Verem, sor, kivételkezelés, lengyelforma

ADATSZERKEZETEK ÉS ALGORITMUSOK
3. GYAKORLAT

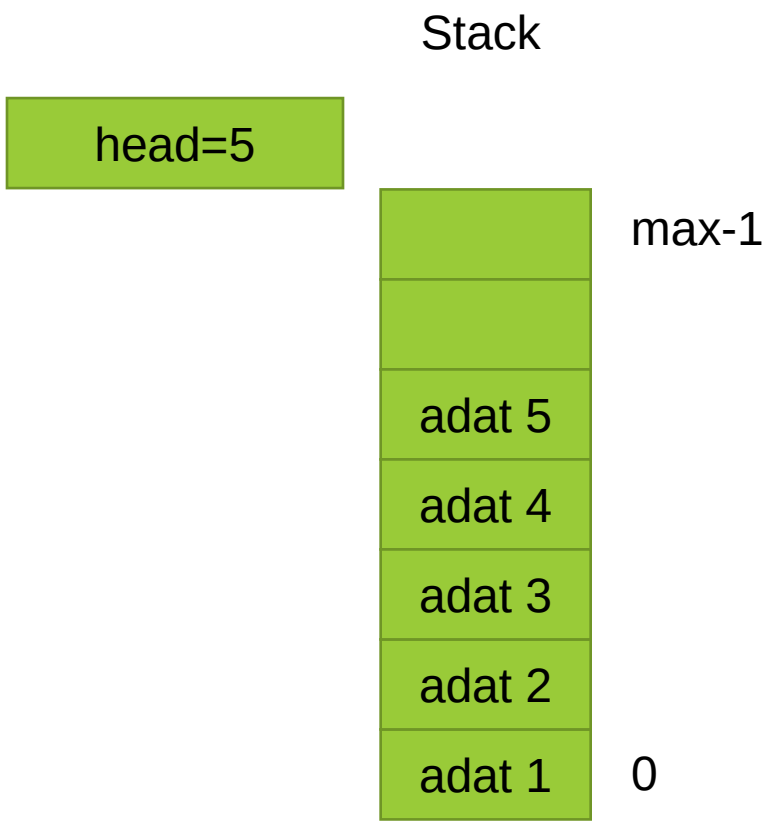
Verem – statikus megvalósítás

A verem egy `max` hosszú tömb és a `head` (**`int`**) direkt szorzata

(a tömb elemei `[0…max-1]` között indexeltek)

