1.4 Arrays

This lecture. Store and manipulate huge quantities of data.

Array. Indexed sequence of values of the same type.

Examples.
- 52 playing cards in a deck.
- 10 thousand undergrads at Penn.
- 1 million characters in a book.
- 10 million audio samples in an MP3 file.
- 4 billion nucleotides in a DNA strand.
- 73 billion Google queries per year.
- 50 million cells in the human body.
- $6.02 \times 10^{23}$ particles in a mole.

Many Variables of the Same Type

Goal. 10 variables of the same type.

```
// tedious and error-prone
double a0, a1, a2, a3, a4, a5, a6, a7, a8, a9;
a0 = 0.0;
a1 = 0.0;
a2 = 0.0;
a3 = 0.0;
a4 = 0.0;
a5 = 0.0;
a6 = 0.0;
a7 = 0.0;
a8 = 0.0;
a9 = 0.0;
a4 = 3.0;
a8 = 8.0;
double x = a4 + a8;
```

```
// easy alternative
double[] a = new double[10];
a[4] = 3.0;
a[8] = 8.0;
double x = a[4] + a[8];
```

Many Variables of the Same Type

Goal. 1 million variables of the same type.

```
// scales to handle large arrays
double[] a = new double[1000000];
a[123456] = 3.0;
a[987654] = 8.0;
double x = a[123456] + a[987654];
```

```
// scales to handle large arrays
double[] a = new double[1000000];
a[123456] = 3.0;
a[987654] = 8.0;
double x = a[123456] + a[987654];
```
Arrays in Java

Java has special language support for arrays.
- To make an array: declare, create, and initialize it.
- To access entry i of array named a, use a[i].
- Array indices start at 0.

```java
int N = 10; // size of array
double[] a; // declare the array
a = new double[N]; // create the array
for (int i = 0; i < N; i++) // initialize the array
    a[i] = 0.0; // all to 0.0
```

Compact alternative:
- Declare, create, and initialize in one statement.
- Default initialization: all numbers automatically set to zero.

```java
int N = 10; // size of array
double[] a = new double[N]; // declare, create, init
```

Vector Dot Product

Given two vectors x and y of length N, their dot product is the sum of the products of their corresponding components.

```java
double[] x = {0.3, 0.6, 0.1};
double[] y = {0.3, 0.1, 0.4};
int N = x.length;
double sum = 0.0;
for (int i = 0; i < N; i++)
    sum += x[i] * y[i];
```

Array Processing Examples

- create an array with random values
- print the array values, one per line
- find the maximum of the array values
- compute the average of the array values
- copy to another array
- reverse the elements within an array

Shuffling a Deck

Ex. Print a random card.

```java
String[] rank = {
    "2", "3", "4", "5", "6", "7", "8", "9",
    "10", "Jack", "Queen", "King", "Ace"
};
String[] suit = {
    "Clubs", "Diamonds", "Hearts", "Spades"
};
int i = (int) (Math.random() * 13); // between 0 and 12
int j = (int) (Math.random() * 4); // between 0 and 3
System.out.println(rank[i] + " of " + suit[j]);
```
In iteration i, exchange it with Texas hold ‘em poker.

Ex. Create a deck of playing cards and print them out.

```
String[] deck = new String[52];
for (int j = 0; j < 13; j++)
    deck[j*4 + j] = rank[j] + " of " + suit[j];
for (int i = 0; i < 52; i++)
    System.out.println(deck[i]);
```

Q. In what order does it output them?

A. two of clubs  
B. two of diamonds  
C. two of hearts 
D. two of spades 
E. three of clubs

```
public class Deck {
    public static void main(String[] args) {
        String[] suit = {"Clubs", "Diamonds", "Hearts", "Spades"};
        String[] rank = {"2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "Queen", "King", "Ace"};
        int N = SUITS.length;
        int length = RANKS.length;
        String[] deck = new String[52];
        for (int i = 0; i < N; i++) {
            int j = (int) (Math.random() * (N-1));
            String t = deck[i];
            deck[i] = deck[j];
            deck[j] = t;
        }
        for (int i = 0; i < N; i++)
            System.out.println(deck[i]);
    }
}
```

**Coupon Collector**

**Goal.** Given an array, rearrange its elements in random order.

**Shuffling algorithm.**

1. In iteration i, pick random card from deck[i] through deck[N-1], with each card equally likely.
2. Exchange it with deck[i].

