Analogy between software process models and learning programming fundamentals

Katerina Zdravkova

Faculty of Computer Science and Engineering

University “Ss Cyril and Methodius”, Skopje, Macedonia

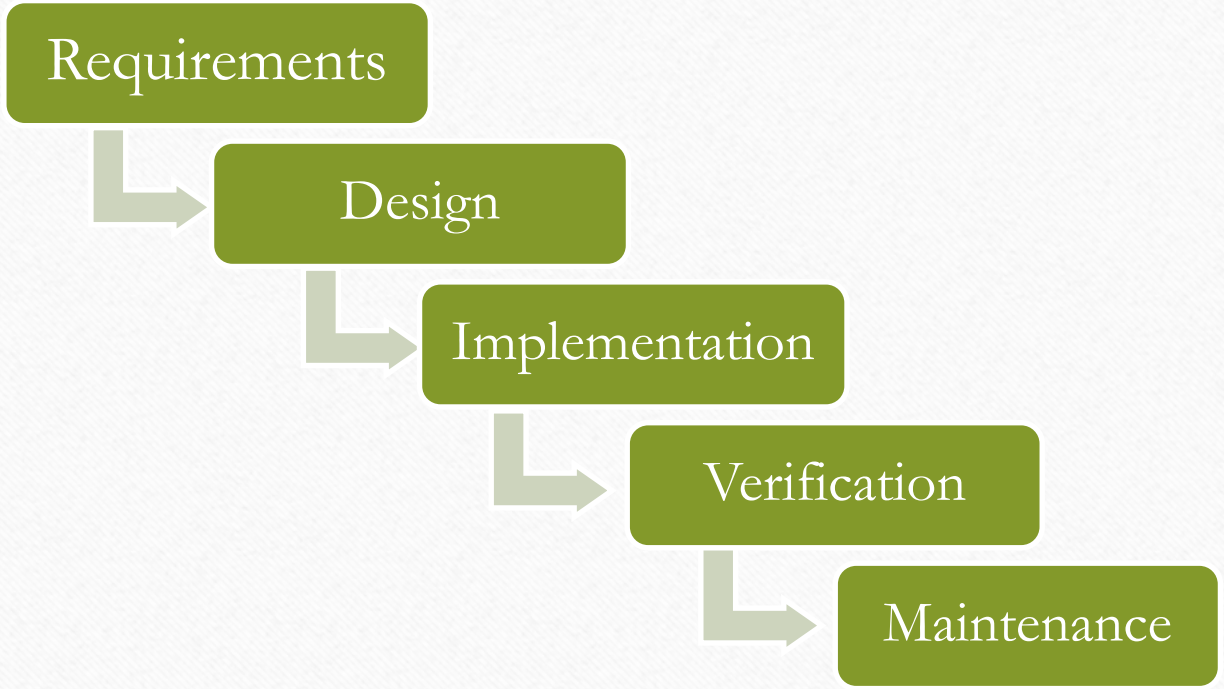
Overview of the presentation

- Typical programming fundamentals
- Typical organization of the assignments
- Software process models
- Analogy of both waterfall models
- Analogy of both incremental models
- Extension of the analogies for prototyping, spiral model and agile software development
- Possibility to practically evaluate the suggested approaches
- Conclusion

