Example: resizable circular buffers

- Using an array implementation for a queue
- Can insert/remove one element at a time
- Insert new elements after the last/back current element
- Remove elements from the front
- Inserts might reach end of array when space is available at the front, so will have the end “wrap-around” to the front
- Dynamically-allocate the array with arbitrary size at start
- If the array is completely full we dynamically allocate a new one, twice as big, move everything across and delete old one

Wrap-around example

- array of size 5 (so positions 0,1,2,3,4)
- sequentially insert values 10, 17, 30, 29 (in positions 0,1,2,3)
- do 2 removes (takes out the 10,17, positions 0,1 now free)
- insert 63 (position 4)
- insert 8 - have reached end of array for insert positions, but the front (positions 0 and 1 right now) are available, so insert at position 0

Keeping track of front/back

- keep track of which array positions currently hold the first (front) element and the last (back) element
- set front and back to -1 whenever buffer is empty
- when insert into empty buffer, put in position 0 (front and back both 0 since it is both first and last right now)
- increment front on inserts, increment back on removes
- for array of size N, when front == N move front to 0 and when back == N move back to 0

Keeping track of current “size”

- Can keep track of number of currently stored elements using with separate variable, e.g. currsize=0 initially, increment/decrement with insert/remove
- always remember to check currsize > 0 before remove and currsize < array size before insert
- special cases when we remove only element in buffer (set front/back to -1) or when we insert element in empty buffer (set front/back to 0)

Don't absolutely need size variable

- Can keep track of current size using just front/back
- if no wrap around currently in use, current size is simply back + 1 - front
- if wrap around is in use (i.e. back < front) then this will be negative, off by N (allocated array size), so add N
- we know buffer is empty if front/back are both -1

Use of modulo with wrap around

- suppose we want the i'th element of those currently stored (i.e. offset from the front)
- if we know front, back, and N then we can compute whre in the array the ith element is stored using

  (front + i) % N

- if no wrap around in use, or the ith element is before the wrap around point, this gives same as front + i
- if i is after a wrap around point then this gives correct position at front of array

Resizing the buffer

- If we ever completely fill the buffer and need to do another insert then we need a new bigger buffer
  - will allocate a new one twice as big (and check it worked)
  - copy the content across to new buffer
  - change our array pointer to point to the new one
  - delete the old one