Compiler Design

1. Implement following Programs Using Lex

1. Generate Histogram of words

**Code:**

%{

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct {

char word[100]; /\* Word \*/

int count; /\* Frequency of the word \*/

} WordFreq;

WordFreq wordList[1000]; /\* A simple array to hold words and their frequencies \*/

int wordCount = 0; /\* Number of words processed \*/

void addWord(char \*word) {

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

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

wordList[i].count++;

return;}}

strcpy(wordList[wordCount].word, word);

wordList[wordCount].count = 1;

wordCount++;}

void printHistogram() {

printf("\nWord Histogram:\n");

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

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

printf("%s: ", wordList[i].word);

for (int j = 0; j < wordList[i].count; j++) {

printf("\*"); }

printf(" (%d)\n", wordList[i].count);}}

%}

%%

[ \t\n]+ { /\* Do nothing, skip whitespace \*/ }

[a-zA-Z]+ { addWord(yytext); }

%%

int main() {

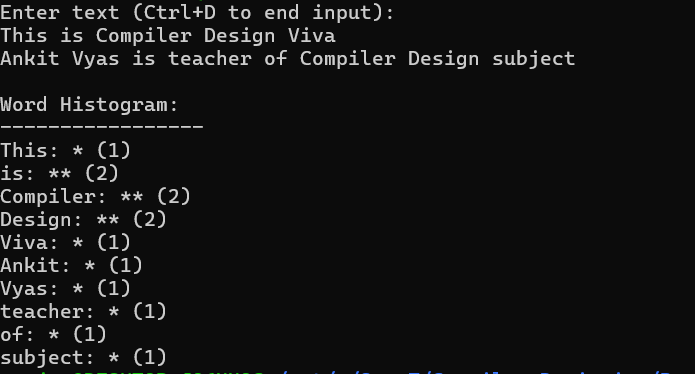
printf("Enter text (Ctrl+D to end input):\n");

yylex(); /\* Start the lexer \*/

printHistogram(); /\* Print the histogram \*/

return 0;}

**Output:**

****

1. Ceasor Cypher

**Code:**

%{

#include <stdio.h>

#include <ctype.h>

int shift = 3; // Default shift value for Caesar Cipher

char caesarShift(char ch, int shift) {

if (isupper(ch)) {

return ((ch - 'A' + shift) % 26) + 'A';

} else if (islower(ch)) {

return ((ch - 'a' + shift) % 26) + 'a';}

return ch; // Non-alphabetic characters remain unchanged}

%}

%%

[a-zA-Z] { putchar(caesarShift(yytext[0], shift)); } /\* Shift alphabetic characters \*/

. { putchar(yytext[0]); } /\* Print other characters as they are \*/

\n { putchar('\n'); } /\* Preserve newlines \*/

%%

int main(int argc, char \*argv[]) {

if (argc > 1) {

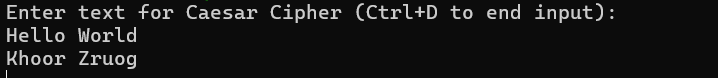
shift = atoi(argv[1]); // Optional shift argument from the command line}

printf("Enter text for Caesar Cipher (Ctrl+D to end input):\n");

yylex(); // Start the lexer

return 0;}

**Output:**

****

1. Extract single and multiline comments from C Program

**Code:**

%{

#include <stdio.h>

void printComment(const char \*comment) {

printf("%s\n", comment);}

%}

%%

"//".\* { printComment(yytext); } /\* Match single-line comments starting with // \*/

"/\*"(.|\n)\*?"\*/" { printComment(yytext); } /\* Match multi-line comments, allowing newlines \*/

