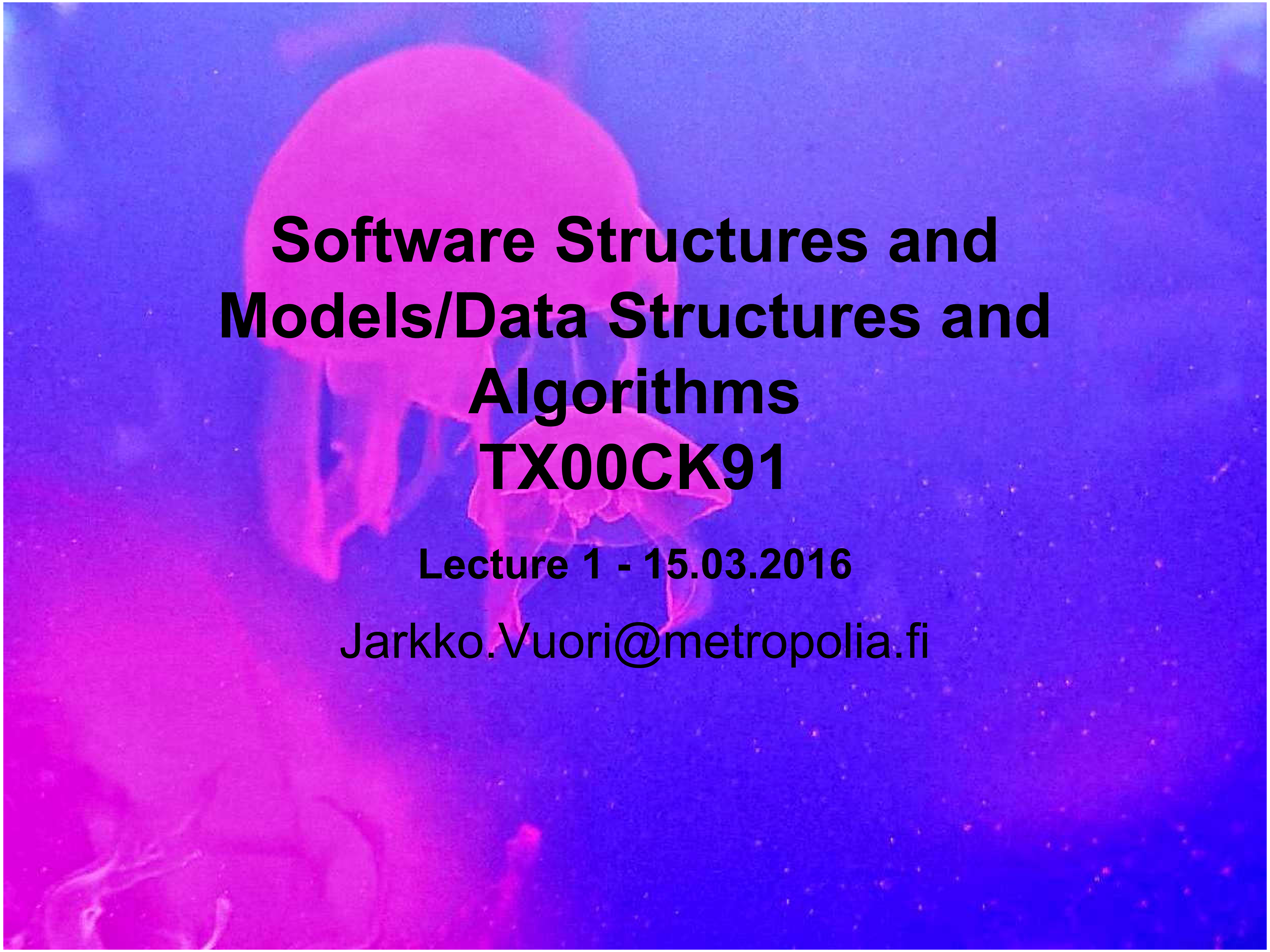
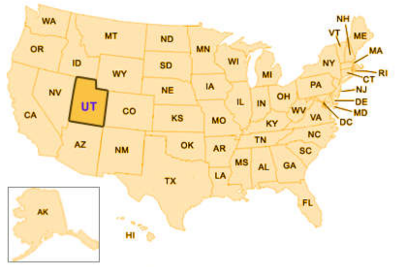


