Computer Lab #7.

Operator Overloading in C++

Vectors are a kind of sequence container. As such, their elements are ordered following a strict linear sequence. Vector containers are implemented as dynamic arrays; just as regular arrays, vector containers have their elements stored in contiguous storage locations, which means that their elements can be accessed not only using iterators but also using offsets on regular pointers to elements. But unlike regular arrays, storage in vectors is handled automatically, allowing it to be expanded and contracted as needed.
Vectors are good at:

- *Accessing individual elements by their position index (constant time).*
- *Iterating over the elements in any order (linear time).*
- *Add and remove elements from its end (constant amortized time).*

Compared to arrays, they provide almost the same performance for these tasks, plus they have the ability to be easily resized.

Task 1.

Download the source code from the Course Webpage. Open the downloaded archive and add ***Vector.h*** and ***VectorSample.cpp*** files to your C++ project. Create the missing ***Vector.cpp*** source file and add it to the project. Follow comments and instructions in the downloaded code.

a) Look into the ***Vector.h*** header file and write implementations for declared class methods in ***Vector.cpp***. Follow the hints in the header file.

b) Examine the *operator=* and answer the following questions:
   - Under what constraints can we call *operator=* from a copy constructor?
   - Why are both parameter and returning value passed by reference?
   - What happens with self-assignment like *v=v*?

c) What modifications need to be done at *Vector* class to be able to manipulate user-defined data types instead of a built-in *int* type (e.g. to have a vector of *String* objects)?

Decomposition of the program:

- *Vector.h* file: class declaration header;
- *Vector.cpp*: implementation of class members declared in the header;
- *VectorSample.cpp*: the main program with testing code.

Homework. (optional)

Add a sorting function to the *Vector* class. Call it from *VectorSample.cpp* and make sure that everything works properly.