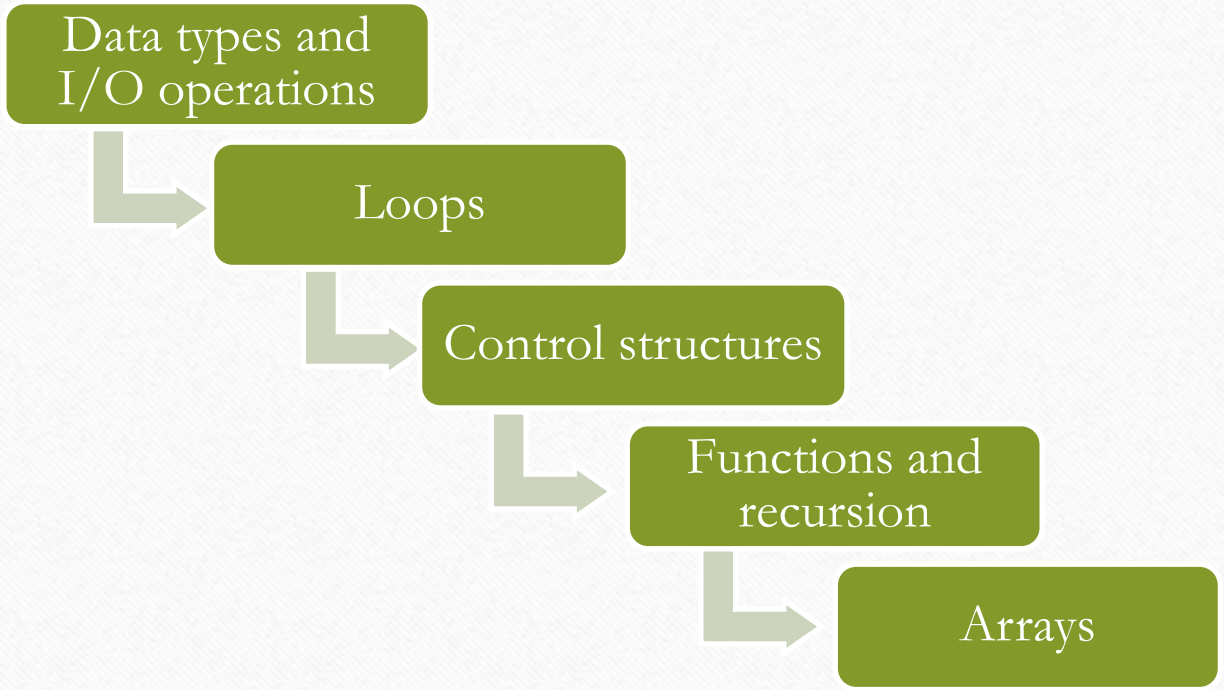
Typical programming fundamentals

- Basic scalar data types, operators, flow control:
  - Primitive data types: variables, constants, assignments, operators and expressions
  - Input / output operations
  - Mathematical functions, characters and strings
  - Loops
  - Control structures
  - Functions and recursion
  - Arrays

Waterfall model in software development



Waterfall model for teaching programming fundamentals



What can students solve after phase 1: Data types and I/O operations

- Hello world ☺
- Variables, declarations, formatted I/O, arithmetic operators, incrementing / decrementing
- Tasks with mathematical formulas, swapping of values
- Conversions:
  - Degree / radian
  - Centimeter / inch
  - Uppercase / lowercase conversions
- Well formatted I/O
- More advanced: Result of a logical operation

What can students make after phase 2: Loops

- Determination of sum, average
- Nicely formatted tables with even / odd members
- Printing permutations of numbers / characters
- Calculation of:
  - Exponents
  - Functions with equidistant arguments
  - Factorials, Fibonacci numbers, binomial series
- Estimation of a value of a function using Taylor series

What can students make after phase 3: Control structures

- Determination of:
  - Positive / negative values
  - Even / odd integers
  - Vowels / consonants
- Determination of minimum / maximum
- Sort algorithms

What can students make after phase 4: Functions and recursion

- Creation of functions for the most important tasks solved in the previous phases
- Factorials, Binomial coefficients, Fibonacci numbers calculated using recursion

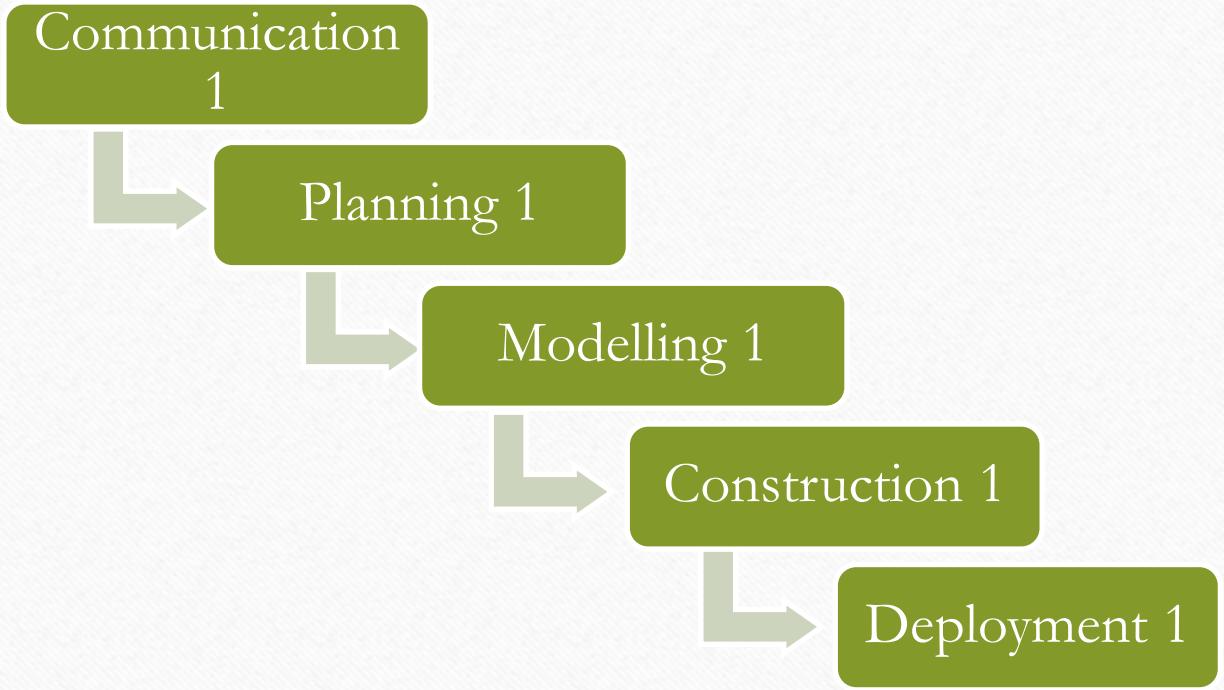
What can students make after phase 5: Arrays

