Mérnöki programozás 6

Szerkesztette: dr. Vass Péter Tamás

Mutatók

Mutató (pointer)

Egy változó definíciója tulajdonképpen a változó típusának megfelelő méretű memóriaterület lefoglalását eredményezi a program futása során.
A változó aktuális értékét ezen a területen tárolja a program kódolt formában.
Amikor egy változó azonosítóját használjuk egy értékadásra szolgáló kifejezésben (akár az értékadás operátorának baloldalán, akár jobboldalán), akkor tulajdonképpen a változóhoz tartozó memóriaterület tartalmához férünk hozzá, ill. változtatjuk meg.
A változó mindig ugyanarra a memóriaterületre hivatkozik (ahhoz van hozzárendelve), tehát memóriafoglalás szempontjából statikusnak minősül.

Mutatók

Mutató (pointer)

Gyakran felmerülő probléma azonban, hogy előre (a programkód elkészítésekor) nem ismerjük pontosan milyen mennyiségű adat számára kell, mekkora méretű memóriaterületet lefoglalni, hanem az csak a program futása során dől el (pl. adatok beolvasása fájlokból, és a fájl mérete nem ismert előre).

Ilyenkor dinamikusan (a program futása során) kell memóriaterületet lefoglalni, ahová ideiglenesen tároljuk el az adatokat, majd ha már nincs szükség rájuk, a memória területet felszabadíthatjuk.

A memóriát úgy kell elképzelnünk, mint egy nagyon sok fiókos szekrényt, melynek minden fiókja 1 bájt kódolt információt képes tárolni.

Mutatók

Mutató (pointer)

A fiókok helyének egyértelmű azonosíthatósága érdekében minden egyes fiók rendelkezik egy címmel. A címek egy címtartományból vannak kölcsönösen egyértelmű hozzárendelés alapján kiosztva a fiókokhoz (a memória bájtjaihoz).

A memóriaterület egyes bájtjainak eléréséhez tehát ismernünk kell a címüket. A dinamikusan lefoglalásra kerülő memória-területen tárolt adatok elérésére nem alkalmazhatunk közönséges változókat, hiszen ezek számát, azonosítóját és típusát (ami a lefoglalásra kerülő memóriaterület méretét meghatározza) előre kell deklarálni.

Emiatt szükség van egy speciális változóra, ami nem közvetlenül egy memóriaterületen tárolt adat értékére hivatkozik, hanem egy memória címre, azaz a lefoglalt memórián belül egy meghatározott bájtra (fiókra).

Mutatók

Mutató (pointer)

Tehát a mutató (pointer) egy olyan változó, amelynek segítségével memóriacímek tárolása és elérése valósítható meg.

Egy mutató definíciójának általános formája:

```
típus *azonosítónév;
```

pl.

```
int *ptr;   /*egy mutató amivel egy int típusú változó címét lehet tárolni és elérni*/
```

A * szimbólum jelzi a deklarációban, hogy nem egy közönséges változóról, hanem egy meghatározott típusú változó címének tárolására alkalmas mutatóról van szó.

A *típus* * (pl. int *) előírás tehát szétválaszthatatlan a deklarációban.

A definiált mutató azonban még nem "mutat" sehova, azaz nem tárol a program szempontjából jelentőséggel bíró memóriacímet!

Mutatók

Mutató (pointer)

A mutatónak értéket kell adni ahhoz, hogy egy érvényes memóriacímre mutasson. Példa:

```
int a;
int *ptr;
a = 5;
ptr = &a;
```

Az & operátor adja meg a mutató számára az utána következő változó memóriacímét. (ld. *scanf* függvény bemenő paraméterei).

