Deklaratív programozás, 1. gyakorlat
Cékla programozás: deKLAratív CÉ++
2011. 09. 16.

Írjon olyan Cékla-függvényt, amely megfelel az adott fejkommentnek. Bármely feladat megoldásához felhasználhat korábbi sorszámú feladatokban definiált eljárásokat.

Ha külön nem kérjük, akkor nem szükséges, hogy jobbrekurzív (iteratív) függvényeket írjon, de természetesen a jobbrekurzív változatot ilyenkor is elfogadjuk.

Hibakezeléssel nem kell foglalkoznia, azaz a megírt függvényeknek csak értelmes argumentumértékek esetén kell az előírt módon viselkedniük. (Pl. egy lista utolsó elemét előállító függvény üres lista esetén, vagy egy lista 5. elemét elővevő függvény egy 3-elemű lista esetén bármilyen módon viselkedhet.)

Az áttekinthetőség kedvéért az egészlistákat nem Cékla nyelven, hanem rövidített jelöléssel adjuk meg: [E1,E2,...,En]. Például [1,3,5] jelentése: cons(1, cons(3, cons(5, nil))).

A Céklában – a C / C++-hoz hasonlóan – a logikai hamis értéknek a 0 felel meg, egyéb szám igazat jelent.

---

1. Csupa 0 és/vagy 1 számjegy

```
// csupa01(N) igaz, ha az N nemnegatív egész szám decimális alakja
// kizárólag a 0 és/vagy 1 számjegyekből áll.

csupa01(100) == 1;
csupa01(2011) == 0;
```

---

2. Osztók száma

Nem kell "okos", azaz az N prímfelbontásán alapuló megoldást adnia, elegendő megszámolnia, hogy az 1..N intervallumban hány osztója van az N számnak.

```
// osztok(N) az N osztóinak száma.

osztok(12) == 6;   // 1,2,3,4,6 és 12
```

---

3. Legnagyobb közös osztó -- naív megoldás

Elegendő "naív" megoldást adnia: pl. A-tól elindulva egyesével csökkenő számokkal osztjuk A-t és B-t; az első közös osztó lesz az eredmény.

```
// lnko(A, B) az A és B szám legnagyobb közös osztója.

lnko(6, 11) == 1;
```

---

4. Legnagyobb közös osztó -- euklideszi algoritmus

Írja át az alábbi, az euklideszi algoritmust imperatív módon megvalósító függvényt Cékla-függvénnyé!

```
// lnko2(A, B) az A és B szám legnagyobb közös osztója.
int lnko2(int A, int B)  // imperatív C nyelven
    while (B != 0) {
        int T = B;
        B = A % B;
        A = T;
    }
    return A;
}
```

---

5. Lista hossza

Egy lista elemeinek számát a lista hosszának nevezzük.

```
// length(L) az L egészlista hossza.

length([1,3,5]) == 3;
```

5.* Lista hossza – jobbrekurzív változat

Írjon iteratív (jobbrekurzív) függvényt az alábbi feladat megoldására!

```
// lengthi(L) az L egészlista hossza.

lengthi([1,3,5]) == 3;
```

---

6. Számlista minden elemének növelése

```
// lista_noveltje(L) az L egészlistának olyan másolata, amelynek
// ugyanannyi eleme van, mint az L-nek, de minden eleme pontosan
// eggyel nagyobb értékű, mint az L megfelelő eleme.

lista_noveltje([1,5,2]) == [2,6,3];
```

---

7. Lista utolsó elemének meghatározása

```
// last(L) az L nemüres egészlista utolsó eleme.

last([5,1,2,8,7]) == 7;
```

---

8. Beszúrás listába adott helyre

```
// Az insert_nth(L, N, E) lista az L lista olyan másolata, amelyben az
// L lista N-edik és (N+1)-edik eleme közé be van szúrva az E elem (a
// lista számozása 1-től kezdődik).

insert_nth([1,8,3,5], 2, 6) == [1,8,6,3,5];
insert_nth([1,3,8,5], 3, 3) == [1,3,8,3,5];
```

---

9. Lista adott sorszámú eleme

```
// nth1(L, N) az L lista N-edik eleme (a lista számozása 1-től kezdődik).

nth1([10,20,30], 3) == 30;
```

