Óbudai Egyetem
Kandó Kálmán Villamosmérnöki Kar

C programozási nyelv
Mutatók, tömbök, sztringek

Dr. Schuster György

2017. december 29.

Pointerek (mutatók)

A pointerek olyan speciális adattípusok, amelyek nem értéket tárolnak, hanem objektumoknak (változónak, függvénynek) a címét tartalmazzák.
Egy egész típusú változóra mutató pointer deklarációja:

```
int *ip;
```

Pointerek (mutatók)

A pointerek olyan speciális adattípusok, amelyek nem értéket tárolnak, hanem objektumoknak (változónak, függvénynek) a címét tartalmazzák.
Egy egész típusú változóra mutató pointer deklarációja:

```
int *ip;
```

Itt a $*$ karakter jelzi, hogy pointerről van szó.

Pointerek (mutatók)

A pointerek olyan speciális adattípusok, amelyek nem értéket tárolnak, hanem objektumoknak (változónak, függvénynek) a címét tartalmazzák.
Egy egész típusú változóra mutató pointer deklarációja:

```
int *ip;
```

Itt a $*$ karakter jelzi, hogy pointerről van szó.
FIGYELEM!!! a mutatókat használatuk előtt inicializálni kell!!!

Pointerek (mutatók)

A pointerek olyan speciális adattípusok, amelyek nem értéket tárolnak, hanem objektumoknak (változónak, függvénynek) a címét tartalmazzák.
Egy egész típusú változóra mutató pointer deklarációja:

```
int *ip;
```

Itt a $*$ karakter jelzi, hogy pointerről van szó.
FIGYELEM!!! a mutatókat használatuk előtt inicializálni kell!!!
Hogyan kap értéket a pointer?

Pointerek (mutatók)

A pointerek olyan speciális adattípusok, amelyek nem értéket tárolnak, hanem objektumoknak (változónak, függvénynek) a címét tartalmazzák.

Egy egész típusú változóra mutató pointer deklarációja:

```
int *ip;
```

Itt a $*$ karakter jelzi, hogy pointerről van szó.

FIGYELEM!!! a mutatókat használatuk előtt inicializálni kell!!!

Hogyan kap értéket a pointer?

```
int *ip;
int i;
  :
ip=&i;
```

Pointerek (mutatók)

A pointerek olyan speciális adattípusok, amelyek nem értéket tárolnak, hanem objektumoknak (változónak, függvénynek) a címét tartalmazzák.

Egy egész típusú változóra mutató pointer deklarációja:

```
int *ip;
```

Itt a $*$ karakter jelzi, hogy pointerről van szó.

FIGYELEM!!! a mutatókat használatuk előtt inicializálni kell!!!

Hogyan kap értéket a pointer?

```
int *ip;
int i;
  :
ip=&i;
```

Itt a `&` operátor az úgynevezett címképző operátor. Ez unáris, ne tévesszük össze a bitenkénti és-sel.

Pointerek (mutatók)

A pointerek olyan speciális adattípusok, amelyek nem értéket tárolnak, hanem objektumoknak (változónak, függvénynek) a címét tartalmazzák.

Egy egész típusú változóra mutató pointer deklarációja:

```
int *ip;
```

Itt a $*$ karakter jelzi, hogy pointerről van szó.

FIGYELEM!!! a mutatókat használatuk előtt inicializálni kell!!!

Hogyan kap értéket a pointer?

```
int *ip;
int i;
   :
ip=&i;
```

Itt a `&` operátor az úgynevezett címképző operátor. Ez unáris, ne tévesszük össze a bitenkénti és-sel.

Mostmár az`ip` az `i` címét tartalmazza.

Indirekció

Az indirekt értékadás azt jelenti, hogy egy változónak mutatón keresztül adunk értéket.

Indirekció

Az indirekt értékadás azt jelenti, hogy egy változónak mutatón keresztül adunk értéket.Például:

```
int *ip;
int i=5,j;
   ⋮
ip=&i;
   ⋮
j=*ip;
```

Indirekció

Az indirekt értékadás azt jelenti, hogy egy változónak mutatón keresztül adunk értéket.Például:

```
int *ip;
int i=5,j;
   ⋮
ip=&i;
   ⋮
j=*ip;
```

Szó szerint!!

`j` változó értéke legyen egyenlő az `ip` által címzett változó értékével.

Indirekció

Az indirekt értékadás azt jelenti, hogy egy változónak mutatón keresztül adunk értéket.Például:

```
int *ip;
int i=5,j;
   ⋮
ip=&i;
   ⋮
j=*ip;
```

Szó szerint!!

`j` változó értéke legyen egyenlő az `ip` által címzett változó értékével.

Mondom

SZÓ SZERINT!!!

Indirekció

Az indirekt értékadás azt jelenti, hogy egy változónak mutatón keresztül adunk értéket.Például:

```
int *ip;
int i=5,j;
   ⋮
ip=&i;
   ⋮
j=*ip;
```

Szó szerint!!

`j` változó értéke legyen egyenlő az `ip` által címzett változó értékével.

Mondom

SZÓ SZERINT!!!

Ezt nevezzük indirekt értékadásnak.

Indirekció

Az indirekt értékadás azt jelenti, hogy egy változónak mutatón keresztül adunk értéket.Például:

```
int *ip;
int i=5,j;
   ⋮
ip=&i;
   ⋮
j=*ip;
```

Szó szerint!!
`j` változó értéke legyen egyenlő az `ip` által címzett változó értékével.
Mondom
SZÓ SZERINT!!!
Ezt nevezzük indirekt értékadásnak.

Ha elfelejtjük `ip` inicializálni, a program rendszertől függő hibákat generál. Kultúráltabb rendszereknél futás közben hibaüzenetet kapunk. Sokkal veszélyesebb, ha nem kapunk visszajelzést.
Még visszatérünk a mutatókhoz.

Tömbök

A tömbök a legegyszerűbb összetett adatszerkezetek.

Tömbök

A tömbök a legegyszerűbb összetett adatszerkezetek. Jellemzőik:

- szigorúan azonos típusú adatokat tartalmaznak,

Tömbök

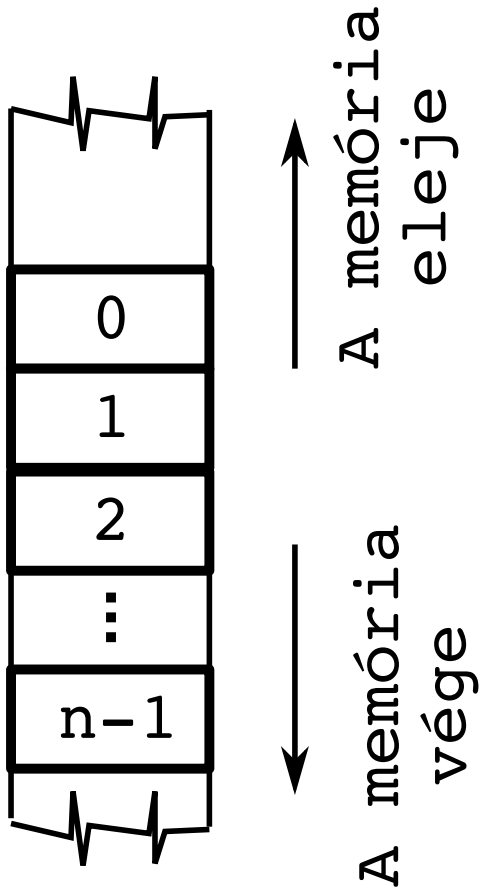
A tömbök a legegyszerűbb összetett adatszerkezetek. Jellemzőik:

- szigorúan azonos típusú adatokat tartalmaznak,
- az adatok egymás után helyezkednek el a tárban.

