**COMPILER DESIGN LAB PROGRAMS**

1.

#include <stdio.h>

#include <ctype.h>

#include <string.h>

#define MAX\_IDENTIFIER\_LENGTH 31

void checkToken(char \*token) {

if (isdigit(token[0])) {

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

} else if (isalpha(token[0])) {

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

} else {

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

}

}

int main() {

char code[1000], token[100];

int i = 0, j = 0;

printf("Enter your code (end with 'EOF' on a new line):\n");

while (fgets(code, sizeof(code), stdin)) {

if (strncmp(code, "EOF", 3) == 0) break;

for (i = 0; code[i] != '\0'; i++) {

if (isspace(code[i])) {

if (j > 0) {

token[j] = '\0';

checkToken(token);

j = 0;

}

} else if (ispunct(code[i])) {

if (j > 0) {

token[j] = '\0';

checkToken(token);

j = 0;

}

token[0] = code[i];

token[1] = '\0';

checkToken(token);

} else {

token[j++] = code[i];

}

}

}

return 0;

}

INPUT

int main() {

int x = 10 + 20;

return 0;

}

OUTPUT

Identifier: int

Identifier: main

Operator: (

Operator: )

Operator: {

Identifier: int

Identifier: x

Operator: =

Constant: 10

Operator: +

Constant: 20

Operator: ;

Identifier: return

Constant: 0

Operator: ;

Operator: }

2.

#include <stdio.h>

#include <string.h>

void checkComment(char \*line) {

if (strstr(line, "//") == line) {

// Single-line comment (starts with //)

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

} else if (strstr(line, "/\*") == line) {

// Multi-line comment (starts with /\* and ends with \*/)

if (strstr(line, "\*/")) {

printf("Multi-line comment: %s", line);

} else {

printf("Multi-line comment starts but does not end: %s", line);

}

} else {

printf("Not a comment: %s", line);

}

}

int main() {

char line[256];

printf("Enter your code line by line (type 'EOF' to end):\n");

while (fgets(line, sizeof(line), stdin)) {

if (strncmp(line, "EOF", 3) == 0) break;

checkComment(line);

}

return 0;

}

INPUT

// This is a single-line comment

/\* This is a multi-line comment \*/

int x = 10; // Variable declaration

/\* This is a multi-line comment that

spans multiple lines \*/

OUTPUT

Single-line comment: // This is a single-line comment

Multi-line comment: /\* This is a multi-line comment \*/

Not a comment: int x = 10; // Variable declaration

Multi-line comment starts but does not end: /\* This is a multi-line comment that

Not a comment: spans multiple lines \*/

3.

#include <stdio.h>

#include <string.h>

#include <ctype.h>

void validateOperator(char ch) {

switch (ch) {

case '+':

printf("Operator: Addition (+)\n");

break;

case '-':

printf("Operator: Subtraction (-)\n");

break;

case '\*':

printf("Operator: Multiplication (\*)\n");

break;

case '/':

printf("Operator: Division (/)\n");

break;

default:

printf("Not a recognized arithmetic operator: %c\n", ch);

}

}

int main() {

char input[100];

int i;

printf("Enter an expression or line of code: ");

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

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

if (input[i] == '+' || input[i] == '-' || input[i] == '\*' || input[i] == '/') {

validateOperator(input[i]);

}

}

return 0;

}

INPUT

a + b - c \* d / e

OUTPUT

Operator: Addition (+)

Operator: Subtraction (-)

Operator: Multiplication (\*)

Operator: Division (/)

4.

#include <stdio.h>

#include <ctype.h>

int main() {

char ch;

int whitespaces = 0, newlines = 0;

printf("Enter your input (Press Ctrl+D or Ctrl+Z to end):\n");

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

if (isspace(ch)) {

if (ch == '\n') {

newlines++;

} else {

whitespaces++;

}

}

}

printf("\nNumber of whitespaces: %d\n", whitespaces);

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

return 0;

}

INPUT

This is a sample text.

