A szerszámosláda

3. Tömb adatszerkezet - Array

Definíció 3.0.1 A tömb (array) egy olyan adatszerkezet, amely sok azonos típusú adatot tartalmaz egy egységes, indexelhető tárolóban. Az indexelés azt jelenti, hogy minden elemnek van egy sorszáma, ami lehetővé teszi, hogy gyorsan és egyszerűen hozzáférjünk bármelyik elemhez a tömbben. Az indexek 0-tól kezdődnek, és a tömb méretéig tartanak.

A C nyelv csak homogén (azonos típusú) adatok tárolását tudja biztosítani egy adott adatszerkezeten belül, azonban számos orientált nyelv (pl. Java) heterogén, úgynevezett polimorfikus tárolókat is tud alkalmazni. Amint az a fenti definícióból is kiderült, a tömböknek két alapvető része van:

- **elem (element)**: a tömbben eltárolt értékek.
- **pozíció (index)**: a tömbben található elemek egy-egy numerikus indexe, amely az elem azonosítására szolgál.

A tömbök gyors hozzáférést biztosítanak az elemeikhez, és lehetővé teszik több adat tárolását egyetlen logikai egységben. A C tömbök korlátozottak abban, hogy a méretük nem változik dinamikusan, hacsak nem biztosítjuk a memóriaterület újrafoglalását. Mindez azt jelenti, hogy a létrehozáskor meg kell határozni a méretüket, melyet a későbbiekben nem módosítunk. Az elemek, a tömbben számozott indexelés miatt folyamatos (egymást követő) memóriaterületen helyezkednek el, ami a gyors hozzáférést biztosítja.

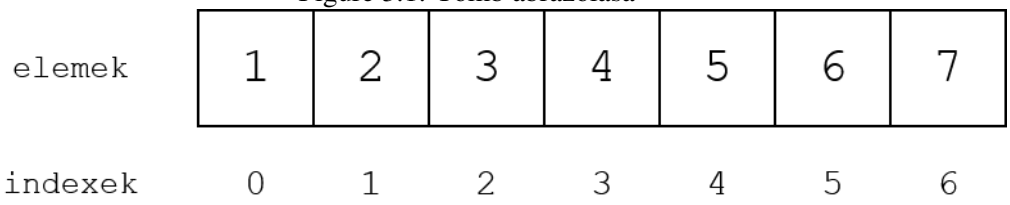
3.1 Tömb definiálása

Tömböket nagyon sokféleképpen tudunk definiálni. Vegyünk példának egy egyszerű deklarációt C nyelvben:

```
int array[7] = {1,2,3,4,5,6,7};
```

Ebben az esetben a következőket állapíthatjuk meg a tömbről: int típusú elemeket tartalmaz, az azonosítója array, a tömb mérete 7, és a következő elemeket tartalmazza sorban egymás után: 1, 2, 3, 4, 5, 6, 7. Fontos megjegyezni, hogy az indexelés 0-tól indul, így az elemek indexei a következők:

Figure 3.1: Tömb ábrázolása



3.2 Tulajdonságok

Amikor tömb adatszerkezetről beszélünk, valószínű mindenkinek, aki már rendelkezik egy kis programozás tapasztalattal eszébe jutnak a tömbbel kapcsolatos alapvető műveletek, melyek a következők:

1. **bejárás (traverse)**: kiírja a tömb összes elemét
2. **beszúrás (insertion)**: beszúr egy új elemet a tömbbe
3. **törlés (deletion)**: töröl egy adott indexen található elemet
4. **keresés (search)**: megkeres egy adott elemet a tömbben
5. **frissítés (update)**: megváltoztat egy adott indexen található elemet

Már korábban a bezető részben beszéltünk a különféle adatszerkezetek idő és hely komplexitásáról, azért ismételjük át, hogy milyen bonyultság jellemzi a tömb adatszerkezet működését. Egy adott elem elérése $O(1)$ bonyolultságú akár legjobb, akár legrosszabb esetről legyen szó. Ennek magyarázata az, hogy az indexelés következtében konstans időben meg tudunk határozni egy adott pozicíón található elemet. A tömbben való keresés bonyolultsága legjobb esetben $\Omega(1)$ míg legrosszabb esetben $O(n)$. Könnyen belátható, hogy amennyiben a keresett elem a legelső összehasonlított elem, azt mondhatjuk, hogy az algoritmus egy lépésből megtalálta azt. Azonban, ha a keresett elem az utolsó, vagy éppenséggel nincs is jelen a bemeneti adathalmazban, az összehasonlítások száma n-re nő. A beszúrás és törlés egyaránt $O(n)$ bonyolultságú, hiszen bárhová szúrunk be új elemet, vagy törlünk egy meglévő elemet, arra is figyelnünk kell, hogy a többi elemet megfelelőképpen tovább csúsztassuk, amely lineáris időbonyolultsághoz vezet.

