COURSE CSE1010: COMPUTER SCIENCE 1

Level: Introductory

Prerequisite: None

Description: Students explore hardware, software and processes. This includes an introduction to the algorithm as a problem-solving tool, to programming languages in general and to the role of programming as a tool for implementing algorithms.

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

Supporting Courses: CSE1110: Structured Programming 1
CSE1120: Structured Programming 2, and/or any
Intermediate project course involving imperative programming

Outcomes: The student will:

1. **identify and describe the nature, approaches and areas of interest of computer science**
   - 1.1 define and describe computer science with consideration of:
     - 1.1.1 the main goal of the discipline
     - 1.1.2 the use of algorithms
     - 1.1.3 computer systems used to test and/or implement algorithms
     - 1.1.4 the translation of algorithms through programming
   - 1.2 describe the general areas of interest of computer science including:
     - 1.2.1 the theory of computation
     - 1.2.2 algorithms and data structures
     - 1.2.3 programming methodology and languages
     - 1.2.4 computer elements and architecture
     - 1.2.5 human–machine and machine–machine interfacing
     - 1.2.6 automata
     - 1.2.7 artificial intelligence
     - 1.2.8 visual and auditory rendering
     - 1.2.9 general development of information technology applications
   - 1.3 compare and contrast computer science, computer engineering and information technology; e.g., theoretical versus applied, general versus specific, exploratory versus applicatory
   - 1.4 describe some of the misconceptions associated with computer science; e.g., synonymous with programming, reliant on solitary individuals for the bulk of its advances, relatively little real-world contact, the learning of various computer applications
   - 1.5 computer science's role in an information society
2. **demonstrate an understanding of the nature, design and use of basic algorithms associated with problems involving the sequential inputting, processing and outputting of data**
   - 2.1 define algorithms and explain how they are used
   - 2.2 compare and contrast the "iterative and incremental" and "waterfall" models of software development

2.3 demonstrate the analysis and design stages of a Systems Development Life Cycle model using appropriate tools; e.g., flowcharts, pseudocode, input/processing/output (IPO) charting
2.4 demonstrate a number of core algorithms including:
2.4.1 accumulation (keeping a running total)
2.4.2 determining the mean
2.4.3 determining minimums and maximums

3. explain and demonstrate the nature of structured programming
3.1 consider the rationale for structured programming
3.2 consider GOTO-less programming
3.3 consider three fundamental control structures—sequential, decision and iterative

4. explain and demonstrate an understanding of the nature, evolution, types and role of programming languages
4.1 describe how various programming languages have dealt with data representation; e.g., binary and hexadecimal systems, standard data types, data storage
4.2 describe the nature of programming language, specifically that these languages:
4.2.1 reflect a simplified version of natural language
4.2.2 evolved in tandem with algorithms and hardware over a number of generations
4.2.3 reflect the IPO data processing paradigm
4.3 describe and demonstrate how programming languages are used in the coding stage of a Systems Development Life Cycle model by converting a representative set of algorithms into executable code

5. explain the nature, evolution and basic architecture of a von Neumann computer system
5.1 create a block diagram of a stereotypical von Neumann machine
5.2 describe a number of typical devices associated with each block
5.3 show the flow of data through the computer under the direction of a program

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. make personal connections to the cluster content and processes to inform possible pathway choices
7.1 complete/update a personal inventory; e.g., interests, values, beliefs, resources, prior learning and experiences
7.2 create a connection between a personal inventory and occupational choices

COURSE CSE1110: STRUCTURED PROGRAMMING 1

Level: Introductory

Prerequisite: None

Description: Students are introduced to a general programming environment in which they write simple structured algorithms and programs that input, process and output data, use some of the more basic operators and data types, and follow a sequential flow of control.

Parameters: Access to appropriate computer equipment, software, the Internet and support materials. Specifically, students must have access to a programming environment that encourages structured programming.

Supporting Course: CSE1010: Computer Science 1

Outcomes: The student will:

1. **demonstrate introductory structured programming skills by writing sequential algorithms to solve problems involving input, processing and output**
   - 1.1 describe the purpose and nature of an algorithm
   - 1.2 analyze a variety of simple algorithms and describe the task or tasks the algorithms are attempting to carry out
   - 1.3 analyze problems and determine if they can be solved using algorithms that employ an input/processing/output (IPO) approach
   - 1.4 decompose the problem into its input, processing and output components, and identify what data is already available to the program and what must be inputted
   - 1.5 sequence components appropriately so that processing occurs only when all required data is available and output occurs only after appropriate processing has occurred
   - 1.6 write the algorithm in an acceptable format; e.g., pseudocode, structured chart
   - 1.7 test the algorithm for failure as well as success with appropriate data
   - 1.8 revise the algorithm, as required
2. **translate algorithms into source code, convert the source code into machine executable form, execute and debug, as required**
   - 2.1 describe a typical programming development environment commenting on the role of the key components; e.g., the source code editor, code translator (compiler and/or interpreter), executor, debugger
   - 2.2 compare and contrast integrated development environments specifically developed for programming with user assembled collections of applications and system software; e.g., text processor for coding, command line compiler
   - 2.3 describe and demonstrate the use of key components in a programming development environment
   - 2.4 convert algorithms into a sequence of statements in an appropriate programming language being sure to:
     - 2.4.1 maintain the IPO structure of the algorithm
     - 2.4.2 use appropriate internal and external documentation
     - 2.4.3 use appropriate data types such as integers, real numbers, characters and strings
     - 2.4.4 use appropriate variables and constants to hold data

2.4.5 use literals and input commands, e.g., methods or operators, to provide data for processing
2.4.6 use assignment, arithmetical and concatenation and interpolation operators, where appropriate, to process data
2.4.7 use output commands; e.g., methods or operators, to display processed data
2.5 test the algorithm for failure or success with appropriate data
2.6 revise the algorithm, as required

3. analyze and compare the results of the program with the intent of the algorithm and modify as required

3.1 use appropriate test data and debugging techniques to track and correct errors including:
3.1.1 run-time errors; e.g., compiler, linker, syntax
3.1.2 logic errors

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. make personal connections to the cluster content and processes to inform possible pathway choices

5.1 complete/update a personal inventory; e.g., interests, values, beliefs, resources, prior learning and experiences
5.2 create a connection between a personal inventory and occupational choices

COURSE CSE1120: STRUCTURED PROGRAMMING 2

Level: Introductory

Prerequisite: CSE1110: Structured Programming 1

Description: Students work with structured programming constructs by adding the selection and iteration program control flow mechanisms to their programming repertoire. They write structured algorithms and programs that use blocks to introduce an element of modularity into their programming practice.

