Exercise 6a. (Dynamically allocated linked list)

An address of the first node is capable to represent the whole list in dynamic memory, provided that the next member in the last node is NULL to indicate the end of the list. In this case the data type Tlist of class LinkedList can be defined as follows:

```
template <class T>
class LinkedList {
private:
     // definition of the list node class
     class Node {
          …
     };

     Node *first;   // no *last variable in this implementation!!
     …
};
```

When operation functions are defined for that type we get the class LinkedList. Write the following operation functions for the class LinkedList defined above

```
     LinkedList();
     ~LinkedList();
     LinkedList<T>& insert_to_end(T data);
     void print(std::ostream &out = std::cout) const;
```

Start working from the given example program (dynlist.h and dynlist.cpp, available from the Oma-portal). This list is a genuine ADT including the test application. First check how the program works, then modify the class definition by removing the `*last` variable in the LinkedList class definition and modify the list implementation in such a way that it works again.

Exercise 6b. (Extra, Universality of the dynamically allocated linked list, 0.25p)

Warning: this exercise requires good knowledge of C++ polyformism (i.e. this is quite a demanding exercise).

In this exercise we demonstrate that it is possible to store different kind of data items to the given dynlist.h list template implementation. This utilization of the linked list rely on the C++ polyformism concept where one pointer can point to different kind of objects if they are related (inheriting classes can be pointed by the parent class pointer).

Now the object to be stored to the linked list is as follows

```
     class NodeData {
          friend ostream& operator<< (ostream& stream, const NodeData *p);
```

```
	public:
		virtual ostream& print(ostream &stream) const = 0;
		virtual bool     operator<(NodeData *) = 0;
	};
```

And the main program is

```
int main() {
	LinkedList<NodeData *> list;

	list.insert_to_end(new String("First complex number")).insert_to_end(new Complex(1, 3));
	list.insert_to_end(new String("Second complex number")).insert_to_end(new Complex(5, 9));

	list.print();
}
```

Your task is to make the given main function given NodeData class, and given dynlist.h linked list implementation work together by defining the missing String and Complex classes and output operator for the NodeData. String class stores strings (you can use the standard C++ string class for that) and Complex stores complex values (real and imaginary part represented by two float numbers).