DRAFT Assignment 2

COMP SCI 2ME3 and SFWR ENG 2AA4

January 22, 2018

1 Dates and Deadlines

Assigned: February 1, 2018

Part 1: February 12, 2018

Receive Partner Files: February 18, 2018

Part 2: March 2, 2018

Last Revised: January 22, 2018

All submissions are made through git, using your own repo located at:

`https://gitlab.cas.mcmaster.ca/se2aa4_cs2me3_assignments_2018/[macid].git`

where `[macid]` should be replaced with your actual macid. The time for all deadlines is 11:59 pm. If you notice problems in your Part 1 `*.py` files after the deadline, you should fix the problems and discuss them in your Part 2 report. However, the code files submitted for the Part 1 deadline will be the ones graded.

2 Introduction

The purpose of this software design exercise is to write a Python program that creates, uses, and tests an ADT for points, curves and data. Plus something. As for the previous assignment, you will use doxygen, make, LaTeX and Python. In addition, this assignment will use PyTest for unit testing.

This assignment takes advantage of functional programming in Python. In a few cases functions are passed as arguments and returned as output. (Check passed as inputs).

Examples of glass capacity. Example of thermocouples for temperature.

All of your code (all files) should be documented using doxygen. Your report should be written using LaTeX. Your code should follow the given specification exactly. In particular, you should not add public methods or procedures that are not specified and you should not change the number or order of parameters for methods or procedures. If you need private methods or procedures, please use the Python convention of naming the files with the double underscore (`__methodName__`) (dunders). Python coding conventions.

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 curve ADT. It should consist of a Python file named `curveSeqADT.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 second curve ADT. Same interface, but different state variables. It should consist of a Python file named `circleFuncADT.py`. The new module should follow the specification given at the end of the assignment.

Step 4

Write a module that implements data - a sequence of curves. It should consist of a Python file named `data.py`. The new module should follow the specification 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.

- interpolation, regression, plotting modules?

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 the rule `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 report.) Please make an effort to test normal cases, boundary cases, and exception cases. Your test program should compare the calculated output to the expected output and provide a summary of the number of test case that have passed or failed.

Step 6

Add to your makefile a rule for `doc`. This rule should compile your source code documentation into an html and LaTeX version. Your documentation should be generated to the `A2` folder.

Step 7

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. Please use the names and locations for these files already given in your git project repo. You should also push your doxygen configuration file to the repo. You will have to add this file to the repo. Ideally, you should place it in the `A2` folder. You should NOT sumbit your generated documentation (html and latex folders). In general, files that can be regenerated are not put under version control. You should tag your final submission of part 1 of the assignment with the name `A2Part1`.

Step 8

Your `circleADT.py` file will automatically be pushed to your partner's repo and vice versa. You actually do not have to take any overt action for this part. I will happen automatically about a day after the deadline for part 1 of the assignment. The location in your repo of your partner's file is given in the Notes section below.

Step 9

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 10

Write a report and push it to your project repo. The final submission should have the tag `A2Part2`. The report should include 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. (You can push this to the repo in the folder srcPartner.)
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. The summary of the results should consist of the following: the number of passed and failed test cases, and brief details on any failed test cases.
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. How did using a formal module interface specification for this assignment compare to the informal specification provided for Assignment 1? What are the advantages of using a testing framework, such as PyUnit for testing your code?
7. 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 your report. You should write the specification as LaTeX equations in your report. You are not required to implement these two access programs.
8. Provide a critique of the Circle Module's interface. In particular, comment on whether the exported access programs provide an interface that is consistent, essential, general, minimal and opaque.

9. What is the output of the mathematical specification of Deq_disjoint() when there is one circle in the deque? Explain your answer. Does this answer make sense? Is it the same result as calculated by your code?

Your commit (push) to the repository should include the file `report.tex` as given in your initial folder structure. You should also push the file `report.pdf` in the same folder. Although the pdf file is a generated file, we'll make an exception to the general rule of avoiding version control for generated files. The purpose of the exception is for the convenience of the TAs doing the grading.

The final submission of your report, including your tex file, should be done using git by 11:59 pm on the assigned due date. If you notice problems in your original `*.py` files, you should discuss these problems, and what changes you would make to fix them, in your report. However, the code files submitted on the first deadline will be the ones that are graded.

Notes

1. Your git repo will be organizes with the following directories at the top level: `A1`, `A2`, `A3`, and `A4`.

2. Inside the `A2` folder you will start with initial stubs of the files and folders that you need to use. Please do not change the names or locations of any of these files or folders. The structure of your project files and folders should look like:

   - A2
     - ∗ doxConfig
     - ∗ Makefile
     - – report
       - ∗ report.tex
       - ∗ report.pdf
     - – src
       - – srcPartner
         - ∗ circleADT.py
       - ∗ pointADT.py
       - ∗ lineADT.py
       - ∗ circleADT.py
       - ∗ deque.py
       - ∗ testCircleDeque.py

3. Please put your name and macid at the top of each of your source files.

4. Your program must work in the ITB labs on mills when compiled with its versions of Python (version 2), LaTeX, doxygen and make.

