Exercise Sheet #6

Deadline: 27.05.2024, 12:00h

Exam The exam (Klausur) will take place

July 24, 2024 in Phys__.102, starting 10:15h.

Problem 1 (*Discrete Dynamical Systems: The Tent Map*) (10 points)

The *tent map*,

$$x_{n+1} = T_\mu(x_n) = \mu \min\{x_n, 1 - x_n\},$$

is a discrete dynamical system defined on the unit interval $[0, 1]$, with a parameter $\mu \in [0, 2]$.

(a) For now, let $\mu < 1$. Find all fixpoints of the tent map, i.e. find all points that fulfill $x^* = T_\mu(x^*)$[1]. You may present your solution to this part in a comment of your submitted code.

(④ points)

(b) Now implement the tent map in `C++`. Define a function

```
float tent_map(float x, float mu)
```

that performs one iteration of the tent map. To get a feeling for the stability of the fixpoint(s) found in part (a), experiment with your code by setting $\mu < 1$ and iterating the map for different initial values $x_0 \in [0, 1]$ and printing $|x_n - x^*|$ to the console for $n = 1, \ldots, N$ with a sufficiently large number of iterations $N \in \mathbb{N}$.

Optional: You may verify your intuition analytically.

Hint: For better readability use `#include <iomanip>` to print a value `x` in scientific notation using

```
std::cout << std::scientific << std::setprecision(3)<< x << std::endl;
```

(③ points)

(c) Perform a numerical stability analysis as described in part (b) with a stable fixpoint you identified.

Explicitly use the values $x_0 = 0.6$ and $\mu = 0.8$ trying different maximum iterations $N = 10, 100, 1\,000$ and $10\,000$. Do you notice anything fishy? If so, what and how could you fix it?

(③ points)

[1] Note that this equation means that starting the iteration of the map from the fixpoint x^*, the time series never moves away from the fixpoint.

Problem 2 (*Classes and Inheritence*) (10 points)

A zoo wants to keep track of its animals and wants you to write the framework to do so. You should define each animal as a class instance, such that data about them can be accessed and manipulated easily. Please follow these steps to implement the framework:

(a) Create a class `Animal` with the member variables `std::string name` storing the animal's nickname, `int cageNumber` storing the cage the animal occupies, `int birthYear` storing the year of birth and a method
```
void printAge(int year)
```
that receives the current year as input and prints the animal's age in years to the console. Set the constructor to initialize all the variables.
(④ points)

(b) To make sure the zoo employees don't corrupt the data, define all variables as private. The method `printAge` shall be public. Further, implement another public method
```
void printData()
```
that prints all available data about an animal to the console.
(② points)

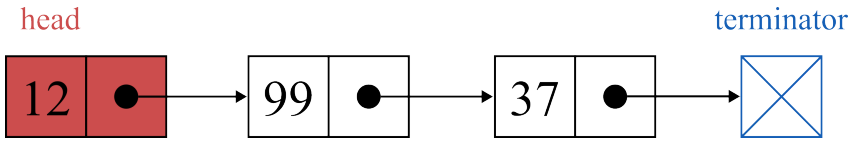
(c) The zoo has grown much larger and would like to keep a count of the different species it has. Write a species-specific child class, `Wolf`, which inherits from `Animal`. Add a counter to the new class `static int counter` and change the constructor and destructor such that they increase and decrease the counter by 1, respectively. Print out the updated count each time an instance is created or destroyed. Remember to initialize the counter to zero before running your code.
(② points)

(d) Override the method `printAge` in `Wolf` such that it prints the age multiplied by a factor of 7 (measured in dog years). Can you access `birthYear`? Change its definition to `protected` in the base class, so it becomes visible to the child class.
(① point)

(e) Try out the framework with a meaningful example.
(① point)

Problem 3 (Advanced: *Linked List*) (10 points)

A linked list is a commonly used data structure comprising a sequence of nodes that contain two fields: a data field and a link (pointer) to the next node. The last node is linked to a terminator used to signify the end of the list. The following illustration demonstrates a linked list storing integer values.



In this exercise we will implement a linked list (or more specifically a singly linked list) in `C++`.

- Create a templated class `Node` with two public variables storing data of the templated type `T` and a pointer to the next node in the list.
- Write a templated class `LinkedList` with a public variable `head` that is a pointer to the head node of the list (i.e. the first node in the list). In the constructor, initialize `head` with the `nullptr`.
- Implement a public method for the `LinkedList`
  `void addNode(T val)`
  that appends an element to the head of the list. Make sure to adjust the `head` variable.
- Implement a public method for the `LinkedList`
  `void printList()`
  which prints out the list to the console.
- Implement a public method for the `LinkedList`
  `void removeNode(T val)`
  which removes the first occurrence of a node holding the data `val`. Again, make sure to adjust `head` accordingly and mind the case that the given data `val` is not in the list.
- Test your implementation with a meaningful example.