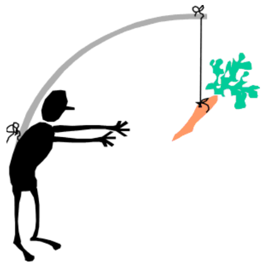
Software Structures and Models/Data Structures and Algorithms TX00CK91

Lecture 1 - 15.03.2016

Jarkko.Vuori@metropolia.fi



Motivation



- What every computer science major should know (http://matt.might.net/articles/what-cs-majors-should-know/, Matt Might is an assistant Professor in University of Utah):
  - Students should certainly see the common (or rare yet unreasonably effective) data structures and algorithms
  - But, more important than knowing a specific algorithm or data structure (which is usually easy enough to look up), computer scientists must understand how to design algorithms (e.g., greedy, dynamic strategies) and how to span the gap between an algorithm in the ideal and the nitty-gritty of its implementation
  - Specific recommendations
    - At a minimum, computer scientists seeking stable long-run employment should know all of the following:

– hash tables	(will be discussed in advanced algorithm course)
– linked lists	(to be discussed in this course)
– trees	(to be discussed in this course)
– binary search trees, and	(to be discussed in this course)
– directed and undirected graphs	(will be discussed in advanced algorithm course)

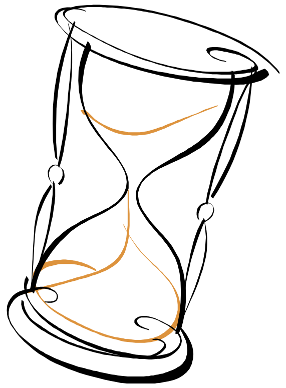
    - Computer scientists should be ready to implement or extend an algorithm that operates on these data structures, including the ability to search for an element, to add an element and to remove an element

General information

- This lecture series is part of the Software Structures and Models course
- The most important issue of the lecture series is to learn to understand the significance of abstraction, genericity and sw-components, and to learn to use them effectively in programming
  - So that you don't need to reinvent the wheel all the times
- In this course we learn how to use and build re-usabe software components
  - We study the problematics of common, reusable software components
  - We build those components using Java-language and the class model
  - The idea is to see how it is possible (using Java-language) to implement common re-usable software components
- We will study widely used common software components: lists, stacks and trees
  - This is the data structures part in the course name
- Considering algorithms we study recursion in-depth
  - We will learn how to use recursion in problem definition, in problem solving, and in program implementation
  - We will also check out how the microprocessor is executing recursive subroutine
- We will also do timing measurement of some algorithm implementation
  - In order to be able to understand the performance differencies of various kind of data structure implementations
- In addition, we will also familiarize ourselves with Java generics, iterators etc.

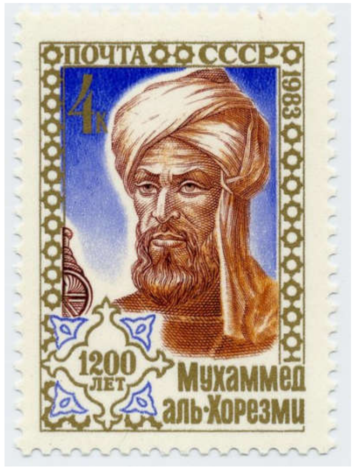
Timetable (preliminary)

Lecture	Topic
1	Intro, algorithm, abstraction as a tool to manage complicated things
2	Class as a software component, Java generics, Containers, List
3	Stack
4	Stack, Enumeration type, Queue
5	Queue, static memory, dynamic memory
6	Dynamically linked lists
7	Stack with dynamically linked lists
8	Linked lists (ordered, circular, double linked), Binary search tree
9	Binary search tree, recursion

