Kvíz

- http://www.menti.com
- mindenki a saját azonosítójával (bbbbnnnn) lépjen be

! aki **nem** az azonosítóját használja, annak nem fogjuk tudni beírni a pontokat a kvízre

4170 6972

Alprogramok, függvények I

KURZUS

Alprogramok, azaz függvények

programok építőkövei

utasítás gyűjtemény

közös név alatt

→ a függvények meghívásakor kerülnek lefuttatásra

- a forráskód alprogramokba/függvényekbe való rendezése megkönnyíti a forráskód rendezését/áttekintését
  - elősegíti a nagy és komplex programok fejlesztését
    - megírás, megértés, módosítás, tesztelés és karbantartás
- eddig a programokat a következő képpen írtuk
  - főprogram: a **`main`** – amely nem hiányozhat egy C programból sem
  - a **standard C könyvtár**ból felhasznált függvények – amelyeket a **`main`** programrészből hívtunk meg
    - a megfelelő header file beillesztésével

Egy program szerkezete

- egy program egy vagy több függvényből áll

```c
#include <stdio.h>

int main()
{
    int a, b, max, min, sum;

    printf("> adj meg 2 egesz szamot: ");
    scanf("%d %d", &a, &b);

    max = a > b ? a : b;
    min = a < b ? a : b;
    sum = a + b;

    printf("> max(%d, %d) = %d\n", a, b, max);
    printf("> min(%d, %d) = %d\n", a, b, min);
    printf("> sum(%d, %d) = %d\n", a, b, sum);

    return 0;
}
```

```c
#include <stdio.h>

int maximum(int a, int b)
{
    return a > b ? a : b;
}

int minimum(int a, int b)
{
    return a < b ? a : b;
}

int osszeg(int a, int b)
{
    return a + b;
}

int main()
{
    int a, b, max, min, sum;

    printf("> adj meg 2 egesz szamot: ");
    scanf("%d %d", &a, &b);

    printf("> max(%d, %d) = %d\n", a, b, maximum(a, b));
    printf("> min(%d, %d) = %d\n", a, b, mininum(a, b));
    printf("> sum(%d, %d) = %d\n"`, a, b, osszeg(a, b));

    return 0;
}
```

hasonlóak a matematikai függvényekhez

$$z = osszeg(x, y)$$

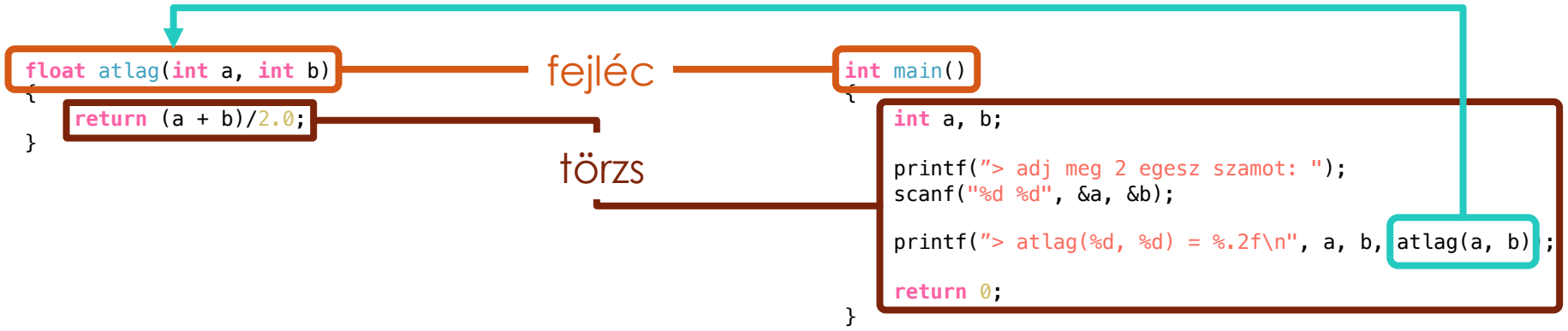
$$osszeg(x, y) = x + y$$

Definíció és meghívás

- alprogram/függvény felépítése

```
visszatérésiTípus függvénynév(paraméterlista)
{
    utasítás(ok);

    return visszatérésiÉrték;
}
```

