10. Implement a C program to eliminate left factoring from a given CFG.

Code:

#include <stdio.h>

#include <string.h>

#define MAX 10

#define MAX\_SYMBOLS 20

// Structure to store grammar

struct Grammar {

char nonTerminal;

char production[MAX][MAX\_SYMBOLS];

};

struct Grammar grammar[MAX];

int numProds;

// Function to find the common prefix of two strings

int commonPrefix(char \*str1, char \*str2) {

int i = 0;

while (str1[i] == str2[i] && str1[i] != '\0' && str2[i] != '\0') {

i++;

}

return i;

}

// Function to eliminate left factoring for a given non-terminal

void eliminateLeftFactoring(int i) {

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

char temp[MAX\_SYMBOLS];

// Check for the common prefix between the production rules

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

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

for (int k = j + 1; k < MAX; k++) {

if (grammar[i].production[k][0] == '\0') break; // End of productions

int prefixLen = commonPrefix(grammar[i].production[j], grammar[i].production[k]);

if (prefixLen > 0) {

// Left factoring required, so we have a common prefix

// Step 1: Create the new production A -> αA'

strncpy(temp, grammar[i].production[j], prefixLen);

temp[prefixLen] = '\0';

strcpy(grammar[i].production[j], temp);

// Step 2: Create the new non-terminal A' and update its productions

sprintf(temp, "%s%c", grammar[i].production[j], newNonTerminal); // New non-terminal A' for A -> A'

strcpy(grammar[numProds].production[0], temp); // A' -> β or γ

// Add the remaining parts of the productions after the common prefix

strcpy(temp, &grammar[i].production[j][prefixLen]);

strcpy(grammar[numProds].production[1], temp);

numProds++;

strcpy(grammar[numProds].production[1], &grammar[i].production[k][prefixLen]);

numProds++;

// Clear the second part of the production

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

return;

}

}

}

}

// Function to process the grammar and eliminate left factoring

void processGrammar() {

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

eliminateLeftFactoring(i);

}

}

// Function to print the 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 factoring:\n");

printGrammar();

return 0;

}

