COEN 79L - Object-Oriented Programming and Advanced Data Structures

Lab 8

- **Deque**

In this project you implement a template deque class, similar to the STL's deque. Although a deque could be easily implemented using a linked list, as we mentioned before, using arrays enables the storage of data in contiguous memory locations, which results in higher performance due to efficient utilization of cache memory. In this project you also implement an iterator class for deque.

The deque data structure has two main components:

- **Data blocks**: Each block is an array of data values. Data blocks are added and removed when items are added or removed. Please note that all the data blocks have the same size.
- **An array of block pointers**: All the entries of this array are initially NULL. When a data block is added, an entry of this array points to the new data block. In order to avoid frequent resizing of the array of block pointers, the pointer to the first block is allocated in the middle of this array. The `reserve()` function is provided for you in the given files.

Please note that the size of data blocks never changes. However, by allocating a larger array of data pointers, more data blocks can be allocated to store more elements.

The initial size of the two arrays are:

```
1. Static const size_t BLOCK_SIZE = 5; // Number of data items per data block
2. static const size_t BLOCKPOINTER_ARRAY_SIZE = 5; // Number of entries in the array of block
   pointers. The minimum acceptable value is 2.
```

Before implementing the functions, you should completely understand the pointers used.

Two pointer variables point to the array of block pointers:

```
1. // A pointer to the dynamic array of block pointers
2. value_type** block_pointers;
3.
4. // A pointer to the final entry of the array of block pointers
5. value_type** block_pointers_end;
```

The first pointer points to the beginning of the array, and the second one points to the last entry of the array.

Two other pointer variables point to the first and last used entry of the array of block pointers:

```
1. // A pointer to the first block pointer that is being used (i.e., points to a data block)
2. value_type** first_bp;
3.
4. // A pointer to the last block pointer that is being used
```

```
5.  value_type** last_bp;
```

For example, assume the array of block pointers has 5 entries. Entry 2 of this array points to the first data block and entry 4 points to the last data block. In this case, `first_bp` and `last_bp` point to entry 2 and 4, respectively.

There are two pointers to point to the first and last element of the deque, which are stored in data blocks:

```
1.  value_type * front_ptr; // A pointer to the front element of the deque
2.  value_type * back_ptr;  // A pointer to the back element of the deque
```

In addition to the deque class, we implement a forward iterator. The forward iterator supports these operators: `prefix ++, postfix ++, dereferencing *, equality ==, not equal !=`
The set of pointer variables of the iterator class is similar to the deque class. However, in addition to those, the iterator class needs a cursor that moves towards the end of the deque. We also need to keep track of the current data block as well as the last entry of the data block. These three pointers enable us to move the cursor and jump from one data block to another when necessary.

The provided files include comments to simplify the implementation of functions. Please read these comments carefully.

As mentioned in the lectures, the implementation file of template classes should be included in the header file. In this project, we implement functions in .impl files and include them at the bottom of their respective .h files. Therefore, we have two files for the deque class and two files for the deque iterator.

Given files:

- `deque.h:` Declaration of the deque template class.
- `deque.impl:` Incomplete implementation of the deque template class for you to finish.
- `deque_iterator.h:` Declaration of an iterator for the deque class.
- `deque_iterator.impl:` Incomplete implementation of the iterator for the deque class for you to finish.
- `deque_test.cpp:` A test file for the deque class.