Tömbök és mutatók:

A C nyelvben a tömbök kezelése és a mutatók használata összefügg. Minden tömbindexeléssel elvégezhető művelet mutatók használatával is végrehajthatók, ezért e két témával együtt foglalkozunk.

1. Tömbök:

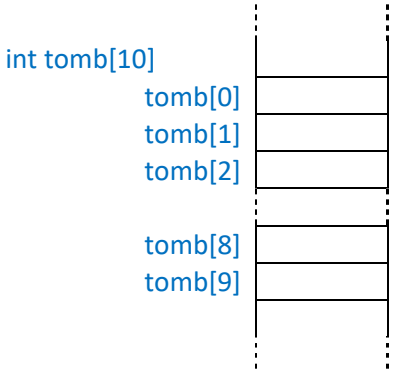
A tömbökben összetartozó értékeket tárolunk egyetlen név alatt, és azokra indexükkel (egész számokkal) hivatkozhassunk. A tömb matematikai megfelelője: egydimenziós esetben a vektor, többdimenziónál pedig a mátrix.

A tömb létrehozása (deklarálása), elhelyezkedése a memóriában:

Hozzunk létre egy egész (int) típusú tömböt, amelybe 10 db adatot tárolhatunk:

```
int tomb[10];
```

Tehát a tömbnek van típusa, jelen esetben 4 byte hosszú integer. Ami azt jelenti, hogy minden egyes eleme ugyanilyen típusú lesz. A neve tomb, amiben 10 db különböző egész számot tárolhatunk.



A C nyelvben a tömbök fordítási időben jönnek létre, tehát futási időben nem változhat a mérete (statikusak). Ezért deklaráláskor meg kell adni az elemek számát, ami csak egész konstans lehet.

Példa:

Töltsünk fel egy tömböt egész számokkal, majd írassuk ki a tartalmát a képernyőre.

Keressük meg a tömbben lévő legkisebb és legnagyobb számot.

Rendezzük a tömb elemeit növekvő (csökkenő) sorrendbe.

Megbeszélés:

1. Minimum és maximum kereső algoritmus elkészítése.
2. Két változó tartalmának kicserélése.
3. Rendező algoritmusok megbeszélése, a buborékrendezés részletes ismertetése.

A buborékrendezés:

Az algoritmus lényege, hogy az elejéről (0 indexű elemtől) indulva többször végig járjuk a tömböt, és összehasonlítjuk a szomszédos elemeket. Amennyiben nem a kívánt sorrendben vannak, akkor megcseréljük azokat. Az első menetben végéig haladva az elemeken, a legnagyobb (legkisebb) elem a lista végére kerül (felszáll, mint egy buborék). Így a következő meneteben elegendő egyel kisebb indexig elvégezni a vizsgálatokat és szükség esetén a cseréket. Összesen $((n-1)\cdot n)/2$ összehasonlítást kell elvégezni. Ezt az algoritmus hatékonyságának nevezzük.

Példa: rendezzük le a 12, 8, 5, 7, 3 számokat növekvő sorrendbe.

Index		1.	2.	3.	4.
0	12	8	5	5	3
1	8	5	7	3	5
2	5	7	3	7	
3	7	3	8		
4	3	12			

Megoldás:

```
#include <stdio.h>
#define MAX 5
int main(){
	int tomb[MAX];
	int i,j;
//Tömb feltöltése
	for (i=0; i<MAX; i++) { printf("Kerem a %d. szamot ", i+1); scanf("%d", &tomb[i]); }
//Minimum- és maximumkeresés
	int min, max;
	min = max = tomb[0];
	for (i=1; i<MAX; i++){
		if(tomb[i] > max)max = tomb[i];
		if(tomb[i] < min)min = tomb[i];
	}
	printf("Minimum = %d\nMaximun = %d\n", min, max);
//Buborékrendezés
	int csere;
	for(j=0; j<MAX-1; j++)
		for(i=0; i<MAX-1-j; i++)
			if(tomb[i] > tomb[i+1]){
				csere = tomb[i];
				tomb[i] = tomb[i+1];
				tomb[i+1] = csere;
			}
	for(i=0; i<MAX; i++) printf("%d\n",tomb[i]);
}
```

Példa:

Töltsünk fel egy tömböt tetszőleges számú elemmel, 0 végjelig. (Előre ne kérjük be az elemek számát.)

Megoldás:

```
#include <stdio.h>
int main(){
	int tomb[10];
	int db=0;
	printf("Kerek egy szamot: "); scanf("%d", &tomb[db]);
	while(tomb[db]){
		db++;
		printf("Kerek egy masik szamot: "); scanf("%d", &tomb[db]);
	}
	for (char i=0; i<db; i++) printf("%d\n",tomb[i]);
}
```

Példa:

Töltsünk fel egy tömböt számokkal, majd vizsgáljuk meg, hogy egy adott számot tartalmaz-e a tömb. Ha igen, írjuk ki, hogy a tömbben hol helyezkedik el, ha nem írjuk ki, hogy „Nincs”.

