5. Kis zárthelyi, megoldások és pontozás

a kijavított kis zárthelyik megtekinthetők az utolsó hét péntekén, gyakorlat időpontjában, a QB.330. szobában (vagy ott lesz kiírva egy pointer...)

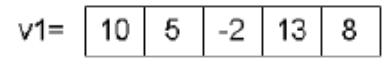
Adott az alábbi Vector dinamikus tömböt reprezentáló osztály:

```
// Egy dinamikus tömb osztály
template<class TYPE> class Vector
{
        // A tömb mérete
        unsigned int elementNum;
        // Az adatokra mutató pointer
        TYPE *pData;
        // Figyeljünk, hogy a friend középen van
        template<class U> friend std::ostream & operator << (std::ostream& os, const Vector<U>& v);
public:
        // Konstruktorok, destruktor
        Vector() { elementNum = 0; pData = NULL; }
        Vector(const Vector& theOther);
        ~Vector() { delete[]pData; }
        // Visszatér a tömb méretével.
        int size()const{ return elementNum; }
        // Törli a tömböt
        void clear();
        // Törli a megadott indexu elemet. A 0 és size()-1 közötti indexek érvényesek.
        void erase(unsigned int position);
        // Visszatér a megadott indexu elemmel, amely módosítható is egyben.
        // A 0 és size()-1 közötti indexek érvényesek.
        TYPE& at(unsigned int position);
        // Visszatér a megadott indexu elemmel, amely csak olvasható.
        // A 0 és size()-1 közötti indexek érvényesek.
        const TYPE& at(unsigned int position)const;
        // Beszúr egy elemet a megadott indexu helyre.
        // Ha az index nagyobb, mint a tömb mérete, megnöveli a tömb méretét,
        // és a szükséges új helyeket nullákkal tölti fel.
        bool insert(unsigned int position, TYPE element);
        // Operator=
        const Vector& operator= (const Vector & theOther);
        // Két operator[]. Az at() tagfüggvény operator formában is.
        TYPE & operator [](unsigned int position);
        const TYPE & operator [](unsigned int position)const;
};
```

Írj egy min() sablon függvényt (template), ami visszadja a legkisebb elemet a vectorban (dinamikus tömbben). (6 pont)

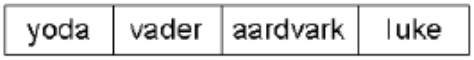
Milyen követelményeket támaszt az osztály a tárolandó típusokkal szemben (pl. operator > rendelkezésre állása)? (4 pont)

```
vector<int> v1;
int x = getMin(v1);
```



x=-2

```
vector<string> v2;
string s = getMin(v2);
```



s=aardvark

Megoldás:

```
template <typename T> T getMin(const vector<T>& v)
{
	T minValue = v[0];
	for (int i = 1; i< v.size(); i++)
		if (v[i] < minValue)
			minValue = v[i];
	return minValue;
}
```

Pontozás:
3 pont: helyes template deklaráció
3 pont: helyes kód

Milyen követelményeket támaszt az osztály a tárolandó típusokkal szemben (pl. operator > rendelkezésre állása)?
operator<
Pontozás:
2 pont: Eltalálja, csak olyat ír, ami tényleg szerepel a kódjában és helyes
2 pont: helyesen rámutat a kódban

Megjegyzések:

- template függvényt kellett írni, nem pedig metódust!
- template<class T> vagy template<typename T> mindegy.
- a getMin-ben az érték szerinti átadás működik, de nem egészséges. (feleslegesen hívódik copy constructor)
- A const “illik.”
- A szöveg félreérthető, mert az operator< a template függvényhez kell, az osztály működni fog enélkül is, de ezért szóltam az elején.