

Tugas Besar
IF2124 Teori Bahasa Formal dan Automata

PYTHON COMPILER



Disiapkan oleh:

Kelas 01 / Kelompok Y X G Nubes

Indra Febrio Nugroho	135 18 016
Anindya Prameswari Ekaputri	135 18 034
Fadhil Muhammad Rafi'	135 18 079

Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2019

DASAR TEORI

A. Python

Python adalah bahasa pemrograman yang interaktif dan berorientasi objek, diciptakan oleh Guido van Rossum pada tahun 1990. Bahasa *Python* terkenal akan penggunaan *whitespace* dan kesederhanaannya, yang membuat pengguna dapat fokus menyelesaikan masalah dan menciptakan algoritma, bukan memusingkan soal bahasa.

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.

Contoh masukan dari bahasa Python :

```
nama = input("Masukkan nama Anda: ")
```

Contoh keluaran dari bahasa Python :

```
print ("Halo", nama, "!")
```

B. Context Free Grammar

Context free language adalah sebuah bahasa yang memiliki cakupan lebih luas daripada *regular language*, yang menggunakan notasi rekursif bernama *context free grammar* (CFG). CFG memiliki empat komponen penting:

1. Terminal (T), berisi simbol-simbol yang akan menjadi *string* dalam sebuah bahasa
2. Variabel atau non terminal (V) yang mendefinisikan satu set bahasa
3. *Start symbol* (S) adalah salah satu dari variabel yang akan dipanggil pertama kali
4. Aturan produksi (P) merupakan definisi rekursif suatu bahasa. Suatu aturan produksi memiliki bentuk:

$$V1 \rightarrow X$$

di mana V1 adalah *head*, sebuah variabel dan X, *body* dari aturan, dapat berupa terminal atau variabel lagi yang akan membentuk sebuah *string*.

Komponen CFG biasanya dinotasikan dengan *four-tuple* sebagai $G = (V, T, P, S)$. Misalnya, *grammar* $G = (\{P\}, \{0, 1\}, A, P)$ memiliki sebuah variabel P, dua terminal 0 dan 1, aturan produksi A, dan *start symbol* P.

Grammar kemudian dapat digunakan untuk menentukan apakah sebuah *string* termasuk dalam suatu *language* atau tidak. Untuk mengetahuinya, digunakan *parse tree* yang menggambarkan penurunan suatu *string*.

C. **Chomsky Normal Form**

Chomsky Normal Form (CNF) merupakan salah satu bentuk normal yang sangat berguna untuk Context Free Grammar (CFG). Chomsky Normal Form dapat dibuat dari sebuah Context Free Grammar yang telah mengalami penyederhanaan yaitu penghilangan produksi *useless*, unit, dan ϵ . Dengan kata lain, suatu tata bahasa bebas konteks dapat dibuat menjadi Chomsky Normal Form dengan syarat tata bahasa bebas konteks tersebut:

1. Tidak memiliki produksi *useless*
2. Tidak memiliki produksi unit
3. Tidak memiliki produksi ϵ

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

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

D. **Cocke—Younger—Kasami**

Di dalam dunia ilmu komputer, algoritma Cocke—Younger—Kasami (CYK) adalah algoritma parsing untuk Context Free Grammar.

Versi standar dari CYK hanya dapat dijalankan pada context-free grammars yang ditulis dalam Chomsky normal form (CNF). Bagaimanapun juga, context-free grammar apapun dapat diubah ke dalam format CNF yang mengekspresikan bahasa yang sama (Sipser 1997).

Algoritma CYK menjadi cukup penting dikarenakan efisiensinya yang tinggi dalam berbagai situasi. Hal ini menjadikan algoritma CYK sebagai salah satu algoritma parsing yang paling efisien dalam hal kompleksitas asimptotik kasus terburuk, meskipun ada algoritma lain dengan waktu berjalan rata-rata yang lebih baik dalam banyak skenario praktis.

Contoh grammar:

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

Gambar 1.1 Contoh Grammar

Kalimat *she eats a fish with a fork* dianalisis menggunakan CYK.

