Week 8 Discussion

Liu Yang
lyang422@wisc.edu

This week's topic

- **Recursive algorithm**
- **Program correctness** (for recursive program)

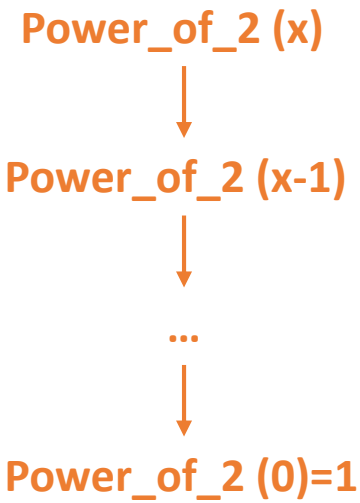
To understand recursion, you first have to understand recursion.

This week's topic

- **Recursive algorithm**
  - An algorithm that solves a problem by reducing it to one or more instances of the same problem with smaller input.

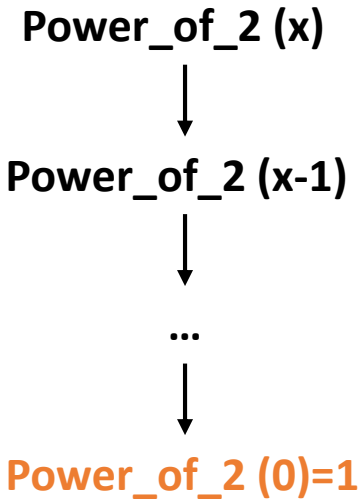
Algorithm **Power_of_2 (** natural number **x)**
Input: ***x***, a natural number
Output: ***x***-th power of ***2***

if ***x = 0***, **then** return ***1***;
else return **2*Power_of_2(*x - 1*)**



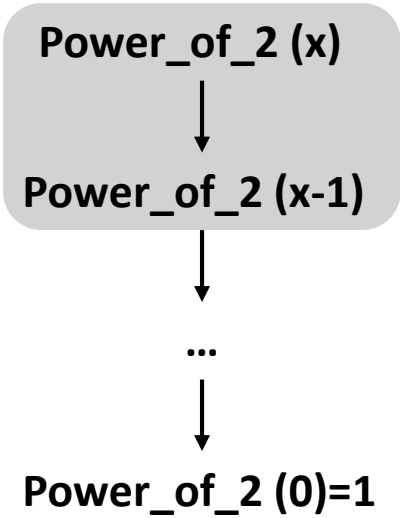
This week's topic

- **Program correctness** (for recursive program) - **Partial correctness**
  show that for all valid inputs *x*
- **base case**: correct result is returned in non-recursive case (assuming program terminates)
- **recursive step** (or **recursive call**):
- valid input is provided to all recursive calls (if any) made by program with input *x*
- assuming the recursive calls return the correct output and assuming the program terminates, the program produces the correct output on input *x*



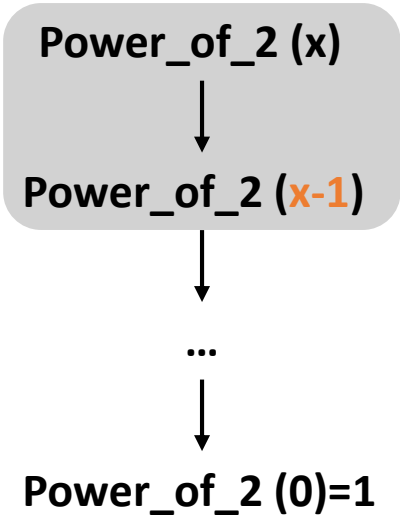
This week's topic

- **Program correctness** (for recursive program) - **Partial correctness**
  show that for all valid inputs *x*
- **base case**: correct result is returned in non-recursive case (assuming program terminates)
- **recursive step** (or **recursive call**):
- valid input is provided to all recursive calls (if any) made by program with input *x*
- assuming the recursive calls return the correct output and assuming the program terminates, the program produces the correct output on input *x*



