For each course we identified two types of outcomes - the skills/graduatate attributes we want students to develop, and the technical content we want students to learn. The skills are developed and reinforced in each course, the content should belong to a single course.

Skills material will usually be addressed in course design and assessment design predominantly with only a small footprint in lecture time. As an aid to those teaching courses where it is not clear how to include skills material we should publishing a table with suggestions each course.

	1917	1927	2911
Content			
	Basic C: selection/loops /functions /operators	link between problem and algorithm complexity: sorting and searching a priori and a posteriora analysis NPC problems	Design (all followed by implementation) by contract TDD CRC cards
	Thorough understanding of how C uses memory: representation of basic types ints chars strings arrays pointers (2s compliment and floating point representation is optional) binary, hex signed and unsigned types and overflow memory segments mechanisms of pass by reference and copy malloc writing secure code, buffer	Trees + Graphs + Hash tables	OO programming (and design simultaneously) using collections framework (generics optional)

	overflows (stack frames optional) (microcontrollers and assembly optional)		
	Thorough understanding of pointers, linked lists, & * operators, sizeof	algorithms on Trees + Graphs + Hash tables	Patterns: Template Method, Factory Method/Abstract Factory, Iterator, Decorator, Composite, Strategy (optional Singleton, Observer, ...)
	Recursion, what is happening in memory	ADTS in C: implementing, designing	Methodologies for designing algorithms: brute force, greedy, divide and conquer, dynamic programming, backtrack, branch and bound, heuristic search
Skills/Attributes			
problem solving	topdown, XP, abstraction, not giving up, going from algorithm to implementation	selecting/adapting standard solutions, systematic approaches, going from problem to algorithm/data structure for known algorithms/data structures	design methodologies, going from problem to algorithm/data structure for unknown algorithms/data structures
groupwork			
style			
professional issues / ethics / real world practise			familiarity with IDE eclipse or similar
testing	assert, writing unit tests first	assert, writing unit tests first	TDD

debugging	printf, gdb or similar	printf, gdb or similar	
abstraction	functions / top down design / structs / APIs / standards	ADTS - using implementing enforcing violating advantages of	objects

text for 1927 - aho and ullman?