CYK table

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

Gambar 1.2 Analisis kalimat dengan CYK

ANALISIS PERSOALAN

Secara umum, untuk dapat membuat menyelesaikan tugas besar atau compiler Python, kami harus terlebih dahulu membuat *grammar* dalam CFG yang memuat tata bahasa dan kata kunci bawaan Python yang didefinisikan di dalam spesifikasi tugas besar. Kami membuat grammar dalam CFG terlebih dahulu agar lebih mudah dalam mengimplementasikan bahasa Python ke dalam aturan grammar. Setelah melakukan pembuatan CFG, konversikan ke dalam bentuk CNF untuk nantinya digunakan sebagai masukan dari CYK_Parser. Sebelum itu, kami membaca input bahasa dengan Lexer agar masukan bahasa dibagi-bagi menjadi token-token yang dapat dicocokkan dengan grammar CNF yang telah dibuat.

Untuk membuat grammar, kami menganalisis beberapa bahasa yang diterima oleh Python, yaitu:

1. IF Statement : *if (elmt_if) :*
2. ELIF Statement : *IF Statement :*
Elif (elmt_elif) :
3. ELSE Statement : *IF Statement*
Elif Statement
else :
4. Function Def : *def -string- (-parameters-) :*
5. False, None, True
6. and, or, not
7. Traversal For dan While
8. Arithmetical operation
9. Comparison operation
10. Import Statement
11. Break, return, pass, raise
12. Class def

A. Grammar CFG

Untuk membuat sebuah CFG, kami menentukan komponen-komponen penyusunnya terlebih dahulu, yaitu Terminal, Non Terminal, Start Symbol, dan Production Rule. Pada laporan ini kami tidak mencantumkan production rule dari CFG yang kami buat karena kurang efektif apabila ditulis semua. Production rule yang kami buat dapat dilihat pada folder yang akan dikumpulkan pada pranala tugas besar ini.

1. Terminal

Terminal pada CFG kami terdiri dari 40 buah yang mendefinisikan kata kunci, beberapa tanda baca, dan operator. Di antaranya yaitu FALSE, NONE, TRUE, FALSE, AND, AS, BREAK, CLASS, CONTINUE, DEF, ELIF, ELSE, FOR, FROM, IF, IMPORT, IN, IS, NOT, OR, PASS, RAISE, RETURN, WHILE, WITH, COMPARE, NUMBER, IDENTIFIER, COMPARISON, ASSIGN, ARITHMETIC, COLON, DOT, COMMA, NEWLINE, LP, RP, LSB, RSB, COMMENT, dan COMMENT_MULTILINE.

Terminal merupakan karakter yang membentuk string. String yang dihasilkan kemudian akan diperiksa menggunakan algoritma CYK apakah merupakan bagian dari bahasa yang tertentu, dalam hal ini yaitu Python.

2. Non Terminal

Non Terminal pada CFG kami terdiri dari 139 buah yang merupakan set bahasa yang dapat dibentuk berdasarkan bahasa Python.

3. Start Symbol

Start symbol dari grammar kami yaitu S, dimana S nantinya akan kemudian memproses kalimat-kalimat yang mungkin dibuat dalam Python. Beberapa contoh dari kalimat tersebut adalah import, conditionals, loop, class, def, dan comment.

Grammar yang kami buat nantinya akan dikonversi menjadi CNF agar dapat dijadikan parameter masukan CYK Parser.

B. Grammar CNF

Seperti yang telah didefinisikan, CNF adalah CFG yang telah disederhanakan dan disesuaikan dengan aturan dari CNF. Aturan tersebut yaitu tidak adanya *useless production*, *unit production*, dan *epsilon production*. Produksi CNF hanya dapat menghasilkan dua Variabel dan/atau menghasilkan terminal.

Ada beberapa langkah untuk mengonversi CFG ke CYK, yaitu:

1. Mengeliminasi *useless production*. *Useless production* adalah aturan yang tidak akan dipanggil (tidak terjangkau dari *start state*) atau tidak menghasilkan terminal apa pun.
2. Mengeliminasi *unit production*, yaitu aturan yang langsung mengarah kepada aturan lain tanpa menambah apa-apa (mempunyai aturan perantara). Aturan ini dapat dipersingkat dengan menghilangkan aturan perantara tersebut.
3. Mengeliminasi *epsilon production*, yaitu aturan-aturan yang memiliki nilai *epsilon*
4. *Binarization*, yaitu mengubah aturan yang sangat panjang atau sangat pendek menjadi dalam format Chomsky, hanya menghasilkan dua variabel atau satu terminal.