Tömbök

A tömbök a legegyszerűbb összetett adatszerkezetek. Jellemzőik:

- szigorúan azonos típusú adatokat tartalmaznak,
- az adatok egymás után helyezkednek el a tárban.



Deklaráció és hozzáférés

Egydimenziós tömb deklarációja, példa:

```
int t[10];
```

Deklaráció és hozzáférés

Egydimenziós tömb deklarációja, példa:

```
int t[10];
```

Egy tíz elemű egészeket tartalmazó tömb deklarációja.

A kérdéses tömb elemeinek indexei 0, 1, . . . , 9 tartományban vannak.

Deklaráció és hozzáférés

Egydimenziós tömb deklarációja, példa:

```
int t[10];
```

Egy tíz elemű egészeket tartalmazó tömb deklarációja.

A kérdéses tömb elemeinek indexei 0, 1, . . . , 9 tartományban vannak.

A tömb egy elemének az elérése, példa:

```
t[5]=23;
```

Deklaráció és hozzáférés

Egydimenziós tömb deklarációja, példa:

```
int t[10];
```

Egy tíz elemű egészeket tartalmazó tömb deklarációja.

A kérdéses tömb elemeinek indexei 0, 1, . . . , 9 tartományban vannak.

A tömb egy elemének az elérése, példa:

```
t[5]=23;
```

Tehát az elemet indexeléssel érhetjük el. Az index csak pozitív egész szám lehet. Tehát konstans szám, vagy egész jellegű változó.

Deklaráció és hozzáférés

Egydimenziós tömb deklarációja, példa:

```
int t[10];
```

Egy tíz elemű egészeket tartalmazó tömb deklarációja.

A kérdéses tömb elemeinek indexei 0, 1, . . . , 9 tartományban vannak.

A tömb egy elemének az elérése, példa:

```
t[5]=23;
```

Tehát az elemet indexeléssel érhetjük el. Az index csak pozitív egész szám lehet. Tehát konstans szám, vagy egész jellegű változó.

```
t[i]=23;
```

ahol `i` egy pl. `int`.

A tömb kérédéses elemét úgy használhatjuk, mint egy azonos típusú közönséges változót.

Deklaráció és hozzáférés

Egydimenziós tömb deklarációja, példa:

```
int t[10];
```

Egy tíz elemű egészeket tartalmazó tömb deklarációja.

A kérdéses tömb elemeinek indexei 0, 1, . . . , 9 tartományban vannak.

A tömb egy elemének az elérése, példa:

```
t[5]=23;
```

Tehát az elemet indexeléssel érhetjük el. Az index csak pozitív egész szám lehet. Tehát konstans szám, vagy egész jellegű változó.

```
t[i]=23;
```

ahol `i` egy pl. `int`.

A tömb kérédéses elemét úgy használhatjuk, mint egy azonos típusú közönséges változót.

Probléma a túlindexelés. Változóval történő indexelés esetén futásközben az index kívül van az idexelési tartományon.

Pl. `i` értéke 11.

A program rendszertől függő hibákat generál.

Többdimenziós tömbök

Kétdimenziós egész tömb deklarálása, pl.:

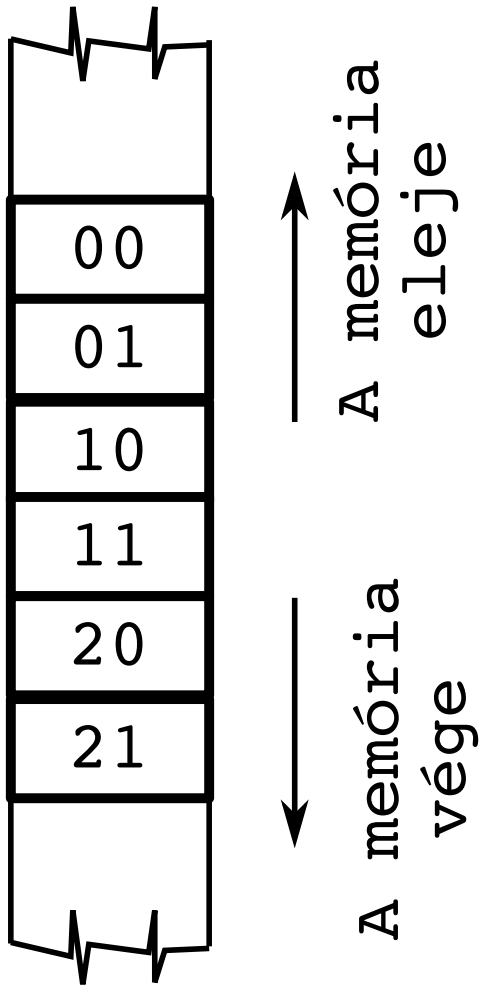
```
int t[3][2];
```

Többdimenziós tömbök

Kétdimenziós egész tömb deklarálása, pl.:

```
int t[3][2];
```

A leghátsó index változik a leggyorsabban.
Vagyis.

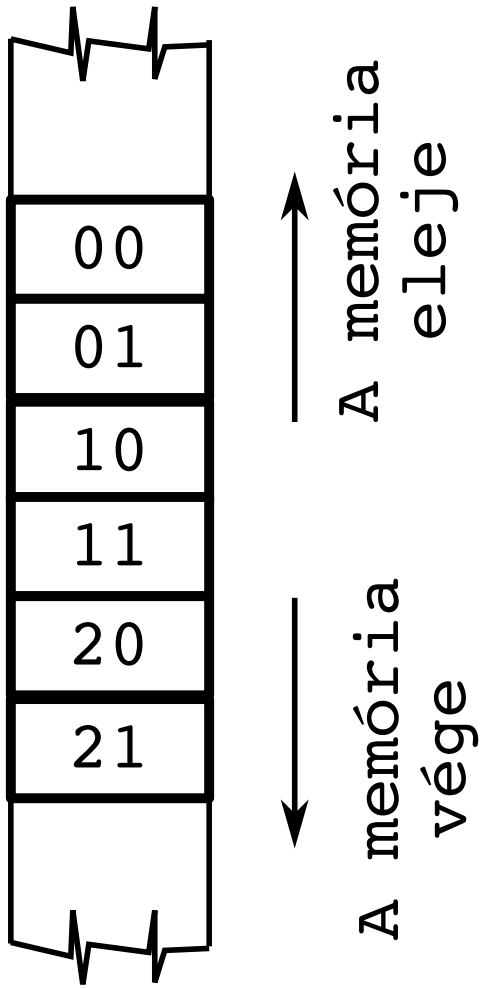


Többdimenziós tömbök

Kétdimenziós egész tömb deklarálása, pl.:

```
int t[3][2];
```

A leghátsó index változik a leggyorsabban.
Vagyis.