- One-dimensional arrays:
  - Scalar and vector multiplication
  - Palindromes
  - Creation of various sort algorithms
- Two-dimensional arrays:
  - Matrices (addition, multiplication)
  - Calculation of determinants
  - Inverting matrices

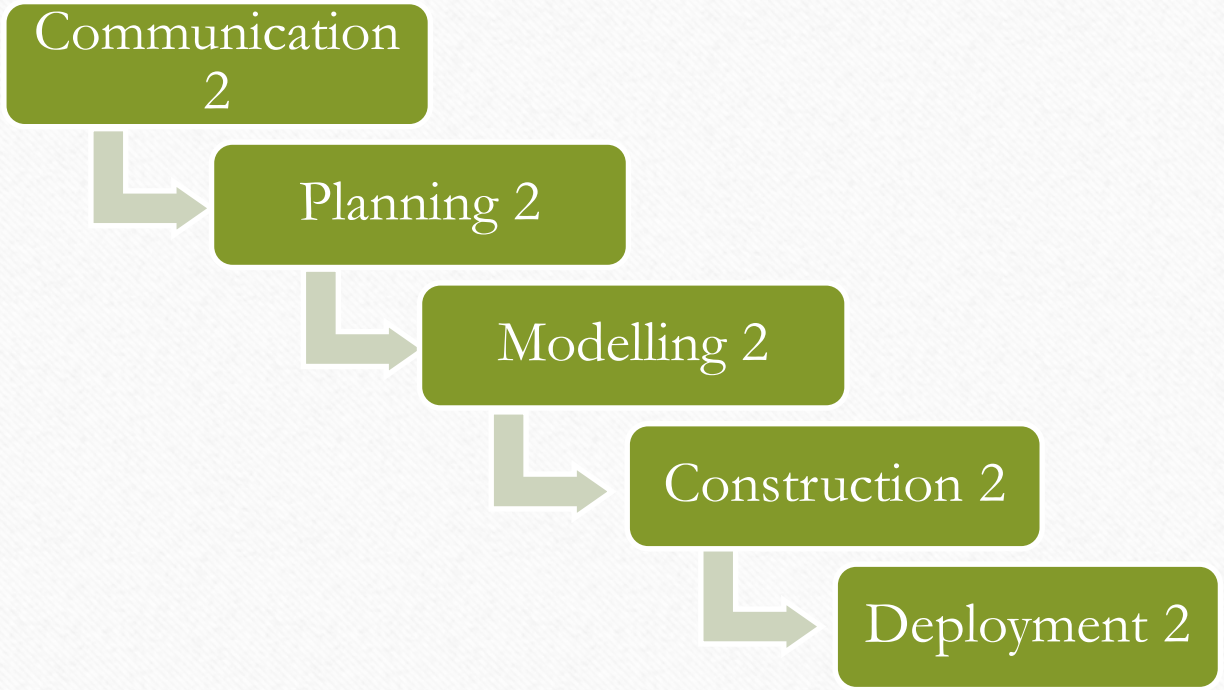
The analogy of both waterfall models

- Both models are sequential
- The order is very strict
- Changes in the previous stages are (almost) impossible
- Running models are visible too late
- Problems arising in the first phases rapidly accumulate
- But, they are still massively used, particularly by those who are used to them

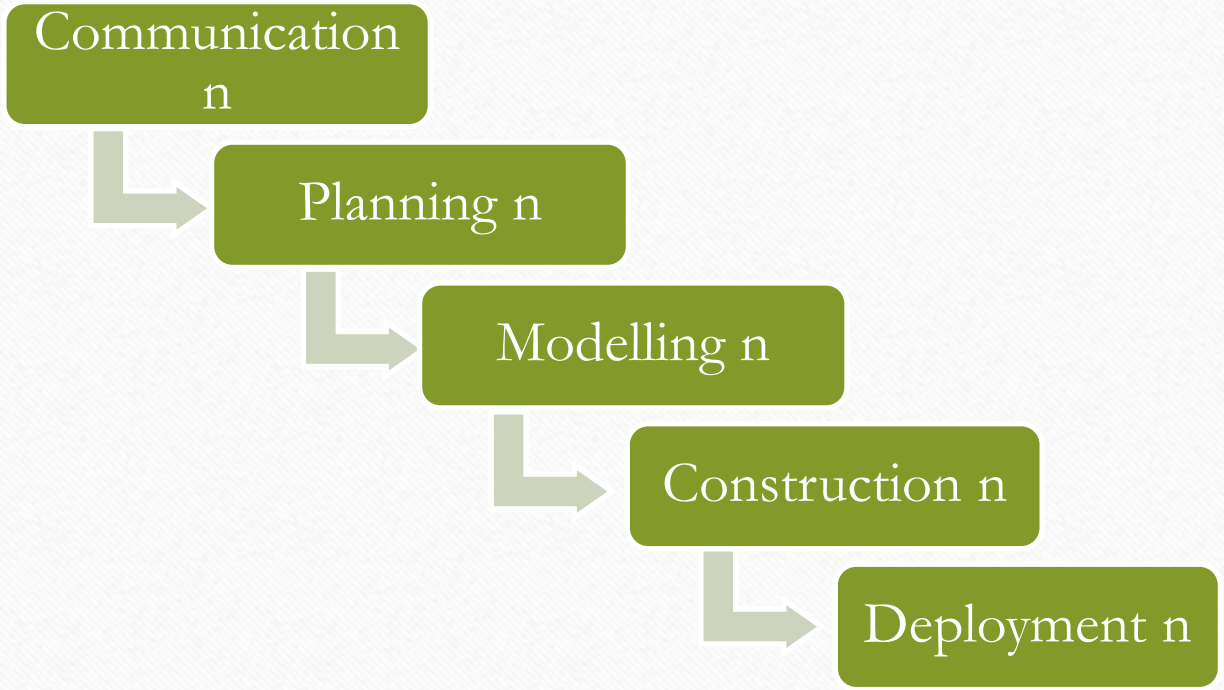
Incremental model in software development (increment 1)