Megoldás:

```
#include <stdio.h>
#include <iostream>
#define MAX 5
int main(){
	setlocale (LC_ALL, "");
	int tomb[MAX], temp;
	int i,j;
	for (i=0;i<MAX;i++){
		printf("Kérem a %d. számot ", i+1); scanf("%d",&tomb[i]);
	}
	int keres;
	printf("Kerem a keresendo szamot "); scanf("%d",&keres);
	for (i=0;i<MAX;i++) if(tomb[i]==keres) break;
	if(i==MAX)
		printf("\nNincs ilyen elem\n");
	else
		printf("A %d. indexu elem = %d\n\n", i, keres);
}
```

Példa:

Az előző feladatot módosítsuk úgy, hogy a megtalált elemet törölje ki. Majd írjuk ki a tömböt.

Megoldás:

```
#include <stdio.h>
#include <iostream>
#define MAX 5
int main(){
	setlocale (LC_ALL, "");
	int tomb[MAX], temp, vege=MAX;
	int i,j;
	for (i=0;i<MAX;i++){
		printf("Kerem a %d. szamot ", i+1); scanf("%d",&tomb[i]);
	}
	int keres;
	printf("Kerem a torlendo szamot "); scanf("%d",&keres);
	for (i=0;i<MAX;i++) if(tomb[i]==keres) break;
	if(i==MAX)
		printf("\nNincs ilyen elem\n");
	else {
		for(; i<vege; i++)tomb[i]=tomb[i+1];
		vege--;
	}
	for(i=0; i<vege;i++) printf("%d\n",tomb[i]);
}
```

2. Mutatók (pointerek):

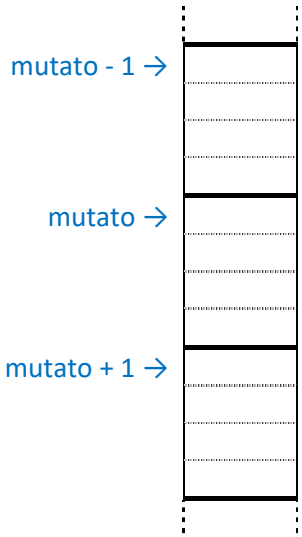
Korábban tárgyaltuk a változók tulajdonságait (típus, méret, név), de nem volt szó arról a memória címről, ahol az adott változó található. Ezt a memóriacímet (a változó első bájtjának a címét) hívjuk mutatónak (pointernek). Vagyis a pointer egy olyan változó, ami egy memóriacímre mutat. Ahogy egy változóval, a mutatóval is írhatjuk, illetve olvashatjuk az adott memória elemet.

A mutató deklarálása:

```
int *mutato;
```

Jelentése: annak a memóriaelemnek, amelyikre mutat a típusa int (4 bájtos egész).

A mutatókkal aritmetikai műveletek végezhetők. Ekkor az egység mindig az adott változó (amire mutat) a bájtban kifejezett mérete. Vagyis: mutato + 1 azt jelenti, hogy a következő változóra fog mutatni és nem a változó következő bájtjára.



Használata:

1. A mutató nevét változóként használva, tartalma az adott (mutatott) változó tényleges helye a memóriában. (Ez futásonként változhat.) Mérete 8 bájt.
2. Ha a mutató nevét csillag (*) előzi meg, akkor az a mutatott memóriahely tartalmát (értékét) jelenti:
   ```
   ertek = *mutato;
   ```
3. Ha egy változónak a címét szeretnénk használni, akkor a változó elé az és (&) jelet írjuk:
   ```
   mutato = &valtozo;      //Ekkor a mutató az adott változóra fog mutatni.
   ```
4. Ha a mutató nem valós memóriacímre (sehova sem) mutat, akkor az értéke NULL.

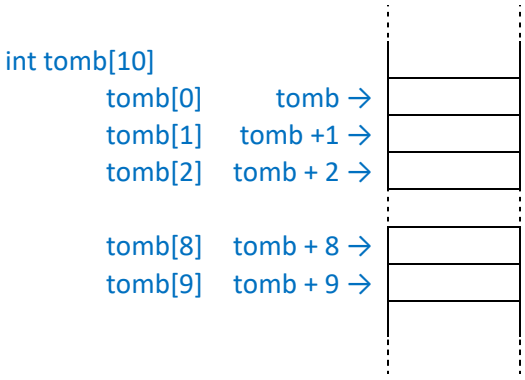
Példa:
Írjunk programot, amely bemutatja a változó és a mutatója használatát.

