

**LAPORAN TUGAS BESAR**  
**IF2124 TEORI BAHASA FORMAL DAN AUTOMATA**

**PARSER JAVASCRIPT (NODE.JS)**



**DISUSUN OLEH :**  
**KELOMPOK 三位棉兰男神**  
**13521046 Jeffrey Chow**  
**13521054 Wilson Tansil**  
**13521102 Jimly Firdaus**

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

# Daftar Isi

<b>BAB I</b> .....	3
<b>BAB II</b> .....	4
<b>2.1 Finite State Automata (FSA)</b> .....	4
<b>2.1 Context Free Grammar (CFG)</b> .....	5
<b>BAB III</b> .....	7
<b>3.1 Finite State Automata</b> .....	7
<b>3.2 Context Free Grammar</b> .....	8
<b>BAB IV</b> .....	10
<b>4.1 Module file lexer.py</b> .....	10
<b>4.2 Module file FA.py</b> .....	10
<b>4.3 Module file CYK.py</b> .....	10
<b>4.4 Module file CFG.py</b> .....	10
<b>4.5 File main.py</b> .....	11
<b>BAB V</b> .....	12
<b>BAB VI</b> .....	16
<b>BAB VII</b> .....	16
<b>BAB VIII</b> .....	16

# **BAB I**

## **Latar Belakang**

Dalam proses pembuatan program dari sebuah bahasa menjadi instruksi yang dapat dieksekusi oleh mesin, terdapat pemeriksaan sintaks bahasa atau parsing yang dibuat oleh programmer untuk memastikan program dapat dieksekusi tanpa menghasilkan error. Parsing ini bertujuan untuk memastikan instruksi yang dibuat oleh programmer mengikuti aturan yang sudah ditentukan oleh bahasa tersebut. Baik bahasa berjenis interpreter maupun compiler, keduanya pasti melakukan pemeriksaan sintaks. Perbedaannya terletak pada apa yang dilakukan setelah proses pemeriksaan (kompilasi/compile) tersebut selesai dilakukan.

Dibutuhkan grammar bahasa dan algoritma parser untuk melakukan parsing. Sudah sangat banyak grammar dan algoritma yang dikembangkan untuk menghasilkan compiler dengan performa yang tinggi. Terdapat CFG, CNF-e, CNF+e, 2NF, 2LF, dll untuk grammar yang dapat digunakan, dan terdapat LL(0), LL(1), CYK, Earley's Algorithm, LALR, GLR, Shift-reduce, SLR, LR(1), dll untuk algoritma yang dapat digunakan untuk melakukan parsing.

Pada program parsing ini, kami akan menggunakan grammar CFG yang akan di convert menjadi CNF serta algoritma CYK untuk melakukan parsing.

## **BAB II**

### **Teori Dasar**

#### **2.1 Finite State Automata (FSA)**

Finite State Automata (FSA) berupa sistem model matematika dengan masukan dan keluaran diskrit yang dapat mengenali bahasa paling sederhana (bahasa reguler) dan dapat diimplementasikan secara nyata.

FSA dapat menerima input dan mengeluarkan output yang memiliki state yang berhingga banyaknya. Selain itu, FSA memiliki sekumpulan aturan untuk berpindah dari satu state ke state lain berdasarkan input dan fungsi transisi yang diterapkan.

Finite State Automata pada dasarnya digunakan untuk mengenali pola. Dibutuhkan string simbol sebagai input dan statusnya berubah sesuai dengan input tersebut. Ketika ditemukan simbol input yang diinginkan, maka terjadi transisi.

Pada saat transisi, automata dapat berpindah ke keadaan berikutnya atau tetap dalam keadaan yang sama. Finite State Automata memiliki dua status, status Terima atau status Tolak. Ketika string input berhasil diproses, dan automata mencapai state akhir, maka statusnya adalah Terima, demikian juga sebaliknya.

Finite State Automata dapat didefinisikan dengan persamaan berikut:

$$M = (Q, \Sigma, \delta, S, F)$$

$Q$  = himpunan state

$\Sigma$  = himpunan simbol input

$\delta$  = fungsi transisi  $\delta : Q \times \Sigma$

$S$  = state awal / initial state,  $S \in Q$

$F$  = state akhir,  $F \subseteq Q$

Finite State Automata memiliki beberapa karakteristik berikut:

- Setiap Finite Automata memiliki keadaan dan transisi yang terbatas.

