Structuri de Date
Laboratorul 3: Liste dublu-înlănțuite

Dan Novischi

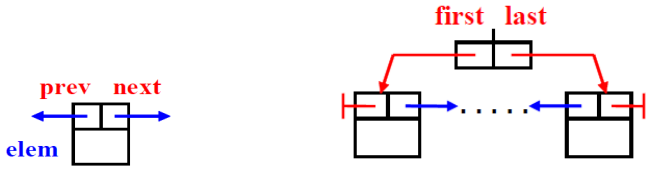
2 martie 2019

1. Introducere

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

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

2. Liste dublu înlănțuite



(a) (b)

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). Într-un nod al listei, alături de fiecare element (`elem`), se rețin două legaturi (pointeri) către adresaele de memorie la care se găsesc atât nodul următor (`next`) cât și cel anterior (`prev`) (vezi Figura 1a). Mai mult, pentru o reprezentare facilă putem defini lista gererică ca fiind o structură cu două legaturi: `first` – legatura la primul nod din listă și `last` – legatura la ultimul nod din listă (vezi Figura 1b). În limbajul `C` avem următoarele definiții pentru un nod și lista propriu-zisă:

```
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

In acest laborator dispuneti de mai multe fisiere inclusiv scheletul de cod dupa cum urmeaza:

- `DoubleLinkedList.h` – interfata generica a listei dublu înlănțuite, care trebuie implementata conform cerințelelor de mai jos.
- `example.c` – un exemplu de program simplu care utlizeaza interfata cu elemente de tip caracter introduse de la tastatura.
- `testList.c` – checker pentru validarea implementarii interfetei pentru liste dublu înlănțuite.
- `problem.c` – apilicatia pentru problema din cerintele de mai jos.
- `input` – fisier ce contine input-ul pentru problema in format text.
- `Makefile` – fiserul pe baza caruia se vor compila si rula testele (interfata si probleme).

Pentru compilara tuturor aplicatiilor folositi comanda "`make build`". Aceasta are urmatorul output pentru un program fara erori de sintaxa 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 stergerea automata a fiserelor generate prin compilare folositi comanda "`make clean`":

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

Cerința 1 (8p) In fisierul `DoubleLinkedList.h` implementati functiile de interfata ale listei dublu inaltuite urmarind atat indicatiile/prototipurile din platforma/schelet cat si ordinea de mai jos:

a) `createList` – creeaza o noua lista vida prin alocare dinamica.

b) `isEmpty` – verifica daca o lista este sau nu goala.

c) `contains` – verifica existenta unui anumit element (`Item elem`) in lista.

d) `insertAt` – introduce un nou element in lista la o pozitie data ca parametru daca acesta exisista. Numerotarea pozitiilor din lista incepe cu numarul zero.

e) `deleteOnce` – sterge prima aparitie a unui element din lista daca acesta exista.

f) `length` – calculeaza lungimea unei liste.

g) `destroyList` – distruge o lista de-alocand memoria utilizata 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)
```

Cerința 2 (2p) In fiserul `problem.c` implementati functia `isPalindrome` care determina daca o lista dublu inalantuita de caractere formeaza (sau nu) un palindrom. Implementarea acestei functii va folosi un singur ciclcu pentru parcurgerea listei. NU este permisa utilizarea unor liste auxiliare, array-uri sau orice alta forma care folosete memorie aditionala. Pentru validarea solutie puteti utiliza comanda "`make test-problem`" care are urmatorul output pentru o implementare 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 palindorme.
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 palindorme.
Input5: "level" --- List: [l, e, v, e, l] is a palindrome.
Input6: "student" --- List: [s, t, u, d, e, n, t] is NOT a palindorme.
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 palindorme.
==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)
```