Házi feladat

Programozás alapjai 2.

Specifikáció a kész programhoz

Grób László
EAS7U0

2025.04.23.

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

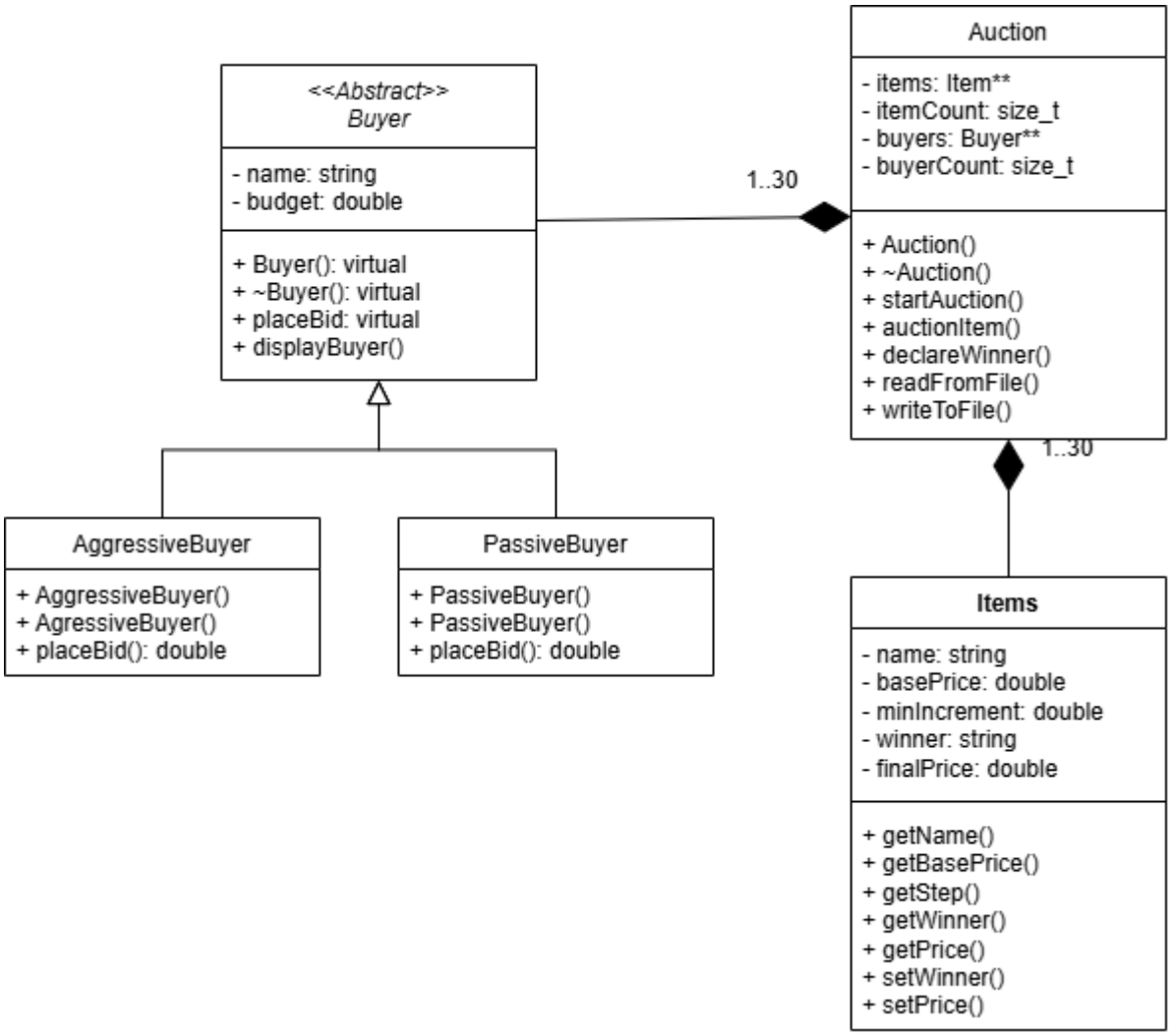
A program modellez agresszívebb és visszafogottabb vevőket is, akik más viselkedéssel licitálnak. A tesztprogramban lehetőség van limitálni a körök számát, a túl hosszú árverés elkerülése érdekében. A tesztprogram futtatásánál egy rögzített seed-del lehetőség van determinisztikus tesztelésre.

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:

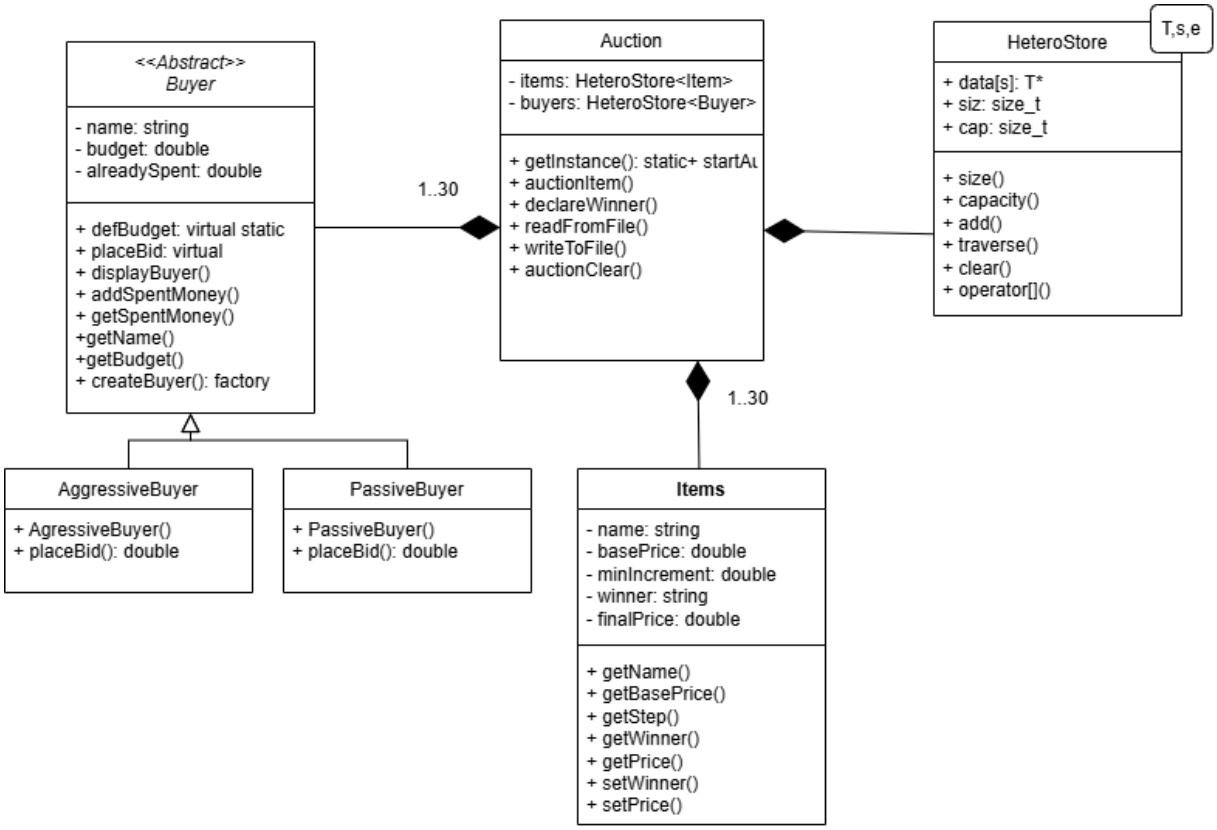
Az árverést lebonyolító Auction class a readFrom file fügvénnyel olvassa be a szükséges információkat a mellékelt fileból, ami tartalmazza az eladásra kínált termékek listáját. Ezután generál vevőket, és egyesével végigiterál az eladásra kínált termékeken, és elárverezi őket az auctionItem függvény segítségével. 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.

5. A kész prograról:

5.1. A program teszteléséről:

A program elkészítéséhez felhasználtam a gyakorlaton korábban elkészített generikus HeteroStore templát osztályt, azonban ennek működésének bemutatását most mellőzöm, mivel gyakorlaton kimerítően foglalkoztunk vele. A program hiba nélkül fordul és az automatikus tesztek is hiba nélkül futnak. A program tesztelése során a memtrace nem észlelt memóriaszivárgást. A tesztek a program minden részletét (a korábban említett HeteroStore klassz kivételével) érintik, és funkcionálisan is tesztelik a működését.

6. A program doxygen által generált dokumentációja:

A továbbiakban csatolom a doxygen által készített meglehetősen terjedelmes programozói dokumentációt.

Nagy házifeladat Programozói dokumentáció

Programozás alapjai 2.
2025.04.23
Grób László

1. fejezet

Hierarchikus mutató

1.1. Osztályhierarchia

Majdnem (de nem teljesen) betűrendbe szedett leszármazási lista:

2. fejezet

Osztálymutató

2.1. Osztálylista

Az összes osztály, struktúra, unió és interfész listája rövid leírásokkal:

3. fejezet

Fájlmutató

3.1. Fájllista

Az összes fájl listája rövid leírásokkal:

4. fejezet

Osztályok dokumentációja

4.1. AggressiveBuyer osztályreferencia

Az agresszív vásárlót leíró osztály, nagyobbat és nagyobb eséllyel licitál.

`#include <Buyer.h>`

Az AggressiveBuyer osztály származási diagramja:

Az AggressiveBuyer osztály együttműködési diagramja:

Publikus tagfüggvények

- AggressiveBuyer (const std::string n, const int b)
- double placeBid (const double currentPrice, const double step) const

  Eldönti hogy licitál e az adott feltételek mellett, majd visszatér a feltett értékkel.

