A Deque allows you to add and remove elements from both ends.

Question 1. How should a deque be implemented?

Question 2. Why stacks and queues, when Deques are more powerful?

Question 3. Why Deques, when a vector is more powerful?

Operator Overloading

Can you define both of these functions simultaneously?

```c
int minimum(int x, int y);
double minimum(double x, double y);
```

How about these?

```c
bool foo(int x);
void foo(int y);
```

Operators such as ==, <=, etc, are functions too. They can be overloaded as well.
class IntArray {
    public:
    bool operator==(const IntArray& otherArray) {  
        if (this->size != otherArray.size) return false;
        for (int i = 0; i < size; i++)  
            if (this->data[i] != otherArray.data[i]) return false;
        return true;
    }
    int& operator[](int index) {  
        return data[index];
    }
    IntArray& operator++() {  
        for (int i = 0; i < size; i++) data[i]++;
        return *this;
    }
private:
    int size;
    int *data;
};

We can now write code such as the following:
if (firstArray == secondArray) ++firstArray;
secondArray[0] = 0;

Question 4. Why do you suppose operator++ has a return type, instead of return void?

Question 5. Why do you suppose operator[] returns by reference?

Let's look at the implementation of operator++.
IntArray IntArray::operator++(int dummy) {  
    IntArray copy = *this;
    ++(*this);
    return copy;
}

Let's also look at the implementation of operator*.
IntArray IntArray::operator* (int multiplier) {  
    IntArray newArray;
    newArray.size = size;
    for (int i = 0; i < size; i++) newArray.data[i] = data[i]*multiplier;
    return newArray;
}

Question 6. Why did we not return by reference?