LAPORAN TUGAS PEMROGRAMAN
IF2124 – Teori Bahasa Formal dan Automata

Parser Bahasa Python menggunakan Algoritma Cocke-Younger-Kasami

13520124 – Owen Christian Wijaya

13520152 – Muhammad Fahmi Irfan

13520157 – Thirafi Najwan Kurniatama

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

DAFTAR ISI

BAB 1 TEORI DASAR

1.1 Finite Automata

Sebuah *finite automata* (FA), biasanya disebut juga *finite-state machine* (FSM), adalah sebuah model matematis dari komputasi yang biasanya digunakan untuk mengenali sebuah pola *input*. Sebuah FA dapat diibaratkan sekumpulan *states* atau keadaan yang menerima sebuah *input* dan berpindah *state* dari satu *state* ke *state* lainnya apabila FA menerima *input* tertentu. Perpindahan ini disebut dengan transisi.

Sebuah FA biasanya diibaratkan dengan sebuah *tuple* yang berisi nilai *input*, *output*, perubahan otomata, hubungan antar *state* (fungsi transisi), dan *state* akhir. Ada dua jenis dari FA:

- *Deterministic finite automata* (DFA), yaitu sebuah FA di mana setiap *input* mempunyai hanya satu *state* yang dituju. Setiap *state* menerima sebuah input dan meneruskannya ke tepat satu *state* selanjutnya, dan tidak ada *input* yang diabaikan.
- *Non-deterministic finite automata* (NFA), yaitu sebuah FA dimana setiap *input* bisa mempunyai lebih dari satu *satate* yang dituju. Setiap input dapat diteruskan ke lebih dari satu *state*, dan *input* bisa saja diabaikan.

Finite automata biasanya digunakan untuk memvalidasi sebuah *input* dengan menganalisis konteks dari *input* dan memberikan *Boolean output* mengenai apakah *input* tersebut diterima atau tidak. Dalam tugas pemrograman ini, konsep FA digunakan untuk memvalidasi nama variabel dalam bahasa Python. Penamaan variabel dalam bahasa Python mempunyai aturan-aturan nama yang harus diimplementasikan di tugas ini. Awalan dari variabel tidak boleh berupa angka, dan karakter dalam variabel tidak boleh mengandung karakter '-'. Untuk melakukan validitas ini, digunakan FA dalam bentuk *class* dengan pendefinisian fungsi-fungsi sebagai perwakilan *states* dalam FA. Apabila variabel tidak memenuhi kriteria, maka variabel akan dimasukkan ke dalam *dead state*. Sebaliknya, FA akan mengembalikan nilai True dan nama variabel valid.

1.2 Context-Free Language dan Grammar

Context-free language merupakan sebuah himpunan string yang memenuhi *context-free grammar* (CFG). CFG sendiri merupakan sebuah *grammar* yang menggunakan aturan rekursif yang disebut *production.* Sebuah CFG memiliki suatu himpunan variabel, himpunan terminal, variabel awal, dan *production*. Salah satu algoritma untuk menguji apakah suatu CFG menerima suatu *string* ialah algoritma Cocke-Younger-Kasami, namun CFG harus berupa Chomsky Normal Form terlebih dahulu.

Context-free grammar adalah beberaupa *rule* yang mengatur tentang seluruh kemungkinan string dalam *context-free language* yang diterima oleh sebuah CFG. Untuk menentukan apakah sebuah string valid terhadap sebuah CFG, digunakan algoritma CYK untuk menentukah apakah string tersebut dapat dicapai dari simbol start non-terminal. Jika ya, maka string masuk dan valid ke dalam CFL. Jika tidak, maka string tidak valid terhadap grammar tertentu, dan harus dicek ulang dengan grammar lainnya.

1.3 Algoritma Cocke-Younger-Kasami (CYK)

Algoritma *Cocke-Younger-Kasami* (CYK) adalah algoritma yang digunakan untuk melakukan proses *parsing* pada *context-free grammar*. Algoritma ini menerima sebuah *context-free grammar* dalam bentuk *Chomsky normal form* dan menguji kemungkinan-kemungkinan untuk membagi *grammar* tersebut ke bentuk yang lebih kecil.