Publikus tagfüggvények a(z) Buyer osztályból származnak

- Buyer (const std::string n, const double b)
- virtual void displayBuyer () const

  Kiírja az adott vásárlót, csak a teszteléshez kell.
- virtual void addSpentMoney (const double i)

  Hozzáadja a már elköltött pénzhez a kapott argumentumot.
- virtual double getSpentMoney () const
- virtual ~Buyer ()
- std::string getName () const
- double getBudget () const

További örökölt tagok

Statikus publikus tagfüggvények a(z) Buyer osztályból származnak

- static Buyer * createBuyer (const std::string name, const double budget, const bool aggressive)

  Factory fv. a vásárlók példányosítására. Statikus hogy objektum nélkül is hívható legyen.

Statikus publikus attribútumok a(z) Buyer osztályból származnak

- static double defBudget = 500

  A default budget érték a vásárlóknak, egyszer inicializálva, hogy utána az Auction class elérje.

4.1.1. Részletes leírás

Az agresszív vásárlót leíró osztály, nagyobbat és nagyobb eséllyel licitál.

Definíció a(z) Buyer.h fájl 56. sorában.

4.1.2. Konstruktorok és destruktorok dokumentációja

4.1.2.1. AggressiveBuyer()

```
AggressiveBuyer::AggressiveBuyer (
            const std::string n,
            const int b)  [inline]
```

Definíció a(z) Buyer.h fájl 59. sorában.

```
00059 : Buyer(n, b) {}
```

A függvény hívási gráfja:

4.1.3. Tagfüggvények dokumentációja

4.1.3.1. placeBid()

```
double AggressiveBuyer::placeBid (
            const double currentPrice,
            const double step) const  [virtual]
```

Eldönti hogy licitál e az adott feltételek mellett, majd visszatér a feltett értékkel.

Paraméterek

currentPrice	Az eladási tárgy jelenlegi ára
step	A minimumm licitlépcső

Visszatérési érték

Visszatér a licit értékével, vagy 0-val ha nem licitál.

Megvalósítja a következőket: Buyer.

Definíció a(z) Buyer.cpp fájl 59. sorában.

```
00060 {
00061     int bidDeviation = currentPrice / 5; // A licitek szórása, agresszív vásárlónál az ár 20%a
00062     if (bidDeviation == 0)               // Ha nulla lenne akkor később okozhat bugokat ezért van itt
      ez a safeguard
00063     {
00064         bidDeviation = 1;
00065     }
00066     double newBid = currentPrice + step + (std::rand() % bidDeviation); // Az új licit értéke
00067     double remainingMoney = getBudget() - getSpentMoney();
00068     if (remainingMoney < newBid) // Ha a licit nagyobb mint a maradék pénze akkor 0-val visszatér
00069     {
00070         return 0;
00071     }
00072     if (getSpentMoney() == 0) // Ha még nem vett semmit akkor mindenképpen licitál
00073     {
00074         return newBid;
00075     }
00076     bool bid;                           // Eldönti hogy licitáljon-e a jelenlegi ár alapján
00077     if (remainingMoney * 0.5 > newBid) // Ha a maradék pénzének a 50%ánál kevesebb a licit, akkor 70%
      eséllyel licitál
00078     {
00079         bid = (rand() % 10 < 7);
00080     }
00081     else // Egyébként 50% eséllyel licitál
00082     {
00083         bid = (rand() % 10 < 5);
00084     }
00085     if (bid) // Ha a licit igaz leadja  a licitet
00086     {
00087         return newBid;
00088     }
00089     return 0;
00090 }
```

A függvény hívási gráfja:

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

- Buyer.h
- Buyer.cpp

4.2. Auction osztályreferencia

Az aukció lebonyolításáért felelős singleton osztály, heterogén kollekcióban tartalmazza az a Buyer classokat.

`#include <Auction.h>`

Az Auction osztály együttműködési diagramja:

Publikus tagfüggvények

- void readFromFile (const bool debug=false, const char *file="items.txt")

  Beolvassa a file-ból az eladni kívánt termékeket, és az items tömbbe rakja.
- void startAuction (const float agressiveRatio, const size_t turnLimit, const double budget, const size_t buyer↩ Count)

  Generál vevőket majd elárverezi a tárgyakat.
- void auctionItem (Item *item, const size_t turnLimit)

  Adott tárgyat elárverez.
- void declareWinner (Item *i, Buyer *b, const double finalPrice)

  Bejegyzi az Item osztály objektumába az eladási árat, és a vevő nevét.
- void writeToFile (const bool debug=true, const char *file="itemsSold.txt")

  Kiírja file-ba vagy stdoutra a tárgyakat a nyertesekkel és végső árral együtt.
- ~Auction ()

  Destruktor, mivel a tároló kezeli a foglalt terület felszabadítását így itt nem kell.
- void AuctionClear ()

  Takarító fv. ha manuálisan akarjuk a memóriát törölni.

Statikus publikus tagfüggvények

- static Auction & getInstance ()

  Függvény az Auction sigleton egyetlen statikus példányának létrehozására.

4.2.1. Részletes leírás

Az aukció lebonyolításáért felelős singleton osztály, heterogén kollekcióban tartalmazza az a Buyer classokat.

Definíció a(z) Auction.h fájl 18. sorában.

4.2.2. Konstruktorok és destruktorok dokumentációja

4.2.2.1. ~Auction()

```
Auction::~Auction ()  [inline]
```

Destruktor, mivel a tároló kezeli a foglalt terület felszabadítását így itt nem kell.

Definíció a(z) Auction.h fájl 64. sorában.

```
00064 {}
```

4.2.3. Tagfüggvények dokumentációja

4.2.3.1. AuctionClear()

```
void Auction::AuctionClear ()  [inline]
```

Takarító fv. ha manuálisan akarjuk a memóriát törölni.

Definíció a(z) Auction.h fájl 66. sorában.

```
00067     {
00068         items.clear();
00069         buyers.clear();
00070     }
```

4.2.3.2. auctionItem()

```
void Auction::auctionItem (
            Item * item,
            const size_t turnLimit)
```

Adott tárgyat elárverez.

Paraméterek

i	Pointer a tárgyra
turnLimit	A beállított maximum körök száma

Definíció a(z) Auction.cpp fájl 70. sorában.

```
00071 {
00072     Buyer *winner = nullptr;                          // Pointer a győztesre, nullpointerre
      inicializélva
00073     double currentRoundPrice = item->getBasePrice(); // Az aktuális körben közben a legmagasabb ár
00074     double step = item->getStep();
00075     double previousRoundHighest = currentRoundPrice; // Az előző kör legmagasabb licitje
00076     int BidsPlaced = buyers.size();                   // A licitek száma a körben
00077     bool mask[buyers.size()];                         // Tömb a kiesett vevők maszkolására
00078     for (size_t i = 0; i < buyers.size(); i++)
00079     {
00080         mask[i] = true;
00081     }
00082     for (size_t i = 0; i < turnLimit && BidsPlaced >= 2; i++) // Külső loop a körökre
00083     {
00084         double bid;
00085         BidsPlaced = 0;                              // A kör elején nullára állítjuk a licitek számát
00086         for (size_t j = 0; j < buyers.size(); j++) // Belső loop az adott körben végigiterálni a
      vevőkön.
00087         {
00088             if (!mask[j])
00089             {
00090                 continue; // A kiesett vásálókat átugorja
00091             }
00092             bid = buyers[j]->placeBid(previousRoundHighest, step);
00093             if (bid == 0) // Ha nem licitál akkor kiesik
00094             {
00095                 mask[j] = false;
00096             }
00097             else
00098             {
00099                 BidsPlaced++;                 // Növeljük a számlálót
00100                 if (bid > currentRoundPrice) // ha magasabb akkor feljegyezzük
00101                 {
00102                     currentRoundPrice = bid;
00103                     winner = buyers[j];
00104                 }
00105             }
00106         }
00107         previousRoundHighest = currentRoundPrice; // átállítjuk az aktuális árat a körben legmagasabb
      árra
00108     }
00109     declareWinner(item, winner, previousRoundHighest);
00110 }
```

A függvény hívási gráfja:

4.2.3.3. declareWinner()

```
void Auction::declareWinner (
            Item * i,
            Buyer * b,
            const double finalPrice)
```

Bejegyzi az Item osztály objektumába az eladási árat, és a vevő nevét.

Paraméterek

i	Az adott tárgy
b	A vevőre mutató
finalPrice	A végső eladási ár

Definíció a(z) Auction.cpp fájl 111. sorában.

```
00112 {
00113     if (b == nullptr) // Hibakezelés, ha nem kellt el a tárgy, akkor itt kezeljük, az eladási árnál -1
      el jelezzük.
00114     {
00115         i->setWinner("senki nem");
00116         i->setFinalPrice(-1);
00117         return;
00118     }
00119     b->addSpentMoney(finalPrice);
00120     i->setWinner(b->getName());
00121     i->setFinalPrice(finalPrice);
00122 }
```

A függvény hívási gráfja:

4.2.3.4. getInstance()

```
static Auction & Auction::getInstance ()  [inline], [static]
```

Függvény az Auction sigleton egyetlen statikus példányának létrehozására.

Paraméterek

b	A vevők száma amivel inicializáljuk

Visszatérési érték

Visszatér a statikus példány referenciájával

Definíció a(z) Auction.h fájl 35. sorában.

```
00036     {
00037         static Auction instance;
00038         return instance;
00039     }
```

4.2.3.5. readFromFile()

```
void Auction::readFromFile (
            const bool debug = false,
            const char * file = "items.txt")
```

Beolvassa a file-ból az eladni kívánt termékeket, és az items tömbbe rakja.

