Programozás I.

4.

Programozási tételek, hasznos dolgok

Egy pár hasznos dolog

#define

- A **#define** kulcsszó egy úgy nevezett makró. Ennek segítségével tudunk olyan ál-változókat beállítani, amelyek értéke az egész program során állandó.
- Ez egy olyan parancs, ami nem a program futása közben hajtódik végre, hanem még fordításkor.

A #define működése

- Pl.:
  - #define N 10
- Később a kódban:
  - int a=N+2;
- Ebből a fordító a következő csinálja:
  - int a=10+2;
- Majd ezt a sort fordítja le

A #define működése

- Pl.:
  - #define N 20
- Később ami a kódban van:
  - int tomb[N];
- Amit a fordító értelmezni fog:
  - int tomb[20];

Miért jó ez?

- Ha a feladat azt mondja, hogy csináljunk egy 10 elemű tömböt, akkor beírhatjuk a fájl elejére ezt:
  - #define N 10
- És később létrehozhatjuk a tömböt:
  - int tomb[N];
- És használhatjuk:
  - for (i=0; i<N; i++){…}

Miért jó ez?

- Ha később változtatni kellene, és a feladat azt mondaná, hogy legyen a tömb inkább 20 elemű, csak ezt kell átírni
  - #define N 20
- A kód többi része változatlan marad, mert mindenhol az **N**-et használtuk.

Átirányítás

- Tegyük fel, hogy egy programban van egy 10 elemű tömb, és a program elején be kell olvasni az elemeket.
- Írjuk be egy bemenet.txt-be a 10 számot, amit általában használunk.
- A **cat** paranccsal ki tudjuk íratni a fájl tartalmát, pl.:
  - cat bemenet.txt
  - 4 3 5 2 6 4 3 36 65 2

Átirányítás

- Linux rendszereken van lehetőség programok kimenetének átirányítására.
- Ezt a ‘ | ’ jellel lehet elvégezni.
- Ezt a tulajdonságot lehet program bemenetének automatizálására is használni.

Átirányítás

- A bemenet.txt-ben egy gyakran használt számsort mentettünk el.
- Tegyünk róla, hogy a program ezeket a számokat kapja meg, de anélkül, hogy be kellene gépelnünk:
  - cat bemenet.txt | ./program
- Ami történik:
  - A **cat** program kiírja a kimenetére a fájl tartalmát
  - Az átirányítás a **cat** kiemenetét átküldi a programunk bemenetére

Átirányítás

- Másik megoldás, amivel ugyanezt lehet elérni:
  - ./program < bemenet.txt
  - Itt a ‘<‘ jelet használjuk.

Átirányítás

- Harmadik megoldás: ne is használjuk az átirányítást.
- Ha a fájlból kimásoljuk a számokat, azok a parancssorba beilleszthetőek a **Ctrl + Shift + V** billentyű kombinációval.
  - A sima **Ctrl + V** nem jó!

Programozási alaptételek

Programozási alaptételek

- A programozási alaptételek olyan egyszerű algoritmusok, melyek megvalósítását minden programozónak tudnia kell, mert a legtöbb programban akár mindegyikük is előjöhet többször is.
- Fontos, gyakran használt algoritmusok
- Ezek az algoritmusok a következők:
  - Csere
  - Megszámlálás
  - Keresés
  - Minimumkeresés
  - Rendezés

Csere

- **4.1.1 Feladat**
  - Olvasson be két számot a billentyűzetről és tárolja el két változóba
  - Cserélje meg a két változó értékét

Rossz megoldás

```
int a, b;
scanf("%d %d", &a, &b);
a=b; // Az a változó felveszi b értékét
// Ezen a ponton a beolvasott a érték már nem létezik
b=a; // A két változó értéke ugyanaz, így ez nem csinál semmit
```

Jó megoldás

```
int a, b;
scanf("%d %d", &a, &b);
// Ahhoz, hogy a értéke ne tűnjön el, elmentjük egy másik változóba
int c=a;
a=b; // Az a változó felveszi b értékét
b=c; // A b változónak az eredeti a érték kell, amit szerencsére elmentettünk c-ben.
```

Megszámlálás

- **4.2.1 Feladat**
  - Hozzon létre egy hatelemű tömböt
  - Töltse fel a tömböt billentyűzetről beolvasott értékekkel
  - Olvasson be egy számot, majd számolja össze, hogy az adott szám hányszor szerepel a tömbben
  - Írja ki az eredményt

