A rendelkezésre álló idő:18 **perc**

Programozás alapja 2. 4. ellenőrző dolgozat. 2016.04.29. Kurz/**Terem**: G5/	Elért pontszám:
Név: **Neptun:**	

1. feladat 6 p

Hozzon létre egy olyan generikus indexelhető tárolót (*Tomb*), aminek sablonparaméterként a maximális méret mellett megadható az index értékének alsó határa is, azaz nem csak 0 lehet az első elem indexe. A tároló az *operator[]* és az *at()* függvényeket leszámítva pontosan úgy viselkedjen, mint a gyakorlatokon elkészített iterátoros generikus tömb (*Array<T, size_t maxsiz = 6>*). Ügyeljen arra, hogy az új osztálynak legyenek olyan konstruktorai, amelyek az *Array* sablonnak is vannak (paraméter nélküli, egyparaméteres, kétparaméteres, kétiterátoros)! Az alábbi kódrészlet a használatra mutat példát:

```
int a[] = { 0, 1, 2, 3 };
Tomb<int, 10, 100> a10_100(a, a+4);          // 10 elemű 100-tól indexelhető tömb
std::cout << a10_100[100] << std::endl;      // kiír: 0
std::cout << a10_100.at(102) << std::endl;   // kiír: 2
```

Egy lehetséges megoldás:

```
template <class T, size_t maxsiz = 6, size_t idx_1 = 0>
struct Tomb : public Array<T, maxsiz> {
    Tomb(size_t n = maxsiz, const T& val = T())
        :Array<T, maxsiz>(n, val) {}
    template <class Iter>
    Tomb(Iter first, Iter last)
        :Array<T, maxsiz>(first, last) {}
    T& operator[](size_t i) {
        return Array<T, maxsiz>::operator[](i-idx_1); }
    T operator[](size_t i) const {
        return Array<T, maxsiz>::operator[](i-idx_1); }
    T& at(size_t i) {
        return Array<T, maxsiz>::at(i-idx_1); }
    T at(size_t i) const {
        return Array<T, maxsiz>::at(i-idx_1); }
};
```

---

2. feladat 2 p

Készítsen függvénysablont (*kiir*), ami iterátorokkal megadott sorozat elemeit kiírja a paraméterként megadott *std::ostream* típusú objektumra! (pl: `char cpp[] = "C++";  kiir(cpp, cpp+3, std::cout);`)

Egy lehetséges megoldás:

```
template <typename Iter>
void kiir(Iter first, Iter last, std::ostream& os) {
    while (first != last) os << *first++;
}
```

---

3. feladat 2 p

Írjon kódrészletet, melyben a fenti példa a10_100 tömbjét – aminek a tartalma már lehet, hogy megváltozott – átmásolja egy olyan tömbbe, amelynek 20 elme van és az első elem indexe 50. Két módszert is mutasson, az egyikben használja a tömb iterátoros konstruktorát!

Egy lehetséges megoldás:

```
Tomb<int, 20, 50>a20_50(a10_100.begin(), a10_100.end());
for (size_t i = 0; i < a10_100.size(); ++i)
    a20_50[50+i] = a10_100[100+i];
```

BBBBBBBBBBBBBBBBBBBBBBBBBBBBB rendelkezésre álló idő:18 **perc**

Programozás alapja 2. 4. ellenőrző dolgozat. 2016.04.29. Kurz/**Terem**: G5/	Elért pontszám:
Név: **Neptun:**	

1. feladat 2 p

Készítsen függvénysablont (*print*), ami iterátorokkal megadott sorozat elemeit kiírja a paraméterként megadott *std::ostream* típusú objektumra! (pl: `char cpp[] = "C++"; print(cpp, cpp+3, std::cout);`)

Egy lehetséges megoldás:

```
template <typename Iter>
void print(Iter first, Iter last, std::ostream& os) {
    while (first != last) os << *first++;
}
```

---

2. feladat 6 p

Hozzon létre egy olyan generikus indexelhető tárolót (*Vektor*), ami pontosan úgy viselkedik, mint a gyakorlatokon elkészített iterátoros generikus tömb (*Array<T, size_t maxsiz = 6>*), de van *push_back()*, és *back()* tagfüggvénye is, amivel a tömb végére lehet elemet tenni, illetve az utolsó elemet el lehet érni (módosításra is). A *back()* függvénynek van konstans környezetben használható változata is. Ügyeljen arra, hogy az új osztálynak legyenek olyan konstruktorai, amelyek az *Array* sablonnak is vannak (paraméter nélküli, egyparaméteres, kétparaméteres, kétiterátoros)! Az *Array* maximális mérete fix, de túlindexeléskor növeli a ténylegesen tárolt elemek számát (*size()*). Az alábbi kódrészlet a használatra mutat példát, melyben használjuk az 1. feladat sablonját is.

```
    char duma[] = "Hello ";
    Vektor<char, 100> ct1(duma, duma+6);        // maximálisan 100 elemű tömb Hello-val
    ct1.push_back('C');
    ct1.push_back('+');
    ct1.push_back('C');
    ct1.back() = '+';
    print(ct1.begin(), ct1.end(), std::cout);   // kiír: Hello C++
```

Egy lehetséges megoldás:

```
template <class T, size_t maxsiz = 6>
struct Vektor : public Array<T, maxsiz> {
    Vektor(size_t n = maxsiz, const T& val = T())
        :Array<T, maxsiz>(n, val) {}
    template <class Iter>
    Vektor(Iter first, Iter last)
        :Array<T, maxsiz>(first, last) {}
    T& back() { return Array<T, maxsiz>::at(Array<T, maxsiz>::size()-1); }
    T back() const { return Array<T, maxsiz>::at(Array<T, maxsiz>::size()-1); }
    void push_back(const T& val) {
        Array<T, maxsiz>::at(Array<T, maxsiz>::size()) = val;
    }
};
```

---

3. feladat 2 p

Írjon kódrészletet, melyben a fenti példa *ct1* vektorát – aminek a tartalma már lehet, hogy megváltozott – átmásolja egy olyan vektorba, amelynek 50 elme van! Két módszert is mutasson, használja a vektor iterátoros konstruktorát!

Egy lehetséges megoldás:

```
Vektor<char, 50>ct2(ct1.begin(), ct1.end());
for (size_t i = 0; i < ct1.size(); ++i)
    ct2[i] = ct1[i];
```