Consider consider consider consider the following subgrammar:

```plaintext
| T -> T . mult_op F       | OW reduce (by 3, recognizing E) |
| F -> . ID               | on ID shift and reduce (by 1, recognizing F) |
| T -> T mul_op . F       | on F shift and reduce (by 3, recognizing T) |
| T -> . F                | on F shift and reduce (by 1, recognizing T) |
```

This is part of a context-free grammar (CFG) for a calculator language. The grammar defines the syntax of calculator expressions and statements.

The grammar consists of productions that define the syntax of calculator expressions and statements. Each production rules a particular structure of tokens in the input.

- **T**: Term, which represents a factor or a term followed by a multiplier or divider.
- **F**: Factor, which represents a single number.
- **ID**: Identifier, which represents a variable name.
- **stmt_list**: List of statements.
- **WRITE**: Output operation.
- **term**: Mathematical term, which can be a factor or a term followed by a multiplier or divider.
- **finite number of symbols onto/off the stack**: Indicates the stack operations.
- **PDAs**: Pushdown automata.
- **return**
- **cost of deleting additional tokens exceeds the cost of**: Modified the grammar production.
- **most practical languages are at least close**: Indicates the complexity of building a parser.
- **token**: Input symbol.
- **prefix :=**
- **- common operators have lower cost than uncommon operators in**: Indicates the priority of operators.
- **- locally optimal**: Indicates the strategy for selecting operators.
- **- modifying the input stream and re-starting parsing**: Indicates the action when an error is detected.
- **add FIRST(Xi) to return_value**: Adds symbols to the return value.
- **<inner> for i in 1..k**: Indicates a loop construct.
- **Fig. 2.22 shows the "obvious" facts in the calculator grammar**: Indicates a specific example.
- **valid program, and one possible RHS is epsilon**: Indicates a valid grammar rule.
- **to predict epsilon productions**: Indicates the need forε productions.
- **FOLLOW, to predict epsilon productions**: Indicates the inclusion of FOLLOW sets.
- **a parser is based on a pushdown automaton**: Indicates the type of automaton.
- **order to parse top-down left-to-right, we end up with a tree that tends**: Indicates the parsing strategy.
- **gcc switched from bottom-up to top-down parsing around 2005**: Indicates a specific parser change.
- **Next lecture: algorithms for top-down vs bottom-up parsing.**: Indicates the upcoming lecture topic.

The grammar and parser examples illustrate the construction of a calculator program and the process of parsing calculator expressions. The grammar rules define the syntax, while the parser processes the input to generate the corresponding tree structure.