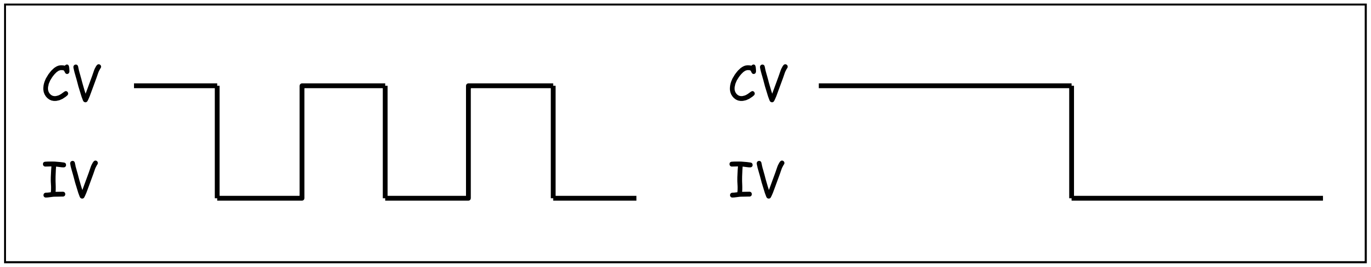
Client View First: An Exodus From Implementation-Biased Teaching

Timothy J. Long, Bruce W. Weide, Paolo Bucci
Computer and Information Science
The Ohio State University
Columbus, OH 43210
{long,weide,bucci}@cis.ohio-state.edu

Murali Sitaraman
Computer Science and Electrical Engineering
West Virginia University
Morgantown, WV 26506
murali@csee.wvu.edu

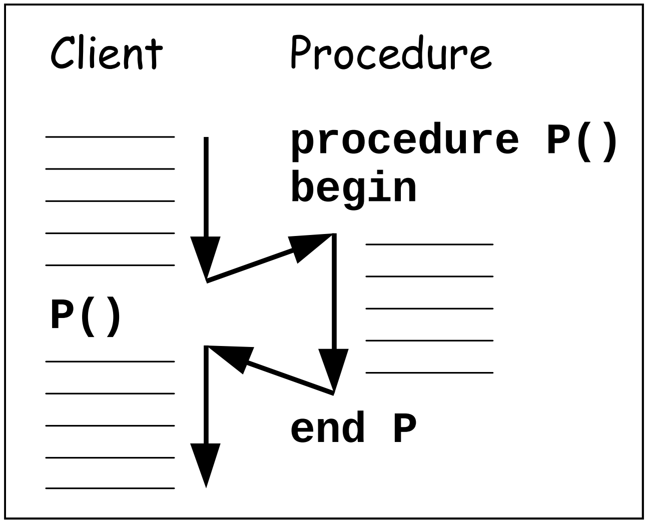
Client View vs. Implementer View

- Fundamental to system thinking (e.g., ADTs)
- Distinct mental models
- Client view *first*



Our Problem

- Teaching how to use programming language features (e.g., procedures, functions, parameter passing, inheritance, and recursion) without revealing how they are implemented



Two Distinct Questions...

- What cover story best enables someone to ***use***, and to ***reason*** about programs that involve, a certain feature?
- How is the feature ***implemented***?

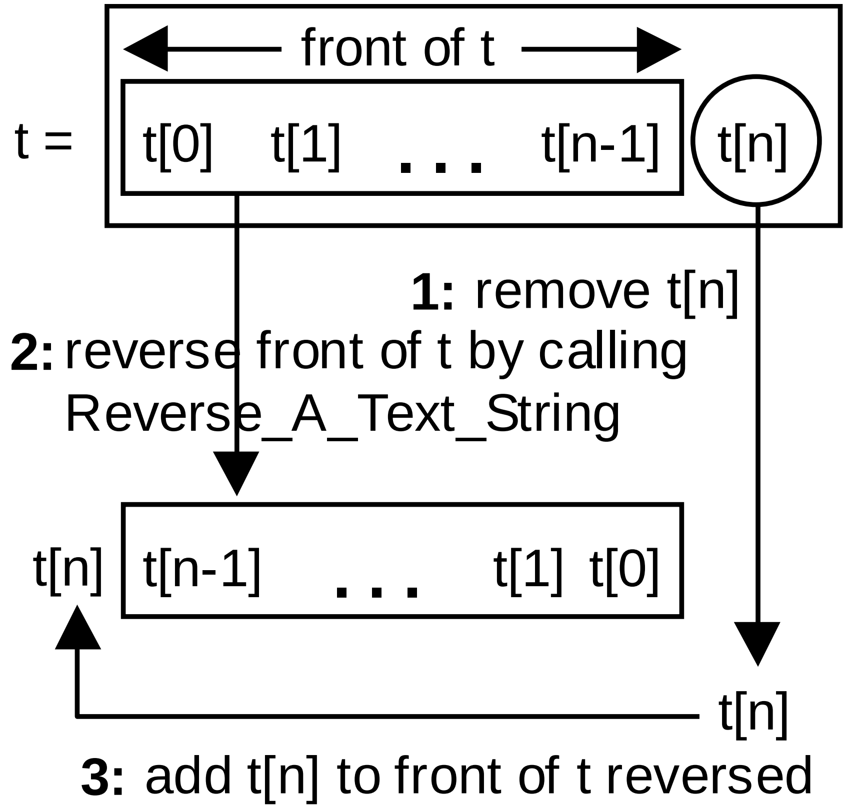
...Two Independent Answers!

