**Practical 3**

**Aim : Implementation of a Lexical Analyzer for C Language Compiler**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <ctype.h>

#define MAX\_TOKENS 1000

#define MAX\_LENGTH 100

const char \*keywords[] = {"int", "char", "return", "void", "long", "float", "struct", "scanf", "printf"};

const int num\_keywords = 9;

const char \*operators[] = {"+", "-", "\*", "/", "=", "==", "<", ">", "<=", ">="};

const int num\_operators = 10;

const char punctuations[] = {';', ',', '(', ')', '{', '}', '[', ']'};

typedef struct { char type[20], value[MAX\_LENGTH]; } Token;

Token tokens[MAX\_TOKENS];

int token\_count = 0;

char symbol\_table[MAX\_TOKENS][MAX\_LENGTH];

int symbol\_count = 0;

void add\_token(const char \*type, char \*value) {

strcpy(tokens[token\_count].type, type);

strcpy(tokens[token\_count].value, value);

token\_count++;

}

int is\_keyword(char \*word) {

for (int i = 0; i < num\_keywords; i++) if (strcmp(word, keywords[i]) == 0) return 1;

return 0;

}

int is\_operator(char \*op) {

for (int i = 0; i < num\_operators; i++) if (strcmp(op, operators[i]) == 0) return 1;

return 0;

}

int is\_punctuation(char ch) {

for (int i = 0; i < sizeof(punctuations); i++) if (ch == punctuations[i]) return 1;

return 0;

}

int is\_identifier(char \*word) {

if (!isalpha(word[0]) && word[0] != '\_') return 0;

for (int i = 1; word[i]; i++) if (!isalnum(word[i]) && word[i] != '\_') return 0;

return 1;

}

int is\_constant(char \*word) {

for (int i = 0; word[i]; i++) if (!isdigit(word[i])) return 0;

return 1;

}

int should\_ignore(char \*identifier) {

return (strcmp(identifier, "main") == 0 || strcmp(identifier, "demo") == 0 || strcmp(identifier, "program") == 0);

}

void add\_symbol(char \*identifier) {

if (should\_ignore(identifier)) return;

for (int i = 0; i < symbol\_count; i++) if (strcmp(symbol\_table[i], identifier) == 0) return;

strcpy(symbol\_table[symbol\_count++], identifier);

}

void tokenize(char \*code) {

char buffer[MAX\_LENGTH];

int i = 0, j = 0;

while (code[i]) {

if (isspace(code[i])) { i++; continue; }

if (isalpha(code[i]) || code[i] == '\_') { // Identifier or Keyword

j = 0;

while (isalnum(code[i]) || code[i] == '\_') buffer[j++] = code[i++];

buffer[j] = '\0';

if (is\_keyword(buffer)) add\_token("Keyword", buffer);

else { add\_token("Identifier", buffer); add\_symbol(buffer); }

}

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

j = 0;

while (isdigit(code[i])) buffer[j++] = code[i++];

buffer[j] = '\0';

add\_token("Constant", buffer);

}

else if (code[i] == '"' || code[i] == '\'') { // String

char quote = code[i++];

j = 0;

buffer[j++] = quote;

while (code[i] != quote && code[i]) buffer[j++] = code[i++];

buffer[j++] = quote;

buffer[j] = '\0';

i++;

add\_token("String", buffer);

}

else if (is\_punctuation(code[i])) { // Punctuation

buffer[0] = code[i++];

buffer[1] = '\0';

add\_token("Punctuation", buffer);

}

else { // Operator

j = 0;

while (!isalnum(code[i]) && !isspace(code[i]) && !is\_punctuation(code[i])) buffer[j++] = code[i++];

buffer[j] = '\0';

if (is\_operator(buffer)) add\_token("Operator", buffer);

}

}

}

void print\_results() {

printf("TOKENS\n");

for (int i = 0; i < token\_count; i++) printf("%s: %s\n", tokens[i].type, tokens[i].value);

printf("\nSYMBOL TABLE\n");

for (int i = 0; i < symbol\_count; i++) printf("%d) %s\n", i+1, symbol\_table[i]);

}

// Student structure and salary calculation part

struct student {

int id;

float cgpa;

long int bs, da, hra, gs;

};

// Function prototype for 'add' function

void add(int x, int y);

int main() {

// Part 1: Tokenizing code

char code[] =

"/\* Demo Program \*/\n"

"void main() {\n"

" int a = 5;\n"

" char b = 'x';\n"

" return a + b;\n"

"}";

tokenize(code);

print\_results();

// Part 2: Salary calculation and student structure

struct student s;

// Take basic salary as input

printf("\nEnter basic salary: ");

scanf("%ld", &s.bs);

// Calculate allowances

s.da = s.bs \* 0.40;

s.hra = s.bs \* 0.20;

s.gs = s.bs + s.da + s.hra;

// Display salary slip

printf("\n\nBasic Salary: %ld", s.bs);

printf("\nDA: %ld", s.da);

printf("\nHRA: %ld", s.hra);

printf("\nGross Salary: %ld", s.gs);

// Initialize student data

s.id = 10;

s.cgpa = 8.7;

// Display student information

printf("\n\nStudent ID: %d", s.id);

printf("\nCGPA: %.2f", s.cgpa);

// Function call for addition

int a = 10, b = 20;

add(a, b);

return 0;

}

// Add function definition

void add(int x, int y) {

int sum = x + y;

printf("\nSum of %d and %d is: %d", x, y, sum);

}