SE 2AA4, CS 2ME3 (Introduction to Software Development)

Winter 2018

10 Abstract Data Types (Ghezzi Ch. 4)

Dr. Spencer Smith

Faculty of Engineering, McMaster University

January 28, 2018

10 Abstract Data Types (Ghezzi Ch. 4)

- Administrative details
- Implementation of a sequence abstract object
- Specification of abstract data types
- Example (similar to A2, 2017)
  - PointADT
  - LineADT
  - CircleADT
  - Deque

Administrative Details

- Assignment 1
  - Partner Files: January 28, 2018
  - Part 2: January 31, 2018
- Questions on assignment?
- NSERC USRA
  - Summer research positions available to top undergrads
  - Details on NSERC's website
  - Some interesting projects will be posted on Avenue
  - You can approach faculty members about other projects
  - Application deadline is Friday, February 9

Homework: Abstract Objects in Python

H&S versus Python for s = [4, 6, -2, 8, 11]

- H&S for s[1:3]?
- Python for s[1:3]?
- H&S for s[0:-1]?
- Python for s[0:-1]?
- H&S for s[0:0]?
- Python for s[0:0]?

Homework: Abstract Objects in Python

H&S versus Python for s = [4, 6, -2, 8, 11]

- H&S for s[1:3]? [6, -2, 8]
- Python for s[1:3]? [6, -2]
- H&S for s[0:-1]?
- Python for s[0:-1]?
- H&S for s[0:0]?
- Python for s[0:0]?

Homework: Abstract Objects in Python

H&S versus Python for s = [4, 6, -2, 8, 11]

- H&S for s[1:3]? [6, -2, 8]
- Python for s[1:3]? [6, -2]
- H&S for s[0:-1]? []
- Python for s[0:-1]? [4, 6, -2, 8]
- H&S for s[0:0]?
- Python for s[0:0]?

Homework: Abstract Objects in Python

H&S versus Python for s = [4, 6, -2, 8, 11]

- H&S for s[1:3]? [6, -2, 8]
- Python for s[1:3]? [6, -2]
- H&S for s[0:-1]? []
- Python for s[0:-1]? [4, 6, -2, 8]
- H&S for s[0:0]? [4]
- Python for s[0:0]? []

Homework: Abstract Objects in Python

See the sample files Python in the repo and compare to Sequence specification.

Specification of ADTs

- Similar template to abstract objects
- "Template Module" as opposed to "Module"
- "Exported Types" that are abstract use a ?
  - pointT = ?
  - pointMassT = ?
- Access routines know which abstract object called them
- Use "self" to refer to the current abstract object
- Use a dot "." to reference methods of an abstract object
  - *p*.xcoord()
  - *self*.*pt*.dist(*p*.point())
- Similar notation to Python or Java

Syntax Point ADT Module

Template Module

pointADT

Uses

N/A

Exported Types

pointT = ?

Syntax Point ADT Module Continued

Routine name	**In**	**Out**	**Exceptions**
new pointT	real, real	pointT	
xcoord		real	
ycoord		real	
dist	pointT	real	
rotate	real		

Semantics Point ADT Module

State Variables

xc: real
yc: real

State Invariant

None

Assumptions

None

Access Routine Semantics Point ADT Module

new pointT (x, y):

- transition: $xc, yc := x, y$
- output: ?
- exception: none

xcoord:

- output: $out := xc$
- exception: none

ycoord:

- output: $out := yc$
- exception: none

Access Routine Semantics Point ADT Module

new pointT (x, y):

- transition: $xc, yc := x, y$
- output: $out := self$
- exception: none

xcoord:

- output: $out := xc$
- exception: none

ycoord:

- output: $out := yc$
- exception: none

Semantics Point ADT Module Continued

dist(p):

- output: $out := \sqrt{(xc - p.\text{xcoord})^2 + (yc - p.\text{ycoord})^2}$
- exception: none

rotate(φ):

- φ is in radians
- transition:

$$\begin{bmatrix} xc \\ yc \end{bmatrix} := \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix} \begin{bmatrix} xc \\ yc \end{bmatrix}$$

- exception: none

Syntax Line ADT Module

Template Module

lineADT

Uses

pointADT

Exported Types

lineT = ?

