Tugas Besar Teori Bahasa Formal dan Automata

Parser Bahasa JavaScript (Node.js)

Kelompok 5 – Beban Alek

Hosea Nathanael Abetnego	**13521057**
Alex Sander	**13521061**
Ariel Jovananda	**13521086**

TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2022

DAFTAR ISI

BAB I : Teori Dasar

1. *Context Free Grammar (CFG)*

Context Free Grammar (CFG), sering dikenal sebagai bahasa bebas konteks, adalah bahasa formal yang aturan tata bahasa produksinya berbentuk $A \rightarrow \alpha$, di mana A adalah simbol nonterminal dan α simbol terminal (yang mungkin berupa string kosong atau dilambangkan dengan ε). CFG pada dasarnya menyelesaikan tugas yang sama seperti tata bahasa biasa, yaitu menghasilkan aturan produksi yang dapat mengkarakterisasi setiap kemungkinan string dalam bahasa formal tertentu.

Context Free Grammar (CFG) secara formal didefinisikan dengan empat tupel, yaitu:

G = (V, T,P, S)

Di mana V adalah variabel dengan himpunan hingga, T adalah terminal dengan himpunan hingga, P adalah produksi dengan himpunan hingga, dan S adalah simbol awal.

Hasil produksi tidak dibatasi oleh pedoman produksi CFG. Ada batasan di sisi kanan produksi tulisan CFG, dan hanya ada satu simbol non-terminal di sebelah kiri. Berikut ilustrasi aturan produksi CFG:

$$S \rightarrow AB$$

$$S \rightarrow aabB$$

2. *Chomsky Normal Form*

Context Free Grammar (CFG) datang dalam bentuk *Chomsky Normal Form (CNF),* yang memiliki batasan lebih ketat daripada *Context Free Grammar (CFG*) . *Chomsky Normal Form (CNF),* adalah penyederhanaan *Context Free Grammar (CFG*) yang memudahkan untuk menentukan apakah menilai suatu string adalah bagian dari bahasa formal tertentu.

Hasil produksi hanya antara dua variabel dan satu terminal menurut . *Chomsky Normal Form (CNF)*. Pedoman produksi adalah:

$$A \rightarrow a$$

atau

$$A \rightarrow BC$$

Di mana α adalah terminal dan A, B, dan C adalah variabel atau nonterminal. *Chomsky Normal Form (CNF)* perlu memenuhi prasyarat *Context Free Grammar (CFG*), yaitu, tidak menghasilkan string kosong (ε), produksi bebas, atau unit produksi.

Berikut adalah langkah-langkah untuk mengonversi tata bahasa bebas konteks ke bentuk normal Chomsky:

1. Apabila start symbol, S, terdapat pada beberapa bagian ruas kanan suatu produksi, buatlah start symbol baru, S′, dan produksi baru;
2. Hapus semua null productions dan unit productions;
3. Ganti setiap produksi A → B1B2 ... Bn, untuk n > 2, dengan A → B1C, dimana C →B2 ... Bn. Ulangi hingga semua produksi hanya mengandung dua nonterminal/variabel pada ruas kanannya;
4. Apabila hasil produksinya berbentuk A → aB, ganti produksi tersebut dengan A → XB dan X → a. Ulangi hingga semua produksi hanya dalam bentuk A → a atau A → BC.

Contoh:

Suatu CFG dengan aturan produksi:

$$P: S \rightarrow ASA \mid aB,\ A \rightarrow B|S,\ B \rightarrow b|\varepsilon$$

diubah menjadi bentuk CNF-nya menjadi

$$P : S' \rightarrow AX \mid YB \mid a \mid AS \mid SA,$$
$$S \rightarrow AX \mid YB \mid a \mid AS \mid SA,$$
$$A \rightarrow b \mid AX \mid YB \mid a \mid AS \mid SA,$$

$$B \rightarrow b,$$
$$X \rightarrow SA,$$
$$Y \rightarrow a$$

3. Cocke-Younger Kasami

Algoritme penguraian yang disebut Cocke-Younger Kasami (CYK) menentukan apakah string milik bahasa formal tertentu untuk *Context Free Grammar (CFG*).

