Tartalomjegyzék

Feladatok

Minden feladathoz nyiss új projektet IDE-ben vagy írd új file-ba ha parancssorból dolgozol!

Mostantól .cpp kiterjesztésû file-okban dolgozzunk!

Súlypont (3. gyakról)

Írjunk C++ programot, mely beolvas a felhasználótól tetszõleges számú, de maximum 10 három dimenziós koordinátát és kiszámolja a súlypontjukat (koordináták átlaga). A beolvasás megáll, ha a (0,0,0) pontot adjuk meg (és ez a pont nem lesz része a számolásnak).

Tipp: nem is kell semmi bonyolult struktúra, hisz elég a koordinátákat egyenként átlagolni. Elég ha a 3 koordinátához létrehozunk 1-1 float tömböt.

Ha kész vagyunk akkor módosítsuk úgy a kódot, hogy elõször olvassuk be pontosan mennyi koordináta érkezik, majd kérjünk be annyit és számoljuk ki a súlypontjukat. Ezesetben nem kell a (0,0,0)-nak speciálisnak lennie. Példa bemenet:

```
3
3.2 -1.2 2.2
3.4 5.7 1.8
4.2 5.0 0.2
```

Ha ezzel is megvagyunk, gondoljuk végig, hogy egyáltalán le kell menteni a koordinátákat a feladat megoldásához? Meg lehet oldani úgy, hogy nem tároljuk le õket tömbben?

String szelet

Írjunk C++ függvényt, ami kap egy C string-et és két nemnegatív egészet. A függvény adja vissza a két egész szám, mint index, közötti rész string-et dinamikusan foglalt tömbként. Például a

"http://wiki.math.bme.hudenever"http://wiki.math.bme.hu, 2, 5 bemenetre a "http://wiki.math.bme.huneve"http://wiki.math.bme.hu string-et adja vissza.

Ne felejtsük el a **main** függvényben felszabadítani ezt a tömböt!

Átlagnál kisebb

Írjunk függvényt, ami kap egy float tömböt (és hosszát), kiszámolja az elemek átlagát, majd visszaadja azon elemek tömbjét amik kisebbek az átlagnál.

Nem kellene még valamit visszaadni? (Ha igen akkor adjuk vissza paraméterként kapott pointeren keresztül.)

Láncolt lista

Az elõadásról idézzük fel a láncolt lista megvalósítását, fordítsuk is le:

```
#include<iostream>

using namespace std;

struct list_e {
  int num;
  struct list_e *next;
};

void append(struct list_e **start, int n) {
  struct list_e *e = new struct list_e;
  e->num = n;
  e->next = NULL;
  if (*start == NULL) {
    *start = e;
  } else {
    struct list_e *p = NULL;
    for(p = *start; p->next != NULL; p = p->next){}
    p->next = e;
  }
}

int main(void) {
  struct list_e *start = NULL;
  append(&start, 1);
  append(&start, 5);
  append(&start, -2);
  append(&start, 15);
  for(struct list_e *e = start; e != NULL; e = e->next) {
    cout << e->num << endl;
  }
  return 0;
}
```

Láncolt lista index

Írjunk függvényt, ami megkap egy láncolt lista elsõ elemére mutató pointert és egy **n** nemnegatív egészet. A függvény adja vissza az **n**-edik lista elemben tárolt értéket (0-tól számolva). Nem kell figyelnünk rá, hogy jó indexet kapunk-e, feltételezhetjük, hogy létezni fog annyiadik elem.

Láncolt lista felszabadítása

Írjunk függvényt ami felszabadítja a kapott láncolt listát. (Elsõ elem pointerét kapja meg.) Tipp: Egy adott elemet már nyugodtan felszabadíthatunk, ha lementettük a rákövetkezõ elem pointerét.

Láncolt lista beszúrás

Írjunk függvényt ami képes beszúrni adott index helyére elemet. Például ha adott az 1, 5, 2-t tartalmazó lista és beszúrjuk a 0. pozícióra a 6-ot, akkor a 6, 1, 5, 2-t kapnánk, ha a 2-es pozícióra akkor az 1, 5, 6, 2-t.

Láncolt lista eleme-e

Írjunk függvényt ami el tudja dönteni, hogy a kapott érték benne van-e a kapott láncolt listában. (Most már C++-ban használhatunk **bool** típusú változókat, ezeknek **true** vagy **false** értéke van.)

Két irányú láncolt lista

Implementáljunk két irányú láncolt listát, amiben minden elemben nem csak a rákövetkezõ van tárolva, hanem az elõzõ is. A start elem elõzõje NULL legyen.

Implementáljuk legalább az **append** függvényt.