Lecture 10: Linked Lists

Brian Hou
July 6, 2016
Announcements

• Project 2 is due 7/12 (+1 EC point if submitted 7/12)
Announcements

• Project 2 is due 7/12 (+1 EC point if submitted 7/12)
  • Run ok --submit to check against hidden tests
Announcements

• Project 2 is due 7/12 (+1 EC point if submitted 7/12)
  • Run ok --submit to check against hidden tests
  • Check your submission at ok.cs61a.org
Announcements

• Project 2 is due 7/12 (+1 EC point if submitted 7/12)
  • Run ok --submit to check against hidden tests
  • Check your submission at ok.cs61a.org
  • Invite your partner (watch this video)
Announcements

• Project 2 is due 7/12 (+1 EC point if submitted 7/12)
  • Run ok --submit to check against hidden tests
  • Check your submission at ok.cs61a.org
  • Invite your partner (watch this video)
• Homework 4 is due 7/7
Announcements

• Project 2 is due 7/12 (+1 EC point if submitted 7/12)
  • Run ok --submit to check against hidden tests
  • Check your submission at ok.cs61a.org
  • Invite your partner (watch this video)
• Homework 4 is due 7/7
• Quiz 3 is tomorrow at the beginning of lecture
Announcements

• Project 2 is due 7/12 (+1 EC point if submitted 7/12)
  • Run ok --submit to check against hidden tests
  • Check your submission at ok.cs61a.org
  • Invite your partner (watch this video)
• Homework 4 is due 7/7
• Quiz 3 is tomorrow at the beginning of lecture
  • If you have an alternate time or are not enrolled in the class, please arrive at 11:45 am
Announcements

• Project 2 is due 7/12 (+1 EC point if submitted 7/12)
  • Run ok --submit to check against hidden tests
  • Check your submission at ok.cs61a.org
  • Invite your partner (watch this video)
• Homework 4 is due 7/7
• Quiz 3 is tomorrow at the beginning of lecture
  • If you have an alternate time or are not enrolled in the class, please arrive at 11:45 am
• Quiz 4 will be released 9 am on 7/11, due 10 am on 7/12
Announcements

- Project 2 is due 7/12 (+1 EC point if submitted 7/12)
  - Run ok --submit to check against hidden tests
  - Check your submission at ok.cs61a.org
  - Invite your partner (watch this video)
- Homework 4 is due 7/7
- Quiz 3 is tomorrow at the beginning of lecture
  - If you have an alternate time or are not enrolled in the class, please arrive at 11:45 am
- Quiz 4 will be released 9 am on 7/11, due 10 am on 7/12
- 61A Potluck on 7/8! 5 – 8 pm (or later) in Wozniak Lounge
Announcements

- Project 2 is due 7/12 (+1 EC point if submitted 7/12)
  - Run ok --submit to check against hidden tests
  - Check your submission at ok.cs61a.org
  - Invite your partner (watch this video)
- Homework 4 is due 7/7
- Quiz 3 is tomorrow at the beginning of lecture
  - If you have an alternate time or are not enrolled in the class, please arrive at 11:45 am
- Quiz 4 will be released 9 am on 7/11, due 10 am on 7/12
- 61A Potluck on 7/8! 5 – 8 pm (or later) in Wozniak Lounge
  - Bring food and board games!
Hog Contest
Hog Contest

- 76 contestants
Hog Contest

• 76 contestants
  • 20 new challengers on the last day
Hog Contest

• 76 contestants
  • 20 new challengers on the last day
  • 11 new challengers in the last 6 hours
Hog Contest

- 76 contestants
  - 20 new challengers on the last day
  - 11 new challengers in the last 6 hours
- The winner:
Hog Contest

• 76 contestants
  • 20 new challengers on the last day
  • 11 new challengers in the last 6 hours

• The winner:
  1. Edgar Orendain
Hog Contest

• 76 contestants
  • 20 new challengers on the last day
  • 11 new challengers in the last 6 hours
• The winner:
  1. Edgar Orendain
  1. Going Deep Blue
Hog Contest

• 76 contestants
  • 20 new challengers on the last day
  • 11 new challengers in the last 6 hours
• The winner:
  1. Edgar Orendain
  1. Going Deep Blue
  1. The best team on the 3rd floor of Davidson (U2)
Hog Contest

