COMP SCI 2ME3 and SFWR ENG 2AA4 Midterm Examination McMaster University

DAY CLASS — Dr. S. Smith
DURATION OF EXAMINATION: 3 hours
MCMASTER UNIVERSITY MIDTERM EXAMINATION — March 4, 2021

NAME: [Rizwan Ahsan —SS]

Student ID: [400232878 —SS]

This examination paper includes 17 pages and 4 questions. You are responsible for ensuring that your copy of the examination paper is complete. Bring any discrepancy to the attention of your instructor.

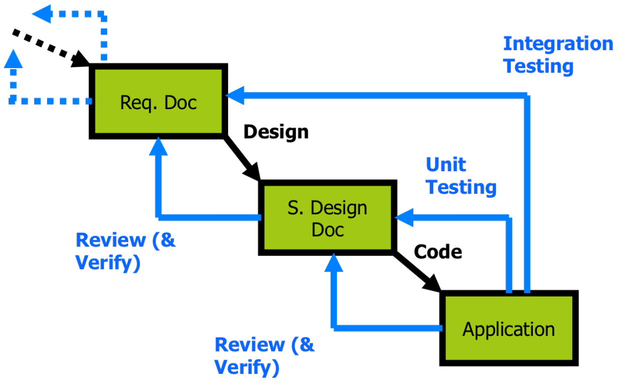
By submitting this work, I certify that the work represents solely my own independent efforts. I confirm that I am expected to exhibit honesty and use ethical behaviour in all aspects of the learning process. I confirm that it is my responsibility to understand what constitutes academic dishonesty under the Academic Integrity Policy.

Special Instructions:

1. For taking tests remotely:
   - Turn off all unnecessary programs, especially Netflix, YouTube, games like Xbox or PS4, anything that might be downloading or streaming.
   - If your house is shared, ask others to refrain from doing those activities during the test.
   - If you can, connect to the internet via a wired connection.
   - Move close to the Wi-Fi hub in your house.
   - Restart your computer, 1-2 hours before the exam. A restart can be very helpful for several computer hiccups.
   - Commit and push your tex file, compiled pdf file, and code files frequently.
   - Ensure that you push your solution (tex file, pdf file and code files) before time expires on the test. The solution that is in the repo at the deadline is the solution that will be graded.
2. It is your responsibility to ensure that the answer sheet is properly completed. Your examination result depends upon proper attention to the instructions.
3. All physical external resources are permitted, including textbooks, calculators, computers, compilers, and the internet.
4. The work has to be completed individually. Discussion with others is strictly prohibited.

5. Read each question carefully.
6. Try to allocate your time sensibly and divide it appropriately between the questions.
7. The set $\mathbb{N}$ is assumed to include 0.

Question 1 [6 marks] Parnas advocates faking a rational design process as depicted in the figure below. The faked documentation follows these steps: Requirements (SRS) → Design (MG and MIS) → Application Implementation (code) → Verification and Validation (Unit Testing, Integration Testing, Review). How are the principles of a) abstraction and b) separation of concerns applied in a rational design process? In your answer you can refer to any aspects of the process, documentation, and/or Parnas's principles.



[Fill in your answer below —SS]

a) Abstraction

- In order to implement the design we are required to use many technologies but we are not required to know the internal details of how those technologies work, we are only required to know how to use it. For example, we might use python as a programming language to implement the design, however, we don't need to know all the inner workings of python to implement the design.
- In the design documentation i.e in the MIS all the required modules and their respective methods are described which are required to implement the design but the exact method of how to achieve that goal is not given, this allows the developers to use their own judgment in deciding the best way to implement it.
- In the application implementation process, there is always more than one way to achieve the goal, like the sorting algorithm used or the data structure used. The designer has the flexibity to choose the best one to use.

b) Separation of Concerns

- In the documentation i.e MIS it is better to create seperate modules that it meant to perform only one task. This allows to seperate the design and break it down into smaller sub-problems to solve.
- We make a Module Guide during the process to decompose the different modules used and identify their uses. This allows the future developers to identify what each module does and make them understand the design rationale.
- The whole design process as a whole serves as a separation of concerns. Each aspect of the process deals with different parts. The requirements process deals with finding out what is required

to implement the design, The documentation process documents all modules implemented and their use cases, the testing process tests the robustness of the application.

