Exercises

Write a generic data type for a *deque* and a *randomized queue*. The goal of this assignment is to implement fundamental collections using resizing arrays and linked lists.

Exercise 1

Dequeue. A *double-ended queue* or *deque* (pronounced "deck") is a generalization of a stack and a queue that supports adding and removing items from either the front or the back of the collection. Create a generic data type Deque that implements the following API:

```
template <typename Item>
class Deque {
public:
    // Construct an empty deque
    Deque();

    // Deallocate memory
    ~Deque();

    // Is the deque empty?
    bool empty() const;

    // Return the number of items on the deque
    int size() const;

    // Add the item to the front
    void push_front(const Item& item);

    // Add the item to the back
    void push_back(const Item& item);

    // Remove and return the item from the front
    Item pop_front();

    // Remove and return the item from the back
    Item pop_back();

    // unit testing
    static void unit_test();
};
```

Corner cases. Throw a `std::runtime_error` if the client calls either `pop_front` or `pop_back` when the deque is empty.

Unit testing. Your `Dequeue::unit_test()` method must call directly every public constructor and method to help verify that they work as prescribed (e.g., by printing results to standard output).

Performance requirements. Your implementation must achieve the following worst-case performance requirements:

- A deque containing n items must use at most $16n$ bytes of memory, not including the memory for the items themselves.
- Each deque operation (including construction) must take constant time.

Exercise 2

Randomized queue. A *randomized queue* is similar to a stack or queue, except that the item removed is chosen uniformly at random among items in the collection. Create a generic data type RandomizedQueue that implements the following API:

```
template<typename Item>
class RandomizedQueue {
public:
    // construct an empty randomized queue
    RandomizedQueue();

    // deallocate memory
    ~RandomizedQueue();

    // is the randomized queue empty?
    bool empty() const;

    // return the number of items on the randomized queue
    int size() const;

    // add the item
    void enqueue(const Item& item);

    // remove and return a random item
    Item dequeue();

    // return a random item (but do not remove it)
    Item sample();

    // unit testing (required)
    static void unit_test();
};
```

Corner cases. Throw a `std::runtime_error` if the client calls either `sample()` or `dequeue()` when the randomized queue is empty.

Unit testing. Your `RandomizedQueue::unit_test()` method must call directly every public constructor and method to help verify that they work as prescribed (e.g., by printing results to standard output).

Performance requirements. Your implementation must achieve the following worst-case performance requirements:

- A randomized queue containing n items must use at most $16n$ bytes of memory, not including the memory for the items themselves.
- Each randomized queue operation must take constant amortized time. That is, starting from an empty randomized queue, any intermixed sequence of m such operations must take $\Theta(m)$ time in the worst case.

Exercise 3

Clients. Write a client program to solve each of the following problems. You may only declare one variable in your client, and it must be of type `Deque` or `RandomizedQueue`.

(a) Given a command line parameter k, read in a sequence of strings from standard input, and print out a subset of exactly k of them, uniformly at random.

```
> cat distinct.txt
A B C D E F G H I

> Permutation 3 < distinct.txt
C
G
A

> Permutation 3 < distinct.txt
E
F
G
```

```
> cat duplicates.txt
AA BB BB BB BB BB CC CC

> Permutation 8 < duplicates.txt
BB
AA
BB
CC
BB
BB
CC
BB
```

(b) Read in a DNA sequence from standard input using `std::cin >> ch`, where `ch` is declared as **`char`** variable. Determine whether the string represents a *Watson-Crick complemented palindrome* (the string equals its reverse when you replace each base with its complement: A-T, C-G).

Palindromes in DNA have have many important biological roles. For example, tumor cells frequently amplify their genes by forming DNA palindromes.

Here are some examples of Watson-Crick complemented palindromes:

- `ACGT` as its reverse is `TGCA` and replacing `A` with `T`, `C` with `G`, `G` with `C`, and `T` with `A`, we get the original string.
- `ACGTACGT`
- `ACGTACGTACGTACGTACGT`

Submission

Submit the programs `RandomizedQueue.cpp`, `Deque.cpp`, `Palindrome.cpp` and `Permutation.cpp`. Your submission may not call library functions except those in `<iostream>`, `<cstdlib>`, `<vector>`, and `<random>`. In particular, do not use either `<dequeue>` or `<list>`. Finally, answer the questions provided in `readme_a1.txt` file and submit the file.

Grading

file	marks
`Deque.cpp`	15
`RandomizedQueue.cpp`	15
`Permutation.cpp`	5
`Palindrome.cpp`	5
`readme.txt`	5