```
1 /*
      LanguageDescriptor.h
      Defines the Language Descriptor class, which is the bridge between a text language descriptor file
 4 *
 5 *
 6 *
      Created: 1/3/2017 by Ryan Tedeschi
 7 */
 9 #ifndef LANGUAGE DESCRIPTOR H
10 #define LANGUAGE DESCRIPTOR H
11
12 #include <vector>
13 #include <string>
14 #include <sstream>
15 #include <regex>
16 #include <iostream>
17 #include <unordered map>
18 #include "../Markup/Markup.h"
19 #include "../Helpers/Helpers.h"
20
21 #define CFG EXT ".cfg"
22 #define CFG DIR "./cfg/"
23
24 using namespace std;
25
26 enum ProductionSetType { _Root, _Terminal, _Group, _Alternation, _Production, _Action };
27
28 class Production;
29 class ProductionSet;
30 class LanguageDescriptorObject;
31 class TokenMatch;
32
33 class ActionRoutine {
34
       public:
35
           virtual Markup* Execute(Markup*, vector<Markup*>) = 0;
36 };
37
38 class DeclareVarAction : public ActionRoutine {
39
40
           Markup* Execute(Markup*, vector<Markup*>);
41 };
42 class AssignVarAction : public ActionRoutine {
43
44
           Markup* Execute(Markup*, vector<Markup*>);
45 };
46 class AccumulateVarAction : public ActionRoutine {
47
48
           Markup* Execute(Markup*, vector<Markup*>);
49 };
50 class ResolveExprAction : public ActionRoutine {
51
52
           Markup* Execute(Markup*, vector<Markup*>);
53
54
55
           Markup* ResolveExpr(Markup*);
56 };
57
58 class ActionRoutines {
59
60
           static Markup* ExecuteAction(string, Markup*);
61
           static Markup* ExecuteAction(string, Markup*, vector<Markup*>);
62
63
      private:
64
           static vector<Markup*> ResolveParameters(string, Markup*);
65
           static Markup* ResolveParameter(string, Markup*);
```

```
67
            static unordered_map<string, ActionRoutine*> actions;
 68 };
 70 class Token : public Printable {
 71
            Token(string, string);
 72
 73
            string id;
 74
            string value;
 75
 76
            void Print();
 77
 78
        private:
 79
 80 };
 81
 82 class LanguageDescriptorObject
 83 {
 84
        public:
 85
            LanguageDescriptorObject(string);
 86
            LanguageDescriptorObject();
 87
            ~LanguageDescriptorObject();
 88
 89
            vector<Token> Tokenize(string);
 90
            vector<Token> Tokenize(Markup*);
 91
 92
            void Parse(string);
 93
 94
            string LookupTerminalValue(string);
 95
            bool IsTerminalIgnored(string);
 96
            Production* findProdById(string);
 97
            int getProdIndex(string);
 98
            vector<Production*> GetOrderedProductions(vector<string>);
 99
            vector<Production*> GetProductions();
100
101
            string GetLanguage();
102
103
104
        private:
105
            void ParseTerminalValues(string);
106
            void ParseFSM(string);
107
            void ParseReservedWords(string);
108
            void ParseIgnores(string);
109
110
            unordered map<string, bool> ignore;
111
            unordered map<string, string> terminals;
112
            unordered_map<string, string> reservedWords;
113
            vector<Production*> productions;
114
            FSM<char> stateMachine;
115
            string language;
116 };
117
118 class TokenMatch {
119
       public:
120
            bool isAction = false;
121
            string prod;
122
            int begin;
123
            int end;
124
            int length;
125
            vector<Token> match;
126
            vector<TokenMatch*> submatches;
127
128
            Markup* GenerateMarkup(Markup* parent = NULL, bool addChildrenToParent = false);
129
            void Print(int);
130
        private:
131
132
133 };
```

```
134
135 class ProductionSet {
136
       public:
137
            ProductionSet(Production*);
138
            void Parse(string);
139
            void SetAction(string);
140
            void SetTerminal(string);
141
            void SetProduction(string);
142
            void SetAlternation(string);
143
            void SetMultiplicity(string);
144
145
            TokenMatch* MatchStrict(vector<Token>, int);
146
            TokenMatch* Match(vector<Token>, int);
147
            TokenMatch* Match(vector<Token>);
148
149
            Production* GetProduction();
150
151
            ProductionSetType GetType();
152
            vector<ProductionSet*> GetChildren();
153
            string GetSource();
154
            string GetMultiplicity();
155
156
            // Markup Parser(vector<string>);
157
158
        private:
159
            TokenMatch* MatchGroup(vector<Token>, int);
160
            TokenMatch* MatchTerminal(vector<Token>, int):
161
            TokenMatch* MatchAlternation(vector<Token>, int);
162
            TokenMatch* MatchProduction(vector<Token>, int);
163
            TokenMatch* MatchAction(string, int);
164
165
            Production* prod;
166
            enum ProductionSetType type = _Root;
167
            string source = "";
168
            vector<ProductionSet*> children;
169
            string multiplicity = "";
170 };
171
172 class Production {
       public:
173
174
            Production(LanguageDescriptorObject*, string, string);
175
            void Parse(string, string);
176
177
            LanguageDescriptorObject* GetLDO();
            ProductionSet* GetRootProductionSet();
178
179
            TokenMatch* Match(vector<Token>);
180
            TokenMatch* Match(vector<Token>, int);
181
            TokenMatch* MatchStrict(vector<Token>);
182
            TokenMatch* MatchStrict(vector<Token>, int);
183
184
            string GetRegex();
185
            string GetId();
186
            vector<Production*> GetContainedProductions();
187
188
        private:
189
            LanguageDescriptorObject* ldo;
190
            string id;
191
            string data;
192
            vector<string> subproductions;
193
            ProductionSet* rootSet;
194 };
195
196 #endif
```