Laporan Tugas Besar Mata Kuliah Teori Bahasa dan Automata

Python While-do Parser and Lexical Analyzer

Disusun Oleh :

IF-45-08

Muhammad Naufal Hawari	1301213069
Reza Muammar Widyanto	1301210513
Deaz Setyo Nugroho	1301210248

BAB 1. CONTEXT FREE GRAMMAR

Pada tugas besar ini kami membuat parser dan lexical analyzer untuk *syntax* while-do pada Bahasa pemrograman Python. Berikut adalah CFG nya:

statement → while <kondisi> : <aksi>

kondisi → <variabel> <comparator> <variabel> | True | False

aksi → <variabel> = <variabel> <operator> <variabel> | print <variabel>

variabel → x | y

comparator → is | in | == | > | >= | < | <= | !=

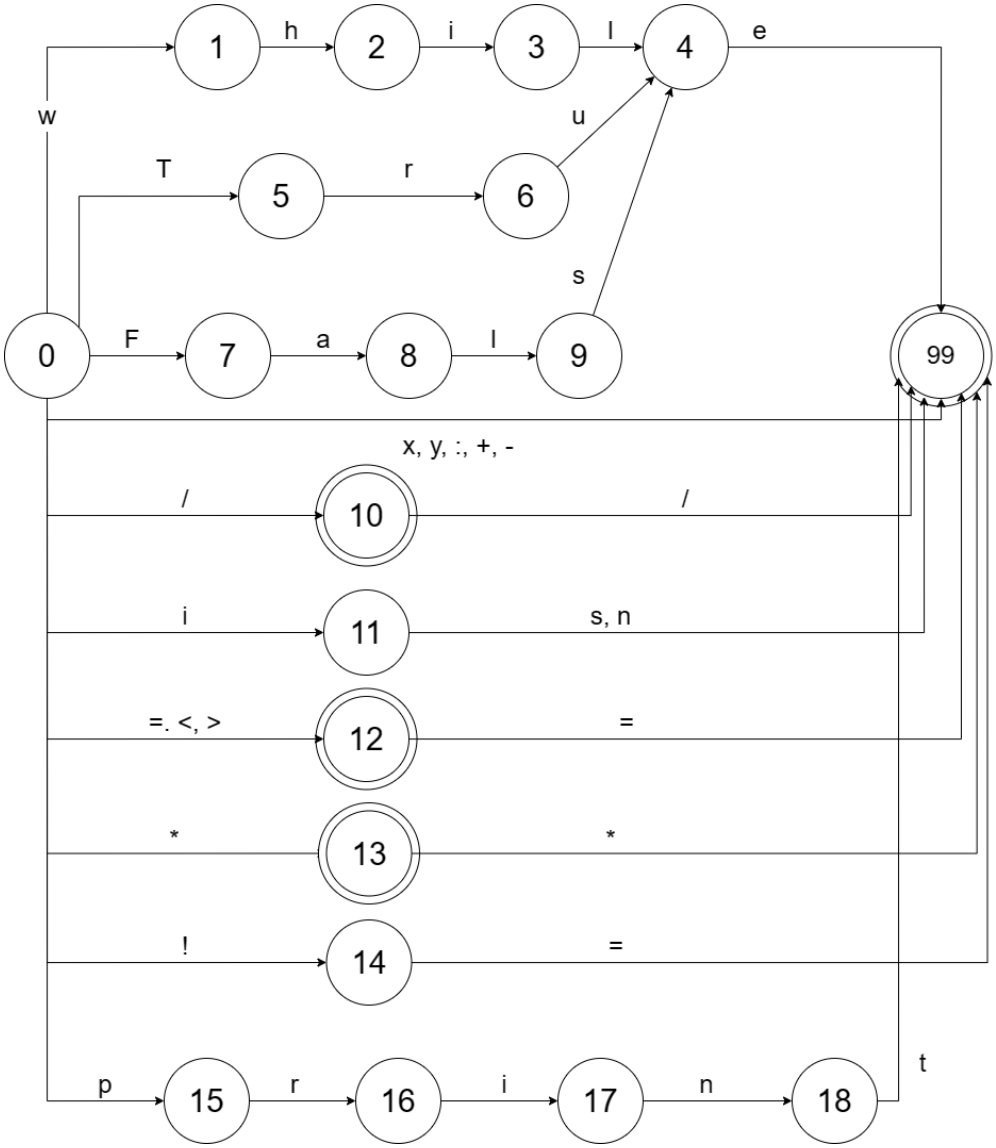
operator → + | - | * | ** | / | //

BAB 2. LEXICAL ANALYZER

1.1 Finite Automata

Berikut adalah diagram finite automata untuk *lexical analyzer* terhadap variabel-variabel terminal.

Gambar 1. Finite Automata untuk Lexical Analyzer



BAB 3. PARSE TABLE

Berikut adalah isi dari *parse table* berdasarkan CFG yang sudah dibuat di BAB 1.

