Today

- Quiz 3
- Resizable Arrays

Assignments

- HW-2 due fri
- Exercises ...
The ArraySeq class

For HW-3, you will implement a “resizable array” (called ArraySeq)

- Similar to Java’s ArrayList and C++’s Vector classes

Resizable arrays address fixed-sized issue of normal arrays

- Arrays in C++ are a fixed size at creation
- Once we reach the capacity of the array, can’t add more elements

Basic idea of a resizable array:

- start with an initial capacity (in our case, 0 slots)
- once all slots full, next add dynamically “resizes” the array

Resizing the array involves four steps:

1. create a new array with twice the current array’s capacity (e.g., 1, 2, 4, 8, ...)
2. copies all current elements to the new array
3. deletes the current array
4. sets the current array to the new array

Resizable arrays introduce a “trade off” in terms of efficiency

- Solves the fixed-size problem
- Maintains fast access to elements
- (Slightly) increases the cost of adding (because of resizing)
The basic structure of ArraySeq for HW-3

```cpp
template<typename T>
class ArraySeq : public Sequence<T> {

public:
    // constructor, copy, move, destructor
    ArraySeq();
    ArraySeq(const ArraySeq& rhs);
    ArraySeq(ArraySeq&& rhs);
    ArraySeq& operator=(const ArraySeq& rhs);
    ArraySeq& operator=(ArraySeq&& rhs);
    ~ArraySeq();

    // sequence operations
    int size() const;
    bool empty() const;
    void clear();
    T& operator[](int index);
    const T& operator[](int index) const;
    void insert(const T& elem, int index);
    void erase(int index);
    bool contains(const T& elem) const;

private:

    // the underlying (resizable) array
    T* array = nullptr;

    // size of list
    int count = 0;

    // max capacity of the array
    int capacity = 0;

    // helper to double capacity of the array
    void resize();

};
```
Q: Why does the array have the following type?

\[
T^* \text{ array } = \text{nullptr};
\]

Q: How do we dynamically allocate an array of size \( n \)?

\[
\text{array } = \text{new} \ T[\text{n}];
\]

Q: How do we deallocate an array (as opposed to a single item)?

\[
\text{delete} \ [\text{}] \ \text{array}; \quad \text{// reclaims all $n$ allocated items}
\]

Q: What does it mean to do a “deep copy” of an array?

- assuming we have an “old” array of size \( m \) and a “new” array of capacity \( n \) ...
- copy \( m \) elements (one at a time) from old array to same indexes in new array

Note we never initialize the elements of the resizable array ...

- we track the amount filled via the \text{count}
- \text{count} \leq \text{capacity} is always true

Q: How do insert and erase work?

- by shifting array values to the left or right

We are never decreasing the capacity of an array ... only growing it

- could lead to wasted space
- in our implementation, array sizes grow from: 0, 1, 2, 4, 8, 16, 32, ... 
- note that from size 0 to 1 is a special case
- from size 1 on we are doubling the capacity