10/02/2025

-COMPILER DESIGN CSA1435

(DAY=02)

PROBLEM - 01-ROLE OF LEXICAL ANALYSER PROGRAM

#include <stdio.h>

#include <ctype.h>

#include <string.h>

#define MAX\_IDENTIFIER\_LENGTH 31

void remove\_comments(FILE \*source, FILE \*cleaned) {

char c, prev = '\0';

while ((c = fgetc(source)) != EOF) {

if (c == '/' && prev == '/') {

while ((c = fgetc(source)) != '\n' && c != EOF);

} else if (c == '\*' && prev == '/') {

while ((c = fgetc(source)) != EOF) {

if (c == '/' && prev == '\*') break;

prev = c;

}

} else {

if (prev != '\0' && prev != '/') fputc(prev, cleaned);

}

prev = c;

}

if (prev != '\0' && prev != '/' && prev != '\*') fputc(prev, cleaned);

}

void lexical\_analyzer(FILE \*file) {

char ch;

char buffer[MAX\_IDENTIFIER\_LENGTH + 1];

int i = 0;

while ((ch = fgetc(file)) != EOF) {

if (isspace(ch)) continue;

if (isalpha(ch) || ch == '\_') { // Identifier

i = 0;

buffer[i++] = ch;

while ((ch = fgetc(file)) != EOF && (isalnum(ch) || ch == '\_')) {

if (i < MAX\_IDENTIFIER\_LENGTH) buffer[i++] = ch;

}

buffer[i] = '\0';

printf("Identifier: %s\n", buffer);

ungetc(ch, file);

} else if (isdigit(ch)) { // Constant

i = 0;

buffer[i++] = ch;

while ((ch = fgetc(file)) != EOF && isdigit(ch)) {

if (i < MAX\_IDENTIFIER\_LENGTH) buffer[i++] = ch;

}

buffer[i] = '\0';

printf("Constant: %s\n", buffer);

ungetc(ch, file);

} else if (strchr("+-\*/=<>!&|", ch)) { // Operator

char next = fgetc(file);

if (next != EOF && strchr("=+-><&|", next)) {

printf("Operator: %c%c\n", ch, next);

} else {

printf("Operator: %c\n", ch);

ungetc(next, file);

}

}

}

}

int main() {

FILE \*source = fopen("input.c", "r");

FILE \*cleaned = fopen("cleaned.c", "w+");

if (!source || !cleaned) {

printf("Error opening file!\n");

return 1;

}

remove\_comments(source, cleaned);

fclose(source);

rewind(cleaned);

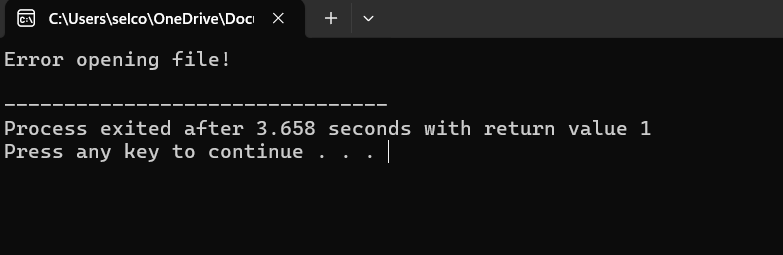
lexical\_analyzer(cleaned);

fclose(cleaned);

return 0;

}

OUTPUT:



PROBLEM - 02

#include <stdio.h>

#include <ctype.h>

#include <string.h>

#define MAX\_IDENTIFIER\_LENGTH 31

void check\_comment(const char \*line) {

if (strncmp(line, "//", 2) == 0) {

printf("Single-line comment: %s\n", line);

} else if (strncmp(line, "/\*", 2) == 0) {

printf("Multi-line comment detected\n");

}

}

void remove\_comments(FILE \*source, FILE \*cleaned) {

char c, prev = '\0';

int in\_multiline\_comment = 0;

while ((c = fgetc(source)) != EOF) {

if (!in\_multiline\_comment && c == '/' && prev == '/') {

printf("Single-line comment found\n");

while ((c = fgetc(source)) != '\n' && c != EOF);

} else if (!in\_multiline\_comment && c == '\*' && prev == '/') {

printf("Multi-line comment start\n");

in\_multiline\_comment = 1;

