C++ Programming techniques

PhD program in Bioengineering and Robotics

Final Project

To develop a program for the management of a computer network.
A network is composed of generic network elements. For the sake of simplicity we assume that they can only be nets and PCs.

A generic network element (`NetworkItem`) is described by an identifier and by an `IP` address (e.g. 192.168.1.1); it will also have the following methods: `Print()` and `Size()`. The first one prints the identifier and the IP address, and in the case of a network, the list of identifiers and IP addresses connected to it. The second returns the memory occupied by all the network elements, i.e. the sum of the memory occupation of all the PCs in the network.

A PC (`PC`) is characterized by a memory occupation: this memory must be managed by a dynamic allocation.

A net (`Net`) is described by a list of pointers to generic network items (`NetItemList`) and by a list of IP addresses (`IPList`); it will have the methods for adding (`Add()` and `AddCopy()`) and removing elements (`remove()`).

When an object of type `Net` is created, the `IP` list is initialized with a list of addresses. The method `AddCopy(const NetworkItem* item)` removes and IP from such a list and assigns it to the generic network item (`item`), then inserts a copy of this in the list `NetItemList`. To do this it is necessary to use the method `Clone()`.

The method `Add(NetworkItem* item)` removes an `IP` from the address list, assigns it to generic network item (`item`) and inserts it into the list `NetItemList`.
The method `remove(const IP ipremove)` searches in a `Net` the generic network element with IP `ipremove`, removes the element from the list `NetItemList` and inserts the `IP` of the removed element into the list of available `IPs`.

Simplifications: the method `remove` searches the element to be removed only in the `NetworkItem` elements directly connected to the network. Moreover, the search of the element to be removed is only by `IP` and not with respect the identifier, since this may not be unique.

Note: objects (`PC` and `Net`) are dynamically created and if they are inserted in a network they are managed (and destroyed) by the network they are connected to.

We suggest to implement the software as follows:

- To implement the `IP` class and to verify its correctness.
- To implement the abstract class `NetworkItem`.
- To implement the `PC` class and to verify its correctness.

- To implement the `Net` class and to verify its correctness.
- To develop a complete example using all the classes (see also the provided code).

```
class IP{
	int ip[4];
	public:
	//..
};

class NetworkItem{
	protected:
		string ItemName;
		IP m_ip;
	public:
		virtual void Print()const;
		virtual int Size()const =0;
		virtual NetworkItem* clone() const =0;
	//..
};

class Net : public NetworkItem{
	list<NetworkItem*> NetItemList;
	list<IP> IPList;
	public:
		bool AddCopy(const NetworkItem* item);
		bool Add(NetworkItem* item);
		bool remove(const IP ipremove);
	//..
};

class PC : public NetworkItem{
	int *mem;
	public:
	//..

};
```

A possible test to assess the developed class hierarchy:

```
Net root("root",IP(10,1,3,1));
PC* pc= new PC("pc",IP());
root.Add(pc);
Net* nodo1 = new Net("node1",IP());
for(int i=0;i<5;i++){
	PC* pc = new PC("pc1",IP());
	nodo1->Add(pc);}
Net* nodo2 = new Net("node2",IP());
for(int i=0;i<5;i++){
	PC* pc = new PC("pc2",IP());
	nodo2->Add(pc);}
nodo1->AddCopy(nodo2);
root.Add(nodo1);
root.AddCopy(nodo2);
root.Add(nodo2);
root.Print("");
cout<<"Net size = "<<root.Size()<<endl;
root.remove(IP(w,x,y,z));//where w.x.y.z is the address of the NetworkItem to be
removed
root.Print("");
cout<<"Net size = "<<root.Size()<<endl;
```