Tabel 1. Parse Table CFG

	while	:	True	False	=	print	x	y	is	in
statement	while <kondisi> : aksi	error	error	error	error	error	error	error	error	error
kondisi	error	error	True	False	error	error	<variabel> <comparator> <variabel>	<variabel> <comparator> <variabel>	error	error
aksi	error	error	error	error	error	print <variabel>	<variabel> = <variabel> <operator> <variabel>	<variabel> = <variabel> <operator> <variabel>	error	error
variabel	error	error	error	error	error	error	x	y	error	error
comparator	error	error	error	error	error	error	error	error	is	in
operator	error	error	error	error	error	error	error	error	error	error

	==	!=	>	>=	<	<=	+	-	/	//	*	**
statement	error	error	error	error	error	error	error	error	error	error	error	error
kondisi	error	error	error	error	error	error	error	error	error	error	error	error
aksi	error	error	error	error	error	error	error	error	error	error	error	error
variabel	error	error	error	error	error	error	error	error	error	error	error	error
comparator	==	!=	>	>=	<	<=	error	error	error	error	error	error
operator	error	error	error	error	error	error	+	-	/	error	*	**

BAB 4. IMPLEMENTASI

Implementasi finite automata dan parse table dilakukan menggunakan bahasa pemrograman JavaScript. Penerimaan teks input dilakukan melalui file html. Link kode program dan cara menjalankan program ada di bagian lampiran laporan ini.

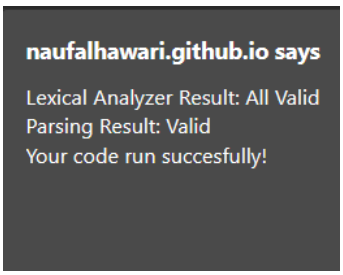
```javascript
function lexicalAnalyzer(terminal) {
    i = 0
    state = 0

    while (i < terminal.length) {
        switch (state) {
            case 0:
                if (terminal[i] === 'w') {
                    state = 1;
                }
                else if (terminal[i] === 'T') {
                    state = 5;
                }
                else if (terminal[i] === 'F') {
                    state = 7;
                }
                else if (terminal[i] === '/') {
                    state = 10;
                }
                else if (terminal[i] === 'i') {
                    state = 11;
                }
                else if (terminal[i] === '=' || terminal[i] === '>' || terminal[i] === '<') {
                    state = 12;
                }
                else if (terminal[i] === '*') {
                    state = 13;
                }
```

Cuplikan Fungsi Lexical Analyzer

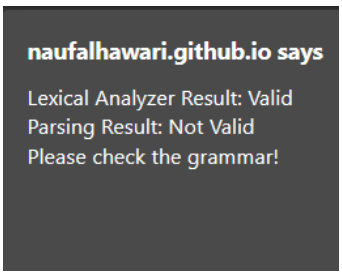
```javascript
function parser(code) {
    let stack = [];

    let state = "i";
    stack.push("#");
    state = "p";
    stack.push("statement");
    state = "q";

    let head = 0;
    let symbol = code[head];

    let topOfStack = stack.at(-1);
    while (topOfStack !== "#" && state !== "error") {
        console.log(state)
        console.log(topOfStack);
        switch (topOfStack) {
            case "statement":
                if (symbol === "while") {
                    stack.pop("statement")
                    stack.push("aksi");
                    stack.push(":");
                    stack.push("kondisi");
                    stack.push("while");
                }
                else {
                    state = "error";
                }
                break;
```

Cuplikan Fungsi Parser

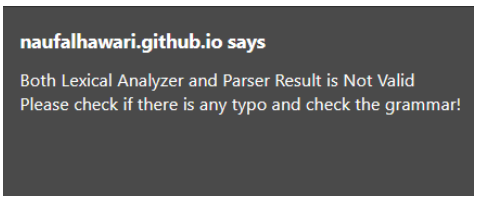
Output program terdiri atas tiga jenis. Jenis pertama adalah input sesuai dengan *grammar* yang sudah ditentukan dan masing-masing variabel terminal sudah sesuai dengan finite automata. Output jenis ini akan memberitahukan bahwa hasil parser adalah valid dan hasil lexical analyzer adalah valid. Jenis kedua adalah masing-masing variabel terminal sesuai dengan finite automata, tetapi tidak sesuai dengan grammar yang telah ditentukan. Output jenis ini akan memberikan informasi bahwa hasil lexical analyzer adalah valid, tetapi hasil parser tidak valid. Jenis ketiga adalah input yang tidak memenuhi grammar dan tidak memenuhi finite automata. Output jenis ini adalah hasil lexical analyzer dan hasil parser tidak valid.