} else if (in\_multiline\_comment && c == '/' && prev == '\*') {

printf("Multi-line comment end\n");

in\_multiline\_comment = 0;

} else if (!in\_multiline\_comment) {

if (prev != '\0' && prev != '/') fputc(prev, cleaned);

}

prev = c;

}

if (prev != '\0' && prev != '/' && prev != '\*') fputc(prev, cleaned);

}

void lexical\_analyzer(FILE \*file) {

char ch;

char buffer[MAX\_IDENTIFIER\_LENGTH + 1];

int i = 0;

while ((ch = fgetc(file)) != EOF) {

if (isspace(ch)) continue;

if (isalpha(ch) || ch == '\_') { // Identifier

i = 0;

buffer[i++] = ch;

while ((ch = fgetc(file)) != EOF && (isalnum(ch) || ch == '\_')) {

if (i < MAX\_IDENTIFIER\_LENGTH) buffer[i++] = ch;

}

buffer[i] = '\0';

printf("Identifier: %s\n", buffer);

ungetc(ch, file);

} else if (isdigit(ch)) { // Constant

i = 0;

buffer[i++] = ch;

while ((ch = fgetc(file)) != EOF && isdigit(ch)) {

if (i < MAX\_IDENTIFIER\_LENGTH) buffer[i++] = ch;

}

buffer[i] = '\0';

printf("Constant: %s\n", buffer);

ungetc(ch, file);

} else if (strchr("+-\*/=<>!&|", ch)) { // Operator

char next = fgetc(file);

if (next != EOF && strchr("=+-><&|", next)) {

printf("Operator: %c%c\n", ch, next);

} else {

printf("Operator: %c\n", ch);

ungetc(next, file);

}

}

}

}

int main() {

FILE \*source = fopen("input.c", "r");

FILE \*cleaned = fopen("cleaned.c", "w+");

if (!source || !cleaned) {

printf("Error opening file!\n");

return 1;

}

remove\_comments(source, cleaned);

fclose(source);

rewind(cleaned);

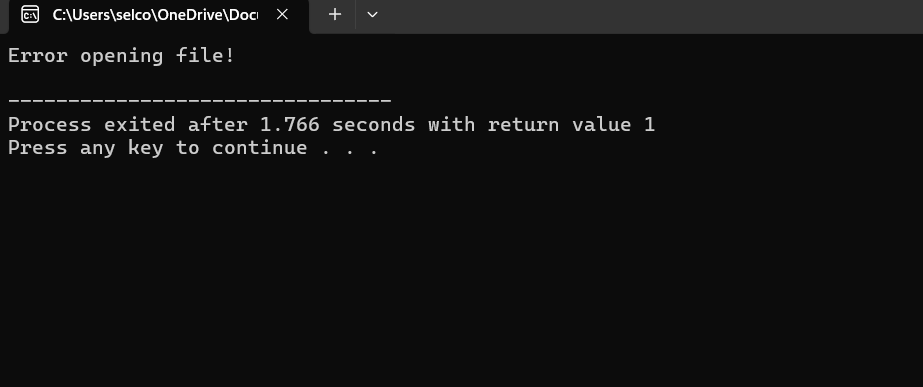
lexical\_analyzer(cleaned);

fclose(cleaned);

