Midterm Exam

Instructions:

- Read and understand all the instructions on this page.
- The **first page** of your answer booklet must contain only the following details:
  - Your full name
  - Your official Ashoka University email address
  - The title "ICS Midterm Exam"

  No additional information should be included on this page.
- You are allowed to use **one A4-sized handwritten cheat-sheet**. No additional notes, printed materials, or digital resources are allowed.
- If you have any questions or need clarification during the exam, please direct them to the teaching staff or invigilators. Discussions with other students are strictly prohibited.
- Any form of plagiarism, including copying from another student, any discussion among students, any form collaboration, or using external resources beyond what is allowed, will result in an immediate **F grade** for the entire course.
- The exam has a total score of **110 points**.
  - You must answer the first two questions (60 points).
  - For the remaining 50 points, you must choose:
    - **Either** Question 3 **or** Question 4 (20 points).
    - **Either** Question 5 **or** Question 6 (30 points)

  Selecting more than the required questions will not provide extra credit.
- Each new solution (not sub-parts within a question) must **begin on a new page**. This ensures clarity and organization in grading.
- Clearly state the type of every OCaml function and expression, and state any assumptions that you make in your responses and solutions.

Question	Points
Hardy–Ramanujan Numbers (compulsory)	40
Wake me up when Summing Ends (compulsory)	20
Don't Panic, Just Repeat! (do this or 4)	20
Mind the Gap! (do this or 3)	20
Have You Started the Assignment? (do this or 6)	30
It's Giving... NPC (do this or 5)	30
Total:	160

Problem 1. [40 points] **Hardy–Ramanujan Numbers (compulsory)** (4 parts)
A taxicab number is a number that can be expressed as the sum of two positive cubes in more than one way. The smallest taxicab number is 1729, since:

$$1729 = 1^3 + 12^3 = 9^3 + 10^3$$

Here is an OCaml code for checking if a number is a taxicab number:

```
1  let cube x = x * x * x ;;
2
3  let rec find_cubic_pairs a b n fst_pair =
4    match (cube a, cube b) with
5    | (ca, cb) when ca > n -> None
6    | (ca, cb) when ca + cb = n -> (
7        match fst_pair with
8        | None -> find_cubic_pairs a (b + 1) n (Some (a, b))
9        | Some _ -> find_cubic_pairs a (b + 1) n fst_pair
10     )
11   | (ca, cb) when ca + cb > n ->
12           find_cubic_pairs (a + 1) (a + 1) n fst_pair
13   | _ -> find_cubic_pairs a b n fst_pair
14 ;;
15
16 let is_taxicab_number n =
17   match find_cubic_pairs 1 1 n None with
18   | (a1, b1) -> true
19   | None -> false
20 ;;
```

(a) [9 points] Annotate the three functions in the program with correct type annotations.

(b) [12 points] Identify three mistakes in the code that need to be fixed so that the code is not only functionally correct but also well typed. Suggest a way to fix each one of them.

(c) [12 points] For four of the match expressions in '`find_cubic_pairs`' (line 8, 9, 11, and 13) give inputs for the function that match with them. The first one (line 5) is done for you:

```
find_cubic_pairs 50 1 1 None
```

(d) [7 points] Write (in OCaml syntax) a recursive program to compute the next taxicab number.

Problem 2. [20 points] **Wake me up when Summing Ends (compulsory)** (3 parts)
Write (in OCaml syntax) the following three functions:

(a) [5 points] `addall: int list -> int` that computes the sum of a list of elements.

(b) [7 points] `addalt: int list -> int` that computes the *sum of alternate elements* in a list, ignoring the rest. For example:

1. '`addalt [1; 2; 3; 4]`' should evaluate to '`4`' (because $1 + 3 = 4$)
2. '`addalt [5; 10; 15]`' should evaluate to '`20`' (because $5 + 15 = 20$)
3. '`addalt []`' should evaluate to '`0`'.

