

**COURSE CSE3010: COMPUTER SCIENCE 3**

**Level:** Advanced

**Prerequisites:** CSE2010: Computer Science 2  
CSE2110: Procedural Programming 1

**Description:** Students explore hardware, software and associated processes at an advanced level. They extend their understanding of software development by moving from procedural programming approaches to an object-oriented approach. In the process they learn how object-oriented programming (OOP) can improve the efficiency and robustness of algorithm development and program construction. They deepen their understanding of the hardware side of computer science by exploring the connection between the binary/hexadecimal number systems and some of the simple logic gates that are the basis of the von Neumann computer. They also add to their understanding of the social implications of computer science by examining the emerging information society.

**Parameters:** Access to an appropriate computer work station, the Internet, a programming language/environment and support resources. It is recommended that the course be taught in tandem with one or more programming courses dealing with OOP.

**Supporting Courses:** CSE3110: Iterative Algorithm 1  
CSE3120: Object-oriented Programming 1 or any  
Advanced project course involving OOP

**Outcomes:** The student will:

- 1. analyze and explain the historical roots and general nature of the information revolution and the emerging information, knowledge-based society**
  - 1.1 analyze and explain the:
    - 1.1.1 technological roots of the information revolution
    - 1.1.2 general economic impact
    - 1.1.3 social impact
    - 1.1.4 political impact
    - 1.1.5 shift from the Industrial Revolution to the Information Age
- 2. explain and represent the nature, rationale and key approaches associated with OOP**
  - 2.1 compare and contrast procedural programming and OOP approaches highlighting the:
    - 2.1.1 approach to modularity
    - 2.1.2 protection and hiding of data
    - 2.1.3 use of interfaces to maintain implementation independence
    - 2.1.4 approaches to organizing algorithms and programs
    - 2.1.5 respective focus of OOP and structured programming

- 2.2 describe key aspects of object-oriented design and OOP including:
  - 2.2.1 abstraction, encapsulation, inheritance and polymorphism
  - 2.2.2 classes, class libraries, objects and instantiation
  - 2.2.3 data members (properties) and member functions (behaviours)
  - 2.2.4 public and private access modifiers
  - 2.2.5 message passing and object networks
- 3. demonstrate object-oriented design techniques**
  - 3.1 demonstrate requirement analysis including:
    - 3.1.1 case analysis
    - 3.1.2 domain analysis
    - 3.1.3 object diagrams
  - 3.2 demonstrate iterative class design using:
    - 3.2.1 principal classes
    - 3.2.2 elaboration of object diagrams
    - 3.2.3 class-responsibility-collaboration cards
    - 3.2.4 iterative prototyping
  - 3.3 demonstrate appropriate relationships including:
    - 3.3.1 dependency
    - 3.3.2 association
    - 3.3.3 aggregation
    - 3.3.4 composition
- 4. explain and demonstrate the relationship between binary and hexadecimal number systems, data encoding, logic gates and the digital computer**
  - 4.1 describe and represent the binary and hexadecimal system by:
    - 4.1.1 comparing and contrasting each system with the decimal system
    - 4.1.2 converting numbers from one system to another
    - 4.1.3 describing and demonstrating the role each system plays in encoding data for digital computing
  - 4.2 describe and represent binary arithmetic by:
    - 4.2.1 creating a simple binary addition truth table
    - 4.2.2 creating simple binary truth tables for basic logical states such as logical conjunction (AND), logical disjunction (OR) and logical negation (NOT)
  - 4.3 describe and represent circuits (general and logic) using binary notation including:
    - 4.3.1 AND
    - 4.3.2 OR
    - 4.3.3 NOT
    - 4.3.4 NAND and NOR
  - 4.4 describe and represent simple operations with logic gates including:
    - 4.4.1 binary addition with full and half adders
    - 4.4.2 binary subtraction with full and half subtractors
    - 4.4.3 tasks involving selection
    - 4.4.4 tasks involving inversion
- 5. demonstrate basic competencies**
  - 5.1 demonstrate fundamental skills to:
    - 5.1.1 communicate
    - 5.1.2 manage information
    - 5.1.3 use numbers
    - 5.1.4 think and solve problems

- 5.2 demonstrate personal management skills to:
  - 5.2.1 demonstrate positive attitudes and behaviours
  - 5.2.2 be responsible
  - 5.2.3 be adaptable
  - 5.2.4 learn continuously
  - 5.2.5 work safely
- 5.3 demonstrate teamwork skills to:
  - 5.3.1 work with others
  - 5.3.2 participate in projects and tasks
- 6. create a transitional strategy to accommodate personal changes and build personal values**
  - 6.1 identify short-term and long-term goals
  - 6.2 identify steps to achieve goals



**COURSE CSE3020: COMPUTER SCIENCE 4**

**Level:** Advanced

**Prerequisites:** CSE3010: Computer Science 3  
CSE3110: Iterative Algorithms 1

**Description:** Students enhance their learning by studying a set of standard abstract data types and the dynamic data structures conventionally used to implement them. They also add to their general understanding of algorithms by learning how to conduct asymptotic analyses of algorithmic efficiency and indicate that efficiency using big O notation. Students continue their exploration of the hardware aspect of computer science by exploring a different type of computer architecture, the Turing machine.

**Parameters:** Access to an appropriate computer work station, the Internet, a programming language/environment, preferably one that supports object-oriented programming, and associated support resources. It is recommended that the course be taught in tandem with one or more programming courses dealing with dynamic abstract data types and their implementation.

**Supporting Courses:** CSE3310: Recursive Algorithms 1  
CSE3320: Dynamic Data Structures 1 or any  
Advanced project course involving abstract data type programming

**Outcomes:** The student will:

**1. analyze and represent the nature, structure, utility and key operations associated with dynamic abstract data types (ADTs) available in high-level programming languages**

- 1.1 list and explain the advantages and disadvantages of dynamic ADTs
- 1.2 describe the nature and structure of common and useful dynamic ADTs such as:
  - 1.2.1 lists
  - 1.2.2 stacks
  - 1.2.3 queues and priority queues
  - 1.2.4 sets
  - 1.2.5 maps
  - 1.2.6 trees
- 1.3 list and describe dynamic data structures used to implement dynamic ADTs including:
  - 1.3.1 user-created dynamic arrays and associative arrays
  - 1.3.2 user-created linear linked structures
  - 1.3.3 user-created hash tables
  - 1.3.4 specialized class libraries
- 1.4 list and describe common data operations associated with the dynamic data structures including:
  - 1.4.1 traversing the items in the data structure
  - 1.4.2 finding and/or retrieving an item
  - 1.4.3 adding, removing or replacing an item
  - 1.4.4 determining the size of the structure