3.3 Struktúra tömb

Ebben a részben az lesz a célunk, hogy struktúra definíció segítségével készítsük el a tömb adatszerkezet típust, majd a hozzá tartozó alap műveleteket. A következőkben minden adtszerkezetet ehhez hasonlóan fogunk szemléltetni. Amint az már kiderült az előző definíciók és fontos tudnivalók alapján a tömb adatszerkezetről a következő információkat kell ismerni: kapacitás (az elemek maximális száma), hossz (az elemek aktuális száma) és az elemek (elements). Ennek értelmében a következőképpen fog kinézni egy egész számokat tartalmazó tömb struktúra definíciója:

```
typedef struct {
    int capacity;
    int size;
    int *elements;
}IntArray;
```

Megjegyzendő, hogy a gyakori hibaüzenetek és hibakódok többszöri felhasználását egy vagy több megosztott fájl felhasználásával oldjuk meg a továbbiakban. A constants.h állomány tartalmazza a továbbiakban a fontosabb hibaüzeneteket és hibakódokat.

A constants.h fájl tartalma:

```
#ifndef CONSTANTS_H
#define CONSTANTS_H

#define MEMORY_ALLOCATION_ERROR_CODE -1
#define MEMORY_ALLOCATION_ERROR_MESSAGE "Error when allocating memory\n"


#define FILE_OPENING_ERROR_CODE -2
#define FILE_OPENING_ERROR_MESSAGE "Error when opening file\n"

#define NULL_POINTER_EXCEPTION_ERROR_MESSAGE "NULL pointer exception\n"
#define NULL_POINTER_EXCEPTION_ERROR_CODE -3

#define INVALID_POSITION_MESSAGE "INVALID position\n"
#define EMPTY_MESSAGE "EMPTY\n"
#define NOT_FOUND_MESSAGE "NOT FOUND\n"

#endif //CONSTANTS_H
```

Ha a továbbiakban is fel szeretnénk használni ezt a megosztott fájlt szükséges elhelyezzük ezt a rendszer beépített könyvtárainak mappájába (pl. ahol a stdio.h, stdlib.h állományok vannak). Ha ezt megtesszük, bármely projekt esetén tudunk hivatkozni ezekre. A továbbiakban ezt az eljárást fogjuk alkalmazni úgy, hogy az említett constants.h fájlt folyamatosan bővítjük.

Térjünk is rá az egész számokat tartalmazó tömb struktúra alapműveleteire.

3.3.1 Helyfoglalás

```
void createIntArray(int capacity, IntArray *pArray) {
```

```
    pArray->capacity = capacity;
    pArray->size = 0;
    pArray->elements = (int *) calloc(capacity, sizeof(int));
    if (!pArray->elements) {
        printf(MEMORY_ALLOCATION_ERROR_MESSAGE);
        exit(MEMORY_ALLOCATION_ERROR_CODE);
    }
}
```

A függvény két bemeneti paramétert kap: a capacity és a pArray. A capacity a létrehozni kívánt tömb méretét határozza meg, míg a pArray egy pointer az IntArray típusra, amely a függvény által inicializált és visszaadott tömbre mutat.

Az első sorban a függvény beállítja a pArray tömb kapacitását a bemeneti capacity értékre. Ez azt jelenti, hogy az IntArray tömb maximális mérete ennyi elemet tartalmazhat.

A második sorban a függvény beállítja a pArray tömb méretét 0-ra. Ez azt jelenti, hogy az IntArray tömb jelenleg nem tartalmaz semmilyen elemet.

A harmadik sorban a függvény dinamikusan lefoglalja a memóriát az IntArray tömb eleminek tárolására. Ez a calloc függvény segítségével történik, amely két bemeneti paramétert kap: a tömb méretét és az egy elem tárolásához szükséges memóriaméretet. A függvény által visszaadott érték egy pointer az IntArray típusú tömb elemeire mutat.

