1. **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. Develop a lexical Analyzer to identify identifiers, constants, operators using C program.**

#include <stdio.h>

#include <ctype.h>

#include <string.h>

#include <stdlib.h>

#define MAX\_IDENTIFIER\_LENGTH 50

int isOperator(char ch) {

return (ch == '+' || ch == '-' || ch == '\*' || ch == '/' || ch == '=' || ch == '>' || ch == '<');

}

int isValidIdentifier(char \*str) {

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

return 0;

}

for (int i = 1; i < strlen(str); i++) {

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

return 0;

}

}

return 1;

}

int isConstant(char \*str) {

for (int i = 0; i < strlen(str); i++) {

if (!isdigit(str[i])) {

return 0;

}

}

return 1;

}

void lexicalAnalyzer(char \*input) {

char token[100];

int index = 0;

for (int i = 0; i < strlen(input); i++) {

if (isspace(input[i])) {

continue;

}

if (input[i] == '/' && input[i + 1] == '/') {

break;

}

if (input[i] == '/' && input[i + 1] == '\*') {

while (input[i] != '\*' || input[i + 1] != '/') {

i++;

}

i += 2;

continue;

}

if (isOperator(input[i])) {

printf("Operator: %c\n", input[i]);

} else if (isalnum(input[i]) || input[i] == '\_') {

token[index++] = input[i];

while (isalnum(input[i + 1]) || input[i + 1] == '\_') {

i++;

token[index++] = input[i];

}

token[index] = '\0';

index = 0;

if (isValidIdentifier(token)) {

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

} else if (isConstant(token)) {

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

}

}

}

}

int main() {

char input[200];

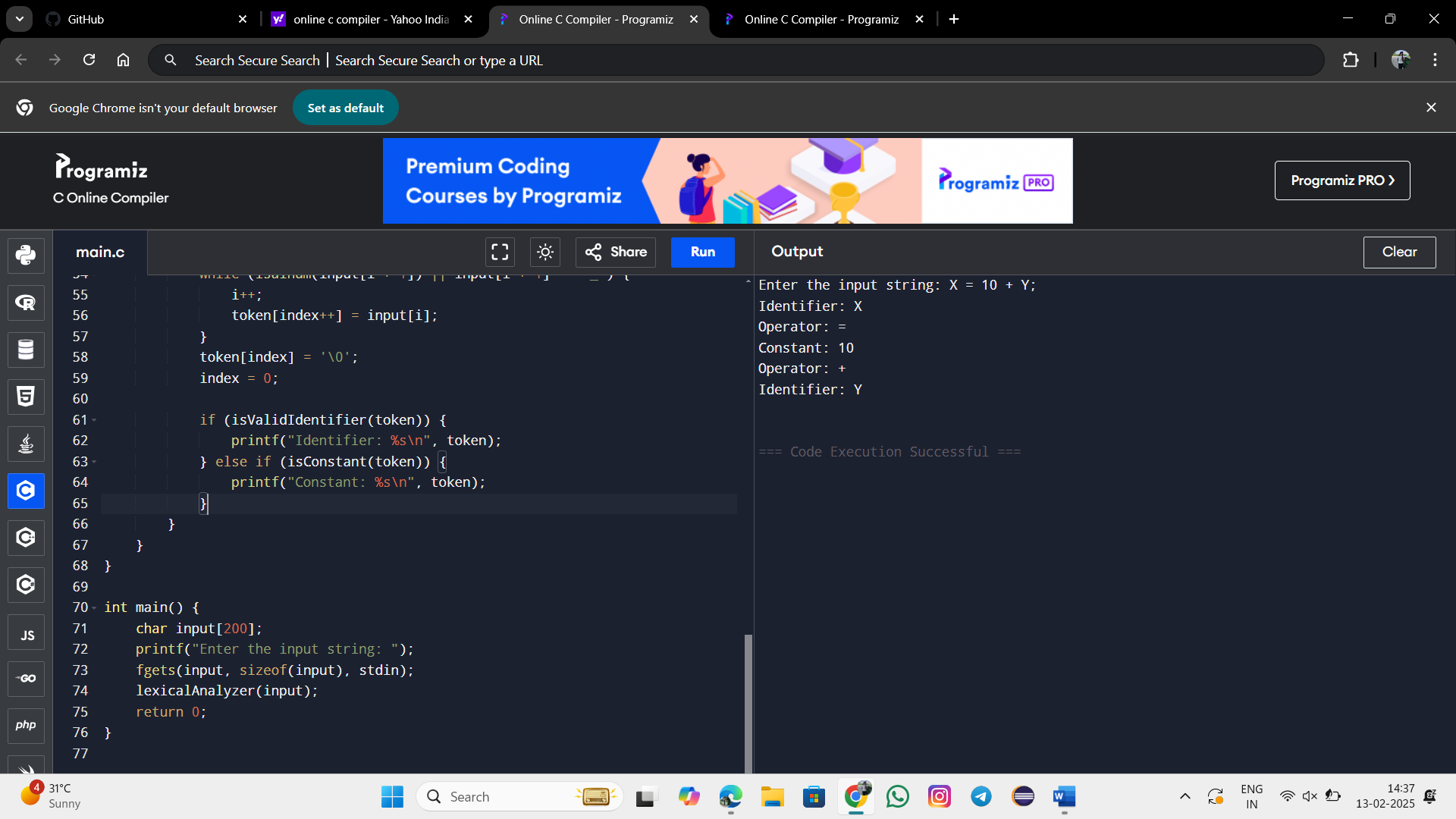
printf("Enter the input string: ");