. { /\* Ignore other characters \*/ }

\n { /\* Ignore newlines unless part of a comment \*/ }

%%

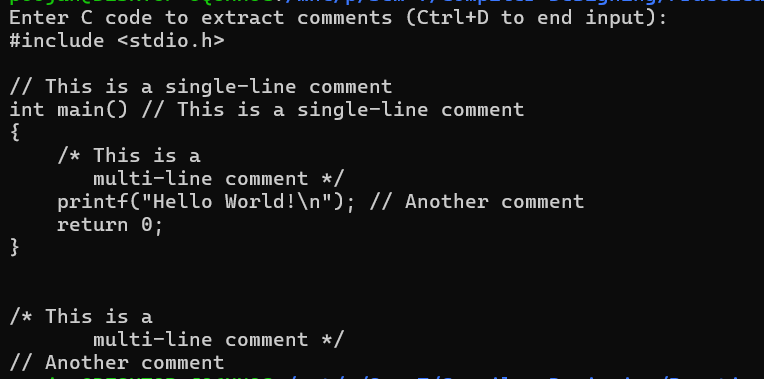
int main() {

printf("Enter C code to extract comments (Ctrl+D to end input):\n");

yylex(); // Start lexical analysis

return 0;}

**Output:**

****

**2.** Implement following Programs Using Lex

1. Convert Roman to Decimal

**Code:**

%{

#include <stdio.h>

int total = 0; // To store the result of Roman to Decimal conversion

void addValue(char roman) {

switch (roman) {

case 'I': total += 1; break;

case 'V': total += 5; break;

case 'X': total += 10; break;

case 'L': total += 50; break;

case 'C': total += 100; break;

case 'D': total += 500; break;

case 'M': total += 1000; break; }}

void subtractValue(char roman) {

switch (roman) {

case 'I': total -= 1; break;

case 'X': total -= 10; break;

case 'C': total -= 100; break; }}

%}

%%

"IV" { subtractValue('I'); addValue('V'); } /\* 4 \*/

"IX" { subtractValue('I'); addValue('X'); } /\* 9 \*/

"XL" { subtractValue('X'); addValue('L'); } /\* 40 \*/

"XC" { subtractValue('X'); addValue('C'); } /\* 90 \*/

"CD" { subtractValue('C'); addValue('D'); } /\* 400 \*/

"CM" { subtractValue('C'); addValue('M'); } /\* 900 \*/

"I" { addValue('I'); }

"V" { addValue('V'); }

"X" { addValue('X'); }

"L" { addValue('L'); }

"C" { addValue('C'); }

"D" { addValue('D'); }

"M" { addValue('M'); }

\n { printf("Decimal value: %d\n", total); total = 0; } /\* On newline, print the result and reset total \*/

. { /\* Ignore other characters \*/ }

%%

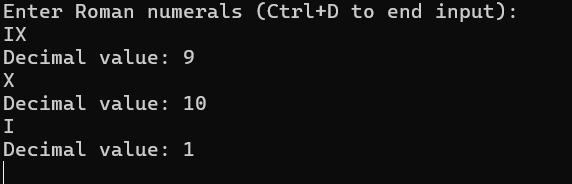
int main() {

printf("Enter Roman numerals (Ctrl+D to end input):\n");

yylex(); // Start lexical analysis

return 0; }

**Output:**

****

1. Check weather given statement is compound or simple

**Code**:

%{

#include <stdio.h>

#include <string.h>

int conjunction\_count = 0; // Variable to track the number of conjunctions

%}

%%

"and"|"or"|"but" { conjunction\_count++; } // Match conjunctions and increment counter

[a-zA-Z]+ { /\* Ignore simple words \*/ }

[ \t\n]+ { /\* Ignore whitespaces and newlines \*/ }

%%

int main() {

char input[256];

while (1) {

printf("\nEnter a statement (type 'exit' to quit):\n");

fgets(input, sizeof(input), stdin); // Read user input

if (strncmp(input, "exit", 4) == 0) {

break;}

conjunction\_count = 0; // Reset conjunction count for the new input

yy\_scan\_string(input); // Pass the input to the lexical analyzer

yylex(); // Begin scanning the input

if (conjunction\_count > 0) {

printf("The given statement is a compound statement.\n");