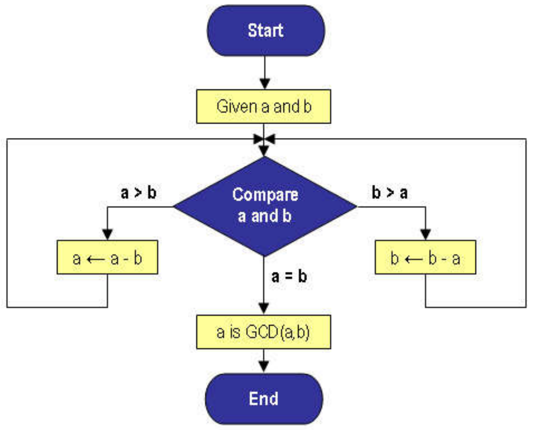


Algorithm

- In mathematics and computing, an algorithm (the word algorithm comes from the name of the 9th century Persian Muslim mathematician al-Khwarizmi) is a procedure (a finite set of well-defined instructions) for accomplishing some task which, given an initial state, will terminate in a defined end-state
  - Recipe is an one kind of algorithm
    - although many algorithms are much more complex; algorithms often have steps that repeat (iterate) or require decisions (such as logic or comparison)
  - Sorting an number list is an another kind of algorithm
  - Algorithm has an initial state and a recognizable end state (as opposed to heuristic operations)
    - E.g. initial state: ingredients for the recipe, end state: meal
  - Most algorithms can be directly implemented by computer programs (using programming languages, algorithms are implemented as functions or procedures); although in a very limited form
    - Errors in the implementation and the limitations of computers (e.g. the amount of memory) could block the correct execution of the algorithm
  - Correct execution of algorithm does not guarantee the solving of the problem if there are errors in the algorithm, or the algorithms does not suit to the given problem solving
    - Potato salad recipe does not give the correct results if the are no potatoes, even in the case all the operations are executed as if we have potatoes
- Algorithms are needed in problem solving
- Different algorithms can solve the given problem using different amount of resources (time, space, energy, etc.)
  - For example, two different potato salad recipes can differ whether we chop up potatoes before or after cooking
- Some countries (e.g. USA) even allow algorithms to be patented when embodied in software or in hardware
  - Multiplication algorithm can be implemented in microprocessor's arithmetic/logic unit (ALU)

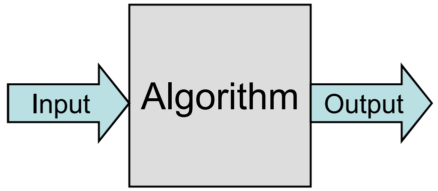


Abu Abdullah Muhammad bin Musa al-Khwarizmi



Flowcharts are often used to graphically represent algorithms
(Shown here is the Euclidean algorithm (also called Euclid's algorithm) which is an algorithm to determine the greatest common divisor (GCD) of two integers)

Algorithm

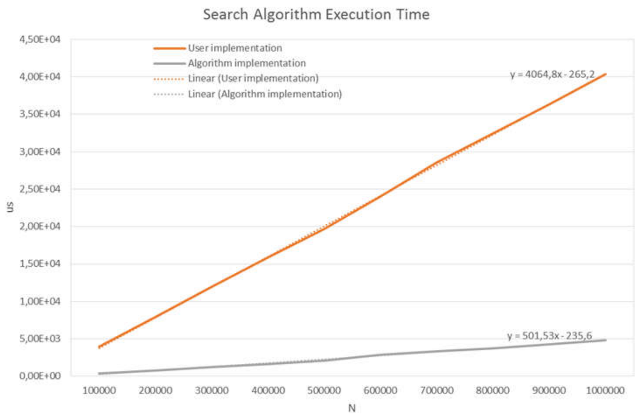


- Computer program is an algorithm which tells what operations and in what sequence is needed to accomplish a given task
  - Tasks could be e.g. the calculation of workers payslip, or printing an extract from a student register
- Often when an algorithm determines the operations to be done to the data, the input data is read from the input device and the results are printed to the output device or stored to be used later
  - Stored data is one part of the internal state of the algorithm executor
- Algorithm must be strictly defined: operation must be defined in all possible cases
  - All the possible cases must be systematically handled case by case (remember the missing potatoes case in potato salad recipe)
- There are also other possibilities (than using a computer) to implement algorithms
  - Biological neural network (human brains performing calculation task, or an insect hunting food)
  - Electrical circuit (PID control algorithm implemented with operation amplifiers)
  - Mechanical device (mechanical ignition advancing control unit)
- Research on algorithms implemented by a computer is called computer science
  - Often it is practiced abstractly (without any specific programming language and implementation)
  - It resembles with mathematics, because analysis is targeted on the general principles “behind” the algorithms