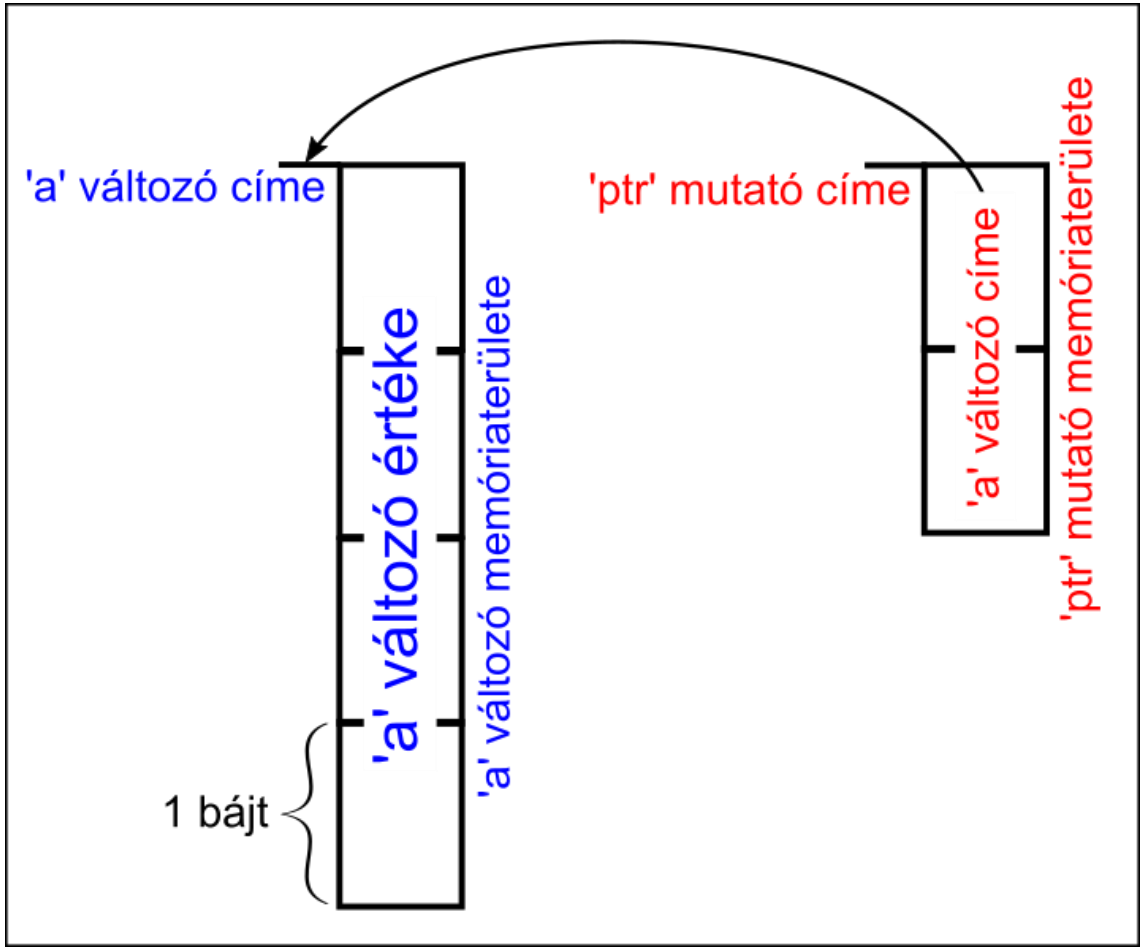
Ennek hatására a mutató értéke egy változóhoz tartozó (érvényes) cím lesz.

Ilyenkor azt mondjuk, hogy a mutató az adott címre mutat.

Mutatók

Mutató (pointer)

```
int a;
int *ptr;
a = 5;
ptr = &a;
```



Mutatók

Mutató (pointer)

Miután egy mutatóhoz hozzárendeltük egy változó memóriacímét, a mutató segítségével elérhetjük a változó értékét is. Példa:

```
#include <stdio.h>
main(){
   int a=5;
   int *ptr;
   ptr = &a;   /*memóriacím hozzárendelés*/
   printf("a = %d\n", a); /*az a nevű változó értékének
közvetlen elérése a változó azonosítóval*/
   printf("a = %d\n", *ptr); /*az a változó értékének
közvetett (indirekt) elérése a mutató segítségével*/
   printf("ptr = %p", ptr);  /*a mutatóban tárolt memóriacím
(az a változó memóriacíme) megjelenítése*/
}
```

Mutatók

Mutató (pointer)

A * operátor neve indirekció operátor.

A mutató azonosítója előtt alkalmazva, a mutatóban tárolt memóriacímen kezdődő memóriaterületen tárolt adatot érhetjük el. Alkalmazhatjuk kifejezésekben, értékadó utasítás bal és jobboldalán egyaránt.

Segítségével megváltoztathatjuk a tárolt adat értékét is. Példa:

```
int a;
int *ptr;
a = 5;
ptr = &a;
a = *ptr + 2;
```

A mutató definíciójában alkalmazott * nem operátor, hanem a mutató típust jelző szimbólum.

