LAB 6

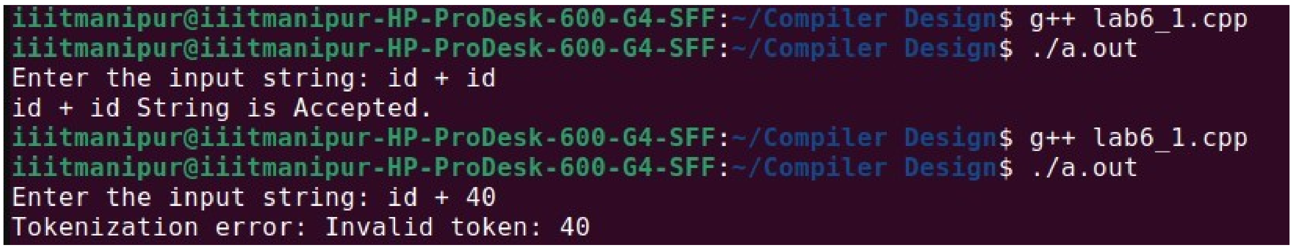
Name :- Amit Kumar
Roll No. :- 220103021
Section :- B

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

E->E+T|T
T->T*F|F
f->(E)|id

check whether the string ‘id+id’ is accepted or not .

```
#include <iostream>
#include <vector>
#include <string>
#include <sstream>
#include <stdexcept>

using namespace std;

class Parser {
public:
    // Constructor
    Parser(const vector<string>& tokens) : tokens(tokens), pos(0) {}

    // Public method to start parsing from the start symbol E
    void parse_E() {
        parse_T();
        parse_E_prime();
    }

private:
    vector<string> tokens; // List of tokens
    size_t pos; // Current position in the token list

    // Get the current token, or return an empty string if out of bounds
    string current_token() const {
        return pos < tokens.size() ? tokens[pos] : "";
    }

    // Consume the current token if it matches the expected type
    void consume(const string& token_type) {
        if (current_token() == token_type) {
            ++pos;
        } else {
            throw runtime_error("Expected " + token_type + ", but found " +
current_token());
        }
    }
    // Parse the T non-terminal
    void parse_T() {
        parse_F();
        parse_T_prime();
    }

    // Parse the E' non-terminal
    void parse_E_prime() {
        if (current_token() == "+") {
            consume("+");
            parse_T();
            parse_E_prime();
        }
        // If ε (empty), just return
    }

    // Parse the T' non-terminal
    void parse_T_prime() {
        if (current_token() == "*") {
            consume("*");
            parse_F();
            parse_T_prime();
        }
        // If ε (empty), just return
    }

    // Parse the F non-terminal
    void parse_F() {
        if (current_token() == "(") {
            consume("(");
            parse_E();
            consume(")");
        } else if (current_token() == "id") {
            consume("id");
        } else {
            throw runtime_error("Unexpected token " + current_token());
        }
    }
};

// Function to tokenize the input string
vector<string> tokenize(const string& input) {
    vector<string> tokens;
    stringstream ss(input);
    string token;

    while (ss >> token) {
        // Remove any surrounding parentheses
        if (token == "+" || token == "*" || token == "(" || token == ")" || token == "id") {
            tokens.push_back(token);
        } else if (token == "id") {
            tokens.push_back("id");
        } else {
            // For simplicity, handle single-character tokens directly
            if (token.length() == 1 && isalnum(token[0])) {
                tokens.push_back("id");
            } else {
                throw runtime_error("Invalid token: " + token);
            }
        }
    }

    return tokens;
}

int main() {
    string input_string;

    // Read the input string from the user
    cout << "Enter the input string: ";
    getline(cin, input_string);

    // Tokenize the input string
    vector<string> tokens;
    try {
        tokens = tokenize(input_string);
    } catch (const runtime_error& e) {
        cerr << "Tokenization error: " << e.what() << endl;
        return 1;
    }

    // Append end marker to the token list
    tokens.push_back("$");

    // Create a parser with the tokens
    try {
        Parser parser(tokens);
        parser.parse_E();
        cout << input_string<<" String is Accepted." << endl;
    } catch (const runtime_error& e) {
        cerr<<"Parsing error: " << e.what() << endl;
        return 1;
    }

    return 0;
}
```



```
iiitmanipur@iiitmanipur-HP-ProDesk-600-G4-SFF:~/Compiler Design$ g++ lab6_1.cpp
iiitmanipur@iiitmanipur-HP-ProDesk-600-G4-SFF:~/Compiler Design$ ./a.out
Enter the input string: id + id
id + id String is Accepted.
iiitmanipur@iiitmanipur-HP-ProDesk-600-G4-SFF:~/Compiler Design$ g++ lab6_1.cpp
iiitmanipur@iiitmanipur-HP-ProDesk-600-G4-SFF:~/Compiler Design$ ./a.out
Enter the input string: id + 40
Tokenization error: Invalid token: 40
```