# CSCI251/CSCI851 Autumn-2020 Advanced Programming (**S1**)

Subject Content Introduction

# Assumed background

- There is no explicit assumption as to prior programming experience.
  - BCompSc students should know Java;
  - MCompSc students may have no programming experience, it's likely going to be a fair bit of work in that case.
- I would have liked there to be a dependence on CSIT113: Problem Solving, for undergraduate students.
  - But there isn't at the moment.
  - This doesn't mean you won't be solving problems!

# Subject Description

- The subject develops a thorough understanding of programming features, which are implemented in the C++ programming language. It comprises of four main components, namely procedural-based, object-based, object-oriented and generic programming. The subject addresses topics including memory management issues and dynamic memory allocation; classes; STL sequential and associative containers; operator overloading; advanced features in object-oriented programming; C++ RTTI; templates and exception handling; the latest C++ features (e.g. C++11 and C++14 standards).

# So … what is this subject about?

- This subject looks beyond the object-based/object-oriented content of CSIT111 and CSIT121, and beyond the limitations of Java.
- We use C++ as a vehicle for exploring a range of programming: Procedural, object-based, object-oriented, and generic.
- We look at some differences between Java and C++, for example the memory management models.
  - Due to the undergraduate structure there will be references to Java.

# Subject Learning Outcomes …

- On successful completion of this subject, students will be able to, to varying degrees:
  - Design and implement solutions to problems with the C++ programming language.
  - Design and implement procedural-based programming to solve problems.
  - Design and implement objects providing encapsulation, inheritance and polymorphism.
  - Design solutions to problems through the use of generic programming.
  - Design object-oriented solutions to problems.
  - Incorporate advanced features in C++ to achieve efficient implementations.

# Looking at the SLOs …

- … the term *design and implement* occurs three times, and design a couple more.
- The programming in the subject title doesn't just refer to coding.
  - Inevitably there will be a fair bit of syntax covered in this subject, but …
  - … I hope the balance will move more towards understanding principles as the subject develops.

