**Table-Driven Parser for Balanced Parentheses**

**First Grammar**

It has three alternatives:

S ::= "" | ( S ) | S S

1. 2. 3.

Goal is to figure out which alternative to use when – convert | into if-then-else.

Compute:

nullable = { S }

first(S) = { ( }

follow(S) = { ( , ) }

Parse Table:

|  |  |  |
| --- | --- | --- |
|  | **(** | **)** |
| **S** | 1,2,3 |  |

Because we have duplicate entries, we cannot use this to build a parser.

**Second Grammar**

S ::= "" | F S

F ::= ( S )

Again compute:

nullable = { S }

first(S) = { ( }

follow(S) = { ) }

|  |  |  |
| --- | --- | --- |
|  | **(** | **)** |
| **S** | 2 | 1 |
| **F** | 1 | {} |

**Recursive Descent Parser**

Constructed mechanically:

[def](http://scala-lang.org) S = {

[if](http://scala-lang.org) (lexer.token==OpenP) { F; S }

[else](http://scala-lang.org) [if](http://scala-lang.org) (lexer.token==ClosedP) ()

[else](http://scala-lang.org) error("Expected '(' or ')'")

}

[def](http://scala-lang.org) F = {

[if](http://scala-lang.org) (lexer.token==OpenP) {

lexer.next

S

skip(ClosedP)

} [else](http://scala-lang.org) error("Expected '('")

}

Simplified:

[def](http://scala-lang.org) S = [if](http://scala-lang.org) (lexer.token==OpenP) { F; S }

[def](http://scala-lang.org) F = { skip(OpenP); S; skip(ClosedP) }

**Top-Down Parser Using a Stack**

stack push "S"

[while](http://scala-lang.org) (!stack empty) {

[val](http://scala-lang.org) X = stack pop

[if](http://scala-lang.org) (X isTerminal) skip(X)

[else](http://scala-lang.org) {

[if](http://scala-lang.org) (X=="S") {

[if](http://scala-lang.org) (lexer.token==OpenP) stack push "S" push "F"

} [else](http://scala-lang.org) [if](http://scala-lang.org) (X=="F") stack push ClosedP push "S" push OpenP

}

}