Parameters: Access to appropriate computer equipment, software, to the Internet and support materials. Specifically, students must have access to a programming environment that encourages structured programming.

Supporting Courses: CSE1010: Computer Science 1, or any
Intermediate project course involving imperative programming

Outcomes: The student will:

1. **demonstrate basic structured programming skills by writing algorithms to solve problems involving selection (decision making) and iteration (repetition)**
   - 1.1 analyze a problem and determine if it can be solved using an algorithm that employs an input/processing/output (IPO) approach
   - 1.2 determine if there is more than one IPO module present
   - 1.3 decompose the problem into its respective modules and identify the IPO components of each module
   - 1.4 identify what data is already available to the programmer and what must be inputted by the end user and organize into appropriate block or blocks using the appropriate program control structures
   - 1.5 identify the processing requirements and organize into appropriate blocks using the appropriate program control structures
   - 1.6 incorporate basic algorithmic idioms as required; e.g., accumulation, determining maximum or minimum values
   - 1.7 identify the output requirements and organize into appropriate blocks using the appropriate program control structures
   - 1.8 order components into an appropriate sequence where processing occurs only when all required data for a module is available and output occurs only after appropriate processing has occurred
   - 1.9 write the algorithm in an acceptable format; e.g., pseudocode, a structured chart
2. **translate algorithms into source code, convert the source code into machine executable form, execute and debug, as required**
   - 2.1 maintain the IPO structure of the algorithm
   - 2.2 use appropriate internal and external documentation
   - 2.3 use appropriate basic (primitive) data types such as integers, real numbers, characters, strings, and Boolean values
   - 2.4 use appropriate variables and constants to hold data
   - 2.5 use literals and input commands, e.g., methods or operators, to provide data for processing

2.6 use assignment, arithmetical, relational, Boolean, and concatenation and interpolation operators, where appropriate, to process data
2.7 use basic processing idioms as required; e.g., accumulation, determining maximum or minimum values
2.8 use appropriate selection and iteration structures to avoid unconditional branching or exiting from the interior of a block including:
  2.8.1 nested conditional blocks
  2.8.2 nested iterative blocks
2.9 use output commands, e.g., methods or operators, to display processed data in an appropriately formatted form

3. analyze and compare the results of the program with the intent of the algorithm and modify, as required

3.1 use appropriate test data and debugging techniques to track and correct errors including:
  3.1.1 run-time errors; e.g., compiler, linker, syntax
  3.1.2 logic errors

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. make personal connections to the cluster content and processes to inform possible pathway choices

5.1 complete/update a personal inventory; e.g., interests, values, beliefs, resources, prior learning and experiences
5.2 create a connection between a personal inventory and occupational choices

COURSE CSE1210: CLIENT-SIDE SCRIPTING 1

Level: Introductory

Prerequisite: None

Description: Students are introduced to Internet computing through the use of one or more Web-specific markup languages. As part of this process, students learn how the Web uses markup languages to provide a client-side approach to display static information. Students also learn how to analyze, modify, write and debug algorithms and documents that use a markup language.

Parameters: Access to appropriate computer equipment, software, support materials and the Internet. More specifically, students must have the tools they will require to design, write and debug markup language-based hypermedia documents.

Supporting Courses: CSE1010: Computer Science 1
CSE1110: Structured Programming 1

Outcomes: The student will:

1. demonstrate an understanding of the general architecture of the Internet as it pertains to client-side scripting

1.1 explain and demonstrate the client/server nature of the Internet including:
- 1.1.1 describe the internetworked nature of the Internet
- 1.1.2 illustrate the client/server relationship that exists among the work stations, stub networks, mid-level networks, regional networks and the backbone that makes up the Internet
- 1.1.3 describe and illustrate how servers, routers, switches, work stations and other hardware components are used to provide the physical matrix required for client/server relationships
- 1.1.4 describe and illustrate the Internet Protocol Suite (TCP/IP) model of networking, in general terms, outlining how this protocol provides the data transfer mechanism required to establish client/server relationships
- 1.1.5 illustrate the client/server relationships set up when a user makes a request for an Internet service and that request is carried out
- 1.1.6 describe at least three examples of Internet services that rely on client/server relationships

1.2 explain and demonstrate the client/server nature of the Web including:
- 1.2.1 describe the hypertext-based nature of the Web
- 1.2.2 describe the relationship between the Web and the Internet as a whole
- 1.2.3 explain why the Web can be thought of as a network of hyperlinked documents
- 1.2.4 describe and illustrate the client/server relationship that allows user agents to interact with the origin servers that make up the information repository components of the Web
- 1.2.5 describe how the Hypertext Transfer Protocol (HTTP) is used to facilitate client/server interaction
- 1.2.6 describe and illustrate the general flow of information through the Internet when a user agent uses a Web browser to interact with origin servers
- 1.2.7 explain how HTTP is used to protect the transmission of data through the Web
- 1.2.8 describe and illustrate the development of the Web in general terms using the Web 1.0, Web 2.0 and Web 3.0 generational paradigms
- 1.2.9 compare and contrast the Web 1.0, Web 2.0 and Web 3.0 stages of development

2. **demonstrate an understanding of the general nature and purpose of Internet-oriented markup languages**
   - 2.1 describe the role markup languages play in the Web
   - 2.2 compare and contrast markup and scripting languages
   - 2.3 describe and illustrate the development of Internet-oriented markup languages in general terms including:
     - 2.3.1 explain the relationship between Standard Generalized Markup Language (SGML), Extensible Markup Language (XML), Hypertext Markup Language (HTML), Extensible Hypertext Markup Language (XHTML) and Dynamic Hypertext Markup Language (DHTML)
     - 2.3.2 explain at least two specialized Internet markup languages
3. **design, write and debug code using an appropriate Internet markup language**
   - 3.1 demonstrate the ability to use an appropriate markup language coding environment
   - 3.2 use appropriate techniques to design a markup language document including:
     - 3.2.1 determine and outline the intent of the document
     - 3.2.2 organize the document into appropriate subsections or pages
     - 3.2.3 describe the content to be carried on each page
     - 3.2.4 illustrate how the content is to be displayed
     - 3.2.5 identify the locations of the required anchors and links
   - 3.3 translate design documents into hypertext documents using code elements such as tags, attributes and hyperlinks to:
     - 3.3.1 mark off the various parts of the document
     - 3.3.2 display text and visual data in a variety of formats
     - 3.3.3 create specialized formats such as lists, tables and frames
     - 3.3.4 create both textual and image-based hyperlinks; e.g., both single images and mapped images
   - 3.4 compare the results of the script with the intent of the design document and modify, as required, including:
     - 3.4.1 use appropriate debugging techniques to compare the original design with the implemented document
     - 3.4.2 make changes, as required, to either the design and/or the document to bring both in line with the original intent
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. **make personal connections to the cluster content and processes to inform possible pathway choices**
   - 5.1 complete/update a personal inventory; e.g., interests, values, beliefs, resources, prior learning and experiences
   - 5.2 create a connection between a personal inventory and occupational choices

