02393 Programming in C++
Module 9: Recursive Programming

Sebastian Mödersheim
Mordor slides courtesy Christian W. Probst

October 31, 2016

Lecture Plan

#	Date	Topic
1	29.8.	Introduction
2	5.9.	Basic C++
3	12.9.	Data Types, Pointers
4	19.9.	Data Types, Pointers / Libraries and Interfaces; Containers
5	26.9.	Libraries and Interfaces; Containers
6	3.10.	Classes and Objects I
7	10.10.	Classes and Objects II
		Efterårsferie
8	24.10.	Classes and Objects III
9	31.10.	Recursive Programming
10	7.11.	Lists
11	14.11.	Trees
12	21.11.	Novel C++ features
13	28.11.	Summary
	5.12.	Exam

Recursion

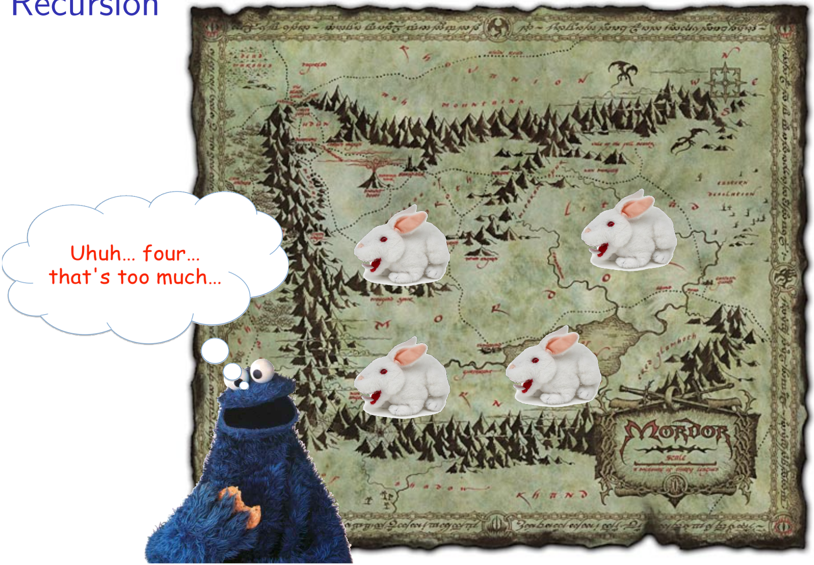
Recursion

Hm, that looks like a nice country... I think I would like to conquer it...
MORDOR

Recursion

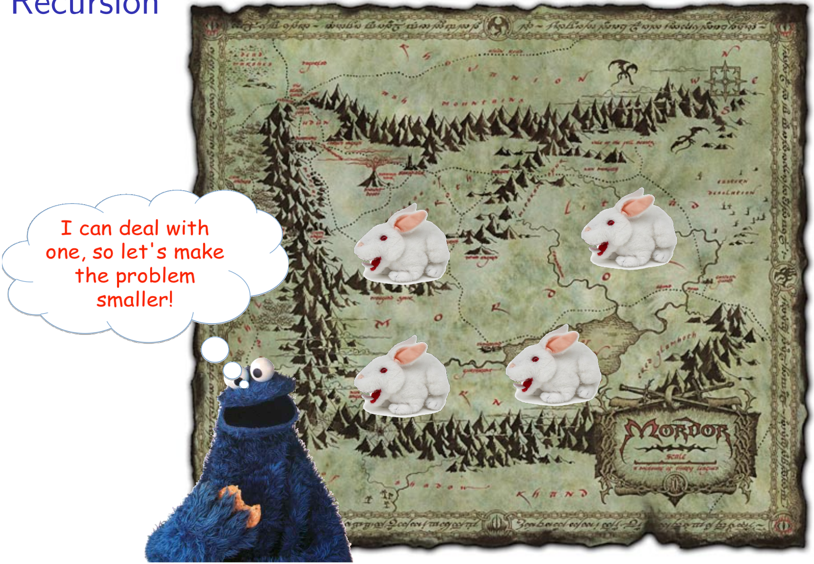
Recursion

Recursion



Recursion

Recursion



Recursion

Recursion

Recursion

Recursion

Recursion

Recursion

Recursion

Recursion

Recursion

Recursion

Recursion

Definition

Recursion (lat.): *see* Recursion
... or Google *recursion*.

What is Recursion?

- Solution technique that **solves large problems** by **reducing them to smaller problems of the same form**
- It is crucial that the smaller problem has the same form
- This means we can use the same technique for the big and the small problem!

Recursion

Definition

Recursion (lat.): *see* Recursion
... or Google *recursion*.

