Házi feladat

Programozás alapjai 2.

Specifikáció a skeleton programhoz

Grób László
EAS7U0

2025.03.26.

1. Feladat ötlet a tantárgy portáljáról:

1.1. Árverés

Készítsen objektummodellt árverés modellezésére! A "Vásárló" objektumok miután megkapták a legutolsó árat, véletlenszerűen licitálnak, vagy nem licitálnak. A "Szervező" objektum kiáltja ki az árat, ill. megfelelő licit esetén leüti (eladja) azt.
Készítsen egyszerű programot, amelyben egy "Szervező" objektum több (~30) vásárló objektum részvételével elárverez egy konkrét árut!

2. A program funkcionális specifikációja:

2.1. Bemenetek

Az adott tárgy neve, árverezéséhez tartozó kiinduló ár, és minimum licitlépcső. Az adott árverésen részt vevő vásárlók száma. A vásárlók milyen arányban tartalmazzanak különböző típusú (pl. agresszív és visszafogott) vásárlókat.

2.2. Kimenetek

Az árverés során megtett licitek listája. Az árverés győztese. Az eladott áru végleges ára.

2.3. A program működésének feltételei

A való élethez hasonlóan (Angol/Japán típusú árverés) a tárgyak csak akkor kerülnek eladásra, ha legalább egy vásárló licitál rájuk. Az aukció körökre bontott, hogy elkerülhetőek legyenek a konkurrenciával járó implementációs nehézségek. Minden licitálónak az adott körben legalább a jelenlegi ár + a licitlépcső értékben kell licitálnia. A kör végén az új ár az adott körben leadott legmagasabb licit értéke lesz. Ha egy vevő az adott körben nem licitál, akkor kiesik és a későbbi körökben sem licitálhat, ezzel

megelőzve az árverések túlzott elhúzódását. Az árverés addig folytatódik, amíg nem marad további licitáló, és a nyertes az utolsó legmagasabb érvényes licitáló.

2.4. Környezeti feltételek

A programnak nem készül grafikus felület, konzolablakban fut. A feladathoz egy olyan tesztprogramot készítek, ami képes egy adott árverést, vagyis több tárgy eladását szimulálni, ugyanazon vevőkkel: A tesztprogram képes file-ból beolvasni az eladásra kínált tárgyakat és azok kiindulási árait és licitlépcsőjét, illetve kiírni a tárgyakat, eladási árakkal és vevőkkel. A többi paraméter (a vásárlók száma és eloszlása) állandó az aukció során.

