COL100: Introduction to Computer Science

5.1: Using helper functions

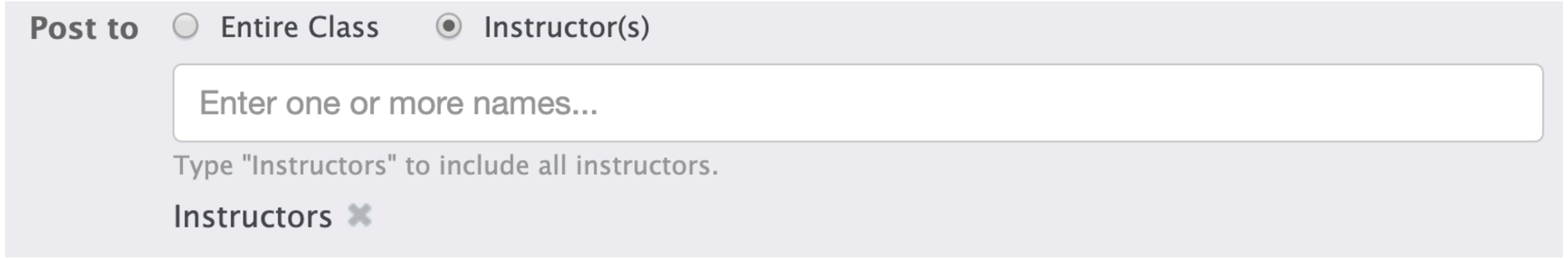
Announcements

1. How to contact us on Piazza
2. Assignments and quizzes
3. Extra classes in Hindi

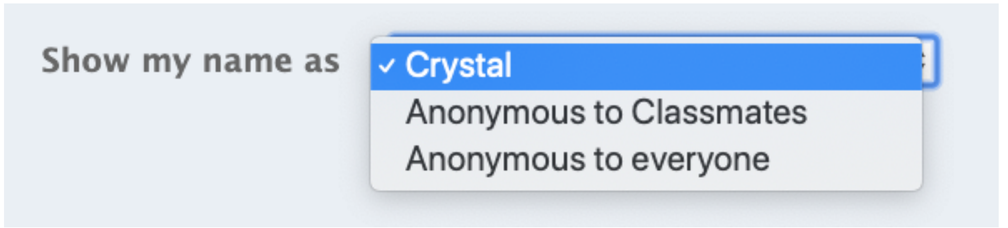
Piazza

Please use Piazza for all questions

- Conceptual doubts, general questions, etc.: make visible to entire class



  - If you don’t want to reveal your identity: post as anonymous



- Only if question is specific to you / has private info: make visible to instructors

Assignments and quizzes

Assignments about every 1.5 weeks

- Submit two files:
  1 PDF file with analysis and proofs (handwritten or typed),
  1 SML file with all your code

Surprise quizzes about once every week

- First ~15 minutes of interactive session

- Lowest 2 quizzes dropped, no extra make-up quizzes for any reason

Extra classes in Hindi

- Sundays at 9:30 AM in main Teams channel (2001-COL100)
- Open to students from all groups
- Will be taught by Prof. Keerti Choudhary and Prof. Venkata Koppula

Quick note: tuples

Cartesian product in set theory:

$$\text{If } a \in A \text{ and } b \in B \text{ then } (a, b) \in A \times B.$$

In SML, same notation `(a, b)`, e.g.

```
(4, true) : int * bool
```

Useful to get multiple values out of a function:

```
fun divMod(n, d) = (n div d, n mod d);
val (q, r) = divMod(10, 3);
```

Then `divMod : int * int → int * int`, and the result is $q = 3$, $r = 1$.

Quick note: tuples

Example: Find the largest power of 2 that divides a number *n*, and the resulting remainder.

- If *n* is odd, largest power is 0, remainder is *n*.
- If *n* is even, largest power is 1 + largest power that divides *n*/2, remainder is remainder of *n*/2.

```
fun factor2(n) =
  if n mod 2 = 1
  then (0, n)
  else let val (p, r) = factor2(n div 2)
       in (p + 1, r)
       end;
```

A more complicated problem

A positive integer is called a *perfect number* if the sum of its proper divisors equals itself.

$$6 = 1 + 2 + 3$$

$$28 = 1 + 2 + 4 + 7 + 14$$

Design a function *perfect* : $\mathbb{N} \rightarrow \mathbb{B}$ to check if a number is perfect or not.

- **Note:** Output type should be Boolean (*true* or *false*), not “yes”, “no”, etc.

Defining *perfect*

$$perfect : \mathbb{N} \to \mathbb{B}$$
$$perfect(n) = ?$$

No good to try recursion directly on n

Instead, apply definition: n is perfect if and only if n = sum of divisors of n

Assume we have a function $sumOfDivisors : \mathbb{N} \to \mathbb{N}$, then

$$perfect(n) = (n = sumOfDivisors(n))$$

Point: Don’t hesitate to introduce helper functions to make the problem simpler.

Defining *sumOfDivisors*

$$sumOfDivisors : \mathbb{N} \to \mathbb{N}$$
$$sumOfDivisors(n) = ?$$

Again, recursion via $sumOfDivisors(n-1)$ is not useful

Many possible approaches

Recall previously defined summation function, $sum : (\mathbb{N} \to \mathbb{N}) \times \mathbb{N} \times \mathbb{N} \to \mathbb{N}$.
$sum(f, a, b)$ computes $\sum f(i)$ over $i = a, a+1, \ldots, b$.

Can we use *sum* to compute *sumOfDivisors*?

Defining *sumOfDivisors* using *sum*

sumOfDivisors should add up divisors of *n* between 1 and *n* – 1, so

$$sumOfDivisors(n) = sum(???, 1, n - 1)$$

What should be the function being summed?

$$f(i) = \begin{cases} i & \text{if } n \bmod i = 0, \\ 0 & \text{otherwise.} \end{cases}$$

$$sumOfDivisors(n) = sum(f, 1, n - 1)$$

Translating into SML

```
fun sumOfDivisors(n) =
  let
    fun f(i) =
      if n mod i = 0
      then i
      else 0;
  in
    sum(f, 1, n - 1)
  end;

fun perfect(n) = (n = sumOfDivisors(n));
```

Question: Why does `f` have to be local in `sumOfDivisors`?

Testing

Check: `perfect(4)` is false, `perfect(5)` is false, `perfect(6)` is true.

One Boolean doesn't give much information. Check the helper functions too!

- `sumOfDivisors(4)` = ?, `sumOfDivisors(5)` = ?, `sumOfDivisors(6)` = ?
- Pull logic out of inner functions if necessary

```
fun ifDivisor(i, n) = if n mod i = 0 then i else 0;
fun sumOfDivisors(n) =
  let fun f(i) = ifDivisor(i, n);
  in sum(f, 1, n - 1)
  end;
```

Defining *sumOfDivisors*

If you didn't want to use *sum*, you could do the summation yourself:

$$sumOfDivisors(n) = ???$$

Need an additional argument to recurse over

$$sumOfDivisors(n) = sumUpto(n, n-1)$$

$$sumUpto(n,k) = \begin{cases} sumUpto(n,k-1)+k & \text{if } k > 0 \text{ and } n \bmod k = 0, \\ \ldots & \ldots \end{cases}$$

Point: Helper functions may need more arguments so recursion is possible.