Consider the specification for two modules: SeqServices and SetOfInt.

Sequence Services Library

Module

SeqServicesLibrary

Uses

None

Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
max_val	seq of $\mathbb{Z}$	$\mathbb{N}$	ValueError
count	$\mathbb{Z}$, seq of $\mathbb{Z}$	$\mathbb{N}$	ValueError
spices	seq of $\mathbb{Z}$	seq of string	ValueError
new_max_val	seq of $\mathbb{Z}$, $\mathbb{Z} \rightarrow \mathbb{B}$	$\mathbb{N}$	ValueError

Semantics

State Variables

None

State Invariant

None

Assumptions

- All access programs will have inputs provided that match the types given in the specification.

Access Routine Semantics

max_val(s)

- output: $out := |m| : \mathbb{N}$ such that $(m \in s) \wedge \forall(x : \mathbb{Z} | x \in s : |m| \geq |x|)$
- exception: $(|s| = 0 \Rightarrow \text{ValueError})$

count(t, s)

- output: $out := +(x : \mathbb{Z} | x \in s \wedge x = t : 1)$
- exception: $(|s| = 0 \Rightarrow \text{ValueError})$

spices(s)

- output: $out := \langle x : \mathbb{Z} | x \in s : (x \leq 0 \Rightarrow \text{“nutmeg”} | \text{True} \Rightarrow \text{“ginger”}) \rangle$
- exception: $(|s| = 0 \Rightarrow \text{ValueError})$

new_max_val(s, f)

- output: $out := \text{max\_val}(\langle x : \mathbb{Z} | x \in s \wedge f(x) : x \rangle)$
- exception: $(|s| = 0 \Rightarrow \text{ValueError})$

Set of Integers Abstract Data Type

Template Module

SetOfInt

Uses

None

Syntax

Exported Types

SetOfInt = ?

Exported Constants

None

Exported Access Programs

Routine name	**In**	**Out**	**Exceptions**
new SetOfInt	seq of $\mathbb{Z}$	SetOfInt	
is_member	$\mathbb{Z}$	$\mathbb{B}$	
to_seq		seq of $\mathbb{Z}$	
union	SetOfInt	SetOfInt	
diff	SetOfInt	SetOfInt	
size		$\mathbb{N}$	
empty		$\mathbb{B}$	
equals	SetOfInt	$\mathbb{B}$	

Semantics

State Variables

s: set of $\mathbb{Z}$

State Invariant

None

Assumptions

- The SetOfInt constructor is called for each object instance before any other access routine is called for that object. The constructor can only be called once. All access programs will have inputs provided that match the types given in the specification.

Access Routine Semantics

new SetOfInt(x_s):

- transition: $s := \cup(x : \mathbb{Z} | x \in x_s : \{x\})$
- output: $out := self$
- exception: none

is_member(x):

- output: $x \in s$
- exception: none

to_seq():

- output: $out :=$ set_to_seq(s)
- exception: none

union(t):

- output: SetOfInt(set_to_seq(s)$||t$.to_seq())
  # in case it is clearer, an alternate version of output is:
  SetOfInt(set_to_seq($s \cup \{x : \mathbb{Z} | x \in t$.to_seq() $: x\}$))
- exception: none

diff(t):

- output: SetOfInt(set_to_seq($s \cap$ complement(t.to_seq())))
- exception: none

size():

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

empty():

- output: $s = \varnothing$
- exception: none

equals(t):

- output: $\forall(x : \mathbb{Z} | x \in \mathbb{Z} : x \in t$.to_seq() $\leftrightarrow x \in s)$ *# this means:* t.to_seq() $= s$
- exception: none

Local Functions

set_to_seq : set of $\mathbb{Z} \rightarrow$ seq of $\mathbb{Z}$
set_to_seq(s) $\equiv \langle x : \mathbb{Z} | x \in s : x \rangle$ # *Return a seq of all of the elems in the set s, order does not matter*

complement : seq of $\mathbb{Z} \rightarrow$ set of $\mathbb{Z}$
complement(A) $\equiv \{x : \mathbb{Z} | x \notin A : x\}$