fgets(input, sizeof(input), stdin);

lexicalAnalyzer(input);

return 0;

}



**2. Extend the lexical Analyzer to Check comments, dened as follows in C:**

**a) A comment begins with // and includes all characters until the end of that line.**

**b) A comment begins with /\* and includes all characters through the next occurrence of the character sequence \*/Develop a lexical Analyzer to identify whether a given line is a comment or not.**

#include <stdio.h>

#include <string.h>

int isComment(char \*line) {

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

return 1;

}

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

int len = strlen(line);

if (len >= 4 && strstr(line, "\*/") != NULL) {

return 1;

}

}

return 0;

}

int main() {

char line[200];

printf("Enter a line of code: ");

fgets(line, sizeof(line), stdin);

if (isComment(line)) {

printf("The given line is a comment.\n");

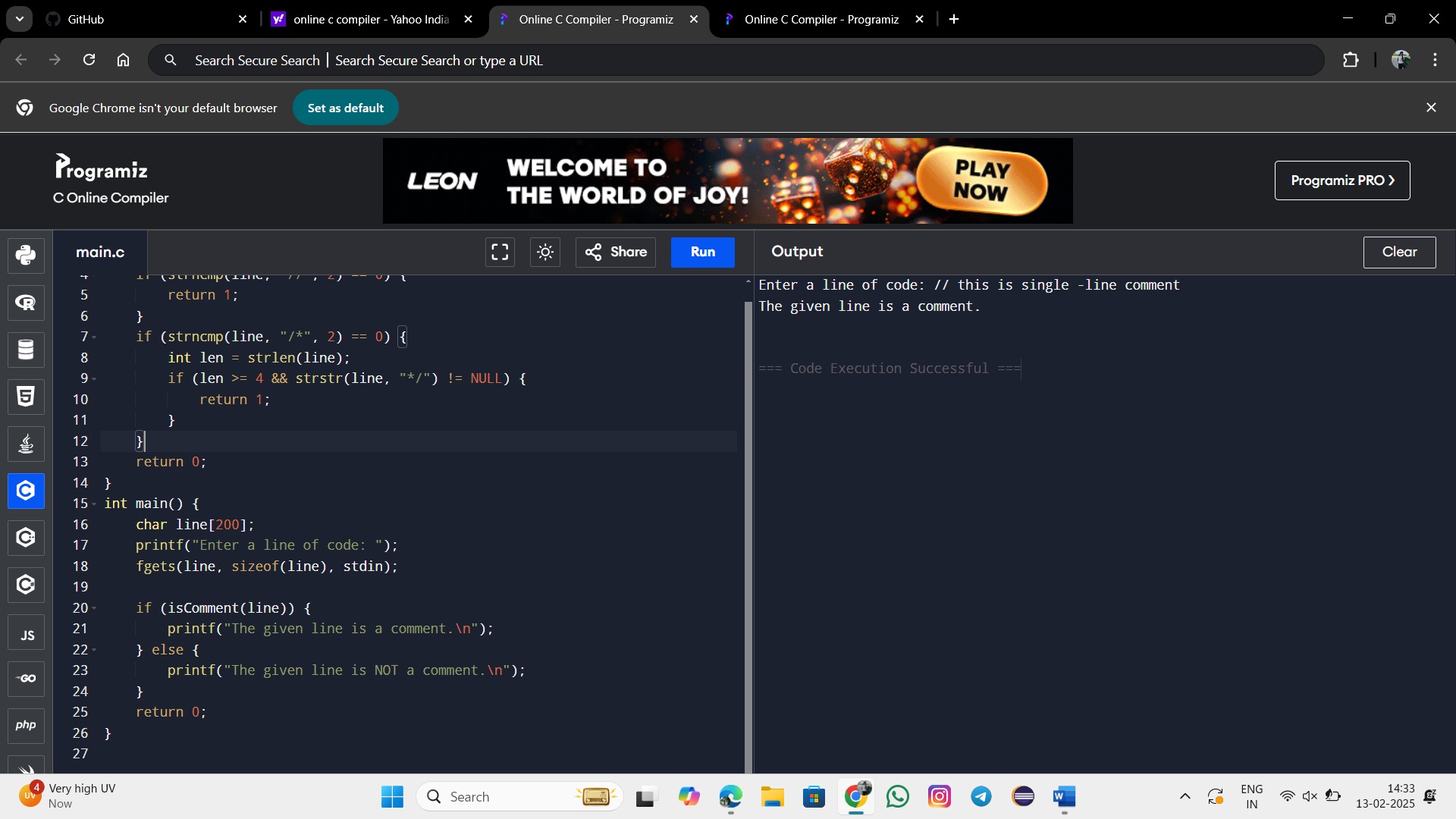
} else {

printf("The given line is NOT a comment.\n");

}

return 0;

}



**3. Design a lexical Analyzer to validate operators to recognize the operators +,-,\*,/ using regular Arithmetic operators .**

#include <stdio.h>

#include <string.h>

int isOperator(char ch) {

return (ch == '+' || ch == '-' || ch == '\*' || ch == '/');

}

int main() {

char input[100];

printf("Enter an expression: ");

fgets(input, sizeof(input), stdin);

printf("Recognized operators: ");

for (int i = 0; i < strlen(input); i++) {

if (isOperator(input[i])) {

printf("%c ", input[i]);