} else {

printf("The given statement is a simple statement.\n"); }}

printf("Program exited.\n");

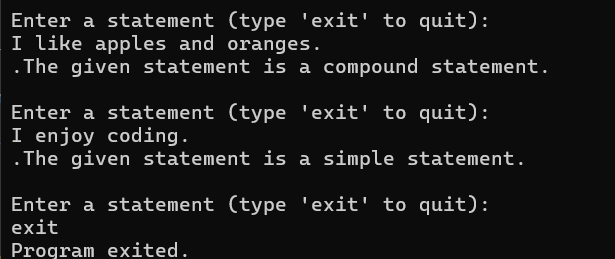
return 0;}

int yywrap() {

return 1;

}

**Output:**

****

1. Extract html tags from .html file

**Code**:

%{

#include <stdio.h>

%}

%%

"<"[^>]\*">" { printf("Tag found: %s\n", yytext); } // Match HTML tags and print them

[^<]+ { /\* Ignore text outside of tags \*/ }

%%

int main(int argc, char \*argv[]) {

if (argc < 2) {

printf("Usage: %s <html\_file>\n", argv[0]);

return 1; }

FILE \*html\_file = fopen(argv[1], "r");

if (!html\_file) {

printf("Error: Could not open file %s\n", argv[1]);

return 1; }

yyin = html\_file;

yylex(); // Start the lexical analysis

fclose(html\_file);

return 0;}

int yywrap() {

return 1;}

**Output:**

****

**3. Implementation of Recursive Descent Parser without backtracking Input: The string to be parsed. Output: Whether string parsed successfully or not. Explanation: Students have to implement the recursive procedure for RDP for a typical grammar. The production no. are displayed as they are used to derive the string.**

**Code:**

#include <stdio.h> #include <ctype.h> #include <string.h>

char input[100]; // To store the input string

int idx = 0; // Current index in the input string

void E(); // For Expression

void E\_prime(); // For Expression Prime

void T(); // For Term

void T\_prime(); // For Term Prime

void F(); // For Factor

void display\_rule(int rule) {

printf("Using production: %d\n", rule); }

int match(char terminal) {

if (input[idx] == terminal) {

idx++; // Move to the next character

return 1; }

return 0; }

void E() {

display\_rule(1); // Production: E -> T E'

T();

E\_prime(); }

void E\_prime() {

if (input[idx] == '+') {

display\_rule(2); // Production: E' -> + T E'

match('+');

T();

E\_prime(); // Recursive call for E'

} else { display\_rule(3); // Production: E' -> ε (empty) }}

void T() {

display\_rule(4); // Production: T -> F T'

F();

T\_prime(); }

void T\_prime() {

if (input[idx] == '\*') {

display\_rule(5); // Production: T' -> \* F T'

match('\*');

F();

T\_prime(); // Recursive call for T'

} else { display\_rule(6); // Production: T' -> ε (empty) }}

void F() {

if (input[idx] == '(') {

display\_rule(7); // Production: F -> ( E )

match('(');

E();

if (!match(')')) {

printf("Error: expected ')'\n");

return; }

} else if (isalpha(input[idx])) { // Match an identifier (id)

display\_rule(8); // Production: F -> id

while (isalnum(input[idx])) { idx++; // Move to the next character }

} else { printf("Error: expected id or '('\n"); }}

int main() {

printf("Enter the string to be parsed: ");

scanf("%s", input); // Read input string

E(); // Start parsing with the start symbol E

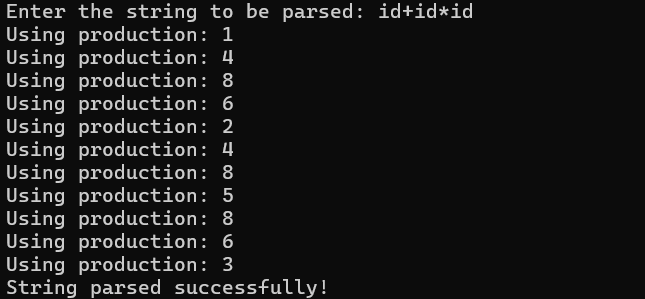
if (input[idx] == '\0') {

printf("String parsed successfully!\n");

