02393 Programming in C++ Module 10
Linked Lists

Teacher: Alberto Lluch Lafuente

Sebastian Mödersheim (slides author)

April 18, 2016

Lecture Plan

#	Date	Topic	Chapter *
1	1.2	Introduction	1
2	8.2	Basic C++	1
3	15.2	Data Types	2
4	22.2		
		Libraries and Interfaces	3
5	29.2		
6	7.3	Classes and Objects	4.1, 4.2 and 9.1, 9.2
7	14.3	Templates	4.1, 11.1
		Påskesferie	
8	4.4	Inheritance	14.3, 14.4, 14.5
9	11.4	Recursive Programming	5
10	18.4	Linked Lists	10.5
11	25.4	Trees	13
12	2.5	Graphs	16
13	9.5	Summary	
	17.5	Exam	

* Recall that the book uses sometimes ad-hoc libraries that are slightly different with respect to the standard libraries (e.g. strings and vectors).

Recursive Data Structures/Types/Classes

Many data types/structures/classes can be recursively defined:

- A natural number is 0 or a natural number plus one.
- A set can be empty, a singleton or the union of two sets;
- A list can be empty, one item or the concatenation of two lists;
- A tree can be empty, one leaf node, or an internal node with two sub-trees;
- etc.

Linked Lists

Recursive Definition

```
struct Node{
    int content;
    Node *next;
}
```

A Node has some content and points to a Node

A list can be then just an pointer to a `Node`:

- A `nullptr` would represent an empty list;
- Otherwise the pointer points to the first node in the list;

Linear Lists

Live Programming

In a previous class we saw an example of how to implement a...

- vector class based on arrays

Now we are going to see examples on how to implement a...

- vector class based on linked lists
- set class based on linked lists

Note: linked lists are provided by the STL. See http://en.cppreference.com/w/cpp/container/list

Array vs. Lists

Recursive data-structures

```
struct Node{
    int content;
    Node *next;
}
```

	Array	*List*
Iterative Access		
Random Access		
Insert/Delete		
Insert/Delete at end		

1

2

Array vs. Lists

Recursive data-structures

```
struct Node{
    int content;
    Node *next;
}
```

	Array	*List*
Iterative Access	$O(1)$	$O(1)$
Random Access		
Insert/Delete		
Insert/Delete at end		

1

2

Array vs. Lists

Recursive data-structures

```
struct Node{
    int content;
    Node *next;
}
```

	Array	*List*
Iterative Access	$O(1)$	$O(1)$
Random Access	$O(1)$	$O(n)$
Insert/Delete		
Insert/Delete at end		

1

2

Array vs. Lists

Recursive data-structures

```
struct Node{
    int content;
    Node *next;
}
```

	Array	*List*
Iterative Access	$O(1)$	$O(1)$
Random Access	$O(1)$	$O(n)$
Insert/Delete	$O(n)$	$O(1)$[1]
Insert/Delete at end		

[1] given pointer to node before insertion/deletion

[2]

Array vs. Lists

Recursive data-structures

```
struct Node{
    int content;
    Node *next;
}
```

	Array	*List*
Iterative Access	$O(1)$	$O(1)$
Random Access	$O(1)$	$O(n)$
Insert/Delete	$O(n)$	$O(1)$[1]
Insert/Delete at end	$O(1)$[2]	$O(1)$

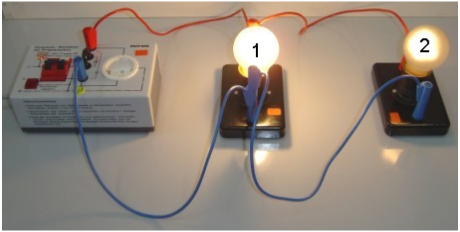
[1] given pointer to node before insertion/deletion

[2] amortized

Doubly-linked Lists

Some annoyances of single-linked lists: delete a pointed element, concatenate two lists, etc.

One possible solution: doubly-linked lists



Implementation	*Insert head*	*Concat*	*Reverse*
Concatenation by connecting the tail of one list with head of other list.	$O(1)$	$O(1)$	$O(N)$

Doubly-Linked Lists

Recursive Definition

```
struct Node{
    int content;
    Node *prev;
    Node *next;
}
```

A Node has some content and points to two Nodes: the previous one and the next one in the list.