Program Code

Lexical.c

```
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "functions.h"

#define file "input.c"

#define debug(x) puts(x)
static int comment_flag = 0;
// #define debug(x) ;

/*Hello World
Good Morning*/

FILE *fp;

int handle_line(char *line) {
  char line_copy[1024];
  strncpy(line_copy, line, 1024);
  int lit_flag = 0;
  int lit_type = 1;  // 1- "  &&  2-'
  char lexeme[1024] = "";
  for (int i = 0; i < strlen(line); i++) {
    char curr_char = line[i];

    // handling multiline comment
    if (i + 1 < strlen(line) && (line[i] == '/' && line[i + 1] ==
'*')) {
      comment_flag = 1;
      fprintf(fp,"multiline comment");
    }
    if (i + 1 < strlen(line) && (line[i] == '*' && line[i + 1] ==
'/')) {
      comment_flag = 0;
      fprintf(fp,"multiline comment");
      lexeme[0]='\0';
      break;
    }
    // if still inside comment
    if (comment_flag == 1) {
      continue;
    }

    // handling single line comment
    if (i + 1 < strlen(line) && (line[i] == '/' && line[i + 1] ==
'/')) {
      fprintf(fp,"Single line comment");
      break;
    }

    // handling string literals
    if (strlen(lexeme) < 1 && lit_flag == 0) {
      if (curr_char == '"') {
        lit_type = 1;
        lit_flag = 1;

      } else if (curr_char == '\'') {
        lit_type = 2;
        lit_flag = 1;
      }
    } else if (lit_flag == 1 && ((lit_type == 1 && curr_char ==
'"') || (lit_type == 2 && curr_char == '\''))) {
      lit_flag = 0;
      int len = strlen(lexeme);
      lexeme[len] = curr_char;
      lexeme[len + 1] = '\0';

      // changes made to print literals and " seperately
      if (lexeme[len] == '"') {
        fprintf(fp,"<%s,%s>\n", "\"", "symbol");
        for (int i = 0; i < len + 1; i++) {
          lexeme[i] = lexeme[i + 1];
        }
        lexeme[len - 1] = '\0';
        fprintf(fp,"<%s,%s>\n", lexeme, "literal");
        fprintf(fp,"<%s,%s>\n", "\"", "symbol");
      }

      // fprintf(fp,"<%s,%s>\n", lexeme, "literal");
      else if(lexeme[len]=='\'')
    {
     fprintf(fp,"<%s,%s>\n", "\"", "symbol");
        for (int i = 0; i < len + 1; i++) {
          lexeme[i] = lexeme[i + 1];
        }
        lexeme[len - 1] = '\0';
        fprintf(fp,"<%s,%s>\n", lexeme, "literal");
        fprintf(fp,"<%s,%s>\n", "\"", "symbol");
      }
      lexeme[0] = '\0';
      continue;
    }


    // checking if encountered a delimiter outside a string
literal
    L1:if (lit_flag == 0 && (isSpaces(curr_char) == 1 ||
isDelim(curr_char) == 1 ||
                          isOperator(curr_char) == 1)) {
      if (isKeyword(lexeme) == 1) {
        fprintf(fp,"<%s,%s>\n", lexeme, "keyword");
      } else if (isIdentifier(lexeme) == 1) {
        fprintf(fp,"<%s,%s>\n", lexeme, "identifier");
      } else if (isInteger(lexeme) == 1) {
        fprintf(fp,"<%s,%s>\n", lexeme, "integer");
      } else if (strlen(lexeme) > 0) {
        if (isOperator2(lexeme) == 1) {
          fprintf(fp,"<%s,%s>\n", lexeme, "operator");
        } else {
          fprintf(fp,"<%s,%s>\n", lexeme, "invalid identifier");
        }
      }
      if (isSpaces(curr_char) == 0) {
        if (isOperator(curr_char) == 1) {
          fprintf(fp,"<%c,%s>\n", curr_char, "operator");
        } else {
          fprintf(fp,"<%c,%s>\n", curr_char, "symbol");
        }
      }
      lexeme[0] = '\0';
    } else {
      // append to lexeme until a delimiter is encountered
      int len = strlen(lexeme);
      lexeme[len] = curr_char;
      lexeme[len + 1] = '\0';
    }
  }
}

int main() {
  FILE *f1;
  f1 = fopen(file, "r");
  fseek(f1, 0, SEEK_SET);
  fp=fopen("lex.txt","w");

  int c=1;
  char line[1024];
  while (fgets(line, 1024, f1)) {
    fprintf(fp,"\n%d. ",c++);
    handle_line(line);
  };
  // fclose(file);
  return 0;
}
```