COURSE CSE1220: CLIENT-SIDE SCRIPTING 2

Level: Introductory

Prerequisite: None

Note: CSE1210: Client-side Scripting 1 or an equivalent course dealing with markup scripting is strongly recommended

Description: Students deepen their understanding of Internet computing by using more advanced markup language techniques and by being introduced to one or more Web-specific scripting languages. As part of this process, students learn how the Web uses these resources as a means of displaying dynamic client-side information. Students learn how to analyze, modify, write and debug algorithms and scripts that use structured programming approaches.

Parameters: Access to appropriate computer equipment, software, support materials and the Internet. More specifically, students must have the tools they will require to design, write and debug hypermedia documents and Internet scripts.

Supporting Courses: CSE1210: Client-side Scripting 1
CSE1110: Structured Programming 1
CSE1120: Structured Programming 2

Outcomes: The student will:

1. **compare and contrast static and dynamic client-side scripting**
   - 1.1 describe and illustrate the main differences, from the user's perspective, between dynamic and static client-side Web sites
   - 1.2 describe and illustrate the main differences between how dynamic and static client-side sites are implemented by the Web
   - 1.3 describe and illustrate the main advantages and disadvantages of dynamic and static client-side sites
2. **design, write and debug scripts that use advanced markup language approaches to provide some aspects of dynamic client-side site construction**
   - 2.1 describe how markup languages can be used to provide a limited amount of client-side dynamic display through the use of forms and style sheets
   - 2.2 use appropriate techniques to design a hypertext document that employs forms and style sheets including:
     - 2.2.1 determine and outline the intent of the document
     - 2.2.2 use appropriate problem decomposition techniques to organize the document into smaller components or pages and to identify locations of any required anchors and links
     - 2.2.3 determine the content to be carried on each page and how that content is to be displayed
     - 2.2.4 identify what role style sheets have in controlling page display
     - 2.2.5 determine what role forms have in managing input and subsequent interaction
   - 2.3 write and debug scripts that translate design documents employing forms and style sheets into client-side sites by:
     - 2.3.1 identifying and using appropriate techniques for separating content and presentation through the use of inline and/or embedded style sheets
     - 2.3.2 identifying and using appropriate techniques for soliciting user input through the use of forms

2.3.3 using appropriate techniques to determine if the script will achieve the original intent
2.3.4 using appropriate internal and external documentation

3. describe the general nature and purpose of Internet-oriented scripting languages

3.1 describe the role scripting languages play in the creation of Web sites
3.2 compare and contrast scripting languages with markup languages and with general purpose programming languages
3.3 describe and illustrate the development of Internet-oriented scripting languages in general terms
3.4 describe and illustrate at least two specialized Internet-oriented scripting languages; e.g., Javascript, PERL

4. design, write and debug scripts that use structured programming approaches with an appropriate Internet-oriented scripting language

4.1 demonstrate the ability to use an appropriate scripting language coding environment
4.2 outline the intent of the script and determine if the intent can be realized using structured programming approaches
4.3 write algorithms that use structured programming approaches to realize the intent of the script including:
4.3.1 use appropriate problem decomposition techniques to break the problem into smaller components
4.3.2 identify the input, processing and output requirements of each component
4.3.3 further decompose each component into smaller blocks, as required, using the appropriate structures to control program flow
4.3.4 write the algorithm in an acceptable format
4.3.5 use appropriate techniques to determine if the algorithm will achieve the original intent
4.4 translate the algorithm into a script using structured programming approaches by:
4.4.1 maintaining the structure of the algorithm
4.4.2 using appropriate basic or primitive data types
4.4.3 using appropriate variables and constants, as required, to hold data
4.4.4 using literals and input commands to provide data for processing
4.4.5 using assignment, arithmetical, relational, Boolean and, where available, concatenation and string construction operators to process data
4.4.6 using basic processing idioms such as accumulation, determining maximum or minimum values, as required
4.4.7 using appropriate selection and iteration structures such as conditional and iterative blocks
4.4.8 using output commands to display processed data in an appropriately formatted form
4.4.9 using appropriate internal and external documentation
4.5 execute the script tracking and eradicating errors including:
4.5.1 embed the script in an appropriate markup document
4.5.2 eliminate run-time and logic errors
4.6 compare the results of the script's execution with the intents 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. make personal connections to the cluster content and processes to inform possible pathway choices

6.1 complete/update a personal inventory; e.g., interests, values, beliefs, resources, prior learning and experiences
6.2 create a connection between a personal inventory and occupational choices

COURSE CSE1240: ROBOTICS PROGRAMMING 1

Level: Introductory

Prerequisite: CSE1110: Structured Programming 1

Description: Students use an appropriate Robot Control Language (RCL) to design, develop, implement and debug robotics programs that employ standard structured programming constructs and simple data structures. In the process, they develop a general understanding of robots and the robotics environment.

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 environment they will require to design, write and debug RCL scripts or programs.

Supporting Courses: CSE1010: Computer Science 1
CSE1120: Structured Programming 2
ELT1130: Robotics 1
ELT1140: Robotics Applications 1

Outcomes: The student will:

1. **demonstrate a general understanding of robotics including the nature of robots, robotic control systems and the robotic environment**
   - 1.1 describe robots in general terms including:
     - 1.1.1 build a general mental model of robotic architecture by outlining the characteristics that are common to all robots
     - 1.1.2 describe the robotic characteristic of "agency", the appearance of operating as autonomous agents and indicating the central role it plays in the concept of robot
     - 1.1.3 indicate how robots interact with their environment
   - 1.2 describe robotic control systems in general terms including:
     - 1.2.1 describe the three robotic primitives associated with robotic behaviour and their role in robotic control
     - 1.2.2 describe the three main approaches to robotic control
     - 1.2.3 use the three robotic primitives to describe how each approach to robotic control is carried out
   - 1.3 describe robotic environments in general terms including:
     - 1.3.1 describe the delimited nature of the robotic environment
     - 1.3.2 explain why robotic environments are circumscribed or limited in comparison with human environments
     - 1.3.3 describe the ideal relationship between a robot and its environment
     - 1.3.4 describe the issues associated with a mixed human/robot environment
   - 1.4 compare and contrast virtual (simulated) robots and physical (real) robots in general terms including:
     - 1.4.1 compare and contrast their architecture
     - 1.4.2 compare and contrast their control systems
     - 1.4.3 compare and contrast their environment