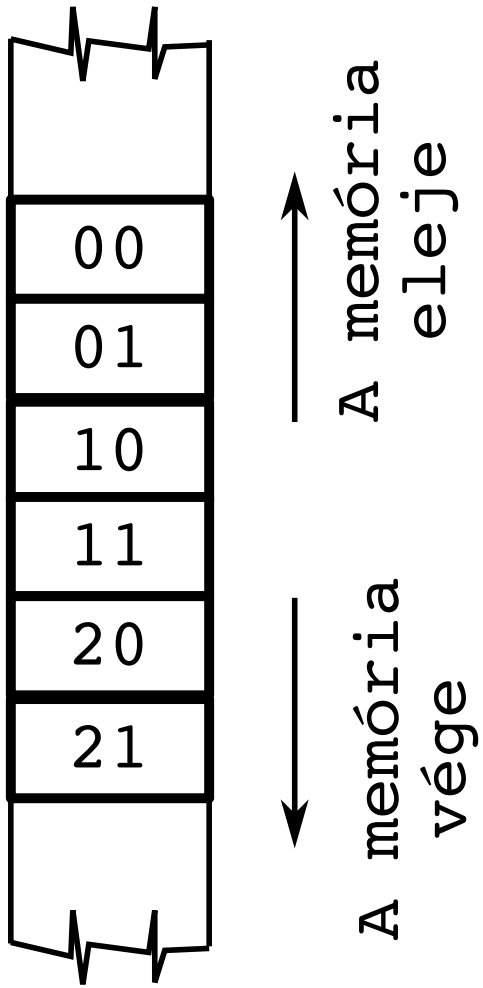
Tetszőleges számú dimenzió lehetséges. De a dimenziók számának növekedésével exponenciálisan növekszik a memória foglalás.

Többdimenziós tömbök

Kétdimenziós egész tömb deklarálása, pl.:

```
int t[3][2];
```

A leghátsó index változik a leggyorsabban.
Vagyis.



Tetszőleges számú dimenzió lehetséges. De a dimenziók számának növekedésével exponenciálisan növekszik a memória foglalás.
Az elem elérése itt is indexeléssel történik. Tehát:

Tömbök inicializálása deklarációnál

Megadhatunk kezdőértéket a tömbnek deklarációnál.

```
int t[10]={6,8,-2,6,123,-8,3,4,2,1};
```

Ezt szépen sorban berakja a tömb elemeibe, de csak deklarációnál!

Tömbök inicializálása deklarációnál

Megadhatunk kezdőértéket a tömbnek deklarációnál.

```
int t[10]={6,8,-2,6,123,-8,3,4,2,1};
```

Ezt szépen sorban berakja a tömb elemeibe, de csak deklarációnál!
Ha inicializálunk, nem kell megadni a tömb méretét.

```
int t[]={6,8,-2,6,123,-8,3,4,2,1};
```

A fordító kitalálja mekkora legyen a tömb.

Tömbök inicializálása deklarációnál

Megadhatunk kezdőértéket a tömbnek deklarációnál.

```
int t[10]={6,8,-2,6,123,-8,3,4,2,1};
```

Ezt szépen sorban berakja a tömb elemeibe, de csak deklarációnál!
Ha inicializálunk, nem kell megadni a tömb méretét.

```
int t[]={6,8,-2,6,123,-8,3,4,2,1};
```

A fordító kitalálja mekkora legyen a tömb.
Ha kevesebb elemet adunk meg, mint amekkora a méret a fennmaradó helyeket 0-val tölti ki.

Tömbök inicializálása deklarációnál

Megadhatunk kezdőértéket a tömbnek deklarációnál.

```
int t[10]={6,8,-2,6,123,-8,3,4,2,1};
```

Ezt szépen sorban berakja a tömb elemeibe, de csak deklarációnál!
Ha inicializálunk, nem kell megadni a tömb méretét.

```
int t[]={6,8,-2,6,123,-8,3,4,2,1};
```

A fordító kitalálja mekkora legyen a tömb.
Ha kevesebb elemet adunk meg, mint amekkora a méret a fennmaradó helyeket 0-val tölti ki.
Többdimenziós esetben.

Szépen:

```
int t[3][2]={ {1,2},
              {3,4},
              {5,6}};
```

Csúnyán:

```
int t[3][2]={ 1,2,3,4,5,6};
```

Sokkal nagyobb figyelmet kíván, mint a baloldali.

Tömb $\leftrightarrow$ pointer

Töltsük be a tömb kezdőcímét egy mutatóba!

Tömb $\leftrightarrow$ pointer

Töltsük be a tömb kezdőcímét egy mutatóba!

```
int t[10];
int *p;
   :
p=t;
```

Tehát a tömb kezdőcíme a tömb neve.

Tömb $\leftrightarrow$ pointer

Töltsük be a tömb kezdőcímét egy mutatóba!

```
int t[10];
int *p;
   :
p=t;
```

Tehát a tömb kezdőcíme a tömb neve.
Vagyis:

`p=t;` azonos `p=&t[0];`

Tömb $\leftrightarrow$ pointer

Töltsük be a tömb kezdőcímét egy mutatóba!

```
int t[10];
int *p;
   :
p=t;
```

Tehát a tömb kezdőcíme a tömb neve.
Vagyis:

`p=t;` azonos `p=&t[0];`

Ha tehát ez igaz, akkor a kezdőcím átadása után, használhatjuk a pointert, mint tömböt, pl.:

```
p[5]=13;
```

Tömb $\leftrightarrow$ pointer

Töltsük be a tömb kezdőcímét egy mutatóba!

```
int t[10];
int *p;
   :
p=t;
```

Tehát a tömb kezdőcíme a tömb neve.
Vagyis:

`p=t;` azonos `p=&t[0];`

Ha tehát ez igaz, akkor a kezdőcím átadása után, használhatjuk a pointert, mint tömböt, pl.:

```
p[5]=13;
```

Ha nem a tömb kezdőcímét akarjuk megadni, akkor csak a címképző operátorral dolgozhatunk.

```
p=&t[5];
```

Tömb $\leftrightarrow$ pointer

Töltsük be a tömb kezdőcímét egy mutatóba!

```
int t[10];
int *p;
   :
p=t;
```

Tehát a tömb kezdőcíme a tömb neve.
Vagyis:

`p=t;` azonos `p=&t[0];`

Ha tehát ez igaz, akkor a kezdőcím átadása után, használhatjuk a pointert, mint tömböt, pl.:

```
p[5]=13;
```

Ha nem a tömb kezdőcímét akarjuk megadni, akkor csak a címképző operátorral dolgozhatunk.

```
p=&t[5];
```

Akkor most vissza a pointerekhez!

Pointer aritmetika

Deklaráljunk egy `int` pointert és egy 10 elemű `int` tömböt!

```
int *p;
int t[10];
   ⋮
p=t;
```

Ezt már láttuk.

Pointer aritmetika

Deklaráljunk egy `int` pointert és egy 10 elemű `int` tömböt!

```
int *p;
int t[10];
   ⋮
```

`p=t;` Ezt már láttuk.

`p++;` Kérdés az, hogy hová mutat a `p` ezután.

Pointer aritmetika

Deklaráljunk egy `int` pointert és egy 10 elemű `int` tömböt!

```
int *p;
int t[10];
   ⋮
```

`p=t;` Ezt már láttuk.

`p++;` Kérdés az, hogy hová mutat a `p` ezután.

A következő bájtra, vagy `t[1]` -re?

Pointer aritmetika

Deklaráljunk egy `int` pointert és egy 10 elemű `int` tömböt!

```
int *p;
int t[10];
   ⋮
```

`p=t;` Ezt már láttuk.

`p++;` Kérdés az, hogy hová mutat a `p` ezután.

A következő bájtra, vagy `t[1]` -re?

A válasz: **`t[1]`** -re.
Tehát egy adott pointer inkrementálása annyival (bájttal) növeli az általa tartalmazott címet, amekkora a mérete (bájtban) annak a típusnak, amelyre mutat.

Pointer aritmetika

További műveletek:

- dekrementálás fordítottja az inkrementálásnak,

Pointer aritmetika

További műveletek:

- dekrementálás fordítottja az inkrementálásnak,
- egész hozzáadása pointerhez

```
int *p,*q;
int t[10];
q=t;
p=q+5;
```

p pointer értéke **q+5*sizeof(int)**.

Pointer aritmetika

További műveletek:

- dekrementálás fordítottja az inkrementálásnak,
- egész hozzáadása pointerhez

```
int *p,*q;
int t[10];
q=t;
p=q+5;
```

p pointer értéke **q+5*sizeof(int)**.

- egész kivonása pointerből

```
int *p,*q;
p=q-5;
```

p pointer értéke **q-5*sizeof(int)**.

Pointer aritmetika

További műveletek:

- dekrementálás fordítottja az inkrementálásnak,
- egész hozzáadása pointerhez

```
int *p,*q;
int t[10];
q=t;
p=q+5;
```

p pointer értéke **q+5*sizeof(int)**.

- egész kivonása pointerből

```
int *p,*q;
p=q-5;
```

p pointer értéke **q-5*sizeof(int)**.

- két pointer kivonása megadja a két pointer által mutatott cím távolságát az adott típusban.

Szóval akkor:

`t[5]=13;` ugyanaz, mint `*(t+5)=13;`

Példa a pointer használatára

Írjunk egy olyan függvényt, amely két `int` tipusú változót összead és egyben ki is von egymásból.
Egy függvény egyetlen paramétert adhat vissza a `return` használatával.

Példa a pointer használatára

Írjunk egy olyan függvényt, amely két `int` tipusú változót összead és egyben ki is von egymásból.
Egy függvény egyetlen paramétert adhat vissza a `return` használatával.
Az ötlet az, hogy azoknak a változóknak, amelyekben majd az eredményt kapjuk, a címét adjuk át a függvénynek.
Hogyanis:

```
1   void muv(int a,int b,int *r1,int *r2)
2      {
3      *r1=a+b;
4      *r2=a-b;
5      }
```

Példa a pointer használatára

Írjunk egy olyan függvényt, amely két `int` tipusú változót összead és egyben ki is von egymásból.
Egy függvény egyetlen paramétert adhat vissza a `return` használatával.
Az ötlet az, hogy azoknak a változóknak, amelyekben majd az eredményt kapjuk, a címét adjuk át a függvénynek.
Hogyanis:

```
1  void muv(int a,int b,int *r1,int *r2)
2     {
3     *r1=a+b;
4     *r2=a-b;
5     }
```

Vagyis a 3. sorban `r1` által címzett változóba kerüljön `a, b` összege.
Vagyis a 4. sorban `r2` által címzett változóba kerüljön `a, b` különbsége.

Példa a pointer használatára

A teljes program:

```
void muv(int,int,int *,int *);
int main(void)
   {
   int x,y,i,j;
   x=6,y=7;
   muv(x,y,&i,&j);
   printf("%i %i",i,j);
   return 0;
   }

void muv(int a,int b,int *r1,int *r2)
   {
   *r1=a+b;
   *r2=a-b;
   }
```

`void` pointer

A `void` pointer egy **típus nélküli** pointer.

Akkor szoktuk használni, ha egy memória címre van szükségünk, de nem ismerjük azt a konkrét típust, amelynek a példányára mutat.

`void` pointer

A `void` pointer egy **típus nélküli** pointer.

Akkor szoktuk használni, ha egy memória címre van szükségünk, de nem ismerjük azt a konkrét típust, amelynek a példányára mutat.

A legnevezetesebb `void` pointer a **`NULL`** pointer, amelyenk a definíciója az `stdio.h` header fájlban található meg.

```
#define NULL (void *)0
```

`void` pointer

A `void` pointer egy **típus nélküli** pointer.

Akkor szoktuk használni, ha egy memória címre van szükségünk, de nem ismerjük azt a konkrét típust, amelynek a példányára mutat.

A legnevezetesebb `void` pointer a **`NULL`** pointer, amelyenk a definíciója az `stdio.h` header fájlban található meg.

```
#define NULL (void *)0
```

Ez konkrétan a nullás memória címre mutat, amely a programozható eszközök nagy többségében nem használható változó tárolására.

`void` pointer

A `void` pointer egy **típus nélküli** pointer.

Akkor szoktuk használni, ha egy memória címre van szükségünk, de nem ismerjük azt a konkrét típust, amelynek a példányára mutat.

A legnevezetesebb `void` pointer a **`NULL`** pointer, amelyenk a definíciója az `stdio.h` header fájlban található meg.

```
#define NULL (void *)0
```

Ez konkrétan a nullás memória címre mutat, amely a programozható eszközök nagy többségében nem használható változó tárolására.
A `NULL` pointer hibajelzésre, szükségtelen paraméterek jelzésére használjuk.
Fájl és dinamikus memória kezelésnél fogjuk látni.

Sztringek

A sztring egy speciális karakter típusú tömb, amelyet egy 0 értékű karakter zár le.

Sztringek

A sztring egy speciális karakter típusú tömb, amelyet egy 0 értékű karakter zár le.

Teljesen mindegy, hogy mekkora a tömb a sztringet a 0 zárja le. Azért ne legyen hosszabb, mint a tömb!

A sztring inicializálása speciális módon történhet deklarációnál és csakis deklarációnál:

```
char nev[]="Schuster";
```

Ez hozzáfüzi a lezáró 0-t is. Tehát a `nev` tömb 9 elemű.

Sztringek

A sztring egy speciális karakter típusú tömb, amelyet egy 0 értékű karakter zár le.

Teljesen mindegy, hogy mekkora a tömb a sztringet a 0 zárja le. Azért ne legyen hosszabb, mint a tömb!

A sztring inicializálása speciális módon történhet deklarációnál és csakis deklarációnál:

```
char nev[]="Schuster";
```

Ez hozzáfüzi a lezáró 0-t is. Tehát a `nev` tömb 9 elemű.

Ez sokkal egyszerűbb, mintha a hagyományos inicializálást csinálnánk:

```
char nev[]={'S','c','h','u','s','t','e','r',0};
```

A lezáró elem 0 és nem '0'!!

Függvény pointerek

Van egy függvényünk, a deklarációja:

```
int fgv(int);
```

Deklaráljunk egy függvény pointert!

```
int (*fgvp)(int);
```

Függvény pointerek

Van egy függvényünk, a deklarációja:

```
int fgv(int);
```

Deklaráljunk egy függvény pointert!

```
int (*fgvp)(int);
```

Adjunk neki értéket!

```
fgvp=fgv;
```

Függvény pointerek

Van egy függvényünk, a deklarációja:

```
int fgv(int);
```

Deklaráljunk egy függvény pointert!

```
int (*fgvp)(int);
```

Adjunk neki értéket!

```
fgvp=fgv;
```

Hívjuk a függvényt a pointeren keresztül!

```
r=fgvp(5);
```

Kérdések

Mik azok a pointerek?

