12. LABOR

TÍPUSKONVERZIÓK KEZELÉSE

Hallgatónak: általános információk

Az összes feladat hibátlan megoldására 1 iMSc pont kapható.

Kötelező feladatok

1. Fraction osztály konverziója

Írj egy osztályt, ami egy törtszámot reprezentál (*Fraction*)! Az osztálynak legyen egy egész és egy tört része (pl. 3 és 1/4). Az egész részt egy int típusú adattaggal, a tört részt pedig két int típusú adattaggal valósítsd meg. Az osztálynak legyen konstruktora, ami megkapja egy törtszám egész és tört részeit.

1. Írj konverziós operátort, ami az osztályt átkonvertálja egy double típusra, ami az egész rész és a tört rész megfelelő tizedesjegyeit reprezentálja (pl. 3 és 1/4 tizedesjegyei: 3.25).
2. Írj továbbá egy konverziós operátort, ami egy std::string-re konvertálja az osztályt, ami megadja a törtszámot szöveges formában (pl. "3 1/4").
3. Írj egy függvényt, ami két ilyen törtszámot kap paraméterként és összeadja őket, majd visszatér az eredménnyel.
4. Írj konverziós konstruktort, amely double értékből Fraction példányt készít EPSILON pontossággal.

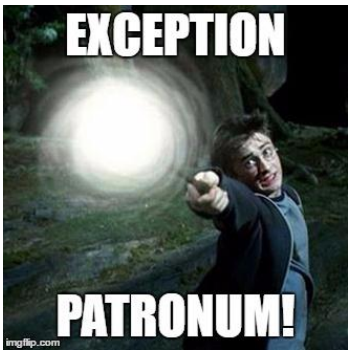
Teszteld a megoldást a megadott main függvény sorainak kikommentelésével.

2. Fraction osztály kivétele

5. Fraction osztályt ne lehessen létrehozni úgy, hogy a tört részben nulla legyen a nevező. Konstruktorból szabad kivételt dobni! Írd meg ehhez a megfelelő kódrészletet, a `FractionException` felhasználásával. A kivételt a Fraction osztályt felhasználó kódban, vagyis a main függvényben kapjuk el. Tedd a teljes main függvény tartalmat védett blokkba. Próbáld meg a kivételt többféle módon elkapni: referenciával, ősosztály referenciával, vagy mindent elkapó ellipszissel. Hogyan számít a sorrend, ha minden catch ágat megírsz?
6. A kivétel a védett blokkon belül bármilyen hívási mélységben történhet. Próbáld meg a reciprocal függvényt a nullás értékkel használni. Hol fut le a catch ág?
   Hogyan lehet kivédeni, hogy a `(reciprocal(0.0)` automatikus konverzió megtörténjen?

Önálló feladatok

3. ExceptionPatronum!



1. ábra Magic

Nyisd meg a mellékelt *ExceptionPatronum* solutiont. Feladatod, hogy a *test.cpp* állomány függvényeiben valósíts meg kivételkezelést, valamint írj saját kivétel osztályt.

- A kivételeket dobják az „f”-fel kezdődő függvények.
- A kivételeket a main()-ben kapd el úgy, hogy az f1, f2, f3, f4, bonus függvények külön-külön is le tudjanak futni!
- Ne használj catch(…)-ot és catch(const exception& ex)-et!
- Az f1, f2, f4 függvényekben használj C++-ba beépített kivételeket!
- Az f3 feladathoz hozz létre egy saját kivétel osztályt:
  - Legyen parametrizált osztály.
    - Egy bármilyen típusú konstans referencia paramétert kapjon meg a konstruktorában, amit tárolj is el, továbbá legyen gettere is!
    - Öröklődjön egy másik exception osztályból (de ne az std::exception-ből!).
  - A neve legyen `element_not_found` (element_not_found.hpp-ben definiálva).
  - Oldd meg, hogy az ősosztály what() függvényét meghívva az `"element not found"` üzenetet kapjuk vissza.
  - A main()-ben való elkapáskor írasd ki a konstruktorban átadott elemet is (feltételezhetjük, hogy van rajta értelmezve operator<<).

