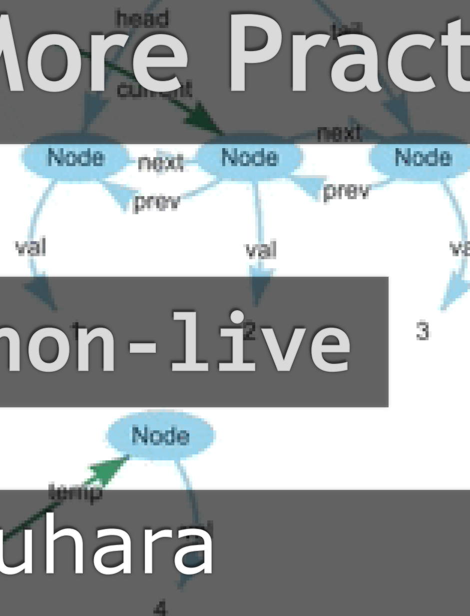
Challenges and Ideas for Making Live Programming More Practical



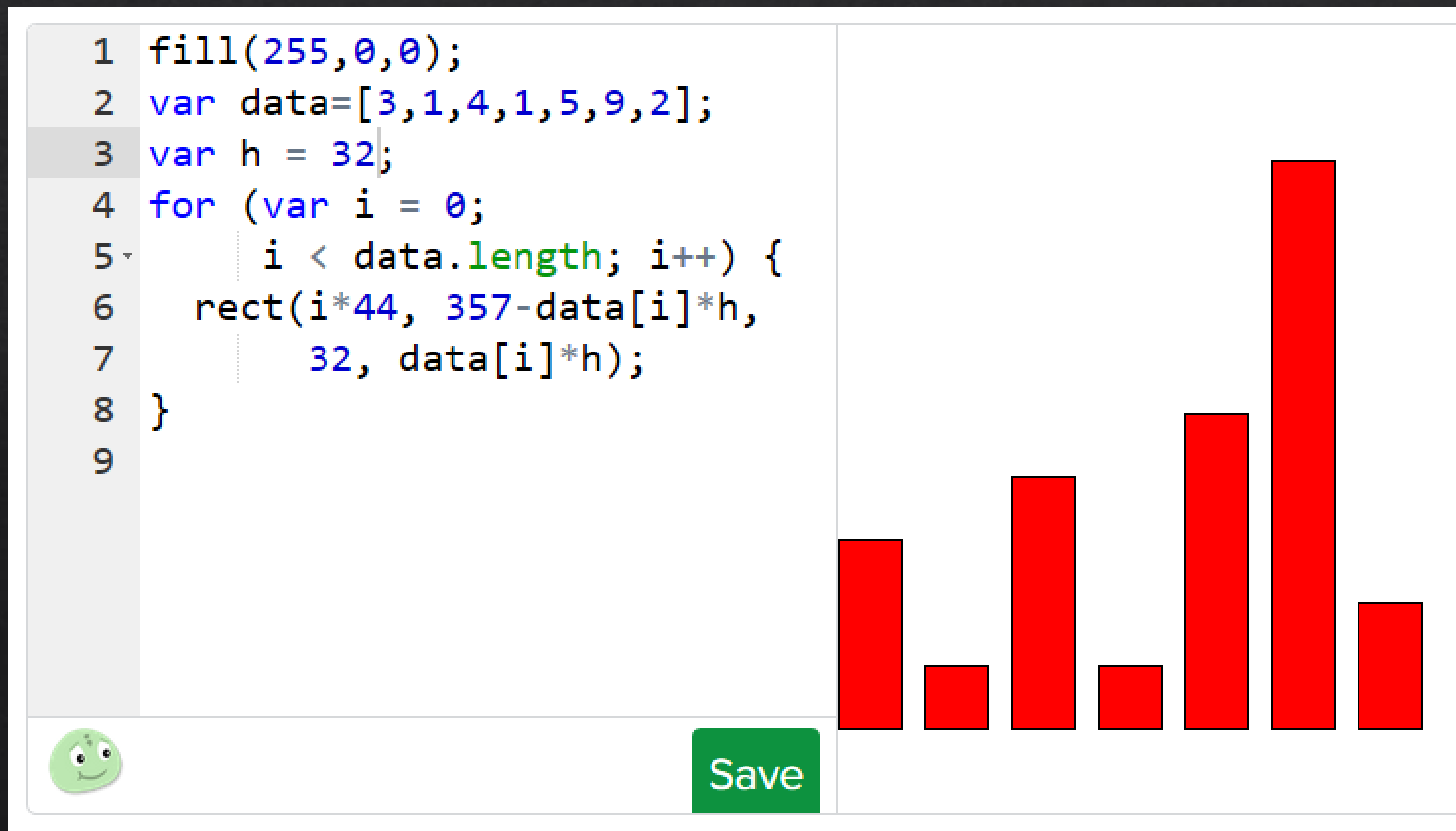
tinyurl.com/kanon-live

Hidehiko Masuhara

Tokyo Tech

Live Programming Environment an example

- Kahn Academy's Live Coding Editor
  https://www.khanacademy.org/computer-programming/new/pjs



Real Programmers don't Use Live Programming!?

