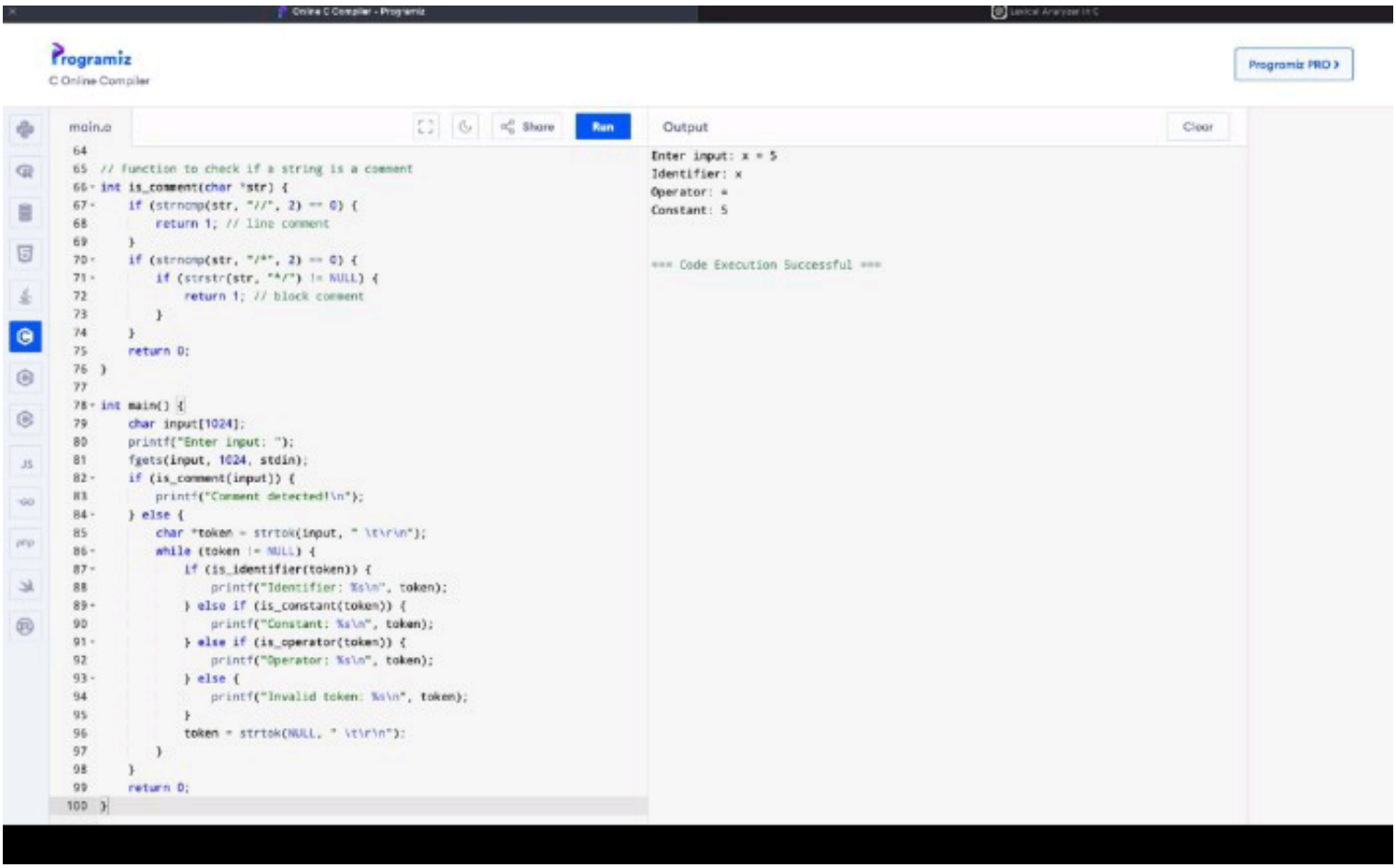
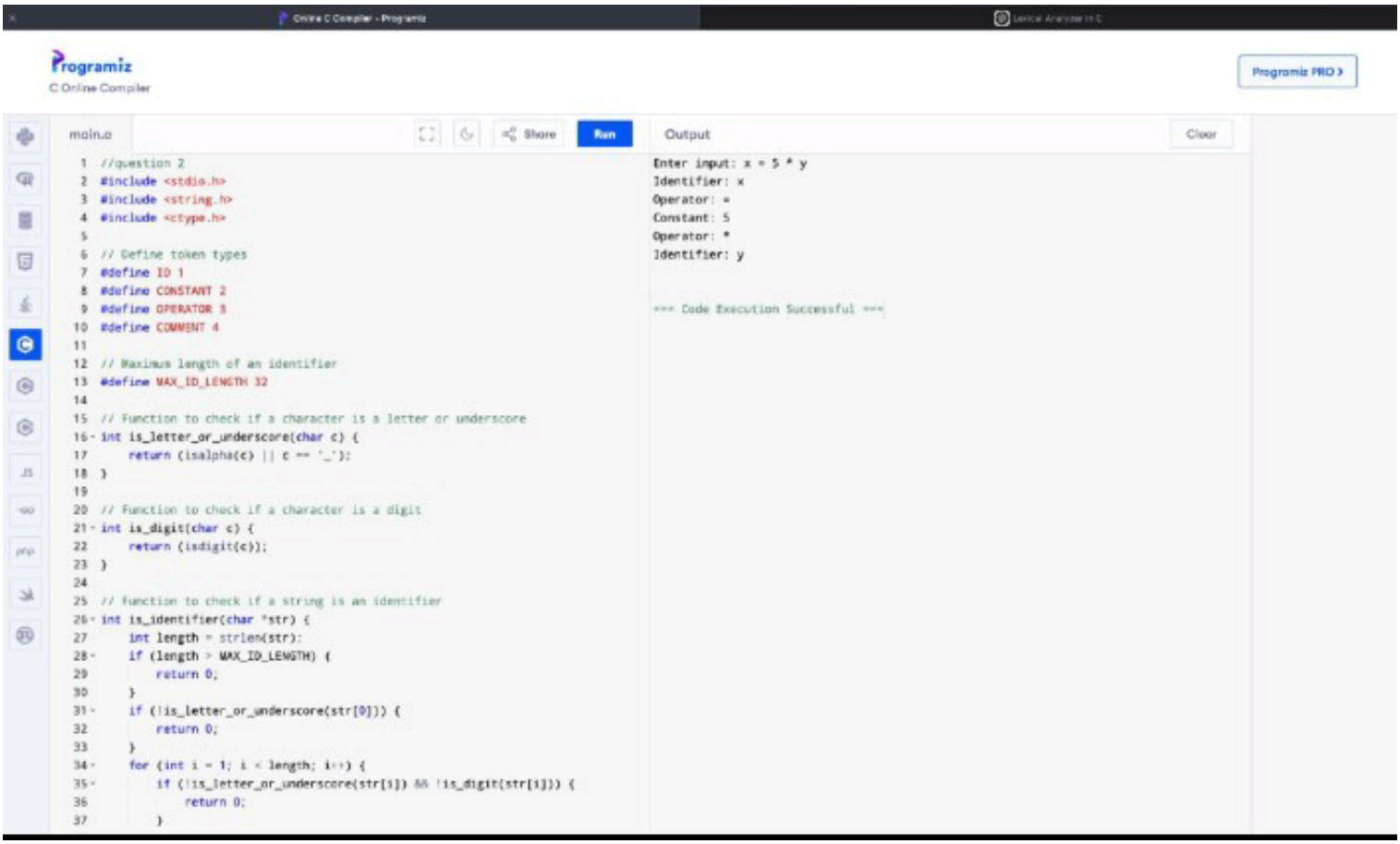
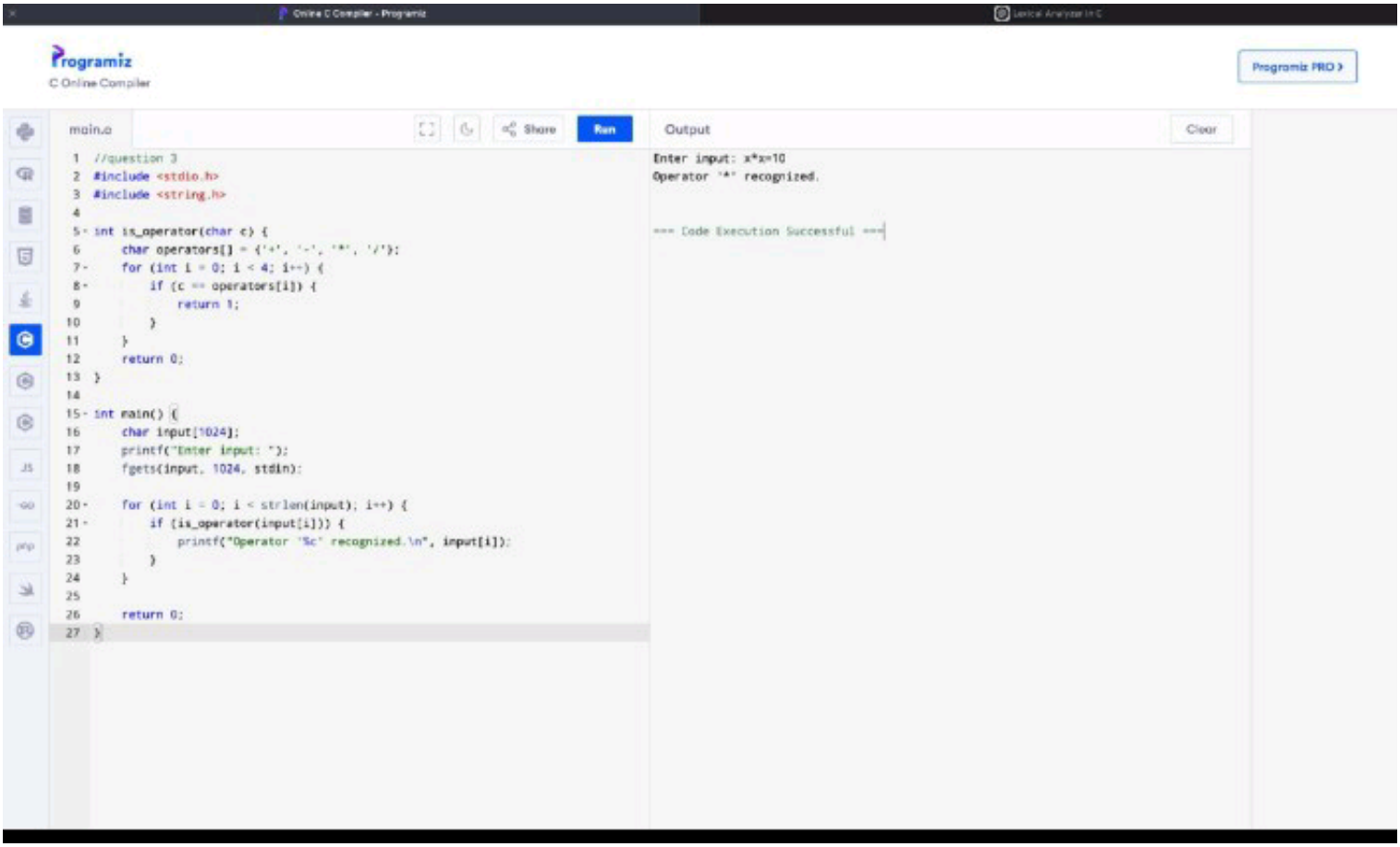
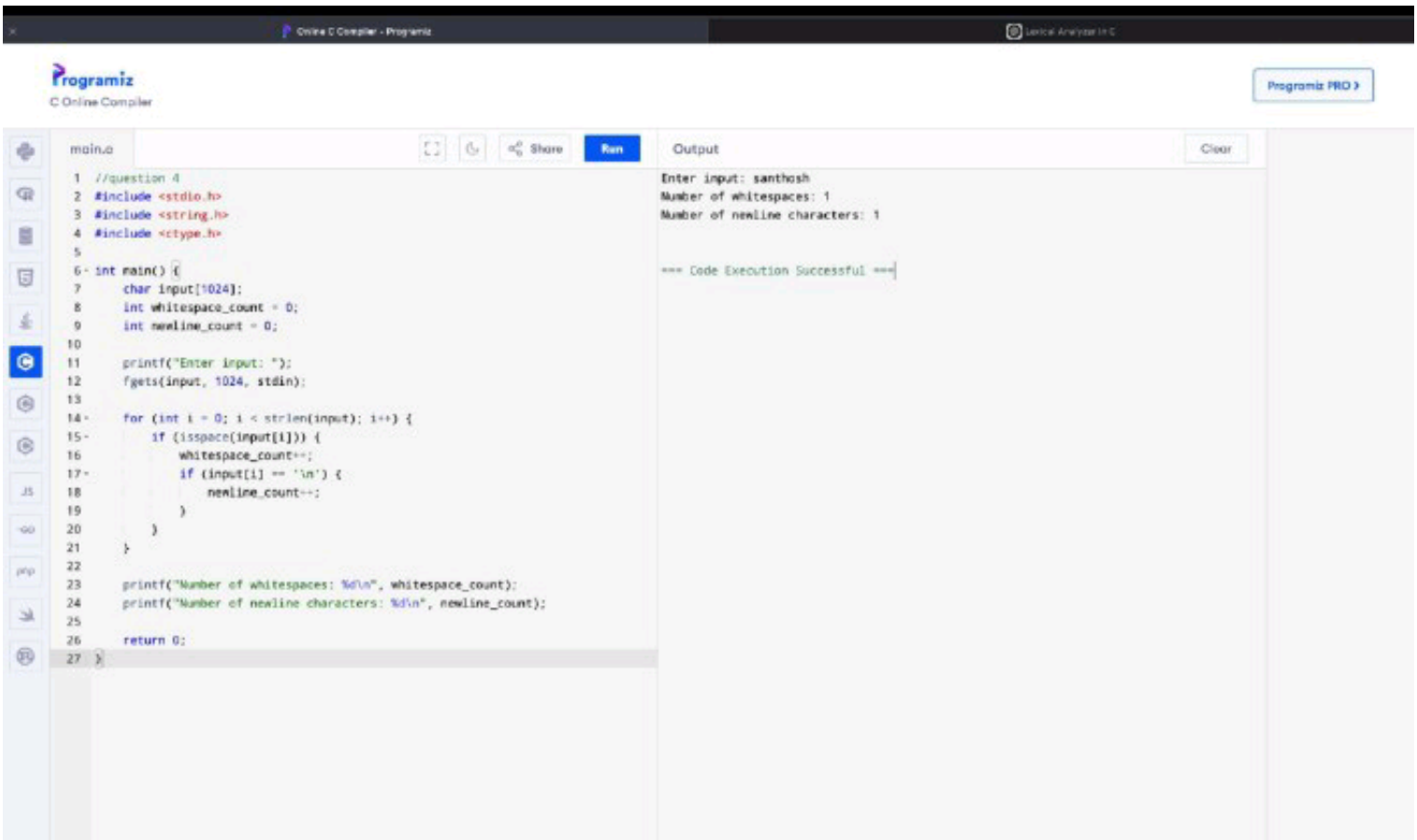
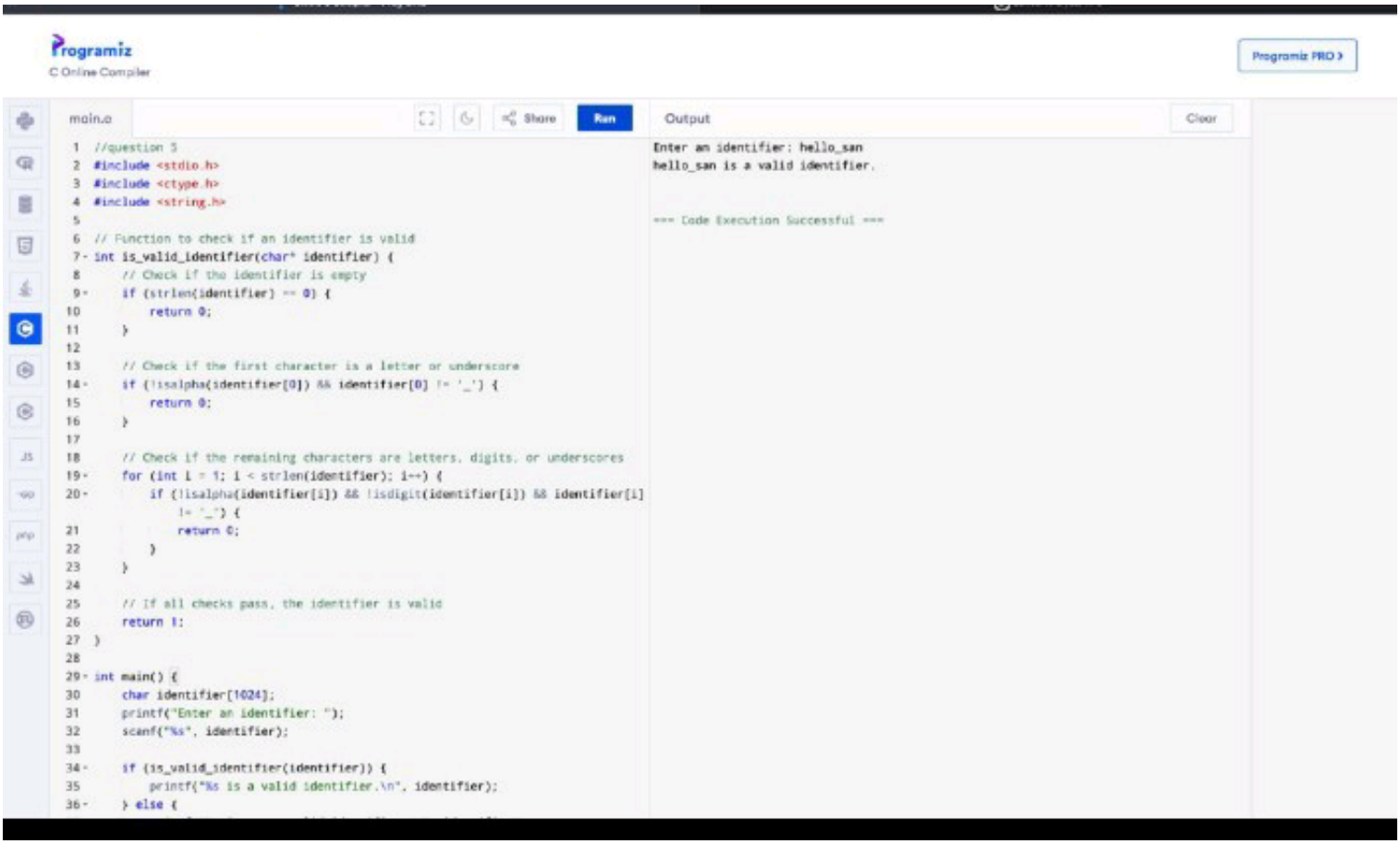
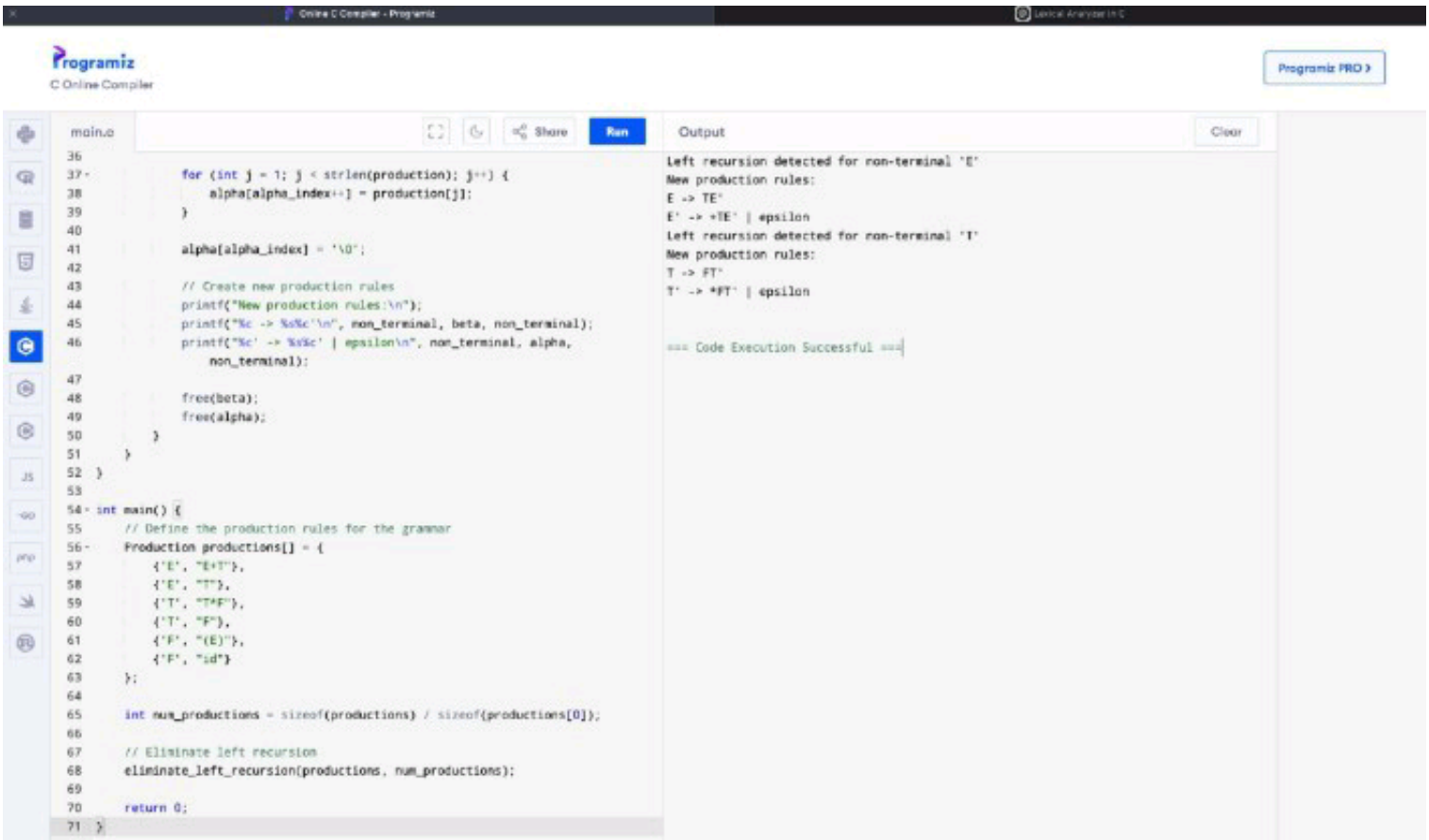

Online C Compiler - Programiz
Lexical Analyzer in C
Programiz
C Online Compiler
Programiz PRO >
main.c
Share
Run
Output
Clear
64
65 // function to check if a string is a comment
66 int is_comment(char *str) {
67 if (strncmp(str, "//", 2) == 0) {
68 return 1; // line comment
69 }
70 if (strncmp(str, "/*", 2) == 0) {
71 if (strstr(str, "*/") != NULL) {
72 return 1; // block comment
73 }
74 }
75 return 0;
76 }
77
78 int main() {
