CRICOS PROVIDER 00123M

School of Computer Science

COMP SCI 1103/2103 Algorithm Design & Data Structure

Recursion

adelaide.edu.au

seek LIGHT

How do you solve this?

- Count the number of children + grand children + great grand children + … (successors) of one person

- Count the number of his/her children
- For each child also add the number of his/her successors to what you have got so far
  - How? Same way!

- You already know the concept of recursion!

Solving Problems

- Ultimately, what we want to do is solve problems.
- We must find a way of expressing the problem:
  - That is easy to understand
  - That allows us to produce an algorithm
  - That allows us to solve the problem

Example: Factorial

- Find n!
- Lets do it for some small n
  - 0!=1! =1 , 2!= 2 , 3!= 6
  - 4!= 4.(what we just had! i.e. 6)= 24
  - 5!= 5.(4!)
  - Generally n!=n.(n-1)!
- Recursive approach:
  - We know about a simple case (1!=1)
  - Write a function that returns n.(the result for n-1)
  - In order to find the result for n-1 it should call itself.
- Any other approach?
  - Of course! Iterative approach!

Self-defined problems

- Some problems are more easily defined in terms of themselves.
- Look at the factorial numbers: n!
  - Can think of this as n*(n-1)*(n-2)*..*2*1
  - OR you can think of this as n*((n-1)!)
- What's the problem with the second?

- How can we turn a mathematical relationship such as n! = n * ((n-1)!) into code?

Code in c++ for recursive factorial

```
int factorial(int n) {
        if (n < 0){
                cout << "ERROR!!!! Negative input\n";
                exit(1);
        }
        if (n == 0){
                return 1;
        }
        return n * factorial(n-1);
}
```

Controlled Recursion

- A recursive function is a function that invokes itself directly or indirectly

- Any controlled recursion is defined in terms of:
  - a recurrence relationship
  - a base case (or stopping condition)
- What happens in the case of uncontrolled recursion?

- Infinite recursion can occur if recursion does not reduce the problem in a manner that allows it to eventually converge to the base case.

Example

- You want to count everyone in the lecture theatre.
  - What's an iterative way to do it?
  - What's a recursive way to do it?

Recursive Counting

- In the lecture theatre, we can think of the people in here as a ‘chain’ of people.
- The first person, or ‘head’, can ask the next person for the count ending before him or her.
  - They ask the next person, until the request reaches the end of the chain.
  - The end says “1”
  - Everyone else adds 1 as they pass it forward
  - The head of the chain is the last one who receives an answer!

- Let’s write a pseudo-code for that!

Another Example

- Count people in one row
- Idea?
- Pseudo-Code
- Any problems with that?!

Common mistakes

- We know we need two things to make recursion work properly. What are they?

- A very common mistake is to set a base case that is never reached!
  - Don't use == when you need <=, >=.
  - You must approach the base case steadily and not 'skip' over it.

- Make sure the input value is moving towards the base case

Recursion and the Stack

- Recursion makes heavy use of the stack as you will be placing lots of activation records on there.
  - This is where the space usage comes from.
  - Stack overflow

- Some problems just make more sense recursively.
  - Easier to write
  - At the expense of heavy stack usage

Recursion versus Iteration

- Anything you can do recursively can be done iteratively.
  - Some languages don't even allow recursion.
  - Remember that everything, ultimately, gets turned into machine code
- Recursive functions are almost always slower and less efficient - but much easier to understand.

Example

- Truckloads problem from TopCoder. https://community.topcoder.com/stat?c=problem_statement&pm=6011

