Linear Collections: Linked Lists
2501ICT/7421ICT
Nathan

René Hexel

School of Information and Communication Technology
Griffith University

Semester 1, 2012
Outline

1. Simple Linked Lists
   - Overview
   - Singly Linked Lists

2. Omnidirectional Lists
   - Doubly Linked Lists
   - Circular Lists
Linked List Implementations
Contents

- Linked Data Structures
- Singly Linked Lists
  - node class implementation
- Doubly Linked Lists
  - node class extension
- Implementations
- Choosing Collection Implementations
The Problems with Arrays were

- Contiguous Memory
- Physically Adjacent Cells
- One-to-One Correspondence between logical position of an item and its Physical position in memory
  - decouple logical and physical position
  - no shifting of items required
Linked Data Structures

- Consist of Nodes
- One Node per item
- Each Node consists of
  - the actual data
  - one or more links to other Nodes
  → link, pointer, and reference are synonymous in this Context
Singly Linked Lists
Singly Linked Lists

- Each Node has 1 Successor:
  - head
  - $D_1$
  - $D_2$
  - $D_3$
  - NULL

- External Head Pointer
  - Points to first Node

- Node Access
  - Traverse the pointers starting from the Head
Changing Items

- To Add or Delete an Entry

- Shuffle Around only Pointers
  - Data remain at their original memory locations
  - Free memory only if no longer referenced
The Cost

- **Linear Searches**
  - Required by (almost) all operations
  - "Intrinsic $O(n)$ overhead"

- **Cache Some Pointers**
  - Reduces some operations to $O(1)$
  - Maintain a second, external tail pointer
  - Keep pointer to "interesting" nodes
  - $\Rightarrow$ Still doesn't help in all cases!
Example (usually a private inner class)

```
@interface SNode: NSObject
{
    SNode *next;                  // pointer to next Node
    id data;                      // data contents
}

- initWithData: d next: (SNode *) n;   // constructor
- data;                               // data getter
- setData: newData;                   // data setter
- (SNode *) next;                     // next node getter
- setNext: (SNode *) newNext;         // next node setter
@end
```
A Node Class Implementation in Objective-C

Example

```objective-c
@implementation SNode

- initWithData: d next: (SNode *) n { // constructor
    return [[[self init] setData: d] setNext: n]; // init and set data/link
}

- (SNode *) next { // next node getter
    return next;
}

- setNext: (SNode *) newNext { // next node setter
    next = newNext;
    return self;
}

- data { // data getter
    return data;
}

- setData: newData { // data setter
    if (data != newData) { // is there a change?
        [data release]; // release old data
    }
    data = [newData retain]; // retain new data
    return self;
}
@end
```

René Hexel  Linear Collections: Linked Lists
A Node Class in C++

Example (usually a *private inner class*)

```cpp
template <class T> class SNode
{
    SNode *nxt;    // pointer to next Node
    T dta;         // data contents

public:
    SNode(T &d, SNode *n = NULL) // constructor
    {
        nxt = n;        // set next node
        dta = d;        // and data
    }

    T data() { return dta; } // data getter

    void setData(T &d) { dta = d; } // data setter

    SNode *next() { return nxt; } // next node getter

    void setNext(SNode<T> *n) // next node setter
    {
        nxt = n;
    }
};
```
Creating the first Node

Example (Objective-C)

```objective-c
SNode *head = [[SNode alloc] initWithData: @"D1" next: nil];
```

Example (C++)

```cpp
string d1("D1");
SNode<string> *head = new SNode<string>(d1);
```
Linking a Second Node

Example (Objective-C)

```objective-c
[head setNext: [[SNode alloc]
    initWithData: @"D2" next: nil]];
```

Example (C++)

```cpp
string d2("D2");
head->setNext(new SNode<string>(d2));
```
Linking a Node to the Head

Example (Objective-C)

```objective-c
head = [[SNode alloc]
    initWithData: @"D0" next: head];
```

