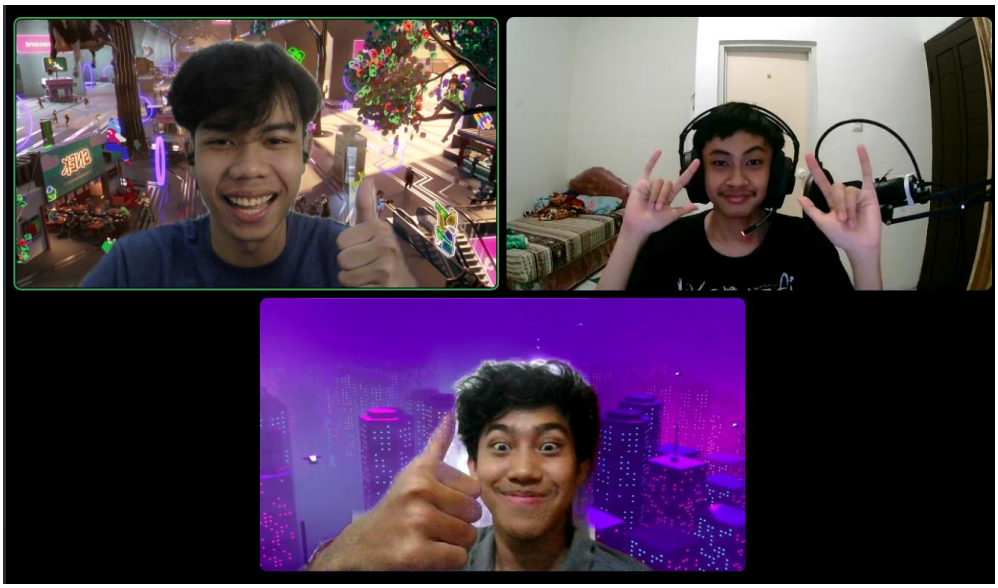
COMPILER BAHASA PYTHON
AYAM BERSIH BERKAH

LAPORAN TUGAS BESAR

Diajukan untuk memenuhi Tugas Teori Bahasa Formal dan Automata



oleh

MUHAMMAD GAREBALDHIE ER RAHMAN	**13520029**
I GEDE ARYA RADITYA PARAMESWARA	**13520036**
ADIYANSA PRASETYA WICAKSANA	**13520044**

TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2021

DAFTAR ISI

BAB I

DESKRIPSI MASALAH

Python adalah bahasa *interpreter* tingkat tinggi (*high level)*, dan juga *general-purpose.* Python dibuat oleh Guido Van Rossuum dan dirilis pertama kali pada tahun 1991. Filosofi desain pemrograman Python mengutamakan *code readability* dengan penggunaan *whitespace*-nya. Python adalah bahasa multiparadigma karena mengimplementasi berbagai paradigma yaitu fungsional, imperatif, berorientasi objek, dan reflektif.

Dalam proses pembuatan program dari sebuah bahasa menjadi instruksi yang dapat dieksekusi oleh mesin, terdapat pemeriksaan sintaks atau kompilasi bahasa yang dibuat oleh programmer. Kompilasi ini bertujuan untuk memastikan instruksi yang dibuat oleh *programmer* mengikuti aturan yang sudah ditentukan oleh bahasa tersebut. Baik bahasa yang menggunakan *interpreter* atau *compiler,* keduanya pasti melakukan pemeriksaan sintaks.

Dibutuhkan suatu *grammar* bahasa dan algoritma *parser* untuk melakukan kompilasi dan pemeriksaan *syntax*. Pada tugas besar ini, kelompok kami akan mengimplementasikan *compiler* untuk bahasa Python yang melakukan pemeriksaan terhadap *statement-statement* dan *syntax* bawaan Python. Pemeriksaan ini akan dilakukan dengan konsep CFG yang telah dipelajari dalam mata kuliah Teori Bahasa Formal dan Automata.

