MCN 7105: Structure and Interpretation of Computer Programs Course Outline

1 Course Details

Course Code: MCN 7105
Course Name: Structure and Interpretation of Computer Programs
Credit Units: 4
Academic Year: 2023/2024

2 Instructor

Name: Marriette Katarahweire (PhD)
Email: kmarriette@cit.ac.ug
Office: Block A - Level 3

3 Description

Effective software engineers need to know efficient techniques that serve as building blocks in the design and implementation of software systems. This course covers a series of techniques for controlling complexity in large software systems. We control complexity by building abstractions that hide details when appropriate. We control complexity by establishing conventional interfaces that enable us to construct systems by combining standard, well-understood pieces in a "mix and match" way. As we confront increasingly complex problems, we find that existing programming languages are not sufficient for our needs and thus we must control complexity by establishing new programming languages in order to express our ideas more effectively. These techniques will be illustrated using a highly expressive language such as Scheme, Python or Clojure. However, the course is not about teaching a particular programming language but rather examines fundamental issues underlying the design decisions of programming languages. This course will forever change the way you think about programming and programming languages i.e., from a language user's view to a designer's view.

4 Course Objectives

The objectives of this course are

- to provide an understanding of the major techniques for controlling complexity in large software systems
- to provide an understanding of the internals of a program execution environment
- to provide an understanding of modifying and creating an interpreter for a new programming language

5 Learning outcomes

Upon completion of the course, students should be able to:

- explain and apply the major techniques for controlling complexity in large software systems: Building abstractions, controlling interaction through Conventional interfaces, and designing new Languages.
- design and implement software systems that demonstrate the concepts covered in the course, specifically: recursive and iterative processes and procedures, higher order procedures, object-oriented methods, data abstractions, procedures with state and dispatch on type.
- understand, modify and create an interpreter for a new programming language, either at the level of a higher order language description or at the level of a register machine description.

6 Detailed Course Content

(i.) Module 1: Building abstractions with procedures (15 hours)

Introduction to Procedure abstraction; procedures as black-box abstraction; computational processes; formulating abstractions with higher-order procedures

(ii.) Module 2: Building abstractions with data (15 hours)

Introduction to data abstraction; compound data; hierarchical data and the closure property; symbolic data; multiple representations of abstract data.

(iii.) Module 3: Controlling interactions (15 hours)

generic operations; self-describing data; message passing; streams and infinite data structures; and object-oriented programming without object-oriented programming language constructs; assignment and state; concurrency.

(iv.) Module 4: Meta-linguistic abstraction (15 hours)

interpretation of programming languages; meta-circular evaluator; evaluation models; machine model; compilation; and embedded languages.

7 Study Materials

The course is based on the famous "Structure and Interpretation of Computer Programs" (SICP) textbook that has been used since the 1980s at the Massachusetts Institute of Technology (MIT). The textbook, recorded videos and other supporting materials are freely available online.

Supporting Videos at `http://groups.csail.mit.edu/mac/classes/6.001/abelson-sussman-lectures/`

Programming Language: Scheme

IDE: DrRacket `http://racket-lang.org/`

8 Mode of Study

The course will be taught on a fast pace. It assumes that students already know how to program in a mainstream imperative language such as Python, C, C++, Java or C#, or, that the students are learning such a language in parallel at the postgraduate level on their own or in other courses.

The course will employ blended learning approaches, combining physical class and online lectures. Videos will be uploaded on the University learning platform and students are expected to watch the videos on their own schedule and pace. Class lectures will be used for introducing and motivating new materials. Problem sets will also be uploaded on the e-learning platform. In addition to the problem sets there will be several substantial programming projects, each involving extensive programming.

9 Mode of Assessment

Substantial weekly problem sets, quizzes, and programming projects will an integral part of the course assessment (40%). Final examination (60%). Part of the final exam may be a practical project.

10 Reading List

1. Abelson, Harold, Gerald Jay Sussman, and Julie Sussman. Structure and Interpretation of Computer Programs 1996: MIT Press. Available: `https://mitpress.mit.edu/books/structure-and-interpretation-computer-programs`