LAPORAN TUGAS BESAR
IF2124 TEORI BAHASA FORMAL DAN OTOMATA

PARSER JAVASCRIPT

Anggota Kelompok :
Naufal Syifa Firdaus (13521050)
Husnia Munzayana (13521077)
Puti Nabilla Aidira (13521088)

Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2022

DAFTAR ISI

BAB I
TEORI DASAR

1.1 Finite Automata (FA)

Automata adalah perangkat komputasi abstrak sebagai alat bantu pemodelan berbagai proses, seperti protocol, sirkuit elektronik, dan software. Finite Automata adalah kumpulan state terbatas dengan transition rules atau aturan perpindahan dari suatu state ke state lain. Finite Automata memiliki 5 elemen atau tuple, antara lain:

$$M = (Q, \Sigma, \delta, q0, F)$$

Q menyatakan himpunan state

Σ menyatakan himpunan alphabet atau simbol input

δ menyatakan fungsi transisi,
dimana $\delta(q, a)$ menyatakan fungsi transisi ketika berada pada state q dan diterima input a

$q0$ menyatakan initial state atau state awal

F menyatakan final state atau state akhir

Dalam finite automata, string adalah kumpulan simbol atau alphabet terbatas, sedangkan languages adalah kumpulan string yang diterima dalam finite automata. Finite Automata akan memulai dari initial state ketika memulai membaca string dan setiap pembacaan string menyebabkan perubahan sesuai dengan transition rules. Ketika pembacaan string selesai dan state akhir yang diperoleh merupakan final state, maka string tersebut diterima oleh FA atau termasuk dalam languages.

Terdapat dua jenis Finite Automata, yaitu Deterministic Finite Automata (DFA) dan Non-deterministic Finite Automata (NFA). transition function pada DFA hanya menuju satu state, sedangkan transition function pada NFA dapat menuju ke beberapa state. Selain direpresentasikan melalui kumpulan transition functionnya, Finite Automata dapat direpresentasikan dengan graf dengan beberapa aturan antara lain:

a. Nodes menyatakan state
b. Arc menyatakan transition function
c. Simbol panah sebagai penanda start state

d. Double circles sebagai penanda final state

1.2 Context-Free Grammar (CFG)

Context-Free Grammar adalah aturan tata bahasa formal dengan konsep penelusuran tata bahasa. Context-Free Grammar memiliki 4 elemen atau tuple, antara lain:

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

V menyatakan himpunan terbatas variabel atau simbol non-terminal

T menyatakan himpunan terbatas simbol terminal

P menyatakan aturan produksi atau production rule,

dalam bentuk $A \rightarrow \alpha$ dengan A adalah simbol non-terminal dan $\alpha \in (V \cup T)^*$

S menyatakan start symbol

Pada CFG, start symbol digunakan untuk menurunkan string. Penelusuran production rule dilakukan untuk mengganti setiap non-terminal hingga akhirnya diperoleh simbol terminal yang termasuk dalam languages. Mesin yang mengenali CFG disebut Push Down Automata (PDA).

1.3 Chomsky Normal Form (CNF)

Chomsky Normal Form adalah CFG khusus yang production rulesnya memenuhi syarat:

a. Tidak memiliki simbol START di ruas kanan.
b. Tidak memiliki product dengan *non solitary terminal* (terminal pada production/ruas kanan yang tidak berdiri sendiri).
c. Tidak memiliki product dengan lebih dari 2 *non-terminal*.
d. Tidak memiliki ε pada product.
e. Tidak memiliki *unit rules* (*rules* yang berbentuk $A \rightarrow B$)

Semua CFG yang tidak mengandung string kosong dapat direpresentasikan dalam CNF.

1.4 Algoritma Cocke–Younger–Kasami (CYK)

Cocke–Younger–Kasami adalah algoritma parsing untuk CFG yang berbentuk CNF. CYK merupakan salah satu algoritma Dynamic Programing. CYK mempertimbangkan setiap kemungkinan substring dari string input. Untuk substring dengan panjang kurang dari sama dengan 2 CYK mengisi tabel dengan simbol non-terminal yang sesuai dengan input (*berdasarkan production rules*). Untuk substring dengan panjang 2 dan lebih besar, CYK mempertimbangkan setiap kemungkinan partisi dari substring menjadi dua bagian, dan mengisi tabel dengan simbol non-terminal yang sesuai dengan input dan sel tabel sebelumnya.

1.5 Javascript *Syntax*

1.5.1 Identifier Rules

a. *Identifier* dapat mengandung huruf, digit, *underscores, dollar sign*.
b. *Identifier* harus diawali huruf
c. *Identifier* khusus dapat diawali $ atau _ .
d. *Identifier* bersifat *case sensitive*.
e. Kata-kata yang digunakan dalam sistem (contoh: JavaScript *keywords*) tidak bisa digunakan sebagai *identifier* (tidak diverifikasi pada program kami).

1.5.2 Function

Function adalah subprogram yang bisa dipanggil oleh program lain, function dapat menerima value dan/atau menghasilkan value. Function terdiri dari nama, argumen, dan statement. Nama function adalah representasi dari function secara keseluruhan dalam kata-kata. Argumen merupakan nilai yang dimasukan kedalam function. Statement adalah kode yang mendefinisikan apa yang function lakukan ketika dipanggil. Mendefinisikan function dimulai dengan pernyataan objek function diikuti dengan nama dan setelahnya sepasang tanda kurung yang didalamnya boleh dimasukan nama dari argumen. Statement ditulis setelah pernyataan objek function di dalam sepasang kurung kurawal.

1.5.3 Object

Object adalah salah satu tipe data pada JavaScript yang digunakan untuk menyimpan berbagai properti. Properti disini merupakan pasangan kunci dengan nilainya atau key-value. Kunci atau key dari sebuah properti dapat berupa string,

sedangkan value atau nilainya dapat berupa tipe data apa saja, seperti string, integer, *array/list*, bahkan fungsi. Object diawali dan diakhiri dengan tanda kurung kurawal "{}".

1.5.4 Switch-Case

Switch digunakan untuk memilih satu dari beberapa statement yang dipilih berdasarkan ekspresi yang dikasih. Switch didefinisikan dengan pernyataan switch diikuti dengan sepasang kurung yang di dalamnya ada ekspresi yang dievaluasi. Setelah itu, diikuti dengan sepasang kurung kurawal. Di dalamnya terdapat pernyataan case yang diikuti dengan ekspresi dan kode algoritma. Case bisa berjumlah berapapun. Isi kurung kurawal yang terakhir adalah kasus default diikuti dengan kode.

1.5.5 Conditional Statement

Pernyataan if..else mengeksekusi salah satu statement yang dinilai benar. if..else memiliki pernyataan if, else, dan else if. Pernyataan diikuti dengan sepasang kurung yang didalamnya ada ekspresi boolean kecuali pernyataan else. Lalu, diikuti dengan sepasang kurung kurawal yang diisi dengan kode program.