---

10. Lista adott hosszúságú prefixuma

Egy N elemű L lista prefixumának nevezzük a H hosszú P listát, ha a P az L első H eleméből áll (a sorrend megtartásával), ahol 0 <= H <= N.

```
// take(L, H) az L lista H hosszú prefixuma.

take([10,20,30,40,50], 3) == [10,20,30];
```

---

11. Lista adott helyen kezdődő szuffixuma

Egy N elemű L lista szuffixumának nevezzük az N-B hosszú S listát, ha az S az L első B eleme utáni elemekből áll (a sorrend megtartásával), ahol 0 =< B =< N.

```
// drop(L, B) az L lista olyan szuffixuma, amely az L első B elemét
// nem tartalmazza.

drop([10,20,30,40,50], 3) == [40,50];
```

---

12. Részlista képzése

```
// sublist(L, H, B) az L lista olyan H hosszúságú (folytonos)
// részlistája, amely előtt B számú elem áll L-ben.

sublist([10,20,30,40,50], 3, 1) == [20,30,40];
```

```
13. Listában párosával előforduló elemek listája

    // parban(L) az L lista összes olyan elemét tartalmazó lista, amelyet
    // vele azonos értékű elem követ. Az eredménylistában az elemek
    // sorrendje legyen ugyanaz, mint a bemenő listában!

    parban([1,1,1,2,3,3,1,2,5,5,4,4]) == [1,1,3,5,4];
    parban("aaabccabeedd") == "aaced";
-------------------------------------------------------------------------------
13.* Listában párosával előforduló elemek listája – jobbrekurzív változat

    Írjon iteratív (jobbrekurzív) függvényt az alábbi feladat megoldására!

    // parbani(L) az L lista összes olyan elemét tartalmazó lista, amelyet
    // vele azonos értékű elem követ.
    // Az eredménylista elemeinek sorrendje tetszőleges lehet, pl:

    parbani([1,1,1,2,3,3,1,2,5,5,4,4]) == [4,5,3,1,1];
    parbani("aaabccabeedd") == "decaa";
-------------------------------------------------------------------------------
14. A 6. feladat újbóli megoldása a map függvénnyel

    // map(F,L) az L=[E1,E2,...En] lista elemeinek F-transzformáltjaiból
    // álló lista, vagyis [F(E1),F(E2),...,F(En)].
    // Egy E érték F-transzformáltjának az F(E) függvényalkalmazás
    // értékét nevezzük.

    Írjon nemrekurzív függvényt lista_noveltje2(L) néven a 6. feladat
    megoldására, a map magasabb rendű függvény felhasználásával!
-------------------------------------------------------------------------------
15. Az 5. feladat újbóli megoldása a foldl függvénnyel

    // foldl(F, A, L) eredménye az L lista elemeire balról jobbra haladva
    // alkalmazott kétargumentumú F függvény eredménye, ahol F első
    // argumentuma a lista soron következő eleme, második argumentuma pedig
    // az előző F függvényalkalmazás eredménye (illetve a lista első eleme
    // esetén A), vagyis
    // foldl(F, A, [E1,E2,E3,...,En]) == F(En, ..., F(E2, F(E1, A))...)

    Példa:
    int rdiv(const int A, const int B) {  return B/A; }
    foldl(rdiv, 64, [4,2,1]) ===> ((64/4)/2)/1 ===> 8;

    Írjon nemrekurzív függvényt length2(L) néven az 5. feladat
    megoldására, a foldl magasabb rendű függvény felhasználásával!
-------------------------------------------------------------------------------
16. A 7. feladat újbóli megoldása a foldl függvénnyel

    Írjon nemrekurzív függvényt last2(L) néven a 7. feladat megoldására a
    foldl magasabb rendű függvény felhasználásával!
-------------------------------------------------------------------------------
17. Lista elemeinek összege a foldl függvénnyel

    Az alábbi függvényt nemrekurzív módon, a foldl felhasználásával írja
    meg!

    // sum(L) az L lista elemeinek összege.

    sum([10,20,30]) == 60;
-------------------------------------------------------------------------------
18. Lista elemeinek négyzetösszege a foldl függvénnyel

    Az alábbi függvényt nemrekurzív módon, a foldl felhasználásával írja
    meg!

    // sumsq(L) az L lista elemeinek négyzetösszege.

    sumsq([10,20,30]) == 1400;
