4. Arrays

- Basic concepts
- Typical array-processing code
- Two-dimensional arrays

Basic building blocks 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

Your first data structure

A **data structure** is an arrangement of data that enables efficient processing by a program.

An **array** is an **indexed** sequence of values of the same type.

**Examples.**
- 52 playing cards in a deck.
- 100 thousand students in an online class.
- 1 billion pixels in a digital image.
- 4 billion nucleotides in a DNA strand.
- 73 billion Google queries per year.
- 86 billion neurons in the brain.
- 50 trillion cells in the human body.
- $6.02 \times 10^{23}$ particles in a mole.

**Main purpose.** Facilitate storage and manipulation of data.
Processing many values of the same type

10 values, without arrays

```java
double a0 = 0.0;
double a1 = 0.0;
double a2 = 0.0;
double a3 = 0.0;
double a4 = 0.0;
double a5 = 0.0;
double a6 = 0.0;
double a7 = 0.0;
double a8 = 0.0;
double a9 = 0.0;
...
double x = a4 + a8;
```

10 values, with an array

```java
double[] a;
a = new double[10];
...
a[4] = 3.0;
...
a[8] = 8.0;
...double x = a[4] + a[8];
```

1 million values, with an array

```java
double[] a;
a = new double[1000000];
...
a[234567] = 3.0;
...
a[876543] = 8.0;
...double x = a[234567] + a[876543];
```

tedious and error-prone code

an easy alternative

scales to handle huge amounts of data

Memory representation of an array

An array is an indexed sequence of values of the same type.

A computer’s memory is also an indexed sequence of memory locations.

• Each primitive type value occupies a fixed number of locations.
• Array values are stored in contiguous locations.

Critical concepts

• The array name a refers to the first value in the array.
• Indices start at 0.
• Given i, the operation of accessing the value a[i] is extremely efficient.
• The assignment b = a makes the names b and a refer to the same array.

Java language support for arrays

Basic support

<table>
<thead>
<tr>
<th>operation</th>
<th>typical code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declare an array</td>
<td>double[] a;</td>
</tr>
<tr>
<td>Create an array of a given length</td>
<td>a = new double[1000];</td>
</tr>
<tr>
<td>Refer to an array entry by index</td>
<td>a[i] = b[j] + c[k];</td>
</tr>
<tr>
<td>Refer to the length of an array</td>
<td>a.length;</td>
</tr>
</tbody>
</table>

Initialization options

<table>
<thead>
<tr>
<th>operation</th>
<th>typical code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explicitly set all entries to some value</td>
<td>for (int i = 0; i &lt; a.length; i++) a[i] = 0.0;</td>
</tr>
<tr>
<td>Default initialization to 0 for numeric types</td>
<td>a = new double[1000];</td>
</tr>
<tr>
<td>Declare, create and initialize in one statement</td>
<td>double[] a = new double[1000];</td>
</tr>
<tr>
<td>Initialize to literal values</td>
<td>double[] x = { 0.3, 0.6, 0.1 };</td>
</tr>
</tbody>
</table>

Importantly, the cost of creating an array is proportional to its length.

Equivalent in Java

Copying an array

To copy an array, create a new array, then copy all the values.

```java
double[] b = new double[a.length];
for (int i = 0; i < a.length; i++)
b[i] = a[i];
```

Important note: The code b = a does not copy an array (it makes b and a refer to the same array).
Q. What does the following code print?

```java
public class PArray1 {
    public static void main(String[] args) {
        int[] a;
        int[] b = new int[a.length];
        b = a;
        for (int i = 1; i < b.length; i++)
            b[i] = 1;
        for (int i = 0; i < a.length; i++)
            System.out.print(a[i] + " ");
        System.out.println();
        for (int i = 0; i < b.length; i++)
            System.out.print(b[i] + " ");
        System.out.println();
    }
}
```