Paraméterek

debug	Flag, ha be van kapcsolva file helyett a konzolról olvassa be az adatokat, alapértelmezetten hamis
file	A file amiből beolvas alapértelmezetten az items.txt ugyanebben a könyvtárban

Definíció a(z) Auction.cpp fájl 7. sorában.

```
00008 {
00009     std::ifstream f(file);
00010     if (!f.is_open())
00011     {
00012         throw std::runtime_error("Nem sikerült megnyitni a tárgyakat tartalmazó file-t!");
00013     }
00014     std::istream *input = &f;
00015     if (debug) // Ha a debug aktív az inputstream át van irányítva a cin-re
00016     {
00017         input = &std::cin;
00018     }
00019     size_t ic = 0; // Az újonnan hozzáadott tárgyak száma
00020     *input » ic;
00021     if (input->fail() || (input->eof()))
00022     {
00023         throw std::runtime_error("Nem sikerült kiolvasni a tárgyakat tartalmazó fileból a tárgyak
      számát!");
00024     }
00025     if (ic + items.size() > items.capacity())
00026     {
00027         throw std::out_of_range("Túl sok tárgyat próbál meg beolvasni!");
00028     }
00029
00030     input->ignore(); // az újsor karaktert átugorja
00031     for (size_t i = 0; i < ic; i++)
00032     {
00033         std::string line;
00034         if (!std::getline(*input, line))
00035         {
00036             throw std::runtime_error("Nem sikerült beolvasni a " + std::to_string(i + 1) + ". sort");
00037         }
00038         std::istringstream iss(line);
00039         std::string name;
00040         double price, step;
00041
00042         if (!(iss » name » price » step)) // Nem lehet space a tárgy nevében
00043         {
00044             throw std::runtime_error("Helytelen beviteli formátum a:" + std::to_string(i + 1) + ".
      sorban");
00045         }
00046         items.add(new Item(name, price, step));
00047     }
00048     f.close();
00049 }
```

4.2.3.6. startAuction()

```
void Auction::startAuction (
             const float agressiveRatio,
             const size_t turnLimit,
             const double budget,
             const size_t buyerCount)
```

Generál vevőket majd elárverezi a tárgyakat.

Paraméterek

turnLimit	A maximális körök száma
agressiveRatio	Az agresszív vevők aránya a passzívokhoz képest, 0 és 1 közötti szám
budget	Az alapértelmezett budget értéket állítja be
buyerCount	A vevők számát állítja be

Végigiterál a tárgyakon és mindegyiket elárverezi

Definíció a(z) Auction.cpp fájl 50. sorában.

```
00051 {
00052     Buyer::defBudget = budget;
00053     size_t an = buyerCount * agressiveRatio; // Az agresszív vevők száma
00054     for (size_t i = 0; i < an; i++)
00055     {
00056         std::string name = "Buyer" + std::to_string(i + 1);
00057         buyers.add(Buyer::createBuyer(name, Buyer::defBudget, true));
00058     }
00059     for (size_t i = an; i < buyerCount; i++)
00060     {
00061         std::string name = "Buyer" + std::to_string(i + 1);
00062         buyers.add(Buyer::createBuyer(name, Buyer::defBudget, false));
00063     }
00065     for (size_t i = 0; i < items.size(); i++)
00066     {
00067         auctionItem(items[i], turnLimit);
00068     }
00069 }
```

A függvény hívási gráfja:

4.2.3.7. writeToFile()

```
void Auction::writeToFile (
            const bool debug = true,
            const char * file = "itemsSold.txt")
```

Kiírja file-ba vagy stdoutra a tárgyakat a nyertesekkel és végső árral együtt.

Paraméterek

file	A file neve ahova írja, alapértelmezetten itemsSold.txt
debug	Flag, ha be van kapcsolva file helyett a konzolra írja ki az eredményeket, alapértelmezetten igaz

Definíció a(z) Auction.cpp fájl 123. sorában.

```
00124 {
00125     std::ofstream f(file);
00126     if (!f.is_open())
00127     {
00128         throw std::runtime_error("Nem sikerült megnyitni az eladott tárgyakat tartalmazó file-t!");
00129     }
00130     std::ostream *output = &f;
00131     if (debug) // Ha a debug aktív az outputstream át van irányítva a cout-ra
00132     {
00133         output = &std::cout;
00134     }
00135     for (size_t i = 0; i < items.size(); i++)
00136     {
00137         (*output) « (*items[i]);
00138     }
00139     if (!debug)
00140     {
00141         std::cout « "Sikeresen kiírva " « items.size() « "db tárgy file-ba!" « std::endl;
00142     }
00143
00144     f.close();
00145 }
```

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

- Auction.h
- Auction.cpp

4.3. Buyer osztályreferencia

A vásárló osztály absztrakt alaposztálya.

`#include <Buyer.h>`

A Buyer osztály származási diagramja:

A Buyer osztály együttműködési diagramja:

Publikus tagfüggvények

- Buyer (const std::string n, const double b)
- virtual double placeBid (const double currentPrice, const double step) const =0
- virtual void displayBuyer () const

  Kiírja az adott vásárlót, csak a teszteléshez kell.
- virtual void addSpentMoney (const double i)

  Hozzáadja a már elköltött pénzhez a kapott argumentumot.
- virtual double getSpentMoney () const
- virtual ∼Buyer ()
- std::string getName () const
- double getBudget () const

Statikus publikus tagfüggvények

- static Buyer ∗ createBuyer (const std::string name, const double budget, const bool aggressive)

  Factory fv. a vásárlók példányosítására. Statikus hogy objektum nélkül is hívható legyen.

Statikus publikus attribútumok

- static double defBudget = 500

  A default budget érték a vásárlóknak, egyszer inicializálva, hogy utána az Auction class elérje.

4.3.1. Részletes leírás

A vásárló osztály absztrakt alaposztálya.

Nem példányosítható.

Definíció a(z) Buyer.h fájl 13. sorában.

4.3.2. Konstruktorok és destruktorok dokumentációja

4.3.2.1. Buyer()

```
Buyer::Buyer (
            const std::string n,
            const double b)  [inline]
```

Definíció a(z) Buyer.h fájl 22. sorában.

```
00022 : name(n), budget(b), alreadySpent(0) {}
```

4.3.2.2. ~Buyer()

```
virtual Buyer::~Buyer ()  [inline], [virtual]
```

Definíció a(z) Buyer.h fájl 30. sorában.

```
00030 {}
```

4.3.3. Tagfüggvények dokumentációja

4.3.3.1. addSpentMoney()

```
virtual void Buyer::addSpentMoney (
            const double i)  [inline], [virtual]
```

Hozzáadja a már elköltött pénzhez a kapott argumentumot.

Paraméterek

i	Ennyivel növeli a már elköltött pénz változó értékét

Definíció a(z) Buyer.h fájl 28. sorában.

```
00028 { alreadySpent += i; }
```

4.3.3.2. createBuyer()

```
Buyer * Buyer::createBuyer (
            const std::string name,
            const double budget,
            const bool aggressive)  [static]
```

Factory fv. a vásárlók példányosítására. Statikus hogy objektum nélkül is hívható legyen.

Paraméterek

name	A vásáló neve
budget	A vásárló pénze
aggressive	Igaz ha agresszív buyert szeretnénk, később bővíthető integerré, ha több leszármazott van

Visszatérési érték

Pointer a létrehozott vásárló objektumra

Definíció a(z) Buyer.cpp fájl 14. sorában.

```
00015 {
00016     if (aggressive)
00017         return new AggressiveBuyer(name + " (Agresszív)", budget);
00018     else
00019         return new PassiveBuyer(name + " (Passzív)", budget);
00020 }
```

A függvény hívási gráfja:

4.3.3.3. displayBuyer()

```
void Buyer::displayBuyer () const  [virtual]
```

Kiírja az adott vásárlót, csak a teszteléshez kell.

Definíció a(z) Buyer.cpp fájl 10. sorában.

```
00011 {
00012     std::cout « "Vásárló: " « name « " | Költségvetése: " « budget « std::endl;
00013 }
```

4.3.3.4. getBudget()

```
double Buyer::getBudget () const  [inline]
```

Definíció a(z) Buyer.h fájl 32. sorában.

```
00032 { return budget; }
```

4.3.3.5. getName()

```
std::string Buyer::getName () const  [inline]
```

Definíció a(z) Buyer.h fájl 31. sorában.

```
00031 { return name; }
```

4.3.3.6. getSpentMoney()

```
virtual double Buyer::getSpentMoney () const  [inline], [virtual]
```

Definíció a(z) Buyer.h fájl 29. sorában.

```
00029 { return alreadySpent; }
```

4.3.3.7. placeBid()

```
virtual double Buyer::placeBid (
            const double currentPrice,
            const double step) const  [pure virtual]
```

Megvalósítják a következők: AggressiveBuyer és PassiveBuyer.

4.3.4. Adattagok dokumentációja

4.3.4.1. defBudget

```
double Buyer::defBudget = 500  [static]
```

A default budget érték a vásárlóknak, egyszer inicializálva, hogy utána az Auction class elérje.

Definíció a(z) Buyer.h fájl 21. sorában.

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

- Buyer.h
- Buyer.cpp

4.4. HeteroStore< T, s, e > osztálysablon-referencia

A gyakorlaton készített tároló templát.

`#include <hetero_store.hpp>`

A HeteroStore< T, s, e > osztály származási diagramja:

A HeteroStore< T, s, e > osztály együttműködési diagramja:

Publikus tagfüggvények

- HeteroStore ()
- size_t size () const
- size_t capacity () const
- void add (T *p)
- template<typename F>
  void traverse (F f)
- void clear ()
- T * operator[ ] (size_t i) const
- ~HeteroStore ()

4.4.1. Részletes leírás

template<class T, size_t s = 30, class e = std::out_of_range>
class HeteroStore< T, s, e >

A gyakorlaton készített tároló templát.

Sablon paraméterek

T	A tárolandó class neve
e	Az error típusa
s	A tároló kapacitása, alapértelmezetten 30

Definíció a(z) hetero_store.hpp fájl 19. sorában.

