

[◀ Return to Classroom](#)

# Memory Management Chatbot

REVIEW

CODE REVIEW 7

HISTORY

▼ src/chatlogic.cpp 5

```
1 #include <fstream>
2 #include <sstream>
3 #include <iostream>
4 #include <vector>
5 #include <iterator>
6 #include <tuple>
7 #include <algorithm>
8
9 #include "graphedge.h"
10 #include "graphnode.h"
11 #include "chatbot.h"
12 #include "chatlogic.h"
13
14 #include <memory>
15 using std::unique_ptr;
16 using std::make_unique;
17 using std::move;
18
19 ChatLogic::ChatLogic()
20 {
21     //// STUDENT CODE : Task 5 , ChatLogic does not have ChatBot instances
22     ////
23
24     // create instance of chatbot
25     // _chatBot = new ChatBot("../images/chatbot.png");
26
27     // add pointer to chatlogic so that chatbot answers can be passed on to the GUI
28     // _chatBot->SetChatLogicHandle(this);
29
30
31     ////
32     //// EOF STUDENT CODE
33 }
34
35 ChatLogic::~ChatLogic()
36 {
37     //// STUDENT CODE : Task 3
38     ////
```

```

39
40 // delete chatbot instance
41 // delete _chatBot; // : Task 5 , ChatLogic does not have ChatBot instances
42
43 // For Task 3
44 // delete all nodes : if _nodes is unique_ptr, this "delete" are not needed. koba
45 // for (auto it = std::begin(_nodes); it != std::end(_nodes); ++it)
46 // {
47 //     delete *it;
48 // }
49
50 // For Task 4
51 // delete all edges
52 // for (auto it = std::begin(_edges); it != std::end(_edges); ++it)
53 // {
54 //     delete *it;
55 // }
56
57 ////
58 //// EOF STUDENT CODE : Task 3

```

AWESOME

Since ChatLogic should have no ownership relation to the ChatBot instance and thus is no longer responsible for me removing all unnecessary allocations and deallocations.

```

59 }
60
61 template <typename T>
62 void ChatLogic::AddAllTokensToElement(std::string tokenID, tokenlist &tokens, T &element)
63 {
64     // find all occurrences for current node
65     auto token = tokens.begin();
66     while (true)
67     {
68         token = std::find_if(token, tokens.end(), [&tokenID](const std::pair<std::string, std::string> &p) {
69             if (token != tokens.end())
70             {
71                 element.AddToken(token->second); // add new keyword to edge
72                 token++; // increment iterator to next element
73             }
74             else
75             {
76                 break; // quit infinite while-loop
77             }
78         });
79     }
80
81 void ChatLogic::LoadAnswerGraphFromFile(std::string filename)
82 {
83     // load file with answer graph elements
84     std::ifstream file(filename);
85
86     // check for file availability and process it line by line
87     if (file)
88     {
89         // loop over all lines in the file
90         std::string lineStr;
91         while (getline(file, lineStr))
92         {
93             // extract all tokens from current line
94             tokenlist tokens;
95             while (lineStr.size() > 0)
96             {
97                 // extract next token
98                 int posTokenFront = lineStr.find("<");
99                 int posTokenBack = lineStr.find(">");
100                 if (posTokenFront < 0 || posTokenBack < 0)
101                     break; // quit loop if no complete token has been found
102                 std::string tokenStr = lineStr.substr(posTokenFront + 1, posTokenBack - 1);

```

```

102
103 // extract token type and info
104 int posTokenInfo = tokenStr.find(":");
105 if (posTokenInfo != std::string::npos)
106 {
107     std::string tokenType = tokenStr.substr(0, posTokenInfo);
108     std::string tokenInfo = tokenStr.substr(posTokenInfo + 1, tokenStr.size() - 1);
109
110     // add token to vector
111     tokens.push_back(std::make_pair(tokenType, tokenInfo));
112 }
113
114 // remove token from current line
115 lineStr = lineStr.substr(posTokenBack + 1, lineStr.size());
116 }
117
118 // process tokens for current line
119 auto type = std::find_if(tokens.begin(), tokens.end(), [](const std::pair<std::string, std::string>& token) {
120     if (token.first != tokens.end())
121     {
122         // check for id
123         auto idToken = std::find_if(tokens.begin(), tokens.end(), [](const std::pair<std::string, std::string>& token) {
124             if (token.first != tokens.end())
125             {
126                 // extract id from token
127                 int id = std::stoi(idToken->second);
128
129                 // node-based processing
130                 if (type->second == "NODE")
131                 {
132                     //// STUDENT CODE : Task 3
133                     ////
134
135                     // check if node with this ID exists already
136                     //auto newNode = std::find_if(_nodes.begin(), _nodes.end(), [&id] (GraphNode *node) {
137                     auto newNode = std::find_if(_nodes.begin(), _nodes.end(), [&id] (unique_ptr<GraphNode> &node) {

```

AWESOME

Nice job.

```

139
140 // create new element if ID does not yet exist
141 if (newNode == _nodes.end())
142 {
143     //_nodes.emplace_back(new GraphNode(id)); // original
144     _nodes.emplace_back(std::make_unique<GraphNode>(id));

