

## **MODULE: COMPUTER PROGRAMMING (course code BSCH-CP)**

**Griffith College Dublin – Computing Science**

**Year-long module (Fall & Spring)**

**This module is open to YEAR ABROAD STUDENTS ONLY. No pre-requisites.**

### **Aims**

This module shows you how to design high-quality programs in a systematic way. All the relevant concepts and techniques are explained and exemplified in the clearest, simplest language.

Many introductory courses on programming only teach the commands of some (popular) programming language, show a few examples of already-written programs, and then leave the student to work out how the programs were made. Our approach is different: we show you *how to design programs*; we demonstrate the techniques with many examples, and then take you through a long series of exercises and problems that have been designed to develop your problem solving and program design skills. Hence, for example, we will use only a small subset of a programming language—just enough to serve our needs.

Our aims are ambitious: we want you to become great programmers. This module is a first step towards achieving this goal. You don't have to know a programming language before taking this module; it's okay if you haven't programmed before. What is required, however, is a willingness to work conscientiously with us throughout the module; you need to develop an appetite for programming, a desire to learn as much as you can.

### **Learning Outcomes**

Upon successful completion of this module, you should be able to:

1. solve programming problems of modest complexity in a systematic, well-organised way
2. specify precisely the syntax and semantics of a programming language construct
3. select an appropriate program construct (or datatype) to achieve a given task
4. document accurately the design of a program on-the-fly
5. determine the basic efficiency of an algorithm
6. reproduce in detail the design and analysis of a range of standard algorithms
7. design a systematic suite of tests for a given program and implement it
8. prepare the text of a program in a well-formatted, conventional manner and develop these programs using an integrated development environment

### **Indicative Content**

<b>Topic</b>	<b>Description</b>
Introduction and motivation (first session)	<p>What is “computational thinking”? Brief tour of fundamental notions (e.g. computation, algorithm, program, programming language, programming, specification, correctness, efficiency, documentation) through simple case studies.</p> <p>Format of this module. How to study programming (regular study and practise essential).</p>
Programming-in-the-small (approx. 12 weeks)	<p>Expressions. Output. Basic program structure. Comments. Variables. Assignment. Input. Specifying alternatives and repetitions.</p> <p>Defining the notation precisely: EBNF, syntax diagrams.</p> <p>Functions. Parameters. Signature. Procedures. Copy rule.</p> <p>Integers. Fractionals. Booleans. Strings. Characters. Operators, precedence, order of association, undefinedness, finiteness, type coercion (light treatment), libraries (light treatment). Classes as records. Arrays.</p> <p>Invariants. Systematic loop design using invariants. Loop archetypes.</p>
Introduction to object-oriented programming (approx. 6 weeks)	<p>Classes. Objects. Instance variables. References. Abstraction. Private. Public. Methods. Constructor. Polymorphism.</p> <p>Exception handling (light treatment). Files.</p> <p>Introduction to inheritance, genericity, abstract datatypes, class invariants.</p> <p>Big case study.</p>
Introduction to algorithm design (approx. 6 weeks)	<p>Review of invariants.</p> <p>Sequence processing archetypes (e.g. Dutch National Flag, Find, select the minimum, reverse, rotate, shift, insert into sorted array).</p> <p>Simple sorting (e.g. insertion sort, selection sort, bubble sort).</p> <p>Binary search.</p> <p>Basic efficiency considerations.</p> <p>Introduction to recursion. Simple examples (e.g. factorial, Fibonacci, Towers of Hanoi, Quicksort).</p>
Systematic design (throughout)	<p>Stepwise design. “Understand the problem”. Making specifications precise. Invariants as loop design “blueprints”.</p> <p>Precise documentation as essential means of managing the design</p>

	process. Informal pre- and postconditions for procedures, functions, and methods. Naming conventions. Notion of software “tool”. Systematic testing. Test suites.
Practical tools (throughout)	Brief overview of compiler, interpreter, integrated development environment.

### **Assessment Methods**

Continuous assessment will be based on a combination of some of the following:

- Programming assignments (at least one per semester and a bigger project between the semesters)
- Selected homeworks
- Class tests
- Quizzes
- Practical tests
- Oral examination
- Take-home exams.

The continuous assessment work addresses all the learning outcomes. The written examination at the end addresses all but the last learning outcome.