4.4.2. Konstruktorok és destruktorok dokumentációja

4.4.2.1. HeteroStore()

```
template<class T, size_t s = 30, class e = std::out_of_range>
HeteroStore< T, s, e >::HeteroStore ()  [inline]
```

Definíció a(z) hetero_store.hpp fájl 28. sorában.

```
00028 : siz(0), cap(s) {}
```

4.4.2.2. ~HeteroStore()

```
template<class T, size_t s = 30, class e = std::out_of_range>
HeteroStore< T, s, e >::~HeteroStore ()  [inline]
```

Definíció a(z) hetero_store.hpp fájl 56. sorában.

```
00056 { clear(); }
```

A függvény hívási gráfja:

4.4.3. Tagfüggvények dokumentációja

4.4.3.1. add()

```
template<class T, size_t s = 30, class e = std::out_of_range>
void HeteroStore< T, s, e >::add (
             T * p)  [inline]
```

Definíció a(z) hetero_store.hpp fájl 31. sorában.

```
00032     {
00033         if (siz >= cap)
00034         {
00035             delete p;
00036             throw e("Túllépi a megadott kapacitást!");
00037         }
00038         data[siz++] = p;
00039     }
```

4.4.3.2. capacity()

```
template<class T, size_t s = 30, class e = std::out_of_range>
size_t HeteroStore< T, s, e >::capacity () const  [inline]
```

Definíció a(z) hetero_store.hpp fájl 30. sorában.

```
00030 { return cap; }
```

4.4.3.3. clear()

```
template<class T, size_t s = 30, class e = std::out_of_range>
void HeteroStore< T, s, e >::clear ()  [inline]
```

Definíció a(z) hetero_store.hpp fájl 46. sorában.

```
00047     {
00048         for (size_t i = 0; i < siz; i++)
00049             delete data[i];
00050         siz = 0;
00051     }
```

4.4.3.4. operator[]()

```
template<class T, size_t s = 30, class e = std::out_of_range>
T * HeteroStore< T, s, e >::operator[] (
             size_t i) const  [inline]
```

Definíció a(z) hetero_store.hpp fájl 52. sorában.

```
00053     {
00054         return data[i];
00055     }
```

4.4.3.5. size()

```
template<class T, size_t s = 30, class e = std::out_of_range>
size_t HeteroStore< T, s, e >::size () const  [inline]
```

Definíció a(z) hetero_store.hpp fájl 29. sorában.

```
00029 { return siz; }
```

4.4.3.6. traverse()

```
template<class T, size_t s = 30, class e = std::out_of_range>
template<typename F>
void HeteroStore< T, s, e >::traverse (
             F f)  [inline]
```

Definíció a(z) hetero_store.hpp fájl 41. sorában.

```
00042     {
00043         for (size_t i = 0; i < siz; i++)
00044             f(data[i]);
00045     }
```

Ez a dokumentáció az osztályról a következő fájl alapján készült:

- hetero_store.hpp

4.5. Item osztályreferencia

Az eladott tárgyak osztálya.

```
#include <Item.h>
```

Az Item osztály együttműködési diagramja:

Publikus tagfüggvények

- Item (std::string n, double startPrice, double minIncrement)
  Konstruktor az Item classhoz.
- Item (const Item &i)
- Item & operator= (const Item &i)
- std::string getName () const
  Standard getter fv.
- double getBasePrice () const
  Standard getter fv.
- double getStep () const
  Standard getter fv.
- std::string getWinner () const
  Standard getter fv.
- double getFinalPrice () const
  Standard getter fv.
- void setWinner (const std::string s)
  Standard setter fv.
- void setFinalPrice (const double p)
  Standard setter fv.

Védett attribútumok

- std::string name
- double basePrice
- double minIncrement
- std::string winner
- double finalPrice

4.5.1. Részletes leírás

Az eladott tárgyak osztálya.

Definíció a(z) Item.h fájl 12. sorában.

4.5.2. Konstruktorok és destruktorok dokumentációja

4.5.2.1. Item() [1/2]

```
Item::Item (
            std::string n,
            double startPrice,
            double minIncrement)  [inline]
```

Konstruktor az Item classhoz.

Paraméterek

n	A tárgy neve
startPrice	A kezdő értéke
minIncrement	A minimum licitlépcső

Definíció a(z) Item.h fájl 26. sorában.

```
00026 : name(n), basePrice(startPrice), minIncrement(minIncrement) {}
```

4.5.2.2. Item() [2/2]

```
Item::Item (
            const Item & i)  [inline]
```

Definíció a(z) Item.h fájl 27. sorában.

```
00027 : name(i.name), basePrice(i.basePrice), minIncrement(i.minIncrement) {}
```

A függvény hívási gráfja:

4.5.3. Tagfüggvények dokumentációja

4.5.3.1. getBasePrice()

```
double Item::getBasePrice () const  [inline]
```

Standard getter fv.

Definíció a(z) Item.h fájl 42. sorában.

```
00042 { return basePrice; }
```

4.5.3.2. getFinalPrice()

```
double Item::getFinalPrice () const  [inline]
```

Standard getter fv.

Definíció a(z) Item.h fájl 48. sorában.

```
00048 { return finalPrice; }
```

4.5.3.3. getName()

```
std::string Item::getName () const  [inline]
```

Standard getter fv.

Definíció a(z) Item.h fájl 40. sorában.

```
00040 { return name; }
```

4.5.3.4. getStep()

```
double Item::getStep () const  [inline]
```

Standard getter fv.

Definíció a(z) Item.h fájl 44. sorában.

```
00044 { return minIncrement; }
```

4.5.3.5. getWinner()

```
std::string Item::getWinner () const  [inline]
```

Standard getter fv.

Definíció a(z) Item.h fájl 46. sorában.

```
00046 { return winner; }
```

4.5.3.6. operator=()

```
Item & Item::operator= (
            const Item & i)  [inline]
```

Definíció a(z) Item.h fájl 28. sorában.

```
00029     {
00030         if (this != &i)
00031         {
00032             name = i.name;
00033             basePrice = i.basePrice;
00034             minIncrement = i.minIncrement;
00035         }
00036         return *this;
00037     }
```

A függvény hívási gráfja:

4.5.3.7. setFinalPrice()

```
void Item::setFinalPrice (
            const double p)  [inline]
```

Standard setter fv.

Definíció a(z) Item.h fájl 52. sorában.

```
00052 { finalPrice = p; }
```

4.5.3.8. setWinner()

```
void Item::setWinner (
            const std::string s)  [inline]
```

Standard setter fv.

Definíció a(z) Item.h fájl 50. sorában.

```
00050 { winner = s; }
```

4.5.4. Adattagok dokumentációja

4.5.4.1. basePrice

```
double Item::basePrice  [protected]
```

Definíció a(z) Item.h fájl 16. sorában.

4.5.4.2. finalPrice

```
double Item::finalPrice  [protected]
```

Definíció a(z) Item.h fájl 19. sorában.

4.5.4.3. minIncrement

```
double Item::minIncrement  [protected]
```

Definíció a(z) Item.h fájl 17. sorában.

4.5.4.4. name

```
std::string Item::name  [protected]
```

Definíció a(z) Item.h fájl 15. sorában.

4.5.4.5. winner

```
std::string Item::winner  [protected]
```

Definíció a(z) Item.h fájl 18. sorában.

Ez a dokumentáció az osztályról a következő fájl alapján készült:

- Item.h

4.6. PassiveBuyer osztályreferencia

A passzív vásárlót leíró osztály, visszafogottabban licitál.

```
#include <Buyer.h>
```

A PassiveBuyer osztály származási diagramja:

A PassiveBuyer osztály együttműködési diagramja:

Publikus tagfüggvények

- PassiveBuyer (const std::string n, const double b)
- double placeBid (const double currentPrice, const double step) const

  Eldönti hogy licitál e az adott feltételek mellett, majd visszatér a feltett értékkel.

Publikus tagfüggvények a(z) Buyer osztályból származnak

- Buyer (const std::string n, const double b)
- virtual void displayBuyer () const

  Kiírja az adott vásárlót, csak a teszteléshez kell.
- virtual void addSpentMoney (const double i)

  Hozzáadja a már elköltött pénzhez a kapott argumentumot.
- virtual double getSpentMoney () const
- virtual ~Buyer ()
- std::string getName () const
- double getBudget () const

További örökölt tagok

Statikus publikus tagfüggvények a(z) Buyer osztályból származnak

- static Buyer * createBuyer (const std::string name, const double budget, const bool aggressive)

  Factory fv. a vásárlók példányosítására. Statikus hogy objektum nélkül is hívható legyen.

Statikus publikus attribútumok a(z) Buyer osztályból származnak

- static double defBudget = 500

  A default budget érték a vásárlóknak, egyszer inicializálva, hogy utána az Auction class elérje.

4.6.1. Részletes leírás

A passzív vásárlót leíró osztály, visszafogottabban licitál.

Definíció a(z) Buyer.h fájl 43. sorában.

4.6.2. Konstruktorok és destruktorok dokumentációja

4.6.2.1. PassiveBuyer()

```
PassiveBuyer::PassiveBuyer (
            const std::string n,
            const double b)  [inline]
```

Definíció a(z) Buyer.h fájl 46. sorában.

```
00046 : Buyer(n, b) {}
```

A függvény hívási gráfja:

4.6.3. Tagfüggvények dokumentációja

4.6.3.1. placeBid()

```
double PassiveBuyer::placeBid (
            const double currentPrice,
            const double step) const  [virtual]
```

Eldönti hogy licitál e az adott feltételek mellett, majd visszatér a feltett értékkel.

Paraméterek

currentPrice	Az eladási tárgy jelenlegi ára
step	A minimumm licitlépcső

Visszatérési érték

Visszatér a licit értékével, vagy 0-val ha nem licitál.

Megvalósítja a következőket: Buyer.

