Tevékenységnapló

Rövid leírás: A program egy tevékenységnaplót valósít meg, melyben a felhasználó egy naptár celláiban vezetheti a napi tevékenységeit (futás, olvasás, sorozatnézés), havi bontásban. Az elvégzett tevékenységeket felveheti az aktuális napra, vagy visszamenőlegesen is. A program több felhasználó adatait is képes vezetni, ezeket fájlban tárolni és onnan kiolvasni. Felhasználók célokat tűzhetnek ki maguknak, amiket nyomon követhetnek az eddigi teljesítményük alapján.

Felhasználói dokumentáció: A program a konzolablakot használja, így futtatáskor az nyílik meg. A felhasználótól szabványos bemeneten a nevét várja, majd enter leütést követően a képernyőn a következők jelennek meg:

Jelenlegi hónap neve, a bal felső sarokban.

Jelenlegi hónap naptárnézete, melynek – a szokásos módon – az első oszlopa mindig a hétfőket jelenti, és melyben az aktuális nap cellája kitüntetett vizuális szerepet kap, a könnyebb átláthatóság céljából.

A következő menü, melyből a felhasználó a magyarázó utasítás melletti betű és az azt követő enter billentyű lenyomásával választhat:

```
Tevekenyseg felvetele - +

Tevekenyseg visszamenoleges felvetele - r

Uj cel kituzese - c

Havi statisztika megtekintese - s

Kilepes - x
```

Amennyiben a futása bármely pontján a felhasználótól hibás vagy nem értelmezhető utasítást kap, a program ebbe az alaphelyzetbe, a „főmenübe” áll vissza.

Tevékenység felvétele: Ha a felhasználó ezt a menüpontot választja, lehetősége van az aznapi adatait módosítani. A képernyőn egy újabb választás után – ahol afelől dönthet, melyik tevékenységet módosítja – az adat megadása után az bevitelre kerül.

```
Melyik adatot szeretned felvenni?

Futas - 1

Olvasas - 2

Filmnezes - 3
```

Új adat felvétele után – a hibás kijelzés elkerülése érdekében – a teljes naptár újra rajzolódik.

```
Add meg a beallitando erteket!
```

A megfelelő szám választása után a program a konkrét számadat bevitelét várja, mértékegység nélkül (pl: 7.2 [km] vagy 36 [oldal]), amit elment aznapra, felülírva az esetleges előző értéket. Ha egy napra nincs megadva érték, vagy törölve lett, a program azt nem írja ki, az átlagszámításban nem veszi figyelembe (különbözik a nulla értéktől!)

Ha egy napra szeretnénk 0-t állítani – ami az átlagba is beleszámít – azt külön fel kell venni.

Ha egy adatot törölni szeretnénk, értéknek egy 0-nál kisebb számot adjunk.

Tevékenység visszamenőleges felvétele: Mindenben megegyezik az előző menüponttal, azzal a különbséggel, hogy a beolvasandó adat típusa és mennyisége megadása előtt azt is meg kell adnia, melyik nap adatát kívánja megváltoztatni.

```
A honap hanyadik napjanak tevekenyseget szeretned modositani?
```

Amint a kiválasztás megtörtént, a naptár cellája vizuálisan is kiemelésre kerül.

```
Melyik adatot szeretned felvenni?

Futas - 1

Olvasas - 2

Filmnezes - 3
```

```
Add meg a beallitando erteket!
```

Tehát egy ***4 -> enter -> 2 -> enter -> 36 -> enter*** billentyűleütés-sorozat eredménye az e-havi negyedikei *(4)* Olvasás adat *(2)* értékét harminchatra *(36)* módosítása.

Új cél kitűzése: A felhasználó képes célt kitűzni maga számára, hogy havonta átlagosan legalább hány kilométert szeretne futni, oldalt olvasni, percet filmnézés közben kikapcsolódni. Ezeket a célokat először a típus kiválasztásával, majd az célul kitűzött érték megadásával tudja beállítani.

```
Melyik tevekenyseg celjat szeretned modositani?

Futas - 1

Olvasas - 2

Filmnezes - 3
```

```
Mennyi legyen a napi atlagos cel?
```

Tehát egy ***1 -> enter -> 5 -> enter*** billentyűleütés-sorozat eredménye a Futás adat *(1)* havi céljaként öt kilométer *(5)* beállítása.