Syntax Line ADT Module Continued

Routine name	**In**	**Out**	**Exceptions**
new lineT	pointT, pointT	lineT	
start		pointT	
end		pointT	
length		real	
midpoint		pointT	
rotate	real		

Semantics Line ADT Module

State Variables

s: pointT
e: pointT

State Invariant

None

Assumptions

None

Access Routine Semantics Line ADT Module

new lineT (p_1, p_2):

- transition: $s, e := p_1, p_2$
- output: $out := self$
- exception: none

start:

- output: $out := s$
- exception: none

end:

- output: $out := e$
- exception: none

Access Routine Semantics Continued

length:

- output: ?
- exception: none

midpoint:

- output: $out :=$

  new pointT(avg(s.xcoord, e.xcoord), avg(s.ycoord, e.ycoord))

- exception: none

rotate (φ):
φ is in radians

- transition: s.rotate(φ), e.rotate(φ)
- exception: none

Access Routine Semantics Continued

length:

- output: $out := s.\text{dist}(e)$
- exception: none

midpoint:

- output: $out :=$

  new pointT(avg(s.xcoord, e.xcoord), avg(s.ycoord, e.ycoord))

- exception: none

rotate (φ):
φ is in radians

- transition: $s.\text{rotate}(\varphi), e.\text{rotate}(\varphi)$
- exception: none

Line ADT Local Functions

Local Functions

avg: real $\times$ real $\rightarrow$ real
avg$(x_1, x_2) \equiv \frac{x_1 + x_2}{2}$

Syntax Circle ADT Module

Template Module

circleADT

Uses

pointADT, lineADT

Exported Types

circleT = ?

Syntax Circle ADT Module Continued

Routine name	**In**	**Out**	**Exceptions**
new circleT	pointT, real	circleT	
centre		pointT	
radius		real	
area		real	
intersect	circleT	boolean	
connection	circleT	lineT	

Semantics Circle ADT Module

State Variables

c: pointT
r: real

State Invariant

None

Assumptions

None

Access Routine Semantics Circle ADT Module

new circleT ($cinput$, $rinput$):

- transition: $c, r := cinput, rinput$
- output: $out := self$
- exception: none

centre:

- output: $out := c$
- exception: none

radius:

- output: $out := r$
- exception: none

area:

- output: $out := \pi r^2$
- exception: none

Access Routine Semantics Continued

intersect(ci):

- output:
  $\exists(p : \text{pointT} | \text{insideCircle}(p, ci) : \text{insideCircle}(p, self))$
- exception: none

connection(ci):

- output: $out := \text{new lineT}(c, ci.centre)$
- exception: none

Circle ADT Local Functions

Local Functions

insideCircle: pointT $\times$ circleT $\rightarrow$ boolean
insideCircle$(p, c) \equiv$?

Circle ADT Local Functions

Local Functions

insideCircle: pointT $\times$ circleT $\rightarrow$ boolean
insideCircle(p, c) $\equiv$ p.dist(c.centre) $\leq$ c.radius

Syntax Deque Of Circles Module

Module

DequeCircleModule

Uses

circleADT

Exported Constants

MAX_SIZE = 20

Syntax Deque Of Circles Module Continued

Routine name	**In**	**Out**	**Exceptions**
init			
pushBack	circleT		FULL
pushFront	circleT		FULL
popBack			EMPTY
popFront			EMPTY
back		circleT	EMPTY
front		circleT	EMPTY
size		integer	
disjoint		boolean	EMPTY
totalArea		real	EMPTY
averageRadius		real	EMPTY

Semantics Deque Of Circles Module

State Variables

s: ?

State Invariant

$|s| \leq$ MAX_SIZE

Assumptions

init() is called before any other access program.

Semantics Deque Of Circles Module

State Variables

s: sequence of circleT

State Invariant

$|s| \leq$ MAX_SIZE

Assumptions

init() is called before any other access program.

Access Routine Semantics Deque Of Circles Module

init():

- transition: $s := <>$
- exception: none

pushBack(c):

- transition: ?
- exception: $exc := (|s| = \text{MAX\_SIZE} \Rightarrow \text{FULL})$

pushFront(c):

- transition: $s := < c > || s$
- exception: $exc := (|s| = \text{MAX\_SIZE} \Rightarrow \text{FULL})$

Access Routine Semantics Deque Of Circles Module

init():

- transition: $s :=<>$
- exception: none

pushBack(c):

- transition: $s := s|| < c >$
- exception: $exc := (|s| = \text{MAX\_SIZE} \Rightarrow \text{FULL})$

pushFront(c):