Definíció a(z) Buyer.cpp fájl 22. sorában.

```
00023 {
00024     int bidDeviation = currentPrice / 10; // A licitek szórása, konzervatív vásárlónál az ár 10%a
00025     if (bidDeviation == 0)                // Ha nulla lenne akkor később okozhat bugokat ezért van itt
      ez a safeguard
00026     {
00027         bidDeviation = 1;
00028     }
00029     double newBid = currentPrice + step + (std::rand() % bidDeviation); // Az új licit értéke
00030     double remainingMoney = getBudget() - getSpentMoney();
00031     if (remainingMoney < newBid) // Ha a licit nagyobb mint a maradék pénze akkor 0-val visszatér
00032     {
00033         return 0;
00034     }
00035     bool bid, weight = true;                  // Eldönti hogy licitáljon-e a jelenlegi ár alapján
00036     if ((getSpentMoney() / getBudget()) > 0.7) // Ha a pénzének már több mint a 70%-át elköltötte
      akkor csak 20% eséllyel licitál, ami multiplikatív a következő esélyekkel.
00037     {
00038         weight = (rand() % 10 < 2);
00039     }
00040     if (remainingMoney * 0.3 > newBid) // Ha a maradék pénzének a 20%ánál kevesebb a licit, akkor 70%
      eséllyel licitál
00041     {
00042         bid = (rand() % 10 < 7);
00043     }
00044     else if (remainingMoney * 0.5 > newBid) // Ha a maradék pénzének a 50%ánál kevesebb a licit, akkor
      40% eséllyel licitál
00045     {
00046         bid = (rand() % 10 < 4);
00047     }
00048     else // Egyébként 20% eséllyel licitál
00049     {
00050         bid = (rand() % 10 < 2);
00051     }
00052     if (bid && weight) // Ha a licit is igaz és a weight(ha a pénze nagy részét elköltötte akkor kicsi
      az esély hogy igaz) is igaz akkor visszatér a licittel
00053     {
00054         return newBid;
00055     }
00056     return 0; // Egyébként visszatér nullával
00057 }
```

A függvény hívási gráfja:

Ez a dokumentáció az osztályról a következő fájlok alapján készült:

- Buyer.h
- Buyer.cpp

5. fejezet

Fájlok dokumentációja

5.1. Auction.cpp fájlreferencia

Az aukció lebonyolításához szükséges függvények.

`#include "Auction.h"`
Az Auction.cpp definíciós fájl függési gráfja:

5.2. Auction.cpp

Ugrás a fájl dokumentációjához.

```
00001
00005 #include "Auction.h"
00006
00007 void Auction::readFromFile(const bool debug, const char *file)
00008 {
00009     std::ifstream f(file);
00010     if (!f.is_open())
00011     {
00012         throw std::runtime_error("Nem sikerült megnyitni a tárgyakat tartalmazó file-t!");
00013     }
00014     std::istream *input = &f;
00015     if (debug) // Ha a debug aktív az inputstream át van irányítva a cin-re
00016     {
00017         input = &std::cin;
00018     }
00019     size_t ic = 0; // Az újonnan hozzáadott tárgyak száma
00020     *input » ic;
00021     if (input->fail() || (input->eof()))
00022     {
00023         throw std::runtime_error("Nem sikerült kiolvasni a tárgyakat tartalmazó fileból a tárgyak
      számát!");
00024     }
00025     if (ic + items.size() > items.capacity())
00026     {
00027         throw std::out_of_range("Túl sok tárgyat próbál meg beolvasni!");
00028     }
00029
00030     input->ignore(); // az újsor karaktert átugorja
00031     for (size_t i = 0; i < ic; i++)
00032     {
00033         std::string line;
00034         if (!std::getline(*input, line))
00035         {
00036             throw std::runtime_error("Nem sikerült beolvasni a " + std::to_string(i + 1) + ". sort");
00037         }
00038         std::istringstream iss(line);
00039         std::string name;
00040         double price, step;
00041
00042         if (!(iss » name » price » step)) // Nem lehet space a tárgy nevében
```

```
00043         {
00044             throw std::runtime_error("Helytelen beviteli formátum a:" + std::to_string(i + 1) + ".
      sorban");
00045         }
00046         items.add(new Item(name, price, step));
00047     }
00048     f.close();
00049 }
00050 void Auction::startAuction(const float agressiveRatio, const size_t turnLimit, const double budget,
      const size_t buyerCount)
00051 {
00052     Buyer::defBudget = budget;
00053     size_t an = buyerCount * agressiveRatio; // Az agresszív vevők száma
00054     for (size_t i = 0; i < an; i++)
00055     {
00056         std::string name = "Buyer" + std::to_string(i + 1);
00057         buyers.add(Buyer::createBuyer(name, Buyer::defBudget, true));
00058     }
00059     for (size_t i = an; i < buyerCount; i++)
00060     {
00061         std::string name = "Buyer" + std::to_string(i + 1);
00062         buyers.add(Buyer::createBuyer(name, Buyer::defBudget, false));
00063     }
00065     for (size_t i = 0; i < items.size(); i++)
00066     {
00067         auctionItem(items[i], turnLimit);
00068     }
00069 }
00070 void Auction::auctionItem(Item *item, const size_t turnLimit)
00071 {
00072     Buyer *winner = nullptr;                          // Pointer a győztesre, nullpointerre
      inicializélva
00073     double currentRoundPrice = item->getBasePrice(); // Az aktuális körben közben a legmagasabb ár
00074     double step = item->getStep();
00075     double previousRoundHighest = currentRoundPrice; // Az előző kör legmagasabb licitje
00076     int BidsPlaced = buyers.size();                   // A licitek száma a körben
00077     bool mask[buyers.size()];                         // Tömb a kiesett vevők maszkolására
00078     for (size_t i = 0; i < buyers.size(); i++)
00079     {
00080         mask[i] = true;
00081     }
00082     for (size_t i = 0; i < turnLimit && BidsPlaced >= 2; i++) // Külső loop a körökre
00083     {
00084         double bid;
00085         BidsPlaced = 0;                              // A kör elején nullára állítjuk a licitek számát
00086         for (size_t j = 0; j < buyers.size(); j++) // Belső loop az adott körben végigiterálni a
      vevőkön.
00087         {
00088             if (!mask[j])
00089             {
00090                 continue; // A kiesett vásálókat átugorja
00091             }
00092             bid = buyers[j]->placeBid(previousRoundHighest, step);
00093             if (bid == 0) // Ha nem licitál akkor kiesik
00094             {
00095                 mask[j] = false;
00096             }
00097             else
00098             {
00099                 BidsPlaced++;                // Növeljük a számlálót
00100                 if (bid > currentRoundPrice) // ha magasabb akkor feljegyezzük
00101                 {
00102                     currentRoundPrice = bid;
00103                     winner = buyers[j];
00104                 }
00105             }
00106         }
00107         previousRoundHighest = currentRoundPrice; // átállítjuk az aktuális árat a körben legmagasabb
      árra
00108     }
00109     declareWinner(item, winner, previousRoundHighest);
00110 }
00111 void Auction::declareWinner(Item *i, Buyer *b, const double finalPrice)
00112 {
00113     if (b == nullptr) // Hibakezelés, ha nem kellt el a tárgy, akkor itt kezeljük, az eladási árnál -1
      el jelezzük.
00114     {
00115         i->setWinner("senki nem");
00116         i->setFinalPrice(-1);
00117         return;
00118     }
00119     b->addSpentMoney(finalPrice);
00120     i->setWinner(b->getName());
00121     i->setFinalPrice(finalPrice);
00122 }
00123 void Auction::writeToFile(const bool debug, const char *file)
00124 {
```

```
00125     std::ofstream f(file);
00126     if (!f.is_open())
00127     {
00128         throw std::runtime_error("Nem sikerült megnyitni az eladott tárgyakat tartalmazó file-t!");
00129     }
00130     std::ostream *output = &f;
00131     if (debug) // Ha a debug aktív az outputstream át van irányítva a cout-ra
00132     {
00133         output = &std::cout;
00134     }
00135     for (size_t i = 0; i < items.size(); i++)
00136     {
00137         (*output) « (*items[i]);
00138     }
00139     if (!debug)
00140     {
00141         std::cout « "Sikeresen kiírva " « items.size() « "db tárgy file-ba!" « std::endl;
00142     }
00143
00144     f.close();
00145 }
```

5.3. Auction.h fájlreferencia

Az aukció osztály deklarációját tartalmazó header file.

```
#include "memtrace.h"
#include "Item.h"
#include "Buyer.h"
#include "hetero_store.hpp"
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
```

Az Auction.h definíciós fájl függési gráfja: Ez az ábra azt mutatja, hogy mely fájlok ágyazzák be közvetve vagy közvetlenül ezt a fájlt:

Osztályok

- class Auction

  Az aukció lebonyolításáért felelős singleton osztály, heterogén kollekcióban tartalmazza az a Buyer classokat.

5.3.1. Részletes leírás

Az aukció osztály deklarációját tartalmazó header file.

Definíció a(z) Auction.h fájlban.

5.4. Auction.h

Ugrás a fájl dokumentációjához.

```
00001
00005 #ifndef AUCTION_H
00006 #define AUCTION_H
00007 #include "memtrace.h"
00008 #include "Item.h"
00009 #include "Buyer.h"
00010 #include "hetero_store.hpp"
00011 #include <iostream>
```

```
00012 #include <fstream>
00013 #include <sstream>
00014 #include <string>
00018 class Auction
00019 {
00020 private:
00021     HeteroStore<Item> items;
00022     HeteroStore<Buyer> buyers; // Ősosztály pointert tárolok
00024     Auction(Auction &);
00026     Auction &operator=(Auction &);
00029     Auction() {}
00030
00031 public:
00035     static Auction &getInstance()
00036     {
00037         static Auction instance;
00038         return instance;
00039     }
00043     void readFromFile(const bool debug = false, const char *file = "items.txt");
00049     void startAuction(const float agressiveRatio, const size_t turnLimit, const double budget, const
      size_t buyerCount);
00053     void auctionItem(Item *item, const size_t turnLimit);
00058     void declareWinner(Item *i, Buyer *b, const double finalPrice);
00062     void writeToFile(const bool debug = true, const char *file = "itemsSold.txt");
00064     ~Auction() {}
00066     void AuctionClear()
00067     {
00068         items.clear();
00069         buyers.clear();
00070     }
00071 };
00072
00073 #endif
```

5.5. Buyer.cpp fájlreferencia

A vásárló osztályok függvényeit tartalmazó cpp file.

