

Programozás alapja 2.	5. ellenőrző dolgozat.	2015.05.08.	Kurz/Terem: G3/	Elért pontszám:
Név:			Neptun:	

1. Tétélezze fel hogy rendelkezésére áll egy olyan generikus sorozattároló, mely rendelkezik a következő tulajdonságokkal:

7p

- van paraméter nélkül hívható konstruktora, ami üres tárolót hoz létre;
- van másoló konstruktora, értékadó operátora, és destruktora;
- van `void push_back(const T&)` tagfüggvénye, ami elemet helyez a tárolóba;
- van `T& back()` tagfüggvénye, ami az utoljára betett elem elérését biztosítja;
- van `void pop_back()` tagfüggvénye, ami kiveszi az utoljára betett elemet a tárolóból;
- van `size_t size()` tagfüggvénye, ami a tárolóban levő elemek számát adja.

Készítsen olyan generikus adapter osztályt, amely egy fenti tulajdonságokkal rendelkező tárolóhoz LIFO jellegű (stack) hozzáférést biztosít! Az új tároló (LIFO) rendelkezzen a következő tulajdonságokkal:

- legyen paraméter nélküli konstruktora, ami üres tárolót hoz létre;
- legyen másoló konstruktora, értékadó operátora és destruktora;
- legyen `void push(const T&)` tagfüggvénye, ami elemet helyez a LIFO-ba;
- legyen `T& top()` tagfüggvénye, ami az utoljára betett elem elérését biztosítja;
- legyen `void pop()` tagfüggvénye, ami kiveszi az utoljára betett elemet a LIFO-ból;
- legyen `bool empty()` tagfüggvénye, ami igaz értékkel tér vissza, ha a LIFO üres.

Hiba esetén (üres tárolón `pop`, vagy `top` művelet) dobjon valamilyen `std::exception&` -ként elfogható hibát!

2. Egy rövid programrészlettel mutassa be az elkészült adapter használatát: olvasson be a standard inputról fájl végéig egész számokat, majd írja ki azokat fordított sorrendben a standard kimenetre! A példában tárolóként `std::vector`-t használjon!

3p

egy lehetséges megoldás:

```
template <class T, class Container >
class LIFO {
    Container tar;
public:
    bool empty() const { return tar.size() == 0; }
    T& top() {
        if (empty()) throw std::out_of_range("empty");
        return tar.back();
    }
    void push(const T& t) { tar.push_back(t); }
    void pop() {
        if (empty()) throw std::out_of_range("empty");
        tar.pop_back();
    }
};

int i; LIFO<int, std::vector<int> > lifo;
while (std::cin >> i) lifo.push(i);
while (!lifo.empty()) {
    std::cout << lifo.top() << std::endl;
    lifo.pop();
}
```

Programozás alapja 2.	5. ellenőrző dolgozat.	2015.05.08.	Kurz/Terem: G3/	Elért pontszám:
Név:			Neptun:	

1. Tételezze fel hogy rendelkezésére áll egy olyan generikus sorozattároló, mely rendelkezik a következő tulajdonságokkal:

7p

- van paraméter nélkül hívható konstruktora, ami üres tárolót hoz létre;
- van másoló konstruktora, értékadó operátora, és destruktora;
- van `void push_back(const T&)` tagfüggvénye: ami új elemet helyez a tárolóba (végére);
- van `T& front()` tagfüggvénye, ami a legrégebben betett elem elérését biztosítja (eleje);
- van `void pop_front()` tagfüggvénye, ami kiveszi a legrégebben betett elemet a tárolóból (elejéről);
- van `size_t size()` tagfüggvénye, ami a tárolóban levő elemek számát adja.

Készítsen olyan generikus adapter osztályt, amely egy fenti tulajdonságokkal rendelkező tárolóhoz FIFO jellegű (sor) hozzáférést biztosít! Az új tároló (SOR) rendelkezzen a következő tulajdonságokkal:

- legyen paraméter nélküli konstruktora, ami üres tárolót hoz létre;
- legyen másoló konstruktora, értékadó operátora és destruktora;
- legyen `void push(const T&)` tagfüggvénye, ami elemet helyez a sorba;
- legyen `T& next()` tagfüggvénye, ami a sor következő elemének elérését biztosítja;
- legyen `void pop()` tagfüggvénye, ami kiveszi következő elemet sorból;
- legyen `bool empty()` tagfüggvénye, ami igaz értékkel tér vissza, ha a sor üres.

Hiba esetén (üres tárolón `pop`, vagy `next` művelet) dobjon valamilyen `std::exception&` -ként elfogható hibát!

2. Egy rövid programrészlettel mutassa be az elkészült adapter használatát: olvasson be a standard inputról fájl végéig egész számokat, majd írja ki azokat a standard kimenetre! A példában tárolóként `std::list` tárolót használjon!

3p

egy lehetséges megoldás:

```
template <class T, class Container >
class SOR {
    Container tar;
public:
    bool empty() const { return tar.size() == 0; }
    T& next() {
        if (empty()) throw std::out_of_range("empty");
        return tar.front ();
    }
    void push(const T& t) { tar.push_back(t); }
    void pop() {
        if (empty()) throw std::out_of_range("empty");
        tar.pop_front();
    }
};

int i; SOR<int, std::list<int> > sor;
while (std::cin >> i) sor.push(i);
while (!sor.empty()) {
    std::cout << sor.next() << std::endl;
    sor.pop();
}
```