Carleton UNIVERSITY | Department of Systems and Computer Engineering

# SYSC 2006
# Foundations of Imperative Programming

## Calendar description

The imperative programming paradigm: assignment and state, types and variables, static and dynamic typing. Memory management and object lifetimes: static allocation, automatic allocation in activation frames, dynamic allocation. Function argument passing. Recursion. Data structures: dynamic arrays, linked lists. Encapsulation and information hiding; object-based programming.

Includes: Experiential Learning Activity.

Lectures three hours a week, laboratory two hours a week.

http://calendar.carleton.ca/undergrad/courses/SYSC/

## Prerequisites

(ECOR 1051 and ECOR 1052 and ECOR 1053 and ECOR 1054) or ECOR 1606 or SYSC 1005, and second-year status in Engineering.

Precludes additional credit for SYSC 1102 (no longer offered), SYSC 2002 (no longer offered) and COMP 2401.

## Prior knowledge

Students should:

- Understand the concepts of software development as an engineering discipline.
- Understand the concepts of engineering problem solving, defining problems, designing solutions, using procedural programming.
- Understand the fundamentals of engineering.
- Be able to apply engineering drawings to help them design solutions to those engineering problems.

## Course objectives

The imperative programming paradigm has been widely used in practice for various applications. The objective of this course is to understand the concepts that underlie most imperative programming languages and to be able to use this knowledge to learn new languages. The course will deal with designing and implementing small-scale programs, in particular the use of C programming language. Specific concepts to be covered include assignment and state, types and variables, static and dynamic typing; memory management and object lifetimes: static allocation, automatic allocation in

activation frames, dynamic allocation; function argument passing; recursion; and data structures including dynamic arrays, and linked lists. The course also prepares students to undertake a course that provides a thorough introduction to object-oriented programming principles.

## List of topics

- Fundamental elements of imperative programming languages: types, variables, expressions, control flow: conditional statements, iteration (loops), functions (subroutines/procedures).
- Introduction to/review of C as an imperative programming language: types, variables, expressions, if, while, for and do-while statements, function definitions vs. function declarations.
- Function calls and parameter-passing mechanisms. Visualizing program state by drawing memory diagrams containing activation frames (activation records) that depict function parameters and local variables.
- Structuring data: arrays and lists:
    - C arrays. Brief introduction to C character strings.
- Motivation for modular programming. Modules: interface vs. implementation:
    - Modules in C: header (.h) and implementation (.c) files. The C preprocessor. Compiling and linking C programs comprised of several modules. Brief overview of the standard C library.
- Pointers. Depicting pointers in memory diagrams:
    - C pointers. The & and * operators. Passing pointers to local variables as function arguments. Drawing memory diagrams to explain how pointers are passed as function arguments.
    - C arrays and pointers. Drawing memory diagrams to explain how pointers to arrays are passed as function arguments.
- Structuring data:
    - Structures
    - C structs. Structures vs. pointers to structures as function arguments. The -> operator. Memory diagrams.
- Introduction to dynamically-allocated memory and the heap:
    - Heap management in C: malloc and free. Drawing memory diagrams to explain how parameters and local variables in activation frames can point to memory blocks allocated on the heap. Memory leaks. Dynamically allocated structs.
- Dynamically-allocated arrays, dynamic arrays:
    - Case study: C implementation of a list collection using a dynamic array.
- Structuring data: linked lists. Drawing memory diagrams to understand the fundamental operations on singly-linked lists:
    - Implementing linked lists in C, using structs and pointers.
- Dynamically-allocated arrays, dynamic arrays:
    - Case study: C implementation of a list collection using a dynamic array.
- Abstract collections: stacks and queues. Comparison of different designs, based on arrays and linked lists:

    - Implementing stacks and queues as C modules.
- Introduction to recursion.
- Introduction to Go.

## Learning outcomes

By the end of this course, students should be able to:

- Trace short C programs and explain what happens, step-by-step, as the computer executes each statement.
- Visualize how code execution changes the program's state; in other words, draw diagrams that depict the contents of the program's global variables, its activation frames (containing function arguments and local variables) and memory that has been allocated from the heap and is accessed through pointers.
- Design, code and test functions that operate on two fundamental data structures: the dynamic (resizable) array and the pointer-based singly-linked list.
- Describe, from a client-side perspective, the operations provided by some linear abstract collections; e.g., lists (vectors), queues and stacks.
- Implement these collections; i.e., given the specification of a collection's operations and a description of its underlying data structure, implement the data structure and the algorithms that provide the required operations.
- Specify simple recursive algorithms, convert these algorithms into recursive functions, and draw memory diagrams to explain their execution.

## Graduate Attributes (GAs)

The Canadian Engineering Accreditation Board requires graduates of engineering programs to possess 12 attributes at the time of graduation. Activities related to the learning outcomes listed above are measured throughout the course and are part of the department's continual improvement process. Graduate attribute measurements will not be taken into consideration in determining a student's grade in the course. For more information, please visit: https://engineerscanada.ca/.

| Graduate Attribute | Learning outcome(s) |
|---|---|
| 1.4.S: Knowledge Base: Introduced: Programming and algorithms | 2, 4, 5 |
| 5.1: Use of Engineering Tools: Introduced: Diagrams and engineering sketches | 1, 5 |
| 5.3: Use of Engineering Tools: Introduced: Tools for design, experimentation, simulation, visualization, and analysis | 1, 5 |
| 7.1: Communication Skills: Introduced: Instructions | 1, 3, 5 |

## Accreditation Units (AUs)

For more information about Accreditation Units, please visit: https://engineerscanada.ca/.

The course has a total of 49 AUs, divided into:

- Engineering Science: 35%
- Engineering Design: 65%

## Instructor and TA contact

Specific to course offering (tbd)

## Textbook (or other resources)

Specific to course offering (tbd)

## Evaluation and grading scheme

Specific to course offering (tbd)

## Breakdown of course requirements

Specific to course offering (tbd)

## Tentative week-by-week breakdown

Specific to course offering (tbd)

## General regulations

Specific to course offering (tbd)