79 char input[1024];
80 printf("Enter input: ");
81 fgets(input, 1024, stdin);
82 if (is_comment(input)) {
83 printf("Comment detected!\n");
84 } else {
85 char *token = strtok(input, " \t\r\n");
86 while (token != NULL) {
87 if (is_identifier(token)) {
88 printf("Identifier: %s\n", token);
89 } else if (is_constant(token)) {
90 printf("Constant: %s\n", token);
91 } else if (is_operator(token)) {
92 printf("Operator: %s\n", token);
93 } else {
94 printf("Invalid token: %s\n", token);
95 }
96 token = strtok(NULL, " \t\r\n");
97 }
98 }
99 return 0;
100 }
Enter input: x = 5
Identifier: x
Operator: =
Constant: 5
=== Code Execution Successful ===

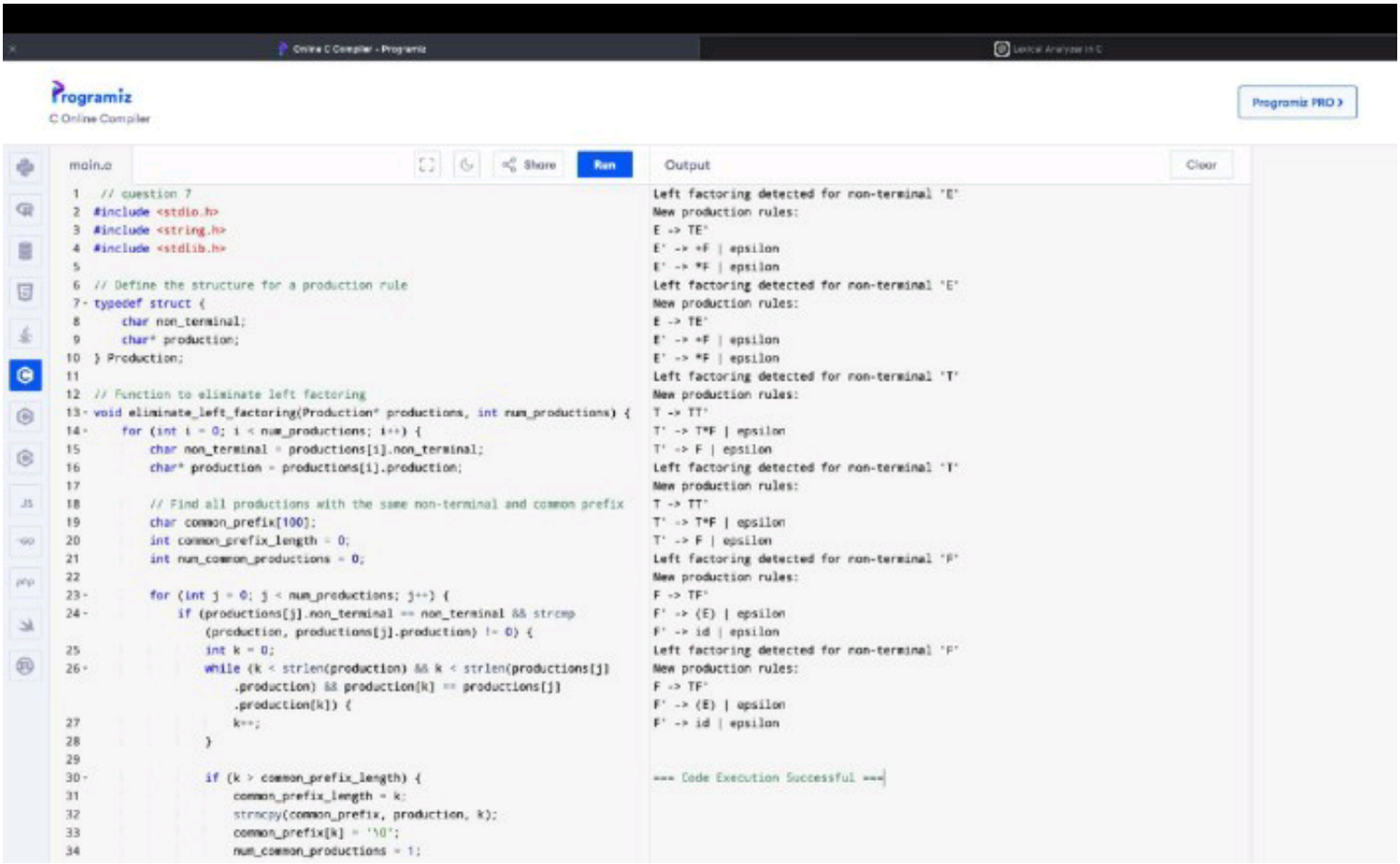

