Megoldás lépései
- #include <iostream>, "using namespace std", csinálok main-t, bemásolom a példakódot
- Projekt struct készítése
- létrehozom a hivatkozott tagfüggvényeket
  > Milyen paramétereket vesznek át a függvények, és mivel térnek vissza (a "semmivel"=void is válasz...)

FUTTASSUNK!

- milyen adatszerkezetben tárolnám el a dolgot?
  > A részfeladatok kulcsembereit a feladat neve alapján fogom lekérdezni. Egy kulcsember egy feladatnál csak egyszer szerepelhet, a sorrend mindegy, viszont szeretném gyorsan előkeresni a kulcsemberek közül, hogy szerepel-e ott az illető neve. >> map<string, set<string>>
  > Egy feladathoz több találkozó is lehetséges, de ugyanazon a napon egy feladathoz maximum egy találkozó lehetséges, a feladatokhoz tartozó találkozókat a feladat neve alapján akarom lekérni. Hasznos lenne, ha egyszerűen tudnám előkeresni, hogy adott feladathoz adott napon van-e találkozó. >> map<string, set<int>>
- hozzuk létre a változókat, és írjuk meg a feladat hozzáadása függvényt!
  > Megjegyzés: set helyett mindkét esetben nyugodtan használhatunk vector-t is, csak annak nincs find függvénye, emiatt nekünk kell manuálisan megírnunk a vector-ban való keresést, ami plusz macera.
  > Tipp: map-hez nagyon egyszerű hozzáadni új elemet: map_neve[uj_elem_kulcsa] = uj_elem_erteke;
    > Egy új üres map-elemet (vagyis, ahol az uj_elem_erteke egy üres változó) még egyszerűbb létrehozni: map_neve[uj_elem_kulcsa];

FUTTASSUNK!

- fontos, hogy a feladat megoldása közben folyamatosan tudjuk követni, hogy milyen adatokat tárol az adatszerkezetünk, hogy ha valami hiba van, azt rögtön megtaláljuk
- Írjuk meg a "teljes_naptar_kiir" függvényben azt, hogy kiírassa a feladatokat!
  > Írassuk ki a feladatok neveit (minden egyes feladatot új sorba)!

FUTTASSUNK!

- Írjuk meg a "kulcsember" függvényt!
  > Előkeresem a részfeladatot, ahova be akarom szúrni és beszúrom a hozzá tartozó set-be: map_neve[feladat_neve].insert(kulcsember_neve);

FUTTASSUNK!

- Ellenőrizzük, hogy jól működik-e! Bővítsük ki a "teljes_naptar_kiir" függvényt azzal, hogy a feladatokhoz egy új sorban a feladat neve alá kiíratjuk azt is vesszővel elválasztva, hogy kik a kulcsemberei.

FUTTASSUNK!

- Készítsük el a találkozók beszúrása függvényt. Először ne vizsgáljuk, hogy van-e ütközés. Ekkor pont ugyanúgy fog kinézni, mint a kulcsember beszúrása:
map_neve[feladat].insert(talakozo_napja);

FUTTASSUNK!

- Ellenőrizzük, hogy jól működik-e! Bővítsük ki a "teljes_naptar_kiir" függvényt azzal, hogy a feladatokhoz megint egy új sorba a feladat neve és kulcsemberei után a harmadik sorba kiíratjuk azt is vesszővel elválasztva, hogy mikor vannak találkozói.

FUTTASSUNK!

- Na, most jön a neheze. Azt kellene megvizsgálnunk, hogy az "uj_talalkozo" függvényben hozzáadandó találkozó ütközik-e valamelyik másikkal. Fogalmazzuk meg pontosabban a feladatot!

> Annak a feladatnak, aminek egy találkozóját akarjuk beszúrni, van-e olyan kulcsembere, akinek aznap már van másik találkozója!

> Még a találkozó beszúrása előtt írjuk ki hogy melyik feladatot akarjuk beszúrni (én itt nem spórolnék hosszú szöveget kiírni magamnak, pl, hogy "A ... feladathoz szeretnek beszurni talalkozot a ... napra", mivel így átláthatóbb számodra is, hogy mikor mit írsz ki).

> Ezután menjünk végig feladat kulcsemberein, és írassuk ki őket!

FUTTASSUNK!

> Most az minden egyes kiíratott kulcsember neve után írjuk ki, hogy az adott ember mely részfeladatokban kulcsember még.

