COMP3141 - Final Exam	0h 00m 00s remaining	Zeal Liang	z5325156	Student

COMP3141

Software System Design and Implementation

FINAL EXAM
Term 2, 2021

- Total Number of **Parts**: 5
- Total Number of **Marks**: 100
- Answer **all** questions.
- Excessively verbose answers may lose marks
- Failure to make the declaration or making a false declaration results in a 100% mark penalty.
- Ensure you are the person listed on the declaration.
- All questions must be attempted **individually** without assistance from anyone else.
- You must **save** your exam paper using the button below **before** time runs out.
- **Late submissions will not be accepted**.
- You may save multiple times before the deadline. Only your final submission will be considered.
- You are not permitted to communicate (email, phone, message, talk, ...) with anyone during this exam, except COMP3141 staff via cs3141@cse.unsw.edu.au using your UNSW email account.
- You are not permitted to get help from anyone but COMP3141 staff during this exam.
- If you have issues during the exam, email the course account at cs3141@cse.unsw.edu.au.
- If you need to refer to a specific question, make sure to state the # number for the question.

You have **2 hours and 10 minutes** in total to complete this is exam.

You have **0 seconds** remaining.

You must complete the exam before **2021/8/21下午4:00:43**.

You last saved your answers at **2021/8/21下午4:00:13**.

Declaration of Original Work	
☑	I, **Zeal Liang** (**z5325156**), declare that these answers are **entirely my own**, and that I did not complete this exam with assistance from anyone else.

30

COMP3141 - Final Exam

0h 00m 00s remaining | Zeal Liang | z5325156 | Student

Answer each of the following questions with short responses of a few sentences or lines of code.

Question 1 #0-0

A 2 marks

Describe what is meant by a curried function in functional programming and provide an example of a function both in a curried form and an uncurried form.

Response

```
Curry is pure and takes in a function as its first argument. After inputting the first
argument, it then applies it to the next two arguments. Furthermore, uncurry is the inverse of
curry. Unlike curry, uncurry takes in a function which includes two values, then apply it the
the second argument which is the components of pair.
```

Save All

Question 2 #0-1-0

A 2 marks

Write a datatype definition in Haskell that represents an Email.

An email has:

- the address of the sender,
- the address of each recipient,
- the subject line, and
- the message body.

An email is either:

- a draft,
- an email sent to other addresses,
- or received from another address.

If an email has been sent, it also has the time at which it was sent. If it was received, it has the time at which the email was received.

You may assume that there exists an `Address` datatype that represents email addresses and a `DateTime` data type that represents a specific date and time.

Response

```
data Mail
  = Mail {add       :: Address,
          [add]     :: [Address] ,
          SubjectLine,
```

COMP3141 - Final Exam

0h 00m 00s remaining | Zeal Liang | z5325156 | Student

Section A.III 4 marks

Consider the following datatype definitions and functions:

```
data Natural = Zero | Successor Natural

zero :: Natural
zero = Zero

one :: Natural
one = Successor Zero

add :: Natural -> Natural -> Natural
add Zero          y = y
add (Successor x) y = Successor (add x y)

multiply :: Natural -> Natural -> Natural
multiply Zero          y = Zero
multiply (Successor x) y = add y (multiply x y)
```

Question 3 #0-2-0-0 A.III 2 marks

Write a term or name a function that could be used as the definition of `<>` in a law-abiding instance of `Semigroup` for the `Natural` datatype.

The term must not use the constructors, i.e., it must not use `Zero` or `Successor`.

Response

```
instance Semigroup Natural where
(<>) ::
```

Save All

Question 4 #0-2-0-1 A.III 2 marks

Write a term or name a function that could be used as the definition of `mempty` in a law-abiding instance of `Monoid` for the `Natural` datatype with the `Semigroup` instance from the previous question.

The term must not use the constructors, i.e., it must not use `Zero` or `Successor`.

Response

```
instance Monoid Natural where mempty =
```

COMP3141 - Final Exam

0h 00m 00s remaining | Zeal Liang | z5325156 | Student

Question 5 #0-3-0

A 2 marks

Describe the following property and explain the difference between an instance of `Eq` that satisfies it and one that does not:

```
a == b ==> f a == f b
```

Response

Save All

Question 6 #0-4-0

A 2 marks

Write the following property as a Haskell function:

> If the argument to the function `fastUnique` is sorted, then the resulting value will contain no duplicates.

