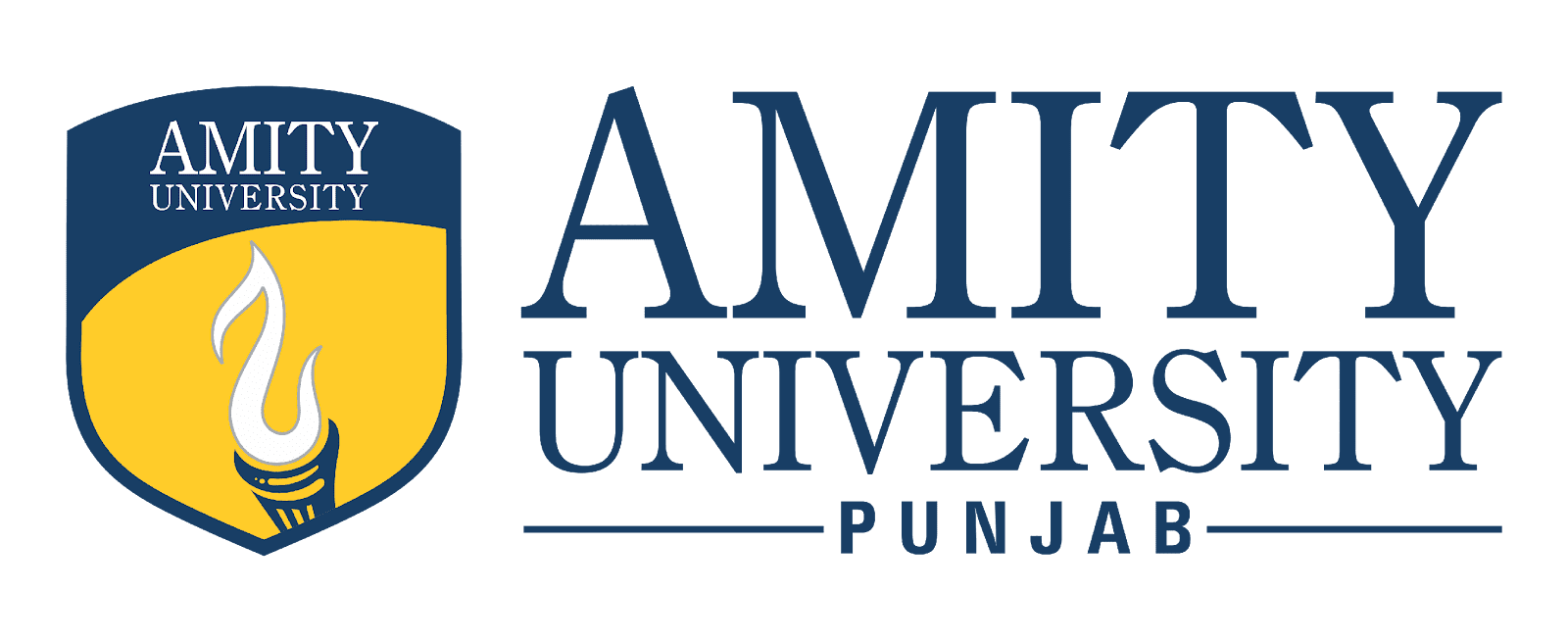
PRACTICAL FILE CD



**B. TECH-CSE-B**

**SEMESTER-6th**

**SUBMITTED TO: SUBMITTED BY:**

**DR. LAW KUMAR SINGH             HRIDYANSH LAKHAN**

**ASET, AMITY UNIVERSITY B.Tech CSE – 6th SEM**

**MOHALI A25305222166**

**CODE-1**

**WRITE A CALCULATOR PROGRAM IN C.**

#include <stdio.h>

void main() {

char operator;

double num1, num2, result;

printf("Enter an operator (+, -, \*, /): ");

scanf(" %c", &operator);

printf("Enter two numbers: ");

scanf("%lf %lf", &num1, &num2);

switch (operator) {

case '+':

result = num1 + num2;

break;

case '-':

result = num1 - num2;

break;

case '\*':

result = num1 \* num2;

break;

case '/':

if (num2 != 0)

result = num1 / num2;

else {

printf("Error! Division by zero.");

return;

