Context-free grammar is used to specify the syntax of a language.

One can provide a formal definition of a context free grammar. It is a 4-tuple (V, Σ, S, P)

Where: • V is a finite set of variables;

• Σ is a finite alphabet of terminals;

• S is the start variable; and

• P is the finite set of productions. Each productions has the form $V \rightarrow (V \cup \Sigma)^*$

A grammar derives strings by beginning with the start symbol and repeatedly replacing a nonterminal by the body of a production for that nonterminal.

• The terminal strings that can be derived from the start symbol form the language defined by the grammar.

Parsing is the problem of taking a string of terminals and figuring out how to derive it from the start symbol of the grammar, and if it cannot be derived from the start symbol of the grammar, then reporting syntax errors within the string.

Parsing is one of the most fundamental problems in all of compiling;

A grammar can have more than one parse tree generating a given string of terminals. Such a grammar is said to be ambiguous.

A context-free grammar can be used to help guide the translation of programs.

Grammar-oriented compiling technique known as syntax-directed translation.

A grammar naturally describes the hierarchical structure of most programming language constructs.

An interior node and its children correspond to a production; the interior node corresponds to the head of the production, the children to the body.

A grammar can have more than one parse tree generating a given string of terminals. Such a grammar is said to be ambiguous.

+ <=< * < / } ->Left

^ , = } ->Right

The terminals of the grammar are the symbols.

In parsing tree:-

1. The root is labeled by the start symbol.
2. Each leaf is labeled by a terminal or by
3. Each interior node is labeled by a nonterminal.

The analysis phase of a compiler breaks up a source program into constituent pieces and produces an internal representation for it, called intermediate code.

The synthesis phase translates the intermediate code into the target program.

Analysis is organized around the \syntax" of the language to be compiled. The syntax of a programming language describes the proper form of its programs, while the semantics of the language defines what its programs mean; that is, what each program does when it executes.

A lexical analyzer allows a translator to handle multi character constructs like identifiers, which are written as sequences of characters, but are treated as units called tokens during syntax analysis;

A context-free grammar has four components:

1. A set of terminal symbols, sometimes referred to as \tokens." The terminals are the elementary symbols of the language defined by the grammar.

2. A set of nonterminal, sometimes called \syntactic variables." Each nonterminal represents a set of strings of terminals, in a manner we shall describe.

3. A set of productions, where each production consists of a nonterminal, called the head or left side of the production, an arrow, and a sequence of terminals and/or nonterminal, called the body or right side of the production. The intuitive intent of a production is to specify one of the written forms of a construct; if the head nonterminal represents a construct, then the body represents a written form of the construct.

4. A designation of one of the nonterminal as the start symbol.

We specify grammars by listing their productions, with the productions for the start symbol listed first.

We assume that digits, signs such as < and <=, and boldface strings such as while are terminals. An italicized name is a nonterminal, and any non-italicized name or symbol may be assumed to be a terminal.1

Any italicized name containing two or more characters will continue to represent a nonterminal.

We say a production is for a nonterminal if the nonterminal is the head of the production. A string of terminals is a sequence of zero or more terminals. The string of zero terminals, written as , is called the empty string.2

We shall call these token names terminals, since they appear as terminal symbols in the grammar for a programming language