BAB II
DASAR TEORI

2.1. CONTEXT-FREE GRAMMAR

CFG atau Context Free Grammar adalah tata bahasa formal di mana setiap aturan produksi adalah dalam bentuk A → B di mana A adalah pemproduksi, dan B adalah hasil produksi. Batasannya hanyalah ruas kiri adalah sebuah simbol variabel. Dan pada ruas kanan bisa berupa terminal, symbol, variable ataupun ε, Contoh aturan produksi yang termasuk CFG adalah seperti berikut ini:

X → bY | Za

Y → aY | b

Z → bZ | ε

CFG adalah tata bahasa yang mempunyai tujuan sama seperti halnya tata bahasa regular yaitu merupakan suatu cara untuk menunjukkan bagaimana menghasilkan suatu untai-untai dalam sebuah bahasa.

Definisi formal dari CFG dapat didefinisikan sebagai berikut.

G = (V, T, P, S)

V = himpunan terbatas variabel

T = himpunan terbatas terminal

P = himpunan terbatas dari produksi

S = start symbol

CFG menjadi dasar dalam pembentukan suatu proses analisis secara sintaksis. Bagian sintaks dalam suatu *compiler* kebanyakna didefinisikan dalam CFG. Pohon penurunan (*parse tree*) berguna untuk menggambarkan simbol-simbol variabel menjadi terminal. Setiap dari seimbol variabel akan diturunkan menjadi terminal sampai tidak ada lagi yang masih variabel.

Proses *parsing* tersebut bisa dilakukan sebagai berikut:

- *Leftmost Derivation*, yaitu simbol variabel yang paling kiri diganti terlebih dahulu.
- *Rightmost Derivation*, yaitu simbol variabel yang paling kanan diganti terlebih dahulu.

2.2. CHOMSKY NORMAL-FORM

Chomsky Normal Form atau biasa disebut sebagai CNF merupakan salah satu bentuk normal dari *Context Free Grammar* (CFG). CNF dapat dibuat dari CFG dengan melalui tahap penyederhanaan yaitu penghilangan produksi yang "tidak penting", unit, dan ε. Sehingga, CNF memiliki syarat yaitu tidak memiliki produksi *useless*, tidak memiliki produksi unit, dan tidak memiliki ε**.** Bentuk dari CNF produksi CNF adalah sebagai berikut.

A → BC atau A → a

A = variabel

B = teminal

Untuk membeuat sebuah CNF terdapat langkah-langkah pembentukannya yang secara umum dapat dideskripiskan sebagai berikut.

- Biarkan aturan produksi yang sudah dalam bentuk CNF.
- Penggantian aturan produksi yang ruas kanannya memuat symbol teriman dan dengan panjang ruas kanannya lebih dari satu.
- Penggantian aturan produksi yang ruas kanannya memuat lebih dari dua simbol variable.
- Penggantian tersebut dilakukan berkali-kali sampai pada akhirnya aturan produksi memenuhi aturan dari CNF.

2.3. COCKE-YOUNGER KASAMI

Cocke-Younger Kasami atau biasa disebut sebagai CYK merupakan salah satu algoritma *parsing* yang diguanakan dalam CFG. Untuk dapat menggunakan CYK ke dalam sebuah *grammar*, bentuk dari *grammar* tersebut harus dalam bentuk CNF. Algoritma menggukan *dynamic programming* untuk mentutkan apakah suatu *string* termasuk ke dalam sebuah grammar.

Sebagai contoh diberikan aturan produksi sebagai berikut.

S -> AB | BC

A -> BA | a

B -> CC | b

C -> AB | a

Dilakukan pengecekan terhadap string "baaba" denga menggunakan algoritma CYK, hasil dari pengecekan dimasukkan ke dalam tabel.

	b	a	a	b	a
b	{B}	{S,A}	Φ	Φ	{S,A,C}
a		{A,C}	{B}	{B}	{S,A,C}
a			{A,C}	{S,C}	{B}
b				{B}	{S,A}
a					{A,C}

Gambar 2.1 Tabel Pengecekan CYK

Dari tabel, bisa dilihat bahwa *start symbol* berada pada tabel (1, n) dimana n disini merupakan panjang dari *string* yang dilakukan pengecekan. Sehingga, karena *start symbol* berada pada tabel (1, n) *string* "baaba" memenuhi aturan produksi dari *grammar* tersebut.

