Manifesto for COSC326

Discard your expectations. This paper is quite different from any other course you have taken so far. There will be no lectures, and there will be no obvious teaching, but you will undoubtedly learn a great deal and you will have to work very hard. We will introduce some elements of 'real life', where you will be the computer professional with the answers, who guarantees the client a working product, and where the client must be able to trust your competence.

Programming includes two important elements—analytical skills, and creative problem solving skills. For this reason, a great deal of time in this course will be spent on developing these skills through a series of problem solving assignments. You will be given problems and you will be expected to continue with a problem until you have solved it or satisfactorily completed the exercise. Some of the assignments will not require use of the computer, although you may find the computer useful for ancillary tasks.

Requirements for Attendance and Assessment

COSC326 is a purely practical course. There are two two-hour supervised laboratories every week and a one-hour 'Town Hall' meeting where we will discuss problems and analyse approaches. We will also use these sessions to drop hints, give out instructions, and listen to your ideas. *You are expected to attend all of these sessions*; they hinge on your participation. According to the standard University workload guidelines, you should also be prepared to spend roughly another 10 hours each week (average) outside the scheduled labs in reading, preparation, and unsupervised laboratory work. If you do not attend and put in the work, you are not likely to pass.

Assignments

Programming is a practical skill and you can learn only by practice. We cannot give you 'real' programming problems because these take weeks or months for professionals to complete. Instead, the course consists of a series of smaller problems to be solved in the laboratories. These exercises have been carefully chosen to make you discover and develop certain skills. In music teaching, such studies or *études* have been used for exactly this purpose for hundreds of years.

In real programming projects, part of the problem is cleaning up the defects in the specification you get from your client. You need to be alert to this because we do not guarantee that our problem specifications are clear or exact. We encourage you to ask for clarification of a problem whenever you are unsure about instructions and *before* you embark on programming.

Sample input, when provided, is just meant to indicate the correct format – it is not intended to stress test your submissions. Generally, your submissions will be tested against much more extensive or difficult data sets. One of the key skills to take away from COSC326 is being able to find ways to test your own work. In COSC326 "But it worked on the sample data" is not a valid reason to claim that a submission should be accepted.

You will do some assignments as individuals or in pairs and others in groups. Generally, a group can solve problems that are beyond the abilities of the individual members. You will also find the workload may be decreased through effective collaboration. To work most effectively in a group, you need to learn how to cooperate. Working in pairs or groups is an integral part of the course and you will be required to participate in these. However, even when assignments are completed by a group, we will reserve the right to monitor individual contributions, participation, and ability level in order to satisfy the success criteria of the course. We also reserve the right to ask for specific individual contributions.

Assessment

In this course, a program that works 'more or less' is not good enough. Computer Science professionals are expected to be able to solve problems, and real software is expected to meet basic professional standards. For this reason, the success criterion for COSC326 is that you complete each assignment to a satisfactory standard. The assignments, however, are designed also to facilitate learning and stretch your ability. The underlying philosophy of this paper ('more or less' is not good enough) means that we cannot just adopt a standard system where each etude counts for a certain number of points and these are added up to obtain a final grade.

The grading system within COSC326 has three levels of completion for each etude: accepted, merit, and excellence.

- **Accepted** Code runs and meets the I/O criteria. In cases where efficiency is an issue, small- and medium-sized examples run (i.e., the algorithm is not fundamentally broken). The code should be sufficiently documented and accompanied by a Readme file. For reports, the answers are given, and grammar/typography is to an acceptable standard.
- **Merit** Code runs efficiently on all examples and is well-structured. For reports, there are no (or very few) grammatical or typographical errors and the presentation is clear and correct.
- **Excellence** Completion will be above and beyond the basic criteria.

