0/24 Questions Answered

COMP1130 Final Exam

STUDENT NAME

Search students by name or email...

Q1 Acknowledgment

0 Points

Australian National University

COMP1130 Final Exam, Semester 1 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, the Prelude documentation, the Data.List documentation, and the CodeWorld documentation useful.

- [ ] You may use any materials you wish but **all work must be your own**.

Save Answer

Q2 Multichoice questions

20 Points

Each correct answer **gains** you 2 marks, while incorrect and unanswered questions neither gain nor lose marks.

Q2.1
2 Points

Which of the following is an **advantage** of low-level programming languages?

- [ ] Programs resemble specifications
- [ ] Their type systems help to catch errors
- [ ] They offer full control of the machine
- [ ] They offer useful abstractions for a variety of applications
- [ ] They support compositional (piece by piece) reasoning about correctness

Save Answer

Q2.2
2 Points

In the library Codeworld, which of the following is **not** a constructor for an `Event`?

- [ ] KeyRelease
- [ ] PointerClick
- [ ] PointerMovement
- [ ] TextEntry
- [ ] TimePassing

Save Answer

Q2.3
2 Points

Which of the following (mathematically identical) functions has the best **style**?

```
safeExp1 :: Int -> Int -> Maybe Int
safeExp1 x y = case (x == 0 && y == 0) of
  True  -> Nothing
  False -> Just (x^y)

safeExp2 :: Int -> Int -> Maybe Int
safeExp2 x y
  | (x == 0) == False = Just (x^y)
  | (y == 0) == False = Just (x^y)
  | otherwise         = Nothing

safeExp3 :: Int -> Int -> Maybe Int
safeExp3 x y
  | x == 0 && y == 0 = Nothing
  | otherwise        = Just (x^y)

safeExp4 :: Int -> Int -> Maybe Int
safeExp4 x y
  | x == 0 && y == 0 = Nothing
  | x /= 0 || y /= 0 = Just (x^y)

safeExp5 :: Int -> Int -> Maybe Int
safeExp5 x y
  | (x == 0 && y == 0) == True = Nothing
  | otherwise                  = Just (x^y)
```

- [ ] `safeExp1`
- [ ] `safeExp2`
- [ ] `safeExp3`
- [ ] `safeExp4`
- [ ] `safeExp5`

Save Answer

Q2.4
2 Points

Which of the following is **not** mathematically identical to the

identity function on lists of type `[Int]`?

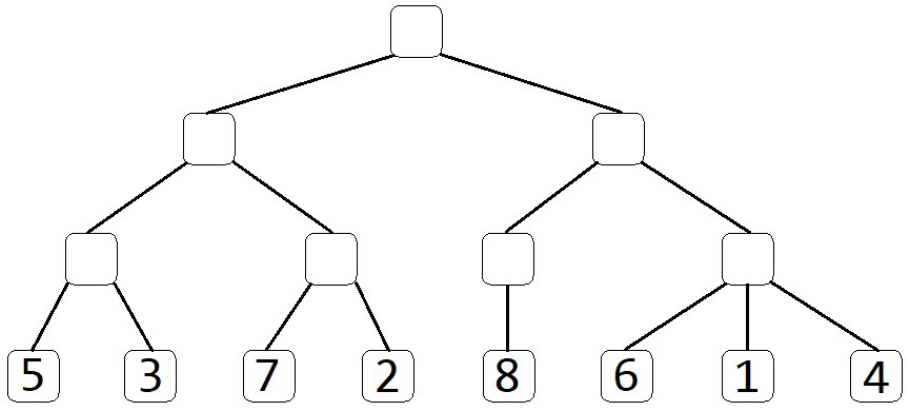
- [ ] `foldl (\x y -> x ++ [y]) []`
- [ ] `foldr (:) []`
- [ ] `filter (const True)`
- [ ] `map (+0)`
- [ ] `zipWith (+) [0..]`

Save Answer

Q2.5

2 Points

Consider the game tree below, where the leaves have been given values according to some heuristic, and each step down the tree represents a change of turn in a two player game. Assuming it is your turn, and you are trying to maximise the heuristic, what value would minimax give the root of the tree?



- [ ] 8
- [ ] 7
- [ ] 6
- [ ] 5
- [ ] 4

Save Answer

Q2.6
2 Points

Which of the below types with context are the most general (i.e. the type variables could be instantiated in the largest number of different ways)?

- [ ] Eq a => a -> a
- [ ] Eq a => a -> b
- [ ] (Eq a, Ord a) => a -> b
- [ ] (Eq a, Ord a) => a -> a
- [ ] Ord a => a -> a
- [ ] Ord a => a -> b

Save Answer

Q2.7
2 Points

Consider the Haskell function:

```
allUnique :: Eq a => [a] -> Bool
allUnique list = case list of
  []     -> True
  x:xs   -> not (elem x xs) && allUnique xs
```

What is the **best case** scenario of `allUnique` with respect to 'big O' time complexity?

- [ ] The first two elements of the list are the same
- [ ] The last two elements of the list are the same
- [ ] The input list contains all unique values and is sorted from largest to smallest
- [ ] The input list contains all unique values and is sorted from smallest to largest
- [ ] The input list is empty

Save Answer

Q2.8

2 Points

What is the **best case** time complexity of `allUnique`?

- [ ] $O(1)$
- [ ] $O(\log n)$
- [ ] $O(n)$
- [ ] $O(n \log n)$
- [ ] $O(n^2)$
- [ ] $O(n^2 \log n)$
- [ ] $O(n^3)$
- [ ] $O(n^3 \log n)$
- [ ] $O(n^4)$
- [ ] $O(2^n)$

Save Answer

Q2.9
2 Points

What is the **worst case** time complexity of `allUnique`?

