Module Interface Specification for GlassBR

Spencer Smith

August 13, 2018

Contents

1 Introduction

The following document details the Module Interface Specification (MIS) for the implemented modules in the program GlassBR. It is intended to ease navigation through the program for design and maintenance purposes. Complementary documents include the Software Requirements Specification (SRS) and Module Guide (MG). The full documentation and implementation can be found at https://github.com/smiths/caseStudies/tree/master/CaseStudies/glass.

2 Notation

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | ... | c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by GlassBR.

Data Type	Notation	Description
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real number	$\mathbb{R}$	any number in (-∞, ∞)
string	$\mathbb{S}$	all finite sequences of symbols from the ASCII character set

The specification of GlassBR uses some derived data types: sequences, strings, and tuples. Sequences are lists that represent a countable number of ordered values of the same data type, where the same value may occur more than once. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, GlassBR uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

3 Module Hierarchy

To view the Module Hierarchy, see section 3 of the MG.

References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.

4 MIS of Control Module

4.1 Module

Control

4.2 Uses

Input (Section 5), LoadASTM (Section 6), Calc (Section 7), Output (Section 13)

4.3 Syntax

4.3.1 Exported Constants

sTSD = *# String for path and filename for TSD file*
sSDF = *# String for path and filename for SDF file*
sIn = *# String for path and filename for Input file*
sOut = *# String for path and filename for Output file*

4.3.2 Exported Access Programs

Name	In	Out	Exceptions
main	-	-	-

4.4 Semantics

4.4.1 State Variables

cTSD: ContoursT
cSDF: ContoursT
$q : \mathbb{R}$
$\hat{q} : \mathbb{R}$
$J_{\text{tol}} : \mathbb{R}$
$J : \mathbb{R}$
$\hat{q}_{\text{tol}} : \mathbb{R}$
$B : \mathbb{R}$
$P_b : \mathbb{R}$
NFL : $\mathbb{R}$
LR : $\mathbb{R}$
is_safePb : $\mathbb{B}$
is_safeLR : $\mathbb{B}$

4.4.2 Access Routine Semantics

main():

- transition: Using the steps below, modify the state variables of this module, the state of the Input module, and the environment variable for the Output module.
  - cTSD = LoadASTM.LoadTSD(sTSD)
  - cSDF = LoadASTM.LoadSDF(sSDF)
  - Input.load_params(sIn)
  - q = cTSD.eval(Input.SD, Input.w * Input.TNT) *# From SRS IM3*
  - $\hat{q}$ = Calc.calc_q_hat(q)
  - J_{tol} = Calc.calc_J_tol()
  - J = cSDF.evaly(Input.AR, $\hat{q}$) *# From SRS DD4*
  - $\hat{q}_{\text{tol}}$ = cSDF.eval(Input.AR, J_{tol}) *# From SRS DD8*
  - B = Calc.calc_B(J)
  - P_b = Calc.calc_Pb(B)
  - NFL = Calc.calc_NFL($\hat{q}_{\text{tol}}$)
  - LR = Calc.calc_LR(NFL)
  - is_safePb = Calc.calc_is_safePb(P_b)
  - is_safeLR = Calc.calc_is_safeLR(LR, q)
  - Output.output(sOut, is_safePb, is_safeLR, P_b, B, LR, q, J, NFL, $\hat{q}$, $\hat{q}_{\text{tol}}$, J_{tol})
  - print("Main has been executed and the results have been written to " + sOut)

5 MIS of Input Module

The secrets of this module are the data structure for input parameters, how the values are input and how the values are verified. The load and verify secrets are isolated to their own access programs. This module follows the singleton pattern; that is, there is only one instance of this module.