An Example: Recursion

- Client view: How do we use recursion? How do we reason about it? Why does it work?
- Implementer view: How does it work?

How We *Use* Recursion

```
operation Reverse_A_Text_String (
        alters t: Text
    )
  ensures
    t = REVERSE (#t)
begin
  if (Length (t) > 0) then
    variable ch: Character
    Remove (t, Length (t)-1, ch)
    Reverse_A_Text_String (t)
    Add (t, 0, ch)
  end if
end Reverse_A_Text_String
```



Step IV: Converting picture to recursive solution

How We *Reason* About Recursion

Statement	**Object Values**
	t="abc"
if (Length (t) > 0) **then** **variable** ch: Character	
	t="abc" ch=' '
Remove (t, Length (t)-1, ch)	
	t="ab" ch='c'
Reverse_A_Text_String (t)	
	t="ba" ch='c'
Add (t, 0, ch)	
	t="cba" ch='c'
end if	
	t="cba"

Why Does Recursion Work?

- Unroll recursive calls
  - What about client view?
- Use inductive proof
  - Can students understand it?

Why Does *Induction* Work?

- E.g., for the natural numbers:
  - establish truth at 0; then
  - establish that truth at 0 implies truth at 1 and thereby conclude truth at 1; then
  - establish that truth at (0 and) 1 implies truth at 2 and thereby conclude truth at 2; then
  - . . .

Why Recursion Works (|t|=0)

|t|	RATS
0	?

Statement	Object Values
	t=""
if (Length (t) > 0) **then** **variable** ch: Character	
Remove (t, Length (t)-1, ch)	
Reverse_A_Text_String (t)	
Add (t, 0, ch)	
end if	
	t=""

Why Recursion Works (|t|=1)

|t|	RATS
0	✓
1	?

Statement	Object Values
	t="a"
if (Length (t) > 0) **then** **variable** ch: Character	
	t="a" ch=' '
Remove (t, Length (t)-1, ch)	
	t="" ch='a'
Reverse_A_Text_String (t)	
	t="" ch='a'
Add (t, 0, ch)	
	t="a" ch='a'
end if	
	t="a"

Why Recursion Works (|t|=2)

|t|	RATS
0	✓
1	✓
2	?

Statement	Object Values
	t="ab"
if (Length (t) > 0) **then** **variable** ch: Character	
	t="ab" ch=' '
Remove (t, Length (t)-1, ch)	
	t="a" ch='b'
Reverse_A_Text_String (t)	
	t="a" ch='b'
Add (t, 0, ch)	
	t="ba" ch='b'
end if	
	t="ba"

Why Recursion Works (|t|=3)

|t|	RATS
0	✓
1	✓
2	✓
3	✓
.	.
.	.
.	.

Statement	Object Values
	t="abc"
if (Length (t) > 0) **then** **variable** ch: Character	
	t="abc" ch=' '
Remove (t, Length (t)-1, ch)	
	t="ab" ch='c'
Reverse_A_Text_String (t)	
	t="ba" ch='c'
Add (t, 0, ch)	
	t="cba" ch='c'
end if	
	t="cba"

What's Good About This Client View?

- Simple and clean
- No reference to implementation details
- Consistent with inductive reasoning
- Encourages faith in inductive reasoning

How Does Recursion Work?

- Unroll recursive calls (it's about time!)

How Recursion Works

Statement	**Object Values**
	t="abc"
if (Length (t) > 0) **then** **variable** ch: Character	
	t="abc" ch=' '
Remove (t, Length (t)-1, ch)	
	t="ab" ch='c'
Reverse_A_Text_String (t)	
Add (t, 0, ch)	
end if	

How Recursion Works

Statement	Object Values
	t="ab"
if (Length (t) > 0) **then** **variable** ch: Character	
	t="ab" ch=' '
Remove (t, Length (t)-1, ch)	
	t="a" ch='b'
Reverse_A_Text_String (t)	
Add (t, 0, ch)	
end if	