Algoritma CYK dapat dilakukan dengan cara membuat tabel triangular dengan sumbu horizontalnya ialah *string* yang akan diuji seperti berikut.

X_{15}				
X_{14}	X_{25}			
X_{13}	X_{24}	X_{35}		
X_{12}	X_{23}	X_{34}	X_{45}	
X_{11}	X_{22}	X_{33}	X_{44}	X_{55}
a_1	a_2	a_3	a_4	a_5

Tabel 1. Tabel CYK untuk mengecek *string* $a_1a_2a_3a_4a_5$

Sel X_{ij} merupakan himpunan variabel-variabel Chomsky Normal Form yang menerima *string* $a_ia_{i+1}...a_j$. Algoritma untuk menentukan X_{ij} ialah sebagai berikut.

Basis
Isi sel X_{ii} dengan variabel-variabel yang dapat menerima a_i

Rekurens
Sebuah variabel A dapat dimasukkan ke dalam sel X_{ij} jika memenuhi kondisi berikut.
1. $i \leq k < j$
2. Terdapat B pada X_{ik}
3. Terdapat C pada $X_{k+1,j}$
4. $A \rightarrow BC$ ada pada *grammar*

String diterima oleh suatu CFG jika dan hanya jika X_{1n} berisi *state* awal, dengan n merupakan panjang *string* tersebut.

1.4 Sintaks *Python*

1.4.1 Sintaks *Assignment* dan Perbandingan

Assignment dalam bahasa Python menggunakan operator =. Bagian kiri dari operator = haruslah berupa nama variabel yang valid dan bagian kanan dapat berupa sebuah *string*, *integer*, *Boolean*, *list*, dan lain-lain. Sementara itu, sintaks perbandingan menggunakan operator-operator == (sama dengan), != (tidak sama dengan), < (lebih kecil), <= (lebih kecil/sama dengan), > (lebih besar), >= (lebih besar atau sama dengan). Dalam bahasa Python, variabel-variabel yang bisa dibandingkan adalah variabel *integer* (misal 5 < 6) atau variabel *string* (mengacu ke tabel ASCII).

1.4.2 Sintaks Kondisional

Sintaks kondisional dalam bahasa Python menggunakan kata kunci `if, else,` dan `elif (else if).` Sintaks diawali dengan kata kunci if yang diikuti sebuah *statement* yang mengembalikan *Boolean* (perbandingan dua variabel, hasil fungsi, nilai sebuah variabel). Penambahan kondisi lainnya menggunakan kata kunci elif (apabila ingin menambahkan lebih dari satu kondisi) atau else (apabila ingin menambahkan hanya satu kondisi saja.

```
if <statement – 1> :
      <kondisi – 1>
elif <statement – 2> :
      <kondisi – 2>
else:
      <kondisi – 3>
```

1.4.3 Sintaks Perulangan

Ada dua cara melakukan perulangan pada Python: menggunakan operator `while` dan `for`. Operator `while` akan melakukan perulangan selama kondisi pada operator `while` benar, dan operator `for` akan melakukan perulangan sebanyak selisih nilai awal dan nilai akhir yang telah ditentukan. Operator `while` efektif untuk digunakan dalam algoritma pencarian sederhana, sementara operator `for` biasanya efektif untuk digunakan untuk mengakses dan mengolah sebuah *list*.

```
while <kondisi>:
    <statement>
if <kondisi terpenuhi>:
    <stop kondisi>
```

Contoh algoritma `while`

```
for i in range(<nilai awal>, <nilai akhir>, <increment/decrement>):
    <statement>
```

Contoh algoritma `for`

1.4.4 Sintaks Boolean

Kata kunci *Boolean* di Python adalah `True` dan `False`. Ada tiga operator *boolean* yang umum di Python: and (menandakan dan), or (menandakan atau), dan not (negasi). *Grammar* dari operasi *boolean* secara umum berbentuk:

```
<not> <True/False> <and/or> <not> False
```

Contoh operasi *boolean*

Selain itu, sintaks Boolean juga bisa menggunakan kata kunci *is* untuk menandakan persamaan antara dua variabel.

