



```
1  #include <stdio.h>
2  #include <ctype.h>
3
4  /* Global declarations */
5
6  /* Variables */
7  int charClass;
8  char lexeme[100];
9  char nextChar;
10 int lexLen;
11 int token;
12 int nextToken;
13 FILE *in_fp;
14
15 /* Function declarations */
16 void addChar();
17 void getChar();
18 void getNonBlank();
19 int lex();
20 int lookup(char ch);
21
22 /* Recursive functions */
23 void expr();
24 void term();
25 void factor();
26 void error();
27
28 /* Character classes */
29 #define LETTER 0
30 #define DIGIT 1
31 #define UNKNOWN 99
32
33 /* Token codes */
34 #define INT_LIT 10
35 #define IDENT 11
36 #define ASSIGN_OP 20
37 #define ADD_OP 21
38 #define SUB_OP 22
39 #define MULT_OP 23
40 #define DIV_OP 24
41 #define LEFT_PAREN 25
42 #define RIGHT_PAREN 26
43 /*****
44  /* main driver */
45  int main(int argc, char *argv[]) {
46      if (argc != 2) {
47          printf("Usage: %s <filename>\n", argv[0]);
48          return 1;
49      }
50
51      if ((in_fp = fopen(argv[1], "r")) == NULL) {
52          printf("ERROR - cannot open %s\n", argv[1]);
53          return 1;
54      }
55
56      getChar();
57      do {
58          lex();
59          expr();
60      } while (nextToken != EOF);
61
62      fclose(in_fp);
63      return 0;
64  }
65
66  void expr() {
67      printf("Enter <expr>\n");
68      term();
69      while (nextToken == ADD_OP || nextToken == SUB_OP) {
70          lex();
71          term();
72      }
73      printf("Exit <expr>\n");
74  }
75
76  void term() {
77      printf("Enter <term>\n");
78      factor();
79      while (nextToken == MULT_OP || nextToken == DIV_OP) {
80          lex();
81          factor();
82      }
83      printf("Exit <term>\n");
84  }
85
86  void factor() {
87      printf("Enter <factor>\n");
88      if (nextToken == INT_LIT || nextToken == IDENT)
89          lex();
90      else if (nextToken == LEFT_PAREN) {
91          lex();
92          expr();
93          if (nextToken == RIGHT_PAREN)
94              lex();
95          else
96              error();
97      } else
98          error();
99      printf("Exit <factor>\n");
100 }
101
102 void error() {
103     printf("SYNTAX ERROR\n");
104 }
105
106 /*****
107  /* lookup - a function to lookup operators and parentheses
108  and return the token */
109  int lookup(char ch)
110  {
111      switch (ch)
112      {
113          case '(':
114              addChar();
115              nextToken = LEFT_PAREN;
116              break;
117          case ')':
118              addChar();
119              nextToken = RIGHT_PAREN;
120              break;
121          case '+':
122              addChar();
123              nextToken = ADD_OP;
124              break;
125          case '-':
126              addChar();
127              nextToken = SUB_OP;
128              break;
129          case '*':
130              addChar();
131              nextToken = MULT_OP;
132              break;
133          case '/':
134              addChar();
135              nextToken = DIV_OP;
136              break;
137          default:
138              addChar();
139              nextToken = UNKNOWN;
140              break;
141      }
142      return nextToken;
143  }
144  /*****
145  /* addChar - a function to add nextChar to lexeme */
146  void addChar()
147  {
148      if (lexLen <= 98)
149      {
150          lexeme[lexLen++] = nextChar;
151          lexeme[lexLen] = '\0';
152      }
153      else
154          printf("Error - lexeme is too long \n");
155  }
156  /*****
157  /* getChar - a function to get the next character of
158  input and determine its character class */
159  void getChar()
160  {
161      if ((nextChar = getc(in_fp)) != EOF)
162      {
163          if (isalpha(nextChar))
164              charClass = LETTER;
165          else if (isdigit(nextChar))
166              charClass = DIGIT;
167          else
168              charClass = UNKNOWN;
169      }
170      else
171          charClass = EOF;
172  }
173  /*****
174  /* getNonBlank - a function to call getChar until it
175  returns a non-whitespace character */
176  void getNonBlank()
177  {
178      while (isspace(nextChar))
179          getChar();
180  }
181  /*
182  *****/
183  /* lex - a simple lexical analyzer for arithmetic
184  expressions */
185  int lex()
186  {
187      lexLen = 0;
188      getNonBlank();
189      switch (charClass)
190      {
191          /* Parse identifiers */
192          case LETTER:
193              addChar();
194              getChar();
195              while (charClass == LETTER || charClass == DIGIT)
196              {
197                  addChar();
198                  getChar();
199              }
200              nextToken = IDENT;
201              break;
202          /* Parse integer literals */
203          case DIGIT:
204              addChar();
205              getChar();
206              while (charClass == DIGIT)
207              {
208                  addChar();
209                  getChar();
210              }
211              nextToken = INT_LIT;
212              break;
213          /* Parentheses and operators */
214          case UNKNOWN:
215              lookup(nextChar);
216              getChar();
217              break;
218          /* EOF */
219          case EOF:
220              nextToken = EOF;
221              lexeme[0] = 'E';
222              lexeme[1] = '0';
223              lexeme[2] = 'F';
224              lexeme[3] = '\0';
225              break;
226      } /* End of switch */
227      printf("Next token is: %d, Next lexeme is %s\n",
228             nextToken, lexeme);
229      return nextToken;
230  } /* End of function lex */
231
```



```
PS C:\Users\amnes\Documents\GitHub\CECS-342\lab2> ./lab2 Test1.txt
Next token is: 25, Next lexeme is (
Enter <expr>
Enter <term>
Enter <factor>
Next token is: 11, Next lexeme is sum
Enter <expr>
Enter <term>
Enter <factor>
Next token is: 21, Next lexeme is +
Exit <factor>
Exit <term>
Next token is: 10, Next lexeme is 47
Enter <term>
Enter <factor>
Next token is: 26, Next lexeme is )
Exit <factor>
Exit <term>
Next token is: 24, Next lexeme is /
Exit <factor>
Next token is: 11, Next lexeme is total
Enter <factor>
Next token is: -1, Next lexeme is EOF
Exit <factor>
Exit <term>
Exit <expr>
```

