COMS 3137: Data Structures and Algorithms

What We Will Accomplish this Semester:

- We will learn about many different data structures commonly used in programming.
- We will understand the importance of selecting the *right* data structure.
- We will understand the relationship between an algorithm and the data structures that are used to implement the algorithm.
- We will further explore and understand the concepts of Object Oriented Programming
- We will understand the classic space-time tradeoff between memory usage and computation time.
- We will greatly expand our ability to use Graphical User Interfaces (GUI's) in programming
- You will be able to take a huge variety of advanced CS courses with this background

Why Learn Data Structures?

A famous equation by Niklaus Wirth about programming is:

$$Programs \ = \ Data\,Structures \ + \ Algorithms \tag{1}$$

To update this for Java-speak, we can say instead:

$$Programs \ = \ Objects \ + \ Methods \tag{2}$$

- **Analogy I:** In this class, we will create a set of new **power** tools from a set of existing, limited hand tools. Would you rather build a large structure like a house with hand tools or power tools?
- **Analogy II:** Programming is WAR!!! Its you against the computer! Would you rather fight this war with simple hand guns and knives or with sophisticated grenades, torpedoes, and laser guided bombs. Nobody wants to write video games, 3-D graphics programs, spreadsheets etc. using the lowest level, pitiful data structures. Why drive a Ford when Ferrari's exist? We will be building the Ferrari's in this class.

Levels of Abstraction in Computer Science

We can get the computer to do what we want in a number of ways. We can write programs at a high or low level of abstraction.

- Level 0: Machine Language
- Level 1: Assembly Language
- Level 2: Programming Language (C, Fortran, C++, Java. LISP etc.)
- Level 3: Application: Spreadsheet, word processor, paint program etc.
- Level 4: User interface, Web-Based application
- Levels 0-1 are *machine dependent*. Levels 2-4 tend to be *machine independent*.

Our current Hand Tools: Primitive Data Structures We Know

- characters
- integers, longs
- floats, doubles

Some Early Power Tools We Know

- Arrays: Contiguous Groupings of primitive data structures. More interesting, but still limited. Need to be pre-allocated.
- Strings: Grouping of chars that follow a simple protocol. Each string is zero or more consecutive char's.
- Where the power comes from: Once we define a string data structure, we can then create methods (functions) that work on the string data structure and not at the char level. String methods let you think at a *higher level of abstraction* about string processing rather than forcing you to focus on what each little char is doing.
- See Java API for String class...

Abstraction in Programming

- Creating the right data model (objects) for thinking about a problem
- Devising the appropriate methods to solve the problem

Definitions

- Data Models: Abstractions used to describe problems
- Data Structures: Programming Language constructs used to represent Abstract Data Models.
- Algorithms: Sequence of steps that can be carried out to solve problems. Algorithms can be represented as English text (e.g. a recipe), pseudo code (e.g. a flowchart) or as computer codes.

Example: A Spreadsheet

- Data Model: table with rows and columns and cells containing different data

Name	Test 1	Test 2	Test3
D. Jones	88	77	82
B. Smith	91	87	93
A. Fron	73	75	62

- Data Structure is (perhaps) a 2-Dimensional array of cells
- Algorithm is a method to sum rows and columns etc.

Example: A Printer Queue

- A printer queue can be modeled as a FIFO list: First In, First out. Jobs are added to the end of the list and then printed from the front of the list in the order they entered the list.
- It is not static, but dynamic - number of entries will always vary.
- Given that the data model is a FIFO list, we can define operations (methods) on the list
- List Operations: Add job to list, Delete job from list, list status inquiry.
- To implement list operations on the computer, we need to write code to add a new job node to the tail of the linked list, to remove a successfully printed job from the head of the list, and to traverse the list to display the status of all jobs in the queue.

Example: A Windowing System

- Writing a window system for a computer is a non-trivial programming exercise. It sounds very difficult (and it is!).
- However, at the correct level of abstraction, we can actually create much of the component machinery necessary to have a windowing system.
- What are (some) attributes of a window on a computer screen?

Window Attributes:

1. An identifier (name) for this window
2. Location of the window on the screen
3. Window dimensions (height and width)
4. Title bar on the window
5. Background color of window
6. Text to be displayed in window
7. Graphics to be displayed in window
8. Is window an Icon (minimized)?
9. Is the window visible (front or back?)
10. Is a mouse cursor in the window?

Key Points:

- Even if we haven't the slightest idea how to write pixels on a screen, we have a reasonable data model of a windowing system. If someone were to give us a piece of code that would actually write pixels to a screen, we could probably easily implement this windowing system.
- It is obvious we have ignored some details. But don't sweat the small stuff - there is always time to figure out the details later! Get the big picture correct, and the rest of it falls into place quite nicely. This is known as *top-down* design.
- How do we draw a computer screen with multiple windows, and still make sure the correct windows appear on top? Well, we can establish a Window List, and draw the windows on the screen from the end of the list first (rearmost windows) until the last window we draw is the front one.
- Hey! We already have a list data structure for Printer Queues, and maybe we can easily reuse this for the Window List! This is what reusable software is all about.
- We can even steal functions from the Print Queue. Add job to queue becomes Create New window. Delete job from queue becomes Kill Window.

A Video Game

- Assume you are in COMS 3137 to make some money - write a new hit video game.
- It sounds very difficult and complex to do this. Once again, if we *abstract* the problem in a *top-down* fashion, it starts to sound more tractable.
- Think of the *Objects* we need in a video game:

  - A set of *Environment Objects* (city, fortress, race track etc.)
  - A set of *Actor Objects* (people, dragons, ninjas, kickboxers, animals etc.)
  - A set of *Action Objects* (knives, cars, torpedos, grenades etc.)
  - A set of *Methods* associated with each *Object*:
    * *Actor Objects* move
    * *Environment Objects* change/transform (trapdoors open, buildings get blown up etc.)
    * *Action Objects* are picked up, discharged, thrown down etc.

- Now we have to find ways to transform these abstractions into code.
- What you will find is that by defining things in this *top-down* fashion, the code actually is easier to write - it *almost writes itself*.
- But first we have to create higher level data structures (i.e. an *Actor Object*) out of lower-level data items. But once we build it, we never have to do it again!