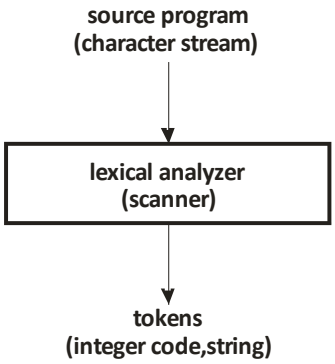

Figure 1. Lexical analyzer input and output

Input: **var** *a*,*b*,*c*:*real*;

Integer code	Integer code name	String spelling
221	**VAR**	var
200	**ID**	a
300	**COMMA**	,
200	**ID**	b
300	**COMMA**	,
200	**ID**	c
301	**COLON**	:
200	**ID**	real
302	**SEMICOLON**	;

Table 1. Lexical analyzer output for "**var** *a*,*b*,*c*:*real*;"

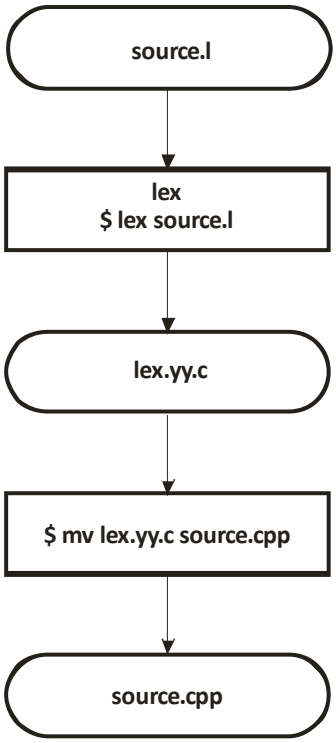

Figure 2. Invocation of lex

Notes:

1. The input file name always has the suffix **.l**
2. The output file name is always **lex.yy.c**
3. The command to invoke the lex utility
   $ lex source.l
4. Every c-program is also a c++-program. To change the output file to be a c++-program only the name needs to be changed.
   $ mv lex.yy.c source.cpp

1. ***Structure of a Lex Specification***

   … definition section
   %%
   … rules section
   %%
   … user subroutines

2. ***Definition Section***
   2.1. literal block
   %{
   … C and C++ comments, directives, and declarations
   %}
   2.2. definitions
   A definition takes the form:

   NAME expression

   The name can contain letters, digits, and underscores, and must not start with a digit.

   In the rules section, patterns may include references to substitutions with the name in braces, for example, "{NAME}". The expression corresponding to the name is substituted literally into pattern. For example.

```
DIGIT                   [0-9]
…
%%
{DIGIT}+                process_integer();
{DIGIT}+\.{DIGIT}*  |
\.{DIGIT}+              process_real();
```

Figure 1. A lex specification that containing a definition

3. ***Rules Section***
   A rule is a pattern followed by C or C++ code. For example:

   substituted literally into pattern. For example.

```
%%
[ \t\n]+ ;
%%
```

Figure 2. A lex specification that discards white space

3.1. Regular Expression Syntax

3.1.1.Metacharacters

Character	Description
.	Matches any single character except the newline character '\n'.
[]	Match any one of the characters with the brackets. A range of characters is indicated with the "-" (dash), e.g., "**[0-9]**" for any of the 10 digits. If the first character after the open bracket is a dash or a close bracket, it is not interpreted as a metacharacter If the first character is a circumflex "^" it changes the meaning to match any character except those within the brackets. (Such a character class *will* match a newline unless you explicitly exclude it.) Other metacharacters have no special meaning within square brackets except that C escape sequences starting with "\" are recognized.
*****	Matches zero or more of the preceding expression. For example, the pattern **a.*z** matches any string that starts with "a" and ends with "z", such as "az", "abz", or "alcatraz".
+	Matches one or more occurrence of the preceding regular expression. For example, **x+** matches "x", "xxx", or "xxxxx", but not an empty string, and **(ab)+** matches "ab", "abab", "ababab", and so forth.
?	Matches zero of one occurrence of the preceding regular expression. For example: **-?[0-9]+** indicates a whole number with an optional leading unary minus sign.