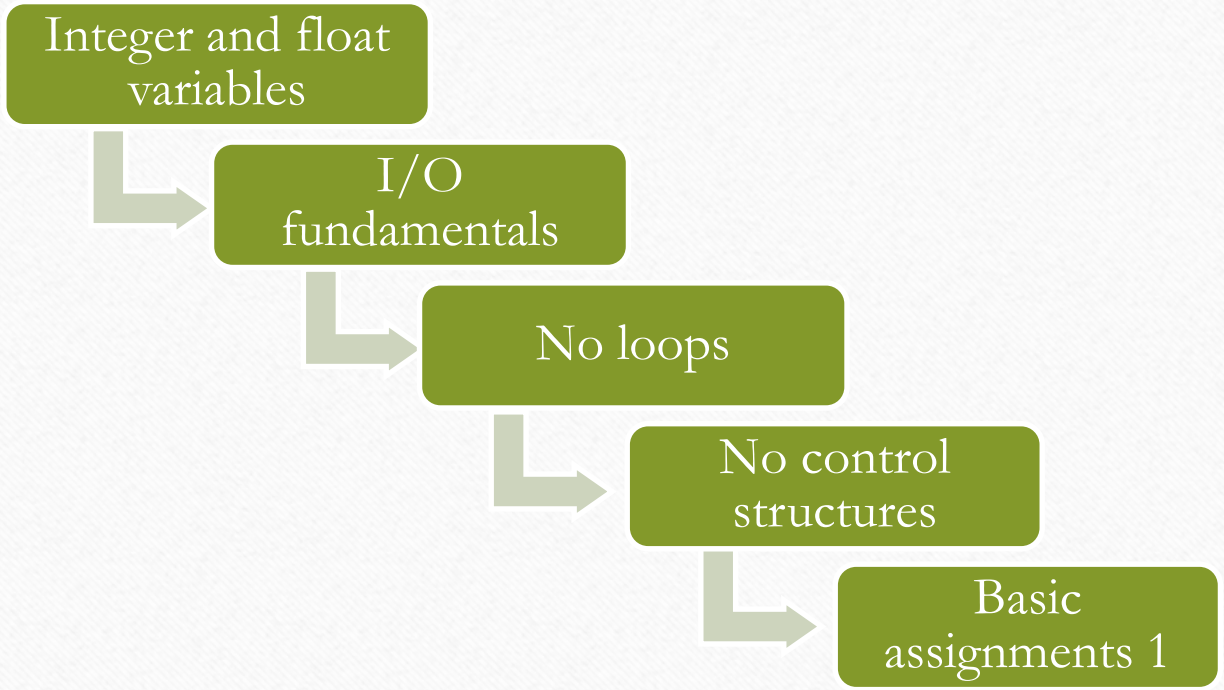
Incremental model in software development (increment 2)



Incremental model in software development (increment n)