1.4.5 Sintaks *Import*

Sintaks *import* di *Python* mempunyai *grammar*:

```
<from> <nama-file> <import> <fungsi> <as> <kata-kunci>
```

Importing di Python biasanya digunakan untuk menggunakan fungsi atau *class* dari file-file lain secara modular. Untuk penggunaan fungsi dari file lain biasanya digunakan `from` diikuti dengan nama file, namun untuk penggunaan fungsi dari modul-modul yang sudah ada biasanya digunakan kata kunci import diikuti nama modul. Untuk mempersingkat nama fungsi, biasanya digunakan as diikuti kata kunci untuk menggunakan fungsi tersebut (mis. `import numpy as np`, berarti fungsi `numpy` akan digunakan dengan imbuhan `np`)

BAB 2 HASIL CFG DAN FA

2.1 Hasil Implementasi CFG

(hasil implementasi CNF dapat dilihat lebih lanjut di *repository* (Bab 5))

2.1.1 CFG Boolean

```
START -> INST | INST BIN INST | INST NI TWO
TWO -> NIINST NI TWO | NIINST
INST -> INT | VAR | BOOLOPS | STR | LIST | FUNCALL | COMP | NEG | INST BIN INST | False | True | None
NIINST -> BOOLOPS | STR | LIST | FUNCALL | VAR
NI -> notin | in
BIN -> and | or
NEG -> not INST
```

Pada *boolean*, sintaks mungkin berisi “True”, “False”, variabel, perbandingan, atau angka. Sintaks juga bisa melakukan perbandingan dengan tipe data lainnya. Apabila sintaks sudah sesuai dengan sintaks boolean, maka persamaan boolean valid. *Grammar* yang dibuat bersifat rekursif, yang berarti *state* pada *grammar* dapat mengacu pada dirinya sendiri.

2.1.2 CFG Assignment

```
S -> S1 EQ NOT S2 | S1 EQOP S3
S1 -> "VAR"
EQ -> "="
S2 -> "VAR" | "LIST" | "ARITH" | "FUNCALL" | "ASSIGN"
NOT -> "not" | ϵ
S3 -> "INT"
EQOP -> "+=" | "-=" | "//=" | "/=" | "%=" | "*=" | "**=" | "@=" | ":="
```

Pada *assignment*, *grammar* berbentuk S1 EQ NOT S2. S1 mewakili nama variabel, EQ mewakili operator =, NOT mewakili operator not (untuk Boolean) dan S2 mewakili variabel yang dapat di-*assign* (variabel, list, persamaan aritmatika, fungsi, dan assignment). Pada implementasi *parsing*, elemen S2 akan dicek menggunakan *parser*. Apabila elemen S2 valid, maka *grammar* yang diterima akan diuji di algoritma CYK menggunakan algoritma ini. Apabila tidak, maka program akan mengembalikan error sesuai error yang ditemukan.

2.1.3 CFG Conditional

```
S -> IF STATEMENT THEN | ELIF STATEMENT THEN | ELSE THEN
IF -> "if"
STATEMENT -> "STATEMENT"
ELIF -> "elif"
ELSE -> "else"
THEN -> ":"
```

Pada *conditional*, sintaks akan diawali oleh *if, elif,* atau *else. If* dan *elif* akan diikuti dengan *statement*. *Statement* diuji kevalidannya, lalu jika *valid* sintaks akan diuji kesesuaiannya dengan *grammar* menggunakan algoritma CYK. Jika tidak valid, program akan memberikan error sesuai error yang ditemukan.

2.1.4 CFG For Loop

```
START → for VAR in FORINST :
FORINST  → VAR | FUNCALL | LIST | STR | True | False | None
```

Pada *for loop*, sintaks akan diawali dengan *keyword for*, diikuti dengan nama variabel yang akan diiterasi, lalu diikuti *in* dan FORINST, yang dapat berupa variabel, fungsi, *list*, atau *string.* Sintaks akan diuji menggunakan algoritma CYK. Jika tidak valid, program akan memberikan *error* sesuai *error* yang ditemukan.

2.1.5 CFG While Loop

```
START → while WHILEINST :
WHILEINST → VAR | BOOLOPS | FUNCALL | COMP | INT | ARITH
```

Pada *while loop*, sintaks akan diawali dengan *keyword while*, kemudian diikuti dengan WHILEINST, yaitu parameter *loop* yang dapat berupa variabel, fungsi, perbandingan. Sintaks akan diuji menggunakan algoritma CYK. Jika tidak valid, program akan memberikan *error* sesuai *error* yang ditemukan.

