HÁZI FELADAT

Programozás alapjai 2.

Végleges Dokumentáció

Ádám Zsombor
X079FB

2022. május 16.

TARTALOM

1. Feladat 3
2. Feladatspecifikáció 3
3. Pontosított feladatspecifikáció 3
4. Terv 4
   4.1. Objektum terv 4
   4.2. Algoritmusok 5
      4.2.1. Havidíj 5
      4.2.2. Beolvasás 5
      4.2.3. Kiírás 5
      4.2.4. Tesztprogram algoritmusai. 6
5. Megvalósítás 6
   5.1. Az elkészített osztályok bemutatása 6
      5.1.1. Alap osztály 6
      5.1.2. Azonosithato osztály 7
      5.1.3. Forgalom osztály 8
      5.1.4. Lista osztály 10
      5.1.5. MobilNet osztály 12
      5.1.6. SMSMax osztály 13
      5.1.7. String osztály 14
      5.1.8. Ugyfel osztály 16
   5.2. További osztályokon kívüli globális függvények 18
6. Tesztelés 19
   6.1. Memóriakezelés tesztje 19
   6.2. Lefedettségi teszt 19

7. Mellékletek 20
7.1. dijcsomagok.h 20
7.2. lista.h 21
7.3. ugyfel.h 22
7.4. azonosithato.h 23
7.5. forgalom.h 23
7.6. string5.h 24

1. Feladat

Egy mobilszolgáltatónál egy egyedi nyilvántartó programmal szeretnék kezelni az ügyfeleket. Az ügyfeleknek van neve és címe, valamint telefonszáma, ami egyben az egyedi azonosítjuk is. A szolgáltató jelenleg három csomagot biztosít ügyfeleinek: Alap, MobilNet és SMSMax, de később több csomag is lehet. Minden csomaghoz más percdíj és SMS díj tartozik, valamint a számítás módszere is eltérő lehet. A MobilNet csomag esetén pl. az is megadható, hogy hány SMS-t küldhet az ügyfél ingyen. A program egy fájlból olvassa be az ügyfelek adatait és választott díjcsomagot. Egy másik fájlból pedig az adott hónapban küldött SMS darabszámot és a lebeszélt perceket. A program írja ki, hogy az egyes ügyfelek mennyit fizetnek a forgalom alapján.
A megoldáshoz **ne** használjon STL tárolót

2. Feladatspecifikáció

A feladat egy mobilszolgáltató nyilvántartó programjának az elkészítése. A mobilszolgáltató szeretné kezelni az ügyfelek adatait, díjcsomagját és hogy mennyi SMS-t küldtek, illetve mennyit telefonáltak adott hónapban (havonta új fájl). Ezt a két eltérő adathalmazt két külön fájlból fogja a program beolvasni. Az ügyfelek adatai a következők: név, cím, telefonszám (vagy egyedi azonosító).

A szolgáltatónak különböző csomagjai vannak különböző ügyfelek számára és ezek alapján számolja ki a program, hogy a forgalom alapján mennyit fizetnek a hónapban. Eleinte 3 csomag van, melyek a következők: Alap, MobilNet és SMSMax. Ezek később bővíthetők más csomagokkal. Minden csomagnak más a perc és az SMS díja, valamint számítási módszere is eltérő lehet.

Az osztályok működéséhez szükség lesz egy String osztályra is, mivel nem használhatunk STL tárolókat. Így a feladat megvalósításához szükséges minden stringműveletet meg fogok valósítani.

A program kiírja, hogy egyes ügyfeleknek mennyit kell az adott hónapban fizetnie a forgalma alapján.

A teszteléséhez egy tesztprogramot készítek. A tesztadatok között hiba is elő fog fordulni. Minden nem megfelelő bejövő adat esetére a program exceptiont fog dobni.

3. Pontosított feladatspecifikáció

A feladat egy mobilszolgáltató nyilvántartó programjának az elkészítése. A mobilszolgáltató szeretné kezelni az ügyfelek adatait, díjcsomagját és hogy mennyi SMS-t küldtek, illetve mennyit telefonáltak adott hónapban (havonta új fájl). Ezt a két eltérő adathalmazt két külön fájlból fogja a program beolvasni. Az ügyfelek adatai a következők: név, cím, telefonszám (vagy egyedi azonosító).

A szolgáltatónak különböző csomagjai vannak különböző ügyfelek számára és ezek alapján számolja ki a program, hogy a forgalom alapján mennyit fizetnek a hónapban.

Eleinte 3 csomag van, melyek a következők: Alap, MobilNet és SMSMax. Ezek később bővíthetők más csomagokkal. Minden csomagnak más a perc és az SMS díja, valamint számítási módszere is eltérő lehet.

