School of CET

System Software and Compiler lab

TY BTech CSE

Assignment No.6

Title: Implementing recursive descent parser for sample language.

Aim: Implement Recursive Descent parser for given grammar.

Objective:

1. To study parsing phase in the compiler.
2. To study types of parsers – top down and bottom up.
3. Problems encountered during top down parser.
4. How to write a top down parser.

Theory: Write in brief for following:

1. CFG, non-terminals, terminals, productions, derivation sequence-

A context-free grammar (CFG) consisting of a finite set of grammar rules is a quadruple (N, T, P, S) where

- N is a set of non-terminal symbols.
- T is a set of terminals where $N \cap T = NULL$.
- P is a set of rules, P: $N \rightarrow (N \cup T)^*$, i.e., the left-hand side of the production rule P does have any right context or left context.
- S is the start symbol.

Example

The grammar ({A}, {a, b, c}, P, A), P : $A \rightarrow aA$, $A \rightarrow abc$.

The grammar ({S, a, b}, {a, b}, P, S), P: $S \rightarrow aSa$, $S \rightarrow bSb$, $S \rightarrow \varepsilon$

The grammar ({S, F}, {0, 1}, P, S), P: S → 00S | 11F, F → 00F | ε

A context-free grammar has four components:

- A set of non-terminals (V). Non-terminals are syntactic variables that denote sets of strings. The non-terminals define sets of strings that help define the language generated by the grammar.
- A set of tokens, known as terminal symbols (Σ). Terminals are the basic symbols from which strings are formed.
- A set of productions (P). The productions of a grammar specify the manner in which the terminals and non-terminals can be combined to form strings. Each production consists of a non-terminal called the left side of the production, an arrow, and a sequence of tokens and/or on- terminals, called the right side of the production.
- One of the non-terminals is designated as the start symbol (S); from where the production begins.

The strings are derived from the start symbol by repeatedly replacing a non-terminal (initially the start symbol) by the right side of a production, for that non-terminal.

2. Introduction to RDP:

Recursive descent is a top-down parsing technique that constructs the parse tree from the top and the input is read from left to right. It uses procedures for every terminal and non-terminal entity. This parsing technique recursively parses the input to make a parse tree, which may or may not require back-tracking. But the grammar associated with it (if not left factored) cannot avoid back-tracking. A form of recursive-descent parsing that does not require any back-tracking is known as predictive parsing.
This parsing technique is regarded recursively as it uses context-free grammar which is recursive in nature.

Back-tracking

Top- down parsers start from the root node (start symbol) and match the input string against the production rules to replace them (if matched). To understand this, take the following example of CFG:
S → rXd | rZd
X → oa | ea
Z → ai
For an input string: read, a top-down parser, will behave like this:
It will start with S from the production rules and will match its yield to the left-most letter of the input, i.e. ‘r’. The very production of S (S → rXd) matches with it. So the top-down parser advances to the next input letter (i.e. ‘e’). The parser tries to expand non-terminal ‘X’ and checks its production from the left (X → oa). It does not match with the next input symbol. So the top-down parser backtracks to obtain the next production rule of X, (X → ea).

Now the parser matches all the input letters in an ordered manner. The string is accepted.

3. Elimination of Left recursion

A production of grammar is said to have left recursion if the leftmost variable of its RHS is the same as the variable of its LHS.
A grammar containing a production having left recursion is called as Left Recursive Grammar.

Example-

$S \rightarrow Sa / \in$
(Left Recursive Grammar)

- Left recursion is considered to be a problematic situation for Top down parsers.
- Therefore, left recursion has to be eliminated from the grammar.

Elimination of Left Recursion

Left recursion is eliminated by converting the grammar into a right recursive grammar.

If we have the left-recursive pair of productions-
$A \rightarrow A\alpha / \beta$
(Left Recursive Grammar)
where β does not begin with an A.

Then, we can eliminate left recursion by replacing the pair of productions with-
$A \rightarrow \beta A'$
$A' \rightarrow \alpha A' / \in$
(Right Recursive Grammar)

This right recursive grammar functions same as left recursive grammar.

Input: String satisfying given grammar, string not satisfying given grammar to test error condition.

Output: Success for correct string, Failure for syntactically wrong string.

Conclusion: The recursive descent parser is successfully implemented.

Platform: Linux (C/C++/JAVA)

FAQ's:

1. What is Parsing?

Parsing, syntax analysis, or syntactic analysis is the process of analyzing a string of symbols, either in natural language, computer languages or data structures, conforming to the rules of a formal grammar.
A parser is a software component that takes input data (frequently text) and builds a data structure – often some kind of parse tree, abstract syntax tree or other hierarchical structure, giving a structural representation of the input while checking for correct syntax. The parsing may be preceded or followed by other steps, or these may be combined into a single step. The parser is often preceded by a separate lexical analyser, which creates tokens from the sequence of input characters; alternatively, these can be combined in scannerless parsing. Parsers may be programmed by hand or may be automatically or semi-automatically generated by a parser generator. Parsing is complementary to templating, which produces formatted output. These may be applied to different domains, but often appear together, such as the scanf/printf pair, or the input (front end parsing) and output (back end code generation) stages of a compiler.

2. What are the different types of Parser?

Syntax analyzers follow production rules defined by means of context-free grammar. The way the production rules are implemented (derivation) divides parsing into two types : top-down parsing and bottom-up parsing.

Top-down Parsing

When the parser starts constructing the parse tree from the start symbol and then tries to transform the start symbol to the input, it is called top-down parsing.

- Recursive descent parsing : It is a common form of top-down parsing. It is called recursive as it uses recursive procedures to process the input. Recursive descent parsing suffers from backtracking.
- Backtracking : It means, if one derivation of a production fails, the syntax analyzer restarts the process using different rules of same production. This technique may process the input string more than once to determine the right production.

Bottom-up Parsing

As the name suggests, bottom-up parsing starts with the input symbols and tries to construct the parse tree up to the start symbol.

3. What are the disadvantages of RDP?

Problem #1: operator associativity

The problem is in the grammar definition (BNF) itself. The way the expr rule is defined makes it inherently right-associative instead of left-associative. The hierarchy of the rules implicitly defines their associativity, because it defines what will be grouped together.

Problem #2: efficiency

There's an inherent performance problem with recursive-descent parsers when dealing with expressions. This problem stems from the need to define operator precedence, and in RD parsers the only way to define this precedence is by using recursive sub-rules

4. Why to eliminate the left recursion?

Why Eliminate Left Recursion?
Consider an example: E->E+T/T,
The above example will go in an infinite loop because the function E keeps calling itself which causes a problem for a parser to go in an infinite loop which is a never-ending process.
So to avoid this infinite loop problem we eliminate Left-recursion problem.
Rules to follow to eliminate left recursion

A-->bA'
A'-->eps/aA' //eps stands for epsilon

Therefore solution for above left recursion problem,
E --> TE'
E'-->eps/+TE'