2.1.6. CFG Import

```
START → from MV import VAR | from MV import VAR as VAR | import MV | import MV as VAR
MV → METHOD | VAR
```

Penggunaan *keyword import* biasanya diikuti dengan *from* atau *as*. MV pada *grammar* di atas ialah nama *module* yang akan diimport, sedangkan setelah *import* akan berisi nama fungsi atau kelas dan setelah *as* akan berisi nama variabel. Nama fungsi, kelas, dan variabel memiliki ketentuan yang sama, sehingga ketiganya diwakili dengan "VAR".

2.1.7. CFG Function dan Class

```
S -> FUNCTION | CLASS
FUNCTION -> def FUNNAME ARGS :
ARGS -> VAR | VAR ARGS
CLASS -> class FUNNAME : | class FUNNAME ARGS :
```

Pada *function* dan *class*, sintaks berturut-turut diawali oleh *def* dan *class*, lalu diikuti dengan nama fungsi atau kelasnya. Fungsi harus diikuti dengan tanda kurung yang berisi argumen yang berupa variabel, sedangkan kelas tidak harus (namun memungkinkan terdapat argumen). Nama fungsi dan kelas memiliki aturan yang sama dengan nama variabel, sehingga variabel, fungsi, dan kelas akan dicek kevalidannya. Apabila nama fungsi/kelas beserta argumennya valid, *string* yang diterima akan diuji kesesuaiannya dengan *grammar* menggunakan algoritma CYK. Jika tidak valid, program akan memberikan *error* sesuai *error* yang ditemukan.

2.1.8. CFG Pass, Return, Raise

```
S -> PASS | RAISE | RETURN
PASS -> pass
RAISE -> raise EXCEPTION
RETURN -> return STATEMENT
```

Pada *class*, *return*, dan *raise*, *string* akan diawali dengan tiga *keyword* tersebut. *String* yang diawali dengan *pass* tidak dapat menerima kata apapun lagi di belakangnya, sedangkan *return* akan diikuti dengan sebuah *statement* yang nantinya akan dicek apakah *statement* valid menjadi *return value*. *Keyword raise* akan diikuti oleh suatu fungsi bernama Exception. Jika *statement* dan fungsi Exception

valid, *string* akan dicek kesesuaiannya dengan grammar menggunakan algoritma CYK. Jika tidak sesuai dengan *grammar*, program akan mengembalikan *error* sesuai *error* yang ditemukan.

2.2 Hasil Implementasi FA

FA yang diimplementasikan digunakan untuk memvalidasi nama variabel dan fungsi yang ada. Ada tiga buah *state*, yaitu *state* awal, *state* akhir dan *dead state*. FA akan menerima *string* berupa nama variabel yang akan dicek. Apabila nama string tersebut kosong atau string tidak dimulai dengan _ atau alfabet, maka nama variabel salah dan langsung dikirimkan ke *dead state* Dalam implementasinya, *dead state* akan menampilkan *exception.* Jika tidak, nama variabel akan valid dan fungsi mengembalikan True.

BAB 3 IMPLEMENTASI DAN PENGUJIAN

3.1 Implementasi Program

Program terdiri atas tiga elemen: main program, parser, dan CNF. File CNF berisikan grammar-grammar yang telah dibuat dalam bentuk CFG yang telah dikonversikan ke bentuk CNF. File parser terdiri atas beberapa bagian yang telah dipartisi sehingga satu file dibuat untuk mengatur kasus-kasus sintaks tertentu:

• BOOL: mengatur kasus validasi boolean dan perbandingan boolean (ada di LOOP)

• CONDITIONAL: mengatur kasus percabangan (ifelse)

• EQUALS: mengatur kasus penggunaan operasi "=" dan assignment operasional aritmatika (+=, -=, *=)

• IMPORT: mengatur kasus penggunaan sintaks file import (from … import … as …)

• LOOP: mengatur kasus sintaks perulangan (for/while loop)

• MISC: mengatur kasus sintaks freetype (def function, raise, pass)