- 76 contestants
  - 20 new challengers on the last day
  - 11 new challengers in the last 6 hours
- The winner:
  1. Edgar Orendain
  1. Going Deep Blue
  1. The best team on the 3rd floor of Davidson (U2)
  1. Going DeepMind
Hog Contest

- 76 contestants
  - 20 new challengers on the last day
  - 11 new challengers in the last 6 hours
- The winner:
  1. Edgar Orendain
  1. Going Deep Blue
  1. The best team on the 3rd floor of Davidson (U2)
  1. Going DeepMind
Hog Contest

- 76 contestants
  - 20 new challengers on the last day
  - 11 new challengers in the last 6 hours
- The winner:
  1. Edgar Orendain
  1. Going Deep Blue
  1. The best team on the 3rd floor of Davidson (U2)
  1. Going DeepMind

Thank you to all the participants!
Hog Contest

• 76 contestants
  • 20 new challengers on the last day
  • 11 new challengers in the last 6 hours
• The winner:
  1. Edgar Orendain
  1. Going Deep Blue
  1. The best team on the 3rd floor of Davidson (U2)
  1. Going DeepMind

Thank you to all the participants!

Full rankings: cs61a.org/proj/hog_contest
• This week (Data), the goals are:
This week (Data), the goals are:

- To continue our journey through abstraction with data abstraction
This week (Data), the goals are:

- To continue our journey through abstraction with *data abstraction*
- To study useful data types we can construct with data abstraction
Data Abstraction
Data Abstraction
Data Abstraction

• Great programmers use data abstraction to separate:
Data Abstraction

• Great programmers use data abstraction to separate:
  • How compound values are used (the unit)
Data Abstraction

• Great programmers use data abstraction to separate:
  • How compound values are used (the unit)
  • How compound values are represented (the parts)
Data Abstraction

• Great programmers use data abstraction to separate:
  • How compound values are *used* (the unit)
  • How compound values are *represented* (the parts)
Data Abstraction

• Great programmers use data abstraction to separate:
  • How compound values are *used* (the unit)
  • How compound values are *represented* (the parts)
Abstraction Barrier Violations
Abstraction Barrier Violations

• Constructors and selectors provide us with abstraction, allowing us to use the data type without having to know its implementation
Abstraction Barrier Violations

- Constructors and selectors provide us with abstraction, allowing us to use the data type without having to know its implementation.
- An abstraction barrier violation is when we assume knowledge about the data type implementation, rather than using constructors and selectors.
Abstraction Barrier Violations

• Constructors and selectors provide us with abstraction, allowing us to use the data type without having to know its implementation.

• An abstraction barrier violation is when we assume knowledge about the data type implementation, rather than using constructors and selectors.

Never violate the abstraction barrier!
Sequences
The Sequence Abstraction
The Sequence Abstraction

The sequence abstraction is a collection of behaviors:
The Sequence Abstraction

The sequence abstraction is a collection of behaviors:

**Length.** A sequence has a finite length.

**Element selection.** A sequence has an element corresponding to any non-negative integer index less than its length, starting at 0.
The Sequence Abstraction

The sequence abstraction is a collection of behaviors:

**Length.** A sequence has a finite length.

**Element selection.** A sequence has an element corresponding to any non-negative integer index less than its length, starting at 0.

Lists and strings are both examples of sequences.
The Sequence Abstraction

The sequence abstraction is a collection of behaviors:

**Length.** A sequence has a finite length.

**Element selection.** A sequence has an element corresponding to any non-negative integer index less than its length, starting at 0.

Lists and strings are both examples of sequences.

We can use built-in syntax associated with this behavior. We can also use functions.
The Sequence Abstraction

The sequence abstraction is a collection of behaviors:

**Length.** A sequence has a finite length.

**Element selection.** A sequence has an element corresponding to any non-negative integer index less than its length, starting at 0.

Lists and strings are both examples of sequences.