Algorithm

- One of the simplest algorithm is the searching of the largest number from the given (unordered) list
  - In the solution, every element of the list must be traversed, but only once. From this principle we can state the basic structure of the algorithm:
    - Check every element of the list, if it is larger than any element we have encountered, take it into account
    - The last element taken into account, is the largest element of the list when every element has been traversed
  - On the right side is the ”pseudocode” description of the algorithm
- When implementing the algorithm, it is usually interesting to know how much resources (e.g. time and memory) the algorithm consumes
  - Analysis of the algorithm gives the answer to this question
    - The given algorithm needs O(n) time units (where n is the length of the list)
      - Time needed is therefore directly related to the length of the list where the largest number must be found
    - Algorithm needs O(1) units of memory, because only the largest value must be stored
      - Memory needed is therefore fixed, it is unrelated to the length of the list

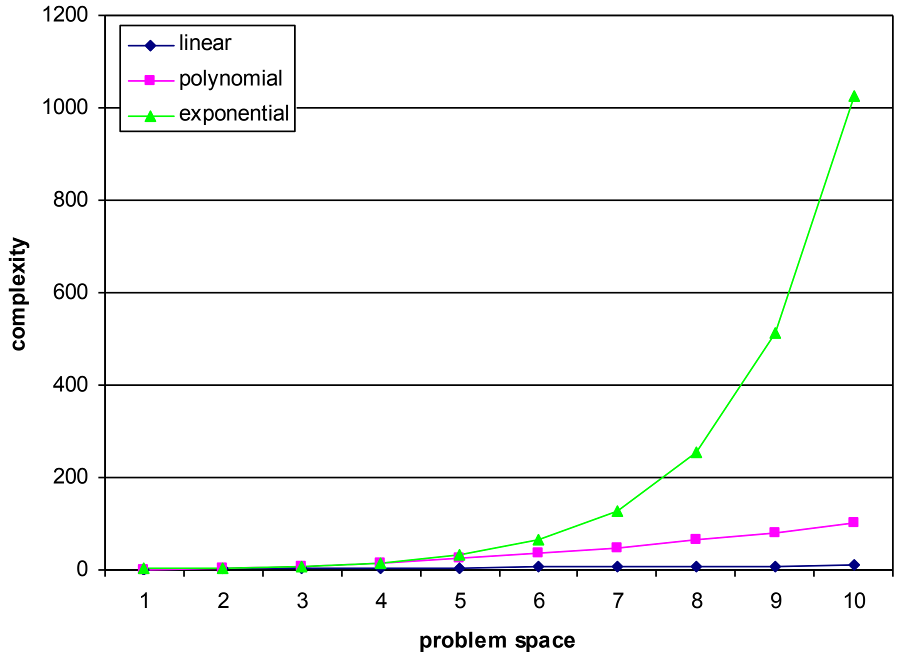
```
Algorithm LargestNumber
  Input: A non-empty list of numbers L.
  Output: The largest number in the list L.

  largest ← -∞
   for each item in the list L, do
    if the item > largest, then
     largest ← the item
  return largest
```



Big-O notation, O(n), is a mathematical notation used to describe the asymptotic behavior of functions. If for example, memory consumption is T(n)=4n^2-2n+3, and if the term n is large, term n^2 dominates. Then the memory consumption is O(n^2)

Algorithms and computability

- Resources (e.g. time or memory) needed by an algorithm is related to the scale (which is described by the number *n*) of the problem
  - It can be
    - linear O(*n*)
    - polynomial O(n^2)
    - exponential O(2^n)
- Polynomial and linear scale problems can be solved by computer
  - Exponential class problems are not solvable because the amount of resources is exploded when *n* goes large
    - Traveling salesman problem, knapsack-problem, optimal routing problem, etc.
    - This means that there exists problems which are not solvable with computer at all
      - But we can provide approximate solutions with limited time
        - » For example, finding a route with GPSR takes finite time, but the route obtained is not necessarily the best possible route between two points (it is good enough)



