03 TÖMB ADATSZERKEZET

Tömbre vonatkozó alapműveletek. Struktúra tömb.

Adatszerkezetek műveletei

módosító, lekérdező

Keres

Beszúr

Töröl

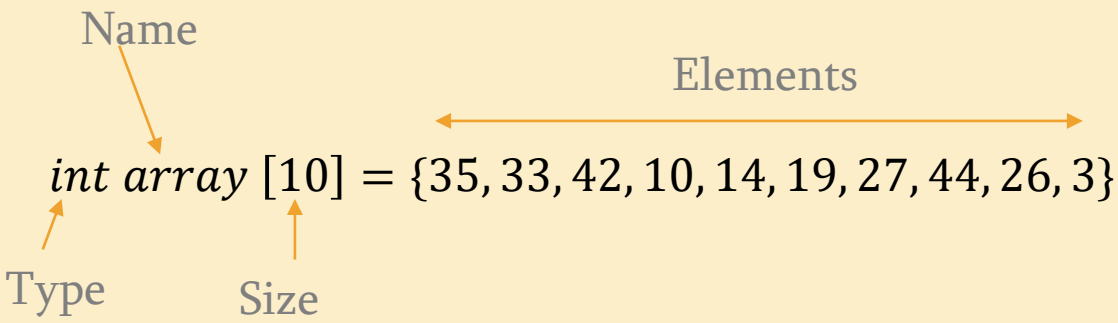
Min, Max

Előző

Következő

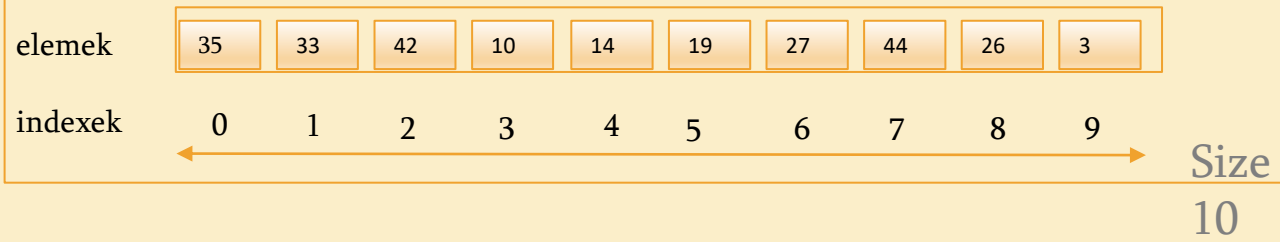
Tömb

- A tömb egy tároló, amely
  - fix számú elemet tud tárolni
  - az elemeknek azonos típusúnak kell lenniük
  - ábrázolása:



Tömb

- fix számú elemet tud tárolni
- az elemeknek azonos típusúnak kell lenniük
- ábrázolása:



Tömb. Alapműveletek.

- **kiírás**– az összes elem egymás után történő kiírása
- **beszúrás** – egy adott indexű elem beszúrása
- **törlés** – egy adott indexű elem törlése
- **keresés** – elem keresése megadott index vagy érték alapján
- **módosítás** – adott indexű elem értékének a módosítása

Tömb inicializálása

int arr[5] = {1, 2, 3, 4, 5};

int a[ ] = {1, 2, 3};

int arr[5] = {1, 2, 3}; ⇔a={1, 2, 3, ?, ?}

int a[6]={0};⇔a={0, 0, 0, 0, 0, 0}

int a[6]={8}; ⇔a={8, 0, 0, 0, 0, 0}

Tömb elemeinek a kiírása

```
int LA[] = {1,3,5,7,8};
int item = 10, k = 3, n = 5;
int i = 0, j = n;
printf("The array elements are :\n");
for(i = 0; i<n; i++) {

    printf("LA[%d] = %d \n", i, LA[i]);
}
```

```
The array elements are :
LA[0] = 1
LA[1] = 3
LA[2] = 5
LA[3] = 7
LA[4] = 8
```

Egy elem beszúrása adott pozícióra

```
int LA[] = {1,3,5,7,8};
 int item = 10, k = 3, n = 5; int i = 0, j = n;
 printf("The original array elements are :\n");
 for(i = 0; i<n; i++)
{ printf("LA[%d] = %d \n", i, LA[i]); }
n = n + 1;
while( j >= k)
{ LA[j+1] = LA[j]; j = j - 1; }
LA[k] = item;
printf("The array elements after insertion :\n");
for(i = 0; i<n; i++) { printf("LA[%d] = %d \n", i, LA[i]); } }
```

```
The original array elements are :
LA[0] = 1
LA[1] = 3
LA[2] = 5
LA[3] = 7
LA[4] = 8
The array elements after insertion :
LA[0] = 1
LA[1] = 3
LA[2] = 5
LA[3] = 10
LA[4] = 7
LA[5] = 8
```