2.4. BAHASA PEMROGRAMAN PYTHON

Python adalah bahasa *interpreter* tingkat tinggi (*high level)*, dan juga *general-purpose.* Python dibuat oleh Guido Van Rossuum dan dirilis pertama kali pada tahun 1991. Filosofi desain pemrograman Python mengutamakan *code readability* dengan penggunaan *whitespace*-nya. Python adalah bahasa multiparadigma karena mengimplementasi berbagai paradigma yaitu fungsional, imperatif, berorientasi objek, dan reflektif.

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 prakteknya 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. Saat ini, kode Python dapat dijalankan di berbagai platform system operasi, beberapa diantaranya adalah Linux/Unix, Windows, Mac, Java Virtual Machine, Amiga, Palm, dan Symbian.

Dalam prosesnya, pembuatan program dari sebuah bahasa menjmadi instruksi yang dapat diekseksi oleh mesin akan terdapat pemeriksaan sintaks atau kompilasi bahasa yang dibuat oleh *programmer*. Kompilasi ini bertujuan untuk memastikan instruksi yang dibuat telah mengikuti aturan yang sudah ditentukan oleh bahasa tersebut. Dalam hal ini, Python secara bawaan sudah menyediakan pemeriksaan sintaks ini, sehingga ketika program akan dieksekusi oleh mesin, sudah diperiksa bahwa *code* yang ditulis sudah valid. Terdapat kata kunci dalam bahasa Python yang sudah memiliki fungsinya sendiri, sehingga tidak bisa digunakan sebagai variabel. Kata kunci tersebut akan digunakan sebagai beberapa terminal pada *grammar* untuk dilakukan pemeriksaan sintaks. Kata kunci tersebut adalah sebagai berikut.

False	class	is	break	return
None	continue	for	not	while
True	def	from	or	with
and	elif	if	pass	in
as	else	import	raise	

Tabel 2.1 Kata Kunci Bahasa Pemrograman Python

BAB III

ANALISIS PERSOALAN

3.1. CFG Production

Berikut ini adalah hasil *Context-Free Grammar* yang telah kelompok kami buat terkait *compiler* bahasa Python.

G = (V, T, P, S)

Non-Terminal Symbol/Variabel (V)

ADD	ALL	AND	ARIT_OP	ARIT_OPERATOR	ARRAY
ARRAY	ARROW	AS	ASSIGN	ASSIGNMENT	ASSIGNMENT_OP
ASSIGNMENT_STMT	BOOL	BREAK	CLASS	CLASS_SENTENCE	CLASS_STMT
COLON	COMMA	CONTINUE	DEF	DICT	DIV
DIVEQ	DOT	DOTBETWEEN	ELIF	ELIF_BANYAK_FUNC	ELIF_STMT
ELIF_STMT_FUNC	ELSE	ELSE_STMT	ELSE_STMT_FUNC	ENTER	EQUAL
FALSE	FOR	FORMAT	FOR_STMT	FROM	FUNCTION
FUNC_SENTENCE	FUNC_STMT	G	GE	IF	IF_BLOCK
IF_BLOCK_FUNC	IF_STMT	IF_STMT_FUNC	IMPORT	IMPORT_STMT	IN
INT	IS	ISEQ	L	LB	LCB
LE	LIST	LOGI_OP	LOGI_OPERATOR	LOGI_VAR	LOOP_SENTENCE
LSB	METHOD_STMT	MOD	MUL	MULEQ	NEQ
NEWLINE	NONE	NOT	OPERATION	OR	PARAM
PARAMDICT	PARAMFUNGSI	PART	PASS	RAISE	RANGE
RANGE_STMT	RB	RCB	RETURN	RSB	S
SENTENCE	SS	STATIC	STRING	STRINGFORMAT	SUBTR
SUBTREQ	SUMEQ	THEN	TRUE	TYPE	VAR
VARINT	VAR_FUNC	WHILE	WHILE_STMT	WITH	WITH_STATE

Tabel 3.1 List Nonterminal Symbols

Terminal Symbol(T)

add	and	arrow	as	break	class
colon	comma	continue	def	div	diveq
dot	dotbetween	elif	else	equal	false
for	format	from	g	ge	if
import	in	int	is	iseq	l
lb	lcb	le	lsb	mod	mul
muleq	multiline	neq	newline	none	not
or	pass	raise	range	rb	rcb
return	rsb	string	subtr	subtreq	sumeq
then	true	type	var	while	with

Tabel 3.2 List Terminal Symbols

Production(P)

Production yang kami buat sejumlah 288 baris yang telah mengimplementasikan CFG, Terminal serta non terminal symbol nya yang nantinya akan diolah menjadi CNF.

