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 16 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.

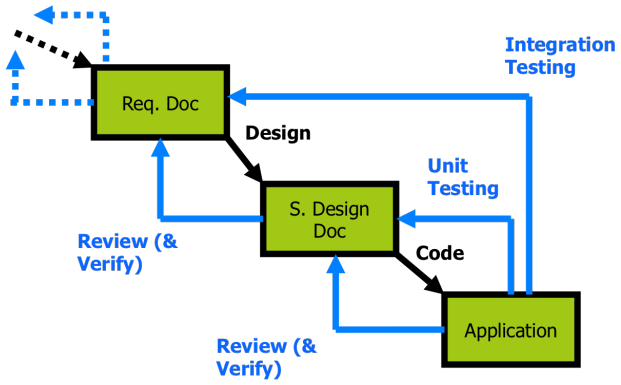
By submitting this work, I certify that the work represents solely my own independent efforts. I confirm that I am expected to exhibit honesty and use ethical behaviour in all aspects of the learning process. I confirm that it is my responsibility to understand what constitutes academic dishonesty under 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 principles of abstraction are applied to every step of the rational design process for software development. In the requirements part of the process, developers collect data that they need to develop a particular software, but this part is abstract to the users because not all projects will share the same requirements.
- At the design level, developers will need to create structured documents such as an MG and MIS document where the document will have specifications for the project's implementation, and again this level is also abstract. Not all projects can use the same specification documents. The MG and MIS documents themselves are usually abstractions of implementation as they contain the implementation's behaviour rather than the solution itself (which follows Parna's Principles).
- The application process is also an abstraction because there is no specific instruction for the developer on how to code or test the solution. The principles of abstraction are applied to every step of the design process; this is important because this minimizes confusion and allows project members to focus on only the tasks relevant to their role.

b) Separation of Concerns

- The principle of concern is the process of separating a complex project into several distinct sections where each section is handled independently; each section will have irrelevant information from other sections kept secret. This is applied to every step of the design process since each level of the process is its own independent process. For example, the requirement phase only focuses on finding the project's knowledge requirements, and it does not concern itself with the implementation phase (and vice versa). This process is necessary because it simplifies a complex project into smaller independent chunks that can be done in an almost linear fashion rather than cycling back and forth between two steps.

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 Safwan Hossain, Hossam18, 400252391
#  @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("List is empty")

    m = 0
    for x in s:
        if m < abs(x):
            m = abs(x)

    return m


def count(t, s):
    if len(s) == 0:
        raise ValueError("List is empty")

    sum = 0
    for x in s:
        if x == t:
            sum += 1

    return sum


def spices(s):
    if len(s) == 0:
        raise ValueError("List is empty")

    return ["nutmeg" if x <= 0 else "ginger" for x in s]


def new_max_val(s, f):
    if len(s) == 0:
        raise ValueError("List is empty")

    return max_val([x if f(x) else False for x in s])
```

Code for SetOfInt.py

```
## @file SetOfInt.py
#  @author Safwan Hossain, Hossam18, 400252391
#  @brief Set of integers
#  @date 03/04/2021

class SetOfInt:

    def __init__(self, x):
        self.s = {a for a in x}

    def set_to_seq(self, s):
        return [x for x in s]

    def is_member(self, x):
        for a in self.s:
            if a == x:
                return True
        return False

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

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

    def diff(self, t):
            diff = []
            ts = t.to_seq()
            ss = self.to_seq()

            for x in ss:
                if x not in ts:
                    diff.append(x)
            return diff


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

    def empty(self):
        return self.size() == 0

    def equals(self, t):
        for x in t.to_seq():
```

```
            if not self.is_member(x):
                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 design is consistent with its naming conventions for example, every word in a name is separated by an underscore. The ordering of variable is also consistent although it should be noted that the design does not frequently use multiple variables as parameters.

- **essentiality**: The design has good essentiality although the method to_seq() is arguably not essential. The method to_seq() does almost the exact same thing as set_to_seq() and is not required when the function set_to_seq() is implemented. But if the developer does not implement set_to_seq() then the design is essential, because it contains only necessary methods and no methods can be used in place of another. I also think the function new_max_ val(s, f) is not very essential because it does almost the same thing as max_val (they both find a max value). I think the design can be more essential by changing new_max_val() to output the sequence that is passed into max_value() instead. This improves the essentiality because max_val() and new_max_val() currently almost does the same thing and one can be used in place of the other, but after this change the two functions will have two different uses.

- **generality**: The design for module SetOfInt is somewhat general because it can be used for any set of integers, but it does not account for other data types. The module can become even more general by changing a few line of codes to allow more data types and objects. The module SequenceServicesLibrary however, has poor generality, some functions are general for example max_val() will return the maximum absolute value of any list, although it only accounts for integers. An example for poor generality is the spices() function, it is not very general because the function has a specific comparison (x is greater than zero) is specific and it outputs a list containing specific values such as nutmeg and ginger. This function cannot be widely used in other implementations and is not considered general.

- **minimality**: The design has excellent minimality as every function only focuses on one task and no function changes a state and outputs a value. The new_max_val() method is questionable for minimality as it modifies a sequence then finds the max value, which may not be minimal.

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. To make it more generic the module can account for more data types (and maybe even reference types) that are not just integers. For example, the design can include all data types such as strings, floats, objects (reference type), etc.

b. The main issue in making the implementation generic is the comparison between two members in a set as comparison is used in several methods of the design (diff(), equals(), is_member()). For data types, python makes it simple because most data types can be compared using ==, but when comparing something more complicated such as objects, the design would need a method for comparing the values of the objects. The design can compare every value of the two objects that are being compared, if the values are the same then they are the same object, otherwise they are two different objects.

c. The relation operator = can be used, but as stated previously, when comparing objects the relation operator would be applied to the values of the objects (data type values) rather than the object themselves. This is because using = on objects would compare their memory addresses instead.

d. (BONUS) a = b