Example (C++)

```cpp
string d0("D0");
head = new SNode<string>(d0, head);
```
Traversing a Linked Structure (\textit{while})

```
SNode *ptr = head; // first element
while (ptr != nil) // as long as node is valid
{
    // do something useful with the data:
    printf("ptr: \%s\n", [[[ptr data] description] UTF8String]);
    ptr = [ptr next]; // next element
}
```
Traversing a Linked Structure (for)

Example (using a for loop in Objective-C)

```objective-c
for (SNode *ptr = head; // first element
    ptr != nil; // until finished
    ptr = [ptr next]) // switch to next element
{
    printf("ptr: %s\n", [[[ptr data] description] UTF8String]);
}
```
Traversing a Linked Structure (C++)

Example (using a **for** loop in C++)

```cpp
for (SNode<string> *ptr = head; // first element
     ptr != NULL; // until finished
     ptr = ptr->next()) // switch to next element
{
    cout << "ptr: " << ptr->data() << endl;
}
```
Inserting Anywhere

- Traverse to specific Node \( O(n) \)
- Save where that Node Points to \( O(1) \)
- Let Node point to new Node \( O(1) \)
- Copy saved pointer to new Node (\( \text{nil} \) if at the end) \( O(1) \)

→ Complexity: \( O(n) \)
Delete First Node

- Point Head to second Node, only then free memory!

```c
SNode *oldHead = head;
head = [head next];
[oldHead release];
```

- Complexity: O(1)
Deleting Anywhere

- Traverse to specific node \( O(n) \)
- Let Predecessor of that Node point to the successor of the Node \( O(1) \)
- Free the Node’s memory \( O(1) \)

\[ \text{Complexity: } O(n) \]
Caveats

- **NULL Pointer Exceptions**

- Whenever you use a pointer w/o checking, e.g.
  
  ```
  head = head->next->next;
  
  head = [SNode new];
  head->data = data;
  ```

- Results in exceptions or program crashes!
  
  ⇒ always check pointers!

- **nil method invocations are safe in Objective-C!**
  
  ⇒ `[head next] better than head->next`
  
  ⇒ safer to use access methods than directly using instance variables!
Pros and Cons of Singly Linked Lists

+ First Item Insertion/Removal: $O(1)$
+ Dynamic Memory Allocation
+ No Resizing Overhead
− Memory-Overhead for Pointers
− Most Operations are $O(n)$
− Costly Binary Search $O(n \log n)$

⇒ binary search more expensive than linear search
Doubly Linked Lists
Doubly Linked Lists

- Add Link to Predecessor Node:

  - Traversal in both directions
  - Easy moving from a given Node to both its successor (next node) and predecessor (previous node)
Example (usually a *private inner class*)

```objective-c
@interface DNode : SNode // extend SNode
{
    DNode *prev; // pointer to previous Node
}
@end
```

- `initWithData: d` // constructor
  - `next: (DNode *) n` // takes both the successor (next)
  - `prev: (DNode *) p;` // and predecessor (prev)
- `(DNode *) prev;` // predecessor node getter
- `setPrev: (DNode *) newPrev;` // predecessor node setter
**Example**

```objective-c
@implementation DNode

- initWithData: d next: (DNode *) n prev: (DNode *) p // constructor
{
    return [[self initWithData: d next: n] setPrev: p];
}

- (DNode *) prev // predecessor node getter
{
    return prev;
}

- setPrev: (DNode *) newPrev // predecessor node setter
{
    prev = newPrev;
    return self;
}
@end
```
Example (usually a *private inner class*)

```cpp
template <class T> class DNode: public SNode<T>
{
    DNode<T> *pre; // pointer to previous Node

public:
    DNode(T &d, DNode *n = NULL, DNode *p = NULL): SNode<T>(d, n)
    {
        pre = p; // set predecessor node
    }

    DNode *prev() { return pre; } // prev node getter
    DNode *next() // next node getter
    {
        return (DNode *) SNode<T>::next();
    } // use super class getter

    void setPrev(DNode *p) { pre = p; } // prev node setter
};
```
Creating A Doubly Linked Node

