Assignment No : A4

1 Title:

YACC Program.

2 Problem Definition

Parser for sample language using YACC.

3 Learning Objectives:

1. To understand the concept of a parser.
2. To use YACC as a parser generator.

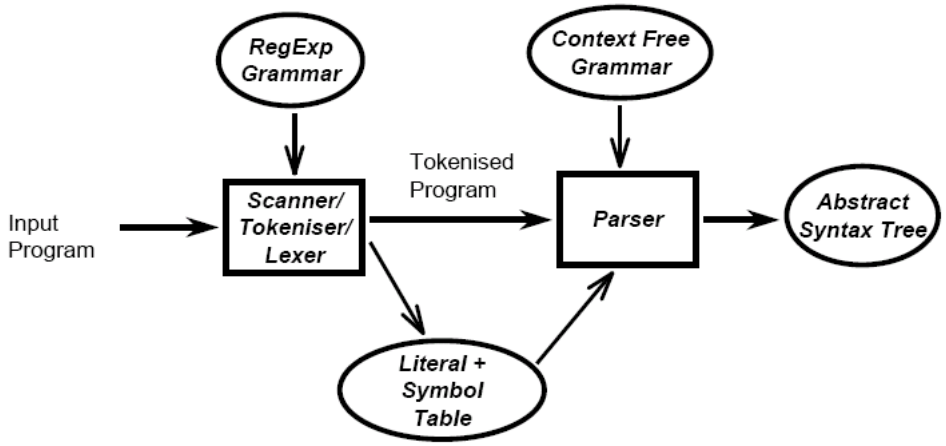
4 S/W and H/W requirements:

1. Open source 64 bit OS.
2. Gedit text editor.
3. bison.

5 Theory

Syntax Analysis:
This is also known as **Parsing**. t takes the token produced by lexical analysis as input and generates a parse tree (or syntax tree). In this phase, token arrangements are checked against the source code grammar, i.e. the parser checks if the expression made by the tokens is syntactically correct.

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.



YACC:
Yacc provides a general tool for imposing structure on the input to a computer program. The Yacc user prepares a specification of the input process; this includes rules describing the input structure, code to be invoked when these rules are recognized, and a low-level routine to do the basic input. Yacc then generates a function to control the input process. This function, called a parser, calls the user-supplied low-level input routine (the lexical analyzer) to pick up the basic items (called tokens) from the input stream. These tokens are organized according to the input structure rules, called grammar rules; when one of these rules has been recognized, then user code supplied for this rule, an action, is invoked; actions have the ability to return values and make use of the values of other actions.

Yacc is written in a portable dialect of C[1] and the actions, and output subroutine, are in C as well. Moreover, many of the syntactic conventions of Yacc follow C.

6 Related Mathematics

Let S be the solution perspective of the given problem.
The set S is defined as:
$S = \{\ s, e, X, Y, F, DD, NDD|_s\}$
Where,
s= Start point
e= End point
F= Set of main functions
DD= set of deterministic data
NDD= set of non deterministic data

X= Input Set.

X = $\{arithematicexpressiontokens, grammearrules\}$.

Y = $\{valid, invalid\}$

s = available expression.
e = expression verified.

F = $\{f_{read}, f_{lex}, f_{match}, f_{ret}\}$

f_{read} :function to read the expression.

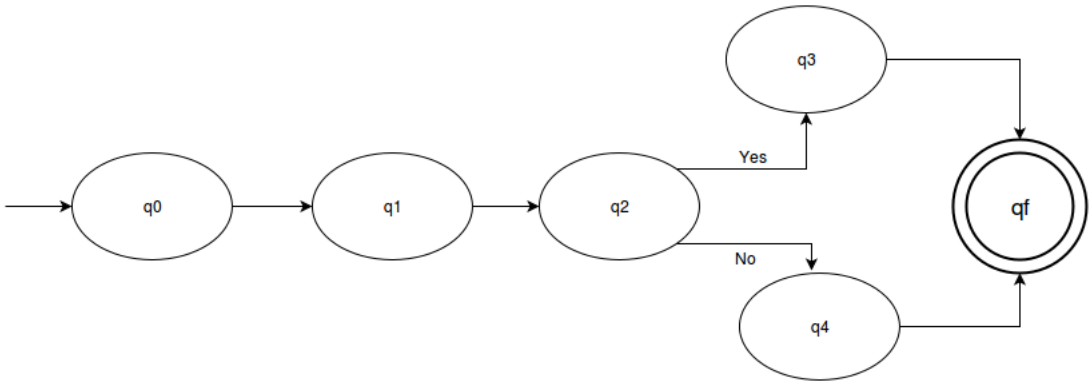
f_{lex} :function to generate tokens for given arithematic expression.

f_{match} :function to match the tokens with grammar rules.

f_{ret} :function to return result for expression.

DD = $\{tokens, grammarrules\}$
NDD = ϕ

7 State Diagram



q0 = read input code state
q1 = token generation state
q2 = grammar rule matching state
q3 = valid expression state
q4 = invalid expression state
qf = final state