Consider the Grammar:  
E->E+T | T  
T->T\*F | F  
F->id

Perform Shift Reduce parsing for the input string “id + id + id”.

#include <stdio.h>  
#include <string.h>  
  
struct ProductionRule  
{  
 char left[10];  
 char right[10];  
};  
  
int main()  
{  
 char input[20], stack[50], temp[50], ch[2], \*token1, \*token2, \*substring;  
 int i, j, stack\_length, substring\_length, stack\_top, rule\_count = 0;  
 struct ProductionRule rules[10];  
  
 stack[0] = '\0';  
  
 // User input for the number of production rules  
 printf("\nEnter the number of production rules: ");  
 scanf("%d", &rule\_count);  
  
 // User input for each production rule in the form 'left->right'  
 printf("\nEnter the production rules (in the form 'left->right'): \n");  
 for (i = 0; i < rule\_count; i++)  
 {  
 scanf("%s", temp);  
 token1 = strtok(temp, "->");  
 token2 = strtok(NULL, "->");  
 strcpy(rules[i].left, token1);  
 strcpy(rules[i].right, token2);  
 }  
  
 // User input for the input string  
 printf("\nEnter the input string: ");  
 scanf("%s", input);  
  
 i = 0;  
 while (1)  
 {  
 // If there are more characters in the input string, add the next character to the stack  
 if (i < strlen(input))  
 {  
 ch[0] = input[i];  
 ch[1] = '\0';  
 i++;  
 strcat(stack, ch);  
 printf("%s\t", stack);  
 for (int k = i; k < strlen(input); k++)  
 {  
 printf("%c", input[k]);  
 }  
 printf("\tShift %s\n", ch);  
 }  
   
 // Iterate through the production rules  
 for (j = 0; j < rule\_count; j++)  
 {  
 // Check if the right-hand side of the production rule matches a substring in the stack  
 substring = strstr(stack, rules[j].right);  
 if (substring != NULL)  
 {  
 // Replace the matched substring with the left-hand side of the production rule  
 stack\_length = strlen(stack);  
 substring\_length = strlen(substring);  
 stack\_top = stack\_length - substring\_length;  
 stack[stack\_top] = '\0';  
 strcat(stack, rules[j].left);  
 printf("%s\t", stack);  
 for (int k = i; k < strlen(input); k++)  
 {  
 printf("%c", input[k]);  
 }  
 printf("\tReduce %s->%s\n", rules[j].left, rules[j].right);  
 j = -1; // Restart the loop to ensure immediate reduction of the newly derived production rule  
 }  
 }  
  
 // Check if the stack contains only the start symbol and if the entire input string has been processed  
 if (strcmp(stack, rules[0].left) == 0 && i == strlen(input))  
 {  
 printf("\nAccepted");  
 break;  
 }  
  
 // Check if the entire input string has been processed but the stack doesn't match the start symbol  
 if (i == strlen(input))  
 {  
 printf("\nNot Accepted");  
 break;  
 }  
 }  
  
 return 0;  
}