Algoritme CYK ini hanya bekerja dengan tata bahasa yang dalam *Chomsky Normal Form (CNF)*. Tabel array dua dimensi dengan ukuran n × n − 1 × ... × 1, digunakan sebagai struktur data untuk *Cocke-Younger Method*, di mana n adalah jumlah kata dalam sebuah string.

Hasil *parsing* dari *array* sebelumnya digunakan oleh metode CYK, untuk memeriksa apakah sebuah *string* memiliki merupakan komponen dari bahasa formal. Untuk melihat apakah suatu *string* merupakan komponen dari bahasa forma,l *start* pada baris di atas (x14 dalam tabel) akan diperiksa . Apabila ada *start symbol* maka *string* merupakan bagian dari bahasa yang di periksa dan sebaliknya.

Contoh:

Akan dicek apakah suatu string “baaba” diterima pada suatu *CFG* yang didefinisikan sebagai berikut:

$$G = (\{S, A, B, C\},\{a, b\}, P, S)$$

dan memiliki aturan produksi

$$P : S \rightarrow AB \mid BC,$$
$$A \rightarrow BA \mid a,$$
$$B \rightarrow CC \mid b,$$
$$C \rightarrow AB \mid a$$

Berdasarkan algoritma *CYK*, akan terbentuk tabel:

{S, A, C}				
{ϕ}	{S, A, C}			
{ϕ}	{B}	{B}		
{S, A}	{B}	{S, C}	{S, A}	
{B}	{A, C}	{A, C}	{B}	{A, C}

Karena *start symbol* (S) terdapat pada baris *cell* paling atas, maka *string* "baaba" diterima oleh *CFG* tersebut.

4. *Finite State Automata*

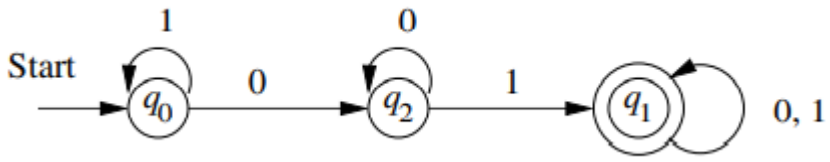
Finite State Automata adalah suatu model matematika dengan masukan dan keluaran diskrit yang dapat mengenali bahasa paling sederhana (bahasa regular) dan dapat diimplementasikan secara nyata. Finite State Automata atau FSA memiliki lima *tuple*, yaitu himpunan *state*, simbol *input*, fungsi transisi, *start state*, dan *final state*. Dapat dituliskan dengan format:

$$A = (Q, \Sigma, \delta, q_0, F)$$

Dengan A adalah nama dari FSA, Q adalah himpunan *state*, $\sum$ adalah himpunan simbol input, δ adalah fungsi transisi, q_0 adalah *start state*, dan *F* adalah *final state*.

Terdapat dua jenis Finite State Automata, yaitu Deterministic Finite Automata (DFA) dan Non-Deterministic Finite Automata (NFA). Pada tugas besar ini, jenis Finite State Automata yang digunakan adalah Deterministic Finite Automata atau DFA. DFA adalah jenis Finite State Automata yang input karakternya hanya dapat mengarah ke satu state saja dan fungsi transisi digunakan pada setiap state untuk setiap simbol input dan DFA tidak dapat berpindah state dengan karakter null (ϵ).

Berikut contoh DFA yang menerima 01 sebagai *substring*:



Pada contoh diatas, q_0 merupakan *start state* (terdapat panah bertulisan start mengarah ke *state*-nya) dengan himpunan simbol input {0, 1}, himpunan *state* {q_0, q_1, q_2} dan *final state* adalah q_1 (*final state* digambarkan dengan lingkaran dua lapis). Dapat dilihat bahwa setiap inputnya hanya memiliki satu tujuan dan tidak terdapat transisi null. DFA tersebut juga dapat dituliskan dalam bentuk tabel seperti berikut:

	0	1
$\rightarrow q_0$	q_2	q_0
$*q_1$	q_1	q_1
q_2	q_2	q_1

Tabel DFA tersebut ekuivalen dengan DFA bentuk graf sebelumnya dengan *start state* adalah state yang memiliki panah mengarah ke dirinya, *final state* yang memiliki tanda bintang dan kolom yang merupakan *state* tujuan dari *state* yang berada pada kolom kiri dengan input pada bagian baris pertama (0 atau 1).

