Assignment 3, Part 1, Specification

SFWR ENG 2AA4

March 12, 2018

The purpose of this software design exercise is to design and implement a portion of the specification for a Geographic Information System (GIS). This document shows the complete specification, which will be the basis for your implementation and testing. In this specification natural numbers (ℕ) include zero (0).

[The parts that you need to fill in are marked by comments, like this one. In several of the modules local functions are specified. You can use these local functions to complete the missing specifications. —SS]

Map Types Module

Module

MapTypes

Uses

N/A

Syntax

Exported Constants

None

Exported Types

CompassT = {N, S, E, W}
LanduseT = {Recreational, Transport, Agricultural, Residential, Commercial}
RotateT = {CW, CCW}

Exported Access Programs

None

Semantics

State Variables

None

State Invariant

None

Point ADT Module

Template Module

PointT

Uses

N/A

Syntax

Exported Types

PointT = ?

Exported Access Programs

Routine name	In	Out	Exceptions
PointT	$\mathbb{Z}$, $\mathbb{Z}$	PointT	
x		$\mathbb{Z}$	
y		$\mathbb{Z}$	
translate	$\mathbb{Z}$, $\mathbb{Z}$	PointT	

Semantics

State Variables

xc: $\mathbb{Z}$
yc: $\mathbb{Z}$

State Invariant

None

Assumptions

The constructor PointT is called for each object instance before any other access routine is called for that object. The constructor cannot be called on an existing object.

Access Routine Semantics

PointT(x, y):

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

x():

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

y():

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

translate(Δx, Δy):

- output: x+ Δx, y + Δy
- exception: None

Line ADT Module

Template Module

LineT

Uses

PointT, CompassT, RotateT,

Syntax

Exported Types

LineT = ?

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
LineT	PointT, CompassT, $\mathbb{N}$	LineT	invalid_argument
strt		PointT	
end		PointT	
orient		CompassT	
len		$\mathbb{N}$	
flip		LineT	
rotate	RotateT	LineT	
translate	$\mathbb{Z}$, $\mathbb{Z}$	LineT	

Semantics

State Variables

s: PointT
o: CompassT
L: $\mathbb{N}$

State Invariant

None

Assumptions

The constructor LineT is called for each object instance before any other access routine is called for that object. The constructor cannot be called on an existing object.

Access Routine Semantics

LineT($st, ornt, l$):

- transition: $s, o, L := st, ornt, l$
- output: $out := self$
- exception: exc := (L = 0 $\implies$ invalid_argument)

strt():

- output: $out :=$ PointT($s.x(), s.y()$)
- exception: None

end():

- output: $out := \{o = N \implies \text{PointT}(s.x, s.y + L) | o = S \implies \text{PointT}(s.x, s.y - L) | o = E \implies \text{PointT}(s.x + L, s.y) o = W \implies \text{PointT}(s.x - L, s.y)\}$
- exception: None

orient():

- output: $out := o$
- exception: None

len():

- output: $out := L$
- exception: None

flip():

- output: $out := \{o = N \implies \text{LineT}(st, S, l) | o = S \implies \text{LineT}(st, N, l) | o = W \implies \text{LineT} st, E, l | o = E \implies \text{LineT}(st, W, l)\}$
- exception: None

rotate(r):

- output:

		$out :=$
$r =$ CW	$o =$ N	LineT(st, E, l)
	$o =$ S	LineT(st, W, l)
	$o =$ W	LineT(st, N, l)
	$o =$ E	LineT(st, S, l)
$r =$ CCW	$o =$ N	LineT(st, W, l)
	$o =$ S	LineT(st, E, l)
	$o =$ W	LineT(st, S, l)
	$o =$ E	LineT(st, N, l)

- exception: None

translate(Δx, Δy):

- output:

$$\forall (i : \mathbb{N} | i \in [0..|s| - 1] : s'[i] = s[i].\text{translate}(\Delta x, \Delta y))$$

- exception: None

Path ADT Module

Template Module

PathT

Uses

PointT, LineT, MapTypes

Syntax

Exported Types

PathT = ?