Jika konversi dilakukan secara manual, langkah-langkah yang lebih dulu dikerjakan adalah mengeliminasi, baru melakukan *binarization*. Namun, karena konversi dilakukan dengan *script* (dilakukan oleh komputer), konversi dimulai dengan langkah *binarization*. Hal ini dilakukan untuk mempermudah iterasi saat pengecekan aturan, apakah ada yang harus dieliminasi atau tidak.

Meskipun dilakukan eliminasi, karena ada *binarization*, jumlah aturan produksi serta jumlah variabel dalam CNF akan lebih banyak dari pada CFG. Sementara itu, dari CFG dan CNF akan tetap sama.

IMPLEMENTASI DAN PENGUJIAN

A. Spesifikasi Teknis Program

Struktur Data

Tiga file utama yang membentuk program kami ialah `grammar_converter.py`, `cyk_parser.py`, dan `lexer.py`. `grammar_converter.py` melakukan konversi CFG menjadi CNF yang hasilnya disimpan di dalam file `CNF.txt`. `cyk_parser.py` melakukan mekanisme pencocokkan bahasa masukan dengan CNF. Keluaran dari `cyk_parser.py` adalah "Accepted" atau "Syntax Error". Yang terakhir adalah `lexer.py` yang digunakan untuk membaca masukan bahasa dan kemudian dijadikan token-token untuk dicocokkan dengan `cyk_parser`. Di dalam file `lexer.py` menggunakan import `cyk_parser` agar dapat sekaligus membaca dan menghasilkan output yang diharapkan dalam satu file.

Program ini menggunakan paradigma pemrograman berorientasi objek.

Fungsi dan Prosedur

Di dalam file `grammar_converter.py`, terdapat tiga fungsi dan prosedur sebagai berikut.

1. Fungsi `read_grammar()` berfungsi untuk membaca grammar yang sudah dibuat di file dan mengonversinya menjadi list rules
2. prosedur `add_rule()` berfungsi untuk menambahkan rule baru ke dalam kamus, prosedur akan mengecek apakah rule tersebut sudah ada di head rule atau belum, jika sudah maka rule akan ditambahkan pada body rule.
3. prosedur `convert_rules()` berfungsi untuk mengonversi rule yang sudah ada

Di dalam file `cyk_parser.py` terdapat dua kelas yang berisi masing-masing fungsi dan prosedur sebagai berikut.

1. Kelas `Node`, berfungsi untuk menyimpan informasi grammar yang terdiri dari dua non terminal, berdasarkan aturan CNF
2. Kelas `parser`, berfungsi untuk melakukan parsing dari CNF dengan bahasa yang dimasukan.

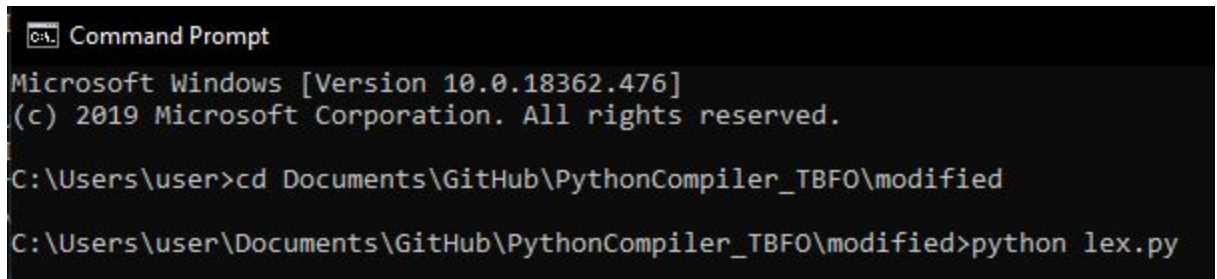
Di dalam file `lexer.py` terdapat tiga kelas dan dua fungsi utama sebagai berikut.