4. ExceptionHandling: exception1.cpp

Nyisd meg az *ExceptionHandling* solution *exception1.cpp* állományát, majd végezd el a következőket:

1. Rakj egy töréspontot (break point) a *main()* függvény elejére!
2. Debuggold végig a kódban található három variációt (1.1, 1.2, 1.3)!
3. Figyeld meg, hogy hogyan adódik át a vezérlés a *catch* blokkokra!
4. Hol folytatódik a vezérlés egy kivétel elkapása esetén, és hol folytatódik egyébként?
5. Mikor melyik *break pointon* megy át a vezérlés?
6. Tedd megjegyzéssé a kódot úgy, hogy csak a *cout*ot tartalmazó sorok és az első *throw*-t tartalmazó sor legyen érvényes!
7. Hova kerül a vezérlés a *throw* utasításról a jelzések tükrében?
8. Hogyan reagál az operációs rendszer a programon belül el nem kapott kivételre?
9. Miért szoktuk a *main()* függvényt körülvenni az alábbihoz hasonló *try-catch* blokkokkal, és a main függvény összes "hasznos" kódját a *try* blokkon belül elhelyezni?
10. Próbáld ki a jelenséget *Release* üzemmódban is! (Ezt látja a felhasználó).

5. ExceptionHandling solution: exception2.cpp

1. Vizsgáld meg, hogyan kell saját kivétel osztályt származtatni az *exception* alaposztályból! Melyek az egyes lépések?
2. Figyeld meg a *main()* függvény felépítését a kivételkezelés szemszögéből! Ezt általánosságban így szokás csinálni, érdemes mindig így feltérképezni.
3. A *main()* függvény elejétől debuggold végig a programot! Amikor kivételt dobsz, a vermet (*stack*) vissza kell görgetni és a lokális objektumok destruktorait meg kell hívni. Helyezz egy töréspontot a *dummy* osztály destruktorára és vizsgáld meg, tényleg meghívódik-e!

4. A kivételeket mindig referencia szerint kapd el! Így nem kell felszabadítást végezni és nem lesz *„slicing on-the-fly”*.
5. Alakítsd át a *start()* függvényt, hogy csak az alábbi két utasítás maradjon benne:
   - `dummy d2(2); print(nullptr);`
   - Vagyis ne kezeld a keletkező kivételt, dobódjon tovább a *main()* függvény *catch* blokkjaihoz.
6. Igaz-e hogy nemcsak közvetlenül azon a függvény lokális objektumainak a destruktora hívódik meg, hanem az összes függvényé, amit vissza kell fejtenünk, hogy elérjük a kívánt *catch* blokkot?
   - (A *dummy2* egy ilyen objektum, hiszen nem a *print()* függvényben van, ahol a kivételt dobtad, hanem eggyel kijjebb a, a *start()* függvényben. Azonban ezt is vissza kell fejteni, hiszen a kivételt a *main()* függvényben kapod el.)
7. Vedd észre, hogy mivel a *main()* függvényben az exceptiont referencia szerint kapod el, ezért ott van alatta a dobott *null_pointer_exception*! (Erről a felüldefiniált virtuális *what()* függvény meghívódása biztosít.) Kapd el most a kivételt *const* referencia helyett érték szerint.
   - Melyik függvény hívódik meg?
   - Mit ad vissza a *what()* függvény?
8. Gyakran megesik, hogy elkapunk egy kivételt, mert egy speciális esetet szeretnénk kezelni és ha rájövünk, hogy ez nem az az eset, tovább kell dobnunk. Javítsd ki a *print()* függvényben a hibakódot -1-ről -2-re (a *null_pointer_exception* konstruktora), és debuggold a programot!

6. ExceptionHandling solution: exception3.cpp

- Futtasd le a programot! A három *catch* blokk közül melyik kapja el a kivételt?
- Cseréld meg a két első *catch* blokkot!
  - Igaz-e, hogy ha két *catch* blokk is elkaphatná a kivételt, akkor a sorrendjük számít? Ha igaz, akkor hogyan kell elhelyezni a *catch* blokkokat?
    - **Laborvezetőnek:** a legspeciálisabbtól a legáltalánosabbig haladva kell megadni a *catch* blokkokat. (Ez olyannyira igaz, hogy *warning*ot eredményez, és ha nem a *catch(…)* az utolsó, akkor fordítási idejű hibaüzenetet kapunk).

Gyakorlófeladatok

- Kivételkezelési hibák
- Saját kivétel könyvtárkezelő keretrendszerhez
- Mátrix operátorai kivételekkel
- MathException kivétel készítése