`#include "Buyer.h"`
A Buyer.cpp definíciós fájl függési gráfja:

5.5.1. Részletes leírás

A vásárló osztályok függvényeit tartalmazó cpp file.

Definíció a(z) Buyer.cpp fájlban.

5.6. Buyer.cpp

Ugrás a fájl dokumentációjához.

```
00001
00005 #include "Buyer.h"
00006
00008 double Buyer::defBudget = 500;
00009
00010 void Buyer::displayBuyer() const
00011 {
00012     std::cout « "Vásárló: " « name « " | Költségvetése: " « budget « std::endl;
00013 }
00014 Buyer *Buyer::createBuyer(const std::string name, const double budget, const bool aggressive)
00015 {
00016     if (aggressive)
00017         return new AggressiveBuyer(name + " (Agresszív)", budget);
00018     else
00019         return new PassiveBuyer(name + " (Passzív)", budget);
00020 }
00021
```

```
00022 double PassiveBuyer::placeBid(const double currentPrice, const double step) const
00023 {
00024     int bidDeviation = currentPrice / 10; // A licitek szórása, konzervatív vásárlónál az ár 10%a
00025     if (bidDeviation == 0)                // Ha nulla lenne akkor később okozhat bugokat ezért van itt
      ez a safeguard
00026     {
00027         bidDeviation = 1;
00028     }
00029     double newBid = currentPrice + step + (std::rand() % bidDeviation); // Az új licit értéke
00030     double remainingMoney = getBudget() - getSpentMoney();
00031     if (remainingMoney < newBid) // Ha a licit nagyobb mint a maradék pénze akkor 0-val visszatér
00032     {
00033         return 0;
00034     }
00035     bool bid, weight = true;                  // Eldönti hogy licitáljon-e a jelenlegi ár alapján
00036     if ((getSpentMoney() / getBudget()) > 0.7) // Ha a pénzének már több mint a 70%-át elköltötte
      akkor csak 20% eséllyel licitál, ami multiplikatív a következő esélyekkel.
00037     {
00038         weight = (rand() % 10 < 2);
00039     }
00040     if (remainingMoney * 0.3 > newBid) // Ha a maradék pénzének a 20%ánál kevesebb a licit, akkor 70%
      eséllyel licitál
00041     {
00042         bid = (rand() % 10 < 7);
00043     }
00044     else if (remainingMoney * 0.5 > newBid) // Ha a maradék pénzének a 50%ánál kevesebb a licit, akkor
      40% eséllyel licitál
00045     {
00046         bid = (rand() % 10 < 4);
00047     }
00048     else // Egyébként 20% eséllyel licitál
00049     {
00050         bid = (rand() % 10 < 2);
00051     }
00052     if (bid && weight) // Ha a licit is igaz és a weight(ha a pénze nagy részét elköltötte akkor kicsi
      az esély hogy igaz) is igaz akkor visszatér a licittel
00053     {
00054         return newBid;
00055     }
00056     return 0; // Egyébként visszatér nullával
00057 }
00058
00059 double AggressiveBuyer::placeBid(const double currentPrice, const double step) const
00060 {
00061     int bidDeviation = currentPrice / 5; // A licitek szórása, agresszív vásárlónál az ár 20%a
00062     if (bidDeviation == 0)               // Ha nulla lenne akkor később okozhat bugokat ezért van itt
      ez a safeguard
00063     {
00064         bidDeviation = 1;
00065     }
00066     double newBid = currentPrice + step + (std::rand() % bidDeviation); // Az új licit értéke
00067     double remainingMoney = getBudget() - getSpentMoney();
00068     if (remainingMoney < newBid) // Ha a licit nagyobb mint a maradék pénze akkor 0-val visszatér
00069     {
00070         return 0;
00071     }
00072     if (getSpentMoney() == 0) // Ha még nem vett semmit akkor mindenképpen licitál
00073     {
00074         return newBid;
00075     }
00076     bool bid;                           // Eldönti hogy licitáljon-e a jelenlegi ár alapján
00077     if (remainingMoney * 0.5 > newBid) // Ha a maradék pénzének a 50%ánál kevesebb a licit, akkor 70%
      eséllyel licitál
00078     {
00079         bid = (rand() % 10 < 7);
00080     }
00081     else // Egyébként 50% eséllyel licitál
00082     {
00083         bid = (rand() % 10 < 5);
00084     }
00085     if (bid) // Ha a licit igaz leadja  a licitet
00086     {
00087         return newBid;
00088     }
00089     return 0;
00090 }
```

5.7. Buyer.h fájlreferencia

A vásárlók osztályait tartalmazó header file.

```
#include "memtrace.h"
#include <iostream>
```

A Buyer.h definíciós fájl függési gráfja: Ez az ábra azt mutatja, hogy mely fájlok ágyazzák be közvetve vagy közvetlenül ezt a fájlt:

Osztályok

- class Buyer

  A vásárló osztály absztrakt alaposztálya.
- class PassiveBuyer

  A passzív vásárlót leíró osztály, visszafogottabban licitál.
- class AggressiveBuyer

  Az agresszív vásárlót leíró osztály, nagyobbat és nagyobb eséllyel licitál.

5.7.1. Részletes leírás

A vásárlók osztályait tartalmazó header file.

Definíció a(z) Buyer.h fájlban.

5.8. Buyer.h

Ugrás a fájl dokumentációjához.

```
00001
00005 #ifndef BUYER_H
00006 #define BUYER_H
00007 #include "memtrace.h"
00008 #include <iostream>
00009
00013 class Buyer
00014 {
00015 private:
00016     std::string name;
00017     double budget;
00018     double alreadySpent;
00019
00020 public:
00021     static double defBudget;
00022     Buyer(const std::string n, const double b) : name(n), budget(b), alreadySpent(0) {}
00023     virtual double placeBid(const double currentPrice, const double step) const = 0;
00025     virtual void displayBuyer() const;
00028     virtual void addSpentMoney(const double i) { alreadySpent += i; }
00029     virtual double getSpentMoney() const { return alreadySpent; }
00030     virtual ~Buyer() {}
00031     std::string getName() const { return name; }
00032     double getBudget() const { return budget; }
00038     static Buyer *createBuyer(const std::string name, const double budget, const bool aggressive);
00039 };
00043 class PassiveBuyer : public Buyer
00044 {
00045 public:
00046     PassiveBuyer(const std::string n, const double b) : Buyer(n, b) {}
00051     double placeBid(const double currentPrice, const double step) const;
00052 };
00056 class AggressiveBuyer : public Buyer
00057 {
00058 public:
00059     AggressiveBuyer(const std::string n, const int b) : Buyer(n, b) {}
00064     double placeBid(const double currentPrice, const double step) const;
00065 };
00066
00067 #endif
```

5.9. hetero_store.hpp fájlreferencia

Az órai gyakorlaton elkészített tároló felhasználása a nagyháziban.

```
#include "memtrace.h"
#include <iostream>
#include <stdexcept>
```

A hetero_store.hpp definíciós fájl függési gráfja: Ez az ábra azt mutatja, hogy mely fájlok ágyazzák be közvetve vagy közvetlenül ezt a fájlt:

Osztályok

- class HeteroStore< T, s, e >

  A gyakorlaton készített tároló templát.

5.9.1. Részletes leírás

Az órai gyakorlaton elkészített tároló felhasználása a nagyháziban.

Definíció a(z) hetero_store.hpp fájlban.

5.10. hetero_store.hpp

Ugrás a fájl dokumentációjához.

```
00001
00006
00007 #ifndef HETEROSTORE_HPP
00008 #define HETEROSTORE_HPP
00009
00010 #include "memtrace.h"
00011 #include <iostream>
00012 #include <stdexcept>
00013
00018 template <class T, size_t s = 30, class e = std::out_of_range>
00019 class HeteroStore
00020 {
00021     T *data[s];
00022     size_t siz;
00023     size_t cap;
00024     HeteroStore(HeteroStore &);
00025     HeteroStore &operator+(const HeteroStore &);
00026
00027 public:
00028     HeteroStore() : siz(0), cap(s) {}
00029     size_t size() const { return siz; }
00030     size_t capacity() const { return cap; }
00031     void add(T *p)
00032     {
00033         if (siz >= cap)
00034         {
00035             delete p;
00036             throw e("Túllépi a megadott kapacitást!");
00037         }
00038         data[siz++] = p;
00039     }
00040     template <typename F>
00041     void traverse(F f)
00042     {
00043         for (size_t i = 0; i < siz; i++)
00044             f(data[i]);
00045     }
00046     void clear()
00047     {
00048         for (size_t i = 0; i < siz; i++)
00049             delete data[i];
```

```
00050         siz = 0;
00051     }
00052     T *operator[](size_t i) const
00053     {
00054         return data[i];
00055     }
00056     ~HeteroStore() { clear(); }
00057 };
00058
00059 #endif // HETEROSTORE_HPP
```

5.11. Item.h fájlreferencia

Az aukcióra bocsájtott tárgyak osztájának leírása, az alapvető függvényeivel.

`#include "memtrace.h"`
`#include <iostream>`
Az Item.h definíciós fájl függési gráfja: Ez az ábra azt mutatja, hogy mely fájlok ágyazzák be közvetve vagy közvetlenül ezt a fájlt:

Osztályok

- class Item

  Az eladott tárgyak osztálya.

Függvények

- std::ostream & operator<< (std::ostream &os, const Item &i)

  Kiírja az adott eladási tárgyat output streamre.

5.11.1. Részletes leírás

Az aukcióra bocsájtott tárgyak osztájának leírása, az alapvető függvényeivel.

Mivel a függvények rövidek, és egyértelműek ezért nem csináltam hozzá külön cpp file-t.

Definíció a(z) Item.h fájlban.

