Nagyházi specifikáció Goldschmidt Olivér

RXL623

Feladat:

Mobil szolgáltató

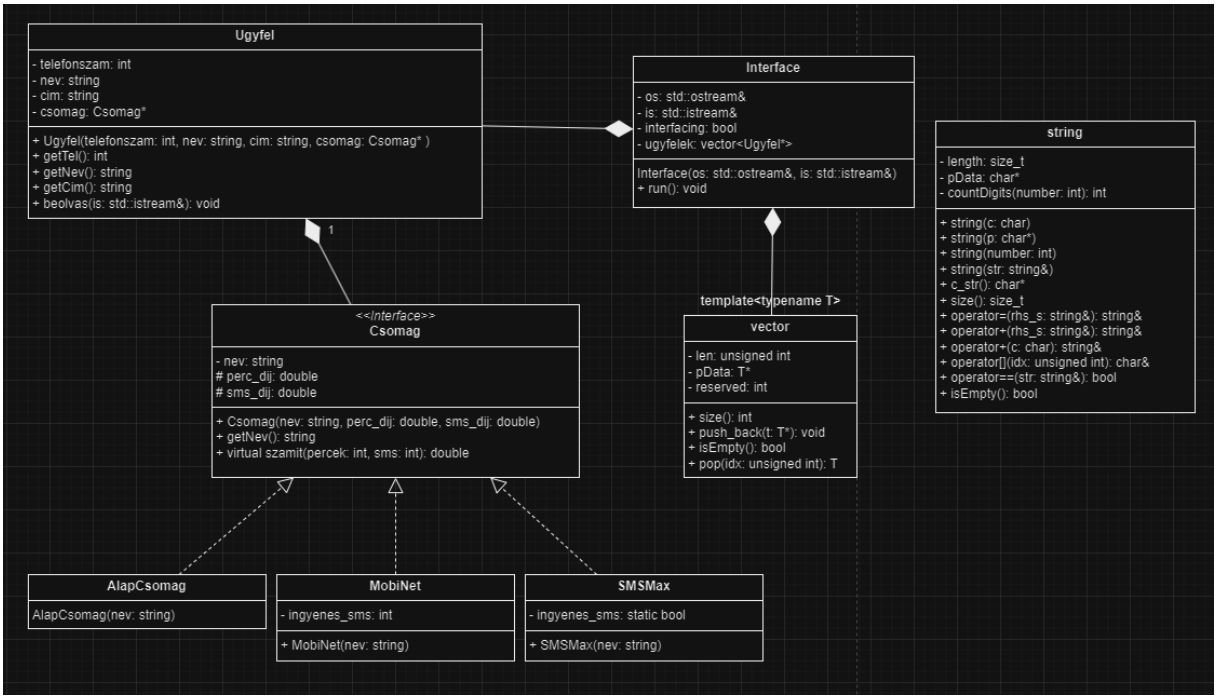
Egy mobilszolgáltatónál egy egyedi nyilvántartó programmal szeretnék kezelni az ügyfeleket. Az ügyfeleknek van neve és címe, valamint telefonszáma, ami egyben az egyedi azonosítjuk is. A szolgáltató jelenleg három csomagot biztosít ügyfeleinek: Alap, MobiNet és SMSMax, de később több csomag is lehet. Minden csomaghoz más percdj és SMS díj tartozik, valamint a számítás módszere is eltérő lehet. A MobiNet 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

A program főmenüje, amelyhez a program minden művelet után visszatér a következőképpen opciókat tartalmazza:

0. Kilépés a programból
   A program futása befejeződik, amennyiben nincs fájlba írva, az állapota elveszik.
1. Ügyfél felvétele
   Az adatok megadása után az ügyfél felvételre kerül.
2. Ügyfelek listázása
   Kilistázza az összes ügyfelet.
3. Ügyfél törlése
   Az ügyfél pontos adatait kell megadni, hogy pontosan be lehessen azonosítani a törlendő ügyfelet.
4. Ügyfelek fájlba írása
   Meg kell adni a fájl nevét, és a program oda menti az ügyfelek adatait. Ha a fájl nem létezik, akkor a program létrehozza, különben felül írja a fájl tartalmát.
5. Ügyfelek betöltése fájlból
   Az ügyfeleket betölti a megadott file-ból. A jelenlegi állapot felülíródik.
6. Számlázás
   Meg kell adni, hogy melyik fájl tartalmazza az ügyfelek e havi lebeszélt perceit, illetve smseinek számát. A program a beolvasott fájl alapján kiszámolja, hogy egyes felhasználó mennyit fizet. A program megkérdezi, hogy ezt a végeredményt szeretnénk-e fájlba írni. Ha nem akkor csak hagyjuk üresen a mezőt. A végeredmény a konzolon mindenképp megjelenik.
7. SMS ingyenességének váltása

Terv

Terv az osztályokról



A string osztály használatát a diagramon nem jelöltem, mert minden osztály használja (a vector-t kivéve).

A vector osztály egy generikus tároló, amelyet a diagramon azzal jelöltem, hogy T típusú pointereket tárol, illetve felé írtam a template-t. Nincs feltüntetve minden tagfüggvénye.

Az Interface osztály valósítja meg a kommunikációt a felhasználóval, kimenetét a kezdetben megadott ostream-re helyezi, bemenete a kezdetben megadott istream. A belső működéshez szükséges függvényeket (egyes menüpontok megnyitása, input kezelése) a diagramon nem tüntettem fel. A fájlkezelés is itt történik. Az interface példányosítása után, run függvényének meghívásával az interface elindul és fut, amíg a felhasználó a kilépés opciót nem választja. Kilépés után az állapota megmarad (amíg meg nem szűnik az objektum), így a run() függvény újbóli meghívásával újra lehet indítani.

