Object Oriented Programming Syllabus

Module description

There are several programming paradigms in computer science. One of the most important, especially when creating large software projects, is object-oriented programming, or OOP. OOP allows the integration of data and processes on that data into discrete software modules. Learning about OOP will allow you to develop more modularised, more complex software designs. It will also let you understand better how many existing software libraries and systems have been designed. In the later stages of the programme, OOP is used extensively.

This module aims to provide you with an object-oriented programming skill set. You will learn what objects and classes are and how to write classes. You will see how objects can interact with each other, including defining and implementing interfaces to control the interaction. You will learn how to use inheritance to inherit and extend functionality from parent classes. You will learn how to write code according to style guidelines and how to write formal code documentation.

1. Text I/O and functions
2. Using classes and variables to model data
3. File I/O, exception handling and algorithms
4. Writing and testing an algorithm
5. Object interactions
6. Libraries, toolkits, frameworks and widgets
7. Event driven programming and inheritance
8. Refactoring and class design
9. Initialiser lists, constructors and threads
10. Advanced class and user interfaces

Module goals and objectives

Upon successful completion of this module, you will be able to:

1. Understand and explain the key principles of object oriented programming
2. Choose appropriate basic data types to represent different data

3. Write classes with data and functions
4. Explain the purpose of interfaces and write classes that implement specific interfaces
5. Use inheritance to implement a hierarchy of classes
6. Write formal code documentation and write code following style guidelines

Textbook and Readings

Specific essential readings for this module will be taken from the following text book:

Ivor Horton and Peter Van Weert , Beginning C++17: From Novice to Professional, Apress, 2018, Fifth edition ISBN-13 (pbk): 978-1-4842-3365-8

The specific pages for the reading activities will be given in the platform, and there is no need to read beyond to recommended pages.

In addition to the text book, there are additional reading activities written by the course author, some of which involve coding exercises.

There will also be discussion prompts asking you to do some independent research using online sources.

Module outline

The module consists of 10 topics, each of which spans two weeks.

Topic 1. Text I/O and functions : the main menu	**Key concepts:** • C++ edit, compile and run cycle • Text I/O • Functions **Learning outcomes:** • Write, compile and run a C++ program that prints messages to the console • Use the standard library to do text I/O in the console • Write and call simple functions

Topic 2. Using classes and variables to model data : the OrderBookEntry class	**Key concepts:** • Basic data types: numbers and strings • Classes and data • Classes and functions **Learning outcomes:** • Select appropriate data types to represent a dataset in a C++ program • Describe how a class can be used to combine multiple pieces of data into one unit • Write a class with functions
Topic 3. File I/O, exception handling and algorithms : the CSVReader class	**Key concepts:** • Translating pseudocode to C++ • Exception handling • File I/O **Learning outcomes:** • Convert pseudocode algorithms involving iteration, logic and string processing into working C++ code • Use exception handling to gracefully recover when processing unreliable data • Read text data from a file using the getline function
Topic 4. Writing and testing an algorithm : taking orders and the order matching engine	**Key concepts:** • Iterating over vectors • Testing algorithms • Exception handling

	Learning outcomes: • Write functions that calculate basic statistics by iterating over vectors of objects • Use test data to evaluate the correctness of an algorithm • Use exception handling to write robust user input processing code
Topic 5. Object interactions : the Wallet class	**Key concepts:** • Object interactions • Modelling real world items with classes • Static and non-static functions11 **Learning outcomes:** • Use class composition to combine classes together which offer higher level functionality • Explain how to model a familiar real world entity as a class with data and functions • Decide when it is appropriate to use static or non-static functions
Topic 6.	**Key concepts:** • GUI libraries, frameworks and toolkits • JUCE toolkit • DJ software **Learning outcomes:** • Describe the functionality of a DJ application • Explain what a GUI library is and give examples of common GUI widgets • Use the JUCE framework to create and compile a starter application

Topic 7.	**Key concepts:** • JUCE application architecture • Event driven programming • JUCE audio playback functionality **Learning outcomes:** • Describe the functions and runtime of a simple JUCE application • Use the JUCE API classes to implement audio playback functionality • Apply event driven programming techniques to create a GUI
Topic 8.	**Key concepts:** • Refactoring into simpler modules and class composition • GUI layout • Drag and drop **Learning outcomes:** • Identify and refactor audio playback functionality and GUI into a new class • Identify and refactor GUI functionality • Use drag and drop events to add more intuitive file loading functionality
Topic 9.	**Key concepts:** • Polymorphism and interfaces • Data organisation

	• Working with background threads **Learning outcomes:** • Implement a graphical widget using polymorphism to control module interaction • Make decisions about duplicate data and sharing data inside a program • Use multithreading to implement GUI components that automatically update
Topic 10.	**Key concepts:** • Linked lists • Storing application state • Making a file library **Learning outcomes:** • Use advanced JUCE GUI components to implement a table • Reason about data sharing between different parts of a program • Implement a data model for a table component

Activities of this module

The module is comprised of the following elements:

- Lecture videos. In each topic, you will find a sequence of videos in which the example programs for the course are coded up. Further videos review the key programming techniques seen in the coding videos.

- Readings. Each topic may include several suggested readings. These are a core part of your learning, and, together with the videos, will cover all of the concepts you need for this module.
- Practice Quizzes. Each topic will include practice quizzes, intended for you to assess your understanding of the topics. You will be allowed unlimited attempts at each practice quiz. There is no time limit on how long you take to complete each attempt at the quiz. These quizzes do not contribute toward your final score in the class.
- Programming Activities. Each topic includes programming activity worksheets. These take you through the steps you have seen in the videos, and provide code excerpts. They also contain challenges activities which challenge you develop the program beyond the functionality seen in the lecture videos.
- Code. Each topic includes the C++ code written in the videos. You can use this in combination with the worksheets to ensure you have the correct code.
- Discussion Prompts. Each topic includes discussion prompts. You will see the discussion prompt alongside other items in the lesson. Each prompt provides a space for you to respond. After responding, you can see and comment on your peers' responses. All prompts and responses are also accessible from the general discussion forum and the module discussion forum.
- Assessed coursework. There are two assessed courseworks, in the middle and at the end of the module. They are based on the two large programs developed during the course.

How to pass this module

The module has two major assessments each worth 50% of your grade:

∉ Midterm coursework. This consists of a programming task wherein you will add functionality to the first example program seen in the module.

∉ End of term coursework. This consists of a programming task wherein you will add functionality to the second example program seen in the module.

Activity	Required?	Deadline week	Estimated time per course	% of final grade

Midterm coursework	Yes	1-10	25 hours	50%
End of term coursework	Yes	22	25 hours	50%