CMP1201:

COMPUTER PROGRAMMING FUNDAMENTALS

Semester II, 2018/2019

A little housekeeping

- **Office No:**

  Room 3014, New CEDAT Building

- **Class email:**

  CMP1201.FUNDAMENTALS@GMAIL.COM

- **Office hours:**

  Mondays, Tuesdays, Wednesdays 2:00pm – 4:00pm

Course Details

Period per Week			Contact Hour per Semester	Weighted Total Mark	Weighted Exam Mark	Weighted Continuous Assessment Mark	Credit Units
LH	PH	TH	CH	WTM	WEM	WCM	CU
30	60	0	60	100	60	40	4

RATIONALE: (From Program Handbook):

"The course provides a thorough understanding of the principles of well-structured and efficient programming in C initially and in Python to fostering a productive and effective programming methodology appropriate for modern day engineering disciplines which require computer programming to carry out simulations, modelling, data gathering and analysis. The course also explores the concepts of Object Oriented Programming (OOP)."

Course Objectives (From Program Handbook)

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

- Analyse a problem by decomposing it into distinct inputs, outputs and processes.
- Use stepwise refinement to design an algorithm from the problem analysis
- Translate a correct algorithm design from pseudo-code into a C program-coding.
- Use programming environment (e.g text editor, compilers, etc.) for development of C and Python programs."

Course Content (1)

1. Introduction to C programming:
   - Basics, Control and Arrays, Pointers, Input/Output.
   - Programming language fundamentals keywords, conditional flow control, iterations, functions.
   - Invocation, parameter passing, recursion, typing, scope and memory management.
   - Introduction to algorithm and complexity (searching and sorting).

2. Data Structures
   - Linked list, queue, tree, stack, hash table. Software engineering formalism, advanced topics in data structures and algorithms.

3. Introduction to Object Oriented Programming (OOP)
   - Philosophy, principles and mechanisms (encapsulation, abstraction, inheritance and polymorphism).
   - Structures, Classes, and Objects, member functions and variables, constructor, destructor, function overloading, virtual functions,
   - Standard Template Libraries – STL, and very briefly exception handling, templates, operator overloading.

Course Content (2)

4. Python Programming:

- Interactive Mode Programming, Script Mode Programming: Identifier conventions and reserved words, Lines and Indentation, Multi-Line Statements, Quotation and Comments, Input from the user and Command Line Arguments.
- Variables and values, Data Types, Numbers, Strings, Lists, Tuples, Dictionary, Data type conversion; Operators and operator precedence.
- Decision Making. if, if . . . else, nested statements. Loops, while, for and nested
- Functions, define, call, passing references and valures, arguments, return.
- Files I/O, keyboard, reading and writing, file operations; Exception Handling.
- Python Modules and packages, locating, import.
- Regular expressions and pattern matching
- Introduction to Python Object Oriented Programming (OOP)

Referencing . . . . . when writing

1. Africa is a continent
2. The sun rises in the east
3. Ohm's law:
4. "Engineering is the purposeful use of science" - - - - -

   Steve Senturia

Plagiarism

ELECTRONIC CIRCUIT MODELING AND SIMULATION IN MODELICA

François E. Cellier[1], Christoph Clauß[2], Alfonso Urquía[3]

Circuit simulators have always remained separate and distinct from the general-purpose dynamical systems modeling and simulation (M&S) tools.

The so-called *Continuous System Simulation Languages (CSSLs)* [1] that are based on state-space descriptions of dynamical systems have primarily been driven by the needs of control engineers. It is no accident that the state-space formalism for the mathematical description of dynamical systems was first proposed by a control engineer, R.E. Kalman [2]. It was further no accident that the *Society for Modeling and Simulation International* was originally founded by a group of aerospace engineers.

The electronic circuit simulators of the past remained outside of the mainstream CSSL development, because electronic circuit models do not lend themselves to manual transformation into a state-space form.

7 References

[1] D.C. Augustin, M.S. Fineberg, B.B. Johnson, R.N. Linebarger, F.J. Sansom, and J.C. Strauss. The SCi Continuous System Simulation Language (CSSL). *Simulation*, 9:281-303, 1967.

[2] R.E. Kalman. Mathematical description of dynamical systems. *SIAM J Control*, 1:152-192, 1963.

[3] H. Elmqvist, S.E. Mattsson, and M. Otter. Modelica – a language for physical system

- You plagiarize if you don't correctly attribute work/statements by others. This is a form of academic misconduct.

e.g:

Circuit simulators have always remained separate and distinct from the general-purpose dynamical systems.

Instead:

"Circuit simulators have always remained separate and distinct from the general-purpose dynamical systems."