5.1 Module

Input

5.2 Uses

GlassTypeADT (Section 8), ThicknessADT (Section 9), Constants (Section 14), Hardware (Section 15)

5.3 Syntax

Name	In	Out	Exceptions
load_params	string	-	FileError
verify_params	-	-	ValueError
a	-	$\mathbb{R}$	
b	-	$\mathbb{R}$	
g	-	GlassTypeT	
$P_{b_{\text{tol}}}$	-	$\mathbb{R}$	
SD_x	-	$\mathbb{R}$	
SD_y	-	$\mathbb{R}$	
SD_z	-	$\mathbb{R}$	
t	-	ThicknessT	
TNT	-	$\mathbb{R}$	
w	-	$\mathbb{R}$	
m	-	$\mathbb{R}$	
k	-	$\mathbb{R}$	
E	-	$\mathbb{R}$	
t_d	-	$\mathbb{R}$	
LDF	-	$\mathbb{R}$	
LSF	-	$\mathbb{R}$	
h	-	$\mathbb{R}$	
GTF	-	$\mathbb{R}$	
SD	-	$\mathbb{R}$	
AR	-	$\mathbb{R}$	

5.4 Semantics

5.4.1 Environment Variables

inputFile: sequence of string *#f[i] is the ith string in the text file f*

5.4.2 State Variables

From SRS R1
a: $\mathbb{R}$
b: $\mathbb{R}$
g: GlassTypeT
$P_{b_{\text{tol}}}$: $\mathbb{R}$
SD_x : $\mathbb{R}$
SD_y : $\mathbb{R}$
SD_z : $\mathbb{R}$
t: ThicknessT
TNT: $\mathbb{R}$
w: $\mathbb{R}$

From SRS R2
m: $\mathbb{R}$
k: $\mathbb{R}$
E: $\mathbb{R}$
t_d: $\mathbb{R}$
LDF: $\mathbb{R}$
LSF: $\mathbb{R}$
h: $\mathbb{R}$
GTF: $\mathbb{R}$
SD: $\mathbb{R}$
AR: $\mathbb{R}$

5.4.3 Assumptions

- load_params will be called before the values of any state variables will be accessed.
- The file contains the string equivalents of the numeric values for each input parameter in order, each on a new line. The order is the same as in the table in R1 of the SRS. Any comments in the input file should be denoted with a '#' symbol.

5.4.4 Access Routine Semantics

a:

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

b:

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

...

GTF:

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

load_params(s):

- transition: The filename s is first associated with the file f. inputFile is used to modify the state variables using the following procedural specification:
  1. Read data sequentially from inputFile to populate the state variables from SRS R1.
  2. Calculate the derived quantities (all other state variables, from SRS R2) as follows:
     - $m, k, E, t_d,$ LSF as defined in Constants (Section 14)
     - LDF $= (\frac{t_d}{60})^{m/16}$ *# From SRS DD3*
     - $h = t$.toMinThick() (Section 9)
     - GTF $= g$.GTF() (Section 8)
     - SD $= \sqrt{\text{SD}_x{}^2 + \text{SD}_y{}^2 + \text{SD}_z{}^2}$ *# From SRS DD10*
     - AR $= a/b$ *# From SRS DD11*
  3. verify_params()
- exception: exc := a file name s cannot be found OR the format of inputFile is incorrect $\Rightarrow$ FileError

verify_params():

