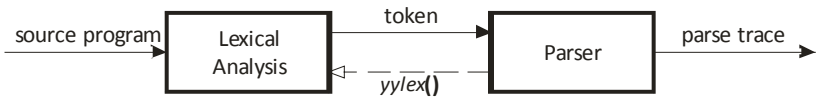

Figure 1. Execution Overview

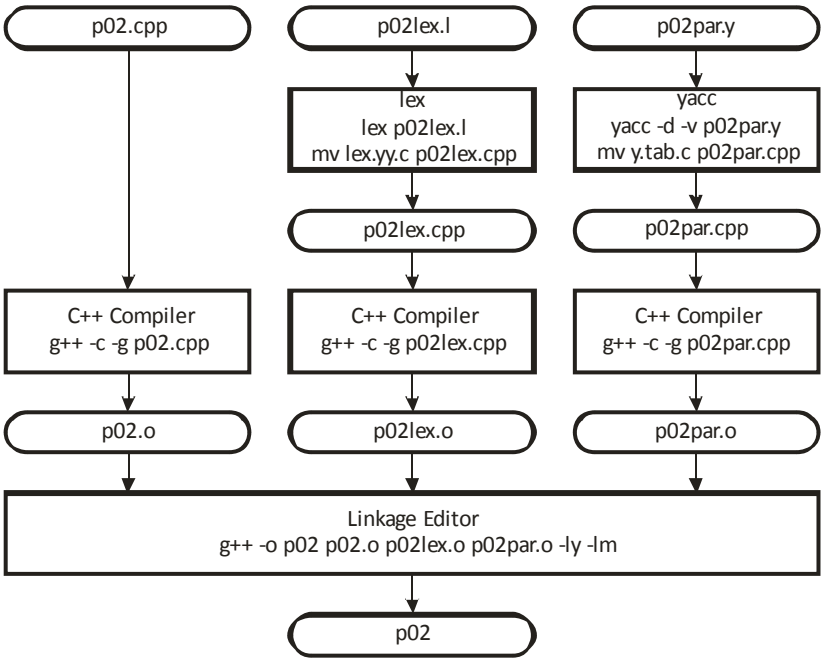

Figure 2. Compilation Overview

```
#---------------------------------------------------------------
# File makep02 contains instructions for creating file p02,
#---------------------------------------------------------------
# Author: Thomas R. Turner
# E-Mail: trturner@ucok.edu
# Date:   March, 2007
#---------------------------------------------------------------
# Copyright March, 2007 by Thomas R. Turner.
# Do not reproduce without permission from Thomas R. Turner.
#---------------------------------------------------------------
# Object files
#---------------------------------------------------------------
obj       =       p02.o p02par.o p02lex.o
#---------------------------------------------------------------
p02:              ${obj}
                  g++ -o p02 ${obj} -ly -lm
```

Figure 3. File p02make (continued)

```
#-----------------------------------------------------------------
# File p02par.cpp processes command line arguments
#-----------------------------------------------------------------
p02.o:              p02.cpp p02lex.h p02par.h
                    g++ -c -g p02.cpp
#-----------------------------------------------------------------
# File p02lex.l is the lexical analyzer
#-----------------------------------------------------------------
p02lex.cpp:     p02lex.l
                    lex p02lex.l
                    mv lex.yy.c p02lex.cpp
#-----------------------------------------------------------------
# File p02lex.cpp is created by lex in the previous step
#-----------------------------------------------------------------
p02lex.o:       p02lex.cpp p02lex.h y.tab.h
                    g++ -c -g p02lex.cpp
#-----------------------------------------------------------------
# File p02par.cpp is the C++ parser created by yacc
#-----------------------------------------------------------------
p02par.o:       p02par.cpp p02par.h
                    g++ -c -g p02par.cpp
#-----------------------------------------------------------------
# File p02par.y contains the specification of the Subset Pascal
# Parser in a format acceptable to yacc
#-----------------------------------------------------------------
y.tab.h\
p02par.cpp:     p02par.y
                    yacc -d -v p02par.y
                    mv y.tab.c p02par.cpp
#-----------------------------------------------------------------
```

