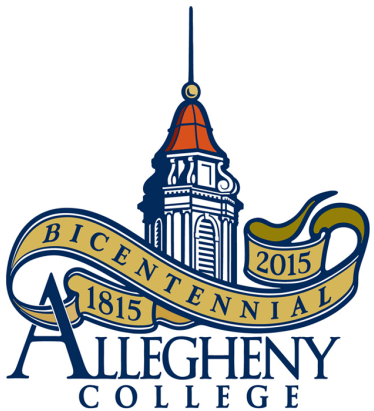
CS101 - Data Abstraction

DS Basics - Module1

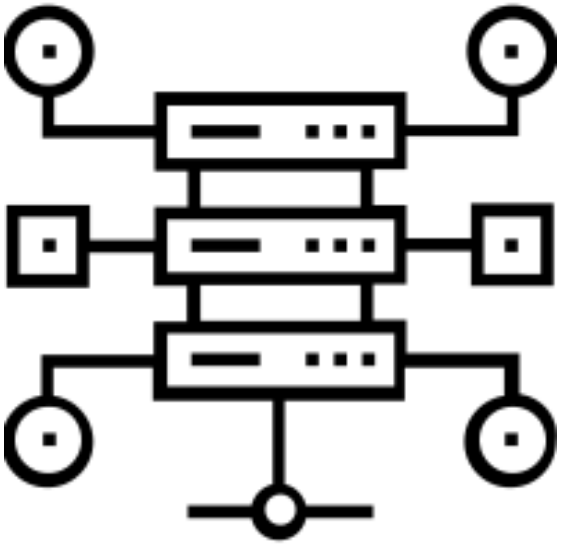
Aravind Mohan

Allegheny College

February 18, 2020



Data Structures



- **Definition** - A data structure is a technique that is primarily used to access, process, store, and organize data.

Data Structures

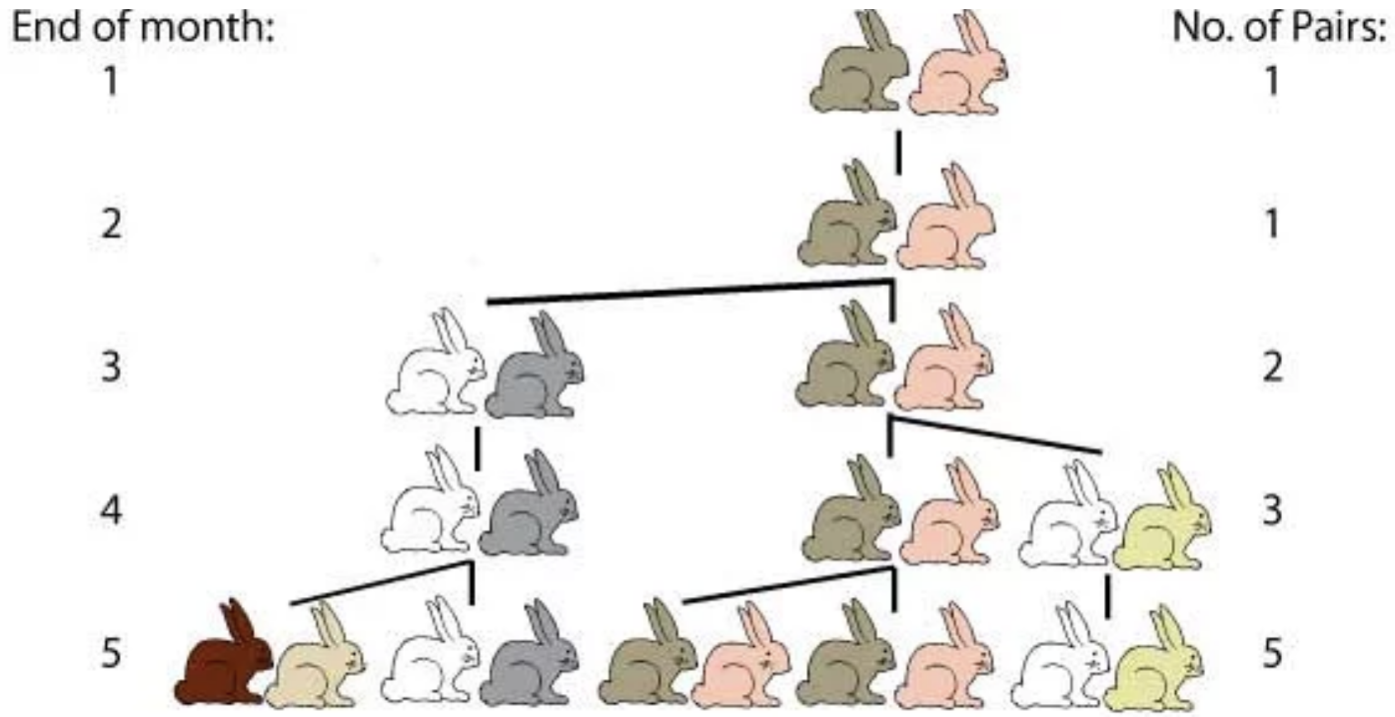
- Core operations supported by a data structure:
  - Add [store]
  - Retrieve [access/read]
  - Remove [organize]
- Other supporting operations are possible based on the data structure.
- Few examples:

  Arrays, Linked List, Stacks, Queues, Hash Maps, Trees, Graphs, etc.

