LAPORAN TUGAS BESAR
IF2124 Teori Bahasa Formal dan Otomata

Compiler Bahasa Python

Disusun oleh:

Kelompok 20

Nathanael Santoso 13520129

Azmi Alfatih Shalahuddin 13520158

Daffa Romyz Aufa 13520162

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung
Jl. Ganesha 10, Bandung 40132

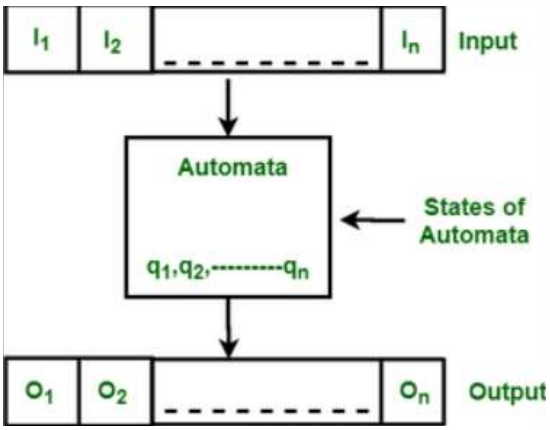
DAFTAR ISI

Teori Dasar

1.1 Finite Automata

Finite Automata yang biasa disingkat sebagai FA adalah mesin paling sederhana untuk mengenali pola. FInite automata (biasa dikenal juga sebagai Finite State Machine) adalah mesin abstrak yang dapat ditulis sebagai sebuah tuple dengan 5 elemen. Finite Automata memiliki sepasang states dan aturan untuk berpindah dari satu state ke state lainnya yang bergantung juga kepada simbol masukan.

Finite automata dapat dikatakan sebagai model abstrak dari komputer digital. Gambar berikut ini menunjukkan bagian bagian penting dari Finite automata



Sumber : geeksforgeeks.com

Bagian-bagian penting dari Finite automata merupakan 5 hal berikut :

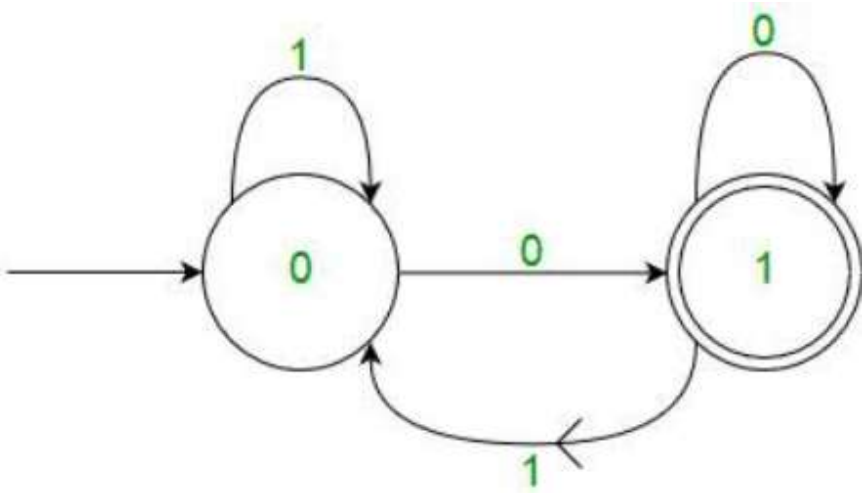
1. Input
2. Output
3. States
4. State Relation
5. Output Relation

atau , elemen tuple yang biasa digunakan adalah sebagai berikut

Q : Finite set of states.
Σ : set of Input Symbols.
q : Initial state.
F : set of Final States.
δ : Transition Function.

FInite automata dapat dikategorikan menjadi dua tipe, yakni DFA (Deterministic Finite Automata) dan juga NFA (Non-deterministic Finite Automata). Dalam sebuah deterministic finite automata, untuk setiap input yang tersedia, subuah mesin berpindah tepat ke sebuah state. Fungsi transisi didefinisikan untuk setiap state untuk masing-masing input. Selain itu, dalam DFA tidak terdapat transisi Null atau Epsilon. Akibatnya, DFA tidak dapat berpindah state jika tidak menerima input apapun.

Sebagai contoh, DFA di bawah dengan Input = {0,1} menerima semua string yang berakhiran 0.