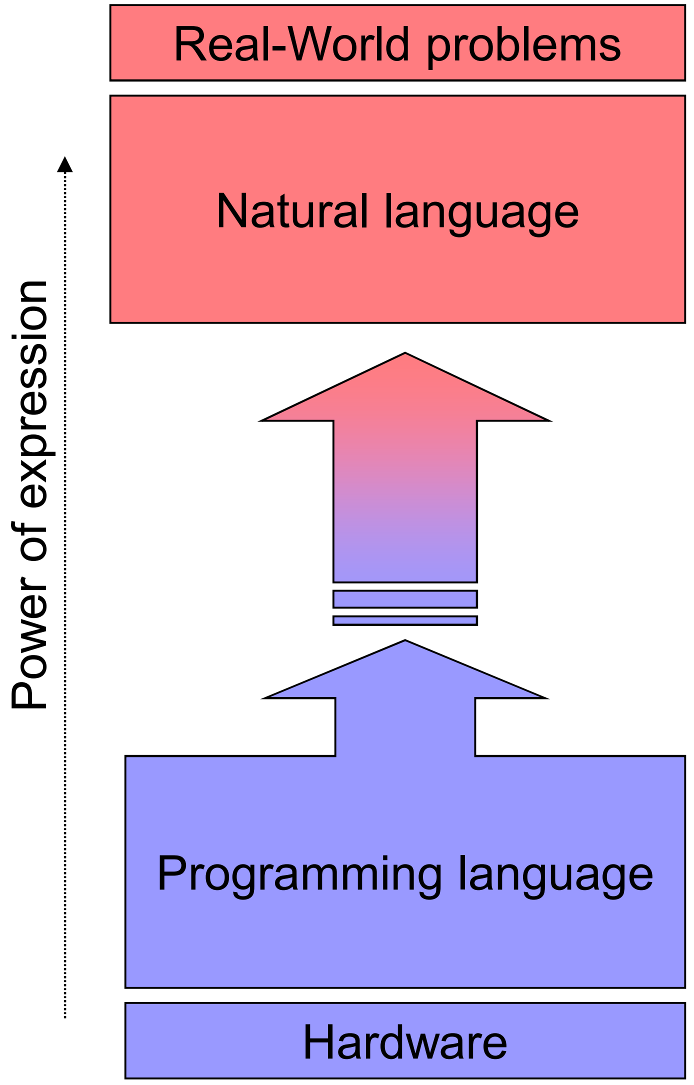
Algorithms and computability

Kurt Gödel (1906-1978)

- Whatever system which is based on logical reasoning cannot solve every problem existing in the world
- Gödel proved in 1931 that logical reasoning of whatever power cannot govern all the mathematical truth
  - this means that ”complete” logical system is impossible to device
  - Continuous hypothesis, Russel’s paradox
    - Suppose there is a town with just one male barber; and that every man in the town keeps himself clean-shaven: some by shaving themselves, some by attending the barber
      - It seems reasonable to imagine that the barber obeys the following rule: He shaves all and only those men who do not shave themselves
      - Under this scenario, we can ask the following question: Does the barber shave himself?
  - All consistent, axiomatic systems of number theory contains propositions which cannot be proven truth
  - On the other hand, the powerfulness of the system can be improved, and in this way we can solve more complicated problems (which cannot be solved before)
    - But still we have improvable problems remaining (but of course they are more complicated)
      - For example, continuous hypothesis has not been solved (H. Woodin proposed in 2000 that the whole question is wrong, it depends on the set theory used)

Algorithm

- How the program which resolves the given problem is designed?
  1. We design an algorithm which solves the problem
     - In this case we should use language that suits to human being
       - So that human being can develop the right algorithm considering the problem solving
     - This algorithm is independent of any programming language
  2. Implement the algorithm with a programming language
     - Algorithm is transformed to the computer's programming language
- In what way we can narrow the gap between phases 1 and 2? = how we can use language close enough to the human languages, but still the description obtained from phase 1 could be used in real programming language (phase 2)
  - By using abstraction



Abstraction

Abstract art describes only
high-level concepts,
not real objects

- In computer science, abstraction is the mechanism and practice to reduce the amount of small details in order to concentrate to a few central concepts at once
  - It is called pelkistäminen (in Finnish)
  - It is analogous to the abstraction used in mathematics
- For example both in computer science and in mathematics, the numbers are (programming language) concepts. Implementation on the other hand depends on the given hardware; this is not a problem because using the number in calculations is based on the mathematical concept of the number
- Human beings use abstraction in his thinking, it is a method to conquer complicated things
  - Man does not keep every detail of complicated things in his mind, but he uses abstraction instead
    - Abstraction is a simplification of the facts
    - In payroll systems we store the name and wage of the employee, but not irrelevant things such than the color of hairs or height or weight – this means that the employee is an abstraction of the real worker
  - Complicated things can have multiple abstraction levels, at the upper level the whole complexity is replaced with one concept (e.g. car)
    - When we are driving with the car, we use a different abstraction model than when we are finding a fault in the car
- In general we can say, that abstraction can exists with both the operations and data
  - Abstraction of the operation is dividing an operation to multiple suboperations (using subroutines)
  - Abstraction of data is dividing the data to the groups (giving a meaning to the databis, e.g. datatype)
  - Object oriented programming tries to abstract both the operations and the data together
- In programming work it is beneficial to use abstraction
  - Complicated things can be divided to subthings and they can be further subdivided …