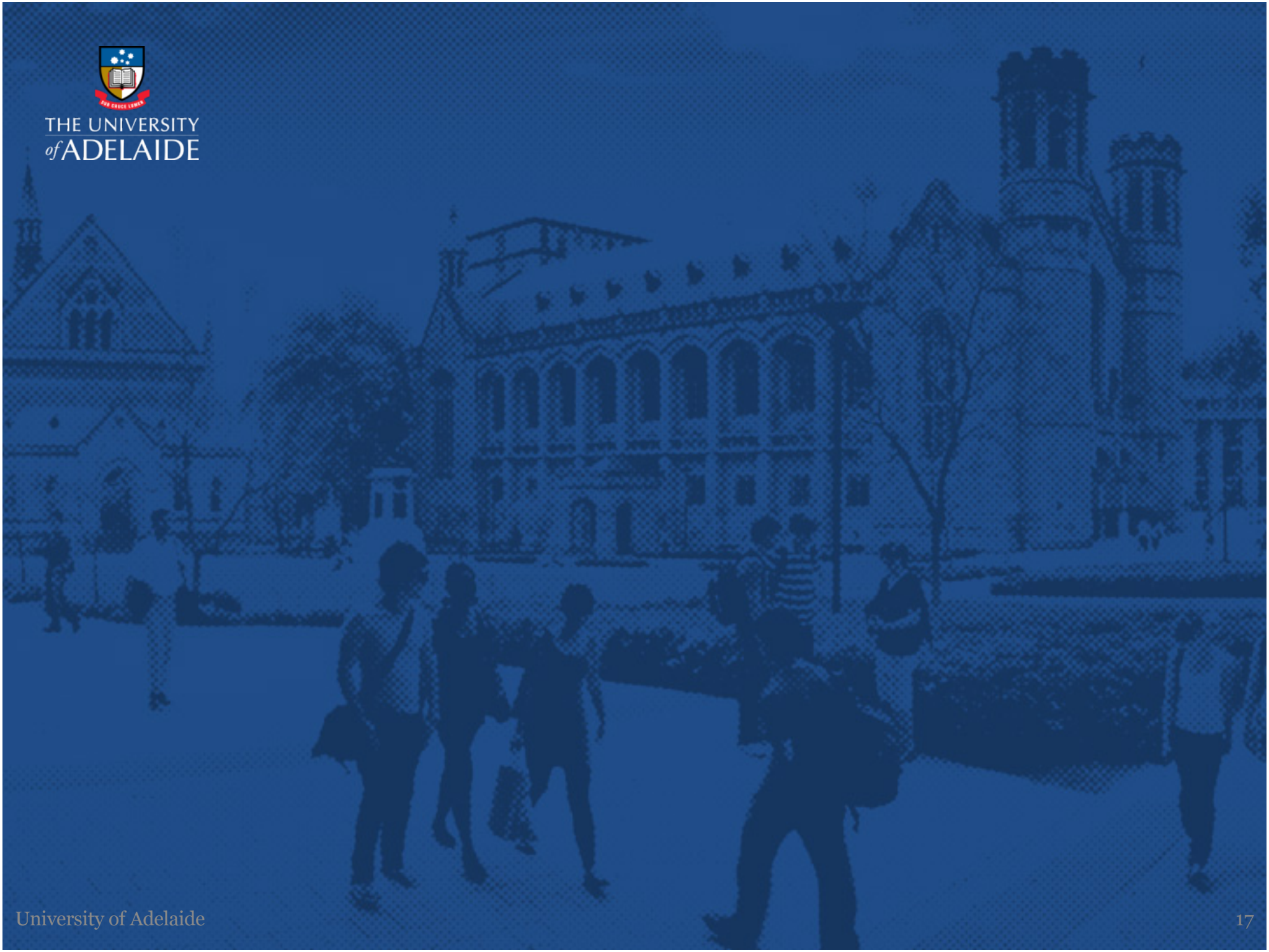
- We have a pile of crates at our warehouse that we want to load onto trucks. Our plan is to divide the pile in half forming two smaller piles, then continuing dividing each of the small piles in half until we get piles that will fit on a truck. (Of course, when we divide an odd number of crates in "half", one of the resulting piles will have one more crate than the other.) Our problem is to determine how many trucks we will need to ship the crates.
- Create a class Truckloads that contains a method numTrucks that is given **numCrates** (the number of crates at the warehouse) and **loadSize** (the maximum number of crates that will fit in a truck) and that returns the number of trucks required.

Truckloads problem

- Iterative approach
- Recursive approach

Summary

- Recursion is defining functions in terms of themselves.
- It can be easier to understand but does come at a cost.
- Some things are naturally recursive.
- We'll talk more about this next lecture.


THE UNIVERSITY
of ADELAIDE