1. Kelas `token`, berfungsi untuk menyimpan informasi token
2. Kelas `lexerError`, berfungsi untuk mengembalikan nilai dimana error terjadi
3. Kelas `Lexer` : berfungsi untuk memproses masukan bahasa menjadi token-token
4. Fungsi `read_files_input`, berfungsi untuk membaca masukan bahasa dari file
5. Fungsi `write_files_output`, berfungsi untuk menulis keluaran bahasa yang telah menjadi token-token untuk kemudian digunakan untuk parsing

Antarmuka

Pada tugas kali ini, tidak ada antarmuka yang 'eye-catchy'. Untuk menjalankan program cukup dengan membuka terminal/command prompt. Pastikan sudah berada di dalam folder yang memuat file program. Kemudian cukup berikan masukan "`python lexer.py`" tanpa tanda kutip. Setelah menekan Enter, maka akan keluar hasil apakah sebuah bahasa diterima atau tidak.

Berikut hasil tangkap layar dari program yang kami buat.



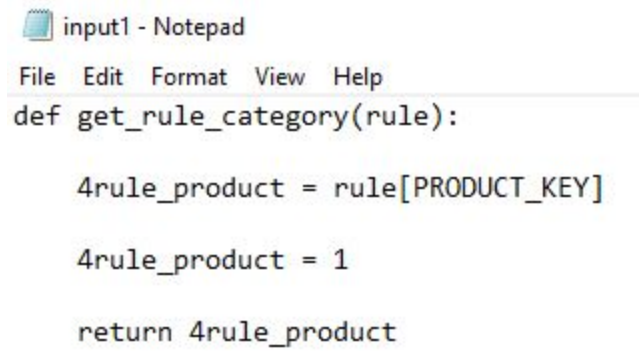
```
Command Prompt
Microsoft Windows [Version 10.0.18362.476]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\user>cd Documents\GitHub\PythonCompiler_TBFO\modified
C:\Users\user\Documents\GitHub\PythonCompiler_TBFO\modified>python lex.py
```

Gambar 3.1 hasil tangkap layar program

B. Capture Berbagai Kasus

1. Input1.txt (nama variable diawali nomor)



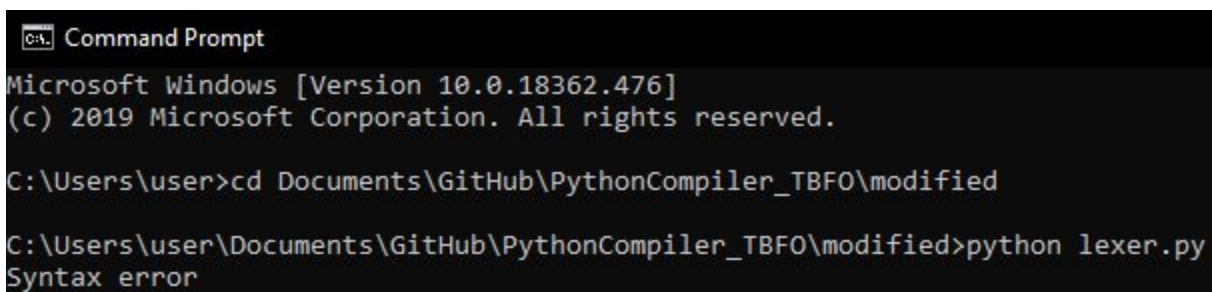
```
input1 - Notepad
File Edit Format View Help
def get_rule_category(rule):

    4rule_product = rule[PRODUCT_KEY]

    4rule_product = 1

    return 4rule_product
```

Gambar 3.2 input1.txt



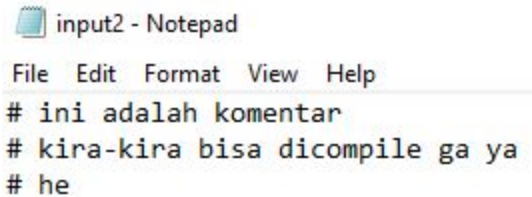
```
Command Prompt
Microsoft Windows [Version 10.0.18362.476]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\user>cd Documents\GitHub\PythonCompiler_TBFO\modified
C:\Users\user\Documents\GitHub\PythonCompiler_TBFO\modified>python lexer.py
Syntax error
```