Figure 3. File p02make (continued)

	left side		*right side*
1	*program*	→	**begin** *statement-list* **end**
2	*statement-list*	→	*statement*
3	*statement-list*	→	*statement-list statement*
4	*statement*	→	**id :=** *expression* **;**
5	*statement*	→	**read (** *id-list* **) ;**
6	*statement*	→	**write (** *expression-list* **) ;**
7	*id-list*	→	**id**
8	*id-list*	→	*id-list* **, id**
9	*expression-list*	→	*expression*
10	*expression-list*	→	*expression-list* **,** *expression*
11	*expression*	→	*primary*
12	*expression*	→	*expression additive-operator primary*
13	*primary*	→	**(** *expression* **)**
14	*primary*	→	**id**
15	*primary*	→	**intlit**
16	*additive-operator*	→	**+**
17	*additive-operator*	→	**-**

Table 1. Set of productions for the *micro* grammar of **Example 2.**

Yacc is a tool that will generate a parser given an LR(0) grammar.

Structure of a Yacc Grammar

… definition section …
%%
… rules section …
%%
… user subroutine section …

Symbol Conventions

Typically, non-terminal symbols are given in lowercase and terminal symbols are assigned all capital letters. For example, the rule:

program → program-head declarations program-body **.**

would be expressed for a yacc grammar as

```
program:
    program_head declarations program_body PERIOD
```

Note that hyphens have been changed to underscores to satisfy the C++ rules for identifiers and the period at the extreme right on the right hand side (RHS) of the rule has been changed to a capitalized spelling.

Definition Section

The definition section can contain

- *literal block*
  Declarations necessary for grammar actions and user subroutines are placed in the *literal block*. The *literal block* includes all **.h** files. A *literal block* is enclosed between **%{** and **%}** on separate lines as shown below.
  %{
  … C++ macro preprocessor definitions, declarations, and code …
  %}

- **%token** declarations
  %token declarations are used to define terminal symbols. Terminal symbols defined by **%token** declarations are made available to a scanner implemented using ***lex***. File y.tab.h is created when ***yacc*** is invoked. File **y.tab.h** assigns positive integer values to terminal symbols defined using **%token** declarations. The values assigned to the terminal symbols are their token codes not the actual values represented by the token. A token is an integer code and a spelling. The spelling is the string of characters recognized by the scanner for that token.

  To make the strings recognized by the scanner available to the parser for the example above, you must add the following statement to the scanner.

Variable *yytext* has type **char*** and points to the most recent string of characters recognized by the scanner.