Online C Compiler - Programiz
Lexical Analyzer in C
Programiz
C Online Compiler
Programiz PRO >
main.c
Share
Run
Output
Clear
//question 2
#include <stdio.h>
#include <string.h>
#include <ctype.h>
// Define token types
#define ID 1
#define CONSTANT 2
#define OPERATOR 3
#define COMMENT 4
// Maximum length of an identifier
#define MAX_ID_LENGTH 32
// Function to check if a character is a letter or underscore
int is_letter_or_underscore(char c) {
return (isalpha(c) || c == '_');
}
// Function to check if a character is a digit
int is_digit(char c) {
return (isdigit(c));
}
// Function to check if a string is an identifier
int is_identifier(char *str) {
int length = strlen(str);
if (length > MAX_ID_LENGTH) {
return 0;
}
if (!is_letter_or_underscore(str[0])) {
return 0;
}
for (int i = 1; i < length; i++) {
if (!is_letter_or_underscore(str[i]) && !is_digit(str[i])) {
return 0;
}
Enter input: x = 5 * y
Identifier: x
Operator: =
Constant: 5
Operator: *
Identifier: y
=== Code Execution Successful ===

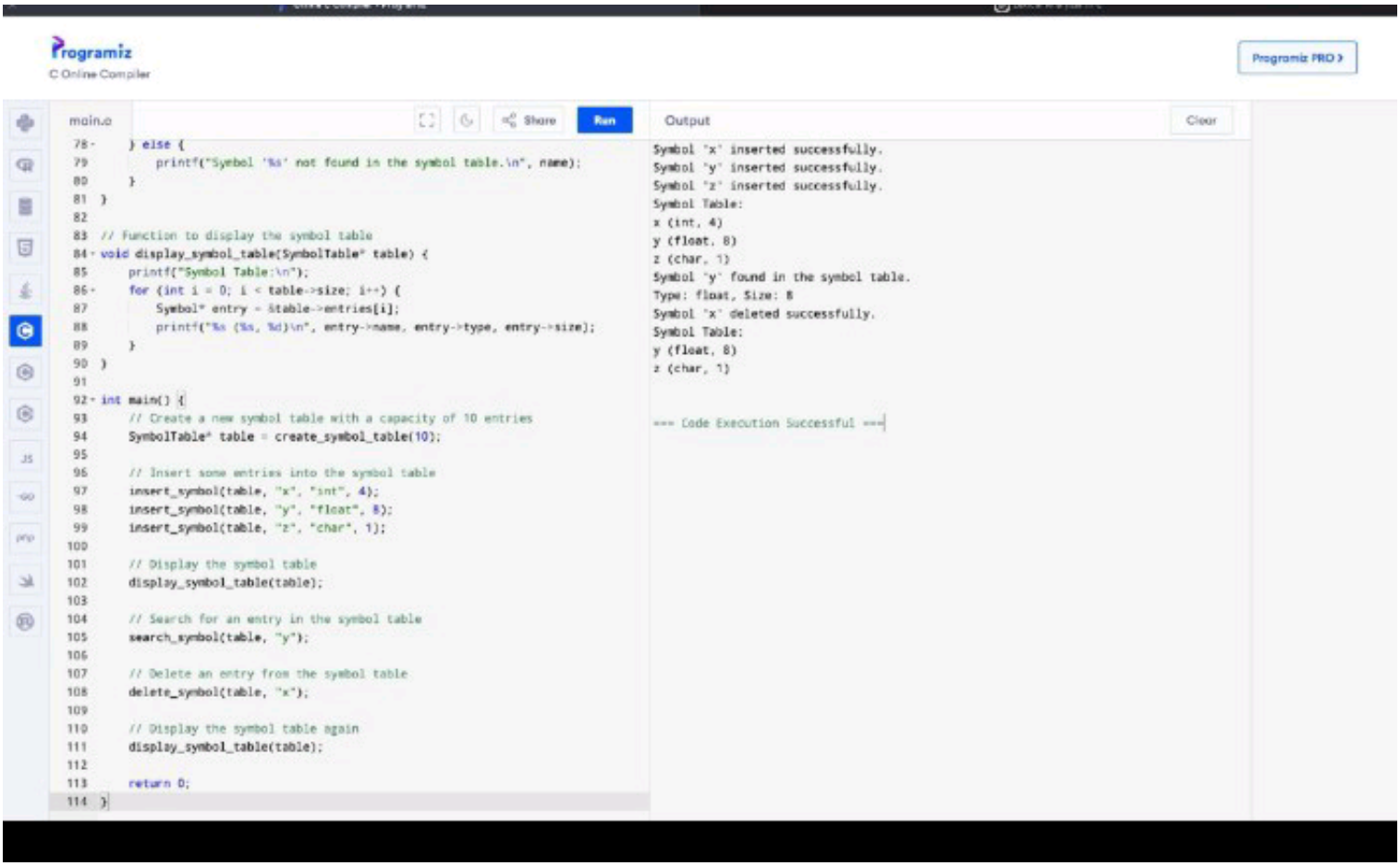

Online C Compiler - Programiz
Lexical Analyzer in C
Programiz
C Online Compiler
Programiz PRO >
main.c
Share
Run
Output
Clear
//question 3
#include <stdio.h>
#include <string.h>
int is_operator(char c) {
char operators[] = {'+', '-', '*', '/'};
for (int i = 0; i < 4; i++) {
