

```
1 /*
2  * CASP_Return.h
3  * Defines a generic return object for the program
4  *
5  * Created: 4/4/2017 by Ryan Tedeschi
6  */
7
8 #ifndef CASP_RETURN_H
9 #define CASP_RETURN_H
10
11 #include <iostream>
12 #include <algorithm>
13 #include <string>
14 #include <unordered_map>
15 #include <vector>
16 #include "../Printable/Printable.h"
17
18 using namespace std;
19
20 class GenericData : public Printable {
21 public:
22     string GetType() {
23         return type;
24     };
25     virtual void Print();
26
27 private:
28     string type = "";
29 };
30
31 template<typename T>
32 class GenericLeaf : public GenericData {
33 public:
34     GenericLeaf(T data) {
35         this->data = data;
36     };
37
38     virtual void Print() {
39         try {
40             cout << specialLookups.at(data);
41         } catch (...) {
42             cout << data;
43         }
44     };
45
46     void Assign(T data) {
47         this->data = data;
48     };
49
50     void AddSpecial(T input, string output) {
51         specialLookups[input] = output;
52     };
53
54 protected:
55     string type = "Leaf";
56     T data;
57     unordered_map<T, string> specialLookups;
58 };
59
60 template<typename T>
61 static inline
62 GenericLeaf<T>* CreateLeaf(T data) {
63     GenericLeaf<T>* leaf = new GenericLeaf<T>(data);
64     return leaf;
65 };
66 template<
```

```

67 static inline
68 GenericLeaf<bool>* CreateLeaf(bool data) {
69     GenericLeaf<bool>* leaf = new GenericLeaf<bool>(data);
70     leaf->AddSpecial(true, "true");
71     leaf->AddSpecial(false, "false");
72     return leaf;
73 };
74 template<>
75 static inline
76 GenericLeaf<string>* CreateLeaf<string>(string data) {
77
78     int index = -1;
79     while ((index = data.find("\\", index + 1)) != -1) {
80         data = data.substr(0, index) + "\\" + data.substr(index, data.size());
81
82         index++;
83     }
84
85     GenericLeaf<string>* leaf = new GenericLeaf<string>("\\\" + data + "\\");
86     return leaf;
87 };
88
89 class GenericObject : public GenericData {
90     public:
91         GenericObject();
92         GenericObject(unordered_map<string, GenericData*>);
93         virtual void Print();
94         void Add(string, GenericData*);
95         GenericData* At(string);
96
97     protected:
98         string type = "Object";
99         unordered_map<string, GenericData*> data;
100
101 };
102 GenericObject* CreateObject();
103 GenericObject* CreateObject(unordered_map<string, GenericData*>);
104
105 class GenericArray : public GenericData {
106     public:
107         GenericArray();
108         GenericArray(vector<GenericData*>);
109         virtual void Print();
110         void Add(GenericData*);
111         GenericData* At(int);
112
113     protected:
114         string type = "Array";
115         vector<GenericData*> data;
116 };
117
118 GenericArray* CreateArray();
119 GenericArray* CreateArray(vector<GenericData*>);
120
121 class CASP_Return : public GenericObject {
122     public:
123         CASP_Return();
124
125         GenericArray* Errors();
126         GenericArray* Warnings();
127         GenericObject* Data();
128
129         void AddStandardWarning(string, int = -1);
130         void AddStandardError(string, int = -1);
131
132     private:
133 };

```

```
134  
135  
136 #endif
```