- 1.4.5 determining if the structure is empty
- 1.4.6 providing a copy or subset of the collection
- 1.4.7 comparing or combining with other data structures of the same type
- 2. analyze and represent the nature, utility, approaches and nomenclature associated with the asymptotic analyses of algorithmic efficiency**
  - 2.1 describe asymptotic analysis
  - 2.2 explain the relationship between efficiency and complexity in the analysis of algorithms
  - 2.3 represent the efficiency and complexity of an algorithm using big O notation including:
    - 2.3.1 constant growth
    - 2.3.2 logarithmic growth
    - 2.3.3 linear growth
    - 2.3.4 linearithmic growth
    - 2.3.5 polynomial growth
    - 2.3.6 exponential growth
- 3. explain and demonstrate the rationale and use of recursive and introductory recursive operations**
  - 3.1 compare and contrast recursion with iteration
  - 3.2 evaluate and identify problems that require recursive processes
  - 3.3 describe and represent recursive program flow
- 4. analyze and represent the nature, architecture, operation and utility of a Turing machine**
  - 4.1 explain the nature and use of Turing machines
  - 4.2 represent a Turing machine
  - 4.3 demonstrate the ability to execute simple programs on a Turing machine
  - 4.4 create simple state representations
- 5. demonstrate an understanding of computer science's impact on society by preparing and delivering a presentation on a personally relevant area of interest where computer science intersects with society**
- 6. demonstrate basic competencies**
  - 6.1 demonstrate fundamental skills to:
    - 6.1.1 communicate
    - 6.1.2 manage information
    - 6.1.3 use numbers
    - 6.1.4 think and solve problems
  - 6.2 demonstrate personal management skills to:
    - 6.2.1 demonstrate positive attitudes and behaviours
    - 6.2.2 be responsible
    - 6.2.3 be adaptable
    - 6.2.4 learn continuously
    - 6.2.5 work safely
  - 6.3 demonstrate teamwork skills to:
    - 6.3.1 work with others
    - 6.3.2 participate in projects and tasks
- 7. create a transitional strategy to accommodate personal changes and build personal values**
  - 7.1 identify short-term and long-term goals
  - 7.2 identify steps to achieve goals

**COURSE CSE3110: ITERATIVE ALGORITHM 1**

**Level:** Advanced

**Prerequisite:** CSE2120: Data Structures 1

**Description:** Students learn a number of standard iterative data processing algorithms useful for working with data structures such as arrays. These include an iterative version of the binary search, the three basic sorts—exchange (bubble), insertion and selection, and a simple merge. In the process, they learn when and where to apply these algorithms.

**Parameters:** Access to appropriate computer equipment, software, the Internet and support materials. Access to an object-oriented programming environment that encourages a formal treatment of objects.

**Supporting Courses:** CSE2130: Files & File Structures 1  
CSE3010: Computer Science 3  
CSE3120: Object-oriented Programming 1

**Outcomes:** The student will:

- 1. analyze and represent the nature, structure and utility of common iterative algorithms**
  - 1.1 compare and contrast search, sort and merge algorithms
  - 1.2 explain the way in which search, sort and merge algorithms manipulate data
  - 1.3 describe the data structures required by search, sort and merge algorithms
  - 1.4 describe how search, sort and merge algorithms are implemented in a programming environment
  - 1.5 describe and represent iterative search algorithms including:
    - 1.5.1 linear search
    - 1.5.2 binary search
    - 1.5.3 compare and contrast how linear and binary searches manipulate data
    - 1.5.4 compare and contrast the data structures required and the computational efficiencies of linear and binary searches
  - 1.6 describe and represent basic iterative sort algorithms including:
    - 1.6.1 exchange sort; e.g., bubble sort, cocktail sort, gnome sort, comb sort
    - 1.6.2 selection sort; e.g., selection sort, strand sort
    - 1.6.3 insertion sort; e.g., insertion sort, library sort
    - 1.6.4 comparing and contrasting how different classes of sorts manipulate data
    - 1.6.5 comparing and contrasting the data structures required and the computational efficiencies of different classes of sorts
  - 1.7 describe and represent simple iterative merge algorithms
- 2. create and/or modify algorithms that use searches, sorts and merges to solve problems**
  - 2.1 demonstrate the use of appropriate general design techniques for the programming environment being considered for implementation
  - 2.2 analyze and decompose the problem into appropriate subsections using the decomposition techniques appropriate for the chosen design approach
  - 2.3 evaluate subsections and identify any that may require some type of search, sort and/or merge algorithm, based on the nature of the data to be processed and the type of processing operations
  - 2.4 identify which algorithms are appropriate or required to search, sort and/or merge data

- 2.5 sequence the various subsections appropriately
- 2.6 test and modify the developing algorithm with appropriate data using a “fail-on-paper” process
- 3. create and/or modify programs that use searches, sorts and merges to solve problems**
  - 3.1 convert algorithms calling for standard iterative structures into programs that reflect the algorithm’s design
  - 3.2 use original (user-created) or pre-existing search, sort and/or merge algorithms appropriate to the data being manipulated
  - 3.3 utilize the appropriate operators, methods, functions or procedures required to carry out the standard algorithms
  - 3.4 use internal and external documentation
- 4. compare program operation and outcomes with the intent of the algorithm and modify, as required**
  - 4.1 use appropriate error-trapping mechanisms built into the programming environment, as well as programmer-directed error-trapping techniques, to eliminate logic errors and debug the program
  - 4.2 compare the congruency between the outcomes of the debugged program and the original intent of the algorithm and modify both, as required
- 5. demonstrate basic competencies**
  - 5.1 demonstrate fundamental skills to:
    - 5.1.1 communicate
    - 5.1.2 manage information
    - 5.1.3 use numbers
    - 5.1.4 think and solve problems
  - 5.2 demonstrate personal management skills to:
    - 5.2.1 demonstrate positive attitudes and behaviours
    - 5.2.2 be responsible
    - 5.2.3 be adaptable
    - 5.2.4 learn continuously
    - 5.2.5 work safely
  - 5.3 demonstrate teamwork skills to:
    - 5.3.1 work with others
    - 5.3.2 participate in projects and tasks
- 6. create a transitional strategy to accommodate personal changes and build personal values**
  - 6.1 identify short-term and long-term goals
  - 6.2 identify steps to achieve goals

**COURSE CSE3120: OBJECT-ORIENTED PROGRAMMING 1**

**Level:** Advanced

**Prerequisite:** CSE2110: Procedural Programming 1

**Description:** Students add to their understanding of programming paradigms by moving from a procedural programming approach, in which modularity is handled through subprograms, to an object-oriented approach, in which it is handled through objects. They learn a simple object-oriented analysis and design approach based on the use of object diagrams and write programs that use objects associated with one another in a client/server relationship.

**Parameters:** Access to appropriate computer equipment, software, the Internet and support materials. Access to an object-oriented programming (OOP) environment that encourages a formal treatment of objects.

**Supporting Courses:** CSE3010: Computer Science 3  
CSE3110: Iterative Algorithm 1

**Outcomes:** The student will:

- 1. explain and represent the nature, rationale and key approaches associated with OOP**
  - 1.1 describe the core concepts of OOP including:
    - 1.1.1 implementation by the exchange of “messages” among “objects”
    - 1.1.2 an outline of the key features of the OOP approach: e.g. encapsulation, modularity, polymorphism, inheritance
    - 1.1.3 use of private, public and protected members, accessors and modifiers to control access to data
    - 1.1.4 use of encapsulation and modularity when writing algorithms and programs
    - 1.1.5 use of classes and objects
    - 1.1.6 an outline the paradigm shift that occurred in the move from imperative and procedural programming to OOP
    - 1.1.7 the advantages of OOP over earlier paradigms
  - 1.2 explain key differences between OOP and procedure-oriented programming in:
    - 1.2.1 designing programs
    - 1.2.2 the storage and access of data
    - 1.2.3 the maintenance of programs
- 2. demonstrate object-oriented design skills by writing algorithms employing an object-oriented approach to solving problems**
  - 2.1 write algorithms and programs that deal with a small number of classes with an associative relationship
  - 2.2 use an iterative and incremental approach in the analysis, design and development (architecture) stages of the software development process
  - 2.3 apply an object-oriented design model to solve a data processing problem including:
    - 2.3.1 requirement analysis
    - 2.3.2 case analysis
    - 2.3.3 domain analysis