It has some spaces and a new line.

OUTPUT

Number of whitespaces: 10

Number of newline characters: 2

5.

#include <stdio.h>

#include <ctype.h>

#include <string.h>

#include <stdbool.h>

// Function to check if a string is a valid identifier

bool isValidIdentifier(char \*identifier) {

// Reserved keywords (add more as needed)

const char \*keywords[] = {"int", "float", "return", "if", "else", "while", "for", "void", NULL};

int i;

// Check if the identifier matches any keyword

for (i = 0; keywords[i] != NULL; i++) {

if (strcmp(identifier, keywords[i]) == 0) {

return false;

}

}

// The first character must be a letter or underscore

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

return false;

}

// Remaining characters can be letters, digits, or underscores

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

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

return false;

}

}

return true;

}

int main() {

char identifier[100];

printf("Enter an identifier: ");

scanf("%s", identifier);

if (isValidIdentifier(identifier)) {

printf("%s is a valid identifier.\n", identifier);

} else {

printf("%s is not a valid identifier.\n", identifier);

}

return 0;

}

INPUT

Int

OUTPUT

int is not a valid identifier.

6.

#include <stdio.h>

#include <string.h>

int main() {

char input[100], alpha[10], beta[10], nonTerminal;

printf("Enter the grammar (e.g., A->Aa|b): ");

scanf("%s", input);

nonTerminal = input[0]; // First character is the non-terminal

char \*alphaStart = strchr(input, '>') + 1;

sscanf(alphaStart, "%[^|]|%s", alpha, beta);

if (alpha[0] == nonTerminal) {

printf("Grammar has left recursion. Eliminating...\n");

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

printf("%c' -> %s%c' | ε\n", nonTerminal, alpha + 1, nonTerminal);

} else {

printf("Grammar does not have left recursion.\n");

}

return 0;

}

INPUT

A->Aa|b

OUTPUT

Grammar has left recursion. Eliminating...

A -> bA'

A' -> aA' | ε

7.

#include <stdio.h>

#include <string.h>

int main() {

char input[100], prefix[10], option1[10], option2[10], nonTerminal;

printf("Enter the grammar (e.g., A->ab|ac): ");

scanf("%s", input);

nonTerminal = input[0]; // First character is the non-terminal

// Extract the production rules after "->"

char \*ruleStart = strchr(input, '>') + 1;

// Split the productions into the common prefix and remaining options

sscanf(ruleStart, "%[^|]|%s", option1, option2);

int i = 0;

while (option1[i] == option2[i] && option1[i] != '\0') {

prefix[i] = option1[i];

i++;

}

prefix[i] = '\0';

// Display the left-factored grammar

printf("After eliminating left factoring:\n");

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

printf("%c' -> %s | %s\n", nonTerminal, &option1[i], &option2[i]);

return 0;

}

INPUT

A->ab|ac

OUTPUT

After eliminating left factoring:

A -> aA'

A' -> b | c

8.

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

// Structure for a symbol table entry

typedef struct {

char name[50];

char type[20];

int size;

} Symbol;

Symbol table[100]; // Array to store symbol table entries

int count = 0; // Count of entries in the symbol table

void insert(char \*name, char \*type, int size) {

strcpy(table[count].name, name);

strcpy(table[count].type, type);

table[count].size = size;

count++;

printf("Symbol '%s' inserted successfully.\n", name);

}

void display() {

printf("\nSymbol Table:\n");

printf("Name\tType\t\tSize\n");

printf("-------------------------------\n");

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

printf("%s\t%s\t%d\n", table[i].name, table[i].type, table[i].size);

}

}

int search(char \*name) {

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

if (strcmp(name, table[i].name) == 0) {

printf("Symbol '%s' found in the table.\n", name);

return i;

}

}

printf("Symbol '%s' not found in the table.\n", name);

return -1;