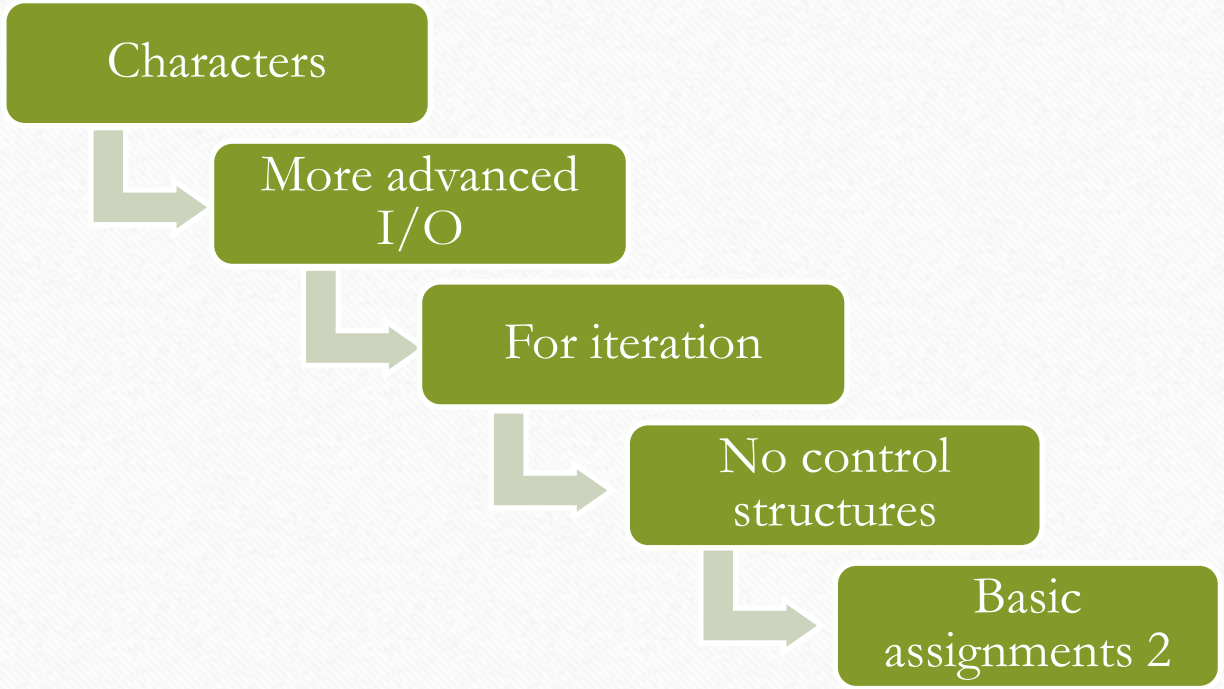
Incremental model for teaching programming fundamentals (increment 1)



Basic assignments after first increment

- Hello world ☺
- Variables, declarations, arithmetic operators, incrementing / decrementing
- Tasks with mathematical formulas, swapping of values
- Conversions:
  - Degree / radian
  - Centimeter / inch
  - Uppercase / lowercase conversions
- FIRST EVALUATION

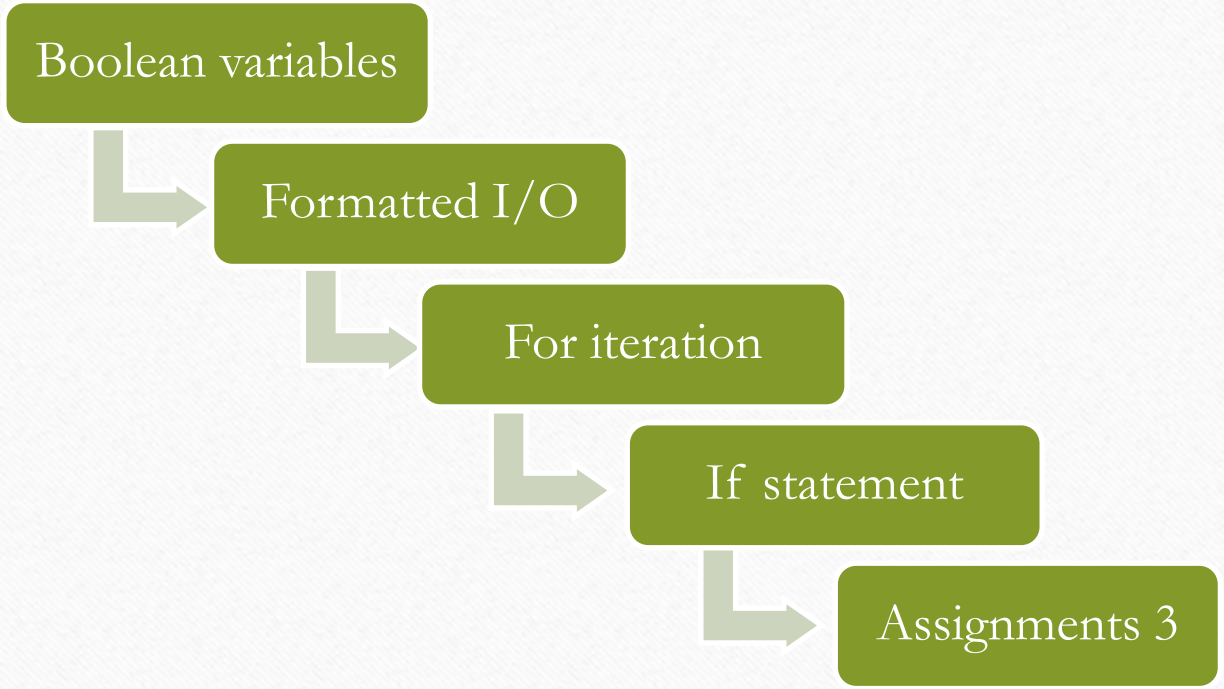
Incremental model for teaching programming fundamentals (increment 2)