1.5.6 Try-Catch

Try-catch digunakan jika ada exception error ketika kode berjalan sebagai cara menanggapi kode tersebut. Try-catch terdiri dari pernyataan try, catch dan finally. Try mengeksekusi sebuah kode program yang akan ditinjau keberjalanannya. Catch adalah kode yang dieksekusi ketika ada exception dalam kode try. Finally adalah kode yang dieksekusi sebelum program utama berakhir. Urutan pernyataan harus try, catch, lalu finally. Masing-masing pernyataan diikuti dengan sepasang kurung kurawal yang berisi kode.

1.5.7 Control Flow (Break, Continue)

Pernyataan break digunakan untuk keluar dari sebuah loop. Continue digunakan untuk melewati satu iterasi sebuah look dan melanjutkan ke iterasi selanjutnya. Pernyataan break dan continue harus ada dalam sebuah loop.

1.5.8 Let, Const, Var

a. Let digunakan untuk mendeklarasikan sebuah variabel. Pernyataan let diikuti dengan nama variabel yang akan dideklarasikan. Lalu dapat dilanjutkan dengan tanda assignment dan suatu ekspresi.
b. Const mempunyai syntax yang sama seperti let tapi variabel yang dideklarasikan dengan const tidak bisa diganti dan diassign kembali.
c. Var mempunyai syntax yang sama dengan let dan const tapi variabel yang dideklarasikan dengan var dapat diakses keseluruhan program.

1.5.9 While & Do-While Loop

Pernyataan while mirip seperti pernyataan if dengan ekspresi kondisional di dalam kurung diikuti dengan kode dalam kurung kurawal. While berfungsi untuk mengulangi kode program sampai ekspresi kondisional bernilai salah.

1.5.10 For Loop

For loop mengulangi kode program didalamnya senilai jumlah yang telah ditetapkan. Pernyataan for diikuti dengan sepasang kurung yang didalamnya terdapat tiga ekspresi yang dipisahkan dengan titik koma. ekspresi pertama adalah assignment pada variabel iterasi. Ekspresi kedua adalah kondisional variabel tersebut dengan sebuah nilai. Terakhir adalah manipulasi variabel iterasi setiap loopnya. Lalu sepasang kurung kurawal yang berisi kode program.

1.5.11 Throw

Throw adalah pernyataan untuk memberi program exception (keluar dari fungsi kemudian mencari catch selanjutnya). Pernyataan throw dapat diikuti dengan pesan exception dari pengguna.

1.5.12 Return

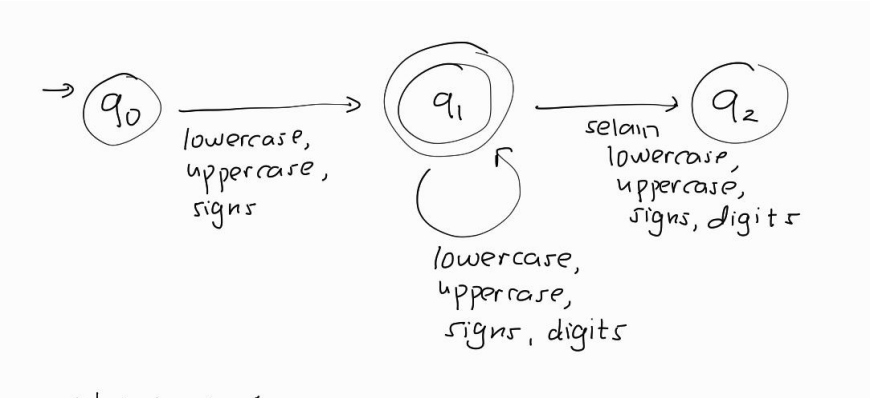
Return adalah mengeluarkan sebuah nilai dari fungsi sekaligus keluar dari fungsi tersebut. Pernyataan return tidak bisa ada diluar fungsi dan harus diikuti dengan sebuah nilai atau variabel.

BAB II

DESKRIPSI CFG DAN FA

2.1. Finite Automata Validasi Variabel

$M = (\{q0,\ q1,\ q2\},\ \{semua\ character\},\ \delta,\ q0, q1)$



Gambar 2.1 Finite Automata Validasi Variabel

lowercase = {a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z}

uppercase = {A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z}

signs = {&, _}

digits = {0,1,2,3,4,55,6,7,8,9}

L = {x, _y, Bobi, b0bi, …}

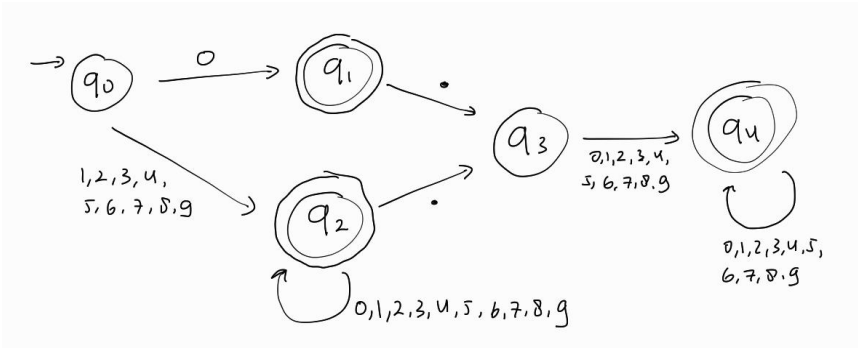
$q0$ = Start state

$q1$ = Variabel valid, final state

$q2$ = Variabel tidak valid, ada karakter selain lowercase, uppercase, signs, dan digits

2.2 Finite Automata Validasi Angka

$M = (\{q0,\ q1,\ q2,\ q3,\ q4\},\ \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9,\ .\},\ \delta,\ q0, \{q1,\ q2,\ q4\})$



Gambar 2.2 Finite Automata Validasi Bilangan/Angka

L = {0, 4, 92, 0.9, 19.2, …}

$q0$ = Start state

$q1$ = Angka diawali dengan 0 valid, Final State

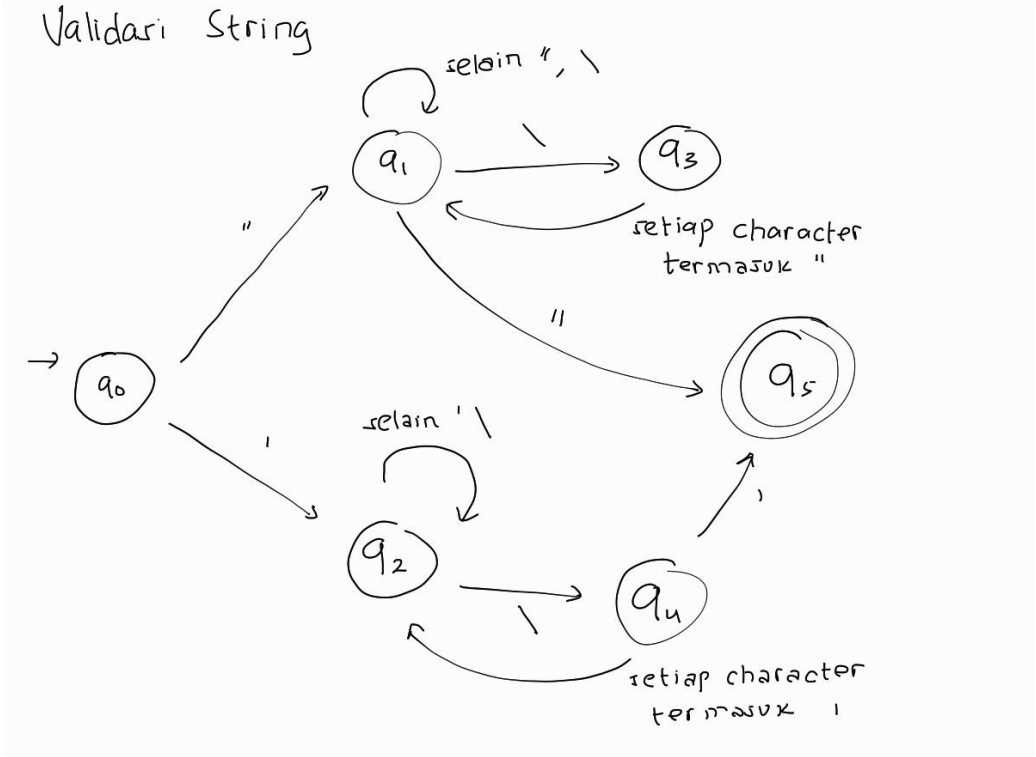
$q2$ = Angka diawali bukan 0 valid, Final State

