JSS MAHAVIDYAPEETHA

SRI JAYACHAMARAJENDRA COLLEGE OF ENGINEERING

Mysore-570006

An Autonomous Institution Affiliated to VTU, Belgaum

Project Topic

“Parsing ’CREATE TABLE’ SQL Query ”

Submitted By

Roll No	Name	USN
60	Veeresh Koti	4JC14CS123
73	Sanganna Hallad	4JC15CS416

Under the guidance of

Mrs.Trisiladevi C. Nagavi and Mr. P M. Shivamurthy

Assistant Professors,Dept.of CS E

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

2017

SRI JAYACHAMARAJENDRA COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Mysore-570006

CERTIFICATE

This is to certify that the work entitled "Parsing 'CREATE TABLE' SQL Query" is a bonafied work carried out by Veeresh Koti, Sanganna Hallad in partial fulfillment of ciriculum for the award II event of SYSTEM SOFTWARE LABORATORY It is certified that all corrections / suggestions indicated during CIE have been incorporated in the report. The mini project report has been approved as it satisfies the academic requirements in respect of mini project work prescribed for the system software laboratory.

Lab In Charge and Guide.
Mrs.Trishila Nagdevi
Assistant Professor,
Dept of CS E,
S.J.C.E,Mysore

Lab In Charge and Guide.
Mr. P M. Shivamurthy
Assistant Professor,
Dept of CS E,
S.J.C.E,Mysore

Dr.T.N Nagabhushan
Principal, SJCE, Mysore

Place :Mysore

Date :21/04/2017

DECLARATION

We hereby declare that the project work entitled **"Parsing 'CREATE TABLE' SQL Query"** submitted to the **Visvesvaraya Technoloical University,Belgaum** is a record of an original work done by us under the guidance of **Mrs. Trisiladevi C. Nagavi and Mr. P M. Shivamurthy,** Assistant Professor, **Department of Computer Science and Engineering, SJCE, Mysore,** and this project work has been submitted in the partial fulfillment of the requirements for the award of the degree **Bachelor of Engineering in Computer Science and Engineering.** The results embodied in this project have not been submitted to any other University or Institute for the award of any degree or diploma.

Veeresh Koti

Sanganna Hallad

ACKNOWLEDGEMENT

It gives us immense pleasure to write an acknowledgement to this project, a contribution of all the people who helped to realize it. We extend our deep regards to **Dr. T.N.Nagabhushan,** Honorable Principal of Sri Jayachamarajendra College of Engineering, for providing an excellent environment for our education and his encouragement throughout our stay in college.

We would like to convey our heartfelt thanks to our HOD, **Dr. H.C.Vijayalakshmi,** for giving us the opportunity to embark upon this topic. We would like to thank our project guide, **Mrs. Trisiladevi C. Nagavi and Mr. P M. Shivamurthy,** for their invaluable guidance and enthusiastic assistance and for providing us support and constructive suggestions for the betterment of the project, without which this project would not have been possible.We appreciate the timely help and kind cooperation of our lecturers, other staff members of the department and our seniors, with whom we have come up all the way during our project work without whose support this project would not have been success. Finally, we would like to thank our friends for providing numerous insightful suggestions. We also convey our sincere thanks to all those who have contributed to this learning opportunity at every step of this project.

Veeresh Koti

Sanganna Hallad

ABSTRACT

Parsing or syntax analysis or syntactic analysis is the process of analysing a string of symbols, either in natural language or in computer languages, conforming to the rules of a formal grammar. The term parsing comes from Latin pars (orationis), meaning part (of speech). Many applications require the parsing of SQL language. The main objective of this library is to provide a fast and easy way of parsing a 'CREATE TABLE' SQL query. Parsing plays an important role with regards to the lex and yacc concepts.