5.11.2. Függvények dokumentációja

5.11.2.1. operator<<()

```
std::ostream & operator<< (
            std::ostream & os,
            const Item & i)  [inline]
```

Kiírja az adott eladási tárgyat output streamre.

Definíció a(z) Item.h fájl 57. sorában.

```
00058 {
00059     os « i.getName() « " tárgyat " « i.getWinner() « " nyerte, "
00060        « i.getFinalPrice() « " leütési áron." « std::endl;
00061     return os;
00062 }
```

A függvény hívási gráfja:

5.12. Item.h

Ugrás a fájl dokumentációjához.

```
00001
00006 #ifndef ITEM_H
00007 #define ITEM_H
00008 #include "memtrace.h"
00009 #include <iostream>
00010
00012 class Item
00013 {
00014 protected:
00015     std::string name;
00016     double basePrice;
00017     double minIncrement;
00018     std::string winner;
00019     double finalPrice;
00020
00021 public:
00026     Item(std::string n, double startPrice, double minIncrement) : name(n), basePrice(startPrice),
      minIncrement(minIncrement) {}
00027     Item(const Item &i) : name(i.name), basePrice(i.basePrice), minIncrement(i.minIncrement) {}
00028     Item &operator=(const Item &i)
00029     {
00030         if (this != &i)
00031         {
00032             name = i.name;
00033             basePrice = i.basePrice;
00034             minIncrement = i.minIncrement;
00035         }
00036         return *this;
00037     }
00038
00040     std::string getName() const { return name; }
00042     double getBasePrice() const { return basePrice; }
00044     double getStep() const { return minIncrement; }
00046     std::string getWinner() const { return winner; }
00048     double getFinalPrice() const { return finalPrice; }
00050     void setWinner(const std::string s) { winner = s; }
00052     void setFinalPrice(const double p) { finalPrice = p; }
00053 };
00057 inline std::ostream &operator«(std::ostream &os, const Item &i)
00058 {
00059     os « i.getName() « " tárgyat " « i.getWinner() « " nyerte, "
00060        « i.getFinalPrice() « " leütési áron." « std::endl;
00061     return os;
00062 }
00063
00064 #endif
```

5.13. Tesztprogram.cpp fájlreferencia

Tesztprogram az elkészített funkciók tesztelésére.

```
#include "memtrace.h"
#include <iostream>
#include "Auction.h"
#include "gtest_lite.h"
#include <cstring>
```

A Tesztprogram.cpp definíciós fájl függési gráfja:

Függvények

- void paramBeolvas ()
- int main ()

5.13.1. Részletes leírás

Tesztprogram az elkészített funkciók tesztelésére.

Definíció a(z) Tesztprogram.cpp fájlban.

5.13.2. Függvények dokumentációja

5.13.2.1. main()

```
int main ()
```

Tesztek:

Lecseréljük az input buffert és elmentjük a régit

Lefuttatjuk a tesztet

Visszaállítjuk a régi inputbuffert

2.teszt

3.teszt

A mesterséges input

Takaritas

Generálunk vevőket, a paraméterek fixek a determinisztikus tesztelésért

Tömb a vevőknek, a 0. helyen azt számolom hányszor nem kellt el

Súlyozom az esetek számával

Eszközfüggő ezért kikapcsoltam

A licitálási szabályok finomhangolásához debug printer

Takaritas

Az eredeti seed visszaállítása

Egy próba aukció bemutatása:

Definíció a(z) Tesztprogram.cpp fájl 18. sorában.

```
00019 {
00020 #ifdef _WIN32
00021     SetConsoleOutputCP(CP_UTF8); // konzol karakterkódolás beállítása, csak windowson.
00022 #endif
00023     int seed = time(0);                         // seed beállítása, alapértelmezetten az idővel seedel
00024     srand(seed);                                // seedelés
00025     size_t buyerCount = 30;                     // globális paraméter: a vevők száma, default 30, max
      30
00026     double defBudget = 500;                     // globális paraméter: a a vevők pénze, default 500
00027     size_t turnLimit = 10;                      // globális paraméter: a körök számának limitje,
      default 10
00028     float aggressiveRatio = 1.0;                // globális paraméter: az agresszív vevők aránya az
      összes vevőhöz képest 0-1 közötti szám, default 1
00029     Auction &auction = Auction::getInstance(); // singleton inicializálás
00030
00034     TEST(Buyer, variables)
```

```
00035     {
00036         Buyer *b1 = Buyer::createBuyer("BuyerTest", 1500, 1);
00037         EXPECT_EQ(1500, b1->getBudget()) << "== Nem jo Budget! ***\n";
00038         EXPECT_STREQ("BuyerTest (Agresszív)", b1->getName().c_str()) << "== Nem jo a buyer nev! ***\n";
00039         b1->addSpentMoney(50);
00040         EXPECT_EQ(50, b1->getSpentMoney()) << "== Nem jo Spent money fv.! ***\n";
00041         Buyer *bp = Buyer::createBuyer("BuyerTest", 1500, false);
00042         EXPECT_STREQ("BuyerTest (Passzív)", bp->getName().c_str()) << "== Nem jo a factory fv.! ***\n";
00043         delete b1;
00044         delete bp;
00045     }
00046     ENDM
00047
00048     TEST(Buyer, behaviour)
00049     {
00050         Buyer *b2 = Buyer::createBuyer("BuyerTest", 1500, true);
00051         bool licital = true;
00052         for (size_t i = 0; i < 200 && licital; i++)
00053         {
00054             if (b2->placeBid(200, 50) == 0)
00055                 licital = false;
00056         }
00057         EXPECT_TRUE(licital) << "== Nem licital mindig az agressziv vevo! ==\n";
00058         delete b2;
00059     }
00060     ENDM
00061
00062     TEST(Item, variables)
00063     {
00064         Item i1("Proba1", 200, 30);
00065         EXPECT_EQ(200, i1.getBasePrice()) << "== Nem jo kezdo ertek! ***\n";
00066         EXPECT_EQ(30, i1.getStep()) << "== Nem jo az increment! ***\n";
00067         EXPECT_STREQ("Proba1", i1.getName().c_str()) << "== Nem jo az item nev! ***\n";
00068         i1.setFinalPrice(400);
00069         i1.setWinner("John");
00070         EXPECT_EQ(400, i1.getFinalPrice()) << "== Nem jo az increment! ***\n";
00071         EXPECT_STREQ("John", i1.getWinner().c_str()) << "== Nem jo az item nev! ***\n";
00072     }
00073     ENDM
00074
00075     TEST(Auction, input)
00076     {
00077         // A mesterséges input
00078         std::istringstream input("2\nPhone 300 50\nLaptop 200 100\n");
00080         std::streambuf *old = std::cin.rdbuf(input.rdbuf());
00082         EXPECT_NO_THROW(auction.readFromFile(true)) << "Nem sikerult beolvasni adatokat\n";
00084         std::cin.rdbuf(old);
00086         std::istringstream input2("3\nCamera 500 50\nYoyo 30 10\nBoardgame 10 2\n");
00087         old = std::cin.rdbuf(input2.rdbuf());
00088         EXPECT_NO_THROW(auction.readFromFile(true)) << "Nem sikerult masodjara adatot beolvasni\n";
00089         std::cin.rdbuf(old);
00091         std::istringstream input3("33\nCamera 500 50\nYoyo 30 10\nBoardgame 10 2\n");
00092         old = std::cin.rdbuf(input3.rdbuf());
00093         try
00094         {
00095             auction.readFromFile(true);
00096             FAIL() << "Nem erkezett az elvart std::out_of_range exception\n";
00097         }
00098         catch (const std::out_of_range &e)
00099         {
00100             SUCCEED();
00101         }
00102         std::cin.rdbuf(old);
00103         auction.AuctionClear();
00104     }
00105     ENDM
00106     TEST(Auction, output)
00107     {
00109         std::istringstream input("1\nCamera 600 50");
00110         std::streambuf *old = std::cin.rdbuf(input.rdbuf());
00111         EXPECT_NO_THROW(auction.readFromFile(true)) << "Nem sikerult beolvasni adatokat\n";
00112         std::cin.rdbuf(old);
00113         EXPECT_NO_THROW(auction.startAuction(aggressiveRatio, turnLimit, defBudget, buyerCount)) <<
      "Nem futott a startAuction fv.\n";
00114         std::ostringstream output;
00115         std::streambuf *oldout = std::cout.rdbuf(output.rdbuf());
00116         EXPECT_NO_THROW(auction.writeToFile(true)) << "Nem sikerult kiirni az adatokat\n";
00117         std::cout.rdbuf(oldout);
00118         std::string op = output.str();
00119         EXPECT_STREQ("Camera tárgyat senki nem nyerte, -1 leütési áron.\n", op.c_str()) << "Nem jo az
      output!\n";
00121         auction.AuctionClear();
00122     }
00123     ENDM
00124
00125     TEST(auction, logic) // A vásárlások eloszlása 30 emberre
00126     {
```

```
00127         srand(3);           // Külön seed a determinisztikus viselkedéshez
00128         size_t cases = 200; // Az esetszám beállítása, a nagyobb pontosság érdekében magasabbat
      érdemes
00129         Item itest1("Teszt", 20, 10);
00131         auction.startAuction(0.5, 10, 50000, 30);
00133         double winners[31];
00134         for (size_t i = 0; i < 30; i++)
00135         {
00136             winners[i] = 0;
00137         }
00138         size_t winner;
00139         size_t start;
00140         size_t len;
00141         // Egy sűrűségfüggvényt csinálok a winners tömbbe, hogy melyik vevő milyen gyakran nyert
00142         for (size_t i = 0; i < cases; i++)
00143         {
00144             auction.auctionItem(&itest1, 10);
00145             std::string winnerStr = itest1.getWinner();
00146             start = 5;
00147             len = 0;
00148             winner = 0;
00149             while (std::isdigit(winnerStr[start + len]))
00150             {
00151                 ++len;
00152             }
00153             if (len != 0)
00154             {
00155                 winner = std::stoi(winnerStr.substr(start, len));
00156             }
00157             winners[winner]++;
00158         }
00160         for (size_t i = 0; i < 30; i++)
00161         {
00162             winners[i] = winners[i] / cases;
00163         }
00165         // EXPECT_FLOAT_EQ(0.04, winners[1]) << "Nem determinisztikus a teszteles!";
00167         /* for (size_t i = 0; i < 30; i++)
00168         {
00169             std::cout << winners[i] << std::endl;
00170         } */
00172         auction.AuctionClear();
00174         srand(seed);
00175     }
00176     ENDM
00177
00178         if (!gtest_lite::test.fail())
00179     {
00180         std::cout << "\nMinden Teszt megfelelt" << std::endl;
00181     }
00185     try
00186     {
00187         auction.readFromFile();                                                    // beolvasás
00188         auction.startAuction(aggressiveRatio, turnLimit, defBudget, buyerCount); // A tárgyak
      elárverezése
00189     }
00190     catch (const std::exception &e) // Input parsing, filekezelési és memory allocation hibák
      fordulhatnak elő
00191     {
00192         std::cerr << "Hiba: " << e.what() << '\n';
00193     }
00194     auction.writeToFile(0); // Az árverés eredményének kiírása az itemsSold.txt tile-ba.
00195     return 0;
00196 }
```