$q3$ = Terdapat karakter ‘.’ , angka merupakan bilangan desimal

$q4$ = Angka desimal valid, Final State

2.3 Finite Automata Validasi String

$M = (\{q0, q1, q2, q3, q4, q5\}, \{semua\ character\}, \delta, q0, q5)$



Gambar 2.3 Finite Automata Validasi String

L = {“Halo”, ‘0hai’, ‘it\’s’, “cont\ ‘oh”, …}

$q0$ = Start state

$q1$ = String diawali karakter “ valid

$q2$ = String diawali karakter ‘ valid

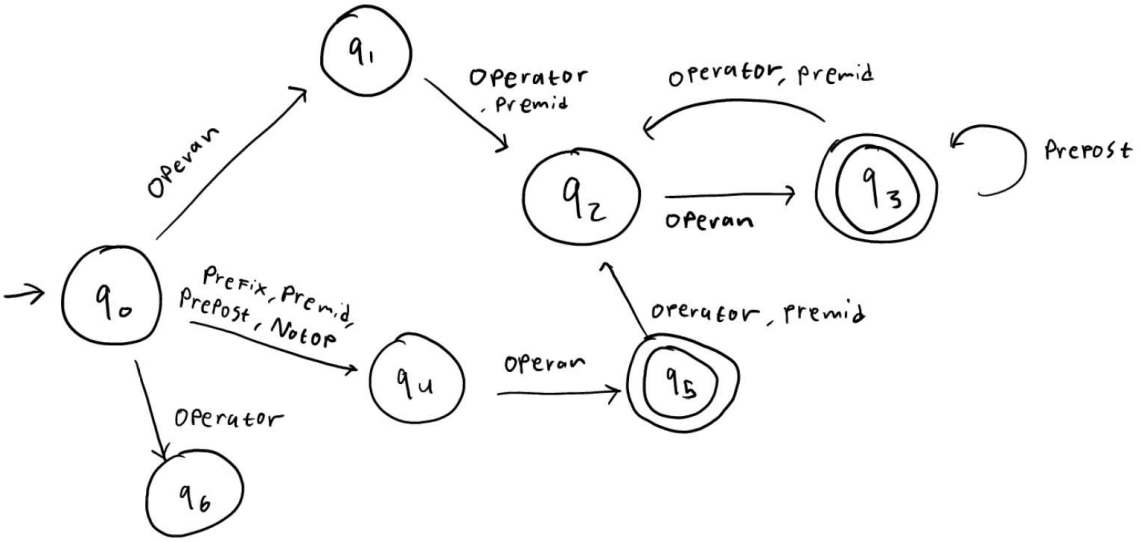
$q3$ = Penanganan escaped character untuk string diawali “

$q4$ = Penanganan escaped character untuk string diawali ‘

$q5$ = String valid, Final State

2.4 Finite Automata Validasi Ekspresi

$M = (\{q0,\ q1,\ q2,\ q3,\ q4, q5\},\ \{operator, operan, prefix, premid, prepost, not\}$
$\delta,\ q0, \{q1,\ q2,\ q4\})$



Gambar 2.4 Finite Automata Validasi Ekspresi

operator = { +, *, /, ==, ===, !=, !==, <, >, <=, >=, ?}
operan = {variabel, huruf, angka, true, false, null}
prefix operator = {~}
premid operator = {-}
prepost operator = {++, –}
not operator = {!}

$q0$ = Start State
$q1$ = String diawali sebuah operan
$q2$ = operan yang diikuti dengan operator dan premid
$q3$ = String diakhri dengan operan atau prepost, Final State
$q4$ = String diawali dengan prefix, premid, prepost, atau not
$q5$ = String valid, Final State
$q6$ = String dengan operator tanpa operan

