COMPILER LAB ASSESSMENT - 4

Name: Karan Sehgal
Registration No: 22BCE3939

Ques 1

Problem Statement: Design a LALR Bottom Up Parser for the given grammar

Design and implement an LALR bottom up Parser for checking the syntax of the statements in the given language.

Code:

```
#include <iostream>
#include <stack>
#include <vector>
#include <string>
#include <map>
using namespace std;

map<string, vector<string>> grammar = {
    {"S", {"A"}},
    {"A", {"a", "B"}},
    {"B", {"b"}}};

map<pair<int, string>, string> parseTable = {
    {{0, "a"}, "S1"},
    {{0, "$"}, "error"},
    {{1, "b"}, "S2"},
    {{1, "$"}, "error"},
```

```
    {{2, "$"}, "accept"},
    {{2, "a"}, "error"},
    {{3, "b"}, "R1"},
    {{3, "$"}, "R1"},
    {{4, "$"}, "R2"}};

class LALRParser
{
public:
    void parse(const string &input)
    {
        stack<int> stateStack;
        stateStack.push(0);
        string action;
        string symbol;
        string inputWithEnd = input + "$";
            string::const_iterator it = inputWithEnd.begin();
        while (true)
        {
            int currentState = stateStack.top();
            symbol = *it;
            auto tableEntry = parseTable.find({currentState, symbol});
            if (tableEntry == parseTable.end())
            {
                cout << "Syntax error at symbol: " << symbol << endl;
                return;
            }
```

```
            action = tableEntry->second;
            if (action[0] == 'S')
            {
                int nextState = action[1] - '0';
                stateStack.push(nextState);
                it++;
                cout << "Shift to state: " << nextState << endl;
            }
            else if (action[0] == 'R')
            {
                int ruleNumber = action[1] - '0';
                applyRule(ruleNumber, stateStack);
                cout << "Reduce by rule: " << ruleNumber << endl;
            }
            else if (action == "accept")
            {
                cout << "Input accepted." << endl;
                return;
            }
            else
            {
                cout << "Error: " << action << endl;
                return;
            }
        }
    }
private:
```

```
    void applyRule(int ruleNumber, stack<int> &stateStack)
    {
        switch (ruleNumber)
        {
        case 1:
            stateStack.pop();
            stateStack.pop();
            stateStack.push(3);
            break;
        case 2:
            stateStack.pop();
            stateStack.push(4);
            break;
        }
    }
};
int main()
{
    LALRParser parser;
    string input;
    cout << "Enter a string to parse (e.g., ab):";
    cin >> input;
    parser.parse(input);
    return 0;
}
```

OUTPUT:

```
Enter a string to parse (e.g., ab):ab
Shift to state: 1
Shift to state: 2
Input accepted.


=== Code Execution Successful ===
```

```
Enter a string to parse (e.g., ab):abx
Shift to state: 1
Shift to state: 2
Syntax error at symbol: x


=== Code Execution Successful ===
```

Ques 2

Problem Statement : Design SLR Parser

Design SLR bottom up parser for the above language

ALGORITHM

SStep1:	Start
Step2:	Initially the parser has s0 on the stack where s0 is the initial state and w$ is in buffer
Step3:	Set ip point to the first symbol of w$
Step4:	repeat forever, begin
Step5:	Let S be the state on top of the stack and a symbol pointed to by ip
Step6:	If action [S, a] =shift S then begin Push S1 on to the top of the stack Advance ip to next input symbol
Step7:	Else if action [S, a], reduce A->B then begin Pop 2* |B| symbols of the stack Let S1 be the state now on the top of the stack
Step8:	Output the production A→B End
Step9:	else if action [S, a]=accepted, then return Else Error() End
Step10:	Stop