- out: $out :=$ none
- exception: exc := *# From the SRS Table 2, R3*

$\neg(a > 0)$	$\Rightarrow$ ValueError("a must be positive")
$\neg(\text{AR} \geq 1.0)$	$\Rightarrow$ ValueError("a must be greater than or equal to b")
$\neg(d_{\min} \leq a \leq d_{\max})$	$\Rightarrow$ ValueError("a too large or small")
$\neg(\text{AR} \leq \text{AR}_{\max})$	$\Rightarrow$ ValueError("allowable aspect ratio exceeded")
$\neg(b > 0)$	$\Rightarrow$ ValueError("b must be positive")
$\neg(d_{\min} \leq b \leq d_{\max})$	$\Rightarrow$ ValueError("b too large or small")
$\neg(0 < P_{b_{\text{tol}}} < 1)$	$\Rightarrow$ ValueError("$P_{b_{\text{tol}}}$ must be between 0 and 1")
$\neg(w > 0)$	$\Rightarrow$ ValueError("charge weight (w) must be greater than zero")
$\neg(w_{\min} \leq w \leq w_{\max})$	$\Rightarrow$ ValueError("charge weight (w) is too small or too large")
$\neg(\text{TNT} > 0)$	$\Rightarrow$ ValueError("TNT must be positive")
$\neg(\text{SD} > 0)$	$\Rightarrow$ ValueError("stand off distance (SD) must be positive")
$\neg(\text{SD}_{\min} \leq \text{SD} \leq \text{SD}_{\max})$	$\Rightarrow$ ValueError("stand off distance (SD) is too small or too large")

5.5 Considerations

The value of each state variable can be accessed through its name (getter). An access program is available for each state variable. There are no setters for the state variables, since the values will be set and checked by load params and will not be changed for the life of the program.

6 MIS of LoadASTM Module

6.1 Module

LoadASTM

6.2 Uses

FunctADT (Section 10), ContoursADT (Section 11)

6.3 Syntax

6.3.1 Exported Constants

None

6.3.2 Exported Access Programs

Routine name	In	Out	Exceptions
LoadTSD	s : string	ContoursT	FileError
LoadSDF	s : string	ContoursT	FileError

6.4 Semantics

6.4.1 Environment Variables

infile: two dimensional sequence of text characters

6.4.2 State Variables

None

6.4.3 State Invariant

None

6.4.4 Assumptions

The input file will match the given specification.

6.4.5 Access Routine Semantics

LoadTSD(s)

- output: Create *out* following the following steps. Read data from the file infile associated with the string s. Use this data to create a ContoursT object. For each value of w, create an object of FunctT and add it to the ContoursT object. Each of the FunctT objects will consist of q versus SD data. The first row of the TSD file contains the values of w. The subsequent columns are grouped in pairs. Each pair corresponds to a column of SD data and a column of q data. There is a pair of columns in this pattern for each value of w.

- exception: exc := a file name s cannot be found OR the format of inputFile is incorrect $\Rightarrow$ FileError

LoadSDF(s)

- output: Create *out* following the following steps. Read data from the file infile associated with the string s. Use this data to create a ContoursT object. For each value of J, create an object of Funct T and add it to the ContoursT object. Each of the FunctT objects will consist of q^* versus AR data. The first row of the SDF file contains the values of J. The subsequent columns are grouped in pairs. Each pair corresponds to a column of AR data and a column of q^* data. There is a pair of columns in this pattern for each value of J.

- exception: exc := a file name s cannot be found OR the format of inputFile is incorrect $\Rightarrow$ FileError

7 MIS of Calc Module

7.1 Module

Calc

7.2 Uses

Input (Section 5), ContoursADT (Section 11), Constants (Section 14)

7.3 Syntax

7.3.1 Exported Constants

None

7.3.2 Exported Access Programs

Routine name	In	Out	Exceptions
calc_q_hat	$q : \mathbb{R}$	$\mathbb{R}$	-
calc_J_tol		$\mathbb{R}$	-
calc_Pb	$B : \mathbb{R}$	$\mathbb{R}$	InvalidOutput
calc_B	$J : \mathbb{R}$	$\mathbb{R}$	-
calc_NFL	$\hat{q}_{\text{tol}} : \mathbb{R}$	$\mathbb{R}$	-
calc_LR	NFL : $\mathbb{R}$	$\mathbb{R}$	-
calc_is_safePb	$P_b : \mathbb{R}$	$\mathbb{B}$	-
calc_is_safeLR	LR : $\mathbb{R}, q : \mathbb{R}$	$\mathbb{B}$	-

