Brute Force & Recursive Descent Parsing

Ver 0.1

Kelompok: Muhammad Irfan Trianto Putra - 09021281621046

Dwi Novitasari – 09021181621032

Mata Kuliah: Teknik Kompilasi

Dosen: Kanda Januar Miraswan

A. TEST CASE

1. Brute Force Parsing

a.

```
String awal: i-i
E jalan
E1 jalan
T jalan
T1 jalan
E3 jalan
E jalan
E1 jalan
T jalan
T1 jalan
i-i : string diterima
```

b.

```
String awal: i/i
E jalan
E1 jalan
T jalan
T1 jalan
T2 jalan
T3 jalan
T jalan
T1 jalan
i/i : string diterima
```

c.

```
String awal: i/i-i
E jalan
E1 jalan
T jalan
T1 jalan
T2 jalan
T3 jalan
T jalan
T1 jalan
E3 jalan
E jalan
E1 jalan
T jalan
T1 jalan
i/i-i : string diterima
```

d.

```
String awal: i-
E jalan
E1 jalan
T jalan
T1 jalan
E3 jalan
E jalan
E1 jalan
T jalan
T1 jalan
T2 jalan
T3 jalan
E2 jalan
E3 jalan
T2 jalan
T3 jalan
E2 jalan
E3 jalan
i- : string ditolak
```

e.

```
String awal: -i
E jalan
E1 jalan
T jalan
T1 jalan
T2 jalan
T3 jalan
E2 jalan
E3 jalan
-i : string ditolak
```

f.

```
String awal: i**
E jalan
E1 jalan
T jalan
T1 jalan
T2 jalan
T jalan
T1 jalan
T2 jalan
T3 jalan
E2 jalan
E3 jalan
i** : string ditolak
```

g.

```
String awal: **
E jalan
E1 jalan
T jalan
T1 jalan
T2 jalan
T3 jalan
E2 jalan
E3 jalan
** : string ditolak
```

h.

```
String awal: i-i/i+i*i
E jalan
E1 jalan
T jalan
T1 jalan
E3 jalan
E jalan
E1 jalan
T jalan
T1 jalan
T2 jalan
T3 jalan
T jalan
T1 jalan
E2 jalan
E jalan
E1 jalan
T jalan
T1 jalan
T2 jalan
T jalan
T1 jalan
i-i/i+i*i : string diterima
```

i.

```
E jalan
E1 jalan
T jalan
T1 jalan
E3 jalan
E jalan
E1 jalan
T jalan
T1 jalan
T2 jalan
T3 jalan
T jalan
T1 jalan
E2 jalan
E jalan
E1 jalan
T jalan
T1 jalan
T2 jalan
T jalan
T1 jalan
T2 jalan
T3 jalan
E2 jalan
E3 jalan
T2 jalan
T3 jalan
E2 jalan
E3 jalan
T2 jalan
T3 jalan
E2 jalan
E3 jalan
i-i/i+i*ii : string ditolak
```

j.

```
String awal: ii-i/i+i*i
E jalan
E1 jalan
T jalan
T1 jalan
T2 jalan
T3 jalan
E2 jalan
E3 jalan
ii-i/i+i*i : string ditolak
```

2. Recursive Descent Parsing

a.

```
string awal: i-i
stack sisa:  i-iZ
token:  i-
pop()
stack sisa:  iZ
token:  iZ
pop()
string diterima
```

b.

```
string awal: i/i
stack sisa:  i/iZ
token:  i/
pop()
stack sisa:  iZ
token:  iZ
pop()
string diterima
```

c.

```
string awal: i/i-i
stack sisa:  i/i-iZ
token:  i/
pop()
stack sisa:  i-iZ
token:  i-
pop()
stack sisa:  iZ
token:  iZ
pop()
string diterima
```

d.

```
string awal: i-
stack sisa:  i-Z
token:  i-
pop()
stack sisa:  Z
token:  Z
string ditolak
```