Black-and-White picture is an abstraction of the color (real) picture/view

Abstraction of operations

- Phases of program development
  1. Divide the problem into smaller pieces (hierarcially)
  2. Group those pieces to functions (but do not solve them yet)
  3. Solve the rules how these subtasks (functions) have to be performed (in what order, on what condition, how many times)
     - Sequential structure (sequential function calls)
     - Conditional structure (if)
     - Repetition structure (for, while, do)
  4. Design and insert data to the program (variables)
  5. Find out the input/output information of the functions (parameters)
  6. Upper level of the program is now ready, then go down one layer (implement functions one by one)
- Procedural approach, operations are the main point
  - Data is on the secondary role

Abstraction example 1: an orienteering program

- The task is to implement an orienteering program: calculate direction and the distance when given the coordinates of the current and target locations
- First divide the problem to pieces, and abstract them as a functions (read_point, distance, print_point)
- Then solve how these functions should be performed (sequentially)
- Finally insert data (variables) and parameters to the program

```
// Orienteering program
public static void main(String[] args) {
    float x1, y1, x2, y2;
    float dist;

    x1 = read_point_x(); y1 = read_point_y();
    x2 = read_point_x(); y2 = read_point_y();
    dist = distance(x1, y1, x2, y2);
    System.out.println("\nDistance to the point ");
    print_point(x2, y2);
    System.out.println(" is ");
}
```

Function calls

Parameters. With parameters same function can be used in different places. E.g. square root can be calculated for different values, sqrt(3), sqrt(7), …

Abstraction of operation

- Operations are targeted on data
- Conquering large data sets is as complicated as controlling elaborate operations
- If we just abstract the operations, and not the data, there is a discrepancy in program structure (look programs on the right side)
  - Control of the program is easy to follow, data is a mess
  - Nobody thinks points as four numbers, but as two points
    - We need data abstraction also

```
// Orienteering program
public static void main(String[] args) {
    float x1, y1, x2, y2;
    float dist;

    x1 = read_point_x(); y1 = read_point_y();
    x2 = read_point_x(); y2 = read_point_y();
    dist = distance(x1, y1, x2, y2);
    System.out.println("\nDistance to the point ");
    print_point(x2, y2);
    System.out.println(" is " + dist);
}
```

Data not abstracted.
We can see **how** things are done, but not **what** are done or **why**

Abstraction of data

- Data abstraction should be also used
  - Man handles the combination of two points with an abstraction ”point” or ”coordinate”
- Data abstraction tools in object oriented programming languages is a class
- Man’s natural abstraction model can be used as it in programming
  - Programs are easy to understand and simpler
  - Man develops the solution to the problem, computer follows these directions (the program)
    - Language used must follow the thinking model of human beings and still be understandable to the computer
- Classes group data (and functions) together, functions group statements together

```
// Class definition
class Point {
    private double x, y;
}
```

Classes (in Java language) make separate variables to appear as a group (an object)

```
// Creating an object using
// a class
Point point1 = new Point(),
      point2 = new Point();
```

Class definition enables us to use it just as a simple datatype (that can be used similarly than standard types (e.g. char, int, …))

An orienteering program with operation and data abstraction

- This version is a program, which uses point as a data abstraction
  - Now the program is clear and easy to read
    - Both the operations and data

```
class Point {
    .
    .
    .
    private double x, y;
}

public static void main(String[] args) {
    Point p1 = new Point(), p2 = new Point();
    double dist;

    System.out.print("Enter point 1 ? "); p1.read();
    System.out.print("Enter point 2 ? "); p2.read();

    dist = p1.distance(p2);
    System.out.println("\nDistance to the point ");
    p2.print();
    System.out.println(" is " + dist);
}
```

Abstraction

- When abstraction is applied to **both** operations and data, the result is a clear program where we can see
  - How things are done
  - Why they are done

```
// Orienteering program
public static void main(String[] args) {
    float x1, y1, x2, y2;
    float dist;

    x1 = read_point_x(); y1 = read_point_y();
    x2 = read_point_x(); y2 = read_point_y();
    dist = distance(x1, y1, x2, y2);
    System.out.println("\nDistance to the point ");
    print_point(x2, y2);
    System.out.println(" is " + dist);
}
```

