

Assignment 2 DRAFT

SFWR ENG 2AA4

Files due Feb 15 (11:59 pm), E-mail partner due Feb 16 (11:59 pm), Lab report due Feb 27 (11:59 pm)

The purpose of this software design exercise is to write a Python program that creates, uses, and tests an ADT for points, lines and circles. A module that stores a deque of circles is also to be implemented and tested. As for the previous assignment, you will use doxygen, make, LaTeX and Python. In addition, this assignment will use PyUnit for unit testing.

Step 1

Write a module that creates a point ADT. It should consist of a Python code file named `pointADT.py`. The specification for this module (Point Module) is given at the end of the assignment.

Step 2

Write a module that creates a line ADT. It should consist of a Python file named `lineADT.py`. The new module should follow the specification (Line Module) given at the end of the assignment.

Step 3

Write a module that creates a circle ADT. It should consist of a Python file named `circleADT.py`. The new module should follow the specification (Circle Module) given at the end of the assignment.

Step 4

Write a module that implements a deque (double ended queue) of circles. It should consist of a Python file named `deque.py`. The new module should follow the specification (Deque of Circles) given at the end of the assignment. Although efficient use of computing resources is always a good goal, your implementation will be judged on correctness and not on performance.

Step 5

Write a module, using PyUnit, that tests all of the other modules together. It should be an Python file named `testCircleDeque.py` that uses all of the other modules. Write a makefile `Makefile` to run `testCircleDeque` via `make test`. Each procedure should have at least one test case. Record your rationale for test case selection and the results of using this module to test the procedures in your modules. (You will submit your rationale with your lab report.) Please make an effort to test normal cases, boundary cases, and exception cases. Your test program should have the test cases “hard coded” into the program, rather than expecting user input. If possible, your test program should also automatically compare the calculated output to the expected output and automatically state whether the test case has passed or not.

Step 6

Submit the files `pointADT.py`, `lineADT.py`, `circleADT.py`, `deque.py`, `testCircleDeque.py` and `Makefile` using git. This must be completed no later than 11:59 pm of the deadline for file submission. You should tag your repo with the name `A2Sub`.

E-mail the `circleADT.py` file to your assigned partner. (You should only send this one file.) Partner assignments will be posted on Avenue. Please do not e-mail your file until after the deadline for the initial submission has passed. Another student will likewise e-mail you his or her files. These e-mails should be traded by midnight of the day following the file submission.

Step 7

After you have received your partner’s files, replace your corresponding files with your partner’s. Do not make any modifications to any of the code. Run your test module and record the results. Your evaluation for this step does not depend on the quality of your partner’s code, but only on your discussion of the testing results.

Step 8

Write a report that includes the following:

1. Your name and macid.
2. Your `pointADT.py`, `lineADT.py`, `circleADT.py`, `deque.py`, `testCircleDeque.py` and `Makefile` files.
3. Your partner's `circleADT.py` file.
4. The results of testing your files (along with the rational for test case selection).
5. The results of testing your files combined with your partner's files.
6. A discussion of the test results and what you learned doing the exercise. List any problems you found with (a) your program, (b) your partner's module, and (c) the specification of the modules.
7. A discussion of ?
8. The specification for the last two access programs (`totalArea()` and `averageRadius()`) is missing the definition for the output. Please complete the specification as part of the assignment submission. You are not required to implement these two access programs?
9. Discussion of principles.

A physical copy of the lab report is due at the beginning of the lecture on the assigned due date. -tag gitLab submissions

Notes

1. Place all submitted files in your svn repository in the folder **Assig2**.
2. Please put your name and student number at the top of each of your source files. (You should remove the student number before e-mailing any files to your partner.)
3. Your program must work in the ITB labs on moore when compiled by `ocamlpt` and `ocamlc`.
4. If your partner fails to provide you with a copy of his or her files by the deadline, please tell the instructor via e-mail as soon as possible.