2. use a general understanding of robotics to analyze a robot operating within its environment

2.1 describe its architecture
2.2 indicate how it displays “agency” or autonomous action
2.3 categorize its control system
2.4 describe its capabilities
2.5 describe its relationship with its environment
2.6 identify at least one task that the robot should be able to accomplish within its environment
2.7 explain how either the robot and/or its environment could be modified to increase the number and type of tasks it could accomplish

3. design a robotics system consisting of at least one robot, associated control systems and environment capable of carrying out a simple set of predetermined tasks

3.1 identify the general tasks the robot will be required to carry out by:
- 3.1.1 breaking tasks into simpler tasks and continuing the process until the tasks are reduced to primitives
- 3.1.2 establishing the task sequence and creating a representation of those tasks

3.2 describe and illustrate the environment in which the robot will be required to operate by:
- 3.2.1 identifying elements of the environment that the robot will be able to manipulate
- 3.2.2 identifying manipulatable elements that will act as task resources and task obstacles
- 3.2.3 identifying variations in the environment that the robot will be able to detect; e.g., light, colour, sound
- 3.2.4 setting the outer limits of the environment
- 3.2.5 determining the location and type of internal barriers in the environment
- 3.2.6 incorporating safety elements into the environment that will protect both the humans and robots operating in the environment

3.3 identify the capabilities the robot will require to carry out set tasks including:
- 3.3.1 sensing requirements
- 3.3.2 mobility requirements
- 3.3.3 manipulation requirements
- 3.3.4 power requirements
- 3.3.5 processing requirements; e.g., both calculation and data storage

3.4 determine the control approach to be used to:
- 3.4.1 determine if the robot has the capacity for autonomous operation
- 3.4.2 determine what level of operator control will be required if the robot is not fully autonomous

3.5 design the robot using the tasks to be accomplished, proposed environment, required capabilities, and control approach as parameters to determine what:
- 3.5.1 type of kinematic chain or body will be required to provide a platform for the other components
- 3.5.2 actuators and end effectors will be required and how they will be mounted on the body
- 3.5.3 sensors will be required and how they will be mounted on the body
- 3.5.4 control components will be required and how they will be mounted on the body
- 3.5.5 power or energy components will be required and how they will be mounted on the body

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 a top-down approach and employing stepwise refinement

4. use an iterative process 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 in the task sequence
4.2 assemble as much of the robot, as is required, to accomplish the task or tasks

4.3 write algorithms that use structured programming approaches to accomplish the task or set of tasks including:
- 4.3.1 use appropriate problem decomposition techniques to break the task into subtasks
- 4.3.2 identify the sense, plan and action requirements of each subtask
- 4.3.3 further decompose each subtask into smaller blocks, as required, using the appropriate structures to control program flow
- 4.3.4 write the algorithm in an approved format such as a structured chart or pseudocode
- 4.3.5 use appropriate techniques to determine if the algorithm will achieve the original intent

4.4 use an RCL capable of writing structured code to translate the algorithm for the set of tasks into a program including:
- 4.4.1 maintain the structure of the algorithm
- 4.4.2 use appropriate basic or primitive data types and variable and constant names, as required, to hold data
- 4.4.3 use literals and input commands to accept data from sensors to provide data for processing
- 4.4.4 use operators and basic processing idioms as required; e.g., accumulation, determination of maximum or minimum values
- 4.4.5 use appropriate selection and iteration structure; e.g., conditional, iterative blocks
- 4.4.6 use output commands to display processed data to the operator as well as activate actuators; e.g., motors, grippers
- 4.4.7 document appropriately

4.5 load and execute the program tracking and eradicating errors by:
- 4.5.1 testing each of the physical subsystems of the robot(s) to eliminate engineering errors
- 4.5.2 testing 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 using self-test code and check points, as well as observation, to eliminate run-time and internal logic errors
- 4.5.4 comparing the robot's actions with the intent of the algorithm
- 4.5.5 modifying the original task list, environment, algorithm and/or program, as required

4.6 participate in interim 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. make personal connections to the cluster content and processes to inform possible pathway choices

6.1 complete/update a personal inventory; e.g., interests, values, beliefs, resources, prior learning and experiences

6.2 create a connection between a personal inventory and occupational choices

COURSE CSE1910: CSE PROJECT A

Level: Introductory

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: Introductory project courses must connect with a minimum of two CTS courses, one of which must be at the introductory level and be in the same occupational area as the project course. The other CTS course(s) can be either at the same level or 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. make personal connections to the cluster content and processes to inform possible pathway choices

5.1 complete/update a personal inventory; e.g., interests, values, beliefs, resources, prior learning and experiences
5.2 create a connection between a personal inventory and occupational choices

COURSE CSE2010: COMPUTER SCIENCE 2

Level: Intermediate

Prerequisites: CSE1010: Computer Science 1
CSE1120: Structured Programming 2

Description: Students explore hardware, software and processes at an intermediate level. Students extend their understanding of software development by learning how to layer modular programming approaches over structured programming techniques to improve the efficiency and robustness of algorithms and programs. They also are introduced to derived data types to provide them with data structures suitable for more demanding problems. Students add to their understanding of the hardware side of computer science by exploring a stylized von Neumann computer system at the machine level, and of the social side of computer science by examining some of the issues that have arisen from the implementation of computer technology.