BAB II : Hasil FA dan CFG

Tujuan dari implementasi CFG adalah untuk membuat keluaran yang dapat dijalankan pada algoritma CYK untuk menilai kebenaran kode sintaktik dalam bahasa pemrograman Python. Keluaran ini selanjutnya disederhanakan menjadi Chomsky-Normal Form (CNF).

1. Terminals

+	-	*	/	%	var	variable	number	!	=
>	<	(	)	{	}	true	false	string	‘
“	&	if	else	:	while	for	in	from	return
,	break	continue	null	function	throw	let	finally	switch	case
default	delete	.	|	try	let	catch			

2. Variables Symbols

S	OPERATOR	VAR	VARCLASS	VAL	STRING	VV	INBRACKET	VARDEC	DELETEVAL
FINALLY	FUNCTION	FUNCDEC	VARVALFUNC	LET	NULL	THROW	RETURN	VAL	BREAK
CONTINUE	SWITCH	CASE	DEFAULT	CONST	VALARRAY	ARRAY	BOOLEAN	TRY	CATCH
RELATION	COMP	COMPORPT	OR	AND	INCDEC	WHILE	FOR	FORINC	IF
ELIF	ELSE	SLOOP	IFLOOP	ELIFLOOP	ELSELOOP	SWITCHLOOP	CASELOOP	DEFAULTLOOP	SFUNC
IFFUNC	ELIFFUNC	ELSEFUNC	SWITCHFUNC	CASEFUNC	DEAULTFUNC	FORFUNC	SLF	SWITCHLF	CASELF
DAFULTLF									

3. Productions

Production	Hasil
S	S S | VAR = VV | VAR + = VV | VAR - = VV | VAR * = VV | VAR / = VV | IF | PRINT | WHILE | FOR | FUNCTION | VARCLASS | COMMENT | ARRAY | SWITCH | DELETE | VARDEC | LET | TRY | FUNCDEC | VAR INCDEC
OPERATOR	+ | - | * | / | %
VAR	variable | VARCLASS
VARCLASS	VARCLASS . VARCLASS | VARCLASS . FUNCTION | variable | FUNCTION
VAL	number | - number | VV OPERATOR VV | VV / / VV | VV * * VV | (VV) | BOOLEAN | (VAL) | VAR | STRING | NUMBER | ARRAY | BOOLEAN
STRING	STRING + STRING | ' string ' | " string " | (STRING)
VV	VAR | VAL | VV OPERATOR VV | VV , VV | VV OPERATOR STRING