Az utolsó sorban a függvény ellenőrzi, hogy a memóriafoglalás sikeres volt-e. Ha a memóriafoglalás nem sikerült, akkor az függvény kiírja a megfelelő üzenetet. Ez megakadályozza, hogy a program futása folytatódjon a hibás memóriafoglalás miatt.

3.3.2 Üres tömb ellenőrzése

```
bool isEmpty(IntArray array) {
    return array.size == 0;
}
```

A isEmpty függvény egy bool típusú értéket ad vissza, ami azt jelzi, hogy a paraméterként kapott IntArray üres-e vagy sem. Az IntArray-t a függvény paraméterként kapja meg, és az array.size értékének vizsgálatával állapítja meg, hogy az üres-e vagy sem. Ha az IntArray size adattagja 0, akkor a függvény true értéket ad vissza, ami azt jelenti, hogy az IntArray üres. Ellenkező esetben, ha az IntArray size adattagja nem 0, akkor a függvény false értéket ad vissza, ami azt jelenti, hogy az IntArray nem üres.

3.3.3 Tele tömb ellenőrzése

```
bool isFull(IntArray array) {
    return array.size == array.capacity;
}
```

Az isFull függvény egy bool típusú értéket ad vissza, ami azt jelzi, hogy a paraméterként kapott IntArray tele van-e vagy sem. Az IntArray-t a függvény paraméterként kapja meg, és az array.size és array.capacity értékek összehasonlításával állapítja meg, hogy az IntArray tele van-e vagy sem. Ha az IntArray size adattagja megegyezik az IntArray capacity adattagjával, akkor a függvény true

értéket ad vissza, ami azt jelenti, hogy az IntArray tele van. Ellenkező esetben, ha az IntArray size adattagja nem egyenlő az IntArray capacity adattagjával, akkor a függvény false értéket ad vissza, ami azt jelenti, hogy az IntArray még nincs tele.

3.3.4 Elemek kiírása

```
void printArray(IntArray array) {
    if (isEmpty(array)) {
        printf(EMPTY_MESSAGE);
        return;
    }
    printf("The elements of the array: ");
    for (int i = 0; i < array.size; ++i) {
        printf("%i ", array.elements[i]);
    }
    printf("\n");
}
```

Ebben az esetben A függvény egy bemeneti paramétert kap, az array-t, amely az IntArray típusú tömbre mutató pointer.

Az első sorban a függvény ellenőrzi, hogy a bemeneti tömb üres-e, használva az isEmpty() függvényt. Ha az IntArray tömb üres, akkor kiírja a megfelelő üzenetet a konzolra, majd visszatér a függvényből.

Ha az IntArray tömb nem üres, akkor a függvény kiírja a a tömb elemeit.

3.3.5 Beszúrás előre

```
void insertFirst(IntArray* pArray, int item) {
    if (isFull(*pArray)) return;
    //shifting the rest of the elements downwards
    for (int i = pArray->size-1; i >= 0; --i) {
        pArray->elements[i+1] = pArray->elements[i];
    }
    pArray->elements[0] = item;
    pArray->size++;
}
```

Az első sorban a függvény ellenőrzi, hogy az IntArray tömb tele van-e, használva az isFull() függvényt. Ha az IntArray tömb tele van, akkor a függvény visszatér azonnal, mivel nem lehet új elemet hozzáadni.

Ha az IntArray tömb nincs tele, akkor a függvény a tömb minden elemét eltolja egy pozicióval jobbra, hogy az új elem beférjen a tömb elejére. Ehhez egy ciklus használatával végigmegy az IntArray tömbön, kezdve a tömb utolsó elemétől. Az elemeket eggyel jobbra tolja, hogy helyet adjon az új elemnek.

Az utolsó sorban a függvény hozzáadja az új elemet az IntArray tömb elejéhez. Az új elem indexe mindig 0 lesz. Ezután növeli a tömb méretét 1-gyel.

Ha az IntArray tömb már tele van, az új elem nem kerül hozzáadásra, és a függvény nem hajt végre semmilyen változtatást a tömbön.