Az osztályok működéséhez szükség lesz egy String osztályra is, mivel nem használhatunk STL tárolókat. Így a feladat megvalósításához szükséges minden sztring műveletet meg fogunk valósítani.

A program kiírja, hogy egyes ügyfeleknek mennyit kell az adott hónapban fizetnie a forgalma alapján, illetve az alábbi műveletekkel lehet módosítani az adatbázisunkat:

- ügyfél felvétele
- ügyfél törlése
- ügyfél adatainak módosítása (beleértve a beszélt perceket és küldött SMS-eket)

A teszteléséhez egy tesztprogramot készítünk. A tesztadatok között hiba is elő fog fordulni. Minden nem megfelelő bejövő adat esetére a program exceptiont fog dobni.

4. Terv

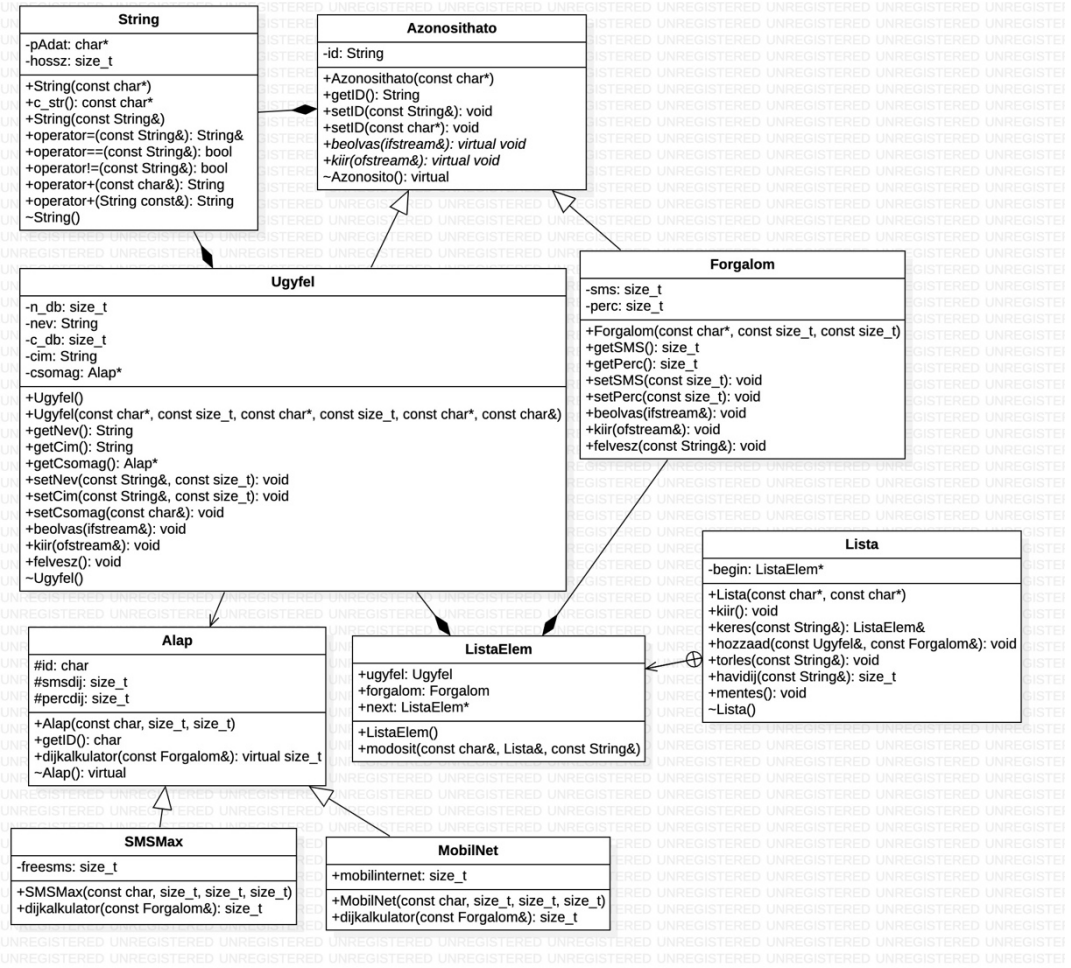
A feladat több objektum és a tesztprogram megtervezését igényli.

4.1. Objektum terv

Van egy alap Azonosithato osztály, amiből leszármaznak az Ugyfel és a Forgalom osztályok. Ezek azért vannak külön objektumban, mivel két külön fájlból kell beolvasni és örökíthetőek. Az Ugyfel osztály egyik privát adattagja a díjcsomagra mutató pointer, amit szintén leszármaztatható az Alap ősosztályból, melyből a két másik díjcsomag származtatik le.

