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: [Enter your name here —SS]

Student ID: [Enter your student number here —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.

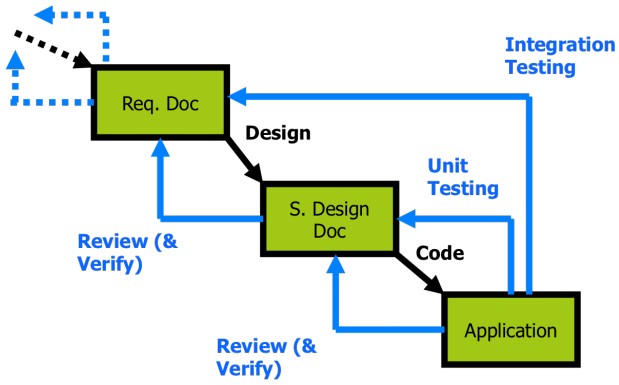
By submitting this work, I certify that the work represents solely my own independent efforts. I confirm that I am expected to exhibit honesty and use ethical behaviour in all aspects of the learning process. I confirm that it is my responsibility to understand what constitutes academic dishonesty under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

- A rational development process was introduced by Parnas and Clements to provide meaning to the ideal process in which programs are derived from its requirements. Despite claiming that a rational design process is needed in order to produce quality software, it is argued that it is impossible to find a process in which the software is designed in a perfectly rational way. Instead, it is suggested to fake this process, by presenting the system and its associated documents to others as if the user had followed this particular idealized design process. By doing these, any unnecessary details or processes that may have occurred during the authentic design process can be left out for future users. This upholds the concept of abstraction, which is the idea of only focusing on what is important in an implementation, and simply ignoring irrelevant processes and details.
- A direct application of the advantages that come with a design that prioritizes abstraction can be described as follows. Once the abstractions of the components public to the user are understood, which in this case would represent the MIS, all of the components of the design would thus be understood without having to look the internals of the modules.
- This serves useful as there is a common practice in the design process called information hiding, where parts of the module that are likely to change are kept hidden from the user. The purpose of information hiding is to anticipate future changes that could occur with the modules themselves. Therefore, the user is able to implement the design as specified by the given documentation, and have a sought after understanding of only the necessary components without having to have access to the modules that are kept hidden. This directly relates to the hierarchy that is created through the module guide that prioritizes information hiding, as repeated applications of abstraction will essentially produce a hierarchy of models.

b) Separation of Concerns

- The separation of concerns is a principle that states that every separate concern must be considered separately in order to reduce a complex problem into a set of simpler ones. Parnas' concept of faking the rational design provides a means of creating this set of simpler problems.
- One way in which this is achieved is through the use of the Module Interface Specification. This specification aims to describe the data and procedures that provide accesses to the services of the modules themselves. An MIS will show what the module does and what services it provides, but it doesn't tell the user how exactly to implement the modules themselves. This is an effective means of taking the pre-defined problem, and splitting it up so that it can be solved in separate modules that will eventually work together to solve that problem.
- Another component of Parnas' rational design process that follows after the design specification is the Module Guide. The module decomposition must be documented to allow for developers to understand and verify the decomposition itself. There exists a hierarchy that defines the module guide by accomplishing the decomposition of information hidden from the clients, otherwise known as secrets. Through behavioural, software and hardware decision hiding modules that separate the hierarchy into three unique categories, separation of change is infact achieved, and the problem is futher subdivided as one continues down the hierarchy.

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: $out :=$ 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 Hamrish Saravanakumar
#  @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
    maximum = 0
    for i in s:
        if abs(i) >= abs(maximum):
            maximum = i
    return abs(maximum)

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

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

def new_max_val(s,f):
    if len(s) == 0:
        raise ValueError
    new = []
    for i in s:
        if f(i):
```

```
            new.append(i)
    return max_val(new)
```

Code for SetOfInt.py

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

class SetOfInt:

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

    def is_member(self, x):
        count = 0
        for i in self.s:
            if i == x:
                count += 1
        if count >= 1:
            return True
        else:
            return False

    def to_seq(self):
        return self.__set_to_seq(self.s)

    def union(self, t):
        return SetOfInt(self.__set_to_seq(self.s) +
           t.__set_to_seq(t.s))

    def diff(self, t):
        return SetOfInt(self.__set_to_seq(self.s - set(t.to_seq())))

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

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

    def equals(self, t):
        return list(self.s) == list(t.to_seq())

    def __set_to_seq(self, s):
        sequence = []
        for i in self.s:
            sequence.append(i)
        return sequence
```

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 MIS achieves consistent naming conventions, as there are no instances where the same variable is given different names to potentially confuse the user. Additionally, the ordering of parameters in the argument list is not necessarily an issue because there is only ever a maximum of one input into any of the methods, but this makes the formatting quite clear to the user. In addition, although there are no explicit exceptions mentioned throughout the implementation of the access routine semantics, a clearly defined assumption is stated saying that the types of the input will be correct and that the constructor will be called before any other access routine. A general assumption for the entire module allows for the consistency to be upheld.

- **essentiality**: An MIS should prioritize being essential in order to omit any unnecessary features that may confuse the user or provide them with more than enough information to create a proper implementation. The entire class itself aims to convert sequences to sets and do different operations on these sets, however python's built-in library has a lot of functionality with such properities through the list() and set() methods. However, the specification itself does do a reasonable job of only including the needed methods and parameters to fully implement the SetOfInt abstract data type. The only exception is the empty() method, that isn't needed because it's purpose can be fulfilled through the use of the size() method, as described in the explanation of minimality below.

- **generality**: Generality is an important quality to take into consideration as it is not good practice to predict how the module will actually be used. However, because the class only works with integers, this reduces the user's ability to use the class with sequences of any other type. Therefore, the MIS would not be considered general unless measures were taken within the specification to explicitly state that the sequences can be of any type.

- **minimality**: This MIS does now allow for a minimal implementation of the SetOfInt class due to the fact that it fails to ensure that every access routine is independent. This is because of the implementation of both the size() method and empty() method. It is unnecessary to have a method that checks the size of the set of integers, as well as a method that checks whether or now a set is empty or not. Instead, a way to make this MIS minimal is to eliminate the empty() method all together. If the user wishes to check if the set is empty, they can call on the size() method, where if a value of 0 is returned, then it would indicate that the set is in fact empty.

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. All inputs and outputs will accept either an integer or a sequence of integers according the the design specification. In order to allow for generalization by allowing multiple types, this would be done by generalizing all such inputs and outputs to type T. Specifically, changing all the occurances of $\mathbb{Z}$ to T would make the specification general. Additionally, the name of the module itself can bs changed to Seq, now that it doesn't only work for type integers and sequences of integers.

b. Making a python implementation generic for type T is useful when the type of data that is wished to be stored is unknown, or if it is desired for the implementation to take on multiple types of data without having to implement extra code for each type. Python achieves this through a method called duck typing, so that it doesn't need special syntax to handle the multiple types. As long as a variable is not defined to a specific type, for example if x = int(3), then python allows for any type to be used.

c. In order to make type T a valid choice, the equals relation operator needs to be defined for type T. This is because the built in python functionality with equals would not suffice for such sets where the type is not defined.

d. In order to specify this in the MIS, a local function for can be defined that would essentially overwrite the built in python equals method, and instead use the one specified that would work for all types.