Az AlapCsomag az absztrakt Csomag osztályból származik, extra funkcióval nem rendelkezik. A MobiNet szintén a Csomagból származik, megadja hány smst lehet ingyenesen küldeni. Az ingyenes smsek felett viszont már drágább az sms küldés darabonként. Az SMSMax (szintén Csomag leszármazott) szerződése szerint az sms csak időszakosan ingyenes. Ezt a programban futásidőben állíthatja az operátor, az interfacen keresztül. Minden csomagnak van neve, amely a csomag azonosításában játszik szerepet, de futásidőben nem változik. A perc és sms díjat protected-re tettem, hogy a származott osztályok kalkulációban egyszerűbben tudják használni, esetleg módosítani, ha a kalkuláció úgy igényli.

Ügyfél osztály az ügyfelek adatait tartja számon. A rendszerben minden telefonszám +36 kezdetű, amelyet az ügyfél telefonszám mezője nem tárol, csak a telefonszám ezt követő részét. Minden ügyfélhez egyetlen csomag tartozhat.

Programozói dokumentáció

A main.cpp-ben található a fő törzs.
A TESZT makró 1-re vagy 0-ra állításával lehet állítani, hogy a tesztesetek fussanak, vagy „production” módban fusson. A teszt program egy külön fordítási egységben található, és a program teljes funkcionalitását teszteli.

Tesztelés.
Teszteléskor fordítási egységenként, azon belül osztályonként, azon belül függvények és funkciók szerint haladok sorban a tesztesetekkel. Az interface osztályból a publikus függvények, amelyek az ügyfelek kezelését és az bemenetről olvasást végzik kerültek tesztelés alá. Szándékosan úgy építettem fel az osztályt, hogy az operátorral való kommunikáció minél jobban elkülönüljön a logikától a tesztelhetőség és átláthatóság szempontjából.

Hibakezelés.
A programban minden függvénynél jelezve van, hogy milyen esetben és milyen típusú (std::exception leszármazott) kivételt dob. Az interface kezeli a felmerülő std::runtime_error-okat, amelyek futás során felmerülhetnek. A további hibák a mainben kerülnek elkapásra. Először std::exception típusúak, majd minden hiba. Bár a program jelenlegi állapotában sosem fut a mindent elkapó catch-be, mégis bele tettem, hogy egy esetleges jövőbeli továbbfejlesztés esetén minimalizáljam program hibáit.
A program kezeli, ha a bemenetére nem megfelelő adatot kap, illetve fájlbeolvasásnál is van minimális hibakezelés, de a program elvárja a fájlok helyességét.

Az osztályokról bővebben:

Interface.
A fő függvénye, a run(). Meghívás után az interface figyel a bemenetére, amíg a 0-s menüpontot nem választjuk. Felépítése a következő. Először beállítja, az interfacing változót igazra, hisz lehet, hogy már másodszor indul. Majd amíg nem lesz az interfacing hamis, kiírja a főmenüt, azaz az opciókat, amelyre a választ elkapja, és meghívja a választás_kezelő() függvényt, amely a beírt szövegnek megfelelően eljár. Vagy meghív egy menüpontot, vagy közli az operátorral, hogy a beírt szöveg hibás.
Tettem bele isInterfacing() getter függvényt, amellyel le lehet kérni, hogy éppen fut-e az interface, azaz, hogy jelenleg figyel-e a bemenetére. Az én applikációmban ezt sosem hívom meg, de egy multithread-es továbbfejlesztett verzióban szükség lehet rá.
A számlázás függvényt kettő függvényre bontottam, de így is határeset a hosszúságuk. További függvényekre bontani viszont nem egyszerű a megnyitott fájlok miatt.

Csomag.
A számít függvényt sokáig nem akartam konstanssá tenni, hisz elméletileg a számítás változtathat a csomag belső állapotán, de mivel ezt nem tette egyik leszármazottja sem, ezért konstanssá tettem mindenhol.
Minden csomagnak van saját neve, amelyet konstans módon a csomag.h elején definiáltam.

Ügyfél.
A programban a telefonszámok mind +36 kezdetűek, amely nincs tárolva. Tehát például a +361-ből csak az 1 kerül eltárolásra. A programban a +360 telefonszám nem érvényes telefonszám, illetve minden telefonszám maximum 11 jegyű lehet (pl: +36308233301). Mivel egyik szám sem kezdődhet nullával így a +360… típusú telefonszámok, sem lehetségesek.
Ezért az ügyfél default konstruktorában a telefonszámot nullára állítom, ami tehát egy érvénytelen telefonszám. Ez egyben az ügyfél egyedi azonosítója is. Emellett a hozzá tartozó csomagot is null-ra teszem, jelezve, hogy még nem tartozik hozzá csomag.

Felhasználói dokumentáció

Csak az angol karaktereket és a számokat kezeli biztonságosan.

Elvárja a fájlok formátumnak való megfelelését: Minden fájl .txt állományú, minden adat új sorban van, és minden fájl az elején tartalmazza, hogy hány bejegyzése van.
Ügyfelek fájlba írása és olvasása esetén egymás alá a következők kerülnek: telefonszáma, neve, címe, választott csomag neve (AlapCsomag / MobiNet / SMSMax).

A program indulástól egészen kilépésig fut. Feldob menüpontokat amelyekből választva lehet funkciókat végrehajtani.