START SYMBOL(S) :

Start symbol yang kami pakai ialah **S** sehingga akan dilakukan pengecekan apabila sebuah grammar bisa menuju **S** maka grammar itu akan diterima

3.2. CNF Production

Chomsky Normal Form yang dibuat merupakan hasil dari Context Free Grammar yang diconvert dengan fungsi yang telah dibuat untuk kelompok kami. Dikarenakan jumlah dari CNF yang dihasilkan terlalu banyak untuk dimuat di laporan, maka hasil dari CNF yang dihasilkan tidak dimuat di laporan ini.

BAB IV

IMPLEMENTASI FUNGSI

4.1. Module lib File tokenizer.py

File tokenizer.py berisi mengenai prosedur dan fungsi untuk membaca sebuah file yang berisi bahasa Python dan akan membaca tiap kata ataupun simbol dalam file tersebut menjadi sebuah token yang merupakan terminal dari grammar.

No.	Fungsi / Prosedur	Tujuan
1	`lex`	Melakukan pencocokan terhadap setiap karakter dan menyimpannya ke dalam sebuah array yang berisi token
2	`createToken`	Membaca sebuah file py ataupun txt dan mengubahnya menjadi token dengan fungsi `lex`

4.2. Module lib File cfgtocnf.py

File cfgtocnf.py berisi mengenai prosedur dan fungsi dalam mengkonversi bentuk CFG menjadi bentuk CNF. File ini akan membaca sebuah *grammar* dalam file cfg.txt dan akan melakukan konversi sekaligus melakukan *write* ke dalam file cnf.txt.

No.	Fungsi / Prosedur	Tujuan
1	`readGrammarFile`	Melakukan pembacaan file yang berisi grammar
2	`printGrammar`	Sebagai fungsi helper untuk mengecek fungsi grammar
3	`addGrammarRule`	Menambahkan rule ke dalam *global variable* yang telah didefinisikan berupa *dictionary/hash table*
4	`convertGrammar`	Melakukan konversi dari bentuk CFG ke bentuk CNF
5	`mapGrammar`	Melakukan mapping dari CNF yang dihasilkan ke dalam bentuk dictionary untuk selanjutnya diolah dengan fungsi *parser* yang telah dibuat.
5	`writeGrammar`	Melakukan penulisan ke dalam sebuah file setelah dilakukan konversi

4.3. Module lib File parser.py

File parser.py berisi mengenai fungsi untuk melakukan pengecekan sintaksis terhadap token yang telah diolah dan akan dicocokan dengan CNF. File ini akna menghasilkan *verdict* dari file py yang akan dilakukan pengecekan.

No.	Fungsi / Prosedur	Tujuan
1	`cykParse`	Melakukan *parsing* terhadap token yang diinput berdasarkan *dictionary* yang berisi CNF, *parsing* dilakukan dengan algoritma CYK. Akan bernilai

		True apabila Start symbol berada pada token yang telah diinput

4.4. File main.py

File main.py merupakan source code yang berisi main program dari *compiler* bahasa Python. File ini akan melakukan import import ke semua file yang lain. Program ini akan melakukan pembacaan file py lalu mengubahnya menjadi token dengan fungsi dari tokenizer.py, lalu CFG yang telah dibuat akan diubah menjadi CNF, dan akhirnya akan dilakukan *parsing* terhadap token dihasilkan dari pembacaan tersebut. File ini akan menampilkan output berupa hasil kompilasi dari program python yang ingin dilakukan pengecekan.