- 2.4 use an iterative and incremental approach to refine the architecture into appropriate class or object diagrams showing their relationships
- 2.5 analyze and refine the diagrams identifying the client/server relationship among the objects and determining the messages that need to be passed between objects and how the objects interface
- 2.6 draft an informal object message sequence indicating the flow of messages in the system
- 2.7 complete the object design by adding private methods, functions and data structures required to implement the various objects
- 2.8 test and modify, as required, the developing algorithm at each stage with appropriate data
- 3. translate algorithms into source code, convert the source code into machine executable form, execute and debug, as required**
  - 3.1 use iterative and incremental approaches in the implementation, testing and maintenance phases of the software development process
  - 3.2 break the algorithm into appropriate sections for implementation using a prototype approach
  - 3.3 create the classes necessary to instantiate the objects called for by the algorithm
  - 3.4 as the classes are constructed, use the server classes to create the client classes establishing the client/server relationships called for by the algorithm
  - 3.5 test and modify the sections, as required
  - 3.6 convert an algorithm into a program of linked classes, objects, instances and methods in a manner that reflects the structure of the algorithm using an iterative and incremental approach
  - 3.7 profile and optimize the code to add additional sections and/or features to the growing program
  - 3.8 where appropriate, collaborate with other students to carry out OOP tasks
  - 3.9 create internal and external documentation
  - 3.10 analyze the program and eliminate syntax, run-time and logic errors by using appropriate test data for each programming task at each stage of development
- 4. compare the results of the program with the intent of the algorithm and modify, as required**
- 5. demonstrate basic competencies**
  - 5.1 demonstrate fundamental skills to:
    - 5.1.1 communicate
    - 5.1.2 manage information
    - 5.1.3 use numbers
    - 5.1.4 think and solve problems
  - 5.2 demonstrate personal management skills to:
    - 5.2.1 demonstrate positive attitudes and behaviours
    - 5.2.2 be responsible
    - 5.2.3 be adaptable
    - 5.2.4 learn continuously
    - 5.2.5 work safely
  - 5.3 demonstrate teamwork skills to:
    - 5.3.1 work with others
    - 5.3.2 participate in projects and tasks
- 6. create a transitional strategy to accommodate personal changes and build personal values**
  - 6.1 identify short-term and long-term goals
  - 6.2 identify steps to achieve goals

**COURSE CSE3130: OBJECT-ORIENTED PROGRAMMING 2**

**Level:** Advanced

**Prerequisite:** CSE3120: Object-oriented Programming 1

**Description:** Students extend their knowledge of object-oriented programming (OOP). They add to their expertise in object-oriented design by using some of the techniques associated with the UML design approach and to their programming expertise by writing programs that explore association between classes. Students work with abstract classes, developing algorithms that employ the object diagram approach and programs that use templated classes, containment and inheritance to promote reusability.

**Parameters:** Access to appropriate computer equipment, software, the Internet and support materials. Access to OOP environment that encourages a formal treatment of objects.

**Supporting Courses:** CSE3010: Computer Science 3  
CSE3110: Iterative Algorithm 1

**Outcomes:** The student will:

**1. explain and represent class and object interactions possible in OOP**

- 1.1 outline the key properties of the OOP approach
- 1.2 describe and demonstrate how coding can be reduced and responsibilities distributed through the appropriate use of polymorphism and inheritance
- 1.3 describe and represent the relationship among the classes, objects, instances and methods including:
  - 1.3.1 inheritance
  - 1.3.2 association
  - 1.3.3 composition and aggregation
- 1.4 describe and represent ways in which inheritance and polymorphism are promoted
- 1.5 outline how static classes, polymorphism and inheritance may be used to hide and/or protect data

**2. demonstrate OOP skills by writing algorithms employing an object-oriented approach to solving problems**

- 2.1 apply an object-oriented analysis and design model to decompose a data processing problem into a form that is accessible to an OOP approach by using:
  - 2.1.1 an informal domain analysis
  - 2.1.2 an informal use case analysis
  - 2.1.3 a general design model
- 2.2 analyze a data processing problem and use a top-down design approach to transform a design model into a class diagram that represents the matrix of interacting classes required to solve the problem
- 2.3 describe and represent the relationship among the classes; e.g., inheritance, association, aggregation, composition
- 2.4 use an iterative and incremental approach to refine the architecture into appropriate class or object diagrams showing their relationships

- 2.5 analyze and refine the diagrams identifying the client/server relationship among the objects and determine the messages that need to be passed between objects and how the objects interface
- 2.6 draft an informal object message sequence indicating the flow of messages in the system
- 2.7 analyze and refine the object design by adding private methods, functions and data structures required to implement the various objects
- 2.8 test and modify, as required, the developing algorithm at each stage with appropriate data
- 3. translate algorithms into source code, convert the source code into machine executable form, execute and debug, as required**
  - 3.1 demonstrate the use iterative and incremental approaches in the implementation, testing and maintenance phases of the software development process
  - 3.2 demonstrate the use of iterative prototyping or a similar approach to break the algorithm into appropriate sections for implementation
  - 3.3 use original (user-created) or pre-existing classes, as necessary, to instantiate the objects called for by the algorithm using an iterative and incremental approach
  - 3.4 as the classes are constructed, use the server classes to create the client classes establishing the client/server relationships called for by the algorithm
  - 3.5 test and modify the sections as required
  - 3.6 where appropriate, collaborate with other students to carry out OOP tasks
  - 3.7 create internal and external documentation
  - 3.8 analyze the program and eliminate syntax, run-time and logic errors by using appropriate test data for each programming task at each stage of development
- 4. compare the results of the program with the intent of the algorithm and modify, as required**
- 5. demonstrate basic competencies**
  - 5.1 demonstrate fundamental skills to:
    - 5.1.1 communicate
    - 5.1.2 manage information
    - 5.1.3 use numbers
    - 5.1.4 think and solve problems
  - 5.2 demonstrate personal management skills to:
    - 5.2.1 demonstrate positive attitudes and behaviours
    - 5.2.2 be responsible
    - 5.2.3 be adaptable
    - 5.2.4 learn continuously
    - 5.2.5 work safely
  - 5.3 demonstrate teamwork skills to:
    - 5.3.1 work with others
    - 5.3.2 participate in projects and tasks
- 6. create a transitional strategy to accommodate personal changes and build personal values**
  - 6.1 identify short-term and long-term goals
  - 6.2 identify steps to achieve goals

## **COURSE CSE3140: SECOND LANGUAGE PROGRAMMING 2**

**Level:** Advanced

**Prerequisite:** CSE2120: Data Structures 1

**Description:** Designed for students who have mastered procedural programming and static data structures in a more accessible programming environment, this course gives students the opportunity to develop a similar skill set in a more demanding language.

**Parameters:** Access to appropriate computer equipment, software, the Internet and support materials. Access to a programming environment that allows structured and modular programming.

**Supporting Courses:** CSE3010: Computer Science 3  
CSE3110: Iterative Algorithms or any  
Advanced project course involving structured and modular programming

**Outcomes:** The student will:

### **1. compare and contrast a new language with a previously learned language**

- 1.1 consider the programming paradigms supported by each language including:
  - 1.1.1 list and describe the paradigms
  - 1.1.2 outline the advantages and disadvantages of the paradigms
- 1.2 consider the source code to machine code translation process used by each language including:
  - 1.2.1 identify and describe the process used by each language
  - 1.2.2 outline the advantages and disadvantages of translation processes in each language
- 1.3 consider the language characteristics including:
  - 1.3.1 language level; e.g., low, high, very high
  - 1.3.2 level of type; e.g., strongly typed, weakly typed
  - 1.3.3 nature of the source code; e.g., iconic, widgets, graphical
  - 1.3.4 difficulty to construct source code
  - 1.3.5 programming resources and aids
- 1.4 consider the modular characteristics of each language including:
  - 1.4.1 types of subprograms supported
  - 1.4.2 how modularity is supported
  - 1.4.3 level of module cohesion possible
  - 1.4.4 amount of module coupling required
  - 1.4.5 object-oriented features
- 1.5 other pertinent characteristics