} else {

printf("Error: unparsed input remaining at index %d: '%s'\n", idx, &input[idx]); } return 0;}

**Output:**

****

**4**. Introduction to YACC and generate Calculator Program

**Code:**

**calc.y**

%{

#include <stdio.h>

#include <stdlib.h>

int yylex(void);

void yyerror(const char \*);

int result; /\* Variable to store the result \*/

%}

%token NUMBER

%left '+' '-'

%left '\*' '/'

%%

calculation:

expression '\n' { printf("Result = %d\n", $1); result = $1; }

| /\* empty \*/

;

expression:

expression '+' expression { $$ = $1 + $3; }

| expression '-' expression { $$ = $1 - $3; }

| expression '\*' expression { $$ = $1 \* $3; }

| expression '/' expression {

if ($3 == 0) {

yyerror("Division by zero");

$$ = 0;

} else {

$$ = $1 / $3; }}

| '(' expression ')' { $$ = $2; }

| NUMBER { $$ = $1; }

;

%%

void yyerror(const char \*s) {

fprintf(stderr, "Error: %s\n", s); }

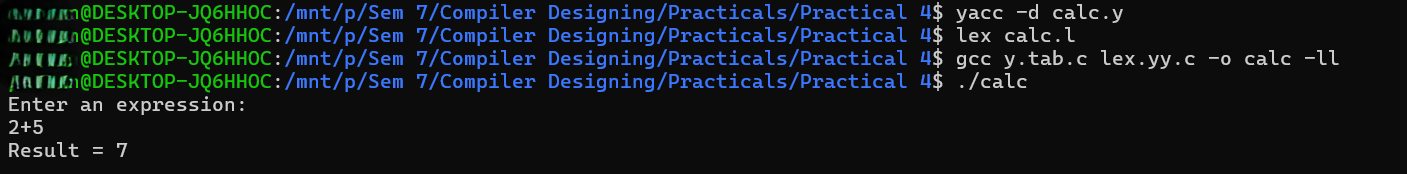
int main(void) {

printf("Enter an expression:\n");

yyparse();

return 0;**}**

**Output:**

****

**5. Finding “First” set Input: The string consists of grammar symbols. Output: The First set for a given string. Explanation: The student has to assume a typical grammar. The program when run will ask for the string to be entered. The program will find the First set of the given string.**

**Code:**

// C program to calculate the First and

// Follow sets of a given grammar

#include <ctype.h>

#include <stdio.h>

#include <string.h>

void followfirst(char, int, int);

void follow(char c);

void findfirst(char, int, int);

int count, n = 0;

char calc\_first[10][100];

char calc\_follow[10][100];

int m = 0;

char production[10][10];

char f[10], first[10];

int k;

char ck;

int e;

int main(int argc, char\*\* argv)

{

int jm = 0;

int km = 0;

int i, choice;

char c, ch;

count = 8;

strcpy(production[0], "X=TnS");

strcpy(production[1], "X=Rm");

strcpy(production[2], "T=q");

strcpy(production[3], "T=#");

strcpy(production[4], "S=p");

strcpy(production[5], "S=#");

strcpy(production[6], "R=om");

strcpy(production[7], "R=ST");

int kay;

char done[count];

int ptr = -1;

for (k = 0; k < count; k++) {

for (kay = 0; kay < 100; kay++) {

calc\_first[k][kay] = '!'; }}

int point1 = 0, point2, xxx;

for (k = 0; k < count; k++) {

c = production[k][0];

point2 = 0;

xxx = 0;

for (kay = 0; kay <= ptr; kay++)

if (c == done[kay])

xxx = 1;

if (xxx == 1)

continue;void follow(char c)

