SNS COLLEGE OF TECHNOLOGY, COIMBATORE –35
(An Autonomous Institution)
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Grammars and BNF Notation

BNF Notation

BNF stands for **Backus-Naur Form**. It is used to write a formal representation of a context-free grammar. It is also used to describe the syntax of a programming language.

BNF notation is basically just a variant of a context-free grammar.

In BNF, productions have the form:

1. Left side → definition

Where leftside ∈ $(V_n \cup V_t)+$ and definition ∈ $(V_n \cup V_t)*$. In BNF, the leftside contains one non-terminal.

We can define the several productions with the same leftside. All the productions are separated by a vertical bar symbol "|".

BNF stands for **Backus Naur Form** notation. It is a formal method for describing the syntax of programming language which is understood as Backus Naur Formas introduced by John Bakus and Peter Naur in 1960. BNF and CFG (Context Free Grammar) were nearly identical. BNF may be a meta-language (a language that cannot describe another language) for primary languages.

For human consumption, a proper notation for encoding grammars intended and called Backus Naur Form (BNF). Different languages have different description and rules but the general structure of BNF is given below –

```
name ::= expansion
```

The symbol ::= means “may expand into” and “may get replaced with.” In some texts, a reputation is additionally called a non-terminal symbol.

- Every name in Backus-Naur form is surrounded by angle brackets, < >, whether it appears on the left- or right-hand side of the rule.

- An expansion is an expression containing terminal symbols and non-terminal symbols, joined together by sequencing and selection.
- A terminal symbol may be a literal like ("+" or "function") or a category of literals (like integer).
- Simply juxtaposing expressions indicates sequencing.
- A vertical bar | indicates choice.

Examples :

```
<expr> ::= <term> "+" <expr>

        |  <term>


<term> ::= <factor> "*" <term>

        |  <factor>


<factor> ::= "(" <expr> ")"

          |  <const>


<const> ::= integer
```

Rules For making BNF :
Naturally, we will define a grammar for rules in BNF –

```
rule → name ::= expansion

name → < identifier >

expansion → expansion expansion

expansion → expansion | expansion

expansion → name

expansion → terminal
```

- We might define identifiers as using the regular expression [-A-Za-z_0-9]+.

- A terminal could be a quoted literal (like "+", "switch" or " "<<=") or the name of a category of literals (like integer).

- The name of a category of literals is typically defined by other means, like a daily expression or maybe prose.

It is common to seek out regular-expression-like operations inside grammars. as an example, the Python lexical specification uses them. In these grammars:

```
postfix * means "repeated 0 or more times"
postfix + means "repeated 1 or more times"
postfix ? means "0 or 1 times"
```

The definition of floating-point literals in Python may be an exemplar of mixing several notations –

```
floatnumber   ::=  pointfloat | exponentfloat
pointfloat    ::=  [intpart] fraction | intpart "."
exponentfloat ::=  (intpart | pointfloat) exponent
intpart       ::=  digit+
fraction      ::=  "." digit+
exponent      ::=  ("e" | "E
```