`fastUnique` has the type `Eq a => [a] -> [a]`.

You must specify the type of the function and the function must return true for all possible arguments. You assume a function `sorted` exists that returns true if a list is sorted and the `nub` that removes all but the first occurence of each element of a list. You can also use the following operator:

```
(==>) :: Bool -> Bool -> Bool
a ==> b = not a || b
```

Response

```
(==>) :: Bool -> Bool -> Bool
a ==> b = not a || b
```

Save All

Section A.VI

6 marks

Consider the following datatype:

```
data Tree a = Leaf a | Node (Tree a) (Tree a)
```

COMP3141 - Final Exam

Question 7 #0-5-1-0 A.VI 2 marks

Provide example arguments for `fmap` and the result of applying those values to `fmap` that show why the following does not satisfy one of the `Functor` laws and name the corresponding law.

```
instance Functor Tree where
  fmap f (Leaf x    ) = Leaf (f x)
  fmap f (Node ls rs) = fmap f ls
```

Response

```
instance Functor Tree where
  fmap f (Leaf x    ) = Leaf (f x)
  fmap f (Node ls rs) = fmap f ls
```

Save All

Question 8 #0-5-1-1 A.VI 4 marks

Provide example arguments and and the result of applying those arguments that show why the identity and interchange laws are not satisfied by the following instance of `Applicative`.

```
instance Applicative Tree where
  pure = Leaf

  Node _ fs <*> xs         = fs <*> xs
  fs        <*> Node xs _  = fs <*> xs
  Leaf f    <*> Leaf x     = Leaf (f x)
```

Response

```
instance Applicative Tree where
  pure = Leaf

  Node   fs <*> xs         = fs <*> xs
```

Save All

Question 9 #0-6 A 2 marks

Explain, briefly, why a pure function has no side effects.

Response

```
Because the same input always gives the same output, meaning that the result of this function
depends only on its arguments, so there are no side effects
```

COMP3141 - Final Exam

Question 10 #0-7-0

A 2 marks

Using `do` notation, show how the following functions can be sequenced together to get the mark for a student in a particular course.

```
lookupStudent :: String -> Maybe Student
```

```
lookupCourse :: String -> Maybe Course
```

```
getMark :: Student -> Course -> Maybe Mark
```

Response

```
fn x = do
  s <- lookupStudent x
  c <- lookupCourse x
  pure (getMark s c)
```

Save All

Question 11 #0-8-1

A 2 marks

Write the following as a Haskell function of type `[a] -> ([a], Int)` without using the `State` monad:

```
do
  xs <- get
  put (map (+1) xs)
  pure (sum xs)
```

Response

```
f :: [Int] -> ([Int], Int)
f xs = (ff xs, sum $ ff xs)
  where ff xss = map (+1) xss
```

Save All

Question 12 #0-9

A 2 marks

Consider the following datatype and functions:

```
data Temperature = Celsius Float | Fahrenheit Float
```

```
toFahrenheit :: Temperature -> Temperature

toFahrenheit (Celsius    x) = Fahrenheit (x * (9 / 5) + 32)
toFahrenheit (Fahrenheit x) = Fahrenheit x

addTemperature :: Temperature -> Temperature -> Temperature
addTemperature (Celsius    x) y = Celsius (x + y')
  where
    Celsius y' = toCelsius y
addTemperature (Fahrenheit x) y = Fahrenheit (x + y')
  where
    Fahrenheit y' = toFahrenheit y
```

Compare the above to the following:

```
data Celsius
data Fahrenheit
newtype Temperature a = T Float

toCelsius :: Temperature Fahrenheit -> Temperature Celsius
toCelsius (T x) = T ((x - 32) * (5 / 9))

toFahrenheit :: Temperature Celsius -> Temperature Fahrenheit
toFahrenheit (T x) = T  (x * (9 / 5) + 32)

addTemperature :: Temperature a -> Temperature a -> Temperature a
addTemperature (T x) (T y) = T (x + y)
```

How are `Celsius` and `Fahrenheit` being used in the second example? What is the advantage to the approach in the second example

Response

```
data Celsius
data Fahrenheit
newtype Temperature a = T Float
```

Save All

Question 13 #0-10 A 2 marks

Consider the following two implementations of a simple language

```
data Term
  = BLit       Bool
  | And        Term Term
  | ILit       Int
  | Plus       Term Term
  | NotGreater Term Term

evalTerm :: Term -> Either Int Bool
evalTerm (BLit x) = Right x
```