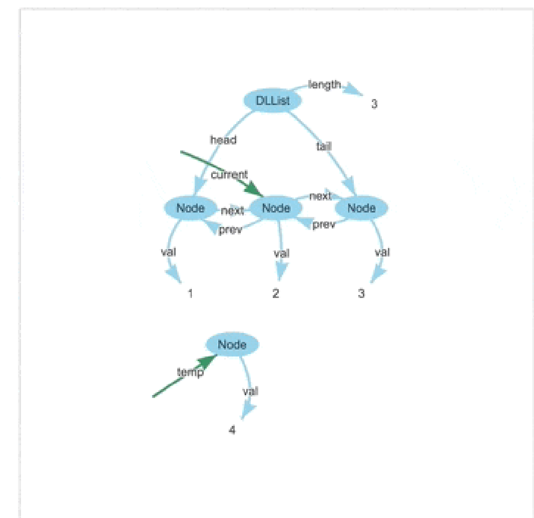
- Use cases:
  - Education (Kahn Academy's)
  - Arcade games [McDirmid'07]
  - Music (aka Live Coding) [Aaron'13]
- Can be used by real programmers?

What can we do for real programmers?

Proposal: Live Data Structure Programming

- DS Programming defines structures and ops.
  - real
  - error prone
  - diagrams in our mind
- Kanon LPE [Onward!18]
  - https://github.com/prg-titech/Kanon

```
1  class Node {…}
8  class DLList {
9      constructor() {…}
14
15     add(val) {…}
28
29     insert(val, index) {
30         if (0 <= index && index <= this.length) {
31             let temp = new Node(val);
32
33             if (index === this.length) {…
37             } else if (index === 0) {…
41             } else {
42                 let current = this.head;
43
44                 for (var i = 0; i < index; i++) {
45                     current = current.next;
46                 }
47
48
49             }
50             this.length++;
51         }
52     }
53 }
54
55 let list = new DLList();
56
57 list.add(1);
58 list.add(2);
59 list.add(3);
60 list.insert(4, 1);
```



Summarized | Snapshot | View Mode: Snapshot

Showing Context | ON

Redraw

A Quick Introduction to Kanon

try at:
tinyurl.com/kanon-live

Kanon is

- a live programming environment
- for data structure programming
- in JavaScript
- runs on a web browser

Demo: define a linked list

A Quick Introduction to Kanon

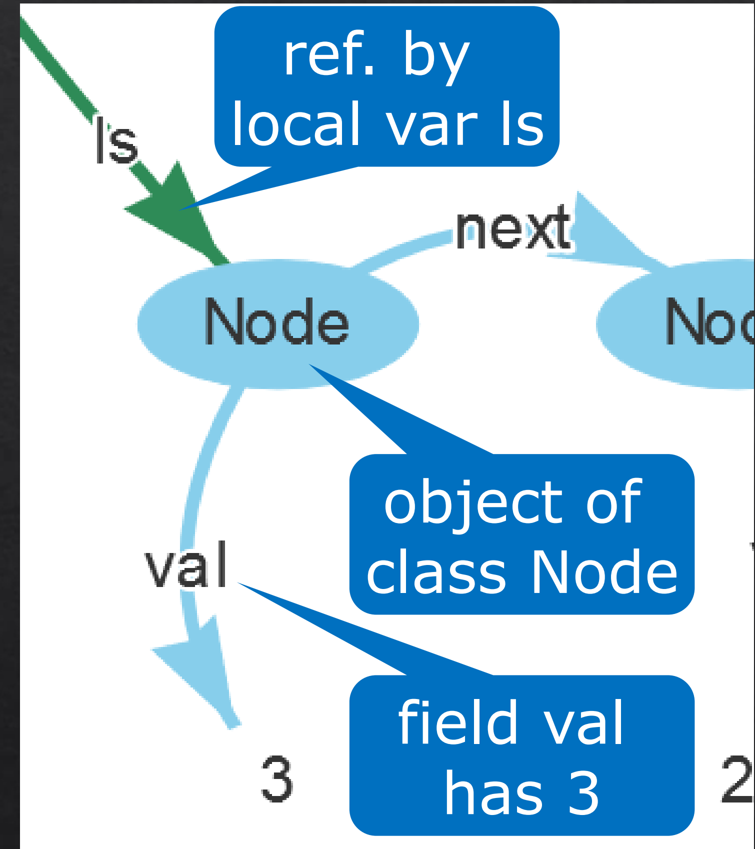
```
1 class Node {
2     constructor(val,next) {
3         this.val = val;
4         this.next = next;
5     }
6 }
7 var ls = new Node(0,null);
8 ls = new Node(1,ls);
9 ls = new Node(2,ls);
10 ls = new Node(3,ls);
```

editor

object diagram

call tree

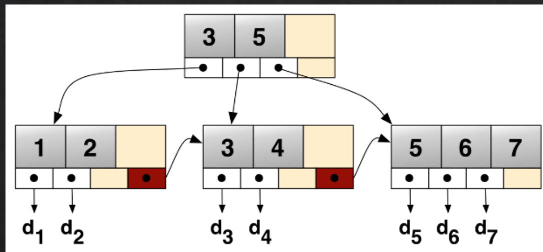
- Object diagram shows a state at the cursor position



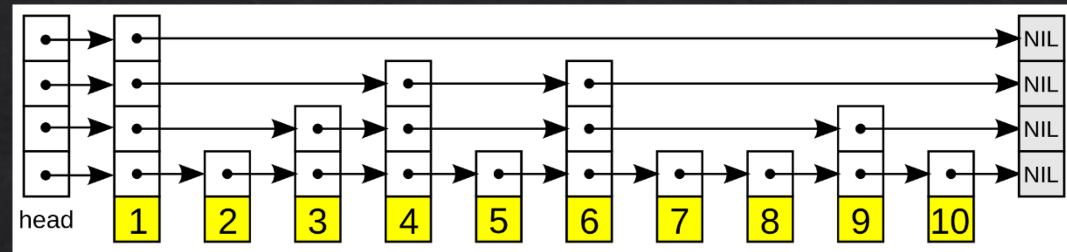
Goal: Live Programming for Real Programmers