DELETE	delete DELETEVAL
DELETEVAL	VAL | VAL . VAL
FINALLY	finally { S }
FUNCTION	VAR (VV) | FUNCTION (FUNCTION) | VAR ( )
FUNCDEC	function FUNCTION { SFUNC RETURN } | function FUNCTION { SFUNC }
VARVALFUNC	VARVALFUNC , VARVALFUNC | VAL | VAR = VAL
VARDEC	var VARVALFUNC
LET	let VARVALFUNC
NULL	null
THROW	throw BOOLEAN | throw VAL | throw STRING
RETURN	return VAR | return BOOLEAN | return VAL | return NULL | return STRING | return VV | return;
BREAK	break
CONTINUE	continue
SWITCH	switch (VAR) { CASE }
CASE	CASE CASE | case VAL : SLOOP | DEFAULT
DEFAULT	default : S
CONST	const VAR = VAL | const { VAR } = VAR
VALARRAY	VAL | VALARRAY , VALARRAY | ARRAY
ARRAY	[ ] | [VALARRAY] | ( ) | (VALARRAY)
BOOLEAN	true | false
TRY	try { S } CATCH
CATCH	catch (VAR) { S } | catch (VAR) { S } FINALLY
RELATION	> | < | = = | ! = | < = | > = | = = = | ! = =
COMP	BOOLEAN | VAR | COMPOPRT
COMPOPRT	COMPOPRT OPERATOR COMPOPRT | COMPOPRT RELATION COMPOPRT | VAL | VAL OPERATOR VAL | VAL * * VAL | VAL / / VAL | VAL > > VAL | VAL < < VAL | VAL RELATION VAL | COMPOPRT OR COMPOPRT | COMPOPRT AND COMPOPRT | (COMPOPRT)
OR	||
AND	&&
INCDEC	+ + | - -
WHILE	while (COMP) { SLOOP }
FOR	for (FORINC1 FORINC2 FORINC3) { SLOOP } | for (const VAR in ARRAY) { SLOOP } | for (const VAR of ARRAY) { SLOOP }
FORINC1	VAL = VAL | LET
FORINC2	VAL RELATION VAL
FORINC3	VAL INCDEC | VAL INCDEC
IF	if (COMP) { S } | IF ELIF | IF ELSE
ELIF	else if (COMP) { S } | ELIF ELIF | ELIF ELSE;
ELSE	else { S };

SLOOP	SLOOP SLOOP | VV | VAR = VV | VAR + = VV | VAR - = VV | VAR * = VV | VAR / = VV | IFLOOP | WHILE | FOR | FUNCTION | VARCLASS | COMMENT | ARRAY | SWITCHLOOP | DELETE | VARDEC | LET | TRY | VAR INCDEC | BREAK | CONTINUE | PASS
IFLOOP	if (COMP) { SLOOP } | IFLOOP ELIFLOOP | IFLOOP ELSELOOP
ELIFLOOP	else if (COMP) { SLOOP } | ELIFLOOP ELIFLOOP | ELIFLOOP ELSELOOP
ELSELOOP	else { SLOOP }
SWITCHLOOP	switch (VAR) { CASELOOP }
CASELOOP	CASELOOP CASELOOP | case VAL : SLOOP | DEFAULTLOOP
DEFAULTLOOP	default : SLOOP
SFUNC	SFUNC SFUNC | THROW | VAR = VV | VAR + = VV | VAR - = VV | VAR * = VV | VAR / = VV | IFFUNC | WHILE | FORFUNC | FUNCTION | VAR INCDEC | VARCLASS | COMMENT | ARRAY | VAR INCDEC | SWITCHFUNC | DELETE | VARDEC | LET | TRY | RETURN | VARVALFUNC
IFFUNC	if (COMP) { SFUNC } | IFFUNC ELIFFUNC | IFFUNC ELSEFUNC | IFFUNC RETURN
ELIFFUNC	else if (COMP) { SFUNC } | ELIFFUNC ELIFFUNC | ELIFFUNC ELSEFUNC
ELSEFUNC	ELSEFUNC -> else { SFUNC }
SWITCHFUNC	SWITCHFUNC -> switch (VAR) { CASEFUNC }
CASEFUNC	CASEFUNC -> CASEFUNC CASEFUNC | case VAL : SFUNC | DEFAULT
DEFAULTFUNC	DEFAULTFUNC -> default : SFUNC
FORFUNC	for (FORINC1 FORINC2 FORINC3) { SLF } | for (const VAR in ARRAY) { SLF } | for (const VAR of ARRAY) { SLF }
SLF	SLF SLF | THROW | VAR = VV | VAR + = VV | VAR - = VV | VAR * = VV | VAR / = VV | IFFUNC | IFLOOP | WHILE | FORFUNC | FUNCTION | VARCLASS | COMMENT | ARRAY | SWITCHFUNC | DELETE | VARDEC | LET | TRY | RETURN | VAR INCDEC | CONTINUE | PASS
SWITCHSLF	switch (VAR) { CASELF }
CASESLF	CASELF CASELF | case VAL : SFUNC | case VAL : SLOOP | DEFAULTFUNC
DEFAULTSLF	DEFAULTFUNC | DEFAULTLOOP

BAB III : Implementasi dan Pengujian

1. Implementasi

Sebelum memulai program, diperlukan file cfg.txt yang memiliki aturan cfg dari bahasa pemrograman Node.js. aturan tersebut kemudian akan dikonversi menjadi cnf menggunakan converter.py.

