Q1 Acknowledgement
0 Points

Australian National University

COMP1100 Mid-Semester Exam, Semester 2 2022

You must acknowledge the following **integrity pledge** before proceeding. Please read carefully and check all the boxes.

- ✓ I am committed to being a person of integrity.
- ✓ I pledge, as a member of the ANU community, to abide by and uphold the standards of academic integrity outlined in the ANU statement on honesty and plagiarism, I am aware of the relevant legislation, and understand the consequences of breaching those rules.
- ✓ I will not communicate in any way with anyone else during this exam. This includes asking questions in any online forum.
- ✓ I acknowledge that this exam is protected by copyright and that copying or sharing any of its content will violate that copyright.

Read and check off the following instructions:

1. This examination is timed.
   - ✓ Note the remaining time at the top right of this screen. Set an alarm for yourself if you need one.

2. Permitted materials. This is an open book exam. You might in particular find the course Website and the Prelude documentation useful.

   - ✓ You may use any documentation you wish but **all work must be your own**.

   - ✓ Importing additional functions or libraries into the programming problems is **strictly prohibited**. Submission which are found importing functions or libraries will receive a **mark of 0**.

3. Autograding. This exam will be automatically graded by gradescope, however this may not be indicative of your official mark for this mid-semester exam. You may gain or lose additional marks. Some examples include:

   3.1. Passing the tests for your programming problems through the use of prohibited libraries (this would result in less marks as per 2. above).

   3.2. Submitting a largely correct solution that does not compile (this will fail the autograder tests, but you may be awarded part marks by the course staff).

   - ✓ I acknowledge that the mark I receive for this assessment on gradescope *may* not be the official mark I receive in Wattle.

4. Marking Scheme

- ✓ This exam is marked out of 100.
- ✓ 40 marks for multiple choice.
- ✓ 60 marks for programming questions.
- ✓ For questions with multiple correct solutions **all** solutions must be checked for the answer to count.

Q2 Sets and Functions
2 Points

Given functions of the following type:

$f :: A \to B$
$g :: B \to C$
$h :: C \to A$

which of the following statements is a valid composition?

- $f \circ f$
- $f \circ g$
- $g \circ f$
- $h \circ f$
- $g \circ h$

Q3 Sets and Functions
2 Points

Given the following sets:

$A = \{1, 2\}$
$B = \{a, b\}$

Which of the following statements is a valid result from $A \times B$?

$\{(1, a), (1, b), (2, a), (2, b)\}$

$\{\}$

$\{(1, a), (2, b)\}$

$\{\{(1, a), (2, a)\}, \{(1, b), (2, b)\}\}$

$\{(2, a \times b)\}$

Q4 Basic Types
4 Points

Given the following function types in Haskell:

```
length :: String -> Int

s :: String
```

which of the following results in a well-typed Haskell expression?

length (s + 10)

length 10 : s

length (10 : s)

length s + 10

length s (+ 10)

Q5 Polymorphic Types
2 Points

Given a Haskell function with the following type:

```
f :: a -> a
```

Tick all valid definitions of this function.

f x = 12

✓ f x = x

✓ f x = snd (x, x)

f x = tail [x]

✓ f x = fst (x, x)

✓ f x = head [x]

Q6 Algebraic Data Types
2 Points

Which of the following is valid syntax for the definition of a `Colour` data type, containing the elements of `Red`, `Green`, `Yellow`, and `Blue`?

Solution A

```
type Colour = Red | Green | Yellow | Blue
```

Solution B

```
data Colour = Red | Green | Yellow | Blue
```

Solution C

```
datatype Colour = Red | Green | Yellow | Blue
```

Solution D

```
data Colour where Red | Green | Yellow | Blue
```

Solution E

```
data Colour = [Red, Green, Yellow, Blue]
```

Select the appropriate solution:

A

B

C

D

E

Q7 Polymorphic ADTs
2 Points

Which of the following types are *not* polymorphic?
Tick all that apply.

Maybe a

[a]

✓ Bool

✓ Integer

✓ ()

✓ ([Int], ())

Either a b

Q8 Recursive ADTs and Recursion
16 Points

Given a list of integers, we would like to write a function that returns the second-to-last element of a list, if that element is followed by the number 3. We need to return a `Maybe Int` as there is no guarantee that the given list *will* have the number 3 in the final

position.

