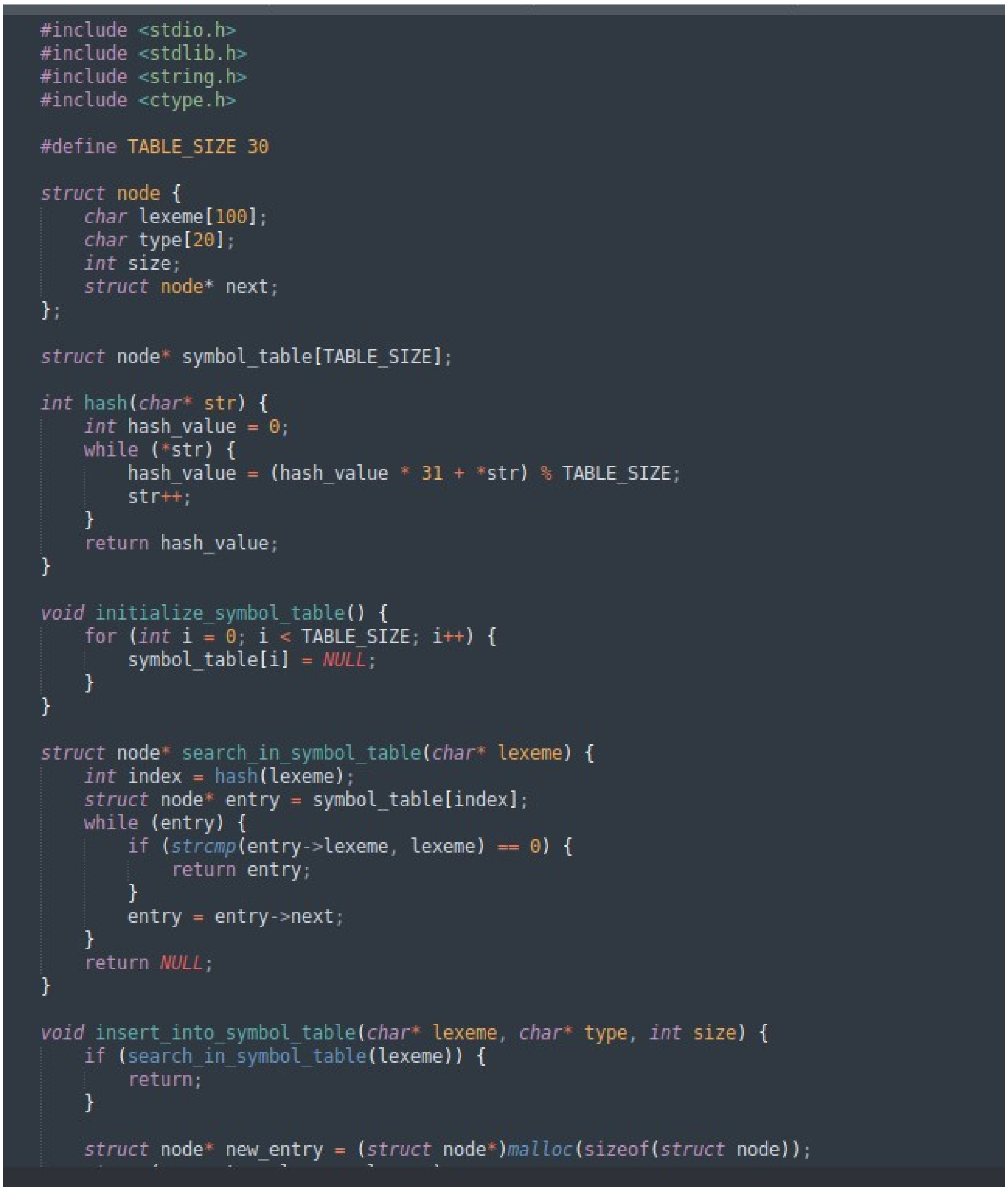
NAME:AYANIKA PAUL
Roll No. 22
D1

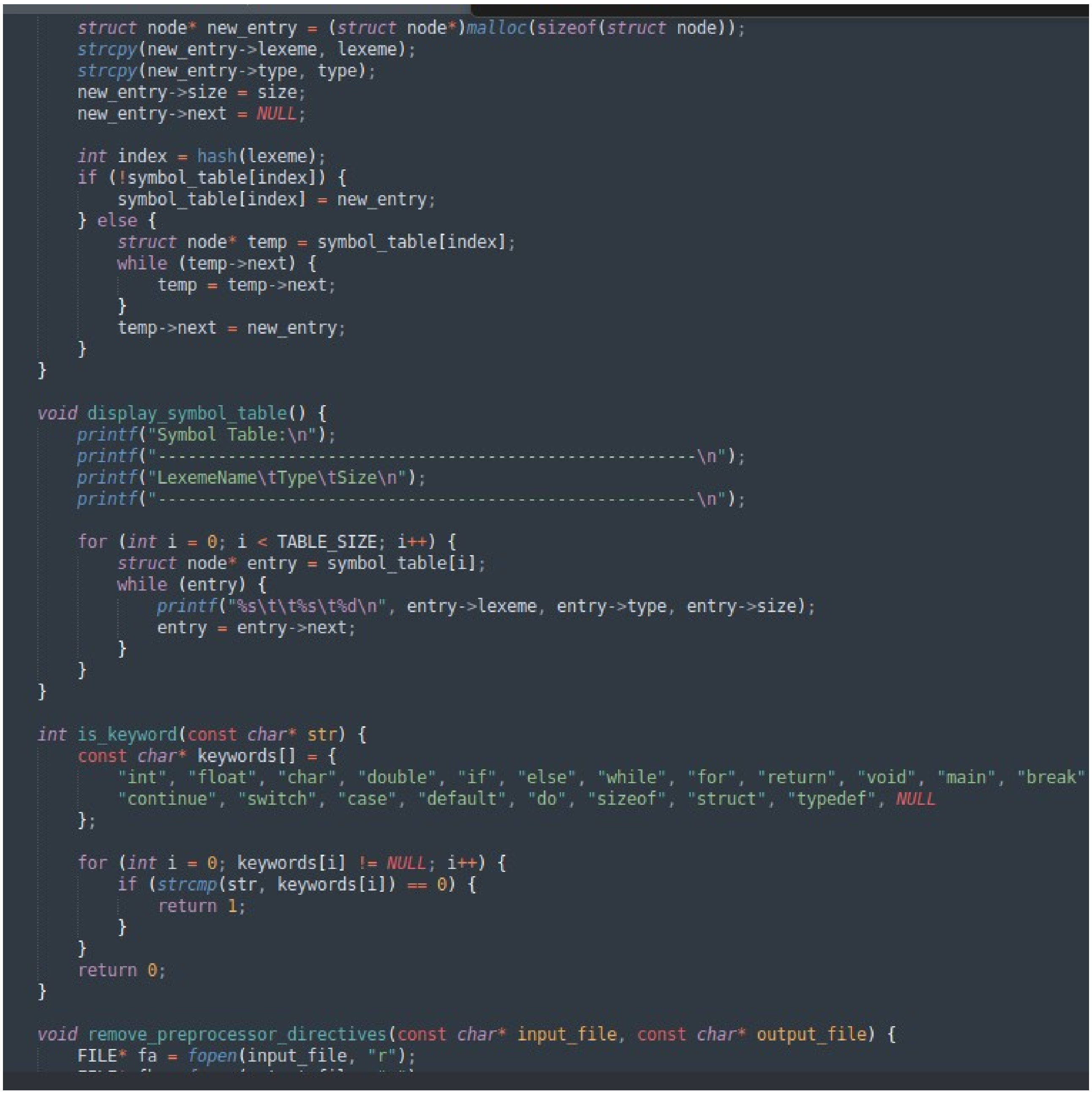
LAB 4

1. Using getNextToken( ) implemented in Lab No 3, design a Lexical Analyser to implement the following symbol tables.
a. local symbol table