The accepted and merit standards will be explicitly given for each etude along with an indication of whether the etude would be considered for excellence and, if so, how this might be achieved. Resubmission to get from accepted to merit is allowed (indeed, encouraged), and resubmission to get from merit to excellence is also possible. Resubmission of not accepted submissions is allowed. We will use the following scheme for grading:

Grade	Completion Required	Criteria
C-	11 etudes	Omitting at most one individual etude
C	12 etudes	Omitting at most one individual etude
C+	All 13 etudes	
B-	All 13 etudes	Three (at least one individual) to merit standard
B	All 13 etudes	Six (at least three individual) to merit standard
B+	All 13 etudes	10 to merit standard (at least four individual)
A-	All 13 etudes	At least 10 to merit standard and at least one individual etude to excellence
A	All 13 etudes to merit standard	Three to excellence (including at least one individual)
A+	All 13 etudes to merit standard	Five to excellence (including at least two individual)

Submission of Assignments

We do not expect you to hand in the assignments at the end of each lab, but you are advised to try and keep up with the work. There may be specific deadlines for some of the assignments for reasons which will be obvious – these deadlines must be met. You will find that in many cases after your first submission you will be asked to do more work to complete an assignment and you need to leave time for this. No submissions of assignments will be accepted after *the last day of lectures*.

Academic integrity

In a paper such as COSC326 maintaining the standards of academic integrity is vitally important. General information on the standards and procedures connected with academic integrity and academic misconduct are available at the University's pages on academic integrity and academic misconduct, but there are a few issues specific to COSC326 as well. These will also be discussed in the first Town Hall session.

For the purpose of the following, "you" means "just you" for individual études, and "your pair/group" for pair/group études.

You may not:

- Ask questions on internet forums specific to any étude.
- Search for code online that's specific to an étude (or closely related to it).

- Provide access to any documents that you've produced in working on an étude to any other student or group (with one exception, test data – see below).
- Write code or reports while discussing them with anyone else (other than an instructor for COSC326).

You may:

- Share test data with any other students in COSC326 (with the limitation to single test cases not complete test files for bulk testing).
- Carry out general research online for the purposes of understanding a problem, finding appropriate data structures, etc.
- Discuss études with other members of the class.

There are two overriding general principles:

- Work that you submit for an étude must be your own, and any exceptions to that (e.g., if using a third-party library) must be clearly indicated.
- In case of any doubt ask permission rather than presuming that something is permitted/forbidden (if you don't ask, you should presume that it's forbidden!)

Course Objectives

This course aims to meet an extensive set of learning objectives with the overall goal of helping you to become more effective programmers through problem solving. After each étude you will see a list of numbers. These refer to skills in the list below, and specify the principal skills that the étude is trying to foster. These are not distributed evenly. Some skills are needed for almost every task and some simply require more attention because they are harder to develop. The most important generic skills are having a good understanding of a problem and working with people and these are therefore implicitly part of the objectives of almost every étude.

1. Understanding a problem

1.1 simplifying

1.2 clarifying

1.3 generalising

1.4 specifying

2. Problem solving strategies

2.1 lateral and creative approaches to problem solving

2.2 solving problems according to specification

2.3 top-down and bottom-up solution designs

2.4 solving a related problem

2.5 working backwards

2.6 choosing appropriate tools, e.g. pen and paper, spreadsheet, programming language

2.7 applying persistence and thoroughness, removing mental blocks

2.8 appropriate, effective, and creative use of resources (time, personal abilities, library etc.)

2.9 simulation, finding metaphors, trial and error

2.10 designing and using appropriate notation

3. Computer related techniques

3.1 understanding the limitations and problems specific to computer program execution (such as overflow, rounding errors, division by zero)

3.2 understanding file formats, platform specifics and proprietary aspects of computing

3.3 understanding recursion and iteration and when and why to use them

3.4 using appropriate data structures, considering suitability, simplicity and size

3.5 adequate testing, debugging and validation

3.6 understanding efficiency, profiling and measuring

3.7 understanding and using fundamental algorithms

4. Working with people

4.1 understanding different approaches required in team and individual projects

4.2 becoming aware of group dynamics; identifying different skills of group members, task allocation, participation, conflict resolution

4.3 collaboration and ego-less programming

4.4 working with other people's programs; reading, understanding, debugging, maintaining, re-using code, libraries

4.5 producing software that is adequately documented, commented, and user-friendly

4.6 an understanding of ethics, and professional responsibilities towards colleagues and clients, including confidentiality and software protection

4.7 recording proceedings, decisions, progress etc.

4.8 writing clear and concise reports