## cs380su21-meta.sg

<u>Dashboard</u> / My courses / <u>cs380su21-meta.sg</u> / <u>10 May - 16 May</u> / <u>Assignment 1 (Tree)</u>

## 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

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

functions.h

```
\file functions.h
 3 \author Vadim Surov, Goh Wei Zhe
 4
     \par DP email: vsurov\@digipen.edu, weizhe.goh\@digipen.edu
     \par Course: CS380
     \par Section: A
  6
     \par Programming Assignment 1
  8
     \date 05-19-2021
 9
     \brief
    This file has declarations and definitions that are required for submission
 10
 11
 12
 13
     #ifndef FUNCTIONS_H
     #define FUNCTIONS_H
 14
 15
    #include <iostream>
 16
 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>
        struct Node
 37
 38 *
 39
            // Member data
 40
 41
            T value;
            Node* parent;
 42
 43
            std::list<Node*> children;
 44
 45
            Node(T value = {}, Node* parent = nullptr,
                      const std::list<Node*>& children = {})
 46
                : value{ value }, parent{ parent }, children{ children }{}
 47
 48
 49
            ~Node()
 50 -
            }
 51
                for (auto child : children)
                   delete child;
 52
 53
 54
            55 🔻
 56
            \brief
 57
            Serialization. An overloading insertion operator function that takes
            and return a stream object.
 58
 59
 60
            \param os
 61
            Output stream to perform output.
 62
 63
            \param rhs
            Right hand side object.
 64
 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
            Output stream to perform output.
 82
 84
            \param rhs
            Right hand side object.
 85
 86
 87
            \return
 88
            Returns the output through ostream.
 89
            *******************************
            static void PrintOutput(std::ostream& os, const Node* rhs)
 90
 91 -
               //std::cout << "os << rhs->value: " << rhs->value << std::endl;</pre>
 92
 93
               os << rhs->value + " {"+std::to_string(rhs->children.size()) + " ";
 94
 95
               //loop through each node in children's list
 96
 97
               for (Node* n : rhs->children)
 98 -
 99
                  PrintOutput(os, n);
100
101
               os << "} ";
102
103
104
105 -
            106
            \brief
107
            Deserialization function to handle input streams and return an istream
108
            object.
```

```
109
            \param is
110
            Input stream to read inputs.
111
112
113
            \param rhs
            Right hand side object.
114
115
            \return
116
117
            Returns the input through istream.
            118
            friend std::istream& operator>>(std::istream& is, Node& rhs)
119
120 🔻
               is >> rhs.value;
121
               //std::cout << "is >> rhs.value: " << rhs.value << std::endl;</pre>
122
123
124
               //Recursive function
125
               ReadInput(is, &rhs);
126
               return is;
127
128
129 🔻
            130
            \brief
            Recursive Function to read input.
131
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
            static void ReadInput(std::istream& is, Node* rhs)
142
143 🔻
144
               std::string s;
145
               while (is >> s)
146
147 🕶
                   //std::cout << "is >> str: " << s << std::endl;
148
149
                   //If found {
150 -
                   if (s.find("{") != std::string::npos)
151
152 🕶
                      //convert str[1] to integer and store as no. of child
153
                      int numChild = stringToInt(s);
154
155
                      //std::cout << "no. of child: " << numChild << std::endl;</pre>
156
157
                      //For each children, check if children has a child
158
                      for (int i = 0; i < numChild; ++i)</pre>
159
160
                         Node* child = new Node;
161
162
163
                          is >> s;
164
                          child->parent = rhs;
165
                          child->value = s;
166
167
                          // std::cout << "child value: " << child->value
168
                         //<< std::endl << std::endl;</pre>
169
170
                         rhs->children.push_back(child);
171
172
173
                          ReadInput(is, child);
174
175
                   }
                   else if (s.find("}") != std::string::npos)
176
177 -
178
                      return;
179
180
181
182
183
            184 🔻
185
            \brief
186
            Function to get path from tree root to current node
187
188
            Returns values from root to this node as an array.
189
            *******************************
190
191
               ::vector<T> getPath() const
192 🔻
193
              std::vector<T> r;
194
              r.push_back(this->value);
195
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
213
        Implementation of the Breadth-First Search algorithm
214
215
        \param node
216
        The node to search from.
```

```
cs380su21-meta.sg Assignment 1 (Tree) Submission view
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>
         Node<T>* BFS(Node<T> & node, const T & lookingfor)
225
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
                if (current->value == lookingfor)
235
                    return current;
236
237
238
                //loop through each node in children's list
                for (Node<T>* n : current->children)
239
240 -
                    Q.push(n);
241
242
243
244
245
             return nullptr;
246
247
         248 🔻
         \brief
249
250
         Implementation of the Depth-First Search algorithm
251
252
         \param node
         The node to search for.
253
254
         \param lookingfor
255
256
         The value of the node we looking for.
257
258
         Returns the node found.
259
                          **********************
260
         template<typename T>
261
262
         Node<T>* DFS(Node<T> & node, const T & lookingfor)
263 🕶
264
             std::stack <Node<T>*> Stack;
             Stack.push(&node);
265
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
               //loop through each node in children's list
275
276
                for (Node<T>* n : current->children)
277
                   Stack.push(n);
278
279
280
             return nullptr;
281
282
     } // end namespace
283
284
    #endif
285
                                                                                                                                VPL
Slides
                                                                                                                 Slides ►
                                          Jump to...
```

You are logged in as Wei Zhe GOH (Log out) cs380su21-meta.sg <u>Data retention summary</u>

Get the mobile app