Kérdések

Mik azok a pointerek?

A pointerek olyan speciális adattpusok, amelyek nem értéket tárolnak, hanem objektumoknak (változónak, függvénynek) a cmét tartalmazzák.

Kérdések

Mik azok a pointerek?

A pointerek olyan speciális adattpusok, amelyek nem értéket tárolnak, hanem objektumoknak (változónak, függvénynek) a cmét tartalmazzák.

Hogyan deklarál egy pointert?

Kérdések

Mik azok a pointerek?

A pointerek olyan speciális adattpusok, amelyek nem értéket tárolnak, hanem objektumoknak (változónak, függvénynek) a cmét tartalmazzák.

Hogyan deklarál egy pointert?

A csillag segítségével `int *ip;`

Kérdések

Mik azok a pointerek?

A pointerek olyan speciális adattpusok, amelyek nem értéket tárolnak, hanem objektumoknak (változónak, függvénynek) a cmét tartalmazzák.

Hogyan deklarál egy pointert?

A csillag segítségével `int *ip;`

Hogyan kap értéket egy pointer változók esetén?

Kérdések

Mik azok a pointerek?

A pointerek olyan speciális adattpusok, amelyek nem értéket tárolnak, hanem objektumoknak (változónak, függvénynek) a cmét tartalmazzák.

Hogyan deklarál egy pointert?

A csillag segítségével `int *ip;`

Hogyan kap értéket egy pointer változók esetén?

A címképző operátor segítségével `&`. Az előző pointer esetén `ip=&i;`

Kérdések

Mik azok a pointerek?
A pointerek olyan speciális adattpusok, amelyek nem értéket tárolnak, hanem objektumoknak (változónak, függvénynek) a cmét tartalmazzák.
Hogyan deklarál egy pointert?
A csillag segítségével `int *ip;`
Hogyan kap értéket egy pointer változók esetén?
A címképző operátor segítségével `&`. Az előző pointer esetén `ip=&i;`
Mi a tipikus hiba a pointerek használatánál?

Kérdések

Mik azok a pointerek?
A pointerek olyan speciális adattpusok, amelyek nem értéket tárolnak, hanem objektumoknak (változónak, függvénynek) a cmét tartalmazzák.
Hogyan deklarál egy pointert?
A csillag segítségével `int *ip;`
Hogyan kap értéket egy pointer változók esetén?
A címképző operátor segítségével `&`. Az előző pointer esetén `ip=&i;`
Mi a tipikus hiba a pointerek használatánál?
Használat előtt nem adnak nekik értéket.

Kérdések

Mik azok a pointerek?
A pointerek olyan speciális adattpusok, amelyek nem értéket tárolnak, hanem objektumoknak (változónak, függvénynek) a cmét tartalmazzák.
Hogyan deklarál egy pointert?
A csillag segítségével `int *ip;`
Hogyan kap értéket egy pointer változók esetén?
A címképző operátor segítségével `&`. Az előző pointer esetén `ip=&i;`
Mi a tipikus hiba a pointerek használatánál?
Használat előtt nem adnak nekik értéket.
Mi az az indirekt értékadás?

Kérdések

Mik azok a pointerek?
A pointerek olyan speciális adattpusok, amelyek nem értéket tárolnak, hanem objektumoknak (változónak, függvénynek) a cmét tartalmazzák.
Hogyan deklarál egy pointert?
A csillag segítségével `int *ip;`
Hogyan kap értéket egy pointer változók esetén?
A címképző operátor segítségével `&`. Az előző pointer esetén `ip=&i;`
Mi a tipikus hiba a pointerek használatánál?
Használat előtt nem adnak nekik értéket.
Mi az az indirekt értékadás?
Amikor egy változó nem közvetlenül egy másik változótól kap értéket, hanem az adatforrás címén keresztül.

Kérdések

Mik azok a pointerek?
A pointerek olyan speciális adattpusok, amelyek nem értéket tárolnak, hanem objektumoknak (változónak, függvénynek) a cmét tartalmazzák.
Hogyan deklarál egy pointert?
A csillag segítségével `int *ip;`
Hogyan kap értéket egy pointer változók esetén?
A címképző operátor segítségével `&`. Az előző pointer esetén `ip=&i;`
Mi a tipikus hiba a pointerek használatánál?
Használat előtt nem adnak nekik értéket.
Mi az az indirekt értékadás?
Amikor egy változó nem közvetlenül egy másik változótól kap értéket, hanem az adatforrás címén keresztül.
Szavakban kifejezve mit jelent a következő kifejezés: `i=*ip;`

Kérdések

Mik azok a pointerek?
A pointerek olyan speciális adattpusok, amelyek nem értéket tárolnak, hanem objektumoknak (változónak, függvénynek) a cmét tartalmazzák.
Hogyan deklarál egy pointert?
A csillag segítségével `int *ip;`
Hogyan kap értéket egy pointer változók esetén?
A címképző operátor segítségével `&`. Az előző pointer esetén `ip=&i;`
Mi a tipikus hiba a pointerek használatánál?
Használat előtt nem adnak nekik értéket.
Mi az az indirekt értékadás?
Amikor egy változó nem közvetlenül egy másik változótól kap értéket, hanem az adatforrás címén keresztül.
Szavakban kifejezve mit jelent a következő kifejezés: `i=*ip;`
Az `i` változó értéke legyen egyenlő az `ip` pointer által címzett változó értékével.

Kérdések

Mik azok a pointerek?
A pointerek olyan speciális adattpusok, amelyek nem értéket tárolnak, hanem objektumoknak (változónak, függvénynek) a cmét tartalmazzák.
Hogyan deklarál egy pointert?
A csillag segítségével `int *ip;`
Hogyan kap értéket egy pointer változók esetén?
A címképző operátor segítségével `&`. Az előző pointer esetén `ip=&i;`
Mi a tipikus hiba a pointerek használatánál?
Használat előtt nem adnak nekik értéket.
Mi az az indirekt értékadás?
Amikor egy változó nem közvetlenül egy másik változótól kap értéket, hanem az adatforrás címén keresztül.
Szavakban kifejezve mit jelent a következő kifejezés: `i=*ip;`
Az `i` változó értéke legyen egyenlő az `ip` pointer által címzett változó értékével.

Kérdések

Mi az a tömb?

Kérdések

Mi az a tömb?

A tömb egy összetett adatszerkezet, amely szigorúan azonos típusú adatokat tartalmaz, az adatok folytonosan egymás után helyezkednek el a tárban.

Kérdések

Mi az a tömb?

A tömb egy összetett adatszerkezet, amely szigorúan azonos típusú adatokat tartalmaz, az adatok folytonosan egymás után helyezkednek el a tárban.

Hogyan deklarál egy tömböt?

Kérdések

Mi az a tömb?

A tömb egy összetett adatszerkezet, amely szigorúan azonos típusú adatokat tartalmaz, az adatok folytonosan egymás után helyezkednek el a tárban.

Hogyan deklarál egy tömböt?

Például: `int t[10];` egy egydimenziós tömb deklarációja.

Kérdések

Mi az a tömb?

A tömb egy összetett adatszerkezet, amely szigorúan azonos típusú adatokat tartalmaz, az adatok folytonosan egymás után helyezkednek el a tárban.

Hogyan deklarál egy tömböt?

Például: `int t[10];` egy egydimenziós tömb deklarációja.