Recap on Arrays

- Consecutive blocks of data in memory
- Homogenous data storage
- Any type of object can be stored in an array: integers, doubles, booleans, Strings, Clocks,...
- The size of the array can be found using **a.length;**

How do we process an Array?



- **Fibonacci Sequence:** $\{0,1,1,2,3,5,8, \ldots\}$
- **Leonardo Problem:** At the end of year, how many pairs of rabbits exist?

Two-Dimensional Arrays

- Solution: Add a second dimension!

```
int myArray[][] = new int[10][5];
```

- Format is [rows][columns]
- Each element still must be the same type
- Can still access each item individually
  - myArray[6][1]

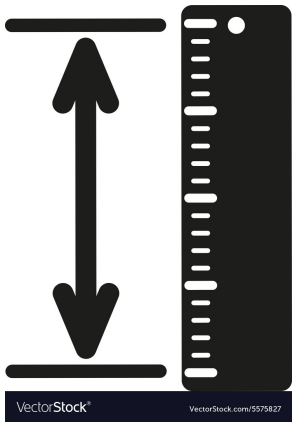
Try to find out?

How do we find the Length?

- Each row doesn't need to have the same length!

```
int myArray[][] = {
            {1, 2, 3, 4, 5},
            {6, 7},
            {8, 9, 0, 1},
            {2, 3, 4}
          }; //myArray[][]
```

Length



- Given myArray[3][4], what is:
- myArray.length?
  - 3 - the number of rows
- myArray[0].length?
  - 4 - the number of columns (in the first row)

Dimensions

- myArray[1][2][3]
- myArray[1][2][3][4]
- myArray[1][2][3][4][5]
- myArray[1][2][3][4][5][6]. . .

. . . if you get this far, you may want to rethink your data storage . . .

Ready for a programming challenge?

- Given the following array, find the indices for the minimum value, maximum value, and calculate average for the entire array.

```
int myArray[][] = {
        {7, 12, 4, -1},
        {-2, 3},
        {8, 9, 0, 1},
        {-12, 3, 4}, {2}
    }; //myArray[][]
```

An array of objects

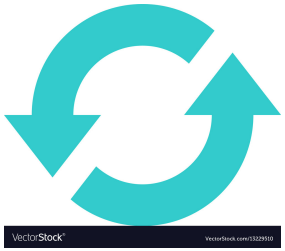
- **An array of objects** is a set of objects, with each object is of heterogeneous type.
- So a homogeneous data store with heterogeneous cells?

An array of objects

- **Example:**

  Students.java and StudentsStub.java in code folder

An alternative approach to Process Data



- **Recursion** is a technique that solves a problem by solving a smaller problem of the same type.
- Sometimes, the best way to solve a problem is by solving a smaller version of the exact same problem first.

Recursion Vs Iteration

- Iteration can be used in place of recursion.
  - An iterative algorithm uses a looping construct.
  - A recursive algorithm uses a branching structure.
- Recursive solutions are often less efficient, in terms of both time and space, than iterative solutions.
- Recursion can simplify the solution of a problem, often resulting in shorter, more easily understood source code.

How do I write a recursive function?

- Determine the size factor
- Determine the base case(s)

  (the one for which you know the answer)
- Determine the general case(s)

  (the one where the problem is expressed as a smaller version of itself)
- Verify the algorithm

  use the (”Three-Question-Method”)

Three-Question Verification Method

1. The Base-Case Question:

   Is there a nonrecursive way out of the function, and does the routine work correctly for this base case?

2. The Smaller-Caller Question:

   Does each recursive call to the function involve a smaller case of the original problem, leading inescapably to the base case?

3. The General-Case Question:

   Assuming that the recursive call(s) work correctly, does the whole function work correctly?

Example 1: Factorial Calculation

- Question: What is ”12!”?
  - 12! = 12*11*10*9*8*7*6*5*4*3*2*1
  - 12! = 479,001,600
- Iterative calculation: Put it in a for-loop.
- Recursive calculation: Use fact(n-1) to calculate fact(n).
- Do each of these provide identical answers?
- Do each of these run at (roughly) the same speed?

Example 2: Fibonacci Calculation

- Question: What are the Fibonacci numbers?
  - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233
  - Each number is the sum of the two numbers preceding it.
- Iterative calculation: Put it in a for-loop.
- Recursive calculation: Use fib(n-1) and fib(n-2) to calculate fib(n).
- Do each of these provide identical answers?
- Do each of these run at (roughly) the same speed?

Reading Assignment

GT Chapter 3 - 3.1
GT Chapter 5 - 5.1, 5.3.1, 5.4

More on Linked Lists.

Questions?

Please ask if there are any Questions!