Rules Section

The rules section contains

- grammar rules
  A rule of the grammar has a Left Hand Side (LHS) and a Right Hand Side (RHS). For example, consider the following expression grammar below with actions enclosed between **{** and **}**.
- actions containing C++ code

```
%token PLUS
%token MINUS
%token LPAREN
%token RPAREN
%token COMMA
%token SEMICOLON
%token ASSIGN
%token INTLIT
%token ID
%token READ
%token WRITE
%token BEGAN
%token END
%%
program:
    BEGAN statement_list END
    {tfs << endl << "#001 program -> begin statement-list end" ;
    }
statement_list:
    statement
    {tfs << endl << "#002 statement-list -> statement";
    }
statement_list:
    statement_list statement
    {tfs << endl << "#003 statement-list -> statement-list statement";
    }
statement:
    ID ASSIGN expression SEMICOLON
    {tfs << endl << "#004 statement -> := expression ;";
    }
statement:
     READ LPAREN id_list RPAREN SEMICOLON
    {tfs << endl << "#005 statement -> READ ( id-list ) ;";
    }
```

```
statement:
      WRITE LPAREN expression_list RPAREN SEMICOLON
     {tfs << endl << "#006 statement -> WRITE ( expression-list ) ;"; }
id_list:
     ID
     {tfs << endl << "#007 id-list -> ID"; }
id_list:
     id_list COMMA ID
     {tfs << endl << "#008 id-list -> id-list , ID"; }
expression_list:
     expression
     {tfs << endl << "#009 expression-list -> expression"; }
expression_list:
     expression_list COMMA expression
     {tfs << endl << "#010 expression-list -> expression-list , expression"; }
expression:
     primary
     {tfs << endl << "#011 expression -> primary"; }
expression:
     expression additive-operator primary
     {tfs << endl << "#012 expression -> expression additive-operator primary"; }
primary:
     LPAREN expression RPAREN
     {tfs << endl << "#013 primary -> ( expression )"; }
primary:
     ID
     {tfs << endl << "#014 primary -> ID"; }
primary:
     INTLIT
     {tfs << endl << "#015 primary -> INTLIT"; }
additive-operator:
     PLUS
     {tfs << endl << "#016 additive-operator -> +"; }
additive-operator:
     MINUS
     {tfs << endl << "#017 additive-operator -> -"; }
```

begin read(*x*); *x*:=*x*+2; write(*x*) end

```
Token:Code=267 Name=     BEGIN line=  1 col=  1 Spelling="begin"
Token:Code=269 Name=      READ line=  1 col=  7 Spelling="read"
Token:Code=263 Name=    LPAREN line=  1 col= 11 Spelling="("
Token:Code=273 Name=IDENTIFIER line=  1 col= 12 Spelling="x"
#007 IDENTIFIER_list->IDENTIFIER
Token:Code=264 Name=    RPAREN line=  1 col= 13 Spelling=")"
#005 READ ( IDENTIFIER_list )
#002 statement_list->statement
Token:Code=262 Name= SEMICOLON line=  1 col= 14 Spelling=";"
Token:Code=273 Name=IDENTIFIER line=  1 col= 16 Spelling="x"
Token:Code=265 Name=    ASSIGN line=  1 col= 17 Spelling=":="
Token:Code=273 Name=IDENTIFIER line=  1 col= 19 Spelling="x"
#014 primary->IDENTIFIER
Token:Code=259 Name=      PLUS line=  1 col= 20 Spelling="+"
#016 addop-> +
Token:Code=272 Name=    INTLIT line=  1 col= 21 Spelling="2"
#015 primary->INTLIT
#012 expression->primary addop primary