A Lista osztály, ami egy láncolt lista az adatok könnyebb kezelhetősége érdekében lett létrehozva, hogy összefűzze az Ugyfel és Forgalom osztályokat, illetve, hogy könnyebb legyen az ügyfeleket hozzáadni és törölni az adatbázisból. Az Ugyfel és a Forgalom osztályokat a Lista osztályon belül található ListaElem struktúrában tárolja a program. A Listának a közvetlen privát adattagja csupán a kezdeti elemre mutató pointer. A ListaElemek továbbá tárolják a következő elemre mutató pointert is.

Ezeken kívül még létre kell hozni egy saját String osztályt, amiben csak az elengedhetetlen függvényeket deklaráljuk.



4.2. Algoritmusok

4.2.1. Havidíj

A havidij() függvény kiszámolja a lista megfelelő elemének, hogy mennyit kell az adott hónapban fizetnie az ügyfélnek a díjcsomagja alapján a díjkalkulátor() függvény segítségével, amely minden csomagban egyedien számolja ki a havi összeget.

4.2.2. Beolvasás

A beolvasó függvény a neki megadott fájlnevet nyitja meg és abból olvassa ki az adatokat, amelyek ügyfelenként sorokba van rendezve és egy ügyfél különböző adatait szóköz választja el.
A beolvasott adatokból láncolt listát épít fel.
Bármikor, amikor nem olyan típusú adatot kap a függvény, amire számít, akkor exceptiont dob.

4.2.3. Kiírás

A beolvasó függvényhez hasonlóan működik, csak a fájlba írja az adatokat, ugyanolyan formában, mint korábban. Hibakezelésre itt csak a fájl nevének a megadásánál van szükség, egyébként korábban már lezajlott.

4.2.4. Tesztprogram algoritmusai.

A tesztprogram több adatra teszteli a beolvasás, adatmódosítás, havidíj számolás és kiírás mechanizmusait. Illetve valamennyi exceptiont is vizsgál.

5. Megvalósítás

A feladat megoldása 6 osztály és egy osztályon belüli struktúra elkészítésével valósult meg. Az osztályok interfésze, mind privát adattagjaiban, mind publikus függvényeiben is változtak az eredetileg tervezetthez képest, mivel megvalósítás során ezek a megoldások tűntek kézenfekvőbbnek.

A díjcsomag osztályok a *dijcsomagok.h (dijcsomagok.cpp)*, a sztring tároló a *string5.h (string5.cpp)*, az Azonosithato osztály az *azonosithato.h*, az Ugyfel osztály az *ugyfel.h (ugyfel.cpp)*, a Forgalom osztály a *forgalom.h (forgalom.cpp)*, illetve a Lista osztály és ListaElem struktúra a *lista.h (lista.cpp)* forrás fájlokban találhatóak meg. A tesztprogramok, illetve a felhasználói felület a *teszt_main.cpp* fájlban lett elkészítve. A főprogram tartalmaz egy felhasználói felületet is, ahol lehet kezelni az adatbázist.

A továbbiakban bemutatom a fontosabb interfészeket és algoritmusokat a program forrása alapján generált dokumentáció felhasználásával.

5.1. Az elkészített osztályok bemutatása

5.1.1. Alap osztály

Alap díjcsomagosztály, amiből a többi származik le (akár bővíthető tovább is)

5.1.1.1 Protected adattagok

- char **id**
  Kódja a díjcsomagnak (egy karakter).

- size_t **smsdij**
  Egy SMS díja.

- size_t **percdij**
  Egy megkezdett percnyi telefonbeszélgetés díja.

5.1.1.2 Konstruktor

Alap::Alap (const char & ***id*** **= 'A', size_t** ***sms*** **= 40, size_t** ***perc*** **= 20)[inline], [explicit]**

Paraméterek

id	Díjcsomag azonosító referenciája.
sms	Az SMS díj értéke a csomagban.

perc	A Percdíj értéke a csomagban.

5.1.1.3 Tagfüggvények

virtual size_t Alap::dijkalkulator (const Forgalom & *forg*) const[inline], [virtual]

Kiszámítja a havi **Forgalom** és a Díjcsomag alapján, hogy mennyit kell fizetnie az Ügyfélnek

Paraméterek

forg	A havi **Forgalom** referenciája.

Visszatérési érték

A havidíj értéke egy Ügyfélnek a csomagban.

char Alap::getID () const[inline]

Visszaadja a díjcsomag kódját

Visszatérési érték

Azonosítója a díjcsomagnak.

5.1.2. Azonosithato osztály

Ősosztálya az Ügyfélnek és a Forgalomnak, mivel mindkettő ugyanazzal az azonosítóval működik. Absztrakt osztály.

5.1.2.1 Privát adattagok

- String **id**
  Azonosítója az Ügyfél és a Forgalom osztályoknak.

