PRACTICAL 1

Design a lexical analyzer for given language and the lexical analyzer should ignore redundant spaces, tabs and new lines. It should also ignore comments. Although the syntax specification states that identifiers can be arbitrarily long, you may restrict the length to some reasonable value. Simulate the same in C language.

Code: lexer.c

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

#define MAX_IDENTIFIER_LENGTH 32
#define MAX_TOKEN_LENGTH 64

typedef enum {
    TOKEN_IDENTIFIER,
    TOKEN_NUMBER,
    TOKEN_OPERATOR,
    TOKEN_KEYWORD,
    TOKEN_EOF,
    TOKEN_UNKNOWN
} TokenType;

typedef struct {
    TokenType type;
    char value[MAX_TOKEN_LENGTH];
} Token;

const char *keywords[] = {"int", "for", "return", "#include", NULL};

int is_keyword(const char *identifier) {
    for (int i = 0; keywords[i] != NULL; i++) {
        if (strcmp(identifier, keywords[i]) == 0) {
            return 1;
        }
    }
    return 0;
}

void skip_whitespace_and_comments(FILE *source) {
    char ch;
    while ((ch = fgetc(source)) != EOF) {
        if (isspace(ch)) {
            continue;
        }
        if (ch == '/') {
            char next = fgetc(source);
            if (next == '/') {
                while ((ch = fgetc(source)) != '\n' && ch != EOF);
            } else if (next == '*') {
                while (1) {
```

```
                ch = fgetc(source);
                if (ch == '*' && (next = fgetc(source)) == '/') {
                    break;
                }
                if (ch == EOF) {
                    break;
                }
            }
        } else {
            ungetc(next, source);
            break;
        }
    } else {
        ungetc(ch, source);
        break;
    }
  }
}

Token get_next_token(FILE *source) {
    Token token;
    token.type = TOKEN_UNKNOWN;
    token.value[0] = '\0';

    skip_whitespace_and_comments(source);

    char ch = fgetc(source);
    if (ch == EOF) {
        token.type = TOKEN_EOF;
        return token;
    }

    if (isalpha(ch) || ch == '#') {
        int index = 0;
        do {
            if (index < MAX_IDENTIFIER_LENGTH - 1) {
                token.value[index++] = ch;
            }
            ch = fgetc(source);
        } while (isalnum(ch) || ch == '_');
        token.value[index] = '\0';
        token.type = is_keyword(token.value) ? TOKEN_KEYWORD : TOKEN_IDENTIFIER;
    } else if (isdigit(ch)) {
        int index = 0;
        do {
            if (index < MAX_TOKEN_LENGTH - 1) {
                token.value[index++] = ch;
            }
            ch = fgetc(source);
        } while (isdigit(ch));
        token.value[index] = '\0';
        token.type = TOKEN_NUMBER;
    } else if (strchr("+-*/=<>;", ch)) {
```

```
        token.value[0] = ch;
        token.value[1] = '\0';
        token.type = TOKEN_OPERATOR;
    } else {
        token.value[0] = ch;
        token.value[1] = '\0';
        token.type = TOKEN_UNKNOWN;
    }

    ungetc(ch, source);
    return token;
}

void print_token(Token token) {
    switch (token.type) {
        case TOKEN_IDENTIFIER:
            printf("IDENTIFIER: %s\n", token.value);
            break;
        case TOKEN_NUMBER:
            printf("NUMBER: %s\n", token.value);
            break;
        case TOKEN_OPERATOR:
            printf("OPERATOR: %s\n", token.value);
            break;
        case TOKEN_KEYWORD:
            printf("KEYWORD: %s\n", token.value);
            break;
        case TOKEN_EOF:
            printf("EOF\n");
            break;
        case TOKEN_UNKNOWN:
        default:
            printf("UNKNOWN: %s\n", token.value);
            break;
    }
}

int main() {
    FILE *source = fopen("source.txt", "r");
    if (!source) {
        perror("Could not open source file");
        return EXIT_FAILURE;
    }

    Token token;
    do {
        token = get_next_token(source);
        print_token(token);
    } while (token.type != TOKEN_EOF);

    fclose(source);
    return EXIT_SUCCESS;
}
```

Input File: source.txt

```
#include <stdio.h>

int main() {
    int count = 0;
    for (int i = 0; i < 10; i++) {
        count += i;
    }
    return count;
}
```

Output:

```
KEYWORD: #include
IDENTIFIER: stdio.h
KEYWORD: int
IDENTIFIER: main
OPERATOR: (
KEYWORD: int
IDENTIFIER: count
OPERATOR: =
NUMBER: 0
OPERATOR: ;
KEYWORD: for
OPERATOR: (
KEYWORD: int
IDENTIFIER: i
OPERATOR: =
NUMBER: 0
OPERATOR: ;
IDENTIFIER: i
OPERATOR: <
NUMBER: 10
OPERATOR: ;
IDENTIFIER: i
OPERATOR: ++
OPERATOR: )
OPERATOR: {
IDENTIFIER: count
OPERATOR: +=
IDENTIFIER: i
OPERATOR: ;
OPERATOR: }
KEYWORD: return
IDENTIFIER: count
OPERATOR: ;
OPERATOR: }
```