Havi statisztika megtekintése: A hónap bármely pontján lehetősége van a felhasználónak megtekinteni, hogy áll adott hónapban a kitűzött céljaival. Ez az almenü nem interaktív, a program által kiszámolt statisztikák megtekintése után, csak a főmenühöz visszatérés lehetséges, az enter billentyű leütésével.

Ez a menüpont minden tevékenység értelmezhető (tehát a felhasználó által manuálisan bevitt) adatból napi átlagot számol az e-havi adatok alapján, melyet összehasonlít a felhasználó által kitűzött célokkal. Ennek az eredménye a következő lehet:

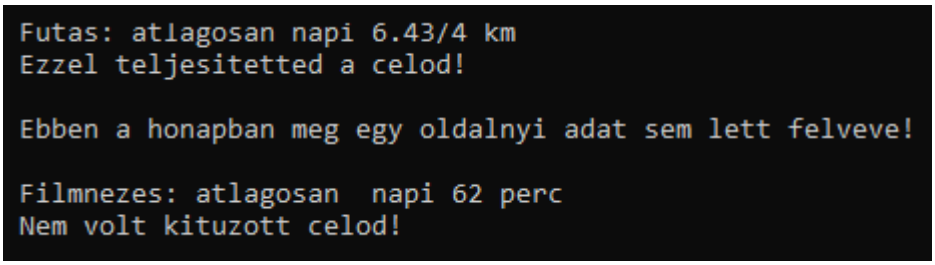
Nincs értelmezhető adat az adott kategóriában (Például ha adott hónapban még nem vett fel olvasás adatot. Ekkor csak ez az üzenet jelenik meg, nem számolható átlag.)

Az átlag kiszámítható, de nincs cél, amihez hasonlítható lenne. (A felhasználó még nem állított be, vagy eltávolította a régebbi célt. Ekkor csak a napi átlag érték jelenik meg.)

Az átlag kiszámítható, de nem éri el a kitűzött célt (Ilyenkor ez az üzenet, illetve egy összehasonlítás ELÉRT / KITŰZÖTT MENNYISÉG formában.)

Az átlag kiszámítható, és eléri a kitűzött célt (Ilyenkor ez az üzenet, illetve egy összehasonlítás ELÉRT / KITŰZÖTT MENNYISÉG formában.)

Példa egy lehetséges kiértékelésre:



Kilépés: Ezt a parancsot választva az aktuális felhasználó adatai fájlba íródnak, hogy következő megnyitáskor a módosított adatok is elérhetőek legyenek, majd a program kilép.

Programozói dokumentáció: A program néhány osztály és egy erre tervezett speciális vektor stílusú sablon együttműködésén alapszik. A belső logikáját nagyrészt a vektor típus tagfüggvényei teszik lehetővé, ám ezek a funkciók kifejezetten egy nap tevékenység-adatainak vektorán – egy napokból álló tömbön – azaz egy hónapot modellező osztályon vannak értelmezve.

Osztályok:

A **Futas**, **Olvasas** és a **Filmnezes** mind a **Tevekenyseg** osztályból publikusan származtatott osztályok. Ezek a hárman alkotják a program működésének alapvető építőköveit. Ezeknek az osztályoknak a deklarációi a `Tevekenyseg.h`, `Futas.h`, `Olvasas.h` és a `Filmnezes.h` fájlokban található, a tagfüggvényei implementációi a `Tevekenyseg.cpp`, `Futas.cpp`, `Olvasas.cpp` és a `Filmnezes.cpp` fájlokban.

A **Nap** osztály egy naptári nap modellezésére szolgál, minden aznap elvégzett tevékenységet, és az azokon végezhető adatfrissítéseket lehetővé tevő függvényeket tartalmazza. Deklarációja a `Nap.h`, a tagfüggvényei implementációi a `Nap.cpp` fájlban találhatók.