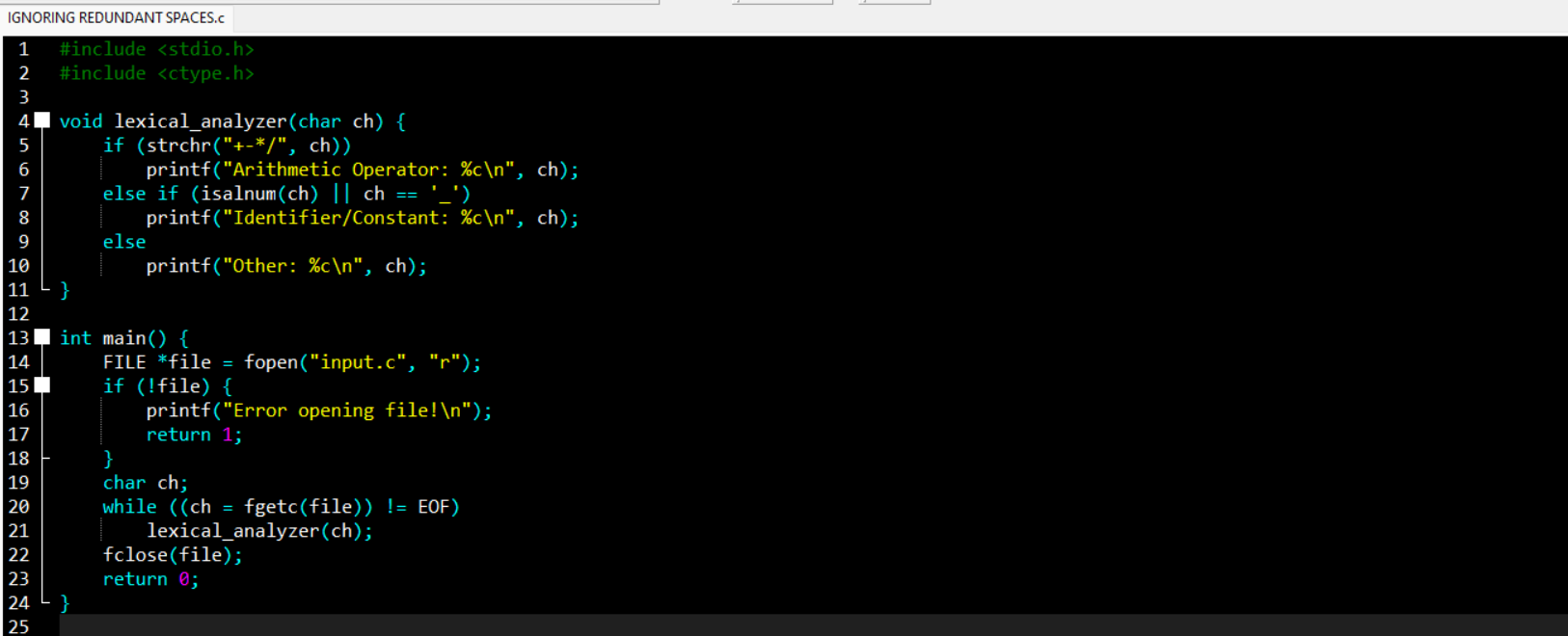
return 0;

}

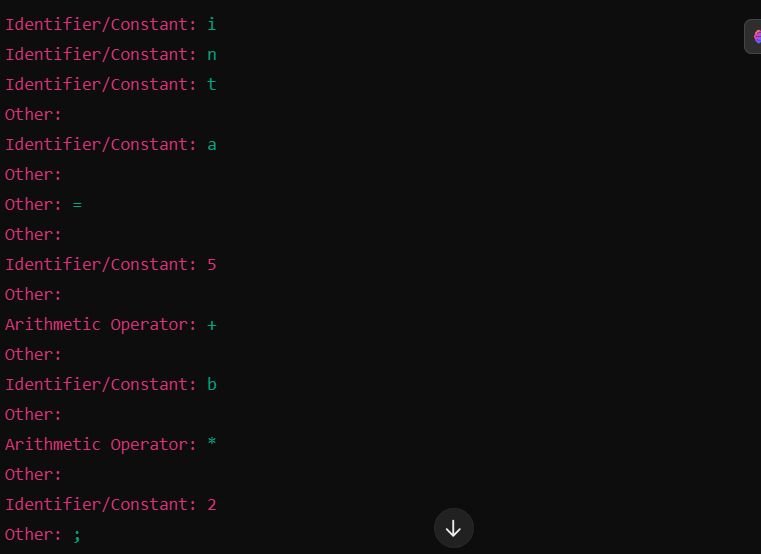
OUTPUT-02



PROBLEM - 03



OUTPUT:03



PROBLEM 04

#include <stdio.h>

#include <ctype.h>

void lexical\_analyzer(char ch, int \*whitespace\_count, int \*newline\_count) {

if (isspace(ch)) {

if (ch == '\n')

(\*newline\_count)++;

else

(\*whitespace\_count)++;

}

}

int main() {

FILE \*file = fopen("input.c", "r");

if (!file) {

printf("Error opening file!\n");

return 1;