Gambar 3.3 hasil output input1.txt

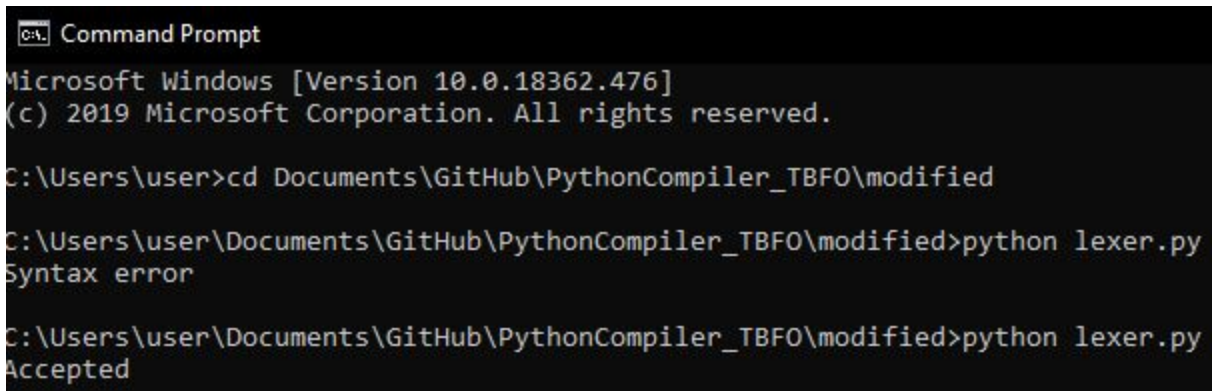
Kode program yang dimasukkan pada input1.txt memiliki syntax error pada line ke-3. Hal ini disebabkan oleh adanya nama variable yang diawali dengan nomor. Nama variable harus diawali oleh huruf, namun setelah huruf pertama dapat berupa nomor dan huruf apapun dengan panjang berapapun.

2. Input2.txt (kode program hanya berisi komentar)



```
input2 - Notepad
File Edit Format View Help
# ini adalah komentar
# kira-kira bisa dicompile ga ya
# he
```

Gambar 3.4 input2.txt



```
Command Prompt
Microsoft Windows [Version 10.0.18362.476]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\user>cd Documents\GitHub\PythonCompiler_TBFO\modified

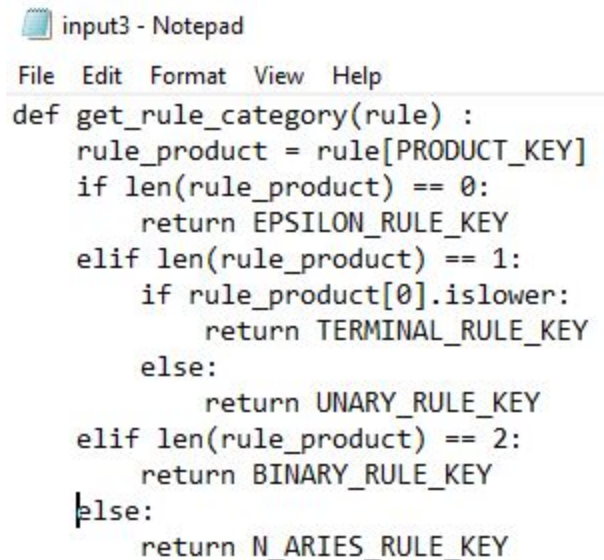
C:\Users\user\Documents\GitHub\PythonCompiler_TBFO\modified>python lexer.py
Syntax error

C:\Users\user\Documents\GitHub\PythonCompiler_TBFO\modified>python lexer.py
Accepted
```

