```
 1 /*
 2  *  AnalyzeModule.h
 3  *
 4  *
 5  *  Created: 3/24/2017 by Ryan Tedeschi
 6  */
 7
 8 #ifndef ANALYZEMODULE_H
 9 #define ANALYZEMODULE_H
10
11 #include <string>
12 #include <vector>
13 #include <iostream>
14 #include "../../shared/CASP_Plugin/CASP_Plugin.h"
15
16 using namespace std;
17
18 enum NodeType { Constant, Exponential, Logarithmic, Undefined };
19
20 class AnalysisNode {
21     public:
22         AnalysisNode();
23
24         void SetToUndefined();
25         void SetToConstant();
26         void SetToExponential(int exponent);
27         void SetToLogarithmic(int base, int exponent);
28
29         string ToString();
30
31         int exponent = 1;
32         int base = 1;
33         NodeType type = Undefined;
34
35         friend bool operator==(const AnalysisNode&, const AnalysisNode&);
36         friend bool operator!=(const AnalysisNode&, const AnalysisNode&);
37         friend bool operator>(const AnalysisNode&, const AnalysisNode&);
38         friend bool operator>=(const AnalysisNode&, const AnalysisNode&);
39         friend bool operator<(const AnalysisNode&, const AnalysisNode&);
40         friend bool operator<=(const AnalysisNode&, const AnalysisNode&);
41         AnalysisNode& operator*(AnalysisNode&);
42         AnalysisNode& operator=(AnalysisNode&);
43         AnalysisNode* operator=(AnalysisNode*);
44
45     private:
46
47 };
48
49 class Analysis {
50
51     public:
52         Analysis();
53         void AddFactor(AnalysisNode*);
54         void AddConstantFactor();
55         void AddExponentialFactor(int);
56         void AddLogarithmicFactor(int, int);
57         string ToString();
58
59         friend bool operator==(const Analysis&, const Analysis&);
60         friend bool operator!=(const Analysis&, const Analysis&);
61         friend bool operator>(const Analysis&, const Analysis&);
62         friend bool operator>=(const Analysis&, const Analysis&);
63         friend bool operator<(const Analysis&, const Analysis&);
64         friend bool operator<=(const Analysis&, const Analysis&);
65         Analysis& operator*(Analysis&);
66
```

```
67          bool IsUndefined();
68
69      private:
70          AnalysisNode* constant = NULL;
71          AnalysisNode* exponential = NULL;
72          AnalysisNode* logarithmic = NULL;
73
74          bool undefined = false;
75  };
76
77  class AnalysisTree {
78      public:
79          AnalysisTree();
80
81          void AddChild(AnalysisTree*);
82          void AddFactor(AnalysisNode*);
83          void AddConstantFactor();
84          void AddExponentialFactor(int);
85          void AddLogarithmicFactor(int, int);
86          void SetAnalysis(Analysis*);
87
88          Analysis* GetAnalysis();
89
90      private:
91
92          vector<AnalysisTree*> children;
93          Analysis* analysis = NULL;
94
95  };
96
97  class AnalyzeModule : public CASP_Plugin {
98      public:
99          AnalyzeModule();
100
101         virtual CASP_Return* Execute(Markup* markup, LanguageDescriptorObject* source_ldo, vector<arg> fnArgs, CASP_Return* inputReturn = NULL);
102
103     private:
104         void GetAllAnalyses(Markup*);
105         Analysis* GetRootAnalysis(vector<Markup*>);
106         Analysis* GetFunctionAnalysis(Markup*);
107
108         void analyzeProcess(Markup*, AnalysisTree*);
109         void analyzeMethodCall(Markup*, AnalysisTree*);
110         void analyzeDecision(Markup*, AnalysisTree*);
111         void analyzeLoop(Markup*, AnalysisTree*);
112         void processStatement(Markup*, AnalysisTree*);
113         void processBlock(Markup*, AnalysisTree*);
114
115         unordered_map<string, Analysis*> functionTable;
116         unordered_map<string, Markup*> markupTable;
117
118 };
119
120 #endif
```