CMPUT 350 Lab 3 Exercise Problems

Rules:

- You can use all course material and man pages, but no other information such as web pages, books, or written notes. Using other information sources during the exercise constitues cheating.
- Your programs must compile without warning using

  ```
  g++ -Wall -Wextra -Wconversion -Wsign-conversion -O -g -std=c++17 ...
  ```

  In case there are compiler warnings or errors, marks will be deducted.
- Test your programs with different values. For now, the speed of your program is irrelevant. So don't spend time on optimization
- You must check for the appropriate preconditions/postconditions. Your program shouldn't crash or have undefined behaviour (**hint**: use asserts)!
- Your programs must be well structured and documented. Use ctrl-x t in Emacs to pretty-print it. Marks are assigned to functionality, program appearance, and comments.
- In case your program hangs, use ctrl-c to terminate it.
- Remember that you need to include the appropriate header files. To find out which ones you need for specific functions such as printf, use then man command.

Submit your solution files `AnimalSim.cpp` on eClass under "Lab 3 / Submission".

Important: Submit often (the system will only accept solutions submitted before 16:50)

[31 marks] In file `AnimalSim.cpp` create a class hierarchy for simulating animal behaviour in the wild. Each animal has an x,y location, an alive flag, an age, a print function, and an act function which may change the animal's location, create at most one new animal ("offspring"), or kill another animal.

The act function in the Animal base-class has the following signature:

```
virtual void act(World &w) = 0;
```

where `w` is a `World` that holds an array of `Animal` pointers which describes the current state of the simulation (see below).

The print function prints animal data and has this signature:

```
virtual void print() const = 0;
```

1. Based on your Animal base-class, define
   - Mouse which always walks West (decrements x)
   - Goose which always flies North (increment y)

- Rabbit which creates one Rabbit each time it acts (at the same location)
- Bear which kills a random animal each time it acts. Bears can't kill themselves and can only choose targets among animals that existed at the beginning of the iteration (i.e. they can kill other bears, but not themselves)

2. Implement missing code parts (marked with "implement ...")

3. To test your implementation use `World::print()` which prints the current list of animals to `stdout`. In `World::print()` a loop must call virtual Animal function `print()` on each animal in turn.

The output format of `World::print()` is as follows:

```
<space> <Animal-Number> animal(s)
<space> <Animal-Type> <Location(x-y)> <Age>
<space> <Animal-Type> <Location(x-y)> <Age>
...
```

E.g. output of `simulate(1)` for a world containing 6 animals:

```
iter 0
  6 animal(s)
  Mouse 100 100 0
  Mouse 50 60 0
  Goose 90 80 0
  Goose 10 20 0
  Rabbit 80 80 0
  Bear 70 90 0

iter 1
  6 animal(s)
  Mouse 101 100 1
  Mouse 51 60 1
  Goose 90 79 1
  Rabbit 80 80 0
  Rabbit 80 80 1
  Bear 70 90 1
```

Note that in this simulation the bear killed the second goose in the first step. The order of animals is implementation-specific. What matters is that all animals are accounted for.

Test your simulation on the starting configuration shown above with 20 steps.

Tips:

- Familiarize yourself with the public interface of class World
- Newly created animals don't act during the simulation step they were created in
- To create new offspring, `w.add_animal(...)` needs to be called
- Use const as often as you can

- For functions that don't use their parameters use this construct

  ```
  void act(World &) { ... }
  ```

  (no parameter name) to avoid compilation warnings, or alternatively,

  ```
  void act([[maybe_unused]] World &w) { ... }
  ```

  This is known as an *attribute*, which is supported in C++17 to indicate to the compiler that this variable may not be used, and thus to not warn about it.

- Use the given starting codebase `AnimalSim.cpp`, and implement and test the missing pieces, designated with `implement ...`