# Feladat

Valósítsa meg az egész számokat tartalmazó zsák típust! Ábrázolja a zsák elemeit (az előfordulás számukkal együtt) egy sorozatban! Implementálja a szokásos műveleteket (elem betevése, kivétele, üres-e a halmaz, egy elem hányszor van a zsákban), valamint két zsák metszetét (a közös elemek a kisebb előfordulási számmal maradnak meg), továbbá egy zsák kiírását, és végül a másoló konstruktort és az értékadás operátort!

# Zsák típus

A feladat lényege egy felhasználói típusnak, a zsák típusnak a megvalósítása.

## Típusérték-halmaz1

Olyan számokat (ebben az esetben egész számokat: ℤ) tartalmazó *zsák* ,melyben tároljuk az elemet, valamint az adott elem előfordulását. Az *n*ℕ ennek a típusnak egy paramétere, amely a típusérték-halmaz zsák méretét határozza meg.

## Típus-műveletek2

1. *Keresés*

Megkeressük, hogy a soron következő elem megegyezik-e a dinamikus tömb i-edik pozíciójában lévő elemmel, majd visszatérünk egy logikai értékkel.

1. *Betevés*

A „keresés” művelet visszatérési értéke alapján(igaz vagy hamis) betesszük az elemet a dinamikusan lefoglalt tömbbe, majd ha az elem már előfordul a tömbben, akkor növeljük az adott elem darabszámát.

1. *Kivétel*

Megnézzük, hogy az adott zsák üres-e, ha igen hibát dobunk, ha nem, akkor a „keres” művelet segítségével csökkentjük az adott elem darabszámát, majd ha az adott elem darabszáma zérus lesz, akkor az elemet kivesszük a zsákból, és minden elemet eggyel balra tolunk a tömbben, majd végső lépésben csökkentjük a tömb hosszát.

1. *Előfordulás*

Megnézzük, hogy az adott zsák üres-e, ha igen hibát dobunk, ha nem, akkor megnézzük, hogy a keresett szám előfordul-e a zsákban, ha igen akkor ezt az elemet értékül adjuk egy változónak, majd kiírjuk azt.

## Üres-e

## Megvizsgáljuk, hogy a zsák elemszáma zérus-e.

## Metszet

## Adott két zsák. Megnézzük, hogy az első zsák i-edik eleme azonos-e a második zsák j-edik elemével, ha azonosak, akkor elhelyezzük őket egy tömbben, majd megvizsgáljuk, hogy a közös elemek előfordulásának száma az első vagy a második zsákban volt kevesebb. Attól függően, hogy melyik zsákban volt a kisebb előfordulás, elhelyezzük a kisebb előfordulás számát egy tömbben.

## Reprezentáció

A zsák típust egy dinamikusan lefoglalt egydimenziós tömb segítségével valósítjuk meg. A tömbben eltároljuk az adott elemet, az előfordulási számukkal együtt. Ahhoz hogy ezt megvalósíthassuk egy rekordra van szükség, melynek kettő egész szám típusú adattagja van ( egyik az adott szám, másik a darabszám).

A zsák létrehozásakor meg kell adni, hogy hány elemet szeretnénk eltárolni a zsákban ( vagyis mekkora legyen a mérete a zsáknak). A zsáknak vizsgáljuk a hosszát ( vagyis azt, hogy hány elem fordul elő benne)

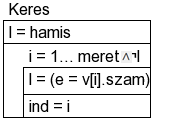
egy db változó segítségével.

A zsákba minden lépésben egy elemet rakhatunk be.

## Implementáció3

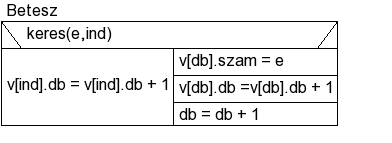
1. *Keresés*

A keresés algoritmusa, melynek 2 paramétere van, az aktuális elem( e ) és egy index ( ind) , majd a függvény egy logikai értékkel tér vissza.



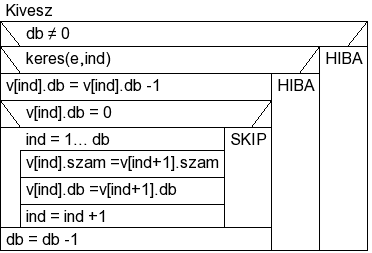
1. *Betevés*

Az elem betevése algoritmusa. Az eljárásnak 1 paramétere van, az aktuális elem ( e ). Az eljárás elején 0-ra inicializálunk egy int típusú ind nevű változót. Az eljárásban a „keres” függvényt használjuk fel, és adjuk neki paraméterként az aktuális elemet, valamint az indexet. Ha a függvény igaz értékkel tér vissza, akkor növeljük az aktuális elem darabszámát, egyébként pedig ha az elem nem fordult elő, betesszük a tömbbe, inicializáljuk az elem darabszámát 1-re majd növeljük a tömb hosszát.



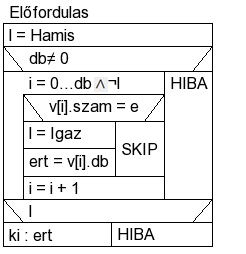
1. *Kivétel*

Az elem kivételének algoritmusa. Az eljárás hasonlít a betevés algoritmusára, azzal a különbséggel, hogy ha a megadott zsák üres ( db=0), akkor hibát dobunk, valamint akkor is, ha olyan elemet szeretnénk kivenni a zsákból, mely nem szerepel benne.



