1.4 Arrays
A Foundation for Programming

any program you might want to write

objects

functions and modules

graphics, sound, and image I/O

arrays

conditionals and loops

Math text I/O

primitive data types assignment statements

store and manipulate huge quantities of data
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.
- 11 thousand undergrads at UM.
- 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.
Many Variables of the Same Type

Goal. 10 variables of the same type.

```c
// 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 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 \( a[i] \).
- Array indices start at 0.

```java
int N = 10;
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
```
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 \( a[i] \).
- Array indices start at 0.

```java
int N = 10;
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;
double[] a = new double[N]; // declare, create, init
```
Dot product. Given two vectors $x[]$ and $y[]$ of length $N$, their dot product (projection) is the sum of the products of their corresponding components.

```java
double[] x = { 0.3, 0.6, 0.1 };
double[] y = { 0.5, 0.1, 0.4 };
double sum = 0.0;
for (int i = 0; i < N; i++) {
    sum += x[i]*y[i];
}
```

<table>
<thead>
<tr>
<th>$i$</th>
<th>$x[i]$</th>
<th>$y[i]$</th>
<th>$x[i]*y[i]$</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.30</td>
<td>0.50</td>
<td>0.15</td>
<td>0.15</td>
</tr>
<tr>
<td>1</td>
<td>0.60</td>
<td>0.10</td>
<td>0.06</td>
<td>0.21</td>
</tr>
<tr>
<td>2</td>
<td>0.10</td>
<td>0.40</td>
<td>0.04</td>
<td>0.25</td>
</tr>
</tbody>
</table>
# Array Processing Code

<table>
<thead>
<tr>
<th>Task</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>create an array with random values</td>
<td><code>double[] a = new double[N];</code></td>
</tr>
<tr>
<td></td>
<td><code>for (int i = 0; i &lt; N; i++)</code></td>
</tr>
<tr>
<td></td>
<td><code>a[i] = Math.random();</code></td>
</tr>
<tr>
<td>print the array values, one per line</td>
<td><code>for (int i = 0; i &lt; N; i++)</code></td>
</tr>
<tr>
<td></td>
<td><code>System.out.println(a[i]);</code></td>
</tr>
<tr>
<td>find the maximum of the array values</td>
<td><code>double max = Double.NEGATIVE_INFINITY;</code></td>
</tr>
<tr>
<td></td>
<td><code>for (int i = 0; i &lt; N; i++)</code></td>
</tr>
<tr>
<td></td>
<td><code>if (a[i] &gt; max) max = a[i];</code></td>
</tr>
<tr>
<td>compute the average of the array values</td>
<td><code>double sum = 0.0;</code></td>
</tr>
<tr>
<td></td>
<td><code>for (int i = 0; i &lt; N; i++)</code></td>
</tr>
<tr>
<td></td>
<td><code>sum += a[i];</code></td>
</tr>
<tr>
<td></td>
<td><code>double average = sum / N;</code></td>
</tr>
<tr>
<td>copy to another array</td>
<td><code>double[] b = new double[N];</code></td>
</tr>
<tr>
<td></td>
<td><code>for (int i = 0; i &lt; N; i++)</code></td>
</tr>
<tr>
<td></td>
<td><code>b[i] = a[i];</code></td>
</tr>
<tr>
<td>reverse the elements within an array</td>
<td><code>for (int i = 0; i &lt; N/2; i++)</code></td>
</tr>
<tr>
<td></td>
<td><code>{</code></td>
</tr>
<tr>
<td></td>
<td><code>double temp = b[i];</code></td>
</tr>
<tr>
<td></td>
<td><code>b[i] = b[N-1-i];</code></td>
</tr>
<tr>
<td></td>
<td><code>b[N-i-1] = temp;</code></td>
</tr>
<tr>
<td></td>
<td><code>}</code></td>
</tr>
</tbody>
</table>
Activity 1

Given two vectors of length N that are represented with one-dimensional arrays, write a code fragment (Dr. Java Interactions) that computes the Euclidean distance between them (the square root of the sums of the squares of the differences between corresponding entries).
Shuffling a Deck
Setting Array Values at Compile Time

**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]);
```
Setting Array Values at Run Time