> Ehhez menjünk végig egy ciklussal minden egyes ember esetében az összes feladaton, hogy ott szerepel-e az adott ember. Mivel set-et választottunk, elég egyszerű lesz a keresés: a count és a find is megfelel nekünk (a count 0-t ad, ha nincs benne, 1-et ha igen, a find pedig egy iterátort ad vissza, a találat helyének megfelelő iterátort, ha megtalálta, set_neve.end()-et, ha nem). Ha igen, akkor írjuk ki a feladat nevét!

FUTTASSUNK!

> Most a kulcsember neve után kiíratott feladatlista után (mondjuk zárójelben) írassuk ki, hogy az egyes feladatokhoz melyik napokon vannak találkozók. Ehhez egy újabb ciklusra lesz szükség. (Szomorú módon már a harmadikat ágyazzuk egymásba, de hát ilyen az élet.)

FUTTASSUNK!

> Igazából innen már nyert ügyünk van, mivel csak annyit kell hozzátenni, hogy ha az éppen kiíratandó nap megegyezik a beszúrni kívánt találkozó napjával, akkor írjunk ki valami hibaüzenetet és lépjünk ki a return segítségével mielőtt hozzáadnánk a találkozót.

FUTTASSUNK!

> Hogy teljes pontszámú legyen a feladatunk, írjuk át a uj_talalkozo függvény visszatérési értékét bool-ra, és "return" helyett "return false;" legyen, és szúrjunk be a függvény végére egy "return true"-t.

FUTTASSUNK! Király, kész az A feladat

- Akkor már rá is térhetünk a B feladatra.

> Mit is akarunk? Azt, hogy ha hozzáadok egy kulcsembert egy feladathoz, akkor csekkolja le, hogy lesz-e ütközés a találkozókban.

> Mikor lehet ütközés? Ha az új kulcsember valamelyik feladatához tartozik olyan találkozó, ami az adott feladat (amihez az új kulcsembert hozzá akarom adni) találkozóival ütközik.

> Idézzünk elő egy olyan esetet a main-ben, amikor ütközés lesz. Miután hozzáadtuk a találkozókat, adjunk hozzá még egy kulcsembert valamelyik feladathoz úgy, hogy ütközés legyen.
> Első lépésben írjuk ki a hozzáadandó kulcsember feladatait. Ezt úgy tudom megtenni, hogy végigmegyek a feladatokat és kulcsembereket tartalmazó adatszerkezeten, és kiíratom a map kulcsában tárolt feladatnevet, ha a kulcsemberei közt megtalálható az új kulcsember.
> A függvény elején érdemes kiíratni valami segítő szöveget, pl "Most ...-t szeretnem hozzaadni a ... feladathoz kulcsemberkent".

FUTTASSUNK!

> Most írassuk ki annak a feladatnak a találkozóit, amihez hozzá akarjuk adni kulcsembert, és azoknak a feladatoknak is a találkozóit, amiket az előbb kikerestünk, vagyis az új kulcsember eddigi feladatainak találkozóit.

FUTTASSUNK!

> Ha ezzel is megvagyunk, akkor már csak annyit kell tennünk, hogy megvizsgáljuk a kulcsember eddigi feladatainak találkozóin végigmenő ciklusban, hogy az épp kiíratásra kerülő feladat megtalálható-e annak a feladatnak a találkozói közt, amihez hozzá akarjunk adni az új kulcsembert. Használjuk a find függvényt!
> Ha igen, akkor írjunk ki hibaüzenetet és lépjünk ki a függvényből a return segítségével.

FUTTASSUNK! Juhéé, kész a B feladat is!

- A C feladatban először ki kéne írni mindenkihez a feladatait egy megadott napon, és ha nincs aznap feladata, akkor a legközelebbit kéne kikeresni.
> Hozzunk létre egy függvényt a struct-ban, és hívjuk meg a main-ből.
> Célszerű itt is segíteni a saját dolgunkat azzal, hogy a függvényünket azzal kezdjük, hogy kiíratjuk a konzolra, hogy melyik nap teendőit akarjuk lekérdezni.
> Első körben írassuk ki az összes embert. Ehhez én célszerűnek látom egy set-be betenni az összes kulcsembert, és ha kész van, akkor ezt a set-et kiírni, mivel így nem kell figyelnünk arra, hogy egy ember többször is szerepel, mivel a set-be csak egyszer fog bekerülni.

FUTTASSUNK!

> Most írjuk ki az egyes emberek után, hogy milyen feladataik vannak. (Ciklussal végigmegyünk a feladatokat és kulcsembereket tartalmazó map-en, és kiírjuk a feladat nevét, ha szerepel a kulcsemberei közt az adott ember).