1. *Előfordulás*

Az adott eleme előfordulásának algoritmusa. Első lépésben megnézzük, hogy a zsák üres e, ha igen hibát jelzünk, továbbá akkor is, ha olyan elem előfordulására vagyunk kíváncsiak, mely nem szerepel az adott zsákban. Az eljárásban végig megyünk a zsák hosszán, és ha találunk olyan elemet, mely a zsákban van, akkor a logikai értéket igazra állítjuk, majd megadjuk az „ert” nevű változónak az aktuális elem darabszámát. Utolsó lépésben ki is írjuk azt.



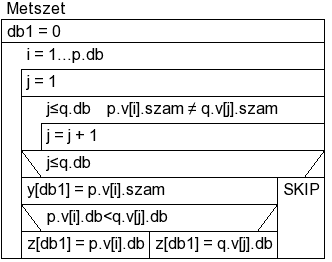
1. *Üres*-e

Az üresség vizsgálatához egy bool típusú függvényt definiálunk, melynek a visszatérési értéke megadja, hogy a zsák üres-e vagy sem. Ehhez a Bag osztály db nevű adattagját használjuk fel.



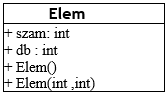
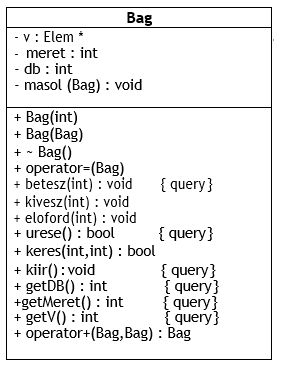
1. *Metszet*

Két zsák metszetének algoritmusa. A műveletet az összeadás (+) operátor felüldefiniálásával valósítjuk meg. Megnézzük hogy mely számok fordulnak elő mindkét zsákban, majd ezeket a számokat eltároljuk egy tömbben, és a számhoz tartozó előfordulását egy másik tömbben.

**

## Osztály

A zsák típusát egy osztály, valamint egy rekord segítségével valósítjuk. A konstruktoron keresztül állítható be zsák mérete.



A zsák elemeit egy dinamikusan lefoglalt tömbben tároljuk el. A konstruktor foglalja le a zsákbeli elemek számára a dinamikus tömböt, a destruktor végzi a felszabadítást, valamint szükség van az értékadás operátor és a másoló konstruktor felüldefiniálására.

A két zsák metszete meghatározásához az összeadás operátor felüldefiniálását használjuk. Az összeadást külső barát függvényként felüldefiniált változataival valósítjuk meg.

Kiegészítjük még az osztályt a zsák kiírására alkalmas metódusokkal, valamint getter metódusokkal.

A hibakezelésre három kivételt definiálunk. Az HIBAS\_MERET a helytelenül megadott zsák méretére utal. Az URES\_ZSAK kivétel akkor aktiválódik, ha a zsák nem tartalmaz egy elemet sem. A HELYTELEN\_SZAM kivétel akkor váltódik ki, mikor olyan elemet keresünk, amelyet a zsák nem tartalmaz.

# Tesztelési terv

Megvalósított műveletek tesztelése (fekete doboz tesztelés)

1,-Konstruktorok tesztelése-

2 zsák létrehozása. Az egyik zsákot üres konstruktorral hívjuk meg, a másikat megadott mérettel. Mindkettőnél vizsgáljuk, hogy az elemeik száma zérus-e, valamint hogy a méretük megfelelő e a követelményeknek.

2, -Üres-e vizsgálat-

Létrehozunk 2 zsákot, az egyiket üres konstruktorral, a másik adott mérettel. Mindkettőnél vizsgáljuk hogy üresek-e, továbbá az második zsákba belepakolunk egy elemet, azt vizsgálva hogy a zsák már nem üres.

3,-Elem berakása-

Létrehozunk 2 zsákot. Mindkettőnek megadjuk a méretét, majd az első zsákba beteszünk egy elemet, a második zsákba pedig két elemet, és vizsgáljuk, hogy a zsákok elemszáma illetve méretük megfelel-e a követelményeknek.

4,-Elem kivétele-

Egy zsákba beleteszünk két elemet, majd megvizsgáljuk, hogy üres-e, ha nem üres, akkor kiveszünk belőle egy elemet, majd megvizsgáljuk, hogy az adott zsák elemszáma megegyezik-e az elvártakkal.

5,-Másoló konstruktor vizsgálata-

Létrehozunk 2 zsákot. Az első zsákba beleteszünk 2 elemet, majd a második zsák paraméteréül adjuk az első zsákot, és megvizsgáljuk, hogy a második zsák adatai (mérete, elemszáma) azonosak-e az első zsák adataival..

6,-Értékadás operátor tesztelése-

Létrehozunk 2 zsákot. Az első zsákba beleteszünk két elemet, majd a második zsák értékéül adjuk az első zsákot ( = ), és vizsgáljuk, hogy a két zsák elemszáma, illetve elemik azonosak-e.

Tesztesetek a kód alapján (fehér doboz tesztelés)

1. Kivételek generálása és elkapása