5.1.2.2 Konstruktor

Azonosithato::Azonosithato (const char * *azon* = "")[inline], [explicit]

Paraméterek

azon	Azonosítóra mutató pointer.

5.1.2.3 Tagfüggvények

virtual void Azonosithato::beolvas (std::ifstream & *fajl*)[pure virtual]

Virtuális beolvasó függvény, amely referenciaként kapott fájlból olvassa be az osztály létrehozásához szükséges adatokat. A leszármazottak határozzák meg ennek a belső működését, mivel önmagában nincs értelme ennél az alaposztálynál.

Paraméterek

fajl	A fájlból beolvasó stream referenciája.

String Azonosithato::getID () const[inline]

Visszaadja az id Stringet.

Visszatérési érték

Azonosító értéke.

virtual void Azonosithato::kiir (std::ofstream & *fajl*) const[pure virtual]

Virtuális kiíró függvény, amely referenciaként kapott fájlba írja be az osztályban lévő adatokat. A leszármazottak határozzák meg ennek a belső működését, mivel önmagában nincs értelme ennél az alaposztálynál.

Paraméterek

fajl	A fájlba kiíró stream referenciája.

void Azonosithato::setID (const char * *azon*)[inline]

Beállítja az id Stringet.

Paraméterek

azon	Azonosítóra mutató pointer.

void Azonosithato::setID (const String & *azon*)[inline]

Beállítja az id Stringet.

Paraméterek

azon	Azonosító referenciája.

5.1.3. Forgalom osztály

Egy adott ügyfélnek az egy havi forgalmát tárolja az alábbi osztály, elküldött sms-ekben és lebeszélt percekben. Az **Azonosithato** osztályból származik le.

5.1.3.1 Privát adattagok

- size_t **sms**
  Elküldött SMS-ek száma.

- size_t **perc**
  Lebeszélt percek száma.

5.1.3.2 Konstruktor

Forgalom::Forgalom (const char * *id* = "", const size_t *db* = 0, const size_t *le* = 0)[inline], [explicit]

Paraméterek

id	Az azonosítóra mutató pointer.
db	A havi elküldött SMS-ek száma.
le	A havi lebeszélt percek száma.

5.1.3.3 Tagfüggvények

void Forgalom::beolvas (std::ifstream & *fajl*)[override], [virtual]

Beolvassa a fájlból az osztályhoz szükséges adatokat.

Paraméterek

fajl	A fájlból kiolvasó stream referenciája.

void Forgalom::felvesz (const String & *id*)

Forgalom felvétele.

Paraméterek

id	A fájlba kiíró stream referenciája.

Kivételek

std::invalid_argument	Ha ahol számot vár a függvény, ott nem számot kap.

size_t Forgalom::getPerc () const[inline]

Visszaadja a lebeszélt percek számát.

Visszatérési érték

A havi lebeszélt percek száma.

size_t Forgalom::getSMS () const[inline]

Visszaadja az elküldött SMS-ek számát.

Visszatérési érték

A havi elküldött SMS-ek száma.

void Forgalom::kiir (std::ofstream & *fajl*) const[override], [virtual]

Kiírja a fájlba az osztályba lévő adatokat.

Paraméterek

fajl	A fájlból beolvasó stream referenciája.

void Forgalom::setPerc (const size_t *db*)[inline]

Beállítja a lebeszélt percek számát.

Paraméterek

db	A havi lebeszélt percek száma.

void Forgalom::setSMS (const size_t *db*)[inline]

Beállítja az elküldött sms-ek számát.

Paraméterek

db	A havi elküldött SMS-ek száma.

5.1.4. Lista osztály

Láncolt **Lista** osztály mindkét fajta adathalmaz tárolásásra.

5.1.4.1 Privát adattagok

- ListaElem struktúra:
    - Ugyfel **ugyfel**
      Egy Ügyfél adatai.
    - Forgalom **forgalom**
      A fentebbi Ügyfél forgalma az adott hónapban.
    - ListaElem***** **next**
      A következő elemre mutató pointer (nullptr, ha nincs következő).

- ListaElem***** **begin**
  Lista kezdetére mutató pointer.

5.1.4.2 Privát tagfüggvényei a ListaElem struktúrában a Listának

String Lista::ListaElem::modosit (const char& m, Lista& lista, const String& id)

ListaElem adatainak a módosítását végzi.

Paraméterek

m	A módosítani kívánt része a ListaElemnek, amit egy char referenciával döntünk el.
lista	A lista referenciája, amiben módosítjuk az elemet.
id	Az azonosító referenciája, hogy melyik ListaElemet kívánjuk módosítani.

