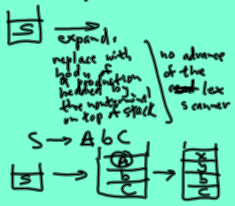
Top-down parsing

stack for the parser:

Given grammar G
start symbol S



$S \to A\,b\,C$

$A \to x\,y$

match move: top of stack is a terminal which must match the next token in the input ("lookahead")
if not, syntax error
if match, pop token, advance lookahead

Initially: start symbol on stack
while stack not empty:
if top is nonterminal, expand by some production labeled by that nonterminal (which production: lookahead tells you)
else if top is token, then match
(also syntax error if no more input)

end-while
if not at end of input, also syntax error

Between actions, take a snapshot of

input (largely read) | stack contents (top down)

follows leftmost derivation of input (forwards)

S
$A\,b\,C$
$x\,y\,b\,C$
$x\,y\,b\,\varepsilon$
$x\,y\,b$ $\quad C \to \varepsilon$
$x\,y\,b$

LL parsing — Left-to-right reading; Leftmost derivation (forward)

LL(1) — LL parsing with one token of lookahead

Top-down parsing
advantage: much easier to implement
disadvantage: only certain grammars are suitable

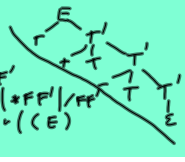
$E \to E + T \mid E - T \mid T$
$T \to T * F \mid T / F \mid F$
$F \to c \mid v \mid (E)$

$2 + 4 * 5$ $\quad E$

$E \to T \pm T \pm T \pm T \pm T$ (E)

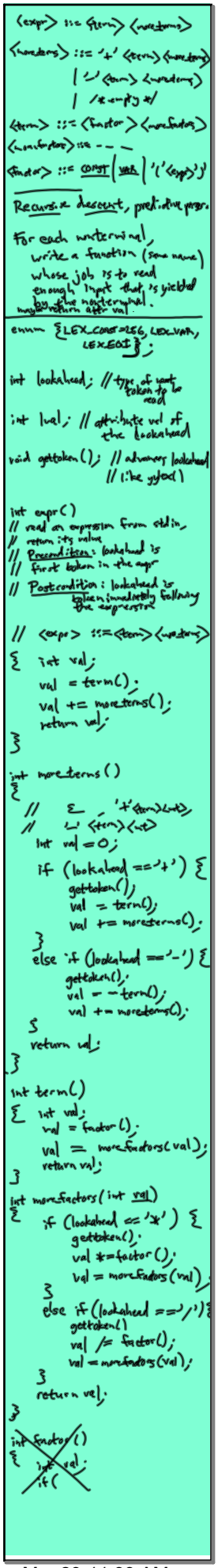
Equivalent LL(1) grammar:

$E \to T\ T'$
$T' \to \varepsilon \mid +\,T\,T' \mid -\,T\,T'$



$T \to F\,F'$
$F' \to \varepsilon \mid *\,F\,F' \mid /\,F\,F'$
$F \to c \mid v \mid (\,E\,)$

Mar 23-10:58 AM



Mar 23-11:33 AM

```
int factor()
{   int val;
    if (lookahead == LEX_CONST) {
        val = lval;
        gettoken();
        return val;
    }
    if (lookahead == LEX_VAR){
        val = get_val(lval);
        gettoken();
        return val;
    }
    if (lookahead == '(') {
        gettoken();
        val = expr();
        if (lookahead != ')')
            error();
        else {
            gettoken();
            return val();
        }
    }
    error("~~left paren expected~~ illegal first token of factor")
}

int main()
{   int val;
    gettoken(); //initial:lookahead
    val = expr();
    if (lookahead != LEX_EOI)
        error("extra token ~~or~~ after expr");
    printf("expr = %d\n", val);
}
```

"right paren expected"