Character	Description
{}	A single number "{***n***}" means *n* repetitions of the preceding pattern, e.g., **[A-Z]{3}** matches any three upper case letters. If the braces contain two numbers separated by a comma, "{*n*,*m*}", they are a minimum and maximum number of repetitions of the preceding pattern. For example: **A{1,3}** matches one to three occurrences of the letter "**A**". If the second number is missing, it is taken to be infinite, so "{**1,**}" means the same as "**+**" and "**{0,}**" means the same as "*****"**.**
****	If the following character is a lowercase letter, then it is a C escape sequence such as "**\t**" for tab. Some implementations also allow octal and hex characters in the form "**\123**" and "**\x3f**". Otherwise "****" quotes the following character, so"*****" matches an asterisk.
()	Group a series of regular expressions together. Each of the "*****", "**+**", and "**[]**" effects only the expression immediately to its left, and "**|**" normally affects everything to its left and right. Parentheses can change this, for example: **(ab|cd)?ef** matches "**abef**", "**cdef**", or just "**|**"
|	Match either the preceding regular expression or the subsequent regular expression. For example: **twelve|12** matches either "**twelve**" or "**12**"
"…"	Match everything withing the quotation marks literally. Metacharacters other than "\" lose their meaning. For example: "/*" matches the two characters
/	Matches the preceding regular expression but only if followed by the following regular expression. For example: **0/1** matches "0" in the string "01" but does not match anything in the strings "**0**" or "**02**". Only one slash is permitted per pattern, and a pattern cannot contain both a slash and a trailing "**$**"

Character	Description
^	As the first character of a regular expression, it matches the beginning of a line; it is also used for negation within square brackets. Otherwise not special.
$	As the last character of a regular expression, it matches the end of a line – otherwise it is not special. The "**$**" has the same meaning as "**/\n**" when at the end of an expression.
<>	A name of list of names in angle brackets at the beginning of a pattern makes that pattern apply only in the given start states.

4. ***User Subroutines***

User subroutines are C and C++ functions. Function prototypes must appear before their implementations in this section.

```
%{
#include <string>
#define ID       1
#define READ     2
#define WRITE    3
#define BEGAN    4
#define END      5
int TokenMgr(int t);
%}
%%
[ \t\n]+                    ;
[a-z]+                      return TokenMgr(ID);
%%
int TokenMgr(int t)
{    string rw[]={"","","read","write","begin","end"};
     for (int k=2;k<6;k++) if ((string)yytext==rw[k]) return k;
     return t;
}
```

Figure 2. A lex specification containing a user subroutine

5. ***lex and C++***

The Unix utility ***lex*** creates a C program and is designed to work with other C programs. Care must be exercised to employ ***lex*** in a C++ environment. Directives shown in figure 3 must be included to ensure the function *yylex*, the lexical analyzer produced by ***lex*** can be called from a C++ program.

```
#ifdef __cplusplus
extern "C"
#endif
int yylex (void);
```

Figure 3. C++ Preprocessor directives allowing function *yylex* to be called from a C++ program

6. ***lex and files***

Since ***lex*** creates a C program, it uses standard input/output text file definitions developed for C in include file <cstdio>. If you wish to have your scanner find tokens in an external file, you will have to redirect the standard input file from the keyboard to a FILE as defined in the include file <cstdio>. Refer to the code fragment included in figure 4.

```
#include <cstdio>
…
char ifn[255];                  //Input file name
FILE* i=fopen(ifn,"r");         //Open the file whose name is stored in string ifn.
…
yyin=i;                         // Redirect the input from the keyboard to FILE i
                                // Variable yyin is the name given to the standard
input file
                                //  by lex.
fclose(i);                      //Close FILE i.
```