7.4 Semantics

7.4.1 State Variables

None

7.4.2 State Invariant

None

7.4.3 Assumptions

None

7.4.4 Access Routine Semantics

calc_q_hat(q) # *From SRS DD7 Dimensionless Load ($\hat{q}$)*

- output: $out := \frac{q(ab)^2}{Eh^4 \text{GTF}}$
- exception: None

calc_J_tol() # *From SRS DD9 Tolerable Stress Distribution Factor (J_{tol})*

- output: $out := \ln[\ln(\frac{1}{1-P_{b_{\text{tol}}}}) \frac{(a \times b)^{m-1}}{k(Eh^2)^m \text{LDF}}]$
- exception: None

calc_Pb(B) # *From SRS IM1 Probability of Glass Breakage*

- output: $out := 1 - e^{-B}$
- exception: $\neg(0 < (1 - e^{-B}) < 1) \Rightarrow$ InvalidOutput # *From SRS Table 3, Output Variables*

calc_B(J) # *From SRS DD1 Risk of Failure (B)*

- output: $out := \frac{k}{(a \times b)^{m-1}} (Eh^2)^m \times \text{LDF} \times e^J$
- exception: None

calc_NFL($\hat{q}_{\text{tol}}$) # *From SRS DD5 Non-Factored Load*

- output: $out := \frac{\hat{q}_{\text{tol}} Eh^4}{(ab)^2}$
- exception: None

calc_LR(NFL) # *From SRS IM2 Calculation of Capacity (LR)*

- output: $out :=$ NFL $\times$ GTF $\times$ LSF
- exception: None

calc_is_safePb(P_b) # *From SRS T1 Safety Req-Pb)*

- output: $out := P_b < P_{b_{\text{tol}}}$
- exception: None

calc_is_safeLR(LR, q) # *From SRS T2 Safety Req-LR)*

- output: $out :=$ LR $> q$
- exception: None

8 MIS of GlassType ADT Module

From DD6 (Glass Type Factor (GTF))

8.1 Template Module

GlassTypeADT

8.2 Uses

None

8.3 Syntax

8.3.1 Exported Constants

None

8.3.2 Exported Types

GlassTypeT = ?

8.3.3 Exported Access Programs

Routine name	In	Out	Exceptions
new GlassTypeT	$\mathbb{S}$	GlassTypeT	ValueError
GTF		$\mathbb{R}$	
toString		$\mathbb{S}$	

8.4 Semantics

8.4.1 State Variables

g: {AN, FT, HS}

8.4.2 State Invariant

None

8.4.3 Assumptions

None

8.4.4 Access Routine Semantics

new GlassTypeT(s):

- transition: $g := (s = \text{“AN”} \Rightarrow \text{AN} | s = \text{“FT”} \Rightarrow \text{FT} | s = \text{“HS”} \Rightarrow \text{HS})$
- output: $out :=$ self
- exception: $(\neg(s \in \{\text{“AN”}, \text{“FT”}, \text{“HS”}\}) \Rightarrow \text{ValueError})$

GTF():

- output: $out := (g = \text{AN} \Rightarrow 1.0 | g = \text{FT} \Rightarrow 4.0 | g = \text{HS} \Rightarrow 2.0)$
- exception: None

toString():

- output: $out := (g = \text{AN} \Rightarrow \text{“AN”} | g = \text{FT} \Rightarrow \text{“FT”} | g = \text{HS} \Rightarrow \text{“HS”})$
- exception: None

9 MIS of Thickness ADT Module

From DD2 (Minimum Thickness from Nominal Thickness)

9.1 Template Module

ThicknessADT

9.2 Uses

None

9.3 Syntax

9.3.1 Exported Constants

None

9.3.2 Exported Types

ThicknessT = ?