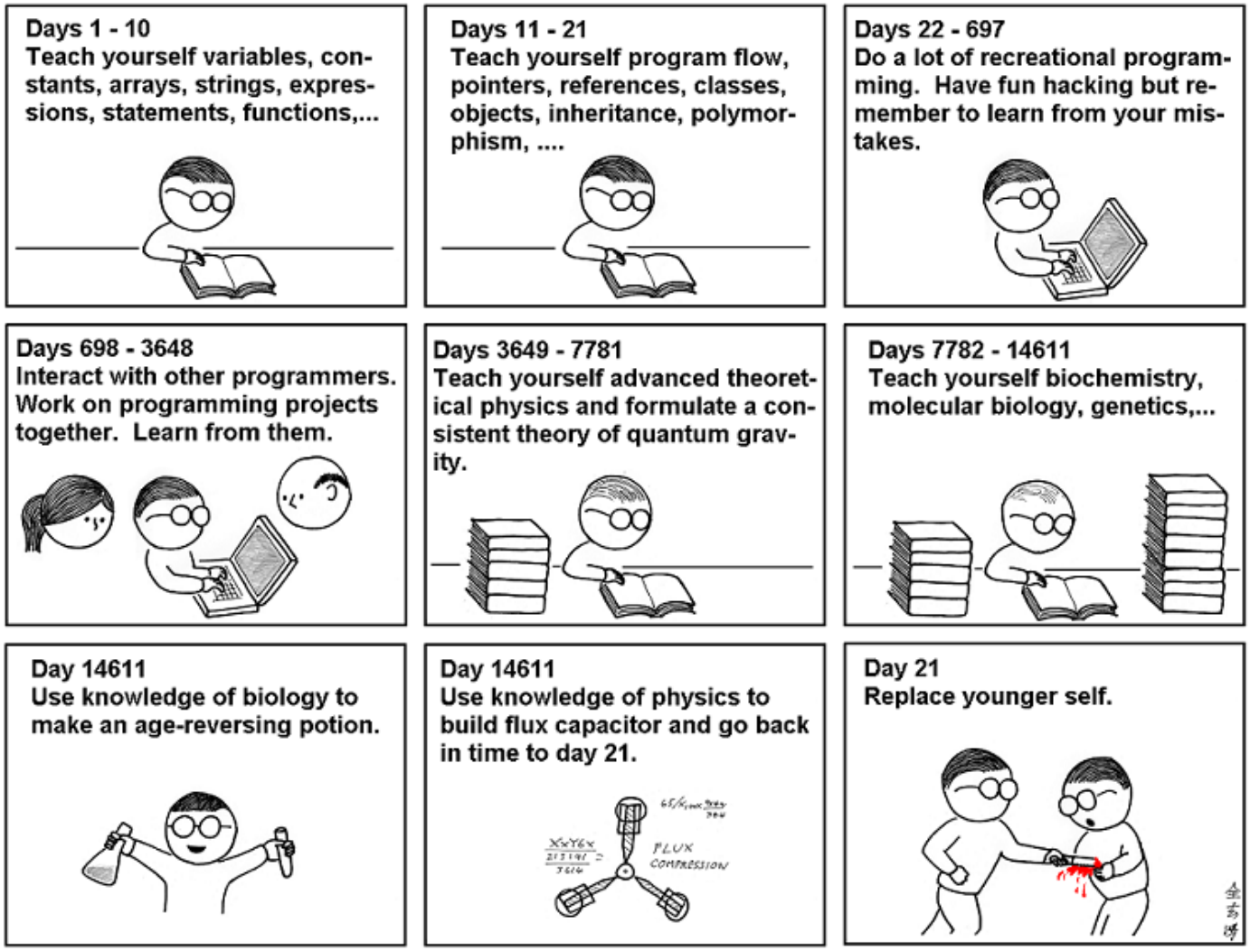
# Topics covered … (roughly)

- Introduction
- C++ Foundations I
- C++ Foundations II: Getting started and Procedural Programming.
- C++ Foundations III: Pointers, classical arrays and ...
- C++ Foundations IV: Control structures, loops, and various other topics ...
- C++ Foundations V: Handling files.
- Getting organised I: Pre-processing, macros, and Makefiles
- Getting organised II: Structs, unions, randomness, and time.
- Getting organised III: Exceptions (Part 1), namespaces, and defensive programming
- Getting organised IV: Debugging and profiling
- Programming with Class I: Fundamental syntax and some introductory UML
- Programming with Class II: Constructors, Destructors, ...
- Programming with Class III: Class/object relations
- Programming with Class IV: Class/class relations
- Programming with Class V: Overloading operators and making friends
- Programming withClass VI: Polymorphism, multiple inheritance
- Runtime type identification and casting
- Creating libraries, moving.
- Generic Programming I: Function templates and compiile time functionality
- Generic Programming II: Class templating
- Generic Programming III: Containers and iterators
- Generic Programming IV: The Standard Template Library (STL)
- Genetic Programming V: Template Compilation models
- Some final bits and pieces

# Languages evolve

- C++ continues to evolve.
- Don’t think of it as static.
  - This is true of most (programming) languages.
- We will try and integrate some of the more modern features of C++ (11 for sure, 14?, 17?) into this subject; but there isn’t time to cover close to everything.

# Learning to code in 21 days …




http://abstrusegoose.com/249

# Practice makes better …

- 21 days isn't enough.
  - For this subject or for competence.
- When there are examples in the lecture notes they are just that, examples, not points to learn.
  - You should practice variations and learn by making mistakes.
  - It's better to make mistakes in your own practice than in the assignment.
- The examples in the labs are there to help.
  - Do them! Ask questions!
- For those of you who haven't done (any/much) programming/coding before it's important that you get started early, yesterday would have been a good time!