5. If you do not send your files to your partner by the deadline, you will be assessed a **20% penalty** to your assignment grade.
6. The exceptions in the specification should simply be generated; you do not need to trap them.
7. For the Python implementation of the modules, you will need to “map” the MIS syntax to Python syntax. In particular, when the input to an access program consists of several parameters, you should provide each parameter separately, as opposed to combining them in a tuple. That is, if function f has two arguments, the type of f is $A \rightarrow (B \rightarrow C)$, not $A \times B \rightarrow C$. A concrete example, in Python syntax, is the constructor for pointT. Please use
`class pointT xc yc = ...` as opposed to
`class pointT (xc ,yc) =`
8. **Your grade will be based to a significant extent on the ability of your code to compile and its correctness. If your code does not compile, then your grade will be significantly reduced.**
9. Any changes to the assignment specification will be announced in class. It is your responsibility to be aware of these changes.

Point ADT Module

Template Module

pointADT

Uses

N/A

Syntax

Exported Types

pointT = ?

Exported Access Programs

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

Semantics

State Variables

xc: real

yc: real

State Invariant

None

Assumptions

None

Access Routine Semantics

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

dist(p):

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

rotate(ϕ):

- ϕ is in radians
- transition:

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

- exception: none

Line Module

Template Module

lineADT

Uses

pointADT

Syntax

Exported Types

lineT = ?

Exported Access Programs

Routine name	In	Out	Exceptions
new lineT	pointT, pointT	lineT	
startpt		pointT	
endpt		pointT	
length		real	
midpoint		pointT	
rotate	real		

Semantics

State Variables

s: pointT

e: pointT

State Invariant

None

Assumptions

None

Access Routine Semantics

new lineT (p_1, p_2):

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

startpt:

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

endpt:

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

length:

- output: $out := s.dist(e)$
- exception: none

midpoint:

- output:

$out := \text{new pointT}(\text{avg}(s.xcoord, e.xcoord), \text{avg}(s.ycoord, e.ycoord))$

- exception: none

rotate (ϕ):

- ϕ is in radians
- transition: $s.rotate(\phi), e.rotate(\phi)$
- exception: none

Local Functions

avg: $\text{real} \times \text{real} \rightarrow \text{real}$

$\text{avg}(x_1, x_2) \equiv \frac{x_1 + x_2}{2}$

Circle Module

Template Module

circleADT

Uses

pointADT, lineADT

Syntax

Exported Types

circleT = ?

Exported Access Programs

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

Semantics

State Variables

c : pointT

r : real

State Invariant

None

Assumptions

None

Access Routine Semantics

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

intersect(*ci*):

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

connection(*ci*):

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

force(*ci*, *f*):

- output: $out := f(r) \cdot self.\text{area}ci.\text{area}$
- exception: none

Local Functions

insideCircle: $\text{pointT} \times \text{circleT} \rightarrow \text{boolean}$

$\text{insideCircle}(p, c) \equiv p.\text{dist}(c.\text{centre}) \leq c.\text{radius}$

Deque Of Circles Module

Module

DequeCircleModule

Uses

circleADT

Syntax

Exported Constants

max_size = 20

Exported Access Programs

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

State Variables

s : sequence of circleT

State Invariant

$|s| \leq \text{max_size}$

Assumptions

`init()` is called before any other access program.

Access Routine Semantics

`init()`:

- transition: $s := \langle \rangle$
- exception: none

`pushBack(c)`:

- transition: $s := s \parallel \langle c \rangle$
- exception: $exc := (|s| = \text{max_size} \Rightarrow \text{FULL})$

`pushFront(c)`:

- transition: $s := \langle c \rangle \parallel s$
- exception: $exc := (|s| = \text{max_size} \Rightarrow \text{FULL})$

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

`front()`:

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

size():

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

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

totalArea():

- output

$$out := ?$$

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

averageRadius():

- output

$$out := ?$$

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