Programming with arrays: typical examples

Access command-line args in system array
```
int stake = Integer.parseInt(args[0]);
int goal = Integer.parseInt(args[1]);
int trials = Integer.parseInt(args[2]);
```  
Copy to another array
```
double[] b = new double[N];
for (int i = 0; i < N; i++)
    b[i] = a[i];
```  
Create an array with \( N \) random values
```
double[] a = new double[N];
for (int i = 0; i < N; i++)
    a[i] = Math.random();
```  
Print array values, one per line
```
for (int i = 0; i < N; i++)
    System.out.println(a[i]);
```  
Compute the average of array values
```
double sum = 0.0;
for (int i = 0; i < N; i++)
    sum += a[i];
    double average = sum / N;
```  
Find the maximum of array values
```
double max = a[0];
for (int i = 1; i < N; i++)
    if (a[i] > max) max = a[i];
```  
Programming with arrays: typical bugs

Array index out of bounds
```
double[] a = new double[10];
for (int i = 1; i <= 10; i++)
    a[i] = Math.random();
```

Uninitialized array
```
double[] a;
for (int i = 0; i < 10; i++)
    a[i] = Math.random();
```

Undeclared variable
```
a = new double[10];
for (int i = 0; i < 10; i++)
    a[i] = Math.random();
```  
What type of data does \( a \) refer to?
Example of array use: create a deck of cards

Define three arrays
- Ranks.
- Suits.
- Full deck.

Use nested for loops to put all the cards in the deck.

Pop quiz 2 on arrays
Q. What happens if the order of the for loops in Deck is switched?

Pop quiz 3 on arrays
Q. Change Deck to put the cards in rank order in the array.
Array application: take a card, any card

**Problem:** Print a random sequence of \( N \) cards.

**Algorithm**
Take \( N \) from the command line and do the following \( N \) times
- Calculate a random index \( p \) between 0 and 1.
- Print \( \text{deck}[p] \).

**Implementation:** Add this code instead of printing deck in Deck.

```java
for (int i = 0; i < N; i++)
    System.out.println(deck[(int) (Math.random() * 52)]);
```

**Note:** Same method is effective for printing a random sequence from any data collection.

---

Array application: random sequence of cards

```java
public class DrawCards
{
    public static void main(String[] args)
    {
        String[] suit = {"♣", "♦", "♥", "♠");
        String[] deck = new String[52];
        for (int i = 0; i < 13; i++)
            for (int j = 0; j < 4; j++)
                deck[i + 33 * j] = rank[i] + " of " + suit[j];
        for (int i = 0; i < N; i++)
            System.out.println(deck[(int) (Math.random() * 52)]);
    }
}
```

**Note:** Sample is with replacement (same card can appear multiple times).

---

Array application: shuffle and deal from a deck of cards

**Problem:** Print \( N \) random cards from a deck.

**Algorithm:** Shuffle the deck, then deal.
- Consider each card index \( i \) from 0 to 51.
- Calculate a random index \( p \) between \( i \) and 51.
- Exchange \( \text{deck}[i] \) with \( \text{deck}[p] \).
- Print the first \( N \) cards in the deck.

```java
for (int i = 0; i < N; i++)
    
### Implementation
```

```java
for (int i = 0; i < 52; i++)
    {
        int p = i + (int) (Math.random() * (52-i));
        String t = deck[p];
        deck[p] = deck[i];
        deck[i] = t;
    }
for (int i = 0; i < N; i++) System.out.println(deck[i]);
```
Array application: shuffle a deck of 10 cards (trace)

```java
for (int i = 0; i < 10; i++)
    int p = i + (int) (Math.random() * (10-i));
    String t = deck[p];
    deck[p] = deck[i];
    deck[i] = t;
```

Q. Why does this method work?
- Uses only exchanges, so the deck after the shuffle has the same cards as before.
- \(\frac{N}{i}\) equally likely values for \(deck[i]\).
- Therefore \(N(x-N)\) equally likely values for \(deck[\]i\].