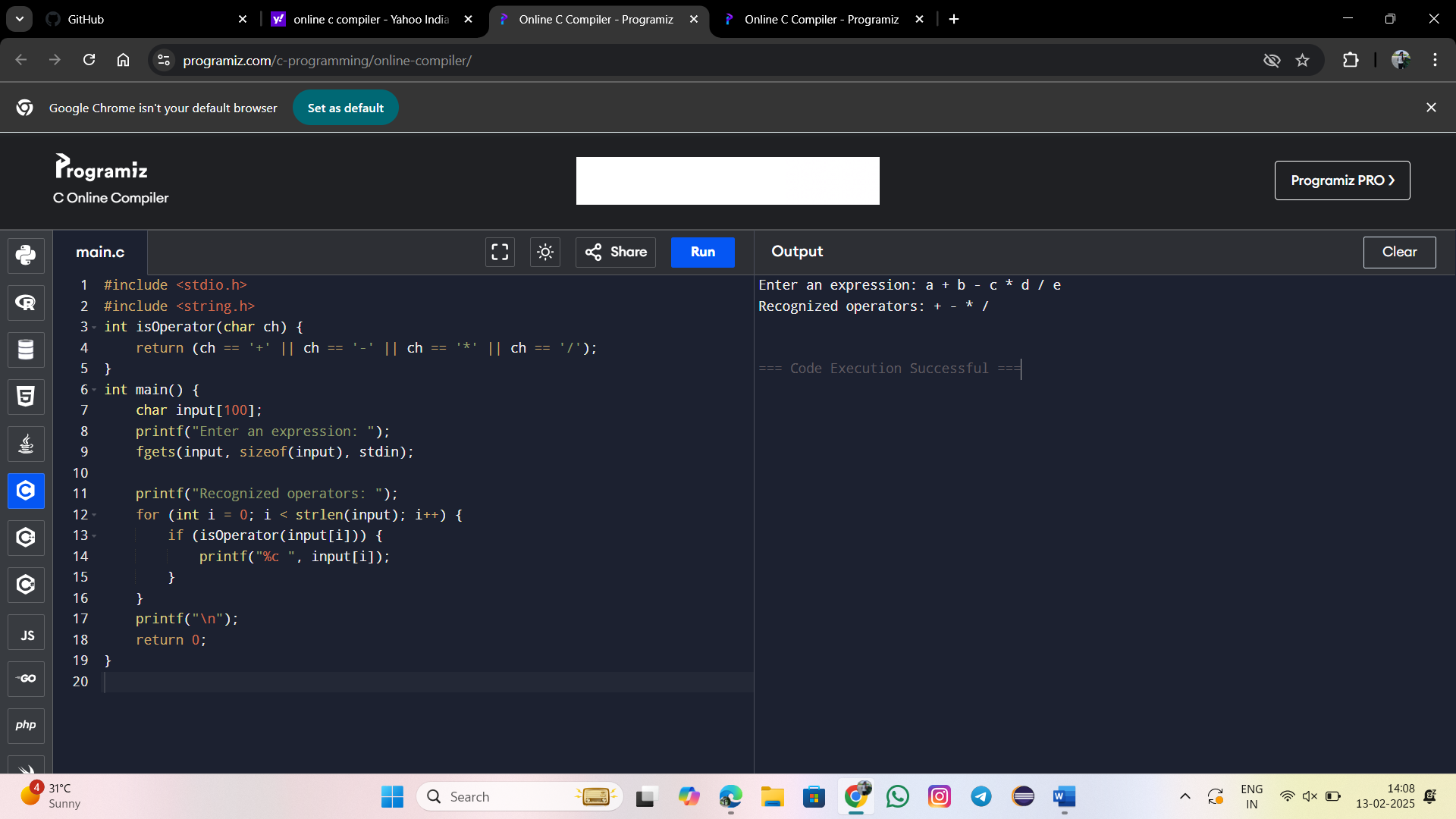
}

}

printf("\n");

return 0;

}



1. **Design a lexical Analyzer to find the number of whitespaces and newline characters.**

#include <stdio.h>

int main() {

char ch;

int whitespaceCount = 0, newlineCount = 0;

printf("Enter the input (Press Ctrl+D to end input on UNIX or Ctrl+Z on Windows):\n");

while ((ch = getchar()) != EOF) {

if (ch == ' ' || ch == '\t') {

whitespaceCount++;

} else if (ch == '\n') {

newlineCount++;

}

}

printf("Number of whitespace characters: %d\n", whitespaceCount);

printf("Number of newline characters: %d\n", newlineCount);

return 0;

}

1. **Develop a lexical Analyzer to test whether a given identifier is valid or not.**

#include <stdio.h>

#include <ctype.h>

#include <string.h>

int isValidIdentifier(char \*identifier) {

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

return 0;

}

for (int i = 1; i < strlen(identifier); i++) {

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

return 0;

