A rendelkezésre álló idő:18 **perc**

Programozás alapja 2. 4. ellenőrző dolgozat. 2016.04.26. Kurz/**Terem**: G4/	Elért pontszám:
Név: **Neptun:**	

1. feladat 6 p

Tervezzen generikus vermet (*Stack*), ami maximum 234 generikus adat tárolására alkalmas! Az osztálysablon definíciójában szerepeltessen minden szokásos veremműveletet (*push, pop, top, empty*), de Önnek a konstruktoron kívül csak a *push(), pop()* és az *empty()* tagfüggvényt kell megvalósítania. Az Ön döntése, hogy ezeket inline függvényként valósítja-e meg. Sablonparaméterként vegyen át egy osztályt is, amit hiba estén (üres verem elérése, vagy helyhiány) eldob. Ennek a paraméternek az alapértelmezett értéke *std::out_of_range* legyen!

Egy lehetséges megoldás:

```
template<class T, class E = std::out_of_range>
class Stack {
    static const size_t siz = 234;
    T data[siz];
    size_t nElem;
public:
    Stack() :nElem(0){ }
    T& top();
    const T& top() const;
    void push(T a);
    void pop();
    bool empty() const { return nElem == 0; };
};
template<class T, class E>
void Stack<T, E>::push(T a) {
    if (nElem >= siz) throw E("Stack: empty");
    data[nElem++] = a;
}
template<class T, class E>
void Stack<T, E>::pop() {
    if (empty()) throw E("Stack: empty");
    --nElem;
}
```

---

2. feladat 4 p

Hozzon létre egy vermet egész számokkal, és egy másikat valós számokkal! Egy rövid kódrészlettel mutassa be az elkészült verem használatát egész számokra: A kódrészlet egy *std::stringstream* típusú objektumból olvasson be számokat, majd írja ki a szabványos kimenetre fordított sorrendben! A hibakezelést is mutassa be!

Egy lehetséges megoldás:

```
std::stringstream ss;
ss << "1 2 3 67 98 100";
try {
    Stack<int> st1;
    Stack<double> st2;
    int i;
    while (ss >> i)
        st1.push(i);
    while (true) {
        std::cout  << st1.top() << std::endl;
        st1.pop();
    }
} catch (std::exception &e) {
    std::cout << e.what() << std::endl;
}
```

BBBBBBBBBBBBBBBBBBBBBBBBBBBBB rendelkezésre álló idő:18 **perc**

Programozás alapja 2. 4. ellenőrző dolgozat. 2016.04.26. Kurz/**Terem**: G4/	Elért pontszám:
Név: **Neptun:**	

1. feladat 2 p

Tervezzen generikus halmazt (*Set*)! A tárolni kívánt halmaz számossága maximum 678. A halmaznak a laborgyakorlaton megvalósított *insert(), isElement(), size()* tagfüggvényén kívül legyen egy *empty()* tagfüggvénye is, ami igaz értékkel tér vissza, ha a halmaz üres. Sablonparaméterként vegyen át egy osztályt is, amit hiba estén (helyhiány) eldob. Ennek a paraméternek az alapértelmezett értéke *std::out_of_range* legyen!

Egy lehetséges megoldás:

```
template <typename T, class E = std::out_of_range>
class Set {
    static const size_t siz = 678;
    T data[siz];
    size_t nElem;
public:
    Set() :nElem(0) { }
    bool isElement(const T& e) const {
        for (size_t i = 0; i < nElem; i++)
            if (e == data[i]) return true;
        return false;
    }
    bool empty() const { return nElem == 0; };
    void insert(const T& e) {
        if (isElement(e)) return;
        if (nElem >= siz) throw E("Set: tele");
        data[nElem++] = e;
    }
};
```

---

2. feladat 5 p

Hozzon létre egy halmazt egész számokból, és egy másikat karakterekből! Egy rövid kódrészlettel mutassa be az elkészült halmaz használatát karakterekre. A kódrészlet egy *std::stringstream* típusú objektumból olvasson be egy karaktersorozatot, a karaktereket tegye halmazba, majd írja ki, hogy a 'A' betű benne van-e a halmazban! A hibakezelést is mutassa be!

Egy lehetséges megoldás:

```
std::stringstream ss;
ss << "Adam batyam pavava valt";
try {
    Set<char> set1;
    Set<int> set2;
    char ch;
    while (ss >> ch)
       set1.insert(ch);
    std::cout << std::boolalpha << set1.isElement('A') << std::endl;
} catch (std::exception &e) {
    std::cout << e.what() << std::endl;
}
```