Mutatók

Mutató (pointer)

Példa változó értékének megváltoztatására mutatón keresztül:

```
#include <stdio.h>
main(){
	int a=5;
	int *ptr;
	ptr = &a;
	printf("a = %d\n", a);
	printf("a = %d\n", *ptr);
	*ptr = *ptr % 2;
	printf("a = %d\n", a);
	printf("a = %d\n", *ptr);
}
```

Mutatók

Mutató (pointer)

Egy mutató segítségével több változót is elérhetünk a programban, ha a mutatóban tárolt memóriacím értékét megváltoztatjuk. Példa.

```
#include <stdio.h>
main(){
        double x=0.5, y;
        double *ptr;
        y = 3 * x;
        ptr = &x;
        printf("ptr = %p\n", ptr);
        *ptr = x + y;
        printf("x = %f\n", x);
        printf("y = %f\n", y);
        printf("x = %f\n", *ptr);
        ptr = &y;
        printf("ptr = %p\n", ptr);
        *ptr = x + y;
        printf("x = %f\n", x);
        printf("y = %f\n", y);
        printf("y = %f\n", *ptr);
}
```

Dinamikus memóriahasználat

Dinamikus memóriahasználat mutató segítségével

Dinamikus memóriahasználatra akkor van szükség, amikor előre nem tudjuk pontosan, hogy mennyi adattal kell műveleteket végeznünk, mert az adatok mennyisége a program futása során válik ismertté.

Ilyenkor nem tudunk megfelelő számú változót definiálni előre a forráskódban.

A dinamikus memóriahasználat lényege, hogy a program futása során foglaljuk le a kívánt méretű memóriablokkot az adatok tárolásához.

A memóriablokk foglalása szabványos könyvtári függvénnyel történik (malloc ← memory allocation), melynek deklarációját az stdlib.h fejlécfájl (header file) tartalmazza.

A legfoglalt memóriablokk kezdőcímét egy mutatóban tároljuk el.

Dinamikus memóriahasználat

Dinamikus memóriahasználat mutató segítségével

A dinamikus memóriafoglalás sikerességét mindig ellenőrizni kell a programban, mert ha nem sikerül, akkor a memóriablokkra vonatkozó további műveletek nem végezhetők el.

A memóriablokk kezdőcímét tároló mutató segítségével

- adatokkal tölthetjük fel a memóriablokkot,
- elérhetjük a tárolt adatokat (azaz műveleteket végezhetünk velük,)
- és az értéküket meg is változtathatjuk.

Ha már nincs szükség a lefoglalt memóriablokkra, akkor azt fel kell szabadítanunk. Ezt a free nevű függvény segítségével tehetjük meg.

A következő példaprogramban a felhasználó döntheti el, hogy hány bemeneti adatot kíván megadni a program számára.

Ennek ismeretében a program már le tudja foglalni az adatok tárolásához szükséges méretű memóriablokkot.

A bevitt számadatok ezen a területen tárolódnak, majd a program kiszámítja, és meg is jeleníti a számok négyzeteit.

Végül a már szükségtelen memóriablokkot felszabadítja.

Dinamikus memóriahasználat

```c
#include <stdio.h>
#include <stdlib.h>

main(){
    int ndata, i;
    float *p;

    printf("Adja meg a beolvasni kivant adatok szamat! (max. 10) ");
    scanf("%d", &ndata);
    if (ndata<1 || ndata>10){
        printf("Hibas adatszam!");
        return;
    }
    p = (float *)malloc(ndata*sizeof(float));
    if (p==NULL){
        printf("A memoria lefoglalasa nem sikerult!");
        return;
    }
    for (i=0; i<ndata; i++){
        printf("Adja meg az %d. adatot: ", i+1);
        scanf("%f", p+i);
    }
    printf("\nA beolvasott adatok negyzetei\n");
    for (i=0; i<ndata; i++)
        printf("Az %d. adat negyzete: %f\n", i+1, *(p+i)**(p+i));
    free(p);
}
```

Dinamikus memóriahasználat

Dinamikus memóriahasználat mutató segítségével

```
#include <stdlib.h>
```

A memóriablokk foglalására szolgáló malloc függvény miatt kell beépíteni a forráskódba a fejlécfájlt.

```
float *p;
```

A beolvasni kívánt adatok nem feltétlenül egész számok ezért float típusú adatok tárolására szolgáló memóriaterületek címeinek tárolására alkalmas mutatót kell definiálnunk.

Kezdetben a mutató egyetlen érvényes címet sem tartalmaz, azaz nem mutat semmilyen memóriacímre. Ilyenkor az értéke a NULL szimbolikus konstans értékével egyenlő. A NULL definícióját is az stdlib.h tartalmazza.

```
p = (float *)malloc(ndata*sizeof(float));
```

A malloc függvény hívása a memóriablokk lefoglalása érdekében. A malloc függvény bemeneti paraméterében meg kell adni a lefoglalni kívánt memóriablokk méretét bájtokban.

Dinamikus memóriahasználat

Dinamikus memóriahasználat mutató segítségével

A lefoglalni kívánt memóriablokk mérete az alábbi kifejezéssel számítható ki

```
ndata*sizeof(float)
```

amiben a `sizeof` operátor segítségével határozzuk meg egy float típusú adat tárolásához szükséges memóriaterületet bájtokban.