- Transisi dari satu keadaan ke keadaan lainnya dapat bersifat deterministik atau non-deterministik.
- Setiap Finite Automata selalu memiliki keadaan awal.
- Finite Automata dapat memiliki lebih dari satu keadaan akhir.
- Jika setelah pemrosesan seluruh string, keadaan akhir dicapai, artinya otomata menerima string tersebut.

## 2.1 Context Free Grammar (CFG)

Context Free Grammar (CFG) / Bahasa Bebas Konteks adalah sebuah tata bahasa dimana tidak terdapat pembatasan pada hasil produksinya, Contoh Pada aturan produksi :

$$\alpha \rightarrow \beta$$

batasannya hanyalah ruas kiri ( $\alpha$ ) adalah sebuah simbol variabel. Sedangkan contoh aturan produksi yang termasuk CFG adalah seperti di bawah :

$$B \rightarrow CDeFg$$

$$D \rightarrow BcDe$$

Context Free Grammar (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.

Context Free Grammar (CFG) menjadi dasar dalam pembentukan suatu parser/proses analisis sintaksis. Bagian sintaks dalam suatu kompilator kebanyakan di definisikan dalam tata bahasa bebas konteks. Pohon penurunan (derivation tree/parse tree) berguna untuk menggambarkan simbol-simbol variabel menjadi simbol-simbol terminal setiap simbol variabel akan di turunkan menjadi terminal sampai tidak ada yang belum tergantikan.

Proses penurunan / parsing bisa dilakukan dengan cara sebagai berikut :

- Penurunan terkiri (leftmost derivation): simbol variabel terkiri yang di perluas terlebih dahulu.
- Penurunan terkanan (rightmost derivation) : simbol variabel terkanan yang diperluas terlebih dahulu.

## 2.3 Syntax JavaScript dalam pembuatan CFG dan FA

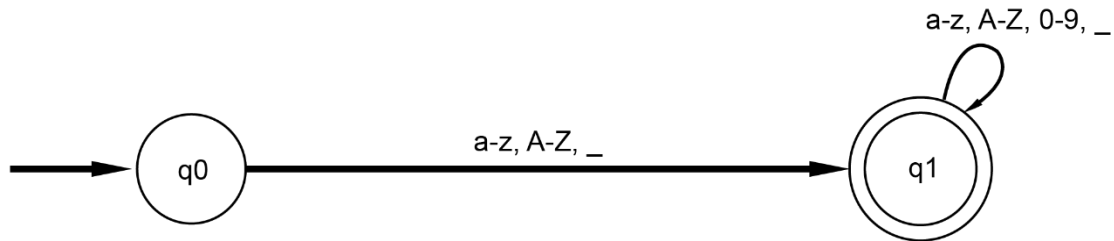
while	break	default	for	return	var
const	delete	function	switch	while	case
else	if	throw	catch	false	let
try	continue	finally	null	true	import
export	else if	export default	in	from	instanceof
as					

Pada syntax Javascript ini, titik koma merupakan hal yang wajib dalam membatasi sebuah command program. Hal ini diimplementasikan agar program Javascript dapat memiliki pembatasan yang signifikan guna membantu dalam hal debugging dan penulisan program. Keterbatasan grammar juga terletak pada penggunaan atau pemanggilan struct, fungsi, atau library pada sebuah operasi aritmatika, misalnya (var1/var2).toFixed(x). Setiap elemen yang berada di dalam { } dan [ ] dianggap sebagai list dan array sehingga syntax tersebut haruslah {x1, x2, x3, ..., xn} atau [x1, x2, x3, ..., xn]. Dalam hal ini, x adalah kumpulan kemungkinan yang dapat diinput ke dalam baik merupakan sebuah expression maupun variabel. Pada tahap pengisian kurung { } dan [ ], elemen paling akhir tidak dapat diakhiri dengan koma. Contohnya program akan menyatakan {x1, x2, x3,} salah dikarenakan terdapat koma pada elemen akhir, sebaliknya program akan menyatakan {x1, x2, x3} benar dikarenakan tidak terdapat koma pada elemen terakhir. Syntax import harus selalu terletak pada bagian atas, sebagai header.