(c) [8 points] `addsub: int list -> int` that computes the *alternating sum or difference* of a list of integers. The zeroth element is added, the first is subtracted, the second added, and so on. For example:

1. '`addsub [1; 2; 3; 4]`' should evaluate to '`-2`' (because $1 - 2 + 3 - 4 = -2$)
2. '`addsub [5; 10; 15]`' should evaluate to '`10`' (because $5 - 10 + 15 = 10$)
3. '`addsub []`' should evaluate to '`0`'.

Problem 3. [20 points] **Don't Panic, Just Repeat! (do this or 4)** (1 part)
Define, that is, write in OCaml syntax, a function:

```
layer: (int -> int) -> int -> int -> int
```

such that it takes as input a function '`f`', an initial value '`x`', and a number '`n`', and applies '`f`' to '`x`' exactly '`n`' times. For example:

1. '`layer (fun x -> x * 2) 1 3`' should evaluate to '`8`' (because $1 \times 2 \times 2 \times 2 = 8$)
2. '`layer (fun x -> x + 1) 5 2`' should evaluate to '`7`' (because $5 + 1 + 1 = 7$)
3. '`layer (fun x -> x) 42 0`' should evaluate to '`42`' (as is the answer to all questions in the Universe.[1])
4. '`layer (fun x -> x * x) 2 2`' should evaluate to '`16`' (because $(2^2)^2 = 4^2 = 16$)

What happens to your function when '`n`' is negative?

Problem 4. [20 points] **Mind the Gap! (do this or 3)** (2 parts)
Given three integer parameters, '`height`', '`width`', and '`gap`', generate a structured grid representation according to the following specifications:

(a) [10 points] Write a program to draw a rectangular grid with the specified dimensions.

1. The spacing between adjacent lines, both horizontally and vertically, is '`gap`' units.
2. The horizontal grid lines must be coloured blue.
3. The vertical grid lines must be coloured red.
4. The grid should contain only vertical and horizontal lines (no diagonal lines).

(b) [10 points] Modify the grid by adding *dots* (say with dot-size 5) at each point where a vertical and horizontal grid line intersect.

Problem 5. [30 points] **Have You Started the Assignment? (do this or 6)** (4 parts)
As discussed in assignment 5, a polynomial with integer coefficients can be represented as a list. For example, the polynomial $1 + 2x + 3x^2$ can be represented as a list `[1.0; 2.0; 3.0]`, where each element denotes the coefficient and the index denotes the corresponding exponent. A higher-order function is a function that takes another function as an input. In this problem, you will use higher-order functions to define polynomial integration.

[1] This is a joke. The expression evaluates to 42 because the lambda function is simply the identity function.

(a) [5 points] The function `eval_poly` evaluates a polynomial '`p`' for a given value '`x`' using Horner's method. The Horner's method is an efficient way to evaluate polynomials by rewriting them in a nested form. For example, the polynomial $1 + 2x + 3x^2$ can be rewritten as $1 + x(2 + x(3))$. In general, we can write:

$$p(x) = \sum_{i=0}^{n} a_i x^i = a_0 + x \times \left(\sum_{i=1}^{n} a_i x^{i-1} \right)$$

Using higher order function `fold_right`, the function is partially implemented as follows:

```
let eval_poly p x = fold_right (__________) p 0.0 ;;
```

Fill in the expression _____ with a lambda function so that `eval_poly` correctly evaluates the polynomial. If you are unable to use a lambda function, define a helper function to achieve the same result for partial credit.

(b) [9 points] The indefinite integral of a polynomial $p(x) = \sum_{i=0}^{n} a_i x^i$ is given by:

$$\int p(x)dx = \left(\sum_{i=0}^{n} a_i \frac{x^{i+1}}{i+1} \right) + C$$

where C is the constant of integration. For example, the indefinite integral of $1 + 2x + 3x^2$ is $x + x^2 + x^3 + C$. Using only higher-order functions (such as `map`, `filter`, and `fold_right`) and lambda functions, define a function `integrate_poly_indef` that computes the indefinite integral of a polynomial. If you are unable to use a lambda function, define a helper function to achieve the same result for partial credit. Clearly state the type definition of the function and any assumptions that you make.

