Lab 02: Preliminary Scanning Applications

Ishaan Vatus, CSE-D, Roll No. 10, 220905043

Makefile:

```
CC = gcc
CFLAGS = -Wall -Wpedantic -Wextra -g

SRCS = lexer.c main.c
OBJS = $(SRCS:.c=.o)

TARGET = main

all: $(TARGET)

%.o: %.c
    $(CC) $(CFLAGS) -c $< -o $@

$(TARGET): $(OBJS)
    $(CC) $(CFLAGS) $(OBJS) -o $(TARGET)

clean:
    rm -f $(OBJS)
```

Header File:

```c
/*
 * lexer.h
 * function signatures
 */
#ifndef LEXER_H
#define LEXER_H
#define TAB 9
#define SPACE 32
void remove_whitespace(char *input_file, char *output_file);
void remove_preprocessor(char *input_file, char *output_file);
void regurge_keywords(char *input_file, char *output_file);
#endif
```

C File:

```c
/*
 * lexer.c
 * function implementations
 */
#include <stdbool.h>
```

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "lexer.h"

void remove_whitespace(char *input_file, char *output_file)
{
    char ch;
    bool flag = false;
    FILE *fp = fopen(input_file, "r");
    FILE *op = fopen(output_file, "w");
    while ((ch = getc(fp)) != EOF) {
        if ((ch == SPACE || ch == TAB) && !flag) {
            putc(SPACE, op);
            flag = true;
        }
        else if ((ch == SPACE || ch == TAB) && flag)
            continue;
        else {
            putc(ch, op);
            flag = false;
        }
    }
    fclose(op);
    fclose(fp);
}
void remove_preprocessor(char *input_file, char *output_file)
{
    FILE *fp = fopen(input_file, "r");
    FILE *op = fopen(output_file, "w");
    char ch;
    while ((ch = getc(fp)) != EOF) {
        while (ch == TAB || ch == SPACE)
            ch = getc(fp);
        if (ch == '#') {
            while (ch != '\n')
                ch = getc(fp);
            continue;
        }
        else {
            putc(ch, op);
            while (ch != '\n') {
                ch = getc(fp);
                putc(ch, op);

            }
```

```
        }
    }
    fclose(fp);
    fclose(op);
}
void regurge_keywords(char *input_file, char *output_file)
{
    int keyword_count = 54;
    char *keywords[] = {"alignas","alignof","auto","bool","break","case","char","const","co
    "else","enum","extern","false","float","for","goto","if","inline","int","long","nullptr
    "return","short","signed","sizeof","static","static_assert","struct","switch","thread_l
    "typeof","typeof_unqual","union","unsigned","void","volatile","while","_Atomic","_BitIn
    "_Decimal32","_Decimal64","_Generic","_Imaginary","_Noreturn"};
    FILE *fp = fopen(input_file, "r");
    FILE *op = fopen(output_file, "w");
    int text_len = 0;
    char ch;
    bool in_slc, in_mlc, in_str, esc;
    in_slc = false;
    in_mlc = false;
    in_str = false;
    esc = false;
    while ((ch = getc(fp)) != EOF)
        text_len++;
    rewind(fp);
    char *text = malloc(text_len*sizeof(char));
    fread(text, sizeof(char), text_len, fp);
    for (int index = 0; index < keyword_count; index++) {
        int pattern_len = strlen(keywords[index]);
        for (int i = 0; i < text_len - pattern_len; i++) {
            if (esc) {
                esc = false;
                continue;
            }
            if (text[i] == '\\') {
                esc = true;
                continue;
            }
            if (!in_str && !in_mlc && text[i] == '/' && text[i + 1] == '/')
                in_slc = true;
            if (in_slc && text[i] == '\n')
                in_slc = false;
            if (!in_str && !in_slc && text[i] == '/' && text[i + 1] == '*')
                in_mlc = true;
            if (in_mlc && text[i] == '*' && text[i + 1] == '/') {
                in_mlc = false;
```

```
                i++;
                continue;
            }
            if (!in_slc && !in_mlc && text[i] == '"')
                in_str = !in_str;
            if (in_slc || in_mlc || in_str)
                continue;
            int matches;
            for (matches = 0; matches < pattern_len; matches++) {
                if (text[i+matches] != keywords[index][matches])
                    break;
            }
            if (matches == pattern_len) {
                fwrite(keywords[index], sizeof(char), pattern_len, op);
                fwrite("\n", sizeof(char), 1, op);
            }
        }
    }
    fclose(fp);
    fclose(op);
}
```

Main File:

```
#include "lexer.h"
#include <stdlib.h>
#include <stdio.h>
#include <stdbool.h>

void main(int argc, char **argv)
{
    char *filename = argv[1];
    remove_whitespace(filename, "sanitized");
    remove_preprocessor("sanitized", "no_pre");
    regurge_keywords("no_pre", "keywords_present");
}
```

Input File:

```
    #include <stdbool.h>
 #include <stdio.h>
#include <stdlib.h>
#define PPM_TYPE 6
#define PPM_DEPTH 255

void main(int argc, char **argv)
```

```
{
    FILE *fp = fopen("pixels.ppm", "wb");
int width = atoi(argv[1]);
    int height = atoi(argv[2]);
    fprintf(fp, "P%d\n%d %d\n%d\n", PPM_TYPE, width, height, PPM_DEPTH);
    printf("#");
    printf("extern");
    char r, g,   b;
    for (int row = 0; row < height; row++) {
        for (int col = 0; col < width; col++) {
            r = row%256;
            g = col%256;
            b = (row + col)%256;
            fwrite(&r, sizeof(char), 1, fp);
            fwrite(&g, sizeof(char), 1, fp);
            fwrite(&b, sizeof(char), 1, fp);
        }
    }
}
```

Output Files:

1. sanitized

```
 #include <stdbool.h>
 #include <stdio.h>
#include <stdlib.h>
#define PPM_TYPE 6
#define PPM_DEPTH 255

void main(int argc, char **argv)
{
 FILE *fp = fopen("pixels.ppm", "wb");
int width = atoi(argv[1]);
 int height = atoi(argv[2]);
 fprintf(fp, "P%d\n%d %d\n%d\n", PPM_TYPE, width, height, PPM_DEPTH);
 printf("#");
 printf("extern");
 char r, g, b;
 for (int row = 0; row < height; row++) {
 for (int col = 0; col < width; col++) {
 r = row%256;
 g = col%256;
 b = (row + col)%256;
 fwrite(&r, sizeof(char), 1, fp);
 fwrite(&g, sizeof(char), 1, fp);
```

```
 fwrite(&b, sizeof(char), 1, fp);
 }
 }
}
```

2. no_pre

```
void main(int argc, char **argv)
{
FILE *fp = fopen("pixels.ppm", "wb");
int width = atoi(argv[1]);
int height = atoi(argv[2]);
fprintf(fp, "P%d\n%d %d\n%d\n", PPM_TYPE, width, height, PPM_DEPTH);
printf("#");
printf("extern");
char r, g, b;
for (int row = 0; row < height; row++) {
for (int col = 0; col < width; col++) {
r = row%256;
g = col%256;
b = (row + col)%256;
fwrite(&r, sizeof(char), 1, fp);
fwrite(&g, sizeof(char), 1, fp);
fwrite(&b, sizeof(char), 1, fp);
}
}
}
```

3. keywords_present

```
char
char
char
char
char
for
for
int
int
int
int
int
int
int
int
sizeof
sizeof
```

```
sizeof
void
```