Sumber : geeksforgeeks.com

Yang perlu diingat ialah bisa terdapat banyak kemungkinan DFA untuk sebuah pattern yang sama. Namun, biasanya untuk mendapatkan efisiensi tertinggi, DFA dengan jumlah state paling sedikitlah yang akan dipilih.

NFA merupakan Finite Automata yang mirip dengan DFA, dengan beberapa aturan tambahan sebagai berikut :

1. Null atau epsilon diperbolehkan, sehingga dalam sebuah NFA kita dapat berpindah dari sebuah state tanpa menerima input apapun.
2. Dapat berpindah ke state berjumlah berapapun (tidak harus 1) untuk setiap input yang diterima.

Akan tetapi, sebenarnya, NFA dan DFA adalah ekuivalen. Satu hal yang penting ialah dalam NFA, semua path atau himpunan deretan input yang diterima yang mengarah state menyebabkan input tersebut diterima.

Secara matematis, setiap DFA merupakan NFA, tetapi tidak berlaku sebaliknya. Akan tetapi, karena kita dapat mengkonversi NFA menjadi DFA, maka sudah pasti tedapat bentuk DFA yang ekuivalen terhadap setiap NFA. NFA merupakan Finite Automata yang lebih menggunakan konsep teoritis dan DFA biasanya digunakan untuk Analisis Leksikal dalam kompiler. Akan tetapi, untuk semua DFA dan NFA dapat terdapat lebih dari satu final state

1.2 Context-Free Grammar

Context Free Grammar atau CFG adalah grammar atau aturan formal yang biasanya digunakan untuk meng-generate semua kemungkinan string dari sebuah bahasa formal.

Context free grammar dapat dituliskan sebagai tuple dengan 4 buah elemen. Misal terdapat Context Free Grammar G, maka G dapat dituliskan sebagai G = (V, T, P, S)
Dimana
G merupakan Grammar,
T merupakan himpunan terbatas dari simbol terminal,
V merupakan himpunan terbatas dari simbol non terminal,
P merupakan himpunan aturan produksi, sedangkan
S merupakan simbol start.

Dalam CFG, start simbol digunakan untuk mendapatkan string. Kita dapat mendapatkan string dengan berulang-ulang menggantikan non terminal di sisi kanan produksi sampai semua non terminal tergantikan dengan terminal.

Misalnya, L= {wcw^R | w € (a, b)*}.
Terdapat aturan produksi sebagai berikut :

1. S → aSa
2. S → bSb
3. S → c

Periksa apakah abbcbba dapat didapatkan dari CFG tersebut.

CFG dapat berguna untuk mendeskripsikan banyak bahasa pemrograman. Jika sebuah gramar didesain dengan baik maka efficient parser dapat terbentuk dengan otomatis. Dengan menggunakan informasi dengan asosiatif dan precedence, grammar yang sesuai dengan ekspresi dapat terbentuk.

CFG dapat mendeskripsikan struktur bersarang seperti tanda kurung yang simbang, melakukan pengecekan begin-end, if-then-else, dan lain sebagainya.

1.3 Kata Kunci dalam Python

1.3.1 False, True

True dan False merupakan nilai kebenaran yang ada dalam Python. Mereka adalah hasil dari operasi perbandingan atau operasi logikal (boolean) dalam python. Sebagai contoh :

1 == 1 bernilai True

True or False bernilai True

3 > 7 bernilai False

True + True bernilai 2 (karena True = 1 , False = 0)

1.3.2 None

None adalah konstanta spesial dalam Python yang merepresentasikan Kekosongan Nilai atau null. None merupakan objek dari tipe data nya sendiri, yakni NoneType. Kita tidak dapat membuat banyak objek None, tetapi kita dapat melakukan assign nilai none terhadap suatu variabel. Variabel tersebut akan bernilai sama, satu sama lain.

Nilai none bukan bernilai False, 0, atau list, dictionary, ataupun string kosong. Sebuah fungsi yang tidak mengembalikan apapun secara otomatis akan mengembalikan nilai None. None juga dikembalikan oleh fungsi yang jika dijalankan dengan sebuah parameter tidak mengembalikan apapun.