Az elve

- Kell egy új változó, ami az eredményt tárolja
- Végig kell menni a tömbön, minden elemre:
  - Ha az adott elem teljesíti a feltételt, akkor a számlálót növeljük

A kód

```
#define N 6
…. // int main() és egyebek
int i, tomb[N]; // A tömb, amiben keresünk
…. // Beolvasás
int ertek; // Az érték, amit keresünk
scanf("%d", &ertek); // Bekérjük
```

A kód

```
int szamlal=0; // Itt tároljuk a találatok számát, kezdetben 0
for (i=0; i<N; i++)
{
        if (tomb[i]==ertek) // Ha megfelelő az elem
        {
                szamlal++; // Növeljük a számlálót
        }
}
printf("%d db talalat.\n", szamlal);
```

Keresés

- **4.3.1 Feladat**
  - Hozzon létre egy tízelemű tömböt és töltse fel értékekkel
  - Olvasson be egy számot és tárolja el
  - Írassa ki, hogy a tömb elemei közt szerepel-e a beolvasott szám
  - Amennyiben szerepel, írja ki az első előfordulásának helyét (indexét)

Az elve

- Ha megtaláltuk, nem kell tovább keresni
- Ezért **while** ciklust használunk
- A tömb minden elemére:
  - Ha az adott elem a keresett, leállhatunk
- A leállást a **while** feltételében vizsgáljuk

A kód

```
#define N 10
…. // int main() és egyebek
int i, tomb[N]; // A tömb, amiben keresünk
…. // Beolvasás
int ertek; // Az érték, amit keresünk
scanf("%d", &ertek); // Bekérjük
```

A kód

```
// i<N, hogy ne menjünk ki a tömbből
// De akkor is álljunk le, ha az aktuális elem megfelelő
// Emlékezzünk a logikai és-re:
// Ha az első kifejezés hamis, nem értékeli ki  a másodikat
while (i<N && tomb[i]!=ertek)
{
        i++;
}
```

A kód

```
// Már csak meg kell vizsgálni, hogy megtaláltuk-e
// Ha megtaláltuk, i értéke a keresett index
// Ha nem, akkor végigértünk a tömbön, vagyis i==N
if (i<N)
        printf("Megtalaltuk a %d. indexen.\n", i);
else
        printf("Nincs ilyen a tombben.\n");
```

Minimumkeresés

- **4.4.1 Feladat**
  - Hozzon létre egy tízelemű tömböt és töltse fel értékekkel
  - Keresse meg és írassa ki a legkisebb érték helyét a tömbben
  - Amennyiben a legkisebb érték többször is szerepel a tömbben, az első előfordulásának a helyét írassa ki

Az elve

- Kezdetben higgyük azt, hogy az első érték (0. index) a legkisebb
- Végigjárjuk a tömb elemeit
  - Ha az adott elem kisebb, mint amiről eddig azt hittük
  - Akkor változik az elképzelésünk, és új jelölt van

A kód

```
#define N 10
…. // int main() és egyebek
int i, tomb[N]; // A tömb, amiben keresünk
…. // Beolvasás
// A legkisebb elemet a tömb-beli helye alapján tároljuk, így tudjuk azt is, hol van
int min_idx=0; // Az első tipp a 0-s indexen lévő elem
```

A kód

```
// A 0. indexet már nem kell vizsgálni
// De a többit igen
for (i=1; i<N; i++)
{
        // Ha az elem kisebb, mint az eddigi tipp
        if (tomb[i]<tomb[min_idx])
                min_idx=i; // Akkor ez az új tipp
}
printf("Legkisebb elem helye: %d, erteke: %d\n",
min_idx, tomb[min_idx]);
```

Rendezés

- **4.5.1 Feladat**
  - Hozzon létre egy tízelemű tömböt és töltse fel értékekkel
  - Rendezze a tömb elemeit növekvő sorrendbe
  - Írassa ki a tömb elemeit

Az elve

- Korábbi tételek használata
- Megkeressük a tömb legkisebb elemét
- Ezt megcseréljük az elsővel
- Majd megkeressük a maradék közül a legkisebbet
- Ezt megcseréljük a másodikkal
- ...