Figure 4. lex and the standard input file

Invoking lex and makefiles

Typically, a programmer will want to automate the creation of a program that includes a scanner. An example ***makefile*** is given in figure 5. Note that the program consists of two source files, pas.cpp and paslex.l. File pas.cpp is compiled in the normal way. The utility lex creates file lex.yy.c from pasles.l. Then, file **lex.yy.c** is renamed to **paslex.cpp**. Next, **paslex.cpp** is translated by the C++ compiler to object file **paslex.o**. Note that *every* C program is also a C++ program. Finally, the two object files **pas.o** and **paslex.o** are bound into and executable program in file **pas**.

```
#-----------------------------------------------------------------------
# File makepas creates a subset Pascal Scanner
#-----------------------------------------------------------------------
# Author: Thomas R. Turner
# E-Mail: trturner@uco.edu
# Date:   November, 2006
#-----------------------------------------------------------------------
# Copyright November, 2006 by Thomas R. Turner.
# Do not reproduce without permission from Thomas R. Turner.
#-----------------------------------------------------------------------
# Object files
#-----------------------------------------------------------------------
obj    =            pas.o \
                    paslex.o
#-----------------------------------------------------------------------
# Bind the subset Pascal Scanneer
#-----------------------------------------------------------------------
pas:                ${obj}
                    g++ -o pas ${obj} -lm -ll
#-----------------------------------------------------------------------
# File pas.cpp processes command line arguments
#-----------------------------------------------------------------------
pas.o:              pas.cpp paslex.h
                    g++ -c -g pas.cpp
#-----------------------------------------------------------------------
# File paslex.cpp is the lex-generated scanner
#-----------------------------------------------------------------------
paslex.cpp:         paslex.l paslex.h
                    lex paslex.l
                    mv lex.yy.c paslex.cpp
#-----------------------------------------------------------------------
#-----------------------------------------------------------------------
paslex.o:           paslex.cpp paslex.h
                    g++ -c -g paslex.cpp
```

Figure 5. File makepas, a makefile that creates a Subset Pascal Scanner.

Reference:

1. Levine, J. R., Mason, T., and Brown D. *lex& yacc 2nd* Ed. O'Reilly & Associates 1992 ISBN: 1-56592-000-7
2. Gardner, J. Linseman, A. Nicol, S., Retterrath, C. and Chartier, M. *MKS LEX & YACC 3rd Ed*. Mortice Kern Systems, Inc. 1993 ISBN 1-895033-26-8

```
#---------------------------------------------------------------------
# File p03make creates executable file p03.
#---------------------------------------------------------------------
# Author: Thomas R. Turner
# E-Mail: tturner@uco.edu
# Date:   September, 2002
#---------------------------------------------------------------------
# Bind p03.o, Scan03.o
#---------------------------------------------------------------------
p03:            p03.o Scan03.o
                g++ -o p03 p03.o Scan03.o -ll
#---------------------------------------------------------------------
# Compile p03.cpp
#---------------------------------------------------------------------
p03.o:          p03.cpp Scan03.h
                g++ -g -c p03.cpp
#---------------------------------------------------------------------
# Compile Scan03.l. First translate the lex specification, then compile
#---------------------------------------------------------------------
Scan03.o:     Scan03.cpp Scan03.h
          g++ -g -c Scan03.cpp
Scan03.cpp:  Scan03.l Scan03.h
          lex Scan03.l
          mv lex.yy.c Scan03.cpp
```

Figure 6. File **p03make**

```
//-----------------------------------------------------------------------
//File p03.cpp processes command line parameters, opens input and output files
//found on the command line, and employs a stack to compute the value of
//a postfix expresssion found in the input file.
//-----------------------------------------------------------------------
//Author: Thomas R. Turner
//E-Mail: tturner@uco.edu
//Date:   September, 2001
//-----------------------------------------------------------------------
//Copyright September, 2001 by Thomas R. Turner
//Do not reproduce without permission from Thomas R. Turner
//-----------------------------------------------------------------------
//Standard C and C++ includes
//-----------------------------------------------------------------------
#include <iostream>
#include <fstream>
#include <cstdio>
#include <string>
using namespace std;
//-----------------------------------------------------------------------
//Application includes
//-----------------------------------------------------------------------
#include "Scan03.h"
//-----------------------------------------------------------------------
//FileException is thrown when a file whose name is given on the command line
//cannot be opened.
//-----------------------------------------------------------------------
struct FileException {
  FileException(char* fn)
  {  cout << endl;
     cout << "File " << fn << " cannot be opened.";
  }
};
//-----------------------------------------------------------------------
//CommandLineException is thrown when too many arguments are given on the command
//line.
//-----------------------------------------------------------------------
struct CommandLineException {
  CommandLineException(int ac)
  {  cout << endl;
     cout << "Too many (" << ac << ") command line arguments.";
  }
};
```