Adott indexű elem törlése

```c
int LA[] = {1,3,5,7,8};
int k = 3, n = 5;
int i, j;
printf("The original array elements are :\n");
for(i = 0; i<n; i++) {
    printf("LA[%d] = %d \n", i, LA[i]);
}
j = k;
while( j < n) {
    LA[j-1] = LA[j];
    j = j + 1;
}
n = n -1;
printf("The array elements after deletion :\n");

for(i = 0; i<n; i++) {
    printf("LA[%d] = %d \n", i, LA[i]); }
```

```
The original array elements are :
LA[0] = 1
LA[1] = 3
LA[2] = 5
LA[3] = 7
LA[4] = 8
The array elements after deletion :
LA[0] = 1
LA[1] = 3
LA[2] = 7
LA[3] = 8
```

Adott értékű elem keresése

```
int LA[] = {1,3,5,7,8};
    int item = 5, n = 5;
    int i = 0, j = 0;
    printf("The original array elements are :\n");
    for(i = 0; i<n; i++) {
        printf("LA[%d] = %d \n", i, LA[i]);
    }
    while( j < n){
        if( LA[j] == item ) {
            break;
        }
        j = j + 1;
    }
    printf("Found element %d at position %d\n", item,
j+1);
```

```
The original array elements are :
LA[0] = 1
LA[1] = 3
LA[2] = 5
LA[3] = 7
LA[4] = 8
Found element 5 at position 3
```

Adott indexű elem értékének módosítása

```c
int LA[] = {1,3,5,7,8};
   int k = 3, n = 5, item = 10;
   int i, j;

   printf("The original array elements are :\n");

   for(i = 0; i<n; i++) {
       printf("LA[%d] = %d \n", i, LA[i]);
   }

   LA[k-1] = item;


   printf("The array elements after updation :\n");

   for(i = 0; i<n; i++) {
       printf("LA[%d] = %d \n", i, LA[i]);
   }
```

```
The original array elements are :
LA[0] = 1
LA[1] = 3
LA[2] = 5
LA[3] = 7
LA[4] = 8
The array elements after updation :
LA[0] = 1
LA[1] = 3
LA[2] = 10
LA[3] = 7
LA[4] = 8
```

Vizualizáció

https://www.simplilearn.com/tutorials/data-structure-tutorial/arrays-in-data-structure

Struktúra tömb - IntArray

Fontos jellemzői:

- a kapacitása (az elemek maximális száma)
- a hossza (az elemek aktuális száma)
- az elemek

Ennek értelmében az egész számokat tartalmazó tömb struktúra definíciója:

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

```
void createIntArray_t(int capacity, IntArray_t* pArray);
void printArray(IntArray_t array);
bool isFull(IntArray_t array);
bool isEmpty(IntArray_t array);
int getItemAt(IntArray_t array, int position);
void insertFirst(IntArray_t* pArray, int item);
void insertLast(IntArray_t* pArray, int item);
void insertAt(IntArray_t* pArray, int position, int item);
void deleteItemAt(IntArray_t* pArray, int position);
int search(IntArray_t pArray, int item);
bool update(IntArray_t* pArray, int position, int newItem);
void deallocateIntArray_t(IntArray_t *pArray);
```

Üres-e a tömb, tele-e a tömb

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

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

Tömb struktúra mezőinek inicializálása, helyfoglalás

```
void createIntArray_t(int capacity, IntArray_t *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);
    }
}
```

Tömb elemeinek kiíratása

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

Új elem bevitele az első pozícióra

```
void insertFirst(IntArray_t* 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++;
}
```

Új elem bevitele az utolsó pozícióra

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

Új elem bevitele egy adott pozícióra

```
void insertAt(IntArray_t* pArray, int position, int item) {

    if (isFull(*pArray)) return;

    if(position < 0 || position > pArray->size) {
printf(ARRAY_POSITION_MESSAGE); return;}

    if(pArray->size == position) {
        insertLast(pArray, item);return;}

    if(position == 0) {
        insertFirst(pArray, item);return;}

        for (int i = pArray->size-1; i >= position; --i) {
        pArray->elements[i+1] = pArray->elements[i];
    }
    pArray->elements[position] = item; pArray->size++;}
```

Adott pozíciójú elem törlése

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

Adott értékű elem keresése (első előfordulásának pozíciója)

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

Adott értékű elem módosítása

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

Adott pozíciójú elem visszatérítése

```
int getItemAt(IntArray_t 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];}
```

Helyfelszabadítás

```
void deallocateIntArray_t(IntArray_t
*pArray) {
    free(pArray->elements);
        pArray->elements = NULL;
        pArray = NULL;
}
```