- Kanon is for ***data structure programming***
  - can be hard, error prone
  - visualization can help us

defining & manipulating DS



B+ tree in Wikipedia



skip list in Wikipedia

- for education? debugging? — maybe

Programming Style with Kanon

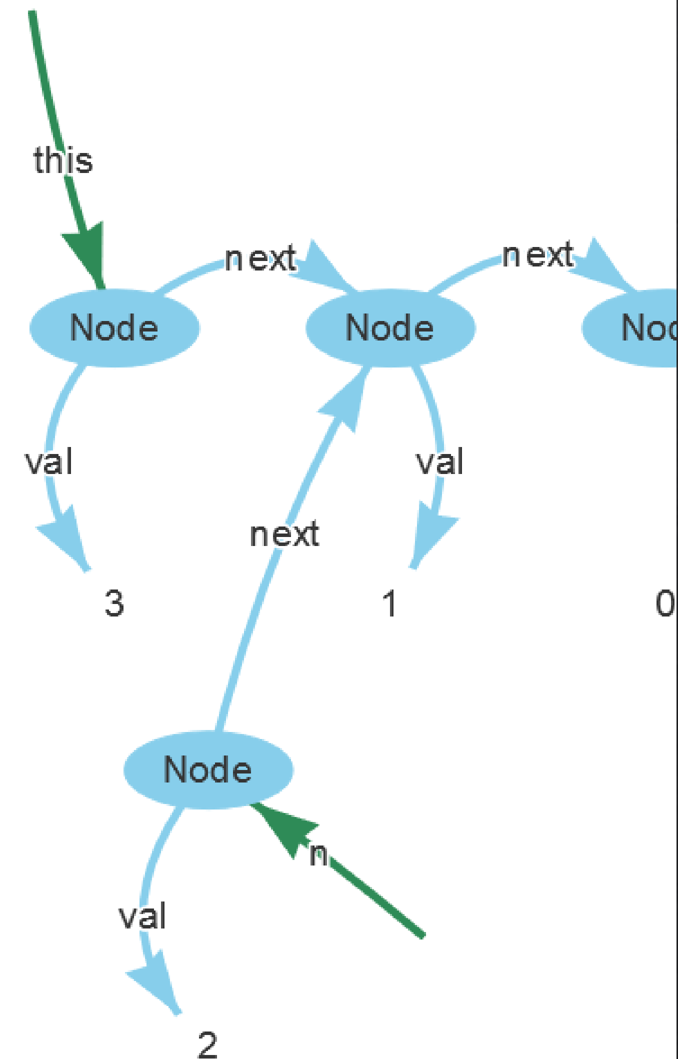
- Test-driven
- Observe-write-confirm
  - observing the current object graph
  - write the next line
  - confirm the effect

Demo: define **`swap()`** that swaps the first two elements of the list

Observe-write-confirm

```
5   }
6   swap() {
7       let n = this.next;
8       this.next = this.next.next;
9       |
10  }
11
```

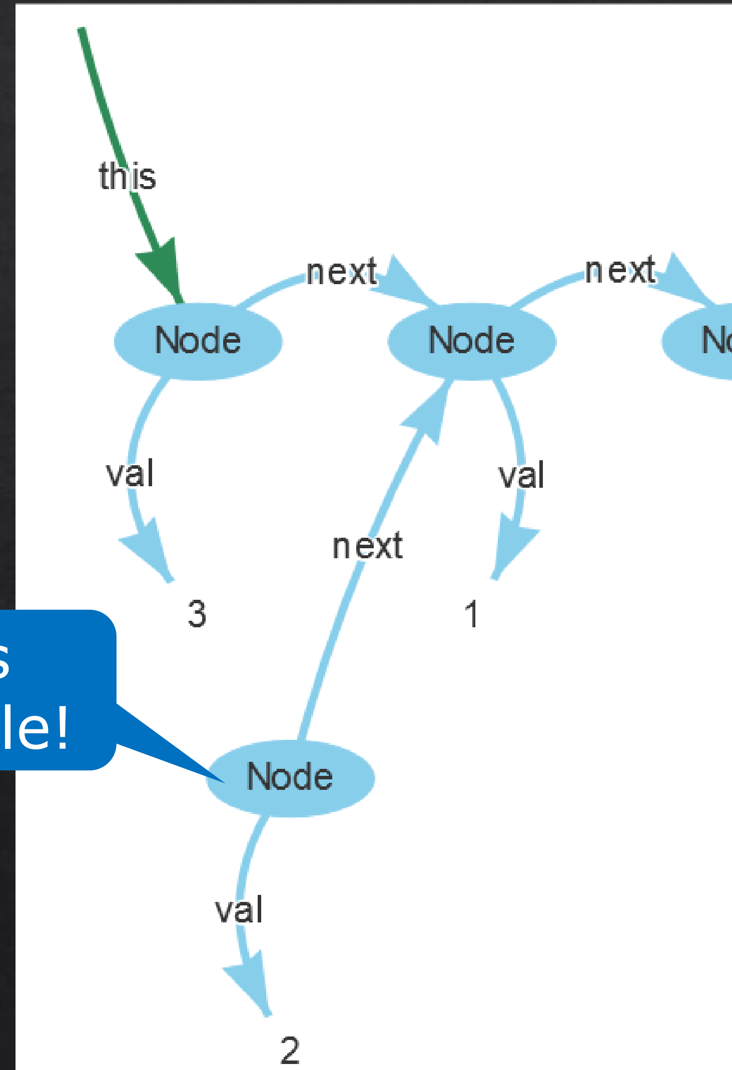
◈ Haven't you drawn a similar diagram in your mind?



