1. **Title of the module**

COMP6580 (CO658) – Programming Language Implementation

1. **School or partner institution which will be responsible for management of the module**

School of Computing

1. **The level of the module (Level 4, Level 5, Level 6 or Level 7)**

Level 6

1. **The number of credits and the ECTS value which the module represents**

15 credits (7.5 ECTS)

1. **Which term(s) the module is to be taught in (or other teaching pattern)**

Autumn or Spring

1. **Prerequisite and co-requisite modules**

Pre-requisites: COMP5450 Functional and concurrent programming; COMP5270 Operating systems and architecture; COMP5180 Algorithms, correctness, and efficiency

1. **The programmes of study to which the module contributes**

Computer Science and related programmes

1. **The intended subject specific learning outcomes.  
   On successfully completing the module students will be able to:**
   1. Understand how a computer program in a high-level, imperative language is translated into machine code;
   2. Understand how a program is executed, including run-time system support;
   3. Understand a variety of techniques that a compiler uses to improve the efficiency of its generated code;
   4. Understand how to represent programs as data in a typed functional language
   5. Implement basic compiler optimisation techniques;
   6. Evaluate a program’s performance; and
   7. Work with and modify an existing code base.
2. **The intended generic learning outcomes.  
   On successfully completing the module students will be able to:**
   1. Be able to clearly communicate the results of performance experiments;
   2. Be able to manage their own learning and development, through self-directed study and working on continuous assessment.
3. **A synopsis of the curriculum**

A study of techniques for interpreting and compiling programming languages, implementing them in a typed functional programming language (e.g., OCaml, Haskell). The module will outline a whole compiler from source to machine code, but will focus in depth on key algorithms and techniques. Possible in-depth topics include:

* writing interpreters,
* Hindley-Milner type inference,
* register allocation,
* garbage collection,
* abstract interpretation,
* static single assignment form.

The implemented language will be based on a simple imperative (e.g., Pascal-like) language with some extensions to address advanced topics in data layout (e.g., closures, objects, pattern matching). The course will be organized around a simple, but complete, example compiler that the student will have to understand and modify.

1. **Reading list (Indicative list, current at time of publication. Reading lists will be published annually)**

Aho, A., Lam, M., Sethi, R., and Ullman, J. (2007). *Compilers: Principles, Techniques, and Tools 2nd ed.,* Prentice Hall.  
Appel, A.W. (2004) *Modern compiler implementation in ML*, Cambridge University Press  
Cooper, K., and Torczan, L. (2011). *Engineering a compiler,* Morgan Kaufmann.  
Minsky, Y., Madhavapeddy, A., and Hickey, J (2013). *Real world OCaml,* O'Reilly Media.

1. **Learning and teaching methods**

Total contact hours: 35 hours

Private study hours: 115 hours

Total study hours: 150 hours

1. **Assessment methods**
   1. Main assessment methods

1 introductory assessment taking approx.. 5 hours (5%)

2 programming assessments taking approx. 25 hours each (25% and 30% respectively)

2 hour unseen written exam (40%)

13.2 Reassessment methods

Like for like.

1. ***Map of module learning outcomes (sections 8 & 9) to learning and teaching methods (section12) and methods of assessment (section 13)***

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Module learning outcome** | *8.1* | *8.2* | *8.3* | *8.4* | *8.5* | *8.6* | *8.7* | *9.1* | *9.2* |
| **Learning/ teaching method** |  |  |  |  |  |  |  |  |  |
| **Private Study** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** | **X** |
| Lectures | **X** | **X** | **X** | **X** |  | **X** | **X** |  |  |
| Terminal sessions |  |  |  | **X** | **X** |  |  |  |  |
| **Assessment method** |  |  |  |  |  |  |  |  |  |
| *Coursework* | x | x | x | x | x | x | x | x | x |
| *Exam* | x | x | x | x | x | x | x |  |  |
|  |  |  |  |  |  |  |  |  |  |

1. **Inclusive module design**

The School recognises and has embedded the expectations of current equality legislation, by ensuring that the module is as accessible as possible by design. Additional alternative arrangements for students with Inclusive Learning Plans (ILPs)/declared disabilities will be made on an individual basis, in consultation with the relevant policies and support services.

The inclusive practices in the guidance (see Annex B Appendix A) have been considered in order to support all students in the following areas:

a) Accessible resources and curriculum

b) Learning, teaching and assessment methods

1. **Campus(es) or centre(s) where module will be delivered**

Canterbury

1. **Internationalisation**

The topics addressed by this module relate to a field which is of international importance, given the global role of computers in today's technological innovation. The topics covered by this module are international in nature, being identical worldwide and independent of traditional spoken language.

**FACULTIES SUPPORT OFFICE USE ONLY**

**Revision record – all revisions must be recorded in the grid and full details of the change retained in the appropriate committee records.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Date approved | Major/minor revision | Start date of the delivery of revised version | Section revised | Impacts PLOs (Q6&7 cover sheet) |
|  |  |  |  |  |
|  |  |  |  |  |

Revised FSO Jan 2018