### **2. demonstrate programming skills by writing algorithms for a second language that uses fundamental data structures**

- 2.1 use appropriate general design techniques for a specific programming environment
- 2.2 analyze and decompose the problem into appropriate subsections using techniques appropriate for the chosen design approach
- 2.3 evaluate subsections and identify any that may require some type of fundamental data structure, based on the nature of the data to be processed and the type of processing operations

- 2.4 identify, use or construct the appropriate data structure; e.g., array using appropriate variant or variants, where required
- 2.5 identify and sequence the operations required to process the data to be contained in the data structure
- 2.6 sequence the various subsections appropriately
- 2.7 create more detailed algorithms for each module identifying the pre- and post-conditions and required program control of flow mechanisms
- 2.8 test and modify the algorithm using appropriate “fail-on-paper” techniques
- 3. demonstrate coding skills by translating algorithms that require fundamental data structures into a second language**
  - 3.1 convert an algorithm into a program of linked subprograms with a main or client module calling other modules in a manner that reflects the structure of the algorithm
  - 3.2 use appropriate types of subprograms to implement the various sections of the algorithm
  - 3.3 analyze and determine, in a second language, the type of scope required to protect and/or hide data and keep implementation decoupled from the calling modules and to avoid unwanted side effects with consideration of the:
    - 3.3.1 use of appropriate parameters for importing and exporting data to and from the subprograms
    - 3.3.2 use of local variables and nested subprograms to enhance cohesion
  - 3.4 analyze for, and maintain, an appropriate balance between the coupling or dependency and cohesion or focus of the subprograms
  - 3.5 create internal and external documentation
  - 3.6 analyze the program and eliminate syntax, run-time and logic errors by using appropriate test data for each programming task at each stage of development
- 4. compare the results of the program with the intent of the algorithm and modify, as required**
- 5. demonstrate basic competencies**
  - 5.1 demonstrate fundamental skills to:
    - 5.1.1 communicate
    - 5.1.2 manage information
    - 5.1.3 use numbers
    - 5.1.4 think and solve problems
  - 5.2 demonstrate personal management skills to:
    - 5.2.1 demonstrate positive attitudes and behaviours
    - 5.2.2 be responsible
    - 5.2.3 be adaptable
    - 5.2.4 learn continuously
    - 5.2.5 work safely
  - 5.3 demonstrate teamwork skills to:
    - 5.3.1 work with others
    - 5.3.2 participate in projects and tasks
- 6. create a transitional strategy to accommodate personal changes and build personal values**
  - 6.1 identify short-term and long-term goals
  - 6.2 identify steps to achieve goals

## **COURSE CSE3210: SERVER-SIDE SCRIPTING 1**

**Level:** Advanced

**Prerequisites:** CSE2210: Client-side Scripting 3  
CSE2110: Procedural Programming 1  
CSE2120: Data Structures 1

**Description:** Students add to their ability to craft dynamic Web sites by exploring the fundamentals of server-side scripting. In the process, they add to their understanding of Internet scripting by employing databases as a repository for the information to be displayed by their sites. Students learn how to analyze, modify, write and debug algorithms and server-side scripts that use simple databases.

**Parameters:** Access to appropriate computer equipment, software, support materials and the Internet. Specifically, students must have access to a scripting environment that includes access to a Web server, a database management system and a server-side scripting language.

**Supporting Courses:** CSE3010: Computer Science 3  
CSE3110: Iterative Algorithm 1  
CSE3120: Object-oriented Programming 1 and/or any  
Advanced project course involving object-oriented programming

**Outcomes:** The student will:

- 1. demonstrate an understanding of the key features of server-side environments and server-side scripting**
  - 1.1 compare and contrast server-side scripting with dynamic client-side scripting including:
    - 1.1.1 outline the limitations of dynamic client-side scripting including security and quality of experience issues
    - 1.1.2 outline the advantages of server-side scripting including broader features available, developer control of application environment, greater interactivity, scalability and maintainability
    - 1.1.3 outline the disadvantages of server-side scripting
    - 1.1.4 describe how client-side and server-side scripting can and should be used to complement one another
    - 1.1.5 describe two Web applications made possible by server-side scripting approaches
    - 1.1.6 describe a server-side language and compare it with one or more client-side languages
  - 1.2 describe the system architecture of a typical server-side environment including:
    - 1.2.1 describe and represent the multitiered nature of a typical server-side environment
    - 1.2.2 describe the role played and capabilities needed by Web server software in a typical server-side environment
    - 1.2.3 describe the role played and capabilities needed by database and/or other information managers in a typical server-side environment
    - 1.2.4 describe the role played by the server-side scripting language in a typical server-side environment
    - 1.2.5 represent how a request from a client would be handled in a typical multitiered server-side environment

- 2. describe the processes and characteristics of a server-side environment**
  - 2.1 describe the various tiers that make up an environment including:
    - 2.1.1 identify the Web server software used in an environment and describe the role(s) it/they will be required to play
    - 2.1.2 identify the database manager and/or other information managers used in an environment and describe the role(s) it/they will be required to play
    - 2.1.3 identify the server-side scripting language(s) used in an environment and describe the role(s) it/they will be required to play
    - 2.1.4 identify and describe at least one significant task or application that a server-side environment could be used to carry out
- 3. use appropriate techniques to design applications for use in a server-side scripting environment**
  - 3.1 outline the intent of the application and determine if that intent can be advantageously realized through the use of a server-side environment including:
    - 3.1.1 determine the data handling requirements of the application, and determine if the application and information tiers of an environment are capable of handling those requirements
    - 3.1.2 determine the input/output requirements of the application, and determine if the client tier of an environment is capable of handling those requirements
  - 3.2 use an appropriate design technique to write the necessary algorithm(s) including:
    - 3.2.1 use an appropriate technique to represent the relationship among the modules
    - 3.2.2 write more detailed algorithms for each module and identify the pre- and post-conditions and program control of flow mechanisms required for any subprograms
  - 3.3 test the developing algorithm with appropriate data using a “fail-on-paper” process
  - 3.4 revise the algorithm, as required
- 4. write and debug the scripts required to implement a server-side application**
  - 4.1 demonstrate the ability to use an appropriate server-side scripting language environment
  - 4.2 convert the algorithms into scripts consisting of collaborating modules that reflect the structure of the algorithm including:
    - 4.2.1 use appropriate subprograms and/or objects to implement the various sections of the algorithm
    - 4.2.2 maintain an appropriate balance between the coupling or dependency and cohesion or focus of the modules
    - 4.2.3 pass data between the modules without unintended side-effects
    - 4.2.4 use internal and external documentation
  - 4.3 execute the script, and track and eradicate errors
  - 4.4 compare the results of the script’s execution with the intent of the algorithm and modify, as required
- 5. demonstrate basic competencies**
  - 5.1 demonstrate fundamental skills to:
    - 5.1.1 communicate
    - 5.1.2 manage information
    - 5.1.3 use numbers
    - 5.1.4 think and solve problems
  - 5.2 demonstrate personal management skills to:
    - 5.2.1 demonstrate positive attitudes and behaviours
    - 5.2.2 be responsible
    - 5.2.3 be adaptable
    - 5.2.4 learn continuously
    - 5.2.5 work safely

- 5.3 demonstrate teamwork skills to:
  - 5.3.1 work with others
  - 5.3.2 participate in projects and tasks
- 6. create a transitional strategy to accommodate personal changes and build personal values**
  - 6.1 identify short-term and long-term goals
  - 6.2 identify steps to achieve goals



## **COURSE CSE3240: ROBOTICS PROGRAMMING 3**

**Level:** Advanced

**Prerequisites:** CSE2240: Robotics Programming 2  
CSE2110: Procedural Programming 1

**Description:** Students continue their work in robotics programming by adding object-oriented programming (OOP) approaches to their skill set. In the process, they learn how to adapt their older procedure-based approaches to an object-oriented approach. They learn how to use object-oriented design approaches to design and write programs that use objects associated with one another in a client/server relationship.