COMP3141 - Final Exam | 0h 00m 00s remaining | Zeal Liang | z5325156 | Student

```
    (Right y') = evalTerm y
evalTerm (ILit x) = Left x

evalTerm (Plus x y) = Left (x' + y')
  where
    (Left x') = evalTerm x
    (Left y') = evalTerm y
evalTerm (NotGreater x y) = Right (x' <= y')
   where
    (Left x') = evalTerm x
    (Left y') = evalTerm y
```

```
data Typed a where
  Lit         :: a          -> Typed a
  TAnd        :: Typed Bool -> Typed Bool -> Typed Bool
  TPlus       :: Typed Int  -> Typed Int  -> Typed Int
  TNotGreater :: Typed Int  -> Typed Int  -> Typed Bool

evalTyped :: Typed a -> a
evalTyped (Lit         x  ) = x
evalTyped (TAnd        x y) = (evalTyped x) && (evalTyped y)
evalTyped (TPlus       x y) = (evalTyped x) + (evalTyped y)
evalTyped (TNotGreater x y) = (evalTyped x) <= (evalTyped y)
```

Explain how the errors produced by the following two terms are different and state what property the function `evalTyped` has that `evalTerm` does not that explains this difference.

```
evalTerm (NotGreater (BLit True) (ILit 1))
```

```
evalTyped (TNotGreater (Lit True) (Lit 1))
```

Response

```
data Typed a where
  Lit         :: a          -> Typed a
  TAnd        :: Typed Bool -> Typed Bool -> Typed Bool
  TPlus       :: Typed Int  -> Typed Int  -> Typed Int
```

Save All

Question 14 #0-11-1 A 2 marks

Write the following proof tree as the equivalent term in Haskell and give the equivalent type for the term. Your haskell program must correspond exactly to the provided proof.

A valid program with the correct type that does not correspond exactly will not be accepted as the correct answer.

$$\frac{}{(A \wedge A) \vdash (A \wedge A)}$$

COMP3141 - Final Exam

Response

```
True
```

Save All

Part B

20 marks

Consider the following definitions:

```
map f [] = [] -- map-nil
map f (x : xs) = f x : map f xs  -- map-cons

foldr f b [] = b -- foldr-nil
foldr f b (x : xs) = f x (foldr f b xs) -- foldr-cons
```

You will show that `foldr (\x -> (:) f) []` is equivalent to `map f` using an inductive proof and equational reasoning.

Question 15 #1-0-0 B 2 marks

What is a correct type for the term `foldr (\x -> (:) (x + 1)) []`?

Response

```
[Int] -> [Int]
```

Save All

Question 16 #1-0-1 B 4 marks

Show the evaluation of the term `foldr (\x -> (:) (f x)) [] [1,2]` using equational reasoning. Make sure to show every step.

Response

```
foldr (\x -> (:) (f x)) [] [1,2]
```

Save All

Question 17 #1-0-2 B 3 marks

Show that `foldr (\x -> (:) (f x)) [] []` is equal to `map f []` using equational reasoning. Make sure to show every step.

Response

Save All

Question 18 #1-0-3 B 3 marks

In order to show that the recursive case is true, i.e., that `foldr (\x -> (:) (f x)) [] (y : ys)` is equivalent to `map f (y : ys)`, we must assume an inductive hypothesis. What is the inductive hypothesis?

Response

```
An inductive hypothesis is a hypothesis in the inductive step, which statement holds for a certain value of n.
```

Save All

Question 19 #1-0-4 B 6 marks

Show that `foldr (\x -> (:) (f x)) [] (y : ys)` is equivalent to `map f (y : ys)` using the inductive hypothesis and equational reasoning.

Response

```
map f (y : ys)
= map f y : f ys   -- cons
= [f y : f ys]     -- cons
```

Save All

Question 20 #1-0-5 B 2 marks

What is the time complexity of `map (+ 1) xs` when `xs` has length n?

O(n)

Save All

Part C

20 marks

Question 21 #2-0

C 2 marks

Describe what it means for all operations on a data type to satisfy that datatype's data invariants.

Response

This means that the data type is safe, the length does not change, and the type does not change . f all externally visible functions maintain data invariances, then no external code can build a value that violates those invariances.

Save All

Question 22 #2-1

C 2 marks

How does an abstraction function relate an abstract model to a concrete implementation?

Response

With data refinement, you can move from an abstract model to a fast, concrete implementation.