Converter.py akan menerima file cfg yang tersedia dan membentuknya menjadi cnf dan diletakkan pada file cnf.txt. Fungsi-fungsi pada converter.py adalah sebagai berikut:

Nama fungsi	Deskripsi
unitary	Memastikan apakah rule memiliki produksi satu variable
simple	Memastikan apakah rule memiliki produksi satu terminal
start	Menambahkan symbol S0 sebagai start symbol
term	Menghapus produksi simbol, terminal, dan variable yang tergabung
bin	Menghapus produksi yang menghasilkan lebih dari 2 variabel
delete	Menghapus produksi yang tidak menghasilkan simbol terminal
changeUnitary	Memastikan produksi merupakan unitary dan bisa diganti
unit	Menghapus unitary production dari suatu produksi

Fungsi-fungsi tersebut juga menggunakan beberapa fungsi tambahan yang terletak pada file supportFunc.py. Fungsi-fungsi pada supportFunc.py adalah sebagai berikut:

Nama fungsi	deskripsi
unionOfList	Menggabungkan dua list tanpa duplikat
pathLoader	Mengelompokkan pembacaan file cfg menjadi terminal, variable, dan productions
prodCleanser	Membersihkan ekspresi grammar dari format cfg dan dikelompokkan menjadi rules dan hasil produksi
alphabetCleanser	Memisahkan rules
removeTargetProd	Menghapus produksi non terminal

makeDict	Menyimpan aturan produksi sebagai key dan value
rewrite	Mengolah ulang aturan produksi yang ada dan menghapus berdasarkan target
convertDictToArr	Mengubah aturan (dictionary) menjadi list
convertForm	Mengubah bentuk aturan produksi menjadi string untuk dituliskan pada file output

Setelah terbentuk cnf, aturan tersebut digunakan pada file mainCYK.py yang kemudian diproses sehingga dapat menentukan apakah suatu kode valid atau tidak. Sebelum algoritma cyk dimulai, dilakukan inisiasi data struktur sehingga memiliki atribut cnfPath (path menuju file cnf), testFile (path menuju file test), chomskyGrammar (grammar yang diperoleh dari cnf.txt), validString (kondisi valid atau tidak string pada file), inputText (hasil input yang dibentuk menjadi list), cykTable, contents (hasil pembacaan file test), err_line (menunjukkan line dimana terdapat error penulisan string). Setelah diinisiasi, algoritma cyk dapat dimulai. Fungsi-fungsi pada mainCYK.py adalah sebagai berikut:

Nama fungsi	**deskripsi**
loadGrammar	Membaca file cnf dari path yang sudah diinisialisasi dan diolah untuk dimasukkan ke dalam chomskyGrammar
string_analyzer	Menganalisis string dan memvalidasikannya
readInputFile	Melakukan formatting pada setiap ekspresi yang ada sehingga membentuk simbol terminal yang valid dan membersihkan string kosong
insertTable	Memasukkan ekspresi ke dalam table cyk
makeCYKTable	Melakukan validasi variable, angka, dan string dan semua ekspresi lainnya dan diletakkan pada cyk table.
result	Menunjukkan hasil apakah kode valid atau tidak
validate	Memanggil setiap fungsi berdasarkan kondisi yang memenuhi

mainCYK.py secara garis besar akan menentukan apakah kode valid atau tidak. Akan tetapi, diperlukan juga fungsi untuk memvalidasi penulisan variabel yaitu dengan Finite Automata. File fa.py memiliki algoritma dalam menentukan apakah penulisan sebuah string sesuai dengan aturan atau tidak menggunakan prinsip Deterministic Finite Automata atau