2.5 Context-Free Grammar

```
START -> S S
S -> FUNC | NONFUNC | VALIDCONST | VALIDLET

//Blok-blok program/statement umum yang bukan fungsi, bukan
pula yang khusus bisa terletak dalam fungsi maupun loop.

NONFUNC -> NONFUNC NONFUNC | NONFUNC NEWLINE NONFUNC |
NONFUNC NEWLINE | NEWLINE NONFUNC | EXPR NEWLINE | SWITCHCASE
| CONDITIONAL | TRYCATCHFIN | VALIDLET NONFUNC | VALIDCONST
NONFUNC | VALIDVAR | WHILELOOP | FORLOOP | DOWHILELOOP |
VALIDCONST NONFUNC | VALIDLET NONFUNC | VALIDVAR |
VALIDDELETE | ASSIGNMENT | OTHERASSIGNMENT


//Blok-blok program/statement yang bisa terletak dalam
fungsi.

INFUNC -> NONFUNC | INFUNC INFUNC | INFUNC NEWLINE INFUNC |
INFUNC NEWLINE | NEWLINE INFUNC | SWITCHCASEINFUNC |
CONDITIONALINFUNC | TRYCATCHFININFUNC | VALIDTHROW | VALIDRET
| FORLOOPINFUNC | WHILELOOPINFUNC | DOWHILELOOPINFUNC


//Blok-blok program/statement yang bisa terletak dalam loop.

INLOOP -> NONFUNC | INLOOP INLOOP | INLOOP NEWLINE INLOOP |
INLOOP NEWLINE | NEWLINE INLOOP | CONDITIONALINLOOP |
CONTROLFLOW


//Blok-blok program/statement yang bisa terletak dalam loop
dan fungsi.

INLOOPANDFUNC -> INLOOPANDFUNC NEWLINE INLOOPANDFUNC |
INLOOPANDFUNC NEWLINE | NEWLINE INLOOPANDFUNC | INLOOP |
INFUNC


//Blok-blok program yang merupakan deklarasi fungsi.

FUNC -> FUNC FUNC | FUNC NEWLINE FUNC | FUNC NEWLINE |
NEWLINE FUNC

//Rules untuk non-terminal yang menjadi placeholder untuk
terminal.

FUNCTION -> function
```

```
SWITCH -> switch
CURLYOPEN -> _curlyOpen_
CURLYCLOSE -> _curlyClose_
COMMONOPEN -> _commonOpen_
COMMONCLOSE -> _commonClose_
SQUAREOPEN -> _squareOpen_
SQUARECLOSE -> _squareClose_
CASE -> case
DEFAULT -> default
COLON -> _colon_
SEMICOLON -> _semicolon_
ASSIGN -> _assign_
EQUALSIGN -> _equalSign_
COMMA -> _comma_
IF -> if
ELSE -> else
EXPR -> _expression_ | ASSIGNMENT | VARIABLE | VALUE |
PARAMLIST
BOOLEAN -> true | false
VARIABLE -> _variable_
VALUE -> NUMBER | STRING | BOOLEAN | NULL
NUMBER -> _number_
NULL -> null
FUNCNAME -> VARIABLE
STRING -> _string_
CONTROLFLOW -> break | continue
RETURN -> return
THROW -> throw
TRY -> try
CATCH -> catch
FINALLY -> finally
CONST -> const
VAR -> var
LET -> let
FOR -> for
WHILE -> while
NEWLINE -> _newline_ | NEWLINE NEWLINE
DELETE -> delete

//Rules untuk setiap kata kunci yang dievaluasi.

//Throw, Return.
VALIDTHROW -> THROW EXPR | THROW VARIABLE | THROW OBJECTDEF |
THROW STRING | THROW FUNC NEWLINE
VALIDRET -> RETURN | RETURN VARIABLE | RETURN EXPR | RETURN
OBJECTDEF | RETURN STRING | RETURN FUNC NEWLINE

//Object definition.
INOBJECT -> VARIABLE COLON VALUE | VARIABLE COLON VARIABLE |
VARIABLE COLON INOBJECT | INOBJECT COMMA INOBJECT
```

```
OBJECTDEF -> VALIDCURLYOPEN INOBJECT VALIDCURLYCLOSE

//Assignment
ASSIGNMENT -> VARIABLE EQUALSIGN EXPR | VARIABLE EQUALSIGN
OBJECTDEF NEWLINE | VARIABLE EQUALSIGN ASSIGNMENT

//Let, Const, Var.
VALIDLET -> LET VARIABLE NEWLINE | LET ASSIGNMENT | VALIDLET
VALIDLET
VALIDCONST -> CONST ASSIGNMENT | VALIDCONST VALIDCONST
VALIDVAR -> VAR VARIABLE NEWLINE | VAR ASSIGNMENT

// +=, -=, **=, /=, ^=, |=, dll .
OTHERASSIGNMENT -> VARIABLE ASSIGN EXPR NEWLINE | VARIABLE
ASSIGN OBJECTDEF NEWLINE | VARIABLE ASSIGN ASSIGNMENT

// Delete operator.
VALIDDELETE -> DELETE VARIABLE SQUAREOPEN VARIABLE
SQUARECLOSE NEWLINE

// Condition (expression).
CONDITION -> COMMONOPEN EXPR COMMONCLOSE

// { } yang digunakan untuk menandakan blok program.
VALIDCURLYOPEN -> CURLYOPEN | NEWLINE CURLYOPEN | NEWLINE
CURLYOPEN NEWLINE | CURLYOPEN NEWLINE
VALIDCURLYCLOSE -> CURLYCLOSE | NEWLINE CURLYCLOSE | NEWLINE
CURLYCLOSE NEWLINE | CURLYCLOSE NEWLINE

// Switch case.
SWITCHCASE -> SWITCH CONDITION VALIDCURLYOPEN VALIDCASE
VALIDCURLYCLOSE | SWITCH CONDITION VALIDCURLYOPEN VALIDCASE
VALIDDEF VALIDCURLYCLOSE
SWITCHCASEINFUNC -> SWITCH CONDITION VALIDCURLYOPEN
VALIDCASEINFUNC VALIDCURLYCLOSE | SWITCH CONDITION
VALIDCURLYOPEN VALIDCASEINFUNC VALIDDEFINFUNC VALIDCURLYCLOSE
VALIDCASE -> VALIDCASE VALIDCASE | CASE VALUE COLON | CASE
VALUE COLON NEWLINE | CASE VALUE COLON NONFUNC | CASE VALUE
COLON NONFUNC NEWLINE | CASE VALUE COLON NONFUNC NEWLINE
CONTROLFLOW NEWLINE
VALIDDEF -> DEFAULT COLON NEWLINE | DEFAULT COLON NONFUNC |
DEFAULT COLON NONFUNC NEWLINE | DEFAULT COLON NONFUNC NEWLINE
CONTROLFLOW NEWLINE
VALIDCASEINFUNC -> VALIDCASEINFUNC VALIDCASEINFUNC |
VALIDCASE | CASE VALUE COLON INFUNC NEWLINE | CASE VALUE
COLON INFUNC
VALIDDEFINFUNC -> VALIDDEF | DEFAULT COLON INFUNC NEWLINE |
DEFAULT COLON INFUNC

// Conditional statement.
```

```
VALIDIF -> IF CONDITION VALIDCURLYOPEN NONFUNC
VALIDCURLYCLOSE | IF CONDITION NONFUNC NEWLINE
VALIDIFINLOOP -> VALIDIF | IF CONDITION VALIDCURLYOPEN INLOOP
VALIDCURLYCLOSE | IF CONDITION INLOOP NEWLINE
VALIDIFINFUNC -> VALIDIF | IF CONDITION VALIDCURLYOPEN INFUNC
VALIDCURLYCLOSE | IF CONDITION INFUNC NEWLINE
VALIDELSE -> ELSE VALIDCURLYOPEN NONFUNC VALIDCURLYCLOSE |
ELSE NONFUNC NEWLINE
VALIDELSEINLOOP -> VALIDELSE | ELSE VALIDCURLYOPEN INLOOP
VALIDCURLYCLOSE | ELSE INLOOP NEWLINE
VALIDELSEINFUNC -> VALIDELSE | ELSE VALIDCURLYOPEN INFUNC
VALIDCURLYCLOSE | ELSE INFUNC NEWLINE
VALIDELSEIF -> ELSE IF CONDITION VALIDCURLYOPEN NONFUNC
VALIDCURLYCLOSE | ELSE IF CONDITION NONFUNC NEWLINE |
VALIDELSEIF VALIDELSEIF
VALIDELSEIFINLOOP -> VALIDELSEIF | ELSE IF CONDITION
VALIDCURLYOPEN INLOOP VALIDCURLYCLOSE | ELSE IF CONDITION
INLOOP NEWLINE | VALIDELSEIFINLOOP VALIDELSEIFINLOOP
VALIDELSEIFINFUNC -> VALIDELSEIF | ELSE IF CONDITION
VALIDCURLYOPEN INFUNC VALIDCURLYCLOSE | ELSE IF CONDITION
INFUNC NEWLINE | VALIDELSEIFINFUNC VALIDELSEIFINFUNC
CONDITIONAL -> VALIDIF | VALIDIF VALIDELSE | VALIDIF
VALIDELSEIF VALIDELSE
CONDITIONALINLOOP -> VALIDIFINLOOP | VALIDIFINLOOP
VALIDELSEINLOOP | VALIDIFINLOOP VALIDELSEIFINLOOP
VALIDELSEINLOOP
CONDITIONALINFUNC -> VALIDIFINFUNC | VALIDIFINFUNC
VALIDELSEINFUNC | VALIDIFINFUNC VALIDELSEIFINFUNC
VALIDELSEINFUNC

// Try Catch.
VALIDTRY -> TRY VALIDCURLYOPEN NONFUNC VALIDCURLYCLOSE
VALIDCATCH -> CATCH VALIDCURLYOPEN NONFUNC VALIDCURLYCLOSE |
CATCH COMMONOPEN VARIABLE COMMONCLOSE VALIDCURLYOPEN NONFUNC
VALIDCURLYCLOSE
VALIDFIN -> FINALLY VALIDCURLYOPEN NONFUNC VALIDCURLYCLOSE
VALIDTRYINFUNC -> VALIDTRY | TRY VALIDCURLYOPEN INFUNC
VALIDCURLYCLOSE
VALIDCATCHINFUNC -> VALIDCATCH | CATCH VALIDCURLYOPEN INFUNC
VALIDCURLYCLOSE | CATCH COMMONOPEN VARIABLE COMMONCLOSE
VALIDCURLYOPEN INFUNC VALIDCURLYCLOSE
VALIDFININFUNC -> VALIDFIN | FINALLY VALIDCURLYOPEN INFUNC
VALIDCURLYCLOSE
TRYCATCHFIN -> VALIDTRY VALIDCATCH | VALIDTRY VALIDFIN |
VALIDTRY VALIDCATCH VALIDFIN
TRYCATCHFININFUNC -> VALIDTRYINFUNC VALIDCATCHINFUNC |
VALIDTRYINFUNC VALIDFININFUNC | VALIDTRYINFUNC
VALIDCATCHINFUNC VALIDFININFUNC

// Function declaration.
```

```
FUNC -> FUNCTION FUNCNAME COMMONOPEN COMMONCLOSE
VALIDCURLYOPEN INFUNC VALIDCURLYCLOSE | FUNCTION FUNCNAME
COMMONOPEN COMMONCLOSE VALIDCURLYOPEN INFUNC NEWLINE VALIDRET
VALIDCURLYCLOSE | FUNCTION FUNCNAME COMMONOPEN PARAMLIST
COMMONCLOSE VALIDCURLYOPEN INFUNC VALIDCURLYCLOSE | FUNCTION
FUNCNAME COMMONOPEN PARAMLIST COMMONCLOSE VALIDCURLYOPEN
INFUNC NEWLINE VALIDRET VALIDCURLYCLOSE | FUNCTION FUNCNAME
COMMONOPEN COMMONCLOSE VALIDCURLYOPEN FUNC VALIDCURLYCLOSE |
FUNCTION FUNCNAME COMMONOPEN COMMONCLOSE VALIDCURLYOPEN FUNC
NEWLINE VALIDRET VALIDCURLYCLOSE | FUNCTION FUNCNAME
COMMONOPEN PARAMLIST COMMONCLOSE VALIDCURLYOPEN FUNC
VALIDCURLYCLOSE | FUNCTION FUNCNAME COMMONOPEN PARAMLIST
COMMONCLOSE VALIDCURLYOPEN FUNC NEWLINE VALIDRET
VALIDCURLYCLOSE

// Parameter.
PARAM -> VARIABLE | VARIABLE EQUALSIGN VALUE
PARAMLIST -> PARAM | PARAM COMMA PARAM | PARAM COMMA
PARAMLIST COMMA PARAM

// Loop.
FORLOOP -> FOR COMMONOPEN EXPR NEWLINE EXPR NEWLINE EXPR
COMMONCLOSE VALIDCURLYOPEN INLOOP VALIDCURLYCLOSE | FOR
COMMONOPEN EXPR SEMICOLON EXPR SEMICOLON EXPR COMMONCLOSE
INLOOP NEWLINE
FORLOOPINFUNC -> FORLOOP | FOR COMMONOPEN EXPR SEMICOLON EXPR
SEMICOLON EXPR COMMONCLOSE VALIDCURLYOPEN INLOOPANDFUNC
VALIDCURLYCLOSE | FOR COMMONOPEN EXPR SEMICOLON EXPR
SEMICOLON EXPR COMMONCLOSE INLOOPANDFUNC NEWLINE
WHILELOOP -> WHILE CONDITION INLOOP NEWLINE | WHILE CONDITION
VALIDCURLYOPEN INLOOP VALIDCURLYCLOSE
WHILELOOPINFUNC -> WHILELOOP | WHILE CONDITION INLOOPANDFUNC
NEWLINE | WHILE CONDITION VALIDCURLYOPEN INLOOPANDFUNC
VALIDCURLYCLOSE
DOWHILELOOP -> DO INLOOP NEWLINE WHILE CONDITION NEWLINE | DO
VALIDCURLYOPEN INLOOP VALIDCURLYCLOSE WHILE CONDITION
DOWHILELOOPINFUNC -> DOWHILELOOP | DO INLOOPANDFUNC NEWLINE
WHILE CONDITION NEWLINE | DO VALIDCURLYOPEN INLOOPANDFUNC
VALIDCURLYCLOSE WHILE CONDITION
```