9.3.3 Exported Access Programs

Routine name	In	Out	Exceptions
new ThicknessT	$\mathbb{R}$	ThicknessT	ValueError
toMinThick		$\mathbb{R}$	
toFloat		$\mathbb{R}$	

9.4 Semantics

9.4.1 State Variables

$t : T$ where $T = \{2.5, 2.7, 3.0, 4.0, 5.0, 6.0, 8.0, 10.0, 12.0, 16.0, 19.0, 22.0\}$

9.4.2 State Invariant

None

9.4.3 Assumptions

None

9.4.4 Access Routine Semantics

new ThicknessT(x):

- transition: $t := x$
- output: $out :=$ self
- exception: $(\neg(x \in T) \Rightarrow \text{ValueError})$

toMinThick():

- output: $out := \frac{1}{1000} \begin{cases} t = 2.5 \Rightarrow 2.16 \\ t = 2.7 \Rightarrow 2.59 \\ t = 3.0 \Rightarrow 2.92 \\ t = 4.0 \Rightarrow 3.78 \\ t = 5.0 \Rightarrow 4.57 \\ t = 6.0 \Rightarrow 5.56 \\ t = 8.0 \Rightarrow 7.42 \\ t = 10.0 \Rightarrow 9.02 \\ t = 12.0 \Rightarrow 11.91 \\ t = 16.0 \Rightarrow 15.09 \\ t = 19.0 \Rightarrow 18.26 \\ t = 22.0 \Rightarrow 21.44 \end{cases}$
- exception: None

toFloat():

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

10 MIS of FunctADT Module

10.1 Template Module

FunctADT

10.2 Uses

SeqServices (Section 12)

10.3 Syntax

10.3.1 Exported Constants

MAX_ORDER = 2

10.3.2 Exported Types

FunctT = ?

10.3.3 Exported Access Programs

Routine name	In	Out	Exceptions
new FunctT	$X_{\text{in}} : \mathbb{R}^n$, $Y_{\text{in}} : \mathbb{R}^n$, $i : \mathbb{N}$	FunctT	IndepVarNotAscending, SeqSizeMismatch, InvalidInterpOrder, TooFewDataPts
minD		$\mathbb{R}$	
maxD		$\mathbb{R}$	
order		$\mathbb{N}$	
eval	$x : \mathbb{R}$	$\mathbb{R}$	OutOfDomain

10.4 Semantics

10.4.1 State Variables

X: $\mathbb{R}^n$
Y: $\mathbb{R}^n$
minX: $\mathbb{R}$
maxX: $\mathbb{R}$
o: $\mathbb{N}$

10.4.2 State Invariant

None

10.4.3 Assumptions

None

10.4.4 Access Routine Semantics

new FunctT($X_{\text{in}}, Y_{\text{in}}, i$):

- transition: $X, Y, \text{minX}, \text{maxX}, o := X_{\text{in}}, Y_{\text{in}}, X_{\text{in}}[0], X_{\text{in}}[|X| - 1], i$
- output: *out* := self
- exception: ($\neg$isAscending(X_{in}) $\Rightarrow$ IndepVarNotAscending$||X_{\text{in}}| \neq |Y_{\text{in}}| \Rightarrow$ SeqSizeMismatch$|i \notin$ [1..MAX_ORDER] $\Rightarrow$ InvalidInterpOrder$||X_{\text{in}}| < 3 \Rightarrow$ TooFewDataPts)

minD():

- output: *out* := minX
- exception: None

maxD():

- output: *out* := maxX
- exception: None

order():

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

eval(x):

- output: *out* :=

  $(o = 1 \Rightarrow \text{interpLin}(X_i, Y_i, X_{i+1}, Y_{i+1}, x) | o = 2 \Rightarrow \text{interpQuad}(X_{i-1}, Y_{i-1}, X_i, Y_i, X_{i+1}, Y_{i+1}, x))$

  where $i = \text{index}(X, x)$
- exception: ($\neg(\text{minX} \leq x \leq \text{maxX}) \Rightarrow$ OutOfDomain)$|\neg(1 \leq \text{index}(X, x) \leq |X| - 2) \Rightarrow$ OutOfDomain) *# first check if within domain, then make sure not too close to edge, so quadratic interpolation is defined*

10.5 Considerations

For simplicity the function evaluation is not defined within one step of the boundaries. By considering the special cases it would be possible to get right to the edge.

