Memory Management Chatbot

Meets Specifications

Dear Excellent Student,
Congratulations!!! You made it. 🙌🏼
By carefully going through the project, it shows a lot of effort, diligence and above all, determination. Excellent work! Your submission has passed all the rubric of this project. All the effort you put in to complete the project are very much appreciated and it was my pleasure reviewing your work. You should be proud of yourself because success is no accident. It is hard work, perseverance, learning, studying, sacrifice and most of all, love of what you are doing or learning to do. Remember, practice makes perfect. So, keep practicing on these projects and I wish you all the best.

Quality of Code

The code compiles and runs with `cmake` and `make`.

Awesome, your project compiles with `cmake` and `make`. Well Done!

More In-depth Knowledge

For more information on this build tool, refer to the following:

- cmake tutorial
- How to Build a CMake-Based Project

Task 1: Exclusive Ownership 1

In file `chatgui.h` / `chatgui.cpp`, `_chatLogic` is made an exclusive resource to class `ChatbotPanelDialog` using an

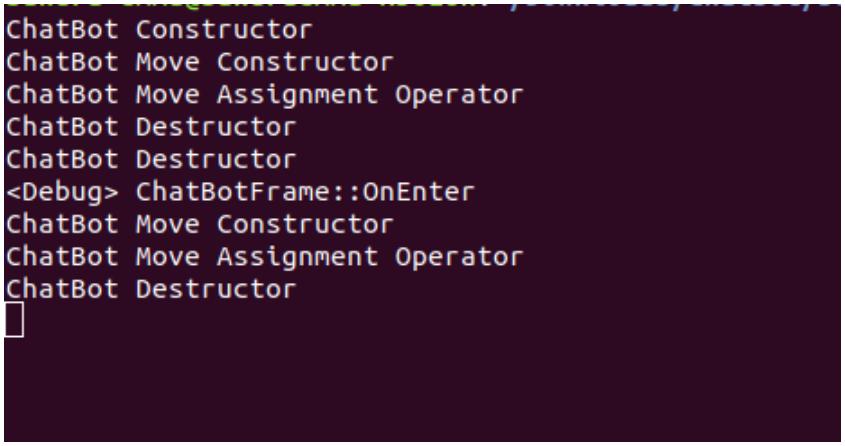
appropriate smart pointer. Where required, changes are made to the code such that data structures and function parameters reflect the new structure.

Well done! `_chatLogic` is constructed with a unique pointer in `ChatbotPanelDialog` on line 128 of `src/chatgui.cpp`.

Task 2: The Rule of Five

In file `chatbot.h` / `chatbot.cpp`, changes are made to the class `ChatBot` such that it complies with the Rule of Five. Memory resources are properly allocated / deallocated on the heap and member data is copied where it makes sense. In each of the methods (e.g. the copy constructor), a string of the type "ChatBot Copy Constructor" is printed to the console so that it is possible to see which method is called in later examples.

Excellent, rule 5 has been implemented correctly.



Task 3: Exclusive Ownership 2

In file `chatlogic.h` / `chatlogic.cpp`, the vector `_nodes` are adapted in a way that the instances of `GraphNodes` to which the vector elements refer are exclusively owned by the class `ChatLogic`. An appropriate type of smart pointer is used to achieve this.

Instances of `Graphnodes` are exclusively owned by the class `ChatLogic` using the correct smart pointer ie. the `unique_ptr`.

When passing the `GraphNode` instances to functions, ownership is not transferred.

Reference is correctly used to avoid ownership transfer of `unique_ptr` object, keep it up!

Task 4: Moving Smart Pointers

In files `chatlogic.h` / `chatlogic.cpp` and `graphnodes.h` / `graphnodes.cpp` all instances of `GraphEdge` are changed in a way such that each instance of `GraphNode` exclusively owns the outgoing `GraphEdges` and holds non-owning references to incoming `GraphEdges`. Appropriate smart pointers are used to do this. Where required, changes are made to the code such that data structures and function parameters reflect the changes.

made to the code such that data structures and function parameters reflect the changes.

Well done with the combinations of `std::unique_ptr` and raw pointer, `GraphNode` exclusively owns the outgoing `GraphEdges` and holds non-owning references to incoming `GraphEdges`.

In files `chatlogic.h` / `chatlogic.cpp` and `graphnodes.h` / `graphnodes.cpp`, move semantics are used when transferring ownership from class `ChatLogic`, where all instances of `GraphEdge` are created, into instances of `GraphNode`.

Nice work.

Task 5: Moving the ChatBot

In file `chatlogic.cpp`, a local `ChatBot` instance is created on the stack at the bottom of function `LoadAnswerGraphFromFile` and move semantics are used to pass the `ChatBot` instance into the root node.