if (c == operators[i]) {
return 1;
}
}
return 0;
}
int main() {
char input[1024];
printf("Enter input: ");
fgets(input, 1024, stdin);
for (int i = 0; i < strlen(input); i++) {
if (is_operator(input[i])) {
printf("Operator '%c' recognized.\n", input[i]);
}
}
return 0;
}
Enter input: x*x=10
Operator '*' recognized.
=== Code Execution Successful ===


Online C Compiler - Programiz
Lexical Analyzer in C
Programiz
C Online Compiler
Programiz PRO ›
main.c
Share
Run
Output
Clear
//question 4
#include <stdio.h>
#include <string.h>
#include <ctype.h>
int main() {
char input[1024];
int whitespace_count = 0;
int newline_count = 0;
printf("Enter input: ");
fgets(input, 1024, stdin);
for (int i = 0; i < strlen(input); i++) {
if (isspace(input[i])) {
whitespace_count++;
if (input[i] == '\n') {
newline_count++;
}
}
}
printf("Number of whitespaces: %d\n", whitespace_count);
printf("Number of newline characters: %d\n", newline_count);
return 0;
}
Enter input: santhosh
Number of whitespaces: 1
Number of newline characters: 1
=== Code Execution Successful ===


Programiz
C Online Compiler
Programiz PRO
main.o
Share
Run
Output
Clear
//question 5
#include <stdio.h>
#include <ctype.h>
#include <string.h>
// Function to check if an identifier is valid
int is_valid_identifier(char* identifier) {
// Check if the identifier is empty
if (strlen(identifier) == 0) {
return 0;
}
// Check if the first character is a letter or underscore
if (!isalpha(identifier[0]) && identifier[0] != '_') {
return 0;
}
// Check if the remaining characters are letters, digits, or underscores
for (int i = 1; i < strlen(identifier); i++) {
if (!isalpha(identifier[i]) && !isdigit(identifier[i]) && identifier[i]
!= '_') {
return 0;
}
}
// If all checks pass, the identifier is valid
return 1;
}
int main() {
char identifier[1024];
printf("Enter an identifier: ");
scanf("%s", identifier);
if (is_valid_identifier(identifier)) {
printf("%s is a valid identifier.\n", identifier);
} else {
Enter an identifier: hello_san
hello_san is a valid identifier.