A **Honap** sablon osztály egy primitív vektor, mely bármilyen adattípussal létrehozható, másolható, indexelhető, de minden tulajdonságát akkor tudjuk kihasználni ha a **Nap** osztályból példányosított objektumok tárolására használjuk. A főprogramrész szinte kizárólag a **Honap** osztály tagfüggvényein keresztül képes minden funkcióra, amit ebben a programban használni szeretnénk. Deklarációja és a tagfüggvényei implementációi a `Honap.hpp` fájlban találhatók. A program feltételezi a `<direct.h>` létezését a benne definiált alkatalógusokat használó fájlkezelési rendszer használatához (alkatalógusok létrehozása).

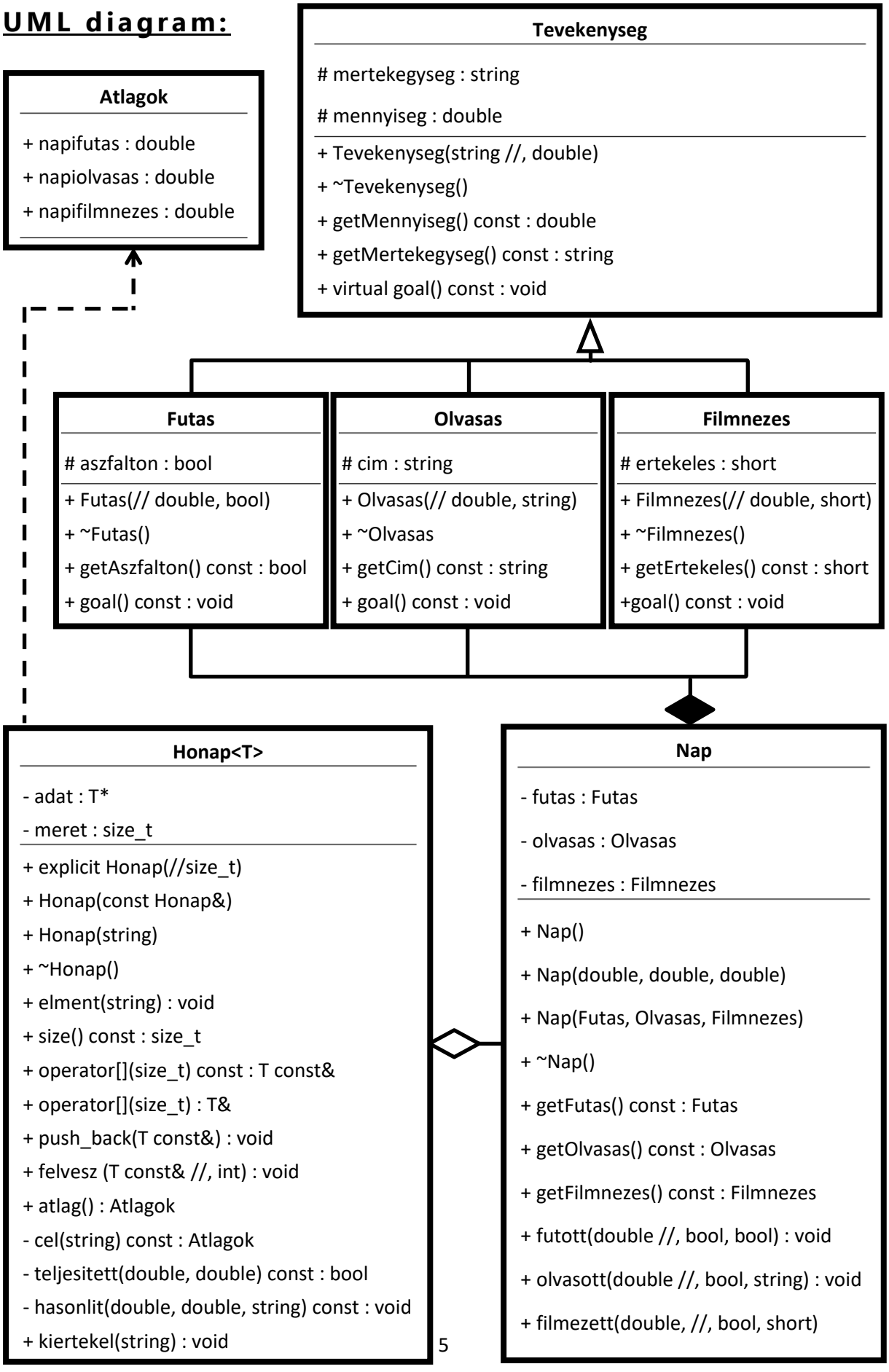
Az **Atlagok** struktúra egy primitív osztály, kizárólag publikus elérhetőségű adattagokkal, mely a **Honap** osztály működését hivatott segíteni. Ez az adatszerkezet alapvetően egy **Nap** objektum három tevékenységének mérőszámait tárolja. Ilyen adatformát használunk a napi átlag számításánál is visszatérési értékként, az azt szolgáló globális függvényekben. Deklarációja a `Honap.hpp` fájlban található.