Exported Access Programs

Routine name	In	Out	Exceptions
PathT	PointT, CompassT, $\mathbb{N}$	PathT	
append	CompassT, $\mathbb{N}$		invalid_argument
strt		PointT	
end		PointT	
line	$\mathbb{N}$	LineT	outside_bounds
size		$\mathbb{N}$	
len		$\mathbb{N}$	
translate	$\mathbb{Z}$, $\mathbb{Z}$	PathT	

Semantics

State Variables

s: sequence of LineT

State Invariant

None

Assumptions

- The constructor PathT is called for each object instance before any other access routine is called for that object. The constructor cannot be called on an existing object.

Access Routine Semantics

PathT($st, ornt, l$):

- transition: s[0] = [st, ornt, l]
- output: $out := self$
- exception: None

append($ornt, l$):

- transition: $o = N \implies LineT(adjPt(ornt), o, l) | o = S \implies LineT(adjPt(ornt), o, l) | o = W \implies LineT(adjPt(ornt), o, l) | o = E \implies LineT(adjPt(ornt), o, l)$
- exception: $exc := (l = 0) \implies invalid\_argument$

strt():

- output: out := s[0].strt()
- exception: None

end():

- output: $out := s[|s| - 1]$

line(i):

- output: out := s[i]
- exception: $exc := (\neg(0 \leq i < |s|) \implies InvalidIndex)$

size:

- output: out := s[i]
- exception: None

len:

- output: $out := +(\forall(i:\mathbb{N}|i \in [0..|s|-1] : pointsInLine(s[i])))$
- exception: None

translate(Δx, Δy):

- output: Create a new PathT object with state variable s' such that:

$$\forall(i:\mathbb{N}|i \in [0..|s|-1] : s'[i] = s[i].\text{translate}(\Delta x, \Delta y))$$

- exception: None

Local Functions

pointsInLine: LineT → (set of PointT)

pointsInLine (l)

$$\equiv \{i:\mathbb{N}|i \in [0..(l.\text{len}-1)] : l.\text{strt.translate.}$$
$$(o = S \implies l.strt.translate(0, -i)|o = N \implies strt.translate(0, i)$$
$$|l.o = E \implies strt.translate(i, 0)|l.o = W \implies strt.translate(-i, 0))$$

adjPt: CompassT → PointT
adjPt($ornt$) ≡

$ornt = \text{N}$	$s[|s|-1]$.end.translate $0, 1$
$ornt = \text{S}$	$s[|s|-1]$.end.translate $0, -1$
$ornt = \text{W}$	$s[|s|-1]$.end.translate $-1, 0$
$ornt = \text{E}$	$s[|s|-1]$.end.translate $1, 0$

Generic Seq2D Module

Generic Template Module

Seq2D(T)

Uses

MapTypes, PointT, LineT, PathT

Syntax

Exported Types

Seq2D(T) = ?

Exported Constants

None

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
Seq2D	seq of (seq of T), $\mathbb{R}$	Seq2D	invalid_argument
set	PointT, T		outside_bounds
get	PointT	T	outside_bounds
getNumRow		$\mathbb{N}$	
getNumCol		$\mathbb{N}$	
getScale		$\mathbb{R}$	
count	T	$\mathbb{N}$	
count	LineT, T	$\mathbb{N}$	invalid_argument
count	PathT, T	$\mathbb{N}$	invalid_argument
length	PathT	$\mathbb{R}$	invalid_argument
connected	PointT, PointT	$\mathbb{B}$	invalid_argument

Semantics

State Variables

s: seq of (seq of T)
scale: $\mathbb{R}$

nRow: $\mathbb{N}$
nCol: $\mathbb{N}$

State Invariant

None

Assumptions

- The Seq2D(T) constructor is called for each object instance before any other access routine is called for that object. The constructor can only be called once.
- Assume that the input to the constructor is a sequence of rows, where each row is a sequence of elements of type T. The number of columns (number of elements) in each row is assumed to be equal. That is each row of the grid has the same number of entries. $s[i][j]$ means the ith row and the jth column. The 0th row is at the bottom of the map and the 0th column is at the leftmost side of the map.