#004 IDENTIFIER := expression
#003 statement_list->statement_list ; statement
Token:Code=262 Name= SEMICOLON line=  1 col= 22 Spelling=";"
Token:Code=270 Name=     WRITE line=  1 col= 24 Spelling="write"
Token:Code=263 Name=    LPAREN line=  1 col= 29 Spelling="("
Token:Code=273 Name=IDENTIFIER line=  1 col= 30 Spelling="x"
#014 primary->IDENTIFIER
Token:Code=264 Name=    RPAREN line=  1 col= 31 Spelling=")"
#011 expression->primary
#009 expression_list->expression
#006 WRITE ( expression_list )
#003 statement_list->statement_list ; statement
Token:Code=268 Name=       END line=  1 col= 33 Spelling="end"
#001 program->BEGIN statement_list END
```

```
rm mcrpar.cpp
rm mcrlex.cpp
rm *.o
rm mcr
make -f makemicro
```

```
#---------------------------------------------------------------------
# File makemcr creates a micro language compiler
#---------------------------------------------------------------------
# Author: Thomas R. Turner
# E-Mail: trturner@uco.edu
# Date:   January, 2012
#---------------------------------------------------------------------
# Copyright January, 2012 by Thomas R. Turner.
# Do not reproduce without permission from Thomas R. Turner.
#---------------------------------------------------------------------
#---------------------------------------------------------------------
# Object files
#---------------------------------------------------------------------
obj =       mcrpar.o \
            mcrlex.o \
            mcr.o
#---------------------------------------------------------------------
# Bind the subset Pascal Scanneer
#---------------------------------------------------------------------
mcr:        ${obj}
            g++ -o mcr ${obj} -lm -ll
#---------------------------------------------------------------------
# File mcr.cpp processes command line arguments
#---------------------------------------------------------------------
mcr.o:      mcr.cpp mcrlex.h
            g++ -c -g mcr.cpp
#---------------------------------------------------------------------
# File mcrlex.cpp is the lex-generated scanner
#---------------------------------------------------------------------
mcrlex.cpp: mcrlex.l mcrlex.h
            lex mcrlex.l
            mv lex.yy.c mcrlex.cpp
#---------------------------------------------------------------------
#---------------------------------------------------------------------
mcrlex.o:   mcrlex.cpp mcrlex.h
            g++ -c -g mcrlex.cpp
#---------------------------------------------------------------
# Create files mcrpar.cpp and mcrtkn.h from file mcrpar.y
#---------------------------------------------------------------
mcrtkn.h    \
mcrpar.cpp:     mcrpar.y
                yacc -d -v mcrpar.y
                mv y.tab.c mcrpar.cpp
                mv y.tab.h mcrtkn.h
#---------------------------------------------------------------
# Compile the parser mcrpar.y
#---------------------------------------------------------------
mcrpar.o:   mcrpar.cpp mcrpar.h
```

```
		g++ -c -g mcrpar.cpp
#-------------------------------------------------------------
```