Sebagai contoh

```
def improper_return_function(a):
    if (a % 2) == 0:
        return True

x = improper_return_function(3)
print(x)
```

Maka output dari terminal akan berupa None

1.3.3 and, or, not

and, or, not merupakan operator logika dalam python. and akan menghasilkan True jika kedua buah operan bernilai benar. Sedangkan or akan menghasilkan true apabila paling tidak terdapat satu buah operan yang bernilai True. sedangkan not merupakan operator untuk membalikkan nilai kebenaran. not True menghasilkan False, begitu juga sebaliknya.

Berikut merupakan tabel kebenaran and, or, dan not.

A	B	A and B
True	True	True
True	False	False
False	True	False
False	False	False
A	B	A or B
True	True	True
True	False	True
False	True	True
False	False	False
A		not A
True		False
False		True

or_test ::= and_test | or_test "or" and_test
and_test ::= not_test | and_test "and" not_test
not_test ::= comparison | "not" not_test

1.3.4 as

as digunakan untuk membuat sebuah alias ketika mengimport suatu modul. Artinya, as dapat memberikan nama lain sesuai keinginan user kepada sebuah modul ketika mengimportnya.

Sebagai contoh, kita ingin menghitung cosinus pi dengan menggunakan modul math yang disediakan oleh Python. Kita dapat menggunakan syntax as sebagai berikut.

```
>>> import math as myAlias
```

```
>>>myAlias.cos(myAlias.pi)
-1.0

import_stmt     ::=  "import" module ["as" name] ( "," module ["as" name] )*
                   | "from" relative_module "import" identifier ["as" name]
                   ( "," identifier ["as" name] )*
                   | "from" relative_module "import" "(" identifier ["as" name]
                   ( "," identifier ["as" name] )* [","] ")"
                   | "from" module "import" "*"
module          ::=  (identifier ".")* identifier
relative_module ::=  "."* module | "."+
name            ::=  identifier
```

1.3.5 break, continue

Break dan continue digunakan dalam pengulangan for dan while untuk mengubah cara kerjanya. Break akan mengakhiri pengulangan terkecil ketika dieksekusi dan eksekusi dilanjutkan tepat dibawah pengulangan. Sedangkan continue hanya melewati iterasi saat ini dan mengeksekusi iterasi selanjutnya. Continue tidak mengakhiri pengulangan.

1.3.6 class

class dapat digunakan untuk mendefinisikan class baru buatan pengguna. Class merupakan kumpulan dari atribut dan method yang berhubungan yang dapat merepresentasikan situasi tertentu. Ide untuk mengumpulkan data dan fungsi bersamaan menjadi sebuah kelas merupakan inti dari konsep Pemrograman Berbasis Objek (Object oriented programming).

```
classdef    ::=  [decorators] "class" classname [inheritance] ":" suite
inheritance ::=  "(" [parameter_list] ")"
classname   ::=  identifier
```

1.3.7 def

Def digunakan untuk mendefinisikan fungsi baru buatan pengguna. Fungsi merupakan kumpulan statemen yang berhubungan, yang secara bersamaan dapat mengerjakan tugas tertentu. Fungsi dapat membantu dalam mengorganisasi kode menjadi lebih mudah di manajemen dan juga untuk melakukan tugas tugas yang berulang. Contoh dari penggunaan def adalah sebagai berikut.

```
def function_name(parameters):
    …
```

```
funcdef        ::=  [decorators] "def" funcname "(" [parameter_list] ")" ["->" expression] ":" suite
decorators     ::=  decorator+
decorator      ::=  "@" dotted_name ["(" [parameter_list [","]] ")"] NEWLINE
dotted_name    ::=  identifier ("." identifier)*
parameter_list ::=  (defparameter ",")*
                  | "*" [parameter] ("," defparameter)* ["," "**" parameter]
                  | "**" parameter
                  | defparameter [","] )
parameter      ::=  identifier [":" expression]
defparameter   ::=  parameter ["=" expression]
funcname       ::=  identifier
```

1.3.8 if, elif, else

If, elif, else digunakan untuk percabangan berkondisi. Ketika ingin mengeksekusi suatu blok kode ketika kondisi yang diinginkan benar, maka gunakan if dan elif. elif adalah singkatan dari else if. Ketika kondisi yang diinginkan salah maka kode blok pada else yang dieksekusi.