FUTTASSUNK!

> Írassuk ki a találkozókat is az egyes feladatokhoz.

FUTTASSUNK!

> Most csak azokat a feladatokat írjuk ki, amiknek az adott napon van találkozója.

FUTTASSUNK!

> Ha a 10.nap teendőit kérdezzük le, akkor látjuk, hogy senkinél nincs kiírva semmilyen feladat. Ilyenkor mindenkinél a legközelebbi teendőjét kéne kiírnia. Érdemes tehát a függvényünket átírni úgy, hogy eltároljuk egy praktikus adatszerkezetbe, hogy kinek mikor vannak feladatai, és utána

ebben az adatszerkezetben keresgélnénk a dolgokat. Az emberek alapján fogjuk csoportosítani a találkozókat, és az emberekre egy string alapján hivatkozunk, tehát a map lesz nekünk jó ide. Minden emberhez eltároljuk a feladatait. A feladatokra szintén string-gel hivatkozunk, emiatt itt szintén map lesz a nyerő. Az egyes feladatokhoz tartozó találkozókat pedig pedig csak simán el kell tárolnom úgy, hogy egy találkozó csak egyszer szerepel, és szeretném, ha egyszerűen ki tudnám keresni, hogy van-e adott feladatnak találkozója az adott napon, így a set adja magát. A javasolt adatszerkezet tehát: map<string, map<string, set<int>>>.

> Csináljuk egy ilyen lokális változót a függvényünkön belül, és töltsük föl adatokkal. A korábban megírt kiíratásunkat csak kicsit kell módosítanunk: Térjünk vissza ahhoz a verzióhoz, amikor az összes feladat összes találkozóját kiírattuk, és amikor kiírjuk az adott ember adott feladatának adott találkozó-időpontját, akkor ezt az időpontot rögtön hozzáadom az adatszerkezethez a megfelelő helyre. Amiatt nem kell aggódnunk, hogy mikor adjuk hozzá az embereket, és hozzájuk a feladatokat, mivel a map egy ügyes jószág, ami létre fogja hozni az embert és a feladatot, ha hivatkozunk rá. Így tehát egyszerű a feladat, egyetlen sort kell hozzáírnunk: lokalis_valtozonk_neve[ember][feladat].insert(talalkozo_idopontja).

> Írassuk is ki a függvény végén, hogy sikerült-e helyesen feltöltenünk ezt az adatszerkezetet. Írassuk ki embereknént az összes feladatot, és az adott feladatok után (mondjuk zárójelben) az összes találkozót. Ehhez 3 egymásba ágyazott ciklusra lesz szükségünk.

FUTTASSUNK!

> Na, most jön a trükk, mivel most csak azokat a találkozókat szeretnénk kiíratni, amik az adott napon vannak, de valakinél nincs ilyen, akkor a legközelebbi teendőjét kéne kiírni. Egyelőre csak annyit csináljuknk, hogy az adatszerkezet kiíratását annyiban módosítjuk, hogy az elején létrehozunk egy ideiglenes bool változót, amibe tároljuk, hogy az adott emberhez találtunk-e már az adott napra találkozót. Ez a változó legyen kezdetben false, és ha találok egy találkozót az adott emberhez, akkor írjuk ki a feladat nevét és állítsuk ezt a változót true-ra.

> Tehát most úgy néz ki ez a rész, hogy végigmegyek az adatszerkezeten egy ciklussal. Ekkor minden egyegy ciklus-iterációban kapok egy párt, ami tartalmazza az adott ember nevét és a feladatait. Írjuk ki az ember nevét, és egy ciklussal menjünk végig a feladatain. Használjuk a find-ot vagy count-ot arra, hogy megnézzük, az adott feladatnak van-e a kérdéses napon találkozója! (A korábbi verzióban egy ciklust használtunk arra, hogy végigmenjünk az adott feladat találkozóin, de itt most nekünk egyszerűbb ehelyett egy sima parancsot használunk a keresésre) Ha igen, akkor írassuk ki a feladat nevét, és állítsuk a bool változónkat true-ra.

> Ha végigértem egy ember összes feladatán, vizsgáljuk meg, hogy a bool típusú változónk értéke hamis-e. Ha igen, akkor írjuk ki, hogy az nincs feladata az adott embernek aznapra.

FUTTASSUNK!