**Parameters:** Access to appropriate computer equipment, software, support materials and the Internet. More specifically, students must have access to either the physical (real) or virtual (simulated) robotic environments they will require to design, write and debug Robot Control Language (RCL) scripts or programs.

**Supporting Courses:** CSE3010: Computer Science 3  
CSE3110: Iterative Algorithm 1  
CSE3120: Object-oriented Programming 1  
ELT3150: Robotics 3  
ELT3180: Robotics Vision Systems  
ELT3190: Robotics Kinematics & Behaviour  
ELT3200: Robotics Artificial Intelligence

**Outcomes:** The student will:

- 1. demonstrate an understanding of basic OOP approaches and how they can be used in robotics**
  - 1.1 demonstrate an understanding of:
    - 1.1.1 classes and objects that can be readily mapped to specific subsections of a robot's architecture
    - 1.1.2 class hierarchies to decompose complex robotic tasks into subtasks improving both the maintainability and extendibility of the programs
    - 1.1.3 potential code reuse both in the same and in other robotics programs
    - 1.1.4 the promotion of data hiding and information protection in robotics programs
    - 1.1.5 enhanced readability of robotics programs
    - 1.1.6 reduced side effect errors
- 2. demonstrate an understanding of basic OOP approaches and how they can be used to create class libraries**
  - 2.1 demonstrate how they:
    - 2.1.1 have the potential to improve design, coding and debugging efficiencies through the reduction of time spent and errors generated
    - 2.1.2 facilitate the development of higher levels of robotic artificial intelligence leading to programs that display greater agency and/or autonomy

3. **design a robotics system consisting of at least one robot, associated control systems and environment that uses OOP approaches to carry out a set of predetermined tasks**
  - 3.1 describe and diagram the environment in which the robot will be required to operate by:
    - 3.1.1 identifying the elements in the environment that are to be manipulated by the robot and determining their location
    - 3.1.2 identifying the elements in the environment to be detected by the robot's sensors and determining their location
    - 3.1.3 determining the type and location of internal barriers in the environment
    - 3.1.4 setting the outer limits of the environment
  - 3.2 identify the general tasks the robot will be required to carry out including:
    - 3.2.1 use an appropriate object-oriented approach to break those tasks into simpler tasks, grouping them into sets of related behaviours and sequencing those behaviours, where appropriate
    - 3.2.2 draft a conceptual model of the robot's behaviour that reflects the task behaviours the robot will be expected to carry out
  - 3.3 identify the capabilities the robot will require to carry out the tasks
  - 3.4 determine the control approach to be used including what level of operator control will be required if the robot cannot support a fully autonomous mode of operation
  - 3.5 design the robot using the tasks to be accomplished, proposed environment, required capabilities and control approach as parameters
  - 3.6 check your design for congruency against the task list to be accomplished and with the proposed environmental specifications
  - 3.7 modify the design, as required
  - 3.8 carry out the design process sequentially using the iterative and incremental approaches associated with object-oriented design and development
4. **use an OOP approach to build the environment, robot and controlling mechanism called for by the design**
  - 4.1 construct that portion of the environment required for the first task or tasks on the task sequence
  - 4.2 assemble as much of the robot, as is required, to accomplish those tasks
  - 4.3 write algorithms that use OOP approaches to outline how the first set of tasks is to be accomplished including:
    - 4.3.1 create a general design that reflects the robot's knowledge of the domain, allows for the acceptance of required data (sense), uses that data in conjunction with the domain model to arrive at decisions (plan) and provides the ability to act (action) to carry out the tasks
    - 4.3.2 use an incremental and stepwise approach to refine the design into a more concrete form such as a set of class or object diagrams showing the key objects and their relationship
    - 4.3.3 refine the design identifying the client/server relationship among the objects and determining the nature of the data or messages that need to be passed between objects
    - 4.3.4 design the objects' interface; e.g., public methods or functions to accommodate the data or messages that need to be passed between objects
    - 4.3.5 draft an informal sequence indicating the flow of messages in the system
    - 4.3.6 complete the object design by adding the private methods, functions and data structures required to implement the various objects
    - 4.3.7 use appropriate techniques to determine if the algorithm will achieve the original intent
  - 4.4 use an object-oriented RCL to translate the algorithms into a program including:
    - 4.4.1 use a technique such as iterative prototyping to break the algorithm or design into functional units or modules that can be encoded using object-oriented approaches
    - 4.4.2 deal with each section; in turn, create the classes necessary to instantiate the objects called for by the design using built-in and other available class libraries, where possible

- 4.4.3 establish the client/server relationships among the classes, as called for by the design, being sure to maintain appropriate access levels to ensure cooperation while preserving boundaries
- 4.4.4 pass data between the subprograms without unintended side effects and modify the sections, as required
- 4.4.5 use internal and external documentation
- 4.5 load and execute the program, and track and eradicate errors including:
  - 4.5.1 test each of the physical subsystems of the robot(s) to eliminate engineering errors
  - 4.5.2 test the robot(s) within the appropriate section of the environment to confirm that the robot is interacting with the environment as called for by the algorithm
  - 4.5.3 use self-test code and check points in each module, as well as observation, to eliminate run-time and internal logic errors
  - 4.5.4 compare the robot's actions with the intent of the algorithm
  - 4.5.5 modify the original task list, environment, algorithm and/or program, as required
- 4.6 participate in intermittent critiques throughout the iterative process; e.g., planning, analysis, design, testing, evaluation
- 5. demonstrate basic competencies**
  - 5.1 demonstrate fundamental skills to:
    - 5.1.1 communicate
    - 5.1.2 manage information
    - 5.1.3 use numbers
    - 5.1.4 think and solve problems
  - 5.2 demonstrate personal management skills to:
    - 5.2.1 demonstrate positive attitudes and behaviours
    - 5.2.2 be responsible
    - 5.2.3 be adaptable
    - 5.2.4 learn continuously
    - 5.2.5 work safely
  - 5.3 demonstrate teamwork skills to:
    - 5.3.1 work with others
    - 5.3.2 participate in projects and tasks
- 6. create a transitional strategy to accommodate personal changes and build personal values**
  - 6.1 identify short-term and long-term goals
  - 6.2 identify steps to achieve goals



## **COURSE CSE3310: RECURSIVE ALGORITHMS 1**

**Level:** Advanced

**Prerequisites:** CSE3110: Iterative Algorithm 1  
CSE3120: Object-oriented Programming 1

**Description:** Students learn how to use a new program control flow mechanism called recursion. They then use this mechanism to write a number of basic recursive algorithms and programs such as a recursive version of the binary search, the quicksort and the merge sort.

**Parameters:** Access to appropriate computer equipment, software, the Internet and support materials. Access to an object-oriented programming environment that encourages a formal treatment of objects.