A `head` az első szabad pozíciót jelzi a tömbben, ahova beszúrhatunk értéket

$0 \leq head \leq max$



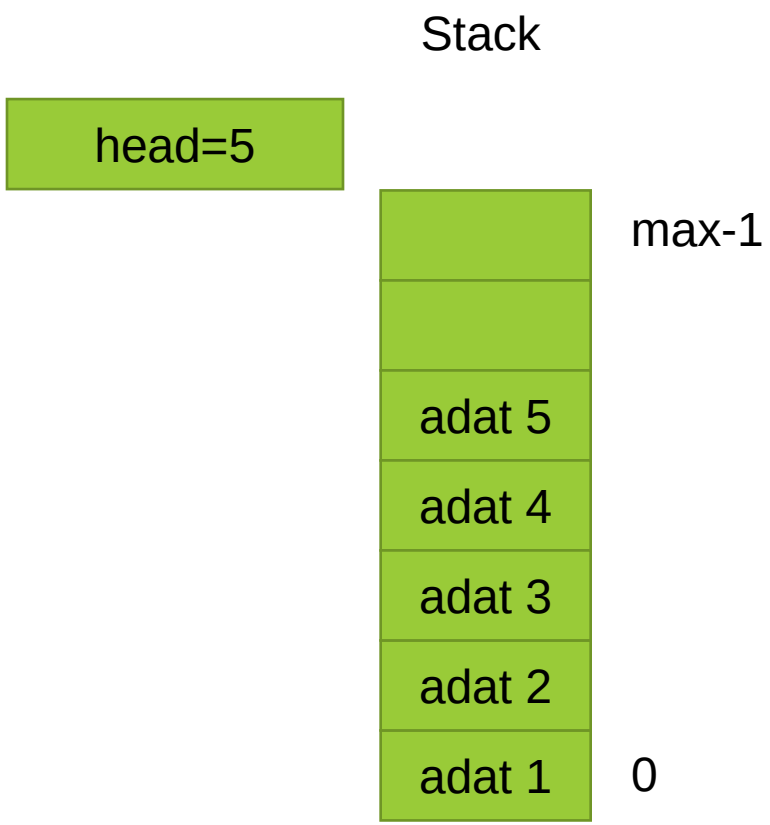
Verem – statikus megvalósítás

Megvalósítás osztály segítségével:

```
class Stack{
  static const int max = 7;
  int tomb[max];
  int head;

  Stack();        //konstruktor
  ~Stack();      //destruktor
  void push(int new_item);
  int pop();
  int top() const;
  bool isEmpty() const;

};
```



Kivételkezelés

A program futása során előfordulhatnak (előre látható) hibák, többek között a következő okokból kifolyólag:

- **Kódolási hiba**: valamit rosszul írtunk meg, így nem az történik, amit elgondoltunk
- **“Külső” hiba**: egy általunk igénybe vett szolgáltatás nem működik megfelelően (például szeretnénk írni a HDD-re, de elfogyott a hely)
- **Adathiba**: a program számára biztosított bemenő adatok nem felelnek meg az elvártnak (például megszegik az specifikáció előfeltételeit)

A kivétel a programnak az **elvárttól eltérő állapot**át leíró objektum.

Amennyiben a futás során valahol abnormális állapotot észlelünk, úgy ott egy kivételt „dobunk”, amivel jelezzük a problémát a külvilág (jó esetben a program többi része) felé.

- Például egy adatbeolvasó függvényben, ami csak egész számokat képes feldolgozni, string bemenet esetén dobunk egy kivételt, mivel ez a függvény előfeltételeinek nem felel meg, és így nem vagyunk felkészülve a bemenet értelmezésére.

A kivételbe elhelyezünk minden információt, ami a hiba kezeléséhez szükséges (például, hogy melyik beolvasott file hibás).

Kivételkezelés

A dobott kivételeket

- elkapjuk (ez történhet a program egy egészen távoli pontján is), és
- megpróbáljuk valamilyen értelmes módon lekezelni (felhasználva a benne kódolt információkat, pl. a hibás file nevét).

A lekezelés lehet

- újbóli próbálkozás (pl. kapcsolathiba esetén),
- a hiba okának megszüntetése (viszonylag ritkán tudjuk megtenni) vagy
- a hiba rögzítése és jelzése a felhasználónak (hibaüzenet, felugró ablak stb.).

A lekezelés során mindig figyelnünk kell arra, hogy az összes objektumot értelmes, konzisztens állapotba juttassuk (azaz megszüntessük az abnormális állapotot).

Kivételkezelés

```
int Verem::top(){
	if (IsEmpty()) {
		throw std::exception();
	}
	else {
		return tomb[head-1];
	}
}

int main(){
	try {
		Verem v;
		cout << "top():" << v.top();
	}
	catch (std::exception& e){
		 cout << "Hiba a verem futása közben!";
	}
	return 0;
}
```

Kivételkezelés

`throw` `[mit]` – kivétel kiváltása, dobása

`try` `{ [programkód] }` – a **`try`** blokkon belül található maga a programkód, melynek végrehajtása közben a kivétel kiváltódhat

`catch` `( [mit] ) { [hibakezelés] }` – a **`catch`** blokk csak és kizárólag a hibakezelést szolgálja

Ha dobsz egy kivételt, akkor a programkód futása ott megszakad

- azaz a **`try`**`{}` blokkon belüli, a kivétel kiváltása utáni utasítások nem fognak lefutni
- Fontos: destruktorból ne dobjunk kivételt, mert nem várt viselkedést érhetünk el!

A **`try`**`{}` blokk után találhatóak a **`catch`**`(){}` ágak, melyekből több is lehet egymás után.

