

# MIS Example

## CS 2ME3/SE 2AA4

Gurankash Singh

Department of Computing and Software  
McMaster University

January 30 - February 3

# Outline

1 What is a module?

2 MIS Template

3 Example

- Point ADT Module
- Point Mass Module

# What is a module?

- A file with encapsulated code to implement a specific functionality
- Ex: For designing a website with a login system, we may have a module that deals with logging out
- A module comes with an "interface"
- An interface includes things like functions and arguments of the function

# What is a MIS?

- Module Interface Specification
- Specifies externally observable behaviour of a module
- Not in language of implementation, but uses mathematical and application language
- Internal implementations are not included in a MIS

# MIS Template Structure

- Uses
  - Imported constants, data types and access programs
- Syntax
  - Exported constants and types
  - Exported functions (access routine interface syntax)
- Semantics
  - State variables
  - State invariants
  - Assumptions
  - Access routine semantics
  - Local functions
  - Local types
  - Local constants
  - Considerations

# Example

Consider implementation of point masses on two dimensional plane

- Position  $P$  is represented by pair of real numbers  $(x,y)$
- Mass  $m$  is represented by a real number
- Considering two point masses that have the corresponding positions and masses of  $P_1$  and  $P_2$  and  $m_1$  and  $m_2$ , respectively, the force  $F$  exerted by one mass on the other is given by Newton's law of universal gravitation:

$$F = G \frac{m_1 m_2}{r^2}$$

where  $r$  is the distance between positions  $P_1$  and  $P_2$  and  $G = 6.672 \times 10^{-11} \text{ N} \cdot \text{m}^2/\text{kg}^2$ .

# Point ADT Module

## Template Module

pointADT

## Uses

N/A

## Syntax

Exported Types:

pointT = ?

## Exported Access Programs

Routine name	In	Out	Exceptions
init	real, real	pointT	
xcoord		real	
ycoord		real	
dist	pointT	real	



## Semantics

State Variables:

xc: real

yc: real

**State Invariant** None

**Assumptions** init() is called for each abstract object before any other access routine is called for that object

# Point ADT Module

## Access Routine Semantics

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

# MIS Interface

From the MIS we can deduce the interface of the code will look like:

```
# Interface
class pointT:

    # Constructor
    def __init__(self, x, y):

    # Selectors
    def xcoord(self):

    def ycoord(self):

    def dist(self, p):
```

# MIS Implementation

See `PointADT.py` for implementation.

# Point Mass Module

## Template Module

pointMassADT

## Uses

PointADT

## Syntax

Exported Types:

pointMassT = ?

## Exported Access Programs

Routine name	In	Out	Exceptions
init	pointT, real	pointMassT	
point		pointT	
mval		real	
force	pointMassT	real	
Fx	pointMassT	real	

## Semantics

State Variables:

*pt*: pointT

*ms*: real

**State Invariant** None

**Assumptions** `init()` is called for each abstract object before any other access routine is called for that object

## Access Routine Semantics

$\text{init}(p, m)$ :

- transition:  $pt, ms := p, m$
- output:  $out := self$
- exception: none

$\text{point}()$ :

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

$\text{mval}()$ :

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



force(*p*):

■ output:

$$out := \text{UNIVERSAL\_G} \frac{self.ms \times p.mval()}{self.pt.dist(p.point())^2}$$

■ exception: none

Fx(*p*):

■ output:

$$out := self.force(p) \frac{p.point().xcoord() - self.pt.xcoord()}{self.pt.dist(p.point())}$$

■ exception: none

## Local Constants

UNIVERSAL\_G =  $6.672 \times 10^{-11}$

# MIS Interface

From the MIS we can deduce the interface of the code will look like:

```
# Interface
class pointMassT:

    #Constructor
    def __init__(self, p, m):

    #Selectors
    def point(self):

    def mval(self):

    def force(self, p):

    def Fx(self, p):
```

# MIS Implementation

See `PointMassADT.py` for implementation.

# Implementation files

- Implementation files PointADT.py and PointMassADT.py can be found in the repo under Tutorial/T4/src