DFA. Sebelum dilakukan pemastian, string akan diinisialisasi agar memiliki atribut yaitu currentState bertanda ‘start’ yang menunjukkan start state, accept bertanda ‘final’ yang menunjukkan final state, dan string yang merupakan string itu sendiri. Kemudian string yang telah diperoleh diproses pada fungsi readVar yang akan memastikan setiap input dan mengarahkan state dari string tersebut berdasarkan karakter yang terdapat pada string. Secara garis besar, di awal string akan berada pada start state dan ketika terdapat karakter yang tidak boleh digunakan untuk mendeklarasi string, state dari string akan menuju ‘dead’ yang menunjukkan bahwa string tersebut tidak valid, jika tidak maka string akan menuju ‘final’ sehingga string tersebut valid.

2. Uji Coba

Uji coba 1

Syntax yang diuji : Delete, else, false, function, if, null, return, true

File : test.txt

```
function do_something(x) {
    if (x == 0) {
      delete P.Xall;
      return 0;
    } else if (x + 4 == 1) {
      if (true) {
        return null;
      } else {
        return false;
      }
    } else if (x == 32) {
      return 4;
    } else if (x == 34) {
      throw 42;
    } else {
      return "Momen";
    }
  }
```

```
PS C:\Users\ASUS\OneDrive\Documents\GitHub\Tubes-TBFO-Parser-Node.js> python cyk.py test.txt
Accepted
```

Percobaan ini menguji syntax delete, else, false, function, if, null, return, dan true. Keluaran program sesuai dengan harapan karena syntax yang diujikan sudah sesuai.

```
function do_something(x) {
    if (x == 0) {
      delete P.Xall;
      return 0;
    } else if (x + 4 = 1) {
      if (true) {
        return null;
      } else {
        return false;
      }
    } else if (x == 32) {
      return 4;
    } else if (x == 34) {
      throw 42;
    } else {
      return "Momen";
    }
  }
```

```
PS C:\Users\ASUS\OneDrive\Documents\GitHub\Tubes-TBFO-Parser-Node.js> python cyk.py test.txt
Syntax error!
```

Percobaan ini seperti percobaan sebelumnya, namun diubah sedemikian sehingga syntax menjadi salah. Keluaran program sesuai dengan harapan karena syntax yang diuji salah. Kesalahan syntax berada di line 5, dimana pernyataan *condition* else if tidak sesuai dengan yang diharapkan.

Uji coba 2

Syntax yang diuji : let, const, var, while, continue, break, if, else

File : test1.txt

```
let x = 10;
const p = 2;
var y = 3;

while (x>=1) {
    x = x+1;
    if (x == 3) {
        continue;
    }
    if (y != 8) {
        y = y+p;
    } else {
        break;
    }
}
```

```
PS C:\Users\ASUS\OneDrive\Documents\GitHub\Tubes-TBFO-Parser-Node.js> python cyk.py test1.txt
Accepted
```

Percobaan ini menguji syntax let, const, var, while, continue, break, if, dan else. Keluaran program sesuai dengan harapan karena syntax yang diujikan sudah sesuai.

```
let x = 10;
const p = 2;
var y = 3;

while (x=1) {
    x = x+1;
    if (x == 3) {
        continue;
    }
    if (y != 8) {
        y = y+p;
    } else {
        break;
    }
}
```

```
PS C:\Users\ASUS\OneDrive\Documents\GitHub\Tubes-TBFO-Parser-Node.js> python cyk.py test1.txt
Syntax error!
```

Percobaan ini seperti percobaan sebelumnya, namun diubah sedemikian sehingga syntax menjadi salah. Keluaran program sesuai dengan harapan karena syntax yang diuji salah. Kesalahan syntax berada di line 5, dimana pernyataan *condition* while tidak sesuai dengan yang diharapkan.

