Programare generică

Azi ninja-ul nostru a ajuns la o altă provocare și anume aceea de a fi mai flexibil la anumite situații care pot să difere chiar și foarte puțin. A auzit că se vor face template-uri și este curios cum poate să nu se repete scriind codul o singură dată, cod care ar urma să funcționeze pentru o paletă mai largă de tipuri de date, inclusiv pentru cele definite de el.

Prin urmare se consideră în limbajul C++ următoarele clase generice:

```
template <typename T>
class Coada;

template <typename T>
class Nod
{
	T info;
	Nod* next;

public:

	friend class Coada<T>;
	Nod(const T& info, Nod* next = nullptr);

	T getInfo() const;
	Nod<T>* getNext() const;
};

template<typename T>
std::ostream& operator<<(std::ostream& out, const Coada<T>& coada);

template <typename T>
class Coada
{
	int size;
	Nod<T>* front;
	Nod<T>* rear;

public:

	Coada();
	~Coada();

	int getSize() const;

	T getRearInfo() const;
	T getFrontInfo() const;

	bool isEmpty() const;

	void deQueue();
	void enQueue(const T& info);

	T* toVector() const; // converteste coada intr-un vector de tipul T
```

```
    friend std::ostream& operator<< <>(std::ostream& out, const Coada& coada);
};
```

Cerințe:

1) Să se implementeze clasa NrComplex care conține ca și atribute **real** și **imaginar** de tip double (2p)

2) Să se implemteze clasele template de mai sus astfel încât să funcționeze corect. (5p)

3) Să se specializeze clasa generică Coada pentru următoarele tipuri de date: int, double și NrComplex. Se va crea o funcție de test în fișierul .cpp (exemplu pe OCW) (1p)

4) Să se sorteze vectorii obținuți cu **toVector** în main. Atenție ca la sortare unul din operatorii < sau > să existe pentru clasa NrComplex ca sortarea să aibă sens. (1p)