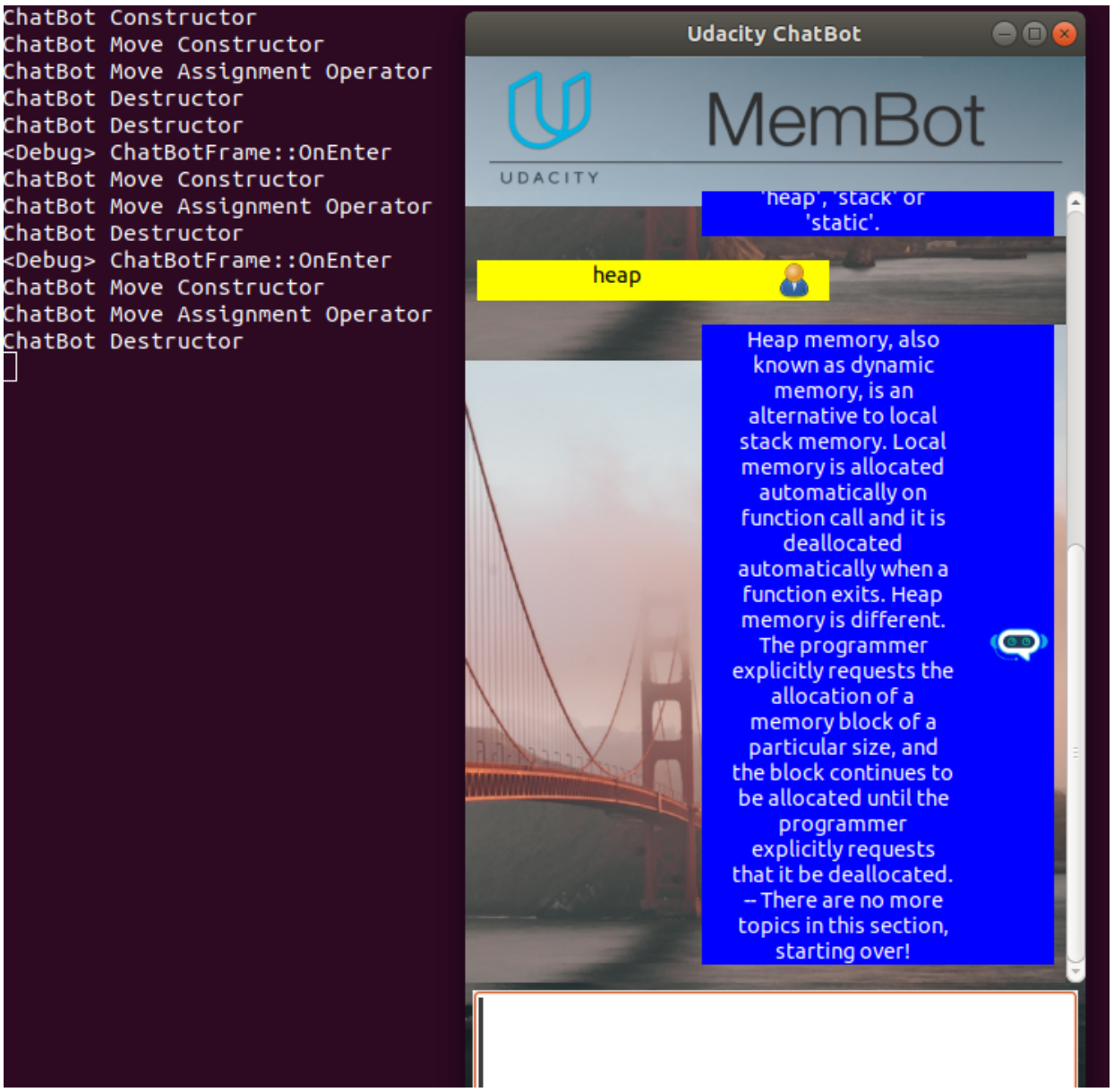
Well done here!

`ChatLogic` has no ownership relation to the `ChatBot` instance and thus is no longer responsible for memory allocation and deallocation.

`ChatLogic` now only holds a non-owning reference to `ChatBot` instance, and this reference is updated in `ChatBot` move semantic. That's neat!

When the program is executed, messages are printed to the console indicating which Rule of Five component of `ChatBot` is being called.

The calling sequence of Rule of five components is displayed as expected. Great work so far!



DOWNLOAD PROJECT

7 CODE REVIEW COMMENTS ›

RETURN TO PATH

Rate this review