You can See Your Mistakes!

```
6  swap() {
7      this.next = this.next.next;
8
9  }
10
```

OK, let's connect the 3rd one after the 1st...



Wait, it's unreachable!

More Features of Kanon

- Program-by-demonstration
- Summarized view
- "Who made this?"

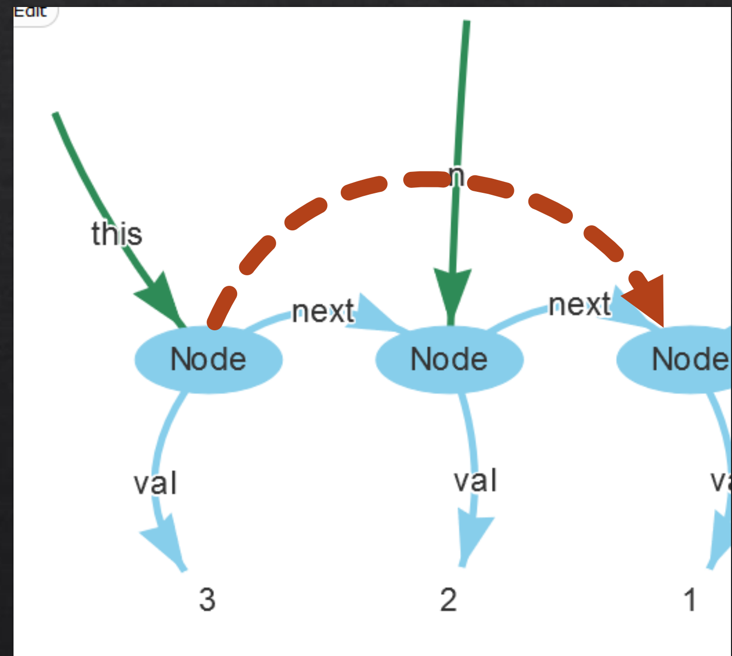
Program-by-Demonstration

Demo:

"make **next** of **this** reference the 3rd node "

◈ easy in the diagram

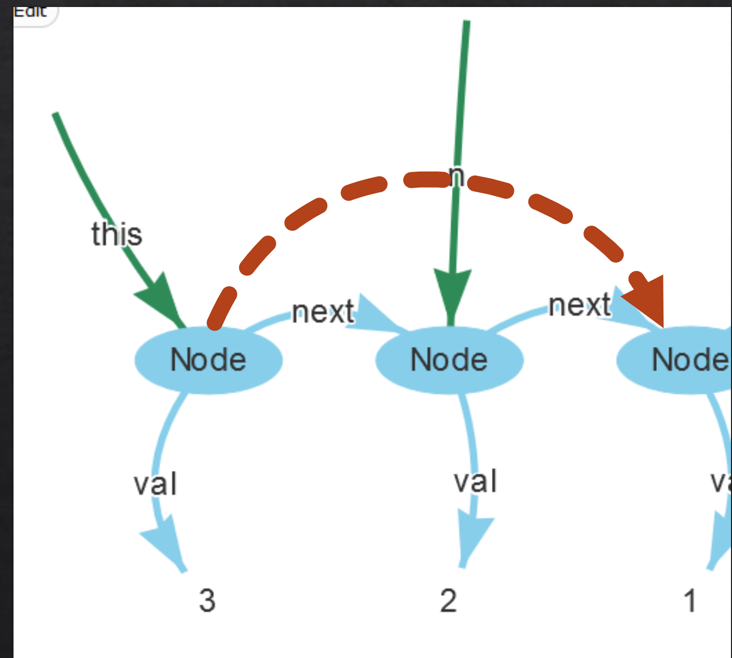
```
swap() {
    let n = this.next;
    |
```



Program-by-Demonstration

- given an edit with an arrow from $o1$ to $o2$ in the diagram
- search paths $p1$,$p2$ from local variables to $o1$ and $o2$ to have candidate "$p1.f = p2$"
- remaining challenges:
  - method calls
  - multiple edits

```
swap() {
    let n = this.next;
    |
```



Summarized View & "Who did This?"

- We sometimes lose track of the code and its effect
  - What does this code do?
  - Which code did make this?
- Summarized view & "Who did this?"

DEMO: a bug in binary search trees

Node
left
val
4
Node
left
val
5
Node
val
1
Node
val
2
Node
val
6

Summarized View & "Who did This?"

- Summarized view
  - shows the object graph at the end of exec.
  - highlights effects of the current code
- "Who did this?": clicking a node/edge moves the cursor to the responsible code

On-the-fly Expectation

- Q: Does the observe-edit-confirm style always work?
- A: There is at least one problematic case: Recursion.