BAB III

IMPLEMENTASI DAN PENGUJIAN

3.1. Spesifikasi Teknis Program

3.1.1 File main.py

Header fungsi/prosedur	Struktur Data yang digunakan	Gambaran Umum Algoritma
slowprint(s)	Primitive : float, char User-defined : Time	Memperindah tampilan dengan memberi jeda output menggambarkan proses sedang berjalan
ReadyToParse(file)	Primitive : String, Boolean, Integer Built-in : List/Array of string, Matrix/array of array	Mempersiapkan text untuk diparsing dengan mengganti simbol seperti brackets, variabel, number, string, dll dengan string tertentu agar mudah dideteksi saat parsing. Dalam fungsi ini juga dilakukan proses validasi variabel dan number.

3.1.2 File globalVariable.py

Gambaran Umum Algoritma	Struktur Data yang digunakan
Digunakan untuk menyimpan variabel global agar dapat diakses dari file program lain dan program tidak memanggil/import satu sama lain.	Primitive : String, Boolean, Integer Built-in : Set

3.1.3 File validNumber.py

Header fungsi/prosedur	Struktur Data yang digunakan	Gambaran Umum Algoritma
numberCheck(word)	Primitive : String, Boolean, Integer Built-in : List/Array of string	Melakukan validasi number dengan Finite Automata

isNumber(list_word)	Primitive : String, Boolean, Integer Built-in : List/Array of string	Melakukan pengecekan apakah kata dalam list_word merupakan bilangan atau angka, jika ya maka lakukan validasi dengan memanggil fungsi numberCheck

3.1.4 File validVariable.py

Header fungsi/prosedur	Struktur Data yang digunakan	Gambaran Umum Algoritma
variableCheck(word)	Primitive : String, Boolean, Integer Built-in : List/Array of string	Melakukan validasi variabel dengan Finite Automata
isVariable(list_word, replacedsymbol, js_grammar)	Primitive : String, Boolean, Integer Built-in : List/Array of string	Melakukan pengecekan apakah kata dalam list_word merupakan variabel. kata dalam list_word merupakan variabel jika tidak termasuk dalam replacedsymbol dan js_grammar. Jika kata tersebut merupakan variabel, maka lakukan validasi dengan memanggil fungsi variableCheck