SQL parser does the parsing of the given 'CREATE TABLE' query following the rules of the grammar.The corresponding lex and yacc specification specifies the importance of this parser.Any sort of 'CREATE TABLE' query can be easily parsed by this parser.

TABLE OF CONTENTS

1 INTRODUCTION

Parsing , syntax analysis or syntactic analysis is the process of analysing a string of symbols, either in natural language or in computer languages, conforming to the rules of a formal grammar. The term parsing comes from Latin pars (orationis), meaning part (of speech).

The term has slightly different meanings in different branches of linguistics and computer science. Traditional sentence parsing is often performed as a method of understanding the exact meaning of a sentence or word, sometimes with the aid of devices such as sentence diagrams. It usually emphasizes the importance of grammatical divisions such as subject and predicate.

Within computational linguistics the term is used to refer to the formal analysis by a computer of a sentence or other string of words into its constituents, resulting in a parse tree showing their syntactic relation to each other, which may also contain semantic and other information.

The term is also used in psycholinguistics when describing language comprehension. In this context, parsing refers to the way that human beings analyze a sentence or phrase (in spoken language or text) ”in terms of grammatical constituents, identifying the parts of speech, syntactic relations, etc.” This term is especially common when discussing what linguistic cues help speakers to interpret garden-path sentences.

Within computer science, the term is used in the analysis of computer languages, referring to the syntactic analysis of the input code into its component parts in order to facilitate the writing of compilers and interpreters. The term may also be used to describe a split or separation.

A parser is a software component that takes input data (frequently text) and builds a data structure often some kind of parse tree, abstract syntax tree or other hierarchical structure giving a structural representation of the input, checking for correct syntax in the process. The parsing may be preceded or followed by other steps, or these may be combined into a single step. The parser is often preceded by a separate lexical analyser, which creates tokens from the sequence of input characters ; alternatively, these can be combined in scannerless parsing. Parsers may be programmed by hand or may be automatically or semi-automatically generated by a parser generator. Parsing is complementary to templating, which produces formatted output. These may be applied to different domains, but often appear together, such as the scanf/printf pair, or the input (front end parsing) and output (back end code generation) stages of a compiler.

The input to a parser is often text in some computer language, but may also be text in a natural language or less structured textual data, in which case generally only certain parts of the text are extracted, rather than a parse tree being constructed. Parsers range from very simple functions such as scanf, to complex programs such as the frontend of a C++ compiler or the HTML parser of a web browser. An important class of simple parsing is done using regular expressions, in which a group of regular expressions defines a regular language and a regular expression engine automatically generating a parser for that language, allowing pattern matching and extraction of text. In other contexts regular expressions are instead used prior to parsing, as the lexing step whose output is then used by the parser.

The use of parsers varies by input. In the case of data languages, a parser is often found as the file reading facility of a program, such as reading in HTML or XML text ; these examples are markup languages. In the case of programming languages, a parser is a component of a compiler or interpreter, which parses the source code of a computer programming language to create some form of internal representation ; the parser is a key step in the compiler frontend. Programming languages tend to be specified in terms of a deterministic context-free grammar because fast and efficient parsers can be written for them.The implied disadvantages of a one-pass compiler can largely be overcome by adding fix-ups, where provision is made for fix-ups during the forward pass, and the fix-ups are applied backwards when the current program segment has been recognized as having been completed.

1.1 PROBLEM STATEMENT :

Parsing of **’CREATE TABLE’ SQL query** in Java.

It takes a ’CREATE TABLE’ query as input and does the parsing and indicates whether it is a valid query or not.

1.2 OBJECTIVES :

Computer programs have some SQL query language parsers. Some softwares provide templates that you may use however insecure it may be. But often we the programmers feel the need to parse sql queries given as strings at runtime. Some of our domain specific applications do need to allow the users enter queries to be parsed and evaluated at runtime.

The SQL queries serve as a basic enhancement incase of lex and yacc specifications.If the query cannot be parsed at the time you parse and request the result, a ParserException will be thrown.

