






# cs380su21-meta.sg

[Dashboard](#) / [My courses](#) / [cs380su21-meta.sg](#) / [10 May - 16 May](#) / [Assignment 1 \(Tree\)](#)

-  [Description](#)
-  [Submission](#)
-  [Edit](#)
-  Submission view

## Grade

Reviewed on Thursday, 20 May 2021, 3:44 AM by Automatic grade  
**grade:** 100.00 / 100.00

**Assessment report**  [-]  
[\[+\]](#) **Summary of tests**

Submitted on Thursday, 20 May 2021, 3:44 AM ([Download](#))

functions.cpp

```
1 1 /*!*****  
2 2 \file functions.cpp  
3 3 \author Vadim Surov, Goh Wei Zhe  
4 4 \par DP email: vsurov\@digipen.edu, weizhe.goh\@digipen.edu  
5 5 \par Course: CS380  
6 6 \par Section: A  
7 7 \par Programming Assignment 1  
8 8 \date 05-19-2021  
9 9 \brief  
10 10 This file has declarations and definitions that are required for submission  
11 11 *****  
12 12  
13 13 #include "functions.h"  
14 14  
15 15 namespace AI  
16 16 {  
17 17     /*!*****  
18 18     \brief  
19 19     Function to convert a string to integer.  
20 20  
21 21     \param str  
22 22     The string passed in to be converted into integer.  
23 23  
24 24     \return  
25 25     Return the string as integer.  
26 26     *****  
27 27     int stringToInt(std::string str)  
28 28     {  
29 29         const char* stringToInt = &str[1];  
30 30         int numChild = std::atoi(stringToInt);  
31 31  
32 32         return numChild;  
33 33     }  
34 34 }  
35 35
```

functions.h

```

1  /*!*****
2  \file functions.h
3  \author Vadim Surov, Goh Wei Zhe
4  \par DP email: vsurov@digipen.edu, weizhe.goh@digipen.edu
5  \par Course: CS380
6  \par Section: A
7  \par Programming Assignment 1
8  \date 05-19-2021
9  \brief
10 This file has declarations and definitions that are required for submission
11 *****/
12
13 #ifndef FUNCTIONS_H
14 #define FUNCTIONS_H
15
16 #include <iostream>
17 #include <sstream>
18 #include <string>
19 #include <list>
20 #include <vector>
21 #include <queue>
22 #include <stack>
23 #include <algorithm>
24
25 #include "data.h"
26
27 namespace AI
28 {
29     #define UNUSED(expr) (void)expr;
30
31     // A simple graph node definition with serialization functions
32
33     //Helper function
34     int stringToInt(std::string str);
35
36     template<typename T>
37     struct Node
38     {
39         // Member data
40
41         T value;
42         Node* parent;
43         std::list<Node*> children;
44
45         Node(T value = {}, Node* parent = nullptr,
46             const std::list<Node*>& children = {})
47             : value{ value }, parent{ parent }, children{ children }{}
48
49         ~Node()
50         {
51             for (auto child : children)
52                 delete child;
53         }
54
55         /*!*****
56         \brief
57         Serialization. An overloading insertion operator function that takes
58         and return a stream object.
59
60         \param os
61         Output stream to perform output.
62
63         \param rhs
64         Right hand side object.
65
66         \return
67         Returns the output through ostream.
68         *****/
69         friend std::ostream& operator<<(std::ostream& os, Node const& rhs)
70         {
71             //Recursive function
72             PrintOutput(os, &rhs);
73
74             return os;
75         }
76
77         /*!*****
78         \brief
79         Recursive Function to print output.
80
81         \param os
82         Output stream to perform output.
83
84         \param rhs
85         Right hand side object.
86
87         \return
88         Returns the output through ostream.
89         *****/
90         static void PrintOutput(std::ostream& os, const Node* rhs)
91         {
92             //std::cout << "os << rhs->value: " << rhs->value << std::endl;
93
94             os << rhs->value + " {" + std::to_string(rhs->children.size()) + " ";
95
96             //loop through each node in children's list
97             for (Node* n : rhs->children)
98             {
99                 PrintOutput(os, n);
100             }
101
102             os << "} ";
103         }
104
105         /*!*****
106         \brief
107         Deserialization function to handle input streams and return an istream
108         object.