Save All

Question 23 #2-2

C 2 marks

How does a refinement function relate an abstract model to a concrete implementation?

Response

Data invariants are statements that must always be true of a data structure. We generally

COMP3141 - Final Exam

Save All

Question 24 #2-3 C 2 marks

What are the two parts of a proof of functional correctness for an operation over a concrete implementation that relates it to an abstract datatype?

Response

```
1 All data invariants are maintained, and
2 The implementation is a refinement of the abstract correctness model.
```

Save All

Section C.V 12 marks

Consider the following datatypes where `Timestamp` represents an amount of time as a total number of seconds and `Duration` represents an amount of time as a number of minutes and seconds. There are 60 seconds to a minute.

```
newtype Timestamp = Timestamp Int

data Duration = Duration
  { minutes :: Int
  , seconds :: Int
  }
```

We will consider the `Timestamp` datatype our abstract model and the `Duration` datatype our concrete implementation.

You may wish to use the following functions in your answers to the following questions:

```
divMod :: Int -> Int -> (Int, Int)
divMod x y = (x `div` y, x `mod` y)
```

```
addMod :: [Int] -> Int -> (Int, Int)
addMod xs m = divMod (sum xs) m
```

Question 25 #2-4-0-0 C.V 2 marks

Write the wellformedness predicate for the `Duration` datatype as a Haskell function called `wellformed` that returns `True` if a value is well formed.

COMP3141 - Final Exam

0h 00m 00s remaining | Zeal Liang | z5325156 | Student

```
wellformed (Duration x y) = if y >= 0 && y <= 60 && x >= 0
                            then True
                            else (Duration x y)
```

Save All

Question 26 #2-4-0-1 C.V 2 marks

Write a refinement function called `refine` that refines a value of the `Timestamp` datatype into a value of the `Duration` datatype.

Response

```
refine (Timestamp x) = Duration divMod x 60
```

Save All

Question 27 #2-4-0-2 C.V 2 marks

Write an abstraction function called `abstract` that abstracts a value of the `Duration` datatype into a value of the `Timestamp` datatype.

Response

```
abstract(Duration x y) = Timestamp (y + x * y)
```

Save All

Question 28 #2-4-0-3 C.V 3 marks

Write a Haskell function called `add100` that adds 100 seconds to a value of the `Duration` datatype, producing another value of `Duration`.

Response

```
add100 x = refine (abstract x + 100)
```

Save All

Question 29 #2-4-0-4 C.V 3 marks

refines `addTimestamp100`.

```
addTimestamp100 :: Timestamp -> Timestamp
addTimestamp100 (Timestamp x) = Timestamp (x + 100)
```

Response

A functional correctness specification is a set of properties that completely specify the definition of correctness for a program. It is often expressed as the combination of data invariants and a refinement from an abstract model.

Save All

Part D

20 marks

XML is a text format that can be used to represent structured data. We can encode Haskell datastructures into XML and use the output to load that data into other applications or save it to disk and load it again later.

Json represents a tree of values with data encoded as text. The structure of the tree is represented with tags. An open tag, e.g., `<Datum>`, represents the start of a named node. The corresponding closing tag, e.g., `</Datum>` represents the end of a node. Everything between the tags is considered the children of that node.

For example, we could represent a person as follows:

```
<Person>
  <Name>Simon Peyton Jones</Name>
  <Age>63</Age>
  <Birthday>
    <Year>1958</Year>
    <Month>January</Month>
    <Day>18</Day>
  </Birthday>
</Person>
```

Consider the following GADT that describes a format used to render a Haskell value as a XML string.

```
data Xml :: * -> * where
  Text :: Xml String
  Seq :: Xml a -> Xml a -> Xml a
  Tag :: String -> Xml a -> Xml a
  Repeat :: Xml a -> Xml [a]
  Action :: (a -> b) -> Xml b -> Xml a
  Optional :: {- to be completed -}
```

COMP3141 - Final Exam


```
asXml (Tag name tx) x = (tag "" name) ++ (asXml tx x) ++ (tag "/" name)
asXml (Repeat tx) xs = concat $ map (asXml tx) xs

asXml (Action f tx) x = asXml tx $ f x
{- Optional cases would appear here -}

tag :: String -> String -> String
tag slash s = "<" ++ slash ++ s ++ ">"
```

Question 30 #3-1-0

D 6 marks

Write the type of the `Optional` constructor and the cases that would need to be added to the implementation of `asXml` that match the `Optional` constructor.