{

int i, j;

if (production[0][0] == c) {

f[m++] = '$'; }

for (i = 0; i < 10; i++) {

for (j = 2; j < 10; j++) {

if (production[i][j] == c) {

if (production[i][j + 1] != '\0') {

// Calculate the first of the next

// Non-Terminal in the production

followfirst(production[i][j + 1], i,

(j + 2));

}

if (production[i][j + 1] == '\0'

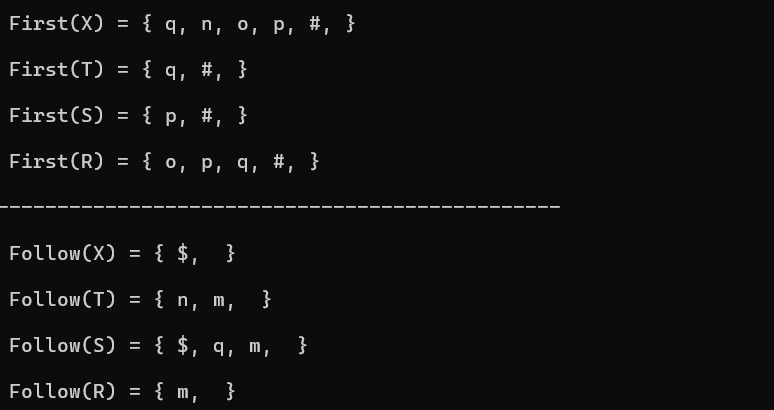
&& c != production[i][0]) {

follow(production[i][0]); }}

void findfirst(char c, int q1, int q2)

return;

**Output:**

****

**6. Extract Predecessor and Successor from given Control Flow Graph**

**Code:**

#include <stdio.h>

#include <stdlib.h>

struct Node {

int vertex;

struct Node\* next;

};

struct AdjacencyList {

struct Node\* head;

};

struct Graph {

int numVertices;

struct AdjacencyList\* array;

};

struct Node\* newNode(int vertex) {

struct Node\* new\_node = (struct Node\*)malloc(sizeof(struct Node));

new\_node->vertex = vertex;

new\_node->next = NULL;

return new\_node;}

new\_node->next = graph->array[dest].head;

graph->array[dest].head = new\_node;}

void printPredecessorsSuccessors(struct Graph\* graph) {

printf("Predecessors:\n");

for (int i = 0; i < graph->numVertices; i++) {

printf("%d: ", i);

struct Node\* current = graph->array[i].head;

int first = 1; // To format output

while (current != NULL) {

if (current->vertex != i) { // Exclude self-loops

if (!first) {

printf(", "); }

printf("%d", current->vertex);

first = 0;}

current = current->next;

}

printf("\n"); }

self-loops

if (!first) {

printf(", ");}

printf("%d", current->vertex);

first = 0;}

current = current->next;}

printf("\n"); }}

int main() {

int vertices = 6; // Define the number of vertices

struct Graph\* graph = createGraph(vertices);

addEdge(graph, 0, 1);

addEdge(graph, 0, 2);

addEdge(graph, 1, 3);

addEdge(graph, 2, 3);

addEdge(graph, 3, 4);

printPredecessorsSuccessors(graph);

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

struct Node\* current = graph->array[i].head;

while (current != NULL) {

struct Node\* temp = current;

current = current->next;

free(temp); }}

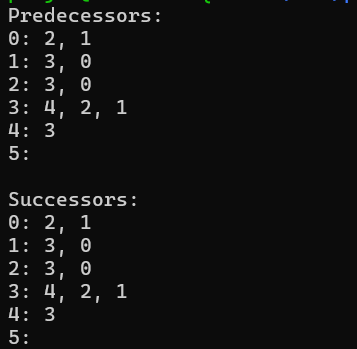
free(graph->array);

free(graph);

return 0;

}

**Output:**

****