}

}

return 1;

}

int main() {

char identifier[100];

printf("Enter an identifier: ");

scanf("%s", identifier);

if (isValidIdentifier(identifier)) {

printf("Valid Identifier\n");

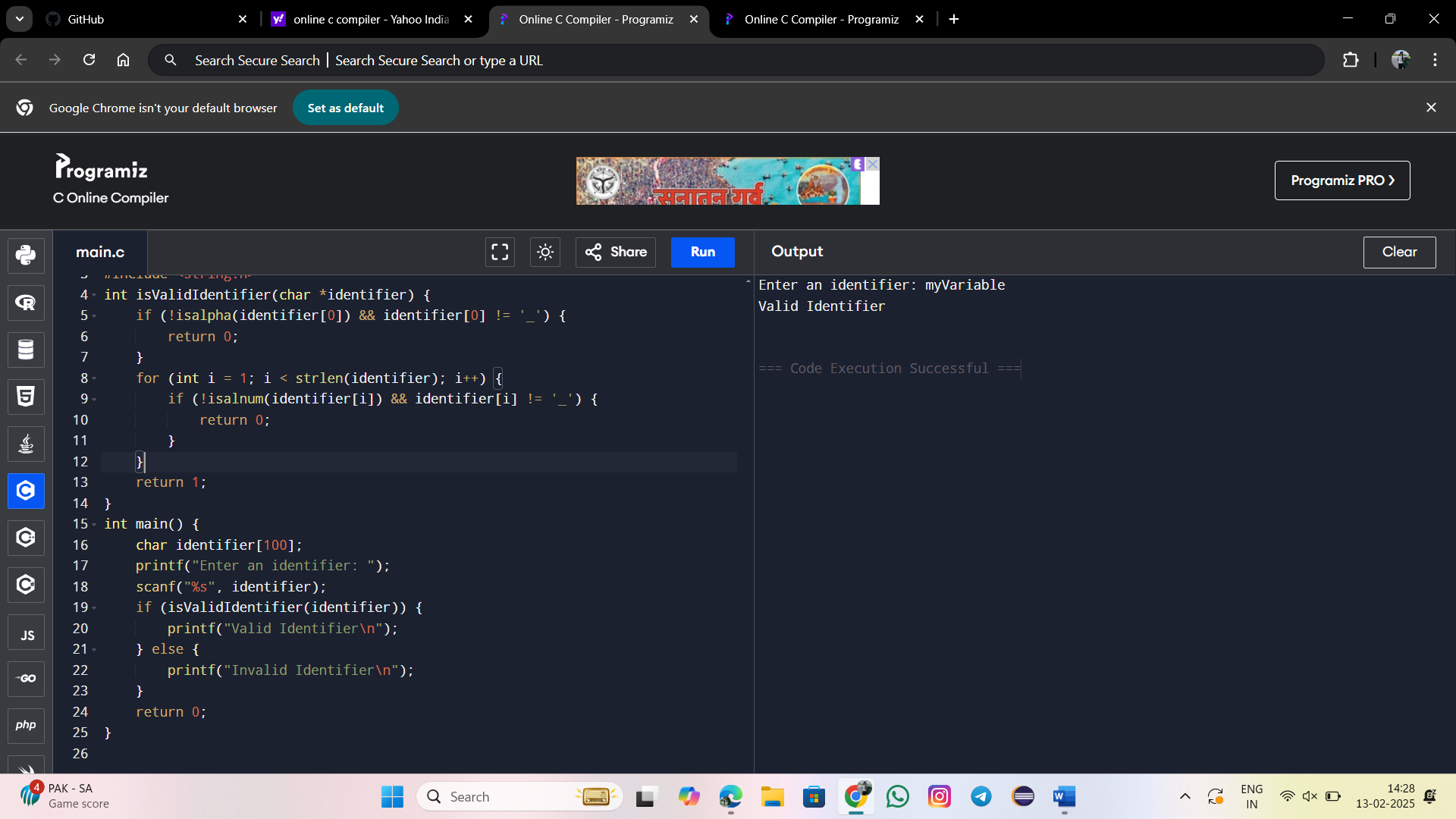
} else {

printf("Invalid Identifier\n");