Demo: define `rev()` that destructively reverses a list

On-the-fly Expectation

- Problem: if you call an incomplete function, you can't go on coding "after the call"
- A feature to rescue:
  On-the-fly expectation

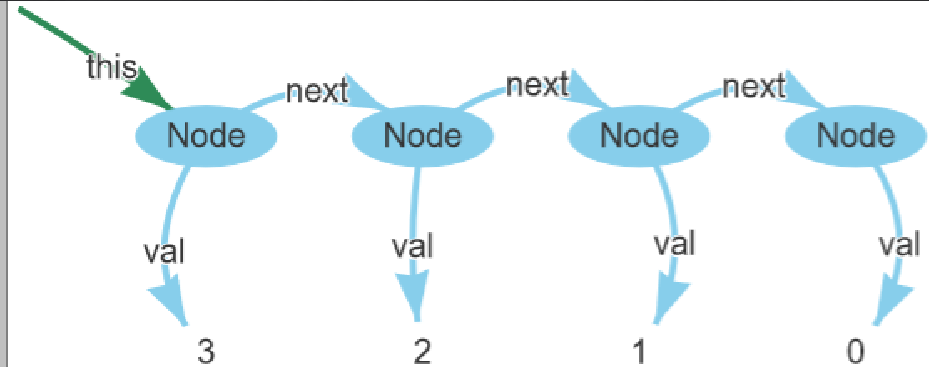
Demo: define `rev()` that destructively reverses a list

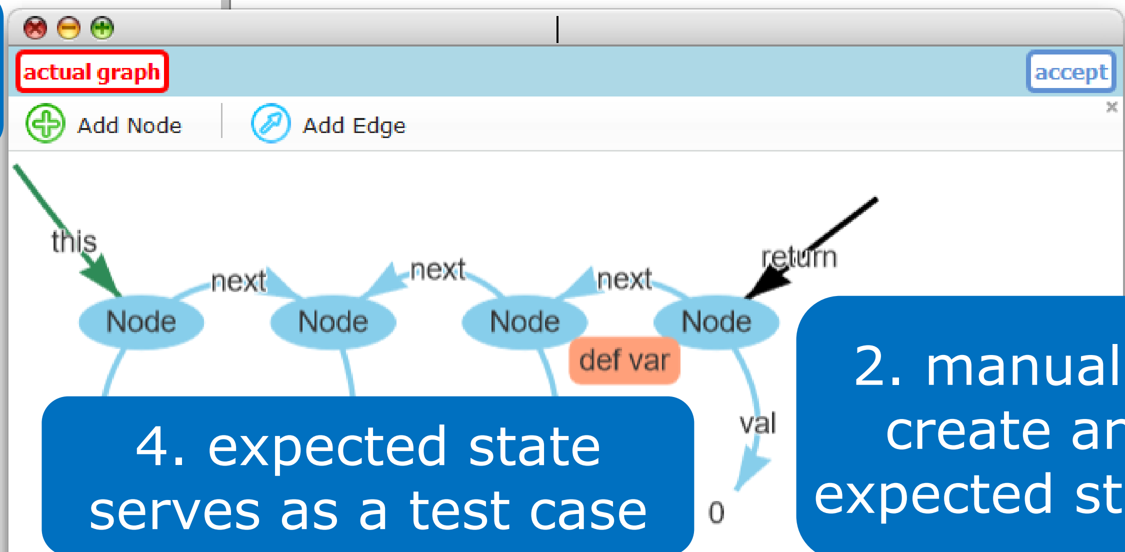
On-the-fly Expectation for Top-down Prog./Recursion

```
ss Node {
  constructor(val,next) {
      this.val =v
      this.next=n
  }
  swap() {}
  rev() {
      var next_rev = this.next.rev();

  }

 ls = new Node(0,null);
= new Node(1,ls);
= new Node(2,ls);
= new Node(3,ls);

s = ls.swap();

 ls.rev();
```

1. call to a unfinished function

3. continue coding wrt the expected state





A Bigger Example: λ

- Demo: define a λ-calculus interpreter

test case: [$((\lambda x.x)(\lambda y.y))z$].ev()

Research Topics

- Visualization algorithms
- Dealing with errors
- Effects on programmer's behavior

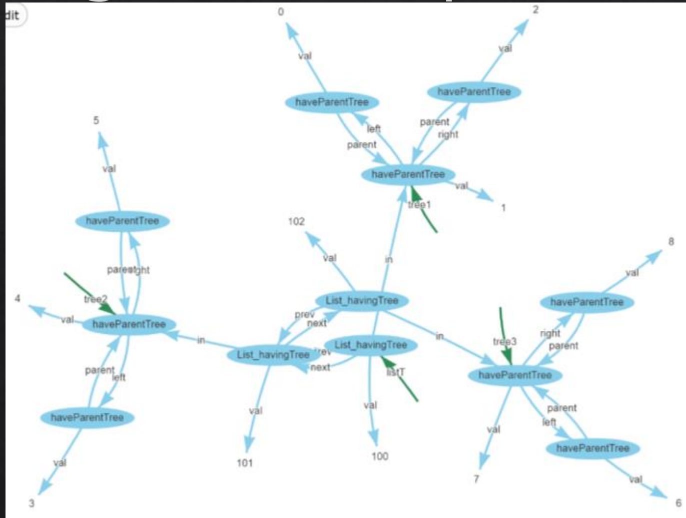
Visualization Algorithms

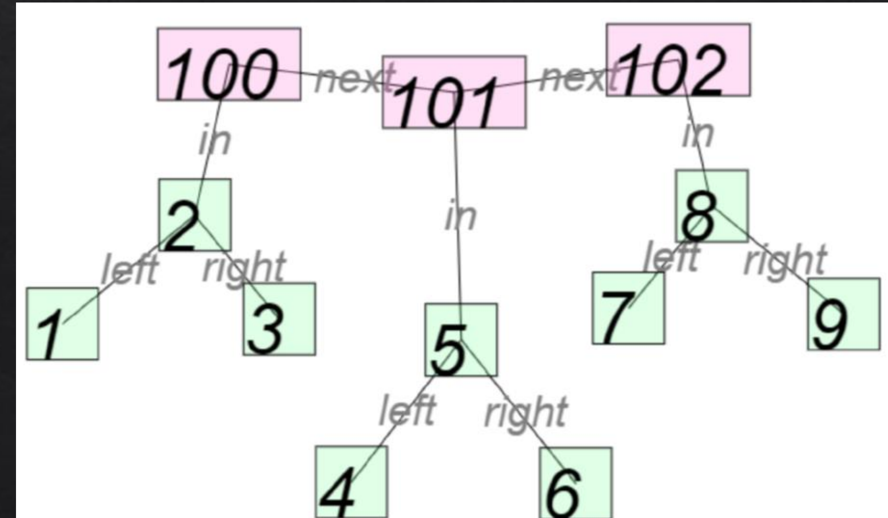
- Current:
  - layout: force-directed
    + special (ad-hoc) for lists and trees
  - mental map preservation [Onward'18]

    (DEMO)
- Future:
  - layout: automatic structure recognition
  - UI: zooming, customization, ...