# BAB III

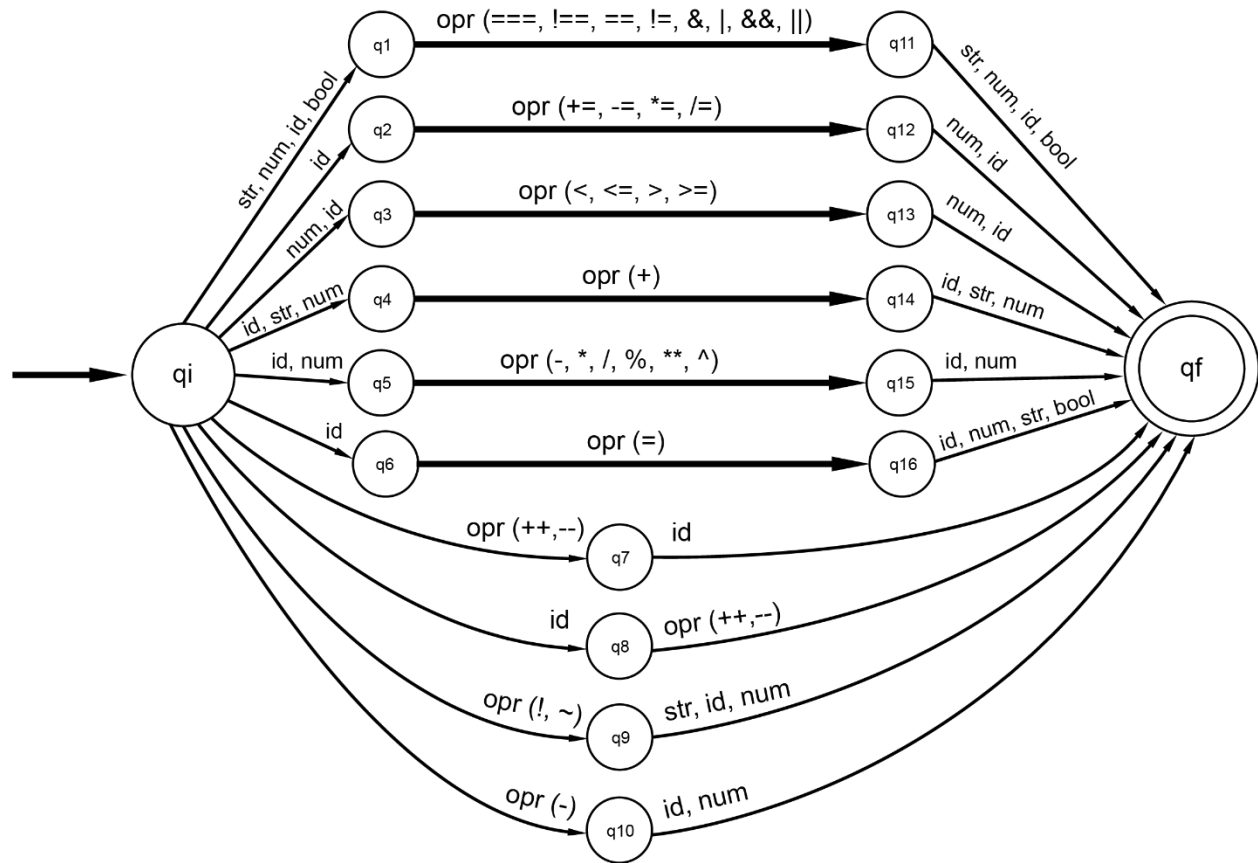
## Finite Automata dan Context Free Grammar

### 3.1 Finite State Automata



Gambar 3.1.1 DFA untuk pengecekan variabel

Program dapat melakukan pengecekan variabel dengan prinsip Finite State Automata. Contoh variabel yang tidak diterima adalah 123cyk, 456FA\_, atau 23\_abc. Contoh variabel yang diterima adalah TBFO\_Mantap, tubesTbfo, atau pembuatProgramIniGanteng. DFA ini diimplementasi pada file FA.py modul checkVar.



Gambar 3.1.2 NFA untuk pengecekan ekspresi

Program dapat melakukan pengecekan ekspresi dengan prinsip Finite State Automata. Contoh ekspresi yang tidak diterima adalah  $a = , b = 2 +$ , atau `"tbfo"*2`. Contoh ekspresi yang diterima adalah `nama = "teori" + "bahasa"`, `a < b`, atau `i++`. NFA ini diimplementasi pada file `FA.py` modul `checkExpr`.