We can use built-in syntax associated with this behavior. We can also use functions.
Linked Lists
Linked Lists
Linked Lists

- Another way to implement the sequence abstraction
Linked Lists

- Another way to implement the sequence abstraction
- Links have two parts
Linked Lists

• Another way to implement the sequence abstraction
• Links have two parts
  • first: the element in the link
Linked Lists

- Another way to implement the sequence abstraction
- Links have two parts
  - first: the element in the link
  - rest: the next link in the list
Linked Lists

- Another way to implement the sequence abstraction
- Links have two parts
  - first: the element in the link
  - rest: the next link in the list
- This is a recursive definition: the rest of a linked list is another linked list
Linked Lists

- Another way to implement the sequence abstraction
- Links have two parts
  - first: the element in the link
  - rest: the next link in the list
- This is a recursive definition: the rest of a linked list is another linked list

1
Linked Lists

- Another way to implement the sequence abstraction
- Links have two parts
  - first: the element in the link
  - rest: the next link in the list
- This is a recursive definition: the rest of a linked list is another linked list

![Diagram of a linked list with a node labeled 1 and another node labeled 2]
Linked Lists

- Another way to implement the sequence abstraction
- Links have two parts
  - first: the element in the link
  - rest: the next link in the list
- This is a recursive definition: the rest of a linked list is another linked list

```
1 -> 2 -> 3
```
Linked Lists

- Another way to implement the sequence abstraction
- Links have two parts
  - first: the element in the link
  - rest: the next link in the list
- This is a recursive definition: the rest of a linked list is another linked list

```
1 → 2 → 3 → [null]
```
Linked Lists

• Another way to implement the sequence abstraction

• Links have two parts
  • first: the element in the link
  • rest: the next link in the list

• This is a recursive definition: the rest of a linked list is another linked list
Linked Lists

- Another way to implement the sequence abstraction
- Links have two parts
  - \textcolor{blue}{first}: the element in the link
  - \textcolor{orange}{rest}: the next link in the list
- This is a recursive definition: the rest of a linked list is another linked list
Linked Lists

• Another way to implement the sequence abstraction
• Links have two parts
  • first: the element in the link
  • rest: the next link in the list
• This is a recursive definition: the rest of a linked list is another linked list

• This data structure has many names:
Linked Lists

- Another way to implement the sequence abstraction
- Links have two parts
  - \textit{first}: the element in the link
  - \textit{rest}: the next link in the list
- This is a recursive definition: the rest of a linked list is another linked list

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{linked_list_diagram.png}
\end{figure}

- This data structure has many names:
  - Linked list (C, Java)
  - List (Lisp)
  - Forward list (C++)
Linked Lists

- Another way to implement the sequence abstraction
- Links have two parts
  - `first`: the element in the link
  - `rest`: the next link in the list
- This is a recursive definition: the rest of a linked list is another linked list

- This data structure has many names:
  - Linked list (C, Java)
  - List (Lisp)
  - Forward list (C++)
  - Linky Listys (TAs)
Linked List Abstraction
def link(first, rest):
    """Construct a linked list from its first element and the rest of the linked list."""
def link(first, rest):
   """Construct a linked list from its first element and the rest of the linked list."""

def first(s):
   """Return the first element of a linked list S."""

def rest(s):
   """Return the rest of the elements of a linked list S."""
If a linked list $s$ is constructed from a first element $h$ and a linked list $t$, then

```python
def link(first, rest):
    """Construct a linked list from its first element and the rest of the linked list.""

def first(s):
    """Return the first element of a linked list $S$.""

def rest(s):
    """Return the rest of the elements of a linked list $S$.""
```
Linked List Abstraction

```python
def link(first, rest):
    """Construct a linked list from its first element and the rest of the linked list.""

def first(s):
    """Return the first element of a linked list S.""

def rest(s):
    """Return the rest of the elements of a linked list S.""
```

If a linked list s is constructed from a first element h and a linked list t, then
- `first(s)` returns h, which is an element of the sequence
def link(first, rest):
    """Construct a linked list from its first element and the rest of the linked list."""

def first(s):
    """Return the first element of a linked list S."""

def rest(s):
    """Return the rest of the elements of a linked list S."""