A függvény hívási gráfja:

5.13.2.2. paramBeolvas()

```
void paramBeolvas ()
```

Definíció a(z) Tesztprogram.cpp fájl 14. sorában.

```
00015 {
00016 }
```

5.14. Tesztprogram.cpp

Ugrás a fájl dokumentációjához.

```
00001
00005 #include "memtrace.h"
00006 #include <iostream>
00007 #ifdef _WIN32
00008 #include <Windows.h>
00009 #endif
00010 #include "Auction.h"
00011 #include "gtest_lite.h"
00012 #include <cstring>
00013
00014 void paramBeolvas()
00015 {
00016 }
00017
00018 int main()
00019 {
00020 #ifdef _WIN32
00021     SetConsoleOutputCP(CP_UTF8); // konzol karakterkódolás beállítása, csak windowson.
00022 #endif
00023     int seed = time(0);                         // seed beállítása, alapértelmezetten az idővel seedel
00024     srand(seed);                                // seedelés
00025     size_t buyerCount = 30;                     // globális paraméter: a vevők száma, default 30, max
    30
00026     double defBudget = 500;                     // globális paraméter: a a vevők pénze, default 500
00027     size_t turnLimit = 10;                      // globális paraméter: a körök számának limitje,
    default 10
00028     float aggressiveRatio = 1.0;                // globális paraméter: az agresszív vevők aránya az
    összes vevőhöz képest 0-1 közötti szám, default 1
00029     Auction &auction = Auction::getInstance(); // singleton inicializálás
00030
00034     TEST(Buyer, variables)
00035     {
00036         Buyer *b1 = Buyer::createBuyer("BuyerTest", 1500, 1);
00037         EXPECT_EQ(1500, b1->getBudget()) « "== Nem jo Budget! ***\n";
00038         EXPECT_STREQ("BuyerTest (Agresszív)", b1->getName().c_str()) « "== Nem jo a buyer nev! ***\n";
00039         b1->addSpentMoney(50);
00040         EXPECT_EQ(50, b1->getSpentMoney()) « "== Nem jo Spent money fv.! ***\n";
00041         Buyer *bp = Buyer::createBuyer("BuyerTest", 1500, false);
00042         EXPECT_STREQ("BuyerTest (Passzív)", bp->getName().c_str()) « "== Nem jo a factory fv.! ***\n";
00043         delete b1;
00044         delete bp;
00045     }
00046     ENDM
00047
00048     TEST(Buyer, behaviour)
00049     {
00050         Buyer *b2 = Buyer::createBuyer("BuyerTest", 1500, true);
00051         bool licital = true;
00052         for (size_t i = 0; i < 200 && licital; i++)
00053         {
00054             if (b2->placeBid(200, 50) == 0)
00055                 licital = false;
00056         }
00057         EXPECT_TRUE(licital) « "== Nem licital mindig az agressziv vevo! ==\n";
00058         delete b2;
00059     }
00060     ENDM
00061
00062     TEST(Item, variables)
00063     {
00064         Item i1("Proba1", 200, 30);
00065         EXPECT_EQ(200, i1.getBasePrice()) « "== Nem jo kezdo ertek! ***\n";
00066         EXPECT_EQ(30, i1.getStep()) « "== Nem jo az increment! ***\n";
00067         EXPECT_STREQ("Proba1", i1.getName().c_str()) « "== Nem jo az item nev! ***\n";
00068         i1.setFinalPrice(400);
00069         i1.setWinner("John");
00070         EXPECT_EQ(400, i1.getFinalPrice()) « "== Nem jo az increment! ***\n";
00071         EXPECT_STREQ("John", i1.getWinner().c_str()) « "== Nem jo az item nev! ***\n";
00072     }
00073     ENDM
00074
00075     TEST(Auction, input)
00076     {
00077         // A mesterséges input
00078         std::istringstream input("2\nPhone 300 50\nLaptop 200 100\n");
00080         std::streambuf *old = std::cin.rdbuf(input.rdbuf());
00082         EXPECT_NO_THROW(auction.readFromFile(true)) « "Nem sikerult beolvasni adatokat\n";
00084         std::cin.rdbuf(old);
00086         std::istringstream input2("3\nCamera 500 50\nYoyo 30 10\nBoardgame 10 2\n");
00087         old = std::cin.rdbuf(input2.rdbuf());
00088         EXPECT_NO_THROW(auction.readFromFile(true)) « "Nem sikerult masodjara adatot beolvasni\n";
00089         std::cin.rdbuf(old);
```

```
00091         std::istringstream input3("33\nCamera 500 50\nYoyo 30 10\nBoardgame 10 2\n");
00092         old = std::cin.rdbuf(input3.rdbuf());
00093         try
00094         {
00095             auction.readFromFile(true);
00096             FAIL() << "Nem erkezett az elvart std::out_of_range exception\n";
00097         }
00098         catch (const std::out_of_range &e)
00099         {
00100             SUCCEED();
00101         }
00102         std::cin.rdbuf(old);
00103         auction.AuctionClear();
00104     }
00105     ENDM
00106     TEST(Auction, output)
00107     {
00109         std::istringstream input("1\nCamera 600 50");
00110         std::streambuf *old = std::cin.rdbuf(input.rdbuf());
00111         EXPECT_NO_THROW(auction.readFromFile(true)) << "Nem sikerult beolvasni adatokat\n";
00112         std::cin.rdbuf(old);
00113         EXPECT_NO_THROW(auction.startAuction(aggressiveRatio, turnLimit, defBudget, buyerCount)) <<
      "Nem futott a startAuction fv.\n";
00114         std::ostringstream output;
00115         std::streambuf *oldout = std::cout.rdbuf(output.rdbuf());
00116         EXPECT_NO_THROW(auction.writeToFile(true)) << "Nem sikerult kiirni az adatokat\n";
00117         std::cout.rdbuf(oldout);
00118         std::string op = output.str();
00119         EXPECT_STREQ("Camera tárgyat senki nem nyerte, -1 leütési áron.\n", op.c_str()) << "Nem jo az
      output!\n";
00121         auction.AuctionClear();
00122     }
00123     ENDM
00124
00125     TEST(auction, logic) // A vásárlások eloszlása 30 emberre
00126     {
00127         srand(3);           // Külön seed a determinisztikus viselkedéshez
00128         size_t cases = 200; // Az esetszám beállítása, a nagyobb pontosság érdekében magasabbat
      érdemes
00129         Item itest1("Teszt", 20, 10);
00131         auction.startAuction(0.5, 10, 50000, 30);
00133         double winners[31];
00134         for (size_t i = 0; i < 30; i++)
00135         {
00136             winners[i] = 0;
00137         }
00138         size_t winner;
00139         size_t start;
00140         size_t len;
00141         // Egy sűrűségfüggvényt csinálok a winners tömbbe, hogy melyik vevő milyen gyakran nyert
00142         for (size_t i = 0; i < cases; i++)
00143         {
00144             auction.auctionItem(&itest1, 10);
00145             std::string winnerStr = itest1.getWinner();
00146             start = 5;
00147             len = 0;
00148             winner = 0;
00149             while (std::isdigit(winnerStr[start + len]))
00150             {
00151                 ++len;
00152             }
00153             if (len != 0)
00154             {
00155                 winner = std::stoi(winnerStr.substr(start, len));
00156             }
00157             winners[winner]++;
00158         }
00160         for (size_t i = 0; i < 30; i++)
00161         {
00162             winners[i] = winners[i] / cases;
00163         }
00165         // EXPECT_FLOAT_EQ(0.04, winners[1]) << "Nem determinisztikus a teszteles!";
00167         /* for (size_t i = 0; i < 30; i++)
00168         {
00169             std::cout << winners[i] << std::endl;
00170         } */
00172         auction.AuctionClear();
00174         srand(seed);
00175     }
00176     ENDM
00177
00178         if (!gtest_lite::test.fail())
00179     {
00180         std::cout << "\nMinden Teszt megfelelt" << std::endl;
00181     }
00185     try
00186     {
```

```
00187         auction.readFromFile();                                                    // beolvasás
00188         auction.startAuction(aggressiveRatio, turnLimit, defBudget, buyerCount); // A tárgyak
      elárverezése
00189     }
00190     catch (const std::exception &e) // Input parsing, filekezelési és memory allocation hibák
      fordulhatnak elő
00191     {
00192         std::cerr « "Hiba: " « e.what() « '\n';
00193     }
00194     auction.writeToFile(0); // Az árverés eredményének kiírása az itemsSold.txt tile-ba.
00195     return 0;
00196 }
```

Tárgymutató