- visszatérés a függvényből: a **`return`** utasításkor vagy az **utasítás lista végrehajtás** végeztével



A fejléc

- a függvény **visszatérési értékének típusa**
  - *pl.* `int` `float` `char` `double` stb.
  - ha a függvény nem térít vissza értéket, akkor `void`
- **KORLÁTOZÁS** nem téríthető vissza tömb (array) vagy más függvény
- ha hiányzik a visszatérési érték típusa
  - **C89** implicit úgy értelmezi, hogy a függvény visszatérít egy egész értéket
  - **C99** és **C11** hibát jelez
- a függvény egyedi neve
  - hasonlóképpen egyedi, mint egy változó azonosítója
- paraméterek listája
  - kerek zárójelben adjuk meg, tartalmazza az `adattípust` és a **`változó azonosító`**ját, vesszővel választjuk el őket
  - az adattípust mindegyik változó esetén fel kell tüntetni, még akkor is, hogyha ugyan azon típusúak
  - ha a lista üres, azaz a függvény nem kap paramétereket, akkor a `void` típust használjuk vagy üresen hagyjuk

```
visszatérésiTípus függvénynév paraméterlista)
{
    utasítás(ok);
    return visszatérésiÉrték;
}
```

```
int tombElemeinekOsszege(int tomb[], int hossz)
int atlag(int a, int b, int c)
int main(void)
int main()
```

A függvények meghívása

- a függvény hívás során megadjuk a függvény **nevét** és kerek zárójelben megadjuk a paraméterek értékét, azaz az **aktuális paraméterek** listáját

```
függvénynév(paraméterlista);
```

- a függvényhívás egy kifejezés
  - szerepelhet, mint egy operandus egy kifejezésben, vagy mint egy aktuális paraméter egy másik függvény hívásakor
  - egy önálló utasításként is szerepelhet, vagyis a hívást a **;** (pontosvessző) karakterrel zárjuk

fejléc

```
int faktorialis(int szam)
```

hívás

```
e = faktorialis(10);
```

Példa

faktorialis.c

```c
#include <stdio.h>

int faktorialis(int szam)
{
    int f = 1;
    int i;

    for(i = 2; i <= szam; i += 1)
    {
        f *= i;
    }

    return f;
}

int main()
{
    int szam;
    int eredmeny;

    printf("> adj meg egy szamot: ");
    scanf("%d", &szam);
    eredmeny = faktorialis(szam);
    printf("> %d! = %d\n", szam, eredmeny);

    return 0;
}
```

```
faktorialis(10)

szam: 10
f: 1 → 3628800
i: 2 → 11

(vissza):
```

Megjegyzések

- azon paramétereket amelyekkel definiáljuk a függvényt **formális paraméterek**nek nevezzük, vagy csak egyszerűen **paraméterek**

```
int maximum(int a, int b)
{
    return a > b ? a : b;
}
```

- az értékeket amelyekkel meghívásra kerül egy függvény **aktuális paraméterek**nek hívjuk, vagy csak egyszerűen **argumensek**

```
int main()
{
    int a, b;

    printf("> adj meg 2 egesz szamot: ");
    scanf("%d %d", &a, &b);
    printf("> max(%d, %d) = %d\n", a, b, maximum(a, b));

    return 0;
}
```

- a formális és aktuális paraméterek szerepelhetnek **ugyanazzal** a névvel (lásd fenti példa) vagy akár **különböző** nevekkel
- az aktuális paramétereket ugyan abban a sorrendben adjuk meg, ahogy azok a formális paraméterek sorrendjében szerepelnek

Megjegyzések

- a paraméterek átadása és feldolgozása **tetszőleges** sorrendben történik

- a **C89** megengedi, hogy a függvény hívása megelőzze annak definícióját vagy deklarálását
- a **C99** és **C11** kötelezik a függvény definícióját vagy deklarálását annak hívása előtt

kvíz 1. kérdés

4170 6972

A függvények deklarálása

- a függvényhívás pillanatában a fordítóprogramnak ismernie kell a függvény **visszatérési értékének típusát** és a **formális paraméterek listáját**, azaz a függvény **prototípusát**