If a linked list s is constructed from a first element h and a linked list t, then
• first(s) returns h, which is an element of the sequence
• rest(s) returns t, which is a linked list
Implementing Linked Lists (v1)
def link(first, rest):
    """Construct a linked list from its first element and the rest of the linked list."""
def link(first, rest):
    """Construct a linked list from its first element and the rest of the linked list."""

def first(s):
    """Return the first element of a linked list S."""

def rest(s):
    """Return the rest of the elements of a linked list S."""
Implementing Linked Lists (v1)

```python
def link(first, rest):
    """Construct a linked list from its first element and the rest of the linked list.""
    return [first, rest]

def first(s):
    """Return the first element of a linked list S.""

def rest(s):
    """Return the rest of the elements of a linked list S.""
```
Implementing Linked Lists (v1)

```python
def link(first, rest):
    """Construct a linked list from its first element and the rest of the linked list.""
    return [first, rest]

def first(s):
    """Return the first element of a linked list S.""
    return s[0]

def rest(s):
    """Return the rest of the elements of a linked list S.""
```
def link(first, rest):
    """Construct a linked list from its first element and the rest of the linked list.""
    return [first, rest]

def first(s):
    """Return the first element of a linked list S.""
    return s[0]

def rest(s):
    """Return the rest of the elements of a linked list S.""
    return s[1]
def link(first, rest):
    """Construct a linked list from its first element and the rest of the linked list.""
    return [first, rest]

def first(s):
    """Return the first element of a linked list S.""
    return s[0]

def rest(s):
    """Return the rest of the elements of a linked list S.""
    return s[1]
Linked Lists are Sequences
Linked Lists are Sequences
def len_link(s):
    """Return the length of the linked list."""
    length = 0
    while s != empty:
        s, length = rest(s), length + 1
    return length
def len_link(s):
    """Return the length of the linked list."""
    length = 0
    while s != empty:
        s, length = rest(s), length + 1
    return length

def getitem_link(s, i):
    """Return the element at index i."""
    while i > 0:
        s, i = rest(s), i - 1
    return first(s)

Linked Lists are Sequences  (demo)
def len_link(s):
    """Return the length of the linked list.""
    length = 0
    while s != empty:
        s, length = rest(s), length + 1
    return length

def getitem_link(s, i):
    """Return the element at index i.""
    while i > 0:
        s, i = rest(s), i - 1
    return first(s)

Never violate the abstraction barrier!
Linked Lists are Recursive
Linked Lists are Recursive
def len_link(s):
    """Return the length of the linked list."""
    if s == empty:
        return 0
    else:
        return 1 + len_link(rest(s))
def len_link(s):
    """Return the length of the linked list.""
    if s == empty:
        return 0
    else:
        return 1 + len_link(rest(s))

defgetitem_link(s, i):
    """Return the element at index i.""
    if i == 0:
        return first(s)
    else:
        returngetitem_link(rest(s), i - 1)
def len_link(s):
    """Return the length of the linked list.""
    if s == empty:
        return 0
    else:
        return 1 + len_link(rest(s))

def getitem_link(s, i):
    """Return the element at index i.""
    if i == 0:
        return first(s)
    else:
        return getitem_link(rest(s), i - 1)

Never violate the abstraction barrier!
Break!
Linked List Processing
Sequences as Containers
Sequences as Containers  (demo)
def contains(s, elem):
    """Return whether ELEM is in the sequence S.
    >>> contains([1, 2, 3], 1)
    True
    >>> contains([1, 2, 3], 4)
    False
    """
    for x in s:
        if x == elem:
            return True
    return False
def contains_link(s, elem):
    """Return whether ELEM is in the sequence S.
    >>> contains_link(link(1, link(2, link(3, empty))), 1)
    True
    >>> contains_link(link(1, link(2, link(3, empty))), 4)
    False
    """
    if s == empty:
        return False
    if first(s) == elem:
        return True
    else:
        return contains(rest(s), elem)
Linked List Examples
def count_partitions(n, m):
    if n == 0:
        return 1
    elif n < 0:
        return 0
    elif m == 0:
        return 0
    else:
        with_m = count_partitions(n-m, m)
        without_m = count_partitions(n, m-1)
        return with_m + without_m
Enumerating Partitions
Enumerating Partitions (demo)
Enumerating Partitions

def partitions(n, m):
def partitions(n, m):
    if n == 0:
def partitions(n, m):
    if n == 0:
        return link(empty, empty)
def partitions(n, m):
    if n == 0:
        return link(empty, empty)
    elif n < 0 or m == 0:
def partitions(n, m):
    if n == 0:
        return link(empty, empty)
    elif n < 0 or m == 0:
        return empty
def partitions(n, m):
    if n == 0:
        return link(empty, empty)
    elif n < 0 or m == 0:
        return empty
    else:
Enumerating Partitions

```python
def partitions(n, m):
    if n == 0:
        return link(empty, empty)
    elif n < 0 or m == 0:
        return empty
    else:
        with_m = partitions(n-m, m)
```

