LAB 6

Alok
220101048
Sec- ‘A’

1. Write a program to implement the predictive parser for the following grammar:

E->E+T|T

T->T*F|F

F->(E)|id

SOURCE CODE:

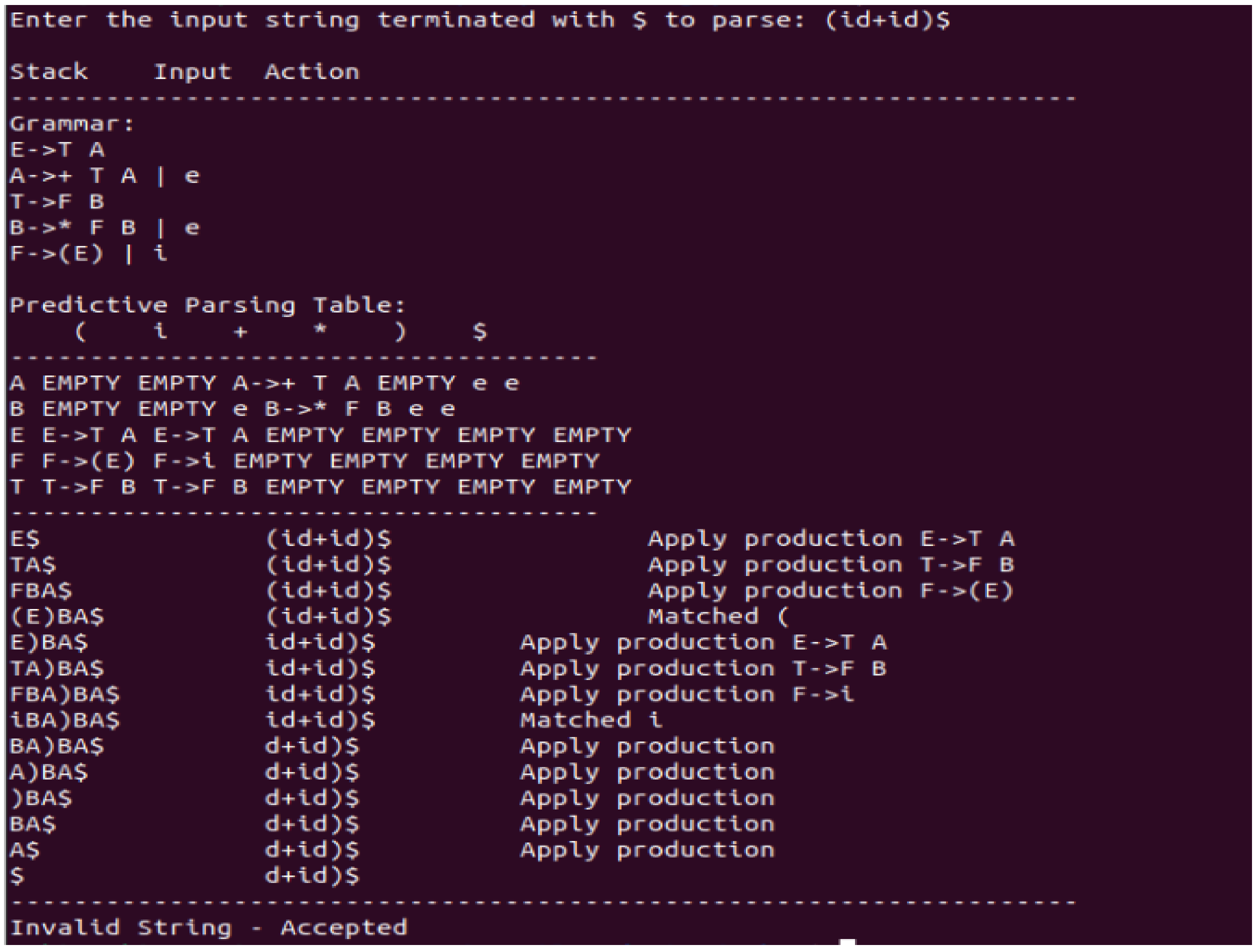
```
#include <iostream>
#include <string>
#include <map>
#include <stack>
using namespace std;
// Grammar
const string prod[] = {"E->T A", "A->+ T A | e", "T->F B", "B->* F B | e",
"F->(E) | i"};
const string first[] = {"i", "+, e", "i", "*, e", "i"};
const string follow[] = {"$, )", "$, )", "+, $, )", "+, $, )", "*, +, $, )"};
// Define the parsing table using map
map<string, map<char, string>> table;
// Initialize the parsing table
```

```
void initializeTable() {
table["E"]['('] = "E->T A";
table["E"]['i'] = "E->T A";
table["E"]['+'] = "EMPTY";
table["E"]['*'] = "EMPTY";
table["E"][')'] = "EMPTY";
table["E"]['$'] = "EMPTY";
table["A"]['('] = "EMPTY";
table["A"]['i'] = "EMPTY";
table["A"]['+'] = "A->+ T A";
table["A"]['*'] = "EMPTY";
table["A"][')'] = "e";
table["A"]['$'] = "e";
table["T"]['('] = "T->F B";
table["T"]['i'] = "T->F B";
table["T"]['+'] = "EMPTY";
table["T"]['*'] = "EMPTY";
table["T"][')'] = "EMPTY";
table["T"]['$'] = "EMPTY";
table["B"]['('] = "EMPTY";
table["B"]['i'] = "EMPTY";
table["B"]['+'] = "e";
table["B"]['*'] = "B->* F B";
table["B"][')'] = "e";
table["B"]['$'] = "e";
table["F"]['('] = "F->(E)";
table["F"]['i'] = "F->i";
table["F"]['+'] = "EMPTY";
table["F"]['*'] = "EMPTY";
table["F"][')'] = "EMPTY";
table["F"]['$'] = "EMPTY";
}
void display(stack<char> s) {
stack<char> temp = s;
while (!temp.empty()) {
```

```
cout << temp.top();
temp.pop();
}
}
void printGrammarAndTable() {
cout << "Grammar:\n";
for (const string& p : prod) {
cout << p << endl;
}
cout << "\nPredictive Parsing Table:\n";
cout << " ( i + * ) $\n";
cout << "-------------------------------------\n";
for (const auto& nonTerminal : table) {
cout << nonTerminal.first << " ";
for (char c : {'(', 'i', '+', '*', ')', '$'}) {
string value = table.at(nonTerminal.first).count(c) ?
table.at(nonTerminal.first).at(c) : "EMPTY";
cout << value << " ";
}
cout << endl;
}
cout << "-------------------------------------\n";
}
int main() {
string input;
cout << "Enter the input string terminated with $ to parse: ";
cin >> input;
if (input.back() != '$') {
cout << "Input String Entered Without End Marker $" << endl;
return 1;
}
stack<char> s;
s.push('$');
s.push('E');
int i = 0;
```

```
cout << "\nStack\t Input\tAction" << endl;
cout <<
"-----------------------------------------------------------------" << endl;
initializeTable();
printGrammarAndTable();
while (input[i] != '$' && !s.empty() && s.top() != '$') {
display(s);
cout << "\t\t" << input.substr(i) << "\t";
char stackTop = s.top();
char inputChar = input[i];
if (stackTop == inputChar) {
cout << "\tMatched " << inputChar << endl;
s.pop();
i++;
} else {
string stackTopStr(1, stackTop);
string inputCharStr(1, inputChar);
string curp = table[stackTopStr][inputChar];
if (curp == "e") {
cout << "\tApply epsilon production\n";
s.pop(); // Pop the non-terminal for epsilon production
// No need to push anything onto the stack for epsilon production
} else if (curp == "EMPTY") {
cout << "\nInvalid String - Rejected\n";
return 1;
} else {
cout << "\tApply production " << curp << endl;
s.pop();
// Push in reverse order, skipping '->'
for (int j = curp.size() - 1; j >= 3; j--) {
if (curp[j] != ' ' && curp[j] != '-') {
s.push(curp[j]);
}
}
}
}
}
display(s);
cout << "\t\t" << input.substr(i) << "\t";
cout <<
"\n------------------------------------------------------------------" <<
endl;
if (s.top() == '$' && input[i] == '$') {
cout << "\nValid String - Accepted\n";
} else {
cout << "Invalid String - Accepted\n";
}
return 0;
}
```