Uji coba 3

Syntax yang diuji : try, catch, finally

File : test2.txt

```
try {
    x = x+1;
} catch (p) {
    x = x-3;
} finally {
    x = x * x;
}
```

```
PS C:\Users\ASUS\OneDrive\Documents\GitHub\Tubes-TBFO-Parser-Node.js> python cyk.py test2.txt
Accepted
```

Percobaan ini menguji syntax try, catch dan finally. Keluaran program sesuai dengan harapan karena syntax yang diujikan sudah sesuai.

```
try {
    x = x+1;
} catch () {
    x = x-3;
} finally {
    x = x * x;
}
```

```
PS C:\Users\ASUS\OneDrive\Documents\GitHub\Tubes-TBFO-Parser-Node.js> python cyk.py test2.txt
Syntax error!
```

```
try {
    x = x+1;
} finally {
    x = x * x;
}
```

```
PS C:\Users\ASUS\OneDrive\Documents\GitHub\Tubes-TBFO-Parser-Node.js> python cyk.py test2.txt
Syntax error!
```

Kedua percobaan diatas menguji syntax try dan catch. Keluaran program berupa *"Syntax error!"* sesuai dengan harapan, dimana program mendeteksi adanya kesalahan syntax. Pada percobaan pertama, terjadi kesalahan syntax dimana parameter untuk catch tidak ada, sedangkan pada percobaan kedua, terjadi kesalahan syntax dimana try langsung dilanjuti dengan finally.

Uji coba 4

Syntax yang diuji : case, default, break

File : test3.txt

```
switch (x) {
    case 1:
        x = x+1;
    case 3:
        x = x-3;
    case 4:
        break;
    default:
        x = x;
}
```

```
PS C:\Users\ASUS\OneDrive\Documents\GitHub\Tubes-TBFO-Parser-Node.js> python cyk.py test3.txt
Accepted
```

Percobaan ini menguji syntax case, default, dan break. Keluaran program sesuai dengan harapan karena syntax yang diujikan sudah sesuai. Dapat diamati bahwa expression yang dapat digunakan pada case ataupun default dapat berupa ekspresi biasa maupun break.

```
switch () {
    case 1:
        x = x+1;
    case 3:
        x = x-3;
    case 4:
        break;
    default:
        x = x;
}
```

```
PS C:\Users\ASUS\OneDrive\Documents\GitHub\Tubes-TBFO-Parser-Node.js> python cyk.py test3.txt
Syntax error!
```

Percobaan ini seperti percobaan sebelumnya, namun diubah sedemikian sehingga syntax menjadi salah. Keluaran program sesuai dengan harapan karena syntax yang diuji salah. Kesalahan syntax berada di line 1, dimana tidak ada masukan pada pernyataan *condition* switch.

Uji coba 5

Syntax yang diuji : for, break, continue, if, else

File : test4.txt

```
for (i=0; i<5; i++) {
    p = p+4;
    if (p == 3) {
        continue;
    } else if (p == 7) {
        break;
    }
}
```

```
PS C:\Users\ASUS\OneDrive\Documents\GitHub\Tubes-TBFO-Parser-Node.js> python cyk.py test4.txt
Accepted
```

Percobaan ini menguji syntax for, break, continue, if, dan else. Keluaran program sesuai dengan harapan karena syntax yang diujikan sudah sesuai. Dapat diamati pada gambar bahwa dapat dimasukkan ekspresi break dan continue dalam for loop, sesuai dengan kegunaan break dan continue.

LAMPIRAN

Repository github

https://github.com/HoseaNA/Tubes-TBFO-Parser-Node.js

Pembagian Tugas

Nama	NIM	Tugas
Hosea Nathanael Abetnego	13521057	Context Free Grammar, Laporan, cyk, converter, helper, dan FA
Alex Sander	13521061	Context Free Grammar, Laporan, cyk, converter, helper, dan FA
Ariel Jovananda	13521086	Context Free Grammar, Laporan, cyk, converter, helper, dan FA

DAFTAR PUSTAKA

1. https://en.wikipedia.org/wiki/CYK_algorithm Diakses pada 16 November 2022
2. https://www.geeksforgeeks.org/cocke-younger-kasami-cyk-algorithm/ Diakses pada 16 November 2022