**Supporting Courses:** CSE3020: Computer Science 4  
CSE3130: Object-oriented Programming 2

**Outcomes:** The student will:

- 1. analyze and represent the nature and utility of the recursive functions or procedures**
  - 1.1 explain and represent the key features of recursive algorithms including:
    - 1.1.1 illustrate how recursive algorithms define themselves in terms of themselves
    - 1.1.2 illustrate the use and purpose of the base case in recursion
  - 1.2 describe and represent the “divide and conquer” approach to creating recursive algorithms
  - 1.3 describe and represent the interchangeability of recursive and iterative operations
  - 1.4 compare and contrast recursion and iteration highlighting:
    - 1.4.1 programmer efficiency
    - 1.4.2 space efficiency
    - 1.4.3 time efficiency
  - 1.5 outline the importance of recursion in creating dynamic data structures
  - 1.6 compare and contrast tail end and head end recursion
  - 1.7 explain and represent how the system stack (or equivalent structure) is used to carry out recursive operations
- 2. analyze and represent the nature, structure and utility of recursive search and sort algorithms**
  - 2.1 describe at least four recursive algorithms used in dynamic data manipulation
  - 2.2 compare and contrast iterative and recursive approaches to binary searching by:
    - 2.2.1 describing and representing iterative and recursive binary search algorithms
    - 2.2.2 explaining the advantages and disadvantages of iterative and recursive approaches to binary searching
  - 2.3 compare and contrast at least two recursive sorts by:
    - 2.3.1 describing and representing the quicksort and the merge sort
    - 2.3.2 describing and representing the heapsort
    - 2.3.3 explaining the advantages and disadvantages of the quicksort, merge sort and heapsort

- 3. create and/or modify recursive algorithms to solve problems**
  - 3.1 demonstrate the use of appropriate general design techniques to draft algorithms that use recursion
  - 3.2 analyze and decompose the problem into appropriate subsections using the decomposition techniques appropriate for the chosen design approach
  - 3.3 evaluate subsections and identify any that may require a recursive approach
  - 3.4 identify which recursive algorithms are appropriate
  - 3.5 sequence the various subsections appropriately
  - 3.6 test and modify the developing algorithm with appropriate data using a “fail-on-paper” process
- 4. create and/or modify programs that use recursion**
  - 4.1 convert algorithms calling for recursive structures into programs that reflect the algorithm’s design
  - 4.2 use original (user-created) or pre-existing recursive merge and/or sort algorithms appropriate to the data being manipulated
  - 4.3 utilize the appropriate operators, methods, functions or procedures required to carry out the recursive algorithms
  - 4.4 use internal and external documentation
- 5. compare program operation and outcomes with the intent of the algorithm and modify, as required**
  - 5.1 use appropriate error-trapping mechanisms built into the programming environment, as well as programmer-directed error-trapping techniques, to eliminate logic errors and debug the program
  - 5.2 compare the congruency between the outcomes of the debugged program and the original intent of the algorithm and modify both, as required
- 6. demonstrate basic competencies**
  - 6.1 demonstrate fundamental skills to:
    - 6.1.1 communicate
    - 6.1.2 manage information
    - 6.1.3 use numbers
    - 6.1.4 think and solve problems
  - 6.2 demonstrate personal management skills to:
    - 6.2.1 demonstrate positive attitudes and behaviours
    - 6.2.2 be responsible
    - 6.2.3 be adaptable
    - 6.2.4 learn continuously
    - 6.2.5 work safely
  - 6.3 demonstrate teamwork skills to:
    - 6.3.1 work with others
    - 6.3.2 participate in projects and tasks
- 7. create a transitional strategy to accommodate personal changes and build personal values**
  - 7.1 identify short-term and long-term goals
  - 7.2 identify steps to achieve goals

**COURSE CSE3320: DYNAMIC DATA STRUCTURES 1**

**Level:** Advanced

**Prerequisite:** CSE3310: Recursive Algorithms 1

**Description:** Students learn how to design, code and debug programs using abstract data types that utilize dynamic data structures. Students explore dynamic memory allocation, in general, and as handled by their programming environment. Students concentrate on how the linked list dynamic data structure(s) can be used to implement abstract data types.

**Parameters:** Access to appropriate computer equipment, software, the Internet and support materials. Access to an object-oriented programming environment that encourages a formal treatment of objects.

**Supporting Courses:** CSE3020: Computer Science 4  
CSE3130: Object-oriented Programming 2

**Outcomes:** The student will:

**1. analyze and represent the nature, structure and utility of linked lists**

- 1.1 describe and represent the nature of dynamic data structures including:
  - 1.1.1 the mechanics of dynamic memory allocation; e.g., the heap, pointers and/or references, linear and non-linear data structures
  - 1.1.2 a comparison and contrast of dynamic and static data structures
- 1.2 describe and represent the nature and mechanics of linked lists by:
  - 1.2.1 describing linked lists as a components of abstract data types
  - 1.2.2 describing the logical structure of the singly linear linked list including nodes, fields, references and pointers
  - 1.2.3 describing the logical structure of other types of linked lists; e.g., double-linked, circularly-linked, ordered linked lists
- 1.3 describe and represent the operators associated with linked lists by:
  - 1.3.1 creating the linked list
  - 1.3.2 inserting a node
  - 1.3.3 traversing the linked list
  - 1.3.4 deleting a node
  - 1.3.5 replacing a node
  - 1.3.6 finding and retrieving data from the linked list
  - 1.3.7 determining the size of the linked list
- 1.4 explain the advantages of the linked list over static data structures

**2. create and/or modify algorithms that use linked lists to solve problems**

- 2.1 demonstrate the use of appropriate general design techniques to draft algorithms that use linked lists
- 2.2 analyze and decompose the problem into appropriate subsections using the decomposition techniques appropriate for the chosen design approach
- 2.3 evaluate subsections and identify any that may require some dynamic data structures, based on the nature of the data to be processed and type of processing operations
- 2.4 identify which dynamic data structures are appropriate or required to merge and/or sort data

- 2.5 sequence the various subsections appropriately
- 2.6 test and modify the developing algorithm with appropriate data using a “fail-on-paper” process
- 3. create and/or modify programs based on appropriate algorithms that make effective use of linked lists**
  - 3.1 convert algorithms calling for dynamic data structures into programs that reflect the algorithm’s design
  - 3.2 use original (user-created) or pre-existing dynamic data structures appropriate to the data being manipulated
  - 3.3 utilize the appropriate operators, methods, functions or procedures required to use dynamic data structures
  - 3.4 use internal and external documentation
- 4. compare program operation and outcomes with the intent of the algorithm and modify, as required**
  - 4.1 use appropriate error-trapping mechanisms built into the programming environment, as well as programmer-directed error-trapping techniques, to eliminate logic errors and debug the program
  - 4.2 compare the congruency between the outcomes of the debugged program and the original intent of the algorithm and modify both, as required
- 5. demonstrate basic competencies**
  - 5.1 demonstrate fundamental skills to:
    - 5.1.1 communicate
    - 5.1.2 manage information
    - 5.1.3 use numbers
    - 5.1.4 think and solve problems
  - 5.2 demonstrate personal management skills to:
    - 5.2.1 demonstrate positive attitudes and behaviours
    - 5.2.2 be responsible
    - 5.2.3 be adaptable
    - 5.2.4 learn continuously
    - 5.2.5 work safely
  - 5.3 demonstrate teamwork skills to:
    - 5.3.1 work with others
    - 5.3.2 participate in projects and tasks
- 6. create a transitional strategy to accommodate personal changes and build personal values**
  - 6.1 identify short-term and long-term goals
  - 6.2 identify steps to achieve goals

## **COURSE CSE3330: DYNAMIC DATA STRUCTURES 2**

**Level:** Advanced

**Prerequisite:** CSE3320: Dynamic Data Structures 1

**Description:** Students enhance their knowledge of abstract data types that utilize dynamic data structures by expanding their repertoire to include stacks and queues. Students also study the unordered data structures, set and map, and learn how to incorporate them into abstract data types. As part of this work, they learn how to use linked lists to create stacks and queues.

**Parameters:** Access to appropriate computer equipment, software, the Internet and support materials. Access to an object-oriented programming environment that encourages a formal treatment of objects.

