Date: 06/09/2023

EXPERIMENT-1

AIM:

Write a program to check whether a given string belongs to a grammar or not.

PROGRAMS:

1. Grammar: S -> aS, S -> Sb, S -> ab

```
#include <bits/stdc++.h>
using namespace std;
int main() {
   string str;
   bool flag = true;
   cout << "The grammar is: S->aS, S->Sb, S->ab" << endl;
   cout << "Enter the string to be checked: ";
   cin >> str;
   int n = str.length();
   if (str[0] == 'a' && str[n - 1] == 'b'){
      for (int i = 1; i < str.length(); i++){
         if (str[i] == 'b'){
            flag = false;
         }
         else if (str[i] == 'a' && flag == false){
            cout << "String is not accepted";
            exit(0);
         }
      }
      cout << "String is accepted";
   }
   else{
      cout << "String is not accepted";
   }
   return 0;
}
```

OUTPUT:

```
The grammar is: S->aS, S->Sb, S->ab
Enter the string to be checked: aaabb
String is accepted
```

Date: 06/09/2023

2. Grammar: S -> aSa, S -> bSb, S -> a, S -> b

```
#include <bits/stdc++.h>
using namespace std;
int main() {
   string str;
   bool flag = true;
   cout << "The grammar is: S->aSa, S->bSb, S->a, S->b" << endl;
   cout << "Enter the string to be checked: ";
   cin >> str;
   int n = str.length();
   int a = 0, b = n - 1;
   if (n % 2 != 0){
      while (b > a){
         if (str[a] == str[b]){
            a++;
            b--;
         }
         else{
            cout << "String is not accepted";
            exit(0);
         }
      }
      cout << "String is accepted";
   }
   else{
      cout << "String is not accepted";
   }
   return 0;
}
```

OUTPUT:

```
The grammar is: S->aSa, S->bSb, S->a, S->b
Enter the string to be checked: abbabba
String is accepted
```

Date: 06/09/2023

3. Grammar: S -> aSbb, S -> abb

```
#include <iostream>
using namespace std;
int main() {
   string str;
   bool flag = true;
   int a_count = 0, b_count = 0;
   cout << "The grammar is: S->aSbb, S->abb" << endl;
   cout << "Enter the string to be checked: ";
   cin >> str;
   int n = str.length();
   if (str[0] == 'a' && str[n - 1] == 'b'){
      for (int i = 0; i < str.length(); i++){
         if (str[i] == 'a' && flag == false){
            cout << "String is not accepted";
            exit(0);
         }
         else if (str[i] == 'a' && flag == true){
            a_count++;
         }
         else if (str[i] == 'b'){
            b_count++;
            flag = false;
         }
      }
      if (b_count == 2 * a_count){
         cout << "String is accepted";
      }
      else {
         cout << "String is not accepted";
      }
   }
   else {
      cout << "String is not accepted";
   }
   return 0;
}
```

OUTPUT:

```
The grammar is: S->aSbb, S->abb
Enter the string to be checked: aaabbbbbb
String is accepted
```

Date: 06/09/2023

4. Grammar: S -> aSb, S -> ab

```
#include <iostream>
using namespace std;
int main() {
	string str;
	bool flag = true;
	int a_count = 0, b_count = 0;
	cout << "The grammar is: S->aSb, S->ab" << endl;
	cout << "Enter the string to be checked: ";
	cin >> str;
	int n = str.length();
	if (str[0] == 'a' && str[n - 1] == 'b'){
	for (int i = 0; i < str.length(); i++){
		if (str[i] == 'a' && flag == false){
			cout << "String is not accepted";
			exit(0);
		}
		else if (str[i] == 'a' && flag == true){
			a_count++;
		}
		else if (str[i] == 'b'){
			b_count++;
			flag = false;
		}
	}
	if (b_count == a_count){
		cout << "String is accepted";
	}
	else{
		cout << "String is not accepted";
	}}
	else{
	cout << "String is not accepted";
	}
	return 0;
}
```

OUTPUT:

```
The grammar is: S->aSb, S->ab
Enter the string to be checked: aaaabbbb
String is accepted
```

Date: 06/09/2023

VIVA-VOCE QUESTIONS:

Ques 1. What is the key feature of a CFG?

Ans 1. CFGs use production rules to generate strings in a language.

Ques 2. How do you determine if a string is in a CFG's language?

Ans 2. By constructing a parse tree for the string.

Ques 3. What are terminal symbols in a CFG?

Ans 3. Symbols that appear in the input string.

Ques 4. What is the significance of the Pumping Lemma for CFGs?

Ans 4. It helps identify non-context-free languages.

Ques 5. Are all programming languages context-free?

Ans 5. No, many programming languages have context-sensitive syntax.

Date: 13/09/2023

EXPERIMENT-2.1

Aim:

Write a program that give output ‘Compiler’ when given input is ‘Hi’ otherwise give output ‘Wrong’.

Code:

```
%option noyywrap
%{
	#include<stdio.h>
%}

%%
"hi" {printf("Compiler");}
.* {printf("Wrong");}

%%
int main()
{
	yylex();
	return 0;
}
```

Output:

```
PS C:\Users\ankus\OneDrive\Desktop\Compiler Design Lab\Exp-1> ./a.exe
hi
Compiler


bye
Wrong
```

Date: 13/09/2023

EXPERIMENT-2.2

Aim:

Write a program to check whether a number is even or odd.

Code:

```
%{
#include <stdio.h>
%}

%%
[0-9]+ {
   int n = atoi(yytext);
   if (n % 2 == 0) {
      printf("%d is Even\n", n);
   } else {
      printf("%d is Odd\n", n);
   }
}
.|\n    ;

%%
int main()
{
   yylex();
   return 0;
}
```

Output:

```
PS C:\Users\ankus\OneDrive\Desktop\Compiler Design Lab\Exp-2> ./a.exe
5
5 is Odd

12
12 is Even
```

Date: 27/09/2023

EXPERIMENT-3.1

Aim:

Write a program to check whether a string include Keyword or not.

Program:

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

int main() {
   string input;
   cout << "Enter a string: ";
   cin >> input;

   string keywords[] = {
      "auto", "break", "case", "char", "const", "continue", "default", "do",
      "double", "else", "enum", "extern", "float", "for", "goto", "if", "int",
      "long", "register", "return", "short", "signed", "sizeof", "static", "struct",
      "switch", "typedef", "union", "unsigned", "void", "volatile", "while"
   };

   bool isKeyword = false;

   for (const string& keyword : keywords) {
      if (input == keyword) {
         isKeyword = true;
         break;
      }
   }

   if (isKeyword) {
      cout << input << " is a C++ keyword." << endl;
   } else {
      cout << input << " is not a C++ keyword." << endl;
   }
   return 0;
}
```

Output:

```
Enter a string: do
do is a C++ keyword.
```

```
Enter a string: hi
hi is not a C++ keyword.
```

Date: 27/09/2023

EXPERIMENT-3.2

Aim:

Write a program to remove left Recursion from a Grammar.

Program:

```
#include<iostream>

#include<string>

using namespace std;

int main() {
   string ip,op1,op2,temp;
   int sizes[10] = {};
   char c;
   int n,j,l;
   cout<<"Enter the Parent Non-Terminal : ";
   cin>>c;
   ip.push_back(c);
   op1 += ip + "\'->";
   ip += "->";
   op2+=ip;
   cout<<"Enter the number of productions : ";
   cin>>n;
   for(int i=0;i<n;i++) {
      cout<<"Enter Production "<<i+1<<" : ";
      cin>>temp;
      sizes[i] = temp.size();
      ip+=temp;
      if(i!=n-1)
         ip += "|";
   }
   cout<<"Production Rule : "<<ip<<endl;
   for(int i=0,k=3;i<n;i++) {
      if(ip[0] == ip[k]) {
         cout<<"Production "<<i+1<<" has left recursion."<<endl;
         if(ip[k] != '#') {
            for(l=k+1;l<k+sizes[i];l++) {
               op1.push_back(ip[l]);
            }
            k=l+1;
            op1.push_back(ip[0]);
            op1 += "\'|";
         }
      }
```

```
        else {
            cout<<"Production "<<i+1<<" does not have left recursion."<<endl;
            if(ip[k] != '#') {
                for(j=k;j<k+sizes[i];j++) {
                    op2.push_back(ip[j]);
                }
                k=j+1;
                op2.push_back(ip[0]);
                op2 += "\'|";
            }
            else {
                op2.push_back(ip[0]);
                op2 += "\'";
            }
        }
    }
    op1 += "#";
    cout<<op2<<endl;
    cout<<op1<<endl;
    return 0;
}
```

Output:

```
Enter the Parent Non-Terminal : S
Enter the number of productions : 2
Enter Production 1 : Sa
Enter Production 2 : bS
Production Rule : S->Sa|bS
Production 1 has left recursion.
Production 2 does not have left recursion.
S->bSS'|
S'->aS'|#
```

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
```

Date: 29-11-23

EXPERIMENT-7

Aim:

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

Code:

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

using namespace std;

struct Production {
    char nonTerminal;
    string production;
};

unordered_map<char, unordered_set<char>> calculateLeading(const
vector<Production>& productions) {
    unordered_map<char, unordered_set<char>> leading;

    for (const Production& production : productions) {
        leading[production.nonTerminal];
        for (char c : production.production) {
            if (!isupper(c)) {
                leading[production.nonTerminal].insert(c);
                break;
            }
        }
    }

    bool changes = true;
    while (changes) {
        changes = false;
        for (const Production& production : productions) {
            char nonTerminal = production.nonTerminal;
            const string& body = production.production;

            for (char c : body) {
                if (isupper(c)) {
                    size_t originalSize = leading[nonTerminal].size();
                    leading[nonTerminal].insert(leading[c].begin(),
leading[c].end());

                    if (leading[nonTerminal].size() > originalSize) {
                        changes = true;
                    }
```

```
                    if (leading[c].find('\0') == leading[c].end()) {
                        break;
                    }
                } else {
                    size_t originalSize = leading[nonTerminal].size();
                    leading[nonTerminal].insert(c);

                    if (leading[nonTerminal].size() > originalSize) {
                        changes = true;
                    }

                    break;
                }
            }
        }
    }

    return leading;
}

int main() {
    vector<Production> productions = {
        {'S', "AB"},
        {'A', "aAB"},
        {'B', "bBc"}
    };

    unordered_map<char, unordered_set<char>> leading =
calculateLeading(productions);

    for (const auto& entry : leading) {
        cout << "Leading(" << entry.first << "): { ";
        for (char c : entry.second) {
            cout << c << ' ';
        }
        cout << '}' << endl;
    }

    return 0;
}
```

Output:

```
Leading(B): { b }
Leading(A): { a }
Leading(S): { a }
```

Date: 29-11-23

EXPERIMENT-8

Aim:

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

Code:

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

using namespace std;

struct Production {
    char left;
    string right;
};

void shift(stack<char> &stateStack, const string &input, size_t
&inputIndex) {
    stateStack.push(input[inputIndex]);
    inputIndex++;
}

void reduce(stack<char> &stateStack, const vector<Production>
&productions) {
    for (const auto &production : productions) {
        string rhs = production.right;
        string stackContents;
        size_t rhsLength = rhs.length();

        while (stateStack.size() >= rhsLength) {
            stackContents = "";
            for (size_t i = 0; i < rhsLength; i++) {
                stackContents = stateStack.top() + stackContents;
                stateStack.pop();
            }

            if (stackContents == rhs) {
                stateStack.push(production.left);
                return;
            }
        }
    }
}
```

```
bool shiftReduceParse(const string &input, const vector<Production>
&productions) {
    stack<char> stateStack;
    size_t inputIndex = 0;

    while (inputIndex < input.length()) {
        shift(stateStack, input, inputIndex);
        reduce(stateStack, productions);
    }

    return stateStack.size() == 1 && stateStack.top() ==
productions[0].left;
}

int main() {
    vector<Production> productions = {
        {'S', "iEtS"},
        {'S', "iEtSeS"},
        {'S', "a"},
        {'E', "b"},
    };

    string input = "iaebbea";

    bool isSuccess = shiftReduceParse(input, productions);

    cout << "Input: " << input << endl;
    if (isSuccess) {
        cout << "Accepted" << endl;
    } else {
        cout << "Rejected" << endl;
    }

    return 0;
}
```

Output:

```
Input: iaebbea
Accepted
```