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.
- 5 thousand undergrads at Princeton.
- 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 trillion cells in the human body.
- $6.02 \times 10^{23}$ particles in a mole.

Arrays in Java

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

Compact alternative.
- Combine declare, create, and initialize in one statement.
- Default initialization: all values automatically set to 0.

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;
a9 = 0.0;
a9 = 0.0;
double x = a4 + a8;

Arrays

index value
0 wayne
1 doug
2 rs
3 dgabai
4 mona
5 cbienia
6 wkj
7 mkc

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.
- 5 thousand undergrads at Princeton.
- 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 trillion cells in the human body.
- $6.02 \times 10^{23}$ particles in a mole.

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

Compact alternative.
- Combine declare, create, and initialize in one statement.
- Default initialization: all values automatically set to 0.

index value
0 wayne
1 doug
2 rs
3 dgabai
4 mona
5 cbienia
6 wkj
7 mkc

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;
a9 = 0.0;
a9 = 0.0;
double x = a4 + a8;
Vector Dot Product

Dot product. Given two vectors \( \mathbf{x} \) and \( \mathbf{y} \) of length \( N \), their dot product is the sum of the products of their corresponding components.

\[
\text{double sum} = 0.0; \\
\text{for (int } i = 0; i < N; i++) \\
\quad \text{sum} += x[i] \times y[i];
\]

<table>
<thead>
<tr>
<th>( i )</th>
<th>( x )</th>
<th>( y )</th>
<th>( x[i] \times y[i] )</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>.30</td>
<td>.10</td>
<td>.03</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>.60</td>
<td>.10</td>
<td>.06</td>
<td>.03</td>
</tr>
<tr>
<td>2</td>
<td>.10</td>
<td>.40</td>
<td>.04</td>
<td>.09</td>
</tr>
</tbody>
</table>

Compile-Time Initialize

Compile-time initialize. Can initialize array by listing values.

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); \\
int j = (int) (Math.random() * 4); \\
String t = deck[r]; \\
deck[r] = deck[i]; \\
deck[i] = t;
```

Shuffling

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

Shuffling algorithm.

- In iteration \( i \), pick card from \( \text{deck}[i] \) through \( \text{deck}[N-1] \) at random, with each card equally likely.
- Exchange it with \( \text{deck}[i] \).

```java
int N = deck.length; \\
for (int i = 0; i < N; i++) \\
{ \\
  int r = i + (int) (Math.random() * (N-i)); \\
  String t = deck[r]; \\
  deck[r] = deck[i]; \\
  deck[i] = t;
}
```
Coupon Collector Problem

**Coupon collector problem.** Given $N$ different card types, how many do you have to collect before you have (at least) one of each type?  

Assuming each possibility is equally likely for each card that you collect.

**Simulation algorithm.** Repeatedly choose an integer $i$ between 0 and $N-1$. Stop if we’ve already collected a card of type $i$.

**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 collected a card of type $i$.

---

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

```java
public class Shuffle {
    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 SUITS = suit.length;
        int RANKS = rank.length;
        int N = SUITS * RANKS;
        // create the deck
        String[] deck = new String[N];
        for (int i = 0; i < RANKS; i++)
            for (int j = 0; j < SUITS; j++)
                deck[RANKS * i + j] = rank[i] + " of " + suit[j];
        // shuffle the deck
        for (int i = 0; i < N; i++) {
            int r = i + (int)(Math.random() * (N-i));
            String t = deck[i];
            deck[i] = deck[r];
            deck[r] = t;
        }
        // print results
        for (int i = 0; i < N; i++)
            System.out.println(deck[i]);
    }
}
```

---

**Shuffling a Deck of Cards**

```java
% java Shuffle
5 of Clubs
Jack of Hearts
9 of Spades
10 of Spades
9 of Clubs
7 of Spades
6 of Diamonds
7 of Hearts
7 of Clubs
4 of Spades
Queen of Diamonds
10 of Hearts
5 of Diamonds
Jack of Clubs
Ace of Hearts
...
5 of Spades
```

---

**Coupon Collector: Java Implementation**