Parser-parser ini menerima sebuah string, kemudian memvalidasi apakah string tersebut masuk ke kategori yang diinginkan (apakah def function(): adalah grammar yang tepat untuk mendeklarasikan sebuah fungsi). Selain itu, parser juga melakukan reevaluasi terhadap string dan menyesuaikan string tersebut ke bentuk yang dapat diolah ke grammar. Penyesuaian ini dilakukan dengan memberikan *word* yang disesuaikan ulang dengan permintaan grammar (misalkan, variabel foo diubah menjadi "VAR", atau persamaan aritmatika (2 * 3 + 6) diubah menjadi "ARITH"). Setelah itu, string yang telah diubah menjadi *array of words* akan divalidasi dengan algoritma CYK menggunakan grammar yang sudah dibuat. Apabila grammar valid, maka parser akan mengirimkan pesan valid, namun jika tidak, parser akan mengembalikan Exception yang telah dibuat di parser.

Pada program utama, program akan meminta nama file yang akan digunakan (file dapat berbentuk .txt atau .py). Setelah itu, program akan membaca tiap baris di dalam file beserta indentasinya. Indentasi akan digunakan untuk mengecek validitas dari return value dan percabangan kondisi. Baris yang akan diolah adalah baris yang tidak kosong, sehingga hasil baris apabila terdapat *error* adalah baris setelah program dimampatkan.

Setelah membaca tiap baris dan menyimpan nilai indentasi di dalam array of array, program akan mencoba mengecek validitas setiap baris menggunakan parser yang sudah dibuat dan algoritma try-except (apabila pengecekan satu parser gagal, program akan mencoba algoritma parser lainnya hingga ditemukan parser yang sesuai). Jika tidak ada yang sesuai, maka program akan menampilkan pesan error beserta baris dimana terletak error. Jika ada yang sesuai, program akan mengecek kesesuaian peletakan.

• Kata kunci break, continue dan pass hanya bisa diletakkan di dalam perulangan. Jika terletak di luar perulangan, maka akan error

• Percabangan (if/else/elif) hanya bisa diikuti oleh statement baru atau percabangan lainnya. Indentasi digunakan untuk mengecek ini

• return hanya dapat dilakukan apabila indentasi return lebih dari indentasi def

Apabila tidak ada kasus yang salah dalam program, maka program akan menampilkan pesan bahwa program berhasil dibaca dan tidak ada kesalahan. Sebaliknya, akan ditampilkan baris kesalahan program.

3.2 Pengujian Program

```
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"
```

```
 py main.py inputAcc.py

Opening file samples/inputAcc.py ...

Program accepted!
```

Pengujian file `inputAcc.py` (contoh dari spesifikasi). Hasil: program diterima

```
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"
```

```
 py main.py inputReject.py

Opening file samples/inputReject.py ...

Error in line 4 :  x + 2 = 3
```

Pengujian file `inputReject.py` (contoh dari spesifikasi). Hasil: kesalahan di line 4 (x + 2 = 3)

```python
from 5fa import FA_equals as equal
def gaming(count):
    for i in range(1, 10, 3):
        if count % 2 == 0 and flag:
            if count == 3:
                count += 1
            elif count == 4:
                if count == 0:
                    count = 1
                else:
                    if count == 0:
                        count = 1
                count -= 1
            else:
                if count == 0:
                    count = 1
                count += 0
        else:
            count -= 2
            if flag:
                nice()
            else:
                notnice()
    while a == 5:
        count //= 2
    return count
```

```
py main.py input1.py

Opening file samples/input1.py ...

Program accepted!
```

```python
from 5fa import FA_equals as equal
def gaming(count):
    for i in "word":
        if count % 2 == 0 and flag:
            if count == 3:
                count += 1
            elif count == 4:
                if count == 0:
                    count = 1
                else:
                    if count == 0:
                        count = 1
                count -= 1
            else:
                if count == 0:
                    count = 1
                count += 0
        else:
            count -= 2
            if flag:
                nice()
        else:
            notnice()
    while a == 5:
        count //= 2
    return count
```

```
py main.py input1.py

Opening file samples/input1.py ...

Error in line 22 :  else:
Error: else initiated before if
```

