

Structuri de Date

Laboratorul 3: Liste dublu-înlănțuite

Dan Novischi

5 martie 2018

1. Introducere

Scopul acestui laborator îl reprezintă lucrul cu listele dublu înlănțuite neordonate si are doua cerințe care urmăresc:

- definirea si implementarea unei interfețe de lucru;
- rezolvarea unei probleme simple cu ajutorul acestei interfețe.

2. Liste dublu înlănțuite

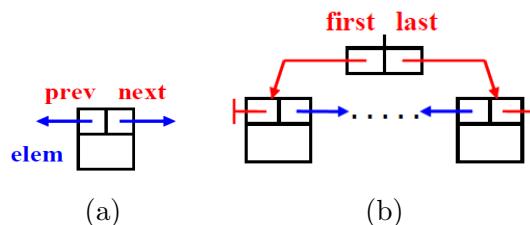


Figura 1: Liste dublu înlănțuite: (a) - un nod al listei; (b) - reprezentare.

Analog listelor simplu înlănțuite si in cazul celor duble se alocă individual memorie pentru fiecare nod (sau celula). Astfel, pentru un nod al listei alături de fiecare element (**elem**) se rețin doua legaturi (pointeri) către adresele de memorie la care se găsesc atat nodul urmator (**next**) cat si cel anterior (**prev**) (vezi Figura 1a). Mai mult, pentru o reprezentare facila putem defini reprezentarea generica a listei ca fiind o structura cu doua legaturi: **first** – legatura la primul nod din lista si **last** – legatura la ultimul nod din lista (vezi Figura 1b). In limbajul C avem urmatoarele definitii pentru un nod si lista propriu-zisa:

```
1 typedef struct ListNode{
2     Item elem;
3     struct ListNode* next;
4     struct ListNode* prev;
5 } ListNode;
```

```
1 typedef struct List{
2     ListNode* first;
3     ListNode* last;
4 }List;
```

3. Cerințe

În acest laborator dispuneți de mai multe fișiere inclusiv scheletul de cod după cum urmează:

- `DoubleLinkedList.h` – interfața generică a listei dublu înlanțuite, care trebuie implementată conform cerințelor de mai jos.
- `example.c` – un exemplu de program simplu care utilizează interfața cu elemente de tip caracter introduse de la tastatură.
- `testList.c` – checker pentru validarea implementării interfeței pentru liste dublu înlanțuite.
- `problem.c` – aplicația pentru problema din cerințele de mai jos.
- `input` – fișier ce conține input-ul pentru problema în format text.
- `Makefile` – fișierul pe baza căruia se vor compila și rula testele (interfața și problemele).

Pentru compilarea tuturor aplicațiilor folosiți comanda `"make build"`. Aceasta are următorul output pentru un program fără erori de sintaxă sau warning-uri:

```
$ make build
gcc -std=c9x -g -O0 problem.c -o problem
gcc -std=c9x -g -O0 example.c -o example
gcc -std=c9x -g -O0 testList.c -o testList
```

Iar pentru ștergerea automată a fișierelor generate prin compilare folosiți comanda `"make clean"`:

```
$ make clean
rm -f problem example testList
```

Cerința 1 (8p) În fișierul `DoubleLinkedList.h` implementați funcțiile de interfață ale listei dublu înlanțuite urmărind atât indicațiile/prototipurile din platformă/schelet cât și ordinea de mai jos:

- a) `createList` – creează o nouă listă vidă prin alocare dinamică.
- b) `isEmpty` – verifică dacă o listă este sau nu goală.
- c) `contains` – verifică existența unui anumit element (`Item elem`) în listă.
- d) `insertAt` – introduce un nou element în listă la o poziție dată ca parametru dacă acesta există. Numerotarea pozițiilor din listă începe cu numărul zero.
- e) `deleteOnce` – șterge prima apariție a unui element din listă dacă acesta există.
- f) `length` – calculează lungimea unei liste.
- g) `destroyList` – distruge o listă de-alocând memoria utilizată de aceasta.

Pentru testare puteti modifica aplicatia din `example.c` dupa bunul plac. In schimb pentru validarea corectitudinii implementarii folositi comanda `"make test"`. In cazul unei implementari corecte a interfetei acesta genereza urmatorul output:

```
$ make test
valgrind --leak-check=full ./testList
==4787== Memcheck, a memory error detector
==4787== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==4787== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==4787== Command: ./testList
==4787==
. Testul Create a fost trecut cu succes! Puncte: 0.05
. Testul IsEmpty a fost trecut cu succes! Puncte: 0.05
. Testul Contains a fost trecut cu succes! Puncte: 0.10
. Testul Insert a fost trecut cu succes! Puncte: 0.20
. Testul DeleteOnce a fost trecut cu succes! Puncte: 0.20
. Testul Length a fost trecut cu succes! Puncte: 0.10
. *Destroy se va verifica cu valgrind* Puncte: 0.10.

Scor total: 0.80 / 0.80

==4787==
==4787== HEAP SUMMARY:
==4787==   in use at exit: 0 bytes in 0 blocks
==4787==   total heap usage: 10 allocs, 10 frees, 1,232 bytes allocated
==4787==
==4787== All heap blocks were freed -- no leaks are possible
==4787==
==4787== For counts of detected and suppressed errors, rerun with: -v
==4787== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Cerinta 2 (2p) In fisierul `problem.c` implementati functia `isPalindrome` care determina daca o lista dublu inlantuita de caractere formeaza (sau nu) un palindrom. Implementarea acestei functii va folosi un singur ciclu pentru parcurgerea listei. NU este permisa utilizarea unor liste auxiliare, array-uri sau orice alta forma care utilizeaza memorie aditionala. Pentru validarea solutiei puteti utiliza comanda `"make test-problem"` care are urmatorul output pentru o implementarea corecta:

```
$ make test-problem
valgrind --leak-check=full ./problem
==5179== Memcheck, a memory error detector
==5179== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==5179== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==5179== Command: ./problem
==5179==
Input1: "abcde" --- List: [a, b, c, d, e] is NOT a palindrome.
Input2: "noon" --- List: [n, o, o, n] is a palindrome.
Input3: "radar" --- List: [r, a, d, a, r] is a palindrome.
Input4: "proffesor" --- List: [p, r, o, f, f, e, s, o, r] is NOT a palindrome.
Input5: "level" --- List: [l, e, v, e, l] is a palindrome.
Input6: "student" --- List: [s, t, u, d, e, n, t] is NOT a palindrome.
Input7: "racecar" --- List: [r, a, c, e, c, a, r] is a palindrome.
Input8: "data-structures" --- List: [d, a, t, a, -, s, t, r, u, c, t, u, r, e, s] is NOT a palindrome.
==5179==
==5179== HEAP SUMMARY:
==5179==   in use at exit: 0 bytes in 0 blocks
==5179==   total heap usage: 68 allocs, 68 frees, 7,168 bytes allocated
==5179==
==5179== All heap blocks were freed -- no leaks are possible
==5179==
==5179== For counts of detected and suppressed errors, rerun with: -v
==5179== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```