3.1.5 File expressionCheck.py

Header fungsi/prosedur	Struktur Data yang digunakan	Gambaran Umum Algoritma
isOperator(word)	Primitive : String	Menghasilkan true jika word adalah sebuah string yang merepresentasikan operator. (+,-, &&, dsb.)
isOperan(word)	Primitive : String	Menghasilkan true jika word adalah sebuah string yang merepresentasikan operan dalam ekspresi

		(variabel, angka, dsb.)
isPrefixOp(word)	Primitive : String	Menghasilkan true jika word adalah Operator yang letaknya sebelum operan.
isPremidOp(word)	Primitive : String	Menghasilkan true jika word adalah operator yang letaknya sebelum atau ditengah operan.
isPrepost(word)	Primitive : String	Menghasilkan true jika word adalah operator yang letaknya sebelum atau sesudah operan.
isNotOp(word)	Primitive : String	Menghasilkan true jika word adalah operator NOT (!).
checkInfix(line, index)	Primitive : Integer, Boolean Built-in : List/Array of String	Fungsi yang memvalidasi sebuah array of string dimulai dari index tertentu, atas keberadaan ekspresi infix yang valid dan boleh berulang. Fungsi menghasilkan boolean validitas dan integer index dimana ekspresi tersebut berakhir
validity(line)	Primitive : Integer, Boolean Built-in : List/Array of String	Fungsi yang memvalidasi sebuah array of string dimulai dari awal. Fungsi memeriksa semua kemungkinan jenis ekspresi dan menghasilkan boolean validitas dan ditemukan atau tidaknya ekspresi serta integer indeks awal dan akhir ekspresi.
elimExpression(start,line)	Primitive : Integer Built-in : List/Array of String	Mengganti semua ekspresi yang valid dengan string “_expression_”.
expressionCheck(matofword)	Built-In : Matrix/Array of Array of String	Memvalidasi keseluruhan isi matriks akan keberadaan ekspresi yang

		benar. Fungsi menghasilkan matriks yang sudah diolah, boolean validitas, dan indeks baris yang error (jika ada).

3.1.6 CNFconverter.py

Header fungsi/prosedur	Struktur Data yang digunakan	Gambaran Umum Algoritma
isNonTerminal(sym)	Primitive : Char, String, Boolean	*Fungsi pembantu simplify(CFG).* Mengembalikan true jika sym adalah non-terminal (huruf besar).
readProdRule(CFGFile)	Primitive : String Built-In : List, Dict User-defined : File	Fungsi yang membaca
elimunitrules(CFG)	Primitive : Boolean Built-in : Dict	Mengeliminasi unit rules (jika ada A → B, mencari B → "..." untuk disubtitusikan).
singleProd(CFG)	Built-in : Dict	Mengeliminasi product dengan panjang lebih dari 2 (jika ada A → B C D membuat non-terminal baru & productionnya untuk disubtitusikan).
CNFconverter(dir)	-	Fungsi yang menyatukan proses readProdRule(CFGFile), simplify(CFG), dan singleProd(CFG). Dipanggil pada main.py.
writeCNF(CNF)	User-defined : File	*Fungsi pembantu debugging hasil CNF.* Mencetak hasil CNF ke file.

3.1.7 CYK.py

Header fungsi/prosedur	Struktur Data yang	Gambaran Umum

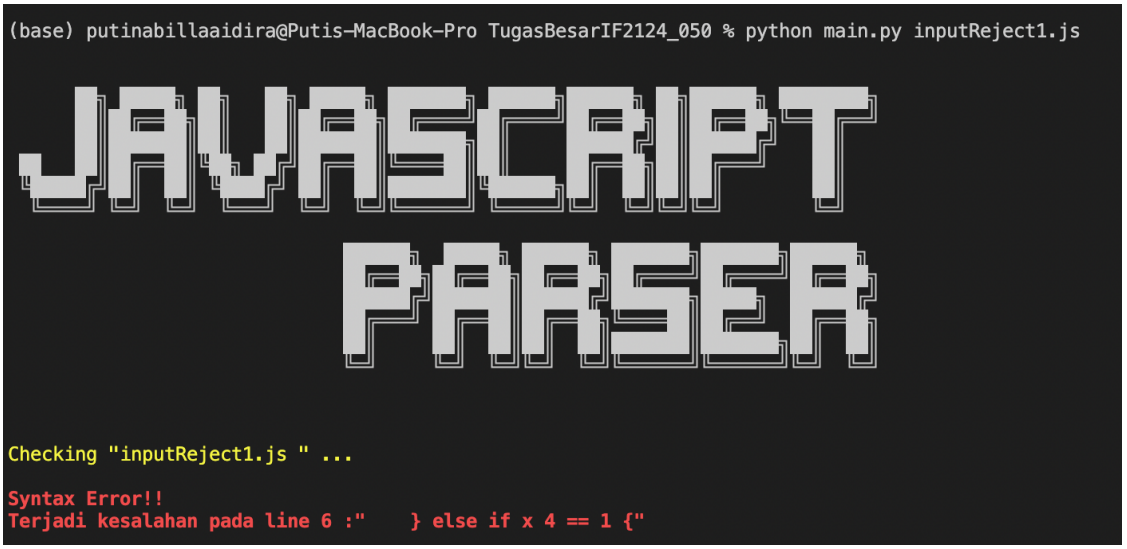
	digunakan	Algoritma
CYK(grammar, arrofword)	Built-in : Matriks/Array of Array, Dict, Array/List	Menerapkan algoritma CYK pada arrofword berdasarkan grammar (mengisi tabel CYK dengan simbol non-terminal yang cocok dengan input).

3.2. Uji Kasus

3.2.1 inputReject1.js

```
// inputReject.js
function do_something(x) {
    // This is a sample multiline comment
    if (x == 0) {
      return 0;
    } else if x 4 == 1 {
      if (true) {
        return 3;
      } else {
        return 2;
      }
    } else if (x == 32) {
      return 4;
    } else {
      return "Momen";
    }
  }
```

Hasil Uji:

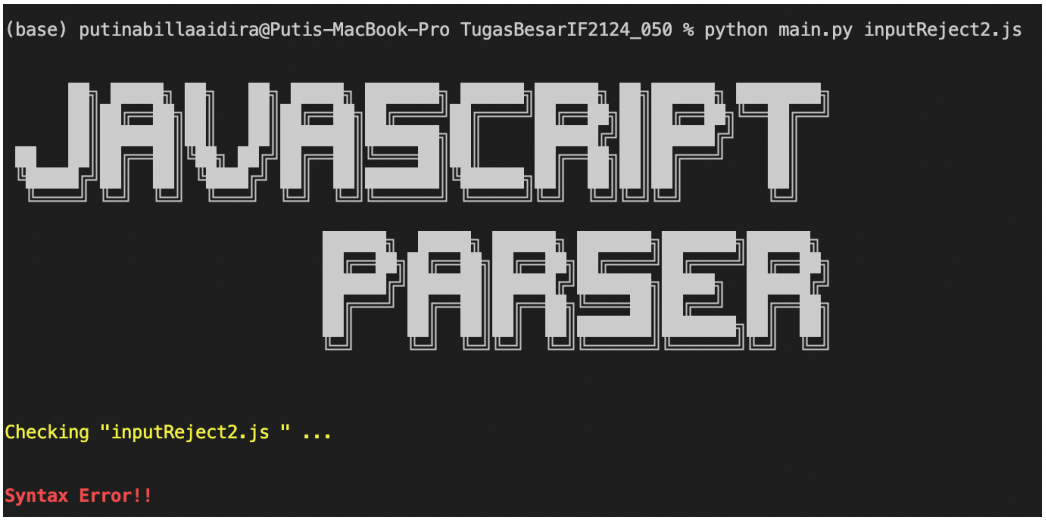


Analisis: Pada line 6 ekspresi x 4 == 1 memiliki syntax yang salah karena antara variabel x dan 4 tidak terdapat operator. Selanjutnya, else if tanpa tanda kurung juga merupakan kesalahan syntax.