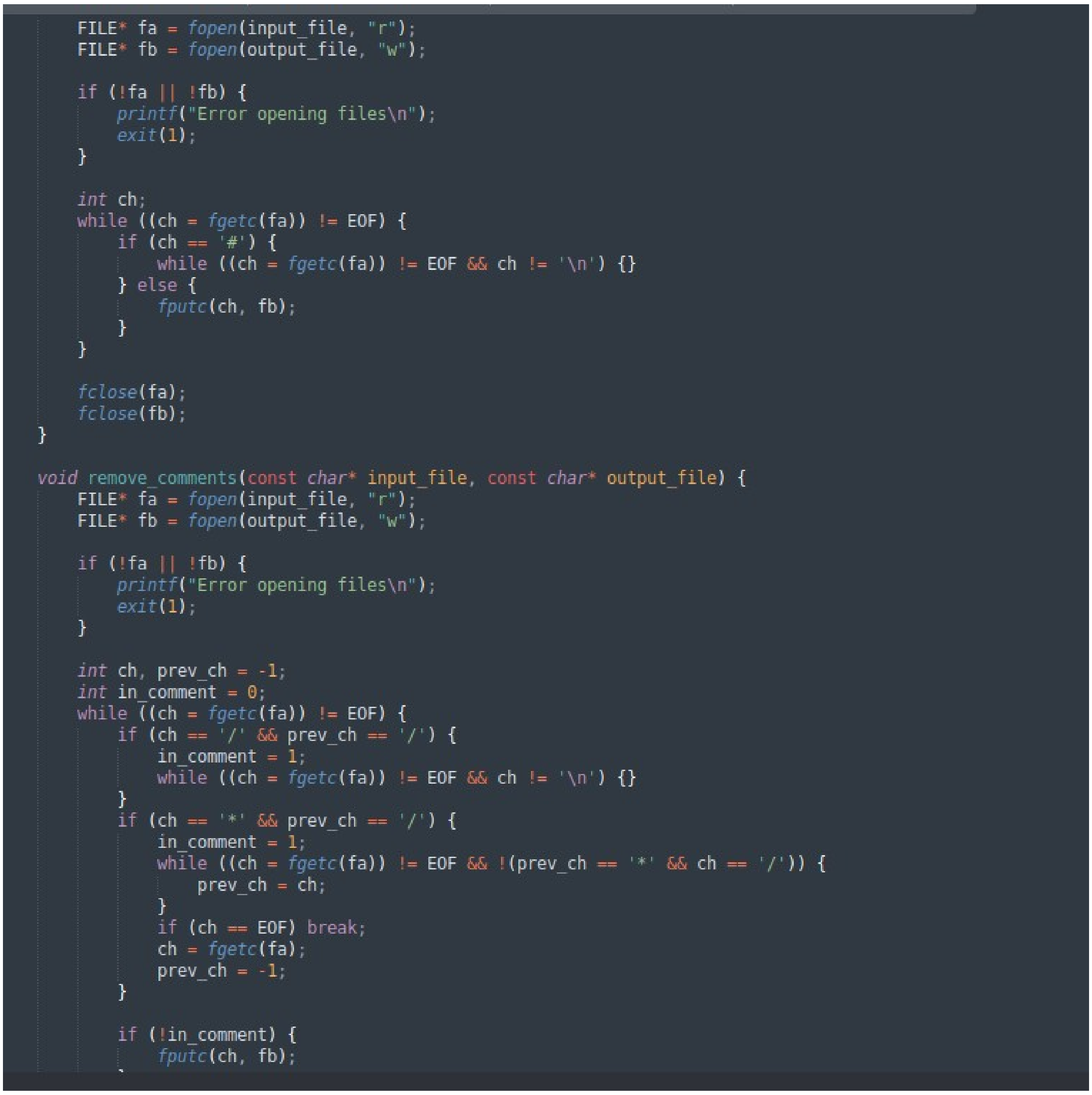
CODE:



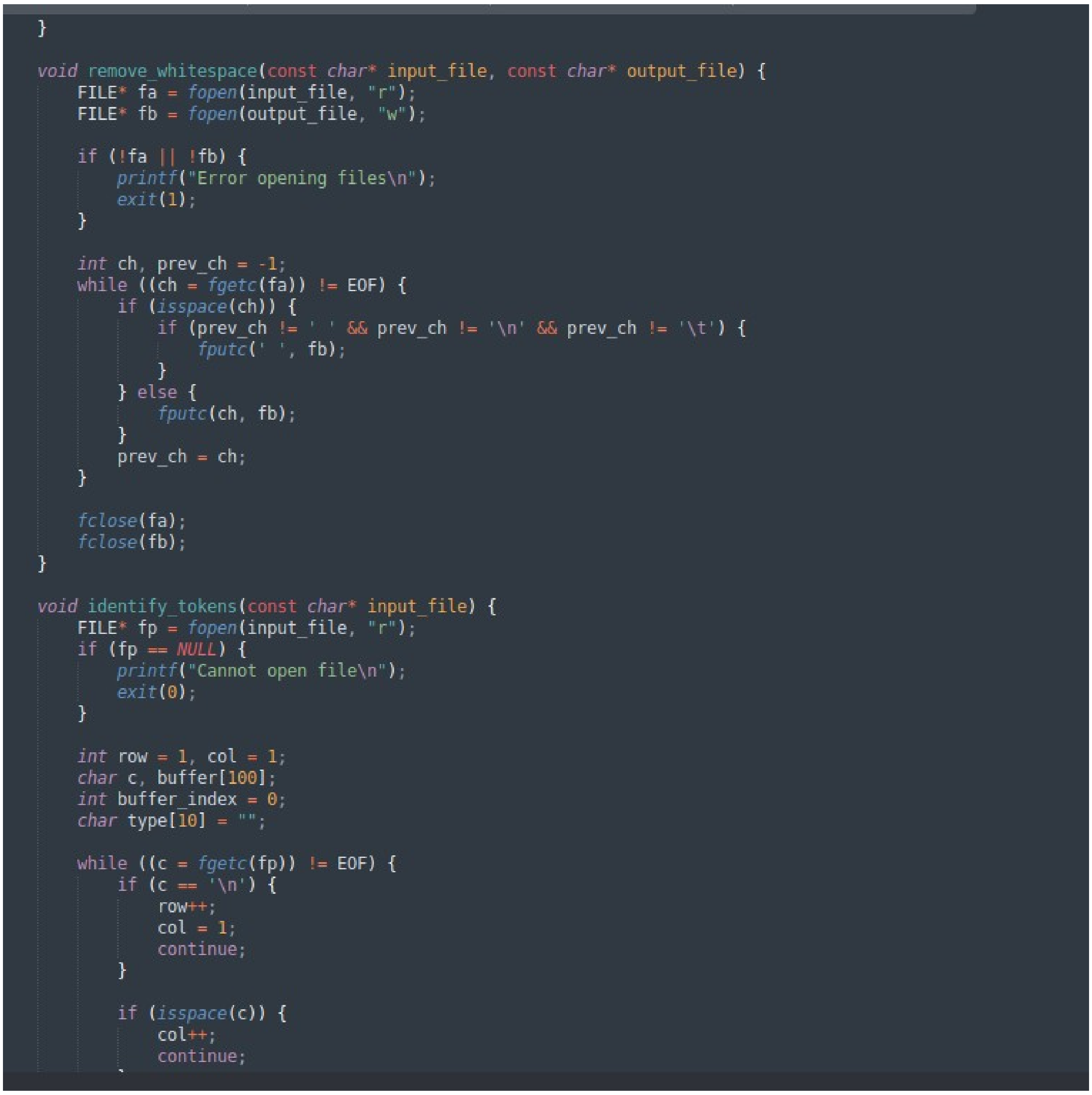
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#define TABLE_SIZE 30

struct node {
    char lexeme[100];
    char type[20];
    int size;
    struct node* next;
};

struct node* symbol_table[TABLE_SIZE];