Automatic Visualization Algorithm (in progress)

- Problems:
  - Force-directed algorithms: hard to read "data structures"
  - Hierarchical algorithms: not good for complex structures
- Hypothesis: aligning angles of same name edges will help us





User Experiment: How do People Use Kanon?

Settings

- 2 small tasks (eg. reverse a double-linked list)
- with Kanon vs with textual environment
- 9 students + 4 industrials

Findings

- Many bugs ;)
- It's usable.
- They liked it!
- **Dealing w/errors**
- **Unique mistakes with Kanon**

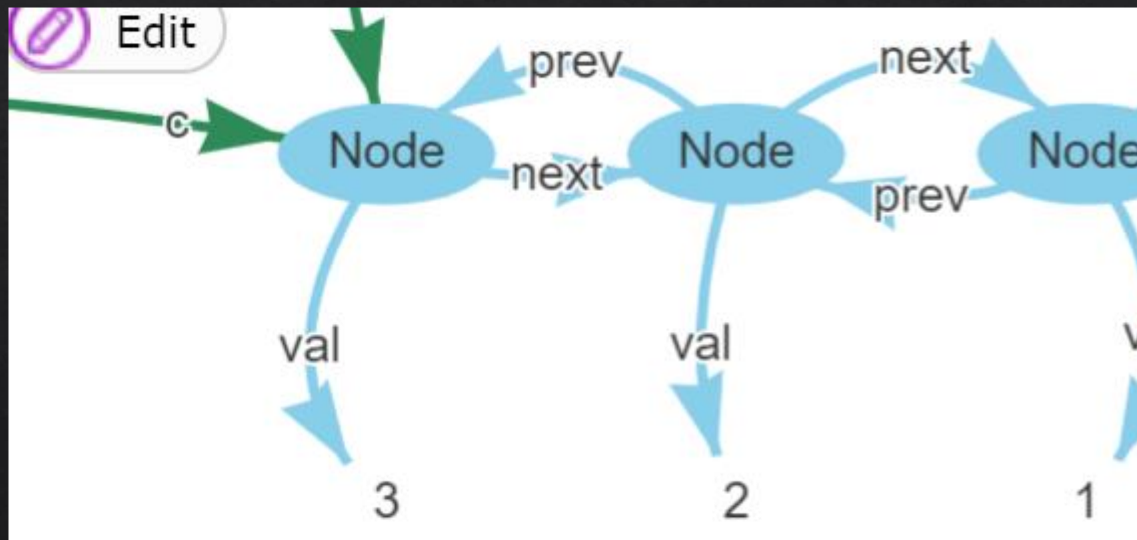
Dealing with Errors in Kanon

- Types of errors: syntax, runtime, semantic
- Runtime errors can happen at different point of execution
- Expected and unexpected errors

Unique Mistakes with Kanon

- Visualization can lead to a different strategy to solve a problem

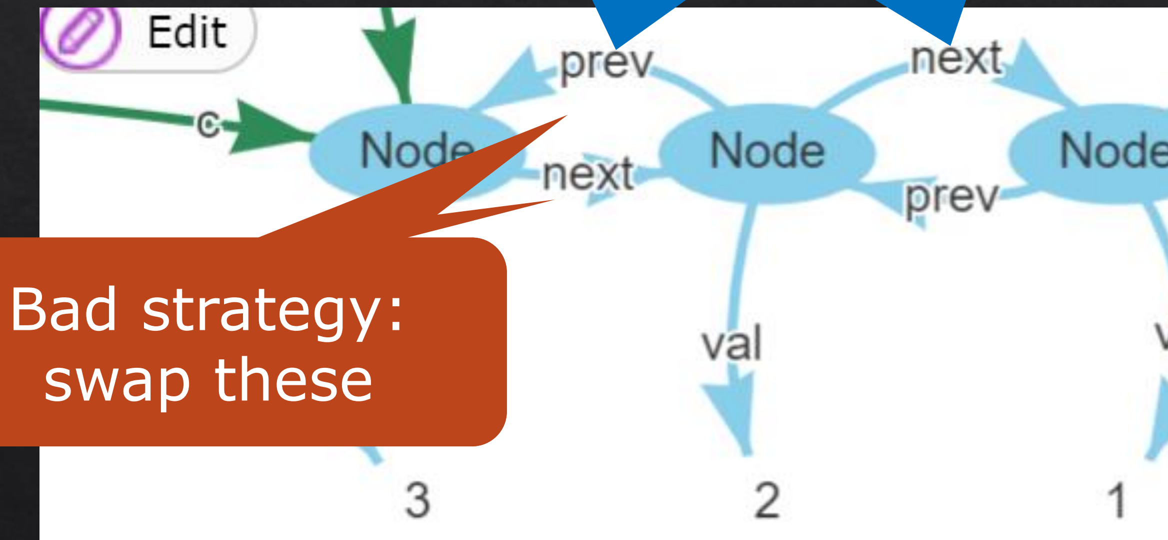
Demo: destructively reverse a double-linked list



Unique Mistakes with Kanon

- With Kanon, some people make a mistake that we don't do with a textual env.

