Needs ANU Account!

pollev.com/fabianm
Register for Engagement

Australian National University

Structured Programming

COMP1110/6710

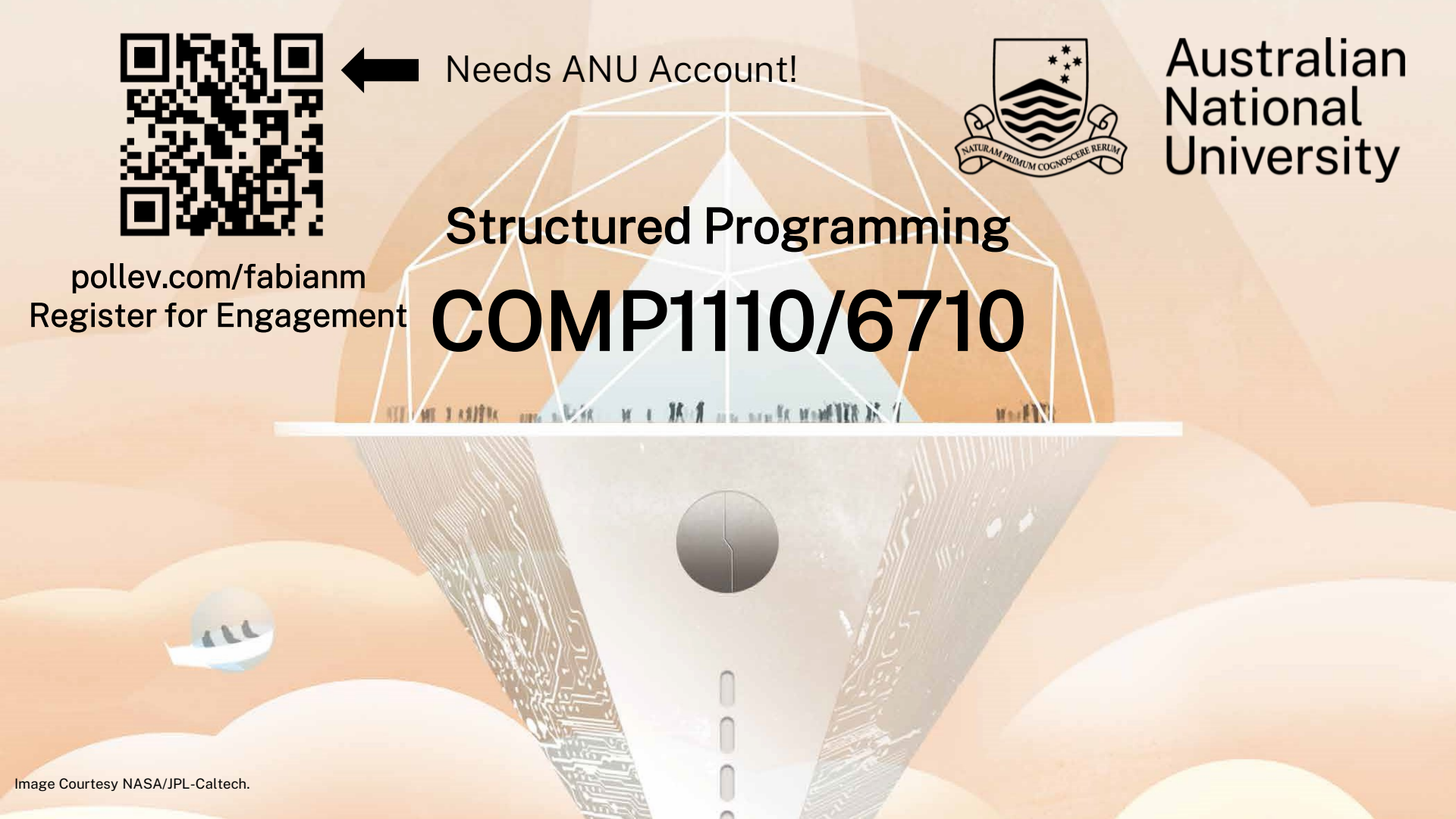


Image Courtesy NASA/JPL-Caltech.