> Most keressünk ki a legközelebbi feladatát, ha nincs teendője. Végig kell tehát mennünk még egyszer az adott ember összes feladatán, de most a belső set-ekben a kérdéses napnál nagyobbat keresünk. Ezt a keresést meg lehet csinálni ciklussal, de ha figyeltünk gyakorlaton (vagy legalábbis az 1. csoport gyakorlatán volt ilyen példa), akkor beugrik, hogy volt valami függvény erre a set-ben. Felmegyünk tehát a cppreference-re, kikeressük a set-ről szóló oldalt és nagy heurkéka-élmény közepette felfedezzük, hogy a set-nek van egy függvénye, ami pont arra jó, hogy egy adott kulcshoz legközelebbi, de annál nagyobb kulcsú elemet visszaadja.

> Na igen, de most minden egyes feladatnál megkapom, hogy mi a kérdéses naphoz legközelebbi feladat, de hogyan döntöm el, hogy ezek közül melyik a legközelebbi? Létrehozok egy ideiglenes változót amibe tárolom az egyes feladatokhoz kikeresett leközelebbi találkozót. Feladatonként egy időpontot szeretnék eltárolni, ezért a map<string, int> adja magát szerintem.

> Ha kikerestem mindegyik feladathoz a legközelebbi találkozót, akkor megvizsgálom, hogy találtam-e egyáltalán valamit. Ha nem, akkor kiírom, hogy később sincs feladata.

> Ha nem üres a map, akkor végrehajtok egy minimum-keresést a map értékei szerint, szépen, ahogy még BevProg1-ből tanultuk... A legkisebb értékű kulcs-elem párt aztán szépen kiíratom.

FUTTASSUNK! Na, már túl vagyunk a nehezén!

- Most már csak a fájlba kimentés és beolvasás van hátra. Csináljuk meg először a kiíratást!
- Itt most trükkösek leszünk. Ugye, még az A feladatban megírtuk azt, hogy kiírattuk a teljes naptárt a konzolra. És most itt a fájlba kiíratástnál szöget üt a fejünkbe az, hogy lám, ezt tök jól lehetne újrahasznosítani. Megtehetjük azt is, hogy kimásoljuk az egész teljes_naptar_kiir függvényt és egy fajlba_ment függvény néven kicsit átírjuk, hogy ne a cout-ra írjon, hanem fájlba. De egy okos programozó ki nem állhatja, ha tök ugyanaz a kód többször szerepel a kódjában, így ha szép és frappáns kódot akarunk, akkor azt csináljuk, hogy átírjuk a teljes_naptar_kiir függvényt úgy, hogy a konzolra és fájlba is tudjon írni. Ez úgy tehetjük meg, hogy a teljes_naptar_kiir paraméterként vegye át az, hogy hova írasson ki. Ezt azért tehetjük meg, mivel a cout és az ofstream egyaránt az ostream osztály leszármazottai, ami azt jelenti, hogy a cout-ot és egy fájlváltozót is el lehet tárolni egy ostream típusú változóba. (Ez nagyon hasonló ahhoz, mint ami a gout és a canvas esetében volt: ott is el tudtuk tárolni a gout-ot egy canvas típusú változóba, mivel a gout a canvas osztály leszármazottja.) Így tehát, ha a teljes_naptar_kiir függvény átírjuk úgy, hogy egy ostream& típusú változót fogadjon paraméterként, aminek mondjuk out legyen a neve (Figyeljünk arra, hogy a stream típusú objektumokat MINDIG REFERENCIA SZERINT vesszük át!). Ekkor csak annyit kell tennünk, hogy átírjuk a teljes_naptar_kiir függvényben a cout-okat simán out-ra. Aztán a main-ben egyszer úgy hívom meg, hogy paraméterként a cout-ot adom át, utána pedig úgy hogy megnyitom a fájlt még a teljes_naptar_kiir függvény meghívása előtt és paraméterként a fájlváltozót adom át.

FUTTASSUNK! MAGIC! Működik!