- **TEHÁT** a függvényt a meghívás előtt **deklarálni** vagy **definiálni** kell

- **DEKLARÁLÁS** a függvény **prototípus**ának a deklarálását jelenti, amely a függvény fejlécéből áll és amelyet **;** (pontosvessző) zár le
- a prototípus meg kell egyezzen a definiáláskor megadott fejléccel

```
void fuggveny(int i, float x, char c);

void fuggveny(int, float, char);
```

a paraméterek változónevei elhagyhatók

ekvivalens prototípusok

Láthatóság

- egy azonosító (változó, konstans, stb.) kizárólag abban a blokkban látható és elérhető, amelyben deklarálva volt:
  - **globális**
    - a teljes programban láthatók és elérhetők
    - a program tetszőleges helyén definiálhatók és módosíthatók
  - **lokális**
    - a függvényen/blokkon belül vannak deklarálva
    - a függvénybe/blokkba való belépéskor jönnek létre
    - a végén megszűnnek → értékük elveszik **!!!**
    - minden függvény/blokk kizárólag csak a sajátjait látja **!!!**

```
char c = 99;
int main()
{
    printf("%c\n", c);
    {
        int a = 5;
        printf("> %d\n", a);
        {
            int a = 6;
            char c = 'A';
            printf("> %d\n", a);
            printf("> %c\n", c);
        }
    }
    printf("> %c\n", c);

    return 0;
}
```

az **a** és **c** változók értéke maszkolva van

- a függvény **globális változók**on, **pointerek**en és a **return** utasításon keresztül tud információt visszaküldeni a meghívó függvényhez

FIGYELEM változók maszkolása megnehezíti a forráskód megértését és javítását!

Példa

maszkolas.c

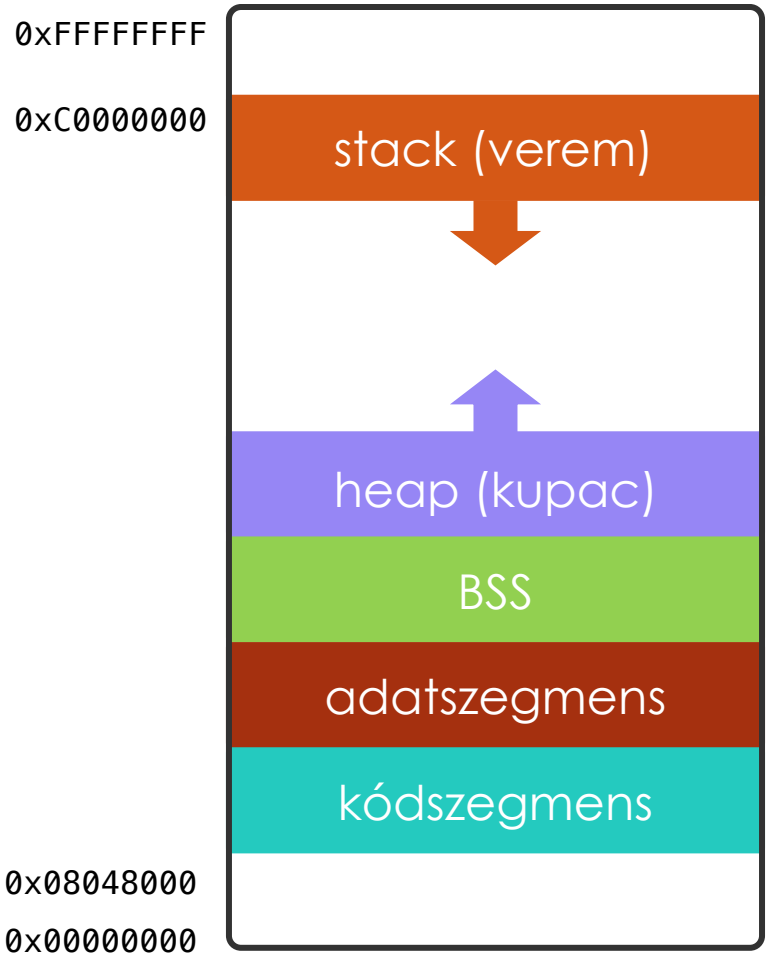
```c
#include <stdio.h>

char c = 98;

int main()
{
    printf("> %c\n", c);                    // → b
    {
        int a = 5;
        printf("> %d\n", a);                // → 5
        {
            int a = 6;
            char c = 'A';
            printf("> %d\n", a);            // → 6
            printf("> %c\n", c);            // → A
        }
        printf("> %d\n", a);                // → 5
    }
    printf("> %c\n", c);                    // → b
                                            //   az a változó itt már nincs definiálva

    return 0;
}
```

