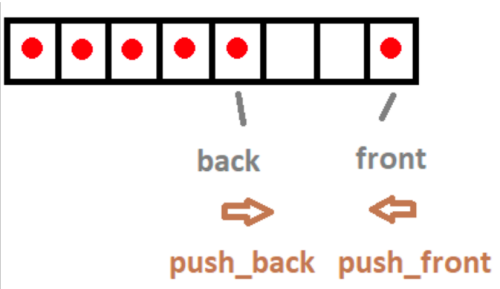
Problem 1(100pt): Safedeque class

Consider the `std::deque` class and implement your own `Safedeque` class which has similar functionality as `std::deque`. The data should be put in a contiguous block of memory and organized as



The `Safedeque` class is templated by a type `T` that has

- member `data` of type `std::unique_ptr<T[]>` to store the data;
- members `cap` and `sz` to track the capacity and size of the `deque`;
- members `front` and `back` to track the "frontmost" and "backmost" indices of data;
- a public nested `iterator` class;
- a default constructor to begin storing nothing, taking no heap memory, initializing the members appropriately;
- copy constructor, copy-assignmnent operator, and a destructor;
- `get_front` and `get_back` functions to access values at the front/back of the deque; throw `logic_error` exceptions if the container is empty;
- `push_front` and `push_back` functions to append values at the front/back of the deque; if the capacity is zero, add one spot to the deque, and print `"One spot added."`; if all spots are taken, double the deque capacity and print `"Capacity doubled."` to the console;
- `pop_front` and `pop_back` functions to remove the frontmost/backmost element; throw `logic_error` exceptions if the container is already empty; you don't need to account for shrinking the deque capacity;
- `begin` and `end` functions, overloaded on `const`, to return iterator objects;
- `size` and `capacity` functions to return the size and capacity;
- a subscript operator, overloaded on `const`, to retrieve the element at a given index;
- `print` function to print all elements from front to back; and
- `real_print` function to print all elements in `data` in their actual array positions from the first to the last. (This function helps us to determine if you implement the class correctly.) Use * to replace spots that are not taken.

The iterator class must have the usual operators as follows, and functions throws `out_of_range` error if bad arguments are supplied, e.g., deferencing or advancing `end()` iterator:

- prefix/postfix `operator++`;
- prefix/postfix `operator--`;
- dereferencing operator;
- `operator==`; and
- `operator!=`.

A `main.cpp` is given to show how this function will be used. The output looks like:

```
********** print empty deque *********

********** add elements *********
One spot added. Current capacity: 1
Capicity doubled. Current capacity: 2
Capicity doubled. Current capacity: 4
Capicity doubled. Current capacity: 8
6 3 2 4 5 7
********* real array elements**********
3 2 4 5 7 * * 6
********** element squares *********
36 9 4 16 25 49
********* work with iterators **********
9
36 9 4 20 25 100
********* copy control **********
36 9 4 20 25 100
********* throw error **********
```

Instructions:

- Your code will be graded based on correctness, efficiency, clearness, and practices. For this homework, the only library header files you are allowed to use are:
  - iostream, iomanip
  - stdexcept, memory
  - functional
- (5pt) Put all of your code in one file, named `Safedeque.h` and submitted it to CCLE. Do not include `int main` function in your code. Add description of this file in the beginning to show your ownership. A sample description may look like:

```
/* PIC 10C Homework 1, Heap.h
    Purpose: Define a template heap class
    Author: John Doe
    Date: 01/01/2021*/
```

- (10pt) Good coding practice includes commenting your code, using descriptive variable/function names, using efficient algorithms, etc. Coding practice part will be graded by three levels: 0, 5, 10.
- (85pt) Implement `Safedeque` class as instructed above.
- The official grading compiler is Visual Studio 2019 and you may lose majority of points if your code does not compile. If you don't have VS2019 installed in your computer, you are welcome to check your homework using virtual machines before submission. Please only check your homework after it has satisfying results on your local computer. Manually log out your account after using the machine.