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

```java
String[] deck = new String[52];
for (int i = 0; i < 13; i++)
    for (int j = 0; j < 4; j++)
        deck[4*i + j] = rank[i] + " 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
two of diamonds
two of hearts
two of spades
three of clubs
...

B. two of clubs
three of clubs
four of clubs
five of clubs
six of clubs
...

Q. In what order does it output them?
Shuffling

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

Shuffling algorithm.
- In iteration $i$, pick random card from $\text{deck}[i]$ through $\text{deck}[N-1]$, 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;
}
```
Shuffling a Deck of Cards

```java
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 SUITS = suit.length;
        int RANKS = rank.length;
        int N = SUITS * RANKS;

        String[] deck = new String[N];
        for (int i = 0; i < RANKS; i++) {
            for (int j = 0; j < SUITS; j++)
                deck[SUITS*i + j] = rank[i] + " of " + suit[j];
        }

        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;
        }

        for (int i = 0; i < N; i++)
            System.out.println(deck[i]);
    }
}
```
Shuffling a Deck of Cards

% java Deck
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

% java Deck
10 of Diamonds
King of Spades
2 of Spades
3 of Clubs
4 of Spades
Queen of Clubs
2 of Hearts
7 of Diamonds
6 of Spades
Queen of Spades
3 of Spades
Jack of Diamonds
6 of Diamonds
8 of Spades
9 of Diamonds
...
10 of Spades
Activity 2

Write a program that takes a command-line argument N and prints N poker hands (five cards each) from a shuffled deck, separated by blank lines.
Coupon Collector
**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?

**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 $i$. 

assuming each possibility is equally likely for each card that you collect
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**

**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 F</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>F F T F F F</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>T F T F F F</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>T F T T F F</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>T F T F T F</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>T T T F T F</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>T T T F T F</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>T T T F T T</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>0</td>
<td>T T T F T T</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>T T T F T T</td>
<td>5</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>T T T T T T</td>
<td>6</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 \((1 + 1/2 + 1/3 + \ldots + 1/N)\).

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

\(\text{under idealized assumptions}\)
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.
Activity 3

Run some experiments to validate the classical mathematical results that the expected number of coupons needed to collect N values is about $N H_N$. For example, if you are observing the cards carefully at the blackjack table (and the dealer has enough decks randomly shuffled together), you will wait until about 235 cards are dealt, on average, before seeing every card value.

$$H_N = 1/1 + \frac{1}{2} + 1/3 + \frac{1}{4} .... + 1/N$$

(Graded activity in Lab)
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.

Reference: Botstein & Brown group
Two Dimensional Arrays in Java

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 = 10;
int 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;
    }
}
```

A 10-by-3 array
Setting 2D Array Values at Compile Time

Initialize 2D array by listing values.

```cpp
double[][[]] p =
{
    { .02, .92, .02, .02, .02 },
    { .02, .02, .32, .32, .32 },
    { .02, .02, .02, .92, .02 },
    { .92, .02, .02, .02, .02 },
    { .47, .02, .47, .02, .02 },
};
```
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 $a[i][j] + b[i][j]$.

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];
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 column 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];
```

all values initialized to 0
Array Challenge 2

Q. How many scalar multiplications multiply two N-by-N matrices?

A. N  B. $N^2$  C. $N^3$  D. $N^4$

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];
Self-Avoiding Walk
Self-Avoiding Walk

**Model.**
- N-by-N lattice.
- Start in the middle.
- Randomly move to a neighboring intersection, avoiding all previous intersections.

**Applications.** Polymers, statistical mechanics, etc.

**Q.** What fraction of time will you escape in an 5-by-5 lattice?
**Q.** In an N-by-N lattice?
**Q.** In an N-by-N-by-N lattice?
Self-Avoiding Walk: Implementation

```java
public class SelfAvoidingWalk {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]); // lattice size
        int T = Integer.parseInt(args[1]); // number of trials
        int deadEnds = 0; // trials resulting in dead end

        for (int t = 0; t < T; t++) {
            boolean[][] a = new boolean[N][N]; // intersections visited
            int x = N/2, y = N/2; // current position

            while (x > 0 && x < N-1 && y > 0 && y < N-1) {
                if (a[x-1][y] && a[x+1][y] && a[x][y-1] && a[x][y+1]) {
                    deadEnds++;
                    break;
                }

                a[x][y] = true; // mark as visited

                double r = Math.random(); // take a random unvisited step
                if (r < 0.25) { if (!a[x+1][y]) x++; }
                else if (r < 0.50) { if (!a[x-1][y]) x--; }
                else if (r < 0.75) { if (!a[x][y+1]) y++; }
                else if (r < 1.00) { if (!a[x][y-1]) y--; }
            }

            System.out.println(100*deadEnds/T + "% dead ends");
        }
    }
}
```
Self-Avoiding Walks
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.
1.4 Arrays: Extra Slides
"You’re always off by 1 in this business." - J. Morris