Pengujian file `input1.py`. Pada pengujian di kiri, program menguji pemanggilan import, definisi fungsi, percabangan ganda, dan return value. Hasil: diterima. Pada pengujian kanan, pengujian menggunakan perulangan menggunakan string dan indentasi else yang berulang. Hasil: kesalahan pada indentasi else.

```python
#Algoritma:
a = int(input("Masukkan bilangan pertama: "))
b = int(input("Masukkan bilangan kedua: "))
c = int(input("Masukkan bilangan ketiga: "))
d = int(input("Masukkan bilangan keempat: "))
max = 0#Kemungkinan range adalah nilai terbesar, karena nilai terb
if d >= c and d <= b and d >= a: #jika d lebih besar dari nilai la
    max = d
elif a >= b and a >= c and a >= d: #jika a lebih besar dari nilai
    max = a
elif c >= a  c >= b and c >= d: #jika c lebih besar dari nilai lai
    max = c
elif b >= a and b >= c and b >= d:
    max = b #jika b lebih besar dari nilai lain

ans = 0
for i in range(1, max + 1):
    if a % i == 0 and b % i == 0 and c % i == 0 and d % i == 0: #j
        ans = i
print("FPB dari keempat bilangan tersebut adalah ", ans)
```

```python
#Algoritma:
a = int(input("Masukkan bilangan pertama: "))
b = int(input("Masukkan bilangan kedua: "))
c = int(input("Masukkan bilangan ketiga: "))
d = int(input("Masukkan bilangan keempat: "))
max = 0#Kemungkinan range adalah nilai terbesar, karena nilai
if d >= c and d <= b and d >= a: #jika d lebih besar dari nil
    max = d
elif a >= b and a >= c and a >= d: #jika a lebih besar dari n
    max = a
elif c >= a and c >= b and c >= d: #jika c lebih besar dari n
    max = c
elif b >= a and b >= c and b >= d:
    max = b #jika b lebih besar dari nilai lain

ans = 0
for i in range(1, max + 1):
    if a % i == 0 and b % i == 0 and c % i == 0 and d % i ==
        ans = i
print("FPB dari keempat bilangan tersebut adalah ", ans)
```

```
py main.py input2.py

Opening file samples/input2.py ...

Error in line 10 :  elif c >= a  c >= b and c >= d:
```

```
py main.py input2.py

Opening file samples/input2.py ...

Program accepted!
```

Pengujian file `input2.py`. Pada pengujian ini, kesalahan terdapat pada percabangan elif (`elif c >= a c >= b and c >= d).` Apabila diuji, maka program akan mengeluarkan *error* berupa kesalahan

di baris ke 10. Selain itu, pengujian juga dilakukan terhadap komentar per baris (*single-line* comment). Apabila kesalahan di line tersebut diperbaiki, maka program akan memberikan hasil benar.

```
x = int(input("Masukkan X: "))

if x > 0:
    if x % 2 == 0:
        print("X adalah bilangan positif genap")
    elif x % 2 != 0:
        print("X adalah bilangan positif ganjil")
else:
    print("X adalah bilangan negatif")
elif x < 0:
    print("X adalah bilangan negatif")
elif x == 0:
    print("X adalah bilangan nol")
```

```
x = int(input("Masukkan X: "))

if x > 0:
    if x % 2 == 0:
        print("X adalah bilangan positif genap")
    elif x % 2 != 0:
        print("X adalah bilangan positif ganjil")

elif x < 0:
    print("X adalah bilangan negatif")
elif x == 0:
    print("X adalah bilangan nol")
```

```
py main.py input4.py

Opening file samples/input4.py ...

Error in line 9 :  elif x < 0:
Error: elif initiated before if
```

```
py main.py input4.py

Opening file samples/input4.py ...

Program accepted!
```