Code:

```
#include <iostream>
```

```
#include <stack>
#include <vector>
#include <string>
#include <map>
using namespace std;

map<int, pair<string, string>> grammar = {
    {1, {"S", "A"}},
    {2, {"A", "aB"}},
    {3, {"B", "bB"}},
    {4, {"B", "b"}}
};

map<pair<int, string>, string> parseTable = {
    {{0, "a"}, "S2"},
    {{0, "S"}, "1"},
    {{0, "A"}, "3"},
    {{1, "$"}, "acc"},
    {{2, "b"}, "S4"},
    {{2, "B"}, "5"},
    {{3, "$"}, "R1"},
    {{4, "b"}, "S6"},
    {{4, "$"}, "R4"},
    {{5, "$"}, "R2"},
    {{6, "b"}, "S6"},
    {{6, "$"}, "R3"}
};
```

```
class SLRParser {
public:
    void parse(const string &input) {
        stack<int> stateStack;
        stack<string> symbolStack;
        stateStack.push(0);
        symbolStack.push("$");

        string inputWithEnd = input + "$";
        size_t inputPos = 0;

        cout << "Parsing process:\n";
        cout << "Input string: " << input << "\n\n";

        while (true) {
            int currentState = stateStack.top();
            string currentSymbol(1, inputWithEnd[inputPos]);

            cout << "Stack: ";
            stack<int> tempStack = stateStack;
            vector<int> states;
            while (!tempStack.empty()) {
                states.push_back(tempStack.top());
                tempStack.pop();
            }
            for (auto it = states.rbegin(); it != states.rend(); ++it) {
                cout << *it << " ";
```

```
}
cout << "\tInput: " << inputWithEnd.substr(inputPos) << "\t";

auto tableEntry = parseTable.find({currentState, currentSymbol});
if (tableEntry == parseTable.end()) {
    cout << "\nError: No valid action for state " << currentState
         << " with symbol '" << currentSymbol << "'\n";
    return;
}

string action = tableEntry->second;
cout << "Action: " << action << "\n";

if (action[0] == 'S') {
    int nextState = stoi(action.substr(1));
    stateStack.push(nextState);
    symbolStack.push(string(1, inputWithEnd[inputPos]));
    inputPos++;
}
else if (action[0] == 'R') {
    int ruleNumber = stoi(action.substr(1));
    const auto &rule = grammar[ruleNumber];
    const string &lhs = rule.first;
    const string &rhs = rule.second;

    for (size_t i = 0; i < rhs.length(); i++) {
        stateStack.pop();
```

```
            symbolStack.pop();
        }

        symbolStack.push(lhs);

        int gotoState = stateStack.top();
        auto gotoEntry = parseTable.find({gotoState, lhs});
        if (gotoEntry == parseTable.end()) {
            cout << "Error: No valid GOTO action for state " << gotoState
                 << " with symbol " << lhs << "\n";
            return;
        }
        stateStack.push(stoi(gotoEntry->second));
      }
      else if (action == "acc") {
        cout << "\nInput string accepted!\n";
        return;
      }
      else {
        cout << "\nError: Invalid action " << action << "\n";
        return;
      }
    }
  }
};
```

```
int main() {
    SLRParser parser;
    string input;
    cout << "Enter a string to parse (e.g., abb): ";
    cin >> input;
    parser.parse(input);
    return 0;
}
```

OUTPUT:

```
Enter a string to parse (e.g., abb): abb
Parsing process:
Input string: abb

Stack: 0    Input: abb$ Action: S2
Stack: 0 2  Input: bb$  Action: S4
Stack: 0 2 4    Input: b$   Action: S6
Stack: 0 2 4 6  Input: $    Action: R3
Stack: 0 2 5    Input: $    Action: R2
Stack: 0 3  Input: $    Action: R1
Stack: 0 1  Input: $    Action: acc

Input string accepted!


=== Code Execution Successful ===
```

Ques 3

write a C program to implement the shift-reduce parsing algorithm.

TOOLS/APPARATUS: Turbo C or gcc / gprof compiler in linux.

Algorithm:

Grammar:

E->E+E

E->E*E

E->E/E

E->a/b

Method:

Stack	Input Symbol	Action
$	id1*id2$	shift
$id1	*id2 $	shift *
$*	id2$	shift id2
$id2	$	shift
$	$	accept

Shift: Shifts the next input symbol onto the stack.

Reduce: Right end of the string to be reduced must be at the top of the stack. Accept: Announce successful completion of parsing.

Error: Discovers a syntax error and call an error recovery routine.

Code:

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;
string s;
vector<char> st;
vector<char> a;
void StackAndInput() {
cout << "\n$";
```

```
for (auto x : st)
cout << x;
cout << "\t";
for (auto x : a)
cout << x;
}
void check() {
while (true) {
bool reduced = false;
if (!st.empty() && st.back() == 'a') {
st.pop_back();
st.push_back('E');
StackAndInput();
cout << "$\tREDUCE E -> a\n";
reduced = true;
}
if (st.size() >= 3 && st[st.size() - 1] == 'E' && st[st.size() -
2] == '+' && st[st.size() - 3] == 'E') {
st.pop_back();
st.pop_back();
st.pop_back();
st.push_back('E');
StackAndInput();
cout << "$\tREDUCE E -> E + E\n";
reduced = true;
}
if (st.size() >= 3 && st[st.size() - 1] == 'E' && st[st.size() -
```

```
2] == '*' && st[st.size() - 3] == 'E') {
st.pop_back();
st.pop_back();
st.pop_back();
st.push_back('E');
StackAndInput();
cout << "$\tREDUCE E -> E * E\n";
reduced = true;
}
if (!reduced) break;
}
}
int main() {
cout << "GRAMMAR is:\nE -> E + E\nE -> E * E\nE -> a\n";
cout << "Enter input string: ";
cin >> s;
for (char c : s)
a.push_back(c);
cout << "\nstack\t input\taction";
for (char c : a) {
st.push_back(c);
StackAndInput();
cout << "$\tSHIFT -> " << st.back() << "\n";
check();
}
if (st.size() == 1 && st[0] == 'E')
cout << "\n\nstring accepted\n";
```

```
else
cout << "\n\nstring rejected\n";
return 0;
}
```

OUTPUT:

```
GRAMMAR is:
E -> E + E
E -> E * E
E -> a
Enter input string: a+a

stack    input  action
$a  a+a$    SHIFT -> a

$E  a+a$    REDUCE E -> a

$E+ a+a$    SHIFT -> +

$E+a    a+a$    SHIFT -> a

$E+E    a+a$    REDUCE E -> a

$E  a+a$    REDUCE E -> E + E


string accepted


=== Code Execution Successful ===
```