### 3.2 Context Free Grammar

$$G = (V, T, P, S)$$

#### Non Terminal Symbol (V)

S	BODY	IMPORT_STMT	EXPORT_STMT	EXPORT_DEFAULT_STMT	PART
VALUE_BOOLEAN	EXPR_BOOLEAN	VARIOUS_EXPR_BOOLEAN	LOGOP	ISSTRICTEQ	NOTSTRICTEQ
ISEQ	ISNEQ	LEQ	L	GEQ	G
LOGOPB	AND	OR	NOTB	IF_STMT	ELSE_STMT
ELSEIF_STMT	IF	ELSE	WHILE_STMT	FUNCTION_STMT	RETURN_STMT
FOR_STMT	DELETE_STMT	THROW_STMT	THROW	SWITCH_STMT	SWITCH_BODY
SWITCH_PARAMETER	CASE_STMT	CASE_BODY	DEFAULT_STMT	SWITCH	CASE
DELETE	FOR	FOR_PARAMETER	TYPE_FOR	IN	OF
INSTANCEOF	RETURN	CONTINUE_STMT	CONTINUE	BREAK_STMT	BREAK
WHILE	TRY_STMT	CATCH_PARAMETER	TRY	CATCH	FINALLY
TITIK_STMT	TITIK_VALUE	ARRAY_CALL	FUNCTION	FUNCTION_CALL	FUNCTION_PARAMETER
VARIOUS_VARIABLE_FUNCTION	TYPE_STMT	BODY_OBJECT	BODY_ARRAY	VARIOUS_VARIABLE_BODY	VALUE
ASSIGN_VARIABLE	ASSIGNMENT	NULL	TD	PMEQ_EXPR	PMEQ_VALUE
PMEQ	PEQ	MEQ	MULEQ	DIVEQ	INCDEC
INC	DEC	EXPR_ARIT	EXPR_ARIT_VALUE	ARIT_OPERATOR	POW
XOR	ANDB	ORB	SUM	SUB	DIV
MOD	HEAD_OBJECT	VARIOUS_VARIABLE	HEAD_VALUE	DEFAULT	EXPORT
IMPORT	KBKI	KBKA	KKKA	KKKI	MUL
FROM	STRING	AS	ID	TK	KM
LET	VAR	CONST	BOOL	TRUE	FALSE
NUMBER	IS	KSKI	KSKA	TITIK	TENARY_STMT

#### Terminal Symbol (T)

isstricteq	notstricteq	iseq	isneq	leq	l
geq	g	and	or	notb	if
else	return_stmt	throw	default_stmt	switch	case
delete	for	in	of	instanceof	return



continue_stmt	continue	break_stmt	break	while	catch_param
try	catch	finally	function	function_param	null
pmeq_expr	peq	meq	muleq	diveq	inc
dec	pow	xor	andb	orb	sum
sub	div	mod	head_object	default	export
import	kbki	kbka	kkka	kkki	mul
from	string	as	id	tk	km
let	var	const	true	false	number
is	kski	kska	titik		

### Production (P)

Terdapat 337 baris production yang kami implementasikan pada CFG.

### Start Symbol (S)

Pada CFG, start symbol yang kami gunakan adalah **S**. Syntax Javascript akan dianggap benar apabila grammar bisa dicapai dari **S**.

### Chomsky Normal Form (CNF)

Pada program ini, Context Free Grammar (CFG) yang telah dibuat akan diubah ke dalam bentuk Chomsky Normal Form (CNF) yang menjadi prasyarat penggunaan algoritma Cocke-Younger-Kasami dalam pengecekan syntax. Program pengubahan CFG ke CNF diimplementasikan dalam file `CFG.py` modul `CFGtoCNF`.

## BAB IV

### Implementasi

#### 4.1 Module file lexer.py

No	Fungsi/Prosedur	Deskripsi
1	lexer	Mem- <i>parsing</i> semua <i>code</i> JavaScript ke dalam token-token yang sudah di- <i>define</i> agar bisa dibaca dan diproses dengan CFG yang dibuat.
2	parseToToken	Mem- <i>parse</i> setiap <i>code by code</i> JavaScript ke dalam token-token (terminal) untuk di- <i>compare</i> dengan <i>grammar</i> CFG yang dibuat.

#### 4.2 Module file FA.py

Modul ini mengimplementasikan Finite Automata untuk melakukan pengecekan variabel dan ekspresi pada *syntax javascript*.

