ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Programming Techniques II
Exercise 7

HS 15
Prof. M. Troyer

Task 7.1 PennaLV

Remember the Adobe polymorphism example back in the week 01 lecture? Let's implement it!

The three cornerstones for a concept wrapper are the following:

- base class: for dynamic polymorphism, specifies interface for the wrapper

```
struct print_concept_wrapper_base {  // long name for clarity :)
    virtual ~print_concept_wrapper_base() {}
    virtual void print() = 0;
};
```

- templated derived class: holds the implementation of the concept

```
template<typename T>
struct print_concept_wrapper_derived: public print_concept_wrapper_base {
    print_concept_wrapper_derived(T const & t): T(t) {}
    // later maybe a move-T-ctor?
    void print() override {
        t.print(); // T has to have a method called print()
                   // this is what we call the concept!
    }
    T t; // holds the implementation
};
```

- wrapper class: templated constructor. Holds a base class pointer and forwards functions

```
struct print_concept_wrapper {
    template<typename T>
    print_concept_wrapper(T const & t):
        ptr(new print_concept_wrapper_derived<T>(t)) {}
    void print() {
        ptr->print(); // dynamic polymorphism happens here
    }
    std::unique_ptr<print_concept_wrapper_base> ptr; // base class pointer
};
```

- sidenote: you cannot forward static methods, since there is no static virtual. Just forward the static methods as normal methods.

Now everything that fullfills the concept (`t.print()`) can be held by the wrapper class and used polymorphically without having to derive from a base class!
For your pennaLV implementation, write a wrapper `animal_concept` class (also `animal_concept_base` and `animal_concept_derived`) and use the `animal_concept` instead of a `std::shared_ptr<animal_base>` in your container. Implement the the concept such that it works in your simulation (with as little change to the simulation as possible). Now your animals do not need to derive from the same base class anymore. Once done, you can try to remove unneccessary constructor calls with the help of universal references / forward and move, but only worry about this after your concept works.

Task 7.2 Fastest Runtime PennaLV Challenge (optional)

Write a C++ header (and a CMakeFiles.txt with compiler flags such as -O3 and -march=native) which provides a function that can be called like this :

```
pennaLV(seed, filename, N_generation
, sheep_gene_size, sheep_repr_age, sheep_threshold
, sheep_mut_rate, sheep_N_init, sheep_N_max
, bear_gene_size, bear_repr_age, bear_threshold
, bear_mut_rate, bear_N_init, bear_N_max); // doesn’t need to return anything
```

The rules are:

- sheep and bear are mixed in the same container (or should behave like they where / do not update a block of bears and later a block of sheep, as this gives an advantage to the earlier one)
- do not optimize the random number generator (use our random number generator)
- the output file (`filename`) should be of the same format (not necessarily content) as the current solution
- children may not be updated in the round they were created
- how/where you insert children is up to you
- aside from the above, feel free to redesign/prune the code where you see fit

For those who know about intrinsics (SSE, AVX ...): evaluate whether they give a speedup for this kind of problem. We will run and benchmark your submissions on an Intel Ivy Bridge single-core consumer CPU. The fastest implementation (at least faster then our current solution with `std::list`) that show the same statistics (not necessarily identical numerics) wins!
The submission deadline for this challenge is Wednesday 04.11.15 at 05:00 in the morning. Notify us via the mailing list (pt2_hs15_ta@lists.phys.ethz.ch) with a link to your uploaded solution on your PT2 repository on GitLab.

Additional Notes:

A plot-tool for the profiler output (`MIB_SAVE(“cycle”, “file1.txt”)`) is now available (extern/tools/profilerplot.py). It can be used with the following arguments:

- `./profilerplot.py file1.txt`
- `./profilerplot.py file1.txt ... file5.txt`
- `./profilerplot.py file1.txt file2.txt .... file5.txt out=pdf_name depth=1 root=main`

The arguments have the following effect (order does not matter):

Name	Effect	Default
`out`	the name of the created pdf file	`noname`
`depth`	how many additional layer of children should be shown	`0`
`root`	only the children of this element are plotted	*all*