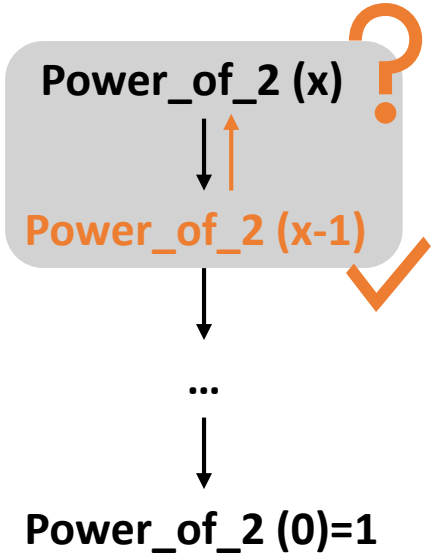
This week's topic

- **Program correctness** (for recursive program) - **Partial correctness**
  show that for all valid inputs *x*
- **base case**: correct result is returned in non-recursive case (assuming program terminates)
- **recursive step** (or **recursive call**):
- valid input is provided to all recursive calls (if any) made by program with input *x*
- assuming the recursive calls return the correct output and assuming the program terminates, the program produces the correct output on input *x*



This week's topic

- **Program correctness** (for recursive program) - **Partial correctness**
  show that for all valid inputs *x*
- **base case**: correct result is returned in non-recursive case (assuming program terminates)
- **recursive step** (or **recursive call**):
- valid input is provided to all recursive calls (if any) made by program with input *x*
- assuming the recursive calls return the correct output and assuming the program terminates, the program produces the correct output on input *x*

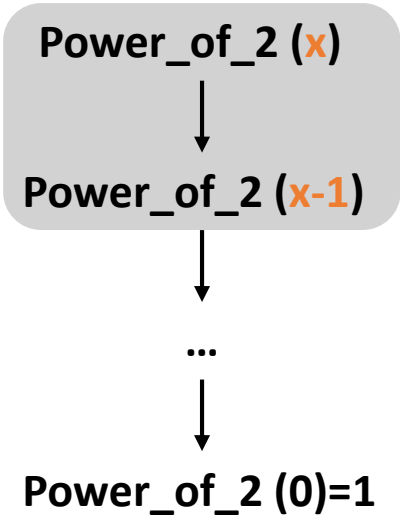


This week's topic

- **Program correctness** (for recursive program) – **Termination**

show that the chain of recursive calls eventually ends

- proved by induction on some quantity (that depends on the input) that **decreases** with each recursive call
- note: decreases = heads towards base (non-recursive) case(s)



	Iterative programs	Recursive programs
	Prove loop invariants using induction on # times loop executes	
Partial correctness	Assume input is valid and program terminates. Use loop invariants to argue program returns correct result.	Assume input is valid and program terminates. *Base case*: Show correct result is returned. *Recursive case:* - Show input to recursive call is valid (i.e., meets program input specification). - Assume recursive call returns correct result (for its input). Show what program does with result of recursive call computes the correct result.
Termination	Use loop invariants to argue that the loop must terminate (i.e., no infinite loops): consider some quantity that decreases on each iteration of the loop and showing that it has a lower bound	Show that the chain of recursive calls must eventually end: use induction on some quantity that decreases with each recursive call (i.e., heads towards the base case(s))

Discussion Handout

- Give the link …

isPalindrome(A, b, e)

- Input: String A, indices b and e
- Output: True if A[b…e] is a palindrome, false otherwise

noon ⇌

- some edge cases:
- “x” is a palindrome
- “ ” (empty string) is a palindrome

isPalindrome(A, b, e)

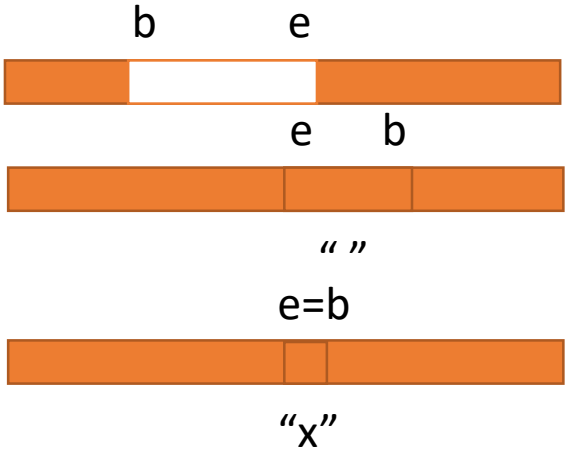
- Input: String A, indices b and e
- Output: True if A[b…e] is a palindrome, false otherwise

1. if b $\geq$ e, then return True
2. if A[b] $\neq$ A[e], then return False
3. else return isPalindrome(A, b+1, e-1)

isPalindrome(A, b, e)

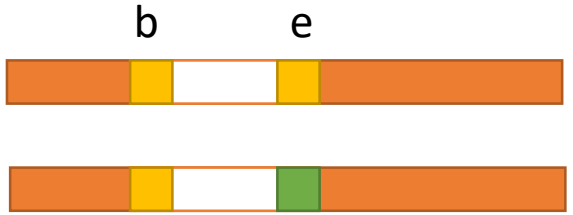
- Input: String A, indices b and e
- Output: True if A[b…e] is a palindrome, false otherwise