}

return 0;

}



**6. Implement a C program to eliminate left recursion****.**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#define MAX\_RULES 10

#define MAX\_LENGTH 100

void eliminateLeftRecursion(char nonTerminal, char productions[][MAX\_LENGTH], int count) {

char alpha[MAX\_RULES][MAX\_LENGTH];

char beta[MAX\_RULES][MAX\_LENGTH];

int alphaCount = 0, betaCount = 0;

for (int i = 0; i < count; i++) {

if (productions[i][0] == nonTerminal) {

strcpy(alpha[alphaCount++], productions[i] + 1); // Store alpha (remove the non-terminal)

} else {

strcpy(beta[betaCount++], productions[i]); // Store beta

}

}

if (alphaCount == 0) {

printf("No left recursion detected. Productions remain unchanged.\n");

for (int i = 0; i < count; i++) {

printf("%c -> %s\n", nonTerminal, productions[i]);

}

return;

}

// Display the new productions without left recursion

printf("Eliminated left recursion:\n");

printf("%c ->", nonTerminal);

for (int i = 0; i < betaCount; i++) {

printf(" %s%c'", beta[i], nonTerminal);

if (i < betaCount - 1) printf(" | ");

}

printf("\n");

printf("%c' ->", nonTerminal);

for (int i = 0; i < alphaCount; i++) {

printf(" %s%c'", alpha[i], nonTerminal);

if (i < alphaCount - 1) printf(" | ");

}

printf(" | ε\n");

}

int main() {

char nonTerminal;

int count;

char productions[MAX\_RULES][MAX\_LENGTH];

printf("Enter the non-terminal: ");

scanf(" %c", &nonTerminal);

printf("Enter the number of productions: ");

scanf("%d", &count);

printf("Enter the productions (without the non-terminal):\n");