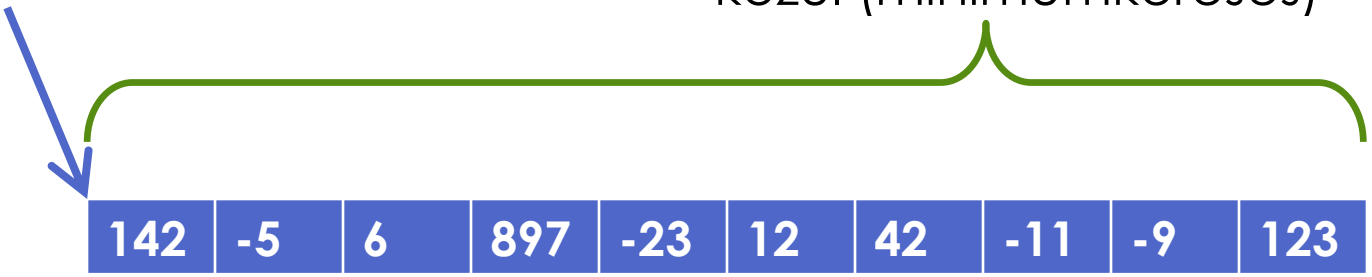
Az elve

- Két ciklus egymásba ágyazva
- A külső jelzi, hogy melyik helyre keressük a következő elemet
- A belső keresi meg a lehetőségek közül a legkisebbet

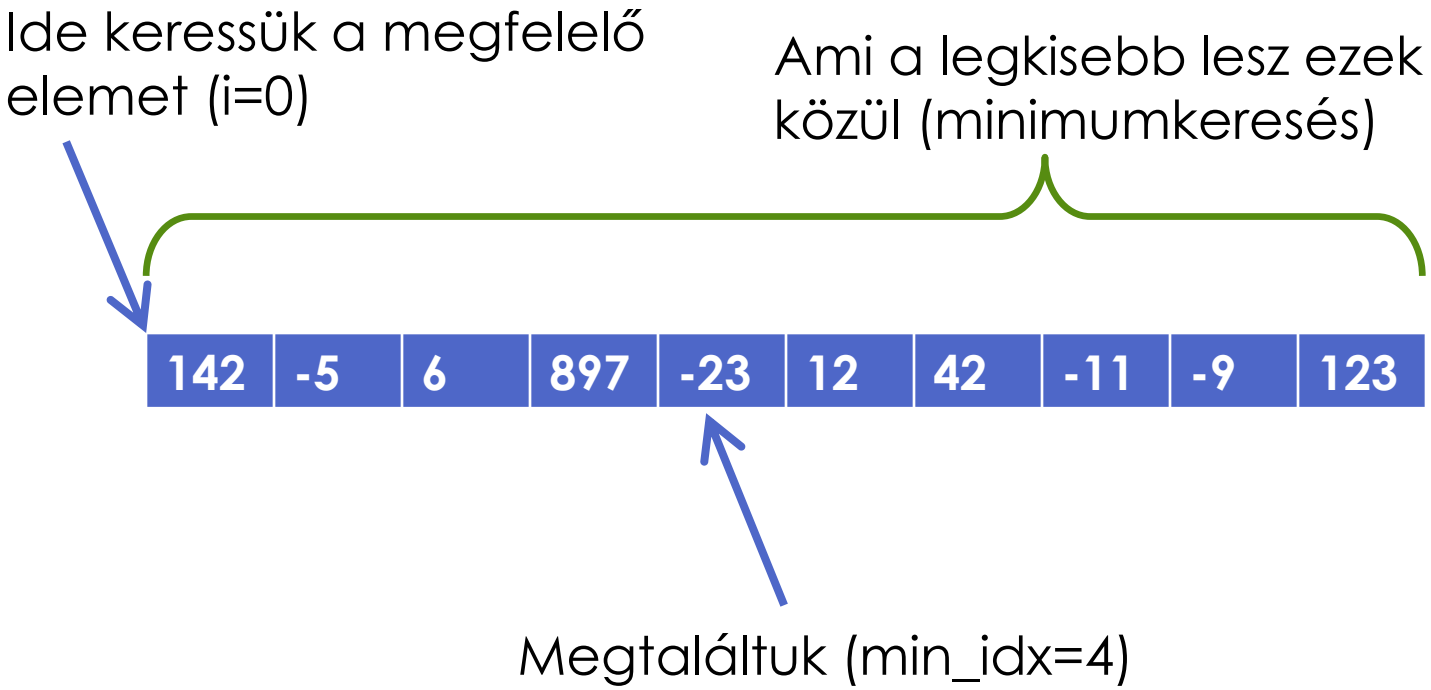
Szemléletesen

Ide keressük a megfelelő elemet (i=0)