BAB V

EKSPERIMEN

Capture Kasus Uji

Dalam bagian ini, kami telah melakukan beberapa kasus uji terhadap compiler bahasa python yang telah kami buat. Berikut ini adalah beberapa kasus uji yang telah kami buat.

Gambar 5.1 Tampilan Awal Python Compiler

5.1 Boolean

Code:

```
1 == 1
5 > 3
True or False
10 <= 1
3 > 7
True and False
```

Hasil:

```
=====VERDICT=====
=========
Loading...
['int', 'iseq', 'int', 'newline', 'int', 'g',
 'int', 'newline', 'true', 'or', 'false', 'ne
wline', 'int', 'le', 'int', 'newline', 'int',
 'g', 'int', 'newline', 'true', 'and', 'false
']
{'FUNC_SENTENCE', 'CLASS_SENTENCE', 'S', 'SEN
TENCE', 'PART', 'LOOP_SENTENCE', 'SS'}
Berhasil di compile yayyy :))
```

5.2. Break

Code:

```
for i in range(1, 11):
    if i == 5:
        break
    print(i)
```

Hasil:

```
======================VERDICT================
=========
Loading...
['for', 'var', 'in', 'range', 'lb', 'int', 'c
omma', 'int', 'rb', 'colon', 'newline', 'if',
 'var', 'iseq', 'int', 'colon', 'newline', 'b
reak', 'newline', 'var', 'lb', 'var', 'rb']
{'LOOP_SENTENCE', 'SENTENCE', 'PART', 'SS', '
S', 'FUNC_SENTENCE', 'CLASS_SENTENCE', 'FOR_S
TMT'}
Berhasil di compile yayyy :))
```

5.3. Class

Code:

```
class ExampleClass:
    def function1(parameters):
        print(a)
    def function2(parameters):
        print(b)
```

Hasil:

```
======================VERDICT================
=========
Loading...
['class', 'var', 'colon', 'newline', 'def', '
var', 'lb', 'var', 'rb', 'colon', 'newline',
'var', 'lb', 'var', 'rb', 'newline', 'def', '
var', 'lb', 'var', 'rb', 'colon', 'newline',
'var', 'lb', 'var', 'rb']
{'SS', 'CLASS_STMT', 'S', 'PART'}
Berhasil di compile yayyy :))
```

5.4. Conditional

Code:

```
if a == 1:
    print('One')
elif a == 2:
    print('Two')
else:
    print('Something else')
```

Hasil:

```
====================VERDICT================
=========
Loading...
['if', 'var', 'iseq', 'int', 'colon', 'newlin
e', 'var', 'lb', 'string', 'rb', 'newline', '
elif', 'var', 'iseq', 'int', 'colon', 'newlin
e', 'var', 'lb', 'string', 'rb', 'newline', '
else', 'colon', 'newline', 'var', 'lb', 'stri
ng', 'rb']
{'IF_BLOCK', 'S', 'IF_BLOCK_FUNC', 'SENTENCE'
, 'PART', 'FUNC_SENTENCE', 'SS', 'LOOP_SENTEN
CE', 'CLASS_SENTENCE'}
Berhasil di compile yayyy :))
```

5.5. Import

Code:

```
from math import ceil
import matplotlib as plt
import PIL
```

Hasil:

```
====================VERDICT=======================
Loading...
['from', 'var', 'import', 'var', 'newline', 'import', 'var'
, 'as', 'var', 'newline', 'import', 'var']
{'S'}
Berhasil di compile yayyy :))
```

5.6. In

Code:

```
5 in a
10 in a

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

Hasil:

```
======================VERDICT=========================
Loading...
['int', 'in', 'var', 'newline', 'int', 'in', 'var', 'newlin
e', 'newline', 'for', 'var', 'in', 'string', 'colon', 'newl
ine', 'var', 'lb', 'var', 'rb']
{'S', 'LOOP_SENTENCE', 'PART', 'SS', 'CLASS_SENTENCE', 'SEN
TENCE', 'FUNC_SENTENCE'}
Berhasil di compile yayyy :))
```

5.7. Is

Code:

```
True is True
None is None
False is False
[] == []
```

Hasil:

```
======================VERDICT=========================
Loading...
['true', 'is', 'true', 'newline', 'none', 'is', 'none', 'ne
wline', 'false', 'is', 'false', 'newline', 'lsb', 'rsb', 'i
seq', 'lsb', 'rsb']
{'SS', 'S', 'FUNC_SENTENCE'}
Berhasil di compile yayyy :))
```

5.8. Logical

Code:

```
True or False
1 == 2 or 3 == 4
False or False
True or True
not True
not False
not 1 or not 2
```

Hasil:

```
======================VERDICT========================
Loading...
['true', 'or', 'false', 'newline', 'int', 'iseq', 'int', 'o
r', 'int', 'iseq', 'int', 'newline', 'false', 'or', 'false'
, 'newline', 'true', 'or', 'true', 'newline', 'not', 'true'
, 'newline', 'not', 'false', 'newline', 'not', 'int', 'or',
 'not', 'int']
{'PART', 'SENTENCE', 'S', 'SS', 'LOOP_SENTENCE', 'CLASS_SEN
TENCE', 'FUNC_SENTENCE'}
Berhasil di compile yayyy :))
```

5.9. Loop

Code:

```
for i in range(10):
    print(i)
```

Hasil:

```
======================VERDICT========================
Loading...
['for', 'var', 'in', 'range', 'lb', 'int', 'rb', 'colon', '
newline', 'var', 'lb', 'var', 'rb']
{'FUNC_SENTENCE', 'FOR_STMT', 'SS', 'SENTENCE', 'LOOP_SENTE
NCE', 'PART', 'S', 'CLASS_SENTENCE'}
Berhasil di compile yayyy :))
```

5.10. Pass

Code:

```
def function(args):
    pass

...
class example:
    pass
```

Hasil:

```
======================VERDICT========================
Loading...
['def', 'var', 'lb', 'var', 'rb', 'colon', 'newline', 'pass
', 'newline', 'newline', 'class', 'var', 'colon', 'newline'
, 'pass']
{'SS', 'S'}
Berhasil di compile yayyy :))
```

5.11. Raise

Code:

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

Hasil:

```
======================VERDICT=========================
Loading...
['if', 'var', 'iseq', 'int', 'colon', 'newline', 'raise', '
var', 'lb', 'string', 'rb']
{'SS', 'IF_STMT_FUNC', 'IF_BLOCK_FUNC', 'FUNC_SENTENCE', 'S
'}
Berhasil di compile yayyy :))
```

BAB VI

PENUTUP

6.1. Kesimpulan

Berdasarkan tugas besar “Compiler Bahasa Python”, program yang kami buat berjalan cukup baik mengingat masih banyaknya kekurangan dalam program kami. Pengujian yang dilakukan juga belum memuat semua sintaks bahasa Python sehingga kami tidak bisa mengetahui batasan dari program yang kami buat. CFG yang kami buat juga tidak sepenuhnya meliputi keseluruhan sintaks dari bahasa Python sehingga untuk kasus yang lebih banyak ataupun asing, bisa terjadi beberapa kesalahan.

6.2. Saran

Berdasarkan pengerjaan yang dilakukan oleh kelompok kami, terdapat beberapa saran yang dapat kami berikan untuk tugas besar ini.

1. Pengaturan waktu yang lebih baik dalam pengerjaan tugas ini sehingga *code* yang dituliskan juga dapat lebih baik dan dapat menangani *error* yang lebih banyak lagi.
2. Perbanyak referensi mengenai *Context-Free Grammar* sehingga bisa menghasilkan algoritma yang lebih sederhana dan efisien.
3. Pengerjaan tugas besar dilakukan pada waktu yang lebih awal, sehingga tidak terlalu mendekati *deadline*, agar program yang dibuat dapat menjadi lebih “bersih” dan bisa melakukan pengujian terhadap lebih banyak *test case*.

REFERENSI

https://www.javatpoint.com/automata-chomskys-normal-form

https://www.tutorialspoint.com/automata_theory/context_free_grammar_introduction.htm

https://www.wikiwand.com/en/CYK_algorithm

https://www.geeksforgeeks.org/cocke-younger-kasami-cyk-algorithm/