Minden **`catch`** ág egy típusú kivételt (pl. `std::exception`-t) vagy annak leszármazottait kapja el.

Kivételkezelés

A **`catch`** ágak egymás után kerülnek kiértékelésre. Amennyiben a try blokkot követő első **`catch`** által várt kivétel típusa megegyezik a kiváltott kivétel típusával, úgy az fog lefutni (és a többi nem).

Amennyiben az első **`catch`** típusa nem kompatibilis, úgy az (esetlegesen meglévő) második **`catch`** kerül vizsgálatra, és így tovább.

Ha a **`try`** blokk után nincs megfelelő típust váró **`catch`**, úgy a kivétel egy szinttel feljebb kerül.

Ha nincs felsőbb szint (azaz try blokk valamelyik hívó függvényben), úgy a program futása megszakad.

Kivételkezelés

Többféleképpen történhet egy kivétel dobása és elkapása:

- A kivételobjektumra állított mutatóval. Ilyenkor megjelennek a memóriakezeléssel kapcsolatos problémák.
- Az objektum eldobásával és érték szerinti elkapásával. Ilyenkor a kivétel csonkulhat (ha ősosztályként kapunk el egy alosztályt), így információt veszthetünk.
- Az objektum eldobásával és referencia szerinti elkapásával. Ilyenkor nem kell foglalkozni a dinamikus memória kezelésével és információt sem vesztünk. Ez a javasolt eljárás.

Kivételkezelés – catch ellipsis

Lehetőség van mindenre illeszkedő **`catch`** ág használatára, ez típustól függetlenül minden dobott kivételt el fog kapni. C++-ban ezt az ún. catch ellipsis-el lehet elérni: **`catch`**`(...) { }`

Fontos a sorrend!

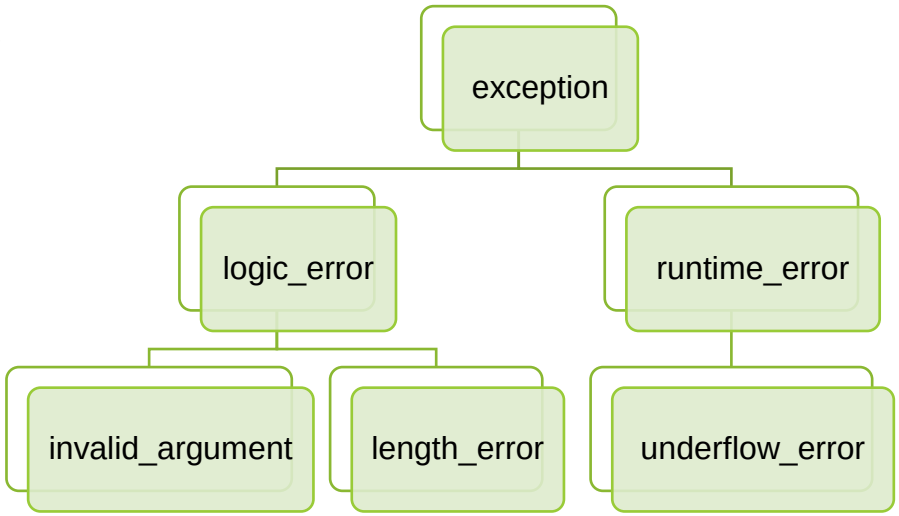
```
try {
        …
} catch (std::logic_error &) {
        std::cout << "logic error" << std::endl;
} catch (std::runtime_error &) {
        std::cout << "runtime error" << std::endl;
} catch (...) {
        std::cout << "other problem" << std::endl;
}
```

Kivételkezelés - Standard kivételek

A C++ Standard Library tartalmaz egy megkezdett kivétel osztály hierarchiát. Ez a stdexcept header fájlban található.

A későbbiekben érdemes a meglévő hierarchiát használni és bővíteni.

Az ábrán a hierarchia egy részlete látható.



Verem – statikus megoldás

Nyissátok meg a projektet, és implementáljátok a verem statikus megvalósítását úgy, hogy a static_stack_main.cpp helyesen lefusson!

A statikus verem 10 elemet tudjon letárolni!