File mcr.cpp

```
//-----------------------------------------------------------------------
//File mcr.cpp contains functions that process command line arguments
//and interface with the lex-generated scanner
//-----------------------------------------------------------------------
//Author: Thomas R. Turner
//E-Mail: trturner@uco.edu
//Date:   January, 2012
//-----------------------------------------------------------------------
//Copyright January, 2012 by Thomas R. Turner
//Do not reproduce without permission from Thomas R. Turner
//-----------------------------------------------------------------------
//C++ Standard include files
//-----------------------------------------------------------------------
#include <cstdlib>
#include <cstring>
#include <iostream>
#include <fstream>
#include <iomanip>
#include <cstdio>
#include <string>
using namespace std;
//-----------------------------------------------------------------------
//Application include files
//-----------------------------------------------------------------------
#include "mcrlex.h"
#include "mcrpar.h"
//-----------------------------------------------------------------------
//Externals
//-----------------------------------------------------------------------
ofstream tfs;                    //trace file stream
//-----------------------------------------------------------------------
//BadSuffixException
//-----------------------------------------------------------------------
struct BadSuffixException {
    BadSuffixException(char* fn)
    {   cout << endl;
        cout << "Input file \"" << fn << "\" does not have a .mcr suffix.";
    }
};
```

```
//-------------------------------------------------------------------
//-------------------------------------------------------------------
class FileNameSuffix {
    char* prefix;
public:
    FileNameSuffix(char* fn)
    {   char* p=strstr(fn,".mcr");
        if (!p) throw BadSuffixException(fn);
        int n=p-fn;
        if (n+4!=strlen(fn)) throw BadSuffixException(fn);
        prefix=new char[strlen(fn)+1];
        strncpy(prefix,fn,n);
        prefix[n]=0;
    }
    ~FileNameSuffix(){if (prefix) delete[] prefix;}
    void Suffix(char* fn,const char* suffix)
    {   strcpy(fn,prefix);
        strcat(fn,suffix);
    }
};
//-------------------------------------------------------------------
//CommandLineException
//-------------------------------------------------------------------
struct CommandLineException {
    CommandLineException(int m,int a)
    {   cout << endl;
        cout << "Too many arguments on the command line.";
        cout << endl;
        cout << m << " argument(s) are permitted on the command line.";
        cout << endl;
        cout << a << " argument(s) appeared on the command line.";
        cout << endl;
    }
};
//-------------------------------------------------------------------
//FileException
//-------------------------------------------------------------------
struct FileException {
    FileException(const char* fn)
    {   cout << endl;
        cout << "File " << fn << " could not be opened.";
        cout << endl;
    }
};
```

```
//-------------------------------------------------------------------
//-------------------------------------------------------------------
void CompilerMgr(FILE* i)
{    Parser P(i);
     P.Parse();
}

//-------------------------------------------------------------------
//Function main processes command line arguments
//-------------------------------------------------------------------
int main(int argc,char* argv[])
{    try {
          char ifn[255];
          switch (argc) {
               case 1:              //Prompt for the input file name
                    cout << "Enter the input file name. ";
                    cin >> ifn;
                    break;
               case 2:              //Read the input file name
                    strcpy(ifn,argv[1]);
                    break;
               default:
                    throw CommandLineException(1,argc-1);
                    break;
          }
          FileNameSuffix F(ifn);    //Find the prefix of the input file name
          char tfn[255];
          F.Suffix(tfn,".trc");     //Create the trace file name
          FILE* i=fopen(ifn,"r");   //Open the input file
          if (!i) throw FileException(ifn);
          tfs.open(tfn); if (!tfs) throw FileException(tfn);
          CompilerMgr(i);
          tfs << endl;              //Put a new line in the trace file
          tfs.close();              //Close the trace file
          fclose(i);                //Close the input file
     } catch (...) {
          cout << endl;
          cout << "Program terminated!";
          cout << endl;
          cout << "I won't be back!";
          cout << endl;
          exit(EXIT_FAILURE);
     }
     return 0;
}
```

```
%{
