COMP 1500 – Computing: Ideas and Innovation

Course Description

Calendar entry

An introduction to the topics of Computer Science and problem solving. Students will learn concepts in computer programming. May not be used to fulfill computer science requirements in a Computer Science Honours, Joint Honours, Major, General or Minor program. May not be taken once in a declared Computer Science Honours, Joint Honours, Major, General or Minor program. May be used as an elective if taken prior to entry.

General Course Description

You may not know anything about computers, and that's totally OK. This course will start by giving you a general overview of some of the fundamental concepts of Computer Science, starting from the bottom (binary, and hardware), working up to abstractions and problem solving (algorithms and data structures), and finally, application of fundamental concepts to a domain area (artificial intelligence, bioinformatics, etc.). You'll also get a chance to do some basic programming in this course using a visual programming environment called Snap!

Detailed Prerequisites

Before entering this course, a student should be able to:

- Evaluate arithmetic expressions, applying the rules of order of operation to basic arithmetic operators (+, −, ×, ÷) and parentheses.
- Use a web browser and have basic document editing skills (e.g., word processing, slide creation).

Course Goals

By the end of this course students will:

- Have a better idea of the breadth of Computer Science (it's not just programming!).
- Represent information digitally.
- Build simple circuits for calculating values and show how those circuits can be combined to perform more complicated calculations.
- Apply basic problem-solving skills.
- Write, use, and evaluate simple algorithms.
- Implement abstractions as software.

Learning Outcomes

Computer Science

Students should be able to:

1. Explain what Computer Science is in a written assignment by giving examples of subdisciplines in Computer Science.
2. Define the term abstraction.
3. Explain why abstractions are necessary to solve complex problems with computers.
4. Describe how a Turing machine behaves.
5. Define and describe the von Neumann architecture.

Representing information digitally

Students should be able to:

1. Convert numbers between number systems (binary, decimal, hexadecimal).
2. Compute how many bits are required to store certain numbers.
3. Explain how text is stored digitally (ASCII vs. Unicode).
4. Explain how images are stored digitally (black and white → grayscale → colour).
5. Predict how an animation might be stored digitally.
6. Explain how formats and encodings are important for communication.

How computers function

Students should be able to:

1. Evaluate Boolean expressions with logical operators AND, OR, NOT, and XOR.
2. Construct a Boolean expression that satisfies a truth table.
3. Explain how XOR and AND can be used to add two single-bit numbers.
4. Translate Boolean expressions into circuits and vice versa.
5. Build a full-adder circuit up to four bits.
6. Determine the output of a circuit given a circuit diagram.
7. Describe the structure of a simple CPU architecture (Arithmetic Logic Unit, Control Unit, memory hierarchy, memory addressing)

Basic assembly language programming

Students should be able to:

1. Use a simple assembly language to write programs that can do the following.
    a. Perform multi-step arithmetic calculations.
    b. Receive inputs and produce an output.

    c. Test a condition and make a decision based on the result.
    d. Repeat a sequence of steps until some condition is met.
2. Determine the output of a simple assembly language program.

Problem solving, algorithms, and complexity

Students should be able to:

1. Explain what an algorithm is.
2. Divide a big problem (e.g., "How do I sort these?") into smaller steps.
3. Find information in a list using linear search and binary search.
4. Sort information using bubble sort and selection sort.
5. Rank searching and sorting algorithms by runtime complexity (Big O notation).

Implement abstractions as software

Students should be able to use a visual programming language to:

1. Move a sprite around on a screen.
2. Repeat blocks using loops (forever, n times, event-based conditions).
3. Detect events using conditional blocks and Boolean expressions.
4. Store and modify values in variables (numbers).
5. Use random numbers in a program.
6. Make custom blocks for code reuse.
7. Use input to customize the behaviour of a program.

Disciplines within Computer Science

Students taking this course should be able to accomplish certain tasks within a discipline of Computer Science, based on the preference of the instructor. The following are examples that have been taught in this course. They are not meant to be prescriptive or required.

Bioinformatics

Students should be able to:

1. Define common terms used in microbiology to describe life sciences data (DNA, base pairs, codons, genes, genomes, sequencing).
2. Explain the purpose of assembly algorithms.
3. Compare and contrast reference-based assembly to de novo assembly.
4. Explain how sequencing reads can be represented in a graph.
5. Define the term de Bruijn graph.
6. Build a de Bruijn graph from a set of sequencing reads.
7. Reconstruct a genome from a de Bruijn graph.
8. Justify the need for sequence alignment.

9. Align sequences using the Needleman-Wunsch algorithm.

Cryptography

Students should be able to:

1. Define the term cryptography.
2. Justify the need for cryptography.
3. Define the terms “plaintext”, “cipher”, “ciphertext”, “encryption”, and “decryption”.
4. Use a shared key cipher (e.g., Caesar Cipher) to encrypt and decrypt a message.
5. Use frequency analysis to decrypt a simple ciphertext without a key.
6. Use brute force to decrypt a simple ciphertext without a key.
7. Define the term “public key cryptography”.
8. Explain the significance of prime numbers and the factoring problem to RSA encryption.
9. Find the prime factors of a (small!) composite number and use the result to decrypt a message that was encrypted with RSA.