COMP9024: Assignment 1

Iterator ADT

(Draft Version)

The following specification may change! A notice on the class web page will be posted after each revision, so please check class notice board frequently.

Change log:

- Nothing as yet!

Objectives

- to give you experience in implementing ADTs in C
- to give you experience with doubly-linked lists
- to give you further practice with C and data structures

Admin

Marks	10 marks
Group?	This assignment is completed **individually**
Due	23:55pm on Friday 04 January 2019.
Late Penalty	2 marks per day off the ceiling (e.g. if you are 2 days late, your maximum possible marks is 6/10) The last day to submit late is 09 January 2019, after this date, you will receive zero mark for this assignment.

Aim

In this assignment, your task is to implement a list Iterator ADT, in the file `listIteratorInt.c`. You need to write wrapper code to test your implementations, and make sure to test a variety of conditions.

Please note that, you need to submit only one file `listIteratorInt.c` , that includes the C code implementing the required list Iterator ADT.

Iterator

An iterator offers an easy way to traverse through a collection, add new elements, delete and modify existing elements. Iterators are widely supported by langauges like C++ and Java. However, C does not offer iterators. In this assignment your task is to implement two list iterators that allow the programmer to traverse the list in either direction, modify the list during iteration, and obtain the iterator's current position in the list.

You will implement list iterators similar to the one available in Java. The following interface specifications are drawn from Java (7) ListIterator API and adapted for C implementations.

A List Iterator has no current element; its cursor position always lies between the element that would be returned by a call to `previous` and the element that would be returned by a call to `next`. An iterator for a list of length n has n+1 possible cursor positions, as illustrated by the carets (^) below:

```
                      Element(0)   Element(1)   Element(2)   ... Element(n-1)
cursor positions:    ^            ^            ^            ^                ^
```

Note that the `delete` and `set` methods are not defined in terms of the cursor position; they are defined to operate on the last element returned by a call to `next` or `previous`.

Integer List iterator

You need to implement a list iterator that can store integer values, and offer the following interface to the programmer. You must use **doubly linked list** to implement your list iterator. You need to implement the following methods in the file `listIteratorInt.c`, and submit this file as described in the section named "Submission".

- Download files: assn1.zip.
- Log from sample tests: sample_tests_log1.txt.

You need to implement the following methods in the file `listIteratorInt.c`. Note that you need to submit ONLY one file `listIteratorInt.c` for this assignment. Please do not modify the header file `listIteratorInt.h`. You can write your test code in `testListIteratorInt.c`. However, you will not submit your test code. We will test your implementation using our test sets, so make sure you do NOT modify function headers (signatures) in the header file `listIteratorInt.h`.

Again, remember that you must use **doubly linked list** to implement your list iterator.

Interface

`IteratorInt IteratorIntNew()`
: Creates a new list iterator that can store integer values.

`int add(IteratorInt it, int v)`
: Inserts the specified value `v` into the list iterator `it`. The element is inserted immediately before the element that would be returned by next(), if any, and after the element that would be returned by previous(), if any. (If the list contains no elements, the new element becomes the sole element on the list.) The new element is inserted before the implicit cursor: a subsequent call to next would be unaffected, and a subsequent call to previous would return the new element.

  Returns 1 if successful, 0 otherwise.

`int hasNext(IteratorInt it)`
: Returns 1 if the given list iterator has more elements when traversing the list in the forward direction, returns 0 otherwise.

`int hasPrevious(IteratorInt it)`
: Returns 1 if the given list iterator has more elements when traversing the list in the reverse direction, returns 0 otherwise.

`int *next(IteratorInt it)`
: Returns the pointer to the next value in the given list iterator and advances the cursor position. This method may be called repeatedly to iterate through the list, or intermixed with calls to `previous` to go back and forth. (Note that alternating calls to next and previous will return the same element repeatedly.)

  The method returns NULL if it has no next value.

`int *previous(IteratorInt it)`
: Returns the pointer to the previous value in the given list iterator and moves the cursor position backwards. This method may be called repeatedly to iterate through the list backwards, or intermixed with calls to `next` to go back and forth. (Note that alternating calls to next and previous will return the same element repeatedly.)

  The method returns NULL if it has no previous value.

`int deleteElm(IteratorInt it)`
: Deletes from the list iterator the last value that was returned by `next` or `previous` or `findNext` or `findPrevious`.

  Precondition: A call to `deleteElm` must be IMMEDIATELY preceded by a successful call to one of `next` or `previous` or `findNext` or `findPrevious`.

  Returns 1 if successful, 0 otherwise (for example, invalid `deleteElm` call).

`int set(IteratorInt it, int v)`

Replaces the last element returned by `next` or `previous` or `findNext` or `findPrevious` with the specified element (v).

Precondition: A call to `set` must be IMMEDIATELY preceded by a successful call to one of `next` or `previous` or `findNext` or `findPrevious`.

Returns 1 if successful, 0 otherwise (for example, invalid `set` call).

`int *findNext(IteratorInt it, int v)`

Returns pointer to the next value in `it` that matches the given value `v` and advances the cursor position **past the value returned.**

The method returns NULL if there is no such next value and the cursor is **not moved from the current position**.

For example:

```
        [ 20 ^ 12 15 5 14 10 5 9 3 ]
findNext(it, 5) will result in
        [ 20 12 15 5 ^ 14 10 5 9 3 ]
findNext(it, 5) will result in
        [ 20 12 15 5 14 10 5 ^ 9 3 ]
```

`int *findPrevious(IteratorInt it, int v)`

Returns pointer to the previous value in `it` that matches the given value `v` and moves the cursor position backwards **before the value returned**.

The method returns NULL if there is no such previous value and the cursor is **not moved from the current position**.

For example:

```
        [ 20 12 15 5 14 10 5 9 ^ 3 ]
findPrevious(it, 5) will result in
        [ 20 12 15 5 14 10 ^ 5 9 3 ]
findPrevious(it, 5) will result in
        [ 20 12 15 ^ 5 14 10 5 9 3 ]
```

`void reset(IteratorInt it)`

Resets `it` to the start of the list.

`void freeIt(IteratorInt it)`

Deletes all the nodes in `it` and frees associated memory.

Assessment Criteria

Your program will be tested thoroughly and objectively. This assignment will be marked out of 10 where 8 marks are for correctness (based on auto marking) and 2 marks are for your solution's complexity, style and comments.

Submission

To be available later ...

Plagiarism

This is an individual assignment. Each student will have to develop their own solution without help from other people. In particular, it is not permitted to exchange code or pseudocode. You are not allowed to use code developed by persons other than yourself. If you have questions about the assignment, ask your tutor Before submitting any work you should read and understand the following very useful guide by the Learning Centre How Not To Plagiarise. All work submitted for assessment must be entirely your own work. We regard unacknowledged copying of material, in whole or part, as an extremely serious offence. For further information, see the Course Information.

-- end --