pertama variabel tersebut adalah angka. Sehingga program tersebut tidak valid.

Test 7

Berikut isi file test 7



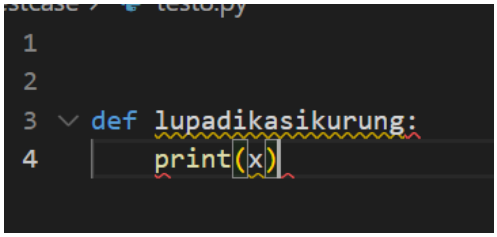
Berikut hasil program terhadap file test 7



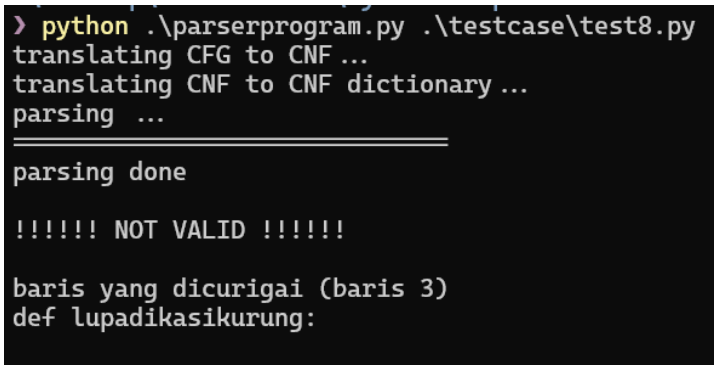
Mirip seperti kesalahan penamaan pada variabel di testcase sebelumnya, file ini juga salah pada penamaan fungsi atau prosedur. Seharusnya tidak boleh diawali dengan angka, sedangkan penamaan fungsi atau variabel disini diawali dengan angka (yaitu 1).

Test 8

Berikut isi file test 8



Berikut hasil program terhadap file test 8



Pada test 8, pada bagian fungsi atau prosedur seharusnya diikuti dengan tanda kurung ‘()’. Namun pada fungsi atau prosedur tersebut tidak terdapat kurung, sehingga tidak valid.

Test 9

Berikut isi file test 9

```
ini tulisan yang lupa dikomen
for a in car:
    print(a)
```

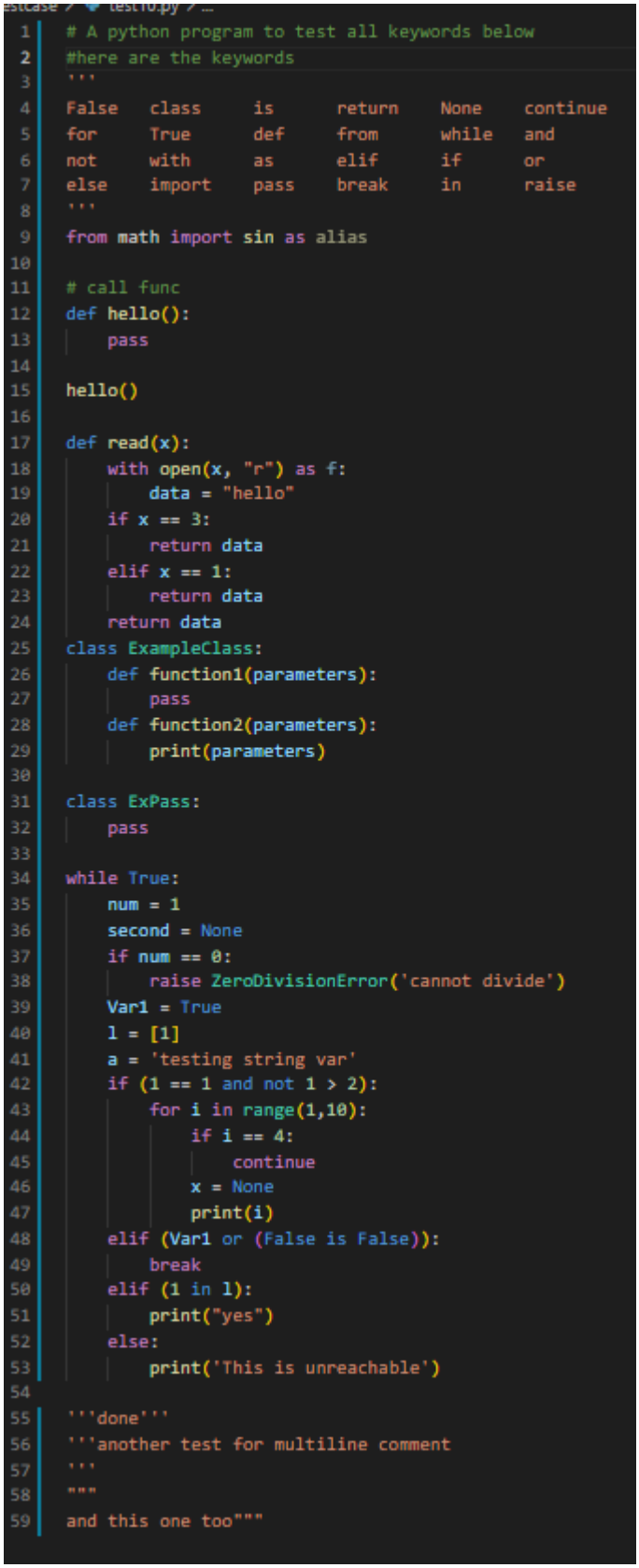
Berikut hasil program terhadap file test 9

```
❯ python .\parserprogram.py .\testcase\test9.py
translating CFG to CNF ...
translating CNF to CNF dictionary ...
parsing  ...
==================================

parsing done

!!!!!! NOT VALID !!!!!!

baris yang dicurigai (baris 1)
ini tulisan yang lupa dikomen
```

Pada fungsi tersebut terdapat sebuah tulisan yang bukan merupakan perintah valid pada python, yang dalam dunia nyata biasanya seharusnya merupakan komentar. Karena tidak diberi tanda komentar, program tidak valid.

