Workshop 30

SafeArray, Copy Constructors, Copy assignment operator

This workshop focuses on memory management and safety, using an implementation of a class `SafeArray`.

`SafeArray` is a template class representing an array of objects of an arbitrary type. Memory inside `SafeArray` is dynamically allocated, and it should provide basic functions while keeping the safety and correctly handling errors.

Parts of `SafeArray` that have to be implemented.

Fields

The class should have two fields.

1. Pointer to an object of the template type `T`. This pointer will point to the first element of the array.
2. Unsigned integer representing size.

There should be getters for both fields as inline functions providing read-only access to them.

Regular constructors

1. The default constructor.
   It should provide default arguments to the fields of the class, so an object of the class can be created without input arguments.
2. A constructor taking a size variable and a variable of the template type `T`.
   It should initialize an array of `size` elements with the value passed into it as the second argument. If only size is passed, it should use default objects of type `T`, created as `T()`.
3. Constructor with an initializer list.
   It should use the size and values of the initializer list to initialize the array.

Constructors must work correctly for all cases, including initialization of an array of size 0. Constructors must never leave any fields uninitialized. Use constructor delegation to achieve this. Constructor delegation is "calling other constructors in the current one", like when you call the constructor of a base class in a derived one.

Copy constructor

The copy constructor takes as an argument another object of the type `SafeArray` and creates a copy of that object.

Copy assignment operator

The class should have an overloaded `operator=` which takes another object of type `SafeArray` as an input by reference. This operator should replace current values of all fields with ones from the input object.

The straightforward approach - delete current values, assign new values - is unsafe. The better approach is the following:

1. Create a new temporary object as a copy of the input object.
2. Swap the values of the temporary object and the current one (the one that calls the operator).
3. Delete the temporary object. This will be done automatically on exiting the function.

This approach requires implementation of a function `swap` that swaps values of fields of two `SafeArray` objects.

Destructor

The array uses dynamically allocated memory. It must be cleaned when the array is destroyed.

Index functions

The class should provide both `operator[]` and method `at()` to access the data. `operator[]` should not perform checks on the size of the array. Method `at()` should throw an `out_of_range` exception if the index is larger than the size of the array.

Both index functions should have const and non-const versions.

operator<<

`operator<<` for `std::ostream&` objects should be overloaded to allow printing data from `SafeArray` objects.

Resize

The class should have a `resize` function which takes two arguments: size as an unsigned variable and the default value as an object of the template type `T`.

It should delete the previously allocated memory and allocate new memory with the new size. If the new array is larger than the old one, it should fill the new free elements with the default

value argument. If the default value argument isn't provided, the function should use default objects of type `T`.

The function should correctly handle edge cases, not performing unnecessary actions.