1. feladat: Milyen hibákat talál az alábbi megoldásokban?

```
m = malloc(80);
m = NULL;
```

megoldás: 80 bájt lefoglalása, majd a pointer nullázása. Probléma: - a lefoglalt 80 byte nem szabadul fel. `m = NULL` sor természetesen nullázza az `m` mutatót, de ettől még a 80 bájt lefoglalva marad.

Egy pointerhez memória lefoglalása majd amikor már nincs rá szükség, akkor a felszabadítása az alábbi váz alapján valósítható meg.

```
m = malloc(80);
// számolás.. egyéb kódok
free(m);
```

Nézzük a következő hibás példát. Mi lehet a hiba?

```
free(n);
n = 5;
```

megoldás: felszabadított memóriaterületre akarunk írni. Ha már meghívtuk a free() memória felszabadító függvényt, akkor nem lehet tovább használni a pointert. (ameddig a malloc()-al újra foglalunk neki memóriát.)

A következő példában nem használunk dinamikus memóriafoglalást, ami elvileg még jó is lehet, de mégsem szabályos. Miért?

```
char *p;
*p = 'a';
```

megoldás: nem inicializált pointert akarunk használni. Azaz a `char *p`, nem ad értéket a `p` pointernek, csak helyet foglal neki, és azon a helyen 0 van, vagy valami memóriaszemét. Ha a következő sorban a `*p = 'a'` -val a p által mutatott címre bele szeretnénk másolni a 'a'-t akkor az nem lesz lehetséges, mert a 0-ás címre nem írhatunk.

Feladat:: foglaljunk le dinamikusan egy 10 elemű int vektort.

```
#include <stdlib.h>

int main() {
    int *v = (int *)malloc(1000 * sizeof(int));
    free(v);
    return 0;
}
```

A kód igazából nem csinál túl sok használhatót, csak demonstrálja a malloc() használatát. A paramétere azt jelenti, hogy hány bájtot szeretnénk lefoglalni. Önmagában 1000-el meghívva csak 250 darab int értéknek foglal helyet, mert minden `int` 4 byte-ot foglal. Mivel ezt nem akarjuk fejből tudni ezért használjuk a beépített sizeof() operátort, ami sizeof(int) esetén 4-et ad vissza. Összefoglalva az 1000 int lefoglalásához 4000 byte-ok kell lefoglalni.

Feladat:: Az alábbi 3 függvényből melyik helyes, illetve hibás?

```
int* f(void)
{
  int x = 10;
  return (&x);
}
int* g(void)
{
  int * p;
  *p = 10;
  return p;
}
int* h(void)
{
  int *p;
  p = (int *) malloc (sizeof(int));
  return p;
}
```

megoldás: csak a h() függvény helyes. A f() és g() hibásak, mert ezekben az esetekben a memóriacím a veremben jön létre és a visszaadott érték helytelen memóriacímre fog mutatni. Az f() és g() esetén lokális változókat hozunk létre, és ezekre mutató memóriacímet adunk vissza. A függvény használatakor, a használat után ezek a memóriacímek nem használhatók tovább. A h() esetén is a p változó lokális lesz, az is megszűnik a függvény használata után, de a malloc() által

visszaadott memória cím (min számérték továbbra is használható) a `return p` itt is nem a p-t, hanem a p-ben tárolt címet adja vissza, ami a h() esetén továbbra is használható.

A példa azt mutatja, hogy a malloc() által létrehozott cím és a mögötte lefoglalt memória globális, nem számít hol hívtuk meg.

Feladat:: Mi lesz az alábbi program kimenete?

```
#include<stdio.h>

void f(int *a)
{
    a = (int*)malloc(sizeof(int));
}

int main()
{
    int *p;
    f(p);
    *p = 10;
    printf("%d",*p);
}
```

megoldás: nem ír ki semmit, hanem lefagy.

Feladat:: Hogyan lehetne kijavítani az előző feladatban szereplő programot?

megoldás: az előző program azért nem működik, mert a pointer nincs inicializálva (csak egy másolat), ezért a 0-s memóriacímet adja át az f() függvénynek, így a malloc lefoglalja a 4 byte-ot, de hibás helyre írja vissza. A javítás során vegyük figyelembe, hogy egy pointerre mutató pointer már ténylegesen a p változó címét adja vissza, amibe már a f() malloc()-ja már be tudja írni a lefoglalt memória címét.

```
#include<stdio.h>

void f(int **a)
{
    *a = (int*)malloc(sizeof(int));
}

int main()
{
    int *p;
```

```
    f(&p);
    *p = 10;
    printf("%d",*p);
}
```

Feladat:: Mi a probléma következő programmal?

```
#include<stdio.h>

int main()
{
    float *p = (float *)malloc(sizeof(float));
    p = NULL;

    free(p);
}
```

megoldás: a p NULL-ázása után, a free() nem tudja, hogy hol van az a memóriacím, amit fel kell szabadítani. A free() nem a p változót, hanem az abban tárolt címet szabadítja fel.

From:
https://edu.iit.uni-miskolc.hu/ - **Institute of Information Science - University of Miskolc**

Permanent link:
https://edu.iit.uni-miskolc.hu/tanszek:oktatas:muszaki_informatika:memoria_kezeles_c-ben?rev=1741806379

Last update: **2025/03/12 19:06**