LAB EXERCISE – 2

(1) //DFA for the language of string over {0,1} in which each string ends with 11

```
#include <iostream>

#include <string>

using namespace std;


int main()

{

   string str;     // string to be checked

   char state = 0; // initial state (q0)


   cout << "Enter the string: ";

   cin >> str;


   // loop to check each character of the string for the DFA

   for (int i = 0; i < str.length(); i++)

   {

      // check if the string is over {0,1} or not

      if (str[i] != '0' && str[i] != '1')

      {

         cout << "String not accepted.\nPlease enter a string over {0,1}" << endl;

         return 0;
```

```
    }

    // dfa transition check
    if (state == 0 && str[i] == '0')
        state = 0;
    else if (state == 0 && str[i] == '1')
        state = 1;
    else if (state == 1 && str[i] == '0')
        state = 0;
    else if (state == 1 && str[i] == '1')
        state = 2;
    else if (state == 2 && str[i] == '0')
        state = 0;
    else if (state == 2 && str[i] == '1')
        state = 1;
}

// check if the string is accepted or not,
// i.e. if the final state is 2 then string is accepted
// else string is not accepted
if (state == 2)
    cout << "String accepted";
else
    cout << "String not accepted";
    return 0;
}
```

(2) /* **NFA for the language of string over {0,1} such that each string contains substring "101" */**

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

using namespace std;

// Define the NFA as a set of states and transitions
vector<int> states = {0, 1, 2, 3}; // States are represented by integers (0, 1, 2, ...)
vector<vector<pair<char, int>>> transitions = {
    {{'0', 0}, {'1', 0}, {'1', 1}},
    {{'0', 2}},
    {{'1', 3}},
    {{'0', 3}, {'1', 3}}}; // Transitions are represented by pairs of characters and states
(character, state)

// Define a function to simulate the NFA on a given string
bool simulate_nfa(string input)
{
    // Start at the initial state (state 0)
    vector<int> current_states = {0};
```

```
// Loop through each character in the input string
for (char c : input)
{
   //Find all possible transitions from the current states for the current character
   vector<int> next_states;
   for (int state : current_states)
   {
      for (auto transition : transitions[state])
      {
         if (transition.first == c)
         {
            next_states.push_back(transition.second);
         }
      }
   }
   // If there are no possible transitions, the input string is not accepted
   if (next_states.empty())
   {
      return false;
   }
   // Update the current states to the next states
   current_states = next_states;
}
```

```
    // If the final state is an accepting state, the input string is accepted
    for (int state : current_states)
    {
        if (state == 3)
        {
            return true;
        }
    }
    return false;
}

// Define the main function to run the program
int main()
{
    // Get input from the user
    string input;
    cout << "Enter a string to check: ";
    cin >> input;

    // Simulate the NFA on the input string and output the result
    if (simulate_nfa(input))
    {
        cout << "String contains substring 101." << endl;
    }
    else
    {
        cout << "String does not contain substring 101." << endl;
    }
    return 0;
}
```

(3) /* WAP to validate C identifiers and keywords */

```
#include <iostream>
#include <string.h>
#include <set>
using namespace std;

int main()
{
    char str[100];
    int i, l, flag = 0;

    cout << "Enter a string: ";
    cin >> str;
    l = strlen(str);

    // Check if first character is a letter
    if (!((str[0] >= 'a' && str[0] <= 'z') || (str[0] >= 'A' && str[0] <= 'Z') || str[0] == '_'))
    {
        cout << "Invalid identifier" << endl;
        return 0;
    }

    // Check if remaining characters are letters, digits or underscore
    for (i = 1; i < l; i++)
    {
        if (!((str[i] >= 'a' && str[i] <= 'z') || (str[i] >= 'A' && str[i] <= 'Z') || (str[i] >= '0'
&& str[i] <= '9') || str[i] == '_'))
        {
            cout << "Invalid identifier" << endl;
            return 0;
        }
    }

    // Check if string is a keyword
    set<string> keywords = {"auto", "break", "case", "char", "const", "continue",
"default", "do", "double", "else", "enum", "extern", "float", "for", "goto", "if", "int",
"long", "register", "return", "short", "signed", "sizeof", "static", "struct", "switch",
"typedef", "union", "unsigned", "void", "volatile", "while"};
    if (keywords.find(str) != keywords.end())
    {
        cout << "Keyword" << endl;
    }
    else
    {
        cout << "Valid identifier" << endl;
    }
    return 0;
}
```