Dinamikus helyfoglalás vektoroknak

Kormányos Andor

Komplex Rendszerek Fizikája Tanszék

2022. szeptember 27.

Dinamikus helyfoglalású vektorok

- a `double szam[meret]` utasítással létrehozott tömbök a memória ún **stack** területén jönnek létre
- ennek a memóriaterületnek a mérete nem túl nagy, be tud telni
- ha több memóriára van szükségünk (nagy tömb), akkor a memóriát kell foglalni az ún **heap** memóriaterületről

Dinamikus helyfoglalású vektorok

- a `double szam[meret]` utasítással létrehozott tömbök a memória ún **stack** területén jönnek létre
- ennek a memóriaterületnek a mérete nem túl nagy, be tud telni
- ha több memóriára van szükségünk (nagy tömb), akkor a memóriát kell foglalni az ún **heap** memóriaterületről

- memóriafoglalás: `malloc`, `calloc` és `realloc` függvényekkel
- a lefoglalt terület kezdetére **mutatókkal** (**pointer**) hivatkozunk
- miután elvégeztük a feladatot, a memóriaterületet felszabadítjuk

Előkészület: void * általános pointer

A void * a pointerek egy univerzális típusa.

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

```
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

```
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

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

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

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

- `*dyn_v` 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 vektor számára elég memóriát 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 vektor számára elég memóriát foglaltunk

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

A végén a memóriát felszabadítani!

- a `free` függvény meghívásával

Vektornak memóriát allokáló függvény

Probléma:

- olyan függvényt szeretnénk írni, ami memóriát allokál egy vektornak

Vektornak memóriát allokáló függvény

Probléma:

- olyan függvényt szeretnénk írni, ami memóriát allokál egy vektornak

Egy lehetséges megoldás: 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  }
```