Question 2 [15 marks]
[Complete Python code to match the above specification. —SS] The files you need to complete are: `SeqServicesLibrary.py` and `SetOfInt.py`. Two testing files are also provided: `expt.py` and `test_driver.py`. The file `expt.py` is pre-populated with some simple experiments to help you see the interface in use, and do some initial test. You are free to add to this file to experiment with your work, but the file itself isn't graded. The `test_driver.py` is also not graded. However, you may want to create test cases to improve your confidence in your solution. The stubs of the necessary files are already available in your `src` folder. The code will automatically be imported into this document when the `tex` file is compiled. You should use the provided Makefile to test your code. You will NOT need to modify the Makefile. The given Makefile will work for `make test`, without errors, from the initial state of your repo. The `make expt` rule will also work, because all lines of code have been commented out. Uncomment lines as you complete work on each part of the modules relevant to those lines in `expt.py` file. The required imports are already given in the code. You should not make any modifications in the provided import statements. You should not delete the ones that are already there. Although you can solve the problem without adding any imports, if your solution requires additional imports, you can add them. As usual, the final test is whether the code runs on mills.
Any exceptions in the specification have names identical to the expected Python exceptions; your code should use exactly the exception names as given in the spec.
You do not need to worry about doxygen comments. However, you should include regular comments in the code where it would benefit from an explanation.
You do not need to worry about PEP8. Adherence to PEP8 will not be part of the grading.
Remember, your code needs to implement the given specification so that the interface behaves as specified. This does NOT mean that the local functions need to all be implemented, or that the types used internally to the spec need to be implemented exactly as given. If you do implement any local functions, please make them private by preceding the name with double underscores.

Code for SeqServicesLibrary.py

```
## @file SeqServicesLibrary.py
#  @author Rizwan Ahsan
#  @brief Library module that provides functions for working with
   sequences
#  @details This library assumes that all functions will be provided
   with arguments of the expected types
#  @date 03/04/2021


def max_val(s):
    if len(s) == 0:
        raise ValueError
    highest = -1
    for i in s:
        if abs(i) > highest:
            highest = abs(i)
    return highest


def count(t, s: list):
    if len(s) == 0:
        raise ValueError
    return sum([1 for i in s if i == t])


def spices(s):
    if len(s) == 0:
        raise ValueError
    return ["nutmeg" if i <= 0 else "ginger" for i in s]


def new_max_val(s, f):
    if len(s) == 0:
        raise ValueError
    lst = []
    for i in s:
        if f:
            lst.append(i)
    return max_val(lst)
```

Code for SetOfInt.py

```python
## @file SetOfInt.py
#  @author Rizwan Ahsan
#  @brief Set of integers
#  @date 03/04/2021


class SetOfInt():

    def __init__(self, x):
        self.s = set()
        for i in x:
            self.s.add(i)

    def is_member(self, x):
        return x in self.s

    def to_seq(self):
        lst = []
        for i in self.s:
            lst.append(i)
        return lst

    def union(self, t):
        lst1 = self.to_seq()
        lst2 = t.to_seq()
        lst = lst1 + lst2
        return SetOfInt(lst)

    def diff(self, t):
        lst1 = self.to_seq()
        lst2 = t.to_seq()
        for i in lst2:
            if i in lst1:
                lst1.remove(i)
        return SetOfInt(lst1)

    def size(self):
        return len(self.s)

    def empty(self):
        return len(self.s) == 0

    def equals(self, t):
        lst1 = t.to_seq()
        lst2 = self.to_seq()
        #sorts the lists in ascending order
        lst2.sort()
        lst1.sort()
        #iterates over the lists to check if all items are equal
        for i in range(len(lst1)):
            if lst1[i] != lst2[i]:
                return False
        return True
```