3.2.2 inputReject2.js

```
// inputReject.js
function do_something(x) {
    /* ffgfh
    else (x == 0) {
      return 0;
    } else if x 4 == 1 {
      if (true) {
        return 3;
      } else {
        return 2;
      }
    } else if (x == 32) {
      return 4;
    } else {
      return "Momen";
    }
  }
```

Hasil Uji:



Analisis: Pada line 4 terdapat kesalahan syntax karena else tidak didahului if.

3.2.3 inputReject3.js

```
if(true){
    const x = 2
}
for(x; y){
    let x
}
yey = 'true
```

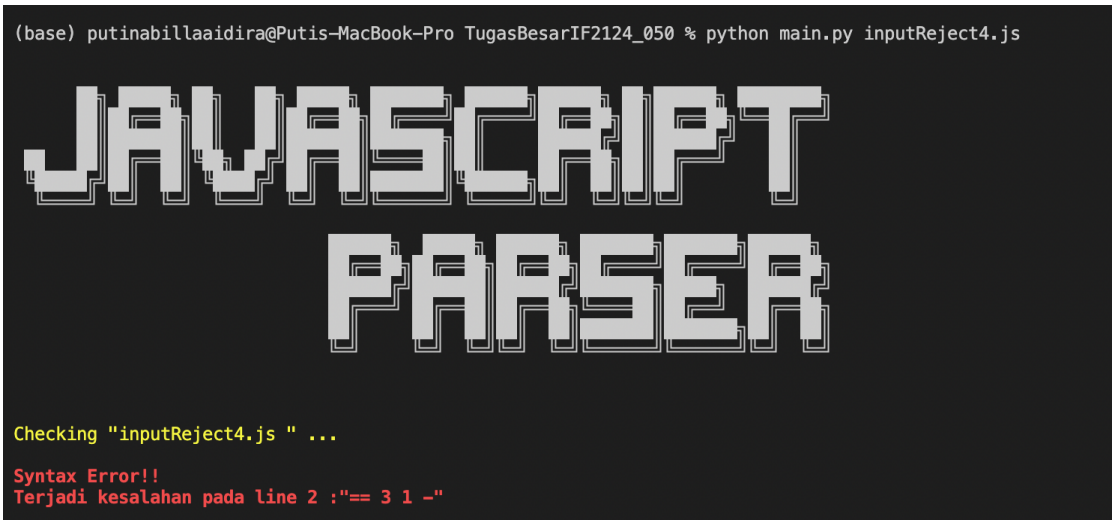
Hasil Uji:

Analisis: Pada line 7 terdapat kesalahan syntax karena terdapat tanda petik satu tetapi tidak terdapat pasangannya sebagai penanda tutup string. Pada line 4 terdapat kesalahan syntax pada for, tetapi pesan error tidak dicetak karena pengecekan dilakukan pada string terlebih dahulu.

3.2.3 inputReject4.js

```
4 +
== 3 1 -
0dfs
1dfdbd
```

Hasil Uji:



Analisis: Pada line 1 terdapat kesalahan syntax karena operator kekurangan operan, tapi pesan error tidak dicetak untuk line 1 karena operan masih 'dicari' di line berikutnya. Selanjutnya, pada line 2 terdapat kesalahan syntax karena *operator precedence* yang salah, pesan error dicetak untuk line ini. Pada line 3 dan 4 terdapat

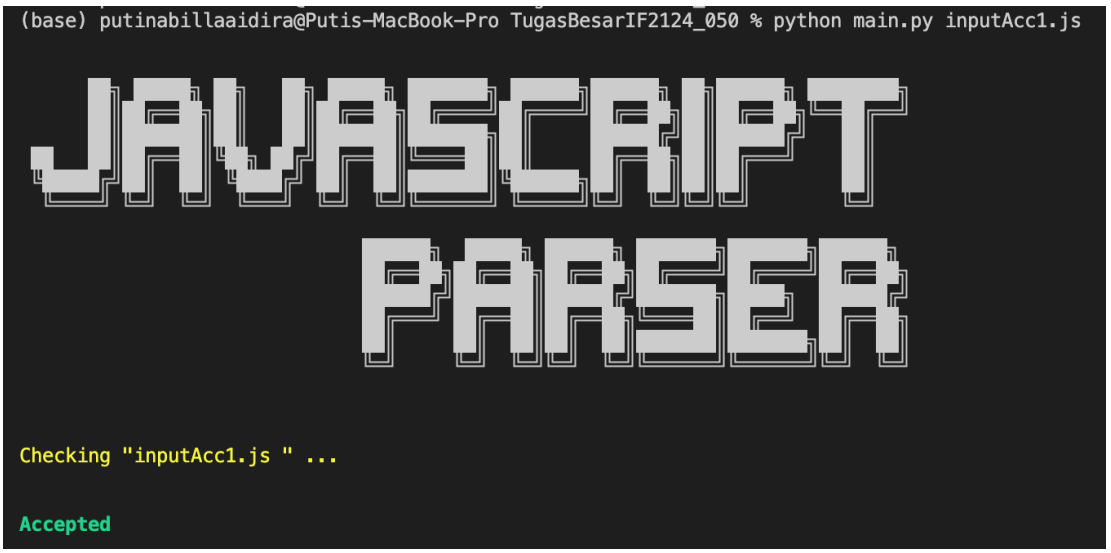
kesalahan penamaan variabel (*identifier*), namun pesan error tidak dicetak karena pesan error hanya dicetak untuk line error pertama.