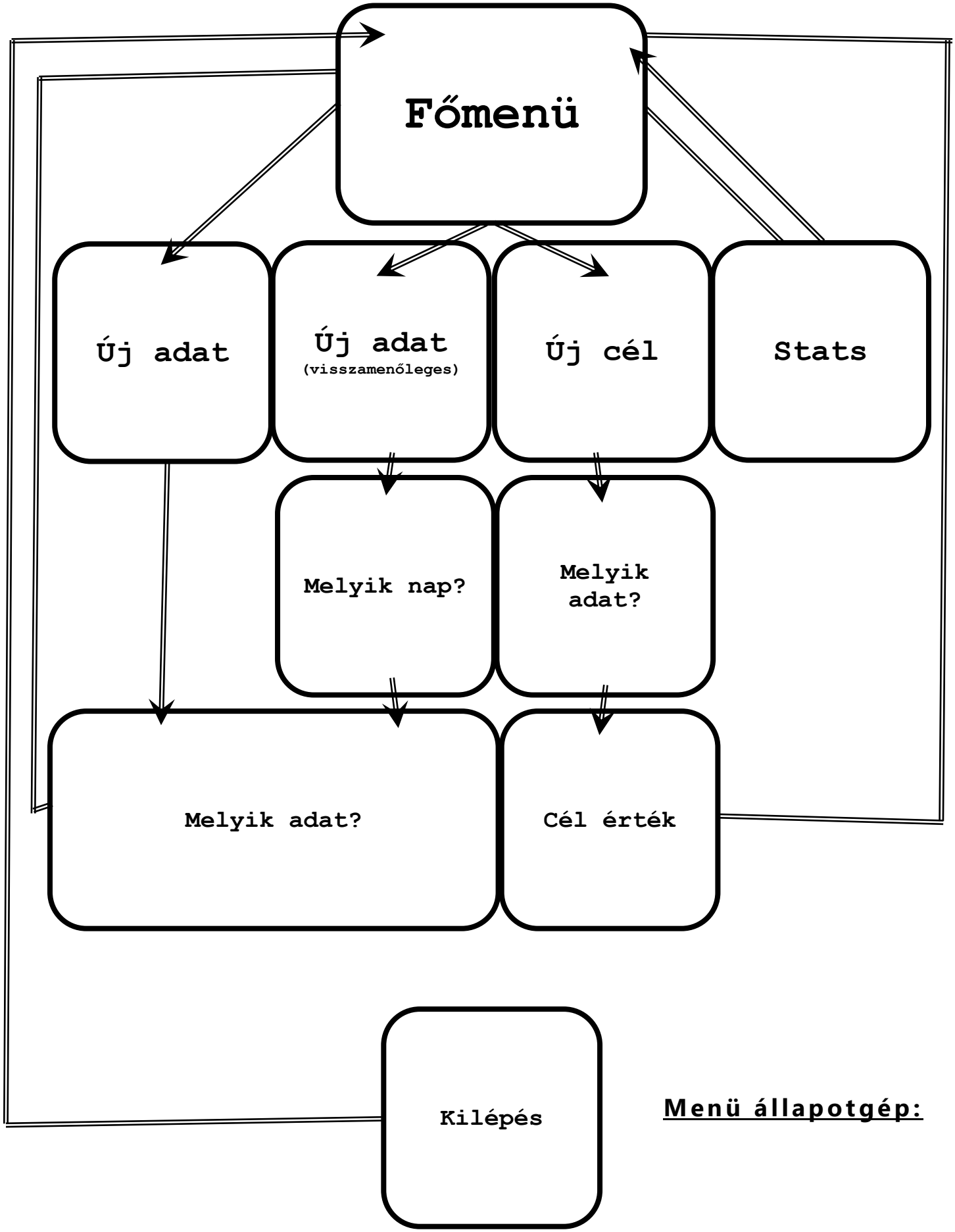
Az osztályokon kívül óriási szerepe van a megjelenést elősegítő függvényeknek, melyek deklarációját a `formatting.h`, implementációját a `formatting.cpp` fájl tartalmazza. Ezek a függvények feltételezik a `<windows.h>` létezését, segítségével módosítják a konzolablak méretét és a kurzor pozícióját.