=== Code Execution Successful ===


Online C Compiler - Programiz
Lexical Analyzer in C
Programiz
C Online Compiler
Programiz PRO >
main.c
Share
Run
Output
Clear
for (int j = 1; j < strlen(production); j++) {
alpha[alpha_index++] = production[j];
}
alpha[alpha_index] = '\0';
// Create new production rules
printf("New production rules:\n");
printf("%c -> %s%c'\n", non_terminal, beta, non_terminal);
printf("%c' -> %s%c' | epsilon\n", non_terminal, alpha,
non_terminal);
free(beta);
free(alpha);
}
}
}
int main() {
// Define the production rules for the grammar
Production productions[] = {
{'E', "E+T"},
{'E', "T"},
{'T', "T*F"},
{'T', "F"},
{'F', "(E)"},
{'F', "id"}
};
int num_productions = sizeof(productions) / sizeof(productions[0]);
// Eliminate left recursion
eliminate_left_recursion(productions, num_productions);
return 0;
}
Left recursion detected for non-terminal 'E'
New production rules:
E -> TE'
E' -> +TE' | epsilon
Left recursion detected for non-terminal 'T'
New production rules:
T -> FT'
T' -> *FT' | epsilon
=== Code Execution Successful ===


Online C Compiler - Programiz
Lexical Analyzer in C
Programiz
C Online Compiler
Programiz PRO >
main.c
Share
Run
Output
Clear
// question 7
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
// Define the structure for a production rule
typedef struct {
char non_terminal;
char* production;
} Production;
// Function to eliminate left factoring
void eliminate_left_factoring(Production* productions, int num_productions) {
for (int i = 0; i < num_productions; i++) {
char non_terminal = productions[i].non_terminal;
char* production = productions[i].production;
// Find all productions with the same non-terminal and common prefix
char common_prefix[100];
int common_prefix_length = 0;
int num_common_productions = 0;
for (int j = 0; j < num_productions; j++) {
if (productions[j].non_terminal == non_terminal && strcmp
(production, productions[j].production) != 0) {
int k = 0;
while (k < strlen(production) && k < strlen(productions[j]
.production) && production[k] == productions[j]
.production[k]) {
k++;
}
if (k > common_prefix_length) {
common_prefix_length = k;
strncpy(common_prefix, production, k);
common_prefix[k] = '\0';
num_common_productions = 1;
Left factoring detected for non-terminal 'E'
New production rules:
E -> TE'
E' -> +F | epsilon
E' -> *F | epsilon
Left factoring detected for non-terminal 'E'
New production rules:
E -> TE'
E' -> +F | epsilon
E' -> *F | epsilon
Left factoring detected for non-terminal 'T'
New production rules:
T -> TT'
T' -> T*F | epsilon
T' -> F | epsilon
Left factoring detected for non-terminal 'T'
New production rules:
T -> TT'
T' -> T*F | epsilon
T' -> F | epsilon
Left factoring detected for non-terminal 'F'
New production rules:
F -> TF'
F' -> (E) | epsilon
F' -> id | epsilon
Left factoring detected for non-terminal 'F'
New production rules:
F -> TF'
F' -> (E) | epsilon
F' -> id | epsilon
=== Code Execution Successful ===


Programiz
C Online Compiler
Programiz PRO
main.c
Share
Run
Output
Clear
} else {
printf("Symbol '%s' not found in the symbol table.\n", name);
}
}
// Function to display the symbol table
void display_symbol_table(SymbolTable* table) {
printf("Symbol Table:\n");
for (int i = 0; i < table->size; i++) {
Symbol* entry = &table->entries[i];
printf("%s (%s, %d)\n", entry->name, entry->type, entry->size);
}
}
int main() {
// Create a new symbol table with a capacity of 10 entries
SymbolTable* table = create_symbol_table(10);
// Insert some entries into the symbol table
insert_symbol(table, "x", "int", 4);
insert_symbol(table, "y", "float", 8);
insert_symbol(table, "z", "char", 1);
// Display the symbol table
display_symbol_table(table);
// Search for an entry in the symbol table
search_symbol(table, "y");
// Delete an entry from the symbol table
delete_symbol(table, "x");
// Display the symbol table again
display_symbol_table(table);
return 0;
}
Symbol 'x' inserted successfully.
Symbol 'y' inserted successfully.
Symbol 'z' inserted successfully.
Symbol Table:
x (int, 4)
y (float, 8)
z (char, 1)
Symbol 'y' found in the symbol table.
Type: float, Size: 8
Symbol 'x' deleted successfully.
Symbol Table:
y (float, 8)
z (char, 1)
=== Code Execution Successful ===