Mutatók használata 2.

Kormányos Andor

Komplex Rendszerek Fizikája Tanszék

2020. szeptember 21.

Előkészület: void * általános pointer

A void * a pointerek egy univerzális típusa.

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

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

Előkészület: void * általános pointer

A void * a pointerek egy univerzális típusa.

- olyankor használjuk, amikor a mutatott adat típusa mindegy
- például memóriafoglaláskor
- 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

Dinamikus helyfoglalású vektorok

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

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 }
```

Mutató deklarálása

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

Vektor feltöltése számokkal

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

int main()
{
    int n = 10;
    double *dyn_v;

    dyn_v = (double *)malloc(n * sizeof(double));

    for (int i = 0; i < n; i ++) {
        dyn_v[i] = i;
    }

    for (int i = 0; i < n; i ++) {
        printf("%f\n", dyn_v[i]);
    }

    free(dyn_v);

    return 0;
}
```

Mutató deklarálása

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

Memóriafoglalás: `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

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]`

Vektor feltöltése számokkal

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

int main()
{
    int n = 10;
    double *dyn_v;

    dyn_v = (double *)malloc(n * sizeof(double));

    for (int i = 0; i < n; i ++) {
        dyn_v[i] = i;
    }

    for (int i = 0; i < n; i ++) {
        printf("%f\n", dyn_v[i]);
    }

    free(dyn_v);

    return 0;
}
```

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

Mutató mutatója

Probléma:

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

Mutató mutatója

Probléma:

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

Emlékezzünk a `Változó átadása egy függvénynek cím szerint` fóliára:

```
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  }
```

Mutató mutatója

Az előbbiek alapján egy lehetséges megoldás egy vektornak memóriát allokáló 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

Vektornak memóriát allokáló függvény

Egy másik lehetséges megoldás a vektornak memóriát allokáló függvényre: visszatérési értékként 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  }
```

Vektor beolvasása fájlból

Összegezve:

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

double *alloc_vec(int n) {
  double *v = (double*)malloc(n * sizeof(double));
  if (v == 0) {
    printf("Not enough memory.\n");
    exit(-1);
  }
  return v;
}

void load_vec(FILE* f, double *v, int n) {
  for (int i = 0; i < n; i ++) {
    fscanf(f, "%lf", v + i);
  }
}

void print_vec(double *v, int n) {
  for (int i = 0; i < n; i ++) {
    printf("%f\n", *(v + i));
  }
}
```

```c
int main(int argc, char* argv[])
{
  if (argc < 3) {
    printf("Not enough arguments.");
    exit(-1);
  }

  int n = atoi(argv[1]);
  double *v = alloc_vec(n);

  FILE* f = fopen(argv[2], "r");
  if (f == 0) {
    printf("Cannot open file %s.", argv[1]);
    exit(-1);
  }
  load_vec(f, v, n);
  fclose(f);

  print_vec(v, n);

  free(v);
  return 0;
}
```