PS C:\Users\amnes\Documents\GitHub\CECS-342\lab2> ./lab2 Test2.txt

Next token is: 10, Next lexeme is 50

Enter <expr>

Enter <term>

Enter <factor>

Next token is: 10, Next lexeme is 47

Exit <factor>

Exit <term>

Exit <expr>

Next token is: 24, Next lexeme is /

Enter <expr>

Enter <term>

SYNTAX ERROR

Exit <factor>

Next token is: 11, Next lexeme is x

Enter <factor>

Next token is: -1, Next lexeme is EOF

Exit <factor>

Exit <term>

Exit <expr>

```
PS C:\Users\amnes\Documents\GitHub\CECS-342\lab2> ./lab2 Test3.txt
Next token is: 25, Next lexeme is (
Enter <expr>
Enter <term>
Enter <factor>
Next token is: 11, Next lexeme is sum
Enter <expr>
Enter <term>
Enter <factor>
Next token is: 21, Next lexeme is +
Exit <factor>
Exit <term>
Next token is: 10, Next lexeme is 47
Enter <term>
Enter <factor>
Next token is: 24, Next lexeme is /
Exit <factor>
Next token is: 11, Next lexeme is total
Enter <factor>
Next token is: -1, Next lexeme is EOF
Exit <factor>
Exit <term>
Exit <expr>
SYNTAX ERROR
Exit <factor>
Exit <term>
Exit <expr>
```