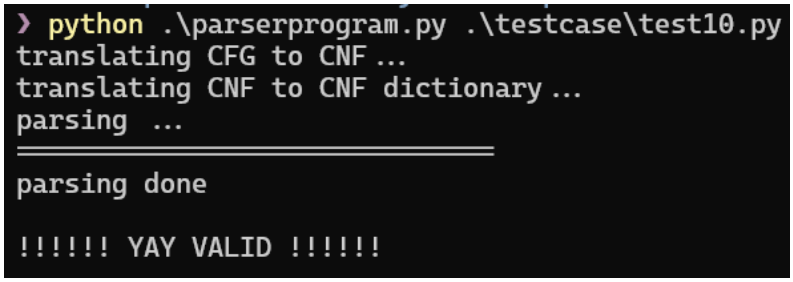
Test 10

Berikut isi file test 10



```
# A python program to test all keywords below
#here are the keywords
'''
False   class     is      return    None    continue
for     True      def     from      while   and
not     with      as      elif      if      or
else    import    pass    break     in      raise
'''
from math import sin as alias

# call func
def hello():
    pass

hello()

def read(x):
    with open(x, "r") as f:
        data = "hello"
    if x == 3:
        return data
    elif x == 1:
        return data
    return data
class ExampleClass:
    def function1(parameters):
        pass
    def function2(parameters):
        print(parameters)

class ExPass:
    pass

while True:
    num = 1
    second = None
    if num == 0:
        raise ZeroDivisionError('cannot divide')
    Var1 = True
    l = [1]
    a = 'testing string var'
    if (1 == 1 and not 1 > 2):
        for i in range(1,10):
            if i == 4:
                continue
            x = None
            print(i)
    elif (Var1 or (False is False)):
        break
    elif (1 in l):
        print("yes")
    else:
        print('This is unreachable')

'''done'''
'''another test for multiline comment
'''
"""
and this one too"""
```

Berikut hasil program terhadap file test 10



Pada test 10, terdapat beberapa keyword python yang diujikan. Karena semua perintah python sesuai dengan ketentuan, maka parser menyatakan program valid.

BAB 5
KESIMPULAN DAN SARAN

A. Kesimpulan

Implementasi CFG dan FA untuk mengevaluasi sintaks pada pemrograman cukup sulit karena sangat banyak aturan produksi yang harus dikerjakan. Meskipun begitu, program yang kami buat cukup baik untuk menangani beberapa kasus kecil. Akan tetapi, untuk kasus besar, program yang kami buat masih mengalami beberapa kendala dan kesalahan evaluasi. Hal tersebut mungkin terjadi karena abstraksi pembuatan CFG kami belum mengatasi semua kemungkinan asli pada sintaks bahasa Python.

B. Saran

Sebaiknya saat membuat program, terutama dalam pembuatan fungsi-fungsi perantara, dilakukan uji coba dengan membuat driver sehingga permasalahan dan bug akan terdeteksi di awal. Selain itu, sebaiknya memperbanyak referensi bahan ajar terkait CFG, CNF, dan CYK.

LAMPIRAN

A. Link Repository Github

https://github.com/FelineJTD/Python-Compiler/tree/main

B. Pembagian Tugas

Nama	NIM	Tugas
Aditya Prawira N	13520049	Pembuatan CFG cfg.txt, algoritma tokenizer.py, CFGtoCNF.py, CYK.py, pembuatan laporan
Felicia Sutandijo	13520050	Pembuatan FA dfa.txt, CFG cfg.txt, algoritma FA.py, CFGtoCNF.py, test case, pembuatan laporan
Christopher Jeffrey	13520055	Pembuatan CFG cfg.txt, algoritma CFGtoCNF.py, CNFtoCNFdict.py, CYK.py, test case, pembuatan laporan

REFERENSI

amanb, “Writing a lexer for a new programming language in python”, diakses pada 21 November 19.24 WIB, https://stackoverflow.com/questions/55571086/writing-a-lexer-for-a-new-programming-language-in-python

Geeks for geeks, “Cocke–Younger–Kasami (CYK) Algorithm”, diakses pada 20 November 2021 pukul 17.16 WIB, https://www.geeksforgeeks.org/cocke-younger-kasami-cyk-algorithm/

Programiz, “List of Keywords in Python”, diakses pada 22 November 18.33 WIB, List of Keywords in Python Programming (programiz.com)

pythonmembers.club, “building a lexer in python – a tutorial”, diakses pada 21 November 2021 pukul 20.47 WIB, https://medium.com/@pythonmembers.club/building-a-lexer-in-python-a-tutorial-3b6de161fe84