Structuri de Date
Laboratorul 3: Liste dublu-înlănțuite

Dan Novischi, Mihai Nan

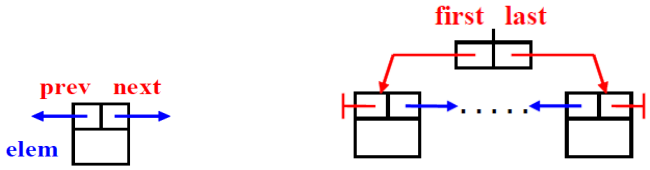
Ultima modificare: Martie 2021

1. Introducere

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

- definirea și implementarea unei interfețe de lucru cu acest tip de structură de date;
- rezolvarea unei probleme simple cu ajutorul acestei interfețe.

2. Liste dublu înlănțuite



(a) (b)

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

Analog listelor simplu înlănțuite, și în cazul celor duble se alocă individual memorie pentru fiecare nod (sau celulă). Într-un nod al listei, alături de fiecare element (`elem`), se reține două legături (pointeri) către adresele 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 generică drept o structură cu două legături: `first` – legătura la primul nod din listă și `last` – legătura la ultimul nod din listă (vezi Figura 1b). În limbajul `C` avem următoarele definiții pentru un nod și lista propriu-zisă:

```c
typedef struct ListNode{
  Item elem;
  struct ListNode* next;
  struct ListNode* prev;
} ListNode;
```

```c
typedef struct List{
  ListNode* first;
  ListNode* last;
}List;
```

3. Cerințe

În acest laborator, scheletul de cod primit este alcătuit din următoarele:

- `DoubleLinkedList.h` – interfața generică a listei dublu înlănțuite, 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 înlănțuite.
- `problem.c` – apilicația 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 aplicațiile).

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

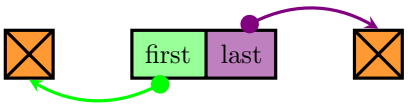
```
$ make build
gcc -std=c9x -g -O0 problem.c -o problem
gcc -std=c9x -g -O0 example.c -o example
gcc -std=c9x -g -O0 testList.c -o testList
```

Pentru ștergerea automată a fișierelor generate prin compilare, folosiți comanda "`make clean`":

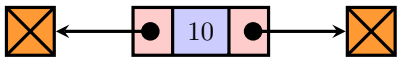
```
$ make clean
rm -f problem example testList
```

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

a) `createList` – creează o noua listă vida prin alocare dinamică.



`createNode` – creează un nod al listei cu o anumită valoare (folosește alocare dinamică) și returnează un pointer la acel nod.



b) `isEmpty` – verifică dacă o listă este sau nu goală. Dacă lista primită ca parametru este vidă, atunci funcția returnează 1. Altfel, funcția întoarce 0.

c) `contains` – verifică existența unui anumit element (`Item elem`) în listă. Dacă elementul se află în listă, funcția returnează 1. Altfel, funcția întoarce 0.

d) `insertAt` – introduce un nou element în listă la o poziție dată ca parametru dacă aceasta există. Numerotarea pozițiilor din listă începe cu numărul zero.

Atenție!

Trebuie să ne asigurăm că actualizăm cele două legături (`first` sau `last`), dacă este cazul.

- `first` – se modifică dacă inserăm la început;
- `last` – se modifică dacă inserăm la final.

e) `deleteOnce` – șterge prima apariție a unui element din listă dacă acesta există.

Atenție!

Trebuie să ne asigurăm că actualizăm cele două legături (`first` sau `last`), dacă este cazul.

- `first` – se modifică dacă ștergem primul nod din listă;
- `last` – se modifică dacă ștergem ultimul nod din listă.

f) `length` – calculează lungimea unei liste.

g) `destroyList` – distruge o listă de-alocând memoria utilizată de aceasta. Returnează lista vidă fără memorie alocată (prin convenție `NULL`).

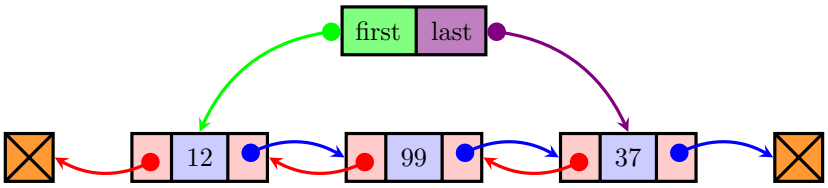


Figura 2: Exemplu de reprezentare grafică a listei dublu înlănțuită

Pentru testare, puteți modifica aplicația din `example.c` după bunul plac. În schimb, pentru validarea corectitudinii implementării, folosiți comanda "`make test`". În cazul unei implementări corecte a interfeței, această comandă generează următorul 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) În fișierul `problem.c` implementați funcția `isPalindrome` care determină dacă o listă dublu înlănțuită de caractere formează (sau nu) un palindrom. Implementarea acestei funcții va folosi un singur ciclu pentru parcurgerea listei. **NU** este permisă utilizarea unor liste auxiliare, array-uri sau orice altă formă care folosește memorie auxiliară. Pentru validarea soluției, puteți utiliza comanda "`make test-problem`" care produce următorul output pentru o implementare corectă:

```
$ 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)
```

Observație

Un **palindrom** este un șir de caractere (de obicei cuvinte, fraze sau numere) care citit de la stânga la dreapta sau de la dreapta la stânga rămâne neschimbat.