Lab 2- Sequence Class

Implement a **template container class that encapsulates a dynamic array** and test it using a suitable driver program (You may use one file for your template class instead of breaking it into 2 files.). **Header file is provided on page 2.**

Your class should include the following **data members**:

- A pointer to the array
- A variable to hold the current array capacity (capacity)
- A variable to keep track of the number of elements currently being used. (num_used)

Include **2 constructors (**or you can write one constructor with a default value for size**)**:

- The default constructor dynamically allocates an array of size 10 and initializes capacity to 10 and num_used variable to 0.
- A one argument constructor dynamically allocates an array of the size passed as an argument and initializes capacity to that number and num_used variable to 0.

Include a **destructor** that will release the allocated memory.

Include methods to:

- Determine if the sequence is **full** (return a bool value)
- Determine if the sequence is **empty** (return a bool value)
- Return the sequence's **capacity**
- Return the **number of elements** currently in the sequence

Additional public methods:

- A method to return an element at a given position. (at)
- A method to overwrite an element at a given position. (set)
- A method to **add** a given element to the end of the array. (push_back)
- A method to **insert** an element at a given position in the array. (insert)
- A method to **remove** an element from a given position in the array. (erase)

Note: Make sure you check for out of bounds conditions and the cases when the sequence is full (positions from 0 to num_used -1). Use the **assert function** when applicable.

Your **driver program** should declare a sequence object; input values from the keyboard; manipulate the array elements by making calls to the public methods – push_back, at, etc.; and print the contents of sequence after the initial input and again after all the manipulations are performed. The main method should implement a menu system with a quit option so the user can make selections for at, set, push_back, etc.

Part B:

Once your program is working properly, modify your methods that **add** elements to the sequence (push_back and insert) by including the following feature:

If the sequence has reached its capacity, the push_back / insert method should call a **private helper function** to double the size of the private array. The helper function will dynamically allocate storage for an array of twice the size, copy all the elements from the original array into the new array, and update all the data members. The private pointer data member should point to this new storage area after the copy is complete. The push_back/insert function will then add the new element to the sequence.

<u>Submit ONE program that includes this feature (parts A & B combined).</u>

```
#ifndef SEQUENCE_H
#define SEQUENCE_H

template <class value_type>
    class sequence
    {
    public:
        sequence( );
        sequence(int size);
        ~sequence();

        bool full( ) const;
        bool empty(  )const;
        int num_element()const;
        int max_capacity()const;
        value_type at(int position) const;
        void set(const value_type& entry, int position);
        void insert(const value_type& entry, int position);
        void push_back(const value_type& entry);
        void remove_current(int position);
        void print()const;
    private:
        value_type* data;
        int num_used;
        int capacity;
    };

#endif
```