```
#include <iostream>
#include <fstream>
#include <cctype>
#include <string>
/* Global declarations */
/* Variables */
int charClass;
std::string lexeme;
char nextChar;
int lexLen;
int token;
int nextToken;
std::ifstream in
_fp;
/* Function declarations */
void addChar();
void getChar();
void getNonBlank();
int lex();
/* Character classes */
#define LETTER 0
#define DIGIT 1
#define UNKNOWN 99
/* Token codes */
#define INT
LIT 10
_
#define IDENT 11#define ASSIGN
OP 20
_
#define ADD
OP 21
_
#define SUB
OP 22
_
#define MULT
OP 23
```

_
#define DIV
_
OP 24
#define LEFT
_
PAREN 25
#define RIGHT
_
PAREN 26
```
/*****************************************************/
/* main driver */
int main() {
/* Open the input data file and process its contents */
in
_fp.open("front.in");
if (!in
_fp) {
std::cerr << "ERROR - cannot open front.in\n";
return 1; // Return with an error code
} else {
getChar();
do {
lex();
} while (nextToken != EOF);
}
in
_fp.close(); // Close the file
return 0; // Successful execution
}
/*****************************************************/
/* lookup - a function to lookup operators and parentheses
and return the token */
int lookup(char ch) {
switch (ch) {
case '(':
addChar();
nextToken = LEFT
```

```
_PAREN;
break;
case ')':case ')':
addChar();
nextToken = RIGHT
_PAREN;
break;
case '+':
addChar();
nextToken = ADD
_OP;
break;
case '-':
addChar();
nextToken = SUB
_OP;
break;
case '*':
addChar();
nextToken = MULT
_OP;
break;
case '/':
addChar();
nextToken = DIV
_OP;
break;
default:
addChar();
nextToken = EOF;
break;
}
return nextToken;
}
/*****************************************************/
/* addChar - a function to add nextChar to lexeme */
void addChar() {
if (lexLen <= 98) {
```

```
lexeme += nextChar;
lexLen++;
} else {
std::cerr << "Error - lexeme is too long \n";
}
}
/*****************************************************//
*****************************************************/
/* getChar - a function to get the next character of
input and determine its character class */
void getChar() {
if (in_fp.get(nextChar)) {
if (isalpha(nextChar))
charClass = LETTER;
else if (isdigit(nextChar))
charClass = DIGIT;
else
charClass = UNKNOWN;
} else {
charClass = EOF;
}
}
/*****************************************************/
/* getNonBlank - a function to call getChar until it
returns a non-whitespace character */
void getNonBlank() {
while (isspace(nextChar))
getChar();
}
/*****************************************************/
/* lex - a simple lexical analyzer for arithmetic
expressions */
int lex() {
lexLen = 0;
lexeme.clear(); // Clear the lexeme
getNonBlank();
switch (charClass) {
/* Parse identifiers */
```

```
case LETTER:
addChar();
getChar();
while (charClass == LETTER || charClass == DIGIT) {
addChar();addChar();
getChar();
}
nextToken = IDENT;
break;
/* Parse integer literals */
case DIGIT:
addChar();
getChar();
while (charClass == DIGIT) {
addChar();
getChar();
}
nextToken = INT
_LIT;
break;
/* Parentheses and operators */
case UNKNOWN:
lookup(nextChar);
getChar();
break;
/* EOF */
case EOF:
nextToken = EOF;
lexeme = "EOF";
break;
} /* End of switch */
std::cout << "Next token is: " << nextToken << ", Next lexeme is " << lexeme << std::endl;
return nextToken;
} /* End of function lex */
```