<u>fucntions.h</u>

```
#define KW_LEN 32
const char *keywords[] = {
    "auto",   "break",  "case",     "char",    "continue", "do",
"default",
    "const",  "double", "else",     "enum",    "extern",   "for",
"if",
    "goto",   "float",  "int",      "long",    "register",
"return",  "signed",
    "static", "sizeof", "short",    "struct",  "switch",
"typedef", "union",
    "void",   "while",  "volatile", "unsigned", "FILE"};

int isKeyword(char *lexeme) {
  for (int i = 0; i < KW_LEN; i++) {
    if (strncmp(lexeme, keywords[i], 10) == 0) return 1;
  }
  return 0;
}

#define OP_LEN 11
const char operators[] = {'-', '+', '/', '*', '#', '=',
                          '&', '!', '|', '^', '%', '\0'};

int isOperator(char lexeme) {
  for (int i = 0; i < sizeof(operators); i++) {
    if (lexeme == operators[i]) return 1;
  }
  return 0;
}

#define OP_LEN2 6
const char *operators2[] = {"&&", "||", "==", ">=", "<=", "!=",
"-", "+", "/",
                            "*",  "#",  "=",  "&",  "!",  "|",
"^", "%"};

int isOperator2(char *lexeme) {
  int len = strlen(lexeme);
  if (len != 2) return 0;
  for (int i = 0; i < OP_LEN2; i++) {
    if (lexeme[0] == operators2[i][0] && lexeme[1] ==
operators2[i][1])
      return 1;
  }
  return 0;
}

int isIdentifier(char *lexeme) {
  if (isdigit(lexeme[0]) || strlen(lexeme) < 1) {
    return 0;
  }
  for (int i = 1; i < strlen(lexeme); i++) {
    if (!isalpha(lexeme[i]) && !isdigit(lexeme[i]) && lexeme[i] !=
'_')
      return 0;
  }
  return 1;
}

int isInteger(char *lexeme) {
  if (strlen(lexeme) < 1) return 0;

  for (int i = 0; i < strlen(lexeme); i++) {
    if (lexeme[i] < '0' || lexeme[i] > '9') return 0;
  }
  return 1;
}

#define DEL_LEN 10
const char delimeters[] = {'{', '}', '[', ']', '(',  ')',
                           '<', '>', ';', ',', '\0', '\n'};

int isDelim(char lexeme) {
  for (int i = 0; i < DEL_LEN; i++) {
    if (lexeme == delimeters[i]) return 1;
  }
  return 0;
}

int isSpaces(char lexeme) {
  if (lexeme == ' ' || lexeme == '\n' || lexeme == '\t')
    return 1;
  else
    return 0;
}
```

Output

```
1. <#,operator>
<include,identifier>
<<,symbol>
<ctype.h,invalid identifier>
<>,symbol>

2. <#,operator>
<include,identifier>
<<,symbol>
<stdio.h,invalid identifier>
<>,symbol>

3. <#,operator>
<include,identifier>
<<,symbol>
<stdlib.h,invalid identifier>
<>,symbol>

4. <#,operator>
<include,identifier>
<<,symbol>
<string.h,invalid identifier>
<>,symbol>

5.
6. <#,operator>
<include,identifier>
<",symbol>
<functions.h,literal>
<",symbol>

7.
8. <#,operator>
<define,identifier>
<file,identifier>
<",symbol>
<input.c,literal>
<",symbol>

9.
10. <#,operator>
<define,identifier>
<debug,identifier>
<(,symbol>
<x,identifier>
<),symbol>
<puts,identifier>
<(,symbol>
<x,identifier>
<),symbol>

11. <static,keyword>
<int,keyword>
<comment_flag,identifier>
<=,operator>
<0,integer>
<;,symbol>
```

```
12. Single line comment
13.
14. multiline comment
15. multiline comment
16.
                                 .
                                 .
                                 .
                                 .
                                 .
                                 .
                                 .

136. <while,keyword>
<(,symbol>
<fgets,identifier>
<(,symbol>
<line,identifier>
<,,symbol>
<1024,integer>
<,,symbol>
<f1,identifier>
<),symbol>
<),symbol>
<{,symbol>

137. <fprintf,identifier>
<(,symbol>
<fp,identifier>
<,,symbol>
<",symbol>
<\n%d. ,literal>
<",symbol>
<,,symbol>
<c,identifier>
<+,operator>
<+,operator>
<),symbol>
<;,symbol>

138. <handle_line,identifier>
<(,symbol>
<line,identifier>
<),symbol>
<;,symbol>

139. <},symbol>
<;,symbol>

140. Single line comment
141. <return,keyword>
<0,integer>
<;,symbol>

142. <},symbol>
```