Good strategy: swap these



Bad strategy: swap these

Programming Experiences of Data Structures with Kanon

read our Onward!'18 paper for more research stuff behind

9

Observe-write-confirm

```
6  swap() {
7      let n = this.next;
8      this.next = this.next.next;
9      |
10 }
11
```

- Haven't you drawn a similar diagram in your mind?

14

Program-by-Demonstration

- given an edit with an arrow from $o1$ to $o2$ in the diagram
- search paths $p1$,$p2$ from local variables to $o1$ and $o2$ to have candidate "$p1.f = p2$"
- remaining challenges:
  - method calls
  - multiple edits

```
swap() {
    let n = this.next;
```

18

Summarized View & "Who did This?"

- Summarized view
  - shows the object graph at the end
  - highlights effects of the current code
- "Who did this?": clicking a node/edge moves the cursor to the responsible code

23

On-the-fly Expectation for Top-down Prog./Recursion

1. call to a unfinished function
2. manually create an expected state
3. continue coding wrt the expected state
4. expected state serves as a test case

```
 constructor(val,next) {
   this.val = v...
   this.next=n...
 }
 swap() {...}
 rev() {
   var next_rev = this.next.rev();
 }

 ls = new Node(0,null);
 = new Node(1,ls);
 = new Node(2,ls);
 = new Node(3,ls);

 s = ls.swap();
```

When can LP be useful?

- when we can ***recognize*** results immediately
  - drawing / playing / animated games / …
- when we cannot ***imagine*** results immediately
  - rect(100,200,10,50)

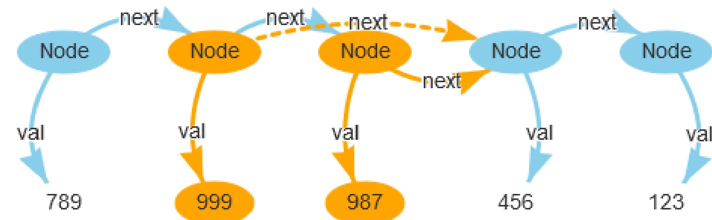
(hypothesis)

LP is useful when we can
- hardly imagine results
- easily recognize results

Kanon Live Programming Env. [Onward!'18]

- Target language: Javascript (but not
- https://github.com/prg-titech/Kanon
- (Demo)

```
class Node {
    constructor(val, next) {
        this.val = val;
        this.next = next;
    }
    insertAfter(pos, value){
        if (pos==0) {
        var n = new Node(value, this.next);
        this.next = n;
        } else {
            this.next.insertAfter(pos-1, valu
        }
    }
}

var x = new Node(123,null);
var y = new Node(456,x);
var z = new Node(789, y);
z.insertAfter(0, 999);
z.insertAfter(1, 987);
```



Research Topics on Live Data Structure Programming

- **How to visualize data structures**
- Programming style
- **Linking between code and visualization**
- Scalability
- Usability evalutaion

Visualizing Data Structures

Goal: diagrams close to ***mental image***

Issues

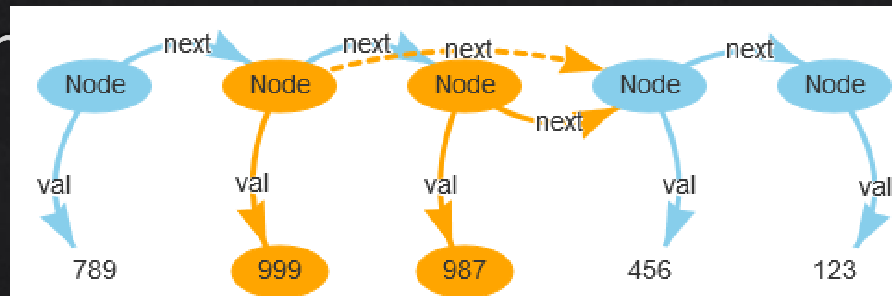
- How should they look like? →
- Which data? →
- How to show changes? →

in Kanon

- object diagram
- everything (spatially) snapshot/overall (temporally)
- tricks
  - **mental map preservation**
  - animation
  - coloring effects

How should they look like?

- Goal: close to programmer's mental images
- Challenges:
  - Can we draw without human intervention?
  - Or, how much can we ask programmers to help?
- in Kanon:
  - Object graph ← common in textbooks
  - force-directed layout + list/tree support
  - (future) customization



Which data?

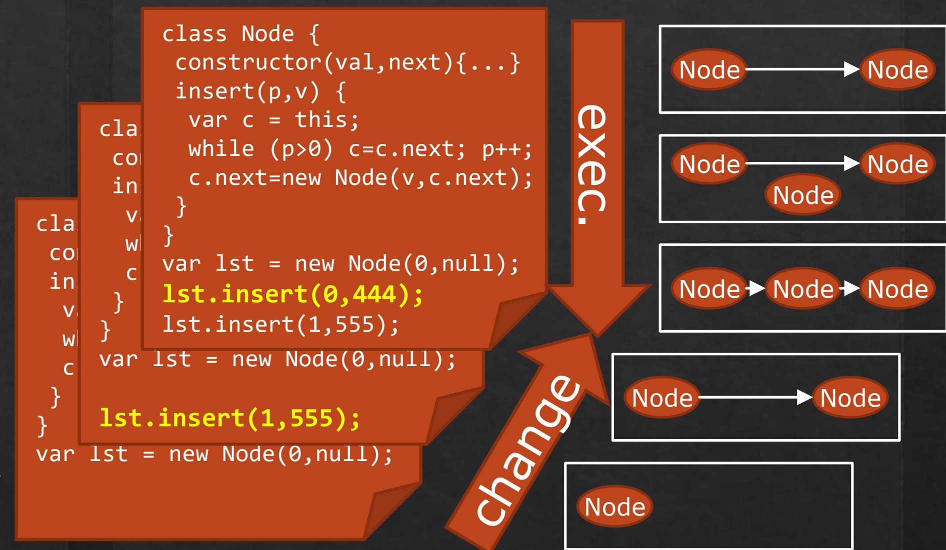
- Goal: close to programmer's mental images
- Issues:
  - Programmers are usually interested in only a part of data
  - How to pick up such a pat?
- in Kanon:
  - draw everything
  - (future) selection by focused test cases[Imai'15] (cf. Example-Centric Programming [Edwars'04])
  - (future) changing levels of details