Statement	Object Values
	t="abc"
if (Length (t) > 0) **then** **variable** ch: Character	
	t="abc" ch=' '
Remove (t, Length (t)-1, ch)	
	t="ab" ch='c'
Reverse_A_Text_String (t)	
Add (t, 0, ch)	
end if	

How Recursion Works

Statement	Object Values
	t="ab"
if (Length (t) > 0) **then** **variable** ch: Character	
	t="ab" ch=' '
Remove (t, Length (t)-1, ch)	
	t="a" ch='b'
Reverse_A_Text_String (t)	
Add (t, 0, ch)	
end if	

Statement	Object Values
	t="abc"
if (Length (t) > 0) **then** **variable** ch: Character	
	t="abc" ch=' '
Remove (t, Length (t)-1, ch)	
	t="ab" ch='c'
Reverse_A_Text_String (t)	
Add (t, 0, ch)	
end if	

Statement	Object Values
	t="a"
if (Length (t) > 0) **then** **variable** ch: Character	
	t="a" ch=' '
Remove (t, Length (t)-1, ch)	
	t="" ch='a'
Reverse_A_Text_String (t)	
Add (t, 0, ch)	
end if	

How Recursion Works

Statement	Object Values
	t="a"
if (Length (t) > 0) **then** **variable** ch: Character	
	t="a" ch=' '
Remove (t, Length (t)-1, ch)	
	t="" ch='a'
Reverse_A_Text_String (t)	
Add (t, 0, ch)	
end if	

Statement	Object Values
	t="ab"
if (Length (t) > 0) **then** **variable** ch: Character	
	t="ab" ch=' '
Remove (t, Length (t)-1, ch)	
	t="a" ch='b'
Reverse_A_Text_String (t)	
Add (t, 0, ch)	
end if	

Statement	Object Values
	t="abc"
if (Length (t) > 0) **then** **variable** ch: Character	
	t="abc" ch=' '
Remove (t, Length (t)-1, ch)	
	t="ab" ch='c'
Reverse_A_Text_String (t)	
Add (t, 0, ch)	
end if	

Statement	Object Values
	t=""
if (Length (t) > 0) **then** **variable** ch: Character	
Remove (t, Length (t)-1, ch)	
Reverse_A_Text_String (t)	
Add (t, 0, ch)	
end if	
	t=""

How Recursion Works

Statement	Object Values
	t="ab"
if (Length (t) > 0) **then** **variable** ch: Character	
	t="ab" ch=' '
Remove (t, Length (t)-1, ch)	
	t="a" ch='b'
Reverse_A_Text_String (t)	
Add (t, 0, ch)	
end if	

Statement	Object Values
	t="abc"
if (Length (t) > 0) **then** **variable** ch: Character	
	t="abc" ch=' '
Remove (t, Length (t)-1, ch)	
	t="ab" ch='c'
Reverse_A_Text_String (t)	
Add (t, 0, ch)	
end if	

Statement	Object Values
	t="a"
if (Length (t) > 0) **then** **variable** ch: Character	
	t="a" ch=' '
Remove (t, Length (t)-1, ch)	
	t="" ch='a'
Reverse_A_Text_String (t)	
	t="" ch='a'
Add (t, 0, ch)	
	t="a" ch='a'
end if	
	t="a"

How Recursion Works

Statement	Object Values
	t="abc"
if (Length (t) > 0) **then** **variable** ch: Character	
	t="abc" ch=' '
Remove (t, Length (t)-1, ch)	
	t="ab" ch='c'
Reverse_A_Text_String (t)	
Add (t, 0, ch)	
end if	

Statement	Object Values
	t="ab"
if (Length (t) > 0) **then** **variable** ch: Character	
	t="ab" ch=' '
Remove (t, Length (t)-1, ch)	
	t="a" ch='b'
Reverse_A_Text_String (t)	
	t="a" ch='b'
Add (t, 0, ch)	
	t="ba" ch='b'
end if	
	t="ba"

How Recursion Works

Statement	Object Values
	t="abc"
if (Length (t) > 0) **then** **variable** ch: Character	
	t="abc" ch=' '
Remove (t, Length (t)-1, ch)	
	t="ab" ch='c'
Reverse_A_Text_String (t)	
	t="ba" ch='c'
Add (t, 0, ch)	
	t="cba" ch='c'
end if	
	t="cba"

What's Good About This Implementer View?

- Simple and clean
- Consistent with client view
- Consistent with how recursion is actually implemented

Ideas About Recursion

- Emphasize methods and strategies to develop recursive solutions
- Use pictures to visualize recursive structure and process
- Explain why recursion works with bottom-up inductive argument
- Formalize manual execution with tracing tables

Summary And Conclusions

- Distinction between client view and implementer view is fundamental
- Client view should come first
- Specifications are essential
- Look for other instances of this problem and come up with reasonable client view (and possibly improved implementer view)

Go Bucks!!!