Abstract Data Types - Lists

Arrays implementation

Linked-lists implementation

Lecture 15

Jérôme Waldispühhl

School of Computer Science
McGill University

Slides by Mathieu Blanchette
Abstract data types (ADT)

• Definition: Model of a data structure that specifies:
  – The type of data stored
  – The operations supported on that data

• An ADT specifies what can be done with the data, but not how it is done

• It is the implementation of the ADT that specifies how operations are performed

• The user of an ADT does not need to know anything about the implementation.
List ADT

Data stored: a ordered set of objects of any kind

\[ [1, 1, 2, 3, 5, 8] \]

\[ [\ln(), \sin(), f(), \exp()] \]

\[ [🍎, 🍊, 🍊, 🍌] \]
Operations on list ADT

- `getFirst()` : returns the first object of the list
- `getLast()` : returns the last object of the list
- `getNth(n)` : returns the n-th object of the list
- `insertFirst(Object o)` : adds o at the beginning of the list
- `insertLast(Object o)` : adds o at the end of the list
- `insertNth(n, o)` : adds the n-th object of the list by o
- `removeFirst()` : removes the first object of the list
- `removeLast()` : removes the last object of the list
- `removeNth(n)` : removes the n-th object of the list
- `getSize()` : returns the number of objects in the list
- `concatenate(List l)` : appends List l to the end of this list
Implementation of the list ADT With an Array

- An 1D array $L$ to store the elements of the list
- An integer `size` to record the number of objects stored.

```
L = [apple, pineapple, apple, banana, empty]
```

`size = 4`
Implementation of the list ADT With an Array

getFirst() { return L[0] }
getLast() { return L[size-1] }
getNth(n) { return L[n] }

L

size = 4
Implementation of the list ADT With an Array

insertLast(banana) $[\text{apple, pineapple, apple}]$

Size = 3

insertLast( Object o ) { L[size] $\leftarrow$ o; size $\leftarrow$ size + 1 }
Implementation of the list ADT
With an Array

\[
\text{insertNth}( 1, \text{Apple} ) \quad [ \text{Apple}, \text{Pineapple}, \text{Banana}, \varnothing, \varnothing ]
\]

\[
\text{Size} = 3
\]

\[
\text{Size} = 4
\]

insertNth(into n, Object o) {
  for i ← size downto n {
    L[i] ← L[i-1];
  }
  L[n] ← o;
  size ← size +1
}

insertFirst(Object o) {
  insertNth(0,o)
}
Implementation of the list ADT With an Array

removeLast(): size ← size - 1

removeNth(n)
  
  for i ← n to size-1 do L[i] ← L[i+1]
  
  size ← size - 1

removeFirst(): removeNth(0)

Size = 4

Size = 3
Limitations of arrays

• In some situations, an array is not a good way to implement a list, because:
  – Size has to be known in advance
  – Memory required may be much larger than the number of elements actually used
  – Inserting or deleting an element can take time up to $O(n)$

• An array implementation is bad when:
  – the number of objects to be stored in not known in advance
  – the user will need to do a lot of insertions or removals
Linked-list implementation

- Linked-list: Sequence of nodes. Each node stores some data and knows the next node in the list.
- A linked-list is a recursive data structure!

- Node: | Value | Next |
- List: "Go" | "Habs" | "Go!" | null

head

null
tail
public class node {

    private Object value;
    private node next;

    // constructor
    public node(Object x, node n) {
        value = x;
        next = n;
    }

    public node getNext() { return next; }
    public Object getValue() { return value; }
    public void setValue(Object x) { value = x; }
    public void setNext(node n) { next = n; }
}
class linkedList {

    node head, tail;

    // default constructor, builds an empty list
    list() {
        head = null;
        tail = null;
    }

    getFirst() { return head.getValue(); }
    getLast() { return tail.getValue(); }
    getNth() { /* we will do later */ }
}

"Go" | "Habs" | "Go!" | null

head

head

tail
/* Add an object at the tail of the list */
void addLast(Object x) {
    if ( tail == null ) {  // list is empty
        tail = head = new node(x, head);
    } else {
        tail.setNext( new node(x,null) );
        tail = tail.getNext();
    }
}

Example: addLast( “Go!” )
/* Add an object at the head of the list */
void addFirst(Object x) {
    head = new node(x, head);
    if (tail == null) tail = head;
}

Example: addFirst(“Go”)
insertNth(n, Object x) is more complicated…

Why? How to code it?

We will come back on that a bit later…

Example: insertNth( 1, "Habs" )