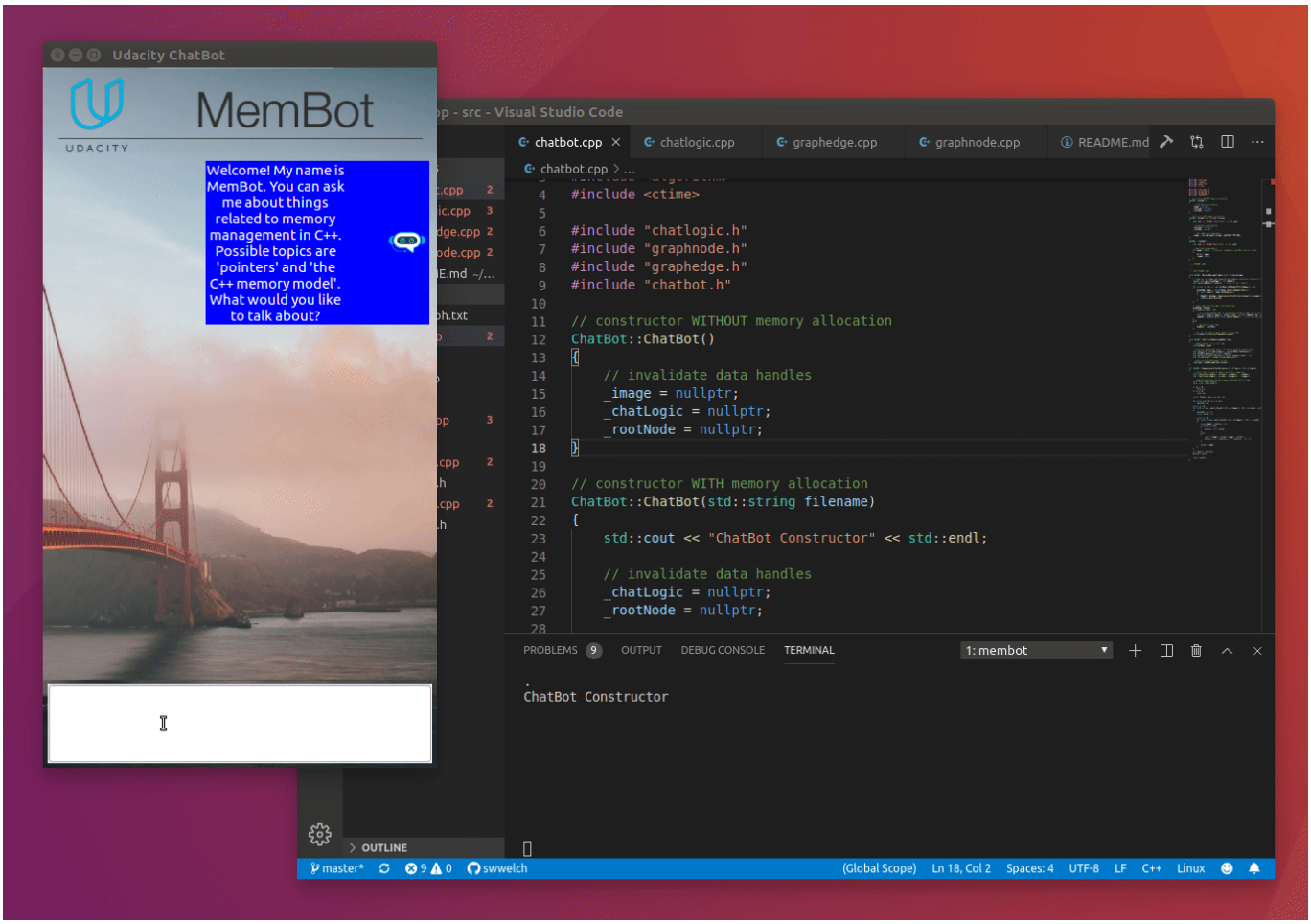
CPPND: Memory Management Chatbot

This is the project for the third course in the Udacity C++ Nanodegree Program: Memory Management.



The ChatBot code creates a dialogue where users can ask questions about some aspects of memory management in C++. After the knowledge base of the chatbot has been loaded from a text file, a knowledge graph representation is created in computer memory, where chatbot answers represent the graph nodes and user queries represent the graph edges. After a user query has been sent to the chatbot, the Levenshtein distance is used to identify the most probable answer. The code is fully functional as-is and uses raw pointers to represent the knowledge graph and interconnections between objects throughout the project.