```

AWESOME

Well done. `std::make_unique` has been used to implement exclusive ownership on the `_nodes` items. This ensures `usestd::move()`

```

145 //_nodes.emplace_back( unique_ptr<GraphNode>(new GraphNode(id)) );
146 newNode = _nodes.end() - 1; // get iterator to last element
147
148 // add all answers to current node
149 AddAllTokensToElement("ANSWER", tokens, **newNode);
150 }
151
152 ////
153 //// EOF STUDENT CODE : Task 3
154 }
155
156 // edge-based processing
157 if (type->second == "EDGE")
158 {

```

```

159         /// STUDENT CODE : Task 3
160         ///
161
162         // find tokens for incoming (parent) and outgoing (child) node
163         auto parentToken = std::find_if(tokens.begin(), tokens.end(), [](const std::pair<int, int>& token) {
164             return token.first == id;
165         });
166         auto childToken = std::find_if(tokens.begin(), tokens.end(), [](const std::pair<int, int>& token) {
167             return token.second == id;
168         });
169
170         if (parentToken != tokens.end() && childToken != tokens.end())
171         {
172             // get iterator on incoming and outgoing node via ID search
173             auto parentNode = std::find_if(_nodes.begin(), _nodes.end(), [&parentToken](const Node& node) {
174                 return node.token.first == parentToken;
175             });
176             auto childNode = std::find_if(_nodes.begin(), _nodes.end(), [&childToken](const Node& node) {
177                 return node.token.second == childToken;
178             });
179
180             // create new edge
181             GraphEdge *edge = new GraphEdge(id); // original
182             unique_ptr<GraphEdge> edge_ptr = make_unique<GraphEdge>(id);
183         }
184     }
185 }
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221

```

AWESOME

Awesome work. Modifying this makes sense since we now use `std::unique_ptr` which has a `get()` method that returns the raw pointer.

```

177         //edge->SetParentNode(*parentNode); // original
178         //edge->SetChildNode(*childNode); // original
179         edge->SetParentNode((*parentNode).get()); // my code
180         edge->SetChildNode((*childNode).get()); // my code
181
182
183
184         //_edges.push_back(edge); // original
185         _edges.push_back(edge); // my code
186
187
188         // find all keywords for current node
189         AddAllTokensToElement("KEYWORD", tokens, *edge);
190
191         // store reference in child node and parent node
192         (*childNode)->AddEdgeToParentNode(edge); // original
193         (*childNode)->AddEdgeToParentNode(edge.get());
194
195         (*parentNode)->AddEdgeToChildNode(edge); // original
196         (*parentNode)->AddEdgeToChildNode(move(edge)); // want to pass unique_ptr
197     }
198
199     ///
200     /// EOF STUDENT CODE : Task 3
201 }
202
203 else
204 {
205     std::cout << "Error: ID missing. Line is ignored!" << std::endl;
206 }
207
208 } // eof loop over all lines in the file
209
210 file.close();
211
212 } // eof check for file availability
213 else
214 {
215     std::cout << "File could not be opened!" << std::endl;
216     return;
217 }
218
219 /// STUDENT CODE
220 ///
221
222 // identify root node

```

```

222     GraphNode *rootNode = nullptr;
224     for (auto it = std::begin(_nodes); it != std::end(_nodes); ++it)
225     {
226         // search for nodes which have no incoming edges
227         if ((*it)->GetNumberOfParents() == 0)
228         {
229             if (rootNode == nullptr)
230             {
231                 // rootNode = *it;        // assign current node to root : original
232                 rootNode = it->get(); // assign current node to root : my code
233             }
234             else
235             {
236                 std::cout << "ERROR : Multiple root nodes detected" << std::endl;
237             }
238         }
239     }
240
241     // Task 5
242     // add chatbot to graph root node
243     //_chatBot->SetRootNode(rootNode);    // original
244     //rootNode->MoveChatbotHere(_chatBot); // original
245
246     // Task 5 : create instance of chatbot on the stack memory.
247     ChatBot localChatBot("../images/chatbot.png");

```

AWESOME

A new local chatBox instance is created as required.

```

248     localChatBot.SetChatLogicHandle(this);
249
250     // Task 5 : add chatbot to graph root node
251     localChatBot.SetRootNode(rootNode);
252     rootNode->MoveChatbotHere(move(localChatBot));
253
254     // add pointer to chatlogic so that chatbot answers can be passed on to the GUI
255     //_chatBot->SetChatLogicHandle(this);
256
257
258
259     ////
260     //// EOF STUDENT CODE
261 }
262
263 void ChatLogic::SetPanelDialogHandle(ChatBotPanelDialog *panelDialog)
264 {
265     _panelDialog = panelDialog;
266 }
267
268 void ChatLogic::SetChatbotHandle(ChatBot *chatbot)
269 {
270     _chatBot = chatbot;
271 }
272
273 void ChatLogic::SendMessageToChatbot(std::string message)
274 {
275     _chatBot->ReceiveMessageFromUser(message);
276 }
277
278 void ChatLogic::SendMessageToUser(std::string message)
279 {
280     _panelDialog->PrintChatbotResponse(message);
281 }
282
283 wxBitmap *ChatLogic::GetImageFromChatbot()
284 {
285     return _chatBot->GetImageHandle();
286 }
287

```

▸ src/graphnode.h 1

▸ src/chatbot.h 1

▸ src/graphnode.cpp

▸ src/graphedge.h

▸ src/graphedge.cpp

▸ src/chatlogic.h

▸ src/chatgui.h

▸ src/chatgui.cpp

▸ src/chatbot.cpp

▸ src/answergraph.txt

▸ CMakeLists.txt

RETURN TO PATH

Rate this review