# **Problem 1. Dynamic Array 2.0**

This problem is based on the Dynarray you wrote in Homework 4 Problem 3. Before adding anything new to it, your Dynarray should meet all the requirements in that problem first.

In this task, your Dynarray should support the following new things.

## Type alias members

The standard library containers have many type alias members for the purpose of supporting generic types. For example, std::vector<T>::value\_type is T, std::vector<T>::size\_type is std::vector<T>::pointer is T.

As an exercise, do the same thing in your Dynarray. You need to define the following type alias members, all of which should be public:

| type alias member         | definition  |
|---------------------------|-------------|
| Dynarray::size_type       | std::size_t |
| Dynarray::value_type      | int         |
| Dynarray::pointer         | int *       |
| Dynarray::reference       | int &       |
| Dynarray::const_pointer   | const int * |
| Dynarray::const_reference | const int & |

Moreover, we will eventually make this <code>Dynarray</code> a class template <code>Dynarray<T></code> that can store any types of data, not only <code>int</code> s. This will be in Homework 7 or 8, depending on the lecture schedule. To make your work easier by then, you'd better make full use of the type alias members you have defined. For example, change <code>new int[n]</code> to <code>new value\_type[n]</code>, change <code>int &</code> to <code>reference</code>, and change (<code>const int \*begin</code>, <code>const int \*end</code>) to (<code>const\_pointer begin</code>, <code>const\_pointer end</code>), etc. By the time we make this a class template, you will just have to modify very few things.

## **Subscript operator**

The Dynarray should support the subscript operator, so that we can use a[i] instead of a.at(i) to access the i-th element.

Let a be an object of type Dynarray or const Dynarray. The behavior of a[i] should be exactly the same as a.at(i), except that the subscript operator does not perform bounds checking. That is, no exception should be thrown if  $i \ge a.size()$ .

#### **Relational operators**

The Dynarray should support the six relational operators: <, <=, >, >=, == and !=. These operators perform *lexicographical comparison* of two Dynarray S.

Lexicographical comparison is an operation with the following properties:

Two ranges are compared element by element.

- The first mismatching element defines which range is lexicographically *less* or *greater* than the other.
- If one range is a prefix of another, the shorter range is lexicographically *less* than the other.
- If two ranges have equivalent elements and are of the same length, then the ranges are lexicographically *equal*.
- An empty range is lexicographically less than any non-empty range.
- Two empty ranges are lexicographically equal.

Since we use C++17, you still have to define all six of them. It is often good practice to implement operator< and operator== first, and define the rest in terms of them. Moreover, **you are not** allowed to write loops or recursions in these six functions. Go through this page and find the appropriate standard library algorithms.

Note that in homework 7 or 8, we will make this <code>Dynarray</code> a class template <code>Dynarray<T></code>, and we should always minimize the requirements on unknown types when we do generic programming. Since C++17 does not have compiler-generated comparison operators, we suggest you making your implementation <code>depend only upon the operator< and operator== of the element type</code>.

You are free to choose to define them as either members or non-members.

# **Output operator**

We want to print a Dynarray directly using operator<<. For example,

```
int arr[] = {1, 2, 3, 5};
Dynarray a(arr, arr + 4);
Dynarray b;
std::cout << a << '\n' << b << std::endl;</pre>
```

The output is as follows.

```
[1, 2, 3, 5]
[]
```

In details:

- Elements are separated by a comma (,) followed by a space.
- The printed content starts with [] and ends with []]. If the dynamic array is empty, just print an empty pair of brackets [].

#### OJ tests

There are three subtasks on OJ. Subtask i will be run only if all the testcases of subtask i-1 are passed.

Subtask 1 is a compile-time check. Subtask 2 contains all the testcases from Homework 5 Problem 2. Subtask 3 contains the new testcases specific for this problem.