for (int i = 0; i < count; i++) {

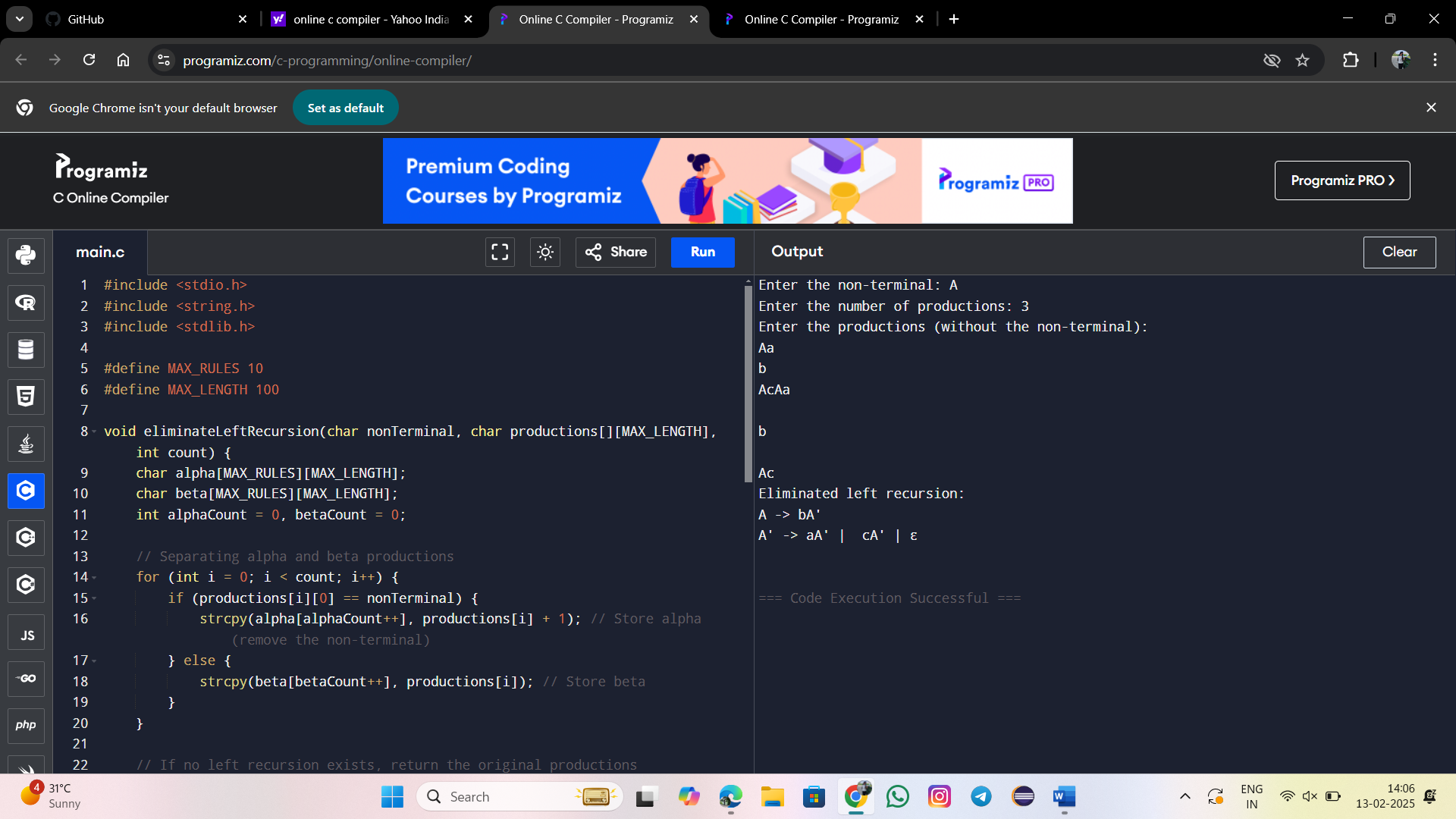
scanf("%s", productions[i]);

}

eliminateLeftRecursion(nonTerminal, productions, count);

return 0;

}



1. **Implement a C program to eliminate left factoring.**

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#define MAX\_RULES 10

#define MAX\_LENGTH 100

void eliminateLeftFactoring(char nonTerminal, char productions[][MAX\_LENGTH], int count) {

char commonPrefix[MAX\_LENGTH] = "";

int prefixLen = strlen(productions[0]);

// Find the longest common prefix

for (int i = 1; i < count; i++) {

int j = 0;

while (j < prefixLen && productions[0][j] == productions[i][j]) {

j++;

}

prefixLen = j;

}

if (prefixLen == 0) {

printf("No left factoring needed. Productions remain unchanged.\n");

for (int i = 0; i < count; i++) {

printf("%c -> %s\n", nonTerminal, productions[i]);

}

return;

}

strncpy(commonPrefix, productions[0], prefixLen);

commonPrefix[prefixLen] = '\0';

// Display new productions

printf("Left Factored Grammar:\n");

printf("%c -> %s%c'\n", nonTerminal, commonPrefix, nonTerminal);

printf("%c' ->", nonTerminal);

int first = 1;

for (int i = 0; i < count; i++) {

if (strlen(productions[i]) > prefixLen) {

if (!first) printf(" | ");

printf("%s", productions[i] + prefixLen);

first = 0;

}

}

printf(" | ε\n");

}

int main() {

char nonTerminal;

int count;

char productions[MAX\_RULES][MAX\_LENGTH];

printf("Enter the non-terminal: ");

scanf(" %c", &nonTerminal);

printf("Enter the number of productions: ");

scanf("%d", &count);

printf("Enter the productions (without the non-terminal):\n");

for (int i = 0; i < count; i++) {

scanf("%s", productions[i]);

}

eliminateLeftFactoring(nonTerminal, productions, count);

return 0;

}