//-------------------------------------------------------------------
//File mcrpar.y contains the grammar for Micro, a language defined by
//Fischer and LeBlanc in their book "Crafting a Compiler" ISBN0-8053-3201-4
//-------------------------------------------------------------------
//Author: Thomas R. Turner
//E-Mail: trturner@uco.edu
//Date:   January, 2012
//-------------------------------------------------------------------
//C and C++ include files
//-------------------------------------------------------------------
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <cmath>
#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
using namespace std;
//-------------------------------------------------------------------
//Application include files
//-------------------------------------------------------------------
#include "mcrpar.h"
//Externals
//-------------------------------------------------------------------
extern ofstream tfs;          //Trace File Stream
extern int line;              //Current Line - defined in mcrlex.l
extern int col;               //Current Column - define in mcrlex.l
//-------------------------------------------------------------------
//Globals
//-------------------------------------------------------------------
//-------------------------------------------------------------------
//User subroutines
//-------------------------------------------------------------------
void yyerror(const char*);
//-------------------------------------------------------------------
%}
```

```
%token TOKEN_BEGIN
/* arithmetic operators:    2*/
%token PLUS
%token MINUS

/* punctuation:           2*/
%token COMMA
%token SEMICOLON

/* parentheses & brackets:  2*/
%token LPAREN
%token RPAREN

/* assignment and range:    1*/
%token ASSIGN

/* reserve words:         4*/
%token RESERVE_WORDS
%token BEGIN_
%token END
%token READ
%token WRITE

/* regular expressions:     2*/
%token REGULAR_EXPRESSIONS
%token INTLIT
%token IDENTIFIER
%token TOKEN_END
/* total:               13*/
```

```
%%
program: BEGIN_ statement_list END
 {tfs << endl << "#001 program->BEGIN statement_list END";
 }
statement_list: statement
 {tfs << endl << "#002 statement_list->statement";
 }
statement_list: statement_list SEMICOLON statement
 {tfs << endl << "#003 statement_list->statement_list ; statement";
 }
statement: IDENTIFIER ASSIGN  expression
 {tfs << endl << "#004 IDENTIFIER := expression";
 }
statement: READ LPAREN identifier_list RPAREN
 {tfs << endl << "#005 READ ( IDENTIFIER_list )";
 }
statement: WRITE LPAREN expression_list RPAREN
 {tfs << endl << "#006 WRITE ( expression_list )";
 }
identifier_list: IDENTIFIER
 {tfs << endl << "#007 IDENTIFIER_list->IDENTIFIER";
 }
identifier_list: identifier_list COMMA IDENTIFIER
 {tfs << endl << "#008 identifier_list->identifier_list , IDENTIFIER";
 }
expression_list: expression
 {tfs << endl << "#009 expression_list->expression";
 }
expression_list: expression_list COMMA expression
 {tfs << endl << "#010 expression_list->expression_list , expression";
 }
expression: primary
 {tfs << endl << "#011 expression->primary";
 }
expression: primary addop primary
 {tfs << endl << "#012 expression->primary addop primary";
 }
primary: LPAREN expression RPAREN
 {tfs << endl << "#013 primary->( expression )";
 }
primary: IDENTIFIER
 {tfs << endl << "#014 primary->IDENTIFIER";
 }
primary: INTLIT
 {tfs << endl << "#015 primary->INTLIT";
 }
addop: PLUS
 {tfs << endl << "#016 addop-> +";
```

```
 }
addop: MINUS
 {tfs << endl << "#017 addop-> -";
 }
%%
//-------------------------------------------------------------------------
//User function section
//-------------------------------------------------------------------------
struct Error {
    Error(const char* m)
    {   cout << endl << "line(" << line << ") col(" << col << ") " << m;
        cout << endl;
    }
};
//-------------------------------------------------------------------------
//Required function yyerror
//-------------------------------------------------------------------------
void yyerror(const char* m){throw Error(m);}
//-------------------------------------------------------------------------
//-------------------------------------------------------------------------
Parser::Parser(FILE* i):Lexer(i){}
int Parser::Parse(){return yyparse();}
//-------------------------------------------------------------------------
//-------------------------------------------------------------------------
Parser::~Parser(){}
```

```
/* A Bison parser, made by GNU Bison 2.4.1.  */

/* Skeleton interface for Bison's Yacc-like parsers in C

   Copyright (C) 1984, 1989, 1990, 2000, 2001, 2002, 2003, 2004, 2005, 2006
  Free Software Foundation, Inc.

  This program is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  This program is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with this program.  If not, see <http://www.gnu.org/licenses/>.  */