No	Fungsi/Prosedur	Deskripsi
1	checkVar	Melakukan pengecekan nama variabel agar sesuai dengan kaidah penulisan variabel. Mengirimkan <i>True</i> jika penamaan variabel benar, <i>vice versa</i> .
2	checkExpr	Melakukan pengecekan ekspresi pada <i>syntax javascript</i> . Mengirimkan <i>True</i> jika ekspresi benar dan mengirimkan indeks token ekspresi yang salah jika ekspresi salah.

#### 4.3 Module file CYK.py

No	Fungsi/Prosedur	Deskripsi
1	cykParse	Melakukan parsing token <i>syntax javascript</i> terhadap CNF yang telah dibuat. Bernilai <i>True</i> jika token bisa dicapai dari start symbol.

#### 4.4 Module file CFG.py

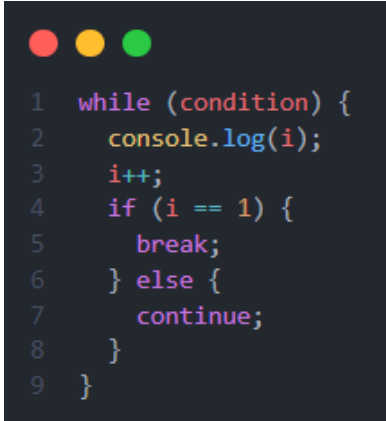

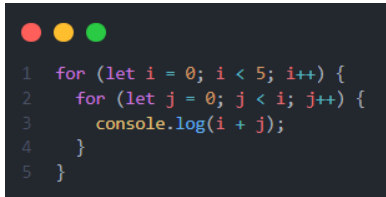
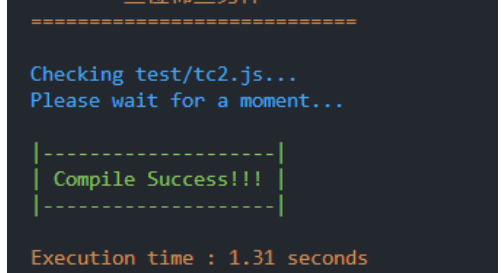
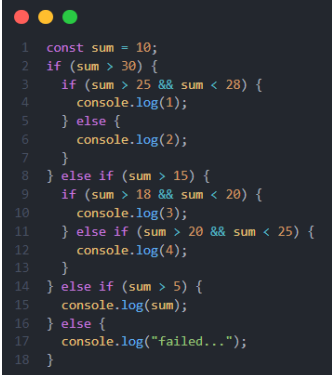
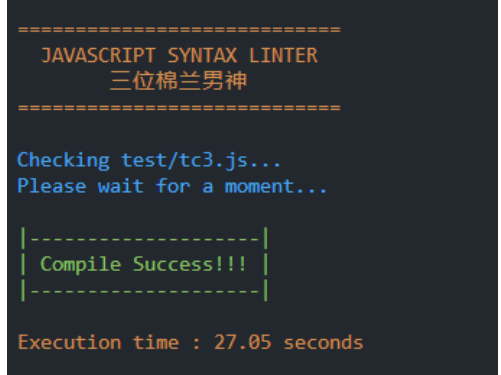
Modul ini diimplementasikan dalam bentuk *Object Oriented Programming* (OOP) untuk memudahkan pengubahan CFG. Penulisan dalam OOP juga membuat program lebih rapi dan terstruktur. *Class* CFG ini diawali dengan memanggil fungsi *parseCFG* lalu akan memanggil fungsi *CFGtoCNF* dan diakhiri dengan mengembalikan CFG yang sudah berbentuk CNF.

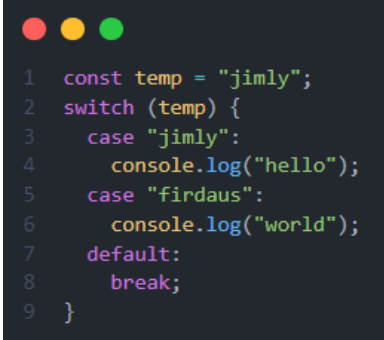
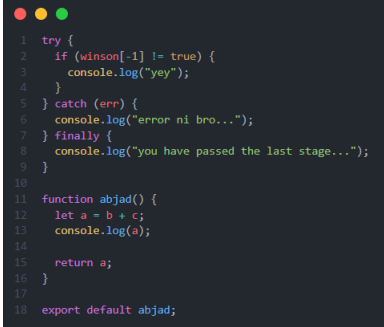