```java
public class CouponCollector {
    public static void main(String[] args) {
        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 i = (int)(Math.random() * N);
            cardcnt++;
            if (!found[i]) valcnt++;
            found[i] = true;
        }
        System.out.println(cardcnt);
    }
}
```
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</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>F</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>F</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>F</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>T</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>T</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>T</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>T</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>1</td>
<td>T</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>T</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

Challenge. Debugging with arrays requires tracing many variables.

Coupon Collector: Mathematical Context

Coupon collector problem. 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 + ... + 1/N).

Ex. N = 30 baseball teams. Expect to wait ~ 120 years before all teams win a World Series. Under idealized assumptions

Coupon Collector: Scientific Context

Q. Given a sequence from nature, does it have same 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.

Compile-Time Initialization

Compile-time initialization. Initialize 2D array by listing values.

```java
double[][] p = {
    { .02, .92, .02, .02, .02 },
    { .02, .02, .32, .32, .32 },
    { .92, .02, .02, .02, .02 },
    { .47, .02, .47, .02, .02 },
};
```

Array access. Use `a[i][j]` to access element in row `i` and column `j`.

Zero-based indexing. Row and column indices start at 0.

```java
int M = 6, N = 3;
double[][] a = new double[M][N];
for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++)
        a[i][j] = 0.0;
```

Matrix Multiplication

Matrix multiplication. Given two N-by-N matrices `a` and `b`, define `c` to be the N-by-N matrix where `c[i][j]` is the dot product of the `i`th row of `a` and the `j`th row of `b`.

```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];
```
Hyperlink Structure of Web

**Relevance.** Use webpage content to determine its relevance to query.

**Importance.** Use hyperlink structure of Web to determine importance of web pages, independent of query.

Random Surfer and Matrix Multiplication

**Q.** What is prob that surfer moves from page $i$ to page $j$ in two steps?

**A.** $P^2 = P \times P$. [Matrix multiplication!]

Random Surfer Model

**90-10 rule.** Web surfer chooses next page:
- 90% of the time surfer clicks random hyperlink.
- 10% of the time surfer types a random page.

**Caveat.** Very crude, but useful, model of reality.

**Transition matrix.** $p[i][j]=$ prob that surfer moves from page $i$ to $j$.

Random Surfer: Mathematical Context

**Q.** What is prob that surfer moves from page $i$ to page $j$ in the limit?

**A.** $\lim_{k \to \infty} P^k = P \times P \times \ldots \times P$.

**Mixing theorem.** $P^k$ converges as $k$ approaches infinity. Moreover, all rows are equal.

For our random surfer model:

- Fraction of time surfer spends on page $j$ is independent of starting point.
- For surfer on page 1 will go to page 2 next 32% of the time.

Squaring a Markov chain

$$
\begin{align*}
P & = \begin{pmatrix}
0.02 & 0.92 & 0.02 & 0.02 \\
0.02 & 0.32 & 0.32 & 0.32 \\
0.92 & 0.02 & 0.02 & 0.02 \\
0.47 & 0.04 & 0.45 & 0.02 \\
\end{pmatrix} \\
\end{align*}
$$

$$
\begin{align*}
P^2 & = \begin{pmatrix}
0.02 & 0.04 & 0.04 & 0.04 \\
0.02 & 0.04 & 0.04 & 0.04 \\
0.92 & 0.04 & 0.04 & 0.04 \\
0.47 & 0.04 & 0.45 & 0.02 \\
\end{pmatrix} \\
\end{align*}
$$
Random Surfer: Scientific Context

*Google's PageRank™ algorithm.* [Sergey Brin and Larry Page, 1998]
- Rank importance of pages based on hyperlink structure of Web, using 90-10 rule.
- Revolutionized access to world's information.

```
0 : .269
1 : .262
2 : .227
3 : .143
4 : .099
```

**Page ranks**

**Scientific challenges.** Cope with 4 billion-by-4 billion matrix!
- Need **data structures** to enable computation.
- Need **linear algebra** to fully understand computation.

---

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.