Figure 7. File **p03.cpp**

```
//---------------------------------------------------------------------------
//Function Mgr processes the input file stream i
//---------------------------------------------------------------------------
void Mgr(FILE* i, ostream& o)
{  Scan L(i);
   for (;;) {
     int t=L.Lex();
     if (t==0) break;
     switch (t) {
       case INTLIT:
         o << endl << "INTLIT=" << L.FetchSpelling();
         break;
       case PLUS:
         o << endl << "PLUS  =" << L.FetchSpelling();
         break;
       case MINUS:
         o << endl << "MINUS =" << L.FetchSpelling();
         break;
       case STAR:
         o << endl << "STAR  =" << L.FetchSpelling();
         break;
       case SLASH:
         o << endl << "SLASH =" << L.FetchSpelling();
         break;
     }
  }
  o << endl;
}
```

Figure 7. File **p03.cpp** (continued)

```
//-------------------------------------------------------------------------
//Function main processes command line arguments and opens files specified
//on the command line.
//-------------------------------------------------------------------------
int main(int argc,char* argv[])
{  try {
    char ifn[255];              //Input File Name
    char ofn[255];               //Output File Name
    switch (argc) {
        case 1:                 //Prompt for both file names
            cout << "Enter the input file name. ";
            cin >> ifn;
            cout << "Enter the output file name. ";
            cin >> ofn;
            break;
        case 2:
            strcpy(ifn,argv[1]);
            cout << "Enter the output file name. ";
            cin >> ofn;
            break;
        case 3:
            strcpy(ifn,argv[1]);
            strcpy(ofn,argv[2]);
            break;
        default:
            throw CommandLineException(argc);
    }
    FILE* ifp=fopen(ifn,"r");  if (!ifp) throw FileException(ifn);
    ofstream ofs(ofn);        if (!ofs) throw FileException(ofn);
    Mgr(ifp,ofs);
    fclose(ifp);
    ofs.close();
  } catch ( ... ) {
    cout << endl;
    cout << "Program terminated.";
    exit(EXIT_FAILURE);
  }

  return 0;
}
```

Figure 7. File **p03.cpp**(continued)

```
#ifndef Scan03_h
#define Scan03_h 1
//---------------------------------------------------------------------
// File: Scan03.h
// Description:
// Recognizes integers and arithmetic operators for project 3 in
// Programming II.
//---------------------------------------------------------------------
// Author: Thomas R. Turner
// E-Mail: trturner.uco.edu
// Date:   September, 2003
//---------------------------------------------------------------------
// Copyright September, 2003 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;
//---------------------------------------------------------------------
//Token code definitions
//---------------------------------------------------------------------
#define INTLIT 1
#define PLUS   2
#define MINUS  3
#define STAR   4
#define SLASH  5
//---------------------------------------------------------------------
//Function: yylex
//Function yylex is the Scanner.  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);
```

Figure 8. File **Scan03.h**

```
//-----------------------------------------------------------------------
//Class Scan defines the attributes of a Scanner
//-----------------------------------------------------------------------
class Scan {
 int tokencode;              //Code for the most recent token found
public:
 Scan(FILE* i);              //Redirect the input source from the
                        //keyboard to input file i.
 int Lex(void);              //Call the scanner yylex and return the code
                        //found by yylex
 int FetchTokenCode(void);     //Return the code of the most recent token
 void StoreTokenCode(int T);   //Store the token code.
 char* FetchSpelling(void);    //Return the spelling of the most recent
                        //token
};
#endif
```