Basic assignments after second increment

- Determination of sum, average
- Well formatted tables with even / odd members
- Printing permutations of numbers / characters
- Calculation of:
  - Exponents
  - Functions with equidistant arguments
  - Factorials, Fibonacci numbers, binomial series
- Estimation of a value of a function using Taylor series
- SECOND EVALUATION

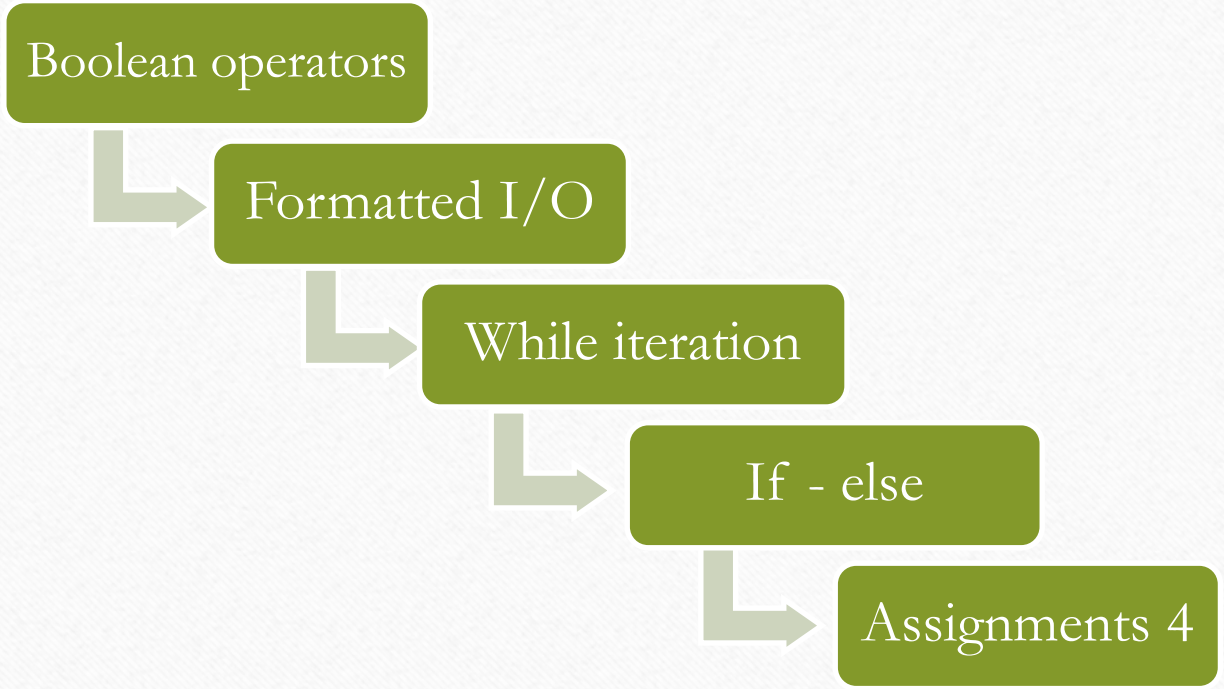
Incremental model for teaching programming fundamentals (increment 3)



