Exercise 7: Array of floating point numbers

Deadline: 7th May 2021

Resource acquisition is initialization (RAII) is a programming idiom used in several object-oriented, statically-typed programming languages to describe a particular language behavior. In RAII, holding a resource is a class invariant, and is tied to object lifetime: resource allocation (or acquisition) is done during object creation (specifically initialization), by the constructor, while resource deallocation (release) is done during object destruction (specifically finalization), by the destructor. In other words, resource acquisition must succeed for initialization to succeed. Thus the resource is guaranteed to be held between when initialization finishes and finalization starts (holding the resources is a class invariant), and to be held only when the object is alive. Thus if there are no object leaks, there are no resource leaks.

Task 1

Define the class `realarray` for representing an array of floating-point numbers according to the design pattern RAII. An array of the given size should be created on the heap while the constructor is running and removed from the heap in the destructor. Implement copy and move semantics in the `realarray` class. Place the class definition in the `calc` namespace.

```
class realarray {
    int len; // numer of bits
    double *arr; // array of bits
public:
    explicit realarray(int siz); // base constructor
    realarray(const realarray &arr); // copy constructor
    realarray(realarray &&arr); // move constructor
    realarray& operator= (const realarray &arr); // copy assignment
    realarray& operator= (realarray &&arr); // move assignment
    ~realarray(); // destructor
// ...
};
```

Raise an exception `invalid_argument` when the size of the array passed to the constructor is not a positive number. Use uniform initialization to clear the array (0 in each cell).

Define index operators in the wrapper class that return a value for constant arrays and a cell reference for mutable arrays, respectively. When the index is outside the acceptable range, throw an exception `out_of_range`.

Thoroughly test the functionality of the wrapper class, with particular emphasis on the exceptions reported.

Task 2

Complete the definition of the wrapper `realarray` with an argumentless constructor that will create an array of floating-point numbers with the maximum possible size, which is the power of 2. Use the `new` operator with the `nothrow` parameter. This constructor should initialize the created array with random values from the interval [0, 1).

What was the size of the array you created?

Task 3

Complete the definition of the wrapper `realarray` with a constructor that will create and initialize an array based on a list of values `initializer_list<double>`.

Next, define a function (or multiplication operator) that will calculate the scalar product of two arrays of the same size. Insert an assertion into the function that will check that the lengths of both arrays are really the same.

Did the assertion work when you provided arrays of different sizes? How to disable assertions?