1. if b ≥ e, then return True
2. if A[b] ≠ A[e], then return False
3. else return isPalindrome(A, b+1, e-1)



isPalindrome(A, b, e)

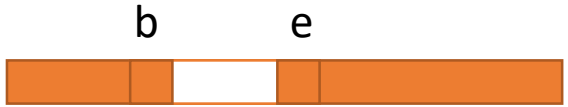
- Input: String A, indices b and e
- Output: True if A[b…e] is a palindrome, false otherwise



1. if b ≥ e, then return True
2. if A[b] ≠ A[e], then return False
3. else return isPalindrome(A, b+1, e-1)

isPalindrome(A, b, e)

- Input: String A, indices b and e
- Output: True if A[b…e] is a palindrome, false otherwise



1. if b ≥ e, then return True
2. if A[b] ≠ A[e], then return False
3. else return isPalindrome(A, b+1, e-1)

Problem 1

- *IsPalindrome*("racecar", 0, 6)

0	1	2	3	4	5	6		b ≥ e ?	A[b] ≠ A[e] ?
r	a	c	e	c	a	**r**	*IsPalindrome*("racecar", 0, 6)	*No*	*No*

Problem 1

- *IsPalindrome*("racecar", 0, 6)

0	1	2	3	4	5	6		b ≥ e ?	A[b] ≠ A[e] ?
r	a	c	e	c	a	**r**	*IsPalindrome*("racecar", 0, 6)	*No*	*No*
r	**a**	c	e	c	**a**	r	*IsPalindrome*("racecar", 1, 5)	*No*	*No*

Problem 1

- *IsPalindrome*("racecar", 0, 6)

0	1	2	3	4	5	6		b ≥ e ?	A[b] ≠ A[e] ?
r	a	c	e	c	a	**r**	*IsPalindrome*("racecar", 0, 6)	*No*	*No*
r	**a**	c	e	c	**a**	r	*IsPalindrome*("racecar", 1, 5)	*No*	*No*
r	a	**c**	e	**c**	a	r	*IsPalindrome*("racecar", 2, 4)	*No*	*No*

Problem 1

- *IsPalindrome*("racecar", 0, 6)

0	1	2	3	4	5	6		b ≥ e ?	A[b] ≠ A[e] ?
r	a	c	e	c	a	**r**	*IsPalindrome*("racecar", 0, 6)	*No*	*No*
r	**a**	c	e	c	**a**	r	*IsPalindrome*("racecar", 1, 5)	*No*	*No*
r	a	**c**	e	**c**	a	r	*IsPalindrome*("racecar", 2, 4)	*No*	*No*
r	a	c	**e**	c	a	r	*IsPalindrome*("racecar", 3, 3)	*Yes*	

Problem 1

- *IsPalindrome*("racecar", 0, 6)

0	1	2	3	4	5	6		b ≥ e ?	A[b] ≠ A[e] ?	
r	a	c	e	c	a	**r**	*IsPalindrome*("racecar", 0, 6)	*No*	*No*	
r	**a**	c	e	c	**a**	r	*IsPalindrome*("racecar", 1, 5)	*No*	*No*	
r	a	**c**	e	**c**	a	r	*IsPalindrome*("racecar", 2, 4)	*No*	*No*	
r	a	c	**e**	c	a	r	*IsPalindrome*("racecar", 3, 3)	*Yes*		**True**

Problem 1

- *IsPalindrome*("racecar", 0, 6)

Call	A	b	e	result
IsPalindrome("racecar", 0, 6)	"racecar"	0	6	since A[0]=r=A[6], return IsPalindrome("racecar", 1, 5)
IsPalindrome("racecar", 1, 5)	"racecar"	1	5	since A[1]=a=A[5], return IsPalindrome("racecar", 2, 4)
IsPalindrome("racecar", 2, 4)	"racecar"	2	4	since A[2]=c=A[4], return IsPalindrome("racecar", 3, 3)
IsPalindrome("racecar", 3, 3)	"racecar"	3	3	since b=e, return True

Group Discussion !

- We assume that if b, e are valid indices to string A, then

  $\forall x \in \mathbb{N}, b \leq x \leq e$, x is valid index to string A.

- If $x, y \in \mathbb{N},\ x < y$, then $x \leq y - 1$
- Prove by cases.