Gambar 3.5 hasil compile input2.txt

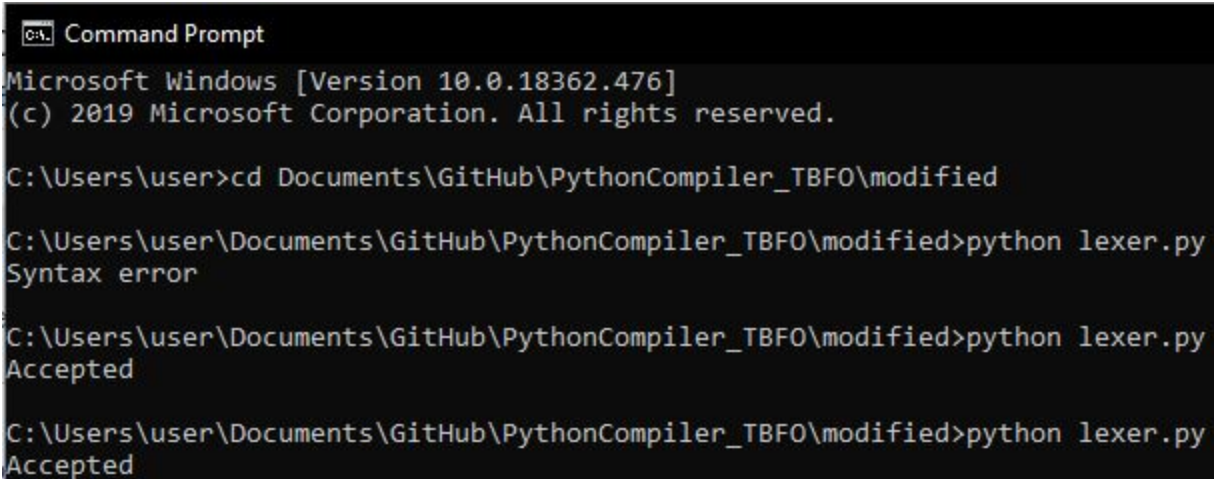
Kode program dalam input2.txt berhasil dicompile, karena hanya berisi komentar single line yang memenuhi aturan bahasa Python.

3. Input3.txt (if-elif-else statement bercabang)



```
input3 - Notepad
File Edit Format View Help
def get_rule_category(rule) :
    rule_product = rule[PRODUCT_KEY]
    if len(rule_product) == 0:
        return EPSILON_RULE_KEY
    elif len(rule_product) == 1:
        if rule_product[0].islower:
            return TERMINAL_RULE_KEY
        else:
            return UNARY_RULE_KEY
    elif len(rule_product) == 2:
        return BINARY_RULE_KEY
    else:
        return N_ARIES_RULE_KEY
```

Gambar 3.6 input3.txt



```
Command Prompt
Microsoft Windows [Version 10.0.18362.476]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\user>cd Documents\GitHub\PythonCompiler_TBFO\modified

C:\Users\user\Documents\GitHub\PythonCompiler_TBFO\modified>python lexer.py
Syntax error

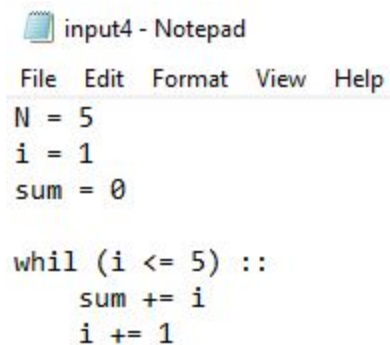
C:\Users\user\Documents\GitHub\PythonCompiler_TBFO\modified>python lexer.py
Accepted

C:\Users\user\Documents\GitHub\PythonCompiler_TBFO\modified>python lexer.py
Accepted
```

Gambar 3.7 hasil compile input3.txt

Kode program dalam input3.txt berhasil dcompile karena memenuhi aturan if-elif-else statement, yaitu elif ada setelah if, elif ada setelah elif yang lain, dan else ada setelah if dan/atau elif.