```

```

109
110 \param is
111 Input stream to read inputs.
112
113 \param rhs
114 Right hand side object.
115
116 \return
117 Returns the input through istream.
118 *****/
119 friend std::istream& operator>>(std::istream& is, Node& rhs)
120 {
121     is >> rhs.value;
122     //std::cout << "is >> rhs.value: " << rhs.value << std::endl;
123
124     //Recursive function
125     ReadInput(is, &rhs);
126     return is;
127 }
128
129 /*!*****
130 \brief
131 Recursive Function to read input.
132
133 \param is
134 Input stream to read inputs.
135
136 \param rhs
137 Right hand side object.
138
139 \return
140 None.
141 *****/
142 static void ReadInput(std::istream& is, Node* rhs)
143 {
144     std::string s;
145
146     while (is >> s)
147     {
148         //std::cout << "is >> str: " << s << std::endl;
149
150         //If found {
151         if (s.find("{") != std::string::npos)
152         {
153             //convert str[1] to integer and store as no. of child
154             int numChild = stringToInt(s);
155
156             //std::cout << "no. of child: " << numChild << std::endl;
157
158             //For each children, check if children has a child
159             for (int i = 0; i < numChild; ++i)
160             {
161                 Node* child = new Node;
162
163                 is >> s;
164
165                 child->parent = rhs;
166                 child->value = s;
167
168                 // std::cout << "child value: " << child->value
169                 //<< std::endl << std::endl;
170
171                 rhs->children.push_back(child);
172
173                 ReadInput(is, child);
174             }
175         }
176         else if (s.find("}") != std::string::npos)
177         {
178             return;
179         }
180     }
181 }
182
183
184 /*!*****
185 \brief
186 Function to get path from tree root to current node
187
188 \return
189 Returns values from root to this node as an array.
190 *****/
191 std::vector<T> getPath() const
192 {
193     std::vector<T> r;
194
195     r.push_back(this->value);
196     Node* node = this->parent;
197
198     while (node)
199     {
200         r.push_back(node->value);
201         node = node->parent;
202     }
203
204     std::reverse(r.begin(), r.end());
205
206     return r;
207 }
208 };
209
210
211 /*!*****
212 \brief
213 Implementation of the Breadth-First Search algorithm
214
215 \param node
216 The node to search from.

```

```
217
218 \param lookingfor
219 The value of node we looking for.
220
221 \return
222 Returns the node found.
223 *****/
224 template<typename T>
225 Node<T>* BFS(Node<T> & node, const T & lookingfor)
226 {
227     std::queue<Node<T>*> Q;
228     Q.push(&node);
229
230     while (!Q.empty())
231     {
232         Node<T>* current = Q.front();
233         Q.pop();
234
235         if (current->value == lookingfor)
236             return current;
237
238         //loop through each node in children's list
239         for (Node<T>* n : current->children)
240         {
241             Q.push(n);
242         }
243     }
244
245     return nullptr;
246 }
247
248 /*!*****
249 \brief
250 Implementation of the Depth-First Search algorithm
251
252 \param node
253 The node to search for.
254
255 \param lookingfor
256 The value of the node we looking for.
257
258 \return
259 Returns the node found.
260 *****/
261 template<typename T>
262 Node<T>* DFS(Node<T> & node, const T & lookingfor)
263 {
264     std::stack <Node<T>*> Stack;
265     Stack.push(&node);
266
267     while (!Stack.empty())
268     {
269         Node<T>* current = Stack.top();
270         Stack.pop();
271
272         if (current->value == lookingfor)
273             return current;
274
275         //loop through each node in children's list
276         for (Node<T>* n : current->children)
277             Stack.push(n);
278     }
279
280     return nullptr;
281 }
282
283 } // end namespace
284
285 #endif
```