/* As a special exception, you may create a larger work that contains
  part or all of the Bison parser skeleton and distribute that work
  under terms of your choice, so long as that work isn't itself a
  parser generator using the skeleton or a modified version thereof
  as a parser skeleton.  Alternatively, if you modify or redistribute
  the parser skeleton itself, you may (at your option) remove this
  special exception, which will cause the skeleton and the resulting
  Bison output files to be licensed under the GNU General Public
  License without this special exception.

  This special exception was added by the Free Software Foundation in
  version 2.2 of Bison.  */


/* Tokens.  */
#ifndef YYTOKENTYPE
# define YYTOKENTYPE
  /* Put the tokens into the symbol table, so that GDB and other debuggers
     know about them.  */
  enum yytokentype {
    TOKEN_BEGIN = 258,
    PLUS = 259,
    MINUS = 260,
    COMMA = 261,
    SEMICOLON = 262,
    LPAREN = 263,
    RPAREN = 264,
    ASSIGN = 265,
```

```
    RESERVE_WORDS = 266,
    BEGIN_ = 267,
    END = 268,
    READ = 269,
    WRITE = 270,
    REGULAR_EXPRESSIONS = 271,
    INTLIT = 272,
    IDENTIFIER = 273,
    TOKEN_END = 274
  };
#endif
/* Tokens.  */
#define TOKEN_BEGIN 258
#define PLUS 259
#define MINUS 260
#define COMMA 261
#define SEMICOLON 262
#define LPAREN 263
#define RPAREN 264
#define ASSIGN 265
#define RESERVE_WORDS 266
#define BEGIN_ 267
#define END 268
#define READ 269
#define WRITE 270
#define REGULAR_EXPRESSIONS 271
#define INTLIT 272
#define IDENTIFIER 273
#define TOKEN_END 274



#if ! defined YYSTYPE && ! defined YYSTYPE_IS_DECLARED
typedef int YYSTYPE;
# define YYSTYPE_IS_TRIVIAL 1
# define yystype YYSTYPE /* obsolescent; will be withdrawn */
# define YYSTYPE_IS_DECLARED 1
#endif

extern YYSTYPE yylval;
```

```
%{
//-------------------------------------------------------------------
// File mcrlex.l defines a scanner for micro, a language defined
// in "Crafting a Compiler" by Fischer and LeBlanc.
//-------------------------------------------------------------------
// Author: Thomas R. Turner
// E-Mail: trturner@uco.edu
// Date:   January, 2012
//-------------------------------------------------------------------
//Copyright January, 2012 by Thomas R. Turner.
//Do not reproduce without permission from Thomas R. Turner
//-------------------------------------------------------------------
//-------------------------------------------------------------------
// Standard C and C++ Library Include Files
//-------------------------------------------------------------------
#include <cstdlib>
#include <cstring>
#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
#include <cstdio>
#include <map>
using namespace std;
//-------------------------------------------------------------------
// Application Includes
//-------------------------------------------------------------------
#include "mcrlex.h"
#include "mcrtkn.h"
//-------------------------------------------------------------------
//Externals
//-------------------------------------------------------------------
extern ofstream tfs;
//-------------------------------------------------------------------
//Global Variables
//-------------------------------------------------------------------
static map<string,int> RW;
static int tokencode;
static string* TokenName;
int line=1;
int col =1;
//-------------------------------------------------------------------
//Functions
//-------------------------------------------------------------------
void ToLower(char* o,char* i,int l);              //Coerce string i to lower case
int TokenMgr(int t);                              //Token post processing
void PrintToken(ostream& o,int tc,int l,int c);   //Print the token and attributes
//-------------------------------------------------------------------
```

```
//Exceptions
//---------------------------------------------------------------------
struct StringTokenException{
    StringTokenException(char* t,int l,int c)
    {   cout << endl;
        cout << "line(" << l << ") col (" << c << ")" ;
        cout << "Lexical error: ";
        cout << "Strings cannot span lines";
        cout << endl;
        cout << "|" << t << "|";
        cout << endl;
    }
};
struct BadCharacterException{
    BadCharacterException(char p,int l,int c)
    {   cout << endl;
        cout << "line(" << l << ") col (" << c << ")" ;
        cout << "Lexical error: ";
        cout << "Illegal character |" << p << "| ASCII code=" << (int)p;
        cout << endl;
    }
};
%}
```

```
%%
[ \t]+              {col+=strlen(yytext);}
[\n]                {line++;col=1;}
[a-zA-Z][a-zA-Z0-9]* return TokenMgr(IDENTIFIER);
[0-9]+              return TokenMgr(INTLIT);
"+"                 return TokenMgr(PLUS);
"-"                 return TokenMgr(MINUS);
","                 return TokenMgr(COMMA);