A beolvasni kívánt adatok számát a felhasználó adja meg, és az értéke az `ndata` változóban tárolódik. A beolvasásra kerülő adatok tárolásához szükséges memóriaterületet tehát *az adatszám * az egy adat tárolásához szükséges terület* összefüggése alapján kapjuk meg.

A malloc függvény miután lefoglalta a megadott méretű memóriablokkot, annak elejére mutató memóriacím értékével tér vissza a hívó függvényhez.

Mivel ezt a memóriacímet jelen esetben egy float típusú adatok memóriacímeinek tárolására alkalmas mutatónak kívánjuk átadni, előtte a malloc visszatérési értékét egy ún. explicit típuskonverziónak kell alávetni.

Dinamikus memóriahasználat

Dinamikus memóriahasználat mutató segítségével

Az explicit típuskonverzióval különböző típusú adatokat alakíthatunk át egymásba. Természetesen ez időnként információvesztéssel jár (pl. float típusú adat átalakítása int típusú adattá a törtrész elvesztését okozza).

Az explicit típuskonverzió kerekzárójelpárba foglalva tartalmazza a megcélzott típus megadását (amivé át kell alakítani az adatot).

A zárójelpár megelőzi azt a változó vagy függvény azonosítót, amelynek értékét át kell alakítani.

Mivel a malloc függvény általánosan használható, mindenféle típusú adatok tárolására szolgáló memóriablokkok lefoglalására, ezért a programban kell a visszatérési értékét olyan típusúra alakítani, amilyen típusú mutató fogadja a memóriablokk kezdőcímét.

Ezért kell alkalmazni a függvény azonosítója előtt a `(float *)` explicit típuskonverziót.

Ha a memóriafoglalás sikertelen akkor a mutató értéke NULL marad, azaz nem mutat sehova.

Dinamikus memóriahasználat

Dinamikus memóriahasználat mutató segítségével

Ezt használhatjuk fel a memóriafoglalás sikerességének ellenőrzésekor, az alábbi kódrészletben

```
if (p==NULL) return;
```

Ha nem sikerült lefoglalni az igényelt memóriaterületet, akkor a program leáll.

Ha sikerült lefoglalni a kívánt méretű memóriablokkot, akkor megkezdődhet az adatok beolvastatása. Az első adat a memóriablokk kezdőcímével azonosított helyre kerül, melynek címét a *p* mutató tárolja. A második adat egy float típusú adat tárolásához szükséges memóriaterülettel távolabbi címmel kezdődő területre kerül, melyet a *p+1* kifejezés értéke ad meg. Ha ugyanis egy mutatóhoz hozzáadunk egy egész számot, akkor az eredmény a mutatóban tárolt címtől távolabbi cím értéke lesz. A távolság mértékét bájtokban úgy kapjuk meg, hogy az egész számot megszorozzuk a mutatóhoz tartozó adattípus tárolásához szükséges terület bájtokban kifejezett hosszával.

Dinamikus memóriahasználat

Dinamikus memóriahasználat mutató segítségével

A memóriablokkon belül visszafelé is mozoghatunk a címekkel, ha a mutató aktuális értékéből egy egész számot kivonunk. Természetesen a kezdőcímtől nem mozoghatunk "hátrább" a lefoglalt memóriablokkon belül.

A beolvasott adatok megfelelő helyekre kerülését az alábbi módon biztosítjuk

```
scanf("%f", p+i);
```

az *i* értéke azért kezdődik 0-val a for ciklusban, hogy az első adat pontosan a *p*-ben tárolt címre kerüljön.

Figyeljük meg, hogy itt nem kell az & (címe) operátort használnunk a scanf függvény második bemeneti paraméterénél. Ugyanis most nem egy változót, hanem a mutató segítségével előírt címet adunk meg közvetlenül.

Végül kiszámítjuk a beolvasott adatok négyzeteit és kiíratjuk az eredményeket egymás alá.

Az egyes számadatokat a mutató segítségével *indirekt* módon érjük el az indirekció (*) operátorát alkalmazva

Dinamikus memóriahasználat

Dinamikus memóriahasználat mutató segítségével

Az egyes számadatok négyzeteinek számításához alkalmazott kifejezés:

```
*(p+i)**(p+i)
```

A zárójelen belüli kifejezések segítségével adjuk meg a soron következő számadatot tároló memóriaterület kezdőcímét.

Ha a mutatóra, ill. a mutatót tartalmazó kifejezésre alkalmazzuk az indirekció operátorát (*), akkor magát a tárolt adatok érhetjük el.

A fenti kifejezésben természetesen a második * a szorzás műveletét jelenti (itt feketével jelölt). Az első és a harmadik * indirekt elérést (nem változó azonosítón keresztüli elérés) biztosít az adathoz a mutatóban tárolt címen keresztül.