What is Recursion?

- Solution technique that **solves large problems** by **reducing them to smaller problems** **of the same form**
- It is crucial that the smaller problem has the same form
- This means we can use the same technique for the big and the small problem!

Why is Recursion... weird for some people?

- Some people are not used to inductive reasoning/abstraction...
- Other programming concepts are common in normal life:
  - repeat an action several times, on different objects (loops);
  - making decisions (if then else);
  - etc.

Recursion

When using recursion we must ensure:

- Every recursion step reduces to a smaller problem.
- There is a smallest problem (or a set of smallest problems) that can be handled directly, without recursion.
- Every chain of recursion steps eventually reaches one of these smallest problems.

Otherwise?

- Risk of non-termination!

Recursion

Recursive Leap of Faith

- When writing a recursive function, we believe that the recursive call computes the right solution, if the argument to the recursive call is smaller.
- Assuming that any recursive call works correctly is called the *Recursive Leap of Faith*.

Recusion

Rules of thumb

- Checking if you have a simple problem before decomposition.
- Solve the simple cases correctly!
- Check that decomposition makes the problem simpler!
- Ensure that decomposition eventually reaches one of the simple cases.
- The arguments to the recursive calls must be simpler versions of the original argument!
- When you take the recursive leap of faith, do the recursive calls provide with a correct solution to all simpler problems possible?

Examples

Mathematical definitions often use recusion:

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1)! & \text{otherwise} \end{cases}$$

Examples

Mathematical definitions often use recusion:

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1)! & \text{otherwise} \end{cases}$$

And can be easily transformed into a C++ program:

```
int fact(int n){
   if (n==0) return 1;
   else return n*fact(n-1);
}
```

Induction Proofs

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1)! & \text{otherwise} \end{cases}$$

Note the similarity of recursion with induction proofs.

Example Theorem and Inductive Proof

$n! > 0$ for all natural numbers n.
Proof:

Induction Proofs

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1)! & \text{otherwise} \end{cases}$$

Note the similarity of recursion with induction proofs.

Example Theorem and Inductive Proof

$n! > 0$ for all natural numbers n.
Proof: trivial.

Induction Proofs

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1)! & \text{otherwise} \end{cases}$$

Note the similarity of recursion with induction proofs.

Example Theorem and Inductive Proof

$n! > 0$ for all natural numbers n.
Proof:

- Induction Base: For $n = 0$: $0! = 1$ by definition.

Induction Proofs

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1)! & \text{otherwise} \end{cases}$$

Note the similarity of recursion with induction proofs.

Example Theorem and Inductive Proof

$n! > 0$ for all natural numbers n.
Proof:

- Induction Base: For $n = 0$: $0! = 1$ by definition.
- Induction Step:
  - ($\star$) Suppose for some number $n - 1$ we have proved that $(n-1)! > 0$.
  - We show: then also $n! > 0$.

Induction Proofs

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1)! & \text{otherwise} \end{cases}$$

Note the similarity of recursion with induction proofs.

Example Theorem and Inductive Proof

$n! > 0$ for all natural numbers n.
Proof:

- Induction Base: For $n = 0$: $0! = 1$ by definition.
- Induction Step:
  - ($\star$) Suppose for some number $n-1$ we have proved that $(n-1)! > 0$.
  - We show: then also $n! > 0$.
    - By definition $n! = n \cdot (n-1)!$.

Induction Proofs

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1)! & \text{otherwise} \end{cases}$$

Note the similarity of recursion with induction proofs.

Example Theorem and Inductive Proof

$n! > 0$ for all natural numbers n.
Proof:

- Induction Base: For $n = 0$: $0! = 1$ by definition.
- Induction Step:
  - ($\star$) Suppose for some number $n-1$ we have proved that $(n-1)! > 0$.
  - We show: then also $n! > 0$.
    - By definition $n! = n \cdot (n-1)!$.
    - We have that $n > 0$ (since $n - 1 \geq 0$) and $(n-1)! > 0$ (by $\star$). It then trivially follows that $n \cdot (n-1)! > 0$.

Induction Proofs

$$n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1)! & \text{otherwise} \end{cases}$$

Note the similarity of recursion with induction proofs.

Example Theorem and Inductive Proof

$n! > 0$ for all natural numbers n.
Proof:

- Induction Base: For $n = 0$: $0! = 1$ by definition.
- Induction Step:
  - ($\star$) Suppose for some number $n-1$ we have proved that $(n-1)! > 0$.
  - We show: then also $n! > 0$.
    - By definition $n! = n \cdot (n-1)!$.
    - We have that $n > 0$ (since $n - 1 \geq 0$) and $(n-1)! > 0$ (by $\star$). It then trivially follows that $n \cdot (n-1)! > 0$.
    - Thus $n! > 0$.