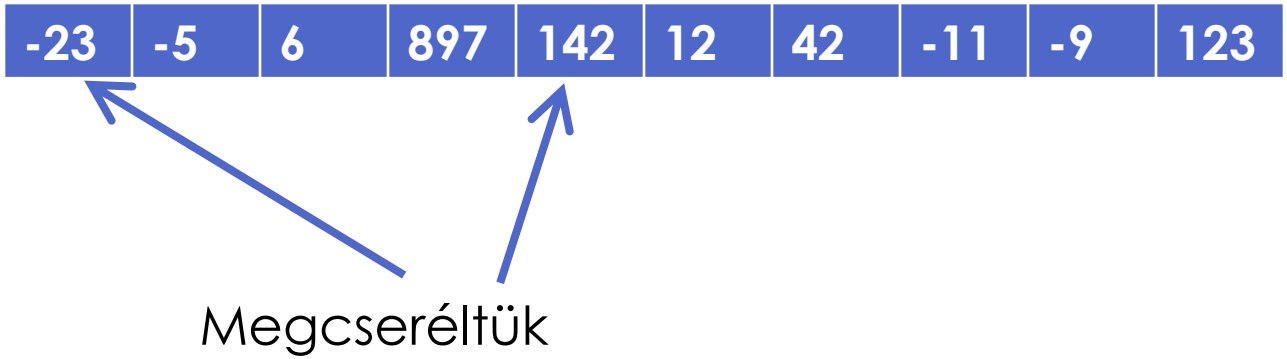
Ami a legkisebb lesz ezek közül (minimumkeresés)