Example (Objective-C)

```objective-c
DNode *head = [[DNode alloc] initWithData: @"D1" next: nil prev: nil];
DNode *tail = head;
```

Example (C++)

```cpp
string dl("D1");
DNode<string> *head = new DNode<string>(dl);
DNode<string> *tail = head;
```
Linking a Second DNode

Example (Objective-C)
```
[head setNext:  [[DNode alloc] initWithData: @"D2" next: nil prev: head]];
tail = [tail next];
```

Example (C++)
```
string d2("D2");
head->setNext(new DNode<string>(d2,NULL,head));
tail = tail->next();
```
Inserting a DNode at the Head

Example (Objective-C)
```objective-c
head = [head prev];
```

Example (C++)
```cpp
string d2("D2");
head->setPrev(new DNode<string>(d2, head));
head = head->prev();
```
Traversing Backwards

Example (using a for loop in Objective-C)

```c
for (DNode *ptr = tail; // last element
     ptr != nil; // until finished
     ptr = [ptr prev]) // switch to predecessor element
{
    printf("ptr: %s\n", [[[ptr data] description] UTF8String]);
}
```
Traversing Backwards (C++)

Example (using a `for` loop in C++)

```cpp
for (DNode<string> *ptr = tail; // last element
     ptr != NULL; // until finished
     ptr = ptr->prev()) // switch to predecessor element
{
    cout << "ptr: " << ptr->data() << endl;
}
```
Caveats

- **NULL Pointer Exceptions**
  - if anything, the number of checks now has increased
  - head and tail as well as prev and next for every node!
  ⇒ more if’s required for correct code
Circular Linked Lists
Circular Lists

- **Problem:** head and tail
  - extra variables
- **Special Cases**
  - inserting/deleting at the beginning/end
- **Solution**
  - use a dummy Node
    - next points to first List element, like head
    - prev points to last List element, like tail
Empty Circular List

Example (Objective-C)

```objective-c
DNode *list = [DNode new];
[list setNext: list];
[list setPrev: list];
```

Example (C++)

```cpp
string dummy('"');
DNode<string> *list = new DNode<string>(dummy);
list->setNext(list); list->setPrev(list);
```
Adding at the Head

Example (Objective-C)

```objective-c
DNode *node = [[DNode alloc] initWithData: @"D1" next: [list next] prev: list];
[[list next] setPrev: node];
[list setNext: node];
```

Example (C++)

```cpp
DNode<string> *node = new DNode<string>(d1, list->next(), list);
list->next()->setPrev(node);
list->setNext(node);
```
Adding at the Head (2)

Example (Objective-C)

```objective-c
DNode *node = [[DNode alloc] initWithData: @"D_0" next: [list next] prev: list];
[[list next] setPrev: node];
[list setNext: node];
```

Example (C++)

```cpp
DNode<string> *node = new DNode<string>(d0, list->next(), list);
list->next()->setPrev(node);
list->setNext(node);
```
Adding at the Tail

Example (Objective-C)

```objective-c
node = [[DNode alloc] initWithData: @"D2" next: list prev: [list prev]];
[[list prev] setNext: node]; [list setPrev: node];
```

Example (C++)

```cpp
DNode<string> *node = new DNode<string>(d2, list, list->prev());
list->prev()->setNext(node); list->setPrev(node);
```
Traversing Circular Lists

Example (using a for loop in Objective-C)

```objective-c
for (DNode *ptr = [list next]; // first element
     [ptr data] != nil; // until finished
     ptr = [ptr next]) // switch to successor element
{
    printf("ptr: %s\n", [[[ptr data] description] UTF8String]);
}
```
Traversing Circular Lists (C++)

Example (using a for loop in C++)

```cpp
for (DNode<string> *ptr = list->next(); // first element
     ptr->data() != dummy; // until finished
     ptr = ptr->next()) // switch to successor element
{
    cout << "ptr: " << ptr->data() << endl;
}
```