Kivételek

std::logic_error	Ha létezik már az azonosító
std::invalid_argument	Ha rossz a névnek vagy a címnek a megadása, vagy ha szám helyett nem számot kap, vagy ha hibás az input karakter.

5.1.4.3 Konstruktor

Lista::Lista (const char * *u_fajl*, const char * *f_fajl*)

Paraméterek

u_fajl	Az Ügyfeleket tartalmazó fájl névre mutató pointer.
f_fajl	A havi Forgalmat tartalmazó fájl névre mutató pointer.

Kivételek

std::invalid_argument	Ha rossz fájlokat kapott a függvény.

5.1.4.4 Tagfüggvények

size_t Lista::havidij (const String & *azon*)[inline]

Megadja egy adott Ügyfél havidíját a Díjcsomagja és a Forgalma alapján a Listában (azonosító alapján).

Paraméterek

azon	Az Ügyfél azonosítója, akinek a havi forgalmát szeretnénk megtudni.

Visszatérési érték

Az össz havidíj értéke.

void Lista::hozzaad (const Ugyfel & *ugy*, const Forgalom & *forg*)

Hozzáad a Listához egy ListaElemet.

Paraméterek

ugy	Az Ügyfél referenciája, akit hozzá kívánunk adni az adaatbázishoz.
forg	A havi **Forgalom** refernciája.

Kivételek

std::logic_error	Ha nem egyezik az Ügyfél és a **Forgalom** osztály azonosítója vagy ha már van ilyen azonosító.

Lista::ListaElem & Lista::keres (const String & *keres*)

Megkeres egy ListaElemet a Listából (azonosító alapján)

Paraméterek

keres	A keresendő ListaElem azonosító referenciája.

Visszatérési érték

A megtalált ListaElem referenciája.

Kivételek

std::invalid_argument	Ha nem található az azonosító.

void Lista::kiir () const

Kiírja a **Lista** összes elemét.

void Lista::mentes () const

Elmenti a **Lista** elemeit az eredeti fájlokba.

void Lista::torol (const String & *torles*)

Töröl egy elemet a Listából (azonosító alapján).

Paraméterek

torles	A törlendő Ügyfél azonosító referenciája.

Kivételek

std::invalid_argument	Ha nem található az azonosító.

5.1.5. MobilNet osztály

MobilNet díjcsomagosztály, amelyben van egy mobilinternet alapdíj. Az **Alap** díjcsomagosztály leszármazottja.

5.1.5.1 Konstruktor

MobilNet::MobilNet (const char & *id* = 'M', size_t *sms* = 60, size_t *perc* = 40, size_t *mint* = 1500)[inline], [explicit]

Paraméterek

id	Díjcsomag azonosító referenciája.
sms	Az SMS díj értéke a csomagban.
perc	A Percdíj értéke a csomagban.
mint	Az alapdíja a csomagnak.

5.1.5.2 Tagfüggvények

size_t MobilNet::dijkalkulator (const Forgalom & *forg*) const[inline], [override], [virtual]

Dijkalkulátor átdefiniálása.

Paraméterek

forg	A havi **Forgalom** referenciája.

Visszatérési érték

A havidíj értéke egy Ügyfélnek a csomagban.

5.1.6. SMSMax osztály

SMSMax dícsomagosztály, amely egy bizonyos mennyiségű ingyen sms-t ad egy adott hónapban. Az **Alap** díjcsomagosztály leszármazottja.

5.1.6.1 Konstruktor

SMSMax::SMSMax (const char & ***id*** **= 'S', size_t** ***sms*** **= 30, size_t** ***perc*** **= 30, size_t** ***db*** **= 50)[inline], [explicit]**

Paraméterek

id	Díjcsomag azonosító referenciája.
sms	Az SMS díj értéke a csomagban.
perc	A Percdíj értéke a csomagban.
db	Ingyen kÃ¼ldhetÅ‘ SMS-ek szÃ¡ma a csomagban.

5.1.6.2 Tagfüggvények

size_t SMSMax::dijkalkulator (const Forgalom & ***forg*****) const[inline], [override], [virtual]**

Dijkalkulátor átdefiniálása.

Paraméterek

forg	A havi **Forgalom** referenciája.

Visszatérési érték

A havidíj értéke egy Ügyfélnek a csomagban.

5.1.7. String osztály

A **String** osztály. A 'pData'-ban vannak a karakterek (a lezáró nullával együtt), 'len' a hossza. A hosszba nem számít bele a lezáró nulla.

5.1.7.1 Konstruktor

String::String (char const * *s* = "")[explicit]

Konstruktor

Paraméterek

s	A tárolandó Stringre mutató pointer.

String::String (String const & *copy*)

Másoló konstruktor

Paraméterek

copy	A lemásolandó **String** referenciája.