";"                 return TokenMgr(SEMICOLON);
"("                 return TokenMgr(LPAREN);
")"                 return TokenMgr(RPAREN);
":="                return TokenMgr(ASSIGN);
.                   {   throw BadCharacterException
                            (*yytext
                            ,line
                            ,col
                            );
                    }
```

```
%%
//-------------------------------------------------------------------
//Class Lexer implementation
//-------------------------------------------------------------------
//-------------------------------------------------------------------
void ToLower(char* o,char* i,int l)
{    for (int a=0;a<l&&a<1024;a++) o[a]=tolower(i[a]);   //To lower case
     o[l]=0;                                             //Null termination
}
//-------------------------------------------------------------------
//Function TokenMgr processes the token after it has been recognized
//-------------------------------------------------------------------
int TokenMgr(int t)
{    int tc=t;
     if (t==IDENTIFIER) {
          char s[1024];
          ToLower(s,yytext,strlen(yytext));
          tc=RW[s];
          if (tc==0) tc=t;
     }
     PrintToken(tfs,tc,line,col);
     col+=yyleng;
     return tc;
}
```

```
//-------------------------------------------------------------------
//Constructor Lexer is used to redirect the input file stream from the
//keyboard to input file stream i.
//-------------------------------------------------------------------
Lexer::Lexer(FILE* i)
{   yyin=i;
    const int MAXSY=17;
    static string sy[]=
        {"TOKEN_BEGIN"      ,"PLUS"                    ,"MINUS"   ,"COMMA"
        ,"SEMICOLON"        ,"LPAREN"                  ,"RPAREN"  ,"ASSIGN"
        ,"RESERVE_WORDS"    ,"BEGIN"                   ,"END"     ,"READ"
        ,"WRITE"            ,"REGULAR_EXPRESSIONS"     ,"INTLIT"  ,"IDENTIFIER"
        , "TOKEN_END"
    };
    TokenName=new string[MAXSY];
    for (int a=0;a<MAXSY;a++) TokenName[a]=sy[a];
    static string rw[]=
        {"begin"            ,"end"                     ,"read"    ,"write"
        };
    static int tc[]=
        {BEGIN_             ,END                       ,READ      ,WRITE
        };
    for (int a=0;a<4;a++) RW[rw[a]]=tc[a];
}
//-------------------------------------------------------------------
//Function Lex calls yylex
//-------------------------------------------------------------------
int Lexer::Lex(void)
{   tokencode=yylex();
    return tokencode;
}
//-------------------------------------------------------------------
//Function PrintToken prints the token code name and the spelling of the
//token.
//-------------------------------------------------------------------
void PrintToken(ostream& o,int tc,int l,int c)
{   o << endl;
    o << "Token";
    o << ":Code="    << setw( 3) << tc;
    o << " Name="    << setw(10) << TokenName[tc-TOKEN_BEGIN];
    o << " line="    << setw( 3) << l;
    o << " col="     << setw( 3) << c;
    o << " Spelling=\"" << (char*)yytext << "\"";
}

//----------------------End of Lex Definition----------------------
```

```
#ifndef mcrlex_h
#define mcrlex_h 1
//-----------------------------------------------------------------------
// File mcrlex.h defines class Lexer.
//-----------------------------------------------------------------------
// Author: Thomas R. Turner
// E-Mail: trturner.uco.edu
// Date:   January, 2012
//-----------------------------------------------------------------------
// Copyright January, 2012 by Thomas R. Turner
// Do not reproduce without permission from Thomas R. Turner.
//-----------------------------------------------------------------------
//-----------------------------------------------------------------------
// Standard C and C++ include files
//-----------------------------------------------------------------------
#include <cstdio>
#include <fstream>
#include <iostream>
//-----------------------------------------------------------------------
//Namespaces
//-----------------------------------------------------------------------
using namespace std;
//-----------------------------------------------------------------------
//Function: yylex
//Function yylex is the mcrner.  Function yylex returns an integer
//token code as defined above or 0 if end-of-file has been
//reached.
//-----------------------------------------------------------------------
#ifdef __cplusplus
extern "C"
#endif
int yylex (void);
//-----------------------------------------------------------------------
//Class Lexer defines the attributes of a Scanner
//-----------------------------------------------------------------------
class Lexer {
public:
    Lexer(FILE* i);        //Constructor used to redirect the keyboard (stdin) to file i.
    int Lex(void);         //Call the scanner yylex and return the code
};
#endif
```