}

char ch;

int whitespace\_count = 0, newline\_count = 0;

while ((ch = fgetc(file)) != EOF)

lexical\_analyzer(ch, &whitespace\_count, &newline\_count);

fclose(file);

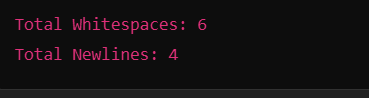
printf("Total Whitespaces: %d\n", whitespace\_count);

printf("Total Newlines: %d\n", newline\_count);

return 0;

}

OUTPUT:04



PROGRAM 05

#include <stdio.h>

#include <ctype.h>

#include <string.h>

int is\_valid\_identifier(const char \*str) {

if (!isalpha(str[0]) && str[0] != '\_')

return 0;

for (int i = 1; str[i] != '\0'; i++) {

if (!isalnum(str[i]) && str[i] != '\_')

return 0;

}

return 1;

}

int main() {

char identifier[100];

printf("Enter an identifier: ");

scanf("%s", identifier);

if (is\_valid\_identifier(identifier))

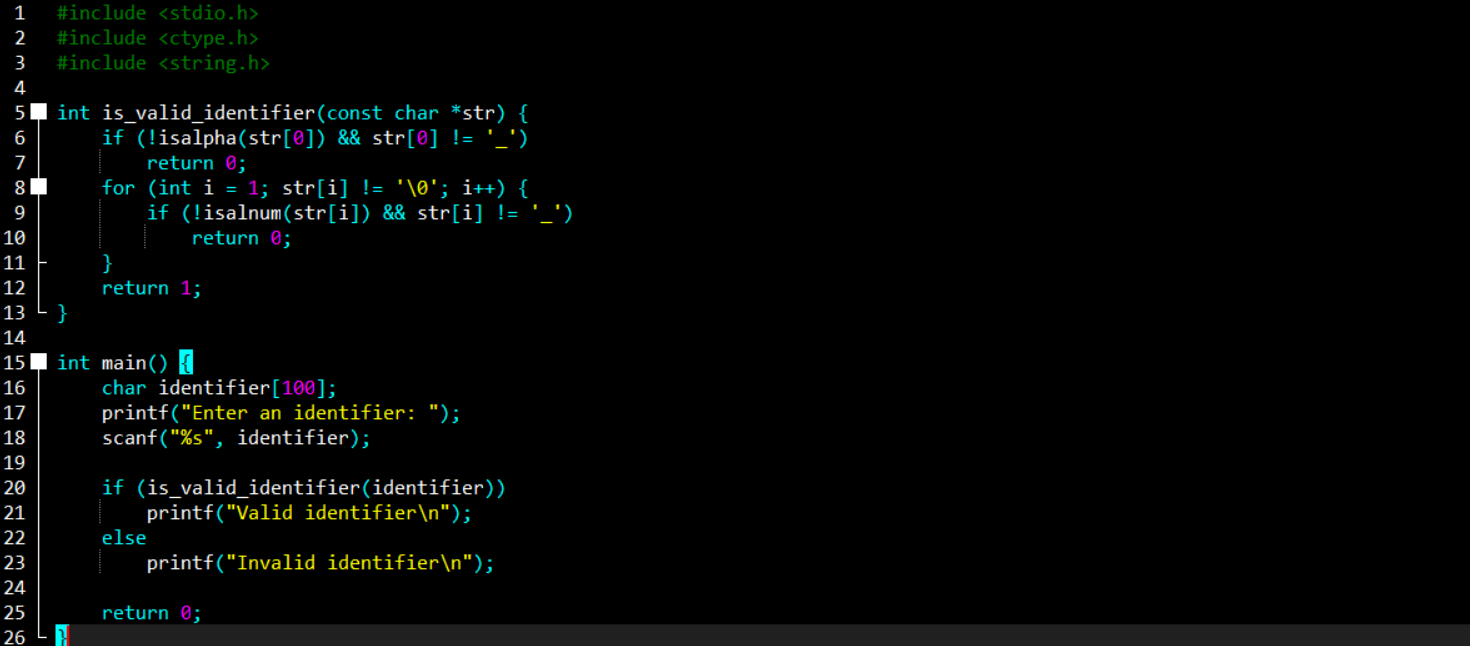
printf("Valid identifier\n");

else

printf("Invalid identifier\n");

return 0;

}



OUTPUT-05