Access Routine Semantics

Seq2D(S, scl):

- transition:s, scale := S, scl
- output: $out := self$
- exception: $exc := scl = 0 \implies invalid\_argument || |S| = 0 \implies invalid\_argument$

set(p, v):

- transition: s[p.y][p.x] := v
- exception: $exc := (ValidPoint(p) \not\equiv true) \implies outside\_bounds$

get(p):

- output: s[p.y][p.x]
- exception: $exc := (ValidPoint(p) \not\equiv true) \implies outside\_bounds$

getNumRow():

- output: out := nRow
- exception: None

getNumCol():

- output: out := nCol
- exception: None

getScale():

- output: out := scale
- exception: None

count(t: T):

- output: $out := +(i, j : \mathbb{N} | 0 \leq i < nRow \wedge 0 \leq j < nCol \wedge s[i][j] \equiv t : 1)$
- exception: None

count(l: LineT, t: T):

- output: $out := +(p : PointT | (p \in pointsInLine(l) \wedge get(p)) \equiv t : 1)$
- exception: $exc := (ValidLine(p) \not\equiv true) \implies outside\_bounds$

count(pth: PathT, t: T):

- output:$out := +(p : PointT | (p \in pointsInPath(pth) \wedge get(p)) \equiv t : 1)$
- exception: $exc := (ValidPath(p) \not\equiv true) \implies outside\_bounds$

length(pth: PathT):

- output: out := pth.size() * s.getScale()
- exception: $exc := (ValidPoint(p) \not\equiv true) \implies outside\_bounds$

connected(p_1: PointT, p_2: PointT):

- output: $out := (\exists p = pointsInPath(PathT) | p_1 < p < p_2) \implies true$
- exception: $exc := (ValidPoint(p_1) \not\equiv true \wedge ValidPoint(p_2) \not\equiv true) \implies invalid\_argument$

Local Functions

validRow: $\mathbb{N} \rightarrow \mathbb{B}$
$\forall i | i \in \mathbb{N} : i \implies \mathit{true}$

validCol: $\mathbb{N} \rightarrow \mathbb{B}$
$\forall i | i \in \mathbb{N} : i \implies \mathit{true}$

validPoint: PointT $\rightarrow \mathbb{B}$
$(0 \leq p.x < nCol \wedge 0 \leq p.y < nRow) \implies \mathit{true}$

validLine: LineT $\rightarrow \mathbb{B}$
$((P = pointsInLine(LineT).(\forall i | i \in [0..|P| - 1])) : validPoint(P[i]) \implies \mathit{true}$
validPath: PathT $\rightarrow \mathbb{B}$
$((P = pointsInPath(PathT).(\forall i | i \in [0..|P| - 1])) : validPoint(P[i]) \implies \mathit{true}$
pointsInLine: LineT $\rightarrow$ (set of PointT)
pointsInLine (l) [The same local function as given in the Path module. —SS]
pointsInPath: PathT $\rightarrow$ (set of PointT)
[Return the set of points that make up the input path. —SS] pointsInPath(p)

LanduseMap Module

Template Module

LanduseMapT is Seq2D(LanduseT)

DEM Module

Template Module

DEMT is Seq2D($\mathbb{Z}$)

Critique of Design

Write a critique of the interface for the modules in this project. Is there anything missing? Is there anything you would consider changing? Why?

The module implementations of different directions(ornt) in both LineADT and PathADT are not very specific, different ornts would cause differernt lines to be generated.
The exceptions are not include all situations for seq2D module, the invalid_argument exceptions could be more specific.