Network register II.

Submission deadline:	**2019-04-28 23:59:59**
Late submission with malus:	**2019-06-30 23:59:59** (Late submission malus: 100.0000 %)
Evaluation:	**2.0000**
Max. assessment:	**2.0000** (Without bonus points)
Submissions:	3 / 20 Free retries + 20 Penalized retries (-2 % penalty each retry)
Advices:	0 / 2 Advices for free + 2 Advices with a penalty (-10 % penalty each advice)

The problem is an extension of the simpler 'Network register' problem. We consider virtualized computers and virtualized networks in this problem. The problem is to design and implement classes that simulate a database of networked computers. We need to store information about networks (`CNetwork`), computers (`CComputer`) and their components: `CCPU`, `CMemory`, and `CDisk`. Compared to the simpler version, this version must allow to attach networks (and their attached computers) into existing computers. Moreover, there is an interface to clone computers/networks and place them elsewhere to the register.

This assignment is focused on class design, where inheritance, polymorphism and abstract methods are used. If these OOP paradigms are used correctly, the implementation is short and clean. On the other hand, if the design is wrong, the implementation will be lengthy with repeated code. Try to identify base class and subclasses, use inheritance.

The classes and the interface:

`CNetwork`
represents a network. The interface is:

- constructor with the network name parameter,
- destructor, copy constructor and operator = (if the automatically generated are not correct),
- method `AddComputer` which adds another computer to the list,
- method `FindComputer(name)` which returns a pointer to `CComputer` object with the given name, or an invalid pointer if the computer does not exist. Caution: the computer is searched in virtualized computers as well,
- method `FindNetwork(name)` which returns a pointer to `CNetwork` object with the given name, or an invalid pointer if the network does not exist. Caution: the network is searched in virtualized networks as well,
- output operator which displays the network in the format shown below. The computers are listed in the order they were added into the network object.

`CComputer`
represents a computer. The interface is:

- constructor with computer name parameter (string),
- destructor, copy constructor and operator = (if the automatically generated are not correct),
- method `AddComponent` which adds another component to the list,
- method `AddAddress` which adds another address to the list of addresses,
- method `FindComputer(name)` which returns a pointer to `CComputer` object with the given name, or or an invalid pointer if the computer does not exist. Caution: the computer is searched in virtualized computers as well,
- method `FindNetwork(name)` which returns a pointer to `CNetwork` object with the given name, or or an invalid pointer if the network does not exist. Caution: the network is searched in virtualized networks as well,
- method `Duplicate(remap)` creates a copy of the computer instance. The copy is identical to the original, including the virtualized computers/networks. The only difference between the original and the copy may be the computer/network names. The method takes `remap` parameter which lists `(original name, new name)` pairs. When creating the copy, the method searches the list for each computer/network name. If the original name is found in the list, it is replaced with the new name. Otherwise (not found in the list), the name is copied unchanged.
- output operator which displays the computer in the format shown below. The addresses are listed first (in the order they were added), followed by the list of components (again in the order they were added).

`CCPU`
represents a CPU. The interface is:

- constructor with the number of cores (int) and frequency (int, MHz) parameters,
- destructor, copy constructor and operator = (if the automatically generated are not correct).

`CMemory`
represents a RAM memory. The interface is:

- constructor with the memory size (int, in MiB),
- destructor, copy constructor and operator = (if the automatically generated are not correct).

`CDisk`
represents a storage. The interface is:

- constructor with the storage type (symbolic constant `SSD` or `MAGNETIC`) and disk size (int, in GiB),
- destructor, copy constructor and operator = (if the automatically generated are not correct),
- method `AddPartition` which adds another partition to the disk description. The method will take two parameters: partition size (int, in GiB) and a partition description (string). The partitions are listed in the order they were added.

Submit a source code with the implementation of classes `CNetwork`, `CComputer`, `CCPU`, `CMemory`, and `CDisk`. All required auxiliary declarations/functions shall be included in the source file submitted. The #include preprocessor definitions and your tests shall be placed in conditional compile blocks as shown in the attached archive.

Notes

- Use the typecast operators (`dynamic_cast`) with caution. The reference implementation does not use any of the RTTI based operators (i.e. it does not use `dynamic_cast`, nor it uses `typeid`). In general, if RTTI based operators are used too often, your design is probably sloppy. A code with RTTI based operators tends to have many branches, is difficult to understand, and is difficult to extend. Pay a special care to the design, prefer polymorphism over RTTI.
- Please note there is no `typeinfo` header file included. Thus, your implementation cannot use `typeid`.
- Your implementation must use classes that form an inheritance hierarchy. This problem is suited for a solution