Object-Oriented Programming 2021-22 Assignment 3 (Creature Community)
Name: Damianakis Damianos

The assignment has been implemented in C++ using the g++ compiler. The code is thoroughly commented for explanations.

For compiling and running the code:
- Compilation command: `make`
- Execution command: `make run`
- You can also use `make valgrind` to check memory allocation and deallocation correctness and to ensure there are no memory leaks.

To change input values, modify the values on line 33 of the `makefile`. For demonstrating the program's functionality and verifying that all requested functions work correctly, I've set the values to 20, 100, 2, 3, and 3 in the order of the assignment.

The organization of files is as follows: there's a header file (`header.hpp`) containing prototypes for all classes, function prototypes, and any other necessary content. Each of the requested classes (creature, good_creature, bad_creature, and creature_society) has a separate .cpp file. An additional file (`other.cpp`) contains various functions implemented for the assignment.

- **creature**: Constructor, copy constructor (using initializer list), and destructor have been implemented. The `bless` function is pure virtual to delegate actions to the subclasses `good_creature` and `bad_creature`. Similarly, the `is_a_good` and `clone` functions are declared and later overridden in the subclasses. An extra function `print` has been implemented for implementation purposes, which prints a creature's name and life.

- **good/bad creature**: Both classes have the same functions with similar implementation, and they share a `thrsh` data member (good and bad, respectively). They each have a constructor, copy constructor (with an object of type creature in their init list), and a destructor. The `clone` function initially returns a creature type, creating a clone using `new`, and then returning it. The clone's memory deallocation is handled in the `society` class. In the `bless` function, after checking that the creature is not bad and that it's healthy, it returns true; otherwise, it returns false.

- **creature society**: A dynamic two-dimensional array is used for storing creatures. In the constructor, two static arrays with 10 names each are created (the variable `var` is given in the `main` and used by extra functions in `other.cpp`). To ensure unique names, a random number corresponding to the position in the array is appended to each name. Then, both good and bad creatures are randomly generated, and memory is allocated accordingly. The `print` function prints the community's state by calling the `print` function of the creature class. In the `bless` function, if the creature is good, the `clone_next` function is executed, and if it's bad, the `clone_zombies` function is executed. The `beat` function simply calls the `beat` function of the creature class. The `clone_next` function loops from the current creature's position to the total number of creatures. If the current creature is not the last

one, the next one's position is deallocated, and the `clone` function is called to place the clone in the freed position. If the current creature is the last one, the first position in the array is deleted, and the clone takes its place. The `clone_zombies` function takes the creature's position as an argument and contains its code within an `if` statement, which only runs if the current creature isn't the last one in the community. It checks if the current zombie is indeed a zombie and if the following creatures are zombies, then performs cloning in the same way as in the `clone_next` function. Finally, it checks the difference between the zombie's position and the current creature's position, ensuring that it's greater than one so that all subsequent creatures are cloned if they're zombies.

- **main**: The input values are converted to integers, and an instance of the `society` class is created. The initial state of the community is printed. In a loop running `M` times, random `bless` and `beat` actions are performed, and at each step, the community's state is printed to demonstrate the functionality of `beat`/`bless` and cloning. Finally, appropriate messages about the community's state are printed as required.