2.5. Egyéb megkötések, egyértelműsítések

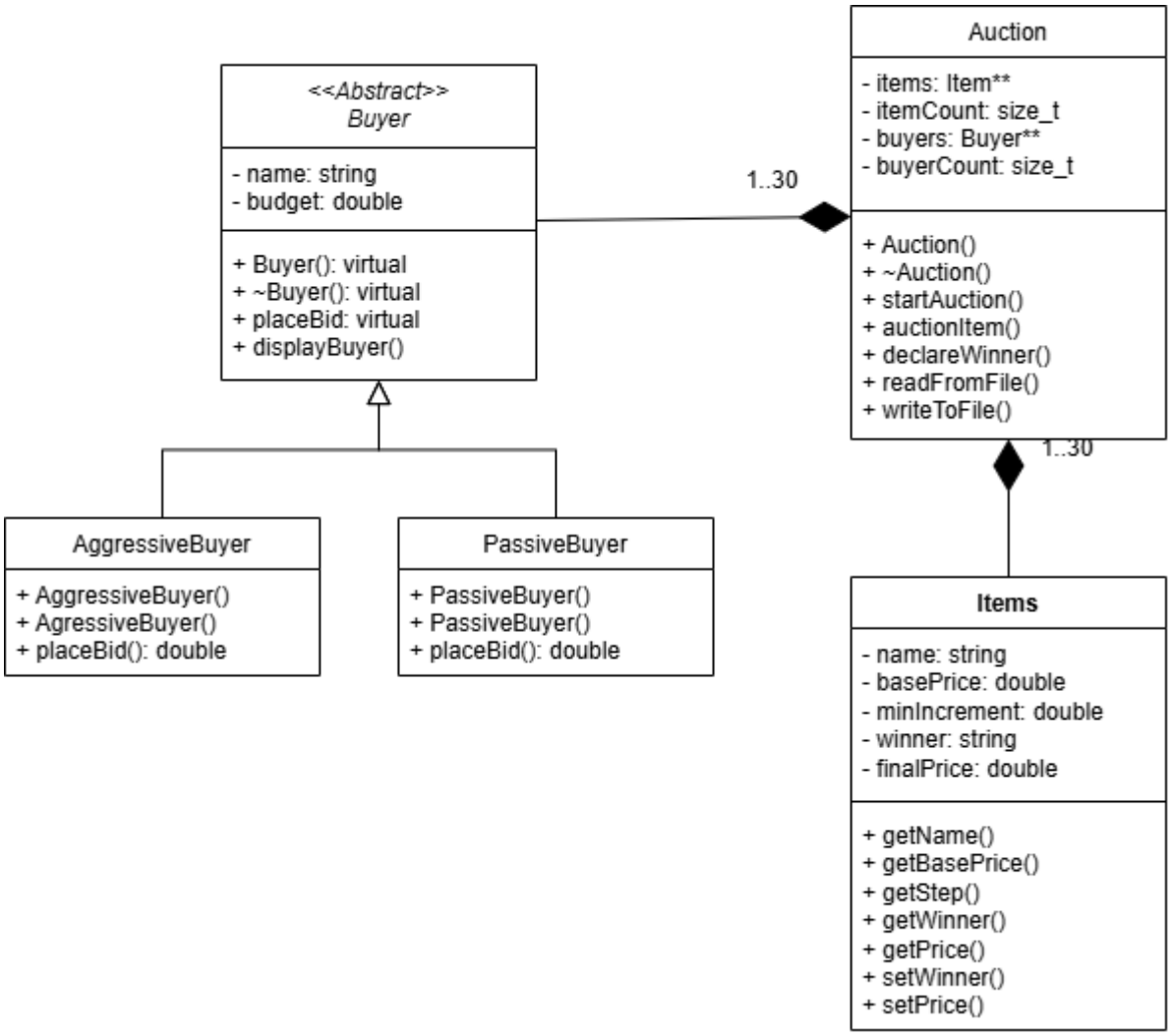
A program modellez agresszívebb és visszafogottabb vevőket is, akik más viselkedéssel licitálnak. A tesztprogramban lehetőség van limitálni a körök számát, a túl hosszú árverés elkerülése érdekében. A tesztprogram futtatásánál egy rögzített seed-del lehetőség van determinisztikus tesztelésre.

2.6. Kiegészítés a laborvezetővel konzultálva

A programban nem használok stl tárolót, kivéve std::stringet.