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

Supporting Courses: CSE2110: Procedural Programming 1
CSE2120: Data Structures 1
CSE2130: Files & File Structures 1 and/or any
Intermediate or advanced project course involving modular programming

Outcomes: The student will:

1. identify and describe past, present and potential developments in computer technology

1.1 analyze and explain the qualitative trends of the application of computer technology over time particularly the shift in focus, in the recent past, from traditional computation, information warehousing (databases), and automation and cybernetics to the present focus on communication, social and commercial networking, entertainment and artificial intelligence, and to a future focus on bionics and cyborganization and artificial life

1.2 analyze and explain the quantitative trends in the application of computer technology over time including the expansion, in the recent past, from the military, scientific community, government, and large and medium-sized institutions, to the present expansion into small institutions, the home, industrial and domestic machines and personal information managers, and to the projected expansion into personal expert systems, implanted systems and artificial life

1.3 identify and describe areas of ethical and moral concerns arising from the permeation of computer technology in society including:

1.3.1 privacy issues; e.g., data mining and database consolidation, tracking of financial transactions, e-mail and other communications monitoring

1.3.2 security issues; e.g., identity and information theft

1.3.3 equality issues; e.g., emergence of the “digital divide”

1.3.4 freedom issues; e.g., privatization of information and control of information flow

2. explain and demonstrate the nature, evolution and key approaches associated with the modular programming paradigm

- 2.1 demonstrate iterative and incremental approaches to the analysis and design stages of the software development process
- 2.2 demonstrate the analysis step of an appropriate Systems Development Life Cycle (SDLC) using modular approaches including:
  - 2.2.1 problem parsing and decomposition
  - 2.2.2 identification of subtasks
  - 2.2.3 data structuring
  - 2.2.4 operation identification
- 2.3 demonstrate the design step of an appropriate SDLC using modular approaches including:
  - 2.3.1 top-down design
  - 2.3.2 stepwise refinement
  - 2.3.3 scope considerations with an emphasis on avoiding global data
  - 2.3.4 modular implementation
  - 2.3.5 appropriate coupling approaches
  - 2.3.6 appropriate levels of cohesion
  - 2.3.7 reusable modules and submodules
  - 2.3.8 data dictionaries, where required
  - 2.3.9 bottom-up coding, where appropriate

3. explain and demonstrate the conversion of general modular algorithms into modular programs through the use of subprograms, procedural abstraction and the use of local scope to protect data, and other tools

- 3.1 explain the following:
  - 3.1.1 hierarchy plus input/process/output (HIPO) charting
  - 3.1.2 structure diagrams
  - 3.1.3 Warnier/Orr diagrams

4. development, structure and use of key algorithms associated with modular approaches and the application of these idioms to create more complex algorithms

- 4.1 demonstrate an understanding of a number of core algorithms associated with derived data types including:
  - 4.1.1 traversing
  - 4.1.2 searching
  - 4.1.3 sorting
  - 4.1.4 merging
- 4.2 demonstrate the ability to prepare the algorithm for the development or coding stage of an appropriate SDLC using modular approaches including:
  - 4.2.1 subprograms
  - 4.2.2 procedures/functions
  - 4.2.3 stub programming
  - 4.2.4 prototyping
  - 4.2.5 libraries

5. explain and demonstrate the rationale, structure and key uses of the fundamental derived data types

5.1 demonstrate the ability to incorporate derived data types including:
- 5.1.1 arrays
- 5.1.2 vectors
- 5.1.3 matrices
- 5.1.4 enumerated data
- 5.1.5 records; e.g., data structures with mixed data types

5.2 demonstrate symbolic data representation, using ASCII coding

6. explain and demonstrate the rationale, structure and key uses of text files

7. explain and analyze the nature, operation and basic architecture of the von Neumann computer system at the machine level

7.1 demonstrate an understanding of the machine level organization of a hypothetical von Neumann machine by describing and representing:
- 7.1.1 the basic components of the Central Processing Unit (CPU), Arithmetic Logic Unit (ALU), control unit, registers, program counter and instruction register
- 7.1.2 the bus
- 7.1.3 the memory

7.2 demonstrate an understanding of the machine language of a hypothetical von Neumann machine by describing and representing:
- 7.2.1 opcodes
- 7.2.2 operands
- 7.2.3 symbolic representation

7.3 demonstrate an understanding of the machine level operations of a hypothetical von Neumann machine by describing and representing:
- 7.3.1 the machine cycle; e.g., fetch, decode, execute
- 7.3.2 the flow of data through the computer under the direction of a hypothetical machine-language program

7.4 demonstrate the mediating role played by system software between the human level and machine level including:
- 7.4.1 operating systems
- 7.4.2 language translators
- 7.4.3 memory managers
- 7.4.4 information managers
- 7.4.5 schedulers
- 7.4.6 utilities

8. demonstrate basic competencies

8.1 demonstrate fundamental skills to:
- 8.1.1 communicate
- 8.1.2 manage information
- 8.1.3 use numbers
- 8.1.4 think and solve problems

8.2 demonstrate personal management skills to:
- 8.2.1 demonstrate positive attitudes and behaviours
- 8.2.2 be responsible
- 8.2.3 be adaptable
- 8.2.4 learn continuously
- 8.2.5 work safely

8.3 demonstrate teamwork skills to:
    8.3.1 work with others
    8.3.2 participate in projects and tasks

9. identify possible life roles related to the skills and content of this cluster

9.1 recognize and then analyze the opportunities and barriers in the immediate environment
9.2 identify potential resources to minimize barriers and maximize opportunities

COURSE CSE2110: PROCEDURAL PROGRAMMING 1

Level: Intermediate

Prerequisite: CSE1120: Structured Programming 2

Description: Students develop their understanding of the procedural programming paradigm. They move from a structured programming approach in which modules were handled through the use of program blocks to a more formal modular programming approach in which they are handled through subprograms. In the process, students also learn to use a number of new design approaches made possible by the new paradigms. As part of this process, they also learn what types of problems are amenable to modular algorithms and programs.

Parameters: Access to appropriate computer equipment, software, support materials, the Internet and a programming environment that encourages modular programming through the use of subprograms.

Supporting Courses: CSE2010: Computer Science 2
CSE2120: Data Structures 1
CSE2130: Files & File Structures 1, and/or any
Intermediate project course involving imperative programming

Outcomes: The student will:

1. **demonstrate an understanding of modular programming**
   - 1.1 describe the advantages of programming with modules or subroutines including:
     - 1.1.1 reducing the duplication of code in a program
     - 1.1.2 enabling the reuse of code in more than one program
     - 1.1.3 decomposing complex problems into simpler pieces to improve maintainability and extendibility
     - 1.1.4 improving the readability of a program
     - 1.1.5 hiding or protecting the program data
   - 1.2 select a programming environment and describe how it supports procedural programming including:
     - 1.2.1 the type of subprograms supported; e.g., procedures, functions, methods
     - 1.2.2 the level or type of modularity provided
     - 1.2.3 the level of protection provided from unwanted side-effects
2. **demonstrate basic procedural programming skills by writing algorithms employing a modular approach to solve problems**
   - 2.1 analyze a data processing problem and use a top-down design approach to decompose it into discreet input, processing and output modules
   - 2.2 analyze and refine modules into submodules that are a manageable size for each process; e.g., input submodules, processing submodules and output submodules
   - 2.3 describe and represent, using pseudocode or an appropriate diagramming approach, the relationship among the modules