**Supporting Courses:** CSE3020: Computer Science 4  
CSE3130: Object-oriented Programming 2 and/or any  
Advanced project course involving the use of abstract data types

**Outcomes:** The student will:

- 1. analyze and represent the nature, structure and utility of stacks, queues, sets and/or maps**
  - 1.1 explain and represent the nature and mechanics of stacks, queues, sets and/or maps including:
    - 1.1.1 the role of stacks, queues, sets and/or maps as containers for abstract data types (ADTs)
    - 1.1.2 the abstract data type and data manipulation each structure is best suited to
    - 1.1.3 the logical structure of stacks, queues, sets and/or maps
  - 1.2 explain and represent the standard operators associated with stacks, queues, sets and/or maps including:
    - 1.2.1 create the data structure
    - 1.2.2 copy the data structure; e.g., cloning, deep copy
    - 1.2.3 push, pop and peek for stacks
    - 1.2.4 enqueue and dequeue for queues
    - 1.2.5 link keys and values for maps
    - 1.2.6 search, insert, remove and modify data elements in sets and/or maps
    - 1.2.7 determine equality between sets
    - 1.2.8 determine union, intersection, difference and symmetric difference
    - 1.2.9 delete the data structure
  - 1.3 explain the advantages and disadvantages of using stacks, queues, sets and/or maps
- 2. create and/or modify algorithms using stacks, queues, sets and/or maps to solve problems**
  - 2.1 demonstrate the use of appropriate general design techniques to draft algorithms that use stacks, queues, sets and/or maps
  - 2.2 analyze and decompose the problem into appropriate subsections using the decomposition techniques appropriate for the chosen design approach
  - 2.3 evaluate subsections and identify any that may require stacks, queues, sets and/or maps, based on the nature of the data to be processed and type of processing operations
  - 2.4 identify which structures (stacks, queues, sets and/or maps) are appropriate or required to manipulate data

- 2.5 sequence the various subsections appropriately
- 2.6 test and modify the developing algorithm with appropriate data using a “fail-on-paper” process
- 3. create and/or modify programs based on appropriate algorithms using stacks, queues, sets and/or maps**
  - 3.1 convert algorithms calling for stacks, queues, sets and/or maps into programs that reflect the algorithm’s design
  - 3.2 use original (user-created) or pre-existing stacks, queues, sets and/or maps appropriate to the data being manipulated
  - 3.3 utilize the appropriate operators, methods, functions or procedures required to use stacks, queues, sets and/or maps
  - 3.4 use internal and external documentation
- 4. compare program operation and outcomes with the intent of the algorithm and modify, as required**
  - 4.1 use appropriate error-trapping mechanisms built into the programming environment, as well as programmer-directed error-trapping techniques, to eliminate logic errors and debug the program
  - 4.2 compare the congruency between the outcomes of the debugged program and the original intent of the algorithm and modify both, as required
- 5. demonstrate basic competencies**
  - 5.1 demonstrate fundamental skills to:
    - 5.1.1 communicate
    - 5.1.2 manage information
    - 5.1.3 use numbers
    - 5.1.4 think and solve problems
  - 5.2 demonstrate personal management skills to:
    - 5.2.1 demonstrate positive attitudes and behaviours
    - 5.2.2 be responsible
    - 5.2.3 be adaptable
    - 5.2.4 learn continuously
    - 5.2.5 work safely
  - 5.3 demonstrate teamwork skills to:
    - 5.3.1 work with others
    - 5.3.2 participate in projects and tasks
- 6. create a transitional strategy to accommodate personal changes and build personal values**
  - 6.1 identify short-term and long-term goals
  - 6.2 identify steps to achieve goals

## **COURSE CSE3340: DYNAMIC DATA STRUCTURES 3**

**Level:** Advanced

**Prerequisite:** CSE3330: Dynamic Data Structures 2

**Description:** Students enhance their knowledge of abstract data types that utilize dynamic data structures by expanding their repertoire to include hierarchically linked data structures. Students study general trees, binary trees, binary search trees and heaps. They learn how to use binary search trees to implement sorted sets, sorted maps and heaps to implement priority queues.

**Parameters:** Access to appropriate computer equipment, software, the Internet and support materials. Access to an object-oriented programming environment that encourages a formal treatment of objects.

**Supporting Courses:** CSE3020: Computer Science 4  
CSE3130: Object-oriented Programming 2 and/or any  
Advanced project course involving the use of abstract data types

**Outcomes:** The student will:

### **1. analyze and represent the nature, structure and utility of tree data structures**

- 1.1 explain and represent the nature and mechanics of tree data structures including:
  - 1.1.1 the role of tree data structures as containers for abstract data types
  - 1.1.2 the abstract data type each tree data structure is best suited to
  - 1.1.3 the logical structure of tree data structures; e.g., general trees, binary trees, binary search trees, heaps
- 1.2 explain and represent the standard operators associated with tree data structures including:
  - 1.2.1 create the data structure
  - 1.2.2 copy the data structure; e.g., cloning and deep copy
  - 1.2.3 preorder, in-order, post-order and level order traversals
  - 1.2.4 search, insert, remove and modify data elements
  - 1.2.5 list data elements accumulated by a tree transversal
  - 1.2.6 list the pop and heapify operation for heaps
  - 1.2.7 delete the data structure
- 1.3 explain the advantages and disadvantages of tree data structures with consideration to their complexity

### **2. create and/or modify algorithms that use various tree data structures to solve problems**

- 2.1 demonstrate the use of appropriate general design techniques to draft algorithms that use tree data structures
- 2.2 analyze and decompose the problem into appropriate subsections using the decomposition techniques appropriate for the chosen design approach
- 2.3 evaluate subsections and identify any that may require tree data structures, based on the nature of the data to be processed and type of processing operations
- 2.4 identify which tree data structures are appropriate or required to manipulate data
- 2.5 sequence the various subsections appropriately
- 2.6 test and modify the developing algorithm with appropriate data using a “fail-on-paper” process

- 3. create and/or modify programs based on appropriate algorithms that use various tree data structures**
  - 3.1 convert algorithms calling for tree data structures into programs that reflect the algorithm's design
  - 3.2 use original (user-created) or pre-existing tree data structures appropriate to the data being manipulated
  - 3.3 utilize the appropriate operators, methods, functions or procedures required to use tree data structures
  - 3.4 use internal and external documentation
- 4. compare program operation and outcomes with the intent of the algorithm and modify, as required**
  - 4.1 use appropriate error-trapping mechanisms built into the programming environment, as well as programmer-directed error-trapping techniques, to eliminate logic errors and debug the program
  - 4.2 compare the congruency between the outcomes of the debugged program and the original intent of the algorithm and modify both, as required
- 5. demonstrate basic competencies**
  - 5.1 demonstrate fundamental skills to:
    - 5.1.1 communicate
    - 5.1.2 manage information
    - 5.1.3 use numbers
    - 5.1.4 think and solve problems
  - 5.2 demonstrate personal management skills to:
    - 5.2.1 demonstrate positive attitudes and behaviours
    - 5.2.2 be responsible
    - 5.2.3 be adaptable
    - 5.2.4 learn continuously
    - 5.2.5 work safely
  - 5.3 demonstrate teamwork skills to:
    - 5.3.1 work with others
    - 5.3.2 participate in projects and tasks
- 6. create a transitional strategy to accommodate personal changes and build personal values**
  - 6.1 identify short-term and long-term goals
  - 6.2 identify steps to achieve goals

## **COURSE CSE3910: CSE PROJECT D**

**Level:** Advanced

**Prerequisite:** None

**Description:** Students develop project design and management skills to extend and enhance competencies and skills in other CTS courses through contexts that are personally relevant.

**Parameters:** Advanced project courses must connect with a minimum of two CTS courses, one of which must be at the advanced level and be in the same occupational area as the project course. The other CTS course(s) must be at least at the intermediate level from any occupational area.

Project courses cannot be connected to other project courses or practicum courses.