11 MIS of ContoursADT Module

11.1 Template Module

ContoursADT

11.2 Uses

FunctADT (Section 10) for FunctT

11.3 Syntax

11.3.1 Exported Constants

MAX_ORDER = 2

11.3.2 Exported Types

ContoursT = ?

11.3.3 Exported Access Programs

Routine name	In	Out	Exceptions
new ContoursT	$i : \mathbb{R}$		InvalidInterpOrder
add	s: FunctT, $z : \mathbb{R}$		IndepVarNotAscending
getC	$i : \mathbb{N}$		InvalidIndex
eval	$x : \mathbb{R}, z : \mathbb{R}$		OutOfDomain
evaly	$x : \mathbb{R}, y : \mathbb{R}$		OutOfDomain
slice	$x : \mathbb{R}$, flip : $\mathbb{B}$	FunctT	

11.4 Semantics

11.4.1 State Variables

S: sequence of FunctT
Z: sequence of $\mathbb{R}$
o: $\mathbb{N}$

11.4.2 State Invariant

None

11.4.3 Assumptions

None

11.4.4 Access Routine Semantics

new ContoursT(i):

- transition: $S, Z, o := <>, <>, i$
- exception: $(i \notin [1..\text{MAX\_ORDER}] \Rightarrow \text{InvalidInterpOrder})$

add(s, z):

- transition: $S, Z := S||<s>, Z||<z>$
- exception: $exc := (|Z| > 0 \wedge z < Z_{|Z|-1} \Rightarrow \text{IndepVarNotAscending})$

getC(i):

- output: $out := S[i]$
- exception: $exc := (\neg(0 \leq i < |S|) \Rightarrow \text{InvalidIndex})$

eval(x, z):

- output: $out := \text{self.slice}(x, \text{False}).\text{eval}(z)$
- exception: none *# appropriate exceptions are generated by slice and eval*

evaly(x, y):

- output: $out := \text{self.slice}(x, \text{True}).\text{eval}(y)$
- exception: none *# appropriate exceptions are generated by slice and eval*

slice(x, flip):

- output and exception:

```
Zdef = []
F = []
for i in [0 .. |S|-1]:
   try y = S[i].eval(x):
     Zdef.append(Z[i])
     F.append(y)
   except OutOfDomain:
     pass
if |Zdef| > 0:
```

```
    if flip:
      return FunctT(F, Zdef, o)
    else:
      return FunctT(Zdef, F, o)
  else:
    raise OutOfDomain
```

12 MIS of SeqServices Module

12.1 Module

SeqServices

12.2 Uses

None

12.3 Syntax

12.3.1 Exported Constants

None

12.3.2 Exported Access Programs

Routine name	In	Out	Exceptions
isAscending	$X : \mathbb{R}^n$	$\mathbb{B}$	
isInBounds	$X : \mathbb{R}^n, x : \mathbb{R}$	$\mathbb{B}$	
interpLin	$x_1 : \mathbb{R}, y_1 : \mathbb{R}, x_2 : \mathbb{R}, y_2 : \mathbb{R}, x : \mathbb{R}$	$\mathbb{R}$	
interpQuad	$x_0 : \mathbb{R}, y_0 : \mathbb{R}, x_1 : \mathbb{R}, y_1 : \mathbb{R}, x_2 : \mathbb{R}, y_2 : \mathbb{R}, x : \mathbb{R}$	$\mathbb{R}$	
index	$X : \mathbb{R}^n, x : \mathbb{R}$	$\mathbb{N}$	

12.4 Semantics

12.4.1 State Variables

None

12.4.2 State Invariant

None

12.4.3 Assumptions

None, unless noted with a particular access program

12.4.4 Access Routine Semantics

isAscending(X)