Initial order is immaterial.

**Note:** Same method is effective for randomly rearranging any type of data.

Array application: shuffle and deal from a deck of cards

```java
int p = i + (int) (Math.random() * (10-i));
String t = deck[p];
deck[p] = deck[i];
deck[i] = t;
```

**Coupon collector**

**Coupon collector problem**
- \(M\) different types of coupons.
- Collector acquires random coupons, one at a time, each type equally likely.

Q. What is the expected number of coupons needed to acquire a full collection?

**Example:** Collect all ranks in a random sequence of cards (\(M = 13\)).

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Collection</th>
</tr>
</thead>
<tbody>
<tr>
<td>9♣ 5♣ A♦ 10♦ 2♣ A♣ 10♥ Q♣ 3♣ 9♥ 5♥ 6♥ 7♥ 8♥</td>
<td>2 3 4 5 6 7 8 9 10 J Q K A</td>
</tr>
</tbody>
</table>

22 cards needed to complete collection

**Array application: coupon collector**

**Coupon collector simulation**
- Generate random int values between 0 and \(M-1\).
- Count number used to generate each value at least once.

Key to the implementation
- Create a boolean array of length \(M\). (Initially all false by default.)
- When \(r\) generated, check \(rth\) value in the array.
  - If \(true\), ignore it (not new).
  - If \(false\), count it as new (and set \(rth\) entry to \(true\))

```java
public class Coupon
    public static void main(String[] args)
        int cardcnt = 0; // number of cards collected
        int cnt = 0; // number of distinct cards
        boolean[] found = new boolean[M];
        while (Cnt < M)
            if (r = (int) (Math.random() * M))
            if (!found[r])
                found[r] = true;
        System.out.println(cardcnt);
```
Array application: coupon collector (trace for \( M = 6 \))

<table>
<thead>
<tr>
<th>( r )</th>
<th>( 0 )</th>
<th>( 1 )</th>
<th>( 2 )</th>
<th>( 3 )</th>
<th>( 4 )</th>
<th>( 5 )</th>
<th>( \text{cnt} )</th>
<th>( \text{cardcnt} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( F )</td>
<td>( F )</td>
<td>( F )</td>
<td>( F )</td>
<td>( F )</td>
<td>( F )</td>
<td>( 0 )</td>
<td>( 0 )</td>
<td></td>
</tr>
<tr>
<td>( 2 )</td>
<td>( F )</td>
<td>( F )</td>
<td>( T )</td>
<td>( F )</td>
<td>( F )</td>
<td>( 1 )</td>
<td>( 1 )</td>
<td></td>
</tr>
<tr>
<td>( 0 )</td>
<td>( T )</td>
<td>( T )</td>
<td>( F )</td>
<td>( F )</td>
<td>( F )</td>
<td>( 2 )</td>
<td>( 2 )</td>
<td></td>
</tr>
<tr>
<td>( 4 )</td>
<td>( T )</td>
<td>( T )</td>
<td>( T )</td>
<td>( F )</td>
<td>( T )</td>
<td>( 3 )</td>
<td>( 3 )</td>
<td></td>
</tr>
<tr>
<td>( 0 )</td>
<td>( T )</td>
<td>( T )</td>
<td>( T )</td>
<td>( F )</td>
<td>( F )</td>
<td>( 4 )</td>
<td>( 4 )</td>
<td></td>
</tr>
<tr>
<td>( 1 )</td>
<td>( T )</td>
<td>( T )</td>
<td>( T )</td>
<td>( F )</td>
<td>( F )</td>
<td>( 4 )</td>
<td>( 5 )</td>
<td></td>
</tr>
<tr>
<td>( 2 )</td>
<td>( T )</td>
<td>( T )</td>
<td>( T )</td>
<td>( F )</td>
<td>( F )</td>
<td>( 4 )</td>
<td>( 6 )</td>
<td></td>
</tr>
<tr>
<td>( 5 )</td>
<td>( T )</td>
<td>( T )</td>
<td>( T )</td>
<td>( T )</td>
<td>( F )</td>
<td>( 5 )</td>
<td>( 7 )</td>
<td></td>
</tr>
<tr>
<td>( 0 )</td>
<td>( T )</td>
<td>( T )</td>
<td>( T )</td>
<td>( T )</td>
<td>( T )</td>
<td>( 5 )</td>
<td>( 8 )</td>
<td></td>
</tr>
<tr>
<td>( 1 )</td>
<td>( T )</td>
<td>( T )</td>
<td>( T )</td>
<td>( T )</td>
<td>( T )</td>
<td>( 5 )</td>
<td>( 9 )</td>
<td></td>
</tr>
<tr>
<td>( 3 )</td>
<td>( T )</td>
<td>( T )</td>
<td>( T )</td>
<td>( T )</td>
<td>( T )</td>
<td>( 6 )</td>
<td>( 10 )</td>
<td></td>
</tr>
</tbody>
</table>