def partitions(n, m):
    if n == 0:
        return link(empty, empty)
    elif n < 0 or m == 0:
        return empty
    else:
        with_m = partitions(n-m, m)
        without_m = partitions(n, m-1)
def partitions(n, m):
    if n == 0:
        return link(empty, empty)
    elif n < 0 or m == 0:
        return empty
    else:
        with_m = partitions(n-m, m)
        without_m = partitions(n, m-1)
        add_m = lambda s: link(m, s)
def partitions(n, m):
    if n == 0:
        return link(empty, empty)
    elif n < 0 or m == 0:
        return empty
    else:
        with_m = partitions(n-m, m)
        without_m = partitions(n, m-1)
        add_m = lambda s: link(m, s)
        with_m = map_link(add_m, with_m)
def partitions(n, m):
    if n == 0:
        return link(empty, empty)
    elif n < 0 or m == 0:
        return empty
    else:
        with_m = partitions(n-m, m)
        without_m = partitions(n, m-1)
        add_m = lambda s: link(m, s)
        with_m = map_link(add_m, with_m)
        return extend(with_m, without_m)
Other Linked List Implementations
Implementing Linked Lists (v1)

```python
def link(first, rest):
    """Construct a linked list from its first element and the rest of the linked list.""
    return [first, rest]

def first(s):
    """Return the first element of a linked list S.""
    return s[0]

def rest(s):
    """Return the rest of the elements of a linked list S.""
    return s[1]
```
Implementing Linked Lists (v2)
Implementing Linked Lists (v2)

def link(first, rest):
def link(first, rest):

def first(s):

def rest(s):
Implementing Linked Lists (v2)

```python
def link(first, rest):
    def dispatch(msg):

def first(s):

def rest(s):
```
Implementing Linked Lists (v2)

def link(first, rest):
    def dispatch(msg):
        if msg == 'first':

def first(s):

def rest(s):
Implementing Linked Lists (v2)

```python
def link(first, rest):
    def dispatch(msg):
        if msg == 'first':
            return first

def first(s):

def rest(s):
```
Implementing Linked Lists (v2)

def link(first, rest):
    def dispatch(msg):
        if msg == 'first':
            return first
        elif msg == 'rest':

    def first(s):

    def rest(s):
Implementing Linked Lists (v2)

```python
def link(first, rest):
    def dispatch(msg):
        if msg == 'first':
            return first
        elif msg == 'rest':
            return rest

def first(s):

def rest(s):
```
def link(first, rest):
    def dispatch(msg):
        if msg == 'first':
            return first
        elif msg == 'rest':
            return rest
    return dispatch

def first(s):

def rest(s):
def link(first, rest):
    def dispatch(msg):
        if msg == 'first':
            return first
        elif msg == 'rest':
            return rest
    return dispatch

def first(s):
    return s('first')

def rest(s):
def link(first, rest):
    def dispatch(msg):
        if msg == 'first':
            return first
        elif msg == 'rest':
            return rest
    return dispatch

def first(s):
    return s('first')

def rest(s):
    return s('rest')
Implementing Linked Lists (v2) (demo)

def link(first, rest):
    def dispatch(msg):
        if msg == 'first':
            return first
        elif msg == 'rest':
            return rest
    return dispatch

def first(s):
    return s('first')

def rest(s):
    return s('rest')
def link(first, rest):
    def dispatch(msg):
        if msg == 'first':
            return first
        elif msg == 'rest':
            return rest
    return dispatch

def first(s):
    return s('first')

def rest(s):
    return s('rest')
Implementing Linked Lists (v3)

```python
def link(first, rest):
    def dispatch(msg):
        if msg == 'brian':
            return first
        elif msg == 'marvin':
            return rest
        return dispatch
    return dispatch

def first(s):
    return s('brian')

def rest(s):
    return s('marvin')
```
Summary

- Linked lists are one implementation of the sequence abstraction.
- Linked lists are composed of two parts:
  - first: the element in the link
  - rest: the next link in the list (may be empty)
- Data abstraction means that the implementation details of the first and rest selectors are unnecessary.
- We can use functions to implement linked lists.
  - We can use lists to implement dictionaries.
  - Therefore, we can use functions to implement dictionaries.