

Mutatók használata 3.

Kormányos Andor

Komplex Rendszerek Fizikája Tanszék

2021. szeptember 22.

A `void *` a pointerek egy univerzális típusa.

- olyankor használjuk, amikor a mutatott adat típusa mindegy
- például memóiafoglaláskor
- nem használható rá a mutató feloldása operátor:

```
1 double d = 1.1234;  
2 void *p = &d;           // ez megy  
3 *p = 5.0;               // hiba
```

A `void *` a pointerek egy univerzális típusa.

- olyankor használjuk, amikor a mutatott adat típusa mindegy
- például memóiafoglaláskor
- nem használható rá a mutató feloldása operátor:

```
1 double d = 1.1234;  
2 void *p = &d;           // ez megy  
3 *p = 5.0;               // hiba
```

Mikor használjuk?

- például ilyen pointert ad vissza a `malloc` függvény
- ilyet vár a `free` függvény is
- ilyet használ az `fread` és `fwrite`
- ezek univerzális függvények, tetszőleges típusú adattal tudnak dolgozni
- az adat típusa nem, de a mérete (hány bájt) számít

Hamarosan látunk példákat

- ha több memóriára van szükségünk (nagy tömb), akkor a halomterületről (heap) kell foglalni
- memóriefoglalás: `malloc`, `calloc` és `realloc` függvényekkel
- a lefoglalt terület kezdetére **mutatókkal (pointer)** hivatkozunk
- miután elvégeztük a feladatot, a memóriaterületet felszabadítjuk

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     int n = 10;
7     double *dyn_v;
8
9     dyn_v = (double *)malloc(n * sizeof(double));
10
11     for (int i = 0; i < n; i++) {
12         dyn_v[i] = i;
13     }
14
15     for (int i = 0; i < n; i++) {
16         printf("%f\n", dyn_v[i]);
17     }
18
19     free(dyn_v);
20
21     return 0;
22 }
```

Mutató deklarálása

- ez a mutató **double** számra (vagy számokra) for mutatni
- nem foglaltunk még memóriát
- még nem mutat sehova

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     int n = 10;
7     double *dyn_v;
8
9     dyn_v = (double *)malloc(n * sizeof(double));
10
11     for (int i = 0; i < n; i++) {
12         dyn_v[i] = i;
13     }
14
15     for (int i = 0; i < n; i++) {
16         printf("%f\n", dyn_v[i]);
17     }
18
19     free(dyn_v);
20
21     return 0;
22 }
```

Mutató deklarálása

- ez a mutató **double** számra (vagy számokra) for mutatni
- nem foglaltunk még memóriát
- még nem mutat sehova

Memória foglalás: **malloc**

- hány bájt memóriát szeretnénk
- elemek száma \times egy elem mérete byte-ban
- a **malloc** függvény egy **void** típusú mutatót ad vissza
- **(double *)** az ún. **cast**-olás, jó programozási gyakorlat

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     int n = 10;
7     double *dyn_v;
8
9     dyn_v = (double *)malloc(n * sizeof(double));
10
11     for (int i = 0; i < n; i++) {
12         dyn_v[i] = i;
13     }
14
15     for (int i = 0; i < n; i++) {
16         printf("%f\n", dyn_v[i]);
17     }
18
19     free(dyn_v);
20
21     return 0;
22 }
```

Egy egész tömböt foglaltunk

- elemek elérése: `dyn_v[i]`
- legelső: `dyn_v[0]`
- legutolsó: `dyn_v[n - 1]`

Vektor feltöltése számokkal

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     int n = 10;
7     double *dyn_v;
8
9     dyn_v = (double *)malloc(n * sizeof(double));
10
11     for (int i = 0; i < n; i++) {
12         dyn_v[i] = i;
13     }
14
15     for (int i = 0; i < n; i++) {
16         printf("%f\n", dyn_v[i]);
17     }
18
19     free(dyn_v);
20
21     return 0;
22 }
```

Egy egész tömböt foglaltunk

- elemek elérése: `dyn_v[i]`
- legelső: `dyn_v[0]`
- legutolsó: `dyn_v[n - 1]`

A végén a memóriát felszabadítani!

- a `free` függvény meghívásával

Probléma:

- olyan függvényt szeretnénk írni, ami memóriát allokal egy vektornak

Vektornak memóriát allokáló függvény

Probléma:

- olyan függvényt szeretnénk írni, ami memóriát allokál egy vektornak

Egy lehetséges megoldás: visszatérési értéként egy memóriacímet adó függvény

```
1
2  double *alloc_vec(int n) {
3      double *v = (double *)malloc(n * sizeof(double));
4      return v;
5  }
6
7  int main()
8  {
9      int n = 25;
10     double *v = alloc_vec(n);
11     // additional steps here
12     free(v);
13     return 0;
14 }
```

Összegezve:

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 double *alloc_vec(int n) {
5     double *v = (double*)malloc(n * sizeof(double));
6     if (v == 0) {
7         printf("Not enough memory.\n");
8         exit(-1);
9     }
10    return v;
11 }
12
13 void load_vec(FILE* f, double *v, int n) {
14     for (int i = 0; i < n; i++) {
15         fscanf(f, "%lf", v + i);
16     }
17 }
18
19 void print_vec(double *v, int n) {
20     for (int i = 0; i < n; i++) {
21         printf("%f\n", *(v + i));
22     }
23 }
24
25 int main(int argc, char* argv[])
26 {
27     if (argc < 3) {
28         printf("Not enough arguments.");
29         exit(-1);
30     }
31
32     int n = atoi(argv[1]);
33     double *v = alloc_vec(n);
34
35     FILE* f = fopen(argv[2], "r");
36     if (f == 0) {
37         printf("Cannot open file %s.", argv[1]);
38         exit(-1);
39     }
40     load_vec(f, v, n);
41     fclose(f);
42
43     print_vec(v, n);
44
45     free(v);
46     return 0;
47 }
```

Probléma:

- olyan függvényt szeretnénk írni, ami memóriát allokál egy vektornak
- a lefoglalt memória címét bele kellene tenni egy átadott pointerbe

Probléma:

- olyan függvényt szeretnénk írni, ami memóriát allokal egy vektornak
- a lefoglalt memória címét bele kellene tenni egy átadott pointerbe

Emlékezzünk a Változó átadása egy függvénynek cím szerint fólíára:

```
1 void increment(int a, int *b) {
2     *b = a + 5;
3 }
4
5 int main() {
6     int a = 25;
7     int b;
8     increment(a, &b);
9     return 0;
10 }
```

Az előbbiek alapján egy másik lehetséges megoldás egy vektornak memóriát allokaló függvényre:

```
1 void alloc_vec(int n, double **v) {
2     *v = malloc(n * sizeof(double));
3 }
4
5 int main()
6 {
7     int n = 25;
8     double *v;
9     alloc_vec(n, &v);
10    // additional steps here
11    free(v);
12    return 0;
13 }
```

- A `double **v` egy mutató mutatója (pointer to pointer) deklaráció
- a függvény egyik paraméterében kapjuk vissza a lefoglalt memória címét