5.1.7.2 Tagfüggvények

const char * String::c_str () const[inline]

C-sztringre mutató pointert ad vissza.

Visszatérési érték

A **String** értékére mutató pointer.

bool String::operator!= (const String & *rhs*) const

Egyenlőtlenség operator.

Paraméterek

rhs	Az egyenlőtlenség vizsgálat jobb oldalán álló **String** referenciája.

Visszatérési érték

Egyenlőtlen-e? (bool)

String String::operator+ (const char & *s*) const

Sztringhez karaktert fűz.

Paraméterek

s	A Stringhez fűzendő karakter referenciája.

Visszatérési érték

A **String** értéke.

String String::operator+ (String const & *s*) const

Összefűzés Stringet Stringgel.

Paraméterek

s	A Stringhez fűzendő **String** referenciája.

Visszatérési érték

A **String** értéke.

String & String::operator= (String const & *rhs*)

Értékadó operator.

Paraméterek

rhs	A lemásolandó **String** referenciája.

Visszatérési érték

A lemásolt **String** referenciája (láncolhatóság miatt).

bool String::operator== (const String & *rhs*) const

Egyenlőség operator

Paraméterek

rhs	Az egyenlőség vizsgálat jobb oldalán álló **String** referenciája.

Visszatérési érték

Egyenlő-e? (bool)

5.1.8. Ugyfel osztály

Egy adott **Ügyfél** adatait tároló osztály, amely az **Azonosítható** osztályból származik le.
Tárolja: hány darab névből áll a teljes neve (nevek, amelyekben kötőjel szerepel egynek számít pl: Moholy-Nagy), teljes nevet, teljes lakcímet, illetve a díjcsomagját.

5.1.8.1 Konstruktor

Ugyfel::Ugyfel (const char * *id*, const size_t *N*, const char * *name*, const size_t *C*, const char * *add*, const char & *c*)[inline], [explicit]

Paraméteres konstruktor

Paraméterek

id	Az azonosítóra mutató pointer.
N	A név részeinek száma.
name	A teljes névre mutató pointer.
C	A cím részeinek a száma.
add	A teljes címre mutató pointer.
c	A Díjcsomag karakter kód referenciája

5.1.8.2 Tagfüggvények

void Ugyfel::beolvas (std::ifstream & *fajl*)[override], [virtual]

Beolvassa a fájlból az osztályhoz szükséges adatokat.

Paraméterek

fajl	A fájlból beolvasó stream referenciája.

Kivételek

std::invalid_argument	Ha rossz a névnek vagy a címnek a megadása.

void Ugyfel::felvesz ()

Ügyfél felvétele.

Kivételek

std::invalid_argument	Ha rossz a névnek vagy a címnek a megadása.

String Ugyfel::getCim () const[inline]

Visszaadja a címet.

Visszatérési érték

A teljes cím.

Alap * Ugyfel::getCsomag () const[inline]

Visszaadja a csomagot.

Visszatérési érték

A díjcsomagra mutató pointer.

String Ugyfel::getNev () const[inline]

Visszaadja a nevet.

Visszatérési érték

A teljes név.

void Ugyfel::kiir (std::ofstream & *fajl*) const[override], [virtual]

Kiírja a fájlba az osztályban lévő adatokat.

Paraméterek

fajl	A fájlba kiíró stream referenciája.

Ugyfel & Ugyfel::operator= (const Ugyfel & *rhs*)

Értékadó operator.

Paraméterek

rhs	A másolandó Ügyfél osztály referenciája.

Visszatérési érték

A lemásolt Ügyfél osztály referenciája.

void Ugyfel::setCim (const String & *str*, const size_t *c*)[inline]

Beállítja a címet.

Paraméterek

str	A teljes cím referenciája.
c	A cím részeinek száma.

void Ugyfel::setCsomag (const char & *c*)

Beállítja a csomagot.

Paraméterek

c	A díjcsomag karakter kódjának referenciája.

Kivételek

std::invalid_argument	Ha nem létezik a díjcsomag vagy rossz kódot kapott a függvény.

void Ugyfel::setNev (const String & *str*, const size_t *n*)[inline]

Beállítja a nevet.

Paraméterek

str	A teljes név referenciája.
n	A név részeinek száma.

5.2. További osztályokon kívüli globális függvények

std::ostream& operator<< (std::ostream& os, const Forgalom& forg)

Paraméterek

os	A kiíró stream referenciája.
forg	A kiírandó havi Forgalom referenciája.

Visszatérési érték

A kiíró stream referenciája (láncolhatóság miatt).

std::ostream& operator<< (std::ostream& os, const Ugyfel& ugy)

Paraméterek

os	A kiíró stream referenciája.
ugy	A kiírandó Ügyfél referenciája.