int hash(char* str) {
    int hash_value = 0;
    while (*str) {
        hash_value = (hash_value * 31 + *str) % TABLE_SIZE;
        str++;
    }
    return hash_value;
}

void initialize_symbol_table() {
    for (int i = 0; i < TABLE_SIZE; i++) {
        symbol_table[i] = NULL;
    }
}

struct node* search_in_symbol_table(char* lexeme) {
    int index = hash(lexeme);
    struct node* entry = symbol_table[index];
    while (entry) {
        if (strcmp(entry->lexeme, lexeme) == 0) {
            return entry;
        }
        entry = entry->next;
    }
    return NULL;
}

void insert_into_symbol_table(char* lexeme, char* type, int size) {
    if (search_in_symbol_table(lexeme)) {
        return;
    }

    struct node* new_entry = (struct node*)malloc(sizeof(struct node));
```



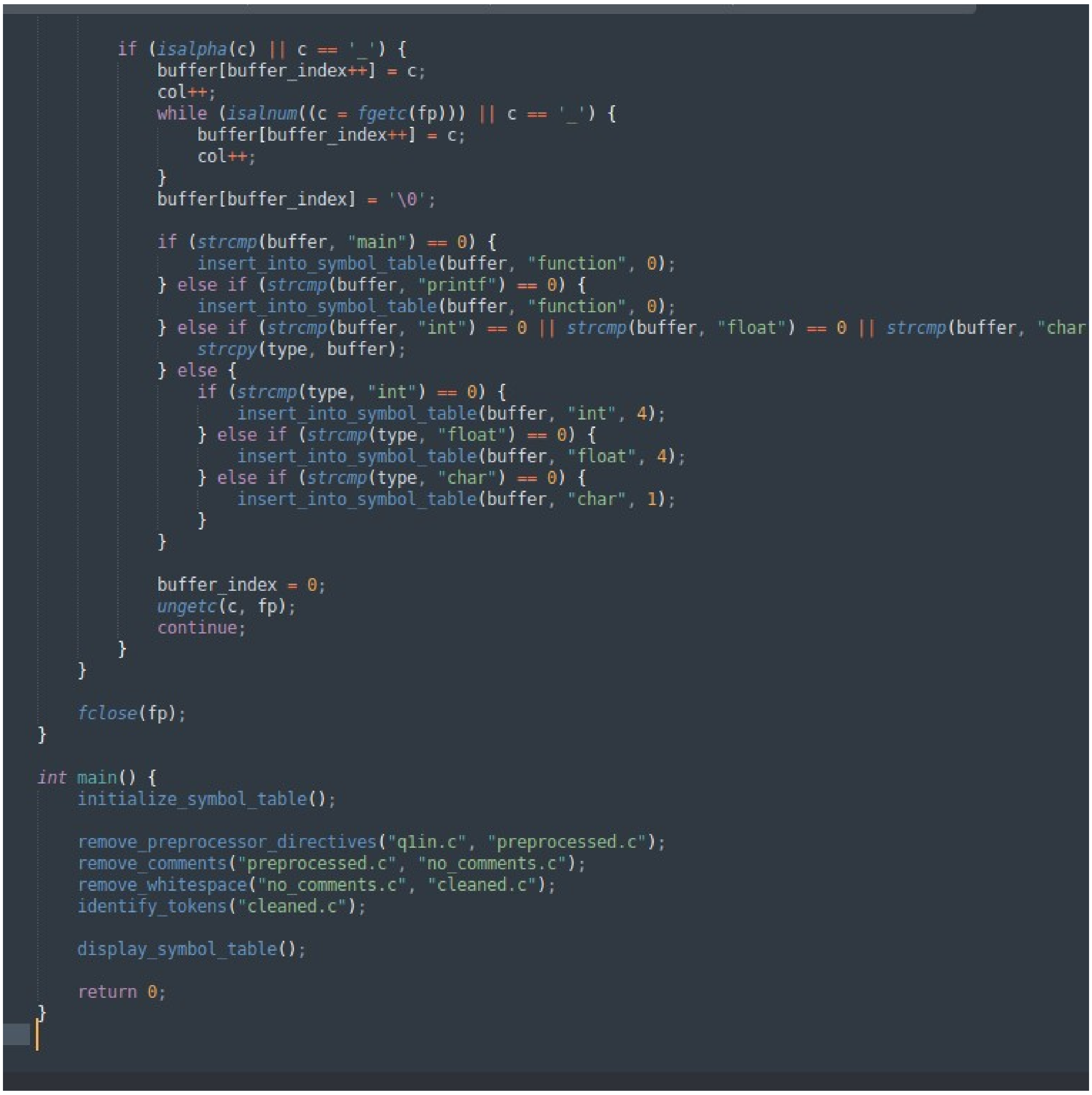
```c
    struct node* new_entry = (struct node*)malloc(sizeof(struct node));
    strcpy(new_entry->lexeme, lexeme);
    strcpy(new_entry->type, type);
    new_entry->size = size;
    new_entry->next = NULL;