- transition: $s :=< c > ||s$
- exception: $exc := (|s| = \text{MAX\_SIZE} \Rightarrow \text{FULL})$

Access Routine Semantics Continued

popBack():

- transition: ?
- exception: $exc := (|s| = 0 \Rightarrow \text{EMPTY})$

popFront():

- transition: $s := s[1..|s| - 1]$
- exception: $exc := (|s| = 0 \Rightarrow \text{EMPTY})$

back():

- output: $out := s[|s| - 1]$
- exception: $exc := (|s| = 0 \Rightarrow \text{EMPTY})$

Access Routine Semantics Continued

popBack():

- transition: $s := s[0..|s| - 2]$
- exception: $exc := (|s| = 0 \Rightarrow \text{EMPTY})$

popFront():

- transition: $s := s[1..|s| - 1]$
- exception: $exc := (|s| = 0 \Rightarrow \text{EMPTY})$

back():

- output: $out := s[|s| - 1]$
- exception: $exc := (|s| = 0 \Rightarrow \text{EMPTY})$

Access Routine Semantics Continued

front():

- output: $out := s[0]$
- exception: $exc := (|s| = 0 \Rightarrow \text{EMPTY})$

size():

- output: $out := |s|$
- exception: none

Access Routine Semantics Disjoint

Disjoint returns true if none of the circles in the deque overlap.

What access program tells you whether two circles overlap?

In words how would you express the predicate for disjoint?

disjoint():

- output
  $out := ?\ (?\qquad |?$
  $\quad : ?\qquad\qquad\qquad)$
- exception: $exc := (|s| = 0 \Rightarrow \text{EMPTY})$

Access Routine Semantics Disjoint

Disjoint returns true if none of the circles in the deque overlap.

What access program tells you whether two circles overlap?

In words how would you express the predicate for disjoint?

disjoint():

- output
  $out := ?\ (?\qquad |?$
  $\quad : ?\qquad\qquad\qquad)$
- exception: $exc := (|s| = 0 \Rightarrow \text{EMPTY})$

Access Routine Semantics Disjoint

Disjoint returns true if none of the circles in the deque overlap.

What access program tells you whether two circles overlap?

In words how would you express the predicate for disjoint?

disjoint():

- output
  $out := ?\ (?\quad |?$
  $\quad : ?\quad)$
- exception: $exc := (|s| = 0 \Rightarrow \text{EMPTY})$

Access Routine Semantics Disjoint

Disjoint returns true if none of the circles in the deque overlap.

What access program tells you whether two circles overlap?

In words how would you express the predicate for disjoint?

disjoint():

- output
  $out := \forall(? \quad |? \quad : ? \quad)$
- exception: $exc := (|s| = 0 \Rightarrow \text{EMPTY})$

Access Routine Semantics Disjoint

Disjoint returns true if none of the circles in the deque overlap.

What access program tells you whether two circles overlap?

In words how would you express the predicate for disjoint?

disjoint():

- output
  $out := \forall(? \quad |?$
  $: \neg s[i].\text{intersect}(s[j]))$
- exception: $exc := (|s| = 0 \Rightarrow \text{EMPTY})$

Access Routine Semantics Disjoint

Disjoint returns true if none of the circles in the deque overlap.

What access program tells you whether two circles overlap?

In words how would you express the predicate for disjoint?

disjoint():

- output
  $out := \forall(\ i, j : \mathbb{N} | ?$
  $: \neg s[i].\text{intersect}(s[j]))$
- exception: $exc := (|s| = 0 \Rightarrow \text{EMPTY})$

Access Routine Semantics Disjoint

Disjoint returns true if none of the circles in the deque overlap.

What access program tells you whether two circles overlap?

In words how would you express the predicate for disjoint?

disjoint():

- output
  $out := \ \forall(\ i, j : \mathbb{N} |\ i \in [0..|s| - 1] \wedge j \in [0..|s| - 1] \wedge i \neq j :\ \neg s[i].\text{intersect}(s[j]))$
- exception: $exc := (|s| = 0 \Rightarrow \text{EMPTY})$

Homework: Access Routine Semantics Continued

totalArea():

- output

$$out := ?$$

- exception: $exc := (|s| = 0 \Rightarrow \text{EMPTY})$

averageRadius():

- output

$$out := ?$$

- exception: $exc := (|s| = 0 \Rightarrow \text{EMPTY})$