Memiliki syntax

```
if_stmt ::=  "if" expression ":" suite
             ( "elif" expression ":" suite )*
             ["else" ":" suite]
```

Selain itu juga memiliki bentuk list comprehension:
INL_if ::= [expression if comparison (else expression)]
Dengan expression dapat berupa INL_if juga sehingga dapat melakukan rekursi yang panjang

1.3.9 raise

Raise digunakan dalam mengelola eksepsi. Eksepsi merupakan eror dasar yang menunjukkan sesuatu berjalan dengan tidak benar ketika mengeksekusi program kita. Kita dapat memunculkan eksepsi secara eksplisit dengan menggunakan raise. Sebagai contoh,

```
if num == 0:
    raise ZeroDivisionError('cannot divide')
```

```
raise_stmt ::=  "raise" [expression ["from" expression]]
```

1.3.10 for

for merupakan keyword yang digunakan untuk proses looping. Biasanya, kita menggunakan for ketika kita mengetahui berapa kali kita akan melakukan looping. Dalam bahasa Python, kita

dapat menggunakan for dengan berbagai macam tipe data dengan beberapa elemen, seperti list atau bahkan string. Sebagai contoh penggunaan for untuk melakukan traversal adalah sebagai berikut.

```
names = ['John','Monica','Steven','Robin']
for i in names:
   print('Hello '+i)
```

Maka outputnya akan sebagai berikut

```
Hello John
Hello Monica
Hello Steven
Hello Robin
```

```
for_stmt ::=  "for" target_list "in" expression_list ":" suite
              ["else" ":" suite]
```

Selain itu juga memiliki bentuk dalam list comprehension:
INL_for ::= [expression “for” var “in” expression]
Dengan expression pertama dapat mengandung INL_for juga sehingga merekursi untuk setiap dimensi array yang dibentuk.
INL_for juga dapat digabung dengan INL_if sebagai berikut:
INL_for + INL_if::= [expression “for” var “in” expression “if” comparison (else expression)]

1.3.11 from, import

Keyword import digunakan untuk mengimport modul kedalam sebuah kode. Sedangkan keyword from…import digunakan untuk mengimport atribut spesifik atau fungsi ke dalam sebuah kode. Sebagai contoh,
import math
Kode tersebut akan mengimport module math dan kita dapat menggunakan fungsi cos() dengan menggunakan math.cos(). Namun, jika kita hanya ingin mengimport fungsi cos(), dapat digunakan from seperti
From math import cos
Sehingga kita cukup menulis cos(), bukan math.cos().

```
import_stmt     ::=  "import" module ["as" name] ( "," module ["as" name] )*
                   | "from" relative_module "import" identifier ["as" name]
                   ( "," identifier ["as" name] )*
                   | "from" relative_module "import" "(" identifier ["as" name]
                   ( "," identifier ["as" name] )* [","] ")"
                   | "from" module "import" "*"
module          ::=  (identifier ".")* identifier
```

```
relative_module ::=  "."* module | "."+
name            ::=  identifier
```

1.3.12 in

In dapat digunakan untuk menguji apakah sebuah sekuens (list, tuple, string, dll.) memiliki elemen tertentu. Kode tersebut akan mengembalikan True apabila nilai tersebut ada dan False jika tidak. Selain itu, in juga dapat digunakan untuk traversal dengan menggunakan for loop. Sebagai contoh,

```
for i in 'hello':
    print(i)
```

Akan menghasilkan

```
h
e
l
l
o
```

```
comparison    ::=  or_expr ( comp_operator or_expr )*
comp_operator ::=  "<" | ">" | "==" | ">=" | "<=" | "<>" | "!="
                 | "is" ["not"] | ["not"] "in"
```

1.3.13 is

Is digunakan untuk mengetes identitas dari suatu objek. Berbeda dengan == yang mengecek apakah kedua variable equal atau tidak, is digunakan untuk mengetes apakah dua variabel mengacu pada objek yang sama. Kode akan bernilai True apabila objek identik dan False jika tidak. List dan dictionary yang sama-sama kosong belum tentu identik karena mengacu pada memori yang berbeda, sedangkan string dan tuple bersifat tetap sehingga dua buah string atau tuple yang equal akan identik juga. Mereka mengacu pada lokasi memori yang sama.


```
comparison    ::=  or_expr ( comp_operator or_expr )*
comp_operator ::=  "<" | ">" | "==" | ">=" | "<=" | "<>" | "!="
                 | "is" ["not"] | ["not"] "in"
```