Output Jenis Pertama



Output Jenis Kedua

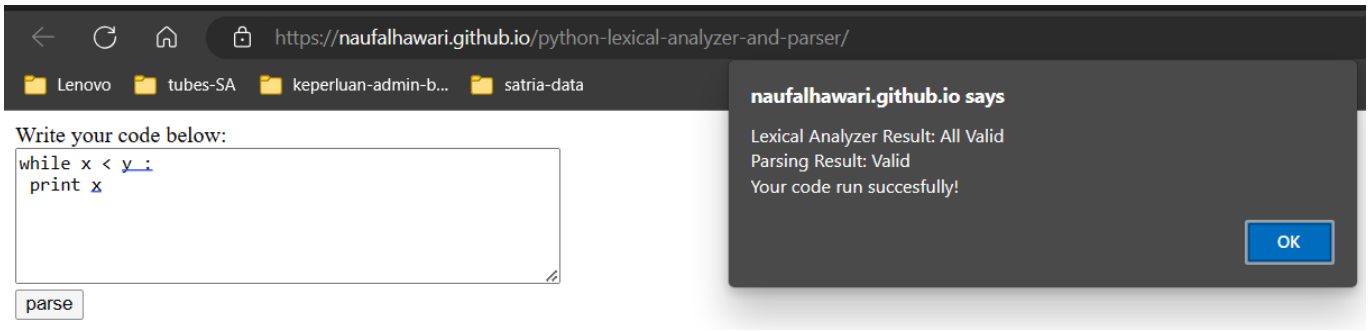


Output Jenis Ketiga

BAB 5. HASIL IMPLEMENTASI

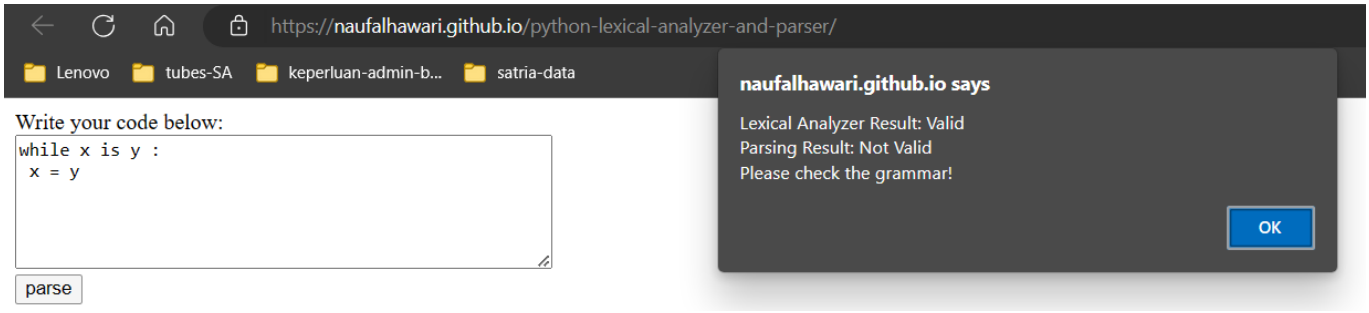
Berikut adalah contoh output jenis pertama. Kode yang dituliskan sudah sesuai dengan *grammar* yang terdefinisi dan masing-masing variabel terminal memenuhi finite automata.

Gambar 2. Contoh Output Jenis Pertama



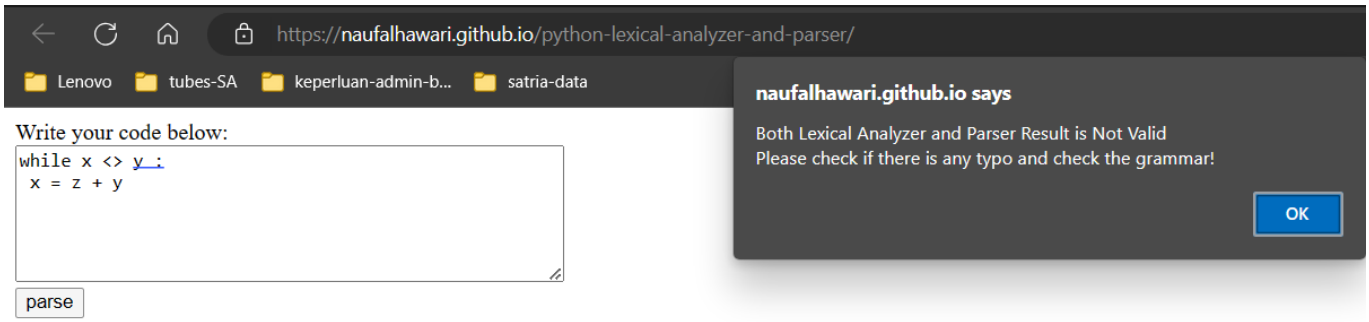
Berikut adalah contoh output jenis kedua ketika semua variabel terminal memenuhi lexical analyzer, tetapi terdapat kesalah grammar. Kesalahan grammar terdapat pada aturan aksi. Aksi seharusnya terdiri atas <variabel> = <variabel> <operator> <variabel>, tetapi aksi pada input hanya terdiri atas <variabel> = <variabel>.

Gambar 3. Contoh Output Jenis Pertama



Berikut adalah contoh output hasil ketiga ketika hasil grammar dan lexical analyzer tidak memenuhi karena tidak ada variabel terminal“<>” dan tidak ada variabel terminal “z”.

Gambar 4. Contoh Output Jenis Pertama



BAB 6. LAMPIRAN

Kode program dan cara menjalankan program dapat dilihat di link berikut:

https://github.com/naufalhawari/python-lexical-analyzer-and-parser

***Cara menjalankan program dapat dilihat melalui README.md**