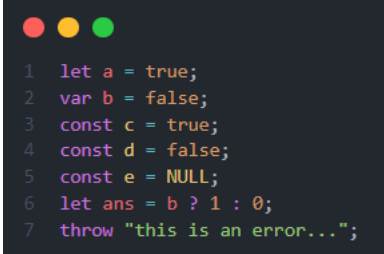
No	Fungsi/Prosedur	Deskripsi
1	updateKeyVal	Mengupdate Key dan Value dari dictionary CFG
2	createProduction	Membuat production baru untuk ditambahkan ke CFG
3	newVar	Membuat nama variabel baru yang digunakan dalam memproduksi rules CNF
4	CFGtoCNF	Mengubah CFG menjadi CNF
5	parseCFG	Melakukan parsing CFG dari textfile

#### 4.5 File main.py


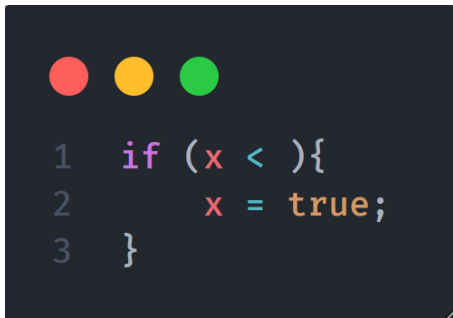
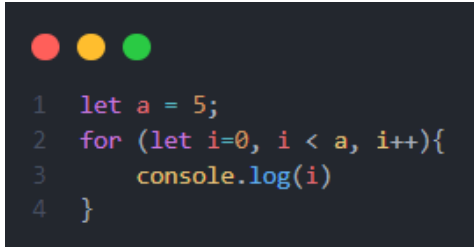
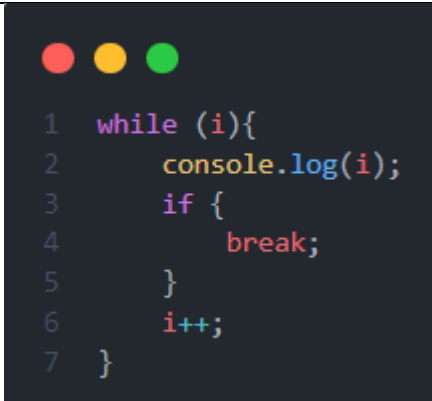
Program utama dari *parser javascript* ini diimplementasikan pada file `main.py`. File ini menggunakan modul-modul yang telah dibuat di berbagai file yang sudah dijelaskan di poin 4.1-4.4. Program dimulai dengan melakukan parsing terhadap argumen yang merupakan nama file javascript yang akan di cek. Selanjutnya program akan menampilkan splash screen program. Program akan melakukan parsing terhadap file `grammar.txt` terlebih dahulu sampai terbentuknya CNF. Lalu program akan melakukan parsing terhadap file javascript yang akan dicek sehingga berbentuk seperti token. Kemudian token dan CNF akan dimasukkan ke algoritma CYK dan akan mengembalikan nilai berupa boolean. Jika True maka akan ditampilkan pesan “Compile Success!!!”, dan jika False akan ditampilkan pesan “Compile Error...”. Jika error sudah ditemukan disaat parsing kode javascript menggunakan lexer dan FA, maka akan menampilkan pesan “Invalid Syntax”. Diakhir program akan ditampilkan waktu eksekusi pengecekan syntax.