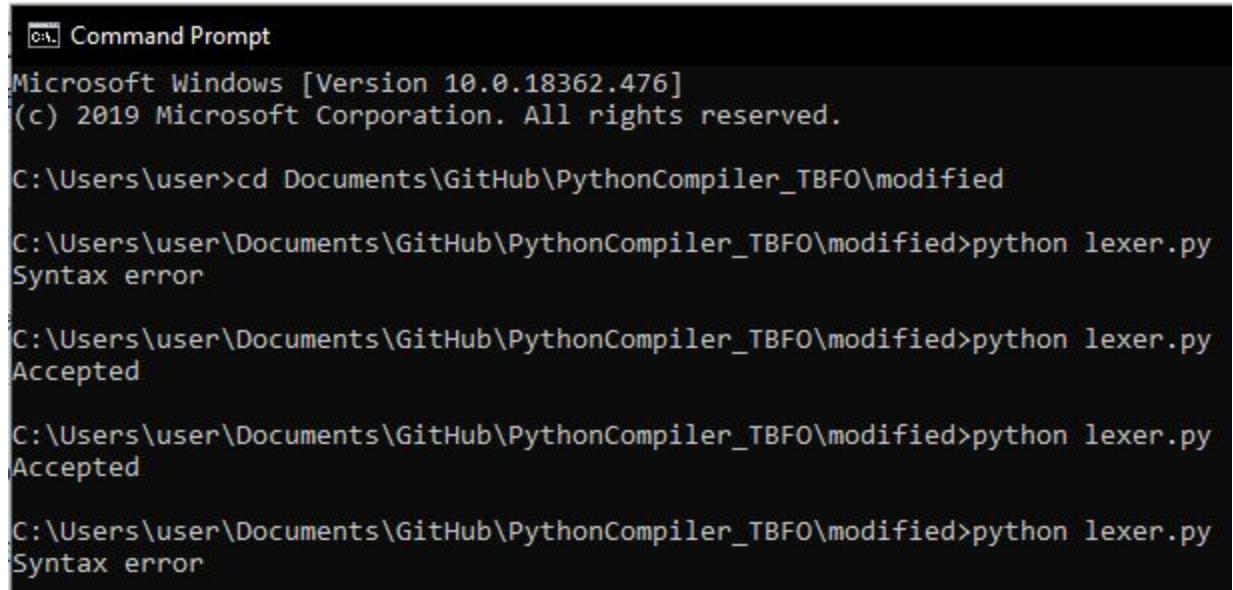
4. Input4.txt (kondisi while statement salah)



```
input4 - Notepad
File Edit Format View Help
N = 5
i = 1
sum = 0

while (i <= 5) ::
    sum += i
    i += 1
```

Gambar 3.8 input4.txt



```
Command Prompt
Microsoft Windows [Version 10.0.18362.476]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\user>cd Documents\GitHub\PythonCompiler_TBFO\modified

C:\Users\user\Documents\GitHub\PythonCompiler_TBFO\modified>python lexer.py
Syntax error

C:\Users\user\Documents\GitHub\PythonCompiler_TBFO\modified>python lexer.py
Accepted

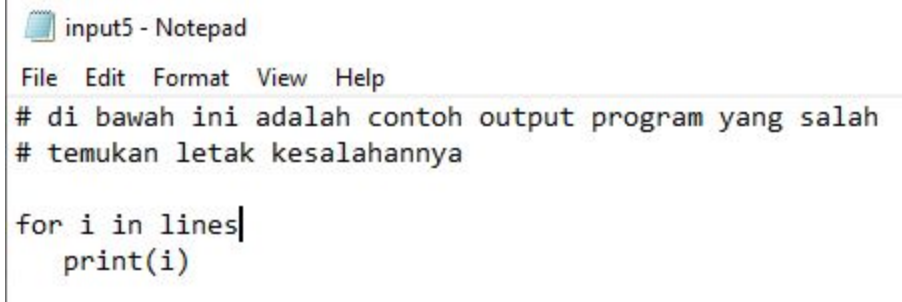
C:\Users\user\Documents\GitHub\PythonCompiler_TBFO\modified>python lexer.py
Accepted

C:\Users\user\Documents\GitHub\PythonCompiler_TBFO\modified>python lexer.py
Syntax error
```

Gambar 3.9 hasil compile input4.txt

Kode program dalam input4.txt mengalami compile error dikarenakan salah kondisi untuk terjadinya while statement. Terjadi salah ketik 'whil' dan tanda titik dua yang ditulis dua kali menyebabkan kode program mengalami compile error

