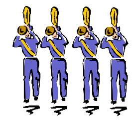
CSE 142

2-D Arrays

Review – One-Dimensional Arrays

- Simple, ordered collections.



- Elements of a particular array all have the same type.
- Size fixed when array created.

```
Person[ ] people = new Person[4];
```

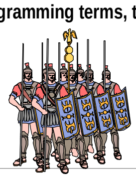
- Indexed access to elements.

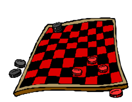
```
people[3] = new Person( );
people[3].moveBy(10, 20);
```


1-D vs. 2-D Arrays

- One-dimensional arrays are very common in programming
  - That's all we used at first

- In everyday life, an array is an regular arrangement, usually rectangular
  - In programming terms, these are two-dimensional arrays





The Game of Life

- Originated by John Conway
- Many interesting variations
- Played on a 2-D board
- Each cell is "alive" or "dead"
- At each time step, a cell looks at its neighbors and may change its own state as a result

Game of Life: Rules

- You can make up your own rules!
- Typical rules:
  - 1. If a cell is surrounded by too many live cells, it dies
  - 2. If a dead cell is surrounded by enough dead cells, it comes to life
- I.e., given a particular cell,
  - let liveNeighborCount = number of adjacent cells which are alive
  - If liveNeighborCount > 7, it dies
  - If liveNeighborCount < 4, it comes to life
  - Otherwise, it doesn't change

Implementing The Game of Life

- 2-D arrays are a natural data structure for the game "board"
- *int[ ] [ ] board;*
  - Each array elements represents a cell on the board
  - Could use boolean[ ] [ ] instead as long as there are only two cell states

```
/** Construct a board with all cells "dead".*/
public GameOfLife() {
  board = new int[ROWS][COLS];
  for (int r = 0; r < ROWS; r++) {
    for (int c = 0; c < COLS; c++) {
      board[r][c] = DEAD;
    }
  }
}
```


2-D Arrays

- Suppose we want to represent a picture
- Want a rectangular, ***2-dimensional*** matrix of Pixel objects
  - Each Pixel contains a red, green, and blue color component
- We can create an array with 2 dimensions to hold the picture
  - **Type pattern:** <*elem type*>[ ][ ]
  - **New expr pattern:** new <*elem type*>[<*dim 1 size*>][<*dim 2 size*>]
  - **Access expr/assignment pattern:** <*array*>[<*dim 1 index*>][<*dim 2 index*>]

```
Pixel[ ][ ] picture = new Pixel [40] [60];
picture[0][0] = new Pixel(128, 0, 255);     // parameters are red, green, blue intensities
```


Picture

2-D Array = Array of Arrays

- A 2-D array is really just an array of arrays
  (In languages like FORTRAN and C/C++, this isn't true)
- It's possible to manipulate each row array separately

```
Pixel[ ][ ] picture = new Pixel[40][60];
picture[0][0] = new Pixel(0, 0, 255);
…
Pixel[ ] firstRow = picture[0];
firstRow[0] = new Pixel(255, 0, 0);
```

- What do the following evaluate to?

```
picture.length
firstRow.length
picture[0][0].length
```


2-D Array Traversal

- Typical traversal is to go through the rows and, for each row, go through the columns. Called "row-major order"

```
/** Create new picture pixels with given rgb color */
public void initialize(Pixel[ ][ ] picture, int r, int g, int b) {
    for (int row = 0; row < picture.length; row++) {
        for (int col = 0; col < picture[row].length; col++) {
            picture[row][col] = new Pixel(r, g, b);
        }
    }
}
```

  - Notice how the upper bounds of the two loops are computed
- "Column-major" order is also possible – go through the columns and, for each column, go through the rows

Exercise: Shift Picture to Left

```
// Copy colors one cell to the left, setting last column to white
public void shiftLeft(Pixel[ ][ ] picture) {

    for (int row = 0; row < ___________________________ ; row++) {

        for (int col = 0; col < ___________________________ ; col++) {

            picture[row][col] = _______________________________________ ;

        }

    }
}
```

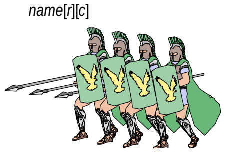

Exercise: Shift Picture Down

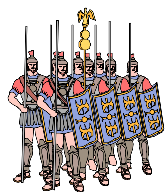
```
// Copy colors one cell downwards, setting first row to white
public void shiftDown(Pixel[ ][ ] picture) {






}
```

- Hint: row-major order might not be the right approach.

Summary

- **2-D arrays**
  - **In Java, just an array of arrays**
    (Similar concepts in other languages)
  - **Syntax is extension of 1-D array case**
    type[ ][ ] *name* = new *type*[*nRows*][*nCols*]
    name[*r*][*c*]





(c) 2001-3, University of Washington