Figyeljetek azokra az esetekre, ahol a feladatot nem lehet (specifikáció szerint) végrehajtani! (kivételkezelés)

Verem – dinamikus megvalósítás

A statikus megvalósítás korlátai:

- Ha betelik a verem, nem tudjuk további elemek tárolására használni.
- Ha túl nagy vermet hozunk létre, feleslegesen foglaljuk a memóriát.

A megoldás természetesen a dinamikus ábrázolás.

- A dinamikus megvalósítást láncolással oldjuk meg.
- Ez az jelenti, hogy létrehozunk egy osztályt a veremhez, amely tartalmazza az értéket, és egy pointerrel mutat a következő elemre.

```
class Node{
      public:
            int value;
            Node* pNext;
};
```

Verem – dinamikus megoldás G03F01

Nyissátok meg a projektet, és implementáljátok (új osztályként, új fájlba) a verem dinamikus megvalósítását úgy, hogy a dynamic_stack_main.cpp helyesen lefusson!

Figyeljetek azokra az esetekre, ahol a feladatot nem lehet (specifikáció szerint) végrehajtani! (kivételkezelés)

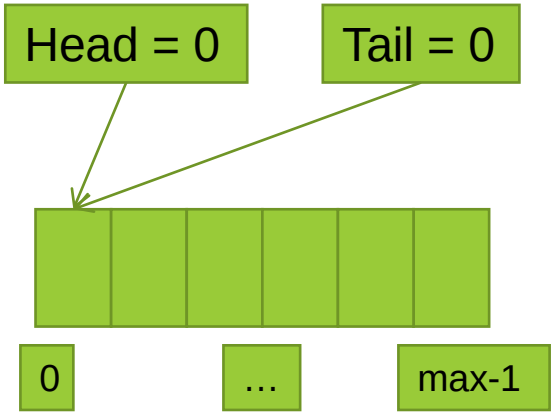
Sor – statikus megvalósítás

A sor elemeit egy statikusan létrehozott `max` méretű tömbbel, a `head` és `tail` mutatókkal, `empty` paraméterrel reprezentáljuk.

elemei: `tomb[0...max-1]`

A veremmel ellentétben a sornál a tömbnek mind a két végére szükségünk van, ezért azok helyét két változó, a head, és a tail fogják megadni.

A megvalósításhoz ciklikus ábrázolást használunk



Sor – statikus megvalósítás

Kezdetben a `head` és a `tail` a tömb ugyanazon elemének indexei.

- `head`: a tömb első elemének indexe
- `tail`: a tömb első szabad helyének indexe

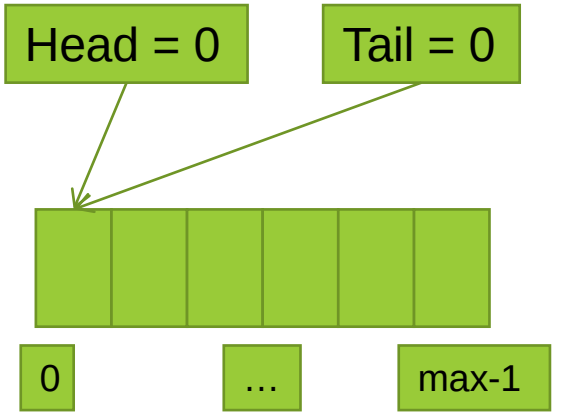
Ha a kettő egyenlő, akkor a sor vagy üres, vagy teli. A kettő megkülönböztetésére használunk egy plusz `empty` nevű változót, amit ha elemet teszünk be hamisra állítjuk, ha pedig egy elemet veszünk ki, és ettől elfogynak az elemek, akkor igazra állítjuk.