mennyi az **a** értéke itt?

kvíz 2. kérdés

4170 6972

Program memóriacímzése



- stack (verem) → lokális változók, függvények argumentumai, visszatérési címek
- heap (kupac) → dinamikusan lefoglalt adatok
- BSS → globális és statikus változók, amelyek nincsenek inicializálva
- adatszegmens → inicializált globális és statikus változók, konstansok
- kódszegmens → a program bináris kódja

Paraméterek átadása

- C nyelvben a paraméterek átadása kizárólag **érték szerint** történik
  - az aktuális paraméter értéke **bemásolásra kerül** a meghívott függvény megfelelő formális paraméterébe

```c
#include <stdio.h>

int summa1N(int n);

int main()
{
    int n, sum;

    printf("> adj meg egy egesz szamot: ");
    scanf("%d", &n);

    sum = summa1N(n);

    printf("> az elso %d szam osszege: %d\n", n, sum);

    return 0;
}

int summa1N(int n)
{
    int s = 0;

    for(; n >= 0; s += n, n--);

    return s;
}
```

függvény prototípusa

beolvasásra kerül az **n** változó értéke

az **n** változó értéke átmásolásra kerül, a másolat értéke teljesen független a főprogramban levő **n** változó értékétől

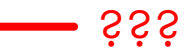
az **n** változó értéke változatlan marad

pointer

előny: megóvja a meghívó függvény változóit
hátrány: a másolás nem hatékony művelet nagy adatszegmensek esetén

Paraméterek átadása – a klasszikus hiba

```c
#include <stdio.h>

void osszeg(int a, int b, int sum)
{
    sum = a + b;
}

int main(void)
{
    int sum;

    sum = 0;
    printf("> elotte: %d\n", sum);

    osszeg(5, 6, sum);
    printf("> utana: %d\n", sum);

    return 0;
}
```



