# DSA/CS 5005 – Computing Structures – Programming Project – 2 – Fall 2016 (Due: October 28, 2016 11:59 PM)

## Objectives

1. [5 pts] Use template in the project
2. [5 pts] Implement exception handling for SortedArray and LinkedSortedArrays classes
3. [5 pts] Implement binary search
4. [35 pts] Implement SortedArray class
   - 5 points for each of the items from ostream operator to join()
5. [30 pts] Implement LinkedSortedArrays class
   - 10 points for insert()
   - 5 points for each of the rest of the items from ostream operator to operator []
6. [20 pts] Implement main() that read and process the input according to the project description
   - 5 points for each of the I, R, F O commands.
7. Document your project thoroughly as the examples in the textbook. This includes, but is not limited to, header comments for all classes/methods, explanatory comments for each section of code, meaningful variable and method names, and consistent indentation. Program that lack or are weak in documentation will receive a *deduction* of up to 30% of the grade.

## Project Description

In this project, you will learn how to use the *list* class from STL (**Standard Template Library**) and implement a new data structure – *Linked Sorted Arrays*.

### *Linked Sorted Arrays*

We know by now that if the elements are stored in sorted order in an array, we can perform binary search to locate the element much faster. The time consuming issues with the sorted arrays are the following:

- **insert** - we have to move the elements to the right of the index position where the insert occurs or create a larger array to move the elements from the old array to the new larger array.
- **remove** - we have to move the elements to the left and if the number of elements in the array becomes very small in proportion to the size of the array, we need to again create a smaller array and copy the elements from the old array to the new array.

Let us describe now, how the *Linked Sorted Arrays* data structure will work. Assume that we have a defined constant **K** (which is set to 4, for illustration). We will create an array of size **K**, call this array A. We will insert elements into array A and make sure the elements are kept in sorted order. As soon as the array A of size **K** is filled up and we want to insert the $(\mathbf{K+1})^{th}$ element. Rather than creating a larger array, we would create another array B (of size **K**); move half the elements in A into the array B such that the largest number in A is less than the smallest number in B. The array A is now linked to array B as in a linked list. You will have to implement at least the following two classes: SortedArray and LinkedSortedArrays in this project as described below.

During the remove operation form the array A of size **K**, if the number of elements after removal

becomes less than a threshold value (say **K/2**), then remaining elements in A are merged with either the array that points to A or the array A points to as long as there is room in either of them.

*SortedArray class*

SortedArray class is used to store an array of a defined size (decided at runtime this is the number **K**), the size of the array (**K**), and the number of elements currently in the array as below.

```
template <class DT>

class SortedArray
{
private:
      int capacity;   // capacity, which is K
      int size;   // num. of elements stored
      DT* elements;   // array used to store the elements, size of K
      // other private methods
public:
     // public methods
};
```

The elements are stored in sorted order in the array. The elements are not by themselves stored in the array, but rather pointers to the elements are stored. This class should implement the standard methods for all classes (as described in your textbook) and the following methods:

- ostream operator – Output the objects stored in the array in a single line, objects are separated by a space and followed by an "endl" at the end of the output.
- DT& overloaded [] (int index), which returns the element (by box) at position index of the array.
- DT& find (DT& lookFor), which performs binary search to return the reference to the element at the location, otherwise throw an exception.
- void insert (DT& newElement) – throw an exception if there is no room to insert into the array, or an object with the same value is already stored in the array. Otherwise, it inserts newElement in the sorted order. Make sure to perform a *binary search* to determine if the element is already stored and to locate the position where to insert the element to.
- void remove (DT& oldElement) – throw an exception if the element is not present in the array, otherwise it removes the element by shifting the elements to the left. Use binary search to find the element.
- SortedArray<DT>& split (int i) – returns a SortedArray containing elements from position i through the last element of the array. The elements from i through the last element are removed from this array and adjust the variable size of this array.
- void join (SortedArray<DT>& P) – Copies the elements from array P to *this* array making sure that the elements are in sorted array. The SortedArray P is destroyed after the copy.

*LinkedSortedArrays class*

This data structure has the following form (again this is only a starting point; you have to change this appropriately).

```
template <class DT>
class LinkedSortedArrays
{
   private:
      list<SortedArray<DT> > nameIT;
      int ArraySizeFactor; //The maximum size of the array in the SortedArray
   // other private methods
   public:
      DT& find (const DT& key);
      void insert (const DT& newOne);
      void remove (const DT& X);
   // other public methods
};
```

This class should implement the standard methods for all classes (as described in your textbook) and the following methods:

- ostream operator – Output the objects (SortedArray object) stored in the list using the ostream operator of SortedArray from the head to the tail of the list.
- DT& find (const DT& key) – The find method will start iterate over the linked list structure until either the element is found or all the nodes of the linked list have been searched. When processing a node during the iteration, it will first check if the element we are looking for is within the bounds of the max and min values stored in the SortedArray class object. If that is the case, then the find method on the object is executed, otherwise an exception will be thrown *if the element you are looking for is smaller than the min value or we have reached the last node of the linked list and the max value of the last node is smaller than the number we are looking for*. If the max value is smaller than the number we are looking for, we move the next element of the linked list, and the process continues. find() returns a reference to the target object.
- void insert (const DT& newOne) – The insert method will also iterate over the linked list. We will encounter three choices: **a)** the element is already in the data structure (thrown an exception), **b)** we find a SortedArray (in the linked list) where the element has to be inserted and has space, and **c)** we find a SortedArray where the element has to be inserted, but it does not have space. Choices a) and b) are easy to handle (and are described previously). In case of choice c), when we apply the insert method on the SortedArray object, it will throw an exception. As soon as the exception is caught, we apply a split operation to the SortedArray object by sending a number ArraySizeFactor/2. Now the new SortedArray object is inserted next to the linked list object (containing the SortedArray).
- remove (const DT& x) – You should now guess how the remove method should work. Of course, we will discuss this in the class.
- DT& overloaded [] (int index), which returns the element (by box) at position index of the array. Example: Let us say that the LinkedSortedArray has two elements (A and B). Further assume that A and B have 4 and 3 elements, respectively. If the index position specified is 0, then the

[] operator should return the first element from A. If the index position specified is a 6, it should return the third element from B (6-4 (no of elements in A) = 2). An exception should be thrown if the index is out of bounds.

*Input File*

We will deal with a set of integers for this project. The first line of the program will be the value for K (described earlier), followed by commands for I (insert), R (remove), F (find), O (output the data structure – LinkedSortedArrays, using the ostream operator). An input file will be posted to D2L later this week.

*Output*

After each operation, display the outcome of the operation.

## Constraints

1. This project must be implemented in C++. The only header file you will have <iostream>, <list> and <cstdlib> (for atoi() which can be used to convert a C string to an integer in the input file). We will not use any other libraries.
2. None of the projects will be a group project. Consulting with other members of this class on programming projects is strictly not allowed and plagiarism charges will be imposed on students who do not follow this.