VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Course Code: BCSE307P

Course Name: Compiler Design Lab

Assessment – 2

Name: Shivam Dave

Reg. No.: 21BCB0107

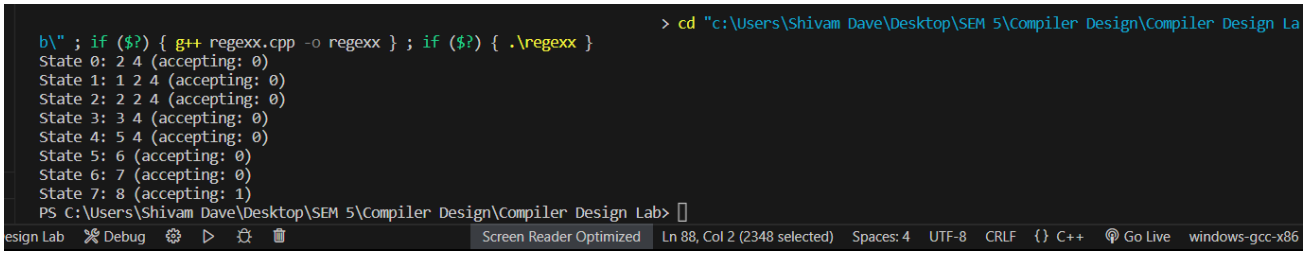
a) Write a program in C/C++ which constructs an equivalent DFA for the given regular expression by using direct method

Code:

```
#include <iostream>
#include <vector>

using namespace std;

// A struct to represent a state in the DFA
struct State {
  int id;
  vector<int> transitions;
  bool accepting;
};

// A function to construct the DFA from the given regular expression
vector<State> constructDFA(string regex) {
  // Create a syntax tree for the regular expression
  vector<State> states;
  State startState;
  startState.id = 0;
  startState.accepting = false;
  states.push_back(startState);

  for (int i = 0; i < regex.length(); i++) {
    char c = regex[i];

    // If the character is a literal, create a new state for it
    if (isalpha(c)) {
      State newState;
      newState.id = states.size();
      newState.transitions.push_back(i);
      newState.accepting = false;
      states.push_back(newState);
    }

    // If the character is a *, create a new state for it and connect it to
the
    // current state
    else if (c == '*') {
      State newState;
      newState.id = states.size();
      newState.transitions.push_back(i);
      newState.accepting = false;
      states.push_back(newState);
```

```
      for (int j = 0; j < states.size(); j++) {
        states[j].transitions.push_back(newState.id);
      }
    }

    // If the character is a |, create a new state for it and connect it to
the
    // current state and the next state
    else if (c == '|') {
      State newState;
      newState.id = states.size();
      newState.transitions.push_back(i);
      newState.accepting = false;
      states.push_back(newState);

      for (int j = 0; j < states.size(); j++) {
        states[j].transitions.push_back(newState.id);
      }
    }

    // If the character is a #, mark the current state as accepting
    else if (c == '#') {
      states[states.size() - 1].accepting = true;
    }
  }

  return states;
}

// A function to print the DFA
void printDFA(vector<State> dfa) {
  for (int i = 0; i < dfa.size(); i++) {
    cout << "State " << dfa[i].id << ": ";
    for (int j = 0; j < dfa[i].transitions.size(); j++) {
      cout << dfa[i].transitions[j] << " ";
    }
    cout << "(accepting: " << dfa[i].accepting << ")" << endl;
  }
}

int main() {
  string regex = "(a|b)*abb#";
  vector<State> dfa = constructDFA(regex);
  printDFA(dfa);

  return 0;
}
```

Output:



b) Write a program in C/C++ which eliminates the left recursion from the given grammar(the program should handle immediate and non immediate left recursion)

Code:

```
#include <iostream>
#include <string>

using namespace std;

int main() {
  int num;
  cout << "Enter Number of Production : ";
  cin >> num;

  string productions[num];
  for (int i = 0; i < num; i++) {
    cout << "Enter the grammar as S->S-A :\n";
    cin >> productions[i];
  }

  for (int i = 0; i < num; i++) {
    cout << "\nGRAMMAR : : : " << productions[i] << endl;
    char non_terminal = productions[i][0];
    int index = 3;

    if (non_terminal == productions[i][index]) {
      cout << " is left recursive.\n";
      while (productions[i][index] != 0 && productions[i][index] != '|') {
        index++;
      }

      if (productions[i][index] != 0) {
        char beta = productions[i][index + 1];
        cout << "Grammar without left recursion:\n";
        cout << non_terminal << "->" << beta << non_terminal;
```

```
        cout << "\n" << non_terminal << "'" << "->" << non_terminal << beta <<
non_terminal;
        cout << "|" << "E\n";
      } else {
        cout << " can't be reduced\n";
      }
    } else {
      cout << " is not left recursive.\n";
    }
  }

  return 0;
}
```

OUTPUT: