ADATSZERKEZETEK ÉS ALGORITMUSOK

Kivételkezelés, Template, Verem

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!

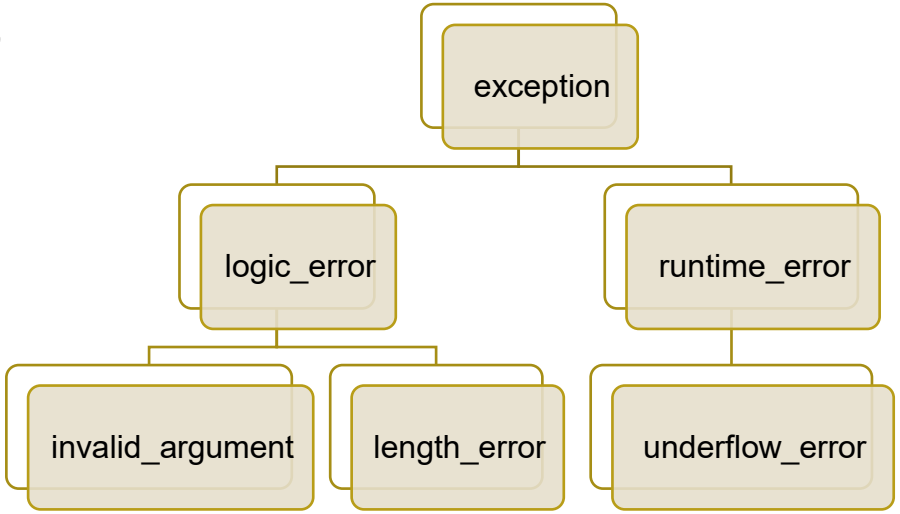
```
try {
        …
} catch (std::logic_error &) {
        std::cout << "logic error" << std::endl;
} catch (std::runtime_error &) {
        std::cout << "runtime error" << std::endl;
} catch (...) {
        std::cout << "other problem" << std::endl;
}
```

Kivételkezelés - Standard kivételek

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

A későbbiekben érdemes a meglévő hierarchiát használni és bővíteni.

Az ábrán a hierarchia egy részlete látható.



Kivételkezelés - Gyakorlás

Töltsük le és nyissuk meg a 03_exception projektet CLion-ban.

Implementáljuk a custom_exception.h és custom_thrower.h hiányzó részeit a kommentek alapján.

Futtassuk le a main.cpp-t és figyeljük meg a kivételek és elkapásuk viselkedését.

Sablon (Template)

Egy függvény vagy osztály definíciója során megtehetjük, hogy a paraméterek, változók vagy épp a visszatérési érték típusát nem egy előre ismert típusként adjuk meg, hanem egy típust jelző változóval helyettesítjük (az ún. típusparaméter).

Tehát az adott kódot csak egyszer kell megírnunk, és az több típussal is működni fog sablonok használatával.

Nevezéktan:

- Osztálytemplate: típusparaméterrel (template paraméterrel) ellátott osztály (típus)
- Függvénytemplate: típusparaméterrel (template paraméterrel) ellátott függvény
- (Template paraméter nem csak típus lehet.)

Template függvény – példa

```
void swap(int &a, int &b)
{
  int c = a;
  a = b;
  b = c;
}

…

// Használat:
int a(3), b(4);
swap(a, b);
```

```
template <typename T>
void swap(T &a, T &b) {
  T c = a;
  a = b;
  b = c;
}

…

// Használat:
int a(3), b(4);
swap<int>(a, b);
swap(a, b);
//automatikus

//behelyettesítés
```

Template

Template-tel paraméterezett kód esetén a fordító a használat alkalmával készíti el az adott típushoz tartozó tárgykódot fordításkor.

Azaz a swap<int>(a, b); és a swap<long>(a, b); hívások során két külön verzió készül a swap függvény kódjából.

Emiatt minden template-tel paraméterezett kódnak a header file-ban kell lennie.

Template – osztály

Az osztálytemplate-k szintaxisa a következő:

```
template <typename T>
class A {
  T data;
  T f();
};
```

```
template <typename T>
T A<T>::f() {
  // ...
}
```

Amint látható, a függvények megvalósításánál is jeleznünk kell a template paramétert (ha osztályon kívül van a megvalósítás).

A megvalósításnak az azt hívó kód számára ugyanúgy láthatónak kell lenni (általában beleírják ugyanabba a header fájlba, mint ahol a template deklarációja van).

A két relációs jel között a **`typename`** és a **`class`** kulcsszó általában ugyanazt jelenti

Verem – fix méretű megvalósítás

Nyissátok meg a 03_fixed_stack projektet, és implementáljátok a verem fix méretű megvalósítását úgy, hogy a fixed_stack_main.cpp helyesen lefusson!

A fix méretű verem 10 elemet tudjon letárolni!

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

Második lépésként templatesítsétek a verem implementációt

Verem – változtatható méretű megvalósítás

A fix méretű 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 változtatható méretű (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;
};
```

Házi feladat – Dinamikus méretű verem (+template, +pointer, +exception)

Töltsd le a khf2 projektet és készítsd el a hiányzó implementációt a pstack.h-ban úgy, hogy:

- A verem dinamikus méretű legyen, de legyen lehetőség felső korlát megadására létrehozáskor
- Template-ezhető
- A megadott típusra mutató pointereket tároljon
- Dobjon kivételt alul- és felülcsorduláskor
- Rendelkezzen egy reverse() metódussal, amely megfordítja a veremeben tárolt elemek sorrendjét
- Plusz teszteket írni ér sőt javallott

Verem - stl megvalósítás G02F01

Hasonlítsuk össze az std::stack<>-el

- Kód szempontjából
- Kivételek kezelése

Gyakorló feladat G02F02

- Hozzatok létre egy absztrakt „Alakzat” osztályt, ennek legyen 2 metódusa, a kerulet_szamol() és a terulet_szamol()
- Hozzatok létre egy „Haromszog”, egy „Teglalap”, és egy „Kor” osztályt, amik az alakzat osztályból származnak.
- A „Teglalap” osztályból származzon le egy „Negyzet” osztály is.
- A kört leszámítva tartalmazzák az osztályok az oldalaik hosszát, a kör pedig egy sugarat.
- A main.cpp-ben hozzatok létre egy „Alakzat” pointer, vagy referencia tömböt, amit töltsetek fel tetszőleges alakzatokkal, és keressétek meg benne a legkisebb területűt, és a legnagyobb kerületűt!
- Írjatok olyan operátort minden alakzathoz, ami kiírja a az alakzat nevét, és paramétereit a „cout”-ra! Erre is csináljatok néhány próbát!

Gyakorló feladat G02F03

Írjunk egy iskolát modellező alkalmazást!

- Először is az iskolába járnak emberek, akiknek van nevük, születési évük és nemük.
- Az iskolában tanítanak tantárgyakat, amelyeknek van nevük, leírásuk és hozzá vannak rendelve évfolyamok, amikor ezeket oktatni kell.
- Az iskolába járó emberek között vannak tanulók, akik egy osztályba tartoznak, akiknek van egy tanulmányi átlaguk, és akiknél el kell tárolni a szülő/nevelő elérhetőségeit.
- Az iskolában tanítanak tanárok, akikhez bizonyos tárgyak vannak hozzárendelve. A tanárok között vannak osztályfőnökök, akiknek van osztályuk. A tanároknak van heti óraszámuk, órabérük, ami alapján a havi fizetésük kiszámolható.
- A tanárok között van egy igazgató, akinek a fizetése nem csak az órabér és a heti óraszám alapján képződik (van valamifajta plusz fizetés).
- Végül van az iskola, amiben vannak tanulók, tanárok, igazgató, aminek van egy címe, neve és ahol kiszámolható, hogy havonta összesen mennyi fizetést kell kiosztani.

Gyakorló feladat G02F04

Tervezzünk meg és írjunk meg egy alkalmazott osztályt!

Útmutatás:

- Gondold át, hogy egy alkalmazottat egy tetszőleges vállalatnál milyen adatokkal lehet reprezentálni.
- Ha ez megvan, gondold át azt, hogy mik lehetnek azok az eljárások/műveletek, amiket ehhez az alkalmazott osztályhoz hozzá lehetne kapcsolni. Gondold át, hogy a meglévő adattagokon milyen értelmes műveleteket/lekérdezéseket lehetne végrehajtani.
- Gondold át, hogy a fentiek összességéből melyek azok, amelyek minden alkalmazott esetén ugyanazok/ugyanazt az eredményt/hatást érik el, és melyek azok, amelyek minden alkalmazott esetén különbözőek.
- Majd gondold át, hogy mi az, ami hozzátartozna az osztály publikus interfészéhez, és mi az, amit érdemes elrejteni.
- Ha mindez megvan, a megfelelő nyelvi elemeket felhasználva írd meg az alkalmazott osztályt!

Gyakorló feladat G02F04 (folyt.)

Feladat: A megírt Alkalmazott osztályból leszármaztatva valósítsunk meg egy Főnök osztályt!

Útmutatás:

- Melyek azok a változók, amelyeket célszerű lenne elfedni?
- Milyen új változókat lehetne bevezetni?
- Melyek azok a metódusok, melyeket felül kéne definiálni?
- Milyen új metódusokat kellene megírni?

Majd ezután írj egy olyan main függvényt, melyben kipróbálod a polimorfizmus és a dinamikus kötés lehetőségeit!

Gyakorló feladat G02F06

Egy városban reggel mindenki megy valahova. Szimuláld a városban élők utazását és készíts róla statisztikát.

- A városban van nyugdíjas, felnőtt és diák. Generálj mindből 1 - 80 között valamennyit.
- Legyen koruk is, tanuló 0 - 25, felnőtt 26 - 61, nyugdíjas 62 - 80.
- Utazhatnak vonattal, autóval, taxival és biciklivel. Véletlenszerűen válaszd ki, hogy az egyes emberek melyik tömegközlekedési eszközt használják.
- Egy ember 15 - 45 km-et utazik, ez is véletlenszerű.
- A járműveknek van kapacitása: vonat 60, autó 5, taxi 4, bicikli 1. Ezeket töltsd fel emberekkel úgy, hogy a járművek mindig maximum távra (45 km) mennek. (ha 2 nyugdíjas 20 km-t megy vonattal, és 3 diák 40 km-t vonattal, ők utazhatnak egy járművön)
- A járműveknek van kilométerenkénti díja. Vonat 18 Ft/km, autó 36 Ft/km, taxi 50 Ft/km és biciklinél elégetett kalória van 33 kalória/km.

Gyakorló feladat G02F06

- Továbbá vannak egyedi tulajdonságok.
- Vonat: típus („Stadler Flirt", „Ganz V43", „Siemens Desiro", „Mallard"), pálya szám (1000, 1001, 1002, 1003). Nem kell minden vonatot megalkotni, elég annyit amennyi éppen használatban van. Értsd, ha csak 100 ember utazik vonattal elég 2 vonatot példányosítani. (Az első vonat így „Stadler Flirt" és 1000 a pályaszáma, a második vonat „Ganz V43" és 1001 a pályaszáma stb.)
- Autó: Mikor gyártották (1990 - 2010).
- Taxi: Mikor gyártották (1990 - 2010), és egy száma (1- 100) véletlenszerűen.
- Bicikli: méret (24 - 32).
- Statisztikák, amiket el kell készíteni:
  - Konzolra: mennyit költöttek összesen az emberek autóra, taxira vonatra, és mind háromra összesen, illetve összesen mennyi kalóriát égettek el. Milyen nevű és pályaszámú vonatok mentek, átlag bicikli méret és autók kora, taxik kora, taxik sorszáma.
  - Fájlba: minden emberről statisztikák így: 10. ember, aki 62 éves, 29 km-et utazik és vonattal.

Megjegyzés: a maximum ponthoz mindenképpen használni kell: absztrakt osztály, konstansok, polimorfizmus, dinamikus kötés, referencia, pointer.

Gyakorló feladat G02F07

- Készíts egy Tárgy osztályt. Minden tárgynak van súlya, és kérésre ki tudja írni, hogy mi az a tárgy, és milyen nehéz.
- Készíts három tetszőleges konkrét alosztályt (pl Toll), amelyeknek mind legalább egy további tulajdonsága van (pl. Toll esetében a színe).
- Készíts egy Táskát, mely egy bizonyos mennyiségű – létrehozáskor megadott – súlyt képes befogadni
  - A Táskába véletlen módon tegyünk véletlen tulajdonságú Tárgyakat, amíg az meg nem telik
  - A Táska legyen képes felsorolni, hogy milyen tárgyak vannak benne, ha megtelt a táska, ezt tegyük is meg.

Megjegyzés: a maximum ponthoz mindenképpen használni kell: absztrakt osztály, dinamikus tömb, polimorfizmus, dinamikus kötés, referencia, pointer.