Figure 8. File **Scan03.h** (continued)

```
%{
//---------------------------------------------------------------------
// File: Scan03.l
// Description:
// Contains the most elementary example use of lex for the purpose of
// building a scanner.
//---------------------------------------------------------------------
// Author: Thomas R. Turner
// E-Mail: trturner@uco.edu
// Date:   September, 2003
//---------------------------------------------------------------------
//Copyright September, 2003 by Thomas R. Turner.
//Do not reproduce without permission from Thomas R. Turner
//---------------------------------------------------------------------
//---------------------------------------------------------------------
// C++ Library Include Files
//---------------------------------------------------------------------
#include <string>
#include <cstdlib>
#include <iostream>
#include <fstream>
using namespace std;
//---------------------------------------------------------------------
// Application Includes
//---------------------------------------------------------------------
#include "Scan03.h"
//---------------------------------------------------------------------
//Function prototypes
//---------------------------------------------------------------------
int TokenMgr(int T);
//---------------------------------------------------------------------
//Global Variables
//---------------------------------------------------------------------
%}
%%
[ \t\n]+                    ;
[+-]?[0-9]+                   {
                            return(TokenMgr(INTLIT));
                          }
"+"                         {
                            return(TokenMgr(PLUS));
                          }
"-"                         {
                            return(TokenMgr(MINUS));
                          }
"*"                         {
                            return(TokenMgr(STAR));
                          }
"/"                         {
                            return(TokenMgr(SLASH));
                          }
%%
```

Figure 9. File **Scan03.l**

```
//-----------------------------------------------------------------
int TokenMgr(int T)
{  return T;
}
//-----------------------------------------------------------------
//Class Scan implementation
//-----------------------------------------------------------------
//Constructor Scan is used to redirect the input file stream from the
//keyboard to input file stream i.
//-----------------------------------------------------------------
Scan::Scan(FILE* i)
{  yyin=i;
}
//-----------------------------------------------------------------
//Function Lex calls yylex
//-----------------------------------------------------------------
int Scan::Lex(void)
{  return tokencode=yylex();
}
//-----------------------------------------------------------------
//Function FetchSpelling returns a pointer to the spelling of the most
//recent token.
//-----------------------------------------------------------------
char* Scan::FetchSpelling(void)
{  return (char*)yytext;
}
//-----------------------------------------------------------------
//Function FetchTokenCode returns the code of the most recent token
//-----------------------------------------------------------------
int Scan::FetchTokenCode(void)
{  return tokencode;
}
//-----------------------------------------------------------------
//Function StoreTokenCode records the most recent token code
//-----------------------------------------------------------------
void Scan::StoreTokenCode(int T)
{  tokencode=T;
}
//----------------------End of Lex Definition-----------------------
```

Figure 9. File **Scan03.l (continued)**

File e00.exp

```
3 + 4 + 5
```

File e00.trc

```
Token(INTLIT,3,258)
Token(PLUS,+,259)
Token(INTLIT,4,258)
Token(PLUS,+,259)
Token(INTLIT,5,258)
```