2.4 analyze and rewrite algorithms for each module identifying the pre- and post-conditions and required program control of flow mechanisms.
2.5 analyze and evaluate algorithms for each developing module with appropriate data and revise, as required

3. translate algorithms into source code, convert the source code into machine executable form, execute and debug, as required

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; e.g., functions (subprograms that return a value) and procedures (subprograms that do not return a value)
3.3 analyze and determine 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 to:
- 3.3.1 use of appropriate parameters for importing and exporting data to and from subprograms
- 3.3.2 use of local variables and nested subprograms to enhance cohesion
- 3.3.3 one- and two-way parameter passing for importing and exporting data to and from subprograms

3.4 analyze for, and maintain, an appropriate balance between the coupling or dependency and cohesion or focus of subprograms
3.5 create both 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

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, 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. identify possible life roles related to the skills and content of this cluster

6.1 recognize and then analyze the opportunities and barriers in the immediate environment
6.2 identify potential resources to minimize barriers and maximize opportunities

COURSE CSE2120: DATA STRUCTURES 1

Level: Intermediate

Prerequisite: CSE2110: Procedural Programming 1

Description: Students learn how to design code and debug programs that use a set of data structures that can be used to handle lists of related data. Building on their knowledge of basic or primitive data types, they learn how to work with fundamental data structures such as the array and the record. As part of this process, they learn what types of problems benefit from the use of these types of data structures.

Parameters: Access to appropriate computer equipment, software, support materials, the Internet and a programming environment that encourages modular programming through the use of subprograms.

Supporting Courses: CSE2010: Computer Science 2
CSE2130: Files & File Structures 1 and/or any
Intermediate project course involving imperative programming

Outcomes: The student will:

1. analyze and represent the nature, structure and utility of fundamental data types

- 1.1 describe and represent the general nature of static data structures including:
  - 1.1.1 how data structures are stored in memory
  - 1.1.2 the advantages and disadvantages of fundamental data structures in relation to primitive data types
  - 1.1.3 the advantages and disadvantages of the various fundamental data structures
- 1.2 describe and represent the nature and mechanics of basic data structures including:
  - 1.2.1 the static array including: use of cells to store data, data homogeneity, use of an index (or indices) to identify the location of data elements, types; e.g., single dimensional arrays (lists), double dimensional arrays (tables) and parallel arrays (look-up or associative tables)
  - 1.2.2 the record including: the use of fields to store data, data heterogeneity, the use of field names to identify the location of data elements
  - 1.2.3 the dynamic array including: sizes, types; e.g., single dimensional arrays (lists), double dimensional arrays (tables) and parallel arrays (look-up or associative tables)
- 1.3 describe and represent the operations associated with data structures including:
  - 1.3.1 creating the structure
  - 1.3.2 inserting, deleting and replacing data in the structure
  - 1.3.3 searching, finding and retrieving data from the structure
  - 1.3.4 determining the size of the structure
  - 1.3.5 copying the structure
  - 1.3.6 comparing two structures of the same type

2. **create and/or modify algorithms that make effective use of fundamental data structures to solve problems**
   - 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 type of processing operations
   - 2.4 identify and 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 test and modify the algorithm using appropriate "fail-on-paper" techniques
3. **create and/or modify programs based on algorithms that make effective use of fundamental data structures**
   - 3.1 convert algorithms calling for the use of data structures into programs that reflect the algorithm's design
   - 3.2 use cohesive subprograms with helper subprograms to hide and/or protect data and separate the implementation of the data structure and its operations from its calling modules
   - 3.3 use original (user-created) or built-in, environment supported data structures and their attendant operations appropriate to the data being manipulated
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. **identify possible life roles related to the skills and content of this cluster**
   - 6.1 recognize and then analyze the opportunities and barriers in the immediate environment
   - 6.2 identify potential resources to minimize barriers and maximize opportunities

COURSE CSE2130: FILES & FILE STRUCTURES 1

Level: Intermediate

Prerequisite: CSE2120: Data Structures 1

Description: Students learn how to design, code and debug programs that use data files to store and retrieve data on secondary storage devices. Building on their knowledge of derived data structures, they learn how to use those structures to organize data for efficient file handling. As part of this process, they learn what types of problems benefit from the use of external files.

Parameters: Access to appropriate computer equipment, software, support materials, the Internet and a programming environment that encourages modular programming through the use of subprograms.

Supporting courses: CSE2010: Computer Science 2, or any
Intermediate project course involving the manipulation and storing of data

Outcomes: The student will:

1. analyze and represent the nature, structure and utility of external data files

- 1.1 identify and illustrate the general characteristics of external data files including:
  - 1.1.1 access methods; e.g., sequential, random, indexed
  - 1.1.2 type of data; e.g., text (encoded in a format such as ASCII code), binary (encoded in binary code)
  - 1.1.3 text files; e.g., data organization, access methods
- 1.2 explain and represent the client/server relationship that exists between a file using application and the operating system with consideration to:
  - 1.2.1 how programming environments access secondary storage devices
  - 1.2.2 how operating systems handle the actual process of manipulating data in secondary memory
  - 1.2.3 how programming environments request file handling services from the operating system
  - 1.2.4 the use of a file buffer, data stream and file descriptor table
- 1.3 describe and represent the logical structure of text files including:
  - 1.3.1 sequential text
  - 1.3.2 random-access text files
  - 1.3.3 Indexed Sequential Access Method (ISAM) text files
- 1.4 describe the main operations associated with text files including:
  - 1.4.1 creating a file buffer or stream
  - 1.4.2 opening an existing file
  - 1.4.3 creating a new file
  - 1.4.4 exporting data to a file
  - 1.4.5 importing data from a file
  - 1.4.6 appending data to a file
  - 1.4.7 closing a file
  - 1.4.8 comparing two files
  - 1.4.9 copying a file
  - 1.4.10 merging two files

1.5 describe and represent the relative advantages of each file type including:
    1.5.1 access speed
    1.5.2 storage space requirement
    1.5.3 difficulty to implement
    1.5.4 maintainability

2. create and/or modify algorithms that make effective use of external data files

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 external file capability, based on the nature and amount of the data to be processed and type of processing operations
2.4 identify and use or construct the appropriate external file structure based on:
    2.4.1 storage space required
    2.4.2 the number and speed of required operations
    2.4.3 programmer efficiency
2.5 create sequential and random-access files, as required
2.6 identify and sequence the operations needed to process the data prior to export and/or process the data after import
2.7 test and modify the algorithm using appropriate "fail-on-paper" techniques

