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: [Bhavna Pereira —SS]

Student ID: [400241502 —SS]

This examination paper includes 18 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.

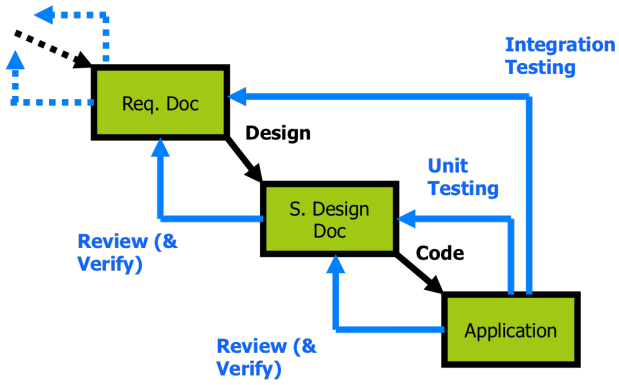
By submitting this work, I certify that the work represents solely my own independent efforts. I confirm that I am expected to exhibit honesty and use ethical behaviour in all aspects of the learning process. I confirm that it is my responsibility to understand what constitutes academic dishonesty under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

- The process of faking a rational design process enables the developer to leave out any mistakes made throughout the actual development process. This uses the principle of abstraction by leaving out details, such as mistakes, which are irrelevant to anyone else attempting to recreate the design process.
- Abstraction is further used in the design process by creating the Module Interface Specifications to highlight key requirements, while not yet determining their methods of implementation.
- The entire rational design process follows a cascade of refinement. The Requirement itself can be extremely abtract. A client may ask the developer to accomplish a task, but not provide any information, or in simpler terms, even care, about how the developer goes about it, as long as thet requirement is being met. The designer then, as mentioned in the point above, decomposes the requirement into important details through an MIS, but still leaves abstraction in that the developer has not yet decided how to implement the criteria. By the actual application process, through code and implementation, the design has becomes rigourous in that it very specifically accomplishes the task at hand. Thus, the process as a whole operates on the principle of abtstraction to refinement as it progesses.

b) Separation of Concerns

- In using design specification to define requirements, the rational design process uses the principle of separation of concerns to detail what exactly needs to be done, for each module that needs to be created. By taking the requirements and decomposing it into smaller problems, this presents a separation of concerns. Within each module itself, the abstraction that comes from what needs to be done and not *how* to do it is yet another separation of concern.

- Integration testing in itself operates on the premise of separation of concerns, in that it is needed to ensure that modules that use other modules work as intended. Moving up in the hierarchy, the design is successful when each module can be run without error. Due to the fact that modules can rely on other modules, the end goal of the Requirements being met is based off of the separation of concerns (individual modules) coming together to accomplish what is being asked.
- Once provided with the requirements, the Design stage of the rational design process allows the developer to extract key details and begin considering likely changes, in order to increase software adaptability. correctivity, and other qualities. Once deciding on modules to take care of these key details, Parnas's design document takes into consideration the MIS, but also the module guide. Within the module guide, there exists a hierarchy, essentially accomplishing decomposition by secrets. Here, there is present the principle of separation of concerns, through behavioural, software and hardware decision hiding modules, which separates key underlying aspects of the requirement and overall design.

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 $\mathbb{Z}$, $\mathbb{Z} \to \mathbb{B}$	$\mathbb{N}$	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($s \cup \{x : \mathbb{Z} | x \in t.\text{to\_seq}() : x\}$))
- 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: $\forall(x : \mathbb{Z} | x \in \mathbb{Z} : x \in t.\text{to\_seq}() \leftrightarrow x \in s)$ *# this means:* $t.\text{to\_seq}() = s$
- 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 Bhavna Pereira
#  @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
        else:
                m = 0
                for i in s:
                        if abs(i) > m:
                                m = abs(i)
                return m



def count(t, s):
        if len(s) == 0:
                raise ValueError
        else:
                result = 0
                for r in s:
                        if r == t:
                                result += 1
        return result


def spices(s):
        if len(s) == 0:
                raise ValueError
        else:
                result = []
                for x in s:
                        if x <= 0:
                                result.append("Nutmeg")
                        else:
                                result.append("Ginger")
        return result
```

```
def new_max_val(s, f):
        if len(s) == 0:
                raise ValueError
        else:
                result = []
                for x in s:
                        if f(x) == True:
                                result.append(x)
                return max_val(result)
```

Code for SetOfInt.py

```
## @file SetOfInt.py
#  @author Your Name
#  @brief Set of integers
#  @date 03/04/2021

class SetOfInt:

        def __init__(self, xs):
                self.__xs = set(xs)


        def is_member(self, x):
                if x in self.__xs:
                        return True
                else:
                        return False

        def to_seq(self):
                seq_equivalent = []
                for x in self.__xs:
                        seq_equivalent.append(x)
                return seq_equivalent

#       def union(self, t):


#       def diff(self, t):

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


        def empty(self):
                if len(self.__xs) == 0:
                        return True
                else:
                        return False

        #def equals(self, t):



@staticmethod
```

```
def set_to_seq(s):
        result = []
        for x in s:
                result.append(x)
        return result

#@staticmethod
#def complement(A):
```

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**: SetofInt lacks consistency in terms of naming convention. Based on `is_member()`, you would expect the `empty()` function to be named `is_empty()` and the `equals` method to be named `is_member`. None of the methods have exceptions, but this consistent with the fact that the assumptions specify that each input will be of the correct type. The inputs and outputs, however are not very consistent in that the programs have a mix of input/output types ranging sets, sequences, booleans, natural numbers, and integers.

- **essentiality**: It is superfluous to implement both the `empty()` and `size()` methods. The `size()` method returns the size of the set, which will return 0 as the natural number output, if the set is, in fact, empty. This omits the necessity for the `empty()` function. Through this, the MIS does not completely adhere to the Quality of essentiality.

- **generality**: The set is not general, in that it operates specifically on sets of integer values. If the module was to be used for a set of any other type (string, character, etc.), in order for it to work.

- **minimality**: Due to the issue of essentiality mentioned above, the ADT is not minimal. In order for it to be minimal, each access routine would perform uniquely. Implementing `size()` **and** `empty()` fails to make this ADT minimal, by accomplishing similar and in one case, the exact same, function, the quality of minimality is tarnished.

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. Instead of an initial input of a set of integer values, it can be made generic by inputting a type itself in order for the constructor method to instantiate a set of that type. By inputting T, where T is any Type, the constructor method can instantiate that Type into a set of that Type, as opposed to a set exclusively of integers.

b. Python uses what is called Duck Typing, in that it "trusts the coder" to adhere to that type throughout the ADT. The way it is implemented in python doesn't specify to the constructor that the set is of integer values, python accepts this and trusts that that object will be treated as the same type throughout the program.

c. *

d. (BONUS)