Problem 1

- *IsPalindrome*("noon", 0, 3)

Call	A	b	e	result
IsPalindrome("noon", 0, 3)	"noon"	0	3	since A[0]=n=A[3], return IsPalindrome("noon", 1, 2)
IsPalindrome("noon", 1, 2)	"noon"	1	2	since A[1]=o=A[2], return IsPalindrome("noon", 2, 1)
IsPalindrome("noon", 2, 1)	"noon"	2	1	since b > e, return True

Problem 1

- *IsPalindrome*("skulks", 0, 5)

Call	A	b	e	result
IsPalindrome("skulks", 0, 5)	"skulks"	0	5	since A[0]=s=A[5], return IsPalindrome("skulks", 1, 4)
IsPalindrome("skulks", 1, 4)	"skulks"	1	4	since A[1]=k=A[4], return IsPalindrome("skulks", 2, 3)
IsPalindrome("skulks", 2, 3)	"skulks"	2	3	since A[2]=u≠A[3], return False

Problem 2: partial correctness

- *Assume input is valid, i.e. b, e are valid indices for string A*
- Base case:
  - b ≥ e:
    - if b = e, then A[b] = A[e] is a palindrome, and the program returns true.
    - if b > e, then A[b..e] is the empty string, which is a palindrome.
    - Thus the program returns True, which is correct.

Problem 2: partial correctness

- *Assume input is valid, i.e. b, e are valid indices for string A*
- Base case:
  - A[b] ≠ A[e]:
    - if A[b] ≠ A[e], then A[b..e] is not a palindrome.
    - Thus the program returns False, which is correct.

Problem 2: partial correctness

- Recursive case:
  - Need to show Valid input provided to recursive calls: IsPalindrome(A, b+1, e-1)
  - It must be the case
    - b < e and,
    - A[b] = A[e]
  - Since b, e are integers, b+1 $\leq$ e and e-1 $\geq$ b.
  - If b and e are valid indices, then b+1 and e-1 must also be valid indices since

    $$b < b+1 \leq e \text{ and } b \leq e-1 < e$$

  - Thus the input to the recursive call is valid.

Problem 2: partial correctness

- Recursive case:
  - Assume IsPalindrome(A, b+1, e-1) provide correct result.
  - Case: IsPalindrome(A, b+1, e-1) returns False.
    - Then A[b+1..e-1] is not a palindrome and neither is A[b..e]
    - So the program returns the correct result.
  - Case: IsPalindrome(A, b+1, e-1) returns True.
    - Then A[b+1..e-1] is a palindrome.
    - Since A[b] = A[e], that means A[b..e] is a palindrome as well.
    - So the program returns the correct result.

Problem 2: partial correctness

- Recursive case:
  - In all cases the program returns the correct result, assuming input is valid and it terminates.

Problem 2: termination

1. if b $\geq$ e, then return True
2. if A[b] $\neq$ A[e], then return False
3. else return isPalindrome(A, b+1, e-1)

- Consider $e - b$. We show by induction on $e - b$ that the program terminates.
- **Base Case:** P(0), $e - b \leq 0$
- If $e - b \leq 0$, then $e \leq b$ and the program terminates on line (1).

Problem 2: termination

1. if b $\geq$ e, then return True
2. if A[b] $\neq$ A[e], then return False
3. else return isPalindrome(A, b+1, e-1)

- Consider $e - b$. We show by induction on $e - b$ that the program terminates.
- **Inductive step**:
- **Induction hypothesis**: program terminates when $e - b \leq k$.
- Show program terminates when $e - b = k + 1$.
- When $e - b = k + 1$, either line (2) or line (3) will execute.
- Case 1: line (2) executes: Then the program returns False and terminates.

Problem 2: termination

1. if b $\geq$ e, then return True
2. if A[b] $\neq$ A[e], then return False
3. else return isPalindrome(A, b+1, e-1)

- Consider $e - b$. We show by induction on $e - b$ that the program terminates.
- Case 2: line (3) executes. Then the program calls isPanlindrome(A, b+1, e-1). Since

$$(e-1)-(b+1) = e-b-2 = k+1-2 = k-1 \leq k$$

- Thus by induction hypothesis (program terminates when $e - b \leq k$), we know isPanlindrome(A, b+1, e-1) terminates. Since isPanlindrome(A, b, e) just returns the result of the recursive call, the program terminates.
- Thus the program terminates when $e - b = k + 1$.

Problem 2: termination

- Thus by strong induction, the program terminates on valid input.

Discussion Participation

- Sec #
- Come up with a string which result in 3 iterations for the isPanlindrome program.
- See example “noon”.

Reference

- Week 8 information page:

https://canvas.wisc.edu/courses/212414/pages/week-8-discussion-info-mlfmo