Which of the following are correct implementations of this function?

Tick the checkbox **ABOVE** the code block.

Solution A (below)

```
secondLastIfLastIs3 :: [Int] -> Maybe Int
secondLastIfLastIs3 xs =
  case xs of
   [] -> Nothing
   (x : xs) ->
     case (head xs) of
       3 -> Just x
       k -> secondLastIfLastIs3 xs
```

✓ Solution B (below)

```
secondLastIfLastIs3 :: [Int] -> Maybe Int
secondLastIfLastIs3 xs =
  let go :: [Int] -> Maybe Int -> Maybe Int
      go []     k = Nothing
      go [x]    k
        | x == 3    = k
        | Nothing
      go (x : xs) = go xs (Just x)
  in go xs Nothing
```

✓ Solution C (below)

```
secondLastIfLastIs3 :: [Int] -> Maybe Int
secondLastIfLastIs3 []       = Nothing
secondLastIfLastIs3 [x]      = Nothing
secondLastIfLastIs3 [x, 3]   = Just x
secondLastIfLastIs3 [x, y]   = Nothing
secondLastIfLastIs3 (x : xs) = secondLastIfLastIs3 xs
```

Solution D (below)

```
secondLastIfLastIs3 :: [Int] -> Maybe Int
secondLastIfLastIs3 xs =
  case xs of
   [] -> Nothing
   (x : xs) ->
     case x of
      3 -> Just x
      _ -> secondLastIfLastIs3 xs
```

Q9 Recursion
2 Points

Which of the following is the most appropriate general definition of recursion for this course?

- Recursion is only recursion if it's from the Recurphiné region of France, otherwise it's just sparkling induction.
- Recursion is where we pattern match on a list and call our function on the tail of that list.
- Recursion is where we recurse towards a base case.
- Recursion is the name of the phenomenon occurring where the function or type we are defining references itself inside its definition.

Q10 Equational Resoning
4 Points

Which of the following definitions are equivalent to `f` below?

```
f :: Int -> Int -> Int
f x y = x + y
```

Tick the checkbox **ABOVE** the code block.

✓ Solution A (below)

```
f :: Int -> Int -> Int
f x = (+ x)
```

Solution B (below)

```
f :: Int -> Int -> Int
f x = (+ y)
```

Solution C (below)

```
f :: Int -> Int -> Int
f y = (x +)
```

✓ Solution D (below)

```
f :: Int -> Int -> Int
f = (+)
```

Q11 Local Definitions
4 Points

Which of the following demonstrates well-typed use of the local definition constructs let and where?

Solution A

```
f :: [(a, b)] -> ([a], [b])
f xs =
 where base = ([], [])
 in go xs base
  let:
    -- go :: [(a, b)] -> ([a], [b]) -> ([a], [b])
    go [] acc = acc
    go ((x, y) : xys) acc  = go xys (x : fst acc, y : snd acc)
```

Solution B

```
f :: [(a, b)] -> ([a], [b])
f xs =
  let base = ([],[])
  in go xs base
    where
      -- go :: [(a, b)] -> ([a], [b]) -> ([a], [b])
      go [] acc = acc
      go ((x, y) : xys) acc  = go xys (x : fst acc, y : snd acc)
```

Solution C

```
f :: [(a, b)] -> ([a], [b])
f xs =
 where in base = ([], [])
 let go xs base
      -- go :: [(a, b)] -> ([a], [b]) -> ([a], [b])
      go [] acc = acc
      go ((x, y) : xys) acc  = go xys (x : fst acc, y : snd acc)
```

Solution D

```
f :: [(a, b)] -> ([a], [b])
f xs = let base = ([], []) where go xs base
 in
 -- go :: [(a, b)] -> ([a], [b]) -> ([a], [b])
 go [] acc = acc
 go ((x, y) : xys) acc  = go xys (x : fst acc, y : snd acc)
```

Select the pip corresponding to the correct solution:

A

B

C

D

E

Q12 Programming Questions
60 Points