-------------------------------------------------------------------------------
19. A 18. feladat megoldása a map és sum függvényekkel

    Írjon nemrekurzív függvényt sumsq2(L) néven a 18. feladat megoldására
    a  sum és map függvények felhasználásával!
-------------------------------------------------------------------------------
20. Az előadáson bemutatott alábbi függvények közül melyikre emlékeztet
    leginkább a foldl függvény: map, filter, append, revapp, nrev? Miben?

    Tipp: ha a foldl függvényt nem Cékla nyelven írjuk, hanem C++-ban
    függvénysablonként (template), és az F függvényként a cons-t adjuk
    át, akkor milyen eredményt kapunk?

    template <class Elem, class Fun>
    Elem foldl(const Fun F, const Elem A, const list L) { ...

    const list L1 = "abc", L2 = "123";
    foldl(cons, L1, L2) == ?
-------------------------------------------------------------------------------
                    TOVÁBBI GYAKORLÓ FELADATOK OTTHONRA
-------------------------------------------------------------------------------
+1. Lista értékkészletének mérete
    // tavolsag(L) az L lista elemeinek legnagyobb távolsága, azaz a
    // legnagyobb és a legkisebb elem különbsége
    tavolsag("alma") == 12;
+2. Listában párosával előforduló részlisták előfordulása
    // dadogo_e(L) igaz, ha az L listának van olyan nemüres, folytonos
    // részlistája, amelyet vele azonos értékű részlista követ.
    dadogo_e([1,2,3,4,2,3,4]) == 1; // [2,3,4]
    dadogo_e([1,2,3,3,2,3,4]) == 1; // [3]
    dadogo_e([1,2,3,5,2,3,4]) == 0; // nincs ismétlődő lista
+3. Beszúrás rendezett listába
    // Az insert_ord(L) szigorúan monoton növekvő egészlista az L szigorúan
    // monoton növekvő egészlistának az E egésszel bővített változata,
    // feltéve, hogy E nem eleme az L listának; egyébként L.
    insert_ord([1,3,5,8], 6) == [1,3,5,6,8];
    insert_ord([1,3,5,8], 3) == [1,3,5,8];
+4. Lista első monoton növekvő részlistája (futama)
    // futam(L) az L lista első monoton növekvő futama.
    futam([1,2,2,3,2,4,5,6,6,6,7,6,8,2,3,3,4,5,6,0,1]) == [1,2,2,3];
+5. Polinom behelyettesítési értéke
    // polinom(L, X) annak a polinomnak az értéke az X helyen, amelynek az
    // együtthatói az L lista elemei. Tegyük fel, hogy szabadon dönthet
    // arról, hogy a listában milyen sorrendben tárolja az együtthatókat
    // (pl. a legmagasabb kitevőhöz tartozó együtthatót tárolhatja a lista
    // első helyén, vagy az utolsó helyen).
    // Hogyan érdemes tárolni a listában az együtthatókat?
    // Mi a megoldás, ha fordítva tároljuk az együtthatókat?
    polinom([1,2,1], 10) == 121;
+6. Melyik tanult függvényre hasonlít a legjobban a foldr függvény? Miben?
+7. Egyesével növekvő számsorozat generálása
    // seq(A,B) az [A,A+1,...,B-1,B] lista.
    seq(3,6) == [3,4,5,6];
+8. Faktoriális számítása az seq és a foldl függvény felhasználásával
    fact(5) == 120;
+9. Részlista keresése
    // search(L, S) az első olyan pozíció az L listában, ahol az S részlista
    // kezdődik; ha nincs ilyen, akkor 0. (A lista számozása 1-től indul.)
    search("alma", "a") == 1;
    search("alma", "lm") == 2;
    search("alma", "lmx") == 0;
+10. Szám visszaállítása számjegyek listájából a foldl függvénnyel
    // list2num(L) az L lista elemeiből képzett decimális szám.
    // Feltételezheti, hogy az L minden eleme 0 és 9 közé esik.
    list2num([1,5,2]) == 152;
+11. Szám számjegyeinek listája 10-es számrendszerben
    // szamjegyek(N) az N számjegyeinek listája 10-es számrendszerben.
    szamjegyek(200) == [2,0,0];
    szamjegyek(0) == [0];
    Javasolt segédfüggvény: szamjegyek(N, L) az N számjegyeinek listája
    10-es számrendszerben az L lista elé fűzve.
--------- $Date: 2011-09-16 17:59:33 +0200 (p, 16 szept 2011) $ ----------
```