

Programozási nyelvek 1

Szathmáry László
Debreceni Egyetem
Informatikai Kar

7. előadás

- stílus (folyt.)
- mutatók
- struct és typedef

(utolsó módosítás: 2025. febr. 20.)

2024-2025, 2. félév



Stílus

„A stílus maga az ember.”

Georges-Louis Leclerc de Buffon

Az, hogy megírok egy programot és az lefut helyesen, az egy dolog.

Törekedjünk arra, hogy maga a forráskód is **szép** legyen. Egy program elkészítése egy szellemi termék, egy mérnöki munka. Nem mindegy, hogy mit adunk ki a kezünk közül!

```
1 #include <stdio.h>
2 int main(){
3     printf("hello\n");
4     return 0;}
5
```



```
1 #include <stdio.h>
2
3 int main( )
4 {
5     printf("hello\n");
6
7     return 0;
8 }
```

Az 5. előadás anyagából átemelve.

Stílus

T. f. h. megírtunk egy programot, s az lefordul. Kész vagyunk? => **NEM!**

További lépések:

- Teszteljük le a programot több esetre is! Teszteljük le szélsőséges esetekre is!
- Ha úgy látjuk, hogy helyesen működik, akkor itt az ideje megszépíteni a kódot.
- Nézzük át a változók / függvények / eljárások neveit. Ha szerencsétlen neveket használtunk, akkor ezeket nevezzük át. (lásd később)

Stílus

Hogyan tudjuk megszépíteni a kódunkat?

- Nekünk is van egy *style guide*-unk, a félév során ehhez próbáljuk tartani magunkat ([link](#), github). [lásd 5. ea.]
- Használunk egy stíuselemző szoftvert (pl. style50). [lásd 5. ea.]
- A VS Code-ban is van automatikus kódformázó funkció, ezt is lehet használni.

Mi lenne a cél?

A kódot már eleve úgy írjuk meg, hogy azon utólag csak keveset kelljen szépíteni. Alakuljon ki egy szépérzék, s mindenféle gépi segítség nélkül eleve szép kódot írjunk.

Ezt természetesen csakis rengeteg gyakorlással lehet elérni.

Stílus

Demó

A bemutatott kódok esetén most csak a stílusra figyeljünk. Hol tudnánk szépíteni a kódon?

Mutatók

Mutató (pointer):

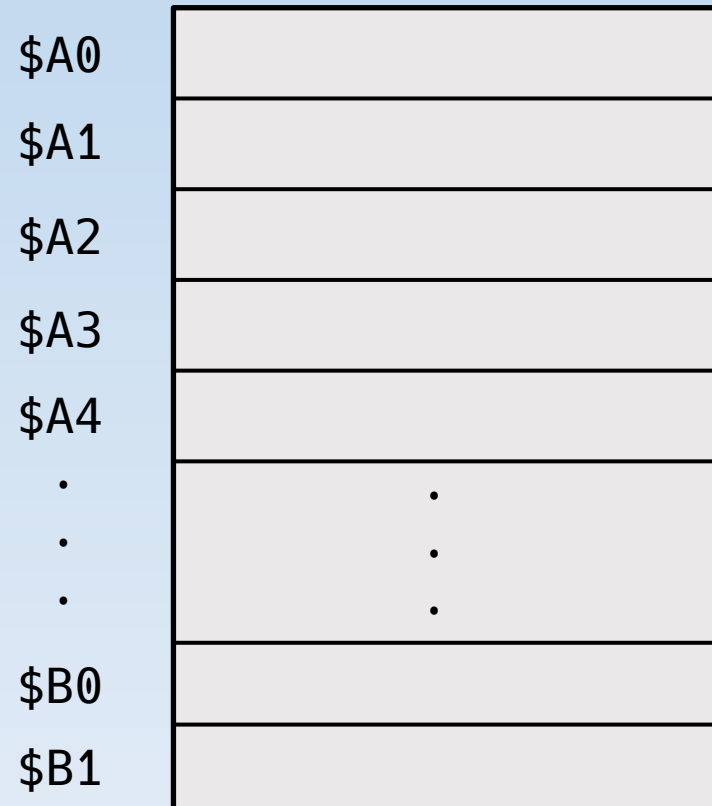
A mutató egy olyan változó, amelynek az értéke egy memóriacím. Magas szintű programozási nyelvekben ehhez hozzárendelődik még az a típus is, amelyre a mutató mutat.

Azt mondjuk, hogy a mutató az értékül felvett memóriacímre *mutat*. Bármely mutatóhoz hozzá lehet rendelni egy speciális NULL értéket; ilyenkor azt mondjuk, hogy a mutató semmire sem mutat.

Mutatók

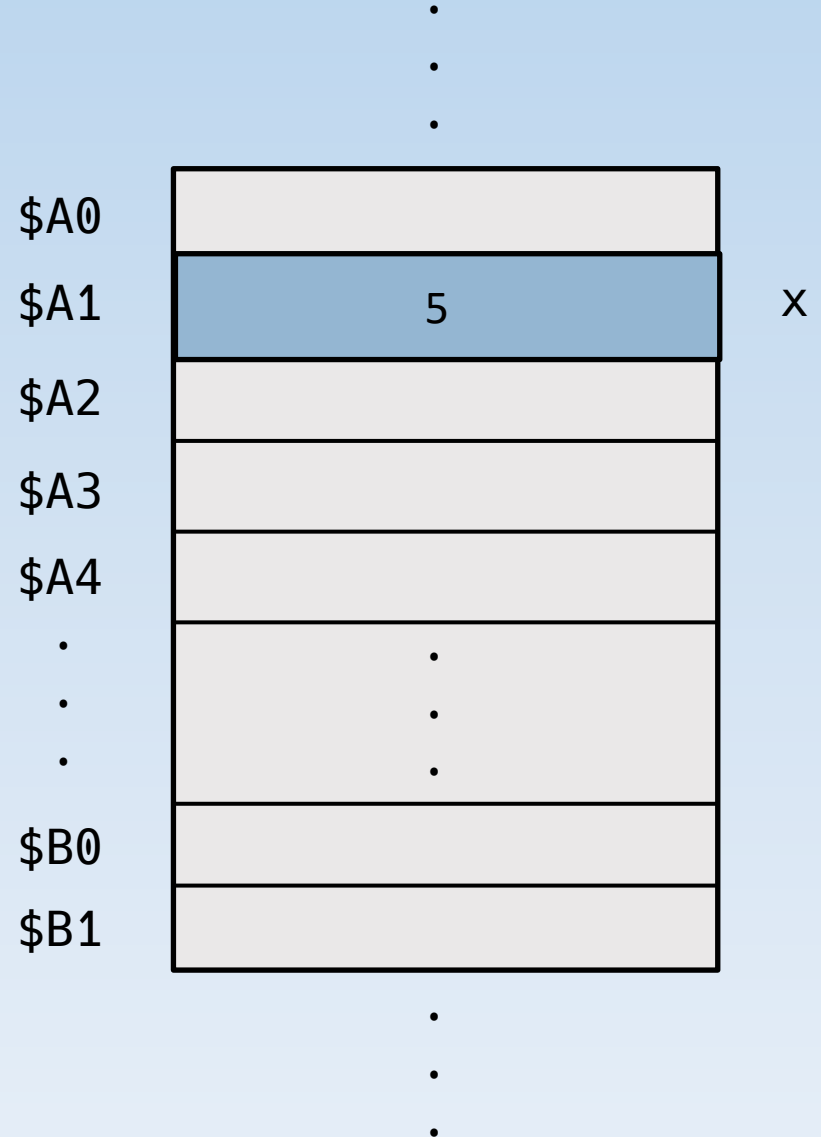
```
int x = 5;  
int y;  
y = x;
```

Példa mutatók nélkül.
Mi történik a memóriában?



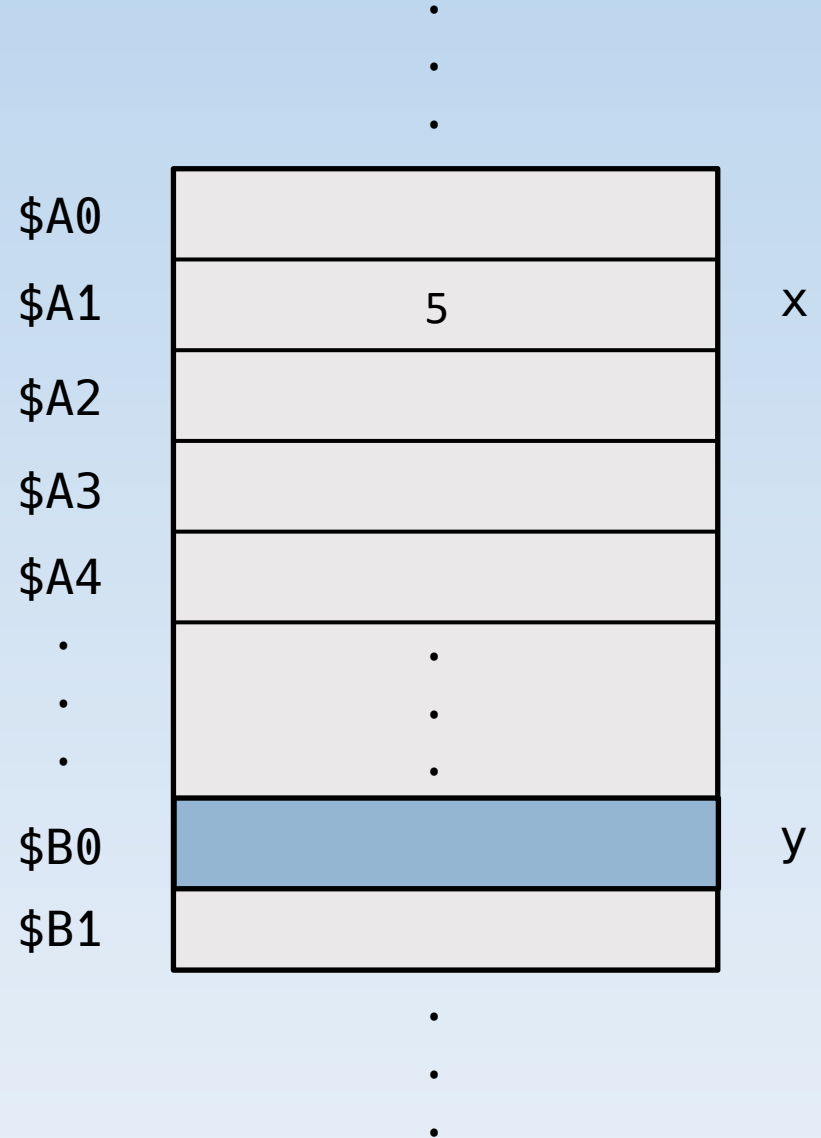
Mutatók

```
int x = 5;  
int y;  
y = x;
```



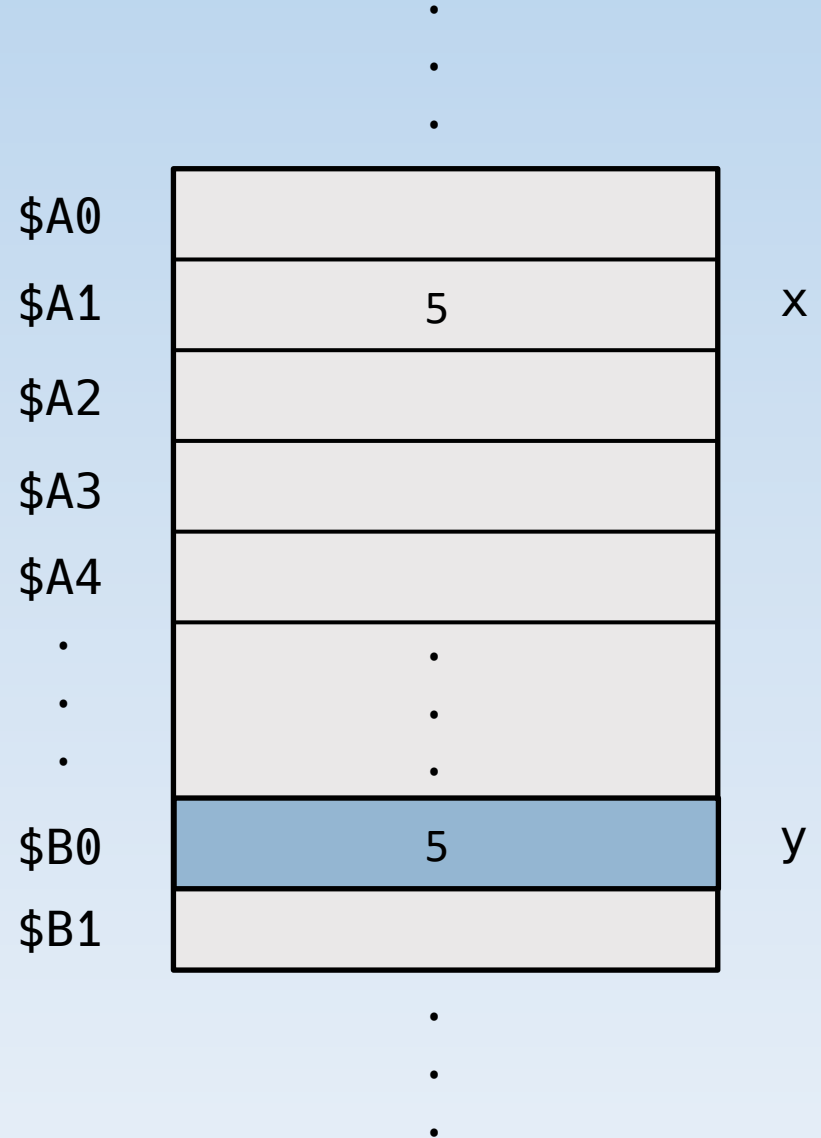
Mutatók

```
int x = 5;  
int y;  
y = x;
```



Mutatók

```
int x = 5;  
int y;  
y = x;
```



Mutatók

```
int z = 5;
int *p;
p = &z;
```

```
int *q;
q = p;
```

Példa mutatókkal.

Mi történik a memóriában?

\$A0

\$A1

\$A2

\$A3

\$A4

•

•

•

\$B0

\$B1

•

•

•

•

•

•

Mutatók

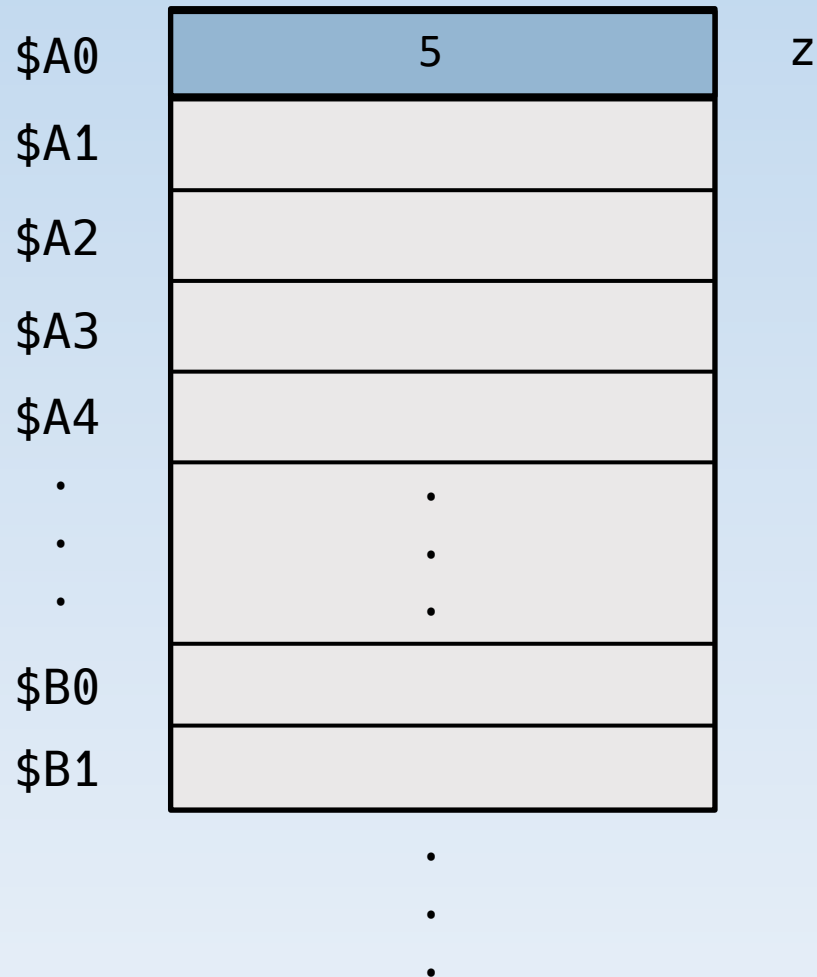
```
int z = 5;
```

```
int *p;
```

```
p = &z;
```

```
int *q;
```

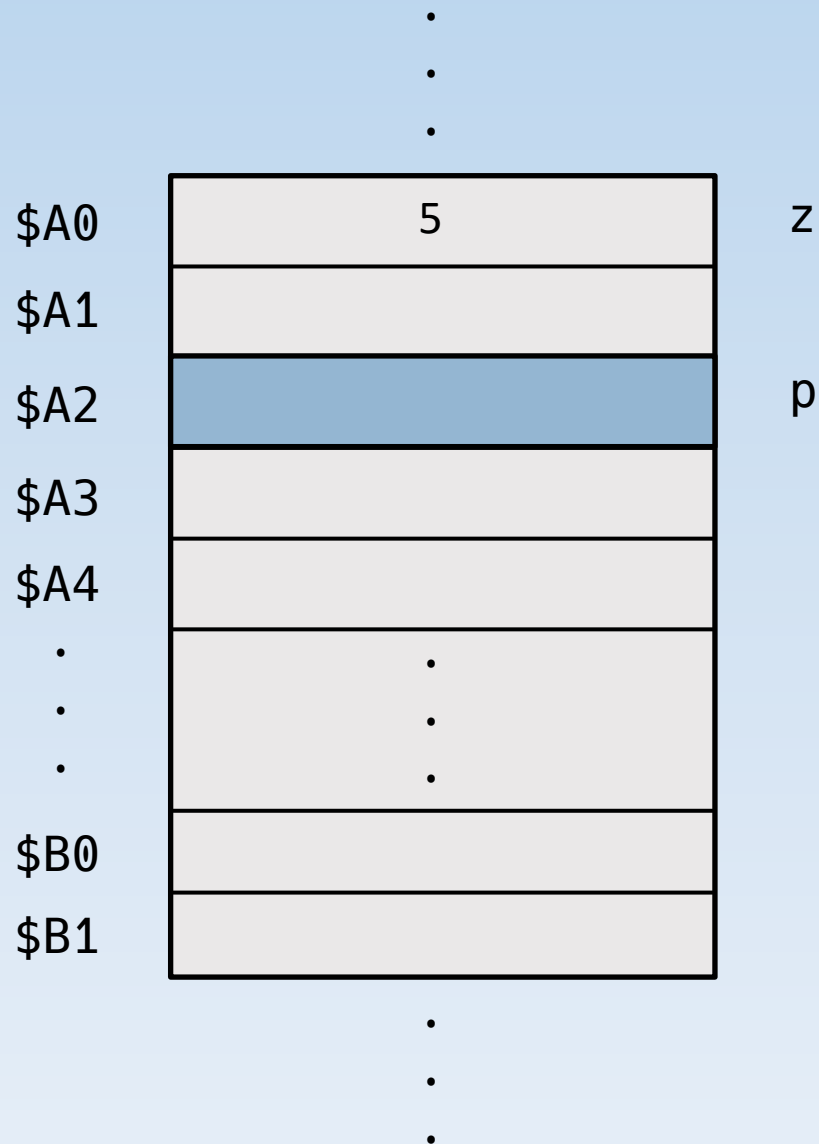
```
q = p;
```



Mutatók

```
int z = 5;
int *p;
p = &z;
```

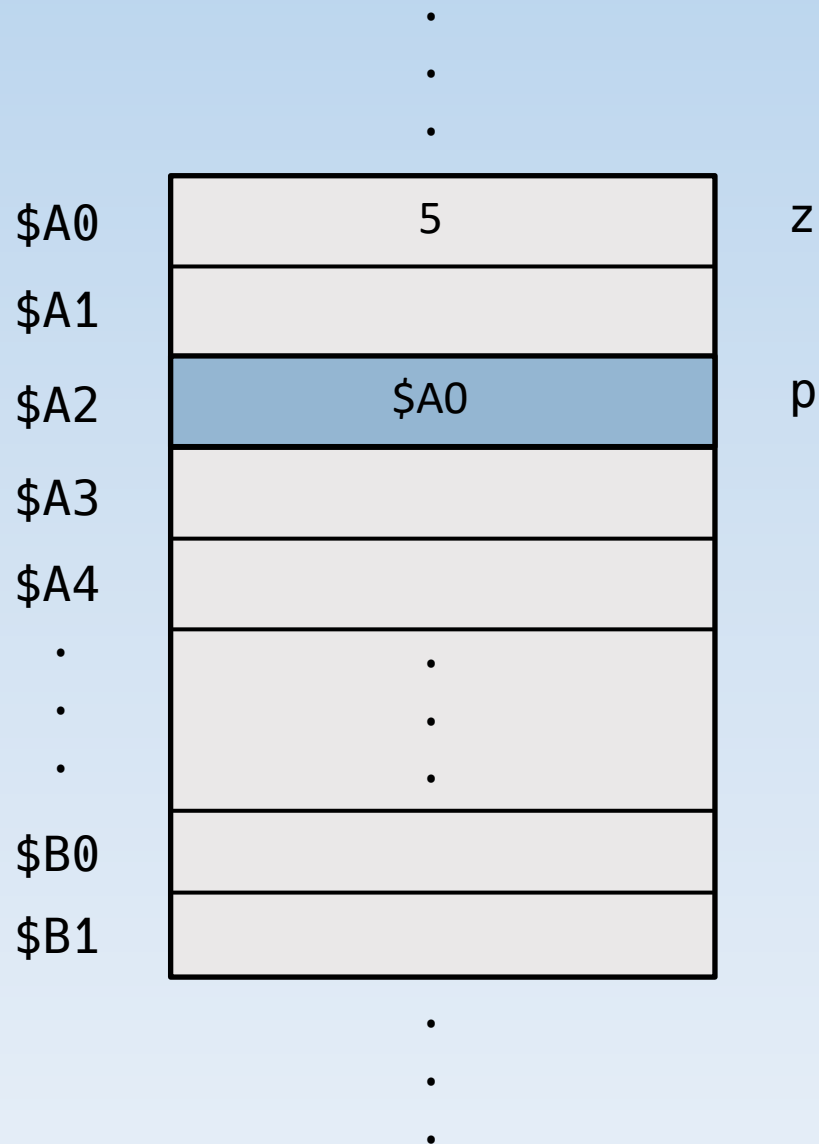
```
int *q;
q = p;
```



Mutatók

```
int z = 5;
int *p;
p = &z;
```

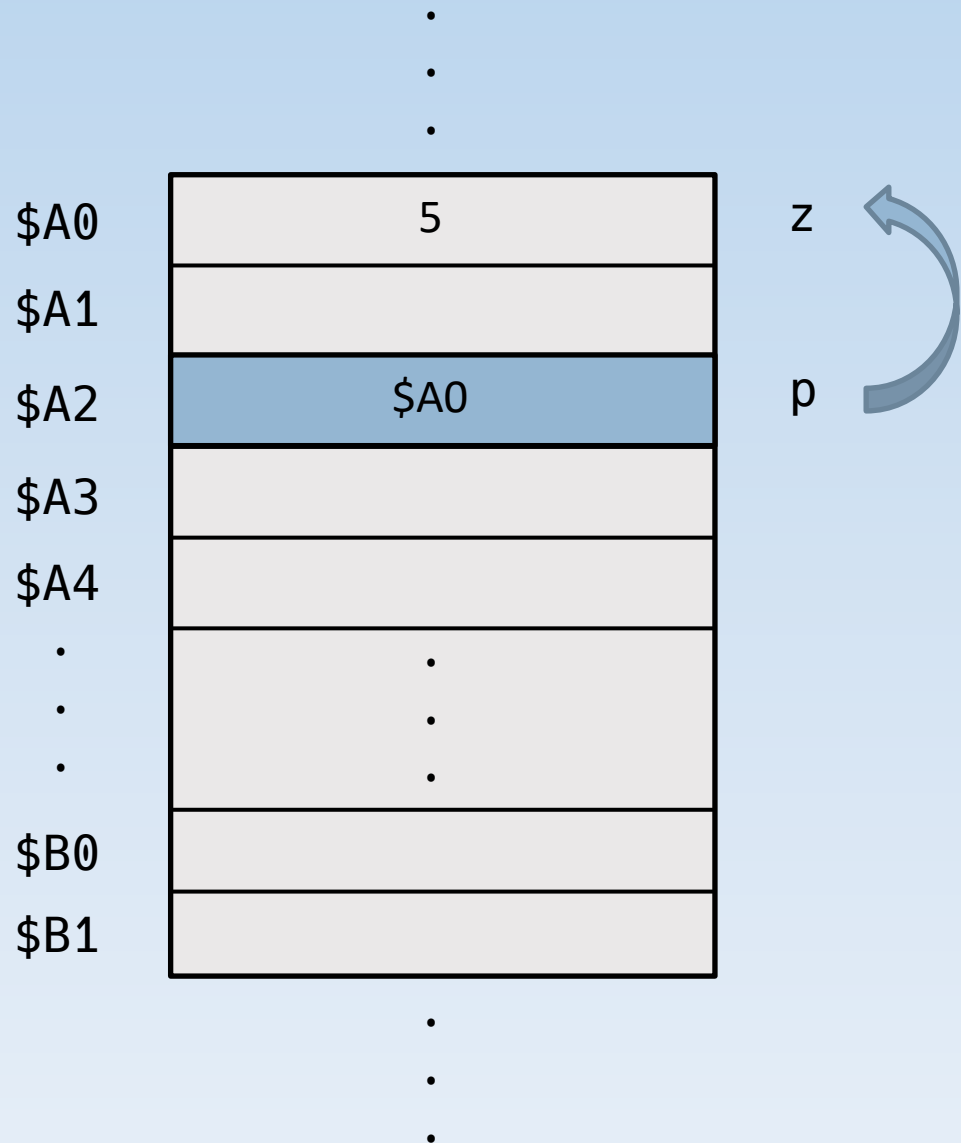
```
int *q;
q = p;
```



Mutatók

```
int z = 5;  
int *p;  
p = &z;
```

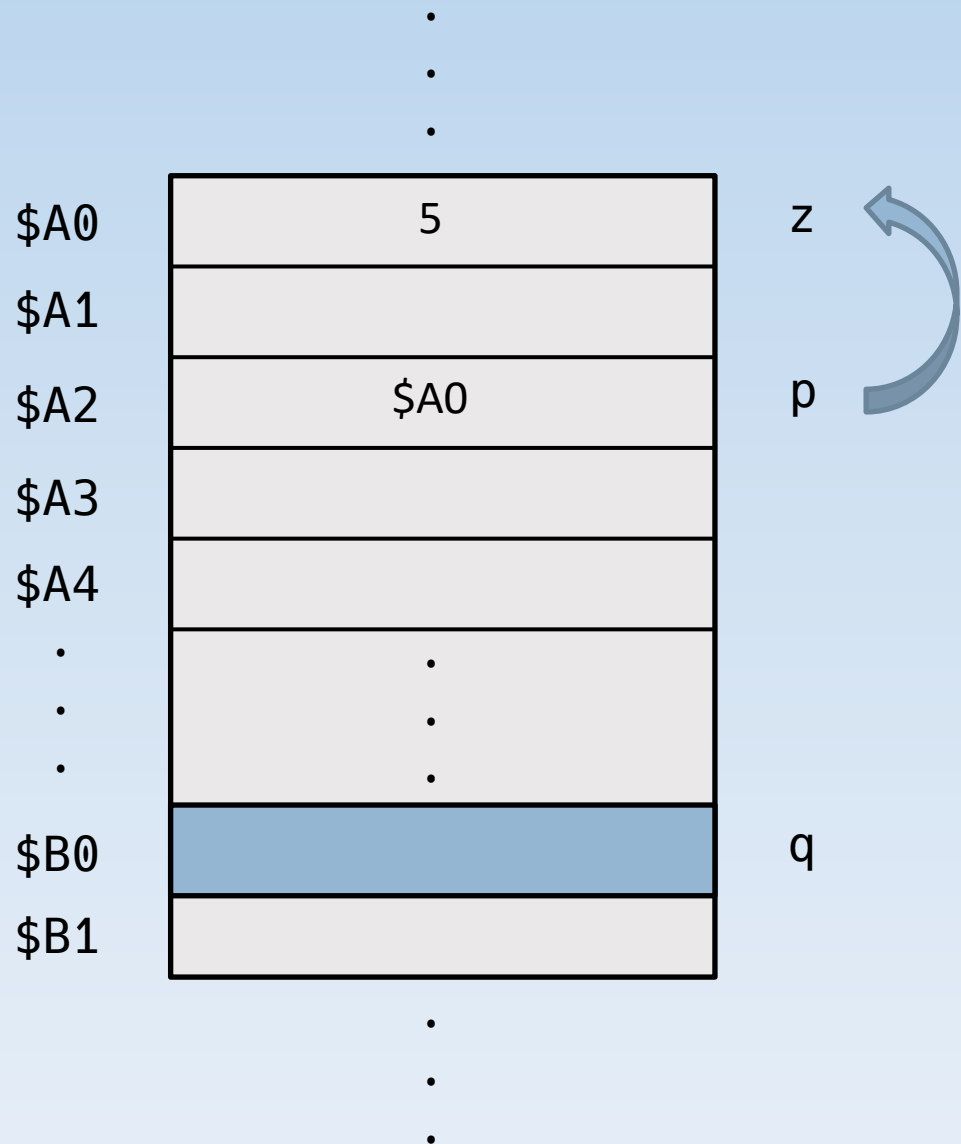
```
int *q;  
q = p;
```



Mutatók

```
int z = 5;
int *p;
p = &z;
```

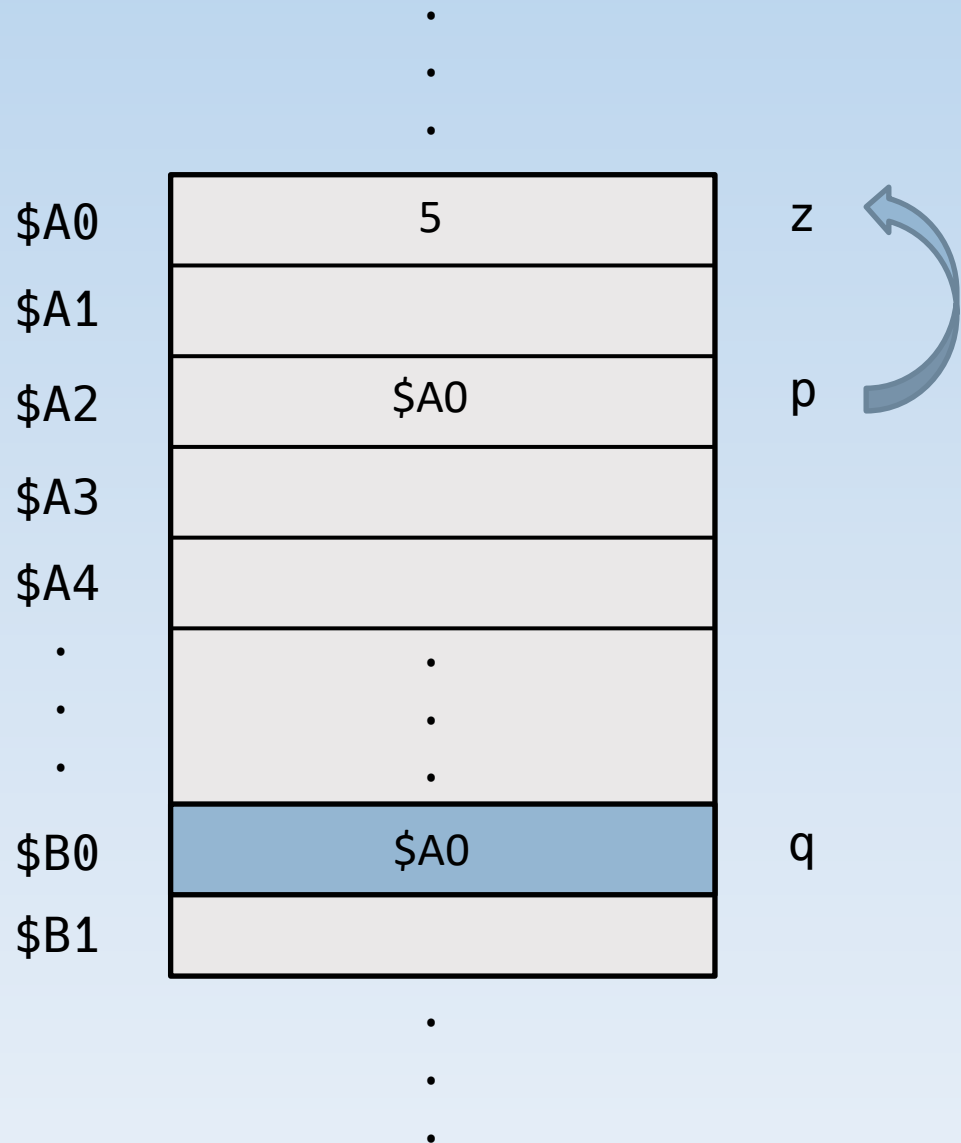
```
int *q;
q = p;
```



Mutatók

```
int z = 5;
int *p;
p = &z;
```

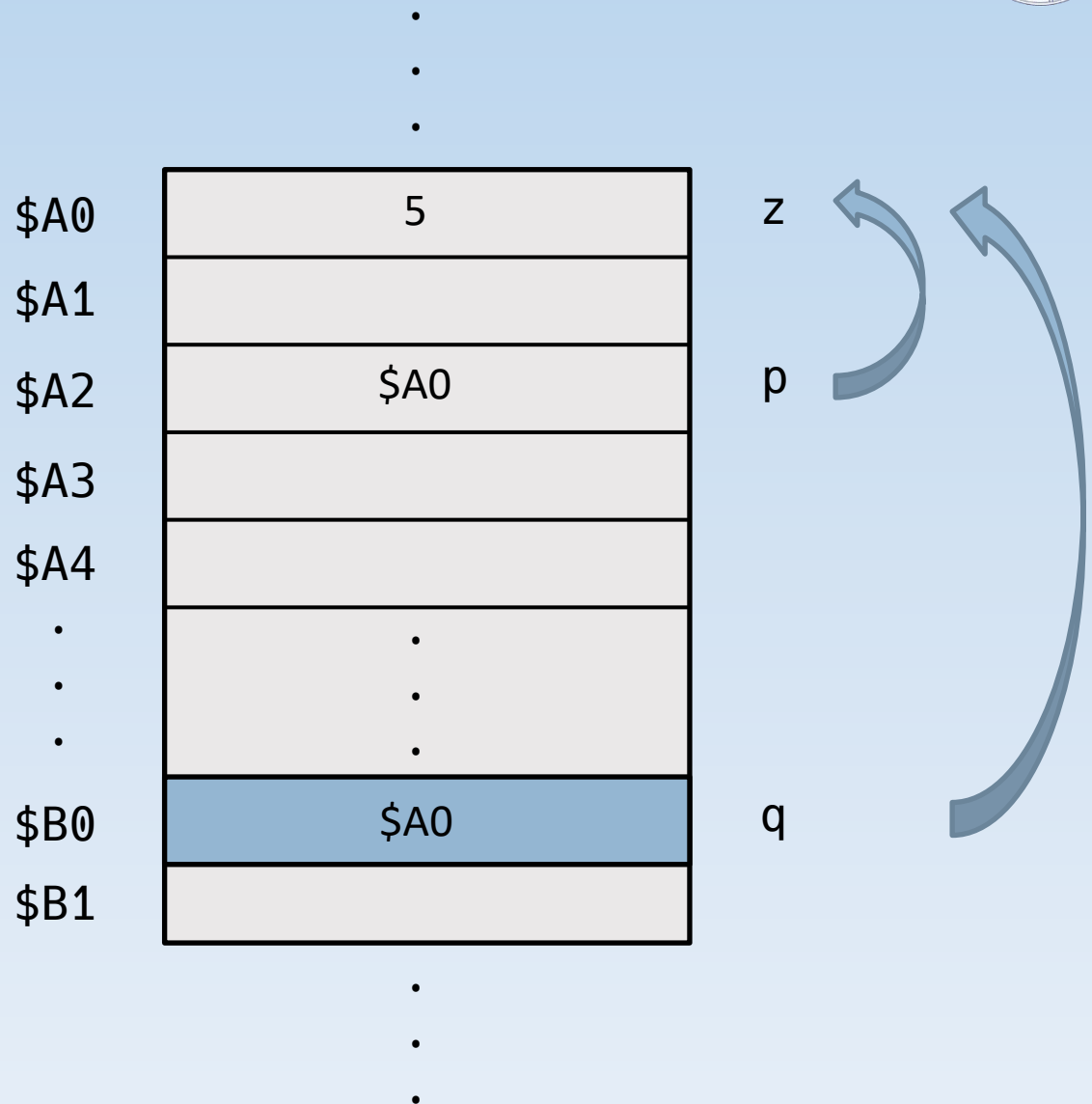
```
int *q;
q = p;
```



Mutatók

```
int z = 5;
int *p;
p = &z;
```

```
int *q;
q = p;
```



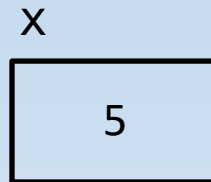
Mutatók

```
int x = 5;  
int y;  
y = x;
```

Példa mutatók nélkül.
Egyszerűsített ábrázolás.

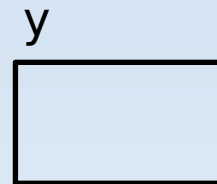
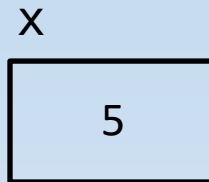
Mutatók

```
int x = 5;  
int y;  
y = x;
```



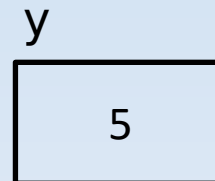
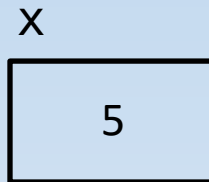
Mutatók

```
int x = 5;  
int y;  
y = x;
```



Mutatók

```
int x = 5;  
int y;  
y = x;
```



Mutatók

```
int z = 5;  
int *p;  
p = &z;
```

```
int *q;  
q = p;
```

Példa mutatókkal.

Egyszerűsített ábrázolás.

Mutatók

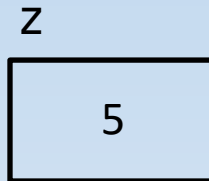
```
int z = 5;
```

```
int *p;
```

```
p = &z;
```

```
int *q;
```

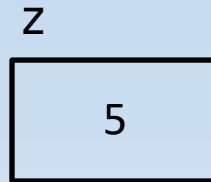
```
q = p;
```



Mutatók

```
int z = 5;  
int *p;  
p = &z;
```

```
int *q;  
q = p;
```



Mutatók

```
int z = 5;  
int *p;  
p = &z;
```

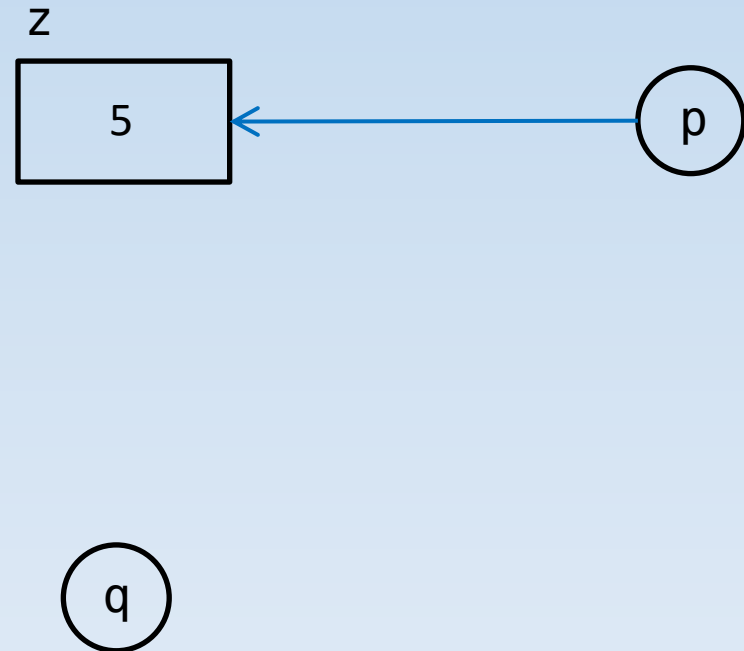
```
int *q;  
q = p;
```



Mutatók

```
int z = 5;  
int *p;  
p = &z;
```

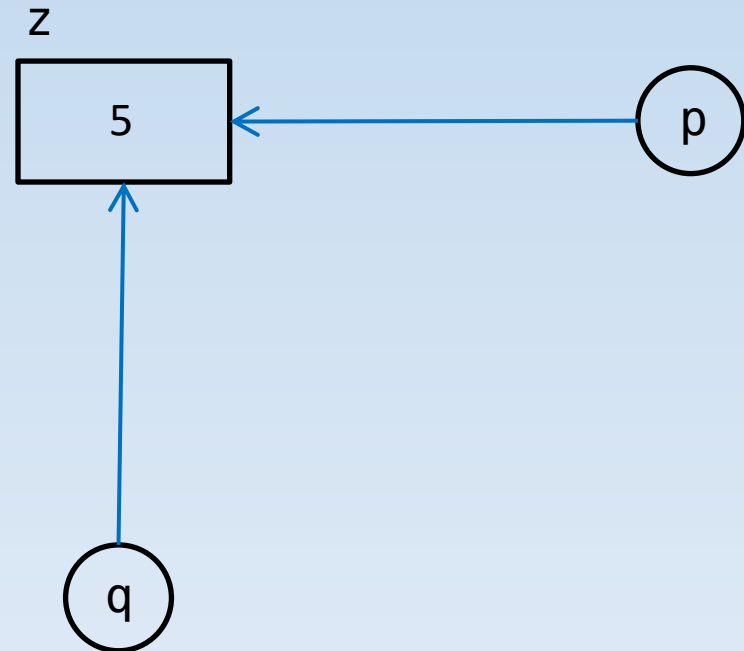
```
int *q;  
q = p;
```



Mutatók

```
int z = 5;  
int *p;  
p = &z;
```

```
int *q;  
q = p;
```



Mutatók

A mutatók esetén az alábbi két operátort használjuk nagyon gyakran:

- **&** operátor (cím operátor)

Ez egy unáris (egyoperandusú) operátor, mely megadja az operandusa memóriabeli címét.