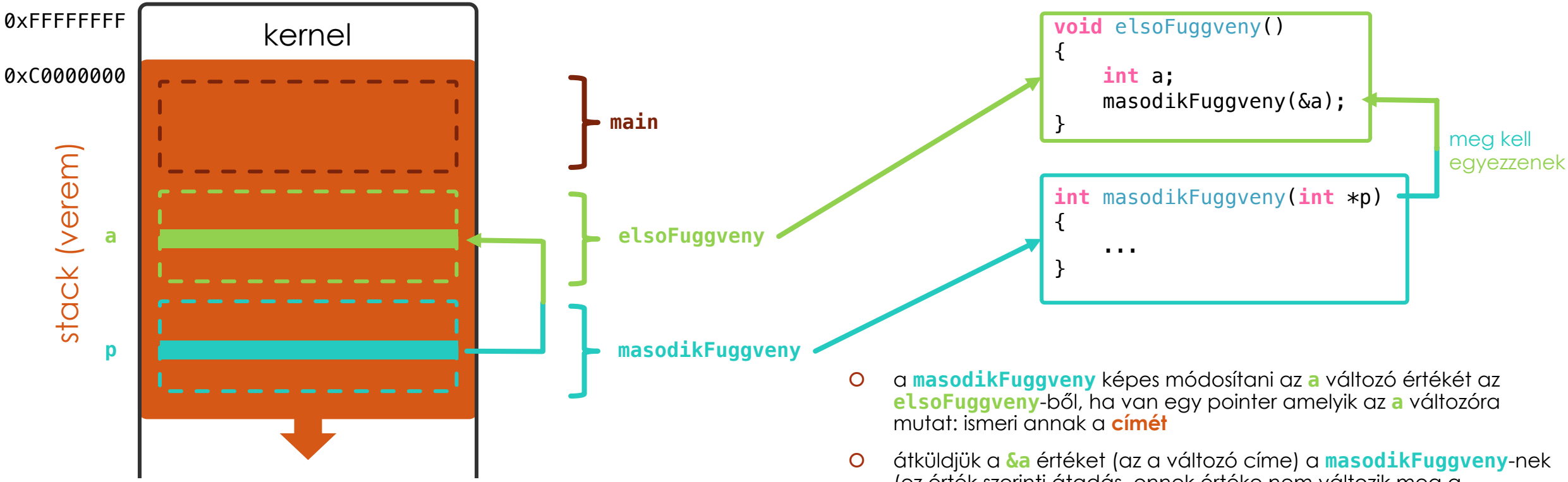
kvíz 3. kérdés

4170 6972

- mivel a paraméterek lokális változók, ezért a függvényből való visszatéréskor megszűnnek
- emiatt paraméteren keresztül közvetlenül nem lehet értéket visszaadni a függvényt meghívó programrésznek
- a példában szereplő **sum** változó nem magát a változót, hanem a **sum** változó értékének egy másolatát kapja meg, vagyis **0**-t
- a függvényben ugyan a **sum** lokális változó felveszi az összeg helyes értékét, de megszűnik létezni a függvényből való visszatéréskor

Paraméterek átadása

- és az örökös kérdés: hogyan módosíthatná egy függvény a kapott paraméter értékét úgy, hogy a módosítás a meghívó függvényben is látható maradjon? ➡ pointerek



- a **masodikFuggveny** képes módosítani az **a** változó értékét az **elsoFuggveny**-ből, ha van egy pointer amelyik az **a** változóra mutat: ismeri annak a **címét**
- átküldjük a **&a** értéket (az a változó címe) a **masodikFuggveny**-nek (ez érték szerinti átadás, ennek értéke nem változik meg a **masodikFuggveny** végrehajtása során)

Paraméterek átadása

érték szerinti átadás

```c
#include <stdio.h>

int fuggveny(int a);

int main()
{
    int a = 1, b;
    b = fuggveny(a);

    return 0;
}
```

- az **a** változó értéke bemásolásra kerül egy lokális változóba a **fuggveny** meghívásakor
- az **a** értéke a **main** programrészben nem változik

```c
#include <stdio.h>

int main()
{
    int a;
    scanf("%d", &a);

    return 0;
}
```

- abban az esetben, ha szeretnénk az **a** változó értékét megváltoztatni, meg kell mondjuk a **scanf** függvénynek, hogy hol találja az **a** változót: a **&a** címen

pointeren keresztül történő átadás

Paraméterek átadása

példa

swap.c

```c
#include <stdio.h>

void swap(int *, int *);

int main()
{
    int a = 2;
    int b = 5;

    printf("> a = %d\tb = %d\n", a, b);
    swap(&a, &b);
    printf("> a = %d\tb = %d\n", a, b);

    return 0;
}

void swap(int *p, int *q)
{
    int tmp;
    tmp = *p;
    *p = *q;
    *q = tmp;
}
```

a formális paraméterek **int** típusú pointerek

a függvény hívásakor átadjuk a változók **címét**

a függvény a * operátort használjuk, hogy hozzáférjünk a változók értékéhez, amelyet a továbbított címen tárolnak

- ***p** ekvivalens a főprogram **a** változójával
- ***q** ekvivalens a főprogram **b** változójával