- Use quotations, even for paraphrased statements.
- Reference the authors – by name or bibliography ordering.
- Referencing styles from respected discipline journals are recommended. For example, ECE journals such as the IEEE style is acceptable.
- Refer to university policies for acceptable academic conduct.

Computer Programming

- Algorithm vs. Computer program
  - ✓Algorithm: step-by-step process
  - ✓Computer program: step-by-step set of instructions for a computer.
  - ✓Every computer program is an algorithm

Programming Languages

- A programming language is a set of commands, instructions and syntax used to create software programs.

- Programming languages thus allow programmers to develop software.

- Three (3) major families of programming languages:
  - ✓ Machine languages
  - ✓ Assembly languages
  - ✓ High-level languages

Programming Languages cont'd

- Machine languages (ML)
  - ✓ The "native" language of computers.
  - ✓ Comprised of 1s and 0s. Any mistake in ordering the 1s and 0s can cause the program to fail.
- Assembly languages (AL)
  - ✓ Comprised of a set of elemental commands tied to a specific processor.
  - ✓ Assembly language code must be translated to machine language for processing by a computer.
- High-level languages (HL) – C, C++, Java, Python, PHP, Ruby, etc.
  - ✓ Are languages that programmers use to write code. This code can then be compiled into a low-level language directly recognized by the computer hardware.
  - ✓ Big step forward in easier programming
  - ✓ Syntax similar to the English language.
  - ✓ Subdivided into two major groups – Procedural and Object oriented languages.

Procedural Languages Vs. OOP

- Procedural languages typically refer to early HL languages.
  - ✧Such languages focused on structure and were characterized by sequential sets of linear commands.

- With OOP *"The real world can be accurately described as a collection of objects that interact".*
  - ✧Focuses on modeling data as opposed to structure in procedural languages.
  - ✧Programmers code using "blueprints" of data models called classes.

The C Programming Language

- C is a high level programming language developed by Dennis Ritchie of AT&T Bell labs of the USA in 1972.
- It is a structured programming language also referred to as a Procedure Oriented Programming Language.
  - ✓Also called a Compiled Language since the source code must first be compiled for it to run.
- C is reliable, simple, easy to use and well suited to business and scientific applications.

Evolution of C

- ALGOL 60 programming language was an international programming development effort.
- It was designed for use in all applications – commercial, scientific, system, etc.
  - ✧ Due to it's very wide applicability, ALGOL 60 turned out to be too abstract and too general.
- A new language, Combined Programming Language (CPL), was developed at Cambridge to overcome the shortcomings of ALGOL 60.
  - ✧ Unfortunately, CPL also ended up being too big, with too many features making it very hard to learn and difficult to implement.

Evolution of C (2)

- Consequently, another language – Basic Combined Programming Language (BCPL) "B" was developed at Cambridge to overcome the shortcomings in CPL.
  - ✧ BCPL unfortunately turned out to be too specific!
- Dennis Ritchie figured out a way of combining features from earlier programming language development and together with more of his own additions developed C.
- The publication of the C language in 1978 by Kernighan & Ritchie caused a revolution in the computing world.

Why C Programming

- Mainly because it produces code that runs nearly as fast as code written in assembly language.
- Also, an entire family of programming languages is built upon the traditions of C.
  - ✓ Knowing C therefore allows you to understand and appreciate other programming languages based on C.
- A simple analogy can be made with Latin.
  - ✓ Latin is the base for many other languages such as French (Français), Spanish (Español), Italian.
  - ✓ Thus, knowing and understanding Latin allows one to smoothly sail through all the associated languages.

Why C Programming (2)

- Some applications of C can be seen in
  - ✓Operating systems
  - ✓Language compilers
  - ✓Language interpreters
  - ✓Assemblers
  - ✓Text editors
  - ✓Network drivers
  - ✓Databases

Features of C

- C is a collection of functions that are supported by the C library.
- C is highly portable.
  - ✓Thus programs written on one computer can be run on a different computer without modification.
- C is a robust language that can be used to write both simple and complex programs easily.
  - ✓It is thus suited to writing business applications and scientific software.
- Allows for dynamic memory allocation.

Features of C(2)

- Programs written in C are fast and efficient.
- It is a “free form” language.
- C uses a top-down approach since it is a structured programming language.
- It is case sensitive and has 32 keywords.
- Works in 4 stages:
  - ✓ Editing: writing the source code by using some IDE or editor
  - ✓ Preprocessing or libraries
  - ✓ Compiling: converts/translates source code to object code for a specific platform.
  - ✓ Linking: resolves external references and produces an executable file.