- Következzen most a beolvasás. Nézzük meg, hogy mi a formátum a fájlnak, amit kiírattam. Nálam úgy néz ki, hogy az első sorban az van, hogy "Teljes naptar kiiratasa", mivel a konzolra kiíratáskor beleraktam ezt is, hogy jobban látszódjon hogy mik is ezek az adatok. Erre a sorra nem lesz szükségünk. Ez után nálam 3 soros perióduban szerepelnek az adatok: első sor a feladat neve, második sorban a hozzá tartozó kulcsemberek (a sor elején szerepel, hogy "Kulcsemberei:"), a harmadik sorban pedig a hozzá tartozó találkozók (a sor elején szerepel, hogy "Talalkozok:").
> Első körben tehát hozzunk létre egy beolvasó függvényt a struct-ban, és a mainből hívjuk meg. Mivel a kiíratást is úgy csináltuk, hogy a fájlt a main-ben nyitjuk meg, és paraméterként adjuk át, ezért itt is tegyünk így, az ifstream típusú változót még a mainben hozzuk létre, és istream& objektumként vegyük át (nem elírás az "f" hiánya az istream-ből, itt ugyanarról van szó, mint az ofstream és az ostream esetében, az istream általánosabb, mint az ifstream, így akár cin-t is átadhatnánk a függvénynek). A main-ben nyissuk meg a fájlt, amibe kiírattuk az adatokat, és hívjuk meg a beolvasó függvényt átadva neki ezt a fájlt paraméterként.
> A beolvasó függvénybe pedig kezdetben csak annyit írjunk, hogy -amennyiben nálad is úgy van, hogy az első sor fölösleges- olvassuk be az első sort, majd írjunk egy ciklust, ami a fájl végéig olvas, és a cikluson belül beolvas három sort egy string változóba, kiírja őket a konzolra (nyugodtan használhatjuk ugyanazt a string változót, nem baj, ha minden egyes beolvasásnál felülírjuk az előző beolvasást).
> Vizsgáljuk meg a függvényben azt is, hogy sikeres volt-e a fájl megnyitása.

FUTTASSUNK!

> Most dolgozzuk fel a beolvasott adatokat. A feladat nevét továbbra is csak simán írassuk ki, de a kulcsembereknél és találkozóknál vizsgáljuk meg a beolvasott sorokat. Erre kiválóan alkalmas a stringstream. Egy kis emlékeztető a stringstream-ről: lényegében felfogható egy virtuális fájlként

(mint amilyenek a plang-os fájlok voltak...), vagyis ugyanúgy lehet beléjük írni, mintha egy fájlba írnánk, és úgy lehet belőlük olvasni, mintha fájlból olvasnánk be.

> Tegyük meg tehát azt, hogy a kulcsembereket tartalmazó sort írassuk ki egy stringstream változóba, majd getline segítségével segítségével beolvasom az egyes kulcsembereket vesszőig olvasva a getline-nal. Ha a sor elejére kiírtad az, hogy "Kulcsemberek:", vagy valami hasonlót (mint ahogy én is tettem), akkor kezd azzal a sor feldolgozását, hogy miután kiírtad a sort a stringstream-be, olvass be egy sort kettőspontig, és csak utána állj neki beolvasni a többi dolgot.

> A beolvasott neveket egyelőre a konzolra írjuk ki.

> FONTOS! A stringstream-et minden egyes használat előtt ürítsük ki a clear() parancs segítségével!

FUTTASSUNK!

> Csináljuk meg ugyanezt a találkozókkal is!

FUTTASSUNK!

> Már csak egy gondunk van: a találkozókat int-ként akarjuk majd tárolni, de mi most egyelőre string-ként olvastuk be. Át kell tehát konvertálnunk a szöveget számmá vagy az atoi vagy egy újabb stringstream segítségével. Én személy szerint a stringstream-et szoktam használni.

FUTTASSUNK!

- Lényegében már kész is vagyunk, már csak a konzolra kiíratás helyett meg kell hívnunk a már megírt függvényeinket, mégpedig az uj_reszfeladat, kulcsember, és uj_talalkozo függvényeket.

> Annyit azért még kell változtatni a dolgot, hogy a beolvasott feladatot mégiscsak egy külön változóban kéne eltárolni, mivel a kulcsember és uj_talalkozo függvényeknél szükség lesz rá.

> Teszteljük is a beolvasást. Az adatokat tartalmazó fájlba írjunk bele még egy embert és még egy találkozót, és nézzük meg, hogy megjelenik-e a konzolon a futtatáskor.

FUTTASSUNK!

> Egy rakás szöveget kapunk a konzolra amiatt, hogy már beolvastuk fájlból az adatokat, és utána a main-ben még egyszer hozzáadjuk a dolgokat. Oldjuk meg a problémát azzal, hogy a kulcsember és uj_talalkozo függvények elejére beszúrunk egy olyan sort, ami megvizsgálja, hogy az adott kulcsembert/találkozót hozzáadtam-e már, és ha igen, akkor kilép egy elegáns return segítségével.