**Setting Array Values at Run Time**

Typical array-processing code change value of run-time.

```
public class Deck {
    public static void main(String[] args) {
        String[] suit = {"Clubs", "Diamonds", "Hearts", "Spades"};
        String[] rank = {"2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "Queen", "King", "Ace"};
        int N = SUITS.length;
        int length = RANKS.length;
        String[] deck = new String[52];
        for (int i = 0; i < N; i++) {
            int j = (int) (Math.random() * (N-1));
            String t = deck[i];
            deck[i] = deck[j];
            deck[j] = t;
        }
        for (int i = 0; i < N; i++)
            System.out.println(deck[i]);
    }
}
```

**Shuffling a Deck of Cards**

```
public static void main(String[] args) {
    String[] suit = {"Clubs", "Diamonds", "Hearts", "Spades"};
    String[] rank = {"2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "Queen", "King", "Ace"};
    int N = SUITS.length;
    int length = RANKS.length;
    String[] deck = new String[52];
    for (int i = 0; i < N; i++) {
        int j = (int) (Math.random() * (N-1));
        String t = deck[i];
        deck[i] = deck[j];
        deck[j] = t;
    }
    for (int i = 0; i < N; i++)
        System.out.println(deck[i]);
}
```

**Shuffling a Deck of Cards: Putting Everything Together**

```
public class Deck {
    public static void main(String[] args) {
        String[] suit = {"Clubs", "Diamonds", "Hearts", "Spades"};
        String[] rank = {"2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "Queen", "King", "Ace"};
        int N = SUITS.length;
        int length = RANKS.length;
        String[] deck = new String[52];
        for (int i = 0; i < N; i++) {
            int j = (int) (Math.random() * (N-1));
            String t = deck[i];
            deck[i] = deck[j];
            deck[j] = t;
        }
        for (int i = 0; i < N; i++)
            System.out.println(deck[i]);
    }
}
```

**War Story (PlanetPoker.com)**

Texas hold ‘em poker: Software must shuffle electronic deck of cards
Coupon Collector Problem

Given \( N \) different card types, how many do you have to collect before you have (at least) one of each type?

Simulation algorithm. Repeatedly choose an integer \( i \) between 0 and \( N - 1 \). Stop when we have at least one card of every type.

Q. How to check if we’ve seen a card of type \( i \)?
A. Maintain a boolean array so that \( \text{found}[i] \) is true if we’ve already collected a card of type \( i \).

Coupon Collector: Java Implementation

```java
public class CouponCollector {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        int cardcnt = 0; // number of cards collected
        int valcnt = 0; // number of distinct cards
        // do simulation
        boolean[] found = new boolean[N];
        while (valcnt < N) {
            int val = (int) (Math.random() * N);
            cardcnt++;
            if (!found[val]) {
                valcnt++;
                found[val] = true;
            }
            } // all N distinct cards found
        System.out.println(cardcnt);
    }
}
```

Coupon Collector: Debugging

Add code to print contents of all variables.

<table>
<thead>
<tr>
<th>val</th>
<th>found</th>
<th>valcnt</th>
<th>cardcnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>F F F F F</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>T T T T T</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>T T T T T</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>T T T T T</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>T T T T T</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>T T T T T</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>T T T T T</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>T T T T T</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>T T T T T</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>T T T T T</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>T T T T T</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>

Challenge. Debugging with arrays requires tracing many variables.

Coupon Collector: Mathematical Context

Given \( N \) different possible cards, how many do you have to collect before you have (at least) one of each type?

Fact. About \( N (1 + 1/2 + 1/3 + \ldots + 1/N) \approx N \ln N \).

Ex. \( N = 30 \) baseball teams. Expect to wait \( 120 \) years before all teams win a World Series.

Coupon Collector: Scientific Context

Q. Given a sequence from nature, does it have some characteristics as a random sequence?
A. No easy answer - many tests have been developed.

Coupon collector test. Compare number of elements that need to be examined before all values are found against the corresponding answer for a random sequence.

Multidimensional Arrays
Two-Dimensional Arrays

Two-dimensional arrays
- Table of data for each experiment and outcome.
- Table of grades for each student and assignments.
- Table of grayscale values for each pixel in a 2D image.

Mathematical abstraction. Matrix.
Java abstraction: 2D array.

Setting 2D Array Values at Compile Time

Initialize 2D array by listing values.

```java
double[][] p = {
    { 0.02, 0.02, 0.02, 0.02, 0.02 },
    { 0.02, 0.02, 0.02, 0.02, 0.02 },
    { 0.02, 0.02, 0.02, 0.02, 0.02 },
    { 0.47, 0.47, 0.47, 0.47, 0.47 }
};
```

Matrix Addition

Matrix addition. Given two N-by-N matrices `a` and `b`, define `c` to be the N-by-N matrix where `c[i][j]` is the sum `a[i][j] + b[i][j].`

```java
double[][] c = new double[N][N];
for (int i = 0; i < N; i++)
   for (int j = 0; j < N; j++)
      c[i][j] = a[i][j] + b[i][j];
```

Array Challenge

Q. How many scalar multiplications multiply two N-by-N matrices?
A. N²  B. N³  C. N⁴  D. N⁵

```java
double[][] c = new double[N][N];
for (int i = 0; i < N; i++)
   for (int j = 0; j < N; j++)
      for (int k = 0; k < N; k++)
         c[i][j] += a[i][k] * b[k][j];
```
Summary

Arrays

- Organized way to store huge quantities of data.
- Almost as easy to use as primitive types.
- Can directly access an element given its index.

Ahead: Reading in large quantities of data from a file into an array.