3.3.6 Beszúrás hátra

```
void insertLast(IntArray* pArray, int item) {
    if (isFull(*pArray)) return;
    pArray->elements[pArray->size++] = item;
}
```

A függvény két bemeneti paramétert kap: a pArray egy pointer az IntArray típusú tömbre, amely az új elem hozzáadására szolgál, míg az item egy új elem, amelyet hozzá kell adni a tömb végéhez.

Az első sorban a függvény ellenőrzi, hogy az IntArray tömb tele van-e, használva az isFull() függvényt. Ha az IntArray tömb tele van, akkor a függvény visszatér azonnal, mivel nem lehet új elemet hozzáadni.

Ha az IntArray tömb nincs tele, akkor a függvény az új elemet hozzáadja a tömb végéhez. Ehhez a függvény a pArray->size indexű pozícióra helyezi az új elemet, majd növeli a tömb méretét 1-gyel. Az utolsó két művelet egyszerűbben is kifejezhető egyetlen sorban, amely először hozzáadja az új elemet a tömbhöz, majd növeli a tömb méretét 1-gyel.

A függvény által végzett műveletek eredményeként az IntArray tömb végén lesz az új elem, és a tömb mérete nőni fog 1-gyel. Ha az IntArray tömb már tele van, az új elem nem kerül hozzáadásra, és a függvény nem hajt végre semmilyen változtatást a tömbön.

3.3.7 Beszúrás egy adott pozícióra

```
void insertAt(IntArray* pArray, int position, int item) {
    if (isFull(*pArray)) return;
    if(position < 0 || position > pArray->size) {
        printf(INVALID_POSITION_MESSAGE); return;}
    if(pArray->size == position) {
        insertLast(pArray, item);
        return;
    }
    if(position == 0) {
        insertFirst(pArray, item);
        return;
    }

    //shifting the rest of the elements downwards
    for (int i = pArray->size-1; i >= position; --i) {
        pArray->elements[i+1] = pArray->elements[i];
    }
    pArray->elements[position] = item;
    pArray->size++;
}
```

A függvény három bemeneti paramétert kap: a pArray egy pointer az IntArray típusú tömbre, amelybe az új elemet szúrja be, a position az a pozíció, ahol az új elemet be kell szúrni, míg az item egy új elem, amelyet hozzá kell adni a tömbhöz.

Az első sorban a függvény ellenőrzi, hogy az IntArray tömb tele van-e, használva az isFull() függvényt. Ha az IntArray tömb tele van, akkor a függvény visszatér azonnal, mivel nem lehet új elemet hozzáadni.

Ha az IntArray tömb nincs tele, akkor a függvény ellenőrzi, hogy a megadott pozíció érvényes-e. Ha a pozíció kívül esik az IntArray méretének határain, akkor a megfelelő üzenetet jeleníti meg, majd visszatér azonnal, mivel nem lehet új elemet hozzáadni.

Ha a pozíció érvényes, akkor a függvény ellenőrzi, hogy az új elemet az IntArray tömb végéhez kell-e hozzáadni. Ha a pozíció megegyezik az IntArray tömb méretével, akkor a függvény az insertLast() függvényt használja az új elem hozzáadásához a tömb végéhez, majd visszatér azonnal.

Ha az új elemet az IntArray tömb elejéhez kell hozzáadni, akkor a függvény az insertFirst() függvényt használja az új elem hozzáadásához a tömb elejéhez, majd visszatér azonnal.

Ha az új elemet valahol a tömb közepére kell beszúrni, akkor a függvény az elemeket eltolja a tömbben, hogy felszabadítsa a helyet az új elemnek. Ehhez a függvény a pArray->size indexű pozíciótól kezdve végigiterál a tömbön az i változó segítségével. Minden iteráció során a függvény a pArray->elements[i] elemet áthelyezi az i+1 indexű pozícióra, így lépésről-lépésre a tömb elemei egy pozícióval lejjebb kerülnek. Ezután a függvény beilleszti az új elemet a megadott pozícióra, majd növeli a tömb méretét

3.3.8 Törlés egy adott pozícióról

```
void deleteItemAt(IntArray *pArray, int position) {
    if(isEmpty(*pArray)) { printf(EMPTY_MESSAGE); return;}
    if ((position < 0 || position >= pArray->size)) {
        printf(INVALID_POSITION_MESSAGE);
        return;
    }
    for (int i = position; i < pArray->size; ++i) {
        pArray->elements[i] = pArray->elements[i + 1];
    }
    pArray->size--;

}
```

Ha az IntArray nem üres, és a position értéke érvényes, akkor a függvény végigmegy az IntArray elemein a position értékétől kezdve egészen az utolsó elemig, és mindig az adott elemet a következő elemmel cseréli ki. Végül a függvény csökkenti az IntArray size adattagjának értékét 1-gyel, hogy jelezze, hogy egy elemet töröltek az IntArray tömbből.