Visszatérési érték

A kiíró stream referenciája (láncolhatóság miatt).

std::ostream& operator<< (std::ostream& os, const String& str)

Paraméterek

os	A kiíró stream referenciája.
str	A kiírandó String referenciája.

Visszatérési érték

A kiíró stream referenciája (láncolhatóság miatt).

std::istream& operator>> (std::istream& is, String& s0)

Paraméterek

is	A beolvasó stream referenciája.
s0	A beolvasandó String referenciája.

Visszatérési érték

A beolvasó stream referenciája (láncolhatóság miatt).

6. Tesztelés

A teszteket a *gtest_lite.h*-ban található függvényekkel valósítottam meg. Mindegyik fontosabb osztálynak, a fontosabb függvényeihez írtam teszteseteket, amelyek kivételeket is tesztelnek.

- Test1: String osztály tesztelése
- Test2: Ügyfél osztály tesztelése
- Test3: Forgalom osztály tesztelése
- Test4: Lista osztály tesztelése
- Test5: Díjcsomag osztályok tesztelése

A kivételeket amiket használ a program az a C++ standard kivételosztályai. Ezek mellett még standard bemenettel is teszteltem a programot saját gépen, illetve a Jporta rendszerben.

6.1. Memóriakezelés tesztje

A memóriakezelés ellenőrzését a laborgyakorlatokon használt MEMTRACE modullal végeztem. Ehhez minden önálló fordítási egységben include-oltam a ”memtrace.h” állományt a standard fejlécállományok után. Memóriakezelési hibát nem tapasztaltam a futtatások során.

6.2. Lefedettségi teszt

A tesztek a program minden ágát lefedték.
Kivételt képeznek az alábbi sorok:

- A bemeneti fájlok hibája miatti eldobó függvények.
- Az üres listához elemet hozzáadó függvény.
- A főprogram végén lévő sikertelen memória foglalás miatti kivételeket elkapó függvény, illetve az egyéb kivételeket elkapó függvény.

7. Mellékletek

7.1. dijcsomagok.h

```
1 #ifndef MOBILSZOL_DIJCSOMAGOK_H
2 #define MOBILSZOL_DIJCSOMAGOK_H
3
4 #include "forgalom.h"
5
7 class Alap{
8 protected:
9     char id;
10     size_t smsdij;
11     size_t percdij;
12 public:
18     inline explicit Alap(const char& id = 'A', size_t sms = 40, size_t perc = 20)
:id(id), smsdij(sms), percdij(perc) {}
19
23     inline char getID() const{ return id;}
24
29     virtual inline size_t dijkalkulator(const Forgalom& forg) const{
30         size_t sum = (forg.getSMS() * smsdij) + (forg.getPerc() * percdij);
31         return sum;
32     }
33
35     virtual ~Alap() = default;
36 };
37
39 class SMSMax :public Alap{
40     size_t freesms;
41 public:
48     inline explicit SMSMax(const char& id = 'S', size_t sms = 30, size_t perc = 30,
size_t db = 50)
49             :Alap(id, sms, perc), freesms(db) {}
50
55     inline size_t dijkalkulator(const Forgalom& forg) const override{
56         size_t sum = 0;
57         if (((int)forg.getSMS() - (int)freesms) >= 0)
58             sum = forg.getSMS() - freesms;
59         sum = (sum * smsdij) + (forg.getPerc() * percdij);
60         return sum;
61     }
62 };
63
65 class MobilNet :public Alap{
66     size_t mobilinternet;
67 public:
74     inline explicit MobilNet(const char& id = 'M', size_t sms = 60, size_t perc =
40, size_t mint = 1500)
75             :Alap(id, sms, perc), mobilinternet(mint) {}
76
81     inline size_t dijkalkulator(const Forgalom& forg) const override{
82         size_t sum = mobilinternet + (forg.getSMS() * smsdij) + (forg.getPerc() *
percdij);
83         return sum;
84     }
85 };
86
87 #endif //MOBILSZOL_DIJCSOMAGOK_H
```

7.2. lista.h

```
1 #ifndef MOBILSZOL_LISTA_H
2 #define MOBILSZOL_LISTA_H
3
4 #include "ugyfel.h"
5 #include "forgalom.h"
6
8 class Lista{
12     struct ListaElem{
13         Ugyfel ugyfel;
14         Forgalom forgalom;
15         ListaElem* next;
16
18         ListaElem() :next(nullptr) {
19             ugyfel = Ugyfel();
20             forgalom = Forgalom();
21         }
22
31         String modosit(const char& m, Lista& lista, const String& id);
32     };
33
34     ListaElem* begin;
35 public:
41     Lista(const char* u_fajl, const char* f_fajl);
42
44     void kiir() const;
45
51     ListaElem& keres(const String& keres);
52
58     void hozzaad(const Ugyfel& ugy, const Forgalom& forg);
59
64     void torol(const String& torles);
65
70     inline size_t havidij(const String& azon){
71         ListaElem* ptr = &keres(azon);
72         size_t sum = ptr->ugyfel.getCsomag()->dijkalkulator(ptr->forgalom);
73         return sum;
74     }
75
77     void mentes() const;
78
80     ~Lista();
81 };
82
83 #endif //MOBILSZOL_LISTA_H
```

