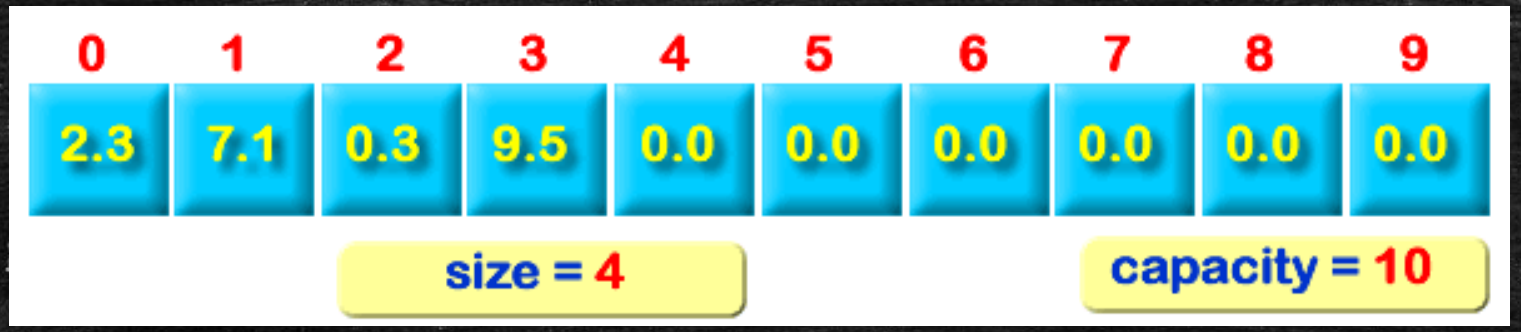
Partially Filled & 2D Arrays

CS 150 – C++ Programming I
Lecture 21

Introducing Partially-filled Arrays

- You'll often use only a portion of an array
  - Since array size can't change, you plan for the worst case
  - Make allocated array large enough to hold maximum data



  - `capacity` represents the "worst case"
  - `size` is the effective size of the array

- Exercise: Partially-Filled Array Basics

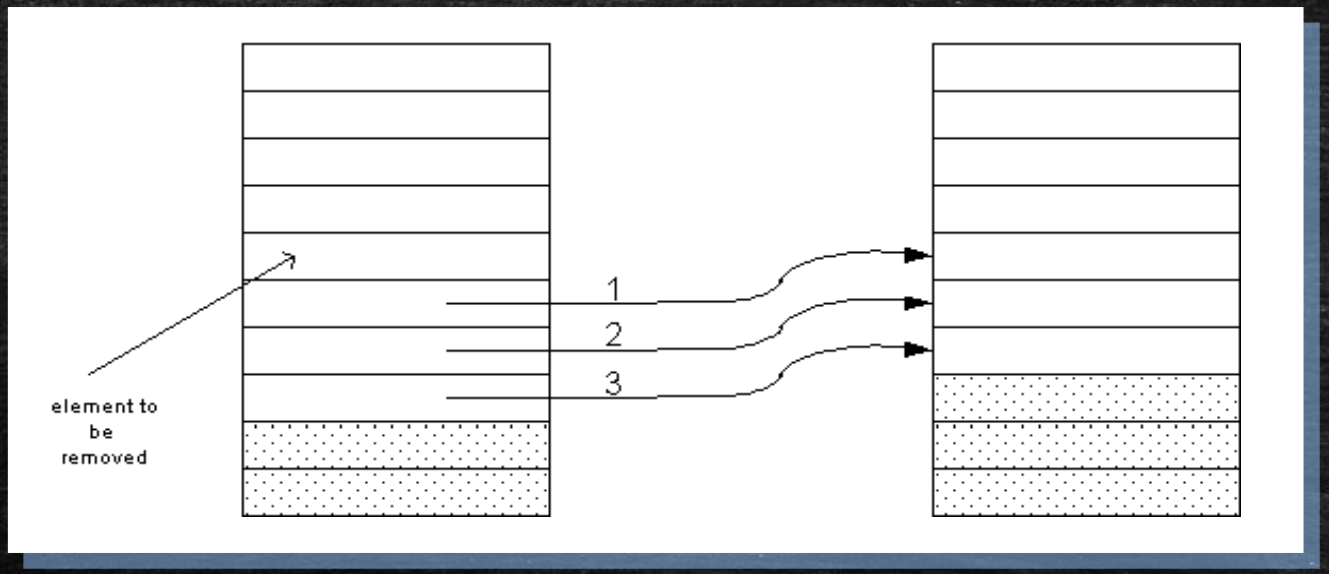
Filling or Reading from Input

- Expand (`push_back`) to a partially-filled array like this:
  - `a[size] = value;`
    `size++;`
- Only works if `size < capacity`