1.3.14 pass

pass adalah operasi null. ketika dieksekusi, tidak ada yang terjadi. Ini berguna sebagai pengganti ketika pernyataan diperlukan secara sintaks, tetapi tidak ada kode yang perlu dieksekusi untuk menghindari error. Kita dapat menggunakan pass pada fungsi dan kelas. Sebagai contoh,

```
def function(args):
    pass

pass_stmt ::=  "pass"
```

1.3.15 return

return hanya dapat terjadi secara sintaks bersarang dalam definisi fungsi, bukan dalam definisi kelas bersarang. return mengakhiri fungsi saat ini dengan ekspresi (atau None) sebagai nilai return. Jika kita tidak menentukan return suatu fungsi secara eksplisit, maka fungsi akan mengembalikan None. Sebagai contoh,

```
def func_return():
    a = 10
    return a

def no_return():
    a = 10

print(func_return())
print(no_return())
```

Outputnya adalah sebagai berikut
10
None

```
return_stmt ::=  "return" [expression_list]
```

1.3.16 while

while digunakan untuk pengulangan selama ekspresi benar. While loop akan berhenti jika loop mengevaluasi sebuah nilai False atau terdapat statement break. Contoh dari while ialah sebagai berikut.

```
i = 5
while(i):
    print(i)
```

```
    i = i – 1
```

Maka outputnya akan sebagai berikut.

```
5
4
3
2
1
```

```
while_stmt ::=  "while" expression ":" suite
                ["else" ":" suite]
```

1.3.17 with

With digunakan untuk mengumpulkan eksekusi dari kumpulan kode dengan method yang didefinisikan dalam context manager. Context manager adalah kelas yang mengimplementasi method __enter dan __exit__. Dengan menggunakan with, maka dipastikan __exit__ akan terpanggil di akhir block kode. Sebagai contoh,

```
with open('example.txt', 'w') as my_file:
    my_file.write('Hello world!')
```

Contoh ini menulis text “Hello World!” ke dalam file example.txt. Karena __enter__ dan __exit__ sudah terdefinisi dalam kode tersebut, maka kode blok tersebut dapat berfungsi selayaknya context manager.

Pertama-tama, method __enter__ dipanggil, lalu kode didalam with dieksekusi dan terakhir, __exit__ dipanggil. __exit__ akan dipanggil bahkan ketika terdapat error.

```
with_stmt ::=  "with" with_item ("," with_item)* ":" suite
with_item ::=  expression ["as" target]
```

Hasil FA dan CFG

2.1 Finite Automata

Adapun Finite Automata untuk melakukan pengecekan terhadap kebenaran penamaan variabel. Finite Automata yang digunakan adalah DFA yang mempunyai tiga state dan language semua karakter ASCII. State DFA ditetapkan dengan variabel global sehingga hanya perlu memanggil fungsi untuk menguji karakter masukan dan mengembalikan keluaran boolean accepted atau tidak.

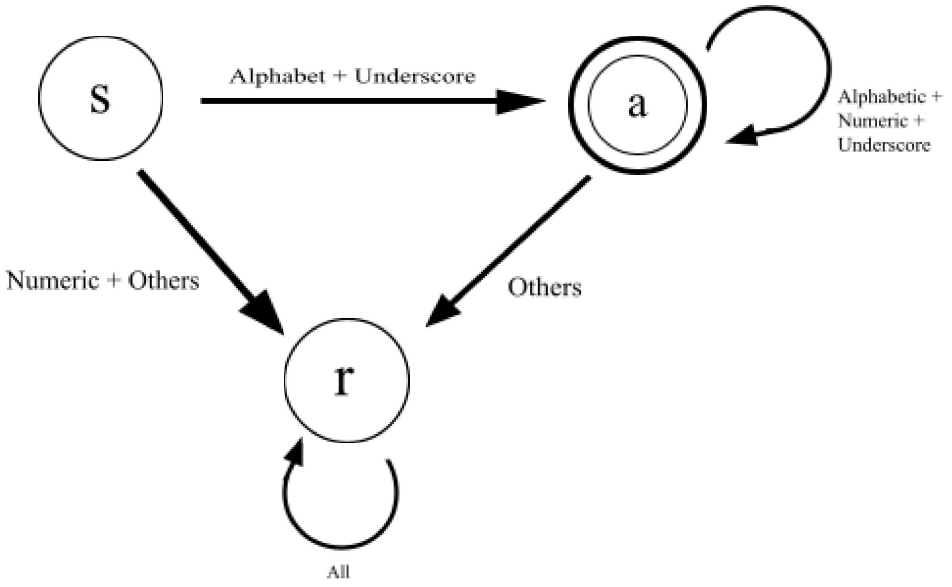
Secara formal, DFA mempunyai Definisi:

$$FA = \{\{s, a, r\}, \{x \mid x \in ASCII\}, \delta, s, \{a\}\}$$

Dengan tabel transisi sebagai berikut:

Σ:	Alphabetic (a-z, A-Z)	Numeric (0-9)	Underscore(_)	Others
s	a	r	a	r
*a	a	a	a	r
r	r	r	r	r

Dari definisi tersebut, DFA dapat dimodelkan dalam bentuk graf berarah sebagai berikut:



Dari Graf di atas dibayangkan bahwa DFA Merupakan Automata yang menerima input ASCII, pada state s akan menuju state a jika input alphabetic atau underscore dan menuju r jika tidak. Accepting state a menerima input alphabetic, numeric, atau underscore dan menuju r jika tidak. State r merupakan dead state dimana semua input akan menghasilkan state yang sama.

2.2 Context-Free Grammar

2.2.1 CFG Single Line

S -> DEF_F | IMP_F | CLS_F | SUITE1 | SUITE2 | WTH_F | Break | Return ANNY
SUITE1 -> ASSNMENT | METHOD | Break
SUITE2 -> If_func | FO_F | WH_F | ELF_F | ELS_F
ASSNMENT -> Var Assign EXPR | Var Assign ANNY
EXPR -> ANNY | Var OPR Singles | Var Per Singles | ARRG | Var Dot METHOD | METHOD | Var Comp Singles | POpen EXPR Pc | Bool | not_func Singles | IN_F | is_Func
not_func -> not_real | not_func not_real
ARRG -> ANNY Comma ARRG | ANNY
ARG_DICT -> Any_Dict Comma ARG_DICT | Any_Dict
Any_Dict -> Singles End ANNY
ANNY -> Singles | Structs | METHOD
Singles -> Var | Str | Num
Structs -> List | Tuple | DCT
METHOD -> Var POpen EXPR Pc | Var POpen EXPR Pc Dot METHOD
If_func -> If_real EXPR End | If_func SUITE1
INL_IF -> ANNY If_real EXPR | ANNY If_real EXPR ELS_F | INL_IF For EXPR
INL_FOR -> ANNY For EXPR | INL_FOR If_real EXPR | INL_FOR If_real EXPR ELS_F
ELF_F -> Elif EXPR End | ELF_F SUITE1
ELS_F -> Else End | ELS_F SUITE1
WH_F -> While EXPR End | WH_F SUITE1
FO_F -> For EXPR End | FO_F SUITE1 | EXPR For EXPR End
DEF_F -> DEF_R Var POpen EXPR Pc End | DEF_R Var POpen EXPR Pc End Return EXPR
IMP_F -> Import Var | Import Var AS_R Var | From Var Import Var | From Var Import Var AS_R Var
CLS_F -> Class Var End
CONTROL -> Break | Continue
WTH_F -> With EXPR AS_R EXPR End
IN_F -> Singles IN_R ANNY | For Singles IN_R ANNY
is_Func -> Var is_real EXPR | POpen is_Func Pc
List -> zqO ARRG zqC | zqO zqC | zqO INL_FOR zqC | zqO INL_IF zqC
Tuple -> POpen ARRG Pc | POpen Pc | POpen INL_FOR Pc | POpen INL_IF Pc
DCT -> COpen ARG_DICT CClose | COpen CClose
Comma -> ","
Import -> "import"
Return -> "return"
DEF_R -> "def"
While -> "while"
For -> "for"
Break -> "break"
TRUE -> "True"
FALSE -> "False"
Assign -> "="
Eq -> Assign Assign
Comp -> Eq | ">" | "<" | Legr Assign | Exclam Assign
Legr -> ">" | "<"
Exclam -> “!”

```
OPR -> "+" | "-" | Mul | "/" | "//" | "%" | Pow
Mul -> "*"
Pow -> Mul Mul
POpen -> "("
Pc -> ")"
zqC -> "]"
zqO -> "["
COpen -> "{"
CClose -> "}"
Elif -> "elif"
Else -> "else"
If_real -> "if"
End -> ":"
Continue -> "continue"
Class -> "class"
AS_R -> "as"
From -> "from"
Pass -> "pass"
Raise -> "raise"
With -> "with"
IN_R -> "in"
not_real -> "not"
is_real -> "is"
EXPS -> Var | Var Ops | Var Comp
Ops -> Per EXPS | Per Bool
Per -> "and" | "or"
Bool -> "True" | "False" | "None"
```