Pengujian file `input4.py`. Pengujian di bagian kiri mempunyai kesalahan adanya percabangan elif setelah percabangan else. Hasil yang diperoleh adalah pesan error berupa elif yang terinisialisasi sebelum if. Apabila program tersusun benar, maka program akan memberikan output program benar. Pengujian pada program input5.py juga akan menghasilkan error yang sama (perujian percabangan berganda)

```
len = int(input("Input string length: "))
word = input("Input string: ")
arr = word.split()

arr_len = 0
count = 0

for i in word:
    if "a" <= arr[i] <= "z" or "A" <= arr[i] <= "Z":
        arr_len += 1

if arr_len == len:
    for i in range(len // 2):
        if arr[i] == arr[len - i - ]:
            count += 1

    if count == len // 2:
        print("palindrome")
    else:
        print("not a palindrome")
else:
    print("Different string length.")
```

```
len = int(input("Input string length: "))
word = input("Input string: ")
arr = word.split()

arr_len = 0
count = 0

for i in word:
    if "a" <= arr[i] <= "z" or "A" <= arr[i] <= "Z":
        arr_len += 1

if arr_len == len:
    for i in range(len // 2):
        if arr[i] == arr[len - i - 1]:
            count += 1

    if count == len // 2:
        print("palindrome")
    else:
        print("not a palindrome")
else:
    print("Different string length.")
```

```
py main.py input6.py

Opening file samples/input6.py ...

Syntax Error in line 11 :  if arr[i] == arr[len - i - ]:
```

```
py main.py input6.py

Opening file samples/input6.py ...

Program accepted!
```

Pada pengujian `input6.py`, program mengecek validitas *list element*, perbandingan, dan perulangan. Apabila ada kesalahan *grammar* pada *list element* (arr[len – i -]) maka program akan menampilkan pesan *error*. Sebaliknya, jika tidak ada kesalahan, program akan mengeluarkan pesan benar.

```
def someFunc(a,b,c):
    return 5 <= 6 and "a" in "flag"
return 0
def list():
    N = int(input("Input N: "))
    a = [1,2,3]

    b = 0
    for i in range(N):
        for j in range(i, N):
            if a[j] - a[i] >= b:
                b = a[j] - a[i]
            else:
                print("something")
    if ((b)>0 and someFunc(a,b,1)):
        print("nice"
    else:
        print("not nice")
```

```
def someFunc(a,b,c):
    return 5 <= 6 and "a" in "flag"

def list():
    N = int(input("Input N: "))
    a = [1,2,3]

    b = 0
    for i in range(N):
        for j in range(i, N):
            if a[j] - a[i] >= b:
                b = a[j] - a[i]
            else:
                print("something")
    if ((b)>0 and someFunc(a,b,1)):
        print("nice")
    else:
        print("not nice")


return 0
```

```
py main.py input7.py

Opening file samples/input7.py ...

Error in line 3 :  return 0
return not initiated with def
```

```
Opening file samples/input7.py ...

Error in line 17 :  return 0
return not initiated with def
```

Pengujian file `input7.py` meliputi pendefinisian fungsi, pengujian aritmatika *array* dan persamaan boolean. Apabila ada pendefinisian return di luar def, maka akan muncul pesan error berupa inisiasi return yang tidak bersamaan dengan def. Sebaliknya, apabila sintaks program benar, maka program akan mengeluarkan input benar.

```
n = int(input("Masukkan banyaknya elemen: "))
arr_a = []
arr_b = []
print("Elemen A: ")
for i in range(n):
    arr_a[i] = int(input())
print("Elemen B: ")
for i in range(n):
    arr_b[i] = int(input())

selisih_a = arr_a[1] - arr_a[0]
selisih_b = arr_b[1] - arr_b[0]
count = 0
if selisih_a == selisih_b:
    for i in range(1, n - 1):
        if arr_a[i + 1] - arr_a[i] == selisih_a and selisih_b == arr_b[i + 1] - arr_b[i]:
            count += 1
if count == n - 2:
    print("A dan B merupakan double magic sequence")
else:
    print("A dan B bukan merupakan double magic sequence")
```

```
py main.py input3.py

Opening file samples/input3.py ...

Program accepted!
```

Pengujian pada file `input3.py` juga melibatkan pengecekan aritmatika string dan assignment. Apabila program dijalankan, program akan memberikan output benar,

```python
def aFunction():
    return "a" and 5 <= 6 * 8

a = [1,2,3,4,5,[6,[7,8]]]
b = int(input("Input number"))
str = "yahoo"
for i in range(1, 3, 1):
    for j in range(1, 3, 2):
        for k in range(6, 9):
            while aFunction() and True:
                if "aho" not in "yahoo":
                    continue
                elif "yah" in "yahoo":
                    pass
                else:
                    break

isTrue = "Yes" if aFunction() == "yahoo" else "No"
```

```
py main.py inputLoops.py

Opening file samples/inputLoops.py ...

Program accepted!
```