- [ ] $O(1)$
- [ ] $O(\log n)$
- [ ] $O(n)$
- [ ] $O(n \log n)$
- [ ] $O(n^2)$
- [ ] $O(n^2 \log n)$
- [ ] $O(n^3)$
- [ ] $O(n^3 \log n)$
- [ ] $O(n^4)$
- [ ] $O(2^n)$

Save Answer

Q2.10
2 Points

Consider the Haskell function (recalling that `sort` is implemented in Data.List as Merge Sort):

```
ordAllUnique :: Ord a => [a] -> Bool
ordAllUnique = sortedAllUnique . sort
  where
    sortedAllUnique list = case list of
      []     -> True
      [_]    -> True
      x:y:ys -> x /= y && sortedAllUnique (y:ys)
```

What is the **worst case** time complexity of `ordAllUnique`?

- [ ] $O(1)$
- [ ] $O(\log n)$
- [ ] $O(n)$
- [ ] $O(n \log n)$
- [ ] $O(n^2)$
- [ ] $O(n^2 \log n)$
- [ ] $O(n^3)$
- [ ] $O(n^3 \log n)$
- [ ] $O(n^4)$
- [ ] $O(2^n)$

Save Answer

Q3 Lambda Calculus

20 Points

In your answers below you may write `\` for λ, and `|-` for $\vdash$. You may write `->` to indicate that an expression beta-reduces to another.

If you are asked to present a typing derivation, use a numbered list of typing judgements, with a clear justification beside each line, e.g.

```
1, x : Bool -> alpha, y : Bool |- x : Bool -> alpha [Variable]
2, x : Bool -> alpha, y : Bool |- y : Bool          [Variable]
3, x : Bool -> alpha, y : Bool |- x y : alpha       [1,2,Application]
4, x : Bool -> alpha |- \y. x y : Bool -> alpha     [3,Lambda]
```

Q3.1
3 Points

Present every step of the **lazy** reduction of
$(\lambda x.(\lambda y.y\ y)((\lambda z.x)z))z$

Enter your answer here

Save Answer

Q3.2
3 Points

Present every step of the **strict** reduction of the same term:
$(\lambda x.(\lambda y.y\ y)((\lambda z.x)z))z$

Enter your answer here

Save Answer

Q3.3
4 Points

Suppose we have access to encodings for Booleans and natural numbers, with `True`, `False`, `if..then..else`, `0`, `isZero`, `succ`, and `pred`, along with the fixed point combinator `Y`. Do not make any assumption about which encoding you are using (e.g. Church vs Scott numbers). You may assume that `pred 0 = 0` in the encoding used.

Using these encodings, write a lambda term `divide` that takes two natural number inputs, and returns the division of the first number by the second **rounded up**. You may treat division by zero as you wish, including by failing to terminate.

Enter your answer here

Save Answer

Q3.4
4 Points

Give the principal type, if any, of
$\lambda x.\lambda y.x(y\,x)$

Justify your answer either by presenting a full typing derivation, or by giving a clear explanation why there is no principal type.

Enter your answer here

Save Answer

Q3.5
3 Points

Suppose we have the simply typed lambda calculus extended with Bool and Nat.

Give a complete type checking derivation to show that
$\vdash \lambda y.(\lambda x.x\ \mathsf{True})y : (\mathsf{Bool} \to \mathsf{Nat}) \to \mathsf{Nat}$

Enter your answer here

Save Answer

Q3.6
3 Points

We could extend the Simply Typed Lambda Calculus with a new finite base type called Ordering, with the following rules for elements and elimination:

$$\frac{}{\Gamma \vdash \texttt{LT} : \texttt{Ordering}} \qquad \frac{}{\Gamma \vdash \texttt{EQ} : \texttt{Ordering}} \qquad \frac{}{\Gamma \vdash \texttt{GT} : \texttt{Ordering}}$$

$$\frac{\Gamma \vdash O : \texttt{Ordering} \quad \Gamma \vdash A : \tau \quad \Gamma \vdash B : \tau \quad \Gamma \vdash C : \tau}{\Gamma \vdash \texttt{case}\ O\ \texttt{of}\ A;\ B;\ C : \tau}$$

and the following reduction rules:

$$\texttt{case}\ \texttt{LT}\ \texttt{of}\ A;\ B;\ C \to A$$
$$\texttt{case}\ \texttt{EQ}\ \texttt{of}\ A;\ B;\ C \to B$$
$$\texttt{case}\ \texttt{GT}\ \texttt{of}\ A;\ B;\ C \to C$$

Suggest an encoding of the operations of Ordering into the **untyped** lambda calculus.

Enter your answer here

Save Answer

Q4 Programming Questions
60 Points

There are **seven** programming questions that you need to complete and submit.
You can find links to submit your programming questions on your dashboard.

Please submit by uploading **each** Haskell file to **each** question.

Each function will be marked individually for correctness and code quality. Each function is worth **5 marks**.

Please download the template Haskell files here.

Q4.1 EtaReduction.hs
5 Points

Submit `EtaReduction.hs` here

Save Answer

Q4.2 RepeatedPowers.hs
5 Points

Submit `RepeatedPowers.hs` here

Save Answer

Q4.3 ListFunctions.hs
10 Points

Submit `ListFunctions.hs` here

Save Answer

Q4.4 BoolFunctions.hs
10 Points

Submit `BoolFunctions.hs` here

Save Answer

Q4.5 Dogs.hs
10 Points

Submit `Dogs.hs` here

Save Answer

Q4.6 RoseTrees.hs
10 Points

Submit `RoseTrees.hs` here

Save Answer

Q4.7 Intervals.hs

10 Points

Submit `Intervals.hs` here

Save Answer

Save All Answers

Submit & View Submission >