Printing

- Can't use `operator<<`, array doesn't know size
  - Write `toString()`, only printing valid elements
- Fencepost algorithm: want values separated
  - 1. Print the delimiter `"["`
  - 2. If there are any elements, print the first one
  - 3. Use a loop to print the remaining elements
    - Precede each by the separator
  - 4. After the *if*, print the closing delimiter `"]"`

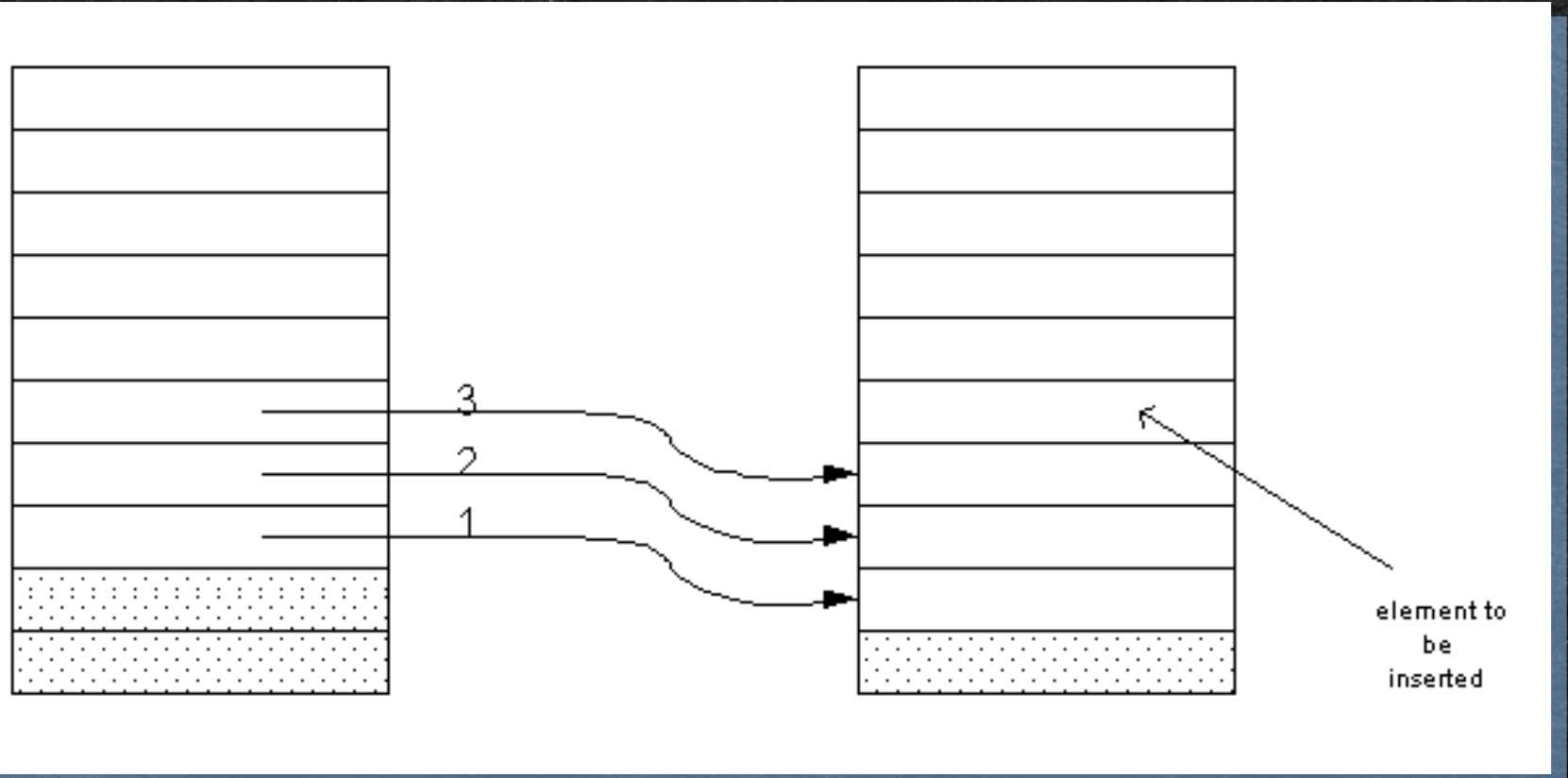
Deleting Elements



- Exercise: deleting all elements from an array
  - Step 1: find element that matches `value`
  - Step 2: Decrement `size`, shift left, update index
  - Step 3: Return elements removed