http://imgs.xkcd.com/comics/donald_knuth.png
Memory Representation

Memory representation. Maps directly to physical hardware.

Consequences.
- Arrays have fixed size.
- Accessing an element by its index is fast.
- Arrays are pointers.

2D array. Array of arrays.

Consequences. Arrays can be ragged.
Sieve of Eratosthenes
Sieve of Eratosthenes

Prime. An integer > 1 whose only positive factors are 1 and itself.
Ex. 2, 3, 5, 7, 11, 13, 17, 23, ...

Prime counting function. \( \pi(N) = \# \text{ primes} \leq N. \)
Ex. \( \pi(17) = 7. \)

Sieve of Eratosthenes.
- Maintain an array isPrime[] to record which integers are prime.
- Repeat for \( i=2 \) to \( \sqrt{N} \)
  - if \( i \) is not still marked as prime
    - \( i \) is not prime since we previously found a factor
  - if \( i \) is marked as prime
    - \( i \) is prime since it has no smaller factors
    - mark all multiples of \( i \) to be non-prime
Sieve of Eratosthenes

**Prime.** An integer > 1 whose only positive factors are 1 and itself.
**Ex.** 2, 3, 5, 7, 11, 13, 17, 23, ...

**Prime counting function.** \( \pi(N) = \# \text{primes} \leq N. \)
**Ex.** \( \pi(17) = 7. \)

<table>
<thead>
<tr>
<th>i</th>
<th>isPrime</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>T</td>
</tr>
<tr>
<td>3</td>
<td>F</td>
</tr>
<tr>
<td>4</td>
<td>T</td>
</tr>
<tr>
<td>5</td>
<td>F</td>
</tr>
<tr>
<td>6</td>
<td>T</td>
</tr>
<tr>
<td>7</td>
<td>T</td>
</tr>
<tr>
<td>8</td>
<td>T</td>
</tr>
<tr>
<td>9</td>
<td>T</td>
</tr>
<tr>
<td>10</td>
<td>T</td>
</tr>
<tr>
<td>11</td>
<td>T</td>
</tr>
<tr>
<td>12</td>
<td>T</td>
</tr>
<tr>
<td>13</td>
<td>T</td>
</tr>
<tr>
<td>14</td>
<td>T</td>
</tr>
<tr>
<td>15</td>
<td>T</td>
</tr>
<tr>
<td>16</td>
<td>T</td>
</tr>
<tr>
<td>17</td>
<td>T</td>
</tr>
<tr>
<td>18</td>
<td>T</td>
</tr>
<tr>
<td>19</td>
<td>T</td>
</tr>
<tr>
<td>20</td>
<td>T</td>
</tr>
<tr>
<td>21</td>
<td>T</td>
</tr>
<tr>
<td>22</td>
<td>T</td>
</tr>
<tr>
<td>23</td>
<td>T</td>
</tr>
<tr>
<td>24</td>
<td>T</td>
</tr>
<tr>
<td>25</td>
<td>T</td>
</tr>
</tbody>
</table>
Sieve of Eratosthenes: Implementation

```java
public class PrimeSieve {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        // initially assume all integers are prime
        boolean[] isPrime = new boolean[N+1];
        for (int i = 2; i <= N; i++)
            isPrime[i] = true;

        // mark non-primes <= N using Sieve of Eratosthenes
        for (int i = 2; i*i <= N; i++)
            if (isPrime[i]) {
                for (int j = i; i*j <= N; j++)
                    isPrime[i*j] = false;
            }

        // count primes
        int primes = 0;
        for (int i = 2; i <= N; i++)
            if (isPrime[i]) primes++;
        StdOut.println("The number of primes <= " + N + " is " + primes);
    }
}
```