COMP90045 Programming Language Implementation

Credit Points:	12.50
Level:	9 (Graduate/Postgraduate)
Dates & Locations:	This subject is not offered in 2014.
Time Commitment:	Contact Hours: 36 hours, comprising of two 1-hour lectures and one 1-hour workshop per week Total Time Commitment: 200 hours
Prerequisites:	None
Corequisites:	None
Recommended Background Knowledge:	Basic proficiency in discrete mathematics.
Non Allowed Subjects:	433-361 Programming Language Implementation
Core Participation Requirements:	For the purposes of considering request for Reasonable Adjustments under the Disability Standards for Education (Cwth 2005), and Student Support and Engagement Policy, academic requirements for this subject are articulated in the Subject Overview, Learning Outcomes, Assessment and Generic Skills sections of this entry. It is University policy to take all reasonable steps to minimise the impact of disability upon academic study, and reasonable adjustments will be made to enhance a student's participation in the University's programs. Students who feel their disability may impact on meeting the requirements of this subject are encouraged to discuss this matter with a Faculty Student Adviser and Student Equity and Disability Support: http://services.unimelb.edu.au/disability
Contact:	email: **harald@unimelb.edu.au (mailto:harald@unimelb.edu.au)**
Subject Overview:	**AIMS** Good craftsmen know their tools, and compilers are amongst the most important tools that programmers use. There are many ways in which familiarity with compilers helps programmers. For example, knowledge of semantic analysis helps programmers understand error messages, and knowledge of code generation techniques helps programmers debug problems at assembly language level. The technologies used in compiler development are also useful when implementing other kinds of programs. The concepts and tools used in the analysis phases of a compiler are useful for any program whose input has a structure that is non-trivial to recognize, while those used in the synthesis phases are useful for any program that generates commands for another system. This subject provides an understanding of the main principles of programming language implementation, as well as first hand experience of the application of those principles. **INDICATIVE CONTENT** The subject describes how compilers analyse source programs, how they translate them to target programs, and what tools are available to support these tasks. Topics covered include compiler structures; lexical analysis; syntax analysis; semantic analysis; intermediate representations of programs; code generation; and optimisation.
Learning Outcomes:	**INTENDED LEARNING OUTCOMES (ILO)** On completion of this subject the student is expected to: 1 Describe important concepts and techniques in programming language implementation 2 Exploit their knowledge of compilers to be more effective programmers 3 Use analysis tools to help implement programs whose input has a structure that is non-trivial to recognize

	4 Use synthesis tools to help implement programs that generate commands for other programs
Assessment:	A multi-stage programming project during semester (30%), expected to take about 45 hours, addressing Intended Learning Outcomes (ILOs) 2, 3 and 4. A 3-hour end-of-semester written examination (70%), addressing all ILOs. Hurdle requirement: To pass the subject, students must obtain: 15/30 in project work 35/70 in the written examination
Prescribed Texts:	TBA
Breadth Options:	This subject is not available as a breadth subject.
Fees Information:	Subject EFTSL, Level, Discipline & Census Date, http://enrolment.unimelb.edu.au/fees
Generic Skills:	On completion of the subject students should have the following skills: # Ability to apply knowledge of science and engineering fundamentals # Ability to undertake problem identification, formulation and solution # Ability to utilise a systems approach to complex problems and to design for performance # Ability to manage information and documentation # Capacity for creativity and innovation
Notes:	**LEARNING AND TEACHING METHODS** The subject involves two 1-hour lectures per week followed by a 1-hour workshop. Weekly tutorial problems are assigned and discussed in class. The programming project is pivotal, as the subject relies heavily on learning-by-doing. Students work in small groups to implement a compiler for a non-trivial programming language. The project is staged, allowing for review and feedback throughout, including student peer review activity. In-plenum discussion of the project is allowed, and encouraged. **INDICATIVE KEY LEARNING RESOURCES** The subject uses online reading materials and an online discussion forum. It offers advance access to teaching materials, including slides used in lectures. **CAREERS / INDUSTRY LINKS** An understanding of program translation techniques, including parsing technologies, is essential in software engineering and software development and maintenance.
Related Course(s):	Master of Engineering in Distributed Computing Master of Information Technology Master of Information Technology Master of Information Technology Master of Philosophy - Engineering Master of Science (Computer Science) Master of Software Systems Engineering Ph.D.- Engineering
Related Majors/Minors/ Specialisations:	B-ENG Software Engineering stream Computer Science Master of Engineering (Software with Business) Master of Engineering (Software)