How to show changes?

Issues:

2 types of chages

- Data changes during execution →at which pt f exec.?
- Changes caused by program changes →two different runs →non trivial correspondences



How to show changes?

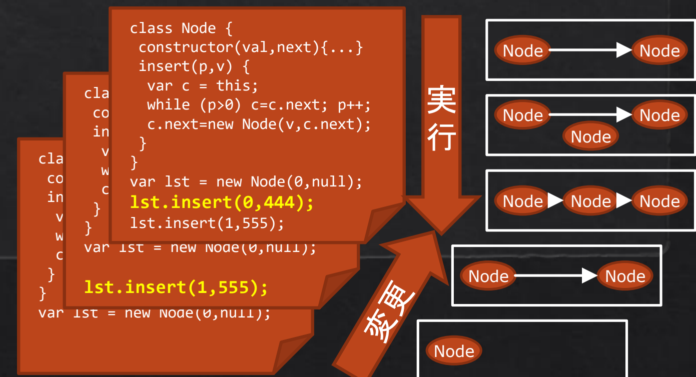
Issues:

2 types of chages

- Data changes during execution
  →at which pt f exec.?
- Changes caused by program changes
  →two different runs
  →non trivial correspondences

in Kanon:

- State at cursor pos. + context selection / summarize view
- Calling context sensitive identification algorithm



Calling context sensitive object identification upon program changes

Issue: Object layout after program changes?

- Programmer: changes only small parts → stable unchanged parts
  may relocate by hand
- Env.: executes the changed program in a new session → need to match objects
- eg:

```
var stack = ...;
stack.push(1);
stack.push(777);
stack.push(2);
stack.push(3);
```

← insert

- serial nr./line nr. won't work well
- edited code may affect indirectly

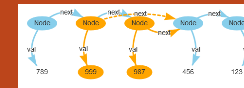
Calling context sensitive object identification upon program changes

- Assumption: editors monitor code changes → identify the same experssions
- Preproc.: add counters at funcalls
- Exec.: record ID of new'ed objects
  ID = pos. of new + list of (caller pos.+counter)
- Layout: preserve positions of objects with the same ID
- Properties: robust agains edits (name changes) / can cope with indirect effects

Linking visualization and code

- Visualization immediately reflects code changes → recognized as the same

mental image

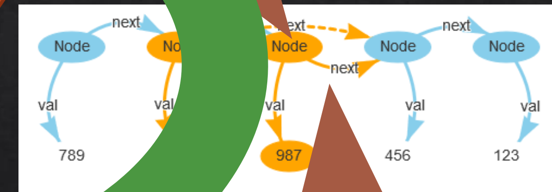


```
1  class Node {
2      constructor(val, next) {
3          this.val = val;
4          this.next = next;
5      }
6      insertAfter(pos, value){
7          if (pos==0) {
8          var n = new Node(value, this.
9          this.next = n;
10         } else {
11             this.next.insertAfter(pos-1, val
12         }
13     }
14 }
15
16 var x = new Node(123,null);
17 var y = new Node(456,x);
```

edit

read

visualize



why this link?

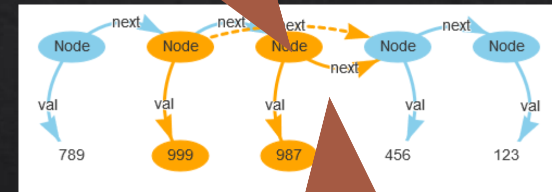
want to change here!

Reflect actions on the vis. into code

Linking visualization and code

- Jump to Construction [Lieberman95]
  Click on an element → jumps to the causal exp.
- Programming by demonstration
  Edit on elements → generate an ex
- (future) showing non object values on obj. graphs

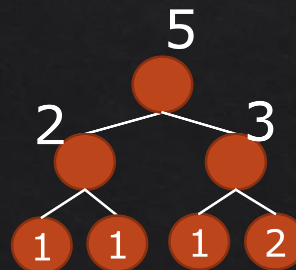
why this link?



want to change here!

1,1,2,1,2,3,5

```
class Tree {
 sum() {
   var sum = this.left.sum()
           + this.right.sum();
   return sum;}}
```



[1] H. Lieberman&C. Fry, Bridging the Gulf Between Code and Behavior in Programming, *CHI*, 1995.

Further opportunities

- Useful in debugging? → need scalability
  - visualization: abstraction, fish-eye view,...
  - runtime overhead: differential execution?
- Useful for teaching?
  - need to learn from previous work on algorithm animation, software visualization
  - difficult to measure effectinevess
- for oher languages?
  - development as a language framework
- Does LP change programmer's behavior?