Havi célok felvételére és törlésére szolgáló függvények deklarációja a `celok.h`, implementációja a `celok.cpp` fájlban található.

Az osztályok együttműködését és kapcsolatait, a bennük definiált függvények láthatóságát, nevét, visszatérési értékét, kötelező és opcionális paramétereit a következő UML diagram vázolja. A függvények dokumentációját lásd az azt követő oldalon.

UML diagram:


Tevekenyseg
mertekegyseg : string
mennyiseg : double
+ Tevekenyseg(string //, double)
+ ~Tevekenyseg()
+ getMennyiseg() const : double
+ getMertekegyseg() const : string
+ virtual goal() const : void
Atlagok
+ napifutas : double
+ napiolvasas : double
+ napifilmnezes : double
Futas
aszfalton : bool
+ Futas(// double, bool)
+ ~Futas()
+ getAszfalton() const : bool
+ goal() const : void
Olvasas
cim : string
+ Olvasas(// double, string)
+ ~Olvasas
+ getCim() const : string
+ goal() const : void
Filmnezes
ertekeles : short
+ Filmnezes(// double, short)
+ ~Filmnezes()
+ getErtekeles() const : short
+goal() const : void
Honap<T>
- adat : T*
- meret : size_t
+ explicit Honap(//size_t)
+ Honap(const Honap&)
+ Honap(string)
+ ~Honap()
+ elment(string) : void
+ size() const : size_t
+ operator[](size_t) const : T const&
+ operator[](size_t) : T&
+ push_back(T const&) : void
+ felvesz (T const& //, int) : void
+ atlag() : Atlagok
- cel(string) const : Atlagok
- teljesitett(double, double) const : bool
- hasonlit(double, double, string) const : void
+ kiertekel(string) : void
Nap
- futas : Futas
- olvasas : Olvasas
- filmnezes : Filmnezes
+ Nap()
+ Nap(double, double, double)
+ Nap(Futas, Olvasas, Filmnezes)
+ ~Nap()
+ getFutas() const : Futas
+ getOlvasas() const : Olvasas
+ getFilmnezes() const : Filmnezes
+ futott(double //, bool, bool) : void
+ olvasott(double //, bool, string) : void
+ filmezett(double, //, bool, short)



Menü állapotgép:

Menü: A program menüjét egy állapotgép vezérli. Ameddig nem a kilépés állapotában vagyunk, addig minden állapoton megáll, beolvassa a megfelelő adatot, vagy a főmenüben kiválasztja a következő állapotot. A működése és implementációja részletesen a formatting.h/formatting.cpp fájl függvényleírásai alatt olvasható.

Függvények: Alább a program által deklarált osztályok tagfüggvényeinek, illetve az egyéb funkciók ellátására szolgáló globális függvények használatának részletes bemutatása következik

main.cpp

```
int main();
```

- Csak a főprogramrészt tartalmazza.

Tevekenyseg.h/Tevekenyseg.cpp

```
Tevekenyseg(std::string mertekegyseg, double mennyiseg = 0);
```

- Konstruktor. A tevékenységhez tartozó mértékegységet kötelező megadni, a mennyiség opcionális.

```
double getMennyiseg() const;
std::string getMertekegyseg() const;
```

- Getterek. Visszatérnek a privát adattagok értékeivel.

```
virtual int goal() const = 0;
```

- Tisztán virtuális tagfüggvény, a belőle származtatott osztályok esetén van csak értelme.