OUTPUT:



```
Enter the input string terminated with $ to parse: (id+id)$

Stack     Input   Action
--------------------------------------------------------------------
Grammar:
E->T A
A->+ T A | e
T->F B
B->* F B | e
F->(E) | i

Predictive Parsing Table:
    (    i    +    *    )    $
-------------------------------------
A EMPTY EMPTY A->+ T A EMPTY e e
B EMPTY EMPTY e B->* F B e e
E E->T A E->T A EMPTY EMPTY EMPTY EMPTY
F F->(E) F->i EMPTY EMPTY EMPTY EMPTY
T T->F B T->F B EMPTY EMPTY EMPTY EMPTY
-------------------------------------
E$               (id+id)$               Apply production E->T A
TA$              (id+id)$               Apply production T->F B
FBA$             (id+id)$               Apply production F->(E)
(E)BA$           (id+id)$               Matched (
E)BA$            id+id)$        Apply production E->T A
TA)BA$           id+id)$        Apply production T->F B
FBA)BA$          id+id)$        Apply production F->i
iBA)BA$          id+id)$        Matched i
BA)BA$           d+id)$         Apply production
A)BA$            d+id)$         Apply production
)BA$             d+id)$         Apply production
BA$              d+id)$         Apply production
A$               d+id)$         Apply production
$                d+id)$
--------------------------------------------------------------------
Invalid String - Accepted
```