3. A program terve:

3.1. A programot leíró UML diagramm bemutatása:



3.2. A fontosabb algoritmusok vázlatos leírása:

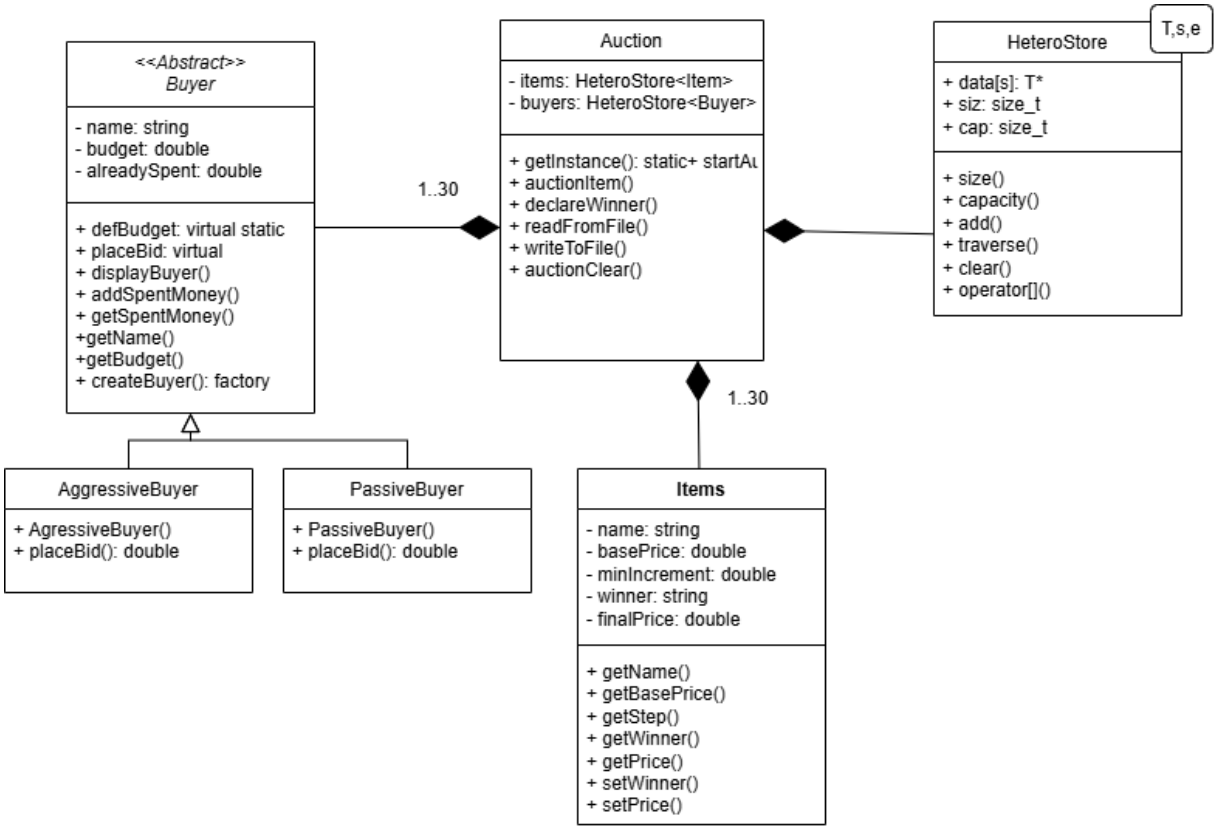
Az árverést lebonyolító Auction class a readFrom file fügvénnyel olvassa be a szükséges információkat a mellékelt fileból, ami tartalmazza az eladásra kínált termékek listáját. Ezután generál vevőket, és egyesével végigiterál az eladásra kínált termékeken, és elárverezi őket az auctionItem függvény segítségével. A függvény egy loopban addig fut amíg sikerül eladnia a terméket. Ezt egy while loopon kersztül valósítja meg egy boolean

változóhoz kötve(while(!sold)). A loopon belül minden résztvevő vásárlónak meghívódik a placeBid függvénye, ami megkapja a termék aktuális árát, és licitlépcsőjét, és visszatér egy licittel. Amennyiben nem licitál az adott körben a vásárló, kiesik és a következő körökben már nem lesz lehetősége licitet leadni. Az adott loopon belüli legnagyobb licit lesz az új ár, ami a következő loopban, mint alapár szerepelni fog. Amennyiben nem licitál az adott körben senki, vagy lejár a beállított kör limit, akkor az aktuális legmagasabbat licitálót beírja a setWinner függvény segítségével a termék adataihoz. Ezt megismétli minden egyes termékkel. Ezután a declareWinners függvény kiírja a nyerteseket és az eladási árat fileba is a writeToFile függvény segítségével.

Az adott vásárló placeBid függvénye az alapár, licitlépcső, és saját pénzösszes függvényében eldönti, hogy licitál-e, és hogy mennyit (az agresszívebb vevő nagyobb eséllyel, és arányosan többet licitál). Ezeket a rand() függvény segítségével szimulálja, amit a tesztprogramban lehet seedelni, hogy determinisztikusan működjön a szimuláció.

4. Felmerülő módosítások skeleton program írása során:

4.1. A módosított UML diagramm:



4.2. Változások az algoritmusokban:

Az auctionItem függvény a korábban említettekkel szemben nem while loopot használ az eladáshoz, hanem 2 beágyazott for loopot, a külsőt az aukció köreire, a belsőt a vevőkön való végigiteráláshoz.

4.3 A skeleton program felépítése:

A programhoz szükséges header file-okat tartalmazza a project, illetve üres cpp fileokat is, hogy lehessen fordítani is. A header file-ok tartalmazzák a kész class, illetve függvénydefiníciókat.

A tesztprogram minden fordítási egységre kiterjedően teszteli a különböző funkciókat, függvényeket, ezt részben automatizálva teszi a gtest_lite segítségével, részben manuális segédfüggvények segítségével, az olyan logikai részek teszteléséhez, amiknek az automatizált tesztelése nehezen megoldható (licitáló algoritmusok tesztelése, eloszlások validálása). A memóriszivárgás vizsgálatát a memtrace.h file segítségével végzem.