When passed to `asXml`, the `Optional` constructor should format a value wrapped in a `Maybe` producing the same output as the value if one is provided and an empty string if no value is provided.

Response

Save All

Section D.II

8 marks

Consider the following untyped datatype for describing formats.

```
data UntypedXml
  = UText
  | USeq UntypedXml UntypedXml
  | UTag String UntypedXml
  | URepeat UntypedXml
  | UOptional UntypedXml
```

Consider how you would implement a function called `asUntypedXml` that is equivalent to `asXml`.

- Its first argument is an `UntypedXml` describes the output format.
- Its second argument is a value to be formatted as the second argument.
- The return value isthe result of formatting.

Question 31 #3-1-1-0

D.II 4 marks

Response

is an Untyped Xml Seq

Save All

Question 32 #3-1-1-1 D.II 4 marks

Assume that `asUntypedXml` must be a total and pure function. What would be the return type of your `asUntypedXml`? Why is this different to `asXml`?

Response

it will return an IO tupe, Unlike asXml, because it returns a string

Save All

Question 33 #3-1-2 D 6 marks

The following are the datatype definitions for a student and the definition of the `Xml` format for a student.

```
data Student = Student
  { name :: String
  , graduated :: Maybe Int
  , grades :: [Grade]
  }

studentXml :: Xml Student
studentXml =
  Tag "Student"
    (     (Action name $ Tag "Name" $ Text)
    `Seq` (Action graduated $ Optional $ Tag "Graduated" $ number)
    `Seq` (Action grades $ Tag "Grades" $ Repeat gradeXml)
    )

number :: Xml Int
number = Action show $ Text

data Grade = Grade
  { course :: String
  , mark :: Maybe Int
  }

gradeXml :: Xml Grade
{- to be completed -}
```

formatting them as XML (with added newlines for clarity).

```
simon = Student
  { name = "Simon The Tortoise"
  , graduated = Nothing
  , grades =
    [ Grade "TORT1917" (Just 94)
    , Grade "TORT2041" (Just 90)
    , Grade "TORT9902" Nothing
    ]
  }
```

The output of `asXml studentXml simon`:

```
<Student>
  <Name>Simon The Tortoise</Name>
  <Grades>
    <Grade>
      <Course>TORT1917</Course>
      <Mark>94</Mark>
    </Grade>
    <Grade>
      <Course>TORT2041</Course>
      <Mark>90</Mark>
    </Grade>
    <Grade>
      <Course>TORT9902</Course>
    </Grade>
  </Grades>
</Student>
```

```
helen = Student
  { name = "Helen The Hare"
  , graduated = Just 2021
  , grades =
    [ Grade "TORT2911" (Just 89)
    , Grade "TURT2631" (Just 54)
    ]
  }
```

The output of `asXml studentXml helen`:

```
<Student>
  <Name>Helen The Hare</Name>
  <Graduated>2021</Graduated>
  <Grades>
    <Grade>
      <Course>TORT2911</Course>
      <Mark>89</Mark>
    </Grade>
    <Grade>
      <Course>TURT2631</Course>
      <Mark>54</Mark>
    </Grade>
```

Provide a definition for `gradeXml` which specifies the `Xml` format for the `Grade` datatype. It should be such that the examples above produce the shown output all on one line.

Response

Save All

Part E

10 marks

Question 34 #4-0-1

E 5 marks

Consider the following natural number and vector datatype:

```
data Nat = Z | S Nat

data Vec (a :: *) :: Nat -> * where
  Nil :: Vec a Z
  Cons :: a -> Vec a n -> Vec a (S n)
```

Provide the type and implementation of the function `dotProduct` which implements vector dot product.

The dot product of two vectors is the sum of the values produced by multiplying together elements from the two vectors in pairs.

Response

```
dotProduct (Vec x1) (Vec x2) = sum (zipWith (*) x1 x2)
```

Save All

Question 35 #4-1-0

E 5 marks

Write the following logical proposition as the equivalent Haskell type then provide a Haskell term with that is total and provide the type.

COMP3141 - Final Exam

0h 00m 00s remaining | Zeal Liang | z5325156 | Student

Response

```
f :: (A -> Void) -> (B -> Void) -> (Either A B -> Viod)
```

Save All

END OF EXAM

(don't forget to save!)

You last saved your answers at **2021/8/21下午4:00:13**.

Submit Exam