Futas.h/Futas.cpp

```
Futas(double mennyiseg = 0, bool aszfalton = false);
```

- Konstruktor. A szülő mértékegységét „km”-re állítja. Mindkét paramétere opcionális.

```
bool getAszfalton() const;
```

- Getter. Visszatér a privát adattag értékével.

```
int goal() const;
```

- A Tevekenyseg osztályból származtatott osztályok megkülönböztetésére szolgál. Visszatérési értéke mindig 1.

Olvasas.h/Olvasas.cpp

```
Olvasas(double mennyiseg = 0, std::string cim = "");
```

- Konstruktor. A szülő mértékegységét „oldal”-ra állítja. Mindkét paramétere opcionális.

```
std::string getCim() const;
```

- Getter. Visszatér a privát adattag értékével.

```
int goal() const;
```

- A Tevekenyseg osztályból származtatott osztályok megkülönböztetésére szolgál. Visszatérési értéke mindig 2.

Filmnezes.h/Filmnezes.cpp

```
Filmnezes(double mennyiseg = 0, short ertekeles = 0);
```

- Konstruktor. A szülő mértékegységét „perc”-re állítja. Mindkét paramétere opcionális.

```
short getErtekeles() const;
```

- Getter. Visszatér a privát adattag értékével.

```
int goal() const;
```

- A Tevekenyseg osztályból származtatott osztályok megkülönböztetésére szolgál. Visszatérési értéke mindig 3.

Nap.h/Nap.cpp

```
Nap(Futas futas, Olvasas olvasas, Filmnezes filmnezes);
Nap(double futas, double olvasas, double filmnezes);
Nap();
```

- Konstruktorok. Inicializálható paraméter nélkül, ekkor mindhárom adattagjára alapértelmezett értékek állítódnak. Inicializálható három valós számmal, ilyenkor csak az adattagok mennyisége állítható be, de gyakran ez elég, átláthatóbb kódot eredményez. Inicializálható a három adattagjával megegyező három paraméterrel, így minden részlete beállítható. Ez precízebb, de gyakran felesleges, nagyon hosszú, nehezebben átlátható kódsort eredményezhet.

```
Futas getFutas() const;
Olvasas getOlvasas() const;
Filmnezes getFilmnezes() const;
```

- Getterek. Visszatérnek a privát adattagok értékével. Ezek Tevékenységből származtatott osztályok, a visszatérési értékre közvetlen meghívható azok gettere is.

```
void futott(double mennyit, bool felulir = false, bool aszf = false);
void olvasott(double mennyit, bool felulir = false, std::string cim = "");
void filmezett(double mennyit, bool felulir = false, short ert = 0);
```

- Már létező Nap adatait módosíthatjuk velük. Egy paraméterrel hívva a már létező adatot inkrementálja. A „felulir” paramétert igazra állítva felülírja azt. Ilyenkor az osztályfüggő paraméter is szabadon beállítható, kihagyása esetén alapértelmezett értékkel felülírja azt is. Szintaktikájukban, viselkedésükben egyformák.

Honap.hpp

```
explicit Honap(size_t meret = 0);
```

- Konstruktor. Opcionális paraméterként a tömb számára kezdetben lefoglalandó hely mérete adható meg, darabban mérve.

```
Honap(const Honap &source);
```

- Másoló konstruktor.

```
Honap(std::string nev);
```

- Stringgel inicializáló konstruktor. Bemenetnek felhasználónevet vár, adatokkal a ./ÉV/HÓ/FELHASZNÁLÓNÉV.txt útvonalon található adatok alapján feltölti magát. Ha a fájl nem létezik létrehozza azt, majd feltölti alapértelmezett adatokkal, és ezzel töltődik fel. Létrehozza a ./celok/FELHASZNÁLÓNÉV.txt fájlt, benne -1-re állítja azokat, ezzel nem beállítottnak tekintve őket.

```
void elment(std::string nev);
```

- A névvel inicializáló konstruktor párja. Paraméterként felhasználónevet vár, a tömb tartalmát a ./ÉV/HÓ/FELHASZNÁLÓNÉV.txt útvonalon fájlba másolja. A tömb tartalma változatlan marad!

```
~Honap();
```

- Destruktor. A dinamikusan foglalt memóriaterületet felszabadítja.

```
size_t size() const;
```

- Getter. A tömb méretét adja vissza.

```
T& operator[](size_t index);
T const& operator[](size_t index) const;
```

- A [] operátor túlterhelése. A tömb „index”-edik adattagjával tér vissza. Konstans tömbre is meghívható.

```
void push_back(T const &uj_ertek);
```

- A standard push_back függvény. A referenciaként átvett értéket a tömb végére beszúrja, a szükséges memóriát dinamikusan foglalja.

