Date: 11/10/2023

EXPERIMENT-5

Aim:

Write a program to perform Left Factoring on a Grammar.

Code:

```
#include <bits/stdc++.h>

using namespace std;

void leftFactor(map<char, vector<string>>& productions, char
nonTerminal) {
    vector<string>& productionList = productions[nonTerminal];

    map<string, vector<string>> prefixGroups;
    for (const string& production : productionList) {
        if (!production.empty()) {
            prefixGroups[production.substr(0,
1)].push_back(production.substr(1));
        }
    }

    bool needLeftFactoring = false;
    for (const auto& group : prefixGroups) {
        if (group.second.size() > 1) {
            needLeftFactoring = true;
            break;
        }
    }

    if (!needLeftFactoring) {
        cout << "No Left Factoring needed for " << nonTerminal <<
endl;
        return;
    }

    cout << "Left Factoring for " << nonTerminal << ":" << endl;
    for (const auto& group : prefixGroups) {
        const vector<string>& groupProductions = group.second;
        if (groupProductions.size() > 1) {
            string commonPrefix = group.first;
            char newNonTerminal = nonTerminal;
```

```
            newNonTerminal++;
            productions[newNonTerminal].push_back(commonPrefix +
newNonTerminal);
            cout << nonTerminal << " -> " << commonPrefix <<
newNonTerminal << endl;

            for (const string& production : groupProductions) {
                if (production.empty()) {
                    productions[nonTerminal].push_back(string(1,
newNonTerminal));
                } else {
                    productions[nonTerminal].push_back(production);
                }
            }
        } else {
            for (const string& production : groupProductions) {
                productions[nonTerminal].push_back(production);
            }
        }
    }
}

int main() {
    map<char, vector<string>> productions;

    productions['S'] = {"abA", "abcB", "aC", "aD"};
    productions['A'] = {"x", "y"};
    productions['B'] = {"pq", "r"};
    productions['C'] = {"st"};
    productions['D'] = {"uv"};

    for (const auto& production : productions) {
        leftFactor(productions, production.first);
    }

    cout << "\nUpdated Grammar:" << endl;
    for (const auto& production : productions) {
        char nonTerminal = production.first;
        const vector<string>& productionList = production.second;
        for (const string& p : productionList) {
            cout << nonTerminal << " -> " << p << endl;
        }
    }

    return 0;
}
```

Date: 11/10/2023

Output:

```
No Left Factoring needed for A
No Left Factoring needed for B
No Left Factoring needed for C
No Left Factoring needed for D
Left Factoring for S:
S -> aT
No Left Factoring needed for T

Updated Grammar:
A -> x
A -> y
B -> pq
B -> r
C -> st
D -> uv
S -> abA
S -> abcB
S -> aC
S -> aD
S -> bA
S -> bcB
S -> C
S -> D
T -> aT
```

Date: 11/10/2023

EXPERIMENT-6

Aim:

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

Code:

```
#include <iostream>
using namespace std;

class Node{
    public:
    Node* prev;
    int data;
    Node* next;

    Node(int data){
        this->prev = NULL;
        this->data = data;
        this->next = NULL;
    }
};

class Stack{
    public:
    Node* top = NULL;

    void push(int d){
        if(top == NULL){
            Node* temp = new Node(d);
            top = temp;
        } else{
            Node* temp = new Node(d);
            top->next = temp;
            temp->next = NULL;
            temp->prev = top;
            top = temp;
        }
    }
    void pop(){
        Node* temp = top;
        top = temp->prev;
        temp->prev = NULL;
        delete temp;
    }
```

```
    void peek(){
        cout<<"\nPeek : "<<top->data<<endl;
    }
    bool empty(){
        return (top == NULL) ? 1 : 0;
    }

    void print(){
        Node* temp = top;
        cout<<"Stack is - "<<endl;
        while(temp != NULL){
            cout<<"|  "<<temp->data<<"  |"<<endl;
            temp = temp->prev;
        }
        cout<<"|_____|"<<endl;
    }
};

int main(){
    Stack st;

    st.push(7);
    st.push(4);

    cout<<"Before any operation: "<<endl;
    st.print();

    st.push(3);
    cout<<"\nAfter Push operation: "<<endl;
    st.print();
    st.peek();

    st.pop();
    cout<<"\nAfter Pop operation: "<<endl;
    st.print();

    st.peek();

    return 0;
}
```

Date: 11/10/2023

Output:

```
Before any operation:
Stack is -
|  4  |
|  7  |
|_____|

After Push operation:
Stack is -
|  3  |
|  4  |
|  7  |
|_____|

Peek : 3

After Pop operation:
Stack is -
|  4  |
|  7  |
|_____|

Peek : 4
```