3. create and/or modify programs based on appropriate algorithms that make effective use of external data files

3.1 convert algorithms calling for the use of external data files into programs that reflect the algorithm's design
3.2 use cohesive subprograms with helper subprograms, if required, to hide and/or protect data, and separate the implementation of the file handling code and attendant data structure and operations from its calling modules
3.3 use original (user-created) or built-in, environment supported file handling code segments and their attendant operations appropriate to the data being manipulated

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. identify possible life roles related to the skills and content of this cluster

6.1 recognize and then analyze the opportunities and barriers in the immediate environment

6.2 identify potential resources to minimize barriers and maximize opportunities

COURSE CSE2140: SECOND LANGUAGE PROGRAMMING 1

Level: Intermediate

Prerequisite: CSE2110: Procedural Programming 1 or
CSE1120: Structured Programming 2

Description: Students who have mastered the basics of one programming language are given the opportunity to learn the basics of another. Designed for students who have learned how to write structured and/or modular programs in a more accessible programming environment, this course gives students an opportunity to develop a similar skill set in a more demanding language. In the process, they have a further opportunity to hone their structured and modular programming skills.

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

Supporting Courses: CSE1010: Computer Science 1
CSE1110: Structured Programming 1
CSE1120: Structured Programming 2 and/or any
Intermediate 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 naming the paradigms supported
     - 1.1.2 outlining the relative advantages and disadvantages of the paradigms
   - 1.2 consider the source code to machine code translation process used by each language by:
     - 1.2.1 identifying and describing the process used by each language
     - 1.2.2 outlining the relative advantages and disadvantages of 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; e.g., attendant learning curve
     - 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

2. **demonstrate programming skills by writing modular structured algorithms in a second language**
   - 2.1 analyze a data processing problem and use a top-down design approach to decompose it into discreet input, processing, output (IPO) modules
   - 2.2 analyze and refine modules into submodules that are a manageable size for each process; e.g., IPO submodules
   - 2.3 describe and represent, using pseudocode or an appropriate diagramming approach, the relationship among the modules
   - 2.4 analyze and rewrite algorithms for each module identifying the pre- and post-conditions and required program control of flow mechanisms
   - 2.5 analyze and evaluate algorithms for each developing module with appropriate data and revise, as required
3. **demonstrate basic coding skills by drawing on first language skills to translate modular structured algorithms into executable programs in the 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; e.g., functions (subprograms that return a value) and procedures (subprograms that do not return a value)
   - 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:
     - 3.3.1 the use of appropriate parameters for importing and exporting data to and from the subprograms
     - 3.3.2 the use of local variables and nested subprograms to enhance cohesion
     - 3.3.3 one- and two-way parameter passing for importing and exporting data to and from the subprograms
   - 3.4 analyze for, and maintain, an appropriate balance between the coupling or dependency and cohesion or focus of the subprograms
   - 3.5 create both 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**
   - 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 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. identify possible life roles related to the skills and content of this cluster

6.1 recognize and then analyze the opportunities and barriers in the immediate environment
6.2 identify potential resources to minimize barriers and maximize opportunities

COURSE CSE2210: CLIENT-SIDE SCRIPTING 3

Level: Intermediate

Prerequisites: CSE1220: Client-side Scripting 2
CSE1120: Structured Programming 2

Description: Students add to their understanding of Internet scripting by employing procedural programming techniques and fundamental data structures to create both static and dynamic client-side sites. Students learn how to analyze, modify, write and debug algorithms and scripts that use subprograms such as functions and data structures such as arrays.

Parameters: Access to appropriate computer equipment, software, support materials and the Internet. Specifically, students must have access to a scripting environment that encourages procedural programming.

Supporting Courses: CSE2010: Computer Science 2
CSE2110: Procedural Programming 1
CSE2120: Data Structures 1

Outcomes: The student will:

1. **demonstrate basic procedural programming approaches and how they can be used to write Internet scripts**
   - 1.1 include the following features:
     - 1.1.1 subprograms that can be readily mapped to specific components of a site's architecture
     - 1.1.2 the decomposition of complex scripting tasks into subtasks improving site design efficiency, maintainability and extendibility
     - 1.1.3 the potential for code reuse both in the same and in other scripts and sites
     - 1.1.4 the enhancement of site security through improved data hiding and information protection
     - 1.1.5 the enhancement of the readability of site scripts
     - 1.1.6 the promotion of collaborative work on site scripts
     - 1.1.7 the reduction of unwanted side effects especially when dealing with multiple scripts
2. **demonstrate basic procedural programming approaches and how they can be used to create development libraries of scriptlets**
   - 2.1 demonstrate how they:
     - 2.1.1 increase design, coding and debugging efficiency
     - 2.1.2 increase user and/or site interactivity
3. **demonstrate the use of data structures in an Internet scripting environment**
   - 3.1 outline the data structures available in a typical Internet scripting environment
   - 3.2 compare and contrast data structures such as arrays with primitive data types
   - 3.3 describe and represent the main operations associated with the fundamental data structures supported by a typical Internet scripting environment
4. **design scripts for an appropriate Internet-oriented scripting environment that uses procedural programming approaches and fundamental data structures**
   - 4.1 outline the intent of the script and determine if the intent can be best realized through the use of procedural programming approaches
   - 4.2 determine the data requirements of the script and determine if the intent can be best realized through the use of fundamental data structures

4.3 create algorithms that use procedural programming approaches to realize the intent of the script including:
    4.3.1 use a top-down design approach to decompose the problem first into modules and then into submodules
    4.3.2 use pseudocode or an appropriate diagramming technique to illustrate the relationship among the modules
    4.3.3 create more detailed algorithms for each module identifying the pre- and post-conditions and required program control of flow mechanisms
    4.3.4 test and modify the developing algorithm with appropriate data using a "fail-on-paper" process

5. write and debug scripts that use procedural programming approaches and fundamental data structures using an appropriate Internet-oriented scripting environment

5.1 demonstrate the ability to use an appropriate scripting language coding environment
5.2 convert the algorithms into scripts consisting of linked modules/subprograms that reflect the structure of the algorithm
5.3 use appropriate types of subprograms to implement the various sections of the algorithm
5.4 maintain an appropriate balance between the coupling or dependency and cohesion or focus of the subprograms
5.5 use internal and external documentation
5.6 execute the script, and track and eradicate errors
5.7 compare the results of the script's execution with the intent of the algorithm and modify, 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. identify possible life roles related to the skills and content of this cluster

7.1 recognize and then analyze the opportunities and barriers in the immediate environment
7.2 identify potential resources to minimize barriers and maximize opportunities

COURSE CSE2240: ROBOTICS PROGRAMMING 2

Level: Intermediate

Prerequisites: CSE1240: Robotics Programming 1
CSE1120: Structured Programming 2

Description: Students add to their understanding of robotics programming by employing procedural programming techniques and fundamental data structures to create programs that display greater agency and autonomy. They learn how to analyze, modify, write and debug robotics algorithms and programs in which modularity is achieved through subprograms such as functions and fundamental data structures such as arrays.