## BAB V Eksperimen

Kata Kunci Javascript	Syntax	Result
While Loop	 <pre> 1  while (condition) { 2    console.log(i); 3    i++; 4    if (i == 1) { 5      break; 6    } else { 7      continue; 8    } 9  } </pre>	 <pre> ===== JAVASCRIPT SYNTAX LINTER 三位棉兰男神 =====  Checking test/tc1.js... Please wait for a moment...   -----    Compile Success!!!    -----   Execution time : 0.58 seconds </pre>
For Loop	 <pre> 1  for (let i = 0; i &lt; 5; i++) { 2    for (let j = 0; j &lt; i; j++) { 3      console.log(i + j); 4    } 5  } </pre>	 <pre> ===== JAVASCRIPT SYNTAX LINTER 三位棉兰男神 =====  Checking test/tc2.js... Please wait for a moment...   -----    Compile Success!!!    -----   Execution time : 1.31 seconds </pre>
Conditional	 <pre> 1  const sum = 10; 2  if (sum &gt; 30) { 3    if (sum &gt; 25 &amp;&amp; sum &lt; 28) { 4      console.log(1); 5    } else { 6      console.log(2); 7    } 8  } else if (sum &gt; 15) { 9    if (sum &gt; 18 &amp;&amp; sum &lt; 20) { 10     console.log(3); 11   } else if (sum &gt; 20 &amp;&amp; sum &lt; 25) { 12     console.log(4); 13   } 14 } else if (sum &gt; 5) { 15   console.log(sum); 16 } else { 17   console.log("failed..."); 18 } </pre>	 <pre> ===== JAVASCRIPT SYNTAX LINTER 三位棉兰男神 =====  Checking test/tc3.js... Please wait for a moment...   -----    Compile Success!!!    -----   Execution time : 27.05 seconds </pre>

Switch, Case, Default	 <pre> 1  const temp = "jimly"; 2  switch (temp) { 3      case "jimly": 4          console.log("hello"); 5      case "firdaus": 6          console.log("world"); 7      default: 8          break; 9  } </pre>	<pre> ===== JAVASCRIPT SYNTAX LINTER 三位棉兰男神 =====  Checking test/tc4.js... Please wait for a moment...   -----    Compile Success!!!    -----   Execution time : 0.84 seconds </pre>
Try, Catch, Finally, Export, Function, Default	 <pre> 1  try { 2      if (winson[-1] != true) { 3          console.log("yey"); 4      } 5  } catch (err) { 6      console.log("error ni bro..."); 7  } finally { 8      console.log("you have passed the last stage..."); 9  } 10 11 function abjad() { 12     let a = b + c; 13     console.log(a); 14 } 15 return a; 16 } 17 18 export default abjad; </pre>	<pre> ===== JAVASCRIPT SYNTAX LINTER 三位棉兰男神 =====  Checking test/tc5.js... Please wait for a moment...   -----    Compile Success!!!    -----   Execution time : 6.42 seconds </pre>
Array, Delete	 <pre> 1  arr = ["a", "b", "c"]; 2  delete arr[0]; 3 4  arr = ["e", "f", "g"]; 5  delete arr[2]; 6 </pre>	<pre> ===== JAVASCRIPT SYNTAX LINTER 三位棉兰男神 =====  Checking test/tc6.js... Please wait for a moment...   -----    Compile Success!!!    -----   Execution time : 0.64 seconds </pre>
Let, var, const, throw, ternary, true, false, null	 <pre> 1  let a = true; 2  var b = false; 3  const c = true; 4  const d = false; 5  const e = NULL; 6  let ans = b ? 1 : 0; 7  throw "this is an error..."; </pre>	<pre> ===== JAVASCRIPT SYNTAX LINTER 三位棉兰男神 =====  Checking test/tc8.js... Please wait for a moment...   -----    Compile Success!!!    -----   Execution time : 0.98 seconds </pre>

