Assignment 5

Create the following files:

1. *myArray.h*: Implement a dynamic array similar to the STL vector, which should support the following operations at least:

   - *push_back*(): Add a new element at the end of the array, after its current last element.
   - *pop_back*(): Remove the last element in the array if there exits elements. Whether reducing the container capacity or not is OK for simplicity.
   - *iterator*: It can be used to access elements and iterate through the array
   - *begin*(): Return an iterator pointing to the first element in the array
   - *end*(): Return an iterator to one beyond the last element
   - *size*(): Return the number of elements in the array
   - *capacity*(): Return the size of the storage space currently allocated for the array, expressed in terms of elements. It can be equal or greater than size(). When the capacity is exhausted and more is needed, you could simply double it.
   - *operator*[ ]: [n] will return the element at position n in the array.
   - *operator* =: Assign new contents to the container, replacing its current contents, and modifying its size accordingly.

```
template<typename Elem>
class myArray {
    // representation and implementation details
public:
    class iterator; // member type: iterator

    void push_back(const Elem& v); // insert v at end
    void pop_back();// remove the last element

    iterator begin(); // iterator to first element
    iterator end( );// iterator to one beyond last element

    int size( ); // the number of elements
    int capacity();

    Elem& operator[] (const int i);
    Elem& operator= (const Elem i);
    // . . .
};
```

Note: Change or add more functions if needed.

2. *myData.h*: Define a Struct or Class *myData* with two fields *val* and *addr*. Complete *Cmp_by_val* and *Cmp_by_addr* to sort arrays according to *myData.val* and *myData.addr* respectively in ascending order.

```
template<typename T>
struct myData{
    T val;
    myData<T>* addr;
    //...
};

template<typename T>
struct Cmp_by_val {
    bool operator()(myData<T> a, myData<T> b) {
        return a.val < b.val; //compare val
    }
};

template<typename T>
struct Cmp_by_addr {
    bool operator()(myData<T> a, myData<T> b) {
        return a.addr < b.addr; //compare addr
    }
};
```

Note: Change or add more functions if needed.

3. *mySort.h*: Implement a generic sorting algorithm like the built-in *sort*, which is able to serve your dynamic array defined in *myArray.h*.

```
    template <class Iterator>
    void mySort(Iterator first, Iterator last)

    template <class Iterator, class Compare>
    void mySort(Iterator first, Iterator last, Compare comp)
```

The *comp* means that users can parameterize sort by the comparison criteria.

Note: You can do with any sorting algorithms, e.g. merge sort, insertion sort, quick sort and so on. But don't use the built-in *sort* directly!

4. *main.cpp*: The *main.cpp* is given by us. Use it to test whether your *myArray.h*, *myData.h* and *mySort.h* are correct. The process of the code is as follows, which also can be seen in *main.cpp*.

   - Test *myArray.h*: Create an *myArray* object and call its members.
   - Test *myData.h*: Create an object of $myArray < myData >$. Randomly input some numerical values. Each element in $myArray < myData >$ object stores each input and the address of this element. Figure 1 illustrates the process.

- Test *mySort.h*: Sort the *myArray* < *myData* > object above using *mySort* by *val* and *addr* fields respectively.

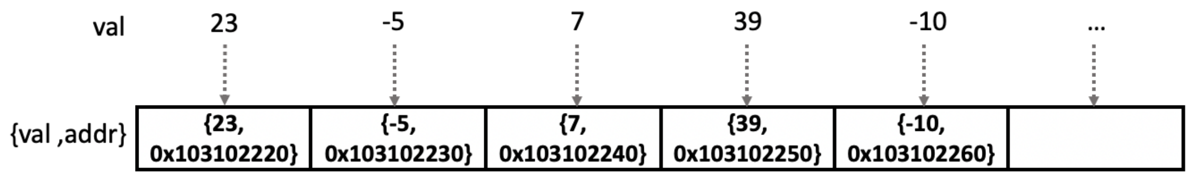

Figure 1: Elements in the dynamic arrays.

Note: You can modify some details of *main.cpp* to satisfy requirements.

Important Notes:

- **Reference**: The book, *Programming: Principles and Practice Using C++, Stroustrup.*, in the Reading Material. Please refer to *Chapter 20 Containers and Iterators* and *Chapter 21 Algorithms and Maps*, which are also covered in Lecture 8.
- Remember to submit your makefile!
- Due: 2019/10/19 11:59pm