Code for expt.py

```
## @file expt.py
#  @author Spencer Smith
#  @brief This file is intended to help test that your interface
   matches the specified interface
#  @date 03/04/2021

from SeqServicesLibrary import *
from SetOfInt import *

# Exercising Sequence Services Library
#print()
# print("SeqServicesLibrary, max_val expt:", max_val([1, 2, -3]))
#print("SeqServicesLibrary, count expt:", count(1, [1, 1, 1, 1]))
#print("SeqServicesLibrary, spices expt:", spices([-5, 0, 23]))
#print("SeqServicesLibrary, new_max_val expt:", new_max_val([-5, 0,
   23], lambda x: x >10))
#print()

# Exercising Set of Integers
#xs = [-9, 6, 23, 21, -5]
#ys = list(xs)
#ys.append(99)
#S = SetOfInt(xs)
#print("SetOfInt, is_member expt:", S.is_member(21))
#print("SetOfInt, to_seq expt:", S.to_seq())
#S2 = SetOfInt(ys)
#S3 = S.union(S2)
#print("SetOfInt, union expt:", S3.to_seq())
#S4 = S2.diff(S)
#print("SetOfInt, diff expt:", S4.to_seq())
#print("SetOfInt, size expt:", S4.size())
#print("SetOfInt, size expt:", S4.empty())
#S5 = SetOfInt([-9, 6, 23, -5, 21])
#print("SetOfInt, equals expt:", S.equals(S5))
#print()
```

Code for test_driver.py

```
## @file test_driver.py
#  @author Your Name
#  @brief Tests implementation of SeqServicesLibrary and SetOfInt ADT
#  @date 03/04/2021

from SeqServicesLibrary import *
from SetOfInt import *

from pytest import *

## @brief Tests functions from SeqServicesLibrary.py
class TestSeqServices:

    # Sample test
    def test_sample_test1(self):
        assert True

## @brief Tests functions from SetOfInt.py
class TestSetOfInt:

    # Sample test
    def test_sample_test2(self):
        assert True
```

Question 3 [5 marks]
Critique the design of the interface for the SetOfInt module. Specifically, review the interface with respect to its consistency, essentiality, generality and minimality. Please be specific in your answer.
[Put your answer for each quality below. —SS]

- **consistency**: The naming conventions of the methods in SetOfInt module is not consistent. The methods that returned a boolean should start with ”is” before them to identitfy them properly. It is consistent in the ordering of the arguments since no method takes in more than 1 argument. None of the methods have any exception handling, so it is consistent in that regard.
- **essentiality**: The module is not essential. The module has a size method and an empty method. The empty method in this case is redundant since the size method can be used to identify whether it is empty or not.
- **generality**: The given module is not general because it only works to create a set of integers and does procedures on that specific set. If the design considered the use of different data type instead of just integers then it could be general. This design is not open-minded meaning it does not leave any future expansion possibility. Down the line there might be a use of set of strings, but since this module only works with set of integers, a whole new module needs to be created.
- **minimality**: Yes, the module is minimal. It is so because none of the access routines are doing more than one task. Each access routine has their own independent service.

Question 4 [4 marks]
The module SetOfInt is for a set of integers. Please answer the following questions related to making that module generic.

a. How would you change the specification to make it generic? (Specifically what changes would you make to the given specification. You don't need to redo the spec, just summarize what changes you would need to make.)

b. What changes would you need to make to the Python implementation to make it generic for type T? (Again, you can describe and characterize the changes; you don't actually have to make them.)

c. What relational operator needs to be defined for type T to be a valid choice?

d. BONUS (1 mark) How would you specify (in the MIS) the relational operator constraint (from the previous question) on the generic type T?

[Put your answer below. —SS]

a. The given specification should say it is a "Generic Template Module" instead of just "Template Module". The name of the class should be changed to SetOfT instead of SetOfInt. So, in all places that have SetOfInt should be replaced with SetOfT. The places where it has integers ($\mathbb{Z}$) and natural ($\mathbb{N}$) should be replaced with T.

b. Since python does not type check the arguments the implementation is already generic. So, no changes needs to be made in the python implementation.

c. In order for the type T to be valid it needs to have the relational operator equal to defined.

d. (BONUS) We need to create a new generic interface module that will be used to check the equality of the items. Then we need to write SetOfT(T with Equality(T)) where Equality(T) is the module that checks for the equality relation for that specific type.