Implementasi dan Pengujian

3.1 Spesifikasi Teknis Program

Program menggunakan algoritma CYK untuk memeriksa kebenaran sintaks. Algoritma tersebut pertama-tama akan memeriksa tiap line secara persatu-satu menentukan apakah sintaks pada line tersebut benar sesuai dengan CNF yang telah dibuat. Kemudian algoritma memeriksa kata pertama pada tiap line untuk memeriksa keyword yang dipakai secara multiline sesuai.

1. `def isLetter(unsigned):`
   Returns True if unsigned is in alphabet a-z, A-Z.
2. `def isNumber(unsigned):`
   Returns True if unsigned is in number 0-9.
3. `def DFA(char):`
   Updates Global Variable state depending on char and returns True if it is an accepting state.
4. `def checkVar(string):`
   Resets Global Variable state to start and feeds a string into the DFA and returns a boolean depending on the end state result.
5. `def getCombinations(verticals, diagonals):`
   Gets all possible grammar combinations from vertical and diagonal directions in a position in the CYK table.
6. `def getCompSets(offset, position):`
   Gets all vertical and horizontal elements of the CYK table for a position in the CYK table.
7. `def makeCYKTable(line):`
   Function makes the CYK table using Functions 5 and 6 for all positions in the table.
8. `def outputTable(Table, line):`
   Calls function 7 and compares the last element of the table with the startstate, if the return value is true then the sentence is in the grammar.
9. `def checkValid(line, readTable=False):`
   Checks the validity of a line in the grammar, takes extra input readTable for debugging purposes
10. `def readFile(file):`
    Reads file from folder based on input or argos.
11. `def checkFile(file, readTable=False):`
    Checks validity of each line of the file based on the CNF grammar that is implemented in the module and checks for out of primitive errors (Return before defs, etc.).
12. `def getIdxLine(lines, list, minline):`
    Gets original line of code that returns an error to display it on standard output.

3.2 Pengujian Kasus

```
C:\Github\TBFO_Tubes\src>py main.py tesfail1.py tesfail2.py tesfail3.py tesfinal.py test1.py cek_if_else.py
File 1 had a Syntax error.
(Error Located at line: 2, in 'print("Hello World!!!"')
File 2 had a Naming error.
(Error Located at line: 4, in '123var = 100')
File 3 had a Outside Primitive error.
(Error Located at line: 2, in 'a = 10')
File 4 had a Outside Primitive error.
(Error Located at line: 9, in 'a=0')
File 5 successfully compiled.
File 6 successfully compiled.
```

3.2.1 File 1 : Syntax Error

tesfail1.py

```
1    print("I love you 3000")
2    print("Hello World!!!"
```

```
  File "c:\Users\azmia\tbfo tubes\tesfail1.py", line 2
    print("Hello World!!!"
          ^
SyntaxError: '(' was never closed
```

3.2.2 File 2 : Naming Error

tesfail2.py > ...

```
1    x = 10
2    y = 20
3    nama = "azmi"
4    123var = 100
```

```
PS C:\Users\azmia\tbfo tubes> & C:/Users/azmia/AppData/Local/Programs/Python/Python310/python.exe
 "c:/Users/azmia/tbfo tubes/tesfail2.py"
  File "c:\Users\azmia\tbfo tubes\tesfail2.py", line 4
    123var = 100
      ^
SyntaxError: invalid decimal literal
```

3.2.3 File 3 : Outside primitive Error

tesfail3.py > ...

```
1    print("Hello world! ")
2    a = 10
3    b = 15
4    break
```

```
PS C:\Users\azmia\tbfo tubes> & C:/Users/azmia/AppData/Local/Programs/Python/Python310/python.exe
 "c:/Users/azmia/tbfo tubes/tesfail3.py"
  File "c:\Users\azmia\tbfo tubes\tesfail3.py", line 4
    break
    ^^^^^
SyntaxError: 'break' outside loop
```