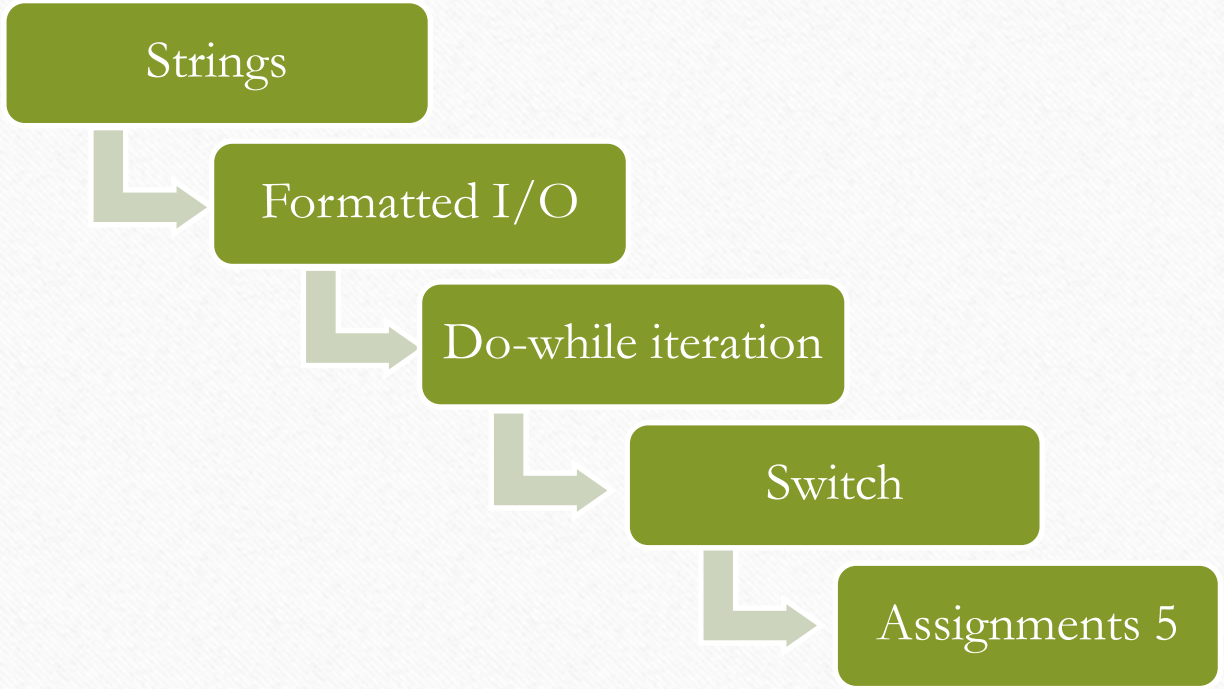
Basic assignments after third increment

- Well formatted I/O
- Result of a logical operation
- Determination of:
  - Positive / negative values
  - Even / odd integers
  - Vowels / consonants
- Determination of minimum / maximum
- THIRD EVALUATION

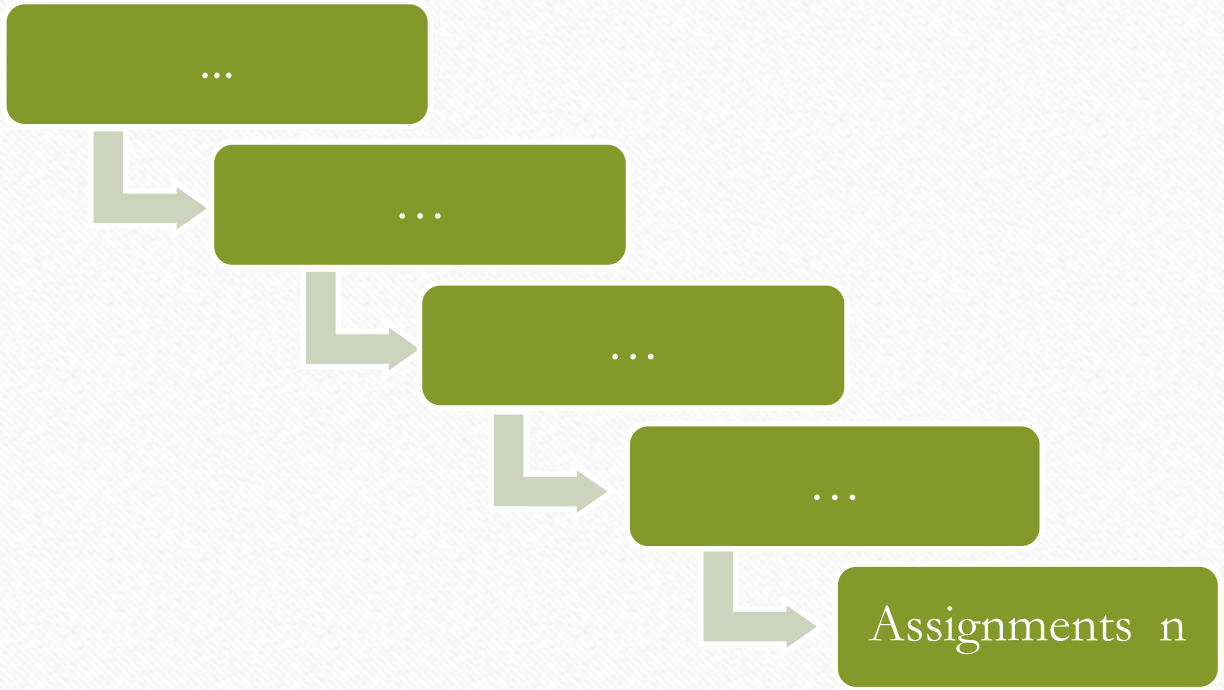
Incremental model for teaching programming fundamentals (increment 4)