1. Queries with zero or more known number of column parameters in the form of : CREATE TABLE table$_name(column1datatype, column2datatype, ...)$;

2. The column parameters specify the names of the columns of the table.

3.The datatype parameter specifies the type of data the column can hold (e.g. varchar, integer, date, etc.).

1.3 APPICATIONS

1.Lookahead establishes the maximum incoming tokens that a parser can use to decide which rule it should use. Lookahead is especially relevant to LL, LR, and LALR parsers, where it is often explicitly indicated by affixing the lookahead to the algorithm name in parentheses, such as LALR(1).2.Most programming languages, the primary target of parsers, are carefully defined in such a way that a parser with limited lookahead, typically one, can parse them, because parsers with limited lookahead are often more efficient. One important change[citation needed] to this trend came in 1990 when Terence Parr created ANTLR for his Ph.D. thesis, a parser generator for efficient LL(k) parsers, where k is any fixed value.3.Parsers typically have only a few actions after seeing each token. They are shift (add this token to the stack for later reduction), reduce (pop tokens from the stack and form a syntactic construct), end, error (no known rule applies) or conflict (does not know whether to shift or reduce). 4.Lookahead has two advantages. i.It helps the parser take the correct action in case of conflicts. For example, parsing the if statement in the case of an else clause. It eliminates many duplicate states and eases the burden of an extra stack.ii. A C language non-lookahead parser will have around 10,000 states. A lookahead parser will have around 300 states. The parse tree and resulting code from it is not correct according to language semantics.5.To correctly parse without lookahead, there are three solutions : The user has to enclose expressions within parentheses. This often is not a viable solution.6.The parser needs to have more logic to backtrack and retry whenever a rule is violated or not complete. The similar method is followed in LL parsers.7.Alternatively, the parser or grammar needs to have extra logic to delay reduction and reduce only when it is absolutely sure which rule to reduce first. This method is used in LR parsers. This correctly parses the expression but with many more states and increased stack depth.8.The input to a parser is often text in some computer language, but may also be text in a natural language or less structured textual data, in which case generally only certain parts of the text are extracted, rather than a parse tree being constructed. Parsers range from very simple functions such as scanf, to complex programs such as the frontend of a C++ compiler or the HTML parser of a web browser. An important class of simple parsing is done using regular expressions, in which a group of regular expressions defines a regular language and a regular expression engine automatically generating a parser for that language, allowing pattern matching and extraction of text.9. In other contexts regular expressions are instead used prior to parsing, as the lexing step whose output is then used by the parser.10.The use of parsers varies by input. In the case of data languages, a parser is often found as the file reading facility of a program, such as reading in HTML or XML text ; these examples are markup languages. In the case of programming languages, a parser is a component of a compiler or interpreter, which parses the source code of a computer programming language to create some form of internal representation ; the parser is a key step in the compiler frontend. Programming languages tend to be specified in terms of a deterministic context-free grammar because fast and efficient parsers can be written for them. For compilers, the parsing itself can be done in one pass or multiple passes see one-pass compiler and multi-pass compiler.

2 SYSTEM REQUIREMENTS

2.1 INPUT REQUIREMENTS

— A ’create table’ sql query.

— A ’create table’ sql query parser.

— User interaction for getting input file and checking the syntax through buttons.

2.2 OUTPUT REQUIREMENTS

— A dialogue box to be displayed after checking the syntax by using input given by the user.

2.3 SOFTWARE REQUIREMENTS

— An operating system.

— A system supporting the parser.

— Any software that supports Lex and Yacc.

— A software that supports JAVA.

2.4 HARDWARE REQUIREMENTS

— System

— Microprocessor

— System RAM 2GB

— Hard disk of 12GB

2.5 FUNCTIONAL REQUIREMENTS

Functional requirements may be calculations, technical details, data manipulation and processing and other specific functionality that define what a system is supposed to accomplish.[1] Behavioral requirements describing all the cases where the system uses the functional requirements are captured in use cases.

Functional requirements are supported by non-functional requirements (also known as quality requirements), which impose constraints on the design or implementation (such as performance requirements, security, or reliability). Generally, functional requirements are expressed in the form ”system must do ¡requirement¿”, while non-functional requirements are ”system shall be ¡requirement¿”.

The plan for implementing functional requirements is detailed in the system design. The plan for implementing non-functional requirements is detailed in the system architecture.

2.6 NON FUNCTIONAL REQUIREMENTS

— Performance-for example response time,throughput, utilization, static volumetric
— Scalability
— Capacity
— Availability
— Reliability
— Recoverability
— Serviceability
— Maintainability

3 TOOLS AND TECHNOLOGY

— Ubuntu 17.04

— flex and bison

— NetBeans-8.2

— GUI

4 SYSTEM DESIGN

— A 'CREATE TABLE' sql query is entered with the aid of java programming language.This query is parsed with the help of lex and yacc provided in the back end.The results of this expression is again being displayed.

5 SYSTEM IMPLEMENTATION

5.1 front-end code in JAVA

```
import java.io.*;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.FileReader;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;


public class lexical extends javax.swing.JFrame {

    public lexical() {
        initComponents();
    }

    private void initComponents() {

        jLabel1 = new javax.swing.JLabel();
        jScrollPane1 = new javax.swing.JScrollPane();
        text = new javax.swing.JTextArea();
        jLabel2 = new javax.swing.JLabel();
        compile = new javax.swing.JButton();
```

```
setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

jLabel1.setFont(new java.awt.Font("Tahoma", 1, 24));
jLabel1.setText("Validation of SQL Statements!");

text.setColumns(20);
text.setRows(5);
jScrollPane1.setViewportView(text);

jLabel2.setFont(new java.awt.Font("Tahoma", 1, 14));
jLabel2.setText("Enter the command");

compile.setFont(new java.awt.Font("Tahoma", 1, 12));
compile.setText("Compile");
compile.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        compileActionPerformed(evt);
    }
});

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
getContentPane().setLayout(layout);
layout.setHorizontalGroup(
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addGroup(layout.createSequentialGroup()
        .addGap(123, 123, 123)
        .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 526,
        .addGap(0, 115, Short.MAX_VALUE))
    .addGroup(layout.createSequentialGroup()
        .addGap(286, 286, 286)
```

```
                .addComponent(jLabel2)
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequential(
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEA
                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSe
                        .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE, 43
                        .addGap(67, 67, 67))
                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSe
                        .addComponent(compile, javax.swing.GroupLayout.PREFERRED_SIZE, 12
                        .addGap(309, 309, 309))))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGap(31, 31, 31)
                .addComponent(jLabel1)
                .addGap(79, 79, 79)
                .addComponent(jLabel2)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 269, 
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addComponent(compile, javax.swing.GroupLayout.DEFAULT_SIZE, 41, Short.MA
        );

        pack();
    }

    private void compileActionPerformed(java.awt.event.ActionEvent evt) {
        try {
            FileWriter fw=new FileWriter("input.txt");
            BufferedWriter br=new BufferedWriter(fw);
            br.write(text.getText());
    br.close();
        } catch (IOException ex) {
            System.out.println("exception");
        }


try
{
Process p=Runtime.getRuntime().exec("./a.out");
p.waitFor();
BufferedReader reader=new BufferedReader(new InputStreamReader(p.getInputStream()));
String line=reader.readLine();
while(line!=null)
{
System.out.println(line);
line=reader.readLine();
}


}
catch(IOException e1) {}
catch(InterruptedException e2) {}

System.out.println("Done");






String out=null;String out2="";
```

```
try{
FileReader fw=new FileReader("output.txt");
BufferedReader bw=new BufferedReader(fw);
while((out=bw.readLine())!=null)
{
out2=out2+"\n"+out;



}


}catch(Exception e){System.out.println("hi");
}





        JTextArea textArea = new JTextArea(20, 50);
      textArea.setText(out2);
      textArea.setEditable(false);

      JScrollPane scrollPane = new JScrollPane(textArea);

      JOptionPane.showMessageDialog(this, scrollPane);
    }

    public static void main(String args[]) {
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getIns
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {
            java.util.logging.Logger.getLogger(lexical.class.getName()).log(java.util.logg
        } catch (InstantiationException ex) {
            java.util.logging.Logger.getLogger(lexical.class.getName()).log(java.util.logg
        } catch (IllegalAccessException ex) {
            java.util.logging.Logger.getLogger(lexical.class.getName()).log(java.util.logg
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            java.util.logging.Logger.getLogger(lexical.class.getName()).log(java.util.logg
        }
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new lexical().setVisible(true);
            }
        });
    }
    private javax.swing.JButton compile;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JTextArea text;
}
```

