EXPANDED COURSE DESCRIPTION
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Lassonde School of Engineering
Electrical Engineering Computer Science

LE / EECS 2001 3.0 SECTION A
INTRO. TO THE THEORY OF COMPUTATION
FALL 2018 / WINTER 2019

Last Modified Date: 08/20/2018

COURSE CALENDAR DESCRIPTION
Introduction to the theory of computing, including automata theory, formal languages and Turing machines; theoretical models and their applications in various fields of computer science. The emphasis is on practical applications of the theory and concepts rather than formal rigour. Prerequisites: cumulative GPA of 4.50 or better over all major EECS courses (without second digit “5”); LE/EECS 1021 3.00 or LE/EECS 1022 3.00 or LE/EECS 1720 3.00 or LE/EECS 1030 3.00; LE/EECS 1028 3.00 or SC/MATH 1028 3.00 or LE/EECS 1019 3.00 or SC/MATH 1019 3.00. Previously offered as: LE/CSE 2001 3.00. PRIOR TO SUMMER 2013: SC/CSE 2001 3.00.

INSTRUCTOR(S)

<table>
<thead>
<tr>
<th>Name</th>
<th>Section / Format / Term</th>
<th>Contact Email</th>
<th>Contact Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edmonds, Jeffrey A</td>
<td>Sec. A / LECT / F</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ADDITIONAL INFORMATION

Course Description
CSE 2001 3.00 Introduction to the Theory of Computation The course introduces different theoretical models of computers and studies their capabilities and theoretical limitations. Topics covered typically include the following.

- Finite automata and regular expressions; practical applications, e.g. text editors
- Pushdown automata and context-free grammars; practical applications, e.g. parsing and compilers
- Turing machines as a general model of computers
- Introduction to unsolvability and the halting problem

Learning Outcomes
In this course, you will be invited to develop your ability to think clearly and carefully about computing, and to improve your skills in expressing those thoughts about computing in a precise way. By the end of this course, you will be able to do the following things.

- **Design machines (e.g., finite automata, Turing machines) to solve specified problems.**
  The central goal of computer science is the design of algorithms to solve computational problems. The ‘machines’ that we describe in this course are not physical machines; they are just ways to formally describe algorithms. We will also talk a bit about how to specify problems formally.

- **Design regular expressions and context-free grammars for a given formal language**
  A formal language is just a set of strings. We will see that they can be used to give formal specifications of computational problems. This part of the course looks at some ways to describe these languages in a compact way. Regular expressions and grammars are also useful in the design of programming languages and compilers, and in natural language understanding.

- **Explain why an object designed according to one of the above two bullets correctly meets its specification**
Computer scientists should be able to design solutions for computational problems. But they must also be able to explain their solution to others. This goal of the course will help you develop your skills in doing this. You will find that proving that your solution is correct also helps you find mistakes in your solution.

- **Prove simple properties about models of computation (e.g., that the class of regular languages is closed under complement)**

  Here, we start to develop an overall theory of computing. Rather than thinking about one problem at a time, we look at the class of problems that can be solved by certain kinds of algorithms and prove properties about that class. This gives us deeper insight into how computers can solve problems.

- **Demonstrate limits of computing by proving that a problem is not solvable within a particular model of computation**

  This is one of the most important components of the course. It's important for a computer scientist to know the limitations of computers, to avoid wasting time on trying to solve a task that turns out to be impossible.

- **Show how one problem can be reduced to another**

  This is a skill that computer scientists use all the time. Instead of solving a problem from scratch, it helps if you can relate it to another problem that has already been solved, and then use the solution to that second problem to solve the original one.

**How to Learn This Material**

Some of the skills that you will develop in this course may be quite new to you, and different from things you have done in previous courses. This is good: it means you’re learning new and (I hope) exciting things. However, it means that you will need practice to master them. Just participating in classes isn't enough. There are suggested exercises from the textbook. Web pages for this course in previous terms also include many more problems to work on. Do lots.

You learn by struggling with problems. However, if you get too stuck or don't know how to begin, help is available. Talk to your classmates (however; see the notes below about academic honesty regarding discussing assignment problems with others). Go to office hours; the instructor and TA are there to help you! You also learn by making mistakes and getting feedback about them. Just make sure that you use the feedback to improve your understanding.

Groups of students can learn a lot by explaining their solutions to the suggested exercises from the textbook to one another and critiquing the solutions of others. After all, learning how to explain solutions clearly is one of the learning objectives of this course. Seeing where other students' solutions are unclear to you helps you make your own explanations clearer. Be aware that a problem may have many different correct solutions; just because someone's solution is different from yours doesn’t necessarily mean that one of them is wrong.

It takes time to build new skills, so it helps if you work on exercises regularly: don't leave all the work to the days right before a test. Sometimes students ask for more exercises with worked-out solutions. (The textbook has some, but maybe not enough.) There is a whole shelf of textbooks that cover the material of this course in the library (some are recommended below), and many have more examples or exercises with solutions.

**ACADEMIC INTEGRITY LINKS**

- Senate Policy on Academic Honesty - [http://secretariat-policies.info.yorku.ca/policies/academic-honesty-senate-policy-on/](http://secretariat-policies.info.yorku.ca/policies/academic-honesty-senate-policy-on/)
- Academic Integrity - [http://lassonde.yorku.ca/academic-integrity](http://lassonde.yorku.ca/academic-integrity)

**STUDENT LINKS**

- Student Rights and Responsibilities - [http://oscr.students.uit.yorku.ca/student-conduct](http://oscr.students.uit.yorku.ca/student-conduct)
- Religious Observance - [https://w2prod.sis.yorku.ca/Apps/WebObjects/cdm.woa/wa/regobs](https://w2prod.sis.yorku.ca/Apps/WebObjects/cdm.woa/wa/regobs)
- Counselling and Disability Services - [http://cds.info.yorku.ca/](http://cds.info.yorku.ca/)
- York University’s Policies on Gender/LGBTQ*/Positive Space - [http://rights.info.yorku.ca/lgbtq/](http://rights.info.yorku.ca/lgbtq/)
LAND ACKNOWLEDGEMENT

- We acknowledge our presence on the traditional territory of many Indigenous Nations. The area known as Tkaronto has been care taken by the Anishinabek Nation, the Haudenosaunee Confederacy, the Huron-Wendat, and the Métis. It is now home to many Indigenous Peoples. We acknowledge the current treaty holders, the Mississaugas of the New Credit First Nation. This territory is subject of the Dish With One Spoon Wampum Belt Covenant, an agreement to peaceably share and care for the Great Lakes region.
- The Indigenous Framework for York University: A Guide to Action can be found here: http://indigenous.info.yorku.ca/
- Meaning of a land acknowledgement: http://healthydebate.ca/opinions/indigenous-land-acknowledgements

Many courses utilize Moodle, York University’s course website system. If your course is using Moodle, click here to access it.
Moodle @ York University