}

break;

default:

printf("Error! Invalid operator.");

return;

}

printf("Result: %.2lf %c %.2lf = %.2lf\n", num1, operator, num2, result);

}

**OUTPUT:**

Enter an operator (+, -, \*, /): -

Enter two numbers: 4 3

Result: 4.00 - 3.00 = 1.00

**CODE-2**

**WRITE A CODE FOR MATRIX ADDITION IN C.**

#include <stdio.h>

#define ROWS 3

#define COLS 3

void inputMatrix(int matrix[ROWS][COLS], int rows, int cols) {

printf("Enter elements of the matrix (%dx%d):\n", rows, cols);

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

for (int j = 0; j < cols; j++) {

printf("Element [%d][%d]: ", i + 1, j + 1);

scanf("%d", &matrix[i][j]);

}

}

}

void displayMatrix(int matrix[ROWS][COLS], int rows, int cols) {

printf("Matrix:\n");

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

for (int j = 0; j < cols; j++) {

printf("%d ", matrix[i][j]);

}

printf("\n");

}

}

void addMatrices(int matrix1[ROWS][COLS], int matrix2[ROWS][COLS], int result[ROWS][COLS], int rows, int cols) {

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

for (int j = 0; j < cols; j++) {

result[i][j] = matrix1[i][j] + matrix2[i][j];

}

}

}

int main() {

int matrix1[ROWS][COLS], matrix2[ROWS][COLS], result[ROWS][COLS];

printf("Input for first matrix:\n");

inputMatrix(matrix1, ROWS, COLS);

printf("Input for second matrix:\n");

inputMatrix(matrix2, ROWS, COLS);

addMatrices(matrix1, matrix2, result, ROWS, COLS);

printf("First Matrix:\n");

displayMatrix(matrix1, ROWS, COLS);

printf("Second Matrix:\n");

displayMatrix(matrix2, ROWS, COLS);

printf("Resultant Matrix (Sum):\n");

displayMatrix(result, ROWS, COLS);

return 0;

}

**OUTPUT:**

First Matrix:

1 2 3

4 5 6

7 8 9

Second Matrix:

9 8 7

6 5 4

3 2 1

Resultant Matrix (Sum):

10 10 10

10 10 10

10 10 10

**CODE-3**

**WRITE A CODE FOR MATRIX MULTIPLICATION IN C.**

#include <stdio.h>

#define ROW1 3

#define COL1 3

#define ROW2 3

#define COL2 3

void inputMatrix(int matrix[][COL1], int rows, int cols) {

printf("Enter elements of the matrix (%dx%d):\n", rows, cols);

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

for (int j = 0; j < cols; j++) {

printf("Element [%d][%d]: ", i + 1, j + 1);

scanf("%d", &matrix[i][j]);

}

}

}

void displayMatrix(int matrix[][COL2], int rows, int cols) {

printf("Matrix:\n");

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

for (int j = 0; j < cols; j++) {

printf("%d ", matrix[i][j]);

}

printf("\n");

}

}

void multiplyMatrices(int matrix1[][COL1], int matrix2[][COL2], int result[][COL2], int row1, int col1, int row2, int col2)

for (int i = 0; i < row1; i++) {

for (int j = 0; j < col2; j++) {

result[i][j] = 0;

for (int k = 0; k < col1; k++) {

result[i][j] += matrix1[i][k] \* matrix2[k][j];

}

}

}

}

int main() {

int matrix1[ROW1][COL1], matrix2[ROW2][COL2], result[ROW1][COL2];

printf("Input for first matrix:\n");

inputMatrix(matrix1, ROW1, COL1);

printf("Input for second matrix:\n");

inputMatrix(matrix2, ROW2, COL2);

multiplyMatrices(matrix1, matrix2, result, ROW1, COL1, ROW2, COL2);

printf("First Matrix:\n");

displayMatrix(matrix1, ROW1, COL1);

printf("Second Matrix:\n");

displayMatrix(matrix2, ROW2, COL2);

printf("Resultant Matrix (Product):\n");

displayMatrix(result, ROW1, COL2);

return 0;