Simple program 1 (import)	<pre> import React from "React"; // program to check if a number is prime or not  // Take input from the user const number = parseInt(prompt("Enter a positive number: ")); let isPrime = true;  // check if number is equal to 1 if (number === 1) {   console.log("1 is neither prime nor composite number."); }  // check if number is greater than 1 else if (number &gt; 1) {   // looping through 2 to number-1   for (let i = 2; i &lt; number; i++) {     if (number % i == 0) {       isPrime = false;       break;     }   }    if (isPrime) {     console.log(number + " is a prime number.");   } else {     console.log(number + " is a not prime number.");   } }  // Check if number is less than 1 else {   console.log("The number is not a prime number."); } </pre>	<pre> ===== JAVASCRIPT SYNTAX LINTER 三位棉兰男神 =====  Checking test/tc9.js... Please wait for a moment...   -----    Compile Success!!!    -----   Execution time : 29.01 seconds </pre>
Simple program 2	<pre> 1 // program to find the factorial of a number 2 function factorial(x) { 3   // if number is 0 4   if (x == 0) { 5     return 1; 6   } 7 8   // if number is positive 9   else { 10    return x * factorial(x - 1); 11  } 12 } 13 14 // take input from the user 15 const num = "Enter a positive number: "; 16 17 // calling factorial() if num is positive 18 if (num &gt;= 0) { 19   const result = factorial(num); 20   console.log("The factorial of \${num} is" + result); 21 } else { 22   console.log("Enter a positive number."); 23 } </pre>	<pre> ===== JAVASCRIPT SYNTAX LINTER 三位棉兰男神 =====  Checking test/tc10.js... Please wait for a moment...   -----    Compile Success!!!    -----   Execution time : 7.14 seconds </pre>
inputAcc.js	<pre> 1 // inputAcc.js 2 function do_something(x) { 3   // This is a sample comment 4   if (x == 0) { 5     return 0; 6   } else if (x + 4 == 1) { 7     if (true) { 8       return 3; 9     } else { 10      return 2; 11    } 12  } else if (x == 32) { 13    return 4; 14  } else { 15    return "Momen"; 16  } 17 } </pre>	<pre> ===== JAVASCRIPT SYNTAX LINTER 三位棉兰男神 =====  Checking test/tc12.js... Please wait for a moment...   -----    Compile Success!!!    -----   Execution time : 4.25 seconds </pre>
InputReject.js	<pre> 1 // inputReject.js 2 function do_something(x) { 3   // This is a sample multiline comment 4   if (x == 0) { 5     return 0; 6   } else if x + 4 == 1 { 7     if (true) { 8       return 3; 9     } else { 10      return 2; 11    } 12  } else if (x == 32) { 13    return 4; 14  } else { 15    return "Momen"; 16  } 17 } </pre>	<pre> ===== JAVASCRIPT SYNTAX LINTER 三位棉兰男神 =====  Checking test/tc11.js... Please wait for a moment...   -----    Compile Error...    -----   Execution time : 3.91 seconds </pre>

syntaxFail.js	 <pre> 1  let `a = 3; </pre>	<pre> ===== JAVASCRIPT SYNTAX LINTER 三位棉兰男神 =====  Checking test/syntaxFail.js... Please wait for a moment...   -----    Syntax Error !!!    -----   &gt;&gt;&gt; Line 1 : "let `a = 3;". </pre>
exprFail.js	 <pre> 1  if (x &lt; ){ 2      x = true; 3  } </pre>	<pre> ===== JAVASCRIPT SYNTAX LINTER 三位棉兰男神 =====  Checking test/exprFail.js... Please wait for a moment...   -----    Invalid Expression !!!    -----   &gt;&gt;&gt; Line 1 : "if (x &lt; ){". </pre>
Simple Program 3 (failed)	 <pre> 1  let a = 5; 2  for (let i=0, i &lt; a, i++){ 3      console.log(i) 4  } </pre>	<pre> ===== JAVASCRIPT SYNTAX LINTER 三位棉兰男神 =====  Checking test/will_fail2.js... Please wait for a moment...   -----    Compile Error...    -----   Execution time : 0.43 seconds </pre>
Simple Program 4 (failed)	 <pre> 1  while (i){ 2      console.log(i); 3      if { 4          break; 5      } 6      i++; 7  } </pre>	<pre> ===== JAVASCRIPT SYNTAX LINTER 三位棉兰男神 =====  Checking test/will_fail3.js... Please wait for a moment...   -----    Compile Error...    -----   Execution time : 0.21 seconds </pre>

## **BAB VI**

### **Github Repository**

<https://github.com/JeffreyChow19/TubesTBFO.git>

## **BAB VII**

### **Pembagian Tugas**

<b>PIC (Person in Charge)</b>	<b>Job Description</b>
<b>13521046 Jeffrey Chow</b>	Grammar parser, CFG to CNF, CYK, main, FA (CFG.py, CYK.py, FA.py, main.py)
<b>13521054 Wilson Tansil</b>	Grammar and Lexer (grammar.txt, lexer.py)
<b>13521102 Jimly Firdaus</b>	Grammar, Lexer, FA (grammar.txt, lexer.py, FA.py)

## **BAB VIII**

### **Referensi**

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

<https://www.geeksforgeeks.org/convertng-context-free-grammar-chomsky-normal-form/>

<https://gomakethings.com/javascript-linters/#:~:text=JavaScript%20linters%20are%20tools%20that,can%20use%20to%20fix%20thin gs.>

<https://fairuzelsaid.wordpress.com/2011/06/16/tbo-context-free-grammar-cfg/>

<https://www.trivusi.web.id/2022/08/finite-state-automata.html>