- ***** operátor (indirekció, indirekt hivatkozás)

Szintén unáris (egyoperandusú) operátor. Ha egy mutatóra alkalmazzuk, akkor a mutató által megcímzett (mutatott) objektumhoz férhetünk hozzá.

Mutatók

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int x = 1;
6     int y = 2;
7     int z[10];
8
9     int *p;                                // p egy olyan mutató, ami int -ekre mutathat
10
11     printf("x értéke: %d\n", x);
12     printf("x címe: %p\n", &x);
13     p = &x;                                // p most x-re mutat
14     printf("p értéke: %p\n", p);           // p most x-re mutat
15
16     *p = 5;                                // a p mutatón keresztül módosítjuk x értékét
17     printf("x értéke: %d\n", x);
18
19     p = &z[0];
20     *p = 99;                                // a p mutatón keresztül módosítjuk a z tömb legelső elemét
21
22     return 0;
23 }
```

Mutatók

A const kulcsszó használata

- hagyományos típusok esetén

```
const int x = 1;    // x read-only, vagyis az  
x = 2;           // x értéke NEM módosítható
```

- mutatók esetén kicsit mást fog jelenteni

```
int x = 1;  
int y = 3;
```

```
const int *p;       // a p által mutatott objektum nem módosítható  
p = &x;             // viszont p -t át lehet állítani más objektumra  
p = &y;  
*p = 10;
```

Mutatók

```
1  #include <stdio.h>
2
3  void swap(int *a, int *b)
4  {
5      int tmp = *a;
6      *a = *b;
7      *b = tmp;
8  }
9
10 int main()
11 {
12     int x = 1;
13     int y = 3;
14
15     printf("(előtte) x: %d, y: %d\n", x, y);    // x: 1, y: 3
16
17     swap(&x, &y);
18
19     printf("(utána) x: %d, y: %d\n", x, y);    // x: 3, y: 1
20
21     return 0;
22 }
```

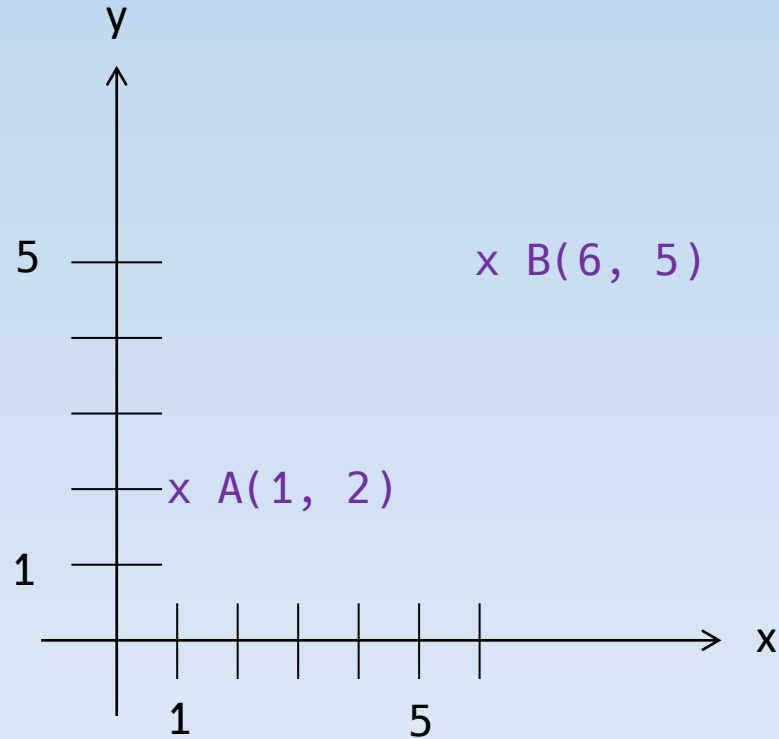
swap:

cseréljük fel
két változó értékét

struct és typedef

A struktúra (nevezzük rekordnak is) egy vagy több, akár különböző típusú változó együttese, amelyet a kényelmes kezelhetőség céljából önálló névvel látunk el.

Lehetővé teszi, hogy az egymással kapcsolatban lévő változók egy csoportját **együtt** kezeljük, szemben az egyedi adatkezeléssel.



struct és typedef

```
10
11 struct pont {
12     int x;
13     int y;
14 };
15
16 int main()
17 {
18     struct pont a;
19     a.x = 1;
20     a.y = 2;
21
22     struct pont b;
23     b.x = 6;
24     b.y = 5;
25
26     printf("Az A pont koordinátái: %d, %d\n", a.x, a.y);
27
28     return 0;
29 }
```

← struktúra
deklarációja

struct és typedef

typedef

Ezzel az utasítással új adattípus-neveket hozhatunk létre.

typedef ... *alias*;

már meglévő
adattípus

másodlagos
név

```
8
9  typedef int egesz;
10
11 int main()
12 {
13     int a = 5;
14     egesz b = 9;
15
16     printf("%d\n", a);    // 5
17     printf("%d\n", b);    // 9
18
19     return 0;
20 }
21
```

struct és typedef

```
10
11 typedef struct {
12     int x;
13     int y;
14 } Pont;
15
16 int main()
17 {
18     Pont a;
19     a.x = 1;
20     a.y = 2;
21
22     printf("Az A pont koordinátái: %d, %d\n", a.x, a.y);
23
24     return 0;
25 }
```

struct és typedef

```
5  #define N 4
6
7  typedef struct {
8      string name;
9      string tel;
10 } Person;
11
12 int main()
13 {
14     Person people[N];
15     people[0].name = "Emma";
16     people[0].tel = "20/123-4567";
17     people[1].name = "Anna";
18     people[1].tel = "30/123-4568";
19     people[2].name = "Cecil";
20     people[2].tel = "30/123-4569";
21     people[3].name = "Eva";
22     people[3].tel = "70/123-4560";
23 }
```

Házi feladat

- A K & R-féle „C Bibliában” nézzék át azokat a részeket, amikről szó volt az előadáson.
- Juhász István jegyzetéből nézzék át azokat a fogalmakat, amikről szó volt az előadáson ([link](#)).