}

**OUTPUT:**

First Matrix:

1 2 3

4 5 6

7 8 9

Second Matrix:

9 8 7

6 5 4

3 2 1

Resultant Matrix (Product):

30 24 18

84 69 54

138 114 90

**CODE- 4**

**WRITE A PROGRAM IN C TO CHECK WHETHER A STRING IS CONSTANT OR NOT.**

#include <stdio.h>

#include <string.h>

int isConstantString(char str[]) {

int length = strlen(str);

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

if (str[i] != str[0]) {

return 0;

}

}

return 1;

}

int main() {

char str[100];

printf("Enter a string: ");

scanf("%s", str);

if (isConstantString(str)) {

printf("The string is a constant string.\n");

} else {

printf("The string is NOT a constant string.\n");

}

return 0;

}

**OUTPUT:**

Enter a string: ABC

The string is NOT a constant string.

**CODE- 5**

**WRITE A PROGRAM TO COUNT NUMBER OF LINES AND SPACES IN C.**

#include <stdio.h>

int main() {

char ch;

int lines = 0, spaces = 0;

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

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

if (ch == ' ') spaces++;

if (ch == '\n') lines++;

}

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

printf("Spaces: %d\n", spaces);

return 0;

**}**

**OUTPUT:**

Hello world!

Welcome to C.

(Press Ctrl+D)

Lines: 2

Spaces: 2

**CODE- 6**

**WRITE A PROGRAM TO CHECK IDENTIFIERS IN C.**

#include <stdio.h>

#include <ctype.h>

#include <string.h>

int isValidIdentifier(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;

}

char \*keywords[] = {"int", "return", "for", "while", "if", "else", "char", "float", "double", "struct", "switch", "case", "break"};

int num\_keywords = sizeof(keywords) / sizeof(keywords[0]);

for (int i = 0; i < num\_keywords; i++) {

if (strcmp(str, keywords[i]) == 0)

return 0;

}

return 1;

}

int main() {

char str[100];

printf("Enter an identifier: ");

scanf("%s", str);

if (isValidIdentifier(str))

printf("Valid identifier\n");

else

printf("Invalid identifier\n");

return 0;

}

**OUTPUT:**

Enter an identifier: my\_var

Valid identifier

Enter an identifier: 123abc

Invalid identifier

**CODE- 7**

**WRITE A PROGRAM TO CHECK KEYWORDS IN C.**

#include <stdio.h>

#include <string.h>

const char \*keywords[] = {

"auto", "break", "case", "char", "const", "continue", "default", "do",

"double", "else", "enum", "extern", "float", "for", "goto", "if",

"inline", "int", "long", "register", "restrict", "return", "short",

"signed", "sizeof", "static", "struct", "switch", "typedef", "union",

"unsigned", "void", "volatile", "while", "\_Bool", "\_Complex", "\_Imaginary",

"\_Alignas", "\_Alignof", "\_Atomic", "\_Generic", "\_Noreturn",

"\_Static\_assert", "\_Thread\_local"

};

#define TOTAL\_KEYWORDS (sizeof(keywords) / sizeof(keywords[0]))

int isKeyword(const char \*word) {

for (int i = 0; i < TOTAL\_KEYWORDS; i++) {

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

return 1;

}

}

return 0;

}

int main() {

char word[50];

printf("Enter a word: ");

scanf("%s", word);

if (isKeyword(word)) {

printf("\"%s\" is a keyword in C.\n", word);

} else {

printf("\"%s\" is NOT a keyword in C.\n", word);

}

return 0;

}

**OUTPUT:**

Enter a word: int

"int" is a keyword in C.

Enter a word: hello

"hello" is NOT a keyword in C.