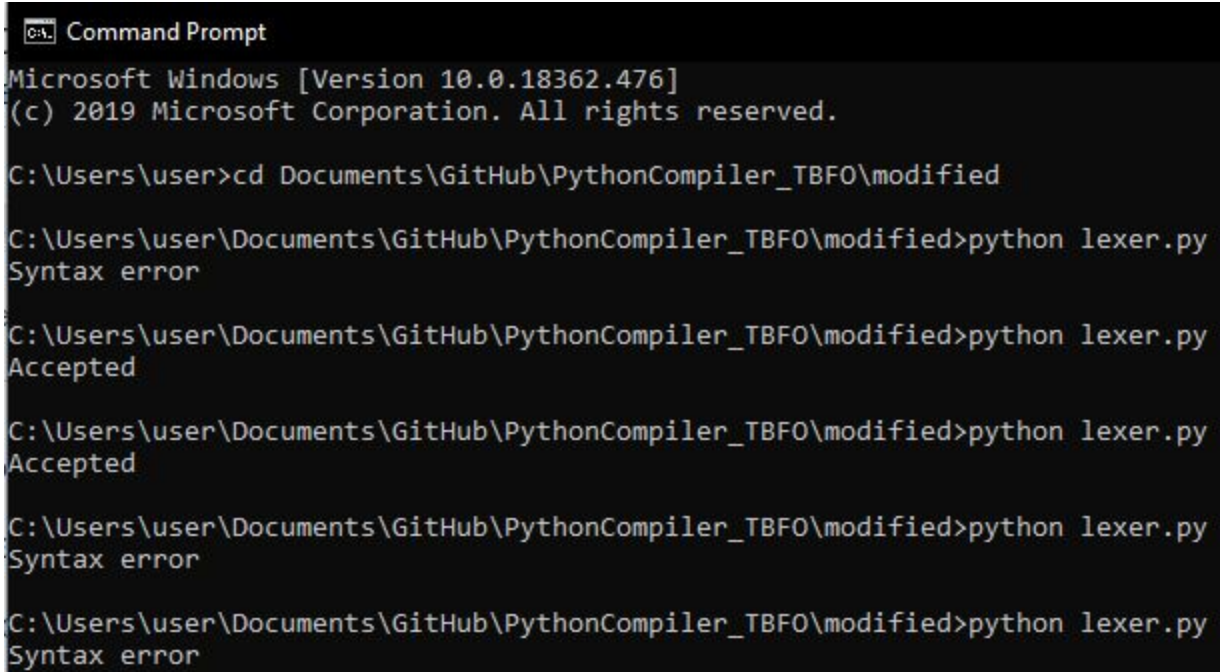
5. Input5.txt (for statement salah kondisi)



```
input5 - Notepad
File Edit Format View Help
# di bawah ini adalah contoh output program yang salah
# temukan letak kesalahannya

for i in lines
    print(i)
```

Gambar 3.10 input5.txt



```
Command Prompt
Microsoft Windows [Version 10.0.18362.476]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\user>cd Documents\GitHub\PythonCompiler_TBF0\modified

C:\Users\user\Documents\GitHub\PythonCompiler_TBF0\modified>python lexer.py
Syntax error

C:\Users\user\Documents\GitHub\PythonCompiler_TBF0\modified>python lexer.py
Accepted

C:\Users\user\Documents\GitHub\PythonCompiler_TBF0\modified>python lexer.py
Accepted

C:\Users\user\Documents\GitHub\PythonCompiler_TBF0\modified>python lexer.py
Syntax error

C:\Users\user\Documents\GitHub\PythonCompiler_TBF0\modified>python lexer.py
Syntax error
```

Gambar 3.11 hasil kompilasi input5.txt

Kode program dalam input5.txt mengalami compile error dikarenakan salah kondisi if statement. Sebelum terjadi if statement, komen pada kode program diterima. Namun saat masuk if statement, terjadi kesalahan dimana pengguna lupa menyertakan tanda titik dua di akhir for statement.

REFERENSI

Lexer: Eli Bendersky @ github (<https://gist.github.com/eliben/5797351>)

Converter & Parser: RobMcH @ github (<https://github.com/RobMcH/CYK-Parser>)

[https://books.google.co.id/books?id=fzUCGtyg0MMC&pg=PA475&dq=python&hl=en&sa=X&ved=0ahUK
EwicxtLdkYTmAhuGzTgGHf-LAqgQ6AEIQDAD#v=onepage&q&f=true](https://books.google.co.id/books?id=fzUCGtyg0MMC&pg=PA475&dq=python&hl=en&sa=X&ved=0ahUKEwicxtLdkYTmAhuGzTgGHf-LAqgQ6AEIQDAD#v=onepage&q&f=true)

<http://web.if.unila.ac.id/ilmukomputer/cnf-chomsky-normal-form/>

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

[https://id.wikipedia.org/wiki/Python_\(bahasa_pemrograman\)](https://id.wikipedia.org/wiki/Python_(bahasa_pemrograman))