}

int main() {

int choice, size;

char name[50], type[20];

while (1) {

printf("\n1. Insert Symbol\n2. Search Symbol\n3. Display Table\n4. Exit\n");

printf("Enter your choice: ");

scanf("%d", &choice);

switch (choice) {

case 1:

printf("Enter Name, Type, and Size: ");

scanf("%s %s %d", name, type, &size);

insert(name, type, size);

break;

case 2:

printf("Enter Name to Search: ");

scanf("%s", name);

search(name);

break;

case 3:

display();

break;

case 4:

exit(0);

default:

printf("Invalid choice. Please try again.\n");

}

}

return 0;

}

INPUT

 Insert: x int 4

 Insert: y float 8

 Search: x

OUTPUT

Symbol 'x' inserted successfully.

Symbol 'y' inserted successfully.

Symbol 'x' found in the table.

Symbol Table:

Name Type Size

------------------------

x int 4

y float 8

9.

#include <stdio.h>

#include <string.h>

// Function to validate the grammar S -> aSb | ε

int isValidGrammar(char \*str) {

int i = 0, j = strlen(str) - 1;

while (i < j) {

if (str[i] == 'a' && str[j] == 'b') {

i++;

j--;

} else {

return 0; // Grammar not satisfied

}

}

// If we reach here, the string satisfies the grammar

return (i == j || i > j); // Allow empty or balanced strings

}

int main() {

char input[100];

printf("Enter the input string: ");

scanf("%s", input);

if (isValidGrammar(input)) {

printf("The input string satisfies the grammar.\n");

} else {

printf("The input string does not satisfy the grammar.\n");

}

return 0;

}

INPUT

S -> aSb | ε

 aabb

 aaabbb

 ab

OUTPUT

The input string satisfies the grammar.

The input string satisfies the grammar.

The input string satisfies the grammar.

10.

#include <stdio.h>

#include <ctype.h>

char \*input;

int error = 0;

void E();

void T();

void F();

void E() {

T();

if (\*input == '+') {

input++;

E();

}

}

void T() {

F();

if (\*input == '\*') {

input++;

T();

}

}

void F() {

if (\*input == '(') {

input++;

E();

if (\*input == ')') {

input++;

} else {

error = 1; // Missing closing parenthesis

}

} else if (isalnum(\*input)) {

input++; // Identifier or number

} else {

error = 1; // Invalid character

}

}

int main() {

char expr[100];

printf("Enter an expression: ");

scanf("%s", expr);

input = expr;

E();

if (\*input == '\0' && error == 0) {

printf("The expression is valid according to the grammar.\n");

} else {

printf("The expression is invalid.\n");

}

return 0;

}

INPUT

E -> T + E | T

T -> F \* T | F

F -> (E) | id

a+b\*(c+d)

OUTPUT

The expression is valid according to the grammar.

11.

#include <stdio.h>

#include <ctype.h>

int precedence(char op) {

if (op == '+' || op == '-') return 1;

if (op == '\*' || op == '/') return 2;

if (op == '^') return 3;

return 0;

}

int applyOperator(int a, int b, char op) {

switch (op) {

case '+': return a + b;

case '-': return a - b;

case '\*': return a \* b;

case '/': return a / b;

case '^': {

int result = 1;

for (int i = 0; i < b; i++) result \*= a;

return result;

}

default: return 0;

}

}

int evaluate(char \*expression) {

int values[100], valueTop = -1;

char operators[100];

int opTop = -1;

for (int i = 0; expression[i]; i++) {

if (isdigit(expression[i])) {

int val = 0;

while (i < strlen(expression) && isdigit(expression[i])) {

val = val \* 10 + (expression[i] - '0');

i++;

}

values[++valueTop] = val;

i--;

} else if (expression[i] == '(') {

operators[++opTop] = expression[i];

} else if (expression[i] == ')') {

while (opTop != -1 && operators[opTop] != '(') {

int b = values[valueTop--];

int a = values[valueTop--];

char op = operators[opTop--];

values[++valueTop] = applyOperator(a, b, op);

}

opTop--; // Pop '('

} else {

while (opTop != -1 && precedence(operators[opTop]) >= precedence(expression[i])) {

int b = values[valueTop--];

int a = values[valueTop--];

char op = operators[opTop--];

values[++valueTop] = applyOperator(a, b, op);

}

operators[++opTop] = expression[i];

}

}

while (opTop != -1) {

int b = values[valueTop--];

int a = values[valueTop--];

char op = operators[opTop--];

values[++valueTop] = applyOperator(a, b, op);

}

return values[valueTop];

}

int main() {

char expression[100];

printf("Enter an arithmetic expression (e.g., 3+5\*2): ");

scanf("%s", expression);

int result = evaluate(expression);

printf("Result: %d\n", result);

return 0;

}

INPUT

3+5\*2

OUTPUT

Result: 13

12.

#include <stdio.h>

#include <string.h>

void generateThreeAddressCode(char \*expression) {

char tempVar[10];

int tempCount = 1;

int i = 0;

printf("\nThree-Address Code:\n");

while (expression[i] != '\0') {

// Check for an operator in the expression

if (expression[i] == '\*' || expression[i] == '/' ||

expression[i] == '+' || expression[i] == '-') {

// Create a temporary variable for the operation

sprintf(tempVar, "t%d", tempCount++);

// Print the three-address code for the operation

printf("%s = %c %c %c\n", tempVar, expression[i - 1], expression[i], expression[i + 1]);

// Replace the result of the operation with the temporary variable

expression[i - 1] = tempVar[0];

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

expression[j - 2] = expression[j];

}

expression[strlen(expression) - 2] = '\0';

// Reset loop to re-evaluate the updated expression

i = 0;

} else {

i++;

}

}

}

int main() {

char expression[50];

printf("Enter the expression (e.g., a+b\*c): ");

scanf("%s", expression);

generateThreeAddressCode(expression);

return 0;

}

INPUT

a+b\*c

OUTPUT

Three-Address Code:

t1 = b \* c

t2 = a + t1

13.

#include <stdio.h>

#include <ctype.h>

int main() {

char ch;

int characters = 0, words = 0, lines = 0;

int inWord = 0;

printf("Enter text (Press Ctrl+D to finish input):\n");

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

// Count characters

characters++;

// Count lines

if (ch == '\n') {

lines++;

}

// Count words

if (isspace(ch)) {

inWord = 0;

} else if (!inWord) {

inWord = 1;

words++;

}

}

printf("\nNumber of characters: %d\n", characters);

printf("Number of words: %d\n", words);

printf("Number of lines: %d\n", lines);

return 0;

}

INPUT

Hello World!

This is a test.

OUTPUT

Number of characters: 27

Number of words: 6

Number of lines: 2

14.

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

typedef struct {

char expression[50];

int isEliminated; // Flag to indicate if the subexpression is eliminated

} Expression;

Expression expressions[100];

int count = 0;

void addExpression(char \*expr) {

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

if (strcmp(expressions[i].expression, expr) == 0) {

printf("Common subexpression found: %s\n", expr);

expressions[i].isEliminated = 1;

return;

}

}

strcpy(expressions[count].expression, expr);

expressions[count].isEliminated = 0;

count++;

}

void displayOptimizedExpressions() {

printf("\nOptimized Code:\n");

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

if (!expressions[i].isEliminated) {

printf("%s\n", expressions[i].expression);

}

}

}

int main() {

char input[100][50];

int n;

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

scanf("%d", &n);

getchar();

printf("Enter the expressions:\n");

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

fgets(input[i], sizeof(input[i]), stdin);

input[i][strcspn(input[i], "\n")] = '\0'; // Remove newline

addExpression(input[i]);

}

displayOptimizedExpressions();

return 0;

}

INPUT

Enter the number of expressions: 5

Enter the expressions:

x = a + b

y = a + b

z = x \* c

w = a + b

v = z + d

OUTPUT

Common subexpression found: x = a + b

Common subexpression found: x = a + b

Optimized Code:

x = a + b

z = x \* c

v = z + d