COMP 2400 - Programming Paradigms

Course Description

Calendar entry

This course is an introduction to common programming paradigms for implementing event-based systems. May not be held with COMP 2150 or COMP 2160 or COMP 3060 when titled Programming Paradigms. Prerequisite: [COMP 1020 or COMP 1021]. Pre- or corequisite: [one of STAT 1150, STAT 2000 (B), STAT 2001 (B), STAT 2220, or PHYS 2496].

General Course Description

There is an infinity of possible solutions to any programming problem. Different programming languages provide their own set of features and strategies to solve problems, which can be categorized across languages into paradigms. Learning about them enables us to adapt to new programming challenges and choose the best tools to solve the problem at hand. And more than that, they provide a framework for us to think about and understand a solution in any programming language.

What kinds of problems does this help us solve? Code represents a model of a system, either real or abstract. The system can be described by data, and the memory of that data at a moment in time is its state. For example, an object can be used to represent the state of some part of a model. Events occur, causing changes to the state. State is at the heart of how a computer functions, and the paradigms we use determine how we instruct the computer to represent and change state.

In this course we will look at a variety of ways to define objects, implement and manage state, and process events. Regardless of the paradigm used, building a computational model will give you the tools you need to develop ever more complex systems and applications as you grow into a Computer Scientist.

Detailed Prerequisites

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

- Design and implement algorithms using arrays.
- Formulate recursive solutions to simple problems and write simple recursive methods.
- Implement simple searching and sorting algorithms using iteration and recursion.
- Use dynamic allocation to create objects.
- Implement list, stack, and queue abstract data types using array and linked list data structures.
- Make use of discrete random variables and probability distributions, including uniform and normal distributions.

- Compute common statistical measures such as mean, standard deviation, and confidence intervals.

Course Goals

By the end of this course students will:

- Implement a solution to a problem using at least one programming language not studied at the 1000 level.
- Begin to understand the advantages and disadvantages of different programming languages and paradigms.
- Make use of object-oriented concepts like abstract classes, polymorphism, encapsulation and overriding in an efficient manner.
- Apply functional programming strategies such as closures to solve programming problems.
- Write programs that respond to events.
- Design, analyze, and implement a finite state machine.
- Define and experiment with a model of a system.

Learning Outcomes

In order to compare the strategies used across paradigms and languages, most of the outcomes below will be demonstrated and/or implemented using more than one programming language.

Procedural Programming

Students should be able to:

1. Write procedures that manipulate numeric and text data.
2. Implement procedural code to read and write text files.
3. Identify mutable and immutable data.
4. Write programs that store and manipulate collections of data using containers.

Simple Object-Oriented Programming

Students should be able to:

1. Differentiate between classes, records, objects, and abstract data types.
2. Define and use classes using appropriate syntax for access modifiers, variables, methods, and constructors.
3. Appropriately use public, protected, and private access modifiers to implement information hiding.
4. Correctly implement object lifecycle management.

Inheritance and Polymorphism

Students should be able to:

1. Describe the meaning of object-oriented concepts such as hierarchies of classes, inheritance, polymorphism, dynamic class binding, and dynamic method-message mapping.
2. Build a hierarchy of classes that makes use of abstract classes and/or interfaces, and polymorphism.
3. Use substitution where an object and an object of a subclass are used interchangeably.
4. Implement and use simple design patterns for tasks such as iteration and comparison.

Functional Programming

Students should be able to:

1. Identify the defining characteristics of the functional programming paradigm.
2. Write programs using higher-order functions.
3. Write programs using lambda expressions and closures, and describe their behaviours.
4. Use method chains/pipelines to process collections of data.

Event-based Programming

Students should be able to:

1. Identify strategies for event handling using procedural programming.
2. Implement responses to events and state changes using object-oriented programming techniques.
3. Implement responses to events and state changes using a functional interface.
4. Compare and contrast the procedural, object-oriented, and functional approaches to event responses.

Finite State Machines

Students should be able to:

1. Define a system's input events, states, transitions between states.
2. Draw a finite state diagram for a given problem.
3. Simulate the execution of a finite state machine, given a sequence of events, to determine the sequence of state transitions and system outputs.
4. Implement a finite state machine using different programming paradigms.

Discrete Event Simulations

Students should be able to:

1. Describe the use of experimentation in Computer Science.
2. Define a model of a system by identifying the events, states, and outputs needed to represent it.
3. Identify the factors and response variables for a model.
4. Implement the simulation of a model.