3.2.4 FIle 5 : Compile Success (dengan berbagai keyword)

```python
import numbers as n

nama = "nathan"
umur_5_tahun_lalu = 23
print ("nama saya")
print(nama)
print("sedangkan umur saat ini adalah")
print(umur_5_tahun_lalu + 5)
a=0



tuplea = ("azmi",19,"South Tangerang",True,[1,2,3,4,5,6,7,8,9,10])

for i in tuplea[4]:
    print("yes",end='')
    if i == 7:
        break
print()
def greet(name):
    """
    This function greets to
    the person passed in as
    a parameter
    """
    print("Hello, " + name + ". Good morning!")
    return("???")

greet('Asprak')

print(greet('Asprak'))

print()
class Person:
    def __init__(mysillyobject, name, age):
        mysillyobject.name = name
        mysillyobject.age = age


    def myfunc(abc):
        print("Hello my name is " + abc.name)

p1 = Person("Azmi", 19)
p1.myfunc()
```

```python
for i in ("Azmi Ganteng"):
    if i == 'z':
        continue
    print(i, end='')


print()
print()
benar = True
benar = not benar

if (benar):
    print("(check not) Benar")
else:
    print("(check not) Salah")



if (True or False == True):
    print("(check or) benerr")
else:
    print("(check or) salahhh")


def reciprocal(num):
    try:
        r = 1/num
    except:
        print('Exception caught')
        return
    return r


print(reciprocal(10))
print(reciprocal(0))



num = 10
x = num
greg = "name"
```

```python
lista = ["dog"]
listb = [1, "lol", greg]
listc = [[x for x in lista] for y in lista]

tupa = ()
tupb = (2, greg, "kek")

dicta = {}
dictb = {"a": 10, greg: "b", 10: num}

while True:
    break

""" Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Cras ut urna erat. Nam mollis justo turpis, sed egestas nisi gravida
tristique. Nam in tristique lacus. Suspendisse sed metus at lacus
ullamcorper gravida. Nullam dictum quam a volutpat lacinia. Phasellus
volutpat cursus lectus, in suscipit leo efficitur nec. Quisque
vitae commodo lorem. Donec suscipit tellus nec eleifend convallis.
Sed leo metus, convallis eget tortor sed, sollicitudin malesuada
ante. Sed facilisis felis id tortor bibendum, et venenatis massa
fermentum. Maecenas magna ligula, tempor eu ligula eu, dignissim
sagittis purus. Pellentesque a dignissim enim. Sed vestibulum feugiat
dui vitae placerat. Cras sapien arcu, tempor ac pulvinar quis,
mattis eget nulla. Fusce non vehicula eros. In hac habitasse platea
dictumst.
"""


x = -1

if x < 0:
    raise Exception("Sorry, no numbers below zero")
```

Link Github

Berikut tertera link untuk repository dari Kelompok 20:

https://github.com/nart4hire/TBFO_Tubes

Pembagian Tugas

Nathanael Santoso

1. Implementasi Algoritma CYK
2. Implementasi Finite Automata Variable Name Checking
3. Penyusunan Laporan

Daffa Romyz Aufa

1. CFG Keyword If
2. CFG Keyword Elif
3. CFG Keyword else
4. CFG Keyword While
5. CFG Keyword For
6. CFG Keyword break
7. CFG Keyword Def
8. CFG Keyword return
9. CFG Keyword Import
10. CFG Keyword False
11. CFG Keyword True
12. CFG Keyword None
13. Penyusunan Laporan

Azmi (Grammar dan CNF)

1. CFG Keyword Class
2. CFG Keyword continue
3. CFG Keyword as
4. CFG Keyword From
5. CFG Keyword Pass
6. CFG Keyword Raise
7. CFG Keyword With
8. CFG Keyword In
9. CFG Keyword Not
10. CFG Keyword And
11. CFG Keyword Or
12. CFG Keyword Is
13. Penyusunan Laporan

Referensi

[1] *Introduction of finite automata*. GeeksforGeeks. (2021, October 6). Retrieved November 24, 2021, from https://www.geeksforgeeks.org/introduction-of-finite-automata/.

[2] *Context free grammars - javatpoint*. www.javatpoint.com. (n.d.). Retrieved November 24, 2021, from https://www.javatpoint.com/context-free-grammar.