    int index = hash(lexeme);
    if (!symbol_table[index]) {
        symbol_table[index] = new_entry;
    } else {
        struct node* temp = symbol_table[index];
        while (temp->next) {
            temp = temp->next;
        }
        temp->next = new_entry;
    }
}

void display_symbol_table() {
    printf("Symbol Table:\n");
    printf("------------------------------------------------------\n");
    printf("LexemeName\tType\tSize\n");
    printf("------------------------------------------------------\n");

    for (int i = 0; i < TABLE_SIZE; i++) {
        struct node* entry = symbol_table[i];
        while (entry) {
            printf("%s\t\t%s\t%d\n", entry->lexeme, entry->type, entry->size);
            entry = entry->next;
        }
    }
}

int is_keyword(const char* str) {
    const char* keywords[] = {
        "int", "float", "char", "double", "if", "else", "while", "for", "return", "void", "main", "break"
        "continue", "switch", "case", "default", "do", "sizeof", "struct", "typedef", NULL
    };

    for (int i = 0; keywords[i] != NULL; i++) {
        if (strcmp(str, keywords[i]) == 0) {
            return 1;
        }
    }
    return 0;
}

void remove_preprocessor_directives(const char* input_file, const char* output_file) {
    FILE* fa = fopen(input_file, "r");
```



```c
    FILE* fa = fopen(input_file, "r");
    FILE* fb = fopen(output_file, "w");

    if (!fa || !fb) {
        printf("Error opening files\n");
        exit(1);
    }

    int ch;
    while ((ch = fgetc(fa)) != EOF) {
        if (ch == '#') {
            while ((ch = fgetc(fa)) != EOF && ch != '\n') {}
        } else {
            fputc(ch, fb);
        }
    }

    fclose(fa);
    fclose(fb);
}

void remove_comments(const char* input_file, const char* output_file) {
    FILE* fa = fopen(input_file, "r");
    FILE* fb = fopen(output_file, "w");

    if (!fa || !fb) {
        printf("Error opening files\n");
        exit(1);
    }

    int ch, prev_ch = -1;
    int in_comment = 0;
    while ((ch = fgetc(fa)) != EOF) {
        if (ch == '/' && prev_ch == '/') {
            in_comment = 1;
            while ((ch = fgetc(fa)) != EOF && ch != '\n') {}
        }
        if (ch == '*' && prev_ch == '/') {
            in_comment = 1;
            while ((ch = fgetc(fa)) != EOF && !(prev_ch == '*' && ch == '/')) {
                prev_ch = ch;
            }
            if (ch == EOF) break;
            ch = fgetc(fa);
            prev_ch = -1;
        }

        if (!in_comment) {
            fputc(ch, fb);
```



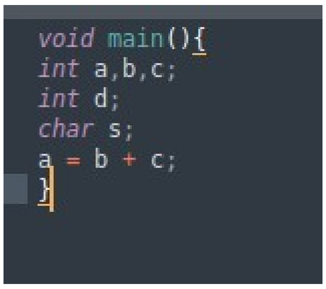
```c
}