e.

```
string awal: -i
stack sisa:  -iZ
token:  -i
string ditolak
```

f.

```
string awal: i**
stack sisa:  i**Z
token:  i*
pop()
stack sisa:  *Z
token:  *Z
string ditolak
```

g.

```
string awal: **
stack sisa:  **Z
token:  **
string ditolak
```

h.

```
string awal: i-i/i+i*i
stack sisa:  i-i/i+i*iZ
token:  i-
pop()
stack sisa:  i/i+i*iZ
token:  i/
pop()
stack sisa:  i+i*iZ
token:  i+
pop()
stack sisa:  i*iZ
token:  i*
pop()
stack sisa:  iZ
token:  iZ
pop()
string diterima
```

i.

```
string awal: i-i/i+i*ii
stack sisa:  i-i/i+i*iiZ
token:  i-
pop()
stack sisa:  i/i+i*iiZ
token:  i/
pop()
stack sisa:  i+i*iiZ
token:  i+
pop()
stack sisa:  i*iiZ
token:  i*
pop()
stack sisa:  iiZ
token:  ii
string ditolak
```

j.

```
string awal: ii-i/i+i*i
stack sisa:  ii-i/i+i*iZ
token:  ii
string ditolak
```

B. SOURCE CODE

1. Brute Force Parsing

```
//ATURAN PRODUKSI:
// E -> T | T+E | T-E
// T -> i | i*T | i/T

string = 'i+i';
index = 0;
function cek(char)
{
	if(char == string.charAt(index))
	{
		index++;
		return true;
	}
	return false;
}
function back(idx)
{
	index = idx;
}
function save()
{
	return index;
}
function E()
{
	console.log('E jalan');
	saveE = save();
	return (back(saveE),E1())
		|| (back(saveE),E2())
		|| (back(saveE),E3());
}
function E1()
{
	console.log('E1 jalan');
	return T();
}
function E2()
```

```
{
	console.log('E2 jalan');
	if(string.charAt(index-1) == 'i')
		return cek('+') && E();
	return false
}
function E3()
{
	console.log('E3 jalan');
	if(string.charAt(index-1) == 'i')
		return cek('-') && E();
	return false;
}
function T()
{
	console.log('T jalan');
	saveT = save();
	return (back(saveT),T1()) ||
		(back(saveT),T2()) ||
		(back(saveT),T3());
}
function T1()
{
	console.log('T1 jalan');
	if( cek('i'))
	{
		head = string.charAt(index);

		if((index == string.length))
			return true;
		else if(head == '+')
			return E2();
		else if(head == '-')
			return E3();
	}
	return false;
}
function T2()
{
	console.log('T2 jalan');
```

```
        return cek('i') && cek('*') && T();
}
function T3()
{
        console.log('T3 jalan');
        return cek('i') && cek('/') && T();
}
console.log('String awal:', string);
if( E() && (index == string.length) )
        console.log(string, ': string diterima');
else
        console.log(string, ': string ditolak');
```

2. Recursive Descent Parsing

```
// E -> T | T+E | T-E
// T -> i | i*T | i/T
// Ekspresi Regular: i(+i | -i | *i | /i)*
string = 'i-i/i+i*i*ii'
index = 0;
head1 = string.charAt(index);
head2 = string.charAt(index+1)
stack = string+'Z';
function pop()
{
        console.log('pop()')
        stack = stack.slice(2);
}
function E()
{
        head1 = stack.charAt(index);
        head2 = stack.charAt(index+1);
        token = head1+head2;
        console.log('stack sisa: ',stack);
        console.log('token: ',token);
        if( (token) == 'iZ')
        {
                pop();
                console.log('string diterima');
                return
```

```
        }
        else if ( (token) == 'i*')
                pop();
        else if ( (token) == 'i/')
                pop();
        else if ( (token) == 'i+')
                pop();
        else if ( (token) == 'i-')
                pop();
        else
        {
                console.log('string ditolak');
                return;
        }
        if(stack != '')
                return E();
}
console.log('string awal:', string);
E();
```