Az ezt követő tagfüggvények Nap objektumok tömbjére, azon belül is stringgel inicializált (28-31 méretű) tömbökre vannak tervezve!

```
void felvesz(T const &ertek, int napra = 0);
```

- Egy adott értékre cseréli a tömb valamelyik elemét. Ha nincs megadva melyik nap elemét cserélje, a lokális óra szerinti aktuális napot cseréli.

```
Atlagok atlag() const;
```

- A tömbben tárolt napok közül az összes figyelembe venni kívánt értékeit átlagolja, egy Atlagok struktúrában mindhárom értéket visszaadja. Ha egy tevékenységből nincs értelmezhető adat, az ahhoz tartozó visszatérési érték -1.

```
Atlagok cel(std::string nev) const;
```

- A paraméterként adott felhasználónév alapján a ./celok/FELHASZNÁLÓNÉV.txt útvonalból kiolvassa a felhasználónévhez tartozó célokat, egy Atlagok struktúrával tér vissza, melynek megfelelő adattagjai ezeket tárolják. Privát! A „kiertekel” függvény belső komponense.

```
bool teljesitett (double eredmeny, double cel) const;
```

- Kiírja a teljesített és a megcélzott mennyiségeket TELJ/CEL formában, igazzal tér vissza ha teljesítve lett a cél (eredmeny >= cel), hamissal, ha nem. Privát! A „kiertekel” függvény belső komponense.

```
void hasonlit (double eredmeny, double cel, std::string mertekegyseg) const;
```

- Az alapján, hogy teljesített-e a cél, a célkiírás menüpontot kezeli. Privát! A „kiertekel” függvény belső komponense.

```
void kiertekel (std::string nev);
```

- Mindhárom tevékenységtípusból havi szintű napi átlagot számol, amit a kitűzött célokkal összehasonlítva a standard kimeneten megjelenít. Paraméterként a felhasználó nevét várja, akinek a cél adatait a fájlból ki fogjuk olvasni.

Formatting.h/Formatting.cpp

```
enum menustate {fomenu, uj, ujback, melyiknap, melyikadat, ujcel, melyikceladat, celertek, statsdisplay, stats, kilep};
```

- A menü rendszer állapotának vezérlésére szolgáló állapotgép lehetséges állapotait felsoroló enumerátor.

```
void naptarkep(tm* ido, int eltol, bool shiftdown, Honap<Nap> const &adat);
```

- A naptárat vizuálisan megjelenítő függvény. Szüksége van az aktuális dátumra, amit az „ido” változó szolgáltat, a behúzás mennyiségére („eltol”), illetve hogy egy sorral lejjebb kezdje-e („shiftdown”). Az aktuális hónapnév kiírásával kezd, majd a naptár keretét rajzolja ki. Ezután a cellákat megtölti az adott felhasználó adataival, ezt a konstans referenciaként átvett Honap<Nap> vektor adatai alapján teszi.

```
menustate nextstate(char selection, menustate src_state);
```

- A főmenüben lehetséges választás, és a jelenlegi állapot alapján a következő állapotra lépteti az állapotgépet. Visszatérési értéke a bemeneti állapot és a választott karakter alapján a következő állapot enumerációja. A menu() függvény hívja.

```
void clearmenuselection();
```

- Szintén a menu() függvény segédfüggvénye, az általa kiírt utasítások és válaszlehetőségek törléséért felelős, a megfelelő sorok felülírásával. Főprogramrészben nem hívandó.

```
menustate menu(menustate state);
```

- Az egyes állapotok utasításainak kiírásáért felelős. A neststate() függvény segítségével a jelenlegi állapottól függően a következőt adja vissza. A főprogramrészben ez használandó az állapot léptetésére.

```
void honapnev(int ho);
```

