

**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 cyborgization 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