7.3. ugyfel.h

```
1 #ifndef MOBILSZOL_UGYFEL_H
2 #define MOBILSZOL_UGYFEL_H
3
4 #include "azonosithato.h"
5 #include "dijcsomagok.h"
6
11 class Ugyfel :public Azonosithato{
12     size_t n_db;
13     String nev;
14     size_t c_db;
15     String cim;
16     Alap* csomag;
17 public:
19     inline Ugyfel() :n_db(0), nev(), c_db(0), cim(), csomag(nullptr) {}
28     inline explicit Ugyfel(const char* id, const size_t N, const char* name, const
size_t C,
29                     const char* add, const char& c)
30             :Azonosithato(id), n_db(N), nev(name), c_db(C), cim(add),
csomag(nullptr) {
31         setCsomag(c);
32     }
33
38     Ugyfel& operator=(const Ugyfel& rhs);
39
43     inline String getNev() const{ return nev;}
44
48     inline String getCim() const{ return cim;}
49
53     inline Alap* getCsomag() const{ return csomag;}
54
59     inline void setNev(const String& str, const size_t n) { nev = str; n_db = n;}
60
65     inline void setCim(const String& str, const size_t c) { cim = str; c_db = c;}
66
71     void setCsomag(const char& c);
72
77     void beolvas(std::ifstream& fajl) override;
78
82     void kiir(std::ofstream& fajl) const override;
83
87     void felvesz();
88
90     inline ~Ugyfel() override{
91         if (csomag != nullptr)
92             delete csomag;
93     }
94 };
95
102 std::ostream& operator<<(std::ostream& os, const Ugyfel& ugy);
103
104 #endif //MOBILSZOL_UGYFEL_H
```

7.4. azonosithato.h

```
1 #ifndef MOBILSZOL_AZONOSITHATO_H
2 #define MOBILSZOL_AZONOSITHATO_H
3
4 #include "string5.h"
5
7 class Azonosithato{
8     String id;
9 public:
13     inline explicit Azonosithato(const char* azon = "") :id(azon) {}
14
18     inline String getID() const{ return id;}
19
23     inline void setID(const String& azon) { id = azon;}
27     inline void setID(const char* azon) { String uj(azon); id = uj;}
28
35     virtual void beolvas(std::ifstream& fajl) = 0;
36
43      virtual void kiir(std::ofstream& fajl) const = 0;
44
46      virtual ~Azonosithato() = default;
47 };
48
49 #endif //MOBILSZOL_AZONOSITHATO_H
```

7.5. forgalom.h

```
1 #ifndef MOBILSZOL_FORGALOM_H
2 #define MOBILSZOL_FORGALOM_H
3
4 #include "azonosithato.h"
5
7 class Forgalom :public Azonosithato{
8     size_t sms;
9     size_t perc;
10 public:
16     inline explicit Forgalom(const char* id = "", const size_t db = 0, const size_t
le = 0)
17             :Azonosithato(id), sms(db), perc(le) {}
18
22     inline size_t getSMS() const{ return sms;}
23
27     inline size_t getPerc() const{ return perc;}
28
32     inline void setSMS(const size_t db) { sms = db;}
33
37     inline void setPerc(const size_t db) { perc = db;}
38
42     void beolvas(std::ifstream& fajl) override;
43
47     void kiir(std::ofstream& fajl) const override;
48
53     void felvesz(const String& id);
54 };
55
61 std::ostream& operator<<(std::ostream& os, const Forgalom& forg);
62
63 #endif //MOBILSZOL_FORGALOM_H
```

7.6. string5.h

```
#ifndef STRING_H
2 #define STRING_H
3
9 class String {
10     char *pData;
11     size_t len;
12 public:
16     explicit String(char const* s = "");
17
21     inline const char* c_str() const { return pData;}
22
26     String(String const& copy);
27
32     String& operator=(String const& rhs);
33
38     bool operator==(const String& rhs) const;
39
44     bool operator!=(const String& rhs) const;
45
50     String operator+(const char& s) const;
51
56     String operator+(String const& s) const;
57
59     inline ~String() { delete[] pData;}
60 };
61
67 std::ostream& operator<<(std::ostream& os, String const& str);
68
74 std::istream& operator>>(std::istream& is, String& s0);
75
76 #endif
```