Szemléletesen



Szemléletesen

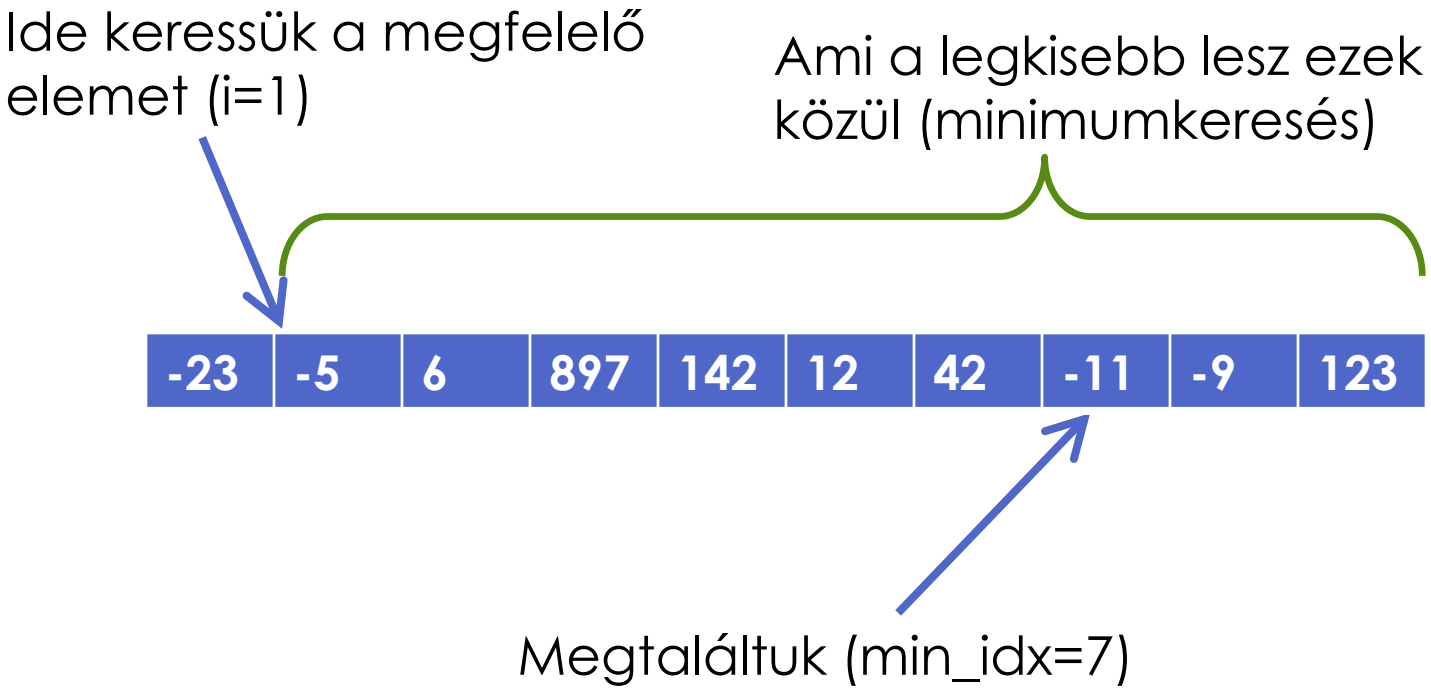


Szemléletesen

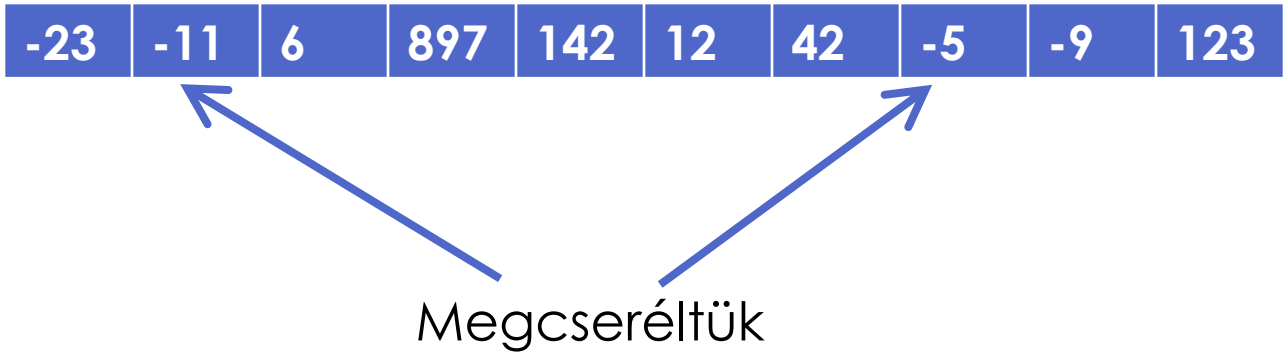
Ide keressük a megfelelő elemet (i=1)

Ami a legkisebb lesz ezek közül (minimumkeresés)

Szemléletesen



Szemléletesen



Szemléletesen

142	-5	6	897	-23	12	42	-11	-9	123
-23	-5	6	897	142	12	42	-11	-9	123
-23	-11	6	897	142	12	42	-5	-9	123
-23	-11	-9	897	142	12	42	-5	6	123
-23	-11	-9	-5	142	12	42	897	6	123
-23	-11	-9	-5	6	12	42	897	142	123
-23	-11	-9	-5	6	12	42	897	142	123
-23	-11	-9	-5	6	12	42	897	142	123
-23	-11	-9	-5	6	12	42	123	142	897
-23	-11	-9	-5	6	12	42	123	142	897

A kód

```
#define N 10
…. // int main() és egyebek
int i, j, tomb[N]; // A tömb, amiben keresünk
// Kell egy j változó is, mert két ciklus lesz
…. // Beolvasás
int min_idx=0; // Itt is minimumkeresés lesz
```

A kód

```
// Ha az első N-1 érték jó, akkor kizárásos alapon az utolsó is
for (i=0; i<N-1; i++)
{
// Az i. indexre keressük a megfelelő elemet, a korábbi helyeken már megfelelő van
        min_idx=i; // Így az i. Index az első tipp
// Itt jön a minimumkeresés
```

A kód

```
// Minimumkeresés, de nem az egész tömbön
// Csak az i. Indextől kezdve
        for (j=i+1; j<N; j++)
        {
                if (tomb[j]<tomb[min_idx])
                        min_idx=j;
        }
// Itt meg van a még helyre nem rakott elemek közül a legkisebb
```

A kód

```
// A megtalált legkisebb elemet rakjuk a helyére, vagyis az i. indexre
// Ha már eleve ott van, akkor felesleges cserélni
        if (min_idx!=i)
        {
                int csere=tomb[i];
                tomb[i]=tomb[min_idx];
                tomb[min_idx]=csere;
        }
}  // Külső ciklus vége
```