3.3.9 Keresés

```
int search(IntArray pArray, int item) {
    if(isEmpty(pArray)) { printf(EMPTY_MESSAGE); return -1;}
    for (int i = 0; i < pArray.size; ++i) {
        if(pArray.elements[i] == item) return i;
    }
    return -1;
}
```

Ha az IntArray nem üres, akkor a függvény végigmegy az IntArray elemein, és ellenőrzi, hogy az adott elem egyenlő-e a item paraméterrel. Ha talál egy olyan elemet, amely egyenlő a item paraméterrel, akkor a függvény visszatér a talált elem indexével. Ha az IntArray objektum elemeit

végigmegyünk, de nem találunk olyan elemet, amely egyenlő a item paraméterrel, akkor a függvény visszatér a -1 értékkel, ami azt jelzi, hogy az elemet nem találták meg.

3.3.10 Módosítás

```
bool update(IntArray *pArray, int position, int newItem) {
    if(isEmpty(*pArray)) { printf(EMPTY_MESSAGE); return false;}
    if ((position < 0 || position >= pArray->size)) {
        printf(INVALID_POSITION_MESSAGE);
        return false;
    }
    pArray->elements[position] = newItem;
    return true;
}
```

A update függvény felelős egy elem frissítéséért az adott pozícióban az adott IntArray-ben. A függvény paraméterként kapja az IntArray-t és annak pozícióját, amelyet frissíteni szeretnénk, valamint az új elem értékét. Először ellenőrzi, hogy az IntArray üres-e, ha igen, akkor kiírja az üres üzenetet, majd hamissal tér vissza. Ha az IntArray nem üres, akkor ellenőrzi, hogy a pozíció érvényes-e, azaz a 0 és az IntArray mérete között van-e. Ha a pozíció érvénytelen, akkor kiírja az érvénytelen pozíció üzenetet, majd hamissal tér vissza. Ha a pozíció érvényes, akkor az adott pozícióban lévő elemet frissíti az új elem értékével, majd igazzal tér vissza.

3.3.11 Adott pozíciójú elem visszatérítése

```
void int getItemAt(IntArray array, int position) {
    if(isEmpty(array)) { printf(EMPTY_MESSAGE); return false;}
    if ((position < 0 || position >= array.size)) {
        printf(INVALID_POSITION_MESSAGE);
        return INT_MIN;
    }
    return array.elements[position];
}
```

Az getItemAt függvény egy adott pozícióban található elem értékét adja vissza az adott IntArray típusú tömbből.

3.3.12 Felszabadítás

```
void deallocateIntArray(IntArray *pArray) {
    free(pArray->elements);
    pArray->elements = NULL;
    pArray = NULL;
}
```

A deallocateIntArray függvény a kapott IntArray típusú tömb felszabadítására szolgál. Az elements pointerre hívja meg a free függvényt, majd a pointer értékét NULL-ra állítja. Ezután a pArray pointer értékét is NULL-ra állítja, hogy az esetleges további használatát megakadályozza és ne okozzon memóriakezelési hibákat. A függvény nem tér vissza semmilyen értékkel, csak végrehajtja a törlési

műveleteket.

3.4 Megoldott feladatok

Íme egy lehetséges array.h fájl struktúra:

```
#ifndef INTARRAY_ARRAY_H
#define INTARRAY_ARRAY_H

#include <stdbool.h>

typedef struct {
    int capacity;
    int size;
    int *elements;
}IntArray;

void createIntArray(int capacity, IntArray* pArray);
void printArray(IntArray array);
bool isFull(IntArray array);
bool isEmpty(IntArray array);
int getItemAt(IntArray array, int position);
void insertFirst(IntArray* pArray, int item);
void insertLast(IntArray* pArray, int item);
void insertAt(IntArray* pArray, int position, int item);
void deleteItemAt(IntArray* pArray, int position);
int search(IntArray pArray, int item);
bool update(IntArray* pArray, int position, int newItem);
void deallocateIntArray(IntArray *pArray);

#endif //INTARRAY_ARRAY_H
```