Once simulation is debugged, experimental evidence is easy to obtain.

Analogous code for coupon collector, this lecture

```java
public class Collector {
    public static void main(String[] args) {
        int M = Integer.parseInt(args[0]);
        int trials = Integer.parseInt(args[1]);
        int cardcnt = 0;
        boolean[] found = new boolean[M];
        for (int r = (int) (Math.random() * M);
             !found[r]; cardcnt++) {
            int cnt = 0;
            for (int i = 0; i < trials; ++i) {
                int t = stake;
                while (t >= 0) {
                    if (Math.random() < 0.5) ++t;
                    else --t;
                    if (t == goal) ++cnt;
                    System.out.println("wins = "+cnt+" of "+trials);
                }
            }
        }
        System.out.println(cardcnt/trials);
    }
}
```

Simulation, randomness, and analysis (revisited)

• Computer simulation can help validate mathematical analysis.
• Computer simulation can also validate software behavior.

Simulation, randomness, and analysis (revisited)

What is the expected number of coupons needed to acquire a full collection?

A. (known via mathematical analysis for centuries) About \( M \ln M + 0.57722 M \).

<table>
<thead>
<tr>
<th>type</th>
<th>( M )</th>
<th>expected wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>playing card suits</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>playing card ranks</td>
<td>13</td>
<td>41</td>
</tr>
<tr>
<td>baseball cards</td>
<td>1200</td>
<td>9201</td>
</tr>
<tr>
<td>Magic™ cards</td>
<td>12534</td>
<td>125508</td>
</tr>
</tbody>
</table>

Predicted by mathematical analysis

Observed by computer simulation

**Hypothesis.** Centuries-old analysis is correct and Math.random() simulates randomness.
4. Arrays

- Basic concepts
- Examples of array-processing code
- Two-dimensional arrays

Two-dimensional arrays

A two-dimensional array is a doubly-indexed sequence of values of the same type.

Examples
- Matrices in math calculations.
- Grades for students in an online class.
- Outcomes of scientific experiments.
- Transactions for bank customers.
- Pixels in a digital image.
- Geographic data

Main purpose. Facilitate storage and manipulation of data.

Java language support for two-dimensional arrays (basic support)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Typical code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declare a two-dimensional array</td>
<td><code>double[][] a;</code></td>
</tr>
<tr>
<td>Create a two-dimensional array of a given length</td>
<td><code>a = new double[1000][1000];</code></td>
</tr>
<tr>
<td>Refer to an array entry by index</td>
<td><code>a[i][j] = b[i][j] * c[j][k];</code></td>
</tr>
<tr>
<td>Refer to the number of rows</td>
<td><code>a.length;</code></td>
</tr>
<tr>
<td>Refer to the number of columns</td>
<td><code>a[i].length;</code></td>
</tr>
<tr>
<td>Refer to row i</td>
<td><code>a[i]</code></td>
</tr>
</tbody>
</table>