File e01.exp
```
3 * (4 + 5)
```

File e02.exp
```
1-2/33
```

```
rm exp
rm *.o
rm explex.cpp
make -f makeexp
```

```
#---------------------------------------------------------------------
# File makeexp contains instructions for creating file exp
#---------------------------------------------------------------------
# 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   =  exp.o explex.o
#---------------------------------------------------------------------
exp:     ${obj}
      g++ -o exp ${obj} -ly -lm
#---------------------------------------------------------------------
# File exppar.cpp processes command line arguments
#---------------------------------------------------------------------
exp.o:     exp.cpp explex.h
      g++ -c -g exp.cpp
#---------------------------------------------------------------------
# File explex.l is the lexical analyzer
#---------------------------------------------------------------------
explex.cpp: explex.l
      lex explex.l
      mv lex.yy.c explex.cpp
#---------------------------------------------------------------------
# File explex.cpp is created by lex in the previous step
#---------------------------------------------------------------------
explex.o:  explex.cpp explex.h exptoken.h
      g++ -c -g explex.cpp
```

```
rm exp
rm *.o
rm explex.cpp
```

```
//-------------------------------------------------------------------
-
//File  exp.cpp  contains  functions  that  process  command  line
arguments
//and interface with the lex-generated scanner
//-------------------------------------------------------------------
--
//Author: Thomas R. Turner
//E-Mail: trturner@ucok.edu
//Date:   March, 2007
//-------------------------------------------------------------------
--
//Copyright March, 2007 by Thomas R. Turner
//Revised September, 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>
//-------------------------------------------------------------------
--
//Application include files
//-------------------------------------------------------------------
--
#include "explex.h"
//-------------------------------------------------------------------
--
//Namespaces
//-------------------------------------------------------------------
--
using namespace std;
//-------------------------------------------------------------------
--
//Function ScanMgr Scans the entire input file.
//-------------------------------------------------------------------
--
void ScanMgr(FILE* i)
{   Lexer L(i);
    for (;L.Scan();) ;
}

//-------------------------------------------------------------------
--
//Function main processes command line arguments
//-------------------------------------------------------------------
--
```

```
int main(int argc,char* argv[])
{   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:
          exit(EXIT_FAILURE);
    }
    FILE* i=fopen(ifn,"r");          //Open the input file
    ScanMgr(i);
    fclose(i);
    return 0;
}
```

```
%{
//---------------------------------------------------------------------
--
// File explex.l defines a prototype scanner for expressions.
// The scanner definition is a lex specification.
//---------------------------------------------------------------------
-
// Author: Thomas R. Turner
// E-Mail: trturner@ucok.edu
// Date:   November, 2006
//---------------------------------------------------------------------
--
//Copyright November, 2006 by Thomas R. Turner.
//Do not reproduce without permission from Thomas R. Turner
//---------------------------------------------------------------------
--
//---------------------------------------------------------------------
--
// Standard C and C++ Library Include Files
//---------------------------------------------------------------------
--
#include <string>
#include <iostream>
#include <iomanip>
#include <fstream>
#include <cstdio>
using namespace std;
//---------------------------------------------------------------------
--
// Application Includes
//---------------------------------------------------------------------
--
#include "explex.h"
#include "exptoken.h"
//---------------------------------------------------------------------
--
//Function prototypes
//---------------------------------------------------------------------
--
int TokenMgr(int t);
void TokenPrint(ostream& o,int t);
//---------------------------------------------------------------------
--
//Global Variables
//---------------------------------------------------------------------
--
static string spelling[]=

{"NOTOKEN","INTLIT","PLUS","MINUS","STAR","SLASH","LPAREN","RPAREN",
"ERROR"
  };
%}

%%
```

```
[ \t\n]+                                ;
[0-9]+                                  return TokenMgr(INTLIT);
"+"                                     return TokenMgr(PLUS);
"-"                                     return TokenMgr(MINUS);
"*"                                     return TokenMgr(STAR);
"/"                                     return TokenMgr(SLASH);
"("                                     return TokenMgr(LPAREN);
")"                                     return TokenMgr(RPAREN);
.                                       return TokenMgr(ERROR);
%%
//------------------------------------------------------------------
--
//Class Lexer implementation
//------------------------------------------------------------------
--
int yywrap(){return 1;}
int TokenMgr(int t)
{ TokenPrint(cout,t);
  return t;
}
void TokenPrint(ostream& o,int t)
{   o << endl;
    o << "Token("<< spelling[t-NOTOKEN]
      << "," << yytext
      << "," << t
      << ")";
    o << " ";
}
//------------------------------------------------------------------
--
//Constructor Lexer is used to redirect the input file stream from
the
//keyboard to input file stream i.
//------------------------------------------------------------------
--
Lexer::Lexer(FILE* i){yyin=i;}
int Lexer::Scan(void){return yylex();}
//----------------------End of Lex Definition---------------------
--
```

```
#ifndef explex_h
#define explex_h 1
//-------------------------------------------------------------------
// File explex.h defines class Lexer.
//-------------------------------------------------------------------
// Author: Thomas R. Turner
// E-Mail: trturner.ucok.edu
// Date:   November, 2006
//-------------------------------------------------------------------
// Copyright November, 2006 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 expner.  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 Scan(void);
};
#endif
```

```
#ifndef YYERRCODE
#define YYERRCODE 256
#endif

#define NOTOKEN 257
#define INTLIT 258
#define PLUS 259
#define MINUS 260
#define STAR 261
#define SLASH 262
#define LPAREN 263
#define RPAREN 264
#define ERROR 265
```