CS210: Data Structures
Week 1

"Get your data structures correct first, and the rest of the program will write itself."

- *David Jones*

Class Overview

- Introduction to many of the basic data structures used in computer software
  - Understand the data structures
  - Analyze the algorithms that use them
  - Know when to apply them
- Practice design and analysis of data structures.
- Practice using these data structures by writing programs.

What is a Program

- A Set of Instructions
- Data Structures + Algorithms
- Data Structure = A Container stores Data
- Algoirthm = Logic + Control

Common Data Structures

- Array
- Stack
- Queue
- Linked List
- Tree
- Heap
- Hash Table
- Priority Queue

How many Algorithms?

- Countless

Which Data Structure or Algorithm is better?

- Must Meet Requirement
- High Performance
- Low RAM footprint
- Easy to implement
  - Encapsulated

Algorithm Specification

- An *algorithm* is a finite set of instructions that accomplishes a particular task.
- Criteria
  - input: zero or more quantities that are externally supplied
  - output: at least one quantity is produced
  - definiteness: clear and unambiguous
  - finiteness: terminate after a finite number of steps
  - effectiveness: instruction is basic enough to be carried out

Algorithm Specification

- Representation
  - A natural language, like English or Chinese.
  - A graphic, like flowcharts.
- Algorithms + Data structures = Programs [Niklus Wirth]

Algorithm Specification

- **Example 1.1 [*Selection sort*]:**
  - From those integers that are currently unsorted, find the smallest and place it next in the sorted list.

i	[0]	[1]	[2]	[3]	[4]
-	30	10	50	40	20
0	10	30	50	40	20
1	10	20	40	50	30
2	10	20	30	40	50
3	10	20	30	40	50

```
for (i = 0; i < n; i++) {
  Examine list[i] to list[n-1] and suppose that the
  smallest integer is  at list[min];

  Interchange list[i] and list[min];
}
```

Program 1.1: Selection sort algorithm

- Program 1.3 contains a complete program which you may run on your computer

```
#include <stdio.h>
#include <math.h>
#define MAX_SIZE 101
#define SWAP(x,y,t) ((t) = (x), (x)= (y), (y) = (t))
void sort(int [],int); /*selection sort */
void main(void)
{
   int i,n;
   int list[MAX_SIZE];
   printf("Enter the number of numbers to generate: ");
   scanf("%d",&n);
   if( n < 1 || n > MAX_SIZE) {
     fprintf(stderr, "Improper value of n\n");
     exit(1);
   }
   for (i = 0; i < n; i++) {/*randomly generate numbers*/
      list[i] = rand() % 1000;
      printf("%d  ",list[i]);
   }
   sort(list,n);
   printf("\n Sorted array:\n ");
   for (i = 0; i < n; i++) /* print out sorted numbers */
     printf("%d  ",list[i]);
   printf("\n");
}
void sort(int list[],int n)
{
   int i, j, min, temp;
   for (i = 0; i < n-1; i++)  {
     min = i;
     for (j = i+1; j < n; j++)
        if (list[j] < list[min])
           min = j;
     SWAP(list[i],list[min],temp);
   }
}
```

Program 1.3: Selection sort

Data Structures: What?

- Need to organize program data according to problem being solved
- Abstract Data Type (ADT) - A data object and a set of operations for manipulating it
  - List ADT with operations **`insert`** and **`delete`**
  - Stack ADT with operations **`push`** and **`pop`**

Data Structures: Why?

- Program design depends crucially on how data is structured for use by the program
  - Implementation of some operations may become easier or harder
  - Speed of program may dramatically decrease or increase
  - Memory used may increase or decrease
  - Debugging may be become easier or harder

Data abstraction

- Data Type
  A data type is a collection of objects and a set of operations that act on those objects.

  - For example, the data type int consists of :
    - the objects {0, +1, -1, +2, -2, …, INT_MAX, INT_MIN}
    - the operations +, -, *, /, and %.

Terminology

- Abstract Data Type (ADT)
  - Mathematical description of an object with set of operations on the object. Useful building block.
- Algorithm
  - A high level, language independent, description of a step-by-step process
- Data structure
  - A specific family of algorithms for implementing an abstract data type.
- Implementation of data structure
  - A specific implementation in a specific language

Data Structure Concepts

- Data Structures are containers:
  - they hold other data
  - arrays are a data structure
  - ... so are lists
- Other types of data structures:
  - stack, queue, tree,
    binary search tree, hash table,
    dictionary or map, set, and on and on

Core Operations

- Data Structures will have 3 core operations
  - a way to add things
  - a way to remove things
  - a way to access things

- More operations added depending on what data structure is designed to do

Performance analysis

- Criteria
  - Is it correct?
  - Is it readable?
  - …
- Performance Analysis
  - space complexity: storage requirement: How much storage does it consume.
  - time complexity: computing time: What is the running time of the algorithm.
- Different algorithms may correctly solve a given task
  - Which one should I use?