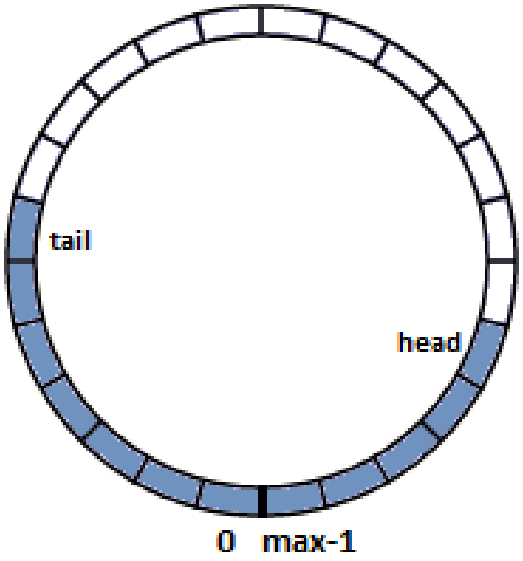
Sor – statikus megvalósítás

Megvalósítás osztály segítségével:

```
class Queue{
 static const int max = 10;
 int tomb[max];
 int head, tail;
 bool empty;


 Queue();      //konstruktor
 ~Queue();     //destruktor
 void in(int new_item);
 int out();
 int first() const;
 bool isEmpty() const;
 bool isFull() const;
};
```





Sor – statikus G03F02

Nyissátok meg a projektet, és (új osztályként, új fájlba) implementáljátok, írjátok meg a sor statikus megvalósítását úgy, hogy a *static_queue_main.cpp* helyesen lefusson!

Figyeljetek azokra az esetekre, ahol a feladatot nem lehet (specifikáció szerint) végrehajtani! (kivételkezelés)

Sor – dinamikus megvalósítás

A statikus megvalósítás korlátai:

- Ha betelik a sor, nem tudjuk további elemek tárolására használni.
- Ha túl nagy sort hozunk létre, feleslegesen foglaljuk a memóriát.

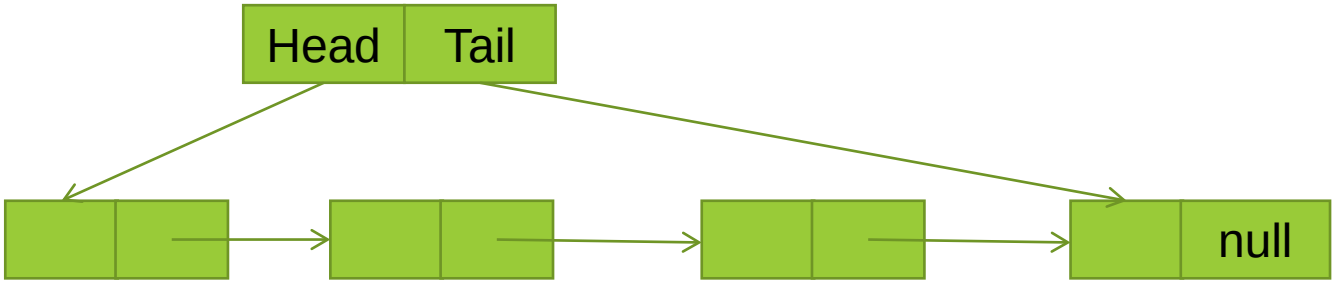
A megoldás természetesen a dinamikus ábrázolás.

- A dinamikus megvalósítást láncolással oldjuk meg.
- Ez az jelenti, hogy létrehozunk egy osztályt a sorhoz, amely tartalmazza az értéket, és egy pointerrel mutat a következő elemre.

```
class Node{
      public:
            int value;
            Node* pNext;
};
```

Sor – dinamikus megvalósítás

A head és tail változók ebben az esetben nem indexek lesznek, hanem mutatók, mégpedig az első ill. az utolsó elemre mutatók. (Node*)



Sor – dinamikus megvalósítás

Nyissátok meg a projektet, és (új osztályként, új fájlba) implementáljátok, írjátok meg a sor dinamikus megvalósítását úgy, hogy a dynamic_queue_main.cpp helyesen lefusson!