Induction proofs also work for recursive programs

```
int fact(int n){
   if (n==0) return 1;
   else return n*fact(n-1);
}
```

Example Theorem and Inductive Proof

$fact(n) > 0$ for all natural numbers n.

Induction proofs also work for recursive programs

```
int fact(int n){
    if (n==0) return 1;
    else return n*fact(n-1);
}
```

Example Theorem and Inductive Proof

$fact(n) > 0$ for all natural numbers n.

Proof:

Induction proofs also work for recursive programs

```
int fact(int n){
   if (n==0) return 1;
   else return n*fact(n-1);
}
```

Example Theorem and Inductive Proof

$fact(n) > 0$ for all natural numbers n.
Proof:

- Induction Base: For $n == 0$: $fact(0) == 1$ by the program.

Induction proofs also work for recursive programs

```
int fact(int n){
   if (n==0) return 1;
   else return n*fact(n-1);
}
```

Example Theorem and Inductive Proof

$fact(n) > 0$ for all natural numbers n.

Proof:

- Induction Base: For $n == 0$: $fact(0) == 1$ by the program.
- Induction Step:
  - ($\star$) Suppose for some number $n-1$ we have proved that $fact(n-1) > 0$.
  - We show: then also $fact(n) > 0$.

Induction proofs also work for recursive programs

```
int fact(int n){
    if (n==0) return 1;
    else return n*fact(n-1);
}
```

Example Theorem and Inductive Proof

$fact(n) > 0$ for all natural numbers n.
Proof:

- Induction Base: For $n == 0$: $fact(0) == 1$ by the program.
- Induction Step:
  - ($\star$) Suppose for some number $n-1$ we have proved that $fact(n-1) > 0$.
  - We show: then also $fact(n) > 0$.
    - By the program $fact(n) == n \cdot fact(n-1)$.

Induction proofs also work for recursive programs

```
int fact(int n){
    if (n==0) return 1;
    else return n*fact(n-1);
}
```

Example Theorem and Inductive Proof

$fact(n) > 0$ for all natural numbers n.

Proof:

- Induction Base: For $n == 0$: $fact(0) == 1$ by the program.
- Induction Step:
  - ($\star$) Suppose for some number $n-1$ we have proved that $fact(n-1) > 0$.
  - We show: then also $fact(n) > 0$.
    - By the program $fact(n) == n \cdot fact(n-1)$.
    - We have that $n > 0$ (since $n-1 \geq 0$) and $fact(n-1) > 0$ (by $\star$). It then trivially follows that $n \cdot fact(n-1) > 0$.

Induction proofs also work for recursive programs

```
int fact(int n){
    if (n==0) return 1;
    else return n*fact(n-1);
}
```

Example Theorem and Inductive Proof

$fact(n) > 0$ for all natural numbers n.

Proof:

- Induction Base: For $n == 0$: $fact(0) == 1$ by the program.
- Induction Step:
  - ($\star$) Suppose for some number $n-1$ we have proved that $fact(n-1) > 0$.
  - We show: then also $fact(n) > 0$.
    - By the program $fact(n) == n \cdot fact(n-1)$.
    - We have that $n > 0$ (since $n - 1 \geq 0$) and $fact(n-1) > 0$ (by $\star$). It then trivially follows that $n \cdot fact(n-1) > 0$.
    - Thus $n! > 0$.
    - Thus $fact(n) > 0$.

Examples/Live programming

- Toy examples: factorial, sum.
- Efficient search binary search
  - Naive search (linear search) of an element in a set takes $O(n)$.
  - Binary search is a divide-and-conquer $O(\log n)$ solution.
- Efficient exponentiation in cryptography ($a^n \ mod \ p$)
  - Naive exponentiation: $O(n)$
  - Efficient exponentiation: $O(\log n)$
  - Efficient solution is hard to program without recursion!
- Efficient sorting: merge sort
  - The recursion paradigm directly triggers an efficient solution!