5.2 back-end code

5.2.1 Lex code

```
%option noyywrap nodefault yylineno case-insensitive
%{
#include<stdio.h>
#include "y.tab.h"
int count;
%}

%%

[ \t\r]    ;
[\n]       count++;
BIT return BIT;
int        return INT;
INT1|TINYINT   return TINYINT;
INT2|SMALLINT   return SMALLINT;
MIDDLEINT|MEDIUMINT   return MEDIUMINT;
INT4?|INTEGER   return;
-?[0-9]+   return INTNUM;
INT8|BIGINT   return BIGINT;
REAL   return REAL;
FLOAT8|DOUBLE   return DOUBLE;
FLOAT4?   return FLOAT;
NUMERIC|DEC|DECIMAL return DECIMAL;
TIMESTAMP return TIMESTAMP;
DATETIME return DATETIME;
YEAR return YEAR;
BINARY return BINARY;
CHAR(ACTER)? return CHAR;
```

```
VARBINARY return VARBINARY;
BLOB return BLOB;
TINYBLOB return TINYBLOB;
MEDIUMBLOB return MEDIUMBLOB;
LONGBLOB return LONGBLOB;
TEXT return TEXT;
TINYTEXT return TINYTEXT;
LONGTEXT return LONGTEXT;
MEDIUMTEXT return MEDIUMTEXT;
ENUM return ENUM;
UNSIGNED return UNSIGNED;
ZEROFILL return ZEROFILL;
COLLATE return COLLATE;
UNIQUE return UNIQUE;
not    return NOT;
null    return NULLX;
set        return SET;
drop       return DROP;
table      return TABLE;
create     return CREATE;
add        return ADD;
column     return COLUMN;
alter      return ALTER;
varchar    return VARCHAR;
date       return DATE;
primary    return PRIMARY;
key        return KEY;
foreign   return FOREIGN;
time       return TIME;
['(']        return LEFT;
[')']        return RIGHT;
```

```
[,]        return COMMA;
[;]        return SEMICOLON;
[a-zA-Z][_a-zA-Z0-9]*     return IDENTIFIER;
[\.]       return DOT;
0B[01]+      |
B'[01]+'  |
X'[0-9A-F]+' |
0X[0-9A-F]+ |
'(\\.|''|[^'\n])*'   |
\"(\\.|\"\"|[^"\n])*\" return STRING;
. ECHO;
%%
```