Figyeljetek azokra az esetekre, ahol a feladatot nem lehet (specifikáció szerint) végrehajtani! (kivételkezelés)

Gyakorlófeladat G03F03

Lengyelforma gyakorlása:

- Nyissátok meg a projektet, és (új osztályként, új fájlba) implementáljátok, írjátok meg a verem dinamikus megvalósítását úgy, hogy a dynamic_stack_main.cpp helyesen lefusson!
  - Figyeljetek azokra az esetekre, ahol a feladatot nem lehet (specifikáció szerint) végrehajtani! (kivételkezelés)

Írjátok át a VEREM és a SOR dinamikus implementációját is úgy, hogy azok char típusú elemeket tudjanak tárolni

Ezeket felhasználva írjátok meg az egyszerűsített lengyelformára hozó algoritmust

- Tipp: Egy jegyű számokkal dolgozzatok!

Gyakorlófeladat G03F04

Adott egy fájl amiben titkosírással elrejtettünk egy üzenetet. Verem és sor használatával, fejtsd vissza mit titkosítottunk.

A fájlban #, & és @ szimbólumok valamint a következő karakterek szerepelhetnek: a-z, A-Z, 0-9

Minden karakter két szimbólum közé esik.

A megfejtés azon karakterek (a fájlban szereplésüknek megfelelő sorrendben) összeolvasása amelyeket két azonos szimbólum határol.

Példa a fájlból:

- #a&#I@#Q@&A##v@@7#@D&#y&@k&@8##W@&V@
- &i@#H#@V&@q&@s#@D#@q&@F##s&&v##e@#L&#a&
- &7##p&&J@#i&&n@@R##M@@f#&j@&V##Y&&T@&4##l

Az eredményt írd ki a konzolra és egy fájlba.

Gyakorlófeladat G03F05

Készítsd el a deque implementációját a statikus sor módosításával. A deque kétirányú sor, tehát mindkét végére tehetünk be és vehetünk ki onnan elemet, de a középső elemeket nem érjük el.

A deque működésének bemutatásához készíts egy piros-zöld papucs kártyajátékot szimuláló programot. (Ez a piros papucs nevű kártyajáték kicsit módosított változata.)

A játékot két játékos játssza egy pakli (32 lapos) magyar kártyával. A kártyalapok értéke nem, csak a színe (piros, zöld, tök, makk) számít.

A lapokat először szétosztják maguk között két megkevert, egyenlő méretű pakliba (figyelj rá, hogy minden színből 8 db legyen). Ezután felváltva pakolnak kártyát a saját paklijuk **tetejéről** az asztalon levő kupac **tetejére**.

(folyt. köv.)

Gyakorlófeladat (folyt.)

Ha valaki *pirosat* rakott, akkor a másik felveszi az asztalon levő teljes paklit, és sorban elhelyezi a saját paklija **aljára**, **felülről lefelé**. Ha valaki *zöldet* rakott, akkor szintén a másik veszi fel a teljes paklit, de fordított sorrendben (**alulról felfelé**) helyezi el a saját paklija **aljára**.

A játéknak vége, ha elfogyott valamelyik játékos paklija, és ő következik soron; ekkor ő nyert (azaz lehet „visszahívni” 1 körig). Ha 200 lépés után sem fogyott el senkinek a kártyája, akkor a játékosok megunják, és kiegyeznek egy döntetlenben.

A megvalósításban a játékosok kezében levő paklikat egy-egy sorral, az asztalon levő paklit pedig egy kétirányú sorral valósítsd meg! A kártyák számértékét nem szükséges figyelembe venned.

Házi feladat

Írj a hallgatók beiratkozását szimuláló programot!

A beiratkozás során a hallgatók érkezési sorrendben, az éppen aktuálisan szabadon lévő TO-s előadóknál iratkoznak be: Bokhara-Rigli Etánál, Szini-Társ Hant Évánál, és Rumba-Rigli Leánál.

A program feladata, hogy feljegyezze a beiratkozás sorrendjét a három TO-s előadónál.

A hallgatók listája érkezési sorrendben egy in.txt nevű fájlban van elmentve, innen kerülnek be a sorba. Mindenkinek véletlenszerűen 1-5 percig tart a beiratkozás.

Az out.txt-ben mentsd el, hogy hogyan zajlott a beiratkozás. Minden hallgató külön sorba kerüljön, és szerepeljen az is, hogy kinél, és hány percet töltött.

Pl.: Kis Eufrozina Rumba-Rigli Leanal 5 perc alatt iratkozott be.