Hány dimenziós lehet egy tömb?

Kérdések

Mi az a tömb?

A tömb egy összetett adatszerkezet, amely szigorúan azonos típusú adatokat tartalmaz, az adatok folytonosan egymás után helyezkednek el a tárban.

Hogyan deklarál egy tömböt?

Például: `int t[10];` egy egydimenziós tömb deklarációja.

Hány dimenziós lehet egy tömb?

Tetszőleges számú dimenzós tömb hozható létre, a memória mérete korlátozhatja.

Kérdések

Mi az a tömb?

A tömb egy összetett adatszerkezet, amely szigorúan azonos típusú adatokat tartalmaz, az adatok folytonosan egymás után helyezkednek el a tárban.

Hogyan deklarál egy tömböt?

Például: `int t[10];` egy egydimenziós tömb deklarációja.

Hány dimenziós lehet egy tömb?

Tetszőleges számú dimenzós tömb hozható létre, a memória mérete korlátozhatja.

Hogyan férünk egy tömb eleméhez?

Kérdések

Mi az a tömb?
A tömb egy összetett adatszerkezet, amely szigorúan azonos típusú adatokat tartalmaz, az adatok folytonosan egymás után helyezkednek el a tárban.
Hogyan deklarál egy tömböt?
Például: `int t[10];` egy egydimenziós tömb deklarációja.
Hány dimenziós lehet egy tömb?
Tetszőleges számú dimenzós tömb hozható létre, a memória mérete korlátozhatja.
Hogyan férünk egy tömb eleméhez?
Indexeléssel `i=t[5];`, vagy `i=t[j];`, ahol `j` egy egész jellegű változó.

Kérdések

Mi az a tömb?
A tömb egy összetett adatszerkezet, amely szigorúan azonos típusú adatokat tartalmaz, az adatok folytonosan egymás után helyezkednek el a tárban.
Hogyan deklarál egy tömböt?
Például: `int t[10];` egy egydimenziós tömb deklarációja.
Hány dimenziós lehet egy tömb?
Tetszőleges számú dimenzós tömb hozható létre, a memória mérete korlátozhatja.
Hogyan férünk egy tömb eleméhez?
Indexeléssel `i=t[5];`, vagy `i=t[j];`, ahol `j` egy egész jellegű változó.
Hogyan helyezkednek a tömb elemei a tárban?

Kérdések

Mi az a tömb?
A tömb egy összetett adatszerkezet, amely szigorúan azonos típusú adatokat tartalmaz, az adatok folytonosan egymás után helyezkednek el a tárban.
Hogyan deklarál egy tömböt?
Például: `int t[10];` egy egydimenziós tömb deklarációja.
Hány dimenziós lehet egy tömb?
Tetszőleges számú dimenzós tömb hozható létre, a memória mérete korlátozhatja.
Hogyan férünk egy tömb eleméhez?
Indexeléssel `i=t[5];`, vagy `i=t[j];`, ahol `j` egy egész jellegű változó.
Hogyan helyezkednek a tömb elemei a tárban?
A legalacsonyabb indexű (`t[0]`) helyezkedik el a legalacsonyabb memóriacímen, majd folyamatosan egészen az utolsó elemig (10 elem esetén `t[9]`).

Kérdések

Mi az a tömb?
A tömb egy összetett adatszerkezet, amely szigorúan azonos típusú adatokat tartalmaz, az adatok folytonosan egymás után helyezkednek el a tárban.
Hogyan deklarál egy tömböt?
Például: `int t[10];` egy egydimenziós tömb deklarációja.
Hány dimenziós lehet egy tömb?
Tetszőleges számú dimenzós tömb hozható létre, a memória mérete korlátozhatja.
Hogyan férünk egy tömb eleméhez?
Indexeléssel `i=t[5];`, vagy `i=t[j];`, ahol `j` egy egész jellegű változó.
Hogyan helyezkednek a tömb elemei a tárban?
A legalacsonyabb indexű (`t[0]`) helyezkedik el a legalacsonyabb memóriacímen, majd folyamatosan egészen az utolsó elemig (10 elem esetén `t[9]`).
Hogyan deklarál többdimenziós tömböt?

Kérdések

Mi az a tömb?
A tömb egy összetett adatszerkezet, amely szigorúan azonos típusú adatokat tartalmaz, az adatok folytonosan egymás után helyezkednek el a tárban.
Hogyan deklarál egy tömböt?
Például: `int t[10];` egy egydimenziós tömb deklarációja.
Hány dimenziós lehet egy tömb?
Tetszőleges számú dimenzós tömb hozható létre, a memória mérete korlátozhatja.
Hogyan férünk egy tömb eleméhez?
Indexeléssel `i=t[5];`, vagy `i=t[j];`, ahol `j` egy egész jellegű változó.
Hogyan helyezkednek a tömb elemei a tárban?
A legalacsonyabb indexű (`t[0]`) helyezkedik el a legalacsonyabb memóriacímen, majd folyamatosan egészen az utolsó elemig (10 elem esetén `t[9]`).
Hogyan deklarál többdimenziós tömböt?
Példa: `int t[3][2];`

Kérdések

Mi az a tömb?
A tömb egy összetett adatszerkezet, amely szigorúan azonos típusú adatokat tartalmaz, az adatok folytonosan egymás után helyezkednek el a tárban.
Hogyan deklarál egy tömböt?
Például: `int t[10];` egy egydimenziós tömb deklarációja.
Hány dimenziós lehet egy tömb?
Tetszőleges számú dimenzós tömb hozható létre, a memória mérete korlátozhatja.
Hogyan férünk egy tömb eleméhez?
Indexeléssel `i=t[5];`, vagy `i=t[j];`, ahol `j` egy egész jellegű változó.
Hogyan helyezkednek a tömb elemei a tárban?
A legalacsonyabb indexű (`t[0]`) helyezkedik el a legalacsonyabb memóriacímen, majd folyamatosan egészen az utolsó elemig (10 elem esetén `t[9]`).
Hogyan deklarál többdimenziós tömböt?
Példa: `int t[3][2];`

Kérdések

Hogyan helyezkednek el többdimenziós tömb esetén az elemek a memóriában?

Kérdések

Hogyan helyezkednek el többdimenziós tömb esetén az elemek a memóriában?
Legelöl a legalacsonyabb indexű elem van (`t[0][0]`), majd a legalacsonyabb (jobboldali) index növekszik (`t[0][1]...t[1][0]` végül a legmagasabb index zárja a tömböt (`t[2][1]`).

Kérdések

Hogyan helyezkednek el többdimenziós tömb esetén az elemek a memóriában?
Legelöl a legalacsonyabb indexű elem van (`t[0][0]`), majd a legalacsonyabb (jobboldali) index növekszik (`t[0][1]...t[1][0]` végül a legmagasabb index zárja a tömböt (`t[2][1]`).
Mi a tipikus hiba a tömbök kezelésénél?

Kérdések

Hogyan helyezkednek el többdimenziós tömb esetén az elemek a memóriában?
Legelöl a legalacsonyabb indexű elem van (`t[0][0]`), majd a legalacsonyabb (jobboldali) index növekszik (`t[0][1]...t[1][0]` végül a legmagasabb index zárja a tömböt (`t[2][1]`).
Mi a tipikus hiba a tömbök kezelésénél?
A túlindexelés, mert a rendszer nem ellenőrzi az indexelő változót.