Pengujian pada file `inputLoops.py` melibatkan perulangan berganda, deklarasi fungsi, deklarasi list berganda, pengecekan string di dalam string menggunakan kata kunci in, dan kata kunci untuk perulangan. Apabila program dijalankan tanpa diubah, maka program akan berjalan dan output yang akan diberikan benar (meskipun program tidak akan berhenti secara semantik karena yang dicek hanyalah sintaks).

```python
def aFunction():
    return "a" and 5 <= 6 * 8

a = [1,2,3,4,5,[6,[7,8]]]
b = int(input("Input number"))
str = "yahoo"
for i in range(1, 3, 1):
    for j in range(1, 3, 2):
        for k in range(6, 9):
            while aFunction() and True:
                if "aho" not in "yahoo":
                    continue
                elif "yah" in "yahoo":
                    pass
                else:
break

isTrue = "Yes" if aFunction() == "yahoo" else "No"
```

```python
def aFunction():
    return "a" and 5 <= 6 * 8

a = [1,2,3,4,5,[6,[7,8]]]
b = int(input("Input number"))
str = "yahoo"
for i in range(1, 3, 1):
    for j in range(1, 3, 2):
        for k in range(6, 9):
            while aFunction() and True:
                if "aho" not in "yahoo":
                    continue
                elif "yah" in "yahoo":
pass
                else:
                    break

isTrue = "Yes" if aFunction() == "yahoo" else "No"
```

Apabila ada kata kunci yang terletak di tempat yang salah, program akan mengembalikan pesan error.

```
py main.py inputLoops.py

Opening file samples/inputLoops.py ...

Error: no loop initiated for context keywords (break/continue/pass)
```

BAB 4 KESIMPULAN

Algoritma CYK adalah algoritma *parsing* yang digunakan untuk melakukan *parsing* terhadap bahasa Python. Algoritma ini mengimplementasikan pencocokan sebuah kalimat/ekspresi dengan sebuah *context-free grammar* yang telah dibuat dalam bentuk *Chomsky normal form*. Dalam implementasinya dalam bahasa Python, grammar yang dibuat diimplementasikan dalam bentuk *tuples* dan digunakan untuk memvalidasi sebuah *array* berisikan kalimat yang telah dikonversi dalam bentuk *grammar* yang tepat. Untuk melakukan *parsing*, dalam digunakan algoritma *try-except* untuk mencoba tiap kemungkinan yang ada bagi sebuah ekspresi sehingga ekspresi tersebut valid. Selain itu, dibutuhkan waktu yang lama untuk melakukan riset, pengetesan, dan penyesuaian ulang ekspresi terhadap format yang dapat divalidasi oleh CYK.

Saran penulis adalah untuk memulai pengerjaan dari validasi tipe data (*integer, string, list*). Apabila validasi tipe data sudah sukses, maka pembuatan *parser* akan lebih mudah dilakukan karena validasi tipe data sudah diatur sehingga yang perlu dilakukan adalah menganalisis *grammar* yang ada. Apabila validasi *grammar* benar, *main program* baru dapat disusun. Penulis juga menyarankan untuk menerapkan algoritma rekursif untuk pengecekan *grammar* sehingga *grammar* yang sifatnya berulang dapat divalidasi.

BAB 5 LAMPIRAN

Repository Github: https://github.com/clumsyyyy/TubesTBFO-CYK-Python-Parser

Pembagian Tugas:

1. 13520124 – Owen Christian Wijaya

- Pembuatan *main program*
- Pembuatan CFG dan *parsing boolean*. kondisional, dan *assignment*
- Validasi persamaan aritmatika dan *list*
- Pengujian program dan sampel program
- Pembuatan laporan

2. 13520152 – Muhammad Fahmi Irfan
- Pembuatan laporan
- Pembuatan CFG dan parsing kondisional dan *freetype*

3. 13520157 – Thirafi Najwan Kurniatama
- Pembuatan CFG dan *parsing* perulangan, *import*
- Validasi struktur data dan *list*
- *Testing* dan *debugging*
- Algoritma CYK
- Penyempurnaan program