5. Python specifics:

   - The exceptions in the specification should be implemented via Python exceptions. Your exceptions should have exactly the same name as given in the specification (FULL, EMPTY). Your exceptions should inherit from the Exception class and they should only be used with one argument, a string explaining what problem has occurred.
   - For the Python implementation of the abstract module, your access programs should be called via, Deq.accessProg, not Deq_accessProg, as shown in the specification. Some sample calls include the following: Deq.init(), Deq.pushBack(c), Deq.pushFront(c), etc.
   - Since the specification is silent on this point, for methods that return an object, you can decide to either return a reference to the appropriate existing object, or construct a new object.

6. **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.**

7. **Any changes to the assignment specification will be announced in class. It is your responsibility to be aware of these changes. Please monitor all pushes to the course git repo.**

8. Python coding convention

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	
xcrd		real	
ycrd		real	
dist	PointT	real	
rot	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

xcrd:

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

ycrd:

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

dist(p):

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

rot(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	
beg		PointT	
end		PointT	
len		real	
mdpt		PointT	
rot	real		

Semantics

State Variables

b: PointT
e: PointT

State Invariant

None

Assumptions

None

Access Routine Semantics

new LineT (p_1, p_2):

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

beg:

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

end:

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

len:

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

mdpt:

- output:

$$out := \text{new PointT}(\text{avg}(b.\text{xcrd}(), e.\text{xcrd}()), \text{avg}(b.\text{ycrd}(), e.\text{ycrd}()))$$

- exception: none

rot (ϕ):

- ϕ is in radians
- transition: $b.\text{rot}(\phi), e.\text{rot}(\phi)$
- exception: none

Local Functions

avg: real $\times$ real $\rightarrow$ real
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	
cen		PointT	
rad		real	
area		real	
intersect	CircleT	boolean	
connection	CircleT	LineT	
force	real $\rightarrow$ real	CircleT $\rightarrow$ real	

Semantics

State Variables

c: PointT
r: real

State Invariant

None

Assumptions

None

Access Routine Semantics

new CircleT (cin, rin):

- transition: $c, r := cin, rin$
- output: $out := self$
- exception: none

cen:

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

rad:

- 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: $out :=$ new LineT$(c, ci.cen())$
- exception: none

force(f):

- output: $out := \lambda x \rightarrow self.\text{area}() \cdot x.\text{area}() \cdot f(self.\text{connection}(x).\text{len}())$
- exception: none

Local Functions

insideCircle: PointT $\times$ CircleT $\rightarrow$ boolean
insideCircle$(p, c) \equiv p.\text{dist}(c.\text{cen}()) \leq c.\text{rad}()$

Deque Of Circles Module

Module

DequeCircleModule

Uses

circleADT

Syntax

Exported Constants

MAX_SIZE = 20

Exported Access Programs

Routine name	In	Out	Exceptions
Deq_init			
Deq_pushBack	CircleT		FULL
Deq_pushFront	CircleT		FULL
Deq_popBack			EMPTY
Deq_popFront			EMPTY
Deq_back		CircleT	EMPTY
Deq_front		CircleT	EMPTY
Deq_size		integer	
Deq_disjoint		boolean	EMPTY
Deq_sumFx	real $\rightarrow$ real	real	EMPTY
Deq_totalArea		real	EMPTY
Deq_averageRadius		real	EMPTY

Semantics

State Variables

s: sequence of CircleT

State Invariant

$|s| \leq$ MAX_SIZE

Assumptions

Deq_init() is called before any other access program.

Access Routine Semantics

Deq_init():

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

Deq_pushBack(c):

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

Deq_pushFront(c):

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

Deq_popBack():

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

Deq_popFront():

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

Deq_back():

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

Deq_front():

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

Deq_size():

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

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

Deq_sumFx(f):

- output

$$out := +(i : \mathbb{N} | i \in ([1..|s|-1]) : \text{Fx}(f, s[i], s[0]))$$

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

Deq_totalArea():

- output

$$out :=?$$

  [The total area is the sum of the area of all of the circles in the deque. You do not need to worry about overlap between circles. The assignment asks you to provide the missing equation, but you do not have to implement this access program.]

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

Deq_averageRadius():

- output

$$out :=?$$

  [The assignment asks you to provide the missing equation, but you do not have to implement this access program.]

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

Local Functions

Fx: (real $\rightarrow$ real) $\times$ CircleT $\times$ CircleT $\rightarrow$ real
$\text{Fx}(f, ci, cj) \equiv \text{xcomp}(ci.\text{force}(f)(cj), ci, cj)$

xcomp: real $\times$ CircleT $\times$ CircleT $\rightarrow$ real

$$\text{xcomp}(F, ci, cj) \equiv F\left[\frac{ci.\text{cen}().\text{xcrd}() - cj.\text{cen}().\text{xcrd}()}{ci.\text{connection}(cj).\text{len}()}\right]$$