NAME: Adejoh Joshua

DEPARTMENT: Computer Science

METRIC NUMBER: Edu1812151

**COMPILER CONSTRUCTION**

Discuss extensively, the following:

1. Top down and Bottom up parsing.

2. Translation Writing System

3. An operator precedence grammar.

## **Parser**

A parser takes input in the form of a sequence of tokens, interactive commands, or program instructions and breaks them up into parts that can be used by other components in programming.

In order for the code written in human-readable form to be understood by a machine, it must be converted into machine language. This task is usually performed by a translator (interpreter or compiler). The parser is commonly used as a component of the translator that organizes linear text in a structure that can be easily manipulated (parse tree). To do so, it follows a set of defined rules called “grammar”

The overall process of parsing involves three stages:

1. **Lexical Analysis:** A lexical analyzer is used to produce tokens from a stream of input string characters, which are broken into small components to form meaningful expressions. A token is the smallest unit in a programming language that possesses some meaning (such as +, -, \*, “function”, or “new” in JavaScript).
2. **Syntactic Analysis:** Checks whether the generated tokens form a meaningful expression. This makes use of a context-free grammar that defines algorithmic procedures for components. These work to form an expression and define the particular order in which tokens must be placed.
3. **Semantic Parsing:** The final parsing stage in which the meaning and implications of the validated expression are determined and necessary actions are taken.

**Top-down Parsing:**

is a parsing technique that first looks at the highest level of the parse tree and works down the parse tree by using the rules of grammar while.

**Bottom-up Parsing:**

is a parsing technique that first looks at the lowest level of the parse tree and works up the parse tree by using the rules of grammar.

There are some differences present to differentiate these two parsing techniques, which are given below:

|  |  |  |
| --- | --- | --- |
| S.No | Top Down Parsing | Bottom Up Parsing |
| 1. | It is a parsing strategy that first looks at the highest level of the parse tree and works down the parse tree by using the rules of grammar. | It is a parsing strategy that first looks at the lowest level of the parse tree and works up the parse tree by using the rules of grammar. |
| 2. | Top-down parsing attempts to find the left most derivations for an input string. | Bottom-up parsing can be defined as an attempts to reduce the input string to start symbol of a grammar. |
| 3. | In this parsing technique we start parsing from top (start symbol of parse tree) to down (the leaf node of parse tree) in top-down manner. | In this parsing technique we start parsing from bottom (leaf node of parse tree) to up (the start symbol of parse tree) in bottom-up manner. |
| 4. | This parsing technique uses Left Most Derivation. | This parsing technique uses Right Most Derivation. |

**Translation Writing System**

The Translator Writing System (TWS) is a programming language designed to automate programming of tasks involving parsing. TWS can be used advantageously in all text processing problems, from simple investigations of the local characteristics of textual information to writing compilers for general or specialized languages. In the statistical practice it will be suitable for forxnatting-reformatting tasks and text editing purpose. Although TWS is a general programming tool, its use for parsing a language whose grammar is originally in non deterministic form requires much more effort than if the grammar were in deterministic form. Because TWS itself is a compiler the method of self-compilling has been used to implement it. This method also suggested tous that the TWS language could be used to des cribe it syntax in this manual. This decision made the manual more instructive,, but on the other hand, it forces the user to read it more carefully and at least twice.

# **Operator-precedence grammar**

Technically, an operator precedence grammar is a [context-free grammar](https://en.wikipedia.org/wiki/Context-free_grammar) that has the property (among others[[1]](https://en.wikipedia.org/wiki/Operator-precedence_grammar" \l "cite_note-1)) that no production has either an empty right-hand side or two adjacent nonterminals in its right-hand side. These properties allow precedence [relations](https://en.wikipedia.org/wiki/Relation_(mathematics)) to be defined between the terminals of the grammar. A [parser that exploits these relations](https://en.wikipedia.org/wiki/Operator-precedence_parser) is considerably simpler than more general-purpose parsers such as [LALR parsers](https://en.wikipedia.org/wiki/LALR_parser). Operator-precedence parsers can be constructed for a large class of context-free grammars.

Operator precedence grammar is kinds of shift reduce parsing method. It is applied to a small class of operator grammars.

A grammar is said to be operator precedence grammar if it has two properties:

* No R.H.S. of any production has a∈.
* No two non-terminals are adjacent.

Operator precedence can only established between the terminals of the grammar. It ignores the non-terminal.

## There are the three operator precedence relations:

a ⋗ b means that terminal "a" has the higher precedence than terminal "b".

a ⋖ b means that terminal "a" has the lower precedence than terminal "b".

a ≐ b means that the terminal "a" and "b" both have same precedence.