- A paraméterként átvett „ho”-adik indexű hónap nevét írja ki a képernyőre word-arttal. A naptarkep() függvény használja. 0-tól indexel! (0 = Jan, 11 = Dec)

```
void gotoxy( int column, int line );
```

- A konzolablakban a kurzort az „coulomn”-adik oszlop „line”-adik sorába helyezi. A naptár rajzolásánál és a menü kiírásánál is használt.

```
void cellarajzol(int x, int y, bool ma, bool shiftdown);
```

- Kirajzol egyet a naptár cellái közül. Az „x” és „y” paraméterek a cella bal felső sarkát jelzik. Ha a „ma” logikai érték igaz, a cella duplavonalú, beszínezett. A „shiftdown” logikai érték ha igaz, egy cellányival lentebb képzeli az első sort.

```
void adatbeir(int nap, int shift, bool shiftdown, Atlagok cellaadat);
void adatbeir(int nap, int shift, bool shiftdown, std::string szoveg, int hova);
```

- A naptár egy cellájába ír adatot. Mindkét verzió paraméterként várja, hogy hányadikai cellába írunk, valamint a behúzás mértékét és hogy lentebb kezdődik-e. Amennyiben egy Atlagok struktúrát adunk még meg, mindhárom adatot kiírja, a megfelelő mértékegységekkel, vízszintesen középre igazítva, egymás alá. Egyéb esetben egy bármilyen kiírandó szöveget vár, illetve hogy függőlegesen a felső (1), középső (2), vagy az alsó (3) sorba írja.

```
void cellakiemel(int nap, int shift, bool shiftdown);
```

- Hányadikai cella, behúzás és letolás paraméterek alapján vizuálisan kiemeli a „nap” paraméter által megadott nap celláját.

```
void statpos();
```

- A statisztikák helyére igazítja a kurzort.

---

celok.h/celok.cpp

```
void setgoal(std::string nev, const Tevekenyseg& melyik, double mire);
void setgoal(std::string nev, Atlagok ujcelok);
```

- A ./celok/NEV.txt fájlba írja vagy egy Atlagok struktúra elemeit, vagy egy csak a „melyik” által kijelölt adatot módosítja a szintén paraméterként átadott értékre, a másik kettőt változatlanul hagyja.

```
void resetgoal(std::string nev, const Tevekenyseg& melyik);
void resetgoal(std::string nev);
```

- Visszaállítja a választott tevékenységtípus adatát alaphelyzetbe, vagy ha csak egy string adott, a ./celok/NEV.txt fájlt három darab „-1”-gyel tölti fel, visszaállítva ezzel mindhármat.

Tesztelési dokumentáció: A program helyességét több megközelítési szempontból ellenőriztem.

1) Felépítés, osztályok tagfüggvényei, kommunikációinak helyessége szempontjából
    a. közel 100 sor, és a függvények lényegi részét lefedő, a „gtest_lite" tesztelési környezet által lehetővé tett tesztesetekkel, és azoknak a várt, hiba nélküli eredményes lefutásának tesztelésével,
    b. a fejlesztés során minden új tagfüggvény/globális függvény folyamatos, közvetlen kipróbálásával, optimalizálásával.
2) A felhasználói felület kezelésének szempontjából
    a. az elkészült felhasználói felület minden lehetséges útvonalának, sorrendjének bejárásával, kipróbálásával,
    b. lehetséges érvénytelen/értelmezhetetlen válaszok bevitelével.
3) Erőforrás-használat szempontjából
    a. a „memtrace" könyvtár memóriaszivárgást ellenőrző funkcióival,
    b. a statikusan létrehozott változók számának és élettartamainak minimálisan szükségesre korlátozásával.

Az elkészült, végleges szoftver mindhárom tesztelési formán átesett, és azoknak megfelelt. A definiált osztályok együtt dolgoznak, kapcsolatuk, tagfüggvényeik ellenőrzöttek.

A felhasználói felület átlátható és bármilyen bemenetet képes racionálisan kezelni, még a hibásokat is.

Az elkészült szoftver jelenlegi állapotában memóriaszivárgás-mentes, a dinamikusan foglalt erőforrások mind felszabadításra kerülnek.