Examples/Live programming

- Toy examples: factorial, sum.
- Efficient search binary search
  - Naive search (linear search) of an element in a set takes $O(n)$.
  - Binary search is a divide-and-conquer $O(\log n)$ solution.
- Efficient exponentiation in cryptography ($a^n \ mod \ p$)
  - Naive exponentiation: $O(n)$
  - Efficient exponentiation: $O(\log n)$
  - Efficient solution is hard to program without recursion!
- Efficient sorting: merge sort
  - The recursion paradigm directly triggers an efficient solution!
  - Naive bubble sort:

Examples/Live programming

- Toy examples: factorial, sum.
- Efficient search binary search
  - Naive search (linear search) of an element in a set takes $O(n)$.
  - Binary search is a divide-and-conquer $O(\log n)$ solution.
- Efficient exponentiation in cryptography ($a^n \ mod \ p$)
  - Naive exponentiation: $O(n)$
  - Efficient exponentiation: $O(\log n)$
  - Efficient solution is hard to program without recursion!
- Efficient sorting: merge sort
  - The recursion paradigm directly triggers an efficient solution!
  - Naive bubble sort: $O(n^2)$ for array of size n.
  - Merge sort:

Examples/Live programming

- Toy examples: factorial, sum.
- Efficient search binary search
  - Naive search (linear search) of an element in a set takes $O(n)$.
  - Binary search is a divide-and-conquer $O(\log n)$ solution.
- Efficient exponentiation in cryptography (a^n mod p)
  - Naive exponentiation: $O(n)$
  - Efficient exponentiation: $O(\log n)$
  - Efficient solution is hard to program without recursion!
- Efficient sorting: merge sort
  - The recursion paradigm directly triggers an efficient solution!
  - Naive bubble sort: $O(n^2)$ for array of size n.
  - Merge sort: $O(n \log n)$ (theoretical optimum).

Finding your way out of a Maze

Maze (Labyrinth)

- Standard example for recursion
- Classic: Daedalus, Theseus, and the Minotaur
- Problem: We want to find the way out of the maze

Finding your way out of a Maze

Maze (Labyrinth)

- Standard example for recursion
- Classic: Daedalus, Theseus, and the Minotaur
- Problem: We want to find the way out of the maze

Difficult to solve with iteration (`while` or `for` loop):

- avoid going in cycles
- detecting if there is no solution

Finding your way out of a Maze

Maze (Labyrinth)

- Standard example for recursion
- Classic: Daedalus, Theseus, and the Minotaur
- Problem: We want to find the way out of the maze

Difficult to solve with iteration (`while` or `for` loop):

- avoid going in cycles
- detecting if there is no solution

Recursive Procedure `findExit`

Finding your way out of a Maze

Maze (Labyrinth)

- Standard example for recursion
- Classic: Daedalus, Theseus, and the Minotaur
- Problem: We want to find the way out of the maze

Difficult to solve with iteration (`while` or `for` loop):

- avoid going in cycles
- detecting if there is no solution

Recursive Procedure `findExit`

- `findExit` recursively calls itself for all neighboring fields that are not walls...

Finding your way out of a Maze

Maze (Labyrinth)

- Standard example for recursion
- Classic: Daedalus, Theseus, and the Minotaur
- Problem: We want to find the way out of the maze

Difficult to solve with iteration (`while` or `for` loop):

- avoid going in cycles
- detecting if there is no solution

Recursive Procedure `findExit`

- `findExit` recursively calls itself for all neighboring fields that are not walls...
- ... and that have not been visited before.

Finding your way out of a Maze

Maze (Labyrinth)

- Standard example for recursion
- Classic: Daedalus, Theseus, and the Minotaur
- Problem: We want to find the way out of the maze

Difficult to solve with iteration (`while` or `for` loop):

- avoid going in cycles
- detecting if there is no solution

Recursive Procedure `findExit`

- `findExit` recursively calls itself for all neighboring fields that are not walls...
- ... and that have not been visited before.
  - If there is a way out, then there is one without going in cycles.

Finding your way out of a Maze

Maze (Labyrinth)

- Standard example for recursion
- Classic: Daedalus, Theseus, and the Minotaur
- Problem: We want to find the way out of the maze

Difficult to solve with iteration (`while` or `for` loop):

- avoid going in cycles
- detecting if there is no solution

Recursive Procedure `findExit`

- `findExit` recursively calls itself for all neighboring fields that are not walls...
- ... and that have not been visited before.
  - If there is a way out, then there is one without going in cycles.
- Backtracking: `findExit` fails if all recursive calls fail.

findExit (Pseudocode)

```
bool findExit(int x, int y){
  if (isExit(x,y)) return true;
  if (isWall(x,y)) return false;
  if (isMarked(x,y)) return false;
  mark(x,y);
  return
  (findExit(x-1,y) ||
   findExit(x,y-1) ||
   findExit(x+1,y) ||
   findExit(x,y+1))
}
```

given suitable implementations of `isExit`, `isWall`, `isMarked`, and `mark`.