NULLABLE(terminal) = false
NULLABLE(sequence of symbols) = true iff none of the symbols are NOT NULLABLE
NULLABLE(production) = NULLABLE(RHS)
NULLABLE(nonterminal) = any of its productions are NULLABLE
FIRST(terminal) = \{ terminal \}
FIRST(X₁ X₂ ... Xₙ) = union of FIRST(X₁), FIRST(X₂), ..., FIRST(Xₖ)
    where X₁, X₂, ... Xₖ₋₁ are NULLABLE and Xₖ is NOT NULLABLE
FIRST(production) = FIRST(RHS)
FIRST(nonterminal) = union of FIRST of RHS of its productions

grammar is LL(\(k\)) iff we can create an LL(\(k\)) parser
language is LL(\(k\)) iff \(\exists\) an LL(\(k\)) grammar for it
context-free language context-free grammars
for each production P in the grammar {
    A = lhs(P)
    for each terminal a in FIRST(P) {
        M[A, a] = P
        if NULLABLE(P) { // equivalently, epsilon in FIRST(P)
            for each terminal b in FOLLOW(A) {
                M[A, b] = P
            }
        }
    }
}

for each production P in the grammar {
    A = lhs(P)
    for each terminal a in FIRST(P) {
        M[A, a] = P
        if NULLABLE(P) { // equivalently, epsilon in FIRST(P)
            for each terminal b in FOLLOW(A) {
                M[A, b] = P
            }
        }
    }
}
// X refers to the top of the stack
// a refers to the next input token
while X != $ {
    if X is a terminal {
        if X == a {
            pop_stack();
            advance_input();
        } else {
            error();
        }
    } else { // X is a nonterminal
        if M[X, a] is empty {
            error();
        } else {
            P = M[X, a];
            output(P);
            pop_stack();
            for Yi in reverse(rhs(P)) {
                push(Yi);
            }
        }
    }
} return a == $;

if_stmt → if ( expr ) { statements } else { statements }