Az alábbi kódrészlet pedig egy lehetséges tesztelése lehet a megírt függvényeknek:

```
#include <stdlib.h>
#include <time.h>
#include <stdio.h>
#include <limits.h>
#include <constants.h>
#include "array.h"

int main() {
    IntArray array;
    int counter = 1;
    createIntArray(10, &array);
    //EMPTY
    printArray(array);
    for (int i = 0; i < array.capacity/2; ++i) {
        insertAt(&array, i, counter);
```

```
    counter++;
}
//1 2 3 4 5
printArray(array);
while(!isFull(array))
{
    if(counter % 2 == 0) insertFirst(&array, counter);
    else insertLast(&array, counter);
    counter++;
}
//10 8 6 1 2 3 4 5 7 9
printArray(array);

int value;
printf("Give an integer value:");
scanf("%i", &value);
int position = search(array, value);
if(position == -1)
{
    printf(NOT_FOUND_MESSAGE);
}
else {
    printf("The first occurrence of the value is: %i\n", position);
    printf("Give an integer value to update:");
    scanf("%i", &value);
    printf("Updating......\n");
    update(&array, position, value);
    printf("After update.\n");
    printArray(array);
}

srand(time(0));
while(!isEmpty(array)) {
    int pos = rand() % array.capacity;
    printf("Finding item at position %i -> ", pos);
    int result = getItemAt(array, pos);
    if(result != INT_MIN) {
        printf("%i\n", result);
        deleteItemAt(&array, pos);
    }
}
printArray(array);
deallocateIntArray(&array);
return 0;
}
```

3.5 Javasolt kérdések

Adott a következő struktúra definíció:

```
typedef struct {int capacity;
```

```
int size;
int *elements;
} IntArray;
```

1. Milyen adattípusokat tartalmaz az IntArray struktúra?
2. Mi a különbség az IntArray "capacity" és "size" mezői között?
3. Milyen értékek tárolhatóak az "elements" tömbben?
4. Hogyan lehet inicializálni egy IntArray struktúrát?
5. Definiálj egy IntArray típusú változót?
6. Ha a1 és a2 két IntArray típusú változó, helyes-e a következő megfeleltetés: a1=a2 ? Indokold meg a válaszod.
   Adott a következő függvény:

```
void insertFirst(IntArray* pArray, int item) {if (isFull(*pArray)) return;
    for (int i = pArray->size-1; i >= 0; --i) { pArray->elements[i+1] =
    pArray->elements[i]; } pArray->elements[0] = item; pArray->size++;}
```

7. Milyen paramétereket vár a függvény?
8. Mit csinál a függvény az "isFull" függvényhívással?
9. Milyen hatása van a ciklusnak a tömbre?
10. Írd meg a függvény meghívását aktuális paramétereket megadva.
11. Miért adjuk át pointerként a függvény első paraméterét?
12. Definiálj egy olyan struktúra tömböt, amely valós elemeket tud tárolni
    Adott a következő függvény:

```
int getItemAt(IntArray array, int position) { if(isEmpty(array)) {
    printf(ARRAY_EMPTY_MESSAGE);
   return false;
   }   if ((position < 0 || position >= array.size))  {
       printf(ARRAY_POSITION_MESSAGE);   return INT_MIN;  }   return
       array.elements[position];}
```

13. Milyen típusú értéket ad vissza a függvény?
14. Milyen paramétereket vár a függvény?
15. Mit térít vissza a függvény, ha az "array" paraméter által jelölt tömb üres?
16. Mit jelent az

```
"ARRAY_EMPTY_MESSAGE"
```

    állandó, és mi a szerepe a kódban?
17. Mit csinál a függvény, ha a "position" paraméter értéke kisebb, mint 0 vagy nagyobb, mint az **array** tömb mérete?
18. Mit jelent az

```
"ARRAY_POSITION_MESSAGE"
```

    állando, és mi a szerepe a kódban?
19. Milyen értéket ad vissza a függvény, ha az **array** tömbben lévő elem, amelyet a **position** paraméter jelöl, nem található?

20. Adj példát a függvény meghívására.