(Mutable) State

The stuff that some of you have already been using, even though they should not have

Still illegal in P2, and only partially allowed (i.e. using HashMaps) in the last part of U2

Australian National University

Functional Programming

As in Functional Java, Haskell, and others

A somewhat mathematical view of the world:
functions always return the same value given the same arguments.

```
var x = f(5);
var y = f(5);
testEqual(x, y); //should always succeed
```

Functional Programming

As in Functional Java, Haskell, and others

A somewhat mathematical view of the world:
functions always return the same value given the same arguments.

```
var x = RandomNumber(5);
var y = RandomNumber(5);
testEqual(x, y); //should always succeed?
```

Imperative Programming

As in Java, Python, and others

Closer to what computers actually do, and useful for modelling things that naturally change.

“We return a new World that is just like the old one, except that the rocket’s Y-coordinate is 5 less than before.”

vs.

“We change the World’s rocket’s Y-coordinate to 5 less than before.”

Imperative Programming

As in Java, Python, and others

Not always great: some things are not meant to be shared.

Date/DateTime in stdlib are based on Java's Time class. If you add 20 years to your birthday, you get a new date, 20 years after your birthday.

This was not Java's original way of modelling Dates/Times.

If you used Java's old Calendar class, and added 20 years to your birthday, you did not get a new value; your birthday is now 20 years later.

Imperative Programming

As in Java, Python, and others

Not always great: may disturb structural recursion.

[Demo]

Functional Programming

- Expressions always have the same result, no matter how often you use them
- Data created with the same arguments are equal
- All data flow is explicit, through function arguments
- Sometimes cumbersome to share/change state

Imperative Programming

- Memory allows for additional inputs/outputs
- Expressions/functions do not just have results, they also have *effects*
- Need to document effects ➔ see next week
- Equality is more complicated (see == vs. Equals)
- How often and in what order you evaluate things matters

Functional Programming

ConsList-based Map (CLM)

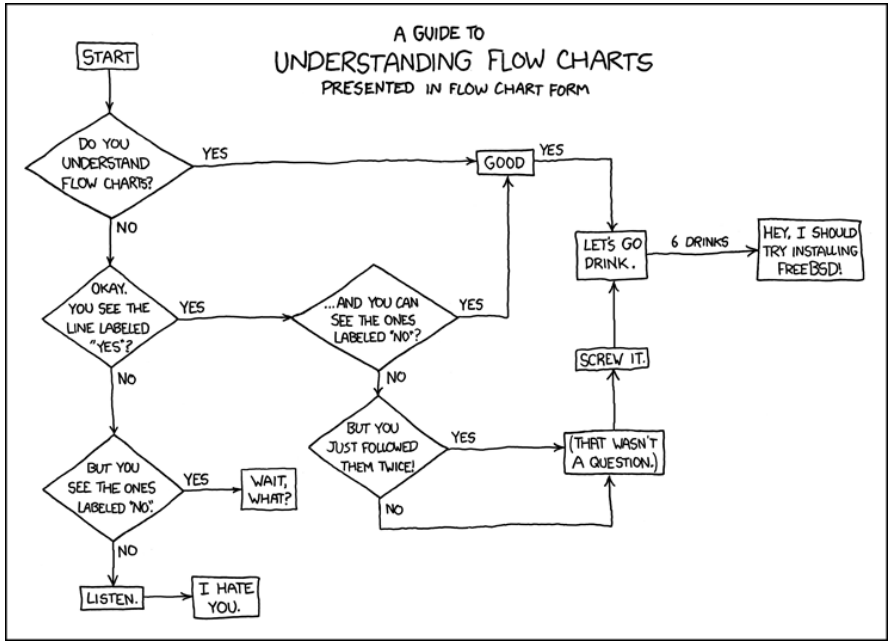
```
CLM MakeConsMap(Pair<K, V>…)
CLM Put(CLM, K, V)
CLM Remove(CLM, K)
Maybe<V> Get(CLM, K)
boolean ContainsKey(CLM, K)
ConsList<K> GetKeys(CLM)
```