Data not abstracted.
We can see **how** things are done, but not **what** are done or **why**

```
public static void main(String[] args) {
    Point p1 = new Point(), p2 = new Point();
    double dist;

    System.out.print("Enter point 1 ? "); p1.read();
    System.out.print("Enter point 2 ? "); p2.read();

    dist = p1.distance(p2);
    System.out.println("\nDistance to the point ");
    p2.print();
    System.out.println(" is " + dist);
}
```

Operations abstracted, data abstracted

Final implementation

```
import java.util.Scanner;

public class Point {
    private double x, y;

    public Point() {
        x = 0; y = 0;
    }

    public void read() {
        Scanner in = new Scanner(System.in);
        x = in.nextDouble();
        y = in.nextDouble();
    }

    public void print() {
        System.out.println("(" + x + "," + y + ")");
    }

    public double distance(Point p) {
        return Math.sqrt(Math.pow(x-p.x, 2) + Math.pow(y-p.y, 2));
    }
}
```

```
public class OrienteeringApp {
    public static void main(String[] args) {
        Point p1 = new Point(), p2 = new Point();
        double dist;

        System.out.print("Enter point 1 ? "); p1.read();
        System.out.print("Enter point 2 ? "); p2.read();

        dist = p1.distance(p2);
        System.out.println("\nDistance to the point ");
        p2.print();
        System.out.println(" is " + dist);
    }
}
```

Multilevel abstraction of the data

- Hierarchy model of data can also be multilevel
- Now we have to design a program which reads the points of triangle, checks if it is an equilateral triangle, and then calculates the area and prints it
- First think the abstraction of operations, functions in the main are targeted to the triangles
  - not, for example, to the points

```
public static void main(String[] args) {
  double x1, y1, x2, y2, x3, y3, triangle_area;

  x1 = read_triangle_x();
  y1 = read_triangle_y();
  z1 = read_triangle_z();
  x2 = read_triangle_x();
  y2 = read_triangle_y();
  z2 = read_triangle_z();

  if (sides_equal(x1, y1, x2, y2, x3, y3)) {
    triangle_area = area(x1, y1, x2, y2, x3, y3);
    System.out.println("\n Area is " + triangle_area);
  } else
    System.out.println("\n Sides are not equal ");
}
```

Operation abstraction ok,
it is easy to follow the program logic

Point handling cluttered,
and it does not equals man’s abstraction (to make things simple)
(Man does not think three points as six numbers)

Multilevel abstraction of the data

- Here the data abstraction is raised to the point level (from plain numbers)
- Large data sets can be handled using grouping and classifying
  - We can reach the single information using hierarchical path when we know to what larger content the information belongs, and to what larger content that content belongs, etc.
- We raise the abstraction level to the triangle
  - Because we are using triangles in the program

```
public static void main(String[] args) {
  Point P1 = new Point(),
        P2 = new Point(),
        P3 = new Point();
  double triangle_area;

  P1.read(); P2.read(); P3.read();

  if (sides_equal(P1, P2, P3)) {
    triangle_area = area(P1, P2, P3);
    System.out.println("\n Area is " + triangle_area);
  } else
    System.out.println("\n Sides are not equal ");
}
```

Multilevel abstraction of the data

- We define datatype Ttriange
  - Every function that handles triangles will use triangle data type
  - Data abstraction raised to the triangle level

```
class Triangle {
    .
    .
    .
  private Point P1;
  private Point P2;
  private Point P3;
}
```

```
public static void main(String[] args) {
  Triangle triangle;
  float    triangle_area;

  triangle.read();

  if (triangle.sides_equal()){
    triangle_area = triangle.area();
    System.out.println("\n Area is " + triangle_area);
  } else
    System.out.println("\n Sides are not equal ");

}
```

Benefits of abstraction

1. It means same than *information hiding*, *programming in large*, *black box thinking*
2. Makes it possible to build large programs
3. Makes it possible to split the task to smaller parts
4. Makes it possible to divide the programming work
    - In this way multiple programmers can work on the same problem simultaneously
5. Enables the verifying of the programs
6. Eases testing of programs
7. Maintenance of programs is made easier
8. Reusing of program modules is more efficient (building programs of components)