Példa

min_max.c

keressük meg a legnagyobb és a legkisebb számjegyet egy egész számban

```c
#include <stdio.h>

void min_max(int szam, int *min, int *max)
{
    int m;
    if(szam < 10)
    {
        printf("> a legkisebb es legnagyobb szamjegy azonos : %d\n", szam);
    }
    else
    {
        *max = szam%10;
        *min = *max;
        szam /= 10;

        while(szam != 0)
        {
            m = szam%10;
            if(m > *max)
            {
                *max = m;
            }
            else if(m < *min)
            {
                *min = m;
            }
            szam /= 10;
        }
    }
}

int main()
{
    int szam, min, max;

    printf("> adj meg egy egesz szamot\n");
    printf("> szam : ");
    scanf("%d", &szam);

    min_max(szam, &min, &max);

    printf("> a %d szam legkisebb szamjegye %d, mig a legnagyobb %d\n", szam, min, max);
    return 0;
}
```

Paraméterek átadása

tömbök

- tömbök függvényeknek való átadásának két ekvivalens módja van
  - tömb formában kezeljük őket `típus t[]` vagy `típus m[][]`
  - pointer formában kezeljük őket `típus *t` vagy `típus **m`

```c
#include <stdio.h>

int atlag(int t[], int hossz)
{
    int i;
    int s = 0;

    for(i = 0; i < hossz; i++)
    {
        s += t[i];
    }

    return (double)s/hossz;
}

int main()
{
    int t[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

    printf("> a tomb elemeinek atlaga: %.2f\n", atlag(t, 10));

    return 0;
}
```

```c
#include <stdio.h>

int atlag(int *t, int hossz)
{
    int i;
    int s = 0;

    for(i = 0; i < hossz; i++)
    {
        s += *(t + i);
    }

    return (double)s/hossz;
}

int main()
{
    int t[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

    printf("> a tomb elemeinek atlaga: %.2f\n", atlag(t, 10));

    return 0;
}
```

Érték és mellékhatás

- **ÉRTÉK** a függvény értéke/a függvény **kiértékelése**
- a függvény lefut és a hívás helyén levő kifejezésbe behelyettesítésre kerül annak visszatérési értéke

```
printf("> %d\n", fakt(6));
printf("> %d\n", fakt(6));
printf("> %d\n", fakt(6));
```

```
> 720
> 720
> 720
```

- **MELLÉKHATÁS** a függvény lefutásakor valahol változást okoz
- ezt abban az esetben is mellékhatásnak nevezzük, hogyha ez volt a függvény kifejezett célja

```
printf("> %d\n", rand());
printf("> %d\n", rand());
printf("> %d\n", rand());
```

```
> 56383
> 123
> 6823420
```

kvíz 4. kérdés

4170 6972

- általában igyekszünk olyan függvényeket írni amelyeknek csak értéke vagy csak mellékhatása van
- eszerint kétféle függvény van:
  - parancsfüggvény (command): azért használjuk, hogy hatása legyen
  - lekérdező függvény (query): kérdéseket teszünk fel, kiszámol valamit, mellékhatása nincs
- **FIGYELEM** ha ez a kettő keveredik, akkor könnyen átláthatatlan programot eredményez

command-query separation

Visszatérés a függvényből

- a függvény törzsében elhelyezett `return` utasítással visszatérünk a függvény hívásának helyére, egy függvényben több ilyen pont is lehet
- ezzel egyben megadjuk a visszatérési értéket is, ezt még a függvény értékének is nevezzük: maximum egy érték `return` utasításonként
- **FONTOS** a `return` utasítás a fenti két szerepet elválaszthatatlanul összeköti
- **FIGYELEM** ami a `return` után van, az már nem kerül végrehajtásra, viszont egy függvényben több helyen is szerepelhet `return` utasítás
- az előre definiált `exit()` utasítás ezzel ellentétben azonnal kilép a program végrehajtásából

érdekesség

- a `main()` is egy függvény
- egy egész számmal kell visszatérjen, amelynek hibajelző szerepe van
  - egyelőre ezt mindig `0` értékre állítjuk, ami azt jelenti, hogy nincs hiba
- a `main()` paraméterei: egyelőre hagyjuk nyitva a kérdést, hogy mik lehetnek az ő paraméterei

További részletek
bibliográfia

- K. N. King C programming – A modern approach, 2nd edition, W. W. Norton & Co., 2008
  - 9. és 10. fejezet
- Deitel & Deitel C How to Program, 6th edition, Pearson, 2009
  - 5. fejezet

kvíz 5. kérdés

4170 6972