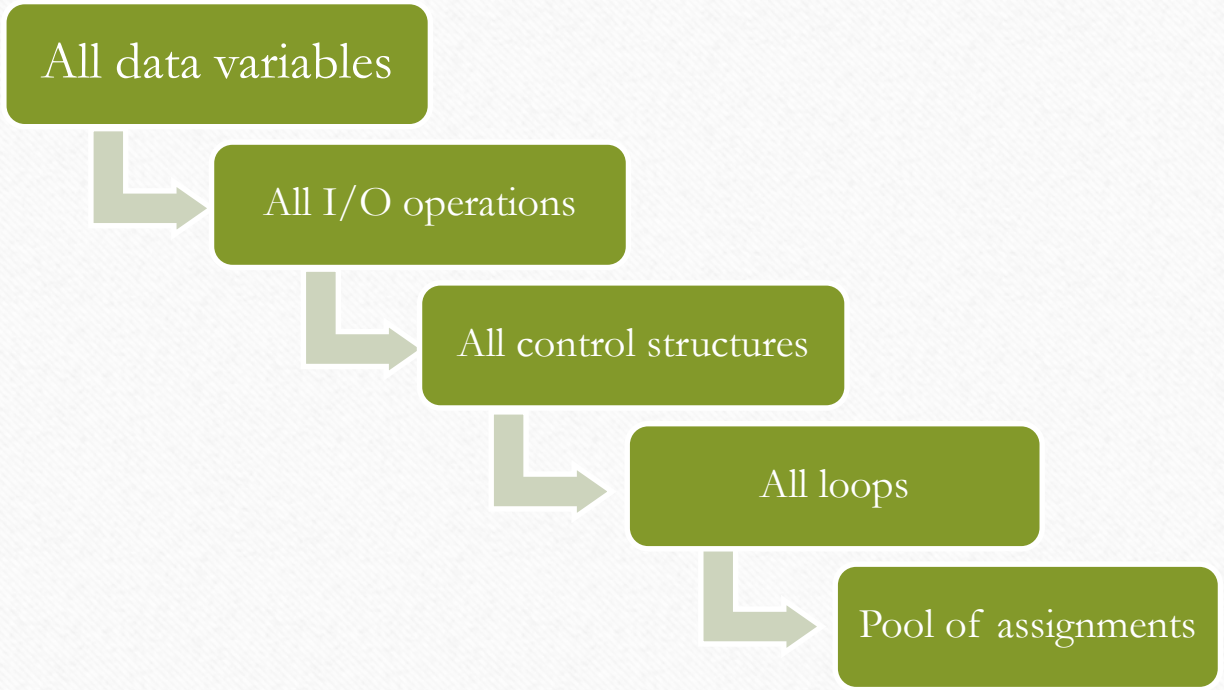
Incremental model for teaching programming fundamentals (increment 5)



Incremental model for teaching programming fundamentals (increment n)



Incremental model for teaching programming fundamentals (final increment)



The analogy of both incremental models (1)

- Each iteration leads to a cheap functional model (software / learning)
- After each iteration (regression) testing / knowledge evaluation can be conducted
- Testing / knowledge evaluation identifies the deficiencies immediately
- Debugging / extra lessons are provided very soon, because very small changes have been made compared with the previous iteration

The analogy of both incremental models (2)

- Management is more complex
- Quick success of the initial iterations can lead to more ambitious final product / learning outcome
- Latter can be a problem to those students which can't stick to the pace

Prototyping

- Initial activity of this model is the creation of a prototype, i.e. an incomplete runnable version of the software, which can be thrown away
- The most convenient types of prototyping to make the analogy with the learning process are:
  - Incremental prototyping
  - Extreme prototyping

Incremental prototyping

- Training is identical to incremental learning of programming fundamentals
- The crucial difference are the tasks students can solve
- Tasks should be directed to visible, running models
- They should not stimulate algorithmic approach that leads to mathematical problems
- On a contrary, the tasks should be very practical
- Incremental prototyping is appropriate for extra-curricular programming courses

Extreme prototyping

- Extreme prototyping is used for Web development
- Therefore, the approach can deal with teaching the initial Web development language (Ruby, Python, PHP)
- Tasks should initially stimulate the design, with no functionalities at all
- Second increment will deal with the navigation
- Each new increment will add new functionalities, starting from trivial towards more complex
- Extreme prototyping is appropriate for extra-curricular design courses

Spiral model

- Risk-driven process model
- Slightly unsuitable for compulsory programming courses
- Extension of the incremental model, with an advanced starting point
- Spiral model is convenient for the preparation of prospective competitors, who have already shown great competence in understanding the basic algorithms
- Spiral model has a great potential for parallelism, so it can be implemented to prepare students for team competitions too

Agile model

- Implementation of the spiral model to prepare team competitors is very time consuming for the instructors, because they should carefully prepare the iterations, and distribute them among the team members
- If the spiral model can't be organized thoroughly and successfully in advance, then the collaborative effort of prospective competitors should be stimulated
- The cross-functional teams can be organized for every training task
- Students' self-initiative should also be stimulated
- In such case, the ultimate goal of the instructor is to observe the progress wisely and thoroughly
- This approach is analogous to agile software development

Possibility to practically evaluate the suggested approaches

- Many of prospective computer science students have no programming skills
- Some faculties organize preparatory programming courses for those prospective students who are interested to improve their skills before starting their studies
- Division of the students into two groups can enable the stimulation of incremental model and comparison of obtained results
- Prototyping can be implemented during extra-curricular activities in secondary schools
- Alternative models can be implemented with the competitors in secondary schools and in academia

Conclusion

- C was invented in 1969, Java in 1995
- The order for delivering the contents is analogous to waterfall software developing model
- Although old-fashioned, it is still very effective
- Current teachers gained their programming skills implementing the old-fashioned manner
- It is time to start experimenting with alternative models, isn't it?

Thank you for your attention

Have you got any questions?