

DEPARTMENT OF COMPUTER SCIENCE	≜UCL	
Recursion		
 A method can call itself - recursion! int factorial(int i) 		
ہ if (i > 0) return i * factorial(i-1) ; else		
return 1 ;		
} // See text book p199		
© 2006, Graham Roberts	2	

DEPARTMENT OF COMPUTER SCIENCE		≜UCL
Or		
<pre>public int factorial(int n) { if (n == 0) return 1; else return n * factorial(n-1); }</pre>	Testing for base case explicitly	
© 2006, Graham Roberts		3

DEPARTMENT OF COMPUTER SCIENCE	⁺UCL
Tracing recursive calls	
6! (6 * 5!) (6 * (5 * 4!)) (6 * (5 * (4 * 3!))) (6 * (5 * (4 * (3 * 2!)))) (6 * (5 * (4 * (3 * (2 * 1!))))) (6 * (5 * (4 * (3 * (2 * (1 * 1)))))) (6 * (5 * (4 * (3 * (2 * (1 * 1))))) (6 * (5 * (4 * (3 * 2)))) (6 * (5 * (4 * (3 * 2)))) (6 * (5 * (4 * (3 * 2)))) (6 * (5 * (4 * 6))) (6 * (5 * (4 * 0))) (6 * (5 * 24)) (6 * 120) 720	
© 2006, Graham Roberts	4

DEPARTMENT OF COMPUTER SCIENCE	≜UCL
A nice demonstration	
• Found this at:	
http://www.cs.princeton.edu/introcs/lectures/	
© 2008, Graham Roberts	5

DEPARTMENT OF COMPUTER SCIENCE

Recursion & iteration

UCL

6

- Recursion provides another way of doing iteration.
 Most use of recursion can be re-written using loops and vice-versa.
- Some algorithms are much easier to express using recursion.
- Remember all the Prolog...

© 2006, Graham Roberts

UCL Another example int sum(final int n) { if (n == 0) Sum the numbers from 1 to n. { return 0; } else { return sum(n-1) + n ; } } © 2006, Graham Roberts 7

DEPARTMENT OF COMPUTER SCIENCE	⁴UCL
Recursive calls made by sum	
call sum(5) sum(5): $n != 0$, call sum(4) sum(4): $n != 0$, call sum(3) sum(3): $n != 0$, call sum(2) sum(2): $n != 0$, call sum(1) sum(1): $n != 0$, call sum(0) sum(0): $n == 0$, return 0 sum(1): return 0 + 1 (1) sum(2): return 1 + 2 (3) sum(3): return 3 + 3 (6) sum(4): return 6 + 4 (10) sum(5): return 10 + 5 (15)	
© 2006, Graham Roberts	8

DEPARTMENT OF COMPUTER SCIENCE	⁺UCL
Linear Search	
int linearSearch(int[] a, int value, int position) { if (position >= a.length) { Base case - value not found }	
else { if (value == (a[position])) { return position; } }	
else { return linearSearch(a, value, position + 1); } }	cursive ase
Called with int n = linearSearch(array, 10, 0);	
© 2006, Graham Roberts	9

DEPARTMENT OF COMPUTER SCIENCE	⁴UCL
Remove else's	
<pre>int linearSearch(int[] a, int value, int position) { if (position >= a.length) { return -1; } if (value == (a[position])) { return position; } return linearSearch(a, value, position + 1); }</pre>	
Shorter but possibly less readable. Not as clear that there are 3 options?	
© 2006, Graham Roberts	10

DEPARTMENT OF COMPUTER SCIENCE		⁴UCL
Binary Search	Array must be sorted	
int binarySearch(int[] a, int low,	int high, int value) {	
if (low > high) { return -1;	B	ase case - ue not found
else {		
int middle = (low + high)	/2;	
if (value == a[middle]) { return middle; }	V	alue found
else {		
if (value < a[middle])		
{ return binarySearch else	(a, low, middle - 1, value); }	Recursive cases
{return binarySearch(a, midule + 1, nigh, value); }	
,		
© 2006, Graham Roberts		11

DEPARTMENT OF COMPUTER SCIENCE	≜UCL
Quicksort	
<pre>void quicksort(double[] a, int left, int right) { if (right <= left) return; int i = partition(a, left, right); quicksort(a, left, i-1); quicksort(a, i+1, right); }</pre>	
 Algorithm: Split array into 2 partitions such that all values in left are less than those in righ recursively apply to partitions. when partition is size one, it is sorted. 	nt.
© 2006, Graham Roberts	12

DEPARTMENT OF COMPUTER SCIENCE	≜U	CL
Partition	t left_int right) {	
int i = left - 1; int j = right;	·····, ·····g···, (
while(true) { while (less(a[++i], a[right])) ; while (less(a[right], a[j]))	Take two partitions and swap values	
if (j == left) break; if (i >= j) break; swap(a, i, j);	such that all values in left partition are less than all values in right.	
} swap(a, i, right); return i:		
}	Here is a demo	
© 2006, Graham Roberts		13

DEPARTMENT OF COMPUTER SCIENCE	⁺UCL
What does this do?	
void f(int a, int b) {	
if (a <= b) {	
int m = $(a + b) / 2$; System out print(m + " "):	
f(a, m-1);	
f(m+1, b);	
}	
}	
© 2006, Graham Roberts	14

DEPARTMENT OF COMPUTER SCIENCE

[≜]UCL

Cost of recursion

• Each recursive call has the 'cost' of a method call. – Recursion is potentially slower than iteration.

- But a recursive algorithm may be more efficient.
- Each recursive call uses up a bit of memory.
- Too many recursive calls will use up all the stack space and the program will fail.
 But not a reason to ignore recursion.

© 2006, Graham Roberts

© 2006, Graham Roberts

Recursion Pitfalls

≜UCL

15

16

- Infinite recursion.
 fail to test for base cases, so recursion never terminates.
 - Program crashes with out of memory error.
- · Mutual recursion.
 - Two method call each other recursively.
 - Directly or indirectly.Usually a sign of poor design and can be difficult to spot.

DEPARTMENT OF COMPUTER SCIENCE	⁴UCL
Questions?	
© 2006, Graham Roberts	17





























DEPARTMENT OF COMPUTER SCIEN	CE	≜UCL
Front in Earniner Basis Course of Action Allscourt	Matters description description	induine foort instantic Technet instanti damefen
Stadert indicates which for enrol Stadert indicates which for enrol Stadert inputs name and number	Control Contro	
3. System verifies student 4. System displays seminar list	2ndister Policies	recention of the second s
Bisteris picks seminar System determines eligibility in enroll System determines eligibility in enroll Oystem determines schedule M		attalation and attalation and attalation attalatio attalation attalation attalation atta
B. System calculates fees. System displays fees. System verifies aludent visites to entail Students indicates yes.		unitating processor bulketing
12. System enrols student in seminar		







DEPARTMENT OF COMPUTER SCIENCE	≜UCL
Summary	
 Recursion another form of repetition More UML diagram examples 	
© 2006, Graham Roberts	29