3.2.5 inputAcc1.js

```
// inputAcc.js
function do_something(x) {
    // This is a sample comment
    x = 5 /*
    Halo bang
    */
    y = "Haig\'ais"
    x = 'Hai\"bang'
    y = "Haig\'ais" // Ini contoh esc
    for(x,y;y;z){
      true
    }
    if (v) {
      return 0
    } else if (x + 4 == 1)
    {
      if (true) {
        return 3.5
      } else {
        y = "ak\"u"
        return 2
      }

    }
    else if (x == 32) {
      return 4
    }
    else {return "Momen\'"}
  }
```

Hasil Uji:

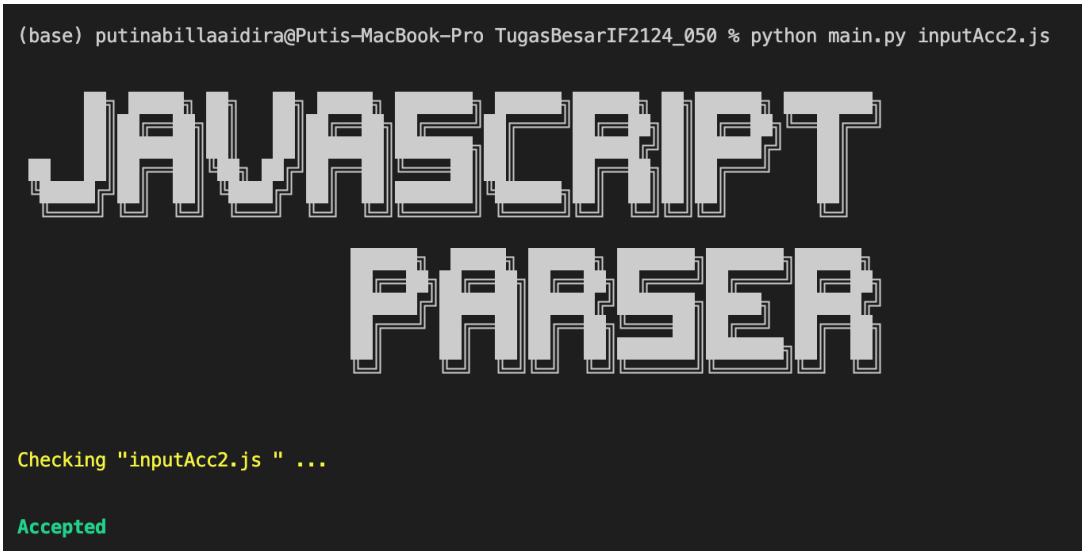


Analisis: Semua syntax sesuai.

3.2.6 inputAcc2.js

```
// inputAcc.js
function do_something(x) {
    // This is a sample comment
    if (x == 0) {
      return 0
    } else if (x + 4 == 1) {
      if (true) {
        return 3
      } else {
        return 2
      }
    } else if (x == 32) {
      return 4
    } else {
      return "Momen"
    }
  }
```

Hasil Uji:

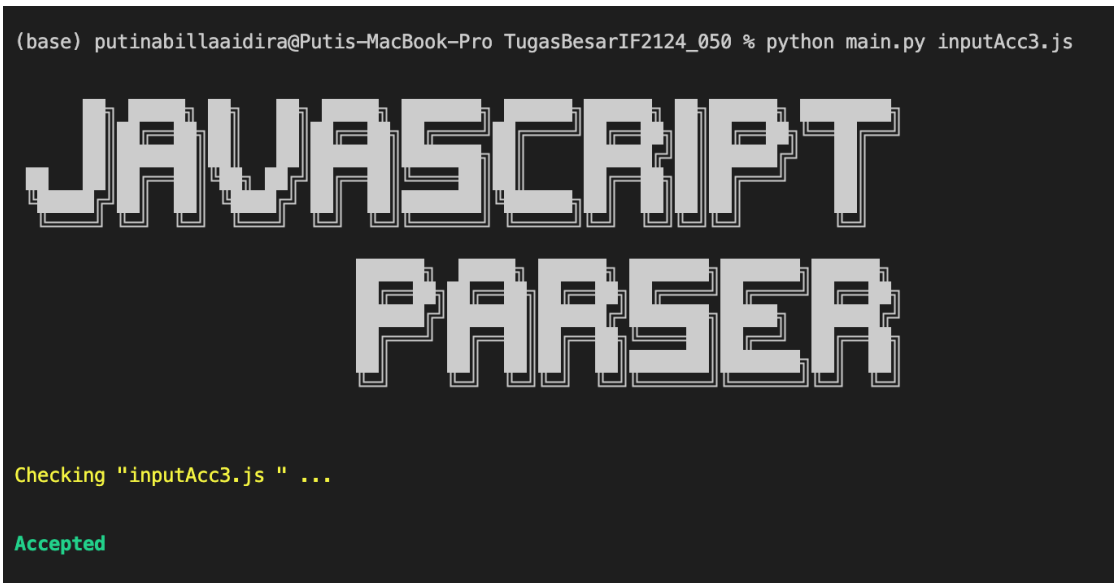


Analisis: Semua syntax sesuai.

3.2.7 inputAcc3.js

```
const message = "wooho"
let x = 2
try {
    if(x == ""){
        y = "empty"
    }
    else if (x == false){
        y = "empty"
    }
    y = "empty"
}
catch(err) {
    x = "Input is " + err
}
finally{
    x = true
}
```

Hasil Uji:



Analisis: Semua syntax sesuai.

BAB IV

DAFTAR REFERENSI

GeeksforGeeks. (2022, August 8). Introduction of Finite Automata. https://www.geeksforgeeks.org/introduction-of-finite-automata/

GeeksforGeeks. (2020, June 22). CYK Algorithm for Context Free Grammar. https://www.geeksforgeeks.org/cyk-algorithm-for-context-free-grammar/

Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2007). Introduction to Automata Theory, Languages, and Computation. Pearson/Addison Wesley.

JavaScript reference - JavaScript | MDN. (2022, October 31). https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference

JavaScript Variables. (n.d.). https://www.w3schools.com/js/js_variables.asp

PENYEDERHANAAN CONTEXT FREE GRAMMAR. (2018, December 20). School of Computer Science. https://socs.binus.ac.id/2018/12/20/penyederhanaan-context-free-grammar/

Tim Dosen Prodi Teknik Informatika STEI ITB. Bahan Kuliah IF2124 Teori Bahasa Formal dan Automata.

Trivusi. (2022, September 17). Finite State Automata: Pengertian, Cara Kerja, dan Jenis-jenisnya. https://www.trivusi.web.id/2022/08/finite-state-automata.html

Wahyu, R. (2022, January 4). Data Structure in Python (Struktur Data dalam Python). Medium. https://medium.com/@rismitawahyu/data-structure-in-python-struktur-data-dalam-python-b4b769c37f87

Link Repository :

https://github.com/Putinabillaa/TugasBesarIF2124_050.git

BAB V

PEMBAGIAN TUGAS

SPESIFIKASI		PIC
PROGRAM		
	Pre-Process	13521077 Husnia Munzayana
	FA Variable	13521077 Husnia Munzayana
	FA Bilangan/Angka	13521077 Husnia Munzayana
	FA Ekspresi	13521050 Naufal Syifa Firdaus
	Grammar	13521088 Puti Nabilla Aidira
	CNF Converter	13521088 Puti Nabilla Aidira
	CYK	13521088 Puti Nabilla Aidira
LAPORAN		
	BAB I	13521050 Naufal Syifa Firdaus 13521077 Husnia Munzayana 13521088 Puti Nabilla Aidira
	BAB II	13521050 Naufal Syifa Firdaus 13521077 Husnia Munzayana 13521088 Puti Nabilla Aidira
	BAB III	13521050 Naufal Syifa Firdaus 13521077 Husnia Munzayana 13521088 Puti Nabilla Aidira
	BAB IV	13521077 Husnia Munzayana
	BAB V	13521077 Husnia Munzayana