(c) [6 points] The definite integral of a polynomial $p(x) = \sum_{i=0}^{n} a_i x^i$ over a closed and bounded interval $[a, b]$ is defined as:

$$\int_a^b p(x).dx = F(b) - F(a) \text{ where } F(x) = \sum_{i=0}^{n} a_i \frac{x^{i+1}}{i+1}$$

Define a function `integrate_poly_def` using only higher-order functions (such as `map`, `filter`, and `fold_right`) and lambda functions to compute the definite integral of a polynomial over a given interval $[a, b]$.

(d) [10 points] Kaagazi, the Origami Club at Ashoka, is forcing their folding propaganda on CS-1102. They claim that both `map` and `filter` can be written as folds with some appropriate lambda functions (or helper functions). Either show how `map` and `filter` can be implemented as specific instances of `fold_right`, or provide a counter argument explaining why this is not possible.

Problem 6. [30 points] **It's Giving... NPC (do this or 5)** (4 parts)
Consider a list of n elements:

$$l = [x_0; x_1; \cdots; x_{n-1}]$$

A sublist $l' \sqsubseteq l$ is a list where every element in l' appears in l in the same order, though not necessarily consecutively. That is, l' is of the form:

$$l' = [x_{i_0}; x_{i_1}; \ldots; x_{i_k}]$$

where $0 \leq i_0 < i_1 < \cdots < i_k < n$. Note that while this definition parallels subsets, it is different because lists are ordered and may contain repeated elements.

(a) [5 points] Design a function $\mathrm{verifySum}$ that checks if elements of l' sum to S. For example:

1. The function $\mathrm{verifySum}$ on sublist $[1; 2; 4]$ and sum 7 returns `true`,
2. While $\mathrm{verifySum}$ on sublist $[1; 1; 2; 4]$ and sum 7 returns `false`.

Write $\mathrm{verifySum}$ in OCaml syntax with the appropriate type declarations.

(b) [10 points] Design a function $\mathrm{sublistSum}$ that given a list l and a sum S, checks if there exists a sublist $l' \sqsubseteq l$ such that elements of l' sum to S. For example:

1. $\mathrm{sublistSum}$ on list $l = [1; 2; 3; 4]$ and sum 5 returns either $l_1 = [1; 4]$ or $l_2 = [2; 3]$ as $l_i \sqsubseteq l$ and the elements in l_i sum up to 5.
2. $\mathrm{sublistSum}$ on list $[1; 2; 4; -8; 16]$ and sum 500 does not return a sublist as none of its sublists can sum up to that large a number.

Write $\mathrm{sublistSum}$ in OCaml syntax with the appropriate type declarations.

(c) [7 points] How would you characterise the correctness of $\mathrm{verifySum}$ and $\mathrm{sublistSum}$? Make sure your definition is rigorous and the following incorrect functions do not satisfy it:

```
1 let sublistSum1 l s = None ;;
2
3 let sublistSum2 l s = if (fold_left (+) 0 l) = s then l else [] ;;
4
5 let sublistSum3 l s = [s] ;;
6
7 let sublistSum4 l s = l ;;
```

Formally prove the correctness of your implementations of $\mathrm{verifySum}$ and $\mathrm{sublistSum}$ with respect to your correctness characterisation.

(d) [8 points] Consider a function $\mathrm{expList}$ that on input $n \in \mathbb{N}$, returns the list $[1; 2; 4; \cdots; 2^n]$. Prove that for all $k \in \mathbb{N}$, if $k \leq 2^n$, then $\mathrm{sublistSum}$ with list $\mathrm{expList}(n)$ and sum k always returns some sublist l'. You may need to use the principle of mathematical induction.