Inserting Into an Array

- Shift right all items from insertion point
  - Then, insert the new item



Inserting Elements

- Exercise: insert keeping elements in order
  - Step 1: If `size >= capacity` return `nullptr`
  - Step 2: Let `pos` = first element larger than value
  - Step 3: Copy elements from `size` to `pos` (right shift)
  - Step 4: Add element at `pos`, increment `size`
  - Step 5: Return address of added element

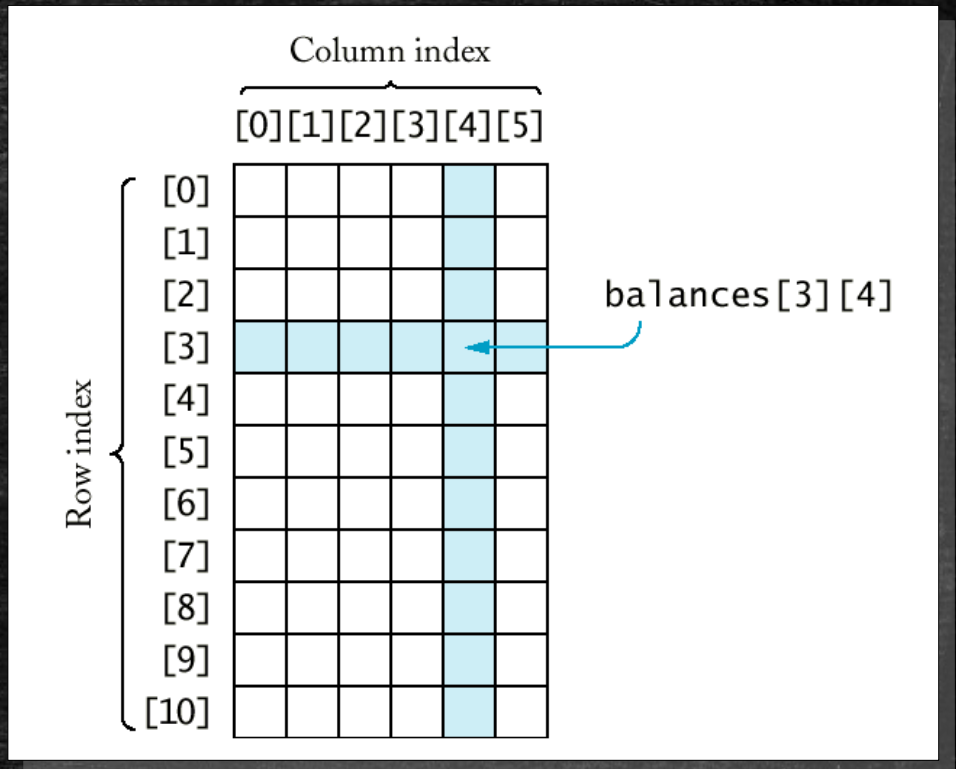
Multidimensional Arrays

- Think rows & columns: AKA a matrix
  - Stored linearly as "array of arrays"
  - Second row follows last element in first row
- Syntax for creating 2D arrays

```
const int ROWS = 2;
const int COLS = 3;
double a2D[ROWS][COLS];          // uninitialized
double b2D[ROWS][COLS]{};        // default init
double c2D[][COLS] = {
    { 1, 2, 3},
    { 4, 5, 6}
};
```

Working with 2D Arrays

- Access elements using `[row][col]`
  - Called row major order
- Access outside array bounds is undefined behavior (UB)
  - `balances[0][6]`
  - Normally works, but you should probably avoid doing it
- Calculate rows, columns like this:
  - `const size_t ROWS = sizeof(a) / sizeof(a[0]), COLS = sizeof(a[0]) / sizeeof(a[0][0])`



2D Arrays and Functions

- When passing a 2D array to a function, specify the number of columns as a constant
  - `int odds(const int n[][COLS], size_t rows)`
  - `int odds(const int n[][] ...) // illegal`
  - If you only want to process some of the columns, then pass an additional argument for the column
- Can pass a single row to a 1D function like this:
  - ```
    int a[3][5] = {...};
    process(a[0], 5); // first row passed
    ```
- Exercise: *a2d* and the *average()* functions
    ```