Kérdések

Hogyan helyezkednek el többdimenziós tömb esetén az elemek a memóriában?
Legelöl a legalacsonyabb indexű elem van (`t[0][0]`), majd a legalacsonyabb (jobboldali) index növekszik (`t[0][1]...t[1][0]` végül a legmagasabb index zárja a tömböt (`t[2][1]`).
Mi a tipikus hiba a tömbök kezelésénél?
A túlindexelés, mert a rendszer nem ellenőrzi az indexelő változót.
Hogyan kapjuk meg egy tömb kezdőcímét?

Kérdések

Hogyan helyezkednek el többdimenziós tömb esetén az elemek a memóriában?
Legelöl a legalacsonyabb indexű elem van (`t[0][0]`), majd a legalacsonyabb (jobboldali) index növekszik (`t[0][1]...t[1][0]` végül a legmagasabb index zárja a tömböt (`t[2][1]`).
Mi a tipikus hiba a tömbök kezelésénél?
A túlindexelés, mert a rendszer nem ellenőrzi az indexelő változót.
Hogyan kapjuk meg egy tömb kezdőcímét?
A tömb neve a tömb kezőcíme.

Kérdések

Hogyan helyezkednek el többdimenziós tömb esetén az elemek a memóriában?
Legelöl a legalacsonyabb indexű elem van (`t[0][0]`), majd a legalacsonyabb (jobboldali) index növekszik (`t[0][1]...t[1][0]` végül a legmagasabb index zárja a tömböt (`t[2][1]`).
Mi a tipikus hiba a tömbök kezelésénél?
A túlindexelés, mert a rendszer nem ellenőrzi az indexelő változót.
Hogyan kapjuk meg egy tömb kezdőcímét?
A tömb neve a tömb kezőcíme.
Milyen műveleteket ismer a pointer aritmetika?

Kérdések

Hogyan helyezkednek el többdimenziós tömb esetén az elemek a memóriában?
Legelöl a legalacsonyabb indexű elem van (`t[0][0]`), majd a legalacsonyabb (jobboldali) index növekszik (`t[0][1]...t[1][0]` végül a legmagasabb index zárja a tömböt (`t[2][1]`).
Mi a tipikus hiba a tömbök kezelésénél?
A túlindexelés, mert a rendszer nem ellenőrzi az indexelő változót.
Hogyan kapjuk meg egy tömb kezdőcímét?
A tömb neve a tömb kezőcíme.
Milyen műveleteket ismer a pointer aritmetika?
`++`, `--`, pointer és egész összeadása, pointer és egész kivonása, és két pointer kivonása.

Kérdések

Hogyan helyezkednek el többdimenziós tömb esetén az elemek a memóriában?
Legelöl a legalacsonyabb indexű elem van (`t[0][0]`), majd a legalacsonyabb (jobboldali) index növekszik (`t[0][1]...t[1][0]` végül a legmagasabb index zárja a tömböt (`t[2][1]`).
Mi a tipikus hiba a tömbök kezelésénél?
A túlindexelés, mert a rendszer nem ellenőrzi az indexelő változót.
Hogyan kapjuk meg egy tömb kezdőcímét?
A tömb neve a tömb kezőcíme.
Milyen műveleteket ismer a pointer aritmetika?
`++`, `--`, pointer és egész összeadása, pointer és egész kivonása, és két pointer kivonása.
Hogyan változik a memória cím, ha egy pointert inkrementálunk?

Kérdések

Hogyan helyezkednek el többdimenziós tömb esetén az elemek a memóriában?
Legelöl a legalacsonyabb indexű elem van (`t[0][0]`), majd a legalacsonyabb (jobboldali) index növekszik (`t[0][1]...t[1][0]` végül a legmagasabb index zárja a tömböt (`t[2][1]`).
Mi a tipikus hiba a tömbök kezelésénél?
A túlindexelés, mert a rendszer nem ellenőrzi az indexelő változót.
Hogyan kapjuk meg egy tömb kezdőcímét?
A tömb neve a tömb kezőcíme.
Milyen műveleteket ismer a pointer aritmetika?
`++`, `--`, pointer és egész összeadása, pointer és egész kivonása, és két pointer kivonása.
Hogyan változik a memória cím, ha egy pointert inkrementálunk?
Annyival növekszik, amekkera a típus mérete (ezért fontos a pointernél a típus megjelölése.)

Kérdések

Hogyan helyezkednek el többdimenziós tömb esetén az elemek a memóriában?
Legelöl a legalacsonyabb indexű elem van (`t[0][0]`), majd a legalacsonyabb (jobboldali) index növekszik (`t[0][1]...t[1][0]` végül a legmagasabb index zárja a tömböt (`t[2][1]`).
Mi a tipikus hiba a tömbök kezelésénél?
A túlindexelés, mert a rendszer nem ellenőrzi az indexelő változót.
Hogyan kapjuk meg egy tömb kezdőcímét?
A tömb neve a tömb kezőcíme.
Milyen műveleteket ismer a pointer aritmetika?
`++`, `--`, pointer és egész összeadása, pointer és egész kivonása, és két pointer kivonása.
Hogyan változik a memória cím, ha egy pointert inkrementálunk?
Annyival növekszik, amekkera a típus mérete (ezért fontos a pointernél a típus megjelölése.)
Mi történik, ha egy tömb kezdőcímét átadjuk egy pointernek?

Kérdések

Hogyan helyezkednek el többdimenziós tömb esetén az elemek a memóriában?
Legelöl a legalacsonyabb indexű elem van (`t[0][0]`), majd a legalacsonyabb (jobboldali) index növekszik (`t[0][1]...t[1][0]` végül a legmagasabb index zárja a tömböt (`t[2][1]`).
Mi a tipikus hiba a tömbök kezelésénél?
A túlindexelés, mert a rendszer nem ellenőrzi az indexelő változót.
Hogyan kapjuk meg egy tömb kezdőcímét?
A tömb neve a tömb kezőcíme.
Milyen műveleteket ismer a pointer aritmetika?
`++`, `--`, pointer és egész összeadása, pointer és egész kivonása, és két pointer kivonása.
Hogyan változik a memória cím, ha egy pointert inkrementálunk?
Annyival növekszik, amekkera a típus mérete (ezért fontos a pointernél a típus megjelölése.)
Mi történik, ha egy tömb kezdőcímét átadjuk egy pointernek?
A pointer pontosan úgy indexelhető, mint az eredeti tömb.

Kérdések

Hogyan helyezkednek el többdimenziós tömb esetén az elemek a memóriában?
Legelöl a legalacsonyabb indexű elem van (`t[0][0]`), majd a legalacsonyabb (jobboldali) index növekszik (`t[0][1]...t[1][0]` végül a legmagasabb index zárja a tömböt (`t[2][1]`).
Mi a tipikus hiba a tömbök kezelésénél?
A túlindexelés, mert a rendszer nem ellenőrzi az indexelő változót.
Hogyan kapjuk meg egy tömb kezdőcímét?
A tömb neve a tömb kezőcíme.
Milyen műveleteket ismer a pointer aritmetika?
`++`, `--`, pointer és egész összeadása, pointer és egész kivonása, és két pointer kivonása.
Hogyan változik a memória cím, ha egy pointert inkrementálunk?
Annyival növekszik, amekkera a típus mérete (ezért fontos a pointernél a típus megjelölése.)
Mi történik, ha egy tömb kezdőcímét átadjuk egy pointernek?
A pointer pontosan úgy indexelhető, mint az eredeti tömb.
Milyen tulajdonságai vannak a `void` pointernek?