Imperative Programming

HashMap (HM)

```
HM MakeHashMap(Pair<K, V>…)
void Put(HM, K, V)
void Remove(HM, K)
Maybe<V> Get(HM, K)
boolean ContainsKey(HM, K)
ConsList<K> GetKeys(HM)
```


Back to Trees



Randall Munroe, https://xkcd.com/518/

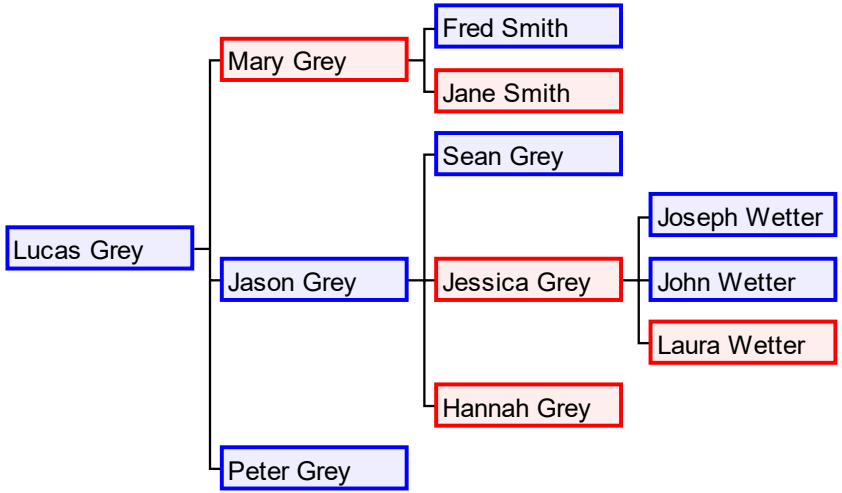
Practice

Fork and clone the comp1110-2025s1-workshops project.
Create a folder “ws4a”, and work in “DescendantsTree.java”.
Commit and push when you are done.

Following the design recipe, design a program able to model a particular person's descendants tree. A person's descendants include the children of that person (immediate descendants), the children of the children of that person (grandchildren) and so on. The program should keep track of the full name (i.e., names + surnames), gender, birth and death date (***if and only if the person died***) of every person in the descendant's tree (including the root of the tree).

You can assume: (1) all dates are in the same time zone, and time of day does not matter; (2) the birth date of the children of any parent in the tree has to be a biologically compatible future date compared to the birth date of the parent; (3) the difference among birth dates of any two siblings is biologically compatible; (4) the death date can never be an earlier date than the birth date; (5) there cannot be two descendants with the same full name.

Person's descendant tree example



Source: Wikipedia

Lucas Grey's descendant tree

Lucas has:

- 3 children (Mary, Jason, Peter)
- 5 grandchildren (Fred, Jane, Sean, Jessica, Hannah); and
- 3 great-grandchildren (Joseph, John, Laura)

Practice

Fork and clone the comp1110-2025s1-workshops project.
Create a folder “ws4a”, and work in “DescendantsTree.java”.
Commit and push when you are done.

Design the following functions:

1. Given the full name of a person (possible different from the person in the root of the tree), count how many descendants of that person are there in a person's descendant tree. Hint: develop first a function that counts how many descendants are there in a person's descendant tree.
2. Given a person's descendant tree, generate a list with the full names of (1) all descendants which are females; (2) descendants still alive; (3) descendants born after a given date; (4) descendants that have had 2 or more children born the same date. *Note 1*: you can develop a single higher-order function to code (1)-(4). *Note 2:* the order of elements in the output list is not important; the output of the function is correct as far all the requested full names are in the output list.
3. Given a person's descendant tree, return the name of the person that had their first children earliest in their life among all parents in the tree. If more than one person fulfills this criterion, return one of such people arbitrarily.
4. Given a person's descendant tree, returns a new tree where all males and their descendants are removed from the input tree. Generalize the function such that this operation (removal of some people and all their descendants) can be applied to any predicate passed as an argument to the function.