Java language support for two-dimensional arrays (initialization)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Typical code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explicitly set all entries to 0</td>
<td>for (int i = 0; i &lt; a.length; i++)</td>
</tr>
<tr>
<td></td>
<td>for (int j = 0; j &lt; a[i].length; j++)</td>
</tr>
<tr>
<td></td>
<td><code>a[i][j] = 0.0;</code></td>
</tr>
<tr>
<td>Default initialization to 0</td>
<td><code>a = new double[1000][1000];</code></td>
</tr>
<tr>
<td>for numeric types</td>
<td></td>
</tr>
<tr>
<td>Declare, create and initialize</td>
<td><code>double[][] a = new double[1000][1000];</code></td>
</tr>
<tr>
<td>in a single statement</td>
<td></td>
</tr>
<tr>
<td>Initialize to literal values</td>
<td><code>double[][] p = {</code></td>
</tr>
<tr>
<td></td>
<td><code>{ .92, .02, .02, .02, .02 }</code>,</td>
</tr>
<tr>
<td></td>
<td><code>{ .02, .92, .32, .32, .32 }</code>,</td>
</tr>
<tr>
<td></td>
<td><code>{ .02, .02, .02, .02, .02 }</code>,</td>
</tr>
<tr>
<td></td>
<td><code>{ .47, .02, .47, .02, .02 }</code>};</td>
</tr>
</tbody>
</table>
Application of arrays: vector and matrix calculations

Mathematical abstraction: vector
Java implementation: 1D array

### Vector addition

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

### Matrix addition

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];

Pop quiz 4 on arrays

Q. How many multiplications to multiply two \(N\times N\) matrices?

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];

- 1. \(N\)
- 2. \(N^2\)
- 3. \(N^3\)
- 4. \(N^4\)

Self-avoiding random walks

A dog walks around at random in a city, never revisiting any intersection.

Q. Does the dog escape?

**Model:** A random process in an \(N\times N\) lattice
- Start in the middle.
- Move to a random neighboring intersection but do not revisit any intersection.
- Outcome 1 (escape): reach edge of lattice.
- Outcome 2 (dead end): no unvisited neighbors.

Q. What are the chances of reaching a dead end?

Approach: Use Monte Carlo simulation, recording visited positions in an \(N\times N\) array.
Self-avoiding random walks

Simulation, randomness, and analysis (revisited again)

Self-avoiding walk in an \( N \)-by-\( N \) lattice
- Start in the middle.
- Move to a random neighboring intersection (do not revisit any intersection).

Applications
- Model the behavior of solvents and polymers.
- Model the physics of magnetic materials.
- (many other physical phenomena)

Q. What is the probability of reaching a dead end?

A. Nobody knows (despite decades of study).
A. 99+\% for \( N > 100 \) (clear from simulations).

Remark: Computer simulation is often the only effective way to study a scientific phenomenon.

Application of 2D arrays: self-avoiding random walks

```java
public class SelfAvoidingWalk {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        int trials = Integer.parseInt(args[1]);
        int deadEnds = 0;
        for (int t = 0; t < trials; t++) {
            boolean[][] a = new boolean[N][N];
            int x = N/2, y = N/2;
            while (x > 0 && x < N-1 && y > 0 && y < N-1) {
                if ((x-1)[y] && a[x+1][y] && a[x][y+1] && a[x][y-1]) {
                    deadEnds++;
                    break;
                }
            }
            System.out.println((100*deadEnds/trials) + " % dead ends");
        }
    }
}
```

Your first data structure

Arrays: A basic building block in programming
- They enable storage of large amounts of data (values all of the same type).
- With an index, a program can instantly access a given value.
- Efficiency derives from low-level computer hardware organization (stay tuned).

Some applications in this course where you will use arrays:

- LFSR
- Digital images
- N-body simulation
4. Arrays