In this project you will analyze and modify the program. Although the program can be executed and works as intended, no advanced concepts as discussed in this course have been used; there are currently no smart pointers, no move semantics and not much thought has been given to ownership or memory allocation.

Your goal is to use the course knowledge to optimize the ChatBot program from a memory management perspective. There are a total of five specific tasks to be completed, which are detailed below.

Dependencies for Running Locally

- cmake >= 3.11
  - All OSes: click here for installation instructions
- make >= 4.1 (Linux, Mac), 3.81 (Windows)

    - Linux: make is installed by default on most Linux distros
    - Mac: install Xcode command line tools to get make
    - Windows: Click here for installation instructions
- gcc/g++ >= 5.4
    - Linux: gcc / g++ is installed by default on most Linux distros
    - Mac: same deal as make - install Xcode command line tools
    - Windows: recommend using MinGW
- wxWidgets >= 3.0
    - Linux: `sudo apt-get install libwxgtk3.0-dev libwxgtk3.0-0v5-dbg`
    - Mac: There is a homebrew installation available.
    - Installation instructions can be found here. Some version numbers may need to be changed in instructions to install v3.0 or greater.

Basic Build Instructions

1. Clone this repo.
2. Make a build directory in the top level directory: `mkdir build && cd build`
3. Compile: `cmake .. && make`
4. Run it: `./membot`.

Project Task Details

Currently, the program crashes when you close the window. There is a small bug hidden somewhere, which has something to do with improper memory management. So your first warm-up task will be to find this bug and remove it. This should familiarize you with the code and set you up for the rest of the upcoming tasks. Have fun debugging!

Aside from the bug mentioned above, there are five additional major student tasks in the Memory Management chatbot project, which are:

Task 1 : Exclusive Ownership 1

In file `chatgui.h` / `chatgui.cpp`, make `_chatLogic` an exclusive resource to class `ChatbotPanelDialog` using an appropriate smart pointer. Where required, make changes to the code such that data structures and function parameters reflect the new structure.

Task 2 : The Rule Of Five

In file `chatbot.h` / `chatbot.cpp`, make changes to the class `ChatBot` such that it complies with the Rule of Five. Make sure to properly allocate / deallocate memory resources on the heap and also copy member data where it makes sense to you. In each of the methods (e.g. the copy constructor), print a string of the type "ChatBot Copy Constructor" to the console so that you can see which method is called in later examples.

Task 3 : Exclusive Ownership 2

In file `chatlogic.h` / `chatlogic.cpp`, adapt the vector `_nodes` in a way that the instances of `GraphNodes` to which the vector elements refer are exclusively owned by the class `ChatLogic`. Use an appropriate type of smart pointer to achieve this. Where required, make changes to the code such that

data structures and function parameters reflect the changes. When passing the `GraphNode` instances to functions, make sure to not transfer ownership and try to contain the changes to class `ChatLogic` where possible.

Task 4 : Moving Smart Pointers

In files `chatlogic.h` / `chatlogic.cpp` and `graphnodes.h` / `graphnodes.cpp` change the ownership of all instances of `GraphEdge` in a way such that each instance of `GraphNode` exclusively owns the outgoing `GraphEdges` and holds non-owning references to incoming `GraphEdges`. Use appropriate smart pointers and where required, make changes to the code such that data structures and function parameters reflect the changes. When transferring ownership from class `ChatLogic`, where all instances of `GraphEdge` are created, into instances of `GraphNode`, make sure to use move semantics.

Task 5 : Moving the ChatBot

In file `chatlogic.cpp`, create a local `ChatBot` instance on the stack at the bottom of function `LoadAnswerGraphFromFile`. Then, use move semantics to pass the `ChatBot` instance into the root node. Make sure that `ChatLogic` has no ownership relation to the `ChatBot` instance and thus is no longer responsible for memory allocation and deallocation. Note that the member `_chatBot` remains so it can be used as a communication handle between GUI and `ChatBot` instance. Make all required changes in files `chatlogic.h` / `chatlogic.cpp` and `graphnode.h` / `graphnode.cpp`. When the program is executed, messages on which part of the Rule of Five components of `ChatBot` is called should be printed to the console. When sending a query to the `ChatBot`, the output should look like the following:

```
ChatBot Constructor
ChatBot Move Constructor
ChatBot Move Assignment Operator
ChatBot Destructor
ChatBot Destructor
```