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: CSE2010: Computer Science 2
CSE2110: Procedural Programming 1
CSE2120: Data Structures 1
ELT2140: Robotics 2
ELT2160: Robotics Sensor 1
ELT2170: Robotics Sensor 2

Outcomes: The student will:

1. **demonstrate how basic procedural programming approaches can be used to create robotics programs**
   - 1.1 include the following:
     - 1.1.1 subprograms that can be readily mapped to specific subsections of a robot's architecture
     - 1.1.2 the decomposition of complex robotic tasks into subtasks improving both the maintainability and extendibility of the programs
     - 1.1.3 the potential for 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 the enhancement of the readability of a robotics program
     - 1.1.6 the reduction in side effect errors
2. **demonstrate how basic procedural programming approaches can be used to create task libraries**
   - 2.1 demonstrate how they:
     - 2.1.1 increase design, coding and debugging efficiency
     - 2.1.2 can improve robotic artificial intelligence leading to programs that display greater agency and/or autonomy
3. **demonstrate an understanding of data structures such as arrays and how they can be used in robotics**
   - 3.1 outline and describe the data structures available in a typical robotic programming environment
   - 3.2 outline the main advantages of data structures such as arrays over primitive data types in robotics programming
   - 3.3 describe and illustrate the main operations associated with the data structures supported by a robotic programming environment

3.4 describe and demonstrate how data structures can be used to simulate aspects of human cognition such as memory in robotics programs

4. design a robotics system consisting of at least one robot, associated control systems and environment that use procedural programming approaches and fundamental data structures to carry out a simple set of predetermined tasks

4.1 identify the general tasks the robot will be required to carry out including:
- 4.1.1 breaking those tasks into simpler tasks by continuing the process until each task can be treated as a subprogram
- 4.1.2 drafting a task hierarchy that associates the tasks and subtasks

4.2 describe and diagram the environment in which the robot will be required to operate by:
- 4.2.1 identifying the elements in the environment that can be manipulated by the robot and determining their location
- 4.2.2 identifying the elements in the environment to be detected by the robot's sensors and determining their location
- 4.2.3 determining the type and location of internal barriers in the environment
- 4.2.4 setting the outer limits of the environment

4.3 identify the capabilities the robot will require to carry out the tasks

4.4 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

4.5 design the robot, using the tasks to be accomplished, the proposed environment, the required capabilities and the control approach as parameters

4.6 check your design for congruency against the task list to be accomplished and with the proposed environmental specifications

4.7 modify the design, as required

4.8 carry out the design process sequentially using a top-down approach and employ stepwise refinement

5. use procedural programming approaches to build the environment, robot and controlling mechanism called for by the design

5.1 construct that portion of the environment required for the first task or tasks on the task sequence

5.2 assemble as much of the robot, as is required, to accomplish those tasks

5.3 write algorithms that use modular programming approaches to outline how the first set of tasks is to be accomplished including:
- 5.3.1 use appropriate problem decomposition techniques to break each task into subtasks capable of being represented as modules
- 5.3.2 identify the sense, plan and action component of each module
- 5.3.3 identify the data requirements of each module and determine which requirements should be met by fundamental data types
- 5.3.4 organize each module, as required, using the appropriate structures to control program flow
- 5.3.5 link the modules into calling and called modules
- 5.3.6 write the algorithm in an acceptable format
- 5.3.7 use appropriate techniques to determine if the algorithm will achieve the original intent

5.4 use an RCL, capable of using structured and procedural approaches and supporting fundamental data structures, to translate the algorithms into a program including:
- 5.4.1 convert the algorithms into programs consisting of linked modules/subprograms that reflect the structure of the algorithm
- 5.4.2 use appropriate types of subprograms to implement the various sections of the algorithm
- 5.4.3 maintain an appropriate balance between the coupling or dependency and cohesion or focus of the subprograms
- 5.4.4 pass data between the subprograms without unintended side effects
- 5.4.5 use internal and external documentation

5.5 load and execute program, and track and eradicate errors by:
- 5.5.1 testing each of the physical subsystems of the robot(s) to eliminate engineering errors
- 5.5.2 testing 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
- 5.5.3 using self-test code and check points in each module, as well as observation, to eliminate run-time and internal logic errors
- 5.5.4 comparing the robot's actions with the intent of the algorithm
- 5.5.5 modifying the original task list, environment, algorithm and/or program, as required

5.6 participate in intermittent critiques throughout the iterative process; e.g., planning, analysis, design, testing, evaluation

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. identify possible life roles related to the skills and content of this cluster

7.1 recognize and then analyze the opportunities and barriers in the immediate environment

7.2 identify potential resources to minimize barriers and maximize opportunities

COURSE CSE2910: CSE PROJECT B

Level: Intermediate

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: Intermediate project courses must connect with a minimum of two CTS courses, one of which must be at the intermediate level and be in the same occupational area as the project course. The other CTS course(s) can be at any 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. identify possible life roles related to the skills and content of this cluster

5.1 recognize and then analyze the opportunities and barriers in the immediate environment
5.2 identify potential resources to minimize barriers and maximize opportunities

COURSE CSE2920: CSE PROJECT C

Level: Intermediate

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: Intermediate project courses must connect with a minimum of two CTS courses, one of which must be at the intermediate level and be in the same occupational area as the project course. The other CTS course(s) can be at any 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. identify possible life roles related to the skills and content of this cluster

5.1 recognize and then analyze the opportunities and barriers in the immediate environment
5.2 identify potential resources to minimize barriers and maximize opportunities

COURSE CSE2950: CSE INTERMEDIATE PRACTICUM

Level: Intermediate

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 advanced (3XXX) 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. identify possible life roles related to the skills and content of this cluster

4.1 recognize and then analyze the opportunities and barriers in the immediate environment

4.2 identify potential resources to minimize barriers and maximize opportunities

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