Megoldás:

```
#include <stdio.h>
int main(){
        int a,*b, c;
        printf("Kerek egy szamot: "); scanf("%d",&a);
        b=&a;
        c=*b;
        printf("%d  %d  %d\n", a, *b, c);
}
```

A mutató és a tömb:

Amikor a deklarált tömbnél (int tomb[10];) csak a tömb nevét használjuk, akkor az a tömb elejére mutató pointert jelenti. Ez a mutató nem változtatható meg, értéket nem adhatunk neki.



Példa:

Készítsünk programot, amely bemutatja a tömb használatát mutató segítségével.

Megbeszélés:

1. Mutatóaritmetika, mutatókkal végzett művelet.
2. Indexelés mutatók segítségével.

Megoldás:

```
#include <stdio.h>
#define MAX 5
int main(){
        int tomb[MAX];
        int i, *p=tomb;
//Tömb feltöltése
        for (i=0; i<MAX; p++, i++){
                printf("Kerem a %d. szamot ", i+1); scanf("%d", p);
        }
//A tömb kiírása
        for(i=0, p=tomb; i<MAX; p++, i++) printf("%d\n",*p);
}
```

Feladatok:

1. A program egy tetszőleges méretű (lebegőpontos) tömböt töltsön fel hőmérsékletekkel. Határozza meg, hogy a tömbben hány 0 és 100 fok közötti hőmérséklet található.

2. A program egy tetszőleges méretű tömböt töltsön fel életkorokkal. Számolja meg, hogy hány 30-59 év közötti életkor található a tömbben.

3. A program egy tetszőleges méretű tömböt töltsön fel osztályzatokkal. Számítsa ki a tömbben lévő osztályzatok átlagát.

4. A program egy tetszőleges méretű (lebegőpontos) tömböt töltsön fel egész számokkal, amelyek egy kocka oldalhosszúságát jelentik. Számítsa ki – a tömbben tárolt oldalhosszak alapján – a kockák térfogatát és felszínét. Ellenőrizze, hogy csak pozitív számokat adtunk-e meg.

5. A program egy tetszőleges méretű tömböt töltsön fel hónapok sorszámaival. A tömbben tárolt adatok alapján írja ki a hónapok melyik évszakban találhatók („tavasz”, „nyár”, „ősz”, „tél”). Hibás adatmegadás esetén adjon hibajelzést!

6. A program egy tetszőleges méretű tömböt töltsön fel napok sorszámaival (1-től 7-ig). Írja ki – a tömbben tárolt adatok alapján – a napok nevét („hétfő”, „kedd”, stb.). Hibás adatmegadás esetén adjon hibajelzést!

7. A program egy tetszőleges méretű tömböt töltsön fel számokkal. Határozza meg, hogy a tömbben hány páros és hány páratlan szám található.

8. A program egy tetszőleges méretű tömböt töltsön fel életkorokkal (maximum 100 éves korig). Majd életkoronként az alábbi szöveget jelenítse meg: ha az életkor 10, 20, 30, 40, 50, 60, 70, 80, 100: „Gratulálunk”; 1-29: „Fiatal”; 30-59: Középkorú”; 60-100: „Idős”. (Hibás adatmegadás esetén adjon hibajelzést!)

9. A program egy tetszőleges méretű tömböt töltsön fel számokkal, majd olvassa be egy [a, b] intervallum alsó és felső határát. Vizsgáljuk meg és írjuk ki, hogy a tömb hány eleme található az intervallumon belül, felette vagy alatta.

10. A program egy tetszőleges méretű (lebegőpontos) tömböt töltsön fel hőmérsékletekkel. A tömbben tárolt értékek alapján határozza meg – az adott hőmérsékleteken – a víz halmazállapotát, és írja ki azt („jég”, „víz”, „gőz”).

11. A program egy tetszőleges méretű tömböt töltsön fel születési évekkel, majd olvassa be az aktuális évszámot. Határozza meg az életkorokat és azt, hogy hány 45 év feletti életkor van.

12. A program egy tetszőleges méretű (lebegőpontos) tömböt töltsön fel számokkal, ezek egy-egy kör sugarát jelentik. A tömbben tárolt adatok alapján számolja ki a körök területét és kerületét.

13. A program egy tetszőleges méretű tömböt töltsön fel születési évekkel, majd olvassa be az aktuális évszámot. Határozza meg – a tömb adatai alapján – az átlagéletkort.

14. Olvasson be egy tömbbe (legalább 3 karakterből álló) karaktersorozatot. Határozza meg és írja ki az első, utolsó és a középső elemet (páros számú karakterek esetén két középső elem van).

15. A program egy tetszőleges méretű tömböt töltsön fel számokkal, majd kérjen be egy tetszőleges értéket. Keresse meg, hogy az utólag megadott érték hol (milyen indexen) található a tömbben. Ha nincs, azt is jelezze.

16. A program egy tetszőleges méretű tömböt töltsön fel számokkal. Keresse meg, a legkisebb és a legnagyobb értéket a tömbben.