```
/*
 *  LanguageDescriptor.h
 *  Defines Helper functions for the application
 *
 *
 *  Created: 2/7/2017 by Ryan Tedeschi
 */

#ifndef HELPERS_H
#define HELPERS_H

#include <string>
#include <vector>
#include <unordered_map>
#include <list>
#include <algorithm>

using namespace std;

struct arg {
    arg(string id, string value) {
        this->id = id;
        this->value = value;
    };
    string id;
    string value;
};

namespace Helpers {
    string ReadFile(string);
    string DupStr(string, int);

    vector<string> ParseArrayArgument(string, vector<arg>);
    string ParseArgument(string, vector<arg>);

    string toLower(string);
    string toUpper(string);

    template<typename T>
    vector<T> concat(vector<T> source, vector<T> addition) {
        source.insert(source.end(), addition.begin(), addition.end());
        return source;
    };
};

template<class T>
class State {
    public:
        State(string id) {
            this->id = id;
        };
        void SetGoal(string token) {
            this->acceptingToken = token;
            isFinal = true;
        };
        void UnsetGoal() {
            this->acceptingToken;
            isFinal = false;
        };
        void AddTransition(State<T>* target, vector<T> input) {
            this->transitionInputs.push_back(input);
            this->transitionStates.push_back(target);
            for (int i = 0; i < input.size(); i++) {
                transitions[input[i]] = target;
            }
        };
```

```
67          State<T>* Transition(T input) {
68              return transitions[input];
69          };
70          string GetId() {
71              return id;
72          };
73          string GetToken() {
74              return acceptingToken;
75          };
76          bool IsGoal() {
77              return isFinal;
78          };
79          void Print() {
80              cout << "State '" << id << "'";
81              if (isFinal)
82                  cout << " (GOAL - '" << acceptingToken << "')";
83              for (int i = 0; i < transitionStates.size(); i++) {
84                  cout << "\n\tTransitions to state '" << transitionStates[i]->GetId() << "' with inputs ";
85                  for (int j = 0; j < transitionInputs[i].size(); j++) {
86                      if (j > 0)
87                          cout << ", ";
88                      cout << transitionInputs[i][j];
89                  }
90              }
91              cout << endl;
92          };
93
94      private:
95          unordered_map<T, State<T>*> transitions;
96          vector<State<T>*> transitionStates;
97          vector<vector<T>> transitionInputs;
98          string id = "";
99          string acceptingToken = "";
100         bool isFinal = false;
101 };
102
103 template<class T>
104 class FSM {
105     public:
106         FSM() {};
107
108         State<T>* AddState(string id) {
109             if (!HasState(id)) {
110                 State<T>* newState = new State<T>(id);
111                 states[id] = newState;
112                 return newState;
113             }
114             return NULL;
115         };
116         void AddTransition(string start, string target, vector<T> transitionInput) {
117             State<T>* Start = GetState(start);
118             State<T>* Target = GetState(target);
119             if (Start != NULL && Target != NULL) {
120                 Start->AddTransition(Target, transitionInput);
121             }
122         };
123         void SetInitialState(string id) {
124             initialState = GetState(id);
125         };
126         void AddGoal(string id, string token) {
127             State<T>* state = GetState(id);
128             if (state != NULL) {
129                 state->SetGoal(token);
130             }
131         };
132         void RemoveGoal(string id) {
133             State<T>* state = GetState(id);
```

```
134             if (state != NULL) {
135                 state->UnsetGoal();
136             }
137         };
138         bool HasState(string id) {
139             return GetState(id) != NULL;
140         };
141         State<T>* GetState(string id) {
142             return states[id];
143         };
144         string Transition(T input) {
145             string ret = "";
146             if (currentState != NULL) {
147                 State<T>* nextState = currentState->Transition(input);
148                 if (nextState != NULL) {
149                     currentState = nextState;
150                 } else {
151                     if (currentState->IsGoal()) {
152                         ret = currentState->GetToken();
153                         Reset();
154                     } else
155                         ret = "ERROR";
156                 }
157             } else {
158                 ret = "ERROR";
159                 Reset();
160             }
161             return ret;
162         };
163         State<T>* CurrentState() {
164             return currentState;
165         };
166         void Reset() {
167             currentState = initialState;
168         };
169 
170         void Print() {
171             cout << "---- FINITE STATE MACHINE ----\n";
172             cout << "Initial State: " << initialState->GetId() << endl;
173 
174             for ( auto it = states.begin(); it != states.end(); ++it )
175                 it->second->Print();
176         };
177 
178     private:
179         unordered_map<string, State<T>*> states;
180         State<T>* initialState = NULL;
181         State<T>* currentState = NULL;
182 };
183 
184 #endif
```