**CODE- 8**

**WRITE A MENU BASED PROGRAM IN C TO CHECK KEYWORD, IDENTIFIER, SPACE AND CONSTANT (LEXICAL ANALYZER).**

#include <stdio.h>

#include <ctype.h>

#include <string.h>

const char \*keywords[] = {

"auto", "break", "case", "char", "const", "continue", "default", "do",

"double", "else", "enum", "extern", "float", "for", "goto", "if",

"inline", "int", "long", "register", "restrict", "return", "short",

"signed", "sizeof", "static", "struct", "switch", "typedef", "union",

"unsigned", "void", "volatile", "while", "\_Bool", "\_Complex", "\_Imaginary",

"\_Alignas", "\_Alignof", "\_Atomic", "\_Generic", "\_Noreturn",

"\_Static\_assert", "\_Thread\_local"

};

#define TOTAL\_KEYWORDS (sizeof(keywords) / sizeof(keywords[0]))

int isKeyword(const char \*word) {

for (int i = 0; i < TOTAL\_KEYWORDS; i++) {

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

return 1;

}

}

return 0;

}

void tokenize(char \*code) {

char token[100];

int i = 0, j = 0;

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

if (isspace(code[i])) {

i++;

continue;

}

if (isalpha(code[i]) || code[i] == '\_') {

j = 0;

while (isalnum(code[i]) || code[i] == '\_') {

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

}

token[j] = '\0';

if (isKeyword(token)) {

printf("[Keyword] -> %s\n", token);

} else {

printf("[Identifier] -> %s\n", token);

}

}

else if (isdigit(code[i])) {

j = 0;

while (isdigit(code[i]) || code[i] == '.') {

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

}

token[j] = '\0';

printf("[Number] -> %s\n", token);

}

else if (strchr("+-\*/=<>!&|", code[i])) {

printf("[Operator] -> %c\n", code[i]);

i++;

}

else if (strchr("(){}[],;:", code[i])) {

printf("[Punctuation] -> %c\n", code[i]);

i++;

}

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

while (code[i] != '\n' && code[i] != '\0') i++;

}

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

i += 2;

while (code[i] != '\0' && !(code[i] == '\*' && code[i+1] == '/')) i++;

if (code[i] != '\0') i += 2;

}

else {

printf("[Unknown] -> %c\n", code[i]);

i++;

}

}

}

int main() {

char code[1000];

printf("Enter C code (end with #):\n");

scanf("%[^#]", code);

printf("\nTokens:\n");

tokenize(code);

return 0;

}

**OUTPUT:**

int main() {

int x = 10;

float y = 3.14;

x = x + 5;

// This is a comment

return 0;

}

#

Tokens:

[Keyword] -> int

[Identifier] -> main

[Punctuation] -> (

[Punctuation] -> )

[Punctuation] -> {

[Keyword] -> int

[Identifier] -> x

[Operator] -> =

[Number] -> 10

[Punctuation] -> ;

[Keyword] -> float

[Identifier] -> y

[Operator] -> =

[Number] -> 3.14

[Punctuation] -> ;

[Identifier] -> x

[Operator] -> =

[Identifier] -> x

[Operator] -> +

[Number] -> 5

[Punctuation] -> ;

[Keyword] -> return

[Number] -> 0

[Punctuation] -> ;

[Punctuation] -> }

**CODE- 9**

**WRITE A PROGRAM TO CHECK LEFT RECURSION IN C.**

#include <stdio.h>

#include <string.h>

void removeLeftRecursion(char nonTerminal, char alpha[], char beta[]) {

char newNonTerminal = nonTerminal + ('\'');

printf("Grammar without left recursion:\n");

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

printf("%c -> %s%c | ε\n", newNonTerminal, alpha, newNonTerminal);

}

int main() {

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

printf("Enter the non-terminal (e.g., A): ");

scanf(" %c", &nonTerminal);

printf("Enter alpha (recursive part): ");

scanf("%s", alpha);

printf("Enter beta (non-recursive part): ");

scanf("%s", beta);

removeLeftRecursion(nonTerminal, alpha, beta);

return 0;

}

**OUTPUT:**

Enter the non-terminal (e.g., A): A

Enter alpha (recursive part): ax

Enter beta (non-recursive part): b

Grammar without left recursion:

A -> bA'

A' -> axA' | ε