5.2.2 Yacc code

```
%{
#include <stdio.h>
#include<stdlib.h>
int status=0;

FILE *yyin,*yyout;

%}

%token COMMA SEMICOLON IDENTIFIER DOT LEFT RIGHT SET DROP TABLE CREATE INT VARCHAR DATE T


%%
```

```
st     : CREATE TABLE IDENTIFIER LEFT collist RIGHT SEMICOLON {fprintf(yyout,"VALID!!!");e

collist: def
|
collist COMMA def
;

def: PRIMARY KEY LEFT column_list RIGHT
    | KEY LEFT column_list RIGHT
    | IDENTIFIER data_type column_atts
    ;
data_type:
     BIT opt_length
   | TINYINT opt_length opt_uz
   | SMALLINT opt_length opt_uz
   | MEDIUMINT opt_length opt_uz
   | INT opt_length opt_uz
   | INTEGER opt_length opt_uz
   | BIGINT opt_length opt_uz
   | REAL opt_length opt_uz
   | DOUBLE opt_length opt_uz
   | FLOAT opt_length opt_uz
   | DECIMAL opt_length opt_uz
   | DATE
   | TIME
   | TIMESTAMP
   | DATETIME
   | YEAR
   | CHAR opt_length opt_csc
   | VARCHAR LEFT INTNUM RIGHT opt_csc
   | BINARY opt_length
   | VARBINARY LEFT INTNUM RIGHT
   | TINYBLOB
   | BLOB
   | MEDIUMBLOB
   | LONGBLOB
   | TINYTEXT opt_binary opt_csc
   | TEXT opt_binary opt_csc
   | MEDIUMTEXT opt_binary opt_csc
   | LONGTEXT opt_binary opt_csc
   | ENUM LEFT enum_list RIGHT opt_csc
   | SET LEFT enum_list RIGHT opt_csc
   ;

opt_length:
   | LEFT INTNUM LEFT
   | LEFT INTNUM COMMA INTNUM RIGHT
   ;

opt_binary:
   | BINARY
   ;

opt_uz:
   | opt_uz UNSIGNED
   | opt_uz ZEROFILL
   ;

opt_csc:
   | opt_csc CHAR SET STRING
   | opt_csc COLLATE STRING
   ;
```

```
enum_list: STRING
   | enum_list ',' STRING
   ;
column_atts:
    |column_atts NOT NULLX
    | column_atts NULLX
    | column_atts UNIQUE LEFT column_list RIGHT
    | column_atts UNIQUE KEY
    | column_atts PRIMARY KEY
    | column_atts KEY
    ;
column_list: IDENTIFIER
  | column_list COMMA IDENTIFIER
  ;



%%

extern char * yytext;

int main(){
yyin = fopen("input.txt","r");
yyout = fopen("output.txt","w");
yyparse();
}

int yyerror(char *s){
fprintf(yyout,"INVALID!!!");
}
```

```
int yywrap(){
return 1;
}
```

6 SYSTEM TESTING AND RESULTS ANALYSIS

The testing phase involves checking of the syntax of a given ’create table’ sql query from the input file . It involves a step by step process :

— Presence of the ’CREATE TABLE’ query

— The declaration of table name

— The number of table elements

— Correct number of paranthesis

— Parsing of the query

— Evaluation of query

The output of the result may be ’valid’ or ’invalid’ depending upon the given query

7 CONCLUSION AND FUTURE WORK

Coming to personal conclusion rst, we have learnt the usage of datastructures, how we can utilise the linkedlists, structures and arrays in a better way with more efficiency. We have learnt the Filehandling functions and creating FIles to store object codes. Docu- mentation has been done using opensource software Latex and thus we have learnt the formatting and documentation of reports in a more productive way, earlier we used to do the documentation using wordsheet or notepad or other editor, but latex is the best tool for documentation which provides a sea of extra formatting features. The given SQL query is analyzed according to the grammar provided in the lex and yacc specification. This parsing and Evaluation has a greater scope in the future.

8 REFERENCES

1. https ://www.sharelatex.com
2. Leland L.Beck, System Software- An Introduction to Systems Programming, 3rd Edition,Pearson Education Asia 2006
3. https ://www.stackovreflow.com
4. https ://www.safaribooksonline.com/library/view/flex-bison/9780596805418/ch04.html