9.Implement a C program to eliminate left recursion from a given CFG.

#include <stdio.h>

#include <string.h>

#define MAX 10

#define MAX\_PROD 10

#define MAX\_SYMBOLS 20

// Structure to store grammar

struct Grammar {

char nonTerminal;

char production[MAX][MAX\_SYMBOLS];

};

struct Grammar grammar[MAX\_PROD];

int numProds;

void eliminateLeftRecursion(int i) {

char newNonTerminal = grammar[i].nonTerminal + '\''; // New non-terminal A' for A -> A'

char temp[MAX\_SYMBOLS];

// Step 1: Separate productions into two lists: direct recursion and non-recursive

char newProduction[MAX][MAX\_SYMBOLS];

int j = 0, k = 0;

for (int p = 0; p < MAX\_PROD; p++) {

if (grammar[i].production[p][0] == '\0') break;

if (grammar[i].production[p][0] == grammar[i].nonTerminal) {

// Left recursive production, A -> Aα

strcpy(newProduction[j], &grammar[i].production[p][1]); // Remove left recursion (A -> α)

j++;

} else {

// Non-recursive production

strcpy(newProduction[k], grammar[i].production[p]);

k++;

}

}

// Step 2: Update the current non-terminal A with the non-recursive productions

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

strcpy(grammar[i].production[p], newProduction[p]);

}

grammar[i].production[k][0] = '\0'; // Mark end of productions

// Step 3: Add new productions for A'

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

sprintf(temp, "%s%c", newProduction[p], newNonTerminal); // A' -> αA'

strcpy(grammar[numProds].production[p], temp);

numProds++;

}

sprintf(temp, "%c%c", newNonTerminal, '\0'); // A' -> ε

strcpy(grammar[numProds].production[0], temp);

numProds++;

printf("Eliminating left recursion for non-terminal: %c\n", grammar[i].nonTerminal);

printf("Updated productions:\n");

for (int p = 0; p < numProds; p++) {

if (grammar[i].production[p][0] != '\0') {

printf("%c -> %s\n", grammar[i].nonTerminal, grammar[i].production[p]);

}

}

}

// Function to eliminate left recursion for all non-terminals

void processGrammar() {

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

eliminateLeftRecursion(i);

}

}

// Function to print the final grammar

void printGrammar() {

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

printf("%c -> ", grammar[i].nonTerminal);

for (int j = 0; j < MAX\_SYMBOLS; j++) {

if (grammar[i].production[j][0] == '\0') break;

printf("%s | ", grammar[i].production[j]);

}

printf("\n");

}

}

int main() {

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

scanf("%d", &numProds);

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

printf("Enter production %d (in the form A->a or B->b etc.): ", i + 1);

char prod[MAX\_SYMBOLS];

scanf("%s", prod);

grammar[i].nonTerminal = prod[0];

int j = 0;

int k = 2;

while (prod[k] != '\0') {

if (prod[k] == '|') {

grammar[i].production[j][k - 2] = '\0';

j++;

} else {

grammar[i].production[j][k - 2] = prod[k];

k++;

}

}

}

processGrammar();

printf("Grammar after eliminating left recursion:\n");

printGrammar();

return 0;

}

