L1 - Introduction

November 1, 2019

1 Course introduction - SYSC2006C

1.1 Fundamentals of Imperative Programming - Prof. Lung

PASS is offered for this course - I feel like it might help....

Email from Carleton email address, with SYSC2006 in the subject line. Check Announcements on CULearn before emailing for FAQs.

1.1.1 Textbook

How to Think Like a Computer Scientist, version 1.09, Downey and Scheffler. Free PDF. Find in Course Files/SYSC2006

1.1.2 Labs

Labs will be posted the Friday before the lab. Labs must be submitted before 11:55pm on Sunday. If the lab is complete, there is no requirement to attend the lab session. Will be finalized by Friday September 6.

Do Lab 0 before the first actual lab session. Lab sessions start Monday, September 9.

No lab exams. Labs are satisfactory, marginal, or unsatisfactory. Satisfactory for clear progress toward a solution = 100%.

2 Imperative Programming Concepts

Imperative programming is a programming paradigm in which computation is specified in terms of program *state* and statements that change program state - a program's state is the contents of its variables at any point in its execution.

The program consists of a sequence of commands for the computer to perform.

An imperative programming language: * has constructs that control the order operations are performed in; * allows *side effects* - state can be modified at different points in execution * side effects can be significant in programs with low-level control

Two common programming paradigms based on imperative programming are *procedural programming* and *object-oriented programming*.

Procedural Programming Program commands are grouped into procedures, functions and subroutines. Procedures in this paradigm are not grouped with program state; variables exist entirely outside the instructions and just have stuff done to them.

Object oriented Programming Instructions are grouped based on the state they operate on; objects encapsulate both data and instructions. Object state is only modified by code that is part of the object.

2.0.1 Concepts

Program state is maintained using *variables*. Assignment statements change the values associated with a variable. By default, statements are executed sequentially from a starting point except when encountering a goto or a function call.

A *loop* executes a sequence of statements multiple times. We may control loop execution: * early exit from the loop (break) * skip execution of the remainder of the loop body, continue at next iteration (continue)

A *conditional branch* executes a sequence of statements only if a specified condiiton is true. *Unconditional branching* includes gotos, subprograms, functions, and procedures. Other constructs - generators, coroutines - will not be covered in this course.

The primary language in this course is C, but the course is about *the fundamental concepts of imperative programming*. These concepts apply equally to all modern languages (Python, Go, Rust...). We may look at Go.

2.0.2 Why use C?

- one of the most popular languages
  - even though it's old af
  - Thompson and Ritchie, Turing Award, yada yada
- small compared to Java or C++ (microcontrollers!)
- good for learning how to manage memory of an executing program
  - not abstracted by the compiler/interpreter
  - we must allocate and deallocate memory on the heap
- commonly used in operating systems, networking equipment, and embedded systems