**Program - 9**

**Aim**

Write a program to perform Left Factoring on a Grammar.

**Code**

#include <stdio.h>  
#include <string.h>  
  
void leftFactoring(char\* nonTerminal, char\* productions) {  
 char commonPrefix[100], remaining1[100], remaining2[100];  
 char\* prod1 = strtok(productions, "|");  
 char\* prod2 = strtok(NULL, "|");  
  
 int i = 0;  
 while (prod1[i] == prod2[i] && prod1[i] != '\0' && prod2[i] != '\0') {  
 commonPrefix[i] = prod1[i];  
 i++;  
 }  
 commonPrefix[i] = '\0';  
  
 if (strlen(commonPrefix) > 0) {  
 printf("%s -> %s%s'\n", nonTerminal, commonPrefix, nonTerminal);  
 printf("%s' -> %s | %s\n", nonTerminal, prod1 + i, prod2 + i);  
 } else {  
 printf("No left factoring needed.\n");  
 }  
}

**Output**

Example Input: A -> ab|ac  
Example Output:  
A -> aA'  
A' -> b | c

**Program - 10**

**Aim**

Write a program to show all the operations of a stack.

**Code**

#include <stdio.h>  
#define MAX 5  
  
int stack[MAX], top = -1;  
  
void push(int x) {  
 if (top < MAX - 1) {  
 stack[++top] = x;  
 printf("Pushed %d onto stack\n", x);  
 } else {  
 printf("Stack Overflow\n");  
 }  
}  
  
void pop() {  
 if (top >= 0) {  
 printf("Popped %d from stack\n", stack[top--]);  
 } else {  
 printf("Stack Underflow\n");  
 }  
}  
  
void display() {  
 if (top >= 0) {  
 printf("Stack: ");  
 for (int i = 0; i <= top; i++) {  
 printf("%d ", stack[i]);  
 }  
 printf("\n");  
 } else {  
 printf("Stack is empty\n");  
 }  
}

**Output**

Operations:  
push(10), push(20), pop(), display()  
  
Expected Output:  
Pushed 10 onto stack  
Pushed 20 onto stack  
Popped 20 from stack  
Stack: 10

**Program - 11**

**Aim**

Write a program to find out the leading of the non-terminals in a grammar.

**Code**

#include <stdio.h>  
#include <string.h>  
  
void findLeading(char nonTerminal, char productions[10][10], int count) {  
 printf("Leading(%c) = {", nonTerminal);  
 for (int i = 0; i < count; i++) {  
 if (productions[i][0] == nonTerminal) {  
 if (productions[i][2] >= 'a' && productions[i][2] <= 'z') {  
 printf(" %c", productions[i][2]);  
 }  
 }  
 }  
 printf(" }\n");  
}

**Program - 12**

**Aim**

Write a program to Implement Shift Reduce parsing for a String.

**Code**

#include <stdio.h>  
#include <string.h>  
  
char stack[50];  
int top = -1;  
  
void push(char symbol) {  
 stack[++top] = symbol;  
}  
  
void pop() {  
 top--;  
}  
  
void display() {  
 for (int i = 0; i <= top; i++) {  
 printf("%c", stack[i]);  
 }  
}  
  
void check() {  
 if (stack[top] == 'b' && stack[top-1] == 'S' && stack[top-2] == 'a') {  
 pop(); pop(); pop();  
 push('S');  
 printf(" Reduced by: S -> aSb\n");  
 display();  
 }  
}  
  
int main() {  
 char input[20];  
 printf("Enter the input string: ");  
 scanf("%s", input);  
 for (int i = 0; i < strlen(input); i++) {  
 push(input[i]);  
 printf(" Shift\n");  
 check();  
 }  
 return 0;  
}

**Program - 13**

**Aim**

Write a program to find out the FIRST of the Non-terminals in a grammar.

**Code**

#include <stdio.h>  
#include <string.h>  
  
void findFirst(char nonTerminal, char productions[10][10], int count) {  
 printf("First(%c) = {", nonTerminal);  
 for (int i = 0; i < count; i++) {  
 if (productions[i][0] == nonTerminal) {  
 printf(" %c", productions[i][2]);  
 }  
 }  
 printf(" }\n");  
}

**Program - 14**

**Aim**

Write a program to check whether a grammar is operator precedence.

**Code**

#include <stdio.h>  
#include <string.h>  
#include <stdbool.h>  
  
bool isOperatorPrecedence(char\* grammar) {  
 for (int i = 0; grammar[i] != '\0'; i++) {  
 if ((grammar[i] == '+' || grammar[i] == '\*') && grammar[i + 1] == grammar[i]) {  
 return false;  
 }  
 }  
 return true;  
}  
  
int main() {  
 char grammar[100];  
 printf("Enter the grammar rules: ");  
 scanf("%s", grammar);  
 if (isOperatorPrecedence(grammar)) {  
 printf("The grammar is operator precedence.\n");  
 } else {  
 printf("The grammar is not operator precedence.\n");  
 }  
}

**Output**

Example Input: E -> E+E|E\*E|id  
Example Output: The grammar is operator precedence.