- output: $out := \neg\exists(i|i \in [0..|X|-2] : X_{i+1} < X_i)$

- exception: none

isInBounds(X, x) # *assuming isAscending is True*

- output: *out* := $X_0 \leq x \leq X_{|X|-1}$
- exception: none

interpLin(x_1, y_1, x_2, y_2, x) # *assuming isAscending is True*

- output: *out* := $\frac{(y_2-y_1)}{(x_2-x_1)}(x - x_1) + y_1$
- exception: none

interpQuad($x_0, y_0, x_1, y_1, x_2, y_2, x$) # *assuming isAscending is True*

- output: *out* := $y_1 + \frac{y_2-y_0}{x_2-x_0}(x - x_1) + \frac{y_2-2y_1+y_0}{2(x_2-x_1)^2}(x - x_1)^2$
- exception: none

index(X, x) # *assuming isAscending is True and isInBounds is True*

- output: *out* := i such that $X_i \leq x < X_{i+1}$
- exception: none

13 MIS of Output Module

13.1 Module

Output

13.2 Uses

Input (Section 5), ThicknessADT (Section 9), GlassTypeADT (Section 8), Hardware (Section 15)

13.3 Syntax

13.3.1 Exported Constants

None

13.3.2 Exported Access Programs

Routine name	In	Out	Exceptions
output	$s : \mathbb{S}$, is_safePb : $\mathbb{B}$, is_safeLR : $\mathbb{B}$, $P_b : \mathbb{R}$, $B : \mathbb{R}$, LR: $\mathbb{R}$, $q : \mathbb{R}$, $J : \mathbb{R}$, NFL: $\mathbb{R}$, $\hat{q} : \mathbb{R}$, $\hat{q}_{\text{tol}} : \mathbb{R}$, $J_{\text{tol}} : \mathbb{R}$	-	-

13.4 Semantics

13.4.1 Environment Variables

outfile: two dimensional sequence of text characters

13.4.2 State Variables

None

13.4.3 State Invariant

None

13.4.4 Assumptions

None

13.4.5 Access Routine Semantics

output(s, is_safePb, is_safeLR, P_b, B, LR, q, J, NFL, $\hat{q}$, $\hat{q}_{\text{tol}}$, J_{tol})

- transition: write data to the file outfile associated with the string s. The data to output follows:
  - From SRS R4: values from SRS R1 (a, b, g, $P_{b_{\text{tol}}}$, SD_x, SD_y, SD_z, t, TNT, w), values from SRS R2 (m, k, E, t_d, LDF, LSF, h, GTF, SD, AR)
  - From SRS R5: (is_safePB $\wedge$ is_safeLR $\Rightarrow$ "For the given input parameters, the glass is considered safe" | True $\Rightarrow$ "For the given input parameters, the glass is NOT considered safe")
  - From SRS R6 (P_b, B, LR, q, J, NFL, $\hat{q}$, $\hat{q}_{\text{tol}}$, J_{tol})
- exception: None

14 MIS of Constants Module

14.1 Module

Constants

14.2 Uses

N/A

14.3 Syntax

14.3.1 Exported Constants

From Table 8 in SRS
$m := 7$
$k := 2.86 \times 10^{-53}$
$E := 7.17 \times 10^{10}$
$t_d := 3$
LSF := 1
$d_{\max} := 5.0$
$d_{\min} := 0.1$
$\text{AR}_{\max} := 5.0$
$w_{\max} := 910.0$
$w_{\min} := 4.5$
$\text{SD}_{\min} := 6.0$
$\text{SD}_{\max} := 130.0$

14.3.2 Exported Types

None

14.3.3 Exported Access Programs

None

14.4 Semantics

14.4.1 State Variables

None

14.4.2 State Invariant

None

15 MIS of Hardware Module

This module hides the underlying hardware for I/O (to the screen, or file, or other device). In general it will be provided by the selected programming language and operating system.

15.1 Module

Hardware

15.2 Uses

None