**All projects and/or performances, whether teacher- or student-led, must include a course outline or student proposal.**

### **Outcomes:**

The teacher/student will:

- 1. identify the connection between this project course and two or more CTS courses**
  - 1.1 identify the outcome(s) from each identified CTS course that support the project and/or performance deliverables
  - 1.2 explain how these outcomes are being connected to the project and/or performance deliverables
- 2. propose the project and/or performance**
  - 2.1 identify the project and/or performance by:
    - 2.1.1 preparing a plan
    - 2.1.2 clarifying the purposes
    - 2.1.3 defining the deliverables
    - 2.1.4 specifying time lines
    - 2.1.5 explaining terminology, tools and processes
    - 2.1.6 defining resources; e.g., materials, costs, staffing
  - 2.2 identify and comply with all related health and safety standards
  - 2.3 define assessment standards (indicators for success)
  - 2.4 present the proposal and obtain necessary approvals

The student will:

- 3. meet goals as defined within the plan**
  - 3.1 complete the project and/or performance as outlined
  - 3.2 monitor the project and/or performance and make necessary adjustments
  - 3.3 present the project and/or performance, indicating the:
    - 3.3.1 outcomes attained
    - 3.3.2 relationship of outcomes to goals originally set

- 3.4 evaluate the project and/or performance, indicating the:
  - 3.4.1 processes and strategies used
  - 3.4.2 recommendations on how the project and/or performance could have been improved
- 4. demonstrate basic competencies**
  - 4.1 demonstrate fundamental skills to:
    - 4.1.1 communicate
    - 4.1.2 manage information
    - 4.1.3 use numbers
    - 4.1.4 think and solve problems
  - 4.2 demonstrate personal management skills to:
    - 4.2.1 demonstrate positive attitudes and behaviours
    - 4.2.2 be responsible
    - 4.2.3 be adaptable
    - 4.2.4 learn continuously
    - 4.2.5 work safely
  - 4.3 demonstrate teamwork skills to:
    - 4.3.1 work with others
    - 4.3.2 participate in projects and tasks
- 5. create a transitional strategy to accommodate personal changes and build personal values**
  - 5.1 identify short-term and long-term goals
  - 5.2 identify steps to achieve goals

## **COURSE CSE3920: CSE PROJECT E**

**Level:** Advanced

**Prerequisite:** None

**Description:** Students develop project design and management skills to extend and enhance competencies and skills in other CTS courses through contexts that are personally relevant.

**Parameters:** Advanced project courses must connect with a minimum of two CTS courses, one of which must be at the advanced level and be in the same occupational area as the project course. The other CTS course(s) must be at least at the intermediate level from any occupational area.

Project courses cannot be connected to other project courses or practicum courses.

**All projects and/or performances, whether teacher- or student-led, must include a course outline or student proposal.**

### **Outcomes:**

The teacher/student will:

- 1. identify the connection between this project course and two or more CTS courses**
  - 1.1 identify the outcome(s) from each identified CTS course that support the project and/or performance deliverables
  - 1.2 explain how these outcomes are being connected to the project and/or performance deliverables
- 2. propose the project and/or performance**
  - 2.1 identify the project and/or performance by:
    - 2.1.1 preparing a plan
    - 2.1.2 clarifying the purposes
    - 2.1.3 defining the deliverables
    - 2.1.4 specifying time lines
    - 2.1.5 explaining terminology, tools and processes
    - 2.1.6 defining resources; e.g., materials, costs, staffing
  - 2.2 identify and comply with all related health and safety standards
  - 2.3 define assessment standards (indicators for success)
  - 2.4 present the proposal and obtain necessary approvals

The student will:

- 3. meet goals as defined within the plan**
  - 3.1 complete the project and/or performance as outlined
  - 3.2 monitor the project and/or performance and make necessary adjustments
  - 3.3 present the project and/or performance, indicating the:
    - 3.3.1 outcomes attained
    - 3.3.2 relationship of outcomes to goals originally set

- 3.4 evaluate the project and/or performance, indicating the:
  - 3.4.1 processes and strategies used
  - 3.4.2 recommendations on how the project and/or performance could have been improved
- 4. demonstrate basic competencies**
  - 4.1 demonstrate fundamental skills to:
    - 4.1.1 communicate
    - 4.1.2 manage information
    - 4.1.3 use numbers
    - 4.1.4 think and solve problems
  - 4.2 demonstrate personal management skills to:
    - 4.2.1 demonstrate positive attitudes and behaviours
    - 4.2.2 be responsible
    - 4.2.3 be adaptable
    - 4.2.4 learn continuously
    - 4.2.5 work safely
  - 4.3 demonstrate teamwork skills to:
    - 4.3.1 work with others
    - 4.3.2 participate in projects and tasks
- 5. create a transitional strategy to accommodate personal changes and build personal values**
  - 5.1 identify short-term and long-term goals
  - 5.2 identify steps to achieve goals

## **COURSE CSE3950: CSE ADVANCED PRACTICUM**

**Level:** Advanced

**Prerequisite:** None

**Description:** Students apply prior learning and demonstrate the attitudes, skills and knowledge required by an external organization to achieve a credential/credentials or an articulation.

**Parameters:** This practicum course, which may be delivered on- or off-campus, should be accessed only by students continuing to work toward attaining a recognized credential/credentials or an articulation offered by an external organization. This course must be connected to at least one CTS course from the same occupational area and cannot be used in conjunction with any introductory (1XXX) level course. A practicum course cannot be delivered as a stand-alone course, cannot be combined with a CTS project course and cannot be used in conjunction with the Registered Apprenticeship Program or the Green Certificate Program.

**Outcomes:** The student will:

- 1. perform assigned tasks and responsibilities, as required by the organization granting the credential(s) or articulation**
  - 1.1 identify regulations and regulatory bodies related to the credential(s) or articulation
  - 1.2 describe personal roles and responsibilities, including:
    - 1.2.1 key responsibilities
    - 1.2.2 support functions/responsibilities/expectations
    - 1.2.3 code of ethics and/or conduct
  - 1.3 describe personal work responsibilities and categorize them as:
    - 1.3.1 routine tasks; e.g., daily, weekly, monthly, yearly
    - 1.3.2 non-routine tasks; e.g., emergencies
    - 1.3.3 tasks requiring personal judgement
    - 1.3.4 tasks requiring approval of a supervisor
  - 1.4 demonstrate basic employability skills and perform assigned tasks and responsibilities related to the credential(s) or articulation
- 2. analyze personal performance in relation to established standards**
  - 2.1 evaluate application of the attitudes, skills and knowledge developed in related CTS courses
  - 2.2 evaluate standards of performance in terms of:
    - 2.2.1 quality of work
    - 2.2.2 quantity of work
  - 2.3 evaluate adherence to workplace legislation related to health and safety
  - 2.4 evaluate the performance requirements of an individual who is trained, experienced and employed in a related occupation in terms of:
    - 2.4.1 training and certification
    - 2.4.2 interpersonal skills
    - 2.4.3 technical skills
    - 2.4.4 ethics

**3. demonstrate basic competencies**

3.1 demonstrate fundamental skills to:

- 3.1.1 communicate
- 3.1.2 manage information
- 3.1.3 use numbers
- 3.1.4 think and solve problems

3.2 demonstrate personal management skills to:

- 3.2.1 demonstrate positive attitudes and behaviours
- 3.2.2 be responsible
- 3.2.3 be adaptable
- 3.2.4 learn continuously
- 3.2.5 work safely

3.3 demonstrate teamwork skills to:

- 3.3.1 work with others
- 3.3.2 participate in projects and tasks

**4. create a transitional strategy to accommodate personal changes and build personal values**

4.1 identify short-term and long-term goals

4.2 identify steps to achieve goals