Task 1. Write a template class that acts like the library find algorithm. We need to use two template parameters, one to represent the function's iterator parameters and another for the type of the value. Instead of returning an iterator to the found element, the function should return a container (a vector for example) with copies of the found elements.

Task 2. Write a template class that acts like the library find_if algorithm. The function receives a template parameter representing the iterators and a second template parameter representing the predicate. The function should return a vector with all the found elements.

Task 3. Write a template class for generating simple linked lists. Your class should at least offer the following interface:
- A default constructor
- A copy constructor
- A copy assignment
- A method for inserting an element
- A method for adding all the content of a list passed in as parameter to the current list
- A method for printing the content of the list

Define the template class in a header file. Define each member function outside the class definition.

Task 4. Add static data member to the class declared before. The static data member should be of a different template type of the one(s) using in exercise 3. For example, if the template class looks like
template<typename T>
class List {…};

The static data member is allowed to be of a different type U. What do you need to change anything in your class template?

Task 5. BONUS EXERCISE (Optional). Have you defined your linked list using regular pointers? The best alternative would be however to implement the linked list using unique_ptr pointers (this way we do not need to worry about the life cycle of elements in our list, and the overhead regarding using regular pointers is rather small). Define a linked list using unique_ptrs.