void remove_whitespace(const char* input_file, const char* output_file) {
    FILE* fa = fopen(input_file, "r");
    FILE* fb = fopen(output_file, "w");

    if (!fa || !fb) {
        printf("Error opening files\n");
        exit(1);
    }

    int ch, prev_ch = -1;
    while ((ch = fgetc(fa)) != EOF) {
        if (isspace(ch)) {
            if (prev_ch != ' ' && prev_ch != '\n' && prev_ch != '\t') {
                fputc(' ', fb);
            }
        } else {
            fputc(ch, fb);
        }
        prev_ch = ch;
    }

    fclose(fa);
    fclose(fb);
}

void identify_tokens(const char* input_file) {
    FILE* fp = fopen(input_file, "r");
    if (fp == NULL) {
        printf("Cannot open file\n");
        exit(0);
    }

    int row = 1, col = 1;
    char c, buffer[100];
    int buffer_index = 0;
    char type[10] = "";

    while ((c = fgetc(fp)) != EOF) {
        if (c == '\n') {
            row++;
            col = 1;
            continue;
        }

        if (isspace(c)) {
            col++;
            continue;
```



```c
            if (isalpha(c) || c == '_') {
                buffer[buffer_index++] = c;
                col++;
                while (isalnum((c = fgetc(fp))) || c == '_') {
                    buffer[buffer_index++] = c;
                    col++;
                }
                buffer[buffer_index] = '\0';

                if (strcmp(buffer, "main") == 0) {
                    insert_into_symbol_table(buffer, "function", 0);
                } else if (strcmp(buffer, "printf") == 0) {
                    insert_into_symbol_table(buffer, "function", 0);
                } else if (strcmp(buffer, "int") == 0 || strcmp(buffer, "float") == 0 || strcmp(buffer, "char
                    strcpy(type, buffer);
                } else {
                    if (strcmp(type, "int") == 0) {
                        insert_into_symbol_table(buffer, "int", 4);
                    } else if (strcmp(type, "float") == 0) {
                        insert_into_symbol_table(buffer, "float", 4);
                    } else if (strcmp(type, "char") == 0) {
                        insert_into_symbol_table(buffer, "char", 1);
                    }
                }

                buffer_index = 0;
                ungetc(c, fp);
                continue;
            }
        }

        fclose(fp);
    }

int main() {
    initialize_symbol_table();

    remove_preprocessor_directives("q1in.c", "preprocessed.c");
    remove_comments("preprocessed.c", "no_comments.c");
    remove_whitespace("no_comments.c", "cleaned.c");
    identify_tokens("cleaned.c");

    display_symbol_table();

    return 0;
}
```

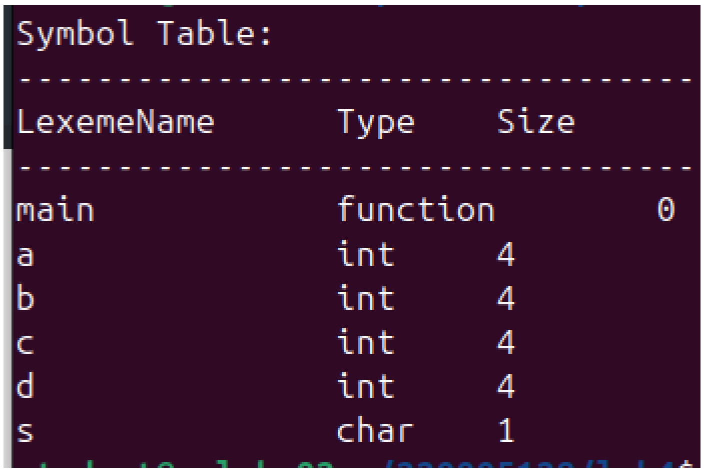
output:

q1in.c



```c
void main(){
int a,b,c;
int d;
char s;
a = b + c;
}
```

output:



```
Symbol Table:
-----------------------------------
LexemeName      Type     Size
-----------------------------------
main            function        0
a               int      4
b               int      4
c               int      4
d               int      4
s               char     1
```