

Assignment No.3

AIM:

Generate the grammar for 'C' language.

THEORY:

Grammar is defined as

$\langle V, \Sigma, S, P \rangle$

Where V – set of nonterminals

Σ - set of terminals

S – Start symbol

P – Set of productions

For C-language

$S \rightarrow \langle \text{Program} \rangle$

$\Sigma \rightarrow \{a, b, c, \dots, A, B, C, \dots, 0, 1, \dots, \&, \dots\}$

$V \rightarrow \{\langle \text{expression} \rangle, \langle \text{statement} \rangle, \langle \text{var} \rangle, \langle \text{expression} \rangle, \langle \text{include-statement} \rangle\}$

$P \rightarrow \{\langle \text{program} \rangle$

$\langle \text{include_statement} \rangle$

main ()

{

$\langle \text{dec_statement} \rangle,$

$\langle \text{statement} \rangle$

}

}

$\langle \text{include_statement} \rangle \rightarrow \#include \text{ '<' } \langle \text{directive} \rangle \text{ '>'}$

$\langle \text{include_statement} \rangle | ^$

$\langle \text{directive} \rangle \rightarrow \text{stdio.h} | \text{conio.h} | \dots$

$\langle \text{dec_statement} \rangle \rightarrow \langle \text{dd} \rangle \langle \text{dec_statement} \rangle | ^$

$\langle \text{dd} \rangle \rightarrow \langle \text{datatype} \rangle \langle \text{variable} \rangle \langle \text{next-var} \rangle;$

$\langle \text{datatype} \rangle \rightarrow \text{int} | \text{char} | \text{float}$

$\langle \text{next-var} \rangle \rightarrow \langle \text{variable} \rangle \langle \text{next-var} \rangle | ^$

$\langle \text{var} \rangle \rightarrow \langle \text{letter} \rangle \langle \text{next} \rangle$

$\langle \text{next} \rangle \rightarrow \langle \text{letter} \rangle \langle \text{next} \rangle | \langle \text{digit} \rangle \langle \text{next} \rangle | \langle \text{next} \rangle | ^$

$\langle \text{letter} \rangle \rightarrow A | B | C | \dots | a | b | c | \dots | z$

$\langle \text{digit} \rangle \rightarrow 0|1|2|3|\dots\dots\dots|9$
 $\langle \text{variable} \rangle \rightarrow \langle \text{var} \rangle = \langle \text{number} \rangle | \langle \text{var} \rangle = ' \langle \text{letter} \rangle ' | \langle \text{var} \rangle$
 $\langle \text{statement} \rangle \rightarrow \langle \text{printf-statement} \rangle$
 $\quad \quad \quad | \langle \text{scanf-statement} \rangle$
 $\quad \quad \quad | \langle \text{if-statement} \rangle$
 $\quad \quad \quad | \langle \text{for-statement} \rangle$
 $\quad \quad \quad | \langle \text{while-statement} \rangle$
 $\langle \text{if-statement} \rangle \rightarrow \text{if} (\langle \text{c-expression} \rangle$
 $\quad \quad \quad | \text{if} (\langle \text{c-expression} \rangle) \{ \langle \text{statement} \rangle \} \text{ else } \{ \langle \text{statement} \rangle \}$
 $\langle \text{expression} \rangle \rightarrow \langle \text{expression} \rangle + \langle \text{expression} \rangle | \langle \text{var} \rangle | \langle \text{number} \rangle$
 $\langle \text{term} \rangle \rightarrow \langle \text{term} \rangle * \langle \text{term} \rangle | \langle \text{term} \rangle | \langle \text{term} \rangle \langle \text{factor} \rangle$
 $\langle \text{factor} \rangle \rightarrow (\langle \text{expression} \rangle) | \langle \text{expression} \rangle | \langle \text{var} \rangle | \langle \text{number} \rangle$
 $\langle \text{number} \rangle \rightarrow \langle \text{digit} \rangle \langle \text{num} \rangle$
 $\langle \text{num} \rangle \rightarrow \langle \text{digit} \rangle \langle \text{num} \rangle | \langle \text{digit} \rangle \langle \text{digit} \rangle \langle \text{var} \rangle | ^$
 $\langle \text{no} \rangle \rightarrow \langle \text{digit} \rangle \langle \text{no} \rangle | ^$
 $\langle \text{while-statement} \rangle \rightarrow \text{while} (\langle \text{c-expression} \rangle) \{ \langle \text{statement} \rangle \}$
 $\langle \text{for-statement} \rangle \rightarrow \text{for} \langle \text{expression} \rangle ; \langle \text{expression} \rangle ; \langle \text{expression} \rangle$
 $\quad \quad \quad \{ \langle \text{statement} \rangle \}$
 $\langle \text{all-statement} \rangle \rightarrow \langle \text{expression} \rangle$
 $\langle \text{scanf-statement} \rangle \rightarrow \text{scanf} (" \% \langle \text{sp} \rangle \langle \text{next-specifications} \rangle " , \langle \text{var} \rangle$
 $\quad \quad \quad \langle \text{scanf-var} \rangle)$
 $\langle \text{scanf-var} \rangle \rightarrow \& \langle \text{scanf-var} \rangle | ^$
 $\langle \text{next-specifications} \rangle \rightarrow \% \langle \text{sp} \rangle \langle \text{next-specifications} \rangle | ^$
 $\langle \text{sp} \rangle \rightarrow \text{d} | \text{f} | \dots$
 $\langle \text{printf-statement} \rangle \rightarrow \text{printf} (" \langle \text{string} \rangle " , \langle \text{printf-var} \rangle)$
 $\langle \text{string} \rangle \rightarrow \langle \text{any} \rangle$
 $\langle \text{printf-statement} \rangle \rightarrow \langle \text{var} \rangle \langle \text{printf-var} \rangle | ^$
 $\langle \text{any} \rangle \rightarrow 0 \langle \text{any} \rangle | \langle \text{any} \rangle | \dots\dots\dots | \text{a} \langle \text{any} \rangle | \dots\dots\dots | = \langle \text{any} \rangle$

CONCLUSION:

Thus the grammar for C-language is generated

REFERENCES:

- <http://marvin.cs.uidaho.edu/Teaching/CS445/c-Grammar.pdf>