Kérdések

Hogyan helyezkednek el többdimenziós tömb esetén az elemek a memóriában?
Legelöl a legalacsonyabb indexű elem van (`t[0][0]`), majd a legalacsonyabb (jobboldali) index növekszik (`t[0][1]...t[1][0]` végül a legmagasabb index zárja a tömböt (`t[2][1]`).
Mi a tipikus hiba a tömbök kezelésénél?
A túlindexelés, mert a rendszer nem ellenőrzi az indexelő változót.
Hogyan kapjuk meg egy tömb kezdőcímét?
A tömb neve a tömb kezőcíme.
Milyen műveleteket ismer a pointer aritmetika?
`++`, `--`, pointer és egész összeadása, pointer és egész kivonása, és két pointer kivonása.
Hogyan változik a memória cím, ha egy pointert inkrementálunk?
Annyival növekszik, amekkera a típus mérete (ezért fontos a pointernél a típus megjelölése.)
Mi történik, ha egy tömb kezdőcímét átadjuk egy pointernek?
A pointer pontosan úgy indexelhető, mint az eredeti tömb.
Milyen tulajdonságai vannak a `void` pointernek?
Nincs típusa, tehát nem igaz rá a pointer aritmetika, csak egy memória cím.

Kérdések

Hogyan helyezkednek el többdimenziós tömb esetén az elemek a memóriában?
Legelöl a legalacsonyabb indexű elem van (`t[0][0]`), majd a legalacsonyabb (jobboldali) index növekszik (`t[0][1]...t[1][0]` végül a legmagasabb index zárja a tömböt (`t[2][1]`).
Mi a tipikus hiba a tömbök kezelésénél?
A túlindexelés, mert a rendszer nem ellenőrzi az indexelő változót.
Hogyan kapjuk meg egy tömb kezdőcímét?
A tömb neve a tömb kezőcíme.
Milyen műveleteket ismer a pointer aritmetika?
`++`, `--`, pointer és egész összeadása, pointer és egész kivonása, és két pointer kivonása.
Hogyan változik a memória cím, ha egy pointert inkrementálunk?
Annyival növekszik, amekkera a típus mérete (ezért fontos a pointernél a típus megjelölése.)
Mi történik, ha egy tömb kezdőcímét átadjuk egy pointernek?
A pointer pontosan úgy indexelhető, mint az eredeti tömb.
Milyen tulajdonságai vannak a `void` pointernek?
Nincs típusa, tehát nem igaz rá a pointer aritmetika, csak egy memória cím.

Kérdések

Hogyan értelmezi a C a sztringeket?

Kérdések

Hogyan értelmezi a C a sztringeket?

A sztring egy karakter kömb, amelyet egy **0** értékű elem zár le.

Kérdések

Hogyan értelmezi a C a sztringeket?
A sztring egy karakter kömb, amelyet egy 0 értékű elem zár le.
Hogyan inicializálunk egy sztringet?

Kérdések

Hogyan értelmezi a C a sztringeket?

A sztring egy karakter kömb, amelyet egy `0` értékű elem zár le.

Hogyan inicializálunk egy sztringet?

Példa: `char str[]="Humpty Dumpty";`

Kérdések

Hogyan értelmezi a C a sztringeket?

A sztring egy karakter kömb, amelyet egy `0` értékű elem zár le.

Hogyan inicializálunk egy sztringet?

Példa: `char str[]="Humpty Dumpty";`

Hogyan deklarálunk egy függvény pointert?

Kérdések

Hogyan értelmezi a C a sztringeket?

A sztring egy karakter kömb, amelyet egy `0` értékű elem zár le.

Hogyan inicializálunk egy sztringet?

Példa: `char str[]="Humpty Dumpty";`

Hogyan deklarálunk egy függvény pointert?

Példa: `int (*fgvpt)(int);` ez egy olyan függvényre fog ”mutatni”, amely `int` típusú és egy `int` típusú paramétere van.

Kérdések

Hogyan értelmezi a C a sztringeket?
A sztring egy karakter kömb, amelyet egy `0` értékű elem zár le.
Hogyan inicializálunk egy sztringet?
Példa: `char str[]="Humpty Dumpty";`
Hogyan deklarálunk egy függvény pointert?
Példa: `int (*fgvpt)(int);` ez egy olyan függvényre fog ”mutatni”, amely `int` típusú és egy `int` típusú paramétere van.
Hogyan adjuk át az `int fgv(int);` függvény címét a fenti pointernek?

Kérdések

Hogyan értelmezi a C a sztringeket?
A sztring egy karakter kömb, amelyet egy `0` értékű elem zár le.
Hogyan inicializálunk egy sztringet?
Példa: `char str[]="Humpty Dumpty";`
Hogyan deklarálunk egy függvény pointert?
Példa: `int (*fgvpt)(int);` ez egy olyan függvényre fog ”mutatni”, amely `int` típusú és egy `int` típusú paramétere van.
Hogyan adjuk át az `int fgv(int);` függvény címét a fenti pointernek?
`fgvpt=fgv;` tehát a függvény neve a függvény címe.

Kérdések

Hogyan értelmezi a C a sztringeket?
A sztring egy karakter kömb, amelyet egy `0` értékű elem zár le.
Hogyan inicializálunk egy sztringet?
Példa: `char str[]="Humpty Dumpty";`
Hogyan deklarálunk egy függvény pointert?
Példa: `int (*fgvpt)(int);` ez egy olyan függvényre fog ”mutatni”, amely `int` típusú és egy `int` típusú paramétere van.
Hogyan adjuk át az `int fgv(int);` függvény címét a fenti pointernek?
`fgvpt=fgv;` tehát a függvény neve a függvény címe.
Hogyan hívjuk meg indirekt módon a fennti `fgv` függvény?

Kérdések

Hogyan értelmezi a C a sztringeket?
A sztring egy karakter kömb, amelyet egy `0` értékű elem zár le.
Hogyan inicializálunk egy sztringet?
Példa: `char str[]="Humpty Dumpty";`
Hogyan deklarálunk egy függvény pointert?
Példa: `int (*fgvpt)(int);` ez egy olyan függvényre fog ”mutatni”, amely `int` típusú és egy `int` típusú paramétere van.
Hogyan adjuk át az `int fgv(int);` függvény címét a fenti pointernek?
`fgvpt=fgv;` tehát a függvény neve a függvény címe.
Hogyan hívjuk meg indirekt módon a fennti `fgv` függvény?

```
r=fgvpt(5);
```

Kérdések

Hogyan értelmezi a C a sztringeket?
A sztring egy karakter kömb, amelyet egy `0` értékű elem zár le.
Hogyan inicializálunk egy sztringet?
Példa: `char str[]="Humpty Dumpty";`
Hogyan deklarálunk egy függvény pointert?
Példa: `int (*fgvpt)(int);` ez egy olyan függvényre fog ”mutatni”, amely `int` típusú és egy `int` típusú paramétere van.
Hogyan adjuk át az `int fgv(int);` függvény címét a fenti pointernek?
`fgvpt=fgv;` tehát a függvény neve a függvény címe.
Hogyan hívjuk meg indirekt módon a fennti `fgv` függvény?

```
r=fgvpt(5);
```