EX.NO.5

Date:

Implementation of Predictive Parsing

AIM

To implement a Predictive Parser using C++ and demonstrate various operations like parsing input strings and validating grammar.

ALGORITHM

Step 1: Define the grammar for predictive parsing and construct the parsing table.
Step 2: Take the input grammar in the form of production rules.
Step 3: Compute the FIRST and FOLLOW sets of the grammar to create the parsing table.
Step 4: Accept the input string to be parsed.
Step 5: Use a stack and the parsing table to match the input string against the grammar.
Step 6: Continue parsing until the stack is empty or an error is encountered.
Step 7: Display whether the input string is accepted or rejected based on the parsing process.

PROGRAM

```
#include <iostream>

#include <stack>

#include <map>

#include <string>

#include <iomanip>


using namespace std;


class PredictiveParser {

private:

   map<string, map<char, string>> parsingTable;

   stack<string> parseStack;


   void displayStack(stack<string> s) {
```

```
    stack<string> temp;
    while (!s.empty()) {
        temp.push(s.top());
        s.pop();
    }
    while (!temp.empty()) {
        cout << temp.top() << " ";
        temp.pop();
    }
}

void displayParsingTable() {
    cout << "\nPredictive Parsing Table:" << endl;
    cout << setw(10) << "NT/T";

    map<char, bool> terminals;
    for (auto& row : parsingTable) {
        for (auto& entry : row.second) {
            terminals[entry.first] = true;
        }
    }

    for (auto& t : terminals) {
        cout << setw(10) << t.first;
    }
    cout << endl;
```

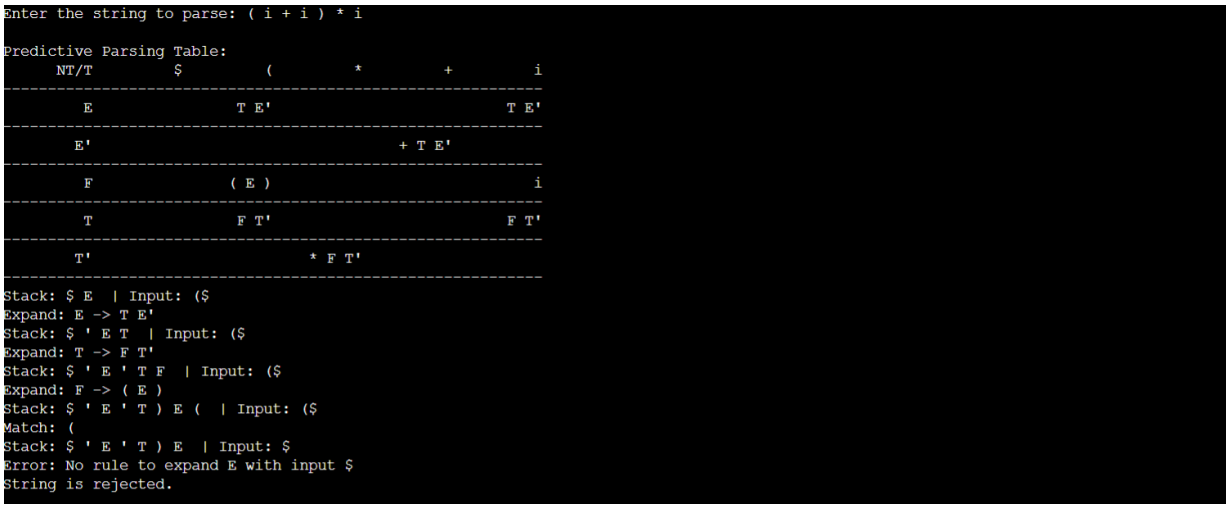
```
        cout << string(10 + terminals.size() * 10, '-') << endl;



        for (auto& row : parsingTable) {
            cout << setw(10) << row.first;
            for (auto& t : terminals) {
                string production = parsingTable[row.first][t.first];
                if (production == "") production = " ";
                cout << setw(10) << production;
            }
            cout << endl;


            cout << string(10 + terminals.size() * 10, '-') << endl;
        }
    }


public:
    PredictiveParser() {
        parsingTable["E"]['i'] = "T E'";
        parsingTable["E"]['('] = "T E'";
        parsingTable["E'"]['+'] = "+ T E'";
        parsingTable["E'"]['$'] = "";
        parsingTable["T"]['i'] = "F T'";
        parsingTable["T"]['('] = "F T'";
        parsingTable["T'"]['*'] = "* F T'";
        parsingTable["T'"]['+'] = "";
        parsingTable["T'"]['$'] = "";
```

```
    parsingTable["F"]['i'] = "i";
    parsingTable["F"]['('] = "( E )";
}

bool parse(string input) {
    input += "$";
    parseStack.push("$");
    parseStack.push("E");



    displayParsingTable();


    int i = 0;
    while (!parseStack.empty()) {
        cout << "Stack: ";
        displayStack(parseStack);
        cout << " | Input: " << input.substr(i) << endl;

        string top = parseStack.top();
        parseStack.pop();

        if (top.size() == 1 && top[0] == input[i]) {
            cout << "Match: " << input[i] << endl;
            i++;
        } else if (isupper(top[0])) {
            if (parsingTable[top][input[i]] != "") {
                string production = parsingTable[top][input[i]];
                cout << "Expand: " << top << " -> " << production << endl;
```

```
                for (int j = production.size() - 1; j >= 0; j--) {
                    if (production[j] != ' ')
                        parseStack.push(string(1, production[j]));
                }
            } else {
                cout << "Error: No rule to expand " << top << " with input " << input[i] << endl;
                return false;
            }
        } else {
            cout << "Error: Unexpected symbol " << top << endl;
            return false;
        }
    }

    return input[i] == '$';
  }
};

int main() {
  string input;
  cout << "Enter the string to parse: ";
  cin >> input;

  PredictiveParser parser;
  if (parser.parse(input)) {
    cout << "String is successfully parsed." << endl;
  